

**GigaDevice Semiconductor Inc.**

**GD32W51x**

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

**固件库  
使用指南**

1.2 版本

(2022 年 12 月)

# 目录

目录.....	2
图索引 .....	5
表索引 .....	6
<b>1. 介绍.....</b>	<b>32</b>
<b>1.1. 文档和固件库规则.....</b>	<b>32</b>
1.1.1. 外设缩写 .....	32
1.1.2. 命名规则 .....	33
<b>2. 固件库概述.....</b>	<b>34</b>
<b>2.1. 文件组织结构.....</b>	<b>34</b>
2.1.1. Examples 文件夹 .....	35
2.1.2. Firmware 文件夹 .....	35
2.1.3. Template 文件夹.....	35
2.1.4. Template_Trustzone 文件夹.....	38
2.1.5. Utilities 文件夹.....	43
<b>2.2. 固件库文件描述 .....</b>	<b>43</b>
<b>3. 外设固件库.....</b>	<b>44</b>
<b>3.1. 外设固件库概述 .....</b>	<b>44</b>
<b>3.2. ADC .....</b>	<b>44</b>
3.2.1. 外设寄存器描述 .....	44
3.2.2. 外设库函数说明 .....	45
<b>3.3. CAU .....</b>	<b>72</b>
3.3.1. 外设寄存器说明 .....	72
3.3.2. 外设库函数说明 .....	72
<b>3.4. CRC .....</b>	<b>100</b>
3.4.1. 外设寄存器说明 .....	100
3.4.2. 外设库函数说明 .....	100
<b>3.5. DBG .....</b>	<b>104</b>
3.5.1. 外设寄存器说明 .....	104
3.5.2. 外设库函数说明 .....	104
<b>3.6. DCI.....</b>	<b>110</b>
3.6.1. 外设寄存器说明 .....	110
3.6.2. 外设库函数说明 .....	111
<b>3.7. DMA.....</b>	<b>124</b>
3.7.1. 外设寄存器说明 .....	124

3.7.2.	外设库函数说明 .....	125
<b>3.8.</b>	<b>EFUSE .....</b>	<b>155</b>
3.8.1.	外设寄存器说明 .....	155
3.8.2.	外设库函数说明 .....	156
<b>3.9.</b>	<b>EXTI .....</b>	<b>169</b>
3.9.1.	外设寄存器说明 .....	169
3.9.2.	外设库函数说明 .....	170
<b>3.10.</b>	<b>FMC .....</b>	<b>180</b>
3.10.1.	外设寄存器说明 .....	180
3.10.2.	外设库函数说明 .....	181
<b>3.11.</b>	<b>FWDGT .....</b>	<b>206</b>
3.11.1.	外设寄存器说明 .....	206
3.11.2.	外设库函数说明 .....	206
<b>3.12.</b>	<b>GPIO .....</b>	<b>211</b>
3.12.1.	外设寄存器说明 .....	211
3.12.2.	外设库函数说明 .....	211
<b>3.13.</b>	<b>HAU .....</b>	<b>224</b>
3.13.1.	外设寄存器说明 .....	224
3.13.2.	外设库函数说明 .....	224
<b>3.14.</b>	<b>HPDF .....</b>	<b>243</b>
3.14.1.	外设寄存器说明 .....	243
3.14.2.	外设库函数说明 .....	243
<b>3.15.</b>	<b>I2C .....</b>	<b>298</b>
3.15.1.	外设寄存器说明 .....	298
3.15.2.	外设库函数说明 .....	299
<b>3.16.</b>	<b>ICACHE .....</b>	<b>338</b>
3.16.1.	外设寄存器说明 .....	338
3.16.2.	外设寄存器说明 .....	338
<b>3.17.</b>	<b>MISC .....</b>	<b>349</b>
3.17.1.	外设寄存器说明 .....	349
3.17.2.	外设库函数说明 .....	350
<b>3.18.</b>	<b>PKCAU .....</b>	<b>357</b>
3.18.1.	外设寄存器说明 .....	357
3.18.2.	外设库函数说明 .....	357
<b>3.19.</b>	<b>PMU .....</b>	<b>386</b>
3.19.1.	外设寄存器说明 .....	386
3.19.2.	外设库函数说明 .....	386
<b>3.20.</b>	<b>QSPI .....</b>	<b>403</b>
3.20.1.	外设寄存器说明 .....	403

3.20.2.	外设库函数说明 .....	404
<b>3.21.</b>	<b>RCU .....</b>	<b>421</b>
3.21.1.	外设寄存器说明 .....	422
3.21.2.	外设库函数说明 .....	423
<b>3.22.</b>	<b>RTC .....</b>	<b>471</b>
3.22.1.	外设寄存器描述 .....	471
3.22.2.	外设库函数描述 .....	472
<b>3.23.</b>	<b>SDIO .....</b>	<b>508</b>
3.23.1.	外设寄存器说明 .....	508
3.23.2.	外设库函数说明 .....	509
<b>3.24.</b>	<b>SPI .....</b>	<b>540</b>
3.24.1.	外设寄存器说明 .....	540
3.24.2.	外设库函数说明 .....	540
<b>3.25.</b>	<b>SQPI .....</b>	<b>568</b>
3.25.1.	外设寄存器说明 .....	568
3.25.2.	外设库函数说明 .....	569
<b>3.26.</b>	<b>SYSCFG .....</b>	<b>574</b>
3.26.1.	外设寄存器说明 .....	575
3.26.2.	外设库函数说明 .....	575
<b>3.27.</b>	<b>TIMER .....</b>	<b>589</b>
3.27.1.	外设寄存器说明 .....	589
3.27.2.	外设库函数说明 .....	589
<b>3.28.</b>	<b>TRNG .....</b>	<b>646</b>
3.28.1.	外设寄存器说明 .....	646
3.28.2.	外设库函数说明 .....	646
<b>3.29.</b>	<b>TSI .....</b>	<b>651</b>
3.29.1.	外设寄存器描述 .....	651
3.29.2.	外设库函数说明 .....	652
<b>3.30.</b>	<b>TZPCU .....</b>	<b>672</b>
3.30.1.	外设寄存器说明 .....	672
3.30.2.	外设库函数说明 .....	673
<b>3.31.</b>	<b>USART .....</b>	<b>695</b>
3.31.1.	外设寄存器说明 .....	696
3.31.2.	外设库函数说明 .....	696
<b>3.32.</b>	<b>WWDGT .....</b>	<b>744</b>
3.32.1.	外设寄存器说明 .....	744
3.32.2.	外设库函数说明 .....	744
<b>4.</b>	<b>版本历史 .....</b>	<b>749</b>



## 图索引

图 2-1. GD32W51x 固件库文件组织结构.....	34
图 2-2. 选择外设例程文件 .....	36
图 2-3. 拷贝外设例程文件 .....	37
图 2-4. 打开工程文件 .....	37
图 2-5. 配置工程文件 .....	38
图 2-6. 编译调试下载 .....	38
图 2-7. 选择外设例程文件 .....	39
图 2-8. 拷贝外设例程文件 .....	40
图 2-9. 打开工程文件 .....	41
图 2-10. 配置工程文件 .....	42
图 2-11. 编译调试下载.....	42

## 表索引

表 1-1. 外设缩写 .....	32
表 2-1. 固件函数库文件描述 .....	43
表 3-1. 外设固件库函数描述格式 .....	44
表 3-2. ADC 寄存器 .....	44
表 3-3. ADC 库函数 .....	45
表 3-4. 函数 adc_deinit .....	46
表 3-5. 函数 adc_clock_config .....	46
表 3-6. 函数 adc_enable .....	47
表 3-7. 函数 adc_disable .....	48
表 3-8. 函数 adc_dma_mode_enable .....	48
表 3-9. 函数 adc_dma_mode_disable .....	49
表 3-10. 函数 adc_dma_request_after_last_enable .....	49
表 3-11. 函数 adc_dma_request_after_last_disable .....	50
表 3-12. 函数 adc_discontinuous_mode_config .....	50
表 3-13. 函数 adc_special_function_config .....	51
表 3-14. 函数 adc_channel_9_to_11 .....	52
表 3-15. 函数 adc_data_alignment_config .....	52
表 3-16. 函数 adc_channel_length_config .....	53
表 3-17. 函数 adc_regular_channel_config .....	54
表 3-18. 函数 adc_inserted_channel_config .....	55
表 3-19. 函数 adc_inserted_channel_offset_config .....	56
表 3-20. 函数 adc_external_trigger_config .....	56
表 3-21. 函数 adc_external_trigger_source_config .....	57
表 3-22. 函数 adc_software_trigger_enable .....	59
表 3-23. 函数 adc_end_of_conversion_config .....	60
表 3-24. 函数 adc_regular_data_read .....	61
表 3-25. 函数 adc_inserted_data_read .....	61
表 3-26. 函数 adc_watchdog_single_channel_enable .....	62
表 3-27. 函数 adc_watchdog_group_channel_enable .....	62
表 3-28. 函数 adc_watchdog_disable .....	63
表 3-29. 函数 adc_watchdog_threshold_config .....	63
表 3-30. 函数 adc_oversample_mode_config .....	64
表 3-31. 函数 adc_oversample_mode_enable .....	66
表 3-32. 函数 adc_oversample_mode_disable .....	66
表 3-33. 函数 adc_flag_get .....	67
表 3-34. 函数 adc_flag_clear .....	67
表 3-35. 函数 adc_interrupt_enable .....	68
表 3-36. 函数 adc_interrupt_disable .....	69
表 3-37. 函数 adc_interrupt_flag_get .....	69
表 3-38. 函数 adc_interrupt_flag_clear .....	70

表 3-39. 函数 <code>adc_regular_software_startconv_flag_get</code> .....	71
表 3-40. 函数 <code>adc_regular_software_startconv_flag_get</code> .....	71
表 3-41. CAU 寄存器 .....	72
表 3-42. CAU 库函数 .....	72
表 3-43. 结构体 <code>cau_key_parameter_struct</code> .....	73
表 3-44. 结构体 <code>cau_iv_parameter_struct</code> .....	74
表 3-45. 结构体 <code>cau_context_parameter_struct</code> .....	74
表 3-46. 结构体 <code>cau_parameter_struct</code> .....	74
表 3-47. 函数 <code>cau_deinit</code> .....	75
表 3-48. 函数 <code>cau_struct_para_init</code> .....	75
表 3-49. 函数 <code>cau_key_struct_para_init</code> .....	76
表 3-50. 函数 <code>cau_iv_struct_para_init</code> .....	76
表 3-51. 函数 <code>cau_context_struct_para_init</code> .....	77
表 3-52. 函数 <code>cau_enable</code> .....	77
表 3-53. 函数 <code>cau_disable</code> .....	78
表 3-54. 函数 <code>cau_dma_enable</code> .....	78
表 3-55. 函数 <code>cau_dma_disable</code> .....	79
表 3-56. 函数 <code>cau_init</code> .....	79
表 3-57. 函数 <code>cau_aes_keysize_config</code> .....	81
表 3-58. 函数 <code>cau_key_init</code> .....	81
表 3-59. 函数 <code>cau_iv_init</code> .....	82
表 3-60. 函数 <code>cau_phase_config</code> .....	82
表 3-61. 函数 <code>cau_fifo_flush</code> .....	83
表 3-62. 函数 <code>cau_enable_state_get</code> .....	83
表 3-63. 函数 <code>cau_data_write</code> .....	84
表 3-64. 函数 <code>cau_data_read</code> .....	84
表 3-65. 函数 <code>cau_context_save</code> .....	85
表 3-66. 函数 <code>cau_context_restore</code> .....	86
表 3-67. 函数 <code>cau_aes_ecb</code> .....	86
表 3-68. 函数 <code>cau_aes_cbc</code> .....	87
表 3-69. 函数 <code>cau_aes_ctr</code> .....	88
表 3-70. 函数 <code>cau_aes_cfb</code> .....	89
表 3-71. 函数 <code>cau_aes_ofb</code> .....	90
表 3-72. 函数 <code>cau_aes_gcm</code> .....	91
表 3-73. 函数 <code>cau_aes_ccm</code> .....	92
表 3-74. 函数 <code>cau_tdes_ecb</code> .....	93
表 3-75. 函数 <code>cau_tdes_cbc</code> .....	94
表 3-76. 函数 <code>cau_des_ecb</code> .....	95
表 3-77. 函数 <code>cau_des_cbc</code> .....	96
表 3-78. 函数 <code>cau_interrupt_enable</code> .....	96
表 3-79. 函数 <code>cau_interrupt_disable</code> .....	97
表 3-80. 函数 <code>cau_interrupt_flag_get</code> .....	98
表 3-81. 函数 <code>cau_flag_get</code> .....	98
表 3-82. CRC 寄存器 .....	100

表 3-83. CRC 库函数 .....	100
表 3-84. 函数 crc_deinit .....	100
表 3-85. 函数 crc_data_register_reset .....	101
表 3-86. 函数 crc_data_register_read .....	101
表 3-87. 函数 crc_free_data_register_read .....	102
表 3-88. 函数 crc_free_data_register_write .....	102
表 3-89. 函数 crc_single_data_calculate .....	103
表 3-90. 函数 crc_block_data_calculate .....	103
表 3-91. DBG 寄存器 .....	104
表 3-92. DBG 库函数 .....	104
表 3-93. 枚举类型 dbg_periph_enum .....	105
表 3-94. 函数 dbg_deinit .....	105
表 3-95. 函数 dbg_id_get .....	106
表 3-96. 函数 dbg_low_power_enable .....	106
表 3-97. 函数 dbg_low_power_disable .....	107
表 3-98. 函数 dbg_periph_enable .....	107
表 3-99. 函数 dbg_periph_disable .....	108
表 3-100. 函数 dbg_trace_pin_enable .....	109
表 3-101. 函数 dbg_trace_pin_disable .....	109
表 3-102. 函数 dbg_trace_pin_mode_set .....	110
表 3-103. DCI 寄存器 .....	110
表 3-104. DCI 库函数 .....	111
表 3-105. 结构体 dci_parameter_struct .....	112
表 3-106. 函数 dci_deinit .....	112
表 3-107. 函数 dci_struct_para_init .....	113
表 3-108. 函数 dci_init .....	113
表 3-109. 函数 dci_enable .....	114
表 3-110. 函数 dci_disable .....	114
表 3-111. 函数 dci_capture_enable .....	115
表 3-112. 函数 dci_capture_disable .....	115
表 3-113. 函数 dci_jpeg_enable .....	116
表 3-114. 函数 dci_jpeg_disable .....	116
表 3-115. 函数 dci_crop_window_enable .....	117
表 3-116. 函数 dci_crop_window_disable .....	117
表 3-117. 函数 dci_crop_window_config .....	118
表 3-118. 函数 dci_embedded_sync_enable .....	118
表 3-119. 函数 dci_embedded_sync_disable .....	119
表 3-120. 函数 dci_sync_codes_config .....	119
表 3-121. 函数 dci_sync_codes_unmask_config .....	120
表 3-122. 函数 dci_data_read .....	120
表 3-123. 函数 dci_flag_get .....	121
表 3-124. 函数 dci_interrupt_enable .....	122
表 3-125. 函数 dci_interrupt_disable .....	122
表 3-126. 函数 dci_interrupt_flag_get .....	123

表 3-127. 函数 dci_interrupt_flag_clear .....	123
表 3-128. DMA 寄存器 .....	124
表 3-129. DMA 库函数 .....	125
表 3-130. 枚举类型 dma_channel_enum .....	126
表 3-131. 枚举类型 dma_subperipheral_enum .....	126
表 3-132. 结构体 dma_multi_data_parameter_struct .....	126
表 3-133. 结构体 dma_single_data_parameter_struct .....	127
表 3-134. 函数 dma_deinit .....	127
表 3-135. 函数 dma_single_data_para_struct_init .....	128
表 3-136. 函数 dma_multi_data_para_struct_init .....	128
表 3-137. 函数 dma_single_data_mode_init .....	129
表 3-138. 函数 dma_multi_data_mode_init .....	130
表 3-139. 函数 dma_periph_address_config .....	131
表 3-140. 函数 dma_memory_address_config .....	132
表 3-141. 函数 dma_transfer_number_config .....	132
表 3-142. 函数 dma_transfer_number_get .....	133
表 3-143. 函数 dma_priority_config .....	134
表 3-144. 函数 dma_memory_burst_beats_config .....	134
表 3-145. 函数 dma_periph_burst_beats_config .....	135
表 3-146. 函数 dma_memory_width_config .....	136
表 3-147. 函数 dma_periph_width_config .....	137
表 3-148. 函数 dma_memory_address_generation_config .....	138
表 3-149. 函数 dma_peripheral_address_generation_config .....	138
表 3-150. 函数 dma_circulation_enable .....	139
表 3-151. 函数 dma_circulation_disable .....	140
表 3-152. 函数 dma_channel_enable .....	140
表 3-153. 函数 dma_channel_disable .....	141
表 3-154. 函数 dma_transfer_direction_config .....	142
表 3-155. 函数 dma_switch_buffer_mode_config .....	142
表 3-156. 函数 dma_using_memory_get .....	143
表 3-157. 函数 dma_channel_subperipheral_select .....	144
表 3-158. 函数 dma_flow_controller_config .....	144
表 3-159. 函数 dma_switch_buffer_mode_enable .....	145
表 3-160. 函数 dma_switch_buffer_mode_disable .....	146
表 3-161. 函数 dma_fifo_status_get .....	146
表 3-162. 函数 dma_flag_get .....	147
表 3-163. 函数 dma_flag_clear .....	148
表 3-164. 函数 dma_interrupt_flag_get .....	149
表 3-165. 函数 dma_interrupt_flag_clear .....	149
表 3-166. 函数 dma_interrupt_enable .....	150
表 3-167. 函数 dma_interrupt_disable .....	151
表 3-168. 函数 dma_security_config .....	152
表 3-169. 函数 dma_security_config .....	153
表 3-170. 函数 dma_illegal_access_flag_get .....	154

表 3-171. 函数 dma_illegal_access_flag_clear .....	154
表 3-172. EFUSE 寄存器 .....	155
表 3-173. EFUSE 库函数 .....	156
表 3-174. 枚举类型 efuse_flag_enum .....	156
表 3-175. 枚举类型 efuse_clear_flag_enum .....	157
表 3-176. 枚举类型 efuse_int_enum .....	157
表 3-177. 枚举类型 efuse_int_flag_enum .....	157
表 3-178. 枚举类型 efuse_clear_int_flag_enum .....	157
表 3-179. 枚举类型 efuse_tz_enum .....	157
表 3-180. 函数 efuse_read .....	158
表 3-181. 函数 efuse_write .....	158
表 3-182. 函数 efuse_boot_config .....	159
表 3-183. 函数 efuse_tz_control_config .....	159
表 3-184. 函数 efuse_fp_config .....	160
表 3-185. 函数 efuse_mcu_init_data_write .....	161
表 3-186. 函数 efuse_aes_key_write .....	161
表 3-187. 函数 efuse_rotpk_key_write .....	162
表 3-188. 函数 efuse_dp_write .....	162
表 3-189. 函数 efuse_iak_write .....	163
表 3-190. 函数 efuse_user_data_write .....	164
表 3-191. 函数 efuse_software_trustzone_enable .....	164
表 3-192. 函数 efuse_software_trustzone_disable .....	165
表 3-193. 函数 efuse_boot_address_get .....	165
表 3-194. 函数 efuse_flag_get .....	166
表 3-195. 函数 efuse_flag_clear .....	166
表 3-196. 函数 efuse_interrupt_enable .....	167
表 3-197. 函数 efuse_interrupt_disable .....	167
表 3-198. 函数 efuse_interrupt_flag_get .....	168
表 3-199. 函数 efuse_interrupt_flag_clear .....	169
表 3-200. EXTI 寄存器 .....	169
表 3-201. EXTI 库函数 .....	170
表 3-202. 枚举类型 exti_line_enum .....	170
表 3-203. 枚举类型 exti_mode_enum .....	171
表 3-204. 枚举类型 exti_trig_type_enum .....	171
表 3-205. 函数 exti_deinit .....	171
表 3-206. 函数 exti_init .....	172
表 3-207. 函数 exti_interrupt_enable .....	173
表 3-208. 函数 exti_interrupt_disable .....	173
表 3-209. 函数 exti_event_enable .....	174
表 3-210. 函数 exti_event_disable .....	174
表 3-211. 函数 exti_security_enable .....	175
表 3-212. 函数 exti_security_disable .....	175
表 3-213. 函数 exti_privilege_enable .....	176
表 3-214. 函数 exti_privilege_disable .....	176

表 3-215. 函数 exti_lock_enable .....	177
表 3-216. 函数 exti_software_interrupt_enable .....	177
表 3-217. 函数 exti_software_interrupt_disable .....	178
表 3-218. 函数 exti_flag_get .....	178
表 3-219. 函数 exti_flag_clear .....	179
表 3-220. 函数 exti_interrupt_flag_get .....	179
表 3-221. 函数 exti_interrupt_flag_clear .....	180
表 3-222. FMC 寄存器 .....	180
表 3-223. FMC 固件库函数 .....	181
表 3-224. 枚举类型 fsmc_state_enum .....	182
表 3-225. 函数 fsmc_unlock .....	183
表 3-226. 函数 fsmc_lock .....	183
表 3-227. 函数 fsmc_page_erase .....	183
表 3-228. 函数 fsmc_mass_erase .....	184
表 3-229. 函数 fsmc_word_program .....	185
表 3-230. 函数 fsmc_continuous_program .....	185
表 3-231. 函数 sram1_reset_enable .....	186
表 3-232. 函数 sram1_reset_disable .....	186
表 3-233. 函数 fsmc_privilege_enable .....	187
表 3-234. 函数 fsmc_privilege_disable .....	187
表 3-235. 函数 ob_unlock .....	188
表 3-236. 函数 ob_lock .....	189
表 3-237. 函数 ob_start .....	189
表 3-238. 函数 ob_reload .....	190
表 3-239. 函数 ob_security_protection_config .....	190
表 3-240. 函数 ob_trustzone_enable .....	191
表 3-241. 函数 ob_trustzone_disable .....	191
表 3-242. 函数 ob_user_write .....	192
表 3-243. 函数 ob_write_protection_config .....	192
表 3-244. 函数 ob_secmark_config .....	193
表 3-245. 函数 ob_dmp_access_enable .....	194
表 3-246. 函数 ob_dmp_access_disable .....	194
表 3-247. 函数 ob_dmp_config .....	195
表 3-248. 函数 ob_dmp_enable .....	196
表 3-249. 函数 ob_dmp_disable .....	196
表 3-250. 函数 fsmc_no_rtdec_config .....	197
表 3-251. 函数 fsmc_offset_region_config .....	198
表 3-252. 函数 fsmc_offset_value_config .....	198
表 3-253. 函数 ob_write_protection_get .....	199
表 3-254. 函数 ob_user_get .....	199
表 3-255. 函数 ob_security_protection_flag_get .....	200
表 3-256. 函数 ob_trustzone_state_get .....	200
表 3-257. 函数 ob_memory_mode_state_get .....	201
表 3-258. 函数 ob_exist_state_get .....	201

表 3-259. 函数 fmc_flag_get.....	202
表 3-260. 函数 fmc_flag_clear.....	203
表 3-261. 函数 fmc_interrupt_enable .....	203
表 3-262. 函数 fmc_interrupt_disable .....	204
表 3-263. 函数 fmc_interrupt_flag_get.....	204
表 3-264. 函数 fmc_interrupt_flag_clear.....	205
表 3-265. FWDGT 寄存器.....	206
表 3-266. FWDGT 库函数.....	206
表 3-267. 函数 fwdgt_write_enable .....	206
表 3-268. 函数 fwdgt_write_disable .....	207
表 3-269. 函数 fwdgt_enable .....	207
表 3-270. 函数 fwdgt_prescaler_value_config .....	208
表 3-271. 函数 fwdgt_reload_value_config .....	208
表 3-272. 函数 fwdgt_counter_reload .....	209
表 3-273. 函数 fwdgt_config.....	209
表 3-274. 函数 fwdgt_flag_get.....	210
表 3-275. GPIO 寄存器 .....	211
表 3-276. GPIO 库函数 .....	211
表 3-277. 函数 gpio_deinit.....	212
表 3-278. 函数 gpio_mode_set.....	212
表 3-279. 函数 gpio_output_options_set.....	213
表 3-280. 函数 gpio_bit_set.....	214
表 3-281. 函数 gpio_bit_reset .....	215
表 3-282. 函数 gpio_bit_write.....	216
表 3-283. 函数 gpio_port_write.....	216
表 3-284. 函数 gpio_input_bit_get.....	217
表 3-285. 函数 gpio_input_port_get .....	217
表 3-286. 函数 gpio_output_bit_get .....	218
表 3-287. 函数 gpio_output_port_get.....	219
表 3-288. 函数 gpio_af_set.....	219
表 3-289. 函数 gpio_pin_lock.....	220
表 3-290. 函数 gpio_bit_toggle .....	221
表 3-291. 函数 gpio_port_toggle .....	221
表 3-292. 函数 gpio_bit_set_sec_cfg .....	222
表 3-293. 函数 gpio_bit_reset_sec_cfg.....	222
表 3-294. 函数 gpio_sec_cfg_bit_get .....	223
表 3-295. HAU 寄存器 .....	224
表 3-296. HAU 库函数 .....	224
表 3-297. 结构体 hau_init_parameter_struct .....	225
表 3-298. 结构体 hau_digest_parameter_struct.....	225
表 3-299. 结构体 hau_context_parameter_struct.....	226
表 3-300. 函数 hau_deinit.....	226
表 3-301. 函数 hau_init .....	226
表 3-302. 函数 hau_init_struct_para_init .....	227



表 3-303. 函数 hau_reset .....	228
表 3-304. 函数 hau_last_word_validbits_num_config .....	228
表 3-305. 函数 hau_data_write .....	229
表 3-306. 函数 hau_infifo_words_num_get .....	229
表 3-307. 函数 hau_digest_read .....	230
表 3-308. 函数 hau_digest_calculation_enable .....	230
表 3-309. 函数 hau_multiple_single_dma_config .....	231
表 3-310. 函数 hau_dma_enable .....	231
表 3-311. 函数 hau_dma_disable .....	232
表 3-312. 函数 hau_context_struct_para_init .....	232
表 3-313. 函数 hau_context_save .....	233
表 3-314. 函数 hau_context_restore .....	233
表 3-315. 函数 hau_hash_sha_1 .....	234
表 3-316. 函数 hau_hmac_sha_1 .....	234
表 3-317. 函数 hau_hash_sha_224 .....	235
表 3-318. 函数 hau_hmac_sha_224 .....	236
表 3-319. 函数 hau_hash_sha_256 .....	236
表 3-320. 函数 hau_hmac_sha_256 .....	237
表 3-321. 函数 hau_hash_md5 .....	238
表 3-322. 函数 hau_hmac_md5 .....	238
表 3-323. 函数 hau_flag_get .....	239
表 3-324. 函数 hau_flag_clear .....	240
表 3-325. 函数 hau_interrupt_enable .....	240
表 3-326. 函数 hau_interrupt_disable .....	241
表 3-327. 函数 hau_interrupt_flag_get .....	241
表 3-328. 函数 hau_interrupt_flag_clear .....	242
表 3-329. HPDF 寄存器 .....	243
表 3-330. HPDF 库函数 .....	243
表 3-331. 枚举类型 hpdf_channel_enum .....	246
表 3-332. 枚举类型 hpdf_filter_enum .....	246
表 3-333. 枚举类型 hpdf_flag_enum .....	246
表 3-334. 枚举类型 hpdf_interrput_flag_enum .....	246
表 3-335. 枚举类型 hpdf_interrput_enum .....	247
表 3-336. 结构体 hpdf_channel_parameter_struct .....	247
表 3-337. 结构体 hpdf_filter_parameter_struct .....	247
表 3-338. 结构体 hpdf_rc_parameter_struct .....	248
表 3-339. 结构体 hpdf_ic_parameter_struct .....	248
表 3-340. 函数 hpdf_deinit .....	248
表 3-341. 函数 hpdf_channel_struct_para_init .....	249
表 3-342. 函数 hpdf_filter_struct_para_init .....	249
表 3-343. 函数 hpdf_rc_struct_para_init .....	250
表 3-344. 函数 hpdf_ic_struct_para_init .....	250
表 3-345. 函数 hpdf_enable .....	251
表 3-346. 函数 hpdf_disable .....	251

表 3-347. 函数 hpdf_channel_init .....	252
表 3-348. 函数 hpdf_filter_init .....	253
表 3-349. 函数 hpdf_rc_init .....	254
表 3-350. 函数 hpdf_ic_init .....	255
表 3-351. 函数 hpdf_clock_output_config .....	255
表 3-352. 函数 hpdf_clock_output_source_config .....	256
表 3-353. 函数 hpdf_clock_output_duty_mode_disable .....	257
表 3-354. 函数 hpdf_clock_output_duty_mode_enable .....	257
表 3-355. 函数 hpdf_clock_output_divider_config .....	257
表 3-356. 函数 hpdf_channel_enable .....	258
表 3-357. 函数 hpdf_channel_disable .....	258
表 3-358. 函数 hpdf_spi_clock_source_config .....	259
表 3-359. 函数 hpdf_serial_interface_type_config .....	260
表 3-360. 函数 hpdf_malfunction_monitor_disable .....	260
表 3-361. 函数 hpdf_malfunction_monitor_enable .....	261
表 3-362. 函数 hpdf_clock_loss_disable .....	261
表 3-363. 函数 hpdf_clock_loss_enable .....	262
表 3-364. 函数 hpdf_channel_pin_redirection_disable .....	262
表 3-365. 函数 hpdf_channel_pin_redirection_enable .....	263
表 3-366. 函数 hpdf_channel_multiplexer_config .....	263
表 3-367. 函数 hpdf_data_pack_mode_config .....	264
表 3-368. 函数 hpdf_data_right_bit_shift_config .....	265
表 3-369. 函数 hpdf_calibration_offset_config .....	265
表 3-370. 函数 hpdf_malfunction_break_signal_config .....	266
表 3-371. 函数 hpdf_malfunction_counter_config .....	266
表 3-372. 函数 hpdf_write_parallel_data_standard_mode .....	267
表 3-373. 函数 hpdf_write_parallel_data_interleaved_mode .....	268
表 3-374. 函数 hpdf_write_parallel_data_dual_mode .....	268
表 3-375. 函数 hpdf_pulse_skip_update .....	269
表 3-376. 函数 hpdf_pulse_skip_read .....	269
表 3-377. 函数 hpdf_filter_enable .....	270
表 3-378. 函数 hpdf_filter_disable .....	270
表 3-379. 函数 hpdf_filter_config .....	271
表 3-380. 函数 hpdf_integrator_oversample .....	272
表 3-381. 函数 hpdf_threshold_monitor_filter_config .....	272
表 3-382. 函数 hpdf_threshold_monitor_filter_read_data .....	273
表 3-383. 函数 hpdf_threshold_monitor_fast_mode_disable .....	273
表 3-384. 函数 hpdf_threshold_monitor_fast_mode_enable .....	274
表 3-385. 函数 hpdf_threshold_monitor_channel .....	274
表 3-386. 函数 hpdf_threshold_monitor_high_threshold .....	275
表 3-387. 函数 hpdf_threshold_monitor_low_threshold .....	276
表 3-388. 函数 hpdf_high_threshold_break_signal .....	276
表 3-389. 函数 hpdf_low_threshold_break_signal .....	277
表 3-390. 函数 hpdf_extremes_monitor_channel .....	278

表 3-391. 函数 hpdf_extremes_monitor_maximum_get .....	278
表 3-392. 函数 hpdf_extremes_monitor_minimum_get .....	279
表 3-393. 函数 hpdf_rc_continuous_disable .....	279
表 3-394. 函数 hpdf_rc_continuous_enable .....	280
表 3-395. 函数 hpdf_rc_start_by_software .....	280
表 3-396. 函数 hpdf_rc_syn_disable .....	281
表 3-397. 函数 hpdf_rc_syn_disable .....	281
表 3-398. 函数 hpdf_rc_dma_disable .....	282
表 3-399. 函数 hpdf_rc_dma_enable .....	282
表 3-400. 函数 hpdf_rc_channel_config .....	283
表 3-401. 函数 hpdf_rc_fast_mode_disable .....	283
表 3-402. 函数 hpdf_rc_fast_mode_enable .....	284
表 3-403. 函数 hpdf_rc_data_get .....	284
表 3-404. 函数 hpdf_rc_channel_get .....	285
表 3-405. 函数 hpdf_ic_start_by_software .....	286
表 3-406. 函数 hpdf_ic_syn_disable .....	286
表 3-407. 函数 hpdf_ic_syn_enable .....	287
表 3-408. 函数 hpdf_ic_dma_disable .....	287
表 3-409. 函数 hpdf_ic_dma_enable .....	288
表 3-410. 函数 hpdf_ic_scan_mode_disable .....	288
表 3-411. 函数 hpdf_ic_scan_mode_enable .....	289
表 3-412. 函数 hpdf_ic_trigger_signal_disbale .....	289
表 3-413. 函数 hpdf_ic_trigger_signal_config .....	290
表 3-414. 函数 hpdf_ic_channel_config .....	290
表 3-415. 函数 hpdf_ic_data_get .....	291
表 3-416. 函数 hpdf_ic_channel_get .....	292
表 3-417. 函数 hpdf_flag_get .....	292
表 3-418. 函数 hpdf_flag_clear .....	293
表 3-419. 函数 hpdf_interrupt_enable .....	295
表 3-420. 函数 hpdf_interrupt_disable .....	295
表 3-421. 函数 hpdf_interrupt_flag_get .....	296
表 3-422. 函数 hpdf_interrupt_flag_clear .....	297
表 3-423. I2C 寄存器 .....	299
表 3-424. I2C 库函数 .....	299
表 3-425. 枚举类型 i2c_interrupt_flag_enum .....	301
表 3-426. 函数 i2c_deinit .....	301
表 3-427. 函数 i2c_timing_config .....	302
表 3-428. 函数 i2c_digital_noise_filter_config .....	302
表 3-429. 函数 i2c_analog_noise_filter_enable .....	303
表 3-430. 函数 i2c_analog_noise_filter_disable .....	304
表 3-431. 函数 i2c_master_clock_config .....	304
表 3-432. 函数 i2c_master_addressing .....	305
表 3-433. 函数 i2c_address10_header_enable .....	305
表 3-434. 函数 i2c_address10_header_disable .....	306

表 3-435. 函数 i2c_address10_enable.....	306
表 3-436. 函数 i2c_address10_disable.....	307
表 3-437. 函数 i2c_automatic_end_enable.....	307
表 3-438. 函数 i2c_automatic_end_disable.....	308
表 3-439. 函数 i2c_slave_response_to_gcall_enable.....	308
表 3-440. 函数 i2c_slave_response_to_gcall_disable.....	309
表 3-441. 函数 i2c_stretch_scl_low_enable.....	309
表 3-442. 函数 i2c_stretch_scl_low_disable.....	310
表 3-443. 函数 i2c_address_config.....	310
表 3-444. 函数 i2c_address_bit_compare_config.....	311
表 3-445. 函数 i2c_address_disable.....	312
表 3-446. 函数 i2c_second_address_config.....	313
表 3-447. 函数 i2c_second_address_disable.....	314
表 3-448. 函数 i2c_receved_address_get.....	314
表 3-449. 函数 i2c_slave_byte_control_enable.....	315
表 3-450. 函数 i2c_slave_byte_control_disable.....	315
表 3-451. 函数 i2c_nack_enable.....	316
表 3-452. 函数 i2c_nack_disable.....	316
表 3-453. 函数 i2c_wakeup_from_deepsleep_enable.....	317
表 3-454. 函数 i2c_wakeup_from_deepsleep_disable.....	317
表 3-455. 函数 i2c_enable.....	318
表 3-456. 函数 i2c_disable.....	318
表 3-457. 函数 i2c_start_on_bus.....	319
表 3-458. 函数 i2c_stop_on_bus.....	319
表 3-459. 函数 i2c_data_transmit.....	320
表 3-460. 函数 i2c_data_receive.....	320
表 3-461. 函数 i2c_reload_enable.....	321
表 3-462. 函数 i2c_reload_disable.....	321
表 3-463. 函数 i2c_transfer_byte_number_config.....	322
表 3-464. 函数 i2c_dma_enable.....	322
表 3-465. 函数 i2c_dma_disable.....	323
表 3-466. 函数 i2c_pec_transfer.....	324
表 3-467. 函数 i2c_pec_enable.....	324
表 3-468. 函数 i2c_pec_disable.....	325
表 3-469. 函数 i2c_pec_value_get.....	325
表 3-470. 函数 i2c_smbus_alert_enable.....	326
表 3-471. 函数 i2c_smbus_alert_disable.....	326
表 3-472. 函数 i2c_smbus_default_addr_enable.....	327
表 3-473. 函数 i2c_smbus_default_addr_disable.....	327
表 3-474. 函数 i2c_smbus_host_addr_enable.....	328
表 3-475. 函数 i2c_smbus_host_addr_disable.....	328
表 3-476. 函数 i2c_extented_clock_timeout_enable.....	329
表 3-477. 函数 i2c_extented_clock_timeout_disable.....	329
表 3-478. 函数 i2c_clock_timeout_enable.....	330

表 3-479. 函数 i2c_clock_timeout_disable .....	330
表 3-480. 函数 i2c_bus_timeout_b_config .....	331
表 3-481. 函数 i2c_bus_timeout_a_config.....	331
表 3-482. 函数 i2c_idle_clock_timeout_config.....	332
表 3-483. 函数 i2c_flag_get .....	332
表 3-484. 函数 i2c_flag_clear .....	333
表 3-485. 函数 i2c_interrupt_enable.....	334
表 3-486. 函数 i2c_interrupt_disable.....	335
表 3-487. 函数 i2c_interrupt_flag_get .....	336
表 3-488. 函数 i2c_interrupt_flag_clear .....	337
表 3-489. ICAHCE 寄存器 .....	338
表 3-490. ICACHE 库函数 .....	338
表 3-491. 结构体 icache_remap_struct .....	339
表 3-492. 函数 icache_enable .....	339
表 3-493. 函数 icache_disable .....	339
表 3-494. 函数 icache_monitor_enable.....	340
表 3-495. 函数 icache_monitor_disable.....	341
表 3-496. 函数 icache_monitor_reset.....	341
表 3-497. 函数 icache_way_configure .....	342
表 3-498. 函数 icache_burst_type_select.....	342
表 3-499. 函数 icache_invalidation.....	343
表 3-500. 函数 icache_hitvalue_get.....	343
表 3-501. 函数 icache_missvalue_get.....	344
表 3-502. 函数 icache_remap_enable .....	344
表 3-503. 函数 icache_remap_disable .....	345
表 3-504. 函数 icache_flag_get.....	346
表 3-505. 函数 icache_flag_clear .....	346
表 3-506. 函数 icache_interrupt_enable .....	347
表 3-507. 函数 icache_interrupt_disable .....	347
表 3-508. 函数 icache_flag_get.....	348
表 3-509. 函数 icache_interrupt_flag_clear .....	348
表 3-510. NVIC 寄存器.....	349
表 3-511. SysTick 寄存器.....	349
表 3-512. 枚举类型 IRQn_Type .....	350
表 3-513. MISC 库函数 .....	352
表 3-514. 函数 nvic_priority_group_set.....	352
表 3-515. 函数 nvic_irq_enable.....	353
表 3-516. 函数 nvic_irq_disable.....	353
表 3-517. 函数 nvic_vector_table_set .....	354
表 3-518. 函数 system_lowpower_set.....	355
表 3-519. 函数 system_lowpower_reset .....	356
表 3-520. 函数 systick_clksource_set.....	356
表 3-521. PKCAU 寄存器 .....	357
表 3-522. PKCAU 库函数 .....	357

表 3-523. 结构体 pkcau_mont_parameter_struct .....	358
表 3-524. 结构体 pkcau_mod_parameter_struct.....	358
表 3-525. 结构体 pkcau_mod_exp_parameter_struct .....	358
表 3-526. 结构体 pkcau_mod_inver_parameter_struct .....	359
表 3-527. 结构体 pkcau_mod_reduc_parameter_struct.....	359
表 3-528. 结构体 pkcau_arithmetic_parameter_struct .....	359
表 3-529. 结构体 pkcau crt_parameter_struct.....	359
表 3-530. 结构体 pkcau_ec_group_parameter_struct .....	360
表 3-531. 结构体 pkcau_point_parameter_struct .....	360
表 3-532. 结构体 pkcau_signature_parameter_struct .....	360
表 3-533. 结构体 pkcau_hash_parameter_struct.....	360
表 3-534. 函数 pkcau_deinit.....	360
表 3-535. 函数 pkcau_mont_struct_para_init.....	361
表 3-536. 函数 pkcau_mod_struct_para_init.....	362
表 3-537. 函数 pkcau_mod_exp_struct_para_init.....	362
表 3-538. 函数 pkcau_mod_inver_struct_para_init .....	363
表 3-539. 函数 pkcau_arithmetic_struct_para_init .....	363
表 3-540. 函数 pkcau crt_struct_para_init.....	364
表 3-541. 函数 pkcau_ec_group_struct_para_init .....	364
表 3-542. 函数 pkcau_point_struct_para_init.....	365
表 3-543. 函数 pkcau_signature_struct_para_init .....	365
表 3-544. 函数 pkcau_hash_struct_para_init .....	366
表 3-545. 函数 pkcau_mod_reduc_struct_para_init.....	367
表 3-546. 函数 pkcau_enable .....	367
表 3-547. 函数 pkcau_disable .....	368
表 3-548. 函数 pkcau_start.....	368
表 3-549. 函数 pkcau_mode_set.....	369
表 3-550. 函数 pkcau_mont_param_operation .....	370
表 3-551. 函数 pkcau_mod_operation .....	371
表 3-552. 函数 pkcau_mod_exp_operation .....	372
表 3-553. 函数 pkcau_mod_inver_operation.....	373
表 3-554. 函数 pkcau_mod_reduc_operation.....	373
表 3-555. 函数 pkcau_arithmetic_operation.....	374
表 3-556. 函数 pkcau crt_exp_operation .....	375
表 3-557. 函数 pkcau_point_check_operation .....	376
表 3-558. 函数 pkcau_point_mul_operation.....	377
表 3-559. 函数 pkcau_ecdsa_sign_operation .....	379
表 3-560. 函数 pkcau_ecdsa_verification_operation .....	380
表 3-561. 函数 pkcau_memread .....	382
表 3-562. 函数 pkcau_flag_get.....	382
表 3-563. 函数 pkcau_flag_clear.....	383
表 3-564. 函数 pkcau_interrupt_enable .....	383
表 3-565. 函数 pkcau_interrupt_disable .....	384
表 3-566. 函数 pkcau_interrupt_flag_get.....	384



表 3-567. 函数 pkcau_interrupt_flag_clear .....	385
表 3-568. PMU 寄存器 .....	386
表 3-569. PMU 库函数 .....	386
表 3-570. 函数 pmu_deinit .....	387
表 3-571. 函数 pmu_lvd_select .....	387
表 3-572. 函数 pmu_lvd_disable .....	388
表 3-573. 函数 pmu_vlvd_enable .....	389
表 3-574. 函数 pmu_vlvd_disable .....	389
表 3-575. 函数 pmu_ldo_output_select .....	390
表 3-576. 函数 pmu_to_sleepmode .....	390
表 3-577. 函数 pmu_to_deepsleepmode .....	391
表 3-578. 函数 pmu_to_standbymode .....	392
表 3-579. 函数 pmu_wakeup_pin_enable .....	392
表 3-580. 函数 pmu_wakeup_pin_disable .....	393
表 3-581. 函数 pmu_backup_write_enable .....	393
表 3-582. 函数 pmu_backup_write_disable .....	394
表 3-583. 函数 pmu_wifi_power_enable .....	394
表 3-584. 函数 pmu_wifi_power_disable .....	395
表 3-585. 函数 pmu_wifi_sram_control .....	395
表 3-586. 函数 pmu_rf_force_enable .....	396
表 3-587. 函数 pmu_rf_force_disable .....	397
表 3-588. 函数 pmu_rf_sequence_config .....	397
表 3-589. 函数 pmu_security_enable .....	398
表 3-590. 函数 pmu_security_disable .....	399
表 3-591. 函数 pmu_privilege_enable .....	400
表 3-592. 函数 pmu_privilege_disable .....	400
表 3-593. 函数 pmu_flag_reset .....	400
表 3-594. 函数 pmu_flag_get .....	401
表 3-595. SPI/I2S 寄存器 .....	403
表 3-596. QSPI 库函数 .....	404
表 3-597. 结构体 qspi_init_struct .....	404
表 3-598. 结构体 qspi_command_struct .....	405
表 3-599. 结构体 qspi_autopolling_struct .....	406
表 3-600. 函数 qspi_deinit .....	406
表 3-601. 函数 qspi_init_struct_para_init .....	406
表 3-602. 函数 qspi_init .....	407
表 3-603. 函数 qspi_enable .....	408
表 3-604. 函数 qspi_disable .....	408
表 3-605. 函数 qspi_dma_enable .....	409
表 3-606. 函数 qspi_dma_disable .....	409
表 3-607. 函数 qspi_command .....	410
表 3-608. 函数 qspi_transmit .....	411
表 3-609. 函数 qspi_receive .....	411
表 3-610. 函数 qspi_autopolling .....	412

表 3-611. 函数 qspi_memorymapped .....	413
表 3-612. 函数 qspi_abort.....	414
表 3-613. 函数 qspi_command_fmc_s .....	415
表 3-614. 函数 qspi_transmit_fmc_s .....	415
表 3-615. 函数 qspi_receive_fmc_s.....	416
表 3-616. 函数 qspi_autopolling_fmc_s.....	416
表 3-617. 函数 qspi_memorymapped_fmc_s .....	418
表 3-618. 函数 qspi_interrupt_enable .....	419
表 3-619. 函数 qspi_interrupt_disable .....	420
表 3-620. 函数 qspi_flag_get.....	420
表 3-621. 函数 qspi_flag_clear .....	421
表 3-622. RCU 寄存器 .....	422
表 3-623. RCU 库函数 .....	423
表 3-624. 枚举类型 rcu_periph_enum .....	425
表 3-625. 枚举类型 rcu_periph_sleep_enum .....	426
表 3-626. 枚举类型 rcu_periph_reset_enum.....	427
表 3-627. 枚举类型 rcu_flag_enum .....	428
表 3-628. 枚举类型 rcu_int_flag_enum.....	429
表 3-629. 枚举类型 rcu_int_flag_clear_enum .....	429
表 3-630. 枚举类型 rcu_int_enum .....	429
表 3-631. 枚举类型 rcu_osci_type_enum.....	430
表 3-632. 枚举类型 rcu_clock_freq_enum .....	430
表 3-633. 枚举类型 rcu_sec_enum .....	430
表 3-634. 枚举类型 rcu_sec_flag_enum .....	431
表 3-635. 枚举类型 rcu_unit_enum .....	432
表 3-636. 函数 rcu_deinit.....	432
表 3-637. 函数 rcu_periph_clock_enable.....	433
表 3-638. 函数 rcu_periph_clock_disable.....	433
表 3-639. 函数 rcu_periph_clock_sleep_enable .....	434
表 3-640. 函数 rcu_periph_clock_sleep_disable .....	434
表 3-641. 函数 rcu_periph_reset_enable .....	435
表 3-642. 函数 rcu_periph_reset_disable .....	435
表 3-643. 函数 rcu_bkp_reset_enable.....	436
表 3-644. 函数 rcu_bkp_reset_disable.....	436
表 3-645. 函数 rcu_hxtal_plli2s_enable .....	437
表 3-646. 函数 rcu_hxtal_plli2s_disable .....	437
表 3-647. 函数 rcu_control_unit_powerup .....	438
表 3-648. 函数 rcu_control_unit_powerdown .....	438
表 3-649. 函数 rcu_rfppl_cal_enable .....	439
表 3-650. 函数 rcu_rfppl_cal_disable .....	439
表 3-651. 函数 rcu_system_clock_source_config .....	440
表 3-652. 函数 rcu_system_clock_source_get .....	441
表 3-653. 函数 rcu_ahb_clock_config.....	441
表 3-654. 函数 rcu_apb1_clock_config.....	442



表 3-655. 函数 rcu_apb2_clock_config .....	442
表 3-656. 函数 rcu_ckout0_config .....	443
表 3-657. 函数 rcu_ckout1_config .....	444
表 3-658. 函数 rcu_pll_config .....	445
表 3-659. 函数 rcu_plli2s_config .....	445
表 3-660. 函数 rcu_plldig_config .....	446
表 3-661. 函数 rcu_plldig_div_sys_config .....	447
表 3-662. 函数 rcu_rtc_clock_config .....	447
表 3-663. 函数 rcu_rtc_div_config .....	448
表 3-664. 函数 rcu_i2s_clock_config .....	448
表 3-665. 函数 rcu_pllfi2s_clock_div_config .....	449
表 3-666. 函数 rcu_pllfi2s_clock_div_config .....	450
表 3-667. 函数 rcu_hpdc_audio_clock_config .....	450
表 3-668. 函数 rcu_sdio_clock_config .....	451
表 3-669. 函数 rcu_sdio_div_config .....	452
表 3-670. 函数 rcu_usbfs_clock_config .....	452
表 3-671. 函数 rcu_usbfs_div_config .....	453
表 3-672. 函数 rcu_i2c0_clock_config .....	453
表 3-673. 函数 usart0_clock_source .....	454
表 3-674. 函数 usart2_clock_source .....	454
表 3-675. 函数 rcu_irc16m_div_config .....	455
表 3-676. 函数 rcu_timer_clock_prescaler_config .....	456
表 3-677. 函数 rcu_lxtal_drive_capability_config .....	456
表 3-678. 函数 rcu_osci_stab_wait .....	457
表 3-679. 函数 rcu_osci_on .....	457
表 3-680. 函数 rcu_osci_off .....	458
表 3-681. 函数 rcu_osci_bypass_mode_enable .....	458
表 3-682. 函数 rcu_osci_bypass_mode_disable .....	459
表 3-683. 函数 rcu_irc16m_adjust_value_set .....	459
表 3-684. 函数 rcu_spread_spectrum_config .....	460
表 3-685. 函数 rcu_spread_spectrum_enable .....	461
表 3-686. 函数 rcu_spread_spectrum_disable .....	461
表 3-687. 函数 rcu_hxtal_clock_monitor_enable .....	462
表 3-688. 函数 rcu_hxtal_clock_monitor_disable .....	462
表 3-689. 函数 rcu_rf_hxtal_clock_monitor_enable .....	463
表 3-690. 函数 rcu_rf_hxtal_clock_monitor_disable .....	463
表 3-691. 函数 rcu_voltage_key_unlock .....	464
表 3-692. 函数 rcu_deepsleep_voltage_set .....	464
表 3-693. 函数 rcu_clock_freq_get .....	465
表 3-694. 函数 rcu_security_enable .....	465
表 3-695. 函数 rcu_security_disable .....	466
表 3-696. 函数 rcu_privilege_enable .....	466
表 3-697. 函数 rcu_privilege_disable .....	467
表 3-698. 函数 rcu_flag_get .....	467

表 3-699. 函数 rcu_all_reset_flag_clear .....	468
表 3-700. 函数 rcu_interrupt_flag_get.....	468
表 3-701. 函数 rcu_interrupt_flag_clear.....	469
表 3-702. 函数 rcu_security_flag_get.....	469
表 3-703. 函数 rcu_interrupt_enable .....	470
表 3-704. 函数 rcu_interrupt_disable .....	470
表 3-705. RTC 寄存器.....	471
表 3-706. RTC 库函数.....	472
表 3-707. 结构体 rtc_parameter_struct .....	473
表 3-708. 结构体 rtc_alarm_struct .....	474
表 3-709. 结构体 rtc_timestamp_struct.....	474
表 3-710. 结构体 rtc_tamper_struct.....	474
表 3-711. 函数 rtc_deinit.....	475
表 3-712. 函数 rtc_init .....	475
表 3-713. 函数 rtc_init_mode_enter .....	476
表 3-714. 函数 rtc_init_mode_exit .....	476
表 3-715. 函数 rtc_register_sync_wait.....	477
表 3-716. 函数 rtc_current_time_get .....	477
表 3-717. 函数 rtc_subsecond_get.....	478
表 3-718. 函数 rtc_alarm_config.....	478
表 3-719. 函数 rtc_alarm_subsecond_config.....	479
表 3-720. 函数 rtc_alarm_get .....	480
表 3-721. 函数 rtc_alarm_subsecond_get .....	481
表 3-722. 函数 rtc_alarm_enable .....	481
表 3-723. 函数 rtc_alarm_disable .....	482
表 3-724. 函数 rtc_timestamp_enable.....	482
表 3-725. 函数 rtc_timestamp_disable.....	483
表 3-726. 函数 rtc_timestamp_get.....	483
表 3-727. 函数 rtc_timestamp_subsecond_get.....	484
表 3-728. 函数 rtc_timestamp_enable.....	484
表 3-729. 函数 rtc_tamper_disable.....	485
表 3-730. 函数 rtc_software_bkp_reset .....	485
表 3-731. 函数 rtc_tamper_without_bkp_seset.....	486
表 3-732. 函数 rtc_interrupt_enable .....	486
表 3-733. 函数 rtc_interrupt_disable .....	487
表 3-734. 函数 rtc_flag_get.....	488
表 3-735. 函数 rtc_flag_clear.....	488
表 3-736. 函数 rtc_nsec_interrupt_flag_get .....	489
表 3-737. 函数 rtc_nsec_interrupt_flag_clear .....	490
表 3-738. 函数 rtc_sec_interrupt_flag_get.....	491
表 3-739. 函数 rtc_sec_interrupt_flag_clear.....	492
表 3-740. 函数 rtc_output_pad_select .....	492
表 3-741. 函数 rtc_alarm_output_config.....	493
表 3-742. 函数 rtc_calibration_output_config.....	494

表 3-743. 函数 rtc_hour_adjust.....	494
表 3-744. 函数 rtc_second_adjust .....	495
表 3-745. 函数 rtc_bypass_shadow_enable .....	495
表 3-746. 函数 rtc_bypass_shadow_disable .....	496
表 3-747. 函数 rtc_refclock_detection_enable .....	496
表 3-748. 函数 rtc_refclock_detection_disable .....	497
表 3-749. 函数 rtc_wakeup_enable.....	497
表 3-750. 函数 rtc_wakeup_disable.....	498
表 3-751. 函数 rtc_wakeup_clock_set.....	498
表 3-752. 函数 rtc_wakeup_timer_set .....	499
表 3-753. 函数 rtc_wakeup_timer_get.....	499
表 3-754. 函数 rtc_smooth_calibration_config .....	500
表 3-755. 函数 rtc_coarse_calibration_enable .....	501
表 3-756. 函数 rtc_coarse_calibration_disable .....	501
表 3-757. 函数 rtc_coarse_calibration_config.....	502
表 3-758. 函数 rtc_pri_pro_enable .....	503
表 3-759. 函数 rtc_pri_pro_disable .....	504
表 3-760. 函数 rtc_sec_pro_enable .....	505
表 3-761. 函数 rtc_sec_pro_disable .....	505
表 3-762. 函数 rtc_bkp_zonea_sec_pro_set.....	506
表 3-763. 函数 rtc_bkp_zoneb_sec_pro_set .....	507
表 3-764. 函数 rtc_bkp_zoneb_sec_pro_check .....	507
表 3-765. SDIO 寄存器 .....	508
表 3-766. SDIO 库函数 .....	509
表 3-767. 函数 sdio_deinit .....	510
表 3-768. 函数 sdio_clock_config .....	510
表 3-769. 函数 sdio_hardware_clock_enable.....	511
表 3-770. 函数 sdio_hardware_clock_disable.....	512
表 3-771. 函数 sdio_bus_mode_set .....	512
表 3-772. 函数 sdio_power_state_set .....	513
表 3-773. 函数 sdio_power_state_get .....	513
表 3-774. 函数 sdio_clock_enable .....	514
表 3-775. 函数 sdio_clock_disable.....	514
表 3-776. 函数 sdio_command_response_config.....	515
表 3-777. 函数 sdio_wait_type_set.....	516
表 3-778. 函数 sdio_csm_enable .....	516
表 3-779. 函数 sdio_csm_disable.....	517
表 3-780. 函数 sdio_command_index_get.....	517
表 3-781. 函数 sdio_response_get .....	518
表 3-782. 函数 sdio_data_config .....	518
表 3-783. 函数 sdio_data_transfer_config.....	520
表 3-784. 函数 sdio_dsm_enable.....	520
表 3-785. 函数 sdio_dsm_disable.....	521
表 3-786. 函数 sdio_data_write.....	521

表 3-787. 函数 <code>sdio_data_read</code> .....	522
表 3-788. 函数 <code>sdio_data_counter_get</code> .....	522
表 3-789. 函数 <code>sdio_data_counter_get</code> .....	523
表 3-790. 函数 <code>sdio_dma_enable</code> .....	523
表 3-791. 函数 <code>sdio_dma_disable</code> .....	524
表 3-792. 函数 <code>sdio_flag_get</code> .....	524
表 3-793. 函数 <code>sdio_flag_clear</code> .....	526
表 3-794. 函数 <code>sdio_interrupt_enable</code> .....	527
表 3-795. 函数 <code>sdio_interrupt_disable</code> .....	528
表 3-796. 函数 <code>sdio_interrupt_flag_get</code> .....	530
表 3-797. 函数 <code>sdio_interrupt_flag_clear</code> .....	531
表 3-798. 函数 <code>sdio_readwait_enable</code> .....	532
表 3-799. 函数 <code>sdio_readwait_disable</code> .....	533
表 3-800. 函数 <code>sdio_stop_readwait_enable</code> .....	533
表 3-801. 函数 <code>sdio_stop_readwait_disable</code> .....	534
表 3-802. 函数 <code>sdio_readwait_type_set</code> .....	534
表 3-803. 函数 <code>sdio_operation_enable</code> .....	535
表 3-804. 函数 <code>sdio_operation_disable</code> .....	535
表 3-805. 函数 <code>sdio_suspend_enable</code> .....	536
表 3-806. 函数 <code>sdio_suspend_disable</code> .....	536
表 3-807. 函数 <code>sdio_ceata_command_enable</code> .....	537
表 3-808. 函数 <code>sdio_ceata_command_disable</code> .....	537
表 3-809. 函数 <code>sdio_ceata_interrupt_enable</code> .....	538
表 3-810. 函数 <code>sdio_ceata_interrupt_disable</code> .....	538
表 3-811. 函数 <code>sdio_ceata_command_completion_enable</code> .....	539
表 3-812. 函数 <code>sdio_ceata_command_completion_disable</code> .....	539
表 3-813. SPI/I2S 寄存器 .....	540
表 3-814. SPI/I2S 库函数 .....	540
表 3-815. 结构体 <code>spi_parameter_struct</code> .....	541
表 3-816. 函数 <code>spi_i2s_deinit</code> .....	542
表 3-817. 函数 <code>spi_struct_para_init</code> .....	542
表 3-818. 函数 <code>spi_init</code> .....	543
表 3-819. 函数 <code>spi_enable</code> .....	544
表 3-820. 函数 <code>spi_disable</code> .....	544
表 3-821. 函数 <code>i2s_init</code> .....	545
表 3-822. 函数 <code>i2s_psc_config</code> .....	546
表 3-823. 函数 <code>i2s_ckin_psc_config</code> .....	547
表 3-824. 函数 <code>i2s_enable</code> .....	548
表 3-825. 函数 <code>i2s_disable</code> .....	549
表 3-826. 函数 <code>spi_nss_output_enable</code> .....	550
表 3-827. 函数 <code>spi_nss_output_disable</code> .....	550
表 3-828. 函数 <code>spi_nss_internal_high</code> .....	551
表 3-829. 函数 <code>spi_nss_internal_low</code> .....	551
表 3-830. 函数 <code>spi_dma_enable</code> .....	552

表 3-831. 函数 spi_dma_disable .....	552
表 3-832. 函数 spi_i2s_data_frame_format_config .....	553
表 3-833. 函数 spi_i2s_data_transmit .....	553
表 3-834. 函数 spi_i2s_data_receive .....	554
表 3-835. 函数 spi_bidirectional_transfer_config .....	555
表 3-836. 函数 i2s_full_duplex_mode_config .....	555
表 3-837. 函数 spi_i2s_format_error_clear .....	557
表 3-838. 函数 spi_crc_polynomial_set .....	557
表 3-839. 函数 spi_crc_polynomial_get .....	558
表 3-840. 函数 spi_crc_on .....	558
表 3-841. 函数 spi_crc_off .....	559
表 3-842. 函数 spi_crc_next .....	559
表 3-843. 函数 spi_crc_get .....	560
表 3-844. 函数 spi_crc_error_clear .....	560
表 3-845. 函数 spi_ti_mode_enable .....	561
表 3-846. 函数 spi_ti_mode_disable .....	561
表 3-847. 函数 spi_quad_enable .....	562
表 3-848. 函数 spi_quad_disable .....	562
表 3-849. 函数 spi_quad_write_enable .....	563
表 3-850. 函数 spi_quad_read_enable .....	563
表 3-851. 函数 spi_quad_io23_output_enable .....	564
表 3-852. 函数 spi_quad_io23_output_disable .....	564
表 3-853. 函数 spi_i2s_flag_get .....	565
表 3-854. 函数 spi_i2s_interrupt_enable .....	566
表 3-855. 函数 spi_i2s_interrupt_disable .....	567
表 3-856. 函数 spi_i2s_interrupt_flag_get .....	567
表 3-857. SQPI 寄存器 .....	568
表 3-858. SQPI 库函数 .....	569
表 3-859. 结构体 sqpi_parameter_struct .....	569
表 3-860. 函数 sqpi_deinit .....	569
表 3-861. 函数 sqpi_struct_para_init .....	570
表 3-862. 函数 sqpi_init .....	570
表 3-863. 函数 sqpi_read_id_command .....	571
表 3-864. 函数 sqpi_special_command .....	571
表 3-865. 函数 sqpi_read_command_config .....	572
表 3-866. 函数 sqpi_write_command_config .....	573
表 3-867. 函数 sqpi_low_id_receive .....	573
表 3-868. 函数 sqpi_high_id_receive .....	574
表 3-869. SYSCFG 寄存器 .....	575
表 3-870. SYSCFG 库函数 .....	575
表 3-871. 函数 syscfg_deinit .....	576
表 3-872. 函数 syscfg_exti_line_config .....	576
表 3-873. 函数 compensation_pwdn_mode_enable .....	577
表 3-874. 函数 compensation_pwdn_mode_disable .....	577

表 3-875. 函数 syscfg_clock_access_security_config .....	578
表 3-876. 函数 classb_access_security_config .....	579
表 3-877. 函数 sram1_access_security_config .....	579
表 3-878. 函数 fpu_access_security_config .....	580
表 3-879. 函数 vtor_ns_write_disable .....	580
表 3-880. 函数 mpu_ns_write_disable .....	581
表 3-881. 函数 vtors_aircr_write_disable .....	581
表 3-882. 函数 vtors_aircr_write_disable .....	582
表 3-883. 函数 sau_write_disable .....	582
表 3-884. 函数 syscfg_lock_config .....	583
表 3-885. 函数 gssacmd_write_data .....	583
表 3-886. 函数 sram1_erase .....	584
表 3-887. 函数 sram1_unlock .....	584
表 3-888. 函数 sram1_lock .....	585
表 3-889. 函数 sram1_write_protect_0_31 .....	585
表 3-890. 函数 sram1_write_protect_32_63 .....	586
表 3-891. 函数 compensation_ready_flag_get .....	586
表 3-892. 函数 sram1_bsy_flag_get .....	587
表 3-893. 函数 fpu_interrupt_enable .....	587
表 3-894. 函数 fpu_interrupt_disable .....	588
表 3-895. TIMER 寄存器 .....	589
表 3-896. TIMER 库函数 .....	589
表 3-897. 结构体 timer_parameter_struct .....	592
表 3-898. 结构体 timer_break_parameter_struct .....	592
表 3-899. 结构体 timer_oc_parameter_struct .....	592
表 3-900. 结构体 timer_ic_parameter_struct .....	593
表 3-901. 函数 timer_deinit .....	593
表 3-902. 函数 timer_struct_para_init .....	594
表 3-903. 函数 timer_init .....	594
表 3-904. 函数 timer_enable .....	595
表 3-905. 函数 timer_disable .....	596
表 3-906. 函数 timer_auto_reload_shadow_enable .....	596
表 3-907. 函数 timer_auto_reload_shadow_disable .....	597
表 3-908. 函数 timer_update_event_enable .....	597
表 3-909. 函数 timer_update_event_disable .....	598
表 3-910. 函数 timer_counter_alignment .....	598
表 3-911. 函数 timer_counter_up_direction .....	599
表 3-912. 函数 timer_counter_down_direction .....	599
表 3-913. 函数 timer_prescaler_config .....	600
表 3-914. 函数 timer_repetition_value_config .....	601
表 3-915. 函数 timer_autoreload_value_config .....	601
表 3-916. 函数 timer_counter_value_config .....	602
表 3-917. 函数 timer_counter_read .....	602
表 3-918. 函数 timer_prescaler_read .....	603



表 3-919. 函数 timer_single_pulse_mode_config.....	604
表 3-920. 函数 timer_update_source_config.....	604
表 3-921. 函数 timer_dma_enable .....	605
表 3-922. 函数 timer_dma_disable .....	606
表 3-923. 函数 timer_channel_dma_request_source_select.....	607
表 3-924. 函数 timer_dma_transfer_config .....	607
表 3-925. 函数 timer_event_software_generate.....	609
表 3-926. 函数 timer_break_struct_para_init .....	610
表 3-927. 函数 timer_break_config.....	610
表 3-928. 函数 timer_break_enable .....	611
表 3-929. 函数 timer_break_disable .....	612
表 3-930. 函数 timer_automatic_output_enable .....	612
表 3-931. 函数 timer_automatic_output_disable .....	613
表 3-932. 函数 timer_primary_output_config.....	613
表 3-933. 函数 timer_channel_control_shadow_config.....	614
表 3-934. 函数 timer_channel_control_shadow_update_config.....	615
表 3-935. 函数 timer_channel_output_struct_para_init.....	615
表 3-936. 函数 timer_channel_output_config .....	616
表 3-937. 函数 timer_channel_output_mode_config.....	617
表 3-938. 函数 timer_channel_output_pulse_value_config.....	618
表 3-939. 函数 timer_channel_output_shadow_config.....	619
表 3-940. 函数 timer_channel_output_fast_config.....	619
表 3-941. 函数 timer_channel_output_clear_config.....	620
表 3-942. 函数 timer_channel_output_polarity_config .....	621
表 3-943. 函数 timer_channel_complementary_output_polarity_config .....	622
表 3-944. 函数 timer_channel_output_state_config.....	623
表 3-945. 函数 timer_channel_complementary_output_state_config .....	624
表 3-946. 函数 timer_channel_input_struct_para_init .....	624
表 3-947. 函数 timer_input_capture_config .....	625
表 3-948. 函数 timer_channel_input_capture_prescaler_config .....	626
表 3-949. 函数 timer_channel_capture_value_register_read .....	627
表 3-950. 函数 timer_input_pwm_capture_config .....	627
表 3-951. 函数 timer_hall_mode_config .....	628
表 3-952. 函数 timer_input_trigger_source_select.....	629
表 3-953. 函数 timer_master_output_trigger_source_select .....	630
表 3-954. 函数 timer_slave_mode_select .....	631
表 3-955. 函数 timer_master_slave_mode_config.....	632
表 3-956. 函数 timer_external_trigger_config.....	632
表 3-957. 函数 timer_quadrature_decoder_mode_config.....	633
表 3-958. 函数 timer_internal_clock_config.....	635
表 3-959. 函数 timer_internal_trigger_as_external_clock_config.....	635
表 3-960. 函数 timer_external_trigger_as_external_clock_config.....	636
表 3-961. 函数 timer_external_clock_mode0_config .....	637
表 3-962. 函数 timer_external_clock_mode1_config .....	638

表 3-963. 函数 timer_external_clock_mode1_disable .....	639
表 3-964. 函数 timer_write_chxval_register_config .....	639
表 3-965. 函数 timer_output_value_selection_config .....	640
表 3-966. 函数 timer_flag_get .....	641
表 3-967. 函数 timer_flag_clear .....	641
表 3-968. 函数 timer_interrupt_enable .....	642
表 3-969. 函数 timer_interrupt_disable .....	643
表 3-970. 函数 timer_interrupt_flag_get .....	644
表 3-971. 函数 timer_interrupt_flag_clear .....	645
表 3-972. TRNG 寄存器 .....	646
表 3-973. TRNG 库函数 .....	646
表 3-974. 枚举 trng_flag_enum .....	646
表 3-975. 枚举 trng_int_flag_enum .....	647
表 3-976. 函数 trng_deinit .....	647
表 3-977. 函数 trng_enable .....	647
表 3-978. 函数 trng_disable .....	648
表 3-979. 函数 trng_get_true_random_data .....	648
表 3-980. 函数 trng_interrupt_enable .....	649
表 3-981. 函数 trng_interrupt_disable .....	649
表 3-982. 函数 trng_flag_get .....	650
表 3-983. 函数 trng_interrupt_flag_get .....	650
表 3-984. 函数 trng_interrupt_flag_clear .....	651
表 3-985. TSI 寄存器 .....	651
表 3-986. TSI 库函数 .....	652
表 3-987. 函数 tsi_deinit .....	653
表 3-988. 函数 tsi_init .....	653
表 3-989. 函数 cec_enable .....	655
表 3-990. 函数 tsi_disable .....	655
表 3-991. 函数 tsi_sample_pin_enable .....	656
表 3-992. 函数 tsi_sample_pin_disable .....	656
表 3-993. 函数 tsi_channel_pin_enable .....	657
表 3-994. 函数 tsi_channel_pin_disable .....	657
表 3-995. 函数 tsi_software_mode_config .....	658
表 3-996. 函数 tsi_software_start .....	658
表 3-997. 函数 tsi_software_stop .....	659
表 3-998. 函数 tsi_hardware_mode_config .....	659
表 3-999. 函数 tsi_pin_mode_config .....	660
表 3-1000. 函数 tsi_extend_charge_config .....	660
表 3-1001. 函数 tsi_plus_config .....	661
表 3-1002. 函数 tsi_max_number_config .....	663
表 3-1003. 函数 tsi_hysteresis_on .....	663
表 3-1004. 函数 tsi_hysteresis_off .....	664
表 3-1005. 函数 tsi_analog_on .....	664
表 3-1006. 函数 tsi_analog_off .....	665



表 3-1007. 函数 tsi_group_enable .....	665
表 3-1008. 函数 tsi_group_disable .....	666
表 3-1009. 函数 tsi_group_status_get .....	666
表 3-1010. 函数 tsi_group0_cycle_get .....	667
表 3-1011. 函数 tsi_group1_cycle_get .....	667
表 3-1012. 函数 tsi_group2_cycle_get .....	668
表 3-1013. 函数 tsi_flag_clear .....	669
表 3-1014. 函数 tsi_flag_get .....	669
表 3-1015. 函数 tsi_interrupt_enable .....	670
表 3-1016. 函数 tsi_interrupt_disable .....	670
表 3-1017. 函数 tsi_interrupt_flag_clear .....	671
表 3-1018. 函数 tsi_interrupt_flag_get .....	671
表 3-1019. TZPCU 寄存器 .....	672
表 3-1020. TZPCU 库函数 .....	673
表 3-1021. 枚举类型 tzpcu_mem .....	673
表 3-1022. 枚举类型 tzpcu_non_secure_mark_region .....	674
表 3-1023. 结构体 tzpcu_non_secure_mark_struct .....	674
表 3-1024. 函数 tzpcu_tzspc_peripheral_attributes_config .....	674
表 3-1025. 函数 tzpcu_tzspc_peripheral_attributes_get .....	676
表 3-1026. 函数 tzpcu_tzspc_emnsm_config .....	679
表 3-1027. 函数 tzpcu_tzspc_items_lock .....	679
表 3-1028. 函数 tzpcu_tzspc_dbg_config .....	680
表 3-1029. 函数 tzpcu_tzmpc_lock .....	681
表 3-1030. 函数 tzpcu_tzmpc_security_state_config .....	681
表 3-1031. 函数 tzpcu_tzmpc_secure_access_config .....	682
表 3-1032. 函数 tzpcu_tzmpc_block_secure_access_mode_config .....	682
表 3-1033. 函数 tzpcu_tzmpc_union_block_lock .....	683
表 3-1034. 函数 tzpcu_tziac_interrupt_enable .....	684
表 3-1035. 函数 tzpcu_tziac_interrupt_disable .....	687
表 3-1036. 函数 tzpcu_tziac_flag_get .....	689
表 3-1037. 函数 tzpcu_tziac_flag_clear .....	692
表 3-1038. USART 寄存器 .....	696
表 3-1039. USART 库函数 .....	696
表 3-1040. 枚举类型 usart_flag_enum .....	698
表 3-1041. 枚举类型 usart_interrupt_flag_enum .....	699
表 3-1042. 枚举类型 usart_interrupt_enum .....	699
表 3-1043. 枚举类型 usart_invert_enum .....	700
表 3-1044. 函数 usart_deinit .....	700
表 3-1045. 函数 usart_baudrate_set .....	701
表 3-1046. 函数 usart_parity_config .....	701
表 3-1047. 函数 usart_word_length_set .....	702
表 3-1048. 函数 usart_stop_bit_set .....	702
表 3-1049. 函数 usart_enable .....	703
表 3-1050. 函数 usart_disable .....	704

表 3-1051. 函数 usart_transmit_config .....	704
表 3-1052. 函数 usart_receive_config .....	705
表 3-1053. 函数 usart_data_first_config .....	705
表 3-1054. 函数 usart_invert_config .....	706
表 3-1055. 函数 usart_overrun_enable .....	707
表 3-1056. 函数 usart_overrun_disable .....	707
表 3-1057. 函数 usart_oversample_config .....	708
表 3-1058. 函数 usart_sample_bit_config .....	709
表 3-1059. 函数 usart_receiver_timeout_enable .....	709
表 3-1060. 函数 usart_receiver_timeout_disable .....	710
表 3-1061. 函数 usart_receiver_timeout_threshold_config .....	710
表 3-1062. 函数 usart_data_transmit .....	711
表 3-1063. 函数 usart_data_receive .....	711
表 3-1064. 函数 usart_address_config .....	712
表 3-1065. 函数 usart_address_detection_mode_config .....	712
表 3-1066. 函数 usart_mute_mode_enable .....	713
表 3-1067. 函数 usart_mute_mode_disable .....	714
表 3-1068. 函数 usart_mute_mode_wakeup_config .....	714
表 3-1069. 函数 usart_lin_mode_enable .....	715
表 3-1070. 函数 usart_lin_mode_disable .....	715
表 3-1071. 函数 usart_lin_break_dection_length_config .....	716
表 3-1072. 函数 usart_halfduplex_enable .....	716
表 3-1073. 函数 usart_halfduplex_disable .....	717
表 3-1074. 函数 usart_clock_enable .....	717
表 3-1075. 函数 usart_clock_disable .....	718
表 3-1076. 函数 usart_synchronous_clock_config .....	718
表 3-1077. 函数 usart_guard_time_config .....	719
表 3-1078. 函数 usart_smartcard_mode_enable .....	720
表 3-1079. 函数 usart_smartcard_mode_disable .....	720
表 3-1080. 函数 usart_smartcard_mode_nack_enable .....	721
表 3-1081. 函数 usart_smartcard_mode_nack_disable .....	721
表 3-1082. 函数 usart_smartcard_mode_early_nack_enable .....	722
表 3-1083. 函数 usart_smartcard_mode_early_nack_disable .....	722
表 3-1084. 函数 usart_smartcard_autoretry_config .....	723
表 3-1085. 函数 usart_block_length_config .....	723
表 3-1086. 函数 usart_irda_mode_enable .....	724
表 3-1087. 函数 usart_irda_mode_disable .....	724
表 3-1088. 函数 usart_prescaler_config .....	725
表 3-1089. 函数 usart_irda_lowpower_config .....	725
表 3-1090. 函数 usart_hardware_flow_rts_config .....	726
表 3-1091. 函数 usart_hardware_flow_cts_config .....	727
表 3-1092. 函数 usart_hardware_flow_coherence_config .....	728
表 3-1093. 函数 usart_rs485_driver_enable .....	728
表 3-1094. 函数 usart_rs485_driver_disable .....	729

表 3-1095. 函数 usart_driver_asserttime_config .....	729
表 3-1096. 函数 usart_driver_deasserttime_config .....	730
表 3-1097. 函数 usart_depolarity_config .....	730
表 3-1098. 函数 usart_dma_receive_config .....	731
表 3-1099. 函数 usart_dma_transmit_config .....	732
表 3-1100. 函数 usart_reception_error_dma_disable .....	732
表 3-1101. 函数 usart_reception_error_dma_enable .....	733
表 3-1102. 函数 usart_wakeup_enable .....	733
表 3-1103. 函数 usart_wakeup_disable .....	734
表 3-1104. 函数 usart_wakeup_mode_config .....	734
表 3-1105. 函数 usart_receive_fifo_enable .....	735
表 3-1106. 函数 usart_receive_fifo_disable .....	735
表 3-1107. 函数 usart_receive_fifo_counter_number .....	736
表 3-1108. 函数 usart_command_enable .....	736
表 3-1109. 函数 usart_flag_get .....	737
表 3-1110. 函数 usart_flag_clear .....	739
表 3-1111. 函数 usart_interrupt_enable .....	740
表 3-1112. 函数 usart_interrupt_disable .....	741
表 3-1113. 函数 usart_interrupt_flag_get .....	742
表 3-1114. 函数 usart_interrupt_flag_clear .....	743
表 3-1115. WWDGT 寄存器 .....	744
表 3-1116. WWDGT 库函数 .....	745
表 3-1117. 函数 wwdgt_deinit .....	745
表 3-1118. 函数 wwdgt_enable .....	745
表 3-1119. 函数 wwdgt_counter_update .....	746
表 3-1120. 函数 wwdgt_config .....	746
表 3-1121. 函数 wwdgt_interrupt_enable .....	747
表 3-1122. 函数 wwdgt_flag_get .....	748
表 3-1123. 函数 wwdgt_flag_clear .....	748
表 4-1. 版本历史 .....	749

## 1. 介绍

本手册介绍了32位基于ARM微控制器GD32W51x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GD32W51x所有外设的性能特征。该固件库还包括每一个外设的驱动描述和基于评估板的固件库使用例程。通过使用本固件库，用户无需深入掌握细节，也可以轻松应用每一个外设。使用本固件库可以大大减少用户的编程时间，从而降低开发成本。

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

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

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

此份固件库使用手册的整体架构如下：

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

### 1.1. 文档和固件库规则

#### 1.1.1. 外设缩写

表 1-1. 外设缩写

外设缩写	说明
ADC	模数转换器
CAU	加密处理器
CRC	循环冗余校验计算单元
DBG	调试模块
DCI	数字摄像头接口
DMA	直接存储器访问控制器
EFUSE	熔丝
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

外设缩写	说明
HAU	哈希处理器
HPDF	高性能数字滤波器
I2C	内部集成电路总线接口
ICACHE	指令缓存
MISC	嵌套中断向量列表控制器
PKCAU	公钥加密处理器
PMU	电源管理单元
QSPI	四线SPI接口
RCU	复位和时钟单元
RTC	实时时钟
SDIO	SDIO接口
SPI/I2S	串行外设接口/片上音频接口
SQPI	SQPI接口
TIMER	定时器
TRNG	真随机数生成器
TSI	触摸传感控制器
TZPCU	Trustzone保护控制器组
USART	通用同步异步收发器
WWDT	窗口看门狗
USBFS	USBFS

### 1.1.2. 命名规则

固件库遵从以下命名规则：

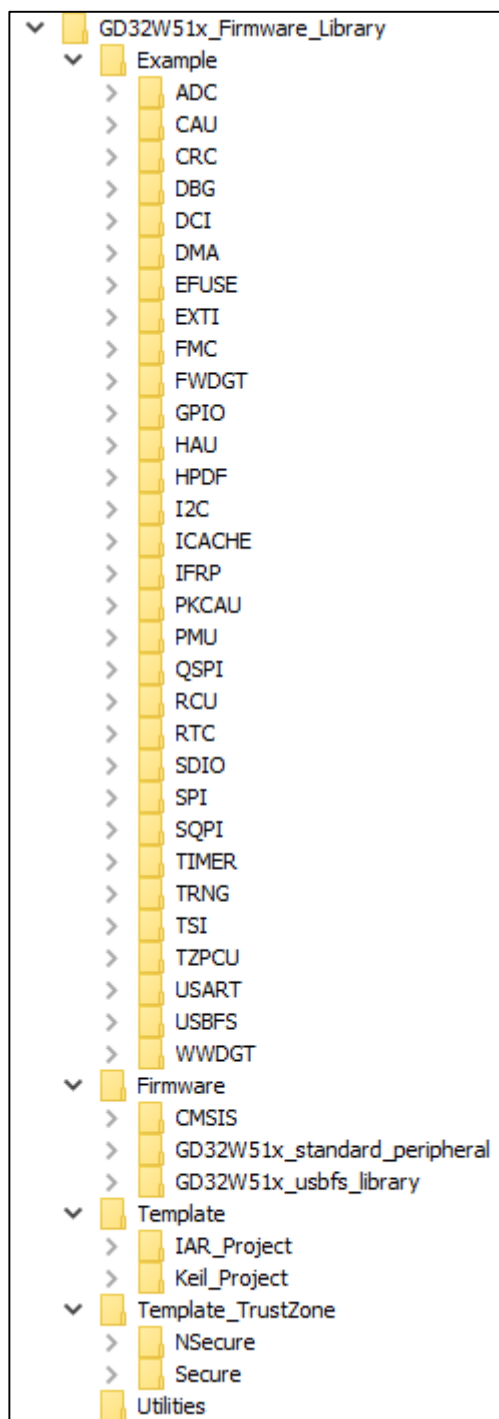
- XXX表示任一外设缩写，例如：ADC。更多缩写相关信息参阅[外设缩写](#)；
- 源文件和头文件命名都以“gd32w51x\_”作为开头，例如：gd32w51x\_adc.h；
- 常量仅被应用于一个文件的，定义于该文件中；被应用于多个文件的，在对应头文件中定义。所有常量都由英文字母大写书写；
- 寄存器作为常量处理。他们的命名都由英文字母大写书写。在大多数情况下，寄存器缩写规范与本用户手册一致；
- 变量名采用全部小写，有多个单词组成的，在单词之间以下划线分隔；
- 外设函数的命名以该外设的缩写加下划线为开头，有多个单词组成的，在单词之间以下划线分隔，所有外设函数都由英文字母小写书写。

## 2. 固件库概述

### 2.1. 文件组织结构

GD32W51x\_Firmware\_Library，文件组织结构见下图：

图 2-1. GD32W51x 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；
- **gd32w51x\_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）；
- **gd32w51x\_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）；
- **gd32w51x\_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。

GD32W51x系列MCU基于Arm®Cortex™-M33内核，并支持Arm®Trustzone®技术。一些外设包括**trustzone**使能例程。在这些例程中，将包含如下文件：

- **partition\_gd32w51x.h**: 该头文件可以配置**SAU/IDAU**以及中断的安全/非安全属性；
- **Project\_S.sct**: 在安全工程中**keil**分散加载文件提供了每个域或段的分组和放置的详细信息；
- **Project\_NS.sct**: 在非安全工程中**keil**分散加载文件提供了每个域或段的分组和放置的详细信息；
- **gd32w51x\_flash\_s.icf**: 在安全工程中**IAR**链接器配置文件，根据要求链接和加载文件；
- **gd32w51x\_flash\_ns.icf**: 在非安全工程中**IAR**链接器配置文件，根据要求链接和加载文件。

### 2.1.2. Firmware 文件夹

**Firmware**文件夹包含组成固件库核心的所有子文件夹和文件：

- **CMSIS**子文件夹包含有**Cortex M33**内核的支持文件、基于**Cortex M33**内核处理器的启动代码和库引导文件以及基于**GD32W51x**的全局头文件和系统配置文件；
- **GD32W51x\_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；
- **GD32W51x\_usbfs\_library**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。

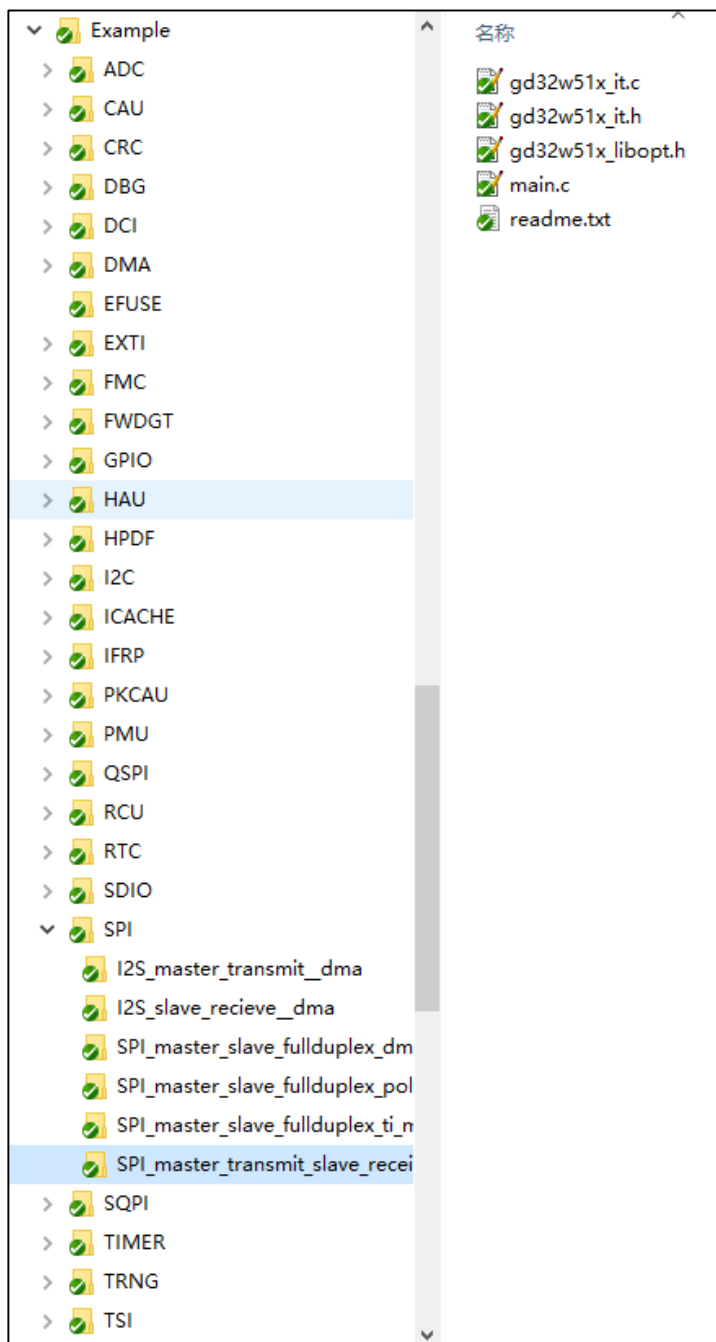
### 2.1.3. Template 文件夹

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

## 选择文件

打开“Examples”文件夹，选择需要测试的模块，如SPI，打开“SPI”文件夹，选择SPI的一个例程，如“SPI\_master\_transmit\_slave\_receive\_interrupt”，如下图所示：

图 2-2. 选择外设例程文件

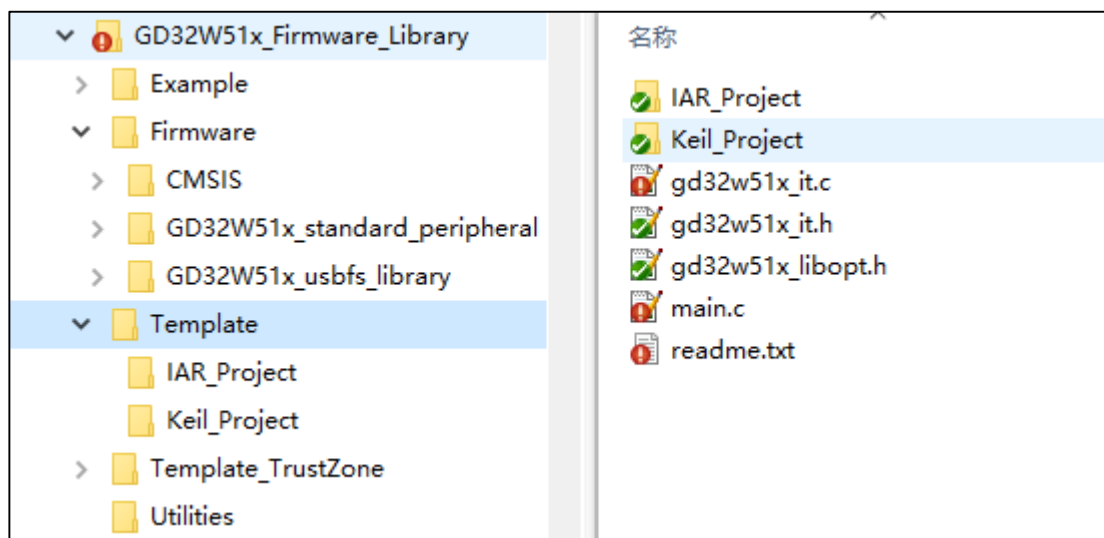


## 拷贝文件

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



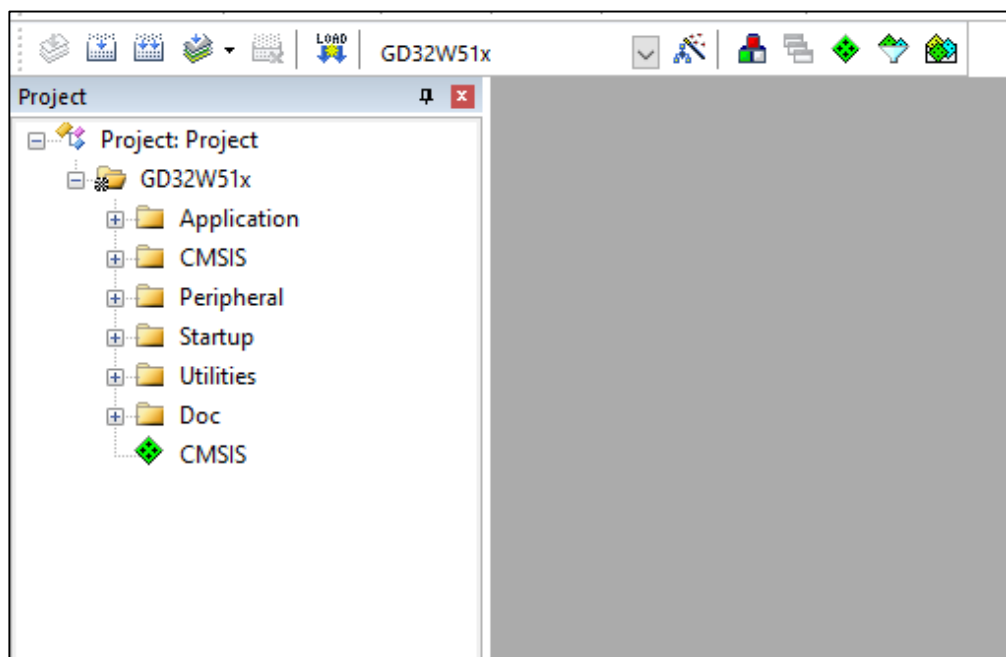
图 2-3. 拷贝外设例程文件



### 打开工程

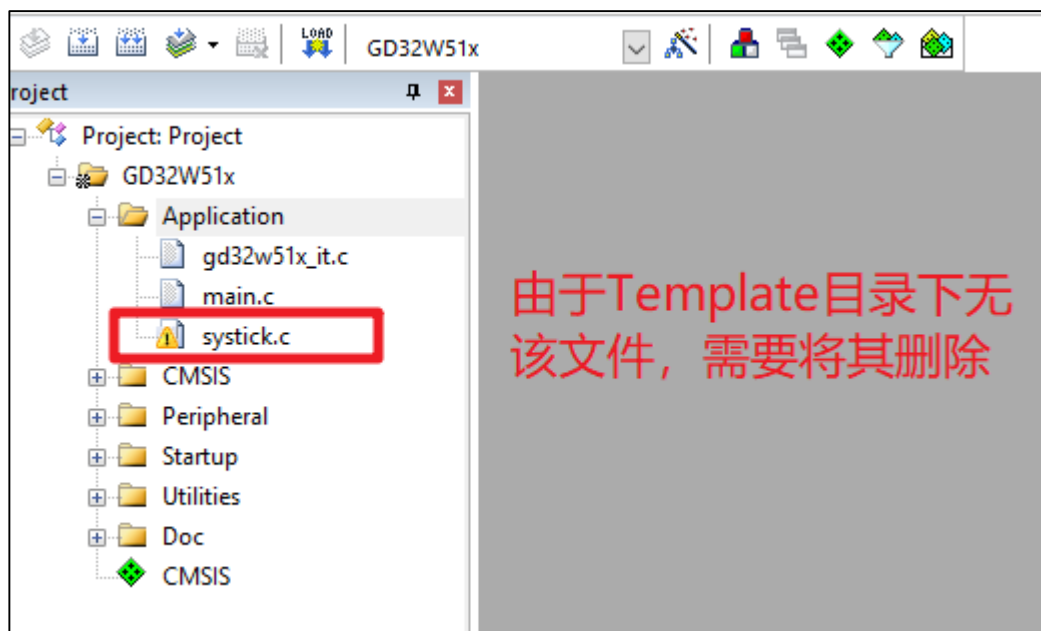
GD提供Keil和IAR两种版本的工程，根据客户所安装的软件，打开不同的project，如“Keil\_project”，打开\Template\Keil\_project\Project.uvprojx，如下图所示：

图 2-4. 打开工程文件



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

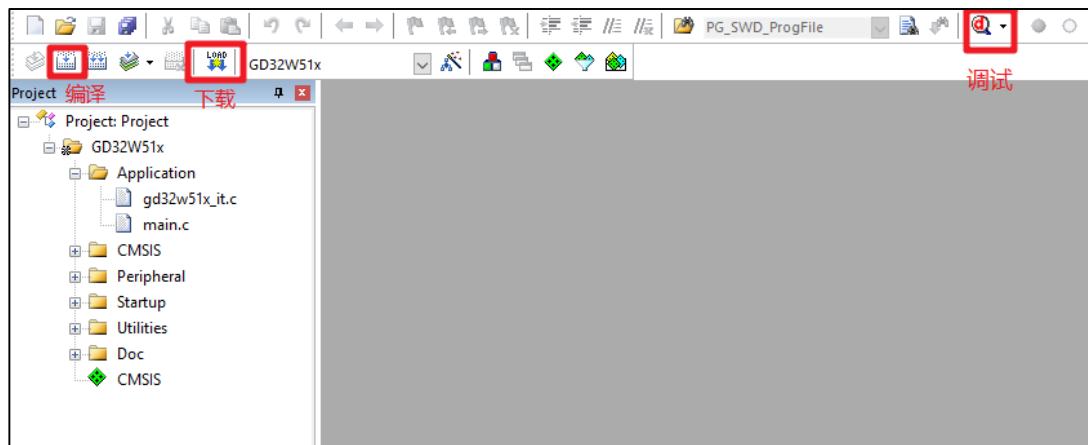
图 2-5. 配置工程文件



### 编译调试下载

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

图 2-6. 编译调试下载



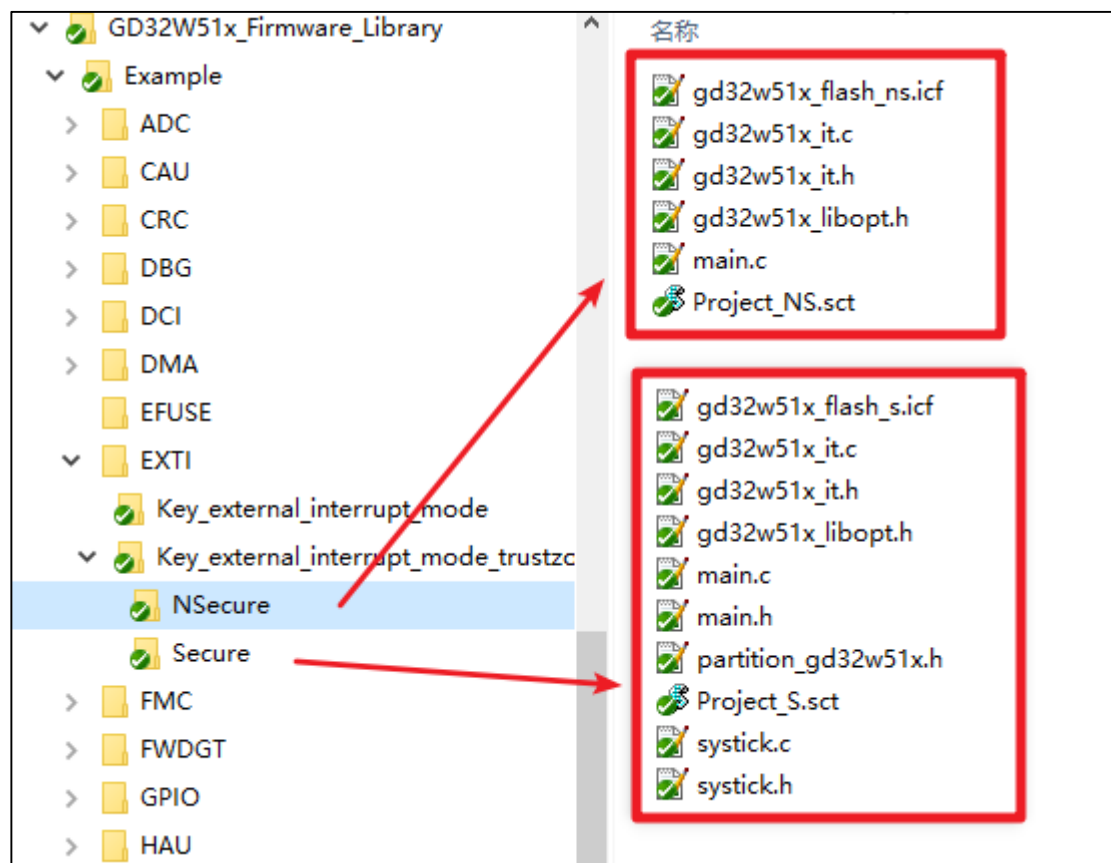
#### 2.1.4. Template\_Trustzone 文件夹

Template\_TrustZone 文件夹包含安全和非安全两个子目录,该例程简单示例如何在安全应用程序和非安全应用程序之间切换, (其中 IAR\_project 用于 IAR 编译环境, Keil\_project 用于 Keil5 编译环境)。用户可以使用该工程模板进行固件库例程的移植编译, 具体使用方法见下:

## 选择文件

打开“Examples”文件夹，选择需要测试的模块，如EXTI，打开“EXTI”文件夹，选择SPI的一个例程，如“Key\_external\_interrupt\_mode\_trustzone”，如下图所示：

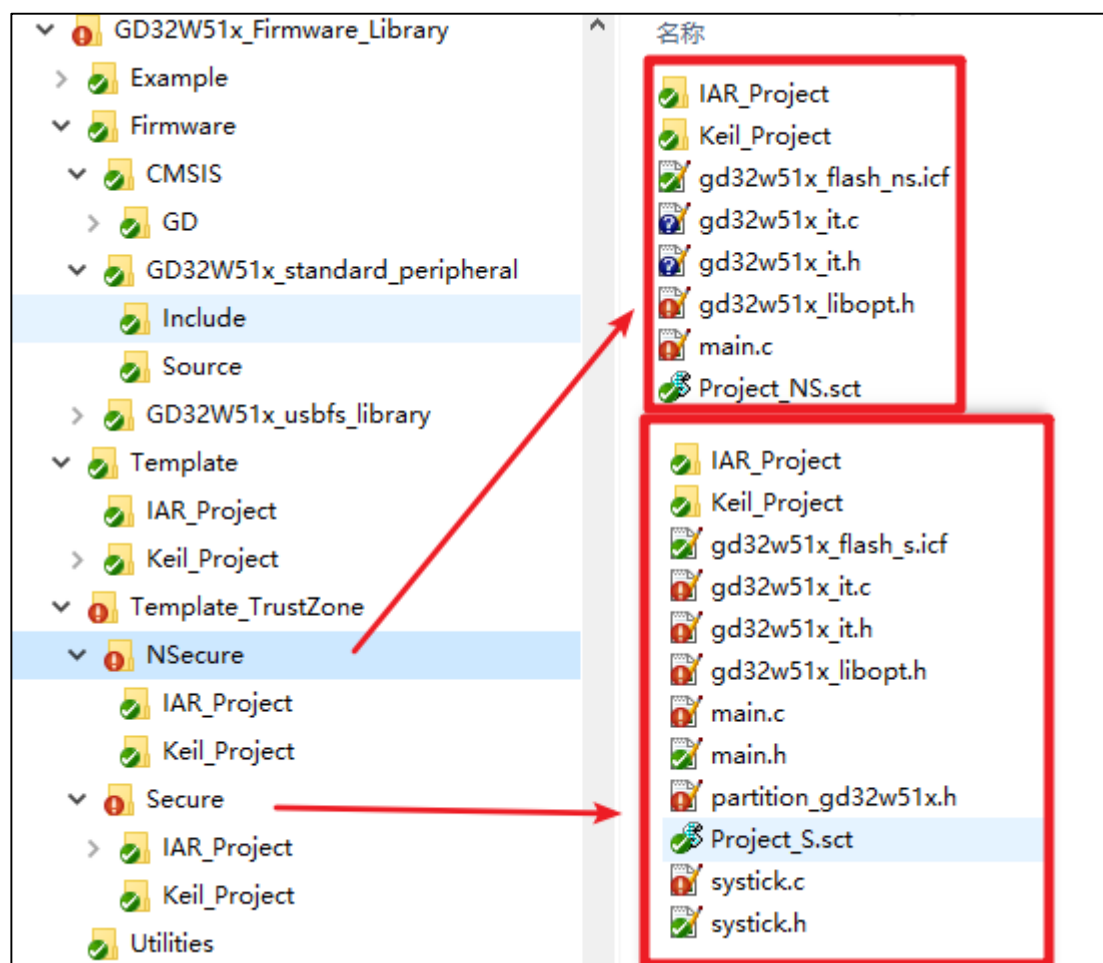
图 2-7. 选择外设例程文件



## 拷贝文件

打开“Template\_TrustZone”文件夹，将Secure和NSecure文件夹中的“IAR\_project”和“Keil\_project”两个子文件夹保留，其他文件都删除，然后将“Key\_external\_interrupt\_mode\_trustzone”子文件夹中的所有文件分别拷到“Template\_TrustZone”文件夹Secure和NSecure文件夹子目录下，如下图所示：

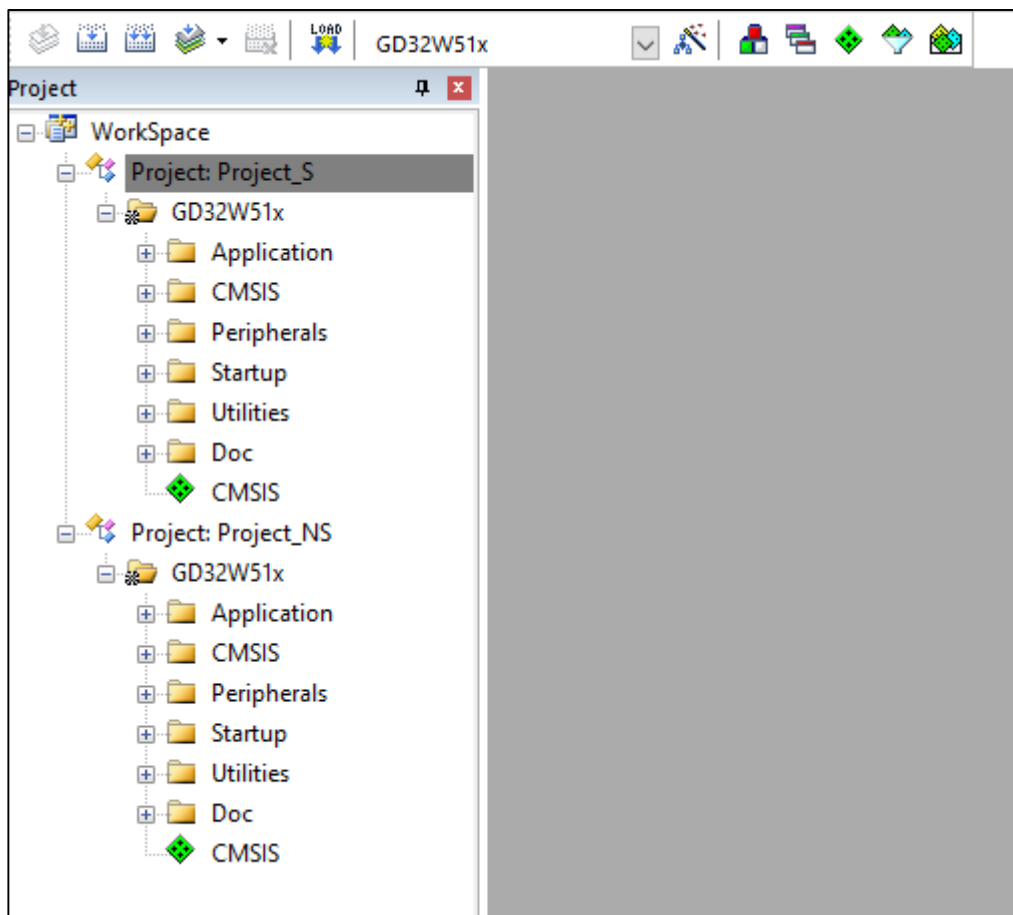
图 2-8. 拷贝外设例程文件



### 打开工作区

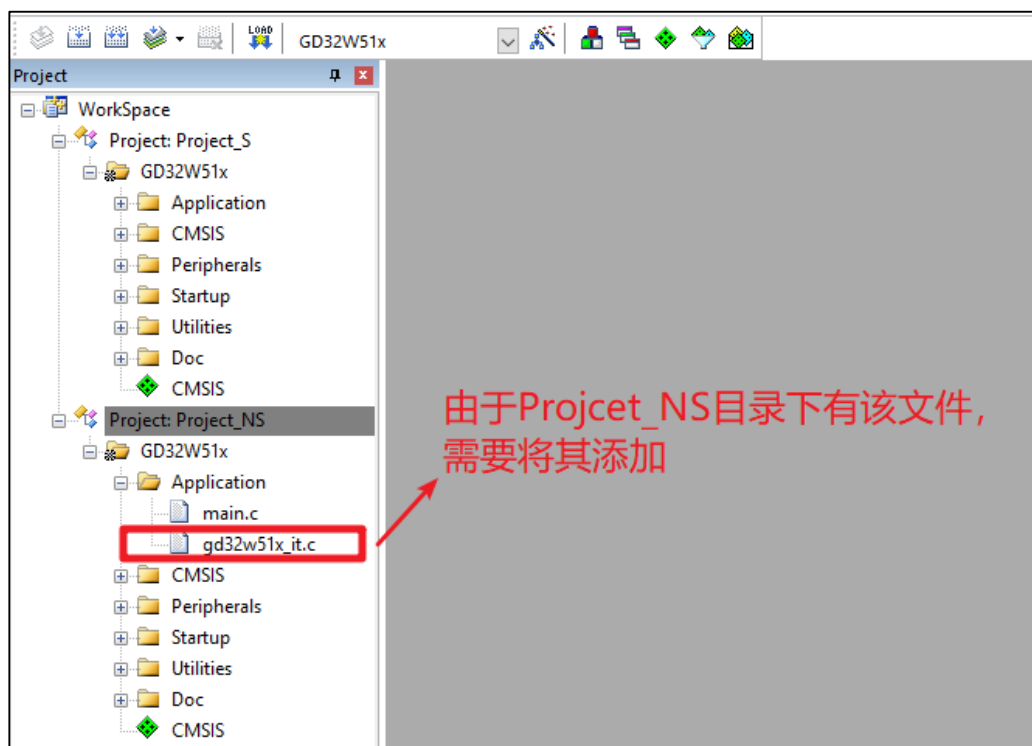
GD提供Keil和IAR两种版本的工作区，根据客户所安装的软件，打开不同的工作区，如“Keil\_project”工作区，打开\ Template\_TrustZone\ Project.uvmpw，如下图所示：

图 2-9. 打开工程文件



工作区包含安全和非安全两个工程，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：

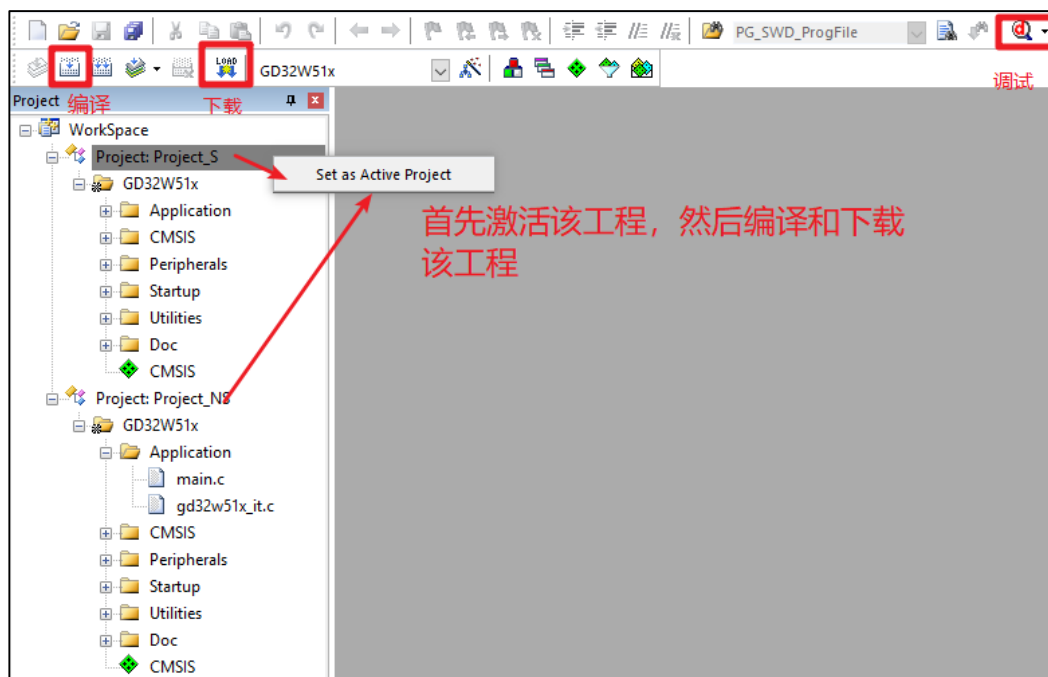
图 2-10. 配置工程文件



### 编译调试下载

首先分别编译两个工程，如果无错误，按照readme中的介绍，选择正确的跳线及连线，然后再将程序分别下载到目标板上，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 2-11. 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：

- gd32w515p\_eval.h文件是运行固件库例程所需关于评估板的头文件；
- gd32w515p\_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
gd32w51x_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。
gd32w51x_it.h	头文件，包含所有中断处理函数原形。
gd32w51x_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。
gd32w51x_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。
gd32w51x_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。
partition_gd32w51x.h	头文件可以配置SAU/IDAU以及中断的安全/非安全属性



## 3. 外设固件库

### 3.1. 外设固件库概述

外设固件库函数的描述格式如下表：

表 3-1. 外设固件库函数描述格式

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

### 3.2. ADC

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

#### 3.2.1. 外设寄存器描述

ADC寄存器列表如下表所示：

表 3-2. ADC 寄存器

寄存器名称	寄存器描述
ADC_STAT	状态寄存器
ADC_CTL0	控制寄存器0
ADC_CTL1	控制寄存器1
ADC_SAMPT0	采样时间寄存器0
ADC_SAMPT1	采样时间寄存器1
ADC_IOFFx(x=0..3)	注入通道数据偏移寄存器x (x=0..3)
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

寄存器名称	寄存器描述
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(x=0..3)
ADC_RDATA	规则数据寄存器
ADC_OVSAMPCTL	过采样控制寄存器
ADC_CCTL	通用控制寄存器

### 3.2.2. 外设库函数说明

ADC库函数列表如下表所示：

**表 3-3. ADC 库函数**

库函数名称	库函数描述
adc_deinit	复位ADC外设
adc_clock_config	ADC时钟配置
adc_enable	使能ADC外设
adc_disable	禁能ADC外设
adc_dma_mode_enable	ADC DMA请求使能
adc_dma_mode_disable	ADC DMA请求禁能
adc_dma_request_after_last_enable	DMA=1时，在每个规则组转换结束时，DMA机制产生一个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禁能
adc_discontinuous_mode_config	配置ADC间断模式
adc_special_function_config	使能或禁能ADC特殊功能
adc_channel_9_to_11	温度传感器、内部参考电压和外部电池引脚功能配置
adc_data_alignment_config	配置ADC数据对齐方式
adc_channel_length_config	配置规则通道组或注入通道组的长度
adc_regular_channel_config	配置ADC规则通道组
adc_inserted_channel_config	配置ADC注入通道组
adc_inserted_channel_offset_config	配置ADC注入通道组数据偏移值
adc_external_trigger_config	配置ADC外部触发
adc_external_trigger_source_config	配置ADC外部触发源
adc_software_trigger_enable	ADC软件触发使能
adc_end_of_conversion_config	转换模式结束配置
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_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
adc_watchdog_disable	ADC模拟看门狗禁能

库函数名称	库函数描述
adc_watchdog_threshold_config	配置ADC模拟看门狗阈值
adc_oversample_mode_config	配置ADC过采样模式
adc_oversample_mode_enable	使能ADC过采样
adc_oversample_mode_disable	禁能ADC过采样
adc_flag_get	获取ADC标志位
adc_flag_clear	清除ADC标志位
adc_interrupt_enable	ADC中断使能
adc_interrupt_disable	ADC中断禁能
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_regular_software_startconv_flag_get	获取ADC规则组转换开始标志位
adc_inserted_software_startconv_flag_get	获取ADC注入组转换开始标志位

### 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

函数名称	adc_clock_config
函数原形	void adc_clock_config(uint32_t prescaler);

功能描述	ADC时钟配置
先决条件	-
被调用函数	-
输入参数{in}	
prescaler	配置ADC分频系数
ADC_ADCCCK_ PCLK2_DIV2	PCLK2 二分频
ADC_ADCCCK_ PCLK2_DIV4	PCLK2 四分频
ADC_ADCCCK_ PCLK2_DIV6	PCLK2 六分频
ADC_ADCCCK_ PCLK2_DIV8	PCLK2 八分频
ADC_ADCCCK_ HCLK_DIV5	HCLK 五分频
ADC_ADCCCK_ HCLK_DIV6	HCLK 六分频
ADC_ADCCCK_ HCLK_DIV10	HCLK 十分频
ADC_ADCCCK_ HCLK_DIV20	HCLK 二十分频
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* configure the ADC clock */
```

```
adc_clock_config(ADC_ADCCCK_PCLK2_DIV8);
```

### 函数 adc\_enable

函数adc\_enable描述见下表：

表 3-6. 函数 adc\_enable

函数名称	adc_enable
函数原形	void adc_enable(void);
功能描述	使能ADC外设
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 ADC */
```

```
adc_enable();
```

### 函数 **adc\_disable**

函数adc\_disable描述见下表：

**表 3-7. 函数 `adc_disable`**

函数名称	adc_disable
函数原形	void adc_disable(void);
功能描述	禁能ADC外设
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 ADC */
```

```
adc_disable();
```

### 函数 **adc\_dma\_mode\_enable**

函数adc\_dma\_mode\_enable描述见下表：

**表 3-8. 函数 `adc_dma_mode_enable`**

函数名称	adc_dma_mode_enable
函数原形	void adc_dma_mode_enable(void);
功能描述	ADC DMA请求使能
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 ADC DMA request */
adc_dma_mode_enable();
```

### 函数 `adc_dma_mode_disable`

函数`adc_dma_mode_disable`描述见下表：

**表 3-9. 函数 `adc_dma_mode_disable`**

函数名称	<code>adc_dma_mode_disable</code>
函数原形	<code>void adc_dma_mode_disable(void);</code>
功能描述	ADC DMA请求禁能
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 ADC DMA request */
adc_dma_mode_disable();
```

### 函数 `adc_dma_request_after_last_enable`

函数`adc_dma_request_after_last_enable`描述见下表：

**表 3-10. 函数 `adc_dma_request_after_last_enable`**

函数名称	<code>adc_dma_request_after_last_enable</code>
函数原形	<code>void adc_dma_request_after_last_enable(void);</code>
功能描述	DMA=1时，在每个规则组转换结束时，DMA机制产生一个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：

```
/* when DMA=1, the DMA engine issues a request at end of each regular conversion */  
adc_dma_request_after_last_enable();
```

## 函数 `adc_dma_request_after_last_disable`

函数`adc_dma_request_after_last_disable`描述见下表：

**表 3-11. 函数 `adc_dma_request_after_last_disable`**

函数名称	<code>adc_dma_request_after_last_disable</code>
函数原形	<code>void adc_dma_request_after_last_disable (void);</code>
功能描述	DMA机制在DMA控制器的传输结束信号之后禁能
先决条件	-
被调用函数	-
输入参数{in}	
-	-
输出参数{out}	
-	-
返回值	
-	-

例如：

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

## 函数 `adc_discontinuous_mode_config`

函数`adc_discontinuous_mode_config`描述见下表：

**表 3-12. 函数 `adc_discontinuous_mode_config`**

函数名称	<code>adc_discontinuous_mode_config</code>
函数原形	<code>void adc_discontinuous_mode_config(uint8_t adc_channel_group, uint8_t length);</code>
功能描述	配置ADC间断模式
先决条件	-
被调用函数	-
输入参数{in}	
<code>adc_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



ADC_CHANNEL_1	规则通道组和注入通道组间断模式禁能
DISCON_DISABLE	
输入参数{in}	
length	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
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* configure ADC discontinuous mode */
```

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

### 函数 adc\_special\_function\_config

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

表 3-13. 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 function, ControlStatus newvalue);
功能描述	使能或禁能ADC特殊功能
先决条件	-
被调用函数	-
输入参数{in}	
function	功能配置
ADC_SCAN_MODE	扫描模式选择
ADC_INSERTED_CHANNEL_AUTO	注入组自动转换
ADC_CONTINUOUS_MODE	连续模式选择
输入参数{in}	
newvalue	功能使能禁能
ENABLE	使能
DISABLE	禁能
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* enable ADC scan mode */
```

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

函数 `adc_channel_9_to_11`

函数`adc_channel_9_to_11`描述见下表：

表 3-14. 函数 `adc_channel_9_to_11`

函数名称	<code>adc_channel_9_to_11</code>
函数原形	<code>void adc_channel_9_to_11(uint32_t function, ControlStatus newvalue);</code>
功能描述	温度传感器、内部参考电压和外部电池引脚功能配置
先决条件	-
被调用函数	-
输入参数{in}	
<b>function</b>	温度传感器、内部参考电压和外部电池引脚
<code>ADC_VBAT_CHANNEL_SWITCH</code>	通道11（外部电池的四分之一电压）
<code>ADC_TEMP_VREF_CHANNEL_SWITCH</code>	通道9（温度传感器）和通道10（内部参考电压）
输入参数{in}	
<b>newvalue</b>	功能使能禁能
<code>ENABLE</code>	使能
<code>DISABLE</code>	禁能
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* configure VBAT channel function */
```

```
adc_channel_9_to_11 (ADC_VBAT_CHANNEL_SWITCH, ENABLE);
```

函数 `adc_data_alignment_config`

函数`adc_data_alignment_config`描述见下表：

表 3-15. 函数 `adc_data_alignment_config`

函数名称	<code>adc_data_alignment_config</code>
函数原形	<code>void adc_data_alignment_config(uint32_t data_alignment);</code>
功能描述	配置ADC数据对齐方式
先决条件	-
被调用函数	-
输入参数{in}	
<b>data_alignment</b>	数据对齐方式选择
<code>ADC_DATAALIGN_RIGHT</code>	右对齐

ADC_DATAALIGN_ LEFT	左对齐
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* configure ADC data alignment */
```

```
adc_data_alignment_config(ADC_DATAALIGN_RIGHT);
```

### 函数 adc\_channel\_length\_config

函数adc\_channel\_length\_config描述见下表：

表 3-16. 函数 adc\_channel\_length\_config

函数名称	adc_channel_length_config
函数原形	void adc_channel_length_config(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	-
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}	
length	通道长度，规则通道组为1-9，注入通道组为1-4
输出参数{out}	
-	-
返回值	
-	-

例如：

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

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

### 函数 adc\_regular\_channel\_config

函数adc\_regular\_channel\_config描述见下表：

表 3-17. 函数 `adc_regular_channel_config`

函数名称	<code>adc_regular_channel_config</code>
函数原形	<code>void adc_regular_channel_config(uint8_t rank, uint8_t adc_channel, uint32_t sample_time);</code>
功能描述	配置ADC规则通道组
先决条件	-
被调用函数	-
输入参数{in}	
<b>rank</b>	规则组通道序列，取值范围为0~8
输入参数{in}	
<b>adc_channel</b>	ADC通道选择
<code>ADC_CHANNEL_x</code> ( $x=0..11$ )	ADC通道x ( $x=0..11$ )
输入参数{in}	
<b>sample_time</b>	采样时间
<code>ADC_SAMPLETIME_2</code>	2周期
<code>ADC_SAMPLETIME_15</code>	15周期
<code>ADC_SAMPLETIME_28</code>	24周期
<code>ADC_SAMPLETIME_56</code>	56周期
<code>ADC_SAMPLETIME_84</code>	84周期
<code>ADC_SAMPLETIME_112</code>	112周期
<code>ADC_SAMPLETIME_144</code>	144周期
<code>ADC_SAMPLETIME_480</code>	480周期
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* configure ADC regular channel */
adc_regular_channel_config(1, ADC_CHANNEL_0, ADC_SAMPLETIME_15);
```

#### 函数 `adc_inserted_channel_config`

函数`adc_inserted_channel_config`描述见下表：

表 3-18. 函数 `adc_inserted_channel_config`

函数名称	<code>adc_inserted_channel_config</code>
函数原形	<code>void adc_inserted_channel_config(uint8_t rank, uint8_t adc_channel, uint32_t sample_time);</code>
功能描述	配置ADC注入通道组
先决条件	-
被调用函数	-
输入参数{in}	
<b>rank</b>	注入组通道序列，取值范围为0~3
输入参数{in}	
<b>adc_channel</b>	ADC通道选择
<code>ADC_CHANNEL_x</code> ( $x=0..11$ )	ADC通道x ( $x=0..11$ )
输入参数{in}	
<b>sample_time</b>	采样时间
<code>ADC_SAMPLETIME_2</code>	2周期
<code>ADC_SAMPLETIME_15</code>	15周期
<code>ADC_SAMPLETIME_28</code>	24周期
<code>ADC_SAMPLETIME_56</code>	56周期
<code>ADC_SAMPLETIME_84</code>	84周期
<code>ADC_SAMPLETIME_112</code>	112周期
<code>ADC_SAMPLETIME_144</code>	144周期
<code>ADC_SAMPLETIME_480</code>	480周期
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* configure ADC inserted channel */
```

```
adc_inserted_channel_config(1, ADC_CHANNEL_0, ADC_SAMPLETIME_15);
```

### 函数 `adc_inserted_channel_offset_config`

函数`adc_inserted_channel_offset_config`描述见下表：

表 3-19. 函数 `adc_inserted_channel_offset_config`

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

例如:

```
/* configure ADC inserted channel offset */
```

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

### 函数 `adc_external_trigger_config`

函数`adc_external_trigger_config`描述见下表:

表 3-20. 函数 `adc_external_trigger_config`

函数名称	<code>adc_external_trigger_config</code>
函数原形	<code>void adc_external_trigger_config(uint8_t 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_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>	外部触发模式
<code>EXTERNAL_TRIGG</code>	外部触发禁能

<i>ER_DISABLE</i>	
<i>EXTERNAL_TRIGG</i> <i>ER_RISING</i>	外部触发上升沿
<i>EXTERNAL_TRIGG</i> <i>ER_FALLING</i>	外部触发下降沿
<i>EXTERNAL_TRIGG</i> <i>ER_RISING_</i> <i>FALLING</i>	外部触发上升沿和下降沿
输出参数{out}	
-	-
返回值	
-	-

例如:

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

```
adc_external_trigger_config(ADC_INSERTED_CHANNEL,  
EXTERNAL_TRIGGER_RISING);
```

### 函数 `adc_external_trigger_source_config`

函数`adc_external_trigger_source_config`描述见下表:

**表 3-21. 函数 `adc_external_trigger_source_config`**

函数名称	<code>adc_external_trigger_source_config</code>
函数原形	<code>void adc_external_trigger_source_config(uint8_t adc_channel_group, uint32_t external_trigger_source);</code>
功能描述	配置ADC外部触发源
先决条件	-
被调用函数	-
输入参数{in}	
<b>adc_channel_group</b>	通道组选择
<i>ADC_REGULAR_CHANNEL</i>	规则通道组
<i>ADC_INSERTED_CHANNEL</i>	注入通道组
输入参数{in}	
<b>external_trigger_source</b>	规则通道组或注入通道组触发源
<i>ADC_EXTTRIG_REGULAR_T0_CH0</i>	TIMER0 CH0事件（规则组）
<i>ADC_EXTTRIG_REGULAR_T0_CH1</i>	TIMER0 CH1事件（规则组）



ADC_EXTTRIG_ REGULAR_T0_CH2	TIMER0 CH2事件（规则组）
ADC_EXTTRIG_ REGULAR_T1_CH1	TIMER1 CH1事件（规则组）
ADC_EXTTRIG_ REGULAR_T1_CH2	TIMER2 CH2事件（规则组）
ADC_EXTTRIG_ REGULAR_T1_CH3	TIMER1 CH3事件（规则组）
ADC_EXTTRIG_ REGULAR_T1_ TRGO	TIMER1 TRGO事件（规则组）
ADC_EXTTRIG_ REGULAR_T2_CH0	TIMER2 CH0事件（规则组）
ADC_EXTTRIG_ REGULAR_T2_ TRGO	TIMER2 TRGO事件（规则组）
ADC_EXTTRIG_ REGULAR_T3_CH3	TIMER3 CH3事件（规则组）
ADC_EXTTRIG_RE GULAR_T4_CH0	TIMER4 CH0事件（规则组）
ADC_EXTTRIG_ REGULAR_T4_CH1	TIMER4 CH1事件（规则组）
ADC_EXTTRIG_ REGULAR_T4_CH2	TIMER4 CH2事件（规则组）
ADC_EXTTRIG_ REGULAR_ EXTI_11	外部中断线11（规则组）
ADC_EXTTRIG_ INSERTED_ T0_CH3	TIMER0 CH3事件（注入组）
ADC_EXTTRIG_ INSERTED_ T0_TRGO	TIMER0 TRGO事件（注入组）
ADC_EXTTRIG_ INSERTED_ T1_CH0	TIMER1 CH0事件（注入组）
ADC_EXTTRIG_ INSERTED_ T1_TRGO	TIMER1 TRGO事件（注入组）
ADC_EXTTRIG_ INSERTED_ T2_CH1	TIMER2 CH1事件（注入组）
ADC_EXTTRIG_ INSERTED_ T2_CH3	TIMER2 CH3事件（注入组）

INSERTED_ T2_CH3	
ADC_EXTTRIG_ INSERTED_ T3_CH0	TIMER3 CH0事件（注入组）
ADC_EXTTRIG_ INSERTED_ T3_CH1	TIMER3 CH1事件（注入组）
ADC_EXTTRIG_ INSERTED_ T3_CH2	TIMER0 CH2事件（注入组）
ADC_EXTTRIG_ INSERTED_ T3_TRGO	TIMER3 TRGO事件（注入组）
ADC_EXTTRIG_ INSERTED_ T4_CH3	TIMER4 CH3事件（注入组）
ADC_EXTTRIG_ INSERTED_ T4_TRGO	TIMER4 TRGO事件（注入组）
ADC_EXTTRIG_ INSERTED_ EXTI_15	外部中断线15（注入组）
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* configure ADC regular channel external trigger source */
adc_external_trigger_source_config (ADC_REGULAR_CHANNEL,
ADC_EXTTRIG_REGULAR_T0_CH0);
```

### 函数 adc\_software\_trigger\_enable

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

表 3-22. 函数 adc\_software\_trigger\_enable

函数名称	adc_software_trigger_enable
函数原形	void adc_software_trigger_enable(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}	
<b>adc_channel_group</b>	通道组选择
<i>ADC_REGULAR_CHANNEL</i>	规则通道组
<i>ADC_INSERTED_CHANNEL</i>	注入通道组
输出参数{out}	
-	-
返回值	
-	-

例如：

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

```
adc_software_trigger_enable(ADC_REGULAR_CHANNEL);
```

### 函数 **adc\_end\_of\_conversion\_config**

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

**表 3-23. 函数 **adc\_end\_of\_conversion\_config****

<b>函数名称</b>	adc_end_of_conversion_config
<b>函数原形</b>	void adc_end_of_conversion_config(uint8_t end_selection);
<b>功能描述</b>	转换模式结束配置
<b>先决条件</b>	-
<b>被调用函数</b>	-
输入参数{in}	
<b>end_selection</b>	转换结束模式
<i>ADC_EOC_SET_SEQUENCE</i>	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，则溢出检测禁能
<i>ADC_EOC_SET_CONVERSION</i>	在每个规则组转换结束时，将EOC置1，溢出检测自动使能
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* configure end of conversion mode */
```

```
adc_end_of_conversion_config(ADC_EOC_SET_SEQUENCE);
```

## 函数 `adc_regular_data_read`

函数`adc_inserted_regular_data_read`描述见下表：

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

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

例如：

```
/* read ADC regular group data register */
uint16_t adc_value = 0;
adc_value = adc_regular_data_read();
```

## 函数 `adc_inserted_data_read`

函数`adc_inserted_regular_data_read`描述见下表：

**表 3-25. 函数 `adc_inserted_data_read`**

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

例如：

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

```
uint16_t adc_value = 0;
```

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

### 函数 `adc_watchdog_single_channel_enable`

函数 `adc_watchdog_single_channel_enable` 描述见下表：

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

函数名称	<code>adc_watchdog_single_channel_enable</code>
函数原形	<code>void adc_watchdog_single_channel_enable(uint8_t adc_channel);</code>
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}	
<code>adc_channel</code>	选择ADC通道
<code>ADC_CHANNEL_x</code> ( $x=0..11$ )	ADC通道 $x(x=0..11)$
输出参数{out}	
-	-
返回值	
-	-

例如：

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

```
adc_watchdog_single_channel_enable(ADC_CHANNEL_1);
```

### 函数 `adc_watchdog_group_channel_enable`

函数 `adc_watchdog_group_channel_enable` 描述见下表：

**表 3-27. 函数 `adc_watchdog_group_channel_enable`**

函数名称	<code>adc_watchdog_group_channel_enable</code>
函数原形	<code>void adc_watchdog_group_channel_enable(uint8_t adc_channel_group);</code>
功能描述	配置ADC模拟看门狗在通道组有效
先决条件	-
被调用函数	-
输入参数{in}	
<code>adc_channel_group</code>	通道组使用模拟看门狗
<code>ADC_REGULAR_CHANNEL</code>	规则通道组
<code>ADC_INSERTED_CHANNEL</code>	注入通道组
<code>ADC_REGULAR_</code>	规则和注入通道组

<i>INSERTED_CHANNEL</i>	
输出参数{out}	
-	-
返回值	
-	-

例如：

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

```
adc_watchdog_group_channel_enable(ADC_REGULAR_CHANNEL);
```

### 函数 `adc_watchdog_disable`

函数`adc_watchdog_disable`描述见下表：

**表 3-28. 函数 `adc_watchdog_disable`**

函数名称	<code>adc_watchdog_disable</code>
函数原形	<code>void adc_watchdog_disable(void);</code>
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}	
-	-
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* disable ADC analog watchdog */
```

```
adc_watchdog_disable();
```

### 函数 `adc_watchdog_threshold_config`

函数`adc_watchdog_threshold_config`描述见下表：

**表 3-29. 函数 `adc_watchdog_threshold_config`**

函数名称	<code>adc_watchdog_threshold_config</code>
函数原形	<code>void adc_watchdog_threshold_config(uint16_t low_threshold, uint16_t high_threshold);</code>
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>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 ADC analog watchdog threshold */
```

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

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

函数adc\_oversample\_mode\_config描述见下表：

**表 3-30. 函数 **adc\_oversample\_mode\_config****

<b>函数名称</b>	adc_oversample_mode_config
<b>函数原形</b>	void adc_oversample_mode_config(uint32_t mode, uint16_t shift, uint8_t ratio);
<b>功能描述</b>	配置ADC过采样模式
<b>先决条件</b>	-
<b>被调用函数</b>	-
<b>输入参数{in}</b>	
<b>mode</b>	ADC过采样触发模式
ADC_OVERSAMPLING_ALL_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
<b>输入参数{in}</b>	
<b>shift</b>	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
ADC_OVERSAMPLING_SHIFT_3B	移3位



ADC_OVERSAMPLING_SHIFT_3B	移4位
ADC_OVERSAMPLING_SHIFT_4B	移5位
ADC_OVERSAMPLING_SHIFT_5B	移6位
ADC_OVERSAMPLING_SHIFT_6B	移7位
ADC_OVERSAMPLING_SHIFT_7B	移8位
ADC_OVERSAMPLING_SHIFT_8B	
输入参数{in}	
ratio	ADC过采样率
ADC_OVERSAMPLING_RATIO_MUL2	2x
ADC_OVERSAMPLING_RATIO_MUL4	4x
ADC_OVERSAMPLING_RATIO_MUL8	8x
ADC_OVERSAMPLING_RATIO_MUL16	16x
ADC_OVERSAMPLING_RATIO_MUL32	32x
ADC_OVERSAMPLING_RATIO_MUL64	64x
ADC_OVERSAMPLING_RATIO_MUL128	128x
ADC_OVERSAMPLING_RATIO_MUL256	256x
输出参数{out}	

-	-
返回值	
-	-

Example:

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

```
adc_oversample_mode_config(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 3-31. 函数 `adc_oversample_mode_enable`

函数名称	<code>adc_oversample_mode_enable</code>
函数原形	<code>void adc_oversample_mode_enable(void);</code>
功能描述	使能ADC过采样
先决条件	-
被调用函数	-
输入参数{in}	
-	-
输出参数{out}	
-	-
返回值	
-	-

Example:

```
/* enable ADC oversample mode */
```

```
adc_oversample_mode_enable ();
```

### 函数 `adc_oversample_mode_disable`

函数`adc_oversample_mode_disable`描述见下表:

表 3-32. 函数 `adc_oversample_mode_disable`

函数名称	<code>adc_oversample_mode_disable</code>
函数原形	<code>void adc_oversample_mode_disable(void);</code>
功能描述	禁能ADC过采样
先决条件	-
被调用函数	-
输入参数{in}	
-	-
输出参数{out}	
-	-

返回值	
-	-

Example:

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

### 函数 adc\_flag\_get

函数adc\_flag\_get描述见下表:

表 3-33. 函数 adc\_flag\_get

函数名称	adc_flag_get
函数原形	FlagStatus adc_flag_get(uint32_t adc_periph , uint32_t flag);
功能描述	获取ADC标志位
先决条件	-
被调用函数	-
输入参数{in}	
flag	ADC标志位
ADC_FLAG_WDE	模拟看门狗事件标志位
ADC_FLAG_EOC	组转换结束标志位
ADC_FLAG_EOIC	注入通道组转换结束标志位
ADC_FLAG_STIC	注入通道组转换开始标志位
ADC_FLAG_STRC	规则通道组转换开始标志位
ADC_FLAG_ROVF	规则组数据寄存器溢出标志位
输出参数{out}	
-	-
返回值	
FlagStatus	SET或RESET

例如:

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

### 函数 adc\_flag\_clear

函数adc\_flag\_clear描述见下表:

表 3-34. 函数 adc\_flag\_clear

函数名称	adc_flag_clear
函数原形	void adc_flag_clear(uint32_t flag);
功能描述	清除ADC标志位

先决条件	-
被调用函数	-
输入参数{in}	
flag	ADC标志位
ADC_FLAG_WDE	模拟看门狗事件标志位
ADC_FLAG_EOC	组转换结束标志位
ADC_FLAG_EOIC	注入通道组转换结束标志位
ADC_FLAG_STIC	注入通道组转换开始标志位
ADC_FLAG_STRC	规则通道组转换开始标志位
ADC_FLAG_ROVF	规则组数据寄存器溢出标志位
输出参数{out}	
-	-
返回值	
-	-

例如：

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

```
adc_flag_clear(ADC_FLAG_WDE);
```

### 函数 adc\_interrupt\_enable

函数adc\_interrupt\_enable描述见下表：

表 3-35. 函数 adc\_interrupt\_enable

函数名称	adc_interrupt_enable
函数原形	void adc_interrupt_enable(uint32_t interrupt);
功能描述	ADC中断使能
先决条件	-
被调用函数	-
输入参数{in}	
interrupt	ADC中断
ADC_INT_WDE	模拟看门狗中断
ADC_INT_EOC	组转换结束中断
ADC_INT_EOIC	注入通道组转换结束中断
ADC_INT_ROVF	规则组数据寄存器溢出中断
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* enable ADC analog watchdog interrupt */
```

```
adc_interrupt_enable(ADC_INT_WDE);
```

## 函数 adc\_interrupt\_disable

函数adc\_interrupt\_disable描述见下表：

**表 3-36. 函数 adc\_interrupt\_disable**

函数名称	adc_interrupt_disable
函数原形	void adc_interrupt_disable(uint32_t interrupt);
功能描述	ADC中断禁能
先决条件	-
被调用函数	-
输入参数{in}	
interrupt	ADC中断
ADC_INT_WDE	模拟看门狗中断
ADC_INT_EOC	组转换结束中断
ADC_INT_EOIC	注入通道组转换结束中断
ADC_INT_ROVF	规则组数据寄存器溢出中断
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* disable ADC analog watchdog interrupt */
```

```
adc_interrupt_disable(ADC_INT_WDE);
```

## 函数 adc\_interrupt\_flag\_get

函数adc\_interrupt\_flag\_get描述见下表：

**表 3-37. 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 int_flag);
功能描述	获取ADC中断标志位
先决条件	-
被调用函数	-
输入参数{in}	
int_flag	ADC中断标志位
ADC_INT_FLAG_WDE	模拟看门狗中断标志位
ADC_INT_FLAG_EOC	组转换结束中断标志位
ADC_INT_FLAG_EOIC	注入通道组转换结束中断标志位
ADC_INT_FLAG_	规则组数据寄存器溢出中断标志位

ROVF	
输出参数{out}	
-	-
返回值	
FlagStatus	SET或RESET

例如：

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

```
FlagStatus flag_value;
```

```
flag_value = adc_interrupt_flag_get(ADC_INT_FLAG_WDE);
```

### 函数 `adc_interrupt_flag_clear`

函数`adc_interrupt_flag_clear`描述见下表：

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

函数名称	<code>adc_interrupt_flag_clear</code>
函数原形	<code>void adc_interrupt_flag_clear(uint32_t int_flag);</code>
功能描述	清除ADC中断标志位
先决条件	-
被调用函数	-
输入参数{in}	
<b>int_flag</b>	ADC中断标志位
<code>ADC_INT_FLAG_WDE</code>	模拟看门狗中断标志位
<code>ADC_INT_FLAG_EOC</code>	组转换结束中断标志位
<code>ADC_INT_FLAG_EOIC</code>	注入通道组转换结束中断标志位
<code>ADC_INT_FLAG_ROVF</code>	规则组数据寄存器溢出中断标志位
输出参数{out}	
-	-
返回值	
-	-

例如：

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

```
adc_interrupt_flag_clear(ADC_INT_FLAG_WDE);
```

### 函数 `adc_regular_software_startconv_flag_get`

函数`adc_regular_software_startconv_flag_get`描述见下表：

表 3-39. 函数 `adc_regular_software_startconv_flag_get`

函数名称	<code>adc_regular_software_startconv_flag_get</code>
函数原形	<code>FlagStatus adc_regular_software_startconv_flag_get(void);</code>
功能描述	获取ADC规则组转换开始标志位
先决条件	-
被调用函数	-
输入参数{in}	
-	-
输出参数{out}	
-	-
返回值	
<b>FlagStatus</b>	SET或RESET

例如:

```
/* get the bit state of ADC software start conversion */

FlagStatus flag_value;

flag_value = adc_regular_software_startconv_flag_get();
```

#### 函数 `adc_inserted_software_startconv_flag_get`

函数`adc_inserted_software_startconv_flag_get`描述见下表:

表 3-40. 函数 `adc_regular_software_startconv_flag_get`

函数名称	<code>adc_inserted_software_startconv_flag_get</code>
函数原形	<code>FlagStatus adc_inserted_software_startconv_flag_get(void);</code>
功能描述	获取ADC注入组转换开始标志位
先决条件	-
被调用函数	-
输入参数{in}	
-	-
输出参数{out}	
-	-
返回值	
<b>FlagStatus</b>	SET或RESET

例如:

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

FlagStatus flag_value;

flag_value = adc_inserted_software_startconv_flag_get();
```

### 3.3. CAU

加密处理单元支持处理DES，三重DES或AES（128，192或256）算法，对数据进行加密或解密。CAU寄存器列举在章节[3.3.1](#)，CAU固件库函数介绍在章节[3.3.2](#)。

#### 3.3.1. 外设寄存器说明

CAU寄存器列表如下表所示：

表 3-41. CAU 寄存器

寄存器名称	寄存器描述
CAU_CTL	CAU控制寄存器
CAU_STAT0	CAU状态寄存器0
CAU_DI	CAU数据输入寄存器
CAU_DO	CAU数据输出寄存器
CAU_DMAEN	CAU DMA使能寄存器
CAU_INTEN	CAU中断使能寄存器
CAU_STAT1	CAU状态寄存器1
CAU_INTF	CAU中断标志寄存器
CAU_KEY0H	CAU密钥0高位寄存器
CAU_KEY0L	CAU密钥0低位寄存器
CAU_KEY1H	CAU密钥1高位寄存器
CAU_KEY1L	CAU密钥1低位寄存器
CAU_KEY2H	CAU密钥2高位寄存器
CAU_KEY2L	CAU密钥2低位寄存器
CAU_KEY3H	CAU密钥3高位寄存器
CAU_KEY3L	CAU密钥3低位寄存器
CAU_IV0H	CAU初始化向量0高位寄存器
CAU_IV0L	CAU初始化向量0低位寄存器
CAU_IV1H	CAU初始化向量1高位寄存器
CAU_IV1L	CAU初始化向量1低位寄存器
CAU_GCMCCMCT XSx (x = 0..7)	GCM或CCM模式上下文交换寄存器x
CAU_GCMCTXSx (x = 0..7)	GCM模式上下文交换寄存器x

#### 3.3.2. 外设库函数说明

CAU库函数列表如下表所示：

表 3-42. CAU 库函数

库函数名称	库函数描述
cau_deinit	复位CAU外设



库函数名称	库函数描述
cau_struct_para_init	初始化CAU加密解密结构体
cau_key_struct_para_init	初始化密钥结构体
cau_iv_struct_para_init	初始化矢量结构体
cau_context_struct_para_init	初始化上下文结构体
cau_enable	使能CAU外设
cau_disable	除能CAU外设
cau_dma_enable	使能CAU DMA接口
cau_dma_disable	除能CAU DMA接口
cau_init	初始化CAU
cau_aes_keysize_config	在使用AES算法的情况下配置密钥大小
cau_key_init	初始化密钥参数
cau_iv_init	初始化矢量参数
cau_phase_config	阶段配置
cau_fifo_flush	清除FIFO内容
cau_enable_state_get	返回CAU外设是否使能的状态值
cau_data_write	将数据写入IN FIFO
cau_data_read	返回最近进入OUT FIFO的数据
cau_context_save	上下文交换之前保存上下文
cau_context_restore	上下文交换之后恢复上下文
cau_aes_ecb	ECB模式下使用AES算法加密和解密
cau_aes_cbc	CBC模式下使用AES算法加密和解密
cau_aes_ctr	CTR模式下使用AES算法加密和解密
cau_aes_cfb	CFB模式下使用AES算法加密和解密
cau_aes_ofb	OFB模式下使用AES算法加密和解密
cau_aes_gcm	GCM模式下使用AES算法加密和解密
cau_aes_ccm	CCM模式下使用AES算法加密和解密
cau_tdes_ecb	ECB模式下使用TDES算法加密和解密
cau_tdes_cbc	CBC模式下使用TDES算法加密和解密
cau_des_ecb	ECB模式下使用DES算法加密和解密
cau_des_cbc	CBC模式下使用DES算法加密和解密
cau_interrupt_enable	使能CAU中断
cau_interrupt_disable	除能CAU中断
cau_interrupt_flag_get	获取中断标志
cau_flag_get	获取CAU标志状态

### 结构体 cau\_key\_parameter\_struct

表 3-43. 结构体 cau\_key\_parameter\_struct

成员名称	功能描述
key_0_high	密钥0高位
key_0_low	密钥0低位
key_1_high	密钥1高位

key_1_low	密钥1低位
key_2_high	密钥2高位
key_2_low	密钥2低位
key_3_high	密钥3高位
key_3_low	密钥3低位

### 结构体 `cau_iv_parameter_struct`

表 3-44. 结构体 `cau_iv_parameter_struct`

成员名称	功能描述
iv_0_high	矢量0高位
iv_0_low	矢量0低位
iv_1_high	矢量1高位
iv_1_low	矢量1低位

### 结构体 `cau_context_parameter_struct`

表 3-45. 结构体 `cau_context_parameter_struct`

成员名称	功能描述
ctl_config	当前配置
iv_0_high	矢量0高位
iv_0_low	矢量0低位
iv_1_high	矢量1高位
iv_1_low	矢量1低位
key_0_high	密钥0高位
key_0_low	密钥0低位
key_1_high	密钥1高位
key_1_low	密钥1低位
key_2_high	密钥2高位
key_2_low	密钥2低位
key_3_high	密钥3高位
key_3_low	密钥3低位
gcmccmctxs[8]	GCM或CCM模式上下文
gcmctxs[8]	GCM模式上下文

### 结构体 `cau_parameter_struct`

表 3-46. 结构体 `cau_parameter_struct`

成员名称	功能描述
alg_dir	算法方向
*key	密钥
key_size	密钥字节长度
*iv	初始化矢量
iv_size	初始化矢量字节长度

*input	输入数据
in_length	输入数据字节长度
*aad	附加身份验证数据
aad_size	附加身份验证数据长度

## 函数 cau\_deinit

函数cau\_deinit描述见下表：

**表 3-47. 函数 cau\_deinit**

函数名称	cau_deinit
函数原形	void cau_deinit(void);
功能描述	复位CAU外设
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 CAU peripheral */
```

```
cau_deinit();
```

## 函数 cau\_struct\_para\_init

函数cau\_struct\_para\_init描述见下表：

**表 3-48. 函数 cau\_struct\_para\_init**

函数名称	cau_struct_para_init
函数原形	void cau_struct_para_init(cau_parameter_struct *cau_parameter);
功能描述	初始化CAU加密解密结构体
先决条件	-
被调用函数	-
输入参数{in}	
-	-
输出参数{out}	
cau_parameter	CAU加密解密结构体，参考结构体 <a href="#">表3-46. 结构体cau_parameter_struct</a>
返回值	
-	-

例如：

```
cau_parameter_struct text;

/* initialize CAU encrypt and decrypt parameter struct */

cau_struct_para_init(&text);
```

### 函数 cau\_key\_struct\_para\_init

The description of cau\_key\_struct\_para\_init描述见下表：

表 3-49. 函数 cau\_key\_struct\_para\_init

函数名称	cau_key_struct_para_init
函数原形	void cau_key_struct_para_init(cau_key_parameter_struct *key_initpara);
功能描述	初始化密钥结构体
先决条件	-
被调用函数	-
输入参数{in}	
-	-
输出参数{out}	
key_initpara	密钥结构体，参考结构体 <a href="#">表3-43. 结构体cau_key_parameter_struct</a>
返回值	
-	-

例如：

```
/* initialize the key parameter struct */

cau_key_parameter_struct key_initpara;

cau_key_struct_para_init(&key_initpara);
```

### 函数 cau\_iv\_struct\_para\_init

函数cau\_iv\_struct\_para\_init描述见下表：

表 3-50. 函数 cau\_iv\_struct\_para\_init

函数名称	cau_iv_struct_para_init
函数原形	void cau_iv_struct_para_init(cau_iv_parameter_struct *iv_initpara);
功能描述	初始化矢量结构体
先决条件	-
被调用函数	-
输入参数{in}	
-	-
输出参数{out}	
iv_initpara	矢量结构体，参考结构体 <a href="#">表3-44. 结构体cau_iv_parameter_struct</a>
返回值	
-	-

例如：

```
/* initialize the vectors parameter struct */

cau_iv_parameter_struct iv_initpara;

cau_iv_struct_para_init(&iv_initpara);
```

### 函数 cau\_context\_struct\_para\_init

函数cau\_context\_struct\_para\_init描述见下表：

表 3-51. 函数 cau\_context\_struct\_para\_init

函数名称	cau_context_struct_para_init
函数原形	void cau_context_struct_para_init(cau_context_parameter_struct *cau_context);
功能描述	初始化上下文结构体
先决条件	-
被调用函数	-
输入参数{in}	
-	-
输出参数{out}	
cau_context	上下文结构体，参考结构体 <a href="#">表3-45. 结构体cau_context_parameter_struct</a>
返回值	
-	-

例如：

```
/* initialize the context parameter struct */

cau_context_parameter_struct context_initpara;

cau_context_struct_para_init (&context_initpara);
```

### 函数 cau\_enable

函数cau\_enable描述见下表：

表 3-52. 函数 cau\_enable

函数名称	cau_enable
函数原形	void cau_enable(void);
功能描述	使能CAU外设
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 CAU peripheral */
```

```
cau_enable();
```

### 函数 cau\_disable

函数cau\_disable描述见下表：

**表 3-53. 函数 cau\_disable**

函数名称	cau_disable
函数原形	void cau_disable(void);
功能描述	除能CAU外设
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 CAU peripheral */
```

```
cau_disable();
```

### 函数 cau\_dma\_enable

函数cau\_dma\_enable描述见下表：

**表 3-54. 函数 cau\_dma\_enable**

函数名称	cau_dma_enable
函数原形	void cau_dma_enable(uint32_t dma_req);
功能描述	使能CAU DMA接口
先决条件	-
被调用函数	-
输入参数{in}	
<b>dma_req</b>	使能CAU指定的DMA传输请求方向
<b>CAU_DMA_INFIFO</b>	DMA用于接收数据
<b>CAU_DMA_OUTFIFO</b>	DMA用于发送数据
输出参数{out}	
-	-

返回值	
-	-

例如：

```
/* enable the CAU DMA interface */
cau_dma_enable(CAU_DMA_INFIFO);
```

### 函数 cau\_dma\_disable

函数cau\_dma\_disable描述见下表：

表 3-55. 函数 cau\_dma\_disable

函数名称	cau_dma_disable
函数原形	void cau_dma_disable(uint32_t dma_req);
功能描述	除能CAU DMA接口
先决条件	-
被调用函数	-
输入参数{in}	
dma_req	除能CAU指定的DMA传输请求方向
CAU_DMA_INFIFO	DMA用于接收数据
CAU_DMA_OUTFIFO	DMA用于发送数据
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* disable the CAU DMA interface */
cau_dma_disable(CAU_DMA_INFIFO);
```

### 函数 cau\_init

函数cau\_init描述见下表：

表 3-56. 函数 cau\_init

函数名称	cau_init
函数原形	void cau_init(uint32_t alg_dir, uint32_t algo_mode, uint32_t swapping);
功能描述	初始化CAU
先决条件	-
被调用函数	-
输入参数{in}	
alg_dir	算法方向
CAU_ENCRYPT	加密
CAU_DECRYPT	解密

输入参数{in}	
<b>algo_mode</b>	算法模式选择
CAU_MODE_TDES_ECB	TDES-ECB
CAU_MODE_TDES_CBC	TDES-CBC
CAU_MODE_DES_ECB	DES-ECB
CAU_MODE_DES_CBC	DES-CBC
CAU_MODE_AES_ECB	AES-ECB
CAU_MODE_AES_CBC	AES-CBC
CAU_MODE_AES_CTR	AES-CTR
CAU_MODE_AES_KEY	AES解密密钥准备模式
CAU_MODE_AES_GCM	AES-GCM
CAU_MODE_AES_CCM	AES-CCM
CAU_MODE_AES_CFB	AES-CFB
CAU_MODE_AES_OFB	AES-OFB
输入参数{in}	
<b>swapping</b>	数据交换选择
CAU_SWAPPING_32BIT	无交换
CAU_SWAPPING_16BIT	半字交换
CAU_SWAPPING_8BIT	字节交换
CAU_SWAPPING_1BIT	位交换
输出参数{out}	
-	-
返回值	
-	-

例如：

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

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



**函数 cau\_aes\_keysize\_config**

函数cau\_aes\_keysize\_config描述见下表：

**表 3-57. 函数 cau\_aes\_keysize\_config**

函数名称	cau_aes_keysize_config
函数原形	void cau_aes_keysize_config(uint32_t key_size);
功能描述	在使用AES算法的情况下配置密钥大小
先决条件	-
被调用函数	-
输入参数{in}	
key_size	密钥长度
CAU_KEYSIZE_128BIT	128位密钥长度
CAU_KEYSIZE_192BIT	192位密钥长度
CAU_KEYSIZE_256BIT	256位密钥长度
输出参数{out}	
-	-
返回值	
-	-

例如：

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

**函数 cau\_key\_init**

函数cau\_key\_init描述见下表：

**表 3-58. 函数 cau\_key\_init**

函数名称	cau_key_init
函数原形	void cau_key_init(cau_key_parameter_struct* key_initpara);
功能描述	初始化密钥参数
先决条件	-
被调用函数	-
输入参数{in}	
key_initpara	密钥参数，参考结构体 <a href="#">表3-43. 结构体cau_key_parameter_struct</a>
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* initialize the key parameters */
```

```
cau_key_parameter_struct key_initpara;
```

```
cau_key_init(&key_initpara);
```

### 函数 cau\_iv\_init

函数cau\_iv\_init描述见下表:

表 3-59. 函数 cau\_iv\_init

函数名称	cau_iv_init
函数原形	void cau_iv_init(cau_iv_parameter_struct* iv_initpara);
功能描述	初始化矢量参数
先决条件	-
被调用函数	-
输入参数{in}	
iv_initpara	矢量参数，参考结构体 <a href="#">表3-44. 结构体cau_iv_parameter_struct</a>
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* initialize the vectors parameters */
cau_iv_parameter_struct iv_initpara;
cau_iv_init(&iv_initpara);
```

### 函数 cau\_phase\_config

函数cau\_phase\_config描述见下表:

表 3-60. 函数 cau\_phase\_config

函数名称	cau_phase_config
函数原形	void cau_phase_config(uint32_t phase);
功能描述	阶段配置
先决条件	-
被调用函数	-
输入参数{in}	
phase	GCM或CCM阶段
CAU_PREPARE_PHASE	准备阶段
CAU_AAD_PHASE	附加身份验证数据阶段
CAU_ENCRYPT_DECRYPT_PHASE	加密解密阶段
CAU_TAG_PHASE	标签阶段

输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* select prepare phase */
cau_phase_config(CAU_PREPARE_PHASE);
```

### 函数 cau\_fifo\_flush

函数cau\_fifo\_flush描述见下表：

**表 3-61. 函数 cau\_fifo\_flush**

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

例如：

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

### 函数 cau\_enable\_state\_get

函数cau\_enable\_state\_get描述见下表：

**表 3-62. 函数 cau\_enable\_state\_get**

函数名称	cau_enable_state_get
函数原形	ControlStatus cau_enable_state_get(void);
功能描述	返回CAU外设是否使能的状态值
先决条件	-
被调用函数	-
输入参数{in}	
-	-
输出参数{out}	
-	-

返回值	
ControlStatus	ENABLE或DISABLE

例如:

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

```
ControlStatus state = cau_enable_state_get();
```

### 函数 cau\_data\_write

函数cau\_data\_write描述见下表:

表 3-63. 函数 cau\_data\_write

函数名称	cau_data_write
函数原形	void cau_data_write(uint32_t data);
功能描述	将数据写入IN FIFO
先决条件	-
被调用函数	-
输入参数{in}	
data	所写的数据0 - 0xFFFFFFFF
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* write data to the IN FIFO */
```

```
cau_data_write(0x10);
```

### 函数 cau\_data\_read

函数cau\_data\_read描述见下表:

表 3-64. 函数 cau\_data\_read

函数名称	cau_data_read
函数原形	uint32_t cau_data_read(void);
功能描述	返回最近进入OUT FIFO的数据
先决条件	-
被调用函数	-
输入参数{in}	
-	-
输出参数{out}	
-	-
返回值	
uint32_t	0x0 - 0xFFFFFFFF

例如：

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

uint32_t data = cau_data_read();
```

### 函数 cau\_context\_save

函数cau\_context\_save描述见下表：

**表 3-65. 函数 cau\_context\_save**

函数名称	cau_context_save
函数原形	void cau_context_save(cau_context_parameter_struct *cau_context, cau_key_parameter_struct* key_initpara);
功能描述	上下文交换之前保存上下文
先决条件	-
被调用函数	-
输入参数{in}	
key_initpara	密钥参数，参考结构体 <a href="#">表3-43. 结构体cau_key_parameter_struct</a>
输出参数{out}	
cau_context	上下文结构体，参考结构体 <a href="#">表3-45. 结构体cau_context_parameter_struct</a>
返回值	
-	-

例如：

```
cau_context_parameter_struct context;

cau_key_parameter_struct key;

cau_parameter_struct cau_parameter;

uint32_t keyaddr;

.....

keyaddr = (uint32_t)(cau_parameter->key);

cau_key_struct_para_init(&key);

key.key_1_high = __REV(*(uint32_t*)(keyaddr));

keyaddr += 4U;

key.key_1_low= __REV(*(uint32_t*)(keyaddr));

/* save context before context switching */

cau_context_save(&context, &key);
```

## 函数 cau\_context\_restore

函数cau\_context\_restore描述见下表：

表 3-66. 函数 cau\_context\_restore

函数名称	cau_context_restore
函数原形	void cau_context_restore(cau_context_parameter_struct *cau_context);
功能描述	上下文交换之后恢复上下文
先决条件	-
被调用函数	-
输入参数{in}	
cau_context	上下文结构体，参考结构体 <a href="#">表3-45. 结构体cau_context_parameter_struct</a>
输出参数{out}	
-	-
返回值	
-	-

例如：

```
cau_context_parameter_struct context;

.....

cau_context_save(&context, &key);

.....

/* restore context after context switching */

cau_context_restore (&context);
```

## 函数 cau\_aes\_ecb

函数cau\_aes\_ecb描述见下表：

表 3-67. 函数 cau\_aes\_ecb

函数名称	cau_aes_ecb
函数原形	ErrStatus cau_aes_ecb(cau_parameter_struct *cau_parameter, uint8_t *output);
功能描述	ECB模式下使用AES算法加密和解密
先决条件	-
被调用函数	-
输入参数{in}	
cau_parameter	CAU加密和解密参数，参考结构体 <a href="#">表3-46. 结构体cau_parameter_struct</a>
输出参数{out}	
output	指针指向返回数组
返回值	
ErrStatus	SUCCESS或ERROR

例如：

```
cau_parameter_struct text;

uint8_t encrypt_result[TEXT_SIZE];

ErrStatus status;

.....

cau_struct_para_init(&text);

key_addr = key_select[i];

key_size = keysize[i];

text.alg_dir    = CAU_ENCRYPT;

text.key        = key_addr;

text.key_size   = key_size;

text.input      = plaintext;

text.in_length  = TEXT_SIZE;

/* encryption in ECB mode */

status = cau_aes_ecb(&text, encrypt_result);
```

### 函数 cau\_aes\_cbc

函数cau\_aes\_cbc描述见下表：

**表 3-68. 函数 cau\_aes\_cbc**

函数名称	cau_aes_cbc
函数原形	ErrStatus cau_aes_cbc(cau_parameter_struct *cau_parameter, uint8_t *output);
功能描述	CBC模式下使用AES算法加密和解密
先决条件	-
被调用函数	-
输入参数{in}	
algo_dir	算法方向
CAU_ENCRYPT	加密
CAU_DECRYPT	解密
输入参数{in}	
cau_parameter	CAU加密和解密参数，参考结构体 <a href="#">表3-46. 结构体cau_parameter_struct</a>
输出参数{out}	
output	指针指向返回数组
返回值	
ErrStatus	SUCCESS或ERROR

例如:

```
cau_parameter_struct text;

uint8_t encrypt_result[TEXT_SIZE];

ErrStatus status;

.....

cau_struct_para_init(&text);

key_addr = key_select[i];

key_size = keysize[i];

text.alg_dir    = CAU_ENCRYPT;

text.key        = key_addr;

text.key_size   = key_size;

text.iv         = vectors;

text.input      = plaintext;

text.in_length  = TEXT_SIZE;

/* encryption in CBC mode */

status = cau_aes_cbc(&text, encrypt_result);
```

### 函数 cau\_aes\_ctr

函数cau\_aes\_ctr描述见下表:

表 3-69. 函数 cau\_aes\_ctr

函数名称	cau_aes_ctr
函数原形	ErrStatus cau_aes_ctr(cau_parameter_struct *cau_parameter, uint8_t *output);
功能描述	CTR模式下使用AES算法加密和解密
先决条件	-
被调用函数	-
输入参数{in}	
cau_parameter	CAU加密和解密参数, 参考结构体 <a href="#">表3-46. 结构体cau_parameter_struct</a>
输出参数{out}	
output	指针指向返回数组
返回值	
ErrStatus	SUCCESS或ERROR

例如:

```
cau_parameter_struct text;
```



```
uint8_t encrypt_result[TEXT_SIZE];

ErrStatus status;

.....

cau_struct_para_init(&text);

key_addr = key_select[i];

key_size = keysize[i];

text.alg_dir    = CAU_ENCRYPT;

text.key        = key_addr;

text.key_size   = key_size;

text.iv         = vectors;

text.input      = plaintext;

text.in_length  = TEXT_SIZE;

/* encryption in CTR mode */

status = cau_aes_ctr(&text, encrypt_result);
```

### 函数 cau\_aes\_cfb

函数cau\_aes\_cfb描述见下表：

**表 3-70. 函数 cau\_aes\_cfb**

函数名称	cau_aes_cfb
函数原形	ErrStatus cau_aes_cfb(cau_parameter_struct *cau_parameter, uint8_t *output);
功能描述	CFB模式下使用AES算法加密和解密
先决条件	-
被调用函数	-
输入参数{in}	
cau_parameter	CAU加密和解密参数，参考结构体 <a href="#">表3-46. 结构体cau_parameter_struct</a>
输出参数{out}	
output	指针指向返回数组
返回值	
ErrStatus	SUCCESS或ERROR

例如：

```
cau_parameter_struct cau_cfb_parameter;

uint8_t encrypt_result[TEXT_SIZE];

ErrStatus status;
```

.....

```
cau_struct_para_init(&cau_cfb_parameter);

/* encryption in CFB mode */

cau_cfb_parameter.alg_dir    = CAU_ENCRYPT;

cau_cfb_parameter.key        = (uint8_t *)key_128;

cau_cfb_parameter.key_size   = KEY_SIZE;

cau_cfb_parameter.iv         = (uint8_t *)vectors;

cau_cfb_parameter.iv_size    = IV_SIZE;

cau_cfb_parameter.input      = (uint8_t *)plaintext;

cau_cfb_parameter.in_length = PLAINTEXT_SIZE;

status = cau_aes_cfb(&cau_cfb_parameter, encrypt_result);
```

### 函数 cau\_aes\_ofb

函数cau\_aes\_ofb描述见下表:

表 3-71. 函数 cau\_aes\_ofb

函数名称	cau_aes_ofb
函数原形	ErrStatus cau_aes_ofb(cau_parameter_struct *cau_parameter, uint8_t *output);
功能描述	OFB模式下使用AES算法加密和解密
先决条件	-
被调用函数	-
输入参数{in}	
cau_parameter	CAU加密和解密参数, 参考结构体 <a href="#">表3-46. 结构体cau_parameter_struct</a>
输出参数{out}	
output	指针指向返回数组
返回值	
ErrStatus	SUCCESS或ERROR

例如:

```
cau_parameter_struct cau_ofb_parameter;

uint8_t encrypt_result[TEXT_SIZE];

ErrStatus status;

.....

cau_struct_para_init(&cau_ofb_parameter);

/* encryption in OFB mode */
```

```

cau_ofb_parameter.alg_dir    = CAU_ENCRYPT;

cau_ofb_parameter.key        = (uint8_t *)key_128;

cau_ofb_parameter.key_size   = KEY_SIZE;

cau_ofb_parameter.iv         = (uint8_t *)vectors;

cau_ofb_parameter.iv_size    = IV_SIZE;

cau_ofb_parameter.input      = (uint8_t *)plaintext;

cau_ofb_parameter.in_length  = PLAINTEXT_SIZE;

status = cau_aes_ofb(&cau_ofb_parameter, encrypt_result);

```

### 函数 cau\_aes\_gcm

函数cau\_aes\_gcm描述见下表：

**表 3-72. 函数 cau\_aes\_gcm**

函数名称	cau_aes_gcm
函数原形	ErrStatus cau_aes_gcm(cau_parameter_struct *cau_parameter, uint8_t *output, uint8_t *tag);
功能描述	GCM模式下使用AES算法加密和解密
先决条件	-
被调用函数	-
输入参数{in}	
cau_parameter	CAU加密和解密参数，参考结构体 <a href="#">表3-46. 结构体cau_parameter_struct</a>
输出参数{out}	
output	指针指向返回数组
输出参数{out}	
tag	指针指向返回标签数组
返回值	
ErrStatus	SUCCESS或ERROR

例如：

```

cau_parameter_struct cau_gcm_parameter;

uint8_t encrypt_result[TEXT_SIZE];

uint8_t gcm_tag[GCM_TAG_SIZE];

ErrStatus status;

.....

cau_struct_para_init(&cau_gcm_parameter);

/* encryption in GCM mode */

```

```

cau_gcm_parameter.alg_dir    = CAU_ENCRYPT;

cau_gcm_parameter.key        = (uint8_t *)key_128;

cau_gcm_parameter.key_size   = KEY_SIZE;

cau_gcm_parameter.iv         = (uint8_t *)vectors;

cau_gcm_parameter.iv_size    = IV_SIZE;

cau_gcm_parameter.input      = (uint8_t *)plaintext;

cau_gcm_parameter.in_length  = PLAINTEXT_SIZE;

cau_gcm_parameter.aad        = (uint8_t *)aadmessage;

cau_gcm_parameter.aad_size = AAD_SIZE;

status = cau_aes_gcm(&cau_gcm_parameter, encrypt_result, gcm_tag);

```

### 函数 cau\_aes\_ccm

函数cau\_aes\_ccm描述见下表：

**表 3-73. 函数 cau\_aes\_ccm**

函数名称	cau_aes_ccm
函数原形	ErrStatus cau_aes_ccm(cau_parameter_struct *cau_parameter, uint8_t *output, uint8_t tag[], uint32_t tag_size, uint8_t aad_buf[]);
功能描述	CCM模式下使用AES算法加密和解密
先决条件	-
被调用函数	-
输入参数{in}	
cau_parameter	CAU加密和解密参数，参考结构体 <a href="#">表3-46. 结构体cau_parameter_struct</a>
输入参数{in}	
tag_size	标签字节长度
输出参数{out}	
output	指针指向返回数组
输出参数{out}	
tag	指针指向返回标签数组
输出参数{out}	
aad_buf	指针指向用户定义的用于填充附加身份验证数据的数组
返回值	
ErrStatus	SUCCESS或ERROR

例如：

```

cau_parameter_struct cau_ccm_parameter;

uint8_t encrypt_result[TEXT_SIZE];

uint8_t ccm_tag[CCM_TAG_SIZE];

```

```
uint8_t aad_buf[AAD_SIZE + 21];

ErrStatus status;

.....

cau_struct_para_init(&cau_ccm_parameter);

/* encryption in CCM mode */

cau_ccm_parameter.alg_dir    = CAU_ENCRYPT;

cau_ccm_parameter.key        = (uint8_t *)ccm_key_128;

cau_ccm_parameter.key_size   = KEY_SIZE;

cau_ccm_parameter.iv         = (uint8_t *)ccm_vectors;

cau_ccm_parameter.iv_size    = CCM_IV_SIZE;

cau_ccm_parameter.input      = (uint8_t *)plaintext;

cau_ccm_parameter.in_length  = PLAINTEXT_SIZE;

cau_ccm_parameter.aad        = (uint8_t *)aadmessage;

cau_ccm_parameter.aad_size   = AAD_SIZE;

status = cau_aes_ccm(&cau_ccm_parameter, encrypt_result, ccm_tag, CCM_TAG_SIZE,
(uint8_t *)aad_buf);
```

### 函数 cau\_tdes\_ecb

函数cau\_tdes\_ecb描述见下表：

**表 3-74. 函数 cau\_tdes\_ecb**

函数名称	cau_tdes_ecb
函数原形	ErrStatus cau_tdes_ecb(cau_parameter_struct *cau_parameter, uint8_t *output);
功能描述	ECB模式下使用TDES算法加密和解密
先决条件	-
被调用函数	-
输入参数{in}	
<b>cau_parameter</b>	CAU加密和解密参数，参考结构体 <a href="#">表3-46. 结构体cau_parameter_struct</a>
输出参数{out}	
<b>output</b>	指针指向返回数组
返回值	
<b>ErrStatus</b>	SUCCESS或ERROR

例如：

```
cau_parameter_struct text;
```

```
uint8_t encrypt_result[DATA_SIZE];

ErrStatus status;

.....

cau_struct_para_init(&text);

text.alg_dir   = CAU_ENCRYPT;

text.key       = tdes_key;

text.input     = plaintext;

text.in_length = DATA_SIZE;

/* encryption in ECB mode */

status = cau_tdes_ecb(&text, encrypt_result);
```

### 函数 cau\_tdes\_cbc

函数cau\_tdes\_cbc描述见下表：

**表 3-75. 函数 cau\_tdes\_cbc**

函数名称	cau_tdes_cbc
函数原形	ErrStatus cau_tdes_cbc(cau_parameter_struct *cau_parameter, uint8_t *output);
功能描述	CBC模式下使用TDES算法加密和解密
先决条件	-
被调用函数	-
输入参数{in}	
cau_parameter	CAU加密和解密参数，参考结构体 <a href="#">表3-46. 结构体cau_parameter_struct</a>
输出参数{out}	
output	指针指向返回数组
返回值	
ErrStatus	SUCCESS或ERROR

例如：

```
cau_parameter_struct text;

uint8_t encrypt_result[DATA_SIZE];

ErrStatus status;

.....

cau_struct_para_init(&text);

text.alg_dir   = CAU_ENCRYPT;

text.key       = tdes_key;
```

```
text.iv          = vectors;

text.input       = plaintext;

text.in_length = DATA_SIZE;

/* encryption in CBC mode */

status = cau_tdes_cbc(&text, encrypt_result);
```

### 函数 cau\_des\_ecb

函数cau\_des\_ecb描述见下表：

表 3-76. 函数 cau\_des\_ecb

函数名称	cau_des_ecb
函数原形	ErrStatus cau_des_ecb(cau_parameter_struct *cau_parameter, uint8_t *output);
功能描述	ECB模式下使用DES算法加密和解密
先决条件	-
被调用函数	-
输入参数{in}	
cau_parameter	CAU加密和解密参数，参考结构体 <a href="#">表3-46. 结构体cau_parameter_struct</a>
输出参数{out}	
output	指针指向返回数组
返回值	
ErrStatus	SUCCESS或ERROR

例如：

```
cau_parameter_struct text;

uint8_t encrypt_result[DATA_SIZE];

ErrStatus status;

.....

cau_struct_para_init(&text);

text.alg_dir    = CAU_ENCRYPT;

text.key        = des_key;

text.input      = plaintext;

text.in_length = DATA_SIZE;

/* encryption in ECB mode */

status = cau_des_ecb(&text, encrypt_result);
```

**函数 cau\_des\_cbc**

函数cau\_des\_cbc描述见下表：

**表 3-77. 函数 cau\_des\_cbc**

函数名称	cau_des_cbc
函数原形	ErrStatus cau_des_cbc(cau_parameter_struct *cau_parameter, uint8_t *output);
功能描述	CBC模式下使用DES算法加密和解密
先决条件	-
被调用函数	-
输入参数{in}	
cau_parameter	CAU加密和解密参数，参考结构体 <a href="#">表3-46. 结构体cau_parameter_struct</a>
输出参数{out}	
output	指针指向返回数组
返回值	
ErrStatus	SUCCESS或ERROR

例如：

```
cau_parameter_struct text;

uint8_t encrypt_result[DATA_SIZE];

ErrStatus status;

.....

cau_struct_para_init(&text);

text.alg_dir   = CAU_ENCRYPT;

text.key       = des_key;

text.iv        = vectors;

text.input     = plaintext;

text.in_length = DATA_SIZE;

/* encryption in CBC mode */

status = cau_des_cbc(&text, encrypt_result);
```

**函数 cau\_interrupt\_enable**

函数cau\_interrupt\_enable描述见下表：

**表 3-78. 函数 cau\_interrupt\_enable**

函数名称	cau_interrupt_enable
函数原形	void cau_interrupt_enable(uint32_t interrupt)



功能描述	使能CAU中断
先决条件	-
被调用函数	-
输入参数{in}	
interrupt	使能CAU指定中断源
CAU_INT_INFIFO	输入FIFO中断
CAU_INT_OUTFIFO	输出FIFO中断
输出参数{out}	
-	-
返回值	
-	-

例如：

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

### 函数 cau\_interrupt\_disable

函数cau\_interrupt\_disable描述见下表：

表 3-79. 函数 cau\_interrupt\_disable

函数名称	cau_interrupt_disable
函数原形	void cau_interrupt_disable(uint32_t interrupt)
功能描述	除能CAU中断
先决条件	-
被调用函数	-
输入参数{in}	
interrupt	使能CAU指定中断源
CAU_INT_INFIFO	输入FIFO中断
CAU_INT_OUTFIFO	输出FIFO中断
输出参数{out}	
-	-
返回值	
-	-

例如：

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

### 函数 cau\_interrupt\_flag\_get

函数cau\_interrupt\_flag\_get描述见下表：

表 3-80. 函数 `cau_interrupt_flag_get`

函数名称	<code>cau_interrupt_flag_get</code>
函数原形	<code>FlagStatus cau_interrupt_flag_get(uint32_t interrupt)</code>
功能描述	获取中断标志
先决条件	-
被调用函数	-
输入参数{in}	
<b>interrupt</b>	CAU中断标志
<code>CAU_INT_FLAG_INFIFO</code>	输入FIFO中断
<code>CAU_INT_FLAG_OUTFIFO</code>	输出FIFO中断
输出参数{out}	
-	-
返回值	
<b>FlagStatus</b>	SET or RESET

例如：

```
/* get the CAU interrupt flag status */
```

```
FlagStatus status;
```

```
status = cau_interrupt_flag_get(CAU_INT_FLAG_INFIFO);
```

### 函数 `cau_flag_get`

函数`cau_flag_get`描述见下表：

表 3-81. 函数 `cau_flag_get`

函数名称	<code>cau_flag_get</code>
函数原形	<code>FlagStatus cau_flag_get(uint32_t flag)</code>
功能描述	获取CAU标志状态
先决条件	-
被调用函数	-
输入参数{in}	
<b>flag</b>	CAU标志状态
<code>CAU_FLAG_INFIFO_EMPTY</code>	输入FIFO空标志
<code>CAU_FLAG_INFIFO_NO_FULL</code>	输入FIFO未滿标志
<code>CAU_FLAG_OUTFIFO_NO_EMPTY</code>	输出FIFO非空标志
<code>CAU_FLAG_OUTFIFO_FULL</code>	输出FIFO滿标志
<code>CAU_FLAG_BUSY</code>	CAU内核忙标志

<code>CAU_FLAG_INFIFO</code>	输入FIFO标志状态
<code>CAU_FLAG_OUTFIFO</code>	输出FIFO标志状态
输出参数{out}	
-	-
返回值	
<b>FlagStatus</b>	SET或RESET

例如:

```
/* get the CAU flag status */
```

```
FlagStatus status;
```

```
status = cau_flag_get(CAU_FLAG_INFIFO_EMPTY);
```

### 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	<code>crc_data_register_reset</code>
函数原形	<code>void crc_data_register_reset(void);</code>
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	<code>crc_data_register_read</code>
函数原形	<code>uint32_t crc_data_register_read(void);</code>
功能描述	读数据寄存器
先决条件	-
被调用函数	-
输入参数{in}	
-	-
输出参数{out}	
-	-
返回值	
<code>uint32_t</code>	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	<code>crc_free_data_register_read</code>
函数原形	<code>uint8_t crc_free_data_register_read(void);</code>
功能描述	读独立数据寄存器
先决条件	-
被调用函数	-
输入参数{in}	
-	-
输出参数{out}	
-	-
返回值	
<code>uint8_t</code>	从独立数据寄存器读取的8位数据 (0-0xFF)

例如：

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

uint8_t crc_value = 0;

crc_value = crc_free_data_register_read();
```

### 函数 `crc_free_data_register_write`

函数 `crc_free_data_register_write` 描述见下表：

表 3-88. 函数 `crc_free_data_register_write`

函数名称	<code>crc_free_data_register_write</code>
函数原形	<code>void crc_free_data_register_write(uint8_t free_data);</code>
功能描述	写独立数据寄存器
先决条件	-
被调用函数	-
输入参数{in}	
<code>free_data</code>	设定的8位数据
输出参数{out}	
-	-
返回值	

-	-
---	---

例如:

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

### 函数 `crc_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	<code>crc_single_data_calculate</code>
函数原形	<code>uint32_t crc_single_data_calculate(uint32_t sdata);</code>
功能描述	CRC 计算一个 32 位数据
先决条件	-
被调用函数	-
输入参数{in}	
<b>sdata</b>	设定的 32 位数据
输出参数{out}	
-	-
返回值	
<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	<code>crc_block_data_calculate</code>
函数原形	<code>uint32_t crc_block_data_calculate(uint32_t array[], uint32_t size);</code>
功能描述	CRC 计算一个 32 位数组
先决条件	-
被调用函数	-
输入参数{in}	
<b>array</b>	32 位数据数组的指针
输入参数{in}	
<b>size</b>	数据长度

输出参数{out}	
-	-
返回值	
uint32_t	32位CRC计算结果 (0-0xFFFFFFFF)

例如:

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

#define BUFFER_SIZE    6

uint32_t valcrc = 0;

static const uint32_t data_buffer[BUFFER_SIZE] = {

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

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

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

### 3.5.1. 外设寄存器说明

DBG寄存器列表如下表所示:

表 3-91. DBG 寄存器

寄存器名称	寄存器描述
DBG_ID	DBG ID寄存器
DBG_CTL0	DBG控制寄存器0
DBG_CTL1	DBG控制寄存器1
DBG_CTL2	DBG控制寄存器2

### 3.5.2. 外设库函数说明

DBG库函数列表如下表所示:

表 3-92. DBG 库函数

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



库函数名称	库函数描述
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 3-93. 枚举类型 `dbg_periph_enum`

成员名称	功能描述
DBG_TIMER1_HOLD	当内核停止时，保持TIMER1计数器计数值不变
DBG_TIMER2_HOLD	当内核停止时，保持TIMER2计数器计数值不变
DBG_TIMER3_HOLD	当内核停止时，保持TIMER3计数器计数值不变
DBG_TIMER4_HOLD	当内核停止时，保持TIMER4计数器计数值不变
DBG_TIMER5_HOLD	当内核停止时，保持TIMER5计数器计数值不变
DBG_RTC_HOLD	当内核停止时，保持RTC计数器计数值不变
DBG_FWDGT_HOLD	当内核停止时，保持FWDGT计数器时钟
DBG_WWDGT_HOLD	当内核停止时，保持WWDGT计数器时钟
DBG_I2C0_HOLD	当内核停止时，保持I2C0的SMBUS状态不变，用于调试
DBG_I2C1_HOLD	当内核停止时，保持I2C1的SMBUS状态不变，用于调试
DBG_TIMER0_HOLD	当内核停止时，保持TIMER0计数器计数值不变
DBG_TIMER15_HOLD	当内核停止时，保持TIMER15计数器计数值不变
DBG_TIMER16_HOLD	当内核停止时，保持TIMER16计数器计数值不变

### 函数 `dbg_deinit`

函数`dbg_deinit`描述见下表：

表 3-94. 函数 `dbg_deinit`

函数名称	<code>dbg_deinit</code>
函数原形	<code>void dbg_deinit(void);</code>
功能描述	复位DEBUG模块
先决条件	-
被调用函数	-
输入参数{in}	
-	-
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* reset the DBG */
```

```
dbg_deinit();
```

**函数 dbg\_id\_get**

函数dbg\_id\_get描述见下表：

**表 3-95. 函数 dbg\_id\_get**

函数名称	dbg_id_get
函数原形	uint32_t dbg_id_get(void);
功能描述	读DBG_ID寄存器
先决条件	-
被调用函数	-
输入参数{in}	
-	-
输出参数{out}	
-	-
返回值	
uint32_t	DBG ID (0-0xFFFFFFFF)

例如：

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

```
uint32_t id_value = 0;
```

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

**函数 dbg\_low\_power\_enable**

函数dbg\_low\_power\_enable描述见下表：

**表 3-96. 函数 dbg\_low\_power\_enable**

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

-	-
---	---

例如：

```
/* keep debugger connection during sleep mode */
```

```
dbg_low_power_enable(DBG_LOW_POWER_SLEEP);
```

### 函数 **dbg\_low\_power\_disable**

函数dbg\_low\_power\_disable描述见下表：

**表 3-97. 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}	
dbg_low_power	低功耗模式调试保持
DBG_LOW_POWER_SLEEP	在睡眠模式下，保持调试器连接，可进行调试
DBG_LOW_POWER_DEEPSLEEP	在深度睡眠模式下，保持调试器连接，可进行调试
DBG_LOW_POWER_STANDBY	在待机模式下，保持调试器连接，可进行调试
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* do not keep debugger connection during sleep mode */
```

```
dbg_low_power_disable(DBG_LOW_POWER_SLEEP);
```

### 函数 **dbg\_periph\_enable**

函数dbg\_periph\_enable描述见下表：

**表 3-98. 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	-
输入参数{in}	

<b>dbg_periph</b>	参考 <a href="#">表3-93. 枚举类型dbg_periph_enum</a>
<i>DBG_FWDGT_HOLD</i> <i>D</i>	当内核停止时，保持FWDGT计数器时钟
<i>DBG_WWDGT_HOLD</i> <i>LD</i>	当内核停止时，保持WWDGT计数器时钟
<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> <i>D</i>	当内核停止时，保持TIMERx计数器计数值不变（x=0,1,2,3,4,5,15,16）
<i>DBG_RTC_HOLD</i>	当内核停止时，保持RTC计数器计数值不变
<b>输出参数{out}</b>	
-	-
<b>返回值</b>	
-	-

例如：

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

```
dbg_periph_enable(DBG_TIMER0_HOLD);
```

### 函数 dbg\_periph\_disable

函数dbg\_periph\_disable描述见下表：

**表 3-99. 函数 dbg\_periph\_disable**

<b>函数名称</b>	dbg_periph_disable
<b>函数原形</b>	void dbg_periph_disable(dbg_periph_enum dbg_periph);
<b>功能描述</b>	禁能外设的MCU调试保持功能
<b>先决条件</b>	-
<b>被调用函数</b>	-
<b>输入参数{in}</b>	
<b>dbg_periph</b>	参考 <a href="#">表3-93. 枚举类型dbg_periph_enum</a>
<i>DBG_FWDGT_HOLD</i> <i>D</i>	当内核停止时，保持FWDGT计数器时钟
<i>DBG_WWDGT_HOLD</i> <i>LD</i>	当内核停止时，保持WWDGT计数器时钟
<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> <i>D</i>	当内核停止时，保持TIMERx计数器计数值不变（x=0,1,2,3,4,5,15,16）
<i>DBG_RTC_HOLD</i>	当内核停止时，保持RTC计数器计数值不变
<b>输出参数{out}</b>	
-	-
<b>返回值</b>	
-	-

例如：

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

```
dbg_periph_disable(DBG_TIMER0_HOLD);
```

### 函数 **dbg\_trace\_pin\_enable**

函数dbg\_trace\_pin\_enable描述见下表:

表 3-100. 函数 **dbg\_trace\_pin\_enable**

函数名称	dbg_trace_pin_enable
函数原形	void dbg_trace_pin_enable(void);
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 3-101. 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 3-102. 函数 `dbg_trace_pin_mode_set`

函数名称	<code>dbg_trace_pin_mode_set</code>
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_DATASIZE_1</code>	跟踪引脚用于同步模式且数据长度为1
<code>TRACE_MODE_SYNC_DATASIZE_2</code>	跟踪引脚用于同步模式且数据长度为2
<code>TRACE_MODE_SYNC_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 used for async mode */
```

```
dbg_trace_pin_mode_set(TRACE_MODE_ASYNC);
```

## 3.6. 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 [3.6.1](#)，DCI固件库函数列举在章节 [3.6.2](#)。

### 3.6.1. 外设寄存器说明

DCI寄存器列表如下表所示：

表 3-103. DCI 寄存器

寄存器名称	寄存器描述
DCI_CTL	DCI控制寄存器

寄存器名称	寄存器描述
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

### 3.6.2. 外设库函数说明

DCI库函数列表如下表所示：

**表 3-104. DCI 库函数**

库函数名称	库函数描述
dc_i_deinit	复位DCI
dc_i_struct_para_init	初始化DCI参数结构体为默认值
dc_i_init	初始化DCI寄存器
dc_i_enable	使能DCI功能
dc_i_disable	除能DCI功能
dc_i_capture_enable	使能DCI捕获功能
dc_i_capture_disable	除能DCI捕获功能
dc_i_jpeg_enable	使能DCI JPEG模式
dc_i_jpeg_disable	除能DCI JPEG模式
dc_i_crop_window_enable	使能裁剪窗口功能
dc_i_crop_window_disable	除能裁剪窗口功能
dc_i_crop_window_config	配置DCI裁剪窗口功能
dc_i_embedded_sync_enable	使能内嵌同步模式
dc_i_embedded_sync_disable	除能内嵌同步模式
dc_i_sync_codes_config	在内嵌同步模式下配置同步码
dc_i_sync_codes_unmask_config	在内嵌同步模式下配置非屏蔽同步码
dc_i_data_read	读取DCI数据寄存器
dc_i_flag_get	获取指定标志
dc_i_interrupt_enable	使能指定的DCI中断
dc_i_interrupt_disable	除能指定的DCI中断
dc_i_interrupt_flag_get	获取指定的中断标志
dc_i_interrupt_flag_clear	清除指定的中断标志

## 结构体 dci\_parameter\_struct

表 3-105. 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 3-106. 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\_struct\_para\_init

函数dci\_struct\_para\_init描述见下表:



表 3-107. 函数 dci\_struct\_para\_init

函数名称	dci_struct_para_init
函数原形	void dci_struct_para_init(dci_parameter_struct* init_struct);
功能描述	初始化DCI参数结构体为默认值
先决条件	-
被调用函数	-
输入参数{in}	
dci_struct	DCI参数初始化结构体的地址
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* initialize DCI struct parameter */
dci_parameter_struct dci_struct;
dci_struct_para_init(&dci_struct);
```

### 函数 dci\_init

函数dci\_init描述见下表：

表 3-108. 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参数初始化结构体的地址
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.hsycn_polarity = DCI_HSYN_POLARITY_LOW;
```

```
dc_i_struct.vsync_polarity = DCI_VSYNC_POLARITY_LOW;
```

```
dc_i_struct.frame_rate = DCI_FRAME_RATE_ALL;
```

```
dc_i_struct.interface_format = DCI_INTERFACE_FORMAT_8BITS;
```

```
dc_i_init(&dc_i_struct);
```

## 函数 `dc_i_enable`

函数`dc_i_enable`描述见下表：

**表 3-109. 函数 `dc_i_enable`**

函数名称	dc_i_enable
函数原形	void dc_i_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 */
```

```
dc_i_enable();
```

## 函数 `dc_i_disable`

函数`dc_i_disable`描述见下表：

**表 3-110. 函数 `dc_i_disable`**

函数名称	dc_i_disable
函数原形	void dc_i_disable(void);
功能描述	除能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：

```
/* disable DCI function */
```

```
dc_i_disable();
```

### 函数 **dc\_i\_capture\_enable**

函数dc\_i\_capture\_enable描述见下表：

**表 3-111. 函数 dc\_i\_capture\_enable**

函数名称	dc_i_capture_enable
函数原形	void dc_i_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 */
```

```
dc_i_capture_enable();
```

### 函数 **dc\_i\_capture\_disable**

函数dc\_i\_capture\_disable描述见下表：

**表 3-112. 函数 dc\_i\_capture\_disable**

函数名称	dc_i_capture_disable
函数原形	void dc_i_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 */
```

```
dc_i_capture_disable();
```

### 函数 `dc_i_jpeg_enable`

函数`dc_i_jpeg_enable`描述见下表:

表 3-113. 函数 `dc_i_jpeg_enable`

函数名称	dc_i_jpeg_enable
函数原形	void dc_i_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 */
```

```
dc_i_jpeg_enable();
```

### 函数 `dc_i_jpeg_disable`

函数`dc_i_jpeg_disable`描述见下表:

表 3-114. 函数 `dc_i_jpeg_disable`

函数名称	dc_i_jpeg_disable
函数原形	void dc_i_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 */
```

```
dcj_peg_disable();
```

### 函数 `dcj_crop_window_enable`

函数`dcj_crop_window_enable`描述见下表:

表 3-115. 函数 `dcj_crop_window_enable`

函数名称	<code>dcj_crop_window_enable</code>
函数原形	<code>void dcj_crop_window_enable(void);</code>
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 */
```

```
dcj_crop_window_enable();
```

### 函数 `dcj_crop_window_disable`

函数`dcj_crop_window_disable`描述见下表:

表 3-116. 函数 `dcj_crop_window_disable`

函数名称	<code>dcj_crop_window_disable</code>
函数原形	<code>void dcj_crop_window_disable(void);</code>
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 */
```

```
dcj_crop_window_disable();
```

**函数 dci\_crop\_window\_config**

函数dci\_crop\_window\_config描述见下表：

**表 3-117. 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 3-118. 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	-
输入参数{in}	
-	-
输出参数{out}	
-	-
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 3-119. 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	-
输入参数{in}	
-	-
输出参数{out}	
-	-
返回值	
-	-

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 3-120. 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	-
输入参数{in}	
frame_start	在内嵌同步模式帧起始码
输入参数{in}	
line_start	在内嵌同步模式行起始码
输入参数{in}	
line_end	在内嵌同步模式行终止码
输入参数{in}	

<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 3-121. 函数 dci\_sync\_codes\_unmask\_config**

<b>函数名称</b>	dci_sync_codes_unmask_config
<b>函数原形</b>	void dci_sync_codes_unmask_config(uint8_t frame_start, uint8_t line_start, uint8_t line_end, uint8_t frame_end);
<b>功能描述</b>	在内嵌同步模式下配置非屏蔽同步码
<b>先决条件</b>	-
<b>被调用函数</b>	-
<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 3-122. 函数 dci\_data\_read**

<b>函数名称</b>	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 3-123. 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	-
被调用函数	-
输入参数{in}	
flag	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
输出参数{out}	
-	-
返回值	
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 3-124. 函数 dci\_interrupt\_enable**

函数名称	dci_interrupt_enable
函数原形	void dci_interrupt_enable(uint32_t interrupt);
功能描述	使能指定的DCI中断
先决条件	-
被调用函数	-
输入参数{in}	
interrupt	DCI中断
DCI_INT_EF	帧结束中断标志
DCI_INT_OVR	FIFO溢出中断标志
DCI_INT_ESE	内嵌码同步错误中断标志
DCI_INT_VSYNC	垂直同步中断标志
DCI_INT_EL	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 3-125. 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}	
interrupt	DCI interrupt
DCI_INT_EF	帧结束中断标志
DCI_INT_OVR	FIFO溢出中断标志
DCI_INT_ESE	内嵌码同步错误中断标志
DCI_INT_VSYNC	垂直同步中断标志
DCI_INT_EL	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 3-126. 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	-
输入参数{in}	
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
输出参数{out}	
-	-
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 3-127. 函数 dci\_interrupt\_flag\_clear

函数名称	dci_interrupt_flag_clear
函数原形	void dci_interrupt_flag_clear(uint32_t interrupt);
功能描述	清除指定的中断标志
先决条件	-

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

例如：

```
/* clear specified interrupt flag */
```

```
dci_interrupt_flag_clear(DCI_INT_FLAG_EF);
```

## 3.7. 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[3.7.1](#)描述了DMA的寄存器列表，章节[3.7.2](#)对DMA库函数进行说明。

### 3.7.1. 外设寄存器说明

DMA寄存器列表如下表所示：

表 3-128. DMA 寄存器

寄存器名称	寄存器描述
DMA_INTF0	中断标志位寄存器 0
DMA_INTF1	中断标志位寄存器 1
DMA_INTC0	中断标志位清除器 0
DMA_INTC1	中断标志位清除器 1
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 外设基地址寄存器
DMA_CHxM0ADDR (x=0..7)	通道 x 存储器 0 基地址寄存器

寄存器名称	寄存器描述
DMA_CHxM1ADDR (x=0..7)	通道 x 存储器 1 基地址寄存器
DMA_CHxFCTL	通道 x FIFO 控制寄存器
DMA_SSTAT	安全状态寄存器
DMA_SSC	安全状态清除寄存器
DMA_CHxSCTL	通道 x 安全控制寄存器

### 3.7.2. 外设库函数说明

DMA库函数列表如下表所示：

**表 3-129. DMA 库函数**

库函数名称	库函数描述
dma_deinit	复位外设 DMAx 的通道 y 的所有寄存器
dma_single_struct_para_struct_init	将单数据传输模式结构体中所有参数初始化为默认值
dma_multi_struct_para_struct_init	将突发传输模式结构体中所有参数初始化为默认值
dma_single_data_mode_init	DMAx 的通道 y 单数据传输模式初始化
dma_multi_data_mode_init	DMAx 的通道 y 突发传输模式初始化
dma_periph_address_config	DMAx 通道 y 传输的外设基地址配置
dma_memory_address_config	DMAx 通道 y 传输的存储器基地址配置
dma_transfer_number_config	配置 DMAx 通道 y 需要传输的数据量
dma_transfer_number_get	获取 DMAx 通道 y 剩余需要传输的数据量
dma_priority_config	DMAx 通道 y 的传输软件优先级配置
dma_memory_burst_beats_config	DMAx 通道 y 突发传输存储器节拍数配置
dma_periph_burst_beats_config	DMAx 通道 y 突发传输外设节拍数配置
dma_memory_width_config	DMAx 通道 y 传输的存储器数据宽度配置
dma_periph_width_config	DMAx 通道 y 传输的外设数据宽度配置
dma_memory_address_generation_config	DMAx 通道 y 存储器地址生成配置
dma_peripheral_address_generation_config	DMAx 通道 y 外设地址生成配置
dma_circulation_enable	DMAx 的通道 y 循环模式使能
dma_circulation_disable	DMAx 的通道 y 循环模式禁能
dma_channel_enable	外设 DMAx 的通道 y 传输使能
dma_channel_disable	外设 DMAx 的通道 y 传输禁能
dma_transfer_direction_config	DMAx 通道 y 的传输方向配置
dma_switch_buffer_mode_config	DMAx 通道 y 的存储切换模式配置
dma_using_memory_get	获取当前正在使用的存储地址
dma_channel_subperipheral_select	DMAx 通道 y 的外设请求选择
dma_flow_controller_config	DMAx 通道 y 的传输控制器选择
dma_switch_buffer_mode_enable	DMAx 通道 y 存储切换模式使能
dma_switch_buffer_mode_disable	DMAx 通道 y 存储切换模式禁能

库函数名称	库函数描述
<code>dma_fifo_status_get</code>	获取 DMAx 通道 y 的 FIFO 状态
<code>dma_flag_get</code>	获取 DMAx 通道 y 的标志位
<code>dma_flag_clear</code>	清除 DMAx 通道 y 的标志位
<code>dma_interrupt_flag_get</code>	获取 DMAx 通道 y 的中断标志位
<code>dma_interrupt_flag_clear</code>	清除 DMAx 通道 y 的中断标志位
<code>dma_interrupt_enable</code>	DMAx 通道 y 中断使能
<code>dma_interrupt_disable</code>	DMAx 通道 y 中断禁能
<code>dma_security_config</code>	DMAx 通道 y 的安全属性配置
<code>dma_priv_config</code>	DMAx 通道 y 的特权模式配置
<code>dma_illegal_access_flag_get</code>	获取 DMAx 通道 y 非法访问标志位
<code>dma_illegal_access_flag_clear</code>	清除 DMAx 通道 y 非法访问标志位

### 枚举类型 `dma_channel_enum`

表 3-130. 枚举类型 `dma_channel_enum`

成员名称	功能描述
<code>DMA_CH0</code>	DMA通道0
<code>DMA_CH1</code>	DMA通道1
<code>DMA_CH2</code>	DMA通道2
<code>DMA_CH3</code>	DMA通道3
<code>DMA_CH4</code>	DMA通道4
<code>DMA_CH5</code>	DMA通道5
<code>DMA_CH6</code>	DMA通道6
<code>DMA_CH7</code>	DMA通道7

### 枚举类型 `dma_subperipheral_enum`

表 3-131. 枚举类型 `dma_subperipheral_enum`

成员名称	功能描述
<code>DMA_SUBPERI0</code>	DMA外设0
<code>DMA_SUBPERI1</code>	DMA外设1
<code>DMA_SUBPERI2</code>	DMA外设2
<code>DMA_SUBPERI3</code>	DMA外设3
<code>DMA_SUBPERI4</code>	DMA外设4
<code>DMA_SUBPERI5</code>	DMA外设5
<code>DMA_SUBPERI6</code>	DMA外设6
<code>DMA_SUBPERI7</code>	DMA外设7

### 结构体 `dma_multi_data_parameter_struct`

表 3-132. 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

periph_width	外设数据传输宽度
periph_inc	外设地址生成算法模式
memory0_addr	存储器 0 基地址
memory_width	存储器数据传输宽度
memory_inc	存储器地址生成算法模式
memory_burst_width	存储器突发传输数据宽度
periph_burst_width	外设突发传输数据宽度
critical_value	FIFO 深度
circular_mode	DMA 循环模式
direction	传输方向
number	传输的数据量
priority	通道优先级

### 结构体 dma\_single\_data\_parameter\_struct

表 3-133. 结构体 dma\_single\_data\_parameter\_struct

成员名称	功能描述
periph_addr	外设基地址
periph_inc	外设地址生成算法模式
memory0_addr	存储器 0 基地址
memory_inc	存储器地址生成算法模式
periph_width	外设数据传输宽度
periph_memory_width	外设/存储器传输数据宽度
circular_mode	DMA 循环模式
direction	DMA 通道数据传输方向
number	DMA 通道数据传输数量
priority	DMA 通道传输软件优先级

### 函数 dma\_deinit

函数dma\_deinit描述见下表：

表 3-134. 函数 dma\_deinit

函数名称	dma_deinit
函数原型	void dma_deinit(uint32_t dma_periph, dma_channel_enum channelx);
功能描述	复位外设 DMAx 的通道 y 的所有寄存器
先决条件	无
被调用函数	无
输入参数{in}	
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, 参考 <a href="#">表 3-130. 枚举类型 dma_channel_enum</a>
输出参数{out}	
-	-
返回值	
-	-

例如:

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

### 函数 dma\_single\_data\_para\_struct\_init

函数dma\_single\_data\_para\_struct\_init描述见下表:

表 3-135. 函数 dma\_single\_data\_para\_struct\_init

函数名称	dma_single_data_para_struct_init
函数原型	void dma_single_data_para_struct_init(dma_single_data_parameter_struct* init_struct);
功能描述	将单数据传输模式结构体中所有参数初始化为默认值
先决条件	无
被调用函数	无
输入参数{in}	
init_struct	已定义的结构体变量地址, 结构体成员参考 <a href="#">表 3-133. 结构体 dma_single_data_parameter_struct</a>
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* initialize the parameters struct of DMA with single data mode*/
dma_single_data_parameter_struct dma_init_struct;
dma_single_data_para_struct_init(&dma_init_struct);
```

### 函数 dma\_multi\_data\_para\_struct\_init

函数dma\_multi\_data\_para\_struct\_init描述见下表:

表 3-136. 函数 dma\_multi\_data\_para\_struct\_init

函数名称	dma_multi_data_para_struct_init
函数原型	void dma_multi_data_para_struct_init(dma_multi_data_parameter_struct* init_struct);
功能描述	将突发传输模式结构体中所有参数初始化为默认值
先决条件	无



被调用函数	无
输入参数{in}	
init_struct	已定义的结构体变量地址，结构体成员参考 <a href="#">表 3-132. 结构体 dma_multi_data_parameter_struct</a>
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* initialize the parameters struct of DMA with multi data mode*/
dma_multi_data_parameter_struct dma_init_struct;
dma_multi_data_para_struct_init(&dma_init_struct);
```

### 函数 dma\_single\_data\_mode\_init

函数dma\_single\_data\_mode\_init描述见下表：

表 3-137. 函数 dma\_single\_data\_mode\_init

函数名称	dma_single_data_mode_init
函数原型	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	DMAx 的通道 y 单数据传输模式初始化
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，参考 <a href="#">表 3-130. 枚举类型 dma_channel_enum</a>
输入参数{in}	
init_struct	已定义的结构体变量地址，结构体成员参考 <a href="#">表 3-133. 结构体 dma_single_data_parameter_struct</a>
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* DMA1 channel0 single data mode initialize */
dma_single_data_parameter_struct dma_init_struct;
dma_deinit(DMA1, DMA_CH0);
dma_single_data_para_struct_init(&dma_init_struct);
```

```

dma_init_struct.direction = DMA_MEMORY_TO_MEMORY;
dma_init_struct.memory0_addr = (uint32_t)g_destbuf;
dma_init_struct.memory_inc = DMA_MEMORY_INCREASE_ENABLE;
dma_init_struct.periph_memory_width = DMA_MEMORY_WIDTH_32BIT;
dma_init_struct.number = TRANSFER_NUM;
dma_init_struct.circular_mode = DMA_CIRCULAR_MODE_DISABLE;
dma_init_struct.periph_addr = (uint32_t)FLASH_WRITE_START_ADDR;
dma_init_struct.periph_inc = DMA_PERIPH_INCREASE_ENABLE;
dma_init_struct.priority = DMA_PRIORITY_ULTRA_HIGH;
dma_single_data_mode_init(DMA1, DMA_CH0, &dma_init_struct);

```

### 函数 dma\_multi\_data\_mode\_init

函数dma\_multi\_data\_mode\_init描述见下表:

表 3-138. 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	DMAx 的通道 y 突发传输模式初始化
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, 参考 <a href="#">表 3-130. 枚举类型 dma_channel_enum</a>
输入参数{in}	
init_struct	已定义的结构体变量地址, 结构体成员参考 <a href="#">表 3-132. 结构体 dma_multi_data_parameter_struct</a>
输出参数{out}	
-	-
返回值	
-	-

例如:

```

/* DMA0 channel0 multi data mode initialize */
dma_multi_data_parameter_struct dma_init_parameter;
dma_multi_data_para_struct_init(&dma_init_parameter);
dma_deinit(DMA1, DMA_CH0);

```

```

dma_init_parameter.periph_addr = (uint32_t)source;

```

```

dma_init_parameter.periph_width = DMA_PERIPH_WIDTH_16BIT;
dma_init_parameter.periph_inc = DMA_PERIPH_INCREASE_ENABLE;
dma_init_parameter.memory0_addr = (uint32_t)destination;
dma_init_parameter.memory_width = DMA_MEMORY_WIDTH_16BIT;
dma_init_parameter.memory_inc = DMA_MEMORY_INCREASE_ENABLE;
dma_init_parameter.memory_burst_width = DMA_MEMORY_BURST_4_BEAT;
dma_init_parameter.periph_burst_width = DMA_PERIPH_BURST_4_BEAT;
dma_init_parameter.critical_value = DMA_FIFO_2_WORD;
dma_init_parameter.circular_mode = DMA_CIRCULAR_MODE_DISABLE;
dma_init_parameter.direction = DMA_MEMORY_TO_MEMORY;
dma_init_parameter.number = BUFFER_SIZE;
dma_init_parameter.priority = DMA_PRIORITY_ULTRA_HIGH;
dma_multi_data_mode_init(DMA1,DMA_CH0, &dma_init_parameter);

```

### 函数 dma\_periph\_address\_config

函数dma\_periph\_address\_config描述见下表：

表 3-139. 函数 dma\_periph\_address\_config

函数名称	dma_periph_address_config
函数原型	void dma_periph_address_config(uint32_t dma_periph, dma_channel_enum channelx, uint32_t address);
功能描述	DMAx 通道 y 传输的外设基地址配置
先决条件	无
被调用函数	无
输入参数{in}	
dma_periph	DMA 外设
DMAx(x=0,1)	DMA 外设选择
输入参数{in}	
channelx	DMA 通道
DMA_CHx(x=0..7)	DMA 通道选择, 参考 <a href="#">表 3-130. 枚举类型 dma_channel_enum</a>
输入参数{in}	
address	外设基地址
输出参数{out}	
-	-
返回值	
-	-

例如：

```

/* configure DMA0 channel0 peripheral address */

#define USART_DATA_ADDR (USART0 + 0x00000024U)

dma_periph_address_config(DMA0, DMA_CH0, USART_DATA_ADDR);

```

## 函数 dma\_memory\_address\_config

函数dma\_memory\_address\_config描述见下表：

**表 3-140. 函数 dma\_memory\_address\_config**

函数名称	dma_memory_address_config
函数原型	void dma_memory_address_config(uint32_t dma_periph, dma_channel_enum channelx, uint32_t address);
功能描述	DMAx 通道 y 传输的存储器基地址配置
先决条件	无
被调用函数	无
输入参数{in}	
dma_periph	DMA 外设
DMAx(x=0,1)	DMA 外设选择
输入参数{in}	
channelx	DMA 通道
DMA_CHx(x=0..7)	DMA 通道选择，参考 <a href="#">表 3-130. 枚举类型 dma_channel_enum</a>
输入参数{in}	
address	存储器基地址
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* configure DMA0 channel0 memory0 address */
uint32_t array[10] = {0};
dma_memory_address_config(DMA0, DMA_CH0, array);
```

## 函数 dma\_transfer\_number\_config

函数dma\_transfer\_number\_config描述见下表：

**表 3-141. 函数 dma\_transfer\_number\_config**

函数名称	dma_transfer_number_config
函数原型	void dma_transfer_number_config(uint32_t dma_periph, dma_channel_enum channelx, uint32_t number);
功能描述	配置 DMAx 通道 y 需要传输的数据量
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}	
<b>channelx</b>	DMA 通道
<i>DMA_CHx(x=0..7)</i>	DMA 通道选择, 参考 <a href="#">表 3-130. 枚举类型 dma_channel_enum</a>
输入参数{in}	
<b>number</b>	数据传输数量 (0x00000000 – 0x0000FFFF)
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* configure DMA0 channel0 transfer number */
#define TRANSFER_NUM                0x400
dma_transfer_number_config(DMA0, DMA_CH0, TRANSFER_NUM);
```

### 函数 dma\_transfer\_number\_get

函数dma\_transfer\_number\_get描述见下表:

表 3-142. 函数 dma\_transfer\_number\_get

函数名称	dma_transfer_number_get
函数原型	uint32_t dma_transfer_number_get(uint32_t dma_periph, dma_channel_enum channelx);
功能描述	获取 DMAx 通道 y 剩余需要传输的数据量
先决条件	无
被调用函数	无
输入参数{in}	
<b>dma_periph</b>	DMA 外设
<i>DMAx(x=0,1)</i>	DMA 外设选择
输入参数{in}	
<b>channelx</b>	DMA 通道
<i>DMA_CHx(x=0..7)</i>	DMA 通道选择, 参考 <a href="#">表 3-130. 枚举类型 dma_channel_enum</a>
输出参数{out}	
-	-
返回值	
<b>uint32_t</b>	DMA 数据传输剩余数量 (0x00000000 – 0x0000FFFF)

例如:

```
/* get channel0 memory0 address */
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 3-143. 函数 dma\_priority\_config

函数名称	dma_priority_config
函数原型	void dma_priority_config(uint32_t dma_periph, dma_channel_enum channelx, uint32_t priority);
功能描述	DMAx 通道 y 的传输软件优先级配置
先决条件	无
被调用函数	无
输入参数{in}	
dma_periph	DMA 外设
DMAx(x=0,1)	DMA 外设选择
输入参数{in}	
channelx	DMA 通道
DMA_CHx(x=0..7)	DMA 通道选择, 参考 <a href="#">表 3-130. 枚举类型 dma_channel_enum</a>
输入参数{in}	
priority	DMA 通道软件优先级
DMA_PRIORITY_LOW	低优先级
DMA_PRIORITY_MEDIUM	中优先级
DMA_PRIORITY_HIGH	高优先级
DMA_PRIORITY_ULTRA_HIGH	极高优先级
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* configure DMA0 channel0 priority */
```

```
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 3-144. 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	DMAx 通道 y 突发传输存储器节拍数配置
先决条件	无
被调用函数	无
输入参数{in}	
dma_periph	DMA 外设
DMAx(x=0,1)	DMA 外设选择
输入参数{in}	
channelx	DMA 通道
DMA_CHx(x=0..7)	DMA 通道选择, 参考 <a href="#">表 3-130. 枚举类型 dma_channel_enum</a>
输入参数{in}	
mbeat	突发传输节拍数
DMA_MEMORY_BURST_SINGLE	存储器单拍传输
DMA_MEMORY_BURST_4_BEAT	存储器 4 拍传输
DMA_MEMORY_BURST_8_BEAT	存储器 8 拍传输
DMA_MEMORY_BURST_16_BEAT	存储器 16 拍传输
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* configure DMA0 channel0 transfer burst beats of memory */
```

```
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 3-145. 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	DMAx 通道 y 突发传输外设节拍数配置
先决条件	无
被调用函数	无
输入参数{in}	
dma_periph	DMA 外设
DMAx(x=0,1)	DMA 外设选择
输入参数{in}	

<b>channelx</b>	DMA 通道
<i>DMA_CHx(x=0..7)</i>	DMA 通道选择, 参考 <a href="#">表 3-130. 枚举类型 dma_channel_enum</a>
<b>输入参数{in}</b>	
<b>pbeat</b>	突发传输节拍数
<i>DMA_PERIPH_BURST_SINGLE</i>	外设单拍传输
<i>DMA_PERIPH_BURST_4_BEAT</i>	外设 4 拍传输
<i>DMA_PERIPH_BURST_8_BEAT</i>	外设 8 拍传输
<i>DMA_PERIPH_BURST_16_BEAT</i>	外设 16 拍传输
<b>输出参数{out}</b>	
-	-
<b>返回值</b>	
-	-

例如:

```
/* configure DMA0 channel0 transfer burst beats of peripheral */
```

```
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 3-146. 函数 dma\_memory\_width\_config

<b>函数名称</b>	dma_memory_width_config
<b>函数原型</b>	void dma_memory_width_config (uint32_t dma_periph, dma_channel_enum channelx, uint32_t mwidth);
<b>功能描述</b>	DMAx 通道 y 传输的存储器数据宽度配置
<b>先决条件</b>	无
<b>被调用函数</b>	无
<b>输入参数{in}</b>	
<b>dma_periph</b>	DMA 外设
<i>DMAx(x=0,1)</i>	DMA 外设选择
<b>输入参数{in}</b>	
<b>channelx</b>	DMA 通道
<i>DMA_CHx(x=0..7)</i>	DMA 通道选择, 参考 <a href="#">表 3-130. 枚举类型 dma_channel_enum</a>
<b>输入参数{in}</b>	
<b>mwidth</b>	存储器数据传输宽度
<i>DMA_MEMORY_WIDTH_8BIT</i>	8 位数据传输宽度
<i>DMA_MEMORY_WIDTH_16BIT</i>	16 位数据传输宽度



<i>DTH_16BIT</i>	
<i>DMA_MEMORY_WIDTH_8BIT</i> <i>DTH_32BIT</i>	32 位数据传输宽度
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* configure DMA0 channel0 transfer memory width */
```

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

### 函数 dma\_periph\_width\_config

函数dma\_periph\_width\_config描述见下表：

表 3-147. 函数 dma\_periph\_width\_config

函数名称	dma_periph_width_config
函数原型	void dma_periph_width_config (uint32_t dma_periph, dma_channel_enum channelx, uint32_t pwidth);
功能描述	DMAx 通道 y 传输的外设数据宽度配置
先决条件	无
被调用函数	无
输入参数{in}	
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，参考 <a href="#">表 3-130. 枚举类型 dma_channel_enum</a>
输入参数{in}	
pwidth	外设数据传输宽度
DMA_PERIPHERAL_WIDTH_8BIT	8 位数据传输宽度
DMA_PERIPHERAL_WIDTH_16BIT	16 位数据传输宽度
DMA_PERIPHERAL_WIDTH_32BIT	32 位数据传输宽度
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* configure DMA0 channel0 transfer peripheral width */
```

```
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 3-148. 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	DMAx 通道 y 存储器地址生成配置
先决条件	无
被调用函数	无
输入参数{in}	
dma_periph	DMA 外设
DMAx(x=0,1)	DMA 外设选择
输入参数{in}	
channelx	DMA 通道
DMA_CHx(x=0..7)	DMA 通道选择, 参考 <a href="#">表 3-130. 枚举类型 dma_channel_enum</a>
输入参数{in}	
generation_algorithm	地址生成模式
DMA_MEMORY_INCREASE_ENABLE	地址增长模式
DMA_MEMORY_INCREASE_DISABLE	固定地址模式
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* configure DMA0 channel0 memory address generation algorithm as memory increase*/
```

```
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 3-149. 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	DMAx 通道 y 外设地址生成配置
先决条件	无
被调用函数	无
输入参数{in}	
<b>dma_periph</b>	DMA 外设
<i>DMAx(x=0,1)</i>	DMA 外设选择
输入参数{in}	
<b>channelx</b>	DMA 通道
<i>DMA_CHx(x=0..7)</i>	DMA 通道选择, 参考 <a href="#">表 3-130. 枚举类型 dma_channel_enum</a>
输入参数{in}	
<b>generation_algorithm</b>	地址生成模式
<i>DMA_PERIPH_INCREMENT_ENABLE</i>	地址增长模式
<i>DMA_PERIPH_INCREMENT_DISABLE</i>	固定地址模式
<i>DMA_PERIPH_INCREMENT_FIX</i>	外设地址固定增量 (4)
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* configure DMA0 channel0 memory address generation algorithm as memory increase */
```

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

## 函数 dma\_circulation\_enable

函数dma\_circulation\_enable描述见下表:

表 3-150. 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	DMAx 的通道 y 循环模式使能
先决条件	无
被调用函数	无
输入参数{in}	
<b>dma_periph</b>	DMA 外设
<i>DMAx(x=0,1)</i>	DMA 外设选择
输入参数{in}	
<b>channelx</b>	DMA 通道

DMA_CHx(x=0..7)	DMA 通道选择, 参考 <a href="#">表 3-130. 枚举类型 dma_channel_enum</a>
输出参数{out}	
-	-
返回值	
-	-

例如:

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

### 函数 dma\_circulation\_disable

函数dma\_circulation\_disable描述见下表:

表 3-151. 函数 dma\_circulation\_disable

函数名称	dma_circulation_disable
函数原型	void dma_circulation_disable(uint32_t dma_periph, dma_channel_enum channelx);
功能描述	DMAx 的通道 y 循环模式禁能
先决条件	无
被调用函数	无
输入参数{in}	
dma_periph	DMA 外设
DMAx(x=0,1)	DMA 外设选择
输入参数{in}	
channelx	DMA 通道
DMA_CHx(x=0..7)	DMA 通道选择, 参考 <a href="#">表 3-130. 枚举类型 dma_channel_enum</a>
输出参数{out}	
-	-
返回值	
-	-

例如:

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

### 函数 dma\_channel\_enable

函数dma\_channel\_enable描述见下表:

表 3-152. 函数 dma\_channel\_enable

函数名称	dma_channel_enable
函数原型	void dma_channel_enable(uint32_t dma_periph, dma_channel_enum channelx);
功能描述	外设 DMAx 的通道 y 传输使能

先决条件	无
被调用函数	无
输入参数{in}	
<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, 参考 <a href="#">表 3-130. 枚举类型 dma_channel_enum</a>
输出参数{out}	
-	-
返回值	
-	-

例如:

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

### 函数 dma\_channel\_disable

函数dma\_channel\_disable描述见下表:

表 3-153. 函数 dma\_channel\_disable

函数名称	dma_channel_disable
函数原型	void dma_channel_disable(uint32_t dma_periph, dma_channel_enum channelx);
功能描述	外设 DMAx 的通道 y 传输禁能
先决条件	无
被调用函数	无
输入参数{in}	
<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, 参考 <a href="#">表 3-130. 枚举类型 dma_channel_enum</a>
输出参数{out}	
-	-
返回值	
-	-

例如:

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

## 函数 dma\_transfer\_direction\_config

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

表 3-154. 函数 dma\_transfer\_direction\_config

函数名称	dma_transfer_direction_config
函数原型	void dma_transfer_direction_config(uint32_t dma_periph, dma_channel_enum channelx, uint32_t direction);
功能描述	DMAx 通道 y 的传输方向配置
先决条件	无
被调用函数	无
输入参数{in}	
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，参考 <a href="#">表 3-130. 枚举类型 dma_channel_enum</a>
输入参数{in}	
direction	数据传输方向
DMA_PERIPHERAL_TO_MEMORY	读取外设中数据，写入存储器
DMA_MEMORY_TO_PERIPHERAL	读取存储器中数据，写入外设
DMA_MEMORY_TO_MEMORY	读取存储器中数据，写入存储器
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* configure DMA0 channel0 transfer direction as peripheral to memory mode*/
dma_transfer_direction_config(DMA0, DMA_CH0, DMA_PERIPHERAL_TO_MEMORY);
```

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

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

表 3-155. 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	DMAx 通道 y 的存储切换模式配置

先决条件	无
被调用函数	无
输入参数{in}	
<b>dma_periph</b>	DMA 外设
<i>DMAx(x=0,1)</i>	DMA 外设选择
输入参数{in}	
<b>channelx</b>	DMA 通道
<i>DMA_CHx(x=0..7)</i>	DMA 通道选择, 参考 <a href="#">表 3-130. 枚举类型 dma_channel_enum</a>
输入参数{in}	
<b>memory1_addr</b>	存储器 1 基地址
输入参数{in}	
<b>memory_select</b>	选择存储器缓冲区
<i>DMA_MEMORY_0</i>	选择存储器 0 作为传输区域
<i>DMA_MEMORY_1</i>	选择存储器 1 作为传输区域
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* configure DMA0 channel0 switch buffer mode*/
```

```
uint32_t mem[32] = {0};
```

```
dma_switch_buffer_mode_config(DMA0, DMA_CH0, mem, DMA_MEMORY_0);
```

### 函数 dma\_using\_memory\_get

函数dma\_using\_memory\_get描述见下表:

表 3-156. 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	存储器到存储器 DMA 传输使能
先决条件	无
被调用函数	无
输入参数{in}	
<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, 参考 <a href="#">表 3-130. 枚举类型 dma_channel_enum</a>
输出参数{out}	

-	-
返回值	
uint32_t	当前传输使用的存储器
DMA_MEMORY_0	存储器 0
DMA_MEMORY_1	存储器 1

例如：

```
/* get DMA0 channel0 transfer buffer currently used */
uint32_t buf_addr = DMA_MEMORY_0;
buf_addr = 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 3-157. 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	DMAx 通道 y 的外设请求选择
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，参考 <a href="#">表 3-130. 枚举类型 dma_channel_enum</a>
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* configure DMA0 channel0 peripheral */
dma_channel_subperipheral_select(DMA0, DMA_CH0, DMA_SUBPERI1);
```

### 函数 dma\_flow\_controller\_config

函数dma\_flow\_controller\_config描述见下表：

**表 3-158. 函数 dma\_flow\_controller\_config**

函数名称	dma_flow_controller_config
函数原型	void dma_flow_controller_config(uint32_t dma_periph, dma_channel_enum



	channelx, uint32_t controller);
功能描述	DMAx 通道 y 的传输控制器选择
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, 参考 <a href="#">表 3-130. 枚举类型 dma_channel_enum</a>
输入参数{in}	
controller	控制器选择
DMA_FLOW_CONTROLLER_DMA	DMA 作为控制器
DMA_FLOW_CONTROLLER_PERI	外设作为控制器
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* configure DMA0 channel0 flow controller */
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 3-159. 函数 dma\_switch\_buffer\_mode\_enable

函数名称	dma_switch_buffer_mode_enable
函数原型	void dma_switch_buffer_mode_enable(uint32_t dma_periph, dma_channel_enum channelx);
功能描述	DMAx 通道 y 存储切换模式使能
先决条件	无
被调用函数	无
输入参数{in}	
dma_periph	DMA 外设
DMAx(x=0,1)	DMA 外设选择
输入参数{in}	
channelx	DMA 通道
DMA_CHx(x=0..7)	DMA 通道选择, 参考 <a href="#">表 3-130. 枚举类型 dma_channel_enum</a>
输出参数{out}	

-	-
返回值	
-	-

例如：

```
/* enable DMA0 channel0 switch buffer mode */
dma_switch_buffer_mode_enable(DMA0, DMA_CH0);
```

### 函数 dma\_switch\_buffer\_mode\_disable

函数dma\_switch\_buffer\_mode\_disable描述见下表：

表 3-160. 函数 dma\_switch\_buffer\_mode\_disable

函数名称	dma_switch_buffer_mode_disable
函数原型	void dma_switch_buffer_mode_disable(uint32_t dma_periph, dma_channel_enum channelx);
功能描述	DMAx 通道 y 存储切换模式禁能
先决条件	无
被调用函数	无
输入参数{in}	
dma_periph	DMA 外设
DMAx(x=0,1)	DMA 外设选择
输入参数{in}	
channelx	DMA 通道
DMA_CHx(x=0..7)	DMA 通道选择，参考 <a href="#">表 3-130. 枚举类型 dma_channel_enum</a>
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* disable DMA0 channel0 switch buffer mode */
dma_switch_buffer_mode_disable(DMA0, DMA_CH0);
```

### 函数 dma\_fifo\_status\_get

函数dma\_fifo\_status\_get描述见下表：

表 3-161. 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	获取 DMAx 通道 y 的 FIFO 状态
先决条件	无
被调用函数	无

输入参数{in}	
<b>dma_periph</b>	DMA 外设
<i>DMAx(x=0,1)</i>	DMA 外设选择
输入参数{in}	
<b>channelx</b>	DMA 通道
<i>DMA_CHx(x=0..7)</i>	DMA 通道选择, 参考 <a href="#">表 3-130. 枚举类型 dma_channel_enum</a>
输出参数{out}	
-	-
返回值	
<b>uint32_t</b>	当前 FIFO 使用情况

例如:

```
/* get DMA0 channel0 FIFO status */

uint32_t fifo_state = 0;
fifo_state = dma_fifo_status_get(DMA0, DMA_CH0);
```

### 函数 dma\_flag\_get

函数dma\_flag\_get描述见下表:

表 3-162. 函数 dma\_flag\_get

函数名称	dma_flag_get
函数原型	FlagStatus dma_flag_get(uint32_t dma_periph, dma_channel_enum channelx, uint32_t flag);
功能描述	获取 DMAx 通道 y 的标志位
先决条件	无
被调用函数	无
输入参数{in}	
<b>dma_periph</b>	DMA 外设
<i>DMAx(x=0,1)</i>	DMA 外设选择
输入参数{in}	
<b>channelx</b>	DMA 通道
<i>DMA_CHx(x=0..7)</i>	DMA 通道选择, 参考 <a href="#">表 3-130. 枚举类型 dma_channel_enum</a>
输入参数{in}	
<b>flag</b>	DMA 标志
<i>DMA_FLAG_FEE</i>	FIFO 异常标志位
<i>DMA_FLAG_SDE</i>	单数据传输异常标志位
<i>DMA_FLAG_TAE</i>	传输错误标志位
<i>DMA_FLAG_HTF</i>	半传输完成标志位
<i>DMA_FLAG_FTF</i>	传输完成标志位
输出参数{out}	
-	-
返回值	

<b>FlagStatus</b>	SET 或 RESET
-------------------	-------------

例如:

```
/* check DMA0 channel0 full transfer finish flag set or not */
```

```
FlagStatus state = RESET;
```

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

### 函数 dma\_flag\_clear

函数dma\_flag\_clear描述见下表:

**表 3-163. 函数 dma\_flag\_clear**

<b>函数名称</b>	dma_flag_clear
<b>函数原型</b>	void dma_flag_clear(uint32_t dma_periph, dma_channel_enum channelx, uint32_t flag);
<b>功能描述</b>	清除 DMAx 通道 y 标志位状态
<b>先决条件</b>	无
<b>被调用函数</b>	无
<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, 参考 <a href="#">表 3-130. 枚举类型 dma_channel_enum</a>
<b>输入参数{in}</b>	
<b>flag</b>	DMA 标志
<i>DMA_FLAG_FEE</i>	清除 FIFO 异常标志位
<i>DMA_FLAG_SDE</i>	清除单数据传输异常标志位
<i>DMA_FLAG_TAE</i>	清除传输错误标志位
<i>DMA_FLAG_HTF</i>	清除半传输完成标志位
<i>DMA_FLAG_FTF</i>	清除传输完成标志位
<b>输出参数{out}</b>	
-	-
<b>返回值</b>	
-	-

例如:

```
/* clear DMA0 channel0 full transfer finish flag */
```

```
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 3-164. 函数 dma\_interrupt\_flag\_get

函数名称	dma_interrupt_flag_get
函数原型	FlagStatus dma_interrupt_flag_get(uint32_t dma_periph, dma_channel_enum channelx, uint32_t int_flag);
功能描述	获取 DMAx 通道 y 的中断标志位
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, 参考 <a href="#">表 3-130. 枚举类型 dma_channel_enum</a>
输入参数{in}	
int_flag	DMA 标志
DMA_INT_FLAG_FEE	FIFO 异常中断标志位
DMA_INT_FLAG_SDE	单数据传输异常中断标志位
DMA_INT_FLAG_TAE	传输错误中断标志位
DMA_INT_FLAG_HTF	半传输完成中断标志位
DMA_INT_FLAG_FTF	传输完成中断标志位
输出参数{out}	
-	-
返回值	
FlagStatus	SET 或 RESET

例如:

```
/* check DMA0 channel0 full transfer finish interrupt flag set or not */
```

```
FlagStatus state = RESET;
```

```
state = dma_interrupt_flag_get(DMA0, DMA_CH0, DMA_INT_FLAG_FTF);
```

### 函数 dma\_interrupt\_flag\_clear

函数 dma\_interrupt\_flag\_clear 描述见下表:

表 3-165. 函数 dma\_interrupt\_flag\_clear

函数名称	dma_interrupt_flag_clear
函数原型	void dma_interrupt_flag_clear(uint32_t dma_periph, dma_channel_enum channelx, uint32_t int_flag);

功能描述	清除 DMAx 通道 y 中断标志位状态
先决条件	无
被调用函数	无
输入参数{in}	
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, 参考 <a href="#">表 3-130. 枚举类型 dma_channel_enum</a>
输入参数{in}	
int_flag	DMA 标志
DMA_INT_FLAG_FEE	FIFO 异常中断标志位
DMA_INT_FLAG_SDE	单数据传输异常中断标志位
DMA_INT_FLAG_TAE	传输错误中断标志位
DMA_INT_FLAG_HTF	半传输完成中断标志位
DMA_INT_FLAG_FTF	传输完成中断标志位
输出参数{out}	
-	-
返回值	
-	-

例如:

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

### 函数 dma\_interrupt\_enable

函数 dma\_interrupt\_enable 描述见下表:

表 3-166. 函数 dma\_interrupt\_enable

函数名称	dma_interrupt_enable
函数原型	void dma_interrupt_enable(uint32_t dma_periph, dma_channel_enum channelx, uint32_t source);
功能描述	DMAx 通道 y 中断使能
先决条件	无
被调用函数	无
输入参数{in}	

<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, 参考 <a href="#">表 3-130. 枚举类型 dma_channel_enum</a>
输入参数{in}	
<b>source</b>	DMA 中断源
<i>DMA_INT_FEE</i>	FIFO 异常中断
<i>DMA_INT_SDE</i>	单数据传输异常中断
<i>DMA_INT_TAE</i>	传输错误中断
<i>DMA_INT_HTF</i>	半传输完成中断
<i>DMA_INT_FTF</i>	传输完成中断
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* enable DMA0 channel0 full transfer finish interrupt */
dma_interrupt_enable(DMA0, DMA_CH0, DMA_INT_FTF);
```

### 函数 dma\_interrupt\_disable

函数dma\_interrupt\_disable描述见下表:

表 3-167. 函数 dma\_interrupt\_disable

函数名称	dma_interrupt_disable
函数原型	void dma_interrupt_disable(uint32_t dma_periph, dma_channel_enum channelx, uint32_t source);
功能描述	DMAx 通道 y 中断禁能
先决条件	无
被调用函数	无
输入参数{in}	
<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, 参考 <a href="#">表 3-130. 枚举类型 dma_channel_enum</a>
输入参数{in}	
<b>source</b>	DMA 中断源
<i>DMA_INT_FEE</i>	FIFO 异常中断
<i>DMA_INT_SDE</i>	单数据传输异常中断
<i>DMA_INT_TAE</i>	传输错误中断

<i>DMA_INT_HTF</i>	半传输完成中断
<i>DMA_INT_FTF</i>	传输完成中断
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* disable DMA0 channel0 full transfer finish interrupt */
```

```
dma_interrupt_disable(DMA0, DMA_CH0, DMA_INT_FTF);
```

### 函数 dma\_security\_config

函数dma\_security\_config描述见下表：

表 3-168. 函数 dma\_security\_config

函数名称	dma_security_config
函数原型	void dma_security_config(uint32_t dma_periph, dma_channel_enum channelx, uint32_t security);
功能描述	DMAx 通道 y 的安全属性配置
先决条件	无
被调用函数	无
输入参数{in}	
<b>dma_periph</b>	DMA 外设
<i>DMAx(x=0,1)</i>	DMA 外设选择
输入参数{in}	
<b>channelx</b>	DMA 通道
<i>DMA_CHx(x=0..7)</i>	DMA 通道选择，参考 <a href="#">表 3-130. 枚举类型 dma_channel_enum</a>
输入参数{in}	
<b>security</b>	通道安全属性
<i>DMA_CHANNEL_SECURE</i>	安全模式
<i>DMA_CHANNEL_NONSECURE</i>	非安全模式
<i>DMA_CHANNEL_SOURCE_SECURE</i>	通道传输源端安全
<i>DMA_CHANNEL_SOURCE_NONSECURE</i>	通道传输源端非安全
<i>DMA_CHANNEL_DESTINATION_SECURE</i>	通道传输目的端安全
<i>DMA_CHANNEL_DESTINATION_NONSECURE</i>	通道传输目的非端安全



<i>ESTINATION_NON SECURE</i>	
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* configure DMA0 channel0 security attribution */
dma_security_config(DMA0, DMA_CH0, DMA_CHANNEL_SECURE);
```

### 函数 dma\_priv\_config

函数dma\_priv\_config描述见下表：

表 3-169. 函数 dma\_security\_config

函数名称	dma_priv_config		
函数原型	void dma_priv_config(uint32_t dma_periph, dma_channel_enum channelx, uint32_t priv);		
功能描述	DMAx 通道 y 的特权模式配置		
先决条件	无		
被调用函数	无		
输入参数{in}			
dma_periph	DMA 外设		
DMAx(x=0,1)	DMA 外设选择		
输入参数{in}			
channelx	DMA 通道		
DMA_CHx(x=0..7)	DMA 通道选择，参考 <a href="#">表 3-130. 枚举类型 dma_channel_enum</a>		
输入参数{in}			
priv	通道特权模式配置		
DMA_CHANNEL_PRIV	特权模式		
DMA_CHANNEL_NONPRIV	非特权模式		
输出参数{out}			
-	-		
返回值			
-	-		

例如：

```
/* configure DMA0 channel0 privileged mode */
dma_priv_config(DMA0, DMA_CH0, DMA_CHANNEL_PRIV);
```

## 函数 dma\_illegal\_access\_flag\_get

函数dma\_illegal\_access\_flag\_get描述见下表:

表 3-170. 函数 dma\_illegal\_access\_flag\_get

函数名称	dma_illegal_access_flag_get
函数原型	FlagStatus dma_illegal_access_flag_get(uint32_t dma_periph, dma_channel_enum channelx);
功能描述	获取 DMAx 通道 y 非法访问标志位
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, 参考 <a href="#">表 3-130. 枚举类型 dma_channel_enum</a>
输出参数{out}	
-	-
返回值	
FlagStatus	SET 或 RESET

例如:

```
/* check DMA0 channel0 illegal access flag is set or not */
```

```
FlagStatus state = RESET;
```

```
state = dma_illegal_access_flag_get(DMA0, DMA_CH0);
```

## 函数 dma\_illegal\_access\_flag\_clear

函数dma\_illegal\_access\_flag\_clear描述见下表:

表 3-171. 函数 dma\_illegal\_access\_flag\_clear

函数名称	dma_illegal_access_flag_clear
函数原型	void dma_illegal_access_flag_clear(uint32_t dma_periph, dma_channel_enum channelx);
功能描述	清除 DMAx 通道 y 非法访问标志位
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, 参考 <a href="#">表 3-130. 枚举类型 dma_channel_enum</a>
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* clear DMA0 channel0 illegal access flag */
dma_illegal_access_flag_clear(DMA0, DMA_CH0);
```

## 3.8. EFUSE

EFUSE作为一种非易失性存储单元存储了一些必需的系统参数, 其中每一个比特位只允许从0改写为1。章节[3.8.1](#)描述了EFUSE的寄存器列表, 章节[3.8.2](#)对EFUSE库函数进行说明。

### 3.8.1. 外设寄存器说明

EFUSE寄存器列表如下表所示:

表 3-172. EFUSE 寄存器

寄存器名称	寄存器描述
EFUSE_CS	控制及状态寄存器
EFUSE_ADDR	地址寄存器
EFUSE_CTL	控制寄存器
EFUSE_TZCTL	Trustzone 控制寄存器
EFUSE_FP_CTL	安全保护控制寄存器
EFUSE_USER_CTL	用户控制寄存器
EFUSE_MCU_INIT_DATAx(x = 0...2)	MCU 初始化数据寄存器 x(x = 0...2)
EFUSE_AES_KEYx(x = 0...3)	固件 AES 密钥寄存器 x(x = 0...3)
EFUSE_ROTTPK_KEYx(x = 0...7)	RoTPK 密钥寄存器 x(x = 0...7)
EFUSE_DPx(x = 0,1)	调试密钥寄存器 x(x = 0,1)
EFUSE_IAK_GSSAx(x = 0...15)	IAK 密钥或 GSSA 寄存器 x(x = 0...15)
EFUSE_PUIDx(x = 0...3)	产品 UID 寄存器 x(x = 0...3)
EFUSE_HUK_KEYx(x = 0...3)	HUK 密钥寄存器 x(x = 0...3)
EFUSE_RF_DATAx	RF 参数寄存器 x(x = 0...11)

寄存器名称	寄存器描述
(x = 0...11)	
EFUSE_USER_DATAx(x = 0...7)	用户数据寄存器 x(x = 0...7)
EFUSE_PRE_TZEN	TrustZone 预使能寄存器
EFUSE_TZ_BOOT_ADDR	TrustZone 启动地址寄存器
EFUSE_NTZ_BOOT_ADDR	无 TrustZone 启动地址寄存器

### 3.8.2. 外设库函数说明

EFUSE库函数列表如下表所示：

表 3-173. EFUSE 库函数

库函数名称	库函数描述
efuse_read	读 EFUSE 值
efuse_write	写 EFUSE
efuse_boot_config	配置 Boot 引脚
efuse_tz_control_config	配置 EFUSE 中 TrustZone 控制字段
efuse_fp_config	配置 EFUSE 中安全保护控制字段
efuse_mcu_init_data_write	写 EFUSE 中 MCU 初始化数据字段
efuse_aes_key_write	写 EFUSE 中固件 AES 密钥字段
efuse_rotpk_key_write	写 EFUSE 中 RoTPK 密钥字段
efuse_dp_write	写 EFUSE 中调试密钥字段
efuse_iak_write	写 EFUSE 中 IAK 密钥字段
efuse_user_data_write	写 EFUSE 中用户数据字段
efuse_software_trustzone_enable	软件使能 TrustZone
efuse_software_trustzone_disable	软件失能 TrustZone
efuse_boot_address_get	获取启动地址
efuse_flag_get	获取 EFUSE 标志位
efuse_flag_clear	清除 EFUSE 标志位
efuse_interrupt_enable	使能 EFUSE 中断
efuse_interrupt_disable	失能 EFUSE 中断
efuse_interrupt_flag_get	获取 EFUSE 中断标志位
efuse_interrupt_flag_clear	清除 EFUSE 中断标志位

#### 枚举类型 efuse\_flag\_enum

表 3-174. 枚举类型 efuse\_flag\_enum

成员名称	功能描述
EFUSE_PGIF	写操作完成标志位
EFUSE_RDIF	读操作完成标志位

EFUSE_OBERIF	越界错误标志位
--------------	---------

### 枚举类型 `efuse_clear_flag_enum`

表 3-175. 枚举类型 `efuse_clear_flag_enum`

成员名称	功能描述
EFUSE_PGIC	清除写操作完成标志位
EFUSE_RDIC	清除读操作完成标志位
EFUSE_OBERIC	清除越界错误标志位

### 枚举类型 `efuse_int_enum`

表 3-176. 枚举类型 `efuse_int_enum`

成员名称	功能描述
EFUSE_INTEN_PG	使能写操作完成中断
EFUSE_INTEN_RD	使能读操作完成中断
EFUSE_INTEN_OBER	使能越界错误中断

### 枚举类型 `efuse_int_flag_enum`

表 3-177. 枚举类型 `efuse_int_flag_enum`

成员名称	功能描述
EFUSE_INT_PGIF	写操作完成中断标志
EFUSE_INT_RDIF	读操作完成中断标志
EFUSE_INT_OBERIF	越界错误中断标志

### 枚举类型 `efuse_clear_int_flag_enum`

表 3-178. 枚举类型 `efuse_clear_int_flag_enum`

成员名称	功能描述
EFUSE_INT_PGIC	清除写操作完成中断标志
EFUSE_INT_RDIC	清除读操作完成中断标志
EFUSE_INT_OBERIC	清除越界错误中断标志

### 枚举类型 `efuse_tz_enum`

表 3-179. 枚举类型 `efuse_tz_enum`

成员名称	功能描述
EFUSE_TZ	TrustZone已使能
EFUSE_N_TZ	TrustZone未使能

### 函数 `efuse_read`

函数`efuse_read`描述见下表：

表 3-180. 函数 efuse\_read

函数名称	efuse_read
函数原形	ErrStatus efuse_read(uint32_t ef_addr, uint32_t size, uint32_t buf[]);
功能描述	读 EFUSE 值
先决条件	-
被调用函数	-
输入参数{in}	
ef_addr	EFUSE 地址编号 (0x00 – 0xf8)
输入参数{in}	
size	要读出的 EFUSE 字段大小 (0x01 – 0x40 字节)
输出参数{out}	
buf	存储读回 EFUSE 字段数据的数组
返回值	
ErrStatus	ERROR 或 SUCCESS

例如:

```
/* read EFUSE USER DATA value */
```

```
uint32_t buffer[8] = {0};
```

```
ErrStatus flag = ERROR;
```

```
flag = efuse_read(0xd8, 32, buffer);
```

### 函数 efuse\_write

函数 efuse\_write 描述见下表:

表 3-181. 函数 efuse\_write

函数名称	efuse_write
函数原形	ErrStatus efuse_write(uint32_t ef_addr, uint32_t size, uint32_t buf[]);
功能描述	写 EFUSE
先决条件	-
被调用函数	-
输入参数{in}	
ef_addr	EFUSE 地址编号 (0x00 – 0xf8)，地址需是 4 的整数倍
输入参数{in}	
size	要写入的 EFUSE 字段大小 (0x01 – 0x40 字节)
输入参数{in}	
buf	存储写入 EFUSE 字段数据的数组
输出参数{out}	
-	-
返回值	
ErrStatus	ERROR 或 SUCCESS

例如:

```
/* write EFUSE USER DATA*/
```

```
uint32_t buffer[2] = {0x11223344, 0x55667788};
```

```
ErrStatus flag = ERROR;
```

```
flag = efuse_write(0xd8, 8, buffer);
```

### 函数 efuse\_boot\_config

函数efuse\_boot\_config描述见下表：

**表 3-182. 函数 efuse\_boot\_config**

函数名称	efuse_boot_config
函数原形	ErrStatus efuse_boot_config(uint32_t size, uint8_t bt_value[]);
功能描述	配置 Boot 引脚
先决条件	-
被调用函数	efuse_write
输入参数{in}	
size	传入的数组大小（字节），必须为 1
输入参数{in}	
bt_value	要写入的 EFUSE 控制字段值
输出参数{out}	
-	-
返回值	
ErrStatus	ERROR 或 SUCCESS

例如：

```
/* write Efuse control value */
```

```
uint8_t bt_value[1] = 0x32;
```

```
ErrStatus flag = ERROR;
```

```
flag = efuse_boot_config(1, bt_value);
```

### 函数 efuse\_tz\_control\_config

函数efuse\_tz\_control\_config描述见下表：

**表 3-183. 函数 efuse\_tz\_control\_config**

函数名称	efuse_tz_control_config
函数原形	ErrStatus efuse_tz_control_config(uint32_t size, uint8_t tz_ctl[]);
功能描述	配置 EFUSE 中 TrustZone 控制字段
先决条件	-
被调用函数	efuse_write
输入参数{in}	
size	传入的数组大小（字节），必须为 1

输入参数{in}	
<b>tz_ctl</b>	要写入的 Trustzone 控制字段值
输出参数{out}	
-	-
返回值	
<b>ErrStatus</b>	ERROR 或 SUCCESS

例如：

```
/* write Trustzone control value */
uint8_t tz_ctl[1] = 0x32;

ErrStatus flag = ERROR;

flag = efuse_tz_control_config(1, tz_ctl);
```

### 函数 efuse\_fp\_config

函数efuse\_fp\_config描述见下表：

表 3-184. 函数 efuse\_fp\_config

函数名称	efuse_fp_config
函数原形	ErrStatus efuse_fp_config(uint32_t size, uint8_t fp_value[]);
功能描述	配置EFUSE中安全保护控制字段
先决条件	-
被调用函数	efuse_write
输入参数{in}	
<b>size</b>	传入的数组大小（字节），必须为 1
输入参数{in}	
<b>fp_value</b>	要写入的存储保护字段值
输出参数{out}	
-	-
返回值	
<b>ErrStatus</b>	ERROR或SUCCESS

例如：

```
/* write Flash protection value */
uint8_t fp_value[1] = 0x02;

ErrStatus flag = ERROR;

flag = efuse_fp_config(1, fp_value);
```

### 函数 efuse\_mcu\_init\_data\_write

函数efuse\_mcu\_init\_data\_write描述见下表：



表 3-185. 函数 `efuse_mcu_init_data_write`

函数名称	<code>efuse_mcu_init_data_write</code>
函数原形	<code>ErrStatus efuse_mcu_init_data_write(uint32_t size, uint8_t buf[]);</code>
功能描述	写 EFUSE 中 MCU 初始化数据字段
先决条件	-
被调用函数	<code>efuse_write</code>
输入参数{in}	
<b>size</b>	传入的数组大小（字节），必须为 12
输入参数{in}	
<b>buf</b>	要写入的 MCU 初始化参数值
输出参数{out}	
-	-
返回值	
<b>ErrStatus</b>	ERROR 或 SUCCESS

例如：

```
/* write MCU initialization parameters value */
uint32_t buffer[3] = {0x11223344, 0x55667788, 0x9900aabb};
ErrStatus flag = ERROR;
flag = efuse_mcu_init_data_write(12, (uint8_t *)buffer);
```

### 函数 `efuse_aes_key_write`

函数 `efuse_aes_key_write` 描述见下表：

表 3-186. 函数 `efuse_aes_key_write`

函数名称	<code>efuse_aes_key_write</code>
函数原形	<code>ErrStatus efuse_aes_key_write(uint32_t size, uint8_t buf[]);</code>
功能描述	写 EFUSE 中固件 AES 密钥字段
先决条件	-
被调用函数	<code>efuse_write</code>
输入参数{in}	
<b>size</b>	传入的数组大小（字节），必须为 16
输入参数{in}	
<b>buf</b>	要写入的 AES 密钥值
输出参数{out}	
-	-
返回值	
<b>ErrStatus</b>	ERROR 或 SUCCESS

例如：

```
/* write AES key value */
```

```
uint32_t buffer[4] = {0x11223344, 0x55667788, 0x9900aabb, 0xccddeeff};
```

```
ErrStatus flag = ERROR;
```

```
flag = efuse_aes_key_write(16, (uint8_t *)buffer);
```

### 函数 efuse\_rotpk\_key\_write

函数efuse\_rotpk\_key\_write描述见下表:

表 3-187. 函数 efuse\_rotpk\_key\_write

函数名称	efuse_rotpk_key_write
函数原形	ErrStatus efuse_rotpk_key_write(uint32_t size, uint8_t buf[]);
功能描述	写 EFUSE 中 RoTPK 密钥字段
先决条件	-
被调用函数	efuse_write
输入参数{in}	
size	传入的数组大小（字节），必须为 32
输入参数{in}	
buf	要写入的 RoTPK 密钥值
输出参数{out}	
-	-
返回值	
ErrStatus	ERROR 或 SUCCESS

例如:

```
/* write AES key value */
```

```
uint32_t buffer[8] = {0x11223344, 0x55667788, 0x9900aabb, 0xccddeeff
```

```
0x11223344, 0x55667788, 0x9900aabb, 0xccddeeff};
```

```
ErrStatus flag = ERROR;
```

```
flag = efuse_rotpk_key_write(32, (uint8_t *)buffer);
```

### 函数 efuse\_dp\_write

函数efuse\_dp\_write描述见下表:

表 3-188. 函数 efuse\_dp\_write

函数名称	efuse_dp_write
函数原形	ErrStatus efuse_dp_write(uint32_t size, uint8_t buf[]);
功能描述	写 EFUSE 中调试密钥字段
先决条件	-
被调用函数	efuse_write
输入参数{in}	
size	传入的数组大小（字节），必须为 8

输入参数{in}	
<b>buf</b>	要写入的调试密钥值
输出参数{out}	
-	-
返回值	
<b>ErrStatus</b>	ERROR 或 SUCCESS

例如：

```
/* write Debug password value */
uint32_t buffer[2] = {0x11223344, 0x55667788};

ErrStatus flag = ERROR;

flag = efuse_dp_write(8, (uint8_t *)buffer);
```

### 函数 efuse\_iak\_write

函数efuse\_iak\_write描述见下表：

表 3-189. 函数 efuse\_iak\_write

函数名称	efuse_iak_write
函数原形	ErrStatus efuse_iak_write(uint32_t size, uint8_t buf[]);
功能描述	写 EFUSE 中 IAK 密钥字段
先决条件	-
被调用函数	efuse_write
输入参数{in}	
<b>size</b>	传入的数组大小（字），必须为 64
输入参数{in}	
<b>buf</b>	要写入的 IAK 密钥值
输出参数{out}	
-	-
返回值	
<b>ErrStatus</b>	ERROR 或 SUCCESS

例如：

```
/* write IAK value */
uint32_t buffer[16] = {0x11223344, 0x55667788, 0x9900aabb, 0xccddeeff,
                       0x11223344, 0x55667788, 0x9900aabb, 0xccddeeff,
                       0x11223344, 0x55667788, 0x9900aabb, 0xccddeeff,
                       0x11223344, 0x55667788, 0x9900aabb, 0xccddeeff };

ErrStatus flag = ERROR;
```

```
flag = efuse_iak_write(64, (uint8_t *)buffer);
```

### 函数 efuse\_user\_data\_write

函数efuse\_user\_data\_write描述见下表：

表 3-190. 函数 efuse\_user\_data\_write

函数名称	efuse_user_data_write
函数原形	ErrStatus efuse_user_data_write(uint32_t size, uint8_t buf[]);
功能描述	写 EFUSE 中用户数据字段
先决条件	-
被调用函数	efuse_write
输入参数{in}	
size	传入的数组大小（字），必须为 32
输入参数{in}	
buf	要写入的用户数据值
输出参数{out}	
-	-
返回值	
ErrStatus	ERROR 或 SUCCESS

例如：

```
/* write User data value */
```

```
uint32_t buffer[8] = {0x11223344, 0x55667788, 0x9900aabb, 0xccddeeff  
0x11223344, 0x55667788, 0x9900aabb, 0xccddeeff};
```

```
ErrStatus flag = ERROR;
```

```
flag = efuse_user_data_write(32, (uint8_t *)buffer);
```

### 函数 efuse\_software\_trustzone\_enable

函数efuse\_software\_trustzone\_enable描述见下表：

表 3-191. 函数 efuse\_software\_trustzone\_enable

函数名称	efuse_software_trustzone_enable
函数原形	void efuse_software_trustzone_enable(void);
功能描述	软件使能 TrustZone
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 Trustzone function by software */
efuse_software_trustzone_enable();
```

### 函数 efuse\_software\_trustzone\_disable

函数efuse\_software\_trustzone\_disable描述见下表：

表 3-192. 函数 efuse\_software\_trustzone\_disable

函数名称	efuse_software_trustzone_disable
函数原形	void efuse_software_trustzone_disable(void);
功能描述	软件失能 TrustZone
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 Trustzone function by software */
efuse_software_trustzone_disable();
```

### 函数 efuse\_boot\_address\_get

函数efuse\_boot\_address\_get描述见下表：

表 3-193. 函数 efuse\_boot\_address\_get

函数名称	efuse_boot_address_get
函数原形	uint32_t efuse_boot_address_get(efuse_tz_enum tz);
功能描述	获取启动地址
先决条件	-
被调用函数	-
输入参数{in}	
<b>tz</b>	当前 Trustzone 状态，参考 <a href="#">表 3-179. 枚举类型 efuse_tz_enum</a>
<i>EFUSE_TZ</i>	Trustzone 使能
<i>EFUSE_N_TZ</i>	Trustzone 未使能
输出参数{out}	
-	-
返回值	

<b>uint32_t</b>	当前启动地址 (0x0 – 0xFFFFFFFF)
-----------------	---------------------------

例如:

```
/* get boot address when Trustzone disable */

uint32_t addr = 0;

addr = effuse_boot_address_get(EFUSE_N_TZ);
```

### 函数 **efuse\_flag\_get**

函数 **efuse\_flag\_get** 描述见下表:

**表 3-194. 函数 **efuse\_flag\_get****

函数名称	efuse_flag_get
函数原形	FlagStatus efuse_flag_get(efuse_flag_enum efuse_flag);
功能描述	获取 EFUSE 标志位
先决条件	-
被调用函数	-
输入参数{in}	
<b>efuse_flag</b>	EFUSE 状态标志, 参考 <a href="#">表 3-174. 枚举类型 efuse_flag_enum</a>
<i>EFUSE_PGIF</i>	写操作完成标志位
<i>EFUSE_RDIF</i>	读操作完成标志位
<i>EFUSE_OBERIF</i>	越界错误标志位
输出参数{out}	
-	-
返回值	
<b>FlagStatus</b>	SET 或 RESET

例如:

```
/* get EFUSE write operation complete flag status */

FlagStatus state = efuse_flag_get(EFUSE_PGIF);
```

### 函数 **efuse\_flag\_clear**

函数 **efuse\_flag\_clear** 描述见下表:

**表 3-195. 函数 **efuse\_flag\_clear****

函数名称	efuse_flag_clear
函数原形	void efuse_flag_clear(efuse_clear_flag_enum efuse_cflag);
功能描述	清除 EFUSE 标志位
先决条件	-
被调用函数	-
输入参数{in}	
<b>efuse_cflag</b>	EFUSE 状态标志, 参考 <a href="#">表 3-175. 枚举类型 efuse_clear_flag_enum</a>

<i>EFUSE_PGIC</i>	清除写操作完成标志位
<i>EFUSE_RDIC</i>	清除读操作完成标志位
<i>EFUSE_OBERIC</i>	清除越界错误标志位
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* clear EFUSE write operation complete flag status */
```

```
efuse_flag_clear(EFUSE_PGIC);
```

### 函数 **efuse\_interrupt\_enable**

函数efuse\_interrupt\_enable描述见下表：

**表 3-196. 函数 efuse\_interrupt\_enable**

函数名称	efuse_interrupt_enable
函数原形	void efuse_interrupt_enable(efuse_int_enum source);
功能描述	使能 EFUSE 中断
先决条件	-
被调用函数	-
输入参数{in}	
<b>source</b>	要使能的中断源，参考 <a href="#">表 3-176. 枚举类型 efuse_int_enum</a>
<i>EFUSE_INTEN_PG</i>	使能写操作完成中断
<i>EFUSE_INTEN_RD</i>	使能读操作完成中断
<i>EFUSE_INTEN_OBER</i>	使能越界错误中断
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* enable EFUSE write operation complete interrupt */
```

```
efuse_interrupt_enable(EFUSE_INTEN_PG);
```

### 函数 **efuse\_interrupt\_disable**

函数efuse\_interrupt\_disable描述见下表：

**表 3-197. 函数 efuse\_interrupt\_disable**

函数名称	efuse_interrupt_disable
函数原形	void efuse_interrupt_disable(efuse_int_enum source);

功能描述	失能 EFUSE 中断
先决条件	-
被调用函数	-
输入参数{in}	
source	要失能的中断源，参考 <a href="#">表 3-176. 枚举类型 efuse_int_enum</a>
EFUSE_INTEN_PG	失能写操作完成中断
EFUSE_INTEN_RD	失能读操作完成中断
EFUSE_INTEN_OBER	失能越界错误中断
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* disable EFUSE write operation complete interrupt */
efuse_interrupt_disable(EFUSE_INTEN_PG);
```

### 函数 efuse\_interrupt\_flag\_get

函数 efuse\_interrupt\_flag\_get 描述见下表：

表 3-198. 函数 efuse\_interrupt\_flag\_get

函数名称	efuse_interrupt_flag_get
函数原形	FlagStatus efuse_interrupt_flag_get(efuse_int_flag_enum int_flag);
功能描述	获取 EFUSE 中断标志位
先决条件	-
被调用函数	-
输入参数{in}	
int_flag	中断标志位，参考 <a href="#">表 3-177. 枚举类型 efuse_int_flag_enum</a>
EFUSE_INT_PGIF	写操作完成中断标志
EFUSE_INT_RDIF	读操作完成中断标志
EFUSE_INT_OBERIF	越界错误中断标志
输出参数{out}	
-	-
返回值	
FlagStatus	SET 或 RESET

例如：

```
/* get EFUSE write operation complete interrupt flag status */
FlagStatus state = efuse_interrupt_flag_get(EFUSE_INT_PGIF);
```



**函数 efuse\_interrupt\_flag\_clear**

函数efuse\_interrupt\_flag\_clear描述见下表:

**表 3-199. 函数 efuse\_interrupt\_flag\_clear**

函数名称	efuse_interrupt_flag_clear
函数原形	void efuse_interrupt_flag_clear(efuse_clear_int_flag_enum int_cflag);
功能描述	清除 EFUSE 中断标志位
先决条件	-
被调用函数	-
<b>输入参数{in}</b>	
linex	中断标志清除位, 参考 <a href="#">表 3-178. 枚举类型 efuse_clear_int_flag_enum</a>
EFUSE_INT_PGIC	清除写操作完成中断标志
EFUSE_INT_RDIC	清除读操作完成中断标志
EFUSE_INT_OBERI C	清除越界错误中断标志
<b>输出参数{out}</b>	
-	-
<b>返回值</b>	
-	-

例如:

```
/* clear EFUSE write operation complete interrupt flag status */
efuse_interrupt_flag_clear(EFUSE_INT_PGIC);
```

**3.9. EXTI**

EXTI是MCU中的中断/事件控制器, 包括29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[3.9.1](#)描述了EXTI的寄存器列表, 章节[3.9.2](#)对EXTI库函数进行说明。

**3.9.1. 外设寄存器说明**

EXTI寄存器列表如下表所示:

**表 3-200. 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

寄存器名称	寄存器描述
EXTI_SECCFG	安全配置寄存器
EXTI_PRIVCFG	特权配置寄存器
EXTI_LOCK	锁定寄存器

### 3.9.2. 外设库函数说明

EXTI库函数列表如下表所示：

**表 3-201. EXTI 库函数**

库函数名称	库函数描述
exti_deinit	复位 EXTI，将 EXTI 的所有寄存器恢复成初始值
exti_init	初始化 EXTI 线 x，使能 EXTI 的初始化配置
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_security_enable	EXTI 线 x 安全使能
exti_security_disable	EXTI 线 x 安全禁能
exti_privilege_enable	EXTI 线 x 特权使能
exti_privilege_disable	EXTI 线 x 特权禁能
exti_lock_enable	使能 EXTI 安全属性和特权访问配置锁
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

#### 枚举类型 exti\_line\_enum

**表 3-202. 枚举类型 exti\_line\_enum**

成员名称	功能描述
EXTI_0	EXTI中断线0
EXTI_1	EXTI中断线1
EXTI_2	EXTI中断线2
EXTI_3	EXTI中断线3
EXTI_4	EXTI中断线4
EXTI_5	EXTI中断线5
EXTI_6	EXTI中断线6
EXTI_7	EXTI中断线7
EXTI_8	EXTI中断线8
EXTI_9	EXTI中断线9

EXTI_10	EXTI中断线10
EXTI_11	EXTI中断线11
EXTI_12	EXTI中断线12
EXTI_13	EXTI中断线13
EXTI_14	EXTI中断线14
EXTI_15	EXTI中断线15
EXTI_16	EXTI中断线16
EXTI_17	EXTI中断线17
EXTI_18	EXTI中断线18
EXTI_19	EXTI中断线19
EXTI_20	EXTI中断线20
EXTI_21	EXTI中断线21
EXTI_22	EXTI中断线22
EXTI_23	EXTI中断线23
EXTI_24	EXTI中断线24
EXTI_25	EXTI中断线25
EXTI_26	EXTI中断线26
EXTI_27	EXTI中断线27
EXTI_28	EXTI中断线28

#### 枚举类型 `exti_mode_enum`

表 3-203. 枚举类型 `exti_mode_enum`

成员名称	功能描述
EXTI_INTERRUPT	EXTI中断模式
EXTI_EVENT	EXTI事件模式

#### 枚举类型 `exti_trig_type_enum`

表 3-204. 枚举类型 `exti_trig_type_enum`

成员名称	功能描述
EXTI_TRIG_RISING	EXTI上升沿触发
EXTI_TRIG_FALLING	EXTI下降沿触发
EXTI_TRIG_BOTH	EXTI双边沿触发
EXTI_TRIG_NONE	EXTI双边沿均不触发

#### 函数 `exti_deinit`

函数`exti_deinit`描述见下表：

表 3-205. 函数 `exti_deinit`

函数名称	<code>exti_deinit</code>
函数原形	<code>void exti_deinit(void);</code>
功能描述	复位 EXTI，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 3-206. 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，使能 EXTI 的初始化配置
先决条件	-
被调用函数	-
输入参数{in}	
linex	EXTI 线 x，参考 <a href="#">表 3-202. 枚举类型 exti_line_enum</a>
EXTI_x	x=0,1,2..28
输入参数{in}	
mode	EXTI 模式，参考 <a href="#">表 3-203. 枚举类型 exti_mode_enum</a>
EXTI_INTERRUPT	中断模式
EXTI_EVENT	事件模式
输入参数{in}	
trig_type	触发类型，参考 <a href="#">表 3-204. 枚举类型 exti_trig_type_enum</a>
EXTI_TRIG_RISING	上升沿触发
EXTI_TRIG_FALLING	下降沿触发
EXTI_TRIG_BOTH	上升沿和下降沿均触发
EXTI_TRIG_NONE	上升沿和下降沿均不触发
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* configure EXTI_0 */
```

```
exti_init(EXTI_0, EXTI_INTERRUPT, EXTI_TRIG_BOTH);
```

### 函数 exti\_interrupt\_enable

函数exti\_interrupt\_enable描述见下表：

表 3-207. 函数 exti\_interrupt\_enable

函数名称	exti_interrupt_enable
函数原形	void exti_interrupt_enable(exti_line_enum linex);
功能描述	EXTI 线 x 中断使能
先决条件	-
被调用函数	-
输入参数{in}	
linex	EXTI 线 x, 参考 <a href="#">表 3-202. 枚举类型 exti_line_enum</a>
EXTI_x	x=0,1,2..28
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 3-208. 函数 exti\_interrupt\_disable

函数名称	exti_interrupt_disable
函数原形	void exti_interrupt_disable(exti_line_enum linex);
功能描述	EXTI 线 x 中断禁能
先决条件	-
被调用函数	-
输入参数{in}	
linex	EXTI 线 x, 参考 <a href="#">表 3-202. 枚举类型 exti_line_enum</a>
EXTI_x	x=0,1,2..28
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 3-209. 函数 exti\_event\_enable**

函数名称	exti_event_enable
函数原形	void exti_event_enable(exti_line_enum linex);
功能描述	EXTI线x事件使能
先决条件	-
被调用函数	-
输入参数{in}	
linex	EXTI线x, 参考 <a href="#">表3-202. 枚举类型exti_line_enum</a>
EXTI_x	x=0,1,2..28
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 3-210. 函数 exti\_event\_disable**

函数名称	exti_event_disable
函数原形	void exti_event_disable(exti_line_enum linex);
功能描述	EXTI 线 x 事件禁能
先决条件	-
被调用函数	-
输入参数{in}	
linex	EXTI 线 x, 参考 <a href="#">表 3-202. 枚举类型 exti_line_enum</a>
EXTI_x	x=0,1,2..28
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\_security\_enable

函数exti\_security\_enable描述见下表：

表 3-211. 函数 exti\_security\_enable

函数名称	exti_security_enable
函数原形	void exti_security_enable(exti_line_enum linex);
功能描述	EXTI 线 x 安全使能
先决条件	-
被调用函数	-
输入参数{in}	
linex	EXTI 线 x, 参考 <a href="#">表 3-202. 枚举类型 exti_line_enum</a>
EXTI_x	x=0,1,2..28
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* enable the security attribution */
```

```
exti_security_enable(EXTI_0);
```

### 函数 exti\_security\_disable

函数exti\_security\_disable描述见下表：

表 3-212. 函数 exti\_security\_disable

函数名称	exti_security_disable
函数原形	void exti_security_disable(exti_line_enum linex);
功能描述	EXTI 线 x 安全禁能
先决条件	-
被调用函数	-
输入参数{in}	
linex	EXTI 线 x, 参考 <a href="#">表 3-202. 枚举类型 exti_line_enum</a>
EXTI_x	x=0,1,2..28
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* disable the security attribution */
```

```
exti_security_disable(EXTI_0);
```

### 函数 exti\_privilege\_enable

函数exti\_privilege\_enable描述见下表:

表 3-213. 函数 exti\_privilege\_enable

函数名称	exti_privilege_enable
函数原形	void exti_privilege_enable(exti_line_enum linex);
功能描述	EXTI 线 x 特权使能
先决条件	-
被调用函数	-
输入参数{in}	
linex	EXTI 线 x, 参考 <a href="#">表 3-202. 枚举类型 exti_line_enum</a>
EXTI_x	x=0,1,2..28
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* enable the privileged access */
```

```
exti_privilege_enable(EXTI_0);
```

### 函数 exti\_privilege\_disable

函数exti\_privilege\_disable描述见下表:

表 3-214. 函数 exti\_privilege\_disable

函数名称	exti_privilege_disable
函数原形	void exti_privilege_disable(exti_line_enum linex);
功能描述	EXTI 线 x 特权禁能
先决条件	-
被调用函数	-
输入参数{in}	
linex	EXTI 线 x, 参考 <a href="#">表 3-202. 枚举类型 exti_line_enum</a>
EXTI_x	x=0,1,2..28
输出参数{out}	
-	-
返回值	
-	-

例如:



```
/* disable the privileged access */
```

```
exti_privilege_disable(EXTI_0);
```

### 函数 exti\_lock\_enable

函数exti\_lock\_enable描述见下表：

表 3-215. 函数 exti\_lock\_enable

函数名称	exti_lock_enable
函数原形	void exti_lock_enable(void);
功能描述	使能 EXTI 安全属性和特权访问配置锁
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 lock EXTI security attribution and privileged access configuration */
```

```
exti_lock_enable();
```

### 函数 exti\_software\_interrupt\_enable

函数exti\_software\_interrupt\_enable描述见下表：

表 3-216. 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，参考 <a href="#">表 3-202. 枚举类型 exti_line_enum</a>
EXTI_x	x=0,1,2..28
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 3-217. 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, 参考 <a href="#">表 3-202. 枚举类型 exti_line_enum</a>
EXTI_x	x=0,1,2..28
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 3-218. 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, 参考 <a href="#">表 3-202. 枚举类型 exti_line_enum</a>
EXTI_x	x=0,1,2..28
输出参数{out}	
-	-
返回值	
FlagStatus	SET 或 RESET

例如：

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

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

### 函数 exti\_flag\_clear

函数exti\_flag\_clear描述见下表:

表 3-219. 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	-
输入参数{in}	
linex	EXTI 线 x, 参考 <a href="#">表 3-202. 枚举类型 exti_line_enum</a>
EXTI_x	x=0,1,2..28
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* clear EXTI line 0 flag status */
exti_flag_clear(EXTI_0);
```

### 函数 exti\_interrupt\_flag\_get

函数exti\_interrupt\_flag\_get描述见下表:

表 3-220. 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, 参考 <a href="#">表 3-202. 枚举类型 exti_line_enum</a>
EXTI_x	x=0,1,2..28
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 3-221. 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, 参考 <a href="#">表 3-202. 枚举类型 exti_line_enum</a>
EXTI_x	x=0,1,2..28
输出参数{out}	
-	-
返回值	
-	-

例如：

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

```
exti_interrupt_flag_clear(EXTI_0);
```

## 3.10. 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[3.10.1](#)描述了FMC的寄存器列表，章节[9](#)对FMC库函数进行说明。

### 3.10.1. 外设寄存器说明

FMC寄存器列表如下：

表 3-222. FMC 寄存器

寄存器	描述
FMC_KEY	解锁寄存器
FMC_OBKEY	选项字节解锁寄存器
FMC_STAT	状态寄存器
FMC_CTL	控制寄存器
FMC_ADDR	地址寄存器
FMC_OBSTAT	选项字节状态寄存器
FMC_SECKEY	安全解锁寄存器
FMC_SECSTAT	安全状态寄存器

寄存器	描述
FMC_SECCTL	安全控制寄存器
FMC_SECADDR	安全地址寄存器
FMC_OBR	选项字节寄存器
FMC_OBUSER	选项字节用户寄存器
FMC_SECMCFG0	安全标记配置寄存器0
FMC_DMP0	安全特定标记保护区域寄存器0
FMC_OBWRP0	选项字节写保护/擦除保护寄存器0
FMC_SECMCFG1	安全标记配置寄存器1
FMC_DMP1	安全特定标记保护区域寄存器1
FMC_OBWRP1	选项字节写保护/擦除保护寄存器1
FMC_SECMCFG2	安全标记配置寄存器2
FMC_SECMCFG3	安全标记配置寄存器3
FMC_NODEC0	NO-RTDEC区域寄存器0
FMC_NODEC1	NO-RTDEC区域寄存器1
FMC_NODEC2	NO-RTDEC区域寄存器2
FMC_NODEC3	NO-RTDEC区域寄存器3
FMC_OFRG	偏移区域寄存器
FMC_OFVR	偏移值寄存器
FMC_DMPCTL	DMP控制寄存器
FMC_PRIVCFG	特权访问配置寄存器
FMC_PID	产品ID寄存器

### 3.10.2. 外设库函数说明

FMC固件库函数列举如下表：

**表 3-223. FMC 固件库函数**

函数名称	函数描述
fmc_unlock	解锁FMC主编程块操作
fmc_lock	锁定FMC主编程块操作
fmc_page_erase	FMC页擦除
fmc_mass_erase	FMC全片擦除
fmc_word_program	在相应地址字编程
fmc_continuous_program	在相应地址连续字编程
sram1_reset_enable	使能系统复位后自动清除SRAM1功能
sram1_reset_disable	禁能系统复位后自动清除SRAM1功能
fmc_privilege_enable	使能特权访问
fmc_privilege_disable	禁能特权访问
ob_unlock	解锁选项字节操作
ob_lock	锁定选项字节操作
ob_start	发送选项字节修改开始命令
ob_reload	重新载入选项字节

函数名称	函数描述
ob_security_protection_config	配置安全保护
ob_trustzone_enable	使能Trust Zone
ob_trustzone_disable	禁能Trust Zone
ob_user_write	用户选项字节编程
ob_write_protection_config	配置擦写保护区域
ob_secmark_config	配置安全标记保护区域
ob_dmp_access_enable	使能DMP区域访问权限
ob_dmp_access_disable	禁能DMP区域访问权限
ob_dmp_config	配置安全特定标记保护区域
ob_dmp_enable	使能安全特定标记保护区域
ob_dmp_disable	禁能安全特定标记保护区域
fmc_no_rtdec_config	配置不即时解密区域
fmc_offset_region_config	配置读偏移功能应用区域
fmc_offset_value_config	配置读偏移功能偏移值
ob_write_protection_get	获取写保护选项字节状态，仅可以获取由EFUSE设置的闪存写保护状态
ob_user_get	获取选项字节USER
ob_security_protection_flag_get	获取安全保护选项字节
ob_trustzone_state_get	获取Trust Zone状态
ob_memory_mode_state_get	获取MCU存储结构是FMC模式还是QSPI模式
ob_exist_state_get	获取选项字节是否存在
fmc_flag_get	检查FMC标志位是否置位
fmc_flag_clear	清除FMC标志
fmc_interrupt_enable	使能FMC中断
fmc_interrupt_disable	禁能FMC中断
fmc_interrupt_flag_get	获取FMC中断标志状态
fmc_interrupt_flag_clear	清除FMC中断标志状态

### 枚举类型 fmc\_state\_enum

表 3-224. 枚举类型 fmc\_state\_enum

枚举名称	枚举描述
FMC_READY	操作完成
FMC_BUSY	操作进行中
FMC_WPERR	写保护错误
FMC_TOERR	超时错误
FMC_OBERR	选项字节错误
FMC_SECERR	安全错误，仅用于安全模式

### 函数 fmc\_unlock

函数fmc\_unlock描述见下表：

表 3-225. 函数 fmc\_unlock

函数名称	fmc_unlock
函数原型	void fmc_unlock(void);
功能描述	解锁Flash操作
先决条件	-
被调用函数	-
输入参数{in}	
-	-
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* unlock the main FMC operation */
fmc_unlock();
```

### 函数 fmc\_lock

函数fmc\_lock描述见下表:

表 3-226. 函数 fmc\_lock

函数名称	fmc_lock
函数原型	void fmc_lock(void);
功能描述	锁定Flash操作
先决条件	-
被调用函数	-
输入参数{in}	
-	-
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* lock the main FMC operation */
fmc_lock();
```

### 函数 fmc\_page\_erase

函数fmc\_page\_erase描述见下表:

表 3-227. 函数 fmc\_page\_erase

函数名称	fmc_page_erase
------	----------------

函数原型	fmc_state_enum fmc_page_erase(uint32_t page_address);
功能描述	页擦除
先决条件	fmc_unlock
被调用函数	-
输入参数{in}	
page_address	页擦除首地址
输出参数{out}	
-	-
返回值	
fmc_state_enum	FMC状态，详细请参考 <a href="#">表3-224. 枚举类型fmc_state_enum</a>

例如：

```
/* erase page */

fmc_state_enum state;

fmc_unlock();

state = fmc_page_erase( 0x08004000);
```

### 函数 fmc\_mass\_erase

函数fmc\_mass\_erase描述见下表：

**表 3-228. 函数 fmc\_mass\_erase**

函数名称	fmc_mass_erase
函数原型	fmc_state_enum fmc_mass_erase(void);
功能描述	全片擦除
先决条件	fmc_unlock
被调用函数	-
输入参数{in}	
-	-
输出参数{out}	
-	-
返回值	
fmc_state_enum	FMC状态，详细请参考 <a href="#">表3-224. 枚举类型fmc_state_enum</a>

例如：

```
/* erase whole chip */

fmc_unlock();

fmc_state_enum state;

state = fmc_mass_erase();
```



**函数 fmc\_word\_program**

函数fmc\_word\_program描述见下表：

**表 3-229. 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	对相应地址字编程
先决条件	fmc_unlock, fmc_page_erase/fmc_mass_erase
被调用函数	-
输入参数{in}	
address	编程地址
输入参数{in}	
data	编程数据
输出参数{out}	
-	-
返回值	
fmc_state_enum	FMC状态，详细请参考 <a href="#">表3-224. 枚举类型fmc_state_enum</a>

例如：

```
/* program a word at the corresponding address */

fmc_state_enum state;

fmc_unlock();

fmc_page_erase(0x08004000);

state = fmc_word_program (0x08004000, 0xaabbccdd);
```

**函数 fmc\_continuous\_program**

函数fmc\_continuous\_program描述见下表：

**表 3-230. 函数 fmc\_continuous\_program**

函数名称	fmc_continuous_program
函数原型	fmc_state_enum fmc_continuous_program(uint32_t address, uint32_t data[], uint32_t size);
功能描述	对相应地址连续字编程
先决条件	fmc_unlock, fmc_page_erase/fmc_mass_erase
被调用函数	-
输入参数{in}	
address	编程地址
输入参数{in}	
data[]	编程数据
输入参数{in}	

<b>size</b>	编程数据个数，必须是4字节对齐
<b>输出参数{out}</b>	
-	-
<b>返回值</b>	
<b>fmc_state_enum</b>	FMC状态，详细请参考 <a href="#">表3-224. 枚举类型fmc_state_enum</a>

例如：

```
/* program a word at the corresponding address */

uint32_t data[16]= {0x01234567, 0x01234567, 0x01234567, 0x01234567, 0x01234567,
0x01234567, 0x01234567, 0x01234567, 0x01234567, 0x01234567, 0x01234567,
0x01234567, 0x01234567, 0x01234567, 0x01234567, 0x01234567};

fmc_state_enum state;

fmc_unlock();

state = fmc_word_program(0x08004000, data[],16);
```

### 函数 sram1\_reset\_enable

函数sram1\_reset\_enable描述见下表：

**表 3-231. 函数 sram1\_reset\_enable**

<b>函数名称</b>	sram1_reset_enable
<b>函数原型</b>	void sram1_reset_enable(void);
<b>功能描述</b>	使能系统复位后自动清除SRAM1功能
<b>先决条件</b>	ob_unlock
<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 SRAM1 reset automatically function */

ob_unlock();

sram1_reset_enable();
```

### 函数 sram1\_reset\_disable

函数sram1\_reset\_disable描述见下表：

**表 3-232. 函数 sram1\_reset\_disable**

<b>函数名称</b>	sram1_reset_disable
-------------	---------------------

函数原型	void sram1_reset_disable(void);
功能描述	禁能系统复位后自动清除SRAM1功能
先决条件	ob_unlock
被调用函数	-
输入参数{in}	
-	-
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* disable SRAM1 reset automatically function */
ob_unlock();
sram1_reset_disable();
```

### 函数 fmc\_privilege\_enable

函数fmc\_privilege\_enable描述见下表：

表 3-233. 函数 fmc\_privilege\_enable

函数名称	fmc_privilege_enable
函数原型	void fmc_privilege_enable(void);
功能描述	使能特权访问
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 privileged access */
fmc_privilege_enable();
```

### 函数 fmc\_privilege\_disable

函数fmc\_privilege\_disable描述见下表：

表 3-234. 函数 fmc\_privilege\_disable

函数名称	fmc_privilege_disable
------	-----------------------

函数原型	void fmc_privilege_disable(void);
功能描述	禁能特权访问
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 privileged access */
fmc_privilege_disable();

/* enable the privileged access */
fmc_privilege_enable();
```

### 函数 ob\_unlock

函数ob\_unlock描述见下表：

表 3-235. 函数 ob\_unlock

函数名称	ob_unlock
函数原型	void ob_unlock(void);
功能描述	解锁选项字节
先决条件	-
被调用函数	-
输入参数{in}	
-	-
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* unlock the option bytes operation */
ob_unlock();
```

### 函数 ob\_lock

函数ob\_lock描述见下表：

表 3-236. 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 bytes operation */
```

```
ob_lock();
```

#### 函数 ob\_start

函数ob\_start描述见下表：

表 3-237. 函数 ob\_start

函数名称	ob_start
函数原型	void ob_start(void)
功能描述	发送选项字节修改开始命令
先决条件	ob_unlock
被调用函数	-
输入参数{in}	
-	-
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* program option bytes USER data */
```

```
ob_unlock();
```

```
ob_user_write(0xFFFF);
```

```
ob_start();
```

## 函数 ob\_reload

函数ob\_reload描述见下表：

**表 3-238. 函数 ob\_reload**

函数名称	ob_reload
函数原型	void ob_reload(void)
功能描述	发送选项字节修改开始命令
先决条件	ob_unlock
被调用函数	-
输入参数{in}	
-	-
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* program option bytes USER data */
ob_unlock();
ob_user_write(0XFFFF);
ob_start();
ob_reload();
```

## 函数 ob\_security\_protection\_config

函数ob\_security\_protection\_config描述见下表：

**表 3-239. 函数 ob\_security\_protection\_config**

函数名称	ob_security_protection_config
函数原型	fmc_state_enum ob_security_protection_config(uint8_t ob_spc);
功能描述	配置安全保护
先决条件	ob_unlock
被调用函数	-
输入参数{in}	
ob_spc	选项字节安全保护值
FMC_NSPC	无保护
FMC_SPC_P0_5	保护等级0.5
FMC_SPC_P1	保护等级1
输出参数{out}	
-	-
返回值	

<b>fmc_state_enum</b>	FMC状态，详细请参考 <a href="#">表3-224. 枚举类型fmc_state_enum</a>
-----------------------	--

例如：

```
/* configure low security protection */

ob_unlock();

ob_security_protection_config (FMC_NSPC);

ob_start();
```

### 函数 ob\_trustzone\_enable

函数ob\_trustzone\_enable描述见下表：

**表 3-240. 函数 ob\_trustzone\_enable**

函数名称	ob_trustzone_enable
函数原型	fmc_state_enum ob_trustzone_enable(void);
功能描述	使能Trust Zone
先决条件	ob_unlock
被调用函数	-
输入参数{in}	
-	-
输出参数{out}	
-	-
返回值	
<b>fmc_state_enum</b>	FMC状态，详细请参考 <a href="#">表3-224. 枚举类型fmc_state_enum</a>

例如：

```
/* enable trustzone */

ob_unlock();

trustzone_enable();
```

### 函数 ob\_trustzone\_disable

函数ob\_trustzone\_disable描述见下表：

**表 3-241. 函数 ob\_trustzone\_disable**

函数名称	ob_trustzone_disable
函数原型	ErrStatus ob_trustzone_disable(void);
功能描述	禁能Trust Zone
先决条件	ob_unlock
被调用函数	-
输入参数{in}	
-	-

输出参数{out}	
-	-
返回值	
ErrStatus	ERROR或SUCCESS

例如：

```
/* disable trustzone */
```

```
ErrStatus state;
```

```
ob_unlock();
```

```
state = trustzone_disable();
```

### 函数 ob\_user\_write

函数ob\_user\_write描述见下表：

表 3-242. 函数 ob\_user\_write

函数名称	ob_user_write
函数原型	fmc_state_enum ob_user_write(uint16_t ob_user);
功能描述	用户选项字节编程
先决条件	ob_unlock
被调用函数	-
输入参数{in}	
ob_user	选项字节USER值
输出参数{out}	
-	-
返回值	
fmc_state_enum	FMC状态，详细请参考 <a href="#">表3-224. 枚举类型fmc_state_enum</a>

例如：

```
/* program option bytes USER data */
```

```
ob_unlock();
```

```
ob_user_write(0xFFFF);
```

```
ob_start();
```

### 函数 ob\_write\_protection\_config

函数ob\_write\_protection\_config描述见下表：

表 3-243. 函数 ob\_write\_protection\_config

函数名称	ob_write_protection_config
函数原型	fmc_state_enum ob_write_protection_config(uint32_t wrp_spage, uint32_t wrp_epage, uint32_t wrp_register_index);



功能描述	配置擦写保护区域
先决条件	-
被调用函数	-
输入参数{in}	
wrp_spage	擦写保护区域起始页，0~512
输入参数{in}	
wrp_epage	擦写保护区域末尾页，0~512
输入参数{in}	
wrp_register_index	FMC_OBWRP <sub>x</sub> register index
OBWRP_INDEX0	option byte write protection area register 0
OBWRP_INDEX1	option byte write protection area register 1
输出参数{out}	
-	-
返回值	
fmc_state_enum	FMC状态，详细请参考 <a href="#">表3-224. 枚举类型fmc_state_enum</a>

例如：

```
/* configure write protection pages */
```

```
ob_unlock();
```

```
ob_write_protection_config(WRP_REGION_SPAGE, WRP_REGION_EPAGE, OBWRP_INDEX0);
```

```
ob_start();
```

### 函数 ob\_secmark\_config

函数ob\_secmark\_config描述见下表：

表 3-244. 函数 ob\_secmark\_config

函数名称	ob_secmark_config
函数原型	void ob_secmark_config(uint32_t secm_spage, uint32_t secm_epage, uint32_t secm_register_index);
功能描述	配置安全标记保护区域
先决条件	ob_unlock
被调用函数	-
输入参数{in}	
secm_spage	安全标记区域起始页，0~0x03FF
输入参数{in}	
secm_epage	安全标记区域末尾页，0~0x03FF
输入参数{in}	
secm_register_index	安全标记区配置寄存器
SECM_INDEX0	安全标记区配置寄存器0

SECM_INDEX1	安全标记区配置寄存器1
SECM_INDEX2	安全标记区配置寄存器2
SECM_INDEX3	安全标记区配置寄存器3
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* configure secure mark pages */
```

```
ob_unlock();
```

```
ob_secmark_config(SECM_REGION_SPAGE,SECM_REGION_EPAGE,SECM_INDEX3);
```

### 函数 ob\_dmp\_access\_enable

函数ob\_dmp\_access\_enable描述见下表：

表 3-245. 函数 ob\_dmp\_access\_enable

函数名称	ob_dmp_access_enable
函数原型	void ob_dmp_access_enable(uint32_t dmp_register_index);
功能描述	使能DMP区域访问权限
先决条件	-
被调用函数	-
输入参数{in}	
dmp_register_index	DMP区域配置寄存器
DMP_INDEX0	DMP区域配置寄存器0
DMP_INDEX1	DMP区域配置寄存器1
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* enable DMP region access right */
```

```
ob_dmp_access_enable(DMP_INDEX0);
```

### 函数 ob\_dmp\_access\_disable

函数ob\_dmp\_access\_disable描述见下表：

表 3-246. 函数 ob\_dmp\_access\_disable

函数名称	ob_dmp_access_disable
------	-----------------------

函数原型	void ob_dmp_access_disable(uint32_t dmp_register_index);
功能描述	禁能DMP区域访问权限
先决条件	ob_unlock
被调用函数	-
输入参数{in}	
dmp_register_index	DMP区域配置寄存器
DMP_INDEX0	DMP区域配置寄存器0
DMP_INDEX1	DMP区域配置寄存器1
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* disable DMP region access right */
ob_unlock();
ob_dmp_access_disable(DMP_INDEX0);
```

### 函数 ob\_dmp\_config

函数ob\_dmp\_config描述见下表：

表 3-247. 函数 ob\_dmp\_config

函数名称	ob_dmp_config
函数原型	ErrStatus ob_dmp_config(uint32_t dmp_epage, uint32_t dmp_register_index);
功能描述	配置安全特定标记保护区域
先决条件	ob_unlock
被调用函数	-
输入参数{in}	
dmp_epage	DMP区域末尾页，0~0x03FF
输入参数{in}	
dmp_register_index	DMP区域配置寄存器
DMP_INDEX0	DMP区域配置寄存器0
DMP_INDEX1	DMP区域配置寄存器1
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* configure DMP pages */
```

```
ob_unlock();
```

```
ob_dmp_config(DMP_REGION_EPAGE, DMP_INDEX0);
```

### 函数 ob\_dmp\_enable

函数ob\_dmp\_enable描述见下表：

**表 3-248. 函数 ob\_dmp\_enable**

函数名称	ob_dmp_enable
函数原型	fmc_state_enum ob_dmp_enable(uint32_t dmp_register_index);
功能描述	使能安全特定标记保护区
先决条件	ob_unlock
被调用函数	-
输入参数{in}	
dmp_register_index	DMP区域配置寄存器
DMP_INDEX0	DMP区域配置寄存器0
DMP_INDEX1	DMP区域配置寄存器1
输出参数{out}	
-	-
返回值	
fmc_state_enum	FMC状态，详细请参考 <a href="#">表3-224. 枚举类型fmc_state_enum</a>

例如：

```
/* enable DMP function */
ob_unlock();
ob_dmp_enable(DMP_INDEX0);
```

### 函数 ob\_dmp\_disable

函数ob\_dmp\_disable描述见下表：

**表 3-249. 函数 ob\_dmp\_disable**

函数名称	ob_dmp_disable
函数原型	fmc_state_enum ob_dmp_disable(uint32_t dmp_register_index);
功能描述	禁能安全特定标记保护区
先决条件	ob_unlock
被调用函数	-
输入参数{in}	
dmp_register_index	DMP区域配置寄存器
DMP_INDEX0	DMP区域配置寄存器0
DMP_INDEX1	DMP区域配置寄存器1

输出参数{out}	
-	-
返回值	
<b>fmc_state_enum</b>	FMC状态，详细请参考 <a href="#">表3-224. 枚举类型fmc_state_enum</a>

例如：

```
/* disable DMP function */

ob_unlock();

ob_dmp_disable(DMP_INDEX0);
```

### 函数 fmc\_no\_rtdec\_config

函数fmc\_no\_rtdec\_config描述见下表：

**表 3-250. 函数 fmc\_no\_rtdec\_config**

函数名称	fmc_no_rtdec_config
函数原型	void fmc_no_rtdec_config(uint32_t nodec_spage, uint32_t nodec_epage, uint32_t nodec_register_index);
功能描述	配置不即时解密区域
先决条件	ob_unlock
被调用函数	-
输入参数{in}	
<b>nodec_spage</b>	NO-RTDEC区域起始页，0~0x1FFF
输入参数{in}	
<b>nodec_epage</b>	NO-RTDEC区域末尾页，0~0x1FFF
输入参数{in}	
<b>nodec_register_index</b>	NO-RTDEC区域配置寄存器
<i>NODEC_INDEX0</i>	NO-RTDEC区域配置寄存器0
<i>NODEC_INDEX1</i>	NO-RTDEC区域配置寄存器1
<i>NODEC_INDEX2</i>	NO-RTDEC区域配置寄存器2
<i>NODEC_INDEX3</i>	NO-RTDEC区域配置寄存器3
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* configure NO-RTDEC pages */

ob_unlock();

fmc_no_rtdec_config(NODEC_REGION_SPAGE, NODEC_REGION_EPAGE, NODEC_INDEX0);
```

## 函数 fmc\_offset\_region\_config

函数fmc\_offset\_region\_config描述见下表:

表 3-251. 函数 fmc\_offset\_region\_config

函数名称	fmc_offset_region_config
函数原型	void fmc_offset_region_config(uint32_t of_spage, uint32_t of_epage);
功能描述	配置读偏移功能应用区域
先决条件	ob_unlock
被调用函数	-
输入参数{in}	
of_spage	读偏移区域起始页，0~0x1FFF
输入参数{in}	
of_epage	读偏移区域末尾页，0~0x1FFF
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* configure offset region */
```

```
ob_unlock();
```

```
fmc_offset_region_config(SOURCE_START_PAGE, SOURCE_END_PAGE);
```

## 函数 fmc\_offset\_value\_config

函数fmc\_offset\_value\_config描述见下表:

表 3-252. 函数 fmc\_offset\_value\_config

函数名称	fmc_offset_value_config
函数原型	void fmc_offset_value_config(uint32_t of_value);
功能描述	配置读偏移功能偏移值
先决条件	ob_unlock
被调用函数	-
输入参数{in}	
of_value	读偏移值，0~0x1FFF
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* configure offset value */
```

```
ob_unlock();
```

```
fmc_offset_value_config(PAGE_OFFSET_VALUE);
```

### 函数 ob\_write\_protection\_get

函数ob\_write\_protection\_get描述见下表：

表 3-253. 函数 ob\_write\_protection\_get

函数名称	ob_write_protection_get
函数原型	FlagStatus ob_write_protection_get(void);
功能描述	获取擦写保护状态，仅反映通过EFUSE模块设置的擦写保护。
先决条件	-
被调用函数	-
输入参数{in}	
-	-
输出参数{out}	
-	-
返回值	
FlagStatus	SET或RESET

例如：

```
/* get the FMC write protection status */
```

```
FlagStatus status;
```

```
status = ob_write_protection_get();
```

### 函数 ob\_user\_get

函数ob\_user\_get描述见下表：

表 3-254. 函数 ob\_user\_get

函数名称	ob_user_get
函数原型	uint16_t ob_user_get(void);
功能描述	获取选项字节USER
先决条件	-
被调用函数	-
输入参数{in}	
-	-
输出参数{out}	
-	-
返回值	
FlagStatus	SET或RESET

例如：

```
/* get the value of option bytes USER */
```

```
uint16_t user_value;
```

```
user_value = ob_user_get();
```

### 函数 ob\_security\_protection\_flag\_get

函数ob\_security\_protection\_flag\_get描述见下表:

**表 3-255. 函数 ob\_security\_protection\_flag\_get**

函数名称	ob_security_protection_flag_get
函数原型	FlagStatus ob_security_protection_flag_get(uint32_t spc_state);
功能描述	获取安全保护选项字节
先决条件	-
被调用函数	-
输入参数{in}	
ob_spc	选项字节安全保护值
FMC_NSPC	无保护
FMC_SPC_P0_5	保护等级0.5
FMC_SPC_P1	保护等级1
输出参数{out}	
-	-
返回值	
FlagStatus	SET或RESET

例如:

```
/* check FMC option bytes security protection level 1 is set or not */
```

```
FlagStatus status;
```

```
status = ob_security_protection_flag_get(OB_FLAG_SPC1);
```

### 函数 ob\_trustzone\_state\_get

函数ob\_trustzone\_state\_get描述见下表:

**表 3-256. 函数 ob\_trustzone\_state\_get**

函数名称	ob_trustzone_state_get
函数原型	FlagStatus ob_trustzone_state_get(void);
功能描述	获取Trust Zone状态
先决条件	-
被调用函数	-
输入参数{in}	
-	-
输出参数{out}	
-	-



返回值	
FlagStatus	SET或RESET

例如：

```
/* get trustzone state */
```

```
FlagStatus status;
```

```
status = ob_trustzone_state_get();
```

### 函数 ob\_memory\_mode\_state\_get

函数ob\_memory\_mode\_state\_get描述见下表：

表 3-257. 函数 ob\_memory\_mode\_state\_get

函数名称	ob_memory_mode_state_get
函数原型	FlagStatus ob_memory_mode_state_get(void);
功能描述	获取MCU是SIP Flash还是EXT Flash状态
先决条件	-
被调用函数	-
输入参数{in}	
-	-
输出参数{out}	
-	-
返回值	
FlagStatus	SET或RESET

例如：

```
/* get the state of MCU memory structure is FMC mode or QSPI mode */
```

```
FlagStatus status;
```

```
status = ob_memory_mode_state_get();
```

### 函数 ob\_exist\_state\_get

函数ob\_exist\_state\_get描述见下表：

表 3-258. 函数 ob\_exist\_state\_get

函数名称	ob_exist_state_get
函数原型	FlagStatus ob_exist_state_get(void);
功能描述	获取MCU是否存在选项字节状态
先决条件	-
被调用函数	-
输入参数{in}	
-	-
输出参数{out}	

-	-
返回值	
FlagStatus	SET或RESET

例如:

```
/* get the state of whether the option byte exist or not */
```

```
FlagStatus status;
```

```
status = ob_exist_state_get();
```

### 函数 fmc\_flag\_get

函数fmc\_flag\_get描述见下表:

表 3-259. 函数 fmc\_flag\_get

函数名称	fmc_flag_get
函数原型	FlagStatus fmc_flag_get(uint32_t 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}	
flag	FMC标志
FMC_FLAG_BUSY	FMC忙碌标志
FMC_FLAG_OBER R	FMC选项字节操作错误标志
FMC_FLAG_WPER R	FMC写保护编程/擦除错误标志
FMC_FLAG_END	FMC操作完成标志
FMC_FLAG_SECE RR	FMC安全错误标志
输出参数{out}	
-	-
返回值	
FlagStatus	SET或RESET

例如:

```
/* check the FMC_FLAG_END flag set or not*/
```

```
FlagStatus flag;
```

```
flag = fmc_flag_get(FMC_FLAG_END);
```

### 函数 fmc\_flag\_clear

函数fmc\_flag\_clear描述见下表:

表 3-260. 函数 `fmc_flag_clear`

函数名称	<code>fmc_flag_clear</code>
函数原型	<code>void fmc_flag_clear(uint32_t flag);</code>
功能描述	清除FMC标志
先决条件	-
被调用函数	-
输入参数{in}	
<b>flag</b>	清除FMC标志
<code>FMC_FLAG_BUSY</code>	FMC忙碌标志
<code>FMC_FLAG_OBER</code> <i>R</i>	FMC选项字节操作错误标志
<code>FMC_FLAG_WPER</code> <i>R</i>	FMC写保护编程/擦除错误标志
<code>FMC_FLAG_END</code>	FMC操作完成标志
<code>FMC_FLAG_SECE</code> <i>RR</i>	FMC安全错误标志
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 3-261. 函数 `fmc_interrupt_enable`

函数名称	<code>fmc_interrupt_enable</code>
函数原型	<code>void fmc_interrupt_enable(uint32_t interrupt);</code>
功能描述	使能FMC中断
先决条件	<code>fmc_unlock</code>
被调用函数	-
输入参数{in}	
<b>interrupt</b>	FMC中断
<code>FMC_INT_END</code>	FMC编程完成中断
<code>FMC_INT_ERR</code>	FMC错误中断
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* enable FMC interrupt */

fmc_unlock();

fmc_interrupt_enable(FMC_INT_END);
```

### 函数 fmc\_interrupt\_disable

函数fmc\_interrupt\_disable描述见下表：

表 3-262. 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	禁能FMC中断
先决条件	fmc_unlock
被调用函数	-
输入参数{in}	
interrupt	FMC中断
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：

```
/* disable FMC interrupt */

fmc_unlock();

fmc_interrupt_disable(FMC_INT_END);
```

### 函数 fmc\_interrupt\_flag\_get

函数fmc\_interrupt\_flag\_get描述见下表：

表 3-263. 函数 fmc\_interrupt\_flag\_get

函数名称	fmc_interrupt_flag_get
函数原型	FlagStatus fmc_interrupt_flag_get(fmc_interrupt_flag_enum flag);
功能描述	获取FMC中断标志状态
先决条件	-
被调用函数	-
输入参数{in}	
flag	中断标志
FMC_INT_FLAG_W	FMC擦写保护错误中断标志

<i>PERR</i>	
<i>FMC_INT_FLAG_END</i>	FMC操作完成中断标志
<i>FMC_INT_FLAG_SECCERR</i>	FMC安全错误中断标志
输出参数{out}	
-	-
返回值	
FlagStatus	SET或RESET

例如:

```
/* check operation interrupt flag is set or not */
```

```
FlagStatus flag;
```

```
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 3-264. 函数 fmc\_interrupt\_flag\_clear

函数名称	fmc_interrupt_flag_clear
函数原型	FlagStatus fmc_interrupt_flag_clear (fmc_interrupt_flag_enum flag);
功能描述	清除FMC中断标志
先决条件	-
被调用函数	-
输入参数{in}	
flag	清除FMC中断标志
<i>FMC_INT_FLAG_WPERR</i>	FMC擦写保护错误中断标志
<i>FMC_INT_FLAG_END</i>	FMC操作完成中断标志
<i>FMC_INT_FLAG_SECCERR</i>	FMC安全错误中断标志
输出参数{out}	
-	-
返回值	
FlagStatus	SET或RESET

例如:

```
/* clear operation interrupt flag */
```

```
fmc_interrupt_flag_clear(FMC_INT_END);
```

### 3.11. 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[3.11.1](#)描述了FWDGT的寄存器列表，章节[3.11.2](#)对FWDGT库函数进行说明。

#### 3.11.1. 外设寄存器说明

FWDGT寄存器列表如下表所示：

表 3-265. 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

#### 3.11.2. 外设库函数说明

FWDGT库函数列表如下表所示：

表 3-266. FWDGT 库函数

库函数名称	库函数描述
fwdgt_write_enable	使能对寄存器FWDGT_PSC和FWDGT_RLD的写操作
fwdgt_write_disable	失能对寄存器FWDGT_PSC和FWDGT_RLD的写操作
fwdgt_enable	使能FWDGT
fwdgt_prescaler_value_config	配置FWDGT预分频值
fwdgt_reload_value_config	配置FWDGT重装载值
fwdgt_counter_reload	重装载FWDGT计数器
fwdgt_config	设置FWDGT重装载值、预分频值
fwdgt_flag_get	获取FWDGT标志位状态

#### 函数 fwdgt\_write\_enable

函数fwdgt\_write\_enable描述见下表：

表 3-267. 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 3-268. 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 3-269. 函数 fwdgt\_enable

函数名称	fwdgt_enable
函数原型	void fwdgt_enable(void);
功能描述	使能FWDGT
先决条件	-
被调用函数	-
输入参数{in}	
-	-
输出参数{out}	
-	-

返回值	
-	-

例如：

```
/* start the free watchdog timer counter */
```

```
fwdgt_enable ( );
```

### 函数 **fwdgt\_prescaler\_value\_config**

函数fwdgt\_prescaler\_value\_config描述见下表：

**表 3-270. 函数 fwdgt\_prescaler\_value\_config**

函数名称	fwdgt_prescaler_value_config
函数原型	void fwdgt_prescaler_value_config (uint16_t prescaler_value);
功能描述	配置FWDGT预分频值
先决条件	-
被调用函数	-
输入参数{in}	
<b>prescaler_value</b>	指定FWDGT预分频值
<i>FWDGT_PSC_DIV4</i>	FWDGT预分频值设为4
<i>FWDGT_PSC_DIV8</i>	FWDGT预分频值设为8
<i>FWDGT_PSC_DIV16</i>	FWDGT预分频值设为16
<i>FWDGT_PSC_DIV32</i>	FWDGT预分频值设为32
<i>FWDGT_PSC_DIV64</i>	FWDGT预分频值设为64
<i>FWDGT_PSC_DIV128</i>	FWDGT预分频值设为128
<i>FWDGT_PSC_DIV256</i>	FWDGT预分频值设为256
输出参数{out}	
-	-
返回值	
<b>ErrStatus</b>	ERROR或SUCCESS

例如：

```
/* configure the FWDGT counter prescaler value */
```

```
fwdgt_prescaler_value_config (FWDGT_PSC_DIV8);
```

### 函数 **fwdgt\_reload\_value\_config**

函数fwdgt\_reload\_value\_config描述见下表：

**表 3-271. 函数 fwdgt\_reload\_value\_config**

函数名称	fwdgt_reload_value_config
函数原型	void fwdgt_reload_value_config(uint16_t reload_value);
功能描述	配置FWDGT重装载值
先决条件	-



被调用函数	-
输入参数{in}	
reload_value	指定FWDGT重装载值(0x0000-0x0FFF)
输出参数{out}	
-	-
返回值	
ErrStatus	ERROR或SUCCESS

例如：

```
/* configure the FWDGT counter reload value */
```

```
f fwdgt_reload_value_config(625);
```

### 函数 fwdgt\_counter\_reload

函数fwdgt\_counter\_reload描述见下表：

表 3-272. 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	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 3-273. 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 - 0x0FFF)
<b>输入参数{in}</b>	
<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预分频值设为128
<i>FWDGT_PSC_DIV256</i>	FWDGT预分频值设为256
<b>输出参数{out}</b>	
-	-
<b>返回值</b>	
<b>ErrStatus</b>	ERROR or SUCCESS

例如:

```
/* configure FWDGT counter clock: 40KHz(IRC40K) / 64 = 0.625 KHz */
fwdgt_config(2*500, FWDGT_PSC_DIV64);
```

### 函数 fwdgt\_flag\_get

函数fwdgt\_flag\_get描述见下表:

表 3-274. 函数 fwdgt\_flag\_get

<b>函数名称</b>	fwdgt_flag_get
<b>函数原型</b>	FlagStatus fwdgt_flag_get(uint16_t flag);
<b>功能描述</b>	获取FWDGT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>	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_RUD</i>	重装载值更新进行中
<b>输出参数{out}</b>	
-	-
<b>返回值</b>	
<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);
```

## 3.12. GPIO

GPIO用来实现各片上设备的逻辑输入/输出功能。章节[3.12.1](#)描述了GPIO的寄存器列表，章节[3.12.2](#)对GPIO库函数进行说明。

### 3.12.1. 外设寄存器说明

GPIO寄存器列表如下表所示:

表 3-275. GPIO 寄存器

寄存器名称	寄存器描述
GPIOx_CTL	端口控制寄存器
GPIOx_OMODE	端口输出模式寄存器
GPIOx_OSPD	端口输出速度寄存器
GPIOx_PUD	端口上拉/下拉寄存器
GPIOx_ISTAT	端口输入状态寄存器
GPIOx_OCTL	端口输出控制寄存器
GPIOx_BOP	端口位操作寄存器
GPIOx_LOCK	端口配置锁定寄存器
GPIOx_AFSEL0	备用功能选择寄存器0
GPIOx_AFSEL1	备用功能选择寄存器1
GPIOx_BC	位清除寄存器
GPIOx_TG	端口位翻转寄存器
GPIOx_SCFG	端口安全配置寄存器

### 3.12.2. 外设库函数说明

GPIO库函数列表如下表所示:

表 3-276. GPIO 库函数

库函数名称	库函数描述
gpio_deinit	复位外设GPIOx
gpio_mode_set	设置GPIO模式
gpio_output_options_set	设置GPIO输出模式和速度
gpio_bit_set	置位引脚值
gpio_bit_reset	复位引脚值
gpio_bit_write	将特定的值写入引脚

库函数名称	库函数描述
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
gpio_bit_set_sec_cfg	配置引脚安全配置位状态为置位
gpio_bit_reset_sec_cfg	配置引脚安全配置位状态为复位
gpio_sec_cfg_bit_get	获取引脚安全配置位状态

### 函数 gpio\_deinit

函数gpio\_deinit描述见下表:

表 3-277. 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	-
输入参数{in}	
gpio_periph	GPIO端口
GPIOx	端口选择(x = A,B,C)
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 3-278. 函数 gpio\_mode\_set

函数名称	gpio_mode_set
函数原型	void gpio_mode_set(uint32_t gpio_periph, uint32_t mode, uint32_t pull_up_down, uint32_t pin);

功能描述	设置GPIO模式
先决条件	-
被调用函数	-
输入参数{in}	
gpio_periph	GPIO端口
GPIOx	GPIOx(x = A,B,C)
输入参数{in}	
mode	GPIO引脚模式
GPIO_MODE_INPUT	输入模式
GPIO_MODE_OUTPUT	输出模式
GPIO_MODE_AF	备用功能模式
GPIO_MODE_ANALOG	模拟模式
输入参数{in}	
pull_up_down	GPIO引脚上拉下拉电阻设置
GPIO_PUPD_NONE	悬空模式，无上拉和下拉
GPIO_PUPD_PULLUP	带上拉电阻
GPIO_PUPD_PULLDOWN	带下拉电阻
输入参数{in}	
pin	GPIO pin
GPIO_PIN_x	引脚选择(x=0..15)
GPIO_PIN_ALL	所有引脚
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* configure PA0 as input mode with pullup */
```

```
gpio_mode_set(GPIOA, GPIO_MODE_INPUT, GPIO_PUPD_PULLUP, GPIO_PIN_0);
```

### 函数 gpio\_output\_options\_set

函数gpio\_output\_options\_set描述见下表：

表 3-279. 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)
输入参数{in}	
otype	GPIO引脚输出模式
GPIO_OTYPE_PP	推挽输出模式
GPIO_OTYPE_OD	开漏输出模式
输入参数{in}	
speed	GPIO引脚输出最大速度
GPIO_OSPEED_2MHZ	最大输出速度为2MHz
GPIO_OSPEED_10MHZ	最大输出速度为10MHz
GPIO_OSPEED_25MHZ	最大输出速度为25MHz
GPIO_OSPEED_166MHZ	最大输出速度为166MHz
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 3-280. 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	置位引脚值
先决条件	-
被调用函数	-
输入参数{in}	
gpio_periph	GPIO端口

GPIOx	端口选择(x = A,B,C)
输入参数{in}	
pin	GPIO引脚
GPIO_PIN_x	引脚选择 (x=0..15)
GPIO_PIN_ALL	所有引脚
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* set PA0 */
```

```
gpio_bit_set(GPIOA, GPIO_PIN_0);
```

### 函数 gpio\_bit\_reset

函数gpio\_bit\_reset描述见下表:

表 3-281. 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	复位引脚值
先决条件	-
被调用函数	-
输入参数{in}	
gpio_periph	GPIO端口
GPIOx	端口选择(x = A,B,C)
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:

```
/* reset PA0 */
```

```
gpio_bit_reset(GPIOA, GPIO_PIN_0);
```

### 函数 gpio\_bit\_write

函数gpio\_bit\_write描述见下表:

表 3-282. 函数 `gpio_bit_write`

函数名称	<code>gpio_bit_write</code>
函数原型	<code>void gpio_bit_write(uint32_t gpio_periph, uint32_t pin, bit_status bit_value);</code>
功能描述	将特定的值写入引脚
先决条件	-
被调用函数	-
输入参数{in}	
<code>gpio_periph</code>	GPIO端口
<code>GPIOx</code>	端口选择(x = A,B,C)
输入参数{in}	
<code>pin</code>	GPIO引脚
<code>GPIO_PIN_x</code>	引脚选择 (x=0..15)
<code>GPIO_PIN_ALL</code>	所有引脚
输入参数{in}	
<code>bit_value</code>	设置或清除
<code>RESET</code>	清除引脚值
<code>SET</code>	设置引脚值
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* write 1 to PA0 */
```

```
gpio_bit_write(GPIOA, GPIO_PIN_0, SET);
```

### 函数 `gpio_port_write`

函数`gpio_port_write`描述见下表:

表 3-283. 函数 `gpio_port_write`

函数名称	<code>gpio_port_write</code>
函数原型	<code>void gpio_port_write(uint32_t gpio_periph, uint16_t data);</code>
功能描述	将特定的值写入端口
先决条件	-
被调用函数	-
输入参数{in}	
<code>gpio_periph</code>	GPIO端口
<code>GPIOx</code>	端口选择(x = A,B,C)
输入参数{in}	
<code>data</code>	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 3-284. 函数 gpio\_input\_bit\_get

函数名称	gpio_input_bit_get
函数原型	FlagStatus gpio_input_bit_get(uint32_t gpio_periph,uint32_t pin);
功能描述	获取引脚的输入值
先决条件	-
被调用函数	-
输入参数{in}	
gpio_periph	GPIO端口
GPIOx	端口选择(x = A,B,C)
输入参数{in}	
pin	GPIO引脚
GPIO_PIN_x	引脚选择 (x=0..15)
GPIO_PIN_ALL	所有引脚
输出参数{out}	
-	-
返回值	
FlagStatus	SET或RESET

例如：

```
/* get input value of Port A */
```

```
FlagStatus bit_state = gpio_input_bit_get(GPIOA);
```

### 函数 gpio\_input\_port\_get

函数gpio\_input\_port\_get描述见下表：

表 3-285. 函数 gpio\_input\_port\_get

函数名称	gpio_input_port_get
函数原型	uint16_t gpio_input_port_get(uint32_t gpio_periph);
功能描述	获取端口的输入值
先决条件	-
被调用函数	-
输入参数{in}	

<b>gpio_periph</b>	GPIO端口
<i>GPIOx</i>	端口选择(x = A,B,C)
输出参数{out}	
-	-
返回值	
<b>uint16_t</b>	0x0000-0xFFFF

例如:

```
/* get output status of PA0 */
uint16_t port_state;
port_state = gpio_input_port_get(GPIOA);
```

### 函数 gpio\_output\_bit\_get

函数gpio\_output\_bit\_get描述见下表:

表 3-286. 函数 gpio\_output\_bit\_get

函数名称	gpio_output_bit_get
函数原型	FlagStatus gpio_output_bit_get(uint32_t gpio_periph,uint32_t pin);
功能描述	获取引脚的输出值
先决条件	-
被调用函数	-
输入参数{in}	
<b>gpio_periph</b>	GPIO端口
<i>GPIOx</i>	端口选择(x = A,B,C)
输入参数{in}	
<b>pin</b>	GPIO引脚
<i>GPIO_PIN_x</i>	引脚选择 (x=0..15)
<i>GPIO_PIN_ALL</i>	所有引脚
输出参数{out}	
-	-
返回值	
<b>FlagStatus</b>	SET或RESET

例如:

```
/* get output value of Port A */
FlagStatus bit_state;
bit_state = gpio_output_bit_get(GPIOA);
```

### 函数 gpio\_output\_port\_get

函数gpio\_output\_port\_get描述见下表:

表 3-287. 函数 `gpio_output_port_get`

函数名称	<code>gpio_output_port_get</code>
函数原型	<code>uint16_t gpio_output_port_get(uint32_t gpio_periph);</code>
功能描述	获取引脚的输出值
先决条件	-
被调用函数	-
输入参数{in}	
<code>gpio_periph</code>	GPIO端口
<code>GPIOx</code>	端口选择(x = A,B,C)
输出参数{out}	
-	-
返回值	
<code>uint16_t</code>	0x0000-0xFFFF

例如:

```
/*set PA0 alternate function 0 */
```

```
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 3-288. 函数 `gpio_af_set`

函数名称	<code>gpio_af_set</code>
函数原型	<code>void gpio_af_set(uint32_t gpio_periph, uint32_t alt_func_num, uint32_t pin);</code>
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}	
<code>gpio_periph</code>	GPIO端口
<code>GPIOx</code>	GPIOx(x = A,B,C)
输入参数{in}	
<code>alt_func_num</code>	GPIO引脚备用功能, 请参见特定设备的数据手册
<code>GPIO_AF_0</code>	USART0, USART1, TIMER0, SPI0, SPI1, CK_OUT0, RTC_REF, IR_OUT
<code>GPIO_AF_1</code>	USART1, TIMER0, TIMER1, TIMER2, I2S1
<code>GPIO_AF_2</code>	USART0, TIMER0, TIMER2, TIMER3, TIMER4, SPI0, I2S1, SDIO
<code>GPIO_AF_3</code>	QSPI, USART1, TIMER0, TIMER2, TIMER3, SQPI, TSI
<code>GPIO_AF_4</code>	TSI, TIMER4, SPI0, I2C0, QSPI, I2C1, I2S1
<code>GPIO_AF_5</code>	SPI0, SPI1, I2C0, I2S1
<code>GPIO_AF_6</code>	SPI0, SPI1, I2S1, I2C1
<code>GPIO_AF_7</code>	USART0, USART1, USART2, TIMER4, TIMER16, SPI0, SPI1, DCI

	(DCI在GD32W515Tx系列设备上不支持)
<i>GPIO_AF_8</i>	USART2, SQPI, TIMER0, TIMER15
<i>GPIO_AF_9</i>	RTC, TIMER1, IR_OUT, I2C1
<i>GPIO_AF_10</i>	USART2, TIMER16, USBFS
<i>GPIO_AF_11</i>	TIMER15
<i>GPIO_AF_12</i>	SDIO, DCI (DCI在GD32W515Tx系列设备上不支持)
<i>GPIO_AF_13</i>	DCI (DCI在GD32W515Tx系列设备上不支持)
<i>GPIO_AF_14</i>	HPDF,DCI (HPDF和DCI在GD32W515Tx系列设备上不支持)
<i>GPIO_AF_15</i>	EVENTOUT
输入参数{in}	
<b>pin</b>	GPIO引脚
<i>GPIO_PIN_x</i>	引脚选择 (x=0..15)
<i>GPIO_PIN_ALL</i>	所有引脚
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/*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 3-289. 函数 **gpio\_pin\_lock****

函数名称	gpio_pin_lock
函数原型	void gpio_pin_lock(uint32_t gpio_periph,uint32_t pin);
功能描述	相应的引脚配置被锁定
先决条件	-
被调用函数	-
输入参数{in}	
<b>gpio_periph</b>	GPIO端口
<i>GPIOx</i>	端口选择(x = A,B,C)
输入参数{in}	
<b>pin</b>	GPIO引脚
<i>GPIO_PIN_x</i>	引脚选择 (x=0..15)
<i>GPIO_PIN_ALL</i>	所有引脚
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* 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 3-290. 函数 gpio\_bit\_toggle

函数名称	gpio_bit_toggle
函数原型	void gpio_bit_toggle(uint32_t gpio_periph, uint32_t pin);
功能描述	翻转GPIO引脚状态
先决条件	-
被调用函数	-
输入参数{in}	
gpio_periph	GPIO端口
GPIOx	GPIOx(x = A,B,C)
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：

```
/* 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 3-291. 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	GPIOx(x = A,B,C)

输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* toggle GPIOA */
```

```
gpio_port_toggle(GPIOA);
```

### 函数 gpio\_bit\_set\_sec\_cfg

函数gpio\_bit\_set\_sec\_cfg描述见下表:

表 3-292. 函数 gpio\_bit\_set\_sec\_cfg

函数名称	gpio_bit_set_sec_cfg
函数原型	void gpio_bit_set_sec_cfg(uint32_t gpio_periph, uint32_t pin);
功能描述	配置引脚安全配置位状态为置位
先决条件	-
被调用函数	-
输入参数{in}	
gpio_periph	GPIO端口
GPIOx	GPIOx(x = A,B,C)
输入参数{in}	
pin	GPIO pin
GPIO_PIN_x	GPIO_PIN_x(x=0..15)
GPIO_PIN_ALL	GPIO_PIN_ALL
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* configure GPIOA pin 0 secure configuration bit status to set */
```

```
gpio_bit_set_sec_cfg (GPIOA, GPIO_PIN_0);
```

### 函数 gpio\_bit\_reset\_sec\_cfg

函数gpio\_bit\_reset\_sec\_cfg描述见下表:

表 3-293. 函数 gpio\_bit\_reset\_sec\_cfg

函数名称	gpio_bit_reset_sec_cfg
函数原型	void gpio_bit_reset_sec_cfg(uint32_t gpio_periph, uint32_t pin);
功能描述	配置引脚安全配置位状态为复位
先决条件	-

被调用函数	-
输入参数{in}	
<b>gpio_periph</b>	GPIO端口
<i>GPIOx</i>	GPIOx(x = A,B,C)
输入参数{in}	
<b>pin</b>	GPIO引脚
<i>GPIO_PIN_x</i>	引脚选择 (x=0..15)
<i>GPIO_PIN_ALL</i>	所有引脚
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* configure GPIOA pin 0 secure configuration bit status to reset */
```

```
gpio_bit_reset_sec_cfg(GPIOA, GPIO_PIN_0);
```

### 函数 **gpio\_sec\_cfg\_bit\_get**

函数gpio\_sec\_cfg\_bit\_get描述见下表:

表 3-294. 函数 **gpio\_sec\_cfg\_bit\_get**

函数名称	gpio_sec_cfg_bit_get
函数原型	FlagStatus gpio_sec_cfg_bit_get(uint32_t gpio_periph, uint32_t pin);
功能描述	获取引脚安全配置位状态
先决条件	-
被调用函数	-
输入参数{in}	
<b>gpio_periph</b>	GPIO端口
<i>GPIOx</i>	GPIOx(x = A,B,C)
输入参数{in}	
<b>pin</b>	GPIO引脚
<i>GPIO_PIN_x</i>	引脚选择 (x=0..15)
<i>GPIO_PIN_ALL</i>	所有引脚
输出参数{out}	
-	-
返回值	
<b>FlagStatus</b>	SET或RESET

例如:

```
/* get GPIOA pin0 secure configuration bit status */
```

```
FlagStatus sec_cfg_bit_state = gpio_sec_cfg_bit_get(GPIOA, GPIO_PIN_0);
```

### 3.13. HAU

哈希处理器应用于信息安全。支持应用于多种场合的安全哈希算法（SHA-1，SHA-224和SHA-256），消息摘要算法（MD5）和哈希运算消息认证码（HMAC）。HAU寄存器列举在章节[3.13.1](#)，HAU固件库函数介绍在章节[3.13.2](#)。

#### 3.13.1. 外设寄存器说明

HAU寄存器列表如下表所示：

**表 3-295. HAU 寄存器**

寄存器名称	寄存器描述
HAU_CTL	控制寄存器
HAU_DI	数据输入寄存器
HAU_CFG	配置寄存器
HAU_DO0	数据输出寄存器0
HAU_DO1	数据输出寄存器1
HAU_DO2	数据输出寄存器2
HAU_DO3	数据输出寄存器3
HAU_DO4	数据输出寄存器4
HAU_DO5	数据输出寄存器5
HAU_DO6	数据输出寄存器6
HAU_DO7	数据输出寄存器7
HAU_INTEN	中断使能寄存器
HAU_STAT	状态寄存器
HAU_CTXSx (x = 0..53)	上下文交换寄存器

#### 3.13.2. 外设库函数说明

HAU库函数列表如下表所示：

**表 3-296. HAU 库函数**

库函数名称	库函数描述
hau_deinit	复位HAU外设
hau_init	初始化HAU外设参数
hau_init_struct_para_init	初始化结构体hau_initpara
hau_reset	复位HAU内核
hau_last_word_validbits_num_config	配置消息最新字有效位数
hau_data_write	写数据到IN FIFO
hau_infifo_words_num_get	返回已经写入IN FIFO的字数目
hau_digest_read	读消息摘要结果
hau_digest_calculation_enable	使能摘要计算



库函数名称	库函数描述
hau_multiple_single_dma_config	配置configure multiple or single DMA is used, and digest calculation at the end of a DMA transfer or not
hau_dma_enable	使能HAU DMA接口
hau_dma_disable	除能HAU DMA接口
hau_context_struct_para_init	初始化上下文结构体
hau_context_save	保存HAU外设上下文
hau_context_restore	恢复HAU外设上下文
hau_hash_sha_1	在HASH模式下使用SHA1计算摘要
hau_hmac_sha_1	在HMAC模式下使用SHA1计算摘要
hau_hash_sha_224	在HASH模式下使用SHA224计算摘要
hau_hmac_sha_224	在HMAC模式下使用SHA224计算摘要
hau_hash_sha_256	在HASH模式下使用SHA256计算摘要
hau_hmac_sha_256	在HMAC模式下使用SHA256计算摘要
hau_hash_md5	在HASH模式下使用MD5计算摘要
hau_hmac_md5	在HMAC模式下使用MD5计算摘要
hau_flag_get	获取HAU标志状态
hau_flag_clear	清除HAU标志状态
hau_interrupt_enable	使能HAU中断
hau_interrupt_disable	除能HAU中断
hau_interrupt_flag_get	获取HAU中断标志状态
hau_interrupt_flag_clear	清除HAU中断标志状态

### 结构体 hau\_init\_parameter\_struct

表 3-297. 结构体 hau\_init\_parameter\_struct

成员名称	功能描述
algo	算法选择: HAU_ALGO_SHA1, HAU_ALGO_SHA224, HAU_ALGO_SHA256, HAU_ALGO_MD5
mode	HAU模式选择: HAU_MODE_HASH, HAU_MODE_HMAC
datatype	数据类型模式: HAU_SWAPPING_32BIT, HAU_SWAPPING_16BIT, HAU_SWAPPING_8BIT, HAU_SWAPPING_1BIT
keytype	密钥长度模式: HAU_KEY_SHORTER_64, HAU_KEY_LONGGER_64

### 结构体 hau\_digest\_parameter\_struct

表 3-298. 结构体 hau\_digest\_parameter\_struct

成员名称	功能描述
out[8]	消息摘要结果0-7

## 结构体 hau\_context\_parameter\_struct

表 3-299. 结构体 hau\_context\_parameter\_struct

成员名称	功能描述
hau_ctl_bak	HAU_CTL寄存器的备份
hau_cfg_bak	HAU_CFG寄存器的备份
hau_inten_bak	HAU_INTEN寄存器的备份
hau_ctxs_bak[54]	HAU_CTXSx寄存器的备份

## 函数 hau\_deinit

函数hau\_deinit描述见下表:

表 3-300. 函数 hau\_deinit

函数名称	hau_deinit
函数原形	void hau_deinit(void);
功能描述	复位HAU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 the HAU peripheral */
```

```
hau_deinit();
```

## 函数 hau\_init

函数hau\_init描述见下表:

表 3-301. 函数 hau\_init

函数名称	hau_init
函数原形	void hau_init(hau_init_parameter_struct* initpara);
功能描述	初始化HAU外设参数
先决条件	-
被调用函数	-
输入参数{in}	
initpara	HAU外设参数, 参考结构体 <a href="#">表3-297. 结构体hau_init_parameter_struct</a>
输出参数{out}	
-	-

返回值	
-	-

例如：

```
/* initialize the HAU peripheral parameters */

hau_init_parameter_struct hau_initpara;

...

hau_init_struct_para_init(&hau_initpara);

hau_initpara.algo = algo;

hau_initpara.mode = HAU_MODE_HMAC;

hau_initpara.datatype = HAU_SWAPPING_8BIT;

if(key_len > 64U){

    hau_initpara.keytype = HAU_KEY_LONGGER_64;

}else{

    hau_initpara.keytype = HAU_KEY_SHORTER_64;

}

hau_init(&hau_initpara);
```

### 函数 hau\_init\_struct\_para\_init

函数hau\_init\_struct\_para\_init描述见下表：

**表 3-302. 函数 hau\_init\_struct\_para\_init**

函数名称	hau_init_struct_para_init
函数原形	void hau_init_struct_para_init(hau_init_parameter_struct* initpara)
功能描述	初始化结构体hau_initpara
先决条件	-
被调用函数	-
输入参数{in}	
-	-
输出参数{out}	
initpara	HAU外设参数，参考结构体 <a href="#">表3-297. 结构体hau_init_parameter_struct</a>
返回值	
-	-

例如：

```
/* initialize the HAU peripheral parameters */

hau_init_parameter_struct hau_initpara;
```

```
hau_init_struct_para_init(&hau_initpara);
```

### 函数 hau\_reset

函数hau\_reset描述见下表：

**表 3-303. 函数 hau\_reset**

函数名称	hau_reset
函数原形	void hau_reset(void);
功能描述	复位HAU内核
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 HAU processor core */
hau_reset();
```

### 函数 hau\_last\_word\_validbits\_num\_config

函数hau\_last\_word\_validbits\_num\_config描述见下表：

**表 3-304. 函数 hau\_last\_word\_validbits\_num\_config**

函数名称	hau_last_word_validbits_num_config
函数原形	void hau_last_word_validbits_num_config(uint32_t valid_num);
功能描述	配置消息最新字有效位数
先决条件	-
被调用函数	-
输入参数{in}	
valid_num	消息最新字有效位数(0x00 – 0x1F)
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* configure the number of valid bits in last word of the message */
hau_last_word_validbits_num_config(0x10);
```

## 函数 hau\_data\_write

函数hau\_data\_write描述见下表：

**表 3-305. 函数 hau\_data\_write**

函数名称	hau_data_write
函数原形	void hau_data_write(uint32_t data);
功能描述	写数据到IN FIFO
先决条件	-
被调用函数	-
输入参数{in}	
data	所写的数据(0x0 – 0xFFFFFFFF)
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* write data to the IN FIFO */
```

```
hau_data_write(0x10);
```

## 函数 hau\_infifo\_words\_num\_get

函数hau\_infifo\_words\_num\_get描述见下表：

**表 3-306. 函数 hau\_infifo\_words\_num\_get**

函数名称	hau_infifo_words_num_get
函数原形	uint32_t hau_infifo_words_num_get(void);
功能描述	返回已经写入IN FIFO的字数目
先决条件	-
被调用函数	-
输入参数{in}	
-	-
输出参数{out}	
-	-
返回值	
uint32_t	0x0 – 0xFFFFFFFF

例如：

```
/* return the number of words already written into the IN FIFO */
```

```
uint32_t num;
```

```
num = hau_infifo_words_num_get();
```

**函数 hau\_digest\_read**

函数hau\_digest\_read描述见下表：

**表 3-307. 函数 hau\_digest\_read**

函数名称	hau_digest_read
函数原形	void hau_digest_read(hau_digest_parameter_struct* digestpara);
功能描述	读消息摘要结果
先决条件	-
被调用函数	-
输入参数{in}	
-	-
输出参数{out}	
digestpara	消息摘要结果，参考结构体 <a href="#">表3-298. 结构体hau_digest_parameter_struct</a>
返回值	
-	-

例如：

```
/* read the message digest result */
hau_digest_parameter_struct digestpara;
hau_digest_read(&digestpara);
```

**函数 hau\_digest\_calculation\_enable**

函数hau\_digest\_calculation\_enable描述见下表：

**表 3-308. 函数 hau\_digest\_calculation\_enable**

函数名称	hau_digest_calculation_enable
函数原形	void hau_digest_calculation_enable(void);
功能描述	使能摘要计算
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 digest calculation */
hau_digest_calculation_enable();
```

**函数 hau\_multiple\_single\_dma\_config**

函数hau\_multiple\_single\_dma\_config描述见下表：

**表 3-309. 函数 hau\_multiple\_single\_dma\_config**

函数名称	hau_multiple_single_dma_config
函数原形	void hau_multiple_single_dma_config(uint32_t multi_single);
功能描述	配置使用多DMA或单DMA，从未确定是否在DMA传输结束后计算摘要
先决条件	-
被调用函数	-
输入参数{in}	
multi_single	Multiple or single
SINGLE_DMA_AUTO_DIGEST	在DMA传输完成后消息填充和计算摘要
MULTIPLE_DMA_NO_DIGEST	需要多次DMA传输，在DMA传输结束时硬件不自动将CALEN位置1
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* configure multiple or single DMA is used, and digest calculation at the end of a DMA transfer or not */
```

```
hau_multiple_single_dma_config(SINGLE_DMA_AUTO_DIGEST);
```

**函数 hau\_dma\_enable**

函数hau\_dma\_enable描述见下表：

**表 3-310. 函数 hau\_dma\_enable**

函数名称	hau_dma_enable
函数原形	void hau_dma_enable(void);
功能描述	使能HAU DMA接口
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 HAU DMA interface */
```

```
hau_dma_enable();
```

### 函数 `hau_dma_disable`

函数 `hau_dma_disable` 描述见下表：

**表 3-311. 函数 `hau_dma_disable`**

函数名称	<code>hau_dma_disable</code>
函数原形	<code>void hau_dma_disable(void);</code>
功能描述	除能HAU DMA接口
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 HAU DMA interface */
```

```
hau_dma_disable();
```

### 函数 `hau_context_struct_para_init`

函数 `hau_context_struct_para_init` 描述见下表：

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

函数名称	<code>hau_context_struct_para_init</code>
函数原形	<code>void hau_context_struct_para_init(hau_context_parameter_struct* context)</code>
功能描述	初始化上下文结构体
先决条件	-
被调用函数	-
输入参数{in}	
<b>context</b>	上下文结构体，参考结构体 <a href="#">表3-299. 结构体 <code>hau_context_parameter_struct</code></a>
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* initialize the struct context */
```



```
hau_context_parameter_struct context_para;
```

```
hau_context_struct_para_init(&context_para);
```

### 函数 `hau_context_save`

函数 `hau_context_save` 描述见下表：

**表 3-313. 函数 `hau_context_save`**

函数名称	<code>hau_context_save</code>
函数原形	<code>void hau_context_save(hau_context_parameter_struct* context_save)</code>
功能描述	保存HAU外设上下文
先决条件	-
被调用函数	-
输入参数{in}	
-	-
输出参数{out}	
<b>context</b>	上下文结构体，参考结构体 <a href="#">表3-299. 结构体 <code>hau_context_parameter_struct</code></a>
返回值	
-	-

例如：

```
hau_context_parameter_struct context_para;
```

```
hau_context_struct_para_init(&context_para);
```

```
/* save HAU context */
```

```
hau_context_save(&context_para);
```

### 函数 `hau_context_restore`

函数 `hau_context_restore` 描述见下表：

**表 3-314. 函数 `hau_context_restore`**

函数名称	<code>hau_context_restore</code>
函数原形	<code>void hau_context_restore(hau_context_parameter_struct* context_restore)</code>
功能描述	恢复HAU外设上下文
先决条件	-
被调用函数	-
输入参数{in}	
<b>context</b>	上下文结构体，参考结构体 <a href="#">表3-299. 结构体 <code>hau_context_parameter_struct</code></a>
输出参数{out}	
-	-
返回值	

-	-
---	---

例如：

```
hau_context_parameter_struct context_para;
```

```
hau_context_struct_para_init(&context_para);
```

```
hau_context_save(&context_para);
```

```
.....
```

```
/* restore HAU context */
```

```
hau_context_restore(&context_para);
```

### 函数 hau\_hash\_sha\_1

函数hau\_hash\_sha\_1描述见下表：

**表 3-315. 函数 hau\_hash\_sha\_1**

函数名称	hau_hash_sha_1
函数原形	ErrStatus hau_hash_sha_1(uint8_t *input, uint32_t in_length, uint8_t output[20]);
功能描述	在HASH模式下使用SHA1计算摘要
先决条件	-
被调用函数	-
输入参数{in}	
input	指向输入缓存区
输入参数{in}	
in_length	输入缓存区长度
输出参数{out}	
output	摘要结果
返回值	
ErrStatus	SUCCESS或ERROR

例如：

```
/* calculate digest using SHA1 in HASH mode */
```

```
ErrStatus status;
```

```
status = hau_hash_sha_1(&input, 0x10, output[0]);
```

### 函数 hau\_hmac\_sha\_1

函数hau\_hmac\_sha\_1描述见下表：

**表 3-316. 函数 hau\_hmac\_sha\_1**

函数名称	hau_hmac_sha_1
------	----------------

函数原形	ErrStatus hau_hmac_sha_1(uint8_t *key, uint32_t keysize, uint8_t *input, uint32_t in_length, uint8_t output[20]);
功能描述	在HMAC模式下使用SHA1计算摘要
先决条件	-
被调用函数	-
输入参数{in}	
key	指向用于HMAC模式的密钥
输入参数{in}	
keysize	用于HMAC模式的密钥长度
输入参数{in}	
input	指向输入缓存区
输入参数{in}	
in_length	输入缓存区长度
输出参数{out}	
output	摘要结果
返回值	
ErrStatus	SUCCESS或ERROR

例如:

```
/* calculate digest using SHA1 in HMAC mode */
```

```
ErrStatus status;
```

```
status = hau_hmac_sha_1(&key, 0x10, &input, 0x10, output[0]);
```

### 函数 hau\_hash\_sha\_224

函数hau\_hash\_sha\_224描述见下表:

表 3-317. 函数 hau\_hash\_sha\_224

函数名称	hau_hash_sha_224
函数原形	ErrStatus hau_hash_sha_224(uint8_t *input, uint32_t in_length, uint8_t output[28]);
功能描述	在HASH模式下使用SHA224计算摘要
先决条件	-
被调用函数	-
输入参数{in}	
input	指向输入缓存区
输入参数{in}	
in_length	输入缓存区长度
输出参数{out}	
output	摘要结果
返回值	
ErrStatus	SUCCESS或ERROR

例如：

```
/* calculate digest using SHA224 in HASH mode */

ErrStatus status;

status = hau_hash_sha_224 (&input, 0x10, output[0]);
```

### 函数 hau\_hmac\_sha\_224

函数hau\_hmac\_sha\_224描述见下表：

表 3-318. 函数 hau\_hmac\_sha\_224

函数名称	hau_hmac_sha_224
函数原形	ErrStatus hau_hmac_sha_224(uint8_t *key, uint32_t keysize, uint8_t *input, uint32_t in_length, uint8_t output[28]);
功能描述	在HMAC模式下使用SHA224计算摘要
先决条件	-
被调用函数	-
输入参数{in}	
key	指向用于HMAC模式的密钥
输入参数{in}	
keysize	用于HMAC模式的密钥长度
输入参数{in}	
input	指向输入缓存区
输入参数{in}	
in_length	输入缓存区长度
输出参数{out}	
output	摘要结果
返回值	
ErrStatus	SUCCESS或ERROR

例如：

```
/* calculate digest using SHA224 in HMAC mode */

ErrStatus status;

status = hau_hmac_sha_224 (&key, 0x10, &input, 0x10, output[0]);
```

### 函数 hau\_hash\_sha\_256

函数hau\_hash\_sha\_256描述见下表：

表 3-319. 函数 hau\_hash\_sha\_256

函数名称	hau_hash_sha_256
函数原形	ErrStatus hau_hash_sha_256(uint8_t *input, uint32_t in_length, uint8_t output[32]);

功能描述	在HASH模式下使用SHA256计算摘要
先决条件	-
被调用函数	-
输入参数{in}	
input	指向输入缓存区
输入参数{in}	
in_length	输入缓存区长度
输出参数{out}	
output	摘要结果
返回值	
ErrStatus	SUCCESS或ERROR

例如：

```
/* calculate digest using SHA256 in HASH mode */
```

```
ErrStatus status;
```

```
status = hau_hash_sha_256 (&input, 0x10, output[0]);
```

### 函数 hau\_hmac\_sha\_256

函数hau\_hmac\_sha\_256描述见下表：

表 3-320. 函数 hau\_hmac\_sha\_256

函数名称	hau_hmac_sha_256
函数原形	ErrStatus hau_hmac_sha_256(uint8_t *key, uint32_t keysize, uint8_t *input, uint32_t in_length, uint8_t output[32]);
功能描述	在HMAC模式下使用SHA256计算摘要
先决条件	-
被调用函数	-
输入参数{in}	
key	指向用于HMAC模式的密钥
输入参数{in}	
keysize	用于HMAC模式的密钥长度
输入参数{in}	
input	指向输入缓存区
输入参数{in}	
in_length	输入缓存区长度
输出参数{out}	
output	摘要结果
返回值	
ErrStatus	SUCCESS或ERROR

例如：

```
/* calculate digest using SHA256 in HMAC mode */
```

```
ErrStatus status;
```

```
status = hau_hmac_sha_256 (&key, 0x10, &input, 0x10, output[0]);
```

### 函数 hau\_hash\_md5

函数hau\_hash\_md5描述见下表:

表 3-321. 函数 hau\_hash\_md5

函数名称	hau_hash_md5
函数原形	ErrStatus hau_hash_md5(uint8_t *input, uint32_t in_length, uint8_t output[16]);
功能描述	在HASH模式下使用MD5计算摘要
先决条件	-
被调用函数	-
输入参数{in}	
input	指向输入缓存区
输入参数{in}	
in_length	输入缓存区长度
输出参数{out}	
output	摘要结果
返回值	
ErrStatus	SUCCESS或ERROR

例如:

```
/* calculate digest using MD5 in HASH mode */
```

```
ErrStatus status;
```

```
status = hau_hash_md5 (&input, 0x10, output[0]);
```

### 函数 hau\_hmac\_md5

函数hau\_hmac\_md5描述见下表:

表 3-322. 函数 hau\_hmac\_md5

函数名称	hau_hmac_md5
函数原形	ErrStatus hau_hmac_md5(uint8_t *key, uint32_t keysize, uint8_t *input, uint32_t in_length, uint8_t output[16]);
功能描述	在HMAC模式下使用MD5计算摘要
先决条件	-
被调用函数	-
输入参数{in}	
key	指向用于HMAC模式的密钥
输入参数{in}	
keysize	用于HMAC模式的密钥长度

输入参数{in}	
<b>input</b>	指向输入缓存区
输入参数{in}	
<b>in_length</b>	输入缓存区长度
输出参数{out}	
<b>output</b>	摘要结果
返回值	
<b>ErrStatus</b>	SUCCESS或ERROR

例如:

```
/* calculate digest using MD5 in HMAC mode */
```

```
ErrStatus status;
```

```
status = hau_hmac_md5 (&key, 0x10, &input, 0x10, output[0]);
```

### 函数 hau\_flag\_get

函数hau\_flag\_get描述见下表:

表 3-323. 函数 hau\_flag\_get

函数名称	hau_flag_get
函数原形	FlagStatus hau_flag_get(uint32_t flag);
功能描述	获取HAU标志状态
先决条件	-
被调用函数	-
输入参数{in}	
<b>flag</b>	HAU标志状态
<i>HAU_FLAG_DATA_INPUT</i>	输入FIFO有足够空间
<i>HAU_FLAG_CALCULATION_COMPLETE</i>	摘要计算完整
<i>HAU_FLAG_DMA</i>	DMA被使能或传输正在处理
<i>HAU_FLAG_BUSY</i>	数据块正在处理
<i>HAU_FLAG_INFIFO_NO_EMPTY</i>	输入FIFO非空
输出参数{out}	
-	-
返回值	
<b>FlagStatus</b>	SET或RESET

例如:

```
/* get the HAU flag status */
```

```
FlagStatus status;
```

```
status = hau_flag_get(HAU_FLAG_DMA);
```

### 函数 `hau_flag_clear`

函数 `hau_flag_clear` 描述见下表：

表 3-324. 函数 `hau_flag_clear`

函数名称	<code>hau_flag_clear</code>
函数原形	<code>void hau_flag_clear(uint32_t flag);</code>
功能描述	清除HAU标志状态
先决条件	-
被调用函数	-
输入参数{in}	
<b>flag</b>	HAU标志状态
<code>HAU_FLAG_DATA_INPUT</code>	输入FIFO有足够空间
<code>HAU_FLAG_CALCULATION_COMPLETE</code>	摘要计算完整
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* clear the HAU flag status */  
  
hau_flag_clear(HAU_FLAG_DATA_INPUT);
```

### 函数 `hau_interrupt_enable`

函数 `hau_interrupt_enable` 描述见下表：

表 3-325. 函数 `hau_interrupt_enable`

函数名称	<code>hau_interrupt_enable</code>
函数原形	<code>void hau_interrupt_enable(uint32_t interrupt);</code>
功能描述	使能HAU中断
先决条件	-
被调用函数	-
输入参数{in}	
<b>interrupt</b>	HAU标志状态
<code>HAU_INT_DATA_INPUT</code>	新数据块进入IN缓存区
<code>HAU_INT_CALCULATION_COMPLETE</code>	计算完成
输出参数{out}	



-	-
返回值	
-	-

例如：

```
/* enable hau interrupt */
```

```
hau_interrupt_enable(HAU_INT_DATA_INPUT);
```

### 函数 `hau_interrupt_disable`

函数 `hau_interrupt_disable` 描述见下表：

**表 3-326. 函数 `hau_interrupt_disable`**

函数名称	<code>hau_interrupt_disable</code>
函数原形	<code>void hau_interrupt_disable(uint32_t interrupt);</code>
功能描述	除能HAU中断
先决条件	-
被调用函数	-
输入参数{in}	
<b>interrupt</b>	HAU标志状态
<code>HAU_INT_DATA_INPUT</code>	新数据块进入IN缓存区
<code>HAU_INT_CALCULATION_COMPLETE</code>	计算完成
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* disable hau interrupt */
```

```
hau_interrupt_disable(HAU_INT_DATA_INPUT);
```

### 函数 `hau_interrupt_flag_get`

函数 `hau_interrupt_flag_get` 描述见下表：

**表 3-327. 函数 `hau_interrupt_flag_get`**

函数名称	<code>hau_interrupt_flag_get</code>
函数原形	<code>FlagStatus hau_interrupt_flag_get(uint32_t int_flag)</code>
功能描述	获取HAU中断标志状态
先决条件	-
被调用函数	-
输入参数{in}	

<b>int_flag</b>	HAU标志状态
<i>HAU_INT_FLAG_DATA_INPUT</i>	输入FIFO有足够空间
<i>HAU_INT_FLAG_CALCULATION_COMPLETE</i>	摘要计算完整
<b>输出参数{out}</b>	
-	-
<b>返回值</b>	
<b>FlagStatus</b>	SET或RESET

例如：

```
/* get the HAU interrupt flag status */
```

```
FlagStatus status;
```

```
status = hau_interrupt_flag_get (HAU_INT_FLAG_DATA_INPUT);
```

## 函数 **hau\_interrupt\_flag\_clear**

函数hau\_interrupt\_flag\_clear描述见下表：

**表 3-328. 函数 **hau\_interrupt\_flag\_clear****

<b>函数名称</b>	hau_interrupt_flag_clear
<b>函数原形</b>	void hau_interrupt_flag_clear(uint32_t int_flag)
<b>功能描述</b>	清除HAU中断标志状态
<b>先决条件</b>	-
<b>被调用函数</b>	-
<b>输入参数{in}</b>	
<b>int_flag</b>	HAU标志状态
<i>HAU_INT_FLAG_DATA_INPUT</i>	输入FIFO有足够空间
<i>HAU_INT_FLAG_CALCULATION_COMPLETE</i>	摘要计算完整
<b>输出参数{out}</b>	
-	-
<b>返回值</b>	
-	-

例如：

```
/* clear the HAU interrupt flag status */
```

```
hau_interrupt_flag_clear(HAU_INT_FLAG_DATA_INPUT);
```

### 3.14. HPDF

GD32W51x 内部集成了一种专门用于外部  $\Sigma-\Delta$  调制器的高性能数字滤波器模块(HPDF)。章节 [3.14.1](#) 描述了 HPDF 的寄存器列表，章节 [3.14.2](#) 对 HPDF 库函数进行说明。

#### 3.14.1. 外设寄存器说明

HPDF 寄存器列表如下表所示：

表 3-329. HPDF 寄存器

寄存器名称	寄存器描述
HPDF_CHxCTL	通道x控制寄存器
HPDF_CHxCFG0	通道x配置寄存器
HPDF_CHxCFG1	通道x配置寄存器1
HPDF_CHxTMFDT	通道x阈值监视器滤波器数据寄存器
HPDF_CHxPDI	通道x并行数据输入寄存器
HPDF_CHxPS	通道x跳频寄存器
HPDF_FLTyCTL0	滤波器y控制寄存器0
HPDF_FLTyCTL1	滤波器y控制寄存器1
HPDF_FLTySTAT	滤波器y状态寄存器
HPDF_FLTyINTC	滤波器y中断标志清除寄存器
HPDF_FLTyIGCS	滤波器y注入组通道选择寄存器
HPDF_FLTySFCFG	滤波器y SINC滤波器配置寄存器
HPDF_FLTyIDATA	滤波器y注入组转换数据寄存器
HPDF_FLTyRDATA	滤波器y规则通道转换数据寄存器
HPDF_FLTyTMHT	滤波器y阈值监视器上限阈值寄存器
HPDF_FLTyTMLT	滤波器y阈值监视器下限阈值寄存器
HPDF_FLTyTMSTAT	滤波器y阈值监视器状态寄存器
HPDF_FLTyTMFC	滤波器y阈值监视器标志清除寄存器
HPDF_FLTyEMMAX	滤波器y极值监视器最大值寄存器
HPDF_FLTyEMMIN	滤波器y极值监视器最小值寄存器

#### 3.14.2. 外设库函数说明

HPDF库函数列表如下表所示：

表 3-330. HPDF 库函数

库函数名称	库函数描述
hpdf_deinit	复位HPDF外设
hpdf_channel_struct_para_init	初始化HPDF通道结构体参数
hpdf_filter_struct_para_init	初始化HPDF滤波器结构体参数
hpdf_rc_struct_para_init	初始化规则转换结构体参数
hpdf_ic_struct_para_init	初始化注入转换结构体参数

库函数名称	库函数描述
hpdf_enable	使能HPDF模块
hpdf_disable	禁止HPDF模块
hpdf_channel_init	初始化HPDF通道
hpdf_filter_init	初始化HPDF滤波器
hpdf_rc_init	初始化规则转换
hpdf_ic_init	初始化注入转换
hpdf_clock_output_config	配置串行输出时钟
hpdf_clock_output_source_config	配置串行输出时钟源
hpdf_clock_output_duty_mode_disable	禁止串行输出时钟占空比模式
hpdf_clock_output_duty_mode_enable	使能串行输出时钟占空比模式
hpdf_clock_output_divider_config	配置串行输出时钟分频
hpdf_channel_enable	使能HPDF通道
hpdf_channel_disable	禁止HPDF通道
hpdf_spi_clock_source_config	配置SPI接口时钟源
hpdf_serial_interface_type_config	配置串行接口类型
hpdf_malfunction_monitor_disable	禁止故障监视器
hpdf_malfunction_monitor_enable	使能故障监视器
hpdf_clock_loss_disable	禁止时钟丢失检测
hpdf_clock_loss_enable	使能时钟丢失检测
hpdf_channel_pin_redirection_disable	禁止通道输入引脚重定向
hpdf_channel_pin_redirection_enable	使能通道输入引脚重定向
hpdf_channel_multiplexer_config	配置复用通道输入数据源
hpdf_data_pack_mode_config	配置数据封装模式
hpdf_data_right_bit_shift_config	配置数据右移位数
hpdf_calibration_offset_config	配置数据校准偏移
hpdf_malfunction_break_signal_config	配置故障监视器断路信号
hpdf_malfunction_counter_config	配置故障监视器计数器阈值
hpdf_write_parallel_data_standard_mode	写入数据封装标准模式下的并行数据
hpdf_write_parallel_data_interleaved_mode	写入数据封装交错模式下的并行数据
hpdf_write_parallel_data_dual_mode	写入数据封装双通道模式下的并行数据
hpdf_pulse_skip_update	更新跳频脉冲数量
hpdf_pulse_skip_read	读取跳频脉冲数量
hpdf_filter_enable	使能滤波器
hpdf_filter_disable	禁止滤波器
hpdf_filter_config	配置滤波器阶数和过采样率
hpdf_integrator_oversample	配置积分器过采样率
hpdf_threshold_monitor_filter_config	配置阈值监视器的滤波器
hpdf_threshold_monitor_filter_read_data	读取阈值监视器滤波器的数据
hpdf_threshold_monitor_fast_mode_disable	禁止阈值监视器快速模式
hpdf_threshold_monitor_fast_mode_enable	使能阈值监视器快速模式
hpdf_threshold_monitor_channel	配置阈值监视器通道

库函数名称	库函数描述
hpdf_threshold_monitor_high_threshold	配置阈值监视器上限阈值
hpdf_threshold_monitor_low_threshold	配置阈值监视器下限阈值
hpdf_high_threshold_break_signal	配置阈值监视器上限阈值事件断路信号
hpdf_low_threshold_break_signal	配置阈值监视器下限阈值事件断路信号
hpdf_extremes_monitor_channel	配置极值监视器通道
hpdf_extremes_monitor_maximum_get	获取极值监视器最大极值
hpdf_extremes_monitor_minimum_get	获取极值监视器最小极值
hpdf_rc_continuous_disable	禁止规则转换连续模式
hpdf_rc_continuous_enable	使能规则转换连续模式
hpdf_rc_start_by_software	软件启动规则转换
hpdf_rc_syn_disable	禁止规则转换同步
hpdf_rc_syn_enable	使能规则转换同步
hpdf_rc_dma_disable	禁止规则转换DMA
hpdf_rc_dma_enable	使能规则转换DMA
hpdf_rc_channel_config	配置规则转换通道
hpdf_rc_fast_mode_disable	禁止规则转换快速模式
hpdf_rc_fast_mode_enable	使能规则转换快速模式
hpdf_rc_data_get	获取规则转换数据
hpdf_rc_channel_get	获取最近一次规则转换的通道
hpdf_ic_start_by_software	软件启动注入转换
hpdf_ic_syn_disable	禁止注入转换同步
hpdf_ic_syn_enable	使能注入转换同步
hpdf_ic_dma_disable	禁止注入转换DMA
hpdf_ic_dma_enable	使能注入转换DMA
hpdf_ic_scan_mode_disable	禁止注入转换扫描模式
hpdf_ic_scan_mode_enable	使能注入转换扫描模式
hpdf_ic_trigger_signal_disable	禁止注入转换触发信号
hpdf_ic_trigger_signal_config	配置注入转换触发信号和边沿
hpdf_ic_channel_config	配置注入组转换通道
hpdf_ic_data_get	获取注入转换数据
hpdf_ic_channel_get	获取最近一次注入转换的通道
hpdf_flag_get	获取HPDF标志位
hpdf_flag_clear	清除HPDF标志位
hpdf_interrupt_enable	使能HPDF中断
hpdf_interrupt_disable	禁止HPDF中断
hpdf_interrupt_flag_get	获取HPDF中断标志位
hpdf_interrupt_flag_clear	清除HPDF中断标志位

### 枚举类型 `hpdf_channel_enum`

表 3-331. 枚举类型 `hpdf_channel_enum`

成员名称	功能描述
CHANNEL0	HPDF通道0
CHANNEL1	HPDF通道1

### 枚举类型 `hpdf_filter_enum`

表3-332. 枚举类型`hpdf_filter_enum`

成员名称	功能描述
FLT0	HPDF滤波器0
FLT0	HPDF滤波器1

### 枚举类型 `hpdf_flag_enum`

表3-333. 枚举类型`hpdf_flag_enum`

成员名称	功能描述
HPDF_FLAG_FLTy_ICEF	注入转换结束标志
HPDF_FLAG_FLTy_RCEF	规则转换结束标志
HPDF_FLAG_FLTy_ICDOF	注入转换溢出标志
HPDF_FLAG_FLTy_RCDOF	规则转换溢出标志
HPDF_FLAG_FLTy_TMEOF	阈值监视器事件标志
HPDF_FLAG_FLTy_ICPF	注入转换正在进行标志
HPDF_FLAG_FLTy_RCPF	规则转换正在进行标志
HPDF_FLAG_FLT0_CKLF0	通道0时钟丢失标志
HPDF_FLAG_FLT0_CKLF1	通道1时钟丢失标志
HPDF_FLAG_FLT0_MMF0	通道0故障事件标志
HPDF_FLAG_FLT0_MMF1	通道1故障事件标志
HPDF_FLAG_FLTy_RCHPDT	规则通道等待处理数据
HPDF_FLAG_FLTy_LTF0	通道0阈值监视器下限阈值标志
HPDF_FLAG_FLTy_LTF1	通道1阈值监视器下限阈值标志
HPDF_FLAG_FLTy-HTF0	通道0阈值监视器上限阈值标志
HPDF_FLAG_FLTy-HTF1	通道1阈值监视器上限阈值标志

### 枚举类型 `hpdf_interrput_flag_enum`

表3-334. 枚举类型`hpdf_interrput_flag_enum`

成员名称	功能描述
HPDF_INT_FLAG_FLTy_ICEF	注入转换结束中断标志
HPDF_INT_FLAG_FLTy_RCEF	规则转换结束中断标志
HPDF_INT_FLAG_FLTy_ICDOF	注入转换溢出中断标志
HPDF_INT_FLAG_FLTy_RCDOF	规则转换溢出中断标志
HPDF_INT_FLAG_FLTy_TMEOF	阈值监视器事件中断标志

成员名称	功能描述
HPDF_INT_FLAG_FLT0_CKLF0	通道0时钟丢失中断标志
HPDF_INT_FLAG_FLT0_CKLF1	通道1时钟丢失中断标志
HPDF_INT_FLAG_FLT0_MMF0	通道0故障事件中断标志
HPDF_INT_FLAG_FLT0_MMF1	通道1故障事件中断标志

### 枚举类型 `hpdf_interrupt_enum`

表3-335. 枚举类型 `hpdf_interrupt_enum`

成员名称	功能描述
HPDF_INT_FLTy_ICEIE	使能注入转换结束中断
HPDF_INT_FLTy_RCEIE	使能规则转换结束中断
HPDF_INT_FLTy_ICDOIE	使能注入转换溢出中断
HPDF_INT_FLTy_RCDOIE	使能规则转换溢出中断
HPDF_INT_FLTy_TMIE	使能阈值监视器事件中断
HPDF_INT_FLT0_MMIE	使能故障监视中断
HPDF_INT_FLT0_CKLIE	使能时钟丢失中断

### 结构体 `hpdf_channel_parameter_struct`

表 3-336. 结构体 `hpdf_channel_parameter_struct`

成员名称	功能描述
<code>data_packing_mode</code>	并行数据寄存器的数据封装模式
<code>channel_muxlexer</code>	复用通道输入数据源
<code>channel_pin_select</code>	通道输入引脚选择
<code>ck_loss_detector</code>	时钟丢失检测
<code>malfunction_monitor</code>	故障监视器
<code>spi_ck_source</code>	SPI接口时钟源
<code>serial_interface</code>	串行接口类型
<code>calibration_offset</code>	24位校准偏移
<code>right_bit_shift</code>	右移位数
<code>tm_filter</code>	阈值监视器滤波器阶数选择
<code>tm_filter_oversample</code>	阈值监视器滤波器过采样率
<code>mm_break_signal</code>	分配故障监视器断路信号
<code>mm_counter_threshold</code>	故障监视器计数阈值
<code>plsk_value</code>	跳频脉冲数

### 结构体 `hpdf_filter_parameter_struct`

表 3-337. 结构体 `hpdf_filter_parameter_struct`

成员名称	功能描述
<code>tm_fast_mode</code>	阈值监视器快速模式
<code>tm_channel</code>	阈值监视器通道
<code>tm_high_threshold</code>	阈值监视器上限阈值

成员名称	功能描述
tm_low_threshold	阈值监视器下限阈值
extreme_monitor_channel	极值监视器通道
sinc_filter	Sinc滤波器阶数
sinc_oversample	Sinc滤波器过采样率
integrator_oversample	积分器过采样率
ht_break_signal	分配上限阈值断路信号
lt_break_signal	分配下限阈值断路信号

### 结构体 hpdf\_rc\_parameter\_struct

表 3-338. 结构体 hpdf\_rc\_parameter\_struct

成员名称	功能描述
fast_mode	规则转换快速转换模式
rsc_channel	规则转换通道
rscdmaen	使能读取规则转换数据的DMA通道
rscsyn	规则转换同步
continuous_mode	规则转换连续模式

### 结构体 hpdf\_ic\_parameter\_struct

表 3-339. 结构体 hpdf\_ic\_parameter\_struct

成员名称	功能描述
trigger_dege	注入转换触发边沿
trigger_signal	注入转换触发信号
icdmaen	使能读取注入转换数据的DMA通道
scmod	注入转换扫描模式
icsyn	注入转换同步
ic_channel_group	选择注入转换通道组

### 函数 hpdf\_deinit

函数hpdf\_deinit描述见下表：

表 3-340. 函数 hpdf\_deinit

函数名称	hpdf_deinit
函数原型	void hpdf_deinit(void);
功能描述	复位外设HPDF
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 HPDF */
```

```
hpdf_deinit();
```

### 函数 hpdf\_channel\_struct\_para\_init

函数hpdf\_channel\_struct\_para\_init描述见下表：

**表 3-341. 函数 hpdf\_channel\_struct\_para\_init**

函数名称	hpdf_channel_struct_para_init
函数原型	void hpdf_channel_struct_para_init(hpdf_channel_parameter_struct* init_struct);
功能描述	初始化HPDF通道结构体参数
先决条件	-
被调用函数	-
输入参数{in}	
*init_struct	HPDF通道初始化结构体，参考 <a href="#">表3-331. 枚举类型hpdf_channel_enum</a>
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* initialize the parameters of HPDF channel */
```

```
hpdf_channel_parameter_struct hpdf_channel_init_struct;
```

```
hpdf_channel_struct_para_init(&hpdf_channel_init_struct);
```

### 函数 hpdf\_filter\_struct\_para\_init

函数hpdf\_filter\_struct\_para\_init描述见下表：

**表 3-342. 函数 hpdf\_filter\_struct\_para\_init**

函数名称	hpdf_filter_struct_para_init
函数原型	void hpdf_filter_struct_para_init(hpdf_filter_parameter_struct* init_struct);
功能描述	初始化HPDF滤波器结构体参数
先决条件	-
被调用函数	-
输入参数{in}	
*init_struct	HPDF滤波器初始化结构体，参考 <a href="#">表3-337. 结构体hpdf_filter_parameter_struct</a>

输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* initialize the parameters of HPDF filter */

hpdf_filter_parameter_struct hpdf_filter_init_struct;

hpdf_filter_struct_para_init(&hpdf_filter_init_struct);
```

### 函数 hpdf\_rc\_struct\_para\_init

函数hpdf\_rc\_struct\_para\_init描述见下表：

表 3-343. 函数 hpdf\_rc\_struct\_para\_init

函数名称	hpdf_rc_struct_para_init
函数原型	void hpdf_rc_struct_para_init(hpdf_rc_parameter_struct* init_struct);
功能描述	初始化规则转换结构体参数
先决条件	-
被调用函数	-
输入参数{in}	
*init_struct	HPDF规则转换初始化结构体，参考 <a href="#">表3-338. 结构体 hpdf_rc_parameter_struct</a>
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* initialize the parameters of regular conversion */

hpdf_rc_parameter_struct hpdf_rc_init_struct;

hpdf_rc_struct_para_init(&hpdf_rc_init_struct);
```

### 函数 hpdf\_ic\_struct\_para\_init

函数hpdf\_ic\_struct\_para\_init描述见下表：

表 3-344. 函数 hpdf\_ic\_struct\_para\_init

函数名称	hpdf_ic_struct_para_init
函数原型	void hpdf_ic_struct_para_init(hpdf_ic_parameter_struct* init_struct);
功能描述	初始化注入转换结构体参数
先决条件	-
被调用函数	-

输入参数{in}	
*init_struct	HPDF注入转换初始化结构体，参考 <a href="#">表3-339. 结构体</a> <a href="#">hpdf_ic_parameter_struct</a>
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* initialize the parameters of inserted conversion */
```

```
hpdf_ic_parameter_struct hpdf_ic_init_struct;
```

```
hpdf_ic_struct_para_init(&hpdf_ic_init_struct);
```

### 函数 hpdf\_enable

函数hpdf\_enable描述见下表：

表 3-345. 函数 hpdf\_enable

函数名称	hpdf_enable
函数原型	void hpdf_enable(void);
功能描述	使能HPDF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 the HPDF module */
```

```
hpdf_enable();
```

### 函数 hpdf\_disable

函数hpdf\_disable描述见下表：

表 3-346. 函数 hpdf\_disable

函数名称	hpdf_disable
函数原型	void hpdf_disable (void);
功能描述	禁止HPDF模块
先决条件	-

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

例如：

```
/* disable the HPDF module */
```

```
hpdf_disable();
```

### 函数 hpdf\_channel\_init

函数hpdf\_channel\_init描述见下表：

表 3-347. 函数 hpdf\_channel\_init

函数名称	hpdf_channel_init
函数原型	void hpdf_channel_init(hpdf_channel_enum channelx, hpdf_channel_parameter_struct* init_struct);
功能描述	初始化HPDF通道
先决条件	-
被调用函数	-
输入参数{in}	
channelx	HPDF模块通道
CHANNELx(x=0..1)	HPDF通道选择，参考 <a href="#">表3-331. 枚举类型hpdf_channel_enum</a>
输入参数{in}	
*init_struct	初始化HPDF通道参数，结构体成员参考 <a href="#">表3-336. 结构体hpdf_channel_parameter_struct</a>
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* initialize the HPDF channel0 */
```

```
hpdf_channel_parameter_struct hpdf_channel_init_struct;
```

```
/* initialize HPDF channel0 */
```

```
hpdf_channel_init_struct.data_packing_mode = DPM_STANDARD_MODE;
```

```
hpdf_channel_init_struct.malfunction_monitor = MM_ENABLE;
```

```
hpdf_channel_init_struct.spi_ck_source = EXTERNAL_CKIN;
```

```

hpdf_channel_init_struct.channel_muxlexer = SERIAL_INPUT;

hpdf_channel_init_struct.serial_interface = SPI_RISING_EDGE;

hpdf_channel_init_struct.calibration_offset = 0;

hpdf_channel_init_struct.right_bit_shift = 0;

hpdf_channel_init_struct.mm_counter_threshold = 110;

hpdf_channel_init_struct.plsk_value = 0;

hpdf_channel_init(CHANNEL0, &hpdf_channel_init_struct);

```

### 函数 hpdf\_filter\_init

函数hpdf\_filter\_init描述见下表：

**表 3-348. 函数 hpdf\_filter\_init**

函数名称	hpdf_filter_init
函数原型	void hpdf_filter_init(hpdf_filter_enum filtery, hpdf_filter_parameter_struct* init_struct);
功能描述	初始化HPDF滤波器
先决条件	-
被调用函数	-
输入参数{in}	
filtery	HPDF模块滤波器
FLTy(y=0..1)	HPDF滤波器选择，参考 <a href="#">表3-332. 枚举类型hpdf_filter_enum</a>
输入参数{in}	
*init_struct	初始化HPDF滤波器参数，结构体成员参考 <a href="#">表3-337. 结构体 hpdf_filter_parameter_struct</a>
输出参数{out}	
-	-
返回值	
-	-

例如：

```

/* initialize the HPDF fliter0 */

hpdf_filter_parameter_struct hpdf_filter_init_struct;

hpdf_filter_init_struct.tm_fast_mode = TMFM_DISABLE;

hpdf_filter_init_struct.tm_channel = TMCHEN_CHANNEL0;

hpdf_filter_init_struct.tm_high_threshold = tm_high_val;

hpdf_filter_init_struct.tm_low_threshold = tm_low_val;

hpdf_filter_init_struct.extreme_monitor_channel = EM_CHANNEL0;

```

```

hpdf_filter_init_struct.sinc_filter = FLT_SINC3;

hpdf_filter_init_struct.sinc_oversample = FLT_OVER_SAMPLE_32;

hpdf_filter_init_struct.integrator_oversample = INTEGRATOR_BYPASS;

hpdf_filter_init(FLT0, &hpdf_filter_init_struct);

```

### 函数 hpdf\_rc\_init

函数hpdf\_rc\_init描述见下表：

**表 3-349. 函数 hpdf\_rc\_init**

函数名称	hpdf_rc_init
函数原型	void hpdf_rc_init(hpdf_filter_enum filtery, hpdf_rc_parameter_struct* init_struct);
功能描述	初始化规则转换
先决条件	-
被调用函数	-
输入参数{in}	
filtery	HPDF模块滤波器
FLTy(y=0..1)	HPDF滤波器选择，参考 <a href="#">表3-332. 枚举类型hpdf_filter_enum</a>
输入参数{in}	
*init_struct	初始化规则转换参数，结构体成员参考 <a href="#">表3-338. 结构体 hpdf_rc_parameter_struct</a>
输出参数{out}	
-	-
返回值	
-	-

例如：

```

/* initialize regular conversion of the HPDF fliter0 */

hpdf_rc_parameter_struct hpdf_rc_init_struct;

hpdf_rc_init_struct.fast_mode = FAST_DISABLE;

hpdf_rc_init_struct.rcs_channel = RCS_CHANNEL0;

hpdf_rc_init_struct.rcdmaen = RCDMAEN_ENABLE;

hpdf_rc_init_struct.continuous_mode = RCCM_ENABLE;

hpdf_rc_init(FLT0, &hpdf_rc_init_struct);

```

### 函数 hpdf\_ic\_init

函数hpdf\_ic\_init描述见下表：

表 3-350. 函数 `hpdf_ic_init`

函数名称	<code>hpdf_ic_init</code>
函数原型	<code>void hpdf_ic_init(hpdf_filter_enum filtery, hpdf_ic_parameter_struct* init_struct);</code>
功能描述	初始化注入转换
先决条件	-
被调用函数	-
输入参数{in}	
<b>filtery</b>	HPDF模块滤波器
<i>FLTy(y=0..1)</i>	HPDF滤波器选择, 参考 <a href="#">枚举类型hpdf_filter_enum</a>
输入参数{in}	
<b>*init_struct</b>	初始化注入转换参数, 结构体成员参考 <a href="#">表3-339. 结构体 hpdf_ic_parameter_struct</a>
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* initialize inserted conversion of the HPDF filter0 */
hpdf_ic_parameter_struct hpdf_ic_init_struct;
hpdf_ic_init_struct.ic_channel_group = IGCSEL_CHANNEL0_1;
hpdf_ic_init_struct.scmmod = SCMOD_ENABLE;
hpdf_ic_init_struct.icdmaen = ICDMAEN_ENABLE;
hpdf_ic_init_struct.icsyn = ICSYN_DISABLE;
hpdf_ic_init(FLT0, &hpdf_ic_init_struct);
```

### 函数 `hpdf_clock_output_config`

函数`hpdf_clock_output_config`描述见下表:

表 3-351. 函数 `hpdf_clock_output_config`

函数名称	<code>hpdf_clock_output_config</code>
函数原型	<code>void hpdf_clock_output_config(uint32_t source, uint8_t divider, uint32_t mode);</code>
功能描述	配置串行输出时钟
先决条件	禁止HPDF
被调用函数	-
输入参数{in}	
<b>source</b>	串行输出时钟源
<i>SERIAL_SYSTEM_CLK</i>	串行输出时钟源为系统时钟

<i>SERIAL_SYSTEM_CLK</i>	串行输出时钟源为音频时钟
输入参数{in}	
<b>divider</b>	串行输出时钟分频系数（0-255）
输入参数{in}	
<b>mode</b>	串行输出时钟占空比模式
<i>CKOUTDM_DISABLE</i>	禁止串行输出时钟占空比模式
<i>CKOUTDM_ENABLE</i>	使能串行输出时钟占空比模式
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* configure serial clock output */
```

```
hpdf_clock_output_config(SERIAL_SYSTEM_CLK, 175, CKOUTDM_ENABLE);
```

### 函数 **hpdf\_clock\_output\_source\_config**

函数hpdf\_clock\_output\_source\_config描述见下表：

**表 3-352. 函数 hpdf\_clock\_output\_source\_config**

函数名称	hpdf_clock_output_source_config
函数原型	void hpdf_clock_output_source_config(uint32_t source);
功能描述	配置串行输出时钟源
先决条件	禁止HPDF
被调用函数	-
输入参数{in}	
<b>source</b>	串行输出时钟源
<i>SERIAL_SYSTEM_CLK</i>	串行输出时钟源为系统时钟
<i>SERIAL_SYSTEM_CLK</i>	串行输出时钟源为音频时钟
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* configure serial clock output source */
```

```
hpdf_clock_output_config(SERIAL_SYSTEM_CLK);
```

### 函数 **hpdf\_clock\_output\_duty\_mode\_disable**

函数hpdf\_clock\_output\_duty\_mode\_disable描述见下表：



表 3-353. 函数 `hpdf_clock_output_duty_mode_disable`

函数名称	<code>hpdf_clock_output_duty_mode_disable</code>
函数原型	<code>void hpdf_clock_output_duty_mode_disable(void);</code>
功能描述	禁止串行输出时钟占空比模式
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 serial clock output duty mode*/
```

```
hpdf_clock_output_duty_mode_disable();
```

### 函数 `hpdf_clock_output_duty_mode_enable`

函数`hpdf_clock_output_duty_mode_enable`描述见下表：

表 3-354. 函数 `hpdf_clock_output_duty_mode_enable`

函数名称	<code>hpdf_clock_output_duty_mode_enable</code>
函数原型	<code>void hpdf_clock_output_duty_mode_enable(void);</code>
功能描述	使能串行输出时钟占空比模式
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 serial clock output duty mode*/
```

```
hpdf_clock_output_duty_mode_enable();
```

### 函数 `hpdf_clock_output_divider_config`

函数`hpdf_clock_output_divider_config`描述见下表：

表 3-355. 函数 `hpdf_clock_output_divider_config`

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

函数原型	void hpdf_clock_output_divider_config(uint8_t divider);
功能描述	配置串行输出时钟分频系数
先决条件	禁止HPDF
被调用函数	-
输入参数{in}	
divider	串行输出时钟分频系数（0-255）
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* configure serial clock output divider */
hpdf_clock_output_divider_config (255);
```

### 函数 hpdf\_channel\_enable

函数hpdf\_channel\_enable描述见下表：

表 3-356. 函数 hpdf\_channel\_enable

函数名称	hpdf_channel_enable
函数原型	void hpdf_channel_enable(hpdf_channel_enum channelx);
功能描述	使能通道
先决条件	-
被调用函数	-
输入参数{in}	
channelx	HPDF模块通道
CHANNELx(x=0..1)	HPDF通道选择，参考 <a href="#">表3-331. 枚举类型hpdf_channel_enum</a>
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* enable channel0 */
hpdf_channel_enable(CHANNEL0);
```

### 函数 hpdf\_channel\_disable

函数hpdf\_channel\_disable描述见下表：

表 3-357. 函数 hpdf\_channel\_disable

函数名称	hpdf_channel_disable
函数原型	void hpdf_channel_disable(hpdf_channel_enum channelx);

功能描述	禁止通道
先决条件	-
被调用函数	-
输入参数{in}	
channelx	HPDF模块通道
CHANNELx(x=0..1)	HPDF通道选择, 参考 <a href="#">表3-331. 枚举类型hpdf_channel_enum</a>
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* disable channel0 */
```

```
hpdf_channel_disable(CHANNEL0);
```

### 函数 hpdf\_spi\_clock\_source\_config

函数hpdf\_spi\_clock\_source\_config描述见下表:

表 3-358. 函数 hpdf\_spi\_clock\_source\_config

函数名称	hpdf_spi_clock_source_config
函数原型	void hpdf_spi_clock_source_config(hpdf_channel_enum channelx, uint32_t clock_source);
功能描述	配置SPI接口时钟源
先决条件	-
被调用函数	-
输入参数{in}	
channelx	HPDF模块通道
CHANNELx(x=0..1)	HPDF通道选择, 参考 <a href="#">表3-331. 枚举类型hpdf_channel_enum</a>
输入参数{in}	
clock_source	SPI接口时钟源
EXTERNAL_CKIN	外部输入时钟
INTERNAL_CKOUT	内部CKOUT时钟
HALF_CKOUT_FALLING_EDGE	内部每第二个CKOUT时钟的下降沿
HALF_CKOUT_RISING_EDGE	内部每第二个CKOUT时钟的上升沿
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* configure external input clock as SPI clock source */
```

```
hpdf_spi_clock_source_config(CHANNEL0, EXTERNAL_CKIN);
```

### 函数 hpdf\_serial\_interface\_type\_config

函数hpdf\_serial\_interface\_type\_config描述见下表：

**表 3-359. 函数 hpdf\_serial\_interface\_type\_config**

函数名称	hpdf_serial_interface_type_config
函数原型	void hpdf_serial_interface_type_config(hpdf_channel_enum channelx, uint32_t type);
功能描述	配置串行接口类型
先决条件	-
被调用函数	-
输入参数{in}	
channelx	HPDF模块通道
CHANNELx(x=0..1)	HPDF通道选择，参考 <a href="#">表3-331. 枚举类型hpdf_channel_enum</a>
输入参数{in}	
type	串行接口类型
SPI_RISING_EDGE	SPI接口上升沿采样
SPI_FALLING_EDGE	SPI接口下降沿采样
MANCHESTER_CODE 0	曼切斯特编码输入：上升沿=逻辑0，下降沿=逻辑1
MANCHESTER_CODE 1	曼切斯特编码输入：上升沿=逻辑1，下降沿=逻辑0
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/*configure external input clock as SPI clock source */
```

```
hpdf_spi_clock_source_config(CHANNEL0, EXTERNAL_CKIN);
```

### 函数 hpdf\_malfunction\_monitor\_disable

函数hpdf\_malfunction\_monitor\_disable描述见下表：

**表 3-360. 函数 hpdf\_malfunction\_monitor\_disable**

函数名称	hpdf_malfunction_monitor_disable
函数原型	void hpdf_malfunction_monitor_disable(hpdf_channel_enum channelx);
功能描述	禁止故障检测器
先决条件	-
被调用函数	-

输入参数{in}	
<b>channelx</b>	HPDF模块通道
<i>CHANNELx(x=0..1)</i>	HPDF通道选择, 参考 <a href="#">表3-331. 枚举类型hpdf_channel_enum</a>
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* disable malfunction monitor */
```

```
hpdf_malfunction_monitor_disable(CHANNEL0);
```

### 函数 hpdf\_malfunction\_monitor\_enable

函数hpdf\_malfunction\_monitor\_enable描述见下表:

**表 3-361. 函数 hpdf\_malfunction\_monitor\_enable**

<b>函数名称</b>	hpdf_malfunction_monitor_enable
<b>函数原型</b>	void hpdf_malfunction_monitor_enable(hpdf_channel_enum channelx);
<b>功能描述</b>	使能故障检测器
<b>先决条件</b>	-
<b>被调用函数</b>	-
输入参数{in}	
<b>channelx</b>	HPDF模块通道
<i>CHANNELx(x=0..1)</i>	HPDF通道选择, 参考 <a href="#">表3-331. 枚举类型hpdf_channel_enum</a>
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* enable malfunction monitor */
```

```
hpdf_malfunction_monitor_enable(CHANNEL1);
```

### 函数 hpdf\_clock\_loss\_disable

函数hpdf\_clock\_loss\_disable描述见下表:

**表 3-362. 函数 hpdf\_clock\_loss\_disable**

<b>函数名称</b>	hpdf_clock_loss_disable
<b>函数原型</b>	void hpdf_clock_loss_disable(hpdf_channel_enum channelx);
<b>功能描述</b>	禁止时钟检测
<b>先决条件</b>	CHANNELx(x=0...1)禁止
<b>被调用函数</b>	-

输入参数{in}	
<b>channelx</b>	HPDF模块通道
<i>CHANNELx(x=0..1)</i>	HPDF通道选择, 参考 <a href="#">表3-331. 枚举类型hpdf_channel_enum</a>
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* disable clock loss detector */
```

```
hpdf_clock_loss_disable(CHANNEL0);
```

### 函数 hpdf\_clock\_loss\_enable

函数hpdf\_clock\_loss\_enable描述见下表:

表 3-363. 函数 hpdf\_clock\_loss\_enable

<b>函数名称</b>	hpdf_clock_loss_enable
<b>函数原型</b>	void hpdf_clock_loss_enable(hpdf_channel_enum channelx);
<b>功能描述</b>	使能时钟检测
<b>先决条件</b>	CHANNELx(x=0...1)禁止
<b>被调用函数</b>	-
输入参数{in}	
<b>channelx</b>	HPDF模块通道
<i>CHANNELx(x=0..1)</i>	HPDF通道选择, 参考 <a href="#">表3-331. 枚举类型hpdf_channel_enum</a>
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* enable clock loss detector */
```

```
hpdf_clock_loss_enable(CHANNEL0);
```

### 函数 hpdf\_channel\_pin\_redirection\_disable

函数hpdf\_channel\_pin\_redirection\_disable描述见下表:

表 3-364. 函数 hpdf\_channel\_pin\_redirection\_disable

<b>函数名称</b>	hpdf_channel_pin_redirection_disable
<b>函数原型</b>	void hpdf_channel_pin_redirection_disable(hpdf_channel_enum channelx);
<b>功能描述</b>	禁止通道引脚重定向
<b>先决条件</b>	CHANNELx(x=0...1)禁止
<b>被调用函数</b>	-

输入参数{in}	
<b>channelx</b>	HPDF模块通道
<i>CHANNELx(x=0..1)</i>	HPDF通道选择, 参考 <a href="#">表3-331. 枚举类型hpdf_channel_enum</a>
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* disable channel0 inputs pins redirection */
```

```
hpdf_channel_pin_redirection_disable(CHANNEL0);
```

### 函数 hpdf\_channel\_pin\_redirection\_enable

函数hpdf\_channel\_pin\_redirection\_enable描述见下表:

表 3-365. 函数 hpdf\_channel\_pin\_redirection\_enable

<b>函数名称</b>	hpdf_channel_pin_redirection_enable
<b>函数原型</b>	void hpdf_channel_pin_redirection_enable(hpdf_channel_enum channelx);
<b>功能描述</b>	使能通道引脚重定向
<b>先决条件</b>	CHANNELx(x=0...1)禁止
<b>被调用函数</b>	-
输入参数{in}	
<b>channelx</b>	HPDF模块通道
<i>CHANNELx(x=0..1)</i>	HPDF通道选择, 参考 <a href="#">表3-331. 枚举类型hpdf_channel_enum</a>
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* enable channel0 inputs pins redirection */
```

```
hpdf_channel_pin_redirection_enable(CHANNEL0);
```

### 函数 hpdf\_channel\_multiplexer\_config

函数hpdf\_channel\_multiplexer\_config描述见下表:

表 3-366. 函数 hpdf\_channel\_multiplexer\_config

<b>函数名称</b>	hpdf_channel_multiplexer_config
<b>函数原型</b>	void hpdf_channel_multiplexer_config(hpdf_channel_enum channelx, uint32_t data_source);
<b>功能描述</b>	配置复用通道输入数据源
<b>先决条件</b>	CHANNELx(x=0...1)禁止

被调用函数	-
输入参数{in}	
channelx	HPDF模块通道
CHANNELx(x=0..1)	HPDF通道选择, 参考 <a href="#">表3-331. 枚举类型hpdf_channel_enum</a>
输入参数{in}	
data_source	输入数据源
SERIAL_INPUT	输入数据源为串行输入数据
INTERNAL_INPUT	输入数据源为内部并行数据寄存器数据
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* configure channel multiplexer select input data source */
hpdf_channel_mux_config(CHANNEL0, SERIAL_INPUT);
```

### 函数 hpdf\_data\_pack\_mode\_config

函数hpdf\_data\_pack\_mode\_config描述见下表:

表 3-367. 函数 hpdf\_data\_pack\_mode\_config

函数名称	hpdf_data_pack_mode_config
函数原型	void hpdf_data_pack_mode_config(hpdf_channel_enum channelx, uint32_t mode);
功能描述	配置数据封装模式
先决条件	CHANNELx(x=0...1)禁止
被调用函数	-
输入参数{in}	
channelx	HPDF模块通道
CHANNELx(x=0..1)	HPDF通道选择, 参考 <a href="#">表3-331. 枚举类型hpdf_channel_enum</a>
输入参数{in}	
mode	数据封装模式
DPM_STANDARD_MODE	标准模式
DPM_INTERLEAVED_MODE	交错模式
DPM_DUAL_MODE	双通道模式
输出参数{out}	
-	-
返回值	
-	-

例如:



```
/* configure data packing mode */
```

```
hpdf_data_pack_mode_config(CHANNEL0, DPM_STANDARD_MODE);
```

### 函数 hpdf\_data\_right\_bit\_shift\_config

函数hpdf\_data\_right\_bit\_shift\_config描述见下表：

**表 3-368. 函数 hpdf\_data\_right\_bit\_shift\_config**

函数名称	hpdf_data_right_bit_shift_config
函数原型	void hpdf_data_right_bit_shift_config(hpdf_channel_enum channelx, uint8_t right_shift);
功能描述	配置数据封装模式
先决条件	-
被调用函数	-
输入参数{in}	
channelx	HPDF模块通道
CHANNELx(x=0..1)	HPDF通道选择，参考 <a href="#">表3-331. 枚举类型hpdf_channel_enum</a>
输入参数{in}	
right_shift	数据右移的位数（0-31）
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* configure data right bit-shift */
```

```
hpdf_data_right_bit_shift_config(CHANNEL0, 5);
```

### 函数 hpdf\_calibration\_offset\_config

函数hpdf\_calibration\_offset\_config描述见下表：

**表 3-369. 函数 hpdf\_calibration\_offset\_config**

函数名称	hpdf_calibration_offset_config
函数原型	void hpdf_calibration_offset_config(hpdf_channel_enum channelx, int32_t offset);
功能描述	配置偏移校正
先决条件	-
被调用函数	-
输入参数{in}	
channelx	HPDF模块通道
CHANNELx(x=0..1)	HPDF通道选择，参考 <a href="#">表3-331. 枚举类型hpdf_channel_enum</a>
输入参数{in}	
offset	24位偏移校正（-8388608~8388607）

输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* configure calibration offset */
```

```
hpdf_calibration_offset_config (CHANNEL0, -255);
```

### 函数 hpdf\_malfunction\_break\_signal\_config

函数hpdf\_malfunction\_break\_signal\_config描述见下表：

**表 3-370. 函数 hpdf\_malfunction\_break\_signal\_config**

函数名称	hpdf_malfunction_break_signal_config
函数原型	void hpdf_malfunction_break_signal_config(hpdf_channel_enum channelx, uint32_t break_signal);
功能描述	配置故障检测器断路信号
先决条件	-
被调用函数	-
输入参数{in}	
channelx	HPDF模块通道
CHANNELx(x=0..1)	HPDF通道选择，参考 <a href="#">表3-331. 枚举类型hpdf_channel_enum</a>
输入参数{in}	
break_signal	数据封装模式
NO_MM_BREAK	无断路信号被分配
MM_BREAK0	断路信号0被分配至所监视的通道
MM_BREAK1	断路信号1被分配至所监视的通道
MM_BREAK0_1	断路信号0和1均被分配至所监视的通道
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* configure break signal is distributed to malfunction monitor on channel0 */
```

```
hpdf_data_pack_mode_config(CHANNEL0, MM_BREAK0_1);
```

### 函数 hpdf\_malfunction\_counter\_config

函数hpdf\_malfunction\_counter\_config描述见下表：

**表 3-371. 函数 hpdf\_malfunction\_counter\_config**

函数名称	hpdf_malfunction_counter_config
------	---------------------------------

函数原型	void hpdf_malfunction_counter_config(hpdf_channel_enum channelx, uint8_t threshold);
功能描述	配置故障监视器计数器阈值
先决条件	-
被调用函数	-
输入参数{in}	
channelx	HPDF模块通道
CHANNELx(x=0..1)	HPDF通道选择, 参考 <a href="#">表3-331. 枚举类型hpdf_channel_enum</a>
输入参数{in}	
threshold	故障监视器计数器阈值 (0-255)
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* configure malfunction monitor counter threshold */
```

```
hpdf_malfunction_counter_config(CHANNEL0, 255);
```

### 函数 hpdf\_write\_parallel\_data\_standard\_mode

函数hpdf\_write\_parallel\_data\_standard\_mode描述见下表:

**表 3-372. 函数 hpdf\_write\_parallel\_data\_standard\_mode**

函数名称	hpdf_write_parallel_data_standard_mode
函数原型	void hpdf_write_parallel_data_standard_mode(hpdf_channel_enum channelx, int16_t data);
功能描述	写入数据封装标准模式下的并行数据
先决条件	-
被调用函数	-
输入参数{in}	
channelx	HPDF模块通道
CHANNELx(x=0..1)	HPDF通道选择, 参考 <a href="#">表3-331. 枚举类型hpdf_channel_enum</a>
输入参数{in}	
data	并行数据
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* write the parallel data on standard mode of data packing */
```

```
write the parallel data on standard mode of data packing(CHANNEL0, 0xEFFF);
```

## 函数 hpdf\_write\_parallel\_data\_interleaved\_mode

函数hpdf\_write\_parallel\_data\_interleaved\_mode描述见下表：

**表 3-373. 函数 hpdf\_write\_parallel\_data\_interleaved\_mode**

函数名称	hpdf_write_parallel_data_interleaved_mode
函数原型	void hpdf_write_parallel_data_interleaved_mode(hpdf_channel_enum channelx, int32_t data);
功能描述	写入数据封装交错模式下的并行数据
先决条件	-
被调用函数	-
输入参数{in}	
channelx	HPDF模块通道
CHANNELx(x=0..1)	HPDF通道选择，参考 <a href="#">表3-331. 枚举类型hpdf_channel_enum</a>
输入参数{in}	
data	并行数据
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* write the parallel data on interleaved mode of data packing */
hpdf_write_parallel_data_interleaved_mode(CHANNEL0, 0xEFFFFFFF);
```

## 函数 hpdf\_write\_parallel\_data\_dual\_mode

函数hpdf\_write\_parallel\_data\_dual\_mode描述见下表：

**表 3-374. 函数 hpdf\_write\_parallel\_data\_dual\_mode**

函数名称	hpdf_write_parallel_data_dual_mode
函数原型	void hpdf_write_parallel_data_dual_mode(hpdf_channel_enum channelx, int32_t data);
功能描述	写入数据封装双通道模式下的并行数据
先决条件	-
被调用函数	-
输入参数{in}	
channelx	HPDF模块通道
CHANNELx(x=0..1)	HPDF通道选择，参考 <a href="#">表3-331. 枚举类型hpdf_channel_enum</a>
输入参数{in}	
data	并行数据
输出参数{out}	
-	-
返回值	

-	-
---	---

例如：

```
/* write the parallel data on dual mode of data packing */
```

```
hpdf_write_parallel_data_dual_mode(CHANNEL0, 0xFFFFFFFF);
```

### 函数 hpdf\_pulse\_skip\_update

函数hpdf\_pulse\_skip\_update描述见下表：

表 3-375. 函数 hpdf\_pulse\_skip\_update

函数名称	hpdf_pulse_skip_update
函数原型	void hpdf_pulse_skip_update(hpdf_channel_enum channelx, uint8_t number);
功能描述	更新跳频脉冲数量
先决条件	-
被调用函数	-
输入参数{in}	
channelx	HPDF模块通道
CHANNELx(x=0..1)	HPDF通道选择，参考 <a href="#">表3-331. 枚举类型hpdf_channel_enum</a>
输入参数{in}	
number	将要被跳过的串行输入样本数
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* update the number of pulses to skip */
```

```
pdf_pulse_skip_update(CHANNEL0, 63);
```

### 函数 hpdf\_pulse\_skip\_read

函数hpdf\_pulse\_skip\_read描述见下表：

表 3-376. 函数 hpdf\_pulse\_skip\_read

函数名称	hpdf_pulse_skip_read
函数原型	uint8_t hpdf_pulse_skip_read(hpdf_channel_enum channelx);
功能描述	读取跳频脉冲数量
先决条件	-
被调用函数	-
输入参数{in}	
channelx	HPDF模块通道
CHANNELx(x=0..1)	HPDF通道选择，参考 <a href="#">表3-331. 枚举类型hpdf_channel_enum</a>

输出参数{out}	
-	-
返回值	
uint8_t	跳频脉冲数量

例如：

```
/* read the number of pulses to skip */
uint8_t value;
value = hpdf_pulse_skip_read(CHANNEL0);
```

### 函数 hpdf\_filter\_enable

函数hpdf\_filter\_enable描述见下表：

表 3-377. 函数 hpdf\_filter\_enable

函数名称	hpdf_filter_enable
函数原型	void hpdf_filter_enable(hpdf_filter_enum filtery);
功能描述	使能滤波器
先决条件	-
被调用函数	-
输入参数{in}	
filtery	HPDF模块滤波器
FLTy(y=0..1)	HPDF滤波器选择，参考 <a href="#">表3-332. 枚举类型hpdf_filter_enum</a>
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* enable filter0 */
hpdf_filter_enable(FLT0);
```

### 函数 hpdf\_filter\_disable

函数hpdf\_filter\_disable描述见下表：

表 3-378. 函数 hpdf\_filter\_disable

函数名称	hpdf_filter_disable
函数原型	void hpdf_filter_disable(hpdf_filter_enum filtery);
功能描述	禁止滤波器
先决条件	-
被调用函数	-
输入参数{in}	

<b>filtery</b>	HPDF模块滤波器
<i>FLTy(y=0..1)</i>	HPDF滤波器选择, 参考 <a href="#">表3-332. 枚举类型hpdf_filter_enum</a>
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* disable filter0 */
```

```
hpdf_filter_disable(FLT0);
```

### 函数 hpdf\_filter\_config

函数hpdf\_filter\_config描述见下表:

**表 3-379. 函数 hpdf\_filter\_config**

函数名称	hpdf_filter_config
函数原型	void hpdf_filter_config(hpdf_filter_enum filtery, uint32_t order, uint16_t oversample);
功能描述	配置滤波器阶数和过采样率
先决条件	-
被调用函数	-
输入参数{in}	
<b>filtery</b>	HPDF模块滤波器
<i>FLTy(y=0..1)</i>	HPDF滤波器选择, 参考 <a href="#">表3-332. 枚举类型hpdf_filter_enum</a>
输入参数{in}	
<b>order</b>	滤波器阶数
<i>FLT_FASTSINC</i>	FastSinc型滤波器
<i>FLT_SINC1</i>	Sinc1型滤波器
<i>FLT_SINC2</i>	Sinc2型滤波器
<i>FLT_SINC3</i>	Sinc3型滤波器
<i>FLT_SINC4</i>	Sinc4型滤波器
<i>FLT_SINC5</i>	Sinc5型滤波器
<b>oversample</b>	滤波器过采样率 (1-1024)
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* configure sinc filter order and oversample */
```

```
hpdf_filter_config(FLT0, FLT_SINC2, 64);
```

函数 `hpdf_integrator_oversample`

函数 `hpdf_integrator_oversample` 描述见下表：

表 3-380. 函数 `hpdf_integrator_oversample`

函数名称	<code>hpdf_integrator_oversample</code>
函数原型	<code>void hpdf_integrator_oversample(hpdf_filter_enum filtery, uint8_t oversample);</code>
功能描述	配置积分器过采样率
先决条件	-
被调用函数	-
输入参数{in}	
<b>filtery</b>	HPDF模块滤波器
<i>FLTy(y=0..1)</i>	HPDF滤波器选择，参考 <a href="#">表3-332. 枚举类型hpdf_filter_enum</a>
输入参数{in}	
<b>oversample</b>	积分器过采样率（1-256）
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* configure integrator oversampling rate */
```

```
hpdf_integrator_oversample(FLT0, 256);
```

函数 `hpdf_threshold_monitor_filter_config`

函数 `hpdf_threshold_monitor_filter_config` 描述见下表：

表 3-381. 函数 `hpdf_threshold_monitor_filter_config`

函数名称	<code>hpdf_threshold_monitor_filter_config</code>
函数原型	<code>void hpdf_threshold_monitor_filter_config(hpdf_channel_enum channelx, uint32_t order, uint8_t oversample);</code>
功能描述	配置阈值监视器的滤波器
先决条件	-
被调用函数	-
输入参数{in}	
<b>channelx</b>	HPDF模块通道
<i>CHANNELx(x=0..1)</i>	HPDF通道选择，参考 <a href="#">表3-331. 枚举类型hpdf_channel_enum</a>
输入参数{in}	
<b>order</b>	滤波器阶数
<i>TM_FASTSINC</i>	FastSinc型滤波器
<i>TM_SINC1</i>	Sinc1型滤波器
<i>TM_SINC2</i>	Sinc2型滤波器



TM_SINC3	Sinc3型滤波器
输入参数{in}	
oversample	滤波器过采样率（1-32）
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* configure threshold monitor filter order and oversample */
```

```
hpdf_threshold_monitor_filter_config (FLT0, TM_SINC3);
```

### 函数 hpdf\_threshold\_monitor\_filter\_read\_data

函数hpdf\_threshold\_monitor\_filter\_read\_data描述见下表：

表 3-382. 函数 hpdf\_threshold\_monitor\_filter\_read\_data

函数名称	hpdf_threshold_monitor_filter_read_data
函数原型	int16_t hpdf_threshold_monitor_filter_read_data(hpdf_channel_enum channelx);
功能描述	读取阈值监视器滤波器的数据
先决条件	
被调用函数	-
输入参数{in}	
channelx	HPDF模块通道
CHANNELx(x=0..1)	HPDF通道选择，参考 <a href="#">表3-331. 枚举类型hpdf_channel_enum</a>
输出参数{out}	
-	-
返回值	
int16_t	阈值监视器滤波器数据

例如：

```
/* read the threshold monitor filter data */
```

```
int16_t data;
```

```
data = hpdf_threshold_monitor_filter_read_data(CHANNEL0);
```

### 函数 hpdf\_threshold\_monitor\_fast\_mode\_disable

函数hpdf\_threshold\_monitor\_fast\_mode\_disable描述见下表：

表 3-383. 函数 hpdf\_threshold\_monitor\_fast\_mode\_disable

函数名称	hpdf_threshold_monitor_fast_mode_disable
函数原型	void hpdf_threshold_monitor_fast_mode_disable(hpdf_filter_enum filtery);

功能描述	禁止阈值监视器快速模式
先决条件	
被调用函数	-
输入参数{in}	
filtery	HPDF模块滤波器
FLTy(y=0..1)	HPDF滤波器选择, 参考 <a href="#">表3-332. 枚举类型hpdf_filter_enum</a>
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* disable threshold monitor fast mode */
hpdf_threshold_monitor_fast_mode_disable (FLT0);
```

### 函数 hpdf\_threshold\_monitor\_fast\_mode\_enable

函数hpdf\_threshold\_monitor\_fast\_mode\_enable描述见下表:

表 3-384. 函数 hpdf\_threshold\_monitor\_fast\_mode\_enable

函数名称	hpdf_threshold_monitor_fast_mode_enable
函数原型	void hpdf_threshold_monitor_fast_mode_enable(hpdf_filter_enum filtery);
功能描述	使能阈值监视器快速模式
先决条件	-
被调用函数	-
输入参数{in}	
filtery	HPDF模块滤波器
FLTy(y=0..1)	HPDF滤波器选择, 参考 <a href="#">表3-332. 枚举类型hpdf_filter_enum</a>
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* enable threshold monitor fast mode */
hpdf_threshold_monitor_fast_mode_enable(FLT0);
```

### 函数 hpdf\_threshold\_monitor\_channel

函数hpdf\_threshold\_monitor\_channel描述见下表:

表 3-385. 函数 hpdf\_threshold\_monitor\_channel

函数名称	hpdf_threshold_monitor_channel
函数原型	void hpdf_threshold_monitor_channel(hpdf_filter_enum filtery, uint32_t

	channel);
功能描述	配置阈值监视器通道
先决条件	-
被调用函数	-
输入参数{in}	
filtery	HPDF模块滤波器
FLTy(y=0..1)	HPDF滤波器选择, 参考 <a href="#">表3-332. 枚举类型hpdf_filter_enum</a>
输入参数{in}	
channel	阈值监视器所监视的通道
TMCHEN_DISABLE	禁止阈值监视器监视通道0或1
TMCHEN_CHANNEL0	阈值监视器监视通道0
TMCHEN_CHANNEL1	阈值监视器监视通道1
TMCHEN_CHANNEL0_1	阈值监视器监视通道0和1
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* configure threshold monitor channel */
```

```
hpdf_threshold_monitor_channel(FLT0, TMCHEN_CHANNEL0_1);
```

### 函数 hpdf\_threshold\_monitor\_high\_threshold

函数hpdf\_threshold\_monitor\_high\_threshold描述见下表:

表 3-386. 函数 hpdf\_threshold\_monitor\_high\_threshold

函数名称	hpdf_threshold_monitor_high_threshold
函数原型	void hpdf_threshold_monitor_high_threshold(hpdf_filter_enum filtery, int32_t value);
功能描述	配置阈值监视器上限阈值
先决条件	-
被调用函数	-
输入参数{in}	
filtery	HPDF模块滤波器
FLTy(y=0..1)	HPDF滤波器选择, 参考 <a href="#">表3-332. 枚举类型hpdf_filter_enum</a>
输入参数{in}	
value	上限阈值(-8388608~8388607)
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* configure threshold monitor high threshold value */
hpdf_threshold_monitor_high_threshold(FLT0, 32767);
```

### 函数 hpdf\_threshold\_monitor\_low\_threshold

函数hpdf\_threshold\_monitor\_low\_threshold描述见下表：

表 3-387. 函数 hpdf\_threshold\_monitor\_low\_threshold

函数名称	hpdf_threshold_monitor_low_threshold
函数原型	void hpdf_threshold_monitor_low_threshold(hpdf_filter_enum filtery, int32_t value);
功能描述	配置阈值监视器下限阈值
先决条件	-
被调用函数	-
输入参数{in}	
filtery	HPDF模块滤波器
FLTy(y=0..1)	HPDF滤波器选择，参考 <a href="#">表3-332. 枚举类型hpdf_filter_enum</a>
输入参数{in}	
value	下限阈值(-8388608~8388607)
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* configure threshold monitor low threshold value */
hpdf_threshold_monitor_low_threshold(FLT0, -32768);
```

### 函数 hpdf\_high\_threshold\_break\_signal

函数hpdf\_high\_threshold\_break\_signal描述见下表：

表 3-388. 函数 hpdf\_high\_threshold\_break\_signal

函数名称	hpdf_high_threshold_break_signal
函数原型	void hpdf_high_threshold_break_signal(hpdf_filter_enum filtery, uint32_t break_signal);
功能描述	配置阈值监视器上限阈值事件断路信号
先决条件	-
被调用函数	-
输入参数{in}	
filtery	HPDF模块滤波器
FLTy(y=0..1)	HPDF滤波器选择，参考 <a href="#">表3-332. 枚举类型hpdf_filter_enum</a>

输入参数{in}	
<b>break_signal</b>	HPDF断路信号
<i>NO_TM_HT_BREAK</i>	禁止断路信号分配给阈值监视器的上限阈值事件
<i>TM_HT_BREAK0</i>	断路信号0分配给阈值监视器的上限阈值事件
<i>TM_HT_BREAK1</i>	断路信号1分配给阈值监视器的上限阈值事件
<i>TM_HT_BREAK0_1</i>	断路信号0和1分配给阈值监视器的上限阈值事件
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* configure threshold monitor high threshold event break signal */
```

```
hpdf_high_threshold_break_signal(FTL0, TM_HT_BREAK0_1);
```

### 函数 hpdf\_low\_threshold\_break\_signal

函数hpdf\_low\_threshold\_break\_signal描述见下表：

表 3-389. 函数 hpdf\_low\_threshold\_break\_signal

函数名称	hpdf_low_threshold_break_signal
函数原型	void hpdf_low_threshold_break_signal(hpdf_filter_enum filtery, uint32_t break_signal);
功能描述	配置阈值监视器下限阈值事件断路信号
先决条件	-
被调用函数	-
输入参数{in}	
<b>filtery</b>	HPDF模块滤波器
<i>FLTy(y=0..1)</i>	HPDF滤波器选择，参考 <a href="#">表3-332. 枚举类型hpdf_filter_enum</a>
输入参数{in}	
<b>break_signal</b>	HPDF断路信号
<i>NO_TM_LT_BREAK</i>	禁止断路信号分配给阈值监视器的下限阈值事件
<i>TM_LT_BREAK0</i>	断路信号0分配给阈值监视器的下限阈值事件
<i>TM_LT_BREAK1</i>	断路信号1分配给阈值监视器的下限阈值事件
<i>TM_LT_BREAK0_1</i>	断路信号0和1分配给阈值监视器的下限阈值事件
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* configure threshold monitor low threshold event break signal */
```

```
hpdf_low_threshold_break_signal(FTL0, TM_LT_BREAK0_1);
```

函数 `hpdf_extremes_monitor_channel`

函数 `hpdf_extremes_monitor_channel` 描述见下表：

表 3-390. 函数 `hpdf_extremes_monitor_channel`

函数名称	<code>hpdf_extremes_monitor_channel</code>
函数原型	<code>void hpdf_extremes_monitor_channel(hpdf_filter_enum filtery, uint32_t channel);</code>
功能描述	配置极值监视器通道
先决条件	-
被调用函数	-
输入参数{in}	
<b>filtery</b>	HPDF模块滤波器
<i>FLTy(y=0..1)</i>	HPDF滤波器选择，参考 <a href="#">表3-332. 枚举类型hpdf_filter_enum</a>
输入参数{in}	
<b>channel</b>	极值监视器所监视的通道
<i>EM_CHANNEL_DISABLE</i>	禁止极值监视y接收通道0或1的数据
<i>EM_CHANNEL0</i>	极值监视y接收通道0的数据
<i>EM_CHANNEL1</i>	极值监视y接收通道1的数据
<i>EM_CHANNEL0_1</i>	极值监视y接收通道0和1的数据
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* configure extremes monitor channel */
```

```
hpdf_extremes_monitor_channel(FTL0, EM_CHANNEL0_1);
```

函数 `hpdf_extremes_monitor_maximum_get`

函数 `hpdf_extremes_monitor_maximum_get` 描述见下表：

表 3-391. 函数 `hpdf_extremes_monitor_maximum_get`

函数名称	<code>hpdf_extremes_monitor_maximum_get</code>
函数原型	<code>int32_t hpdf_extremes_monitor_maximum_get(hpdf_filter_enum filtery);</code>
功能描述	获取极值监视器最大极值
先决条件	-
被调用函数	-
输入参数{in}	
<b>filtery</b>	HPDF模块滤波器
<i>FLTy(y=0..1)</i>	HPDF滤波器选择，参考 <a href="#">表3-332. 枚举类型hpdf_filter_enum</a>
输出参数{out}	

-	-
返回值	
int32_t	最大极值

例如：

```
/* get the extremes monitor maximum value */
```

```
int32_t vlaue;
```

```
vlaue = hpdf_extremes_monitor_maximum_get(FTL0);
```

### 函数 hpdf\_extremes\_monitor\_minimum\_get

函数hpdf\_extremes\_monitor\_minimum\_get描述见下表：

表 3-392. 函数 hpdf\_extremes\_monitor\_minimum\_get

函数名称	hpdf_extremes_monitor_minimum_get
函数原型	int32_t hpdf_extremes_monitor_minimum_get(hpdf_filter_enum filtery);
功能描述	获取极值监视器最小极值
先决条件	-
被调用函数	-
输入参数{in}	
filtery	HPDF模块滤波器
FLTy(y=0..1)	HPDF滤波器选择，参考 <a href="#">表3-332. 枚举类型hpdf_filter_enum</a>
输出参数{out}	
-	-
返回值	
int32_t	最小极值

例如：

```
/* get the extremes monitor minimum value */
```

```
int32_t vlaue;
```

```
vlaue = hpdf_extremes_monitor_minimum_get(FTL0,);
```

### 函数 hpdf\_rc\_continuous\_disable

函数hpdf\_rc\_continuous\_disable描述见下表：

表 3-393. 函数 hpdf\_rc\_continuous\_disable

函数名称	hpdf_rc_continuous_disable
函数原型	void hpdf_rc_continuous_disable(hpdf_filter_enum filtery);
功能描述	禁止规则转换连续模式
先决条件	-
被调用函数	-
输入参数{in}	

<b>filtery</b>	HPDF模块滤波器
<i>FLTy(y=0..1)</i>	HPDF滤波器选择, 参考 <a href="#">表3-332. 枚举类型hpdf_filter_enum</a>
<b>输出参数{out}</b>	
-	-
<b>返回值</b>	
-	-

例如:

```
/* disable regular conversions continuous mode */
```

```
hpdf_rc_continuous_disable(FTL0);
```

### 函数 hpdf\_rc\_continuous\_enable

函数hpdf\_rc\_continuous\_enable描述见下表:

**表 3-394. 函数 hpdf\_rc\_continuous\_enable**

<b>函数名称</b>	hpdf_rc_continuous_enable
<b>函数原型</b>	void hpdf_rc_continuous_enable(hpdf_filter_enum filtery);
<b>功能描述</b>	使能规则转换连续模式
<b>先决条件</b>	-
<b>被调用函数</b>	-
<b>输入参数{in}</b>	
<b>filtery</b>	HPDF模块滤波器
<i>FLTy(y=0..1)</i>	HPDF滤波器选择, 参考 <a href="#">表3-332. 枚举类型hpdf_filter_enum</a>
<b>输出参数{out}</b>	
-	-
<b>返回值</b>	
-	-

例如:

```
/* enable regular conversions continuous mode */
```

```
hpdf_rc_continuous_enable(FTL0);
```

### 函数 hpdf\_rc\_start\_by\_software

函数hpdf\_rc\_start\_by\_software描述见下表:

**表 3-395. 函数 hpdf\_rc\_start\_by\_software**

<b>函数名称</b>	hpdf_rc_start_by_software
<b>函数原型</b>	void hpdf_rc_start_by_software(hpdf_filter_enum filtery);
<b>功能描述</b>	软件启动规则转换
<b>先决条件</b>	-
<b>被调用函数</b>	-
<b>输入参数{in}</b>	



<b>filtery</b>	HPDF模块滤波器
<i>FLTy(y=0..1)</i>	HPDF滤波器选择, 参考 <a href="#">表3-332. 枚举类型hpdf_filter_enum</a>
<b>输出参数{out}</b>	
-	-
<b>返回值</b>	
-	-

例如:

```
/* start regular channel conversion by software */
```

```
hpdf_rc_start_by_software(FTL0);
```

### 函数 hpdf\_rc\_syn\_disable

函数hpdf\_rc\_syn\_disable描述见下表:

**表 3-396. 函数 hpdf\_rc\_syn\_disable**

<b>函数名称</b>	hpdf_rc_syn_disable
<b>函数原型</b>	void hpdf_rc_syn_disable(hpdf_filter_enum filtery);
<b>功能描述</b>	禁止规则转换同步
<b>先决条件</b>	禁止FLTy(y=0..1)
<b>被调用函数</b>	-
<b>输入参数{in}</b>	
<b>filtery</b>	HPDF模块滤波器
<i>FLTy(y=0..1)</i>	HPDF滤波器选择, 参考 <a href="#">表3-332. 枚举类型hpdf_filter_enum</a>
<b>输出参数{out}</b>	
-	-
<b>返回值</b>	
-	-

例如:

```
/* disable regular conversion synchronously */
```

```
hpdf_rc_syn_disable(FTL0);
```

### 函数 hpdf\_rc\_syn\_enable

函数hpdf\_rc\_syn\_enable描述见下表:

**表 3-397. 函数 hpdf\_rc\_syn\_enable**

<b>函数名称</b>	hpdf_rc_syn_enable
<b>函数原型</b>	void hpdf_rc_syn_enable(hpdf_filter_enum filtery);
<b>功能描述</b>	使能规则转换同步
<b>先决条件</b>	禁止FLTy(y=0..1)
<b>被调用函数</b>	-
<b>输入参数{in}</b>	

<b>filtery</b>	HPDF模块滤波器
<i>FLTy(y=0..1)</i>	HPDF滤波器选择, 参考 <a href="#">表3-332. 枚举类型hpdf_filter_enum</a>
<b>输出参数{out}</b>	
-	-
<b>返回值</b>	
-	-

例如:

```
/* enable regular conversion synchronously */
```

```
hpdf_rc_syn_enable(FTL0);
```

### 函数 hpdf\_rc\_dma\_disable

函数hpdf\_rc\_dma\_disable描述见下表:

**表 3-398. 函数 hpdf\_rc\_dma\_disable**

<b>函数名称</b>	hpdf_rc_dma_disable
<b>函数原型</b>	void hpdf_rc_dma_disable(hpdf_filter_enum filtery);
<b>功能描述</b>	禁止规则转换DMA
<b>先决条件</b>	禁止FLTy(y=0..1)
<b>被调用函数</b>	-
<b>输入参数{in}</b>	
<b>filtery</b>	HPDF模块滤波器
<i>FLTy(y=0..1)</i>	HPDF滤波器选择, 参考 <a href="#">表3-332. 枚举类型hpdf_filter_enum</a>
<b>输出参数{out}</b>	
-	-
<b>返回值</b>	
-	-

例如:

```
/* disable regular conversion DMA channel */
```

```
hpdf_rc_dma_disable(FTL0);
```

### 函数 hpdf\_rc\_dma\_enable

函数hpdf\_rc\_dma\_enable描述见下表:

**表 3-399. 函数 hpdf\_rc\_dma\_enable**

<b>函数名称</b>	hpdf_rc_dma_enable
<b>函数原型</b>	void hpdf_rc_dma_enable(hpdf_filter_enum filtery);
<b>功能描述</b>	使能规则转换DMA
<b>先决条件</b>	禁止FLTy(y=0..1)
<b>被调用函数</b>	-
<b>输入参数{in}</b>	

<b>filtery</b>	HPDF模块滤波器
<i>FLTy(y=0..1)</i>	HPDF滤波器选择, 参考 <a href="#">表3-332. 枚举类型hpdf_filter_enum</a>
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* enable regular conversion DMA channel */
```

```
hpdf_rc_dma_enable(FTL0);
```

### 函数 hpdf\_rc\_channel\_config

函数hpdf\_rc\_channel\_config描述见下表:

**表 3-400. 函数 hpdf\_rc\_channel\_config**

函数名称	hpdf_rc_channel_config
函数原型	void hpdf_rc_channel_config(hpdf_filter_enum filtery, hpdf_channel_enum channelx);
功能描述	配置规则转换通道
先决条件	-
被调用函数	-
输入参数{in}	
<b>filtery</b>	HPDF模块滤波器
<i>FLTy(y=0..1)</i>	HPDF滤波器选择, 参考 <a href="#">表3-332. 枚举类型hpdf_filter_enum</a>
输入参数{in}	
<b>channelx</b>	HPDF模块通道
<i>CHANNELx(x=0..1)</i>	HPDF通道选择, 参考 <a href="#">表3-331. 枚举类型hpdf_channel_enum</a>
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* configure regular conversion channel */
```

```
hpdf_rc_channel_config(FTL0, CHANNEL1);
```

### 函数 hpdf\_rc\_fast\_mode\_disable

函数hpdf\_rc\_fast\_mode\_disable描述见下表:

**表 3-401. 函数 hpdf\_rc\_fast\_mode\_disable**

函数名称	hpdf_rc_fast_mode_disable
函数原型	void hpdf_rc_fast_mode_disable(hpdf_filter_enum filtery);

功能描述	禁止规则转换快速模式
先决条件	禁止FLTy(y=0..1)
被调用函数	-
输入参数{in}	
filtery	HPDF模块滤波器
FLTy(y=0..1)	HPDF滤波器选择, 参考 <a href="#">表3-332. 枚举类型hpdf_filter_enum</a>
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* disable regular conversion fast conversion mode */
```

```
hpdf_rc_fast_mode_disable(FTL0);
```

### 函数 hpdf\_rc\_fast\_mode\_enable

函数hpdf\_rc\_fast\_mode\_enable描述见下表:

表 3-402. 函数 hpdf\_rc\_fast\_mode\_enable

函数名称	hpdf_rc_fast_mode_enable
函数原型	void hpdf_rc_fast_mode_enable(hpdf_filter_enum filtery);
功能描述	使能规则转换快速模式
先决条件	禁止FLTy(y=0..1)
被调用函数	-
输入参数{in}	
filtery	HPDF模块滤波器
FLTy(y=0..1)	HPDF滤波器选择, 参考 <a href="#">表3-332. 枚举类型hpdf_filter_enum</a>
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* enable regular conversion fast conversion mode */
```

```
hpdf_rc_fast_mode_enable(FTL0);
```

### 函数 hpdf\_rc\_data\_get

函数hpdf\_rc\_data\_get描述见下表:

表 3-403. 函数 hpdf\_rc\_data\_get

函数名称	hpdf_rc_data_get
函数原型	int32_t hpdf_rc_data_get(hpdf_filter_enum filtery);

功能描述	获取规则转换数据
先决条件	-
被调用函数	-
输入参数{in}	
filtery	HPDF模块滤波器
FLTy(y=0..1)	HPDF滤波器选择, 参考 <a href="#">表3-332. 枚举类型hpdf_filter_enum</a>
输出参数{out}	
-	-
返回值	
int32_t	规则转换数据

例如:

```
/* get the regular conversion data */
int32_t vlaue;

vlaue = hpdf_rc_data_get(FTL0);
```

### 函数 hpdf\_rc\_channel\_get

函数hpdf\_rc\_channel\_get描述见下表:

表 3-404. 函数 hpdf\_rc\_channel\_get

函数名称	hpdf_rc_channel_get
函数原型	uint8_t hpdf_rc_channel_get(hpdf_filter_enum filtery);
功能描述	获取最近一次规则转换的通道
先决条件	-
被调用函数	-
输入参数{in}	
filtery	HPDF模块滤波器
FLTy(y=0..1)	HPDF滤波器选择, 参考 <a href="#">表3-332. 枚举类型hpdf_filter_enum</a>
输出参数{out}	
-	-
返回值	
uint8_t	通道

例如:

```
/* get the channel of regular channel most recently converted */
uint8_t channel;

channel = hpdf_rc_channel_get(FTL0);
```

### 函数 hpdf\_ic\_start\_by\_software

函数hpdf\_ic\_start\_by\_software描述见下表:

表 3-405. 函数 `hpdf_ic_start_by_software`

函数名称	<code>hpdf_ic_start_by_software</code>
函数原型	<code>void hpdf_ic_start_by_software(hpdf_filter_enum filtery);</code>
功能描述	软件启动注入转换
先决条件	-
被调用函数	-
输入参数{in}	
<b>filtery</b>	HPDF模块滤波器
<i>FLTy(y=0..1)</i>	HPDF滤波器选择, 参考 <a href="#">表3-332. 枚举类型hpdf_filter_enum</a>
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* start inserted channel conversion by software */
hpdf_ic_start_by_software(FTL0);
```

### 函数 `hpdf_ic_syn_disable`

函数`hpdf_ic_syn_disable`描述见下表:

表 3-406. 函数 `hpdf_ic_syn_disable`

函数名称	<code>hpdf_ic_syn_disable</code>
函数原型	<code>void hpdf_ic_syn_disable(hpdf_filter_enum filtery);</code>
功能描述	禁止注入转换同步
先决条件	禁止 <code>FLTy(y=0..1)</code>
被调用函数	-
输入参数{in}	
<b>filtery</b>	HPDF模块滤波器
<i>FLTy(y=0..1)</i>	HPDF滤波器选择, 参考 <a href="#">表3-332. 枚举类型hpdf_filter_enum</a>
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* disable inserted conversion synchronously */
hpdf_ic_syn_disable(FTL0);
```

### 函数 `hpdf_ic_syn_enable`

函数`hpdf_ic_syn_enable`描述见下表:

表 3-407. 函数 `hpdf_ic_syn_enable`

函数名称	<code>hpdf_ic_syn_enable</code>
函数原型	<code>void hpdf_ic_syn_enable(hpdf_filter_enum filtery);</code>
功能描述	使能注入转换同步
先决条件	禁止FLTy(y=0..1)
被调用函数	-
输入参数{in}	
<b>filtery</b>	HPDF模块滤波器
<i>FLTy(y=0..1)</i>	HPDF滤波器选择, 参考 <a href="#">表3-332. 枚举类型hpdf_filter_enum</a>
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* enable inserted conversion synchronously */
hpdf_ic_syn_enable(FTL0);
```

### 函数 `hpdf_ic_dma_disable`

函数`hpdf_ic_dma_disable`描述见下表:

表 3-408. 函数 `hpdf_ic_dma_disable`

函数名称	<code>hpdf_ic_dma_disable</code>
函数原型	<code>void hpdf_ic_dma_disable(hpdf_filter_enum filtery);</code>
功能描述	禁止注入转换DMA
先决条件	禁止FLTy(y=0..1)
被调用函数	-
输入参数{in}	
<b>filtery</b>	HPDF模块滤波器
<i>FLTy(y=0..1)</i>	HPDF滤波器选择, 参考 <a href="#">表3-332. 枚举类型hpdf_filter_enum</a>
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* disable inserted conversion DMA channel */
hpdf_ic_dma_disable(FTL0);
```

### 函数 `hpdf_ic_dma_enable`

函数`hpdf_ic_dma_enable`描述见下表:

表 3-409. 函数 `hpdf_ic_dma_enable`

函数名称	<code>hpdf_ic_dma_enable</code>
函数原型	<code>void hpdf_ic_dma_enable(hpdf_filter_enum filtery);</code>
功能描述	使能注入转换DMA
先决条件	禁止FLTy(y=0..1)
被调用函数	-
输入参数{in}	
<b>filtery</b>	HPDF模块滤波器
<i>FLTy(y=0..1)</i>	HPDF滤波器选择, 参考 <a href="#">表3-332. 枚举类型hpdf_filter_enum</a>
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* enable inserted conversion DMA channel */
hpdf_ic_dma_enable(FTL0);
```

### 函数 `hpdf_ic_scan_mode_disable`

函数`hpdf_ic_scan_mode_disable`描述见下表:

表 3-410. 函数 `hpdf_ic_scan_mode_disable`

函数名称	<code>hpdf_ic_scan_mode_disable</code>
函数原型	<code>void hpdf_ic_scan_mode_disable(hpdf_filter_enum filtery);</code>
功能描述	禁止注入转换扫描模式
先决条件	禁止FLTy(y=0..1)
被调用函数	-
输入参数{in}	
<b>filtery</b>	HPDF模块滤波器
<i>FLTy(y=0..1)</i>	HPDF滤波器选择, 参考 <a href="#">表3-332. 枚举类型hpdf_filter_enum</a>
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* disable scan conversion mode */
hpdf_ic_scan_mode_disable(FTL0);
```

### 函数 `hpdf_ic_scan_mode_enable`

函数`hpdf_ic_scan_mode_enable`描述见下表:



表 3-411. 函数 `hpdf_ic_scan_mode_enable`

函数名称	<code>hpdf_ic_scan_mode_enable</code>
函数原型	<code>void hpdf_ic_scan_mode_enable(hpdf_filter_enum filtery);</code>
功能描述	使能注入转换扫描模式
先决条件	禁止 <code>FLTy(y=0..1)</code>
被调用函数	-
输入参数{in}	
<b>filtery</b>	HPDF模块滤波器
<i>FLTy(y=0..1)</i>	HPDF滤波器选择, 参考 <a href="#">表3-332. 枚举类型hpdf_filter_enum</a>
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* enable scan conversion mode */
hpdf_ic_scan_mode_enable(FTL0);
```

### 函数 `hpdf_ic_trigger_signal_disbale`

函数`hpdf_ic_trigger_signal_disbale`描述见下表:

表 3-412. 函数 `hpdf_ic_trigger_signal_disbale`

函数名称	<code>hpdf_ic_trigger_signal_disbale</code>
函数原型	<code>void hpdf_ic_trigger_signal_disbale(hpdf_filter_enum filtery);</code>
功能描述	禁止注入转换触发信号
先决条件	禁止 <code>FLTy(y=0..1)</code>
被调用函数	-
输入参数{in}	
<b>filtery</b>	HPDF模块滤波器
<i>FLTy(y=0..1)</i>	HPDF滤波器选择, 参考 <a href="#">表3-332. 枚举类型hpdf_filter_enum</a>
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* disable inserted conversions trigger signal */
hpdf_ic_trigger_signal_disbale(FTL0);
```

### 函数 `hpdf_ic_trigger_signal_config`

函数`hpdf_ic_trigger_signal_config`描述见下表:

表 3-413. 函数 `hpdf_ic_trigger_signal_config`

函数名称	<code>hpdf_ic_trigger_signal_config</code>
函数原型	<code>void hpdf_ic_trigger_signal_config(hpdf_filter_enum filtery, uint32_t trigger, uint32_t trigger_edge);</code>
功能描述	配置注入转换触发信号和边沿
先决条件	禁止FLTy(y=0..1)
被调用函数	-
输入参数{in}	
<b>filtery</b>	HPDF模块滤波器
<i>FLTy(y=0..1)</i>	HPDF滤波器选择, 参考 <a href="#">表3-332. 枚举类型hpdf_filter_enum</a>
输入参数{in}	
<b>trigger</b>	注入转换触发信号
<i>HPDF_ITRG0</i>	注入转换触发信号为TIMER1_TRGO
<i>HPDF_ITRG1</i>	注入转换触发信号为TIMER2_TRGO
<i>HPDF_ITRG2</i>	注入转换触发信号为TIMER3_TRGO
<i>HPDF_ITRG3</i>	注入转换触发信号为TIMER4_TRGO
<i>HPDF_ITRG24</i>	注入转换触发信号为EXTI11
<i>HPDF_ITRG25</i>	注入转换触发信号为EXTI15
<i>HPDF_ITRG26</i>	注入转换触发信号为TIMER5_TRGO
输入参数{in}	
<b>trigger_edge</b>	注入转换触发边沿
<i>TRG_DISABLE</i>	禁止触发信号
<i>RISING_EDGE_TRG</i>	触发信号上升沿
<i>FALLING_EDGE_TRG</i>	触发信号下降沿
<i>EDGE_TRG</i>	触发信号双边沿
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* configure inserted conversions trigger signal and trigger edge */
```

```
hpdf_ic_trigger_signal_config(FTL0, HPDF_ITRG3, FALLING_EDGE_TRG);
```

### 函数 `hpdf_ic_channel_config`

函数 `hpdf_ic_channel_config` 描述见下表:

表 3-414. 函数 `hpdf_ic_channel_config`

函数名称	<code>hpdf_ic_channel_config</code>
函数原型	<code>void hpdf_ic_channel_config(hpdf_filter_enum filtery, uint32_t channel);</code>
功能描述	配置注入组转换通道
先决条件	-

被调用函数	-
输入参数{in}	
filtery	HPDF模块滤波器
FLTy(y=0..1)	HPDF滤波器选择, 参考 <a href="#">表3-332. 枚举类型hpdf_filter_enum</a>
输入参数{in}	
channel	注入组通道
IGCSEL_CHANNEL0	通道0属于注入组
IGCSEL_CHANNEL1	通道1属于注入组
IGCSEL_CHANNEL0_1	通道0和1属于注入组
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* configure inserted group conversions channel */
hpdf_ic_channel_config(FTL0, IGCSEL_CHANNEL0_1);
```

### 函数 hpdf\_ic\_data\_get

函数hpdf\_ic\_data\_get描述见下表:

表 3-415. 函数 hpdf\_ic\_data\_get

函数名称	hpdf_ic_data_get
函数原型	int32_t hpdf_ic_data_get(hpdf_filter_enum filtery);
功能描述	获取注入转换数据
先决条件	-
被调用函数	-
输入参数{in}	
filtery	HPDF模块滤波器
FLTy(y=0..1)	HPDF滤波器选择, 参考 <a href="#">表3-332. 枚举类型hpdf_filter_enum</a>
输出参数{out}	
-	-
返回值	
int32_t	注入转换数据

例如:

```
/* get the inserted conversions data */
int32_t vlaue;
vlaue = hpdf_ic_data_get(FTL0);
```

函数 `hpdf_ic_channel_get`

函数 `hpdf_ic_channel_get` 描述见下表：

表 3-416. 函数 `hpdf_ic_channel_get`

函数名称	<code>hpdf_ic_channel_get</code>
函数原型	<code>uint8_t hpdf_ic_channel_get(hpdf_filter_enum filtery);</code>
功能描述	获取最近一次注入转换的通道
先决条件	-
被调用函数	-
输入参数{in}	
<b>filtery</b>	HPDF模块滤波器
<i>FLTy(y=0..1)</i>	HPDF滤波器选择，参考 <a href="#">表3-332. 枚举类型hpdf_filter_enum</a>
输出参数{out}	
-	-
返回值	
<b>uint8_t</b>	通道

例如：

```
/* get the channel of inserted group channel most recently converted */
```

```
uint8_t channel;
```

```
channel = hpdf_ic_channel_get(FTL0);
```

函数 `hpdf_flag_get`

函数 `hpdf_flag_get` 描述见下表：

表 3-417. 函数 `hpdf_flag_get`

函数名称	<code>hpdf_flag_get</code>
函数原型	<code>FlagStatus hpdf_flag_get(hpdf_filter_enum filtery, hpdf_flag_enum flag);</code>
功能描述	获取HPDF标志位
先决条件	-
被调用函数	-
输入参数{in}	
<b>filtery</b>	HPDF模块滤波器
<i>FLTy(y=0..1)</i>	HPDF滤波器选择，参考 <a href="#">表3-332. 枚举类型hpdf_filter_enum</a>
输入参数{in}	
<b>flag</b>	HPDF模块标志位
<i>HPDF_FLAG_FLTy_IC EF</i>	注入转换结束标志
<i>HPDF_FLAG_FLTy_RC EF</i>	规则转换结束标志
<i>HPDF_FLAG_FLTy_IC</i>	注入转换溢出标志

<i>DOF</i>	
<i>HPDF_FLAG_FLTy_RC DOF</i>	规则转换溢出标志
<i>HPDF_FLAG_FLTy_TM EOF</i>	阈值监视器事件标志
<i>HPDF_FLAG_FLTy_IC PF</i>	注入转换正在进行标志
<i>HPDF_FLAG_FLTy_RC PF</i>	规则转换正在进行标志
<i>HPDF_FLAG_FLT0_CK LF0</i>	通道0时钟丢失标志
<i>HPDF_FLAG_FLT0_CK LF1</i>	通道1时钟丢失标志
<i>HPDF_FLAG_FLT0_M MF0</i>	通道0故障事件标志
<i>HPDF_FLAG_FLT0_M MF1</i>	通道1故障事件标志
<i>HPDF_FLAG_FLTy_RC HPDT</i>	规则通道等待处理数据
<i>HPDF_FLAG_FLTy_LT F0</i>	通道0阈值监视器下限阈值标志
<i>HPDF_FLAG_FLTy_LT F1</i>	通道1阈值监视器下限阈值标志
<i>HPDF_FLAG_FLTy_HT F0</i>	通道0阈值监视器上限阈值标志
<i>HPDF_FLAG_FLTy_HT F1</i>	通道1阈值监视器上限阈值标志
输出参数{out}	
-	-
返回值	
FlagStatus	SET或RESET

例如：

```
/* get the FLT0 threshold monitor event occurred flag */
```

```
hpdf_flag_get(FTL0, HPDF_FLAG_FLTy_TMEOF);
```

### 函数 hpdf\_flag\_clear

函数hpdf\_flag\_clear描述见下表：

表 3-418. 函数 hpdf\_flag\_clear

函数名称	hpdf_flag_clear
函数原型	void hpdf_flag_clear(hpdf_filter_enum filtery, hpdf_flag_enum flag);

功能描述	清楚HPDF标志位
先决条件	-
被调用函数	-
输入参数{in}	
filtery	HPDF模块滤波器
FLTy(y=0..1)	HPDF滤波器选择, 参考 <a href="#">表3-332. 枚举类型hpdf_filter_enum</a>
输入参数{in}	
flag	HPDF模块标志位
HPDF_FLAG_FLTy_IC EF	注入转换结束标志
HPDF_FLAG_FLTy_RC EF	规则转换结束标志
HPDF_FLAG_FLTy_IC DOF	注入转换溢出标志
HPDF_FLAG_FLTy_RC DOF	规则转换溢出标志
HPDF_FLAG_FLTy_TM EOF	阈值监视器事件标志
HPDF_FLAG_FLT0_CK LF0	通道0时钟丢失标志
HPDF_FLAG_FLT0_CK LF1	通道1时钟丢失标志
HPDF_FLAG_FLT0_M MF0	通道0故障事件标志
HPDF_FLAG_FLT0_M MF1	通道1故障事件标志
HPDF_FLAG_FLTy_LT F0	通道0阈值监视器下限阈值标志
HPDF_FLAG_FLTy_LT F1	通道1阈值监视器下限阈值标志
HPDF_FLAG_FLTy_HT F0	通道0阈值监视器上限阈值标志
HPDF_FLAG_FLTy_HT F1	通道1阈值监视器上限阈值标志
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* clear the FLT0 threshold monitor event occurred flag */
hpdf_flag_clear(FTL0, HPDF_FLAG_FLTy_TMEOF);
```

**函数 hpdf\_interrupt\_enable**

函数hpdf\_interrupt\_enable描述见下表：

**表 3-419. 函数 hpdf\_interrupt\_enable**

函数名称	hpdf_interrupt_enable
函数原型	void hpdf_interrupt_enable(hpdf_filter_enum filtery, hpdf_interrput_enum interrupt);
功能描述	使能HPDF中断
先决条件	-
被调用函数	-
输入参数{in}	
filtery	HPDF模块滤波器
FLTy(y=0..1)	HPDF滤波器选择，参考 <a href="#">表3-332. 枚举类型hpdf_filter_enum</a>
输入参数{in}	
interrupt	HPDF模块中断
HPDF_INT_FLTy_ICEI E	使能注入转换结束中断
HPDF_INT_FLTy_RCEI E	使能规则转换结束中断
HPDF_INT_FLTy_ICD OIE	使能注入转换溢出中断
HPDF_INT_FLTy_RCD OIE	使能规则转换溢出中断
HPDF_INT_FLTy_TMIE	使能阈值监视器事件中断
HPDF_INT_FLT0_MMI E	使能故障监视中断
HPDF_INT_FLT0_CKLI E	使能时钟丢失中断
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* enable FLT0 threshold monitor interrupt */
```

```
hpdf_interrupt_enable(FTL0, HPDF_INT_FLTy_TMEIOIE);
```

**函数 hpdf\_interrupt\_disable**

函数hpdf\_interrupt\_disable描述见下表：

**表 3-420. 函数 hpdf\_interrupt\_disable**

函数名称	hpdf_interrupt_disable
------	------------------------

函数原型	void hpdf_interrupt_disable(hpdf_filter_enum filtery, hpdf_interrput_enum interrupt);
功能描述	禁止HPDF中断
先决条件	-
被调用函数	-
输入参数{in}	
filtery	HPDF模块滤波器
FLTy(y=0..1)	HPDF滤波器选择, 参考 <a href="#">表3-332. 枚举类型hpdf_filter_enum</a>
输入参数{in}	
interrupt	HPDF模块中断
HPDF_INT_FLTy_ICEI E	使能注入转换结束中断
HPDF_INT_FLTy_RCEI E	使能规则转换结束中断
HPDF_INT_FLTy_ICD OIE	使能注入转换溢出中断
HPDF_INT_FLTy_RCD OIE	使能规则转换溢出中断
HPDF_INT_FLTy_TMIE	使能阈值监视器事件中断
HPDF_INT_FLT0_MMI E	使能故障监视中断
HPDF_INT_FLT0_CKLI E	使能时钟丢失中断
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* disable FLT0 threshold monitor interrupt */
```

```
hpdf_interrupt_disable(FTL0, HPDF_INT_FLT0_CKLIIE);
```

### 函数 hpdf\_interrupt\_flag\_get

函数hpdf\_interrupt\_flag\_get描述见下表:

表 3-421. 函数 hpdf\_interrupt\_flag\_get

函数名称	hpdf_interrupt_flag_get
函数原型	FlagStatus hpdf_interrupt_flag_get(hpdf_filter_enum filtery, hpdf_interrput_flag_enum int_flag);
功能描述	获取HPDF中断标志位
先决条件	-
被调用函数	-



输入参数{in}	
<b>filtery</b>	HPDF模块滤波器
<i>FLTy(y=0..1)</i>	HPDF滤波器选择, 参考 <a href="#">表3-332. 枚举类型hpdf_filter_enum</a>
输入参数{in}	
<b>int_flag</b>	HPDF模块中断标志位
HPDF_INT_FLAG_FLT y_ICEF	注入转换结束中断标志
HPDF_INT_FLAG_FLT y_RCEF	规则转换结束中断标志
HPDF_INT_FLAG_FLT y_ICDOF	注入转换溢出中断标志
HPDF_INT_FLAG_FLT y_RCDOF	规则转换溢出中断标志
HPDF_INT_FLAG_FLT y_TMEOF	阈值监视器事件中断标志
HPDF_INT_FLAG_FLT 0_CKLF0	通道0时钟丢失中断标志
HPDF_INT_FLAG_FLT 0_CKLF1	通道1时钟丢失中断标志
HPDF_INT_FLAG_FLT 0_MMF0	通道0故障事件中断标志
HPDF_INT_FLAG_FLT 0_MMF1	通道1故障事件中断标志
输出参数{out}	
-	-
返回值	
<b>FlagStatus</b>	SET或RESET

例如:

```
/* get the clock loss interrupt flag */
```

```
hpdf_interrupt_flag_get (FTL0, HPDF_INT_FLAG_FLT0_CKLF0);
```

### 函数 hpdf\_interrupt\_flag\_clear

函数hpdf\_interrupt\_flag\_clear描述见下表:

表 3-422. 函数 hpdf\_interrupt\_flag\_clear

函数名称	hpdf_interrupt_flag_clear
函数原型	void hpdf_interrupt_flag_clear(hpdf_filter_enum filtery, hpdf_interrput_flag_enum int_flag);
功能描述	清楚HPDF中断标志位
先决条件	-
被调用函数	-

输入参数{in}	
<b>filtery</b>	HPDF模块滤波器
<i>FLTy(y=0..1)</i>	HPDF滤波器选择, 参考 <a href="#">表3-332. 枚举类型hpdf_filter_enum</a>
输入参数{in}	
<b>int_flag</b>	HPDF模块中断标志位
HPDF_INT_FLAG_FLT y_ICEF	注入转换结束中断标志
HPDF_INT_FLAG_FLT y_RCEF	规则转换结束中断标志
HPDF_INT_FLAG_FLT y_ICDOF	注入转换溢出中断标志
HPDF_INT_FLAG_FLT y_RCDOF	规则转换溢出中断标志
HPDF_INT_FLAG_FLT y_TMEOF	阈值监视器事件中断标志
HPDF_INT_FLAG_FLT 0_CKLF0	通道0时钟丢失中断标志
HPDF_INT_FLAG_FLT 0_CKLF1	通道1时钟丢失中断标志
HPDF_INT_FLAG_FLT 0_MMF0	通道0故障事件中断标志
HPDF_INT_FLAG_FLT 0_MMF1	通道1故障事件中断标志
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* clear the clock loss interrupt flag */
```

```
hpdf_interrupt_flag_get(FTL0, HPDF_INT_FLAG_FLT0_CKLF0);
```

## 3.15. 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[3.15.1](#)描述了I2C的寄存器列表，章节[3.15.2](#)对I2C库函数进行说明。

### 3.15.1. 外设寄存器说明

I2C寄存器列表如下表所示：

表 3-423. I2C 寄存器

寄存器名称	寄存器描述
I2C_CTL0	控制寄存器 0
I2C_CTL1	控制寄存器 1
I2C_SADDR0	从机地址寄存器 0
I2C_SADDR1	从机地址寄存器 1
I2C_TIMING	时序寄存器
I2C_TIMEOUT	超时寄存器
I2C_STAT	状态寄存器
I2C_STATC	状态清除寄存器
I2C_PEC	PEC 寄存器
I2C_RDATA	接收数据寄存器
I2C_TDATA	发送数据寄存器
I2C_CTL2	控制寄存器 2

### 3.15.2. 外设库函数说明

I2C库函数列表如下表所示：

表 3-424. I2C 库函数

库函数名称	库函数描述
i2c_deinit	复位外设 I2C
i2c_timing_config	配置时序参数
i2c_digital_noise_filter_config	配置数字噪声过滤器
i2c_analog_noise_filter_enable	使能数字噪声过滤器
i2c_analog_noise_filter_disable	禁能数字噪声过滤器
i2c_master_clock_config	配置主机模式下 SCL 高低电平时钟周期
i2c_master_addressing	配置 I2C 从机地址以及数据传输方向
i2c_address10_header_enable	主机接收模式下，10 位地址头只执行读操作
i2c_address10_header_disable	主机接收模式下，10 位地址头执行完整的读操作序列
i2c_address10_enable	使能主机模式下 10 位地址寻址模式
i2c_address10_disable	禁能主机模式下 10 位地址寻址模式
i2c_automatic_end_enable	使能主机模式下 I2C 自动结束模式
i2c_automatic_end_disable	禁能主机模式下 I2C 自动结束模式
i2c_slave_response_to_gcall_enable	使能从机响应广播呼叫
i2c_slave_response_to_gcall_disable	禁能从机响应广播呼叫
i2c_stretch_scl_low_enable	当从机数据没有准备好时拉低 SCL
i2c_stretch_scl_low_disable	当从机数据没有准备好时不拉低 SCL
i2c_address_config	配置 I2C 从机地址
i2c_address_bit_compare_config	定义 ADDRESS[7:1]的哪些位和接收到的地址进行比较
i2c_address_disable	禁能从机模式下 I2C 地址
i2c_second_address_config	配置 I2C 从机第二个地址
i2c_second_address_disable	禁能 I2C 从机第二个地址

库函数名称	库函数描述
i2c_receved_address_get	获取从机模式下匹配成功的地址
i2c_slave_byte_control_enable	使能从机字节控制
i2c_slave_byte_control_disable	禁能从机字节控制
i2c_nack_enable	从机模式下产生 NACK
i2c_nack_disable	从机模式下产生 ACK
i2c_wakeup_from_deepsleep_enable	使能从 Deep-sleep 模式中唤醒
i2c_wakeup_from_deepsleep_disable	禁止从 Deep-sleep 模式中唤醒
i2c_enable	使能 I2C
i2c_disable	禁能 I2C
i2c_start_on_bus	在 I2C 总线上生成起始位
i2c_stop_on_bus	在 I2C 总线上生成停止位
i2c_data_transmit	发送数据
i2c_data_receive	接收数据
i2c_reload_enable	使能 I2C 重载模式
i2c_reload_disable	禁能 I2C 重载模式
i2c_transfer_byte_number_config	配置待发送字节数
i2c_dma_enable	使能发送/接收模式下 DMA
i2c_dma_disable	禁能发送/接收模式下 DMA
i2c_pec_transfer	I2C 传输 PEC 值
i2c_pec_enable	使能报文错误校验
i2c_pec_disable	禁能报文错误校验
i2c_pec_value_get	获取报文错误校验值
i2c_smbus_alert_enable	使能 SMBus 报警
i2c_smbus_alert_disable	禁能 SMBus 报警
i2c_smbus_default_addr_enable	使能 SMBus 设备默认地址
i2c_smbus_default_addr_disable	禁能 SMBus 设备默认地址
i2c_smbus_host_addr_enable	使能 SMBus 主机地址
i2c_smbus_host_addr_disable	禁能 SMBus 主机地址
i2c_extented_clock_timeout_enable	使能时钟信号延展超时检测
i2c_extented_clock_timeout_disable	禁能时钟信号延展超时检测
i2c_clock_timeout_enable	使能时钟超时检测
i2c_clock_timeout_disable	禁能时钟超时检测
i2c_bus_timeout_b_config	配置总线超时 B
i2c_bus_timeout_a_config	配置总线超时 A
i2c_idle_clock_timeout_config	配置空闲时钟超时检测
i2c_flag_get	获取 I2C 标志位
i2c_flag_clear	清除 I2C 标志位
i2c_interrupt_enable	中断使能
i2c_interrupt_disable	中断除能
i2c_interrupt_flag_get	获取中断标志位
i2c_interrupt_flag_clear	清除中断标志位

枚举类型 `i2c_interrupt_flag_enum`表 3-425. 枚举类型 `i2c_interrupt_flag_enum`

成员名称	功能描述
<code>I2C_INT_FLAG_TI</code>	发送中断标志
<code>I2C_INT_FLAG_RBNE</code>	接收期间I2C_RDATA非空中断标志
<code>I2C_INT_FLAG_ADDSEND</code>	从机模式下，接收到的地址与自身地址匹配中断标志
<code>I2C_INT_FLAG_NACK</code>	NACK中断标志
<code>I2C_INT_FLAG_STPDET</code>	从机模式下检测到STOP信号中断标志
<code>I2C_INT_FLAG_TC</code>	主机模式下传输完成中断标志
<code>I2C_INT_FLAG_TCR</code>	传输完成重载中断标志
<code>I2C_INT_FLAG_BERR</code>	总线错误中断标志
<code>I2C_INT_FLAG_LOSTARB</code>	仲裁丢失中断标志
<code>I2C_INT_FLAG_OUERR</code>	从机模式下，过载/欠载错误中断标志
<code>I2C_INT_FLAG_PECERR</code>	PEC错误中断标志
<code>I2C_INT_FLAG_TIMEOUT</code>	超时中断标志
<code>I2C_INT_FLAG_SMBALT</code>	SMBus报警中断标志

函数 `i2c_deinit`

函数*i2c\_deinit*描述见下表：

表 3-426. 函数 `i2c_deinit`

函数名称	<code>i2c_deinit</code>
函数原型	<code>void i2c_deinit(uint32_t i2c_periph);</code>
功能描述	复位外设 I2C
先决条件	-
被调用函数	<code>rcu_periph_reset_enable</code> / <code>rcu_periph_reset_disable</code>
输入参数{in}	
<code>i2c_periph</code>	I2C 外设
<code>I2Cx</code>	(x=0,1)
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* reset I2C0 */
```

```
i2c_deinit(I2C0);
```

函数 `i2c_timing_config`

函数*i2c\_timing\_config*描述见下表：

表 3-427. 函数 i2c\_timing\_config

函数名称	i2c_timing_config
函数原型	void i2c_timing_config(uint32_t i2c_periph, uint32_t psc, uint32_t scl_dely, uint32_t sda_dely);
功能描述	配置时序参数
先决条件	-
被调用函数	-
输入参数{in}	
i2c_periph	I2C 外设
I2Cx	(x=0,1)
输入参数{in}	
psc	0-0xf, 时序分频
输入参数{in}	
scl_dely	0-0xf, 数据建立时间
输入参数{in}	
sda_dely	0-0xf, 数据保持时间
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* configure the timing parameters */
```

```
i2c_timing_config(I2C0, 0x1, 0x2, 0x1);
```

### 函数 i2c\_digital\_noise\_filter\_config

函数i2c\_digital\_noise\_filter\_config描述见下表:

表 3-428. 函数 i2c\_digital\_noise\_filter\_config

函数名称	i2c_digital_noise_filter_config
函数原型	void i2c_digital_noise_filter_config(uint32_t i2c_periph, uint32_t filter_length);
功能描述	配置数字噪声过滤器
先决条件	-
被调用函数	-
输入参数{in}	
i2c_periph	I2C 外设
I2Cx	(x=0,1)
输入参数{in}	
filter_length	过滤长度
FILTER_DISABLE	数字噪声过滤器禁能
FILTER_LENGTH_1	数字噪声滤波使能并且可以滤除脉宽宽度不大于 1 t <sub>12CCLK</sub> 的尖峰

<i>FILTER_LENGTH_2</i>	数字噪声滤波使能并且可以滤除脉宽宽度不大于 2 $t_{I2CCLK}$ 的尖峰
<i>FILTER_LENGTH_3</i>	数字噪声滤波使能并且可以滤除脉宽宽度不大于 3 $t_{I2CCLK}$ 的尖峰
<i>FILTER_LENGTH_4</i>	数字噪声滤波使能并且可以滤除脉宽宽度不大于 4 $t_{I2CCLK}$ 的尖峰
<i>FILTER_LENGTH_5</i>	数字噪声滤波使能并且可以滤除脉宽宽度不大于 5 $t_{I2CCLK}$ 的尖峰
<i>FILTER_LENGTH_6</i>	数字噪声滤波使能并且可以滤除脉宽宽度不大于 6 $t_{I2CCLK}$ 的尖峰
<i>FILTER_LENGTH_7</i>	数字噪声滤波使能并且可以滤除脉宽宽度不大于 7 $t_{I2CCLK}$ 的尖峰
<i>FILTER_LENGTH_8</i>	数字噪声滤波使能并且可以滤除脉宽宽度不大于 8 $t_{I2CCLK}$ 的尖峰
<i>FILTER_LENGTH_9</i>	数字噪声滤波使能并且可以滤除脉宽宽度不大于 9 $t_{I2CCLK}$ 的尖峰
<i>FILTER_LENGTH_10</i>	数字噪声滤波使能并且可以滤除脉宽宽度不大于 10 $t_{I2CCLK}$ 的尖峰
<i>FILTER_LENGTH_11</i>	数字噪声滤波使能并且可以滤除脉宽宽度不大于 11 $t_{I2CCLK}$ 的尖峰
<i>FILTER_LENGTH_12</i>	数字噪声滤波使能并且可以滤除脉宽宽度不大于 12 $t_{I2CCLK}$ 的尖峰
<i>FILTER_LENGTH_13</i>	数字噪声滤波使能并且可以滤除脉宽宽度不大于 13 $t_{I2CCLK}$ 的尖峰
<i>FILTER_LENGTH_14</i>	数字噪声滤波使能并且可以滤除脉宽宽度不大于 14 $t_{I2CCLK}$ 的尖峰
<i>FILTER_LENGTH_15</i>	数字噪声滤波使能并且可以滤除脉宽宽度不大于 15 $t_{I2CCLK}$ 的尖峰
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* I2C0 digital filter filters spikes with a length of up to 1 tI2CCLK */
```

```
i2c_digital_noise_filter_config(I2C0, FILTER_LENGTH_1);
```

### 函数 i2c\_analog\_noise\_filter\_enable

函数 i2c\_analog\_noise\_filter\_enable 描述见下表：

表 3-429. 函数 i2c\_analog\_noise\_filter\_enable

函数名称	i2c_analog_noise_filter_enable
函数原型	void i2c_analog_noise_filter_enable(uint32_t i2c_periph);
功能描述	使能模拟噪声滤波器
先决条件	-
被调用函数	-
输入参数{in}	
<b>i2c_periph</b>	I2C外设
<i>I2Cx</i>	(x=0,1)
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* enable analog noise filter */
```

```
i2c_analog_noise_filter_enable(I2C0);
```

### 函数 i2c\_analog\_noise\_filter\_disable

函数i2c\_analog\_noise\_filter\_disable描述见下表：

表 3-430. 函数 i2c\_analog\_noise\_filter\_disable

函数名称	i2c_analog_noise_filter_disable
函数原型	void i2c_analog_noise_filter_disable(uint32_t i2c_periph);
功能描述	禁能模拟噪声滤波器
先决条件	-
被调用函数	-
输入参数{in}	
i2c_periph	I2C外设
I2Cx	(x=0,1)
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* disable analog noise filter */
```

```
i2c_analog_noise_filter_disable(I2C0);
```

### 函数 i2c\_master\_clock\_config

函数i2c\_master\_clock\_config描述见下表：

表 3-431. 函数 i2c\_master\_clock\_config

函数名称	i2c_master_clock_config
函数原型	void i2c_master_clock_config(uint32_t i2c_periph, uint32_t sclh, uint32_t scll);
功能描述	配置主机模式下 SCL 高低电平周期
先决条件	-
被调用函数	-
输入参数{in}	
i2c_periph	I2C 外设
I2Cx	(x=0,1)
输入参数{in}	
sclh	0-0xff, SCL 高电平周期
输入参数{in}	
scll	0-0xff, SCL 低电平周期
输出参数{out}	
-	-



返回值	
-	-

例如：

```
/* configure the SCL and SDA period of clock in master mode */
```

```
i2c_master_clock_config(I2C0, 0x0f, 0x0f);
```

### 函数 i2c\_master\_addressing

函数i2c\_master\_addressing描述见下表：

表 3-432. 函数 i2c\_master\_addressing

函数名称	i2c_master_addressing
函数原型	void i2c_master_addressing(uint32_t i2c_periph, uint32_t address, uint32_t trans_direction);
功能描述	配置 I2C 从机地址以及数据传输方向
先决条件	-
被调用函数	-
输入参数{in}	
i2c_periph	I2C 外设
I2Cx	(x=0,1)
输入参数{in}	
address	除保留地址外的地址，0-0x3FF，由主机发送给从机的地址
输入参数{in}	
trans_direction	主机模式下，I2C 传输方向
I2C_MASTER_TRANSMIT	主机发送
I2C_MASTER_RECEIVE	主机接收
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* send slave address to I2C bus */
```

```
i2c_master_addressing(I2C0, 0x82, I2C_MASTER_TRANSMIT);
```

### 函数 i2c\_address10\_header\_enable

函数i2c\_address10\_header\_enable描述见下表：

表 3-433. 函数 i2c\_address10\_header\_enable

函数名称	i2c_address10_header_enable
------	-----------------------------

函数原型	void i2c_address10_header_enable(uint32_t i2c_periph);
功能描述	主机接收模式下，10 位地址头只执行读操作
先决条件	-
被调用函数	-
输入参数{in}	
i2c_periph	I2C 外设
I2Cx	(x=0,1)
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* 10-bit address header executes read direction only in master receive mode */
```

```
i2c_address10_header_enable(I2C0);
```

### 函数 i2c\_address10\_header\_disable

函数i2c\_address10\_header\_disable描述见下表：

表 3-434. 函数 i2c\_address10\_header\_disable

函数名称	i2c_address10_header_disable
函数原型	void i2c_address10_header_disable(uint32_t i2c_periph);
功能描述	主机接收模式下，10 位地址头执行完整的读操作序列
先决条件	-
被调用函数	-
输入参数{in}	
i2c_periph	I2C 外设
I2Cx	(x=0,1)
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* 10-bit address header executes complete sequence in master receive mode */
```

```
i2c_address10_header_disable(I2C0);
```

### 函数 i2c\_address10\_enable

函数i2c\_address10\_enable描述见下表：

表 3-435. 函数 i2c\_address10\_enable

函数名称	i2c_address10_enable
------	----------------------

函数原型	void i2c_address10_enable(uint32_t i2c_periph);
功能描述	使能主机模式下 10 位地址寻址模式
先决条件	-
被调用函数	-
输入参数{in}	
i2c_periph	I2C 外设
I2Cx	(x=0,1)
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* enable 10-bit addressing mode in master mode */
```

```
i2c_address10_enable(I2C0);
```

### 函数 i2c\_address10\_disable

函数i2c\_address10\_disable描述见下表：

表 3-436. 函数 i2c\_address10\_disable

函数名称	i2c_address10_disable
函数原型	void i2c_address10_disable(uint32_t i2c_periph);
功能描述	禁能主机模式下 10 位地址寻址模式
先决条件	-
被调用函数	-
输入参数{in}	
i2c_periph	I2C 外设
I2Cx	(x=0,1)
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* disable 10-bit addressing mode in master mode */
```

```
i2c_address10_disable(I2C0);
```

### 函数 i2c\_automatic\_end\_enable

函数i2c\_automatic\_end\_enable描述见下表：

表 3-437. 函数 i2c\_automatic\_end\_enable

函数名称	i2c_automatic_end_enable
------	--------------------------

函数原型	void i2c_automatic_end_enable(uint32_t i2c_periph);
功能描述	使能主机模式下 I2C 自动结束模式
先决条件	-
被调用函数	-
输入参数{in}	
i2c_periph	I2C 外设
I2Cx	(x=0,1)
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* enable I2C automatic end mode in master mode */
```

```
i2c_automatic_end_enable(I2C0);
```

### 函数 i2c\_automatic\_end\_disable

函数i2c\_automatic\_end\_disable描述见下表：

表 3-438. 函数 i2c\_automatic\_end\_disable

函数名称	i2c_automatic_end_disable
函数原型	void i2c_automatic_end_disable(uint32_t i2c_periph);
功能描述	禁能主机模式下 I2C 自动结束模式
先决条件	-
被调用函数	-
输入参数{in}	
i2c_periph	I2C 外设
I2Cx	(x=0,1)
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* disable I2C automatic end mode in master mode */
```

```
i2c_automatic_end_disable(I2C0);
```

### 函数 i2c\_slave\_response\_to\_gcall\_enable

函数i2c\_slave\_response\_to\_gcall\_enable描述见下表：

表 3-439. 函数 i2c\_slave\_response\_to\_gcall\_enable

函数名称	i2c_slave_response_to_gcall_enable
------	------------------------------------

函数原型	void i2c_slave_response_to_gcall_enable(uint32_t i2c_periph);
功能描述	使能从机响应广播呼叫
先决条件	-
被调用函数	-
输入参数{in}	
i2c_periph	I2C 外设
I2Cx	(x=0,1)
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* enable the response to a general call */
```

```
i2c_slave_response_to_gcall_enable(I2C0);
```

### 函数 i2c\_slave\_response\_to\_gcall\_disable

函数i2c\_slave\_response\_to\_gcall\_disable描述见下表：

表 3-440. 函数 i2c\_slave\_response\_to\_gcall\_disable

函数名称	i2c_slave_response_to_gcall_disable
函数原型	void i2c_slave_response_to_gcall_disable(uint32_t i2c_periph);
功能描述	禁能从机响应广播呼叫
先决条件	-
被调用函数	-
输入参数{in}	
i2c_periph	I2C 外设
I2Cx	(x=0,1)
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* disable the response to a general call */
```

```
i2c_slave_response_to_gcall_disable(I2C0);
```

### 函数 i2c\_stretch\_scl\_low\_enable

函数i2c\_stretch\_scl\_low\_enable描述见下表：

表 3-441. 函数 i2c\_stretch\_scl\_low\_enable

函数名称	i2c_stretch_scl_low_enable
------	----------------------------

函数原型	void i2c_stretch_scl_low_enable(uint32_t i2c_periph);
功能描述	当从机数据没有准备好时拉低 SCL
先决条件	-
被调用函数	-
输入参数{in}	
i2c_periph	I2C 外设
I2Cx	(x=0,1)
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* enable to stretch SCL low when data is not ready in slave mode */
```

```
i2c_stretch_scl_low_enable(I2C0);
```

### 函数 i2c\_stretch\_scl\_low\_disable

函数i2c\_stretch\_scl\_low\_disable描述见下表：

表 3-442. 函数 i2c\_stretch\_scl\_low\_disable

函数名称	i2c_stretch_scl_low_disable
函数原型	void i2c_stretch_scl_low_disable(uint32_t i2c_periph);
功能描述	当从机数据没有准备好时不拉低 SCL
先决条件	-
被调用函数	-
输入参数{in}	
i2c_periph	I2C 外设
I2Cx	(x=0,1)
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* disable to stretch SCL low when data is not ready in slave mode */
```

```
i2c_stretch_scl_low_disable(I2C0);
```

### 函数 i2c\_address\_config

函数i2c\_address\_config描述见下表：

表 3-443. 函数 i2c\_address\_config

函数名称	i2c_address_config
------	--------------------

函数原型	void i2c_address_config(uint32_t i2c_periph, uint32_t address, uint32_t addr_format);
功能描述	配置 I2C 从机地址
先决条件	-
被调用函数	-
输入参数{in}	
i2c_periph	I2C 外设
I2Cx	(x=0,1)
输入参数{in}	
address	I2C 地址
输入参数{in}	
addr_format	7 位地址或 10 位地址
I2C_ADDFORMAT_7BITS	7 位地址
I2C_ADDFORMAT_10BITS	10 位地址
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* configure i2c slave address */
```

```
i2c_address_config (I2C0, 0x82, I2C_ADDFORMAT_7BITS);
```

### 函数 i2c\_address\_bit\_compare\_config

函数 i2c\_address\_bit\_compare\_config 描述见下表:

表 3-444. 函数 i2c\_address\_bit\_compare\_config

函数名称	i2c_address_bit_compare_config
函数原型	void i2c_address_bit_compare_config(uint32_t i2c_periph, uint32_t compare_bits);
功能描述	定义 ADDRESS[7:1] 的哪些位和接收到的地址进行比较
先决条件	-
被调用函数	-
输入参数{in}	
i2c_periph	I2C peripheral
I2Cx	(x=0,1)
输入参数{in}	
compare_bits	需要进行比较的位
ADDRESS_BIT1_COMPARE	地址的第 1 位需要进行比较

ADDRESS_BIT2_COM PARE	地址的第 2 位需要进行比较
ADDRESS_BIT3_COM PARE	地址的第 3 位需要进行比较
ADDRESS_BIT4_COM PARE	地址的第 4 位需要进行比较
ADDRESS_BIT5_COM PARE	地址的第 5 位需要进行比较
ADDRESS_BIT6_COM PARE	地址的第 6 位需要进行比较
ADDRESS_BIT7_COM PARE	地址的第 7 位需要进行比较
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* define which bits of ADDRESS[7:1] need to compare with the incoming address byte */
```

```
i2c_address_bit_compare_config(I2C0, ADDRESS_BIT1_COMPARE);
```

### 函数 i2c\_address\_disable

函数i2c\_address\_disable描述见下表：

表 3-445. 函数 i2c\_address\_disable

函数名称	i2c_address_disable
函数原型	void i2c_address_disable(uint32_t i2c_periph);
功能描述	禁能从机模式下 I2C 地址
先决条件	-
被调用函数	-
输入参数{in}	
i2c_periph	I2C 外设
I2Cx	(x=0,1)
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* disable i2c address in slave mode */
```

```
i2c_address_disable(I2C0);
```



## 函数 i2c\_second\_address\_config

函数i2c\_second\_address\_config描述见下表:

表 3-446. 函数 i2c\_second\_address\_config

函数名称	i2c_second_address_config
函数原型	void i2c_second_address_config(uint32_t i2c_periph, uint32_t address, uint32_t addr_mask);
功能描述	配置 I2C 从机第二个地址
先决条件	-
被调用函数	-
输入参数{in}	
i2c_periph	I2C 外设
I2Cx	(x=0,1)
输入参数{in}	
address	I2C 地址
输入参数{in}	
addr_mask	不需要进行比较的地址
ADDRESS2_NO_MASK	无屏蔽, 全部都需要进行比较
ADDRESS2_MASK_BIT1	ADDRESS2[1]屏蔽, ADDRESS2[7:2]进行比较
ADDRESS2_MASK_BIT1_2	ADDRESS2[2:1]屏蔽, ADDRESS2[7:3]进行比较
ADDRESS2_MASK_BIT1_3	ADDRESS2[3:1]屏蔽, ADDRESS2[7:4]进行比较
ADDRESS2_MASK_BIT1_4	ADDRESS2[4:1]屏蔽, ADDRESS2[7:5]进行比较
ADDRESS2_MASK_BIT1_5	ADDRESS2[5:1]屏蔽, ADDRESS2[7:6]进行比较
ADDRESS2_MASK_BIT1_6	ADDRESS2[6:1]屏蔽, ADDRESS2[7]进行比较
ADDRESS2_MASK_ALL	ADDRESS2[7:1]屏蔽
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* configure i2c second slave address */
```

```
i2c_second_address_config(I2C0, 0x82, ADDRESS2_MASK_BIT1_2);
```

## 函数 i2c\_second\_address\_disable

函数i2c\_second\_address\_disable描述见下表：

表 3-447. 函数 i2c\_second\_address\_disable

函数名称	i2c_second_address_disable
函数原型	void i2c_second_address_disable(uint32_t i2c_periph);
功能描述	禁能 I2C 从机第二个地址
先决条件	-
被调用函数	-
输入参数{in}	
i2c_periph	I2C 外设
I2Cx	(x=0,1)
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* disable i2c second address in slave mode */
```

```
i2c_second_address_disable(I2C0);
```

## 函数 i2c\_receved\_address\_get

函数i2c\_receved\_address\_get描述见下表：

表 3-448. 函数 i2c\_receved\_address\_get

Table 3-1. Function i2c\_receved\_address\_get

函数名称	i2c_receved_address_get
函数原型	uint32_t i2c_receved_address_get(uint32_t i2c_periph);
功能描述	获取从机模式下匹配成功的地址
先决条件	-
被调用函数	-
输入参数{in}	
i2c_periph	I2C 外设
I2Cx	(x=0,1)
输出参数{out}	
-	-
返回值	
uint32_t	0x00..0x7F

例如：

```
/* get received match address in slave mode */
```

```
uint32_t address;
```

```
address = i2c_receivied_address_get(I2C0);
```

### 函数 i2c\_slave\_byte\_control\_enable

函数i2c\_slave\_byte\_control\_enable描述见下表：

表 3-449. 函数 i2c\_slave\_byte\_control\_enable

函数名称	i2c_slave_byte_control_enable
函数原型	void i2c_slave_byte_control_enable(uint32_t i2c_periph);
功能描述	使能从机字节控制
先决条件	-
被调用函数	-
输入参数{in}	
i2c_periph	I2C 外设
I2Cx	(x=0,1)
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* enable slave byte control */
```

```
i2c_slave_byte_control_enable(I2C0);
```

### 函数 i2c\_slave\_byte\_control\_disable

函数i2c\_slave\_byte\_control\_disable描述见下表：

表 3-450. 函数 i2c\_slave\_byte\_control\_disable

函数名称	i2c_slave_byte_control_disable
函数原型	void i2c_slave_byte_control_disable(uint32_t i2c_periph);
功能描述	禁能从机字节控制
先决条件	-
被调用函数	-
输入参数{in}	
i2c_periph	I2C 外设
I2Cx	(x=0,1)
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* disable slave byte control */
```

```
i2c_slave_byte_control_disable(I2C0);
```

### 函数 i2c\_nack\_enable

函数i2c\_nack\_enable描述见下表:

表 3-451. 函数 i2c\_nack\_enable

函数名称	i2c_nack_enable
函数原型	void i2c_nack_enable(uint32_t i2c_periph);
功能描述	从机模式下产生 NACK
先决条件	-
被调用函数	-
输入参数{in}	
i2c_periph	I2C 外设
I2Cx	(x=0,1)
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* generate a NACK in slave mode */
```

```
i2c_nack_enable(I2C0);
```

### 函数 i2c\_nack\_disable

函数i2c\_nack\_disable描述见下表:

表 3-452. 函数 i2c\_nack\_disable

函数名称	i2c_nack_disable
函数原型	void i2c_nack_disable(uint32_t i2c_periph);
功能描述	从机模式下产生 ACK
先决条件	-
被调用函数	-
输入参数{in}	
i2c_periph	I2C 外设
I2Cx	(x=0,1)
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* generate a ACK in slave mode */
```

```
i2c_nack_disable(I2C0);
```

### 函数 i2c\_wakeup\_from\_deepsleep\_enable

函数i2c\_wakeup\_from\_deepsleep\_enable描述见下表：

**表 3-453. 函数 i2c\_wakeup\_from\_deepsleep\_enable**

函数名称	i2c_wakeup_from_deepsleep_enable
函数原型	void i2c_wakeup_from_deepsleep_enable(uint32_t i2c_periph);
功能描述	使能从Deep-sleep模式中唤醒
先决条件	-
被调用函数	-
输入参数{in}	
i2c_periph	I2C外设
I2Cx	(x=0)
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* enable wakeup from Deep-sleep mode */
```

```
i2c_wakeup_from_deepsleep_enable(I2C0);
```

### 函数 i2c\_wakeup\_from\_deepsleep\_disable

函数i2c\_wakeup\_from\_deepsleep\_disable描述见下表：

**表 3-454. 函数 i2c\_wakeup\_from\_deepsleep\_disable**

函数名称	i2c_wakeup_from_deepsleep_disable
函数原型	void i2c_wakeup_from_deepsleep_disable(uint32_t i2c_periph);
功能描述	禁止从Deep-sleep模式中唤醒
先决条件	-
被调用函数	-
输入参数{in}	
i2c_periph	I2C外设
I2Cx	(x=0)
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* disable wakeup from Deep-sleep mode */
```

```
i2c_wakeup_from_deepsleep_disable(I2C0);
```

### 函数 i2c\_enable

函数i2c\_enable描述见下表：

表 3-455. 函数 i2c\_enable

函数名称	i2c_enable
函数原型	void i2c_enable(uint32_t i2c_periph);
功能描述	使能 I2C
先决条件	-
被调用函数	-
输入参数{in}	
i2c_periph	I2C 外设
I2Cx	(x=0,1)
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* enable I2C0 */
```

```
i2c_enable(I2C0);
```

### 函数 i2c\_disable

函数i2c\_disable描述见下表：

表 3-456. 函数 i2c\_disable

函数名称	i2c_disable
函数原型	void i2c_disable(uint32_t i2c_periph);
功能描述	禁能 I2C
先决条件	-
被调用函数	-
输入参数{in}	
i2c_periph	I2C 外设
I2Cx	(x=0,1)
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* disable I2C0 */
```

```
i2c_disable(I2C0);
```

### 函数 i2c\_start\_on\_bus

函数i2c\_start\_on\_bus描述见下表:

表 3-457. 函数 i2c\_start\_on\_bus

函数名称	i2c_start_on_bus
函数原型	void i2c_start_on_bus(uint32_t i2c_periph);
功能描述	在 I2C 总线上生成起始位
先决条件	-
被调用函数	-
输入参数{in}	
i2c_periph	I2C 外设
I2Cx	(x=0,1)
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* I2C0 send a start condition to I2C bus */
```

```
i2c_start_on_bus(I2C0);
```

### 函数 i2c\_stop\_on\_bus

函数i2c\_stop\_on\_bus描述见下表:

表 3-458. 函数 i2c\_stop\_on\_bus

函数名称	i2c_stop_on_bus
函数原型	void i2c_stop_on_bus(uint32_t i2c_periph);
功能描述	在 I2C 总线上生成停止位
先决条件	-
被调用函数	-
输入参数{in}	
i2c_periph	I2C 外设
I2Cx	(x=0,1)
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* I2C0 generate a STOP condition to I2C bus */
```

```
i2c_stop_on_bus(I2C0);
```

### 函数 i2c\_data\_transmit

函数i2c\_data\_transmit描述见下表:

表 3-459. 函数 i2c\_data\_transmit

函数名称	i2c_data_transmit
函数原型	void i2c_data_transmit(uint32_t i2c_periph, uint32_t data);
功能描述	发送数据
先决条件	-
被调用函数	-
输入参数{in}	
i2c_periph	I2C 外设
I2Cx	(x=0,1)
输入参数{in}	
data	transmit data
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* I2C0 transmit data */
```

```
i2c_data_transmit(I2C0, 0x80);
```

### 函数 i2c\_data\_receive

函数i2c\_data\_receive描述见下表:

表 3-460. 函数 i2c\_data\_receive

函数名称	i2c_data_receive
函数原型	uint32_t i2c_data_receive(uint32_t i2c_periph);
功能描述	接收数据
先决条件	-
被调用函数	-
输入参数{in}	
i2c_periph	I2C 外设
I2Cx	(x=0,1)
输出参数{out}	
-	-
返回值	
uint32_t	0x0000..0x00FF



例如：

```
/* I2C0 receive data */

uint32_t i2c_receiver;

i2c_receiver = i2c_data_receive(I2C0);
```

### 函数 i2c\_reload\_enable

函数i2c\_reload\_enable描述见下表：

表 3-461. 函数 i2c\_reload\_enable

函数名称	i2c_reload_enable
函数原型	void i2c_reload_enable(uint32_t i2c_periph);
功能描述	使能 I2C 重载模式
先决条件	-
被调用函数	-
输入参数{in}	
i2c_periph	I2C 外设
I2Cx	(x=0,1)
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* enable I2C reload mode */

i2c_reload_enable(I2C0);
```

### 函数 i2c\_reload\_disable

函数i2c\_reload\_disable描述见下表：

表 3-462. 函数 i2c\_reload\_disable

函数名称	i2c_reload_disable
函数原型	void i2c_reload_disable(uint32_t i2c_periph);
功能描述	禁能 I2C 重载模式
先决条件	-
被调用函数	-
输入参数{in}	
i2c_periph	I2C 外设
I2Cx	(x=0,1)
输出参数{out}	
-	-
返回值	

-	-
---	---

例如：

```
/* disable I2C reload mode */
```

```
i2c_reload_disable(I2C0);
```

### 函数 i2c\_transfer\_byte\_number\_config

函数i2c\_transfer\_byte\_number\_config描述见下表：

表 3-463. 函数 i2c\_transfer\_byte\_number\_config

函数名称	i2c_transfer_byte_number_config
函数原型	void i2c_transfer_byte_number_config(uint32_t i2c_periph, uint32_t byte_number);
功能描述	配置待发送字节数
先决条件	-
被调用函数	-
输入参数{in}	
i2c_periph	I2C 外设
I2Cx	(x=0,1)
输入参数{in}	
byte_number	0x0-0xFF，待传输的字节数
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* configure number of bytes to be transferred */
```

```
i2c_transfer_byte_number_config(I2C0, 0xFF);
```

### 函数 i2c\_dma\_enable

函数i2c\_dma\_enable描述见下表：

表 3-464. 函数 i2c\_dma\_enable

函数名称	i2c_dma_enable
函数原型	void i2c_dma_enable(uint32_t i2c_periph, uint8_t dma);
功能描述	使能发送/接收模式下 DMA
先决条件	-
被调用函数	-
输入参数{in}	
i2c_periph	I2C 外设
I2Cx	(x=0,1)

输入参数{in}	
<b>dma</b>	I2C DMA
<i>I2C_DMA_TRANSMIT</i>	采用 DMA 方式发送数据
<i>I2C_DMA_RECEIVE</i>	采用 DMA 方式接收数据
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* enable I2C DMA for transmission or reception */
```

```
i2c_dma_enable(I2C0, I2C_DMA_RECEIVE);
```

### 函数 i2c\_dma\_disable

函数i2c\_dma\_disable描述见下表：

**表 3-465. 函数 i2c\_dma\_disable**

函数名称	i2c_dma_disable
函数原型	void i2c_dma_disable(uint32_t i2c_periph, uint8_t dma);
功能描述	禁用发送/接收模式下 DMA
先决条件	-
被调用函数	-
输入参数{in}	
<b>i2c_periph</b>	I2C 外设
<i>I2Cx</i>	(x=0,1)
输入参数{in}	
<b>dma</b>	I2C DMA
<i>I2C_DMA_TRANSMIT</i>	采用 DMA 方式发送数据
<i>I2C_DMA_RECEIVE</i>	采用 DMA 方式接收数据
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* disable I2C DMA for transmission or reception */
```

```
i2c_dma_disable(I2C0, I2C_DMA_RECEIVE);
```

### 函数 i2c\_pec\_transfer

函数i2c\_pec\_transfer描述见下表：

表 3-466. 函数 i2c\_pec\_transfer

函数名称	i2c_pec_transfer
函数原型	void i2c_pec_transfer(uint32_t i2c_periph);
功能描述	I2C 传输 PEC 值
先决条件	-
被调用函数	-
输入参数{in}	
i2c_periph	I2C 外设
I2Cx	(x=0,1)
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* I2C transfers PEC value */
```

```
i2c_pec_transfer(I2C0);
```

### 函数 i2c\_pec\_enable

函数i2c\_pec\_enable描述见下表：

表 3-467. 函数 i2c\_pec\_enable

函数名称	i2c_pec_enable
函数原型	void i2c_pec_enable(uint32_t i2c_periph);
功能描述	使能报文错误校验
先决条件	-
被调用函数	-
输入参数{in}	
i2c_periph	I2C 外设
I2Cx	(x=0,1)
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* enable I2C PEC calculation */
```

```
i2c_pec_enable(I2C0);
```

### 函数 i2c\_pec\_disable

函数i2c\_pec\_disable描述见下表：

表 3-468. 函数 i2c\_pec\_disable

函数名称	i2c_pec_disable
函数原型	void i2c_pec_disable(uint32_t i2c_periph);
功能描述	禁能报文错误校验
先决条件	-
被调用函数	-
输入参数{in}	
i2c_periph	I2C 外设
I2Cx	(x=0,1)
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* disable I2C PEC calculation */
```

```
i2c_pec_disable(I2C0);
```

### 函数 i2c\_pec\_value\_get

函数i2c\_pec\_value\_get描述见下表：

表 3-469. 函数 i2c\_pec\_value\_get

函数名称	i2c_pec_value_get
函数原型	uint32_t i2c_pec_value_get(uint32_t i2c_periph);
功能描述	获取报文错误校验值
先决条件	-
被调用函数	-
输入参数{in}	
i2c_periph	I2C 外设
I2Cx	(x=0,1)
输出参数{out}	
-	-
返回值	
uint32_t	PEC 值

例如：

```
/* I2C0 get packet error checking value */
```

```
uint32_t pec_value;
```

```
pec_value = i2c_pec_value_get(I2C0);
```

**函数 i2c\_smbus\_alert\_enable**

函数i2c\_smbus\_alert\_enable描述见下表：

**表 3-470. 函数 i2c\_smbus\_alert\_enable**

函数名称	i2c_smbus_alert_enable
函数原型	void i2c_smbus_alert_enable(uint32_t i2c_periph);
功能描述	使能 SMBus 报警
先决条件	-
被调用函数	-
输入参数{in}	
i2c_periph	I2C 外设
I2Cx	(x=0,1)
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* enable SMBus Alert */
```

```
i2c_smbus_alert_enable(I2C0);
```

**函数 i2c\_smbus\_alert\_disable**

函数i2c\_smbus\_alert\_disable描述见下表：

**表 3-471. 函数 i2c\_smbus\_alert\_disable**

函数名称	i2c_smbus_alert_disable
函数原型	void i2c_smbus_alert_disable(uint32_t i2c_periph);
功能描述	禁能 SMBus 报警
先决条件	-
被调用函数	-
输入参数{in}	
i2c_periph	I2C 外设
I2Cx	(x=0,1)
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* disable SMBus Alert */
```

```
i2c_smbus_alert_disable(I2C0);
```

**函数 i2c\_smbus\_default\_addr\_enable**

函数i2c\_smbus\_default\_addr\_enable描述见下表：

**表 3-472. 函数 i2c\_smbus\_default\_addr\_enable**

函数名称	i2c_smbus_default_addr_enable
函数原型	void i2c_smbus_default_addr_enable(uint32_t i2c_periph);
功能描述	使能 SMBus 设备默认地址
先决条件	-
被调用函数	-
输入参数{in}	
i2c_periph	I2C 外设
I2Cx	(x=0,1)
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* enable SMBus device default address */
```

```
i2c_smbus_default_addr_enable(I2C0);
```

**函数 i2c\_smbus\_default\_addr\_disable**

函数i2c\_smbus\_default\_addr\_disable描述见下表：

**表 3-473. 函数 i2c\_smbus\_default\_addr\_disable**

函数名称	i2c_smbus_default_addr_disable
函数原型	void i2c_smbus_default_addr_disable(uint32_t i2c_periph);
功能描述	禁能 SMBus 设备默认地址
先决条件	-
被调用函数	-
输入参数{in}	
i2c_periph	I2C 外设
I2Cx	(x=0,1)
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* disable SMBus device default address */
```

```
i2c_smbus_default_addr_disable(I2C0);
```

**函数 i2c\_smbus\_host\_addr\_enable**

函数i2c\_smbus\_host\_addr\_enable描述见下表：

**表 3-474. 函数 i2c\_smbus\_host\_addr\_enable**

函数名称	i2c_smbus_host_addr_enable
函数原型	void i2c_smbus_host_addr_enable(uint32_t i2c_periph);
功能描述	使能 SMBus 主机地址
先决条件	-
被调用函数	-
输入参数{in}	
i2c_periph	I2C 外设
I2Cx	(x=0,1)
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* enable SMBus Host address */
```

```
i2c_smbus_host_addr_enable(I2C0);
```

**函数 i2c\_smbus\_host\_addr\_disable**

函数i2c\_smbus\_host\_addr\_disable描述见下表：

**表 3-475. 函数 i2c\_smbus\_host\_addr\_disable**

函数名称	i2c_smbus_host_addr_disable
函数原型	void i2c_smbus_host_addr_disable(uint32_t i2c_periph);
功能描述	禁能 SMBus 主机地址
先决条件	-
被调用函数	-
输入参数{in}	
i2c_periph	I2C 外设
I2Cx	(x=0,1)
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* disable SMBus Host address */
```

```
i2c_smbus_host_addr_disable(I2C0);
```



## 函数 i2c\_extented\_clock\_timeout\_enable

函数i2c\_extented\_clock\_timeout\_enable描述见下表：

表 3-476. 函数 i2c\_extented\_clock\_timeout\_enable

函数名称	i2c_extented_clock_timeout_enable
函数原型	void i2c_extented_clock_timeout_enable(uint32_t i2c_periph);
功能描述	使能时钟信号延展超时检测
先决条件	-
被调用函数	-
输入参数{in}	
i2c_periph	I2C 外设
I2Cx	(x=0,1)
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* enable extended clock timeout detection */
```

```
i2c_extented_clock_timeout_enable(I2C0);
```

## 函数 i2c\_extented\_clock\_timeout\_disable

函数i2c\_extented\_clock\_timeout\_disable描述见下表：

表 3-477. 函数 i2c\_extented\_clock\_timeout\_disable

函数名称	i2c_extented_clock_timeout_disable
函数原型	void i2c_extented_clock_timeout_disable(uint32_t i2c_periph);
功能描述	禁能扩展时钟超时检测
先决条件	-
被调用函数	-
输入参数{in}	
i2c_periph	I2C 外设
I2Cx	(x=0,1)
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* disable extended clock timeout detection */
```

```
i2c_extented_clock_timeout_disable(I2C0);
```

**函数 i2c\_clock\_timeout\_enable**

函数i2c\_clock\_timeout\_enable描述见下表：

**表 3-478. 函数 i2c\_clock\_timeout\_enable**

函数名称	i2c_clock_timeout_enable
函数原型	void i2c_clock_timeout_enable(uint32_t i2c_periph);
功能描述	禁能时钟信号延展超时检测
先决条件	-
被调用函数	-
输入参数{in}	
i2c_periph	I2C 外设
I2Cx	(x=0,1)
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* enable clock timeout detection */
```

```
i2c_clock_timeout_enable(I2C0);
```

**函数 i2c\_clock\_timeout\_disable**

函数i2c\_clock\_timeout\_disable描述见下表：

**表 3-479. 函数 i2c\_clock\_timeout\_disable**

函数名称	i2c_clock_timeout_disable
函数原型	void i2c_clock_timeout_disable(uint32_t i2c_periph);
功能描述	禁能时钟超时检测
先决条件	-
被调用函数	-
输入参数{in}	
i2c_periph	I2C 外设
I2Cx	(x=0,1)
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* disable clock timeout detection */
```

```
i2c_clock_timeout_disable(I2C0);
```

**函数 i2c\_bus\_timeout\_b\_config**

函数i2c\_bus\_timeout\_b\_config描述见下表:

**表 3-480. 函数 i2c\_bus\_timeout\_b\_config**

函数名称	i2c_bus_timeout_b_config
函数原型	void i2c_bus_timeout_b_config(uint32_t i2c_periph, uint32_t timeout);
功能描述	配置总线超时 B
先决条件	-
被调用函数	-
输入参数{in}	
i2c_periph	I2C 外设
I2Cx	(x=0,1)
输入参数{in}	
timeout	0-0xffff, 总线超时 B
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* configure bus timeout B */
```

```
i2c_bus_timeout_b_config(I2C0, 0xff);
```

**函数 i2c\_bus\_timeout\_a\_config**

函数i2c\_bus\_timeout\_a\_config描述见下表:

**表 3-481. 函数 i2c\_bus\_timeout\_a\_config**

函数名称	i2c_bus_timeout_a_config
函数原型	void i2c_bus_timeout_a_config(uint32_t i2c_periph, uint32_t timeout);
功能描述	配置总线超时 A
先决条件	-
被调用函数	-
输入参数{in}	
i2c_periph	I2C 外设
I2Cx	(x=0,1)
输入参数{in}	
timeout	0-0xffff, 总线超时 A
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* configure bus timeout A */

i2c_bus_timeout_a_config(I2C0, 0xff);
```

### 函数 i2c\_idle\_clock\_timeout\_config

函数i2c\_idle\_clock\_timeout\_config描述见下表：

表 3-482. 函数 i2c\_idle\_clock\_timeout\_config

函数名称	i2c_idle_clock_timeout_config
函数原型	void i2c_idle_clock_timeout_config(uint32_t i2c_periph, uint32_t timeout);
功能描述	配置空闲时钟超时检测
先决条件	-
被调用函数	-
输入参数{in}	
i2c_periph	I2C 外设
I2Cx	(x=0,1)
输入参数{in}	
timeout	总线超时 A
BUSTOA_DETECT_SCL_LOW	BUSTOA 用于检测 SCL 低电平超时
BUSTOA_DETECT_IDLE	BUSTOA 用于检测总线空闲情况下 SCL 和 SDA 高电平超时
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* configure idle clock timeout detection */

i2c_idle_clock_timeout_config(I2C0, BUSTOA_DETECT_SCL_LOW);
```

### 函数 i2c\_flag\_get

函数i2c\_flag\_get描述见下表：

表 3-483. 函数 i2c\_flag\_get

函数名称	i2c_flag_get
函数原型	FlagStatus i2c_flag_get(uint32_t i2c_periph, uint32_t flag);
功能描述	获取 I2C 标志位
先决条件	-
被调用函数	-
输入参数{in}	

<b>i2c_periph</b>	I2C 外设
<i>I2Cx</i>	(x=0,1)
<b>输入参数{in}</b>	
<b>flag</b>	I2C 标志位
<i>I2C_FLAG_TBE</i>	发送期间 I2C_TDATA 寄存器空标志
<i>I2C_FLAG_TI</i>	发送中断标志
<i>I2C_FLAG_RBNE</i>	接收期间 I2C_RDATA 非空标志
<i>I2C_FLAG_ADDSEND</i>	从机模式下，接收到的地址与自身地址匹配
<i>I2C_FLAG_NACK</i>	NACK 标志
<i>I2C_FLAG_STPDET</i>	从机模式下检测到 STOP 信号
<i>I2C_FLAG_TC</i>	主机模式下传输完成标志
<i>I2C_FLAG_TCR</i>	传输完成重载标志
<i>I2C_FLAG_BERR</i>	总线错误标志
<i>I2C_FLAG_LOSTARB</i>	仲裁丢失标志
<i>I2C_FLAG_OUERR</i>	从机模式下，过载/欠载错误标志
<i>I2C_FLAG_PECERR</i>	PEC 错误标志
<i>I2C_FLAG_TIMEOUT</i>	超时标志
<i>I2C_FLAG_SMBALT</i>	SMBus 报警标志
<i>I2C_FLAG_I2CBSY</i>	忙标志
<i>I2C_FLAG_TR</i>	从机模式下，I2C 作为发送器还是接收器标志
<b>输出参数{out}</b>	
-	-
<b>返回值</b>	
<b>FlagStatus</b>	SET / RESET

例如：

```
/* get I2C flag status */
```

```
FlagStatus flag_state = RESET;
```

```
flag_state = i2c_flag_get(I2C0, I2C_FLAG_TBE);
```

### 函数 i2c\_flag\_clear

函数i2c\_flag\_clear描述见下表：

**表 3-484. 函数 i2c\_flag\_clear**

<b>函数名称</b>	i2c_flag_clear
<b>函数原型</b>	void i2c_flag_clear(uint32_t i2c_periph, uint32_t flag);
<b>功能描述</b>	清除 I2C 标志位
<b>先决条件</b>	-
<b>被调用函数</b>	-
<b>输入参数{in}</b>	
<b>i2c_periph</b>	I2C 外设

I2Cx	(x=0,1)
输入参数{in}	
flag	I2C 标志位
I2C_FLAG_ADDSEND	从机模式下，接收到的地址与自身地址匹配
I2C_FLAG_NACK	NACK 标志
I2C_FLAG_STPDET	从机模式下检测到 STOP 信号
I2C_FLAG_BERR	总线错误标志
I2C_FLAG_LOSTARB	仲裁丢失标志
I2C_FLAG_OUERR	从机模式下，过载/欠载错误标志
I2C_FLAG_PECERR	PEC 错误标志
I2C_FLAG_TIMEOUT	超时标志
I2C_FLAG_SMBALT	SMBus 报警标志
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* clear a bus error flag*/
```

```
i2c_flag_clear(I2C0, I2C_FLAG_BERR);
```

### 函数 i2c\_interrupt\_enable

函数i2c\_interrupt\_enable描述见下表：

表 3-485. 函数 i2c\_interrupt\_enable

函数名称	i2c_interrupt_enable
函数原型	void i2c_interrupt_enable(uint32_t i2c_periph, uint32_t interrupt);
功能描述	中断使能
先决条件	-
被调用函数	-
输入参数{in}	
i2c_periph	I2C 外设
I2Cx	(x=0,1)
输入参数{in}	
interrupt	I2C 中断
I2C_INT_ERR	错误中断
I2C_INT_TC	发送完成中断
I2C_INT_STPDET	检测到 STOP 中断
I2C_INT_NACK	接收到 NACK 中断
I2C_INT_ADDM	地址匹配中断
I2C_INT_RBNE	接收中断
I2C_INT_TI	发送中断

输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* enable I2C0 transmit interrupt */
i2c_interrupt_enable(I2C0, I2C_INT_TI);
```

### 函数 i2c\_interrupt\_disable

函数i2c\_interrupt\_disable描述见下表：

**表 3-486. 函数 i2c\_interrupt\_disable**

函数名称	i2c_interrupt_disable
函数原型	void i2c_interrupt_disable(uint32_t i2c_periph, uint32_t interrupt);
功能描述	中断除能
先决条件	-
被调用函数	-
输入参数{in}	
i2c_periph	I2C 外设
I2Cx	(x=0,1)
输入参数{in}	
interrupt	I2C 中断
I2C_INT_ERR	错误中断
I2C_INT_TC	发送完成中断
I2C_INT_STPDET	检测到 STOP 中断
I2C_INT_NACK	接收到 NACK 中断
I2C_INT_ADDM	地址匹配中断
I2C_INT_RBNE	接收中断
I2C_INT_TI	发送中断
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* disable I2C0 transmit interrupt */
i2c_interrupt_disable(I2C0, I2C_INT_TI);
```

### 函数 i2c\_interrupt\_flag\_get

函数i2c\_interrupt\_flag\_get描述见下表：

表 3-487. 函数 i2c\_interrupt\_flag\_get

函数名称	i2c_interrupt_flag_get
函数原型	FlagStatus i2c_interrupt_flag_get(uint32_t i2c_periph, i2c_interrupt_flag_enum int_flag);
功能描述	获取中断标志位
先决条件	-
被调用函数	-
输入参数{in}	
i2c_periph	I2C 外设
I2Cx	(x=0,1)
输入参数{in}	
int_flag	I2C 中断标志位, 参考 <a href="#">表 3-425. 枚举类型 i2c_interrupt_flag_enum</a> 。
I2C_INT_FLAG_TI	发送中断标志
I2C_INT_FLAG_RBNE	接收期间 I2C_RDATA 非空中断标志
I2C_INT_FLAG_ADDS END	从机模式下, 接收到的地址与自身地址匹配中断标志
I2C_INT_FLAG_NACK	NACK 中断标志
I2C_INT_FLAG_STPD ET	从机模式下检测到 STOP 信号中断标志
I2C_INT_FLAG_TC	主机模式下传输完成中断标志
I2C_INT_FLAG_TCR	传输完成重载中断标志
I2C_INT_FLAG_BERR	总线错误中断标志
I2C_INT_FLAG_LOSTA RB	仲裁丢失中断标志
I2C_INT_FLAG_OUER R	从机模式下, 过载/欠载错误中断标志
I2C_INT_FLAG_PEC RR	PEC 错误中断标志
I2C_INT_FLAG_TIMEO UT	超时中断标志
I2C_INT_FLAG_SMBA LT	SMBus 报警中断标志
输出参数{out}	
-	-
返回值	
FlagStatus	SET / RESET

例如:

```
/* get I2C interrupt flag status */
```

```
FlagStatus flag_state = RESET;
```

```
flag_state = i2c_interrupt_flag_get(I2C0, I2C_INT_FLAG_TI);
```



## 函数 i2c\_interrupt\_flag\_clear

函数i2c\_interrupt\_flag\_clear描述见下表：

表 3-488. 函数 i2c\_interrupt\_flag\_clear

函数名称	i2c_interrupt_flag_clear
函数原型	void i2c_interrupt_flag_clear(uint32_t i2c_periph, i2c_interrupt_flag_enum int_flag);
功能描述	清除中断标志位
先决条件	-
被调用函数	-
输入参数{in}	
i2c_periph	I2C 外设
I2Cx	(x=0,1)
输入参数{in}	
int_flag	I2C 中断标志位，， 参考 <a href="#">表 3-425. 枚举类型 i2c_interrupt_flag_enum。</a>
I2C_INT_FLAG_ADDS END	从机模式下，接收到的地址与自身地址匹配中断标志
I2C_INT_FLAG_NACK	NACK 中断标志
I2C_INT_FLAG_STPD ET	从机模式下检测到 STOP 信号中断标志
I2C_INT_FLAG_BERR	总线错误中断标志
I2C_INT_FLAG_LOSTA RB	仲裁丢失中断标志
I2C_INT_FLAG_OUER R	从机模式下，过载/欠载错误中断标志
I2C_INT_FLAG_PECE RR	PEC 错误中断标志
I2C_INT_FLAG_TIMEO UT	超时中断标志
I2C_INT_FLAG_SMBA LT	SMBus 报警中断标志
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* clear a bus error flag */
```

```
i2c_interrupt_flag_clear(I2C0, I2C_INT_FLAG_BERR);
```

### 3.16. ICACHE

指令缓存（ICACHE）是由Cortex-M33内核的C-AHB代码总线引入的，用以提升从内部和外部存储取指令和数据时的性能。章节[3.16.1](#)描述了ICACHE的寄存器列表，章节[3.16.2](#)对ICACHE库函数进行说明。

#### 3.16.1. 外设寄存器说明

ICACHE寄存器列表如下表所示：

表 3-489. ICACHE 寄存器

寄存器名称	寄存器描述
ICACHE_CTL	ICACHE 控制寄存器
ICACHE_STAT	ICACHE 状态寄存器
ICACHE_INTEN	ICACHE 中断使能寄存器
ICACHE_FC	ICACHE 标志清除寄存器
ICACHE_HMC	ICACHE 命中监测计数寄存器
ICACHE_MMC	ICACHE 缺失监测计数寄存器
ICACHE_CFGx	ICACHE 配置寄存器

#### 3.16.2. 外设寄存器说明

ICACHE库函数列表如下表所示：

表 3-490. ICACHE 库函数

库函数名称	库函数描述
icache_enable	使能 ICACHE
icache_disable	除能 ICACHE
icache_monitor_enable	使能 ICACHE 监测功能
icache_monitor_disable	除能 ICACHE 监测功能
icache_monitor_reset	复位 ICACHE 监测功能
icache_way_configure	配置 ICACHE 组相连模式
icache_burst_type_select	选择 ICACHE 突发类型
icache_invalidation	使 ICACHE 无效
icache_hitvalue_get	获取命中监测值
icache_missvalue_get	获取缺失监测值
icache_remap_enable	使能 ICACHE 重映射功能
icache_remap_disable	除能 ICACHE 重映射功能
icache_flag_get	获取 ICACHE 标志
icache_flag_clear	清除 ICACHE 标志
icache_interrupt_enable	使能 ICACHE 中断
icache_interrupt_disable	除能 ICACHE 中断
icache_interrupt_flag_get	获取 ICACHE 中断标志

库函数名称	库函数描述
icache_interrupt_flag_clear	清除 ICACHE 中断标志

### 结构体 icache\_remap\_struct

表 3-491. 结构体 icache\_remap\_struct

成员名称	功能描述
region_num	重映射区域编号
base_address	重映射基地址
remap_address	重映射地址
remap_size	重映射大小
burst_type	重映射突发类型
master_sel	重映射主机选择

### 函数 icache\_enable

函数 icache\_enable 描述见下表:

表 3-492. 函数 icache\_enable

函数名称	icache_enable
函数原形	void icache_enable(void);
功能描述	使能 ICACHE
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 icache */
```

```
icache_enable ();
```

### 函数 icache\_disable

函数 icache\_disable 描述见下表:

表 3-493. 函数 icache\_disable

函数名称	icache_disable
函数原形	void icache_disable(void);
功能描述	除能 ICACHE
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 icache */
```

```
icache_disable ();
```

### 函数 icache\_monitor\_enable

函数icache\_monitor\_enable描述见下表：

表 3-494. 函数 icache\_monitor\_enable

函数名称	icache_monitor_enable
函数原形	void icache_monitor_enable(uint32_t monitor_source);
功能描述	使能 ICACHE 监测功能
先决条件	-
被调用函数	-
输入参数{in}	
<b>monitor_source</b>	使能监测源类型
ICACHE_MONITOR_HIT	命中监测
ICACHE_MONITOR_MISS	缺失监测
ICACHE_MONITOR_HIT_MISS	命中和缺失监测
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* enable the icache monitor */
```

```
icache_monitor_enable (ICACHE_MONITOR_HIT);
```

### 函数 icache\_monitor\_disable

函数icache\_monitor\_disable描述见下表：

表 3-495. 函数 icache\_monitor\_disable

函数名称	icache_monitor_disable
函数原形	void icache_monitor_disable(uint32_t monitor_source);
功能描述	除能 ICACHE 监测功能
先决条件	-
被调用函数	-
输入参数{in}	
monitor_source	除能监测源类型
ICACHE_MONITOR_HIT	命中监测
ICACHE_MONITOR_MISS	缺失监测
ICACHE_MONITOR_HIT_MISS	命中和缺失监测
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* disable the icache monitor */
```

```
icache_monitor_disable (ICACHE_MONITOR_HIT);
```

### 函数 icache\_monitor\_reset

函数 icache\_monitor\_reset 描述见下表:

表 3-496. 函数 icache\_monitor\_reset

函数名称	icache_monitor_reset
函数原形	void icache_monitor_reset(uint32_t reset_monitor_source);
功能描述	复位 ICACHE 监测功能
先决条件	-
被调用函数	-
输入参数{in}	
reset_monitor_source	复位监测源类型
ICACHE_MONITOR_HIT	命中监测
ICACHE_MONITOR_MISS	缺失监测
ICACHE_MONITOR_HIT_MISS	命中和缺失监测
输出参数{out}	
-	-

返回值	
-	-

例如：

```
/* reset the icache monitor */
```

```
icache_monitor_reset (ICACHE_MONITOR_HIT);
```

### 函数 icache\_way\_configure

函数icache\_way\_configure描述见下表：

表 3-497. 函数 icache\_way\_configure

函数名称	icache_way_configure
函数原形	ErrStatus icache_way_configure(void);
功能描述	配置 ICACHE 组相连模式
先决条件	-
被调用函数	-
输入参数{in}	
-	-
输出参数{out}	
-	-
返回值	
ErrStatus	SUCCESS或ERROR

例如：

```
/* Configure icache way */
```

```
ErrStatus flag = icache_way_configure ();
```

### 函数 icache\_burst\_type\_select

函数icache\_burst\_type\_select描述见下表：

表 3-498. 函数 icache\_burst\_type\_select

函数名称	icache_burst_type_select
函数原形	ErrStatus icache_burst_type_select(uint32_t burst_type);
功能描述	选择 ICACHE 突发类型
先决条件	-
被调用函数	-
输入参数{in}	
burst_type	输出突发类型
ICACHE_WRAP_BURS T	ICACHE WRAP 突发类型
ICACHE_INCR_BURS T	ICACHE INCR 突发类型

输出参数{out}	
-	-
返回值	
ErrStatus	SUCCESS或ERROR

例如：

```
/* select icache burst type */
```

```
ErrStatus flag = icache_burst_type_select (ICACHE_WRAP_BURST);
```

### 函数 icache\_invalidation

函数icache\_invalidation描述见下表：

表 3-499. 函数 icache\_invalidation

函数名称	icache_invalidation
函数原形	ErrStatus icache_invalidation(void);
功能描述	使 ICACHE 无效
先决条件	-
被调用函数	-
输入参数{in}	
-	-
输出参数{out}	
-	-
返回值	
ErrStatus	SUCCESS或ERROR

例如：

```
/* invalid icache */
```

```
ErrStatus flag = icache_invalidation();
```

### 函数 icache\_hitvalue\_get

函数icache\_hitvalue\_get描述见下表：

表 3-500. 函数 icache\_hitvalue\_get

函数名称	icache_hitvalue_get
函数原形	uint32_t icache_hitvalue_get(void);
功能描述	获取命中监测值
先决条件	-
被调用函数	-
输入参数{in}	
-	-
输出参数{out}	
-	-

返回值	
uint32_t	命中监测计数寄存器的 32 位数值(0-0xFFFF FFFF)

例如：

```
/* get the hit monitor value */

uint8_t hit_value = 0;

hit_value = icache_hitvalue_get ();
```

### 函数 icache\_missvalue\_get

函数icache\_missvalue\_get描述见下表：

表 3-501. 函数 icache\_missvalue\_get

函数名称	icache_missvalue_get
函数原形	uint32_t icache_missvalue_get(void);
功能描述	获取缺失监测值
先决条件	-
被调用函数	-
输入参数{in}	
-	-
输出参数{out}	
-	-
返回值	
uint32_t	缺失监测计数寄存器的低 16 位数值 (0 – 0xFFFF)

例如：

```
/* get the miss monitor value */

uint8_t miss_value = 0;

miss_value = icache_missvalue_get ();
```

### 函数 icache\_remap\_enable

函数icache\_remap\_enable描述见下表：

表 3-502. 函数 icache\_remap\_enable

函数名称	icache_remap_enable
函数原形	ErrStatus icache_remap_enable(icache_remap_struct * icache_remap_config);
功能描述	使能 ICACHE 重映射功能
先决条件	-
被调用函数	-
输入参数{in}	
icache_remap_config	ICACHE重映射结构体，参见 <a href="#">表3-491. 结构体icache_remap_struct</a>



输出参数{out}	
-	-
返回值	
ErrStatus	SUCCESS或ERROR

例如：

```
/* enable the icache remap function */

icache_remap_struct ICACHE_struct;

ICACHE_struct.region_num = 0U;

ICACHE_struct.base_address = 0x10000000UL;

ICACHE_struct.remap_address = 0x30000000UL;

ICACHE_struct.remap_size = ICACHE_REMAP_2M;

ICACHE_struct.burst_type = ICACHE_WRAP_BURST;

icache_remap_enable(&ICACHE_struct);
```

### 函数 icache\_remap\_disable

函数icache\_remap\_disable描述见下表：

表 3-503. 函数 icache\_remap\_disable

函数名称	icache_remap_disable
函数原形	ErrStatus icache_remap_disable(uint32_t region_num);
功能描述	除能 ICACHE 重映射功能
先决条件	-
被调用函数	-
输入参数{in}	
region_num	所除能重映射区域标号
输出参数{out}	
-	-
返回值	
ErrStatus	SUCCESS或ERROR

例如：

```
/* disable the icache remap function */

icache_remap_disable (1);
```

### 函数 icache\_flag\_get

函数icache\_flag\_get描述见下表：

表 3-504. 函数 icache\_flag\_get

函数名称	icache_flag_get
函数原形	FlagStatus icache_flag_get(uint32_t flag);
功能描述	获取 ICACHE 标志
先决条件	-
被调用函数	-
输入参数{in}	
flag	所获取的 ICACHE 标志
ICACHE_BUSY_FLAG	ICACHE 忙碌标志
ICACHE_END_FLAG	ICACHE 完成标志
ICACHE_ERR_FLAG	ICACHE 错误标志
输出参数{out}	
-	-
返回值	
FlagStatus	SET或RESET

例如：

```
/* get icache flag */
```

```
FlagStatus flag_value;
```

```
flag_value = icache_flag_get (ICACHE_BUSY_FLAG);
```

### 函数 icache\_flag\_clear

函数icache\_flag\_clear描述见下表：

表 3-505. 函数 icache\_flag\_clear

函数名称	icache_flag_clear
函数原形	void icache_flag_clear(uint32_t flag);
功能描述	清除ICACHE标志
先决条件	-
被调用函数	-
输入参数{in}	
flag	所清除的 ICACHE 标志
ICACHE_ENDC_FLAG	ICACHE 完成清除标志
ICACHE_ERRC_FLAG	ICACHE 错误清除标志
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* clear icache flag */
```

icache\_flag\_clear (ICACHE\_ENDC\_FLAG);

### 函数 icache\_interrupt\_enable

函数icache\_interrupt\_enable描述见下表:

表 3-506. 函数 icache\_interrupt\_enable

函数名称	icache_interrupt_enable
函数原形	void icache_interrupt_enable(uint32_t interrupt);
功能描述	使能ICACHE中断
先决条件	-
被调用函数	-
输入参数{in}	
interrupt	所使能的中断
ICACHE_ENDIE:	ICACHE 完成中断
ICACHE_ERRIE	ICACHE 错误中断
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* enable icache interrupt */
```

```
icache_interrupt_enable (ICACHE_ENDIE);
```

### 函数 icache\_interrupt\_disable

函数icache\_interrupt\_disable描述见下表:

表 3-507. 函数 icache\_interrupt\_disable

函数名称	icache_interrupt_disable
函数原形	void icache_interrupt_disable(uint32_t interrupt);
功能描述	除能 ICACHE 中断
先决条件	-
被调用函数	-
输入参数{in}	
interrupt	所除能的中断
ICACHE_ENDIE	ICACHE 完成中断
ICACHE_ERRIE	ICACHE 错误中断
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* disable icache interrupt */
```

```
icache_interrupt_disable (ICACHE_ENDIE);
```

### 函数 icache\_interrupt\_flag\_get

函数icache\_interrupt\_flag\_get描述见下表：

**表 3-508. 函数 icache\_flag\_get**

函数名称	icache_interrupt_flag_get
函数原形	FlagStatus icache_interrupt_flag_get(uint32_t interrupt);
功能描述	获取 ICACHE 中断标志
先决条件	-
被调用函数	-
输入参数{in}	
interrupt	所获取的 ICACHE 中断标志
ICACHE_END_FLAG	ICACHE 完成中断标志
ICACHE_ERR_FLAG	ICACHE 错误中断标志
输出参数{out}	
-	-
返回值	
FlagStatus	SET或RESET

例如：

```
/* get icache flag */
```

```
FlagStatus flag_value;
```

```
flag_value = icache_interrupt_flag_get (ICACHE_END_FLAG);
```

### 函数 icache\_interrupt\_flag\_clear

函数icache\_interrupt\_flag\_clear描述见下表：

**表 3-509. 函数 icache\_interrupt\_flag\_clear**

函数名称	icache_interrupt_flag_clear
函数原形	void icache_interrupt_flag_clear(uint32_t interrupt);
功能描述	清除 ICACHE 中断标志
先决条件	-
被调用函数	-
输入参数{in}	
interrupt	所清除的 ICACHE 中断标志
ICACHE_ENDC_FLAG	ICACHE 完成中断清除标志
ICACHE_ERRC_FLAG	ICACHE 错误中断清除标志
输出参数{out}	
-	-

返回值	
-	-

例如：

```
/* clear icache interrupt flag */
```

```
icache_interrupt_flag_clear (ICACHE_ENDC_FLAG);
```

## 3.17. 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 [3.17.1](#) 描述了 NVIC 和 SysTick 的寄存器列表，章节 [3.17.2](#) 对 MISC 库函数进行说明。

### 3.17.1. 外设寄存器说明

表 3-510. NVIC 寄存器

寄存器名称	寄存器描述
ISER <sup>(1)</sup>	中断使能寄存器
ICER <sup>(1)</sup>	中断禁能寄存器
ISPR <sup>(1)</sup>	中断挂起寄存器
ICPR <sup>(1)</sup>	中断清除寄存器
IABR <sup>(1)</sup>	中断活动状态寄存器
ITNS <sup>(1)</sup>	中断不安全状态寄存器
IPR <sup>(1)</sup>	中断优先级寄存器
STIR <sup>(1)</sup>	软触发中断寄存器
CPUID <sup>(2)</sup>	CPUID 寄存器
ICSR <sup>(2)</sup>	中断控制及状态寄存器
VTOR <sup>(2)</sup>	向量表偏移量寄存器
AIRCR <sup>(2)</sup>	应用程序中断及复位控制寄存器
SCR <sup>(2)</sup>	系统控制寄存器
CCR <sup>(2)</sup>	配置与控制寄存器
SHP <sup>(2)</sup>	系统异常优先级寄存器
SHCSR <sup>(2)</sup>	系统异常控制及状态寄存器

1. 参考 core\_cm33.h 文件中定义的结构体类型 NVIC\_Type

2. 参考 core\_cm33.h 文件中定义的结构体类型 SCB\_Type

表 3-511. SysTick 寄存器

寄存器名称	寄存器描述
CTRL <sup>(1)</sup>	系统控制和状态寄存器
LOAD <sup>(1)</sup>	系统重载值寄存器
VAL <sup>(1)</sup>	系统当前值寄存器
CALIB <sup>(1)</sup>	系统校准寄存器

1. 参考 core\_cm33.h 文件中定义的结构体类型 SysTick\_Type

## 3.17.2. 外设库函数说明

## 枚举类型 IRQn\_Type

表 3-512. 枚举类型 IRQn\_Type

成员名称	功能描述
WWDGT_IRQn	窗口看门狗中断
LVD_IRQn	连接到 EXTI 线的 LVD 中断
TAMPER_STAMP_IRQn	侵入和时间戳中断
RTC_WKUP_IRQn	RTC 唤醒中断
FMC_IRQn	FMC 全局中断
RCU_IRQn	RCU 全局中断
EXTI0_IRQn	EXTI 线 0 中断
EXTI1_IRQn	EXTI 线 1 中断
EXTI2_IRQn	EXTI 线 2 中断
EXTI3_IRQn	EXTI 线 3 中断
EXTI4_IRQn	EXTI 线 4 中断
DMA0_Channel0_IRQn	DMA0 通道 0 全局中断
DMA0_Channel1_IRQn	DMA0 通道 1 全局中断
DMA0_Channel2_IRQn	DMA0 通道 2 全局中断
DMA0_Channel3_IRQn	DMA0 通道 3 全局中断
DMA0_Channel4_IRQn	DMA0 通道 4 全局中断
DMA0_Channel5_IRQn	DMA0 通道 5 全局中断
DMA0_Channel6_IRQn	DMA0 通道 6 全局中断
DMA0_Channel7_IRQn	DMA0 通道 7 全局中断
ADC_IRQn	ADC 中断
TAMPER_STAMP_S_IRQn	RTC 侵入和时间戳安全中断
RTC_WKUP_S_IRQn	RTC 唤醒安全中断
RTC_Alarm_S_IRQn	RTC 闹钟安全中断
EXTI5_9_IRQn	EXTI 线[9:5] 中断
TIMER0_BRK_IRQn	TIMER0 中止中断
TIMER0_UP_IRQn	TIMER0 更新中断
TIMER0_CMT_IRQn	TIMER0 换相中断
TIMER0_Channel_IRQn	TIMER0 通道捕获比较中断
TIMER1_IRQn	TIMER1 全局中断
TIMER2_IRQn	TIMER2 全局中断
TIMER3_IRQn	TIMER3 全局中断
I2C0_EV_IRQn	I2C0 事件中断
I2C0_ER_IRQn	I2C0 错误中断
I2C1_EV_IRQn	I2C1 事件中断
I2C1_ER_IRQn	I2C1 错误中断
SPI0_IRQn	SPI0 全局中断

成员名称	功能描述
SPI1_IRQn	SPI1 全局中断
USART0_IRQn	USART0 全局中断
USART1_IRQn	USART1 全局中断
USART2_IRQn	USART2 全局中断
EXTI10_15_IRQn	EXTI 线[15:10] 中断
RTC_Alarm_IRQn	RTC 闹钟中断
VLVDF_IRQn	VLVDF 中断
TIMER15_IRQn	TIMER15 全局中断
TIMER16_IRQn	TIMER16 全局中断
SDIO_IRQn	SDIO 全局中断
TIMER4_IRQn	TIMER4 全局中断
I2C0_WKUP_IRQn	I2C0 唤醒中断
USART0_IRQn	USART0 唤醒中断
USART2_IRQn	USART2 唤醒中断
TIMER5_IRQn	TIMER5 全局中断
DMA1_Channel0_IRQn	DMA1 通道 0 全局中断
DMA1_Channel1_IRQn	DMA1 通道 1 全局中断
DMA1_Channel2_IRQn	DMA1 通道 2 全局中断
DMA1_Channel3_IRQn	DMA1 通道 3 全局中断
DMA1_Channel4_IRQn	DMA1 通道 4 全局中断
DMA1_Channel5_IRQn	DMA1 通道 5 全局中断
DMA1_Channel6_IRQn	DMA1 通道 6 全局中断
DMA1_Channel7_IRQn	DMA1 通道 7 全局中断
WIFI11N_WKUP_IRQn	WIFI11N 唤醒中断
USBFS_IRQn	USBFS 全局中断
USBFS_WKUP_IRQn	USBFS 唤醒中断
DCI_IRQn	DCI 全局中断
CAU_IRQn	CAU 全局中断
HAU_TRNG_IRQn	HAU/TRNG 全局中断
FPU_IRQn	FPU 全局中断
HPDF_INT0_IRQn	HPDF 全局中断 0
HPDF_INT1_IRQn	HPDF 全局中断 1
WIFI11N_INT0_IRQn	WIFI11N 全局中断 0
WIFI11N_INT1_IRQn	WIFI11N 全局中断 1
WIFI11N_INT2_IRQn	WIFI11N 全局中断 2
EFUSE_IRQn	EFUSE 全局中断
QSPI_IRQn	QSPI 全局中断
PKCAU_IRQn	PKCAU 全局中断
TSI_IRQ	TSI 全局中断
ICACHE_IRQn	ICACHE 全局中断
TZIAC_S_IRQn	TZIAC 安全中断

成员名称	功能描述
FMC_S_IRQn	FMC 安全中断
QSPI_S_IRQn	QSPI 安全中断

MISC库函数列表如下表所示：

**表 3-513. 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_system_reset	复位MCU
nvic_vector_table_set	设置向量表地址
system_lowpower_set	设置系统低功耗模式状态
system_lowpower_reset	复位系统低功耗模式状态
systick_clksource_set	设置系统定时器时钟源

### 函数 nvic\_priority\_group\_set

函数nvic\_priority\_group\_set描述见下表：

**表 3-514. 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
NVIC_PRIGROUP_PRE0_SUB4	0位用于抢占优先级，4位用于响应优先级
NVIC_PRIGROUP_PRE1_SUB3	1位用于抢占优先级，3位用于响应优先级
NVIC_PRIGROUP_PRE2_SUB2	2位用于抢占优先级，2位用于响应优先级
NVIC_PRIGROUP_PRE3_SUB1	3位用于抢占优先级，1位用于响应优先级
NVIC_PRIGROUP_PRE4_SUB0	4位用于抢占优先级，0位用于响应优先级
输出参数{out}	
-	-
返回值	
-	-

例如：



```
/* priority group configuration , 0 bits for pre-emption priority 4 bits for subpriority */
```

```
nvic_priority_group_set(NVIC_PRIGROUP_PRE0_SUB4);
```

### 函数 nvic\_irq\_enable

函数nvic\_irq\_enable描述见下表:

表 3-515. 函数 nvic\_irq\_enable

函数名称	nvic_irq_enable
函数原形	void nvic_irq_enable(uint8_t nvic_irq, uint8_t nvic_irq_pre_priority, uint8_t nvic_irq_sub_priority);
功能描述	使能中断，配置中断的优先级
先决条件	-
被调用函数	nvic_priority_group_set
输入参数{in}	
nvic_irq	NVIC中断，参考 <a href="#">表3-512. 枚举类型IRQn_Type</a>
输入参数{in}	
nvic_irq_pre_priority	抢占优先级（0~4）
输入参数{in}	
nvic_irq_sub_priority	响应优先级（0~4）
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* enable window watchdog timer interrupt , pre-emption priority is 1, subpriority is 1 */
```

```
nvic_irq_enable(WWDGT_IRQn,1,1);
```

### 函数 nvic\_irq\_disable

函数nvic\_irq\_disable描述见下表:

表 3-516. 函数 nvic\_irq\_disable

函数名称	nvic_irq_disable
函数原形	void nvic_irq_disable (uint8_t nvic_irq);
功能描述	禁能中断
先决条件	-
被调用函数	-
输入参数{in}	
nvic_irq	NVIC中断，参考 <a href="#">表3-512. 枚举类型IRQn_Type</a>
输出参数{out}	

-	-
返回值	
-	-

例如：

```
/* disable window watchDog timer interrupt */
```

```
nvic_irq_disable(WWDGT_IRQn);
```

### 函数 nvic\_system\_reset

函数nvic\_system\_reset描述见下表：

**Table 3-2. Function nvic\_system\_reset**

函数名称	nvic_system_reset
函数原形	void nvic_system_reset(void);
功能描述	initiates a system reset request to reset the MCU
先决条件	-
被调用函数	-
输入参数{in}	
-	-
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* initiates a system reset request to reset the MCU */
```

```
nvic_system_reset();
```

### 函数 nvic\_vector\_table\_set

函数nvic\_vector\_table\_set描述见下表：

**表 3-517. 函数 nvic\_vector\_table\_set**

函数名称	nvic_vector_table_set
函数原形	void nvic_vector_table_set(uint32_t nvic_vect_tab, uint32_t offset);
功能描述	设置向量表地址
先决条件	-
被调用函数	-
输入参数{in}	
nvic_vect_tab	RAM 或者 FLASH基地址
NVIC_VECTTAB_RAM	RAM基地址
NVIC_VECTTAB_FLASH	FLASH基地址

LASH	
NVIC_VECTTAB_RAM_SECURE	RAM安全基地址
NVIC_VECTTAB_FLASH_SECURE	FLASH安全基地址
输入参数{in}	
offset	向量表偏移量（向量表地址=基地址+偏移量）
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* set vector table address = NVIC_VECTTAB_FLASH + 0x200 */
nvic_vector_table_set (NVIC_VECTTAB_FLASH, 0x200);
```

### 函数 system\_lowpower\_set

函数system\_lowpower\_set描述见下表：

表 3-518. 函数 system\_lowpower\_set

函数名称	system_lowpower_set
函数原形	void system_lowpower_set(uint8_t lowpower_mode);
功能描述	系统低功耗模式状态的管理
先决条件	-
被调用函数	-
输入参数{in}	
lowpower_mode	系统低功耗模式的状态
SCB_LPM_SLEEP_EXIT_ISR	该位为1时，退出ISR时一直处于低功耗模式
SCB_LPM_DEEPSLEEP	该位为1时，系统处于deep sleep模式
SCB_LPM_WAKE_BY_ALL_INT	该位为1时，低功耗模式可以被所有中断唤醒（无论中断是否被使能）
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* the system always enter low power mode by exiting from ISR */
system_lowpower_set (SCB_LPM_SLEEP_EXIT_ISR);
```

函数 **system\_lowpower\_reset**

函数system\_lowpower\_reset描述见下表：

表 3-519. 函数 **system\_lowpower\_reset**

函数名称	system_lowpower_reset
函数原形	void system_lowpower_reset(uint8_t lowpower_mode);
功能描述	系统低功耗模式状态的管理
先决条件	-
被调用函数	-
输入参数{in}	
lowpower_mode	系统低功耗模式的状态
SCB_LPM_SLEEP_EXIT_ISR	该位为0时，退出ISR时退出低功耗模式
SCB_LPM_DEEPSLEEP	该位为0时，系统进入sleep模式
SCB_LPM_WAKEUP_BY_ALL_INT	该位为0时，系统只能被使能的中断唤醒
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* the system will exit low power mode by exiting from ISR */
```

```
system_lowpower_reset (SCB_LPM_SLEEP_EXIT_ISR);
```

函数 **systick\_clksource\_set**

函数systick\_clksource\_set描述见下表：

表 3-520. 函数 **systick\_clksource\_set**

函数名称	systick_clksource_set
函数原形	void systick_clksource_set(uint32_t systick_clksource);
功能描述	设置SysTick时钟源
先决条件	-
被调用函数	-
输入参数{in}	
systick_clksource	SysTick时钟源
SYSTICK_CLKSOURCE_HCLK	SysTick时钟源为AHB时钟
SYSTICK_CLKSOURCE_HCLK_DIV8	SysTick时钟源为AHB时钟的8分频
输出参数{out}	

-	-
返回值	
-	-

例如:

```
/* systick clock source is HCLK/8 */
```

```
systick_clksource_set (SYSTICK_CLKSOURCE_HCLK_DIV8);
```

## 3.18. PKCAU

公钥加密处理器(PKCAU)支持加速GF(p) (伽罗华域)上的RSA(Rivest、Shamir和Adleman)、Diffie-Hellmann (DH密钥交换)或ECC(椭圆曲线加密)加密算法。章节[3.18.1](#)描述了PKCAU的寄存器列表, 章节[3.18.2](#)对PKCAU库函数进行说明。

### 3.18.1. 外设寄存器说明

PKCAU寄存器列表如下表所示:

表 3-521. PKCAU 寄存器

寄存器名称	寄存器描述
PKCAU_CTL	控制寄存器
PKCAU_STAT	状态寄存器
PKCAU_STATC	状态清除寄存器

### 3.18.2. 外设库函数说明

PKCAU库函数列表如下表所示:

表 3-522. PKCAU 库函数

库函数名称	库函数描述
pkcau_deinit	复位PKCAU
pkcau_mont_struct_para_init	初始化蒙哥马利参数结构体为默认值
pkcau_mod_struct_para_init	初始化模运算参数结构体为默认值
pkcau_mod_exp_struct_para_init	初始化模幂运算参数结构体为默认值
pkcau_mod_inver_struct_para_init	初始化模逆运算参数结构体为默认值
pkcau_arithmetic_struct_para_init	初始化算术运算参数结构体为默认值
pkcau_crt_struct_para_init	初始化CRT参数结构体为默认值
pkcau_ec_group_struct_para_init	初始化ECC椭圆曲线参数结构体为默认值
pkcau_point_struct_para_init	初始化点参数结构体为默认值
pkcau_signature_struct_para_init	初始化签名参数结构体为默认值
pkcau_hash_struct_para_init	初始化哈希参数结构体为默认值
pkcau_mod_reduc_struct_para_init	初始化取模参数结构体为默认值
pkcau_enable	使能PKCAU

库函数名称	库函数描述
pkcau_disable	禁能PKCAU
pkcau_start	开始运算
pkcau_mode_set	配置PKCAU运算模式
pkcau_mont_param_operation	执行蒙哥马利参数运算
pkcau_mod_operation	执行模运算，包含模加，模减以及蒙哥马利乘法
pkcau_mod_exp_operation	执行模幂运算
pkcau_mod_inver_operation	执行模逆运算
pkcau_mod_reduc_operation	执行取模运算
pkcau_arithmetic_operation	执行算术加法运算
pkcau_crt_exp_operation	执行RSA CRT幂运算
pkcau_point_check_operation	执行椭圆上点的检查
pkcau_point_mul_operation	执行点乘运算
pkcau_ecdsa_sign_operation	执行ECDSA签名运算
pkcau_ecdsa_verification_operation	执行ECDSA验证运算
pkcau_memread	从PKCAU RAM读取结果
pkcau_flag_get	获取PKCAU标志位
pkcau_flag_clear	清除PKCAU标志位
pkcau_interrupt_enable	中断使能
pkcau_interrupt_disable	中断禁能
pkcau_interrupt_flag_get	获取PKCAU中断标志位
pkcau_interrupt_flag_clear	清除PKCAU中断标志位

### 结构体 pkcau\_mont\_parameter\_struct

表 3-523. 结构体 pkcau\_mont\_parameter\_struct

成员名称	功能描述
modulus_n	模数n
modulus_len	模长（字节长度）

### 结构体 pkcau\_mod\_parameter\_struct

表 3-524. 结构体 pkcau\_mod\_parameter\_struct

成员名称	功能描述
opr_d_a	操作数A
opr_d_b	操作数B
modulus_n	模数n
modulus_len	模长（字节长度）

### 结构体 pkcau\_mod\_exp\_parameter\_struct

表 3-525. 结构体 pkcau\_mod\_exp\_parameter\_struct

成员名称	功能描述
opr_d_a	操作数A

成员名称	功能描述
exp_e	指数e
e_len	指数e长度（字节长度）
modulus_n	模数n
modulus_len	模长（字节长度）
mont_para	蒙哥马利参数 $R2 \bmod n$

### 结构体 `pkcau_mod_inver_parameter_struct`

表 3-526. 结构体 `pkcau_mod_inver_parameter_struct`

成员名称	功能描述
opr_d_a	操作数A
modulus_n	模数n
modulus_len	模长（字节长度）

### 结构体 `pkcau_mod_reduc_parameter_struct`

表 3-527. 结构体 `pkcau_mod_reduc_parameter_struct`

成员名称	功能描述
opr_d_a	操作数A
opr_d_a_len	操作数A的长度（字节长度）
modulus_n	模数n
modulus_len	模长（字节长度）

### 结构体 `pkcau_arithmetic_parameter_struct`

表 3-528. 结构体 `pkcau_arithmetic_parameter_struct`

成员名称	功能描述
opr_d_a	操作数A
opr_d_b	操作数B
opr_d_len	操作数的长度（字节长度）

### 结构体 `pkcau_crt_parameter_struct`

表 3-529. 结构体 `pkcau_crt_parameter_struct`

成员名称	功能描述
opr_d_a	操作数A
opr_d_len	操作数的长度（字节长度）
opr_d_dp	操作数dp
opr_d_dq	操作数dq
opr_d_qinv	操作数qinv
opr_d_p	质数操作数p
opr_d_q	质数操作数q

结构体 `pkcau_ec_group_parameter_struct`表 3-530. 结构体 `pkcau_ec_group_parameter_struct`

成员名称	功能描述
<code>modulus_p</code>	曲线模长p
<code>coff_a</code>	曲线系数a
<code>coff_b</code>	曲线系数b
<code>base_point_x</code>	曲线基点坐标x
<code>base_point_y</code>	曲线基点坐标y
<code>order_n</code>	曲线质数阶n
<code>a_sign</code>	曲线系数a的符号
<code>modulus_p_len</code>	曲线模数p的长度（字节长度）
<code>order_n_len</code>	曲线质数阶n的长度（字节长度）
<code>multi_k</code>	标量乘数k
<code>k_len</code>	标量乘数k的长度
<code>mont_para</code>	蒙哥马利参数R2 mod n

结构体 `pkcau_point_parameter_struct`表 3-531. 结构体 `pkcau_point_parameter_struct`

成员名称	功能描述
<code>point_x</code>	点的坐标x
<code>point_y</code>	点的坐标y

结构体 `pkcau_signature_parameter_struct`表 3-532. 结构体 `pkcau_signature_parameter_struct`

成员名称	功能描述
<code>sign_r</code>	签名r部分
<code>sign_s</code>	签名s部分

结构体 `pkcau_hash_parameter_struct`表 3-533. 结构体 `pkcau_hash_parameter_struct`

成员名称	功能描述
<code>hash_z</code>	哈希值z
<code>hash_z_len</code>	哈希值z的长度（字节长度）

函数 `pkcau_deinit`

函数`pkcau_deinit`描述见下表：

表 3-534. 函数 `pkcau_deinit`

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



功能描述	复位PKCAU
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 PKCAU */
```

```
pkcau_deinit();
```

### 函数 pkcau\_mont\_struct\_para\_init

函数pkcau\_mont\_struct\_para\_init描述见下表：

表 3-535. 函数 pkcau\_mont\_struct\_para\_init

函数名称	pkcau_mont_struct_para_init
函数原型	void pkcau_mont_struct_para_init(pkcau_mont_parameter_struct* init_para);
功能描述	初始化蒙哥马利参数结构体为默认值
先决条件	-
被调用函数	-
输入参数{in}	
init_para	蒙哥马利参数初始化结构体，结构体成员参考 <a href="#">表3-523. 结构体 pkcau_mont_parameter_struct</a> 。
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* initialize PKCAU montgomery parameter struct with a default value */
```

```
pkcau_mont_parameter_struct pkcau_mont_parameter;
```

```
pkcau_mont_struct_para_init(&pkcau_mont_parameter);
```

### 函数 pkcau\_mod\_struct\_para\_init

函数pkcau\_mod\_struct\_para\_init描述见下表：

表 3-536. 函数 pkcau\_mod\_struct\_para\_init

函数名称	pkcau_mod_struct_para_init
函数原型	void pkcau_mod_struct_para_init(pkcau_mod_parameter_struct* init_para);
功能描述	初始化模运算参数结构体为默认值
先决条件	-
被调用函数	-
输入参数{in}	
init_para	模参数初始化结构体，结构体成员参考 <a href="#">表3-524. 结构体 pkcau_mod_parameter_struct</a> 。
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* initialize PKCAU modular parameter struct with a default value */
pkcau_mod_parameter_struct pkcau_mod_parameter;
pkcau_mod_struct_para_init(&pkcau_mod_parameter);
```

### 函数 pkcau\_mod\_exp\_struct\_para\_init

函数pkcau\_mod\_exp\_struct\_para\_init描述见下表：

表 3-537. 函数 pkcau\_mod\_exp\_struct\_para\_init

函数名称	pkcau_mod_exp_struct_para_init
函数原型	void pkcau_mod_exp_struct_para_init(pkcau_mod_exp_parameter_struct* init_para);
功能描述	初始化模幂运算参数结构体为默认值
先决条件	-
被调用函数	-
输入参数{in}	
init_para	模幂参数初始化结构体，结构体成员参考 <a href="#">表3-525. 结构体 pkcau_mod_exp_parameter_struct</a> 。
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* initialize PKCAU modular exponentation parameter struct with a default value */
pkcau_mod_exp_parameter_struct pkcau_mod_exp_parameter;
pkcau_mod_exp_struct_para_init(&pkcau_mod_exp_parameter);
```

**函数 pkcau\_mod\_inver\_struct\_para\_init**

函数pkcau\_mod\_inver\_struct\_para\_init描述见下表：

**表 3-538. 函数 pkcau\_mod\_inver\_struct\_para\_init**

函数名称	pkcau_mod_inver_struct_para_init
函数原型	void pkcau_mod_inver_struct_para_init(pkcau_mod_inver_parameter_struct* init_para);
功能描述	初始化模逆运算参数结构体为默认值
先决条件	-
被调用函数	-
输入参数{in}	
init_para	模逆参数初始化结构体，结构体成员参考 <a href="#">表3-526. 结构体 pkcau_mod_inver_parameter_struct</a> 。
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* initialize PKCAU modular inversion parameter struct with a default value */
```

```
pkcau_mod_inver_parameter_struct pkcau_mod_inver_parameter;
```

```
pkcau_mod_inver_struct_para_init(&pkcau_mod_inver_parameter);
```

**函数 pkcau\_arithmetic\_struct\_para\_init**

函数pkcau\_arithmetic\_struct\_para\_init描述见下表：

**表 3-539. 函数 pkcau\_arithmetic\_struct\_para\_init**

函数名称	pkcau_arithmetic_struct_para_init
函数原型	void pkcau_arithmetic_struct_para_init(pkcau_arithmetic_parameter_struct* init_para);
功能描述	初始化算术运算参数结构体为默认值
先决条件	-
被调用函数	-
输入参数{in}	
init_para	算术参数初始化结构体，结构体成员参考 <a href="#">表3-528. 结构体 pkcau_arithmetic_parameter_struct</a> 。
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* initialize PKCAU arithmetic parameter struct with a default value */

pkcau_arithmetic_parameter_struct pkcau_arithmetic_parameter;

pkcau_arithmetic_struct_para_init(&pkcau_arithmetic_parameter);
```

### 函数 pkcau\_crt\_struct\_para\_init

函数pkcau\_crt\_struct\_para\_init描述见下表：

表 3-540. 函数 pkcau\_crt\_struct\_para\_init

函数名称	pkcau_crt_struct_para_init
函数原型	void pkcau_crt_struct_para_init(pkcau_crt_parameter_struct* init_para);
功能描述	初始化CRT参数结构体为默认值
先决条件	-
被调用函数	-
输入参数{in}	
init_para	CRT参数初始化结构体，结构体成员参考 <a href="#">表3-529. 结构体 pkcau_crt_parameter_struct</a> 。
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* initialize PKCAU CRT parameter struct with a default value */

pkcau_crt_parameter_struct pkcau_crt_parameter;

pkcau_crt_struct_para_init(&pkcau_crt_parameter);
```

### 函数 pkcau\_ec\_group\_struct\_para\_init

函数pkcau\_ec\_group\_struct\_para\_init描述见下表：

表 3-541. 函数 pkcau\_ec\_group\_struct\_para\_init

函数名称	pkcau_ec_group_struct_para_init
函数原型	void pkcau_ec_group_struct_para_init(pkcau_ec_group_parameter_struct* init_para);
功能描述	初始化ECC椭圆曲线参数结构体为默认值
先决条件	-
被调用函数	-
输入参数{in}	
init_para	ECC椭圆参数初始化结构体，结构体成员参考 <a href="#">表3-530. 结构体 pkcau_ec_group_parameter_struct</a> 。

输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* initialize PKCAU ECC curve parameter struct with a default value */
```

```
pkcau_ec_group_parameter_struct pkcau_ec_group_parameter;
```

```
pkcau_ec_group_struct_para_init(&pkcau_ec_group_parameter);
```

### 函数 pkcau\_point\_struct\_para\_init

函数pkcau\_point\_struct\_para\_init描述见下表：

**表 3-542. 函数 pkcau\_point\_struct\_para\_init**

函数名称	pkcau_point_struct_para_init
函数原型	void pkcau_point_struct_para_init(pkcau_point_parameter_struct* init_para);
功能描述	初始化点参数结构体为默认值
先决条件	-
被调用函数	-
输入参数{in}	
init_para	点参数初始化结构体，结构体成员参考 <a href="#">表3-531. 结构体 pkcau_point_parameter_struct</a> 结构体。
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* initialize PKCAU point parameter struct with a default value */
```

```
pkcau_point_parameter_struct pkcau_point_parameter;
```

```
pkcau_point_struct_para_init(&pkcau_point_parameter);
```

### 函数 pkcau\_signature\_struct\_para\_init

函数pkcau\_signature\_struct\_para\_init描述见下表：

**表 3-543. 函数 pkcau\_signature\_struct\_para\_init**

函数名称	pkcau_signature_struct_para_init
函数原型	void pkcau_signature_struct_para_init(pkcau_signature_parameter_struct* init_para);

功能描述	初始化签名参数结构体为默认值
先决条件	-
被调用函数	-
输入参数{in}	
init_para	签名参数初始化结构体，结构体成员参考 <a href="#">表3-532. 结构体 pkcau_signature_parameter_struct</a> 。
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* initialize PKCAU signature parameter struct with a default value */
pkcau_signature_parameter_struct pkcau_signature_parameter;
pkcau_signature_struct_para_init(&pkcau_signature_parameter);
```

### 函数 pkcau\_hash\_struct\_para\_init

函数pkcau\_hash\_struct\_para\_init描述见下表：

表 3-544. 函数 pkcau\_hash\_struct\_para\_init

函数名称	pkcau_hash_struct_para_init
函数原型	void pkcau_hash_struct_para_init(pkcau_hash_parameter_struct* init_para);
功能描述	初始化哈希参数结构体为默认值
先决条件	-
被调用函数	-
输入参数{in}	
init_para	哈希参数初始化结构体，结构体成员参考 <a href="#">表3-533. 结构体 pkcau_hash_parameter_struct</a> 。
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* initialize PKCAU hash parameter struct with a default value */
pkcau_hash_parameter_struct pkcau_hash_parameter;
pkcau_hash_struct_para_init(&pkcau_hash_parameter);
```

**函数 pkcau\_mod\_reduc\_struct\_para\_init**

函数pkcau\_mod\_reduc\_struct\_para\_init描述见下表：

**表 3-545. 函数 pkcau\_mod\_reduc\_struct\_para\_init**

函数名称	pkcau_mod_reduc_struct_para_init
函数原型	void pkcau_mod_reduc_struct_para_init(pkcau_mod_reduc_parameter_struct* init_para);
功能描述	初始化取模参数结构体为默认值
先决条件	-
被调用函数	-
输入参数{in}	
init_para	取模参数初始化结构体，结构体成员参考 <a href="#">表3-527. 结构体 pkcau_mod_reduc_parameter_struct</a> 。
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* initialize PKCAU modular reduction parameter struct with a default value */
pkcau_mod_reduc_parameter_struct pkcau_mod_reduc_parameter;
pkcau_mod_reduc_struct_para_init(&pkcau_mod_reduc_parameter);
```

**函数 pkcau\_enable**

函数pkcau\_enable描述见下表：

**表 3-546. 函数 pkcau\_enable**

函数名称	pkcau_enable
函数原型	void pkcau_enable(void);
功能描述	使能PKCAU
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 PKCAU */
```

```
pkcau_enable();
```

### 函数 pkcau\_disable

函数pkcau\_disable描述见下表:

表 3-547. 函数 pkcau\_disable

函数名称	pkcau_disable
函数原型	void pkcau_disable(void);
功能描述	禁能PKCAU
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 PKCAU */
```

```
pkcau_disable();
```

### 函数 pkcau\_start

函数pkcau\_start描述见下表:

表 3-548. 函数 pkcau\_start

函数名称	pkcau_start
函数原型	void pkcau_start(void);
功能描述	开始运算
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 operation */
```

```
pkcau_start();
```



## 函数 pkcau\_mode\_set

函数pkcau\_mode\_set描述见下表:

表 3-549. 函数 pkcau\_mode\_set

函数名称	pkcau_mode_set
函数原型	void pkcau_mode_set(uint32_t mode);
功能描述	配置PKCAU运算模式
先决条件	-
被调用函数	-
输入参数{in}	
mode	PKCAU运算模式
PKCAU_MODE_MOD_EXP	先计算蒙哥马利参数，然后进行模幂运算
PKCAU_MODE_MONT_PARAM	仅计算蒙哥马利参数
PKCAU_MODE_MOD_EXP_FAST	仅进行模幂运算
PKCAU_MODE_CRT_EXP	RSA CRT幂运算
PKCAU_MODE_MOD_INVERSION	模逆运算
PKCAU_MODE_ARITHMETIC_ADD	算术加法运算
PKCAU_MODE_ARITHMETIC_SUB	算术减法运算
PKCAU_MODE_ARITHMETIC_MUL	算术乘法运算
PKCAU_MODE_ARITHMETIC_COMP	算术比较运算
PKCAU_MODE_MOD_REDUCTION	取模运算
PKCAU_MODE_MOD_ADD	模加运算
PKCAU_MODE_MOD_SUB	模减运算
PKCAU_MODE_MONT_MUL	蒙哥马利乘法运算
PKCAU_MODE_ECC_MUL	先计算蒙哥马利参数，然后进行ECC标量乘法运算
PKCAU_MODE_ECC_MUL_FAST	仅进行ECC标量乘法运算
PKCAU_MODE_ECDSA_SIGN	ECDSA签名

PKCAU_MODE_ECDSA_VERIFY	ECDSA验证
PKCAU_MODE_POINT_CHECK	椭圆曲线Fp上点的检查
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* configure the PKCAU operation mode as PKCAU_MODE_ARITHMETIC_ADD */
pkcau_mode_set(PKCAU_MODE_ARITHMETIC_ADD);
```

### 函数 pkcau\_mont\_param\_operation

函数pkcau\_mont\_param\_operation描述见下表：

**表 3-550. 函数 pkcau\_mont\_param\_operation**

函数名称	pkcau_mont_param_operation
函数原型	void pkcau_mont_param_operation(pkcau_mont_parameter_struct *mont_para);
功能描述	执行蒙哥马利参数运算
先决条件	-
被调用函数	-
输入参数{in}	
mont_para	蒙哥马利参数初始化结构体，结构体成员参考 <a href="#">表3-523. 结构体 pkcau_mont_parameter_struct</a> 。
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* initialize the montgomery parameter structure */
pkcau_mont_parameter_struct pkcau_mont_parameter;
pkcau_mont_struct_para_init(&pkcau_mont_parameter);

/* initialize the montgomery parameters */
pkcau_mont_parameter.modulus_len = ME_MOD_SIZE;
pkcau_mont_parameter.modulus_n = (uint8_t *)modulus_n;

/* execute montgomery parameter operation */
```

```
pkcau_mod_param_operation(&pkcau_mod_parameter);
```

### 函数 pkcau\_mod\_operation

函数pkcau\_mod\_operation描述见下表：

表 3-551. 函数 pkcau\_mod\_operation

函数名称	pkcau_mod_operation
函数原型	void pkcau_mod_operation(pkcau_mod_parameter_struct *mod_para, uint32_t mode);
功能描述	执行模运算，包含模加，模减以及蒙哥马利乘法
先决条件	-
被调用函数	-
输入参数{in}	
mod_para	模参数初始化结构体，结构体成员参考 <a href="#">表3-524. 结构体 pkcau_mod_parameter_struct</a> 。
输入参数{in}	
mode	模运算模式
PKCAU_MODE_MOD_ADD	模加运算
PKCAU_MODE_MOD_SUB	模减运算
PKCAU_MODE_MONT_MUL	蒙哥马利乘法运算
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* initialize the modular parameter structure */
pkcau_mod_parameter_struct pkcau_mod_parameter;
pkcau_mod_struct_para_init(&pkcau_mod_parameter);
/* initialize the modular parameters */
pkcau_mod_parameter.oprd_a = (uint8_t *)oprd_a;
pkcau_mod_parameter.oprd_b = (uint8_t *)oprd_b;
pkcau_mod_parameter.modulus_len = MA_MOD_SIZE;
pkcau_mod_parameter.modulus_n = (uint8_t *)modulus_n;
/* execute modular addition operation */
pkcau_mod_operation(&pkcau_mod_parameter, PKCAU_MODE_MOD_ADD);
```

## 函数 pkcau\_mod\_exp\_operation

函数pkcau\_mod\_exp\_operation描述见下表：

表 3-552. 函数 pkcau\_mod\_exp\_operation

函数名称	pkcau_mod_exp_operation
函数原型	void pkcau_mod_exp_operation(pkcau_mod_exp_parameter_struct *mod_exp_para, uint32_t mode);
功能描述	执行模幂运算
先决条件	-
被调用函数	-
输入参数{in}	
mod_exp_para	模幂参数初始化结构体，结构体成员参考 <a href="#">表3-525. 结构体 pkcau_mod_exp_parameter_struct</a> 。
输入参数{in}	
mode	模幂运算模式
PKCAU_MODE_MOD_EXP	先计算蒙哥马利参数，然后进行模幂运算
PKCAU_MODE_MOD_EXP_FAST	仅进行模幂运算
输出参数{out}	
-	-
返回值	
-	-

例如：

```

/* initialize the modular exponentiation parameter structure */
pkcau_mod_exp_parameter_struct pkcau_mod_exp_parameter;
pkcau_mod_exp_struct_para_init(&pkcau_mod_exp_parameter);

/* initialize the modular exponentiation parameters */
pkcau_mod_exp_parameter.oprd_a = (uint8_t *)oprd_a;
pkcau_mod_exp_parameter.exp_e = (uint8_t *)exp_e;
pkcau_mod_exp_parameter.e_len = ME_E_SIZE;
pkcau_mod_exp_parameter.modulus_len = ME_MOD_SIZE;
pkcau_mod_exp_parameter.modulus_n = (uint8_t *)modulus_n;
pkcau_mod_exp_parameter.mont_para = (uint8_t *)mont_para;

/* execute modular exponentiation fast operation */
pkcau_mod_exp_operation(&pkcau_mod_exp_parameter,

```

```
PKCAU_MODE_MOD_EXP_FAST);
```

### 函数 pkcau\_mod\_inver\_operation

函数pkcau\_mod\_inver\_operation描述见下表:

表 3-553. 函数 pkcau\_mod\_inver\_operation

函数名称	pkcau_mod_inver_operation
函数原型	void void pkcau_mod_inver_operation(pkcau_mod_inver_parameter_struct *mod_inver_para);
功能描述	执行模逆运算
先决条件	-
被调用函数	-
输入参数{in}	
mod_inver_para	模逆参数初始化结构体, 结构体成员参考 <a href="#">表3-526. 结构体 pkcau_mod_inver_parameter_struct</a> 。
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* initialize the modular inversion parameter structure */
pkcau_mod_inver_parameter_struct pkcau_mod_inver_parameter;
pkcau_mod_inver_struct_para_init(&pkcau_mod_inver_parameter);
/* initialize the modular parameters */
pkcau_inver_exp_parameter.oprd_a = (uint8_t *)oprd_a;
pkcau_inver_exp_parameter.modulus_len = ME_MOD_SIZE;
pkcau_inver_exp_parameter.modulus_n = (uint8_t *)modulus_n;
/* execute modular inversion operation */
pkcau_mod_inver_operation(&pkcau_mod_inver_parameter);
```

### 函数 pkcau\_mod\_reduc\_operation

函数pkcau\_mod\_reduc\_operation描述见下表:

表 3-554. 函数 pkcau\_mod\_reduc\_operation

函数名称	pkcau_mod_reduc_operation
函数原型	void pkcau_mod_reduc_operation(pkcau_mod_reduc_parameter_struct *mod_reduc_para);
功能描述	执行取模运算

先决条件	-
被调用函数	-
输入参数{in}	
mod_reduc_para	取模参数初始化结构体，结构体成员参考 <a href="#">表3-527. 结构体 pkcau_mod_reduc_parameter_struct</a> 。
输出参数{out}	
-	-
返回值	
-	-

例如：

```

/* initialize the modular inversion parameter structure */

pkcau_mod_reduc_parameter_struct pkcau_mod_reduc_parameter;

pkcau_mod_reduc_struct_para_init(&pkcau_mod_reduc_parameter);

/* initialize the modular parameters */

pkcau_mod_reduc_parameter.oprd_a = (uint8_t *)oprd_a;

pkcau_mod_reduc_parameter.oprd_a_len = MA_A_SIZE;

pkcau_mod_reduc_parameter.modulus_len = MA_MOD_SIZE;

pkcau_mod_reduc_parameter.modulus_n = (uint8_t *)modulus_n;

/* execute modular reduction operation */

pkcau_mod_reduc_operation(&pkcau_mod_reduc_parameter);

```

### 函数 pkcau\_arithmetic\_operation

函数pkcau\_arithmetic\_operation描述见下表：

表 3-555. 函数 pkcau\_arithmetic\_operation

函数名称	pkcau_arithmetic_operation
函数原型	void pkcau_arithmetic_operation(pkcau_arithmetic_parameter_struct *arithmetic_para, uint32_t mode);
功能描述	执行算术运算
先决条件	-
被调用函数	-
输入参数{in}	
arithmetic_para	算术参数初始化结构体，结构体成员参考 <a href="#">表3-528. 结构体 pkcau_arithmetic_parameter_struct</a> 。
输入参数{in}	
mode	算术运算模式
PKCAU_MODE_ARITH	算术加法运算

METIC_ADD	
PKCAU_MODE_ARITH METIC_SUB	算术减法运算
PKCAU_MODE_ARITH METIC_MUL	算术乘法运算
PKCAU_MODE_ARITH METIC_COMP	算术比较运算
输出参数{out}	
-	-
返回值	
-	-

例如:

```

/* initialize the arithmetic parameter structure */

pkcau_arithmetic_parameter_struct pkcau_arithmetic_parameter;

pkcau_arithmetic_struct_para_init(&pkcau_arithmetic_parameter);

/* initialize the arithmetic addition parameters */

pkcau_ari_multi_parameter.oprd_a = (uint8_t *)oprd_a;

pkcau_ari_multi_parameter.oprd_b = (uint8_t *)oprd_b;

pkcau_ari_multi_parameter.oprd_len = AM_B_SIZE;

/* execute arithmetic addition operation */

pkcau_arithmetic_operation(&pkcau_arithmetic_parameter,
PKCAU_MODE_ARITHMETIC_ADD);

```

## 函数 pkcau\_crt\_exp\_operation

函数pkcau\_crt\_exp\_operation描述见下表:

表 3-556. 函数 pkcau\_crt\_exp\_operation

函数名称	pkcau_crt_exp_operation
函数原型	void pkcau_crt_exp_operation(pkcau_crt_parameter_struct* crt_para);
功能描述	执行RSA CRT幂运算
先决条件	-
被调用函数	-
输入参数{in}	
crt_para	CRT参数初始化结构体, 结构体成员参考 <a href="#">表3-529. 结构体 pkcau_crt_parameter_struct</a> 。
输出参数{out}	
-	-
返回值	

-	-
---	---

例如：

```
/* initialize the arithmetic parameter structure */

pkcau_crt_parameter_struct pkcau_crt_parameter;

pkcau_crt_exp_operation(&pkcau_crt_parameter);

/* initialize the input ECC curve parameters */

pkcau_crt_parameter.oprd_a    = (uint8_t *)rsa_crt_a;
pkcau_crt_parameter.oprd_dp   = (uint8_t *)rsa_crt_dp;
pkcau_crt_parameter.oprd_dq   = (uint8_t *)rsa_crt_dq;
pkcau_crt_parameter.oprd_qinv = (uint8_t *)rsa_crt_qinv;
pkcau_crt_parameter.oprd_p    = (uint8_t *)rsa_crt_p;
pkcau_crt_parameter.oprd_q    = (uint8_t *)rsa_crt_q;
pkcau_crt_parameter.oprd_len  = sizeof(rsa_crt_a);

/* execute RSA CRT exponentiation operation */

pkcau_crt_exp_operation(&pkcau_crt_parameter);
```

### 函数 pkcau\_point\_check\_operation

函数pkcau\_point\_check\_operation描述见下表：

表 3-557. 函数 pkcau\_point\_check\_operation

函数名称	pkcau_point_check_operation
函数原型	void pkcau_point_check_operation(pkcau_point_parameter_struct* point_para, const pkcau_ec_group_parameter_struct* curve_group_para);
功能描述	执行椭圆上点的检查
先决条件	-
被调用函数	-
输入参数{in}	
point_para	点参数初始化结构体，结构体成员参考 <a href="#">表3-531. 结构体 pkcau_point_parameter_struct</a> 。
输入参数{in}	
curve_group_para	ECC椭圆参数初始化结构体，结构体成员参考 <a href="#">表3-530. 结构体 pkcau_ec_group_parameter_struct</a> 。
输出参数{out}	
-	-
返回值	
-	-



例如：

```
/* initialize point parameter and ECC curve parameter structure */

pkcau_point_parameter_struct pkcau_point_parameter;

pkcau_ec_group_parameter_struct pkcau_curve_group;

pkcau_point_struct_para_init(&pkcau_point_parameter);

pkcau_ec_group_struct_para_init(&pkcau_curve_group);

/* initialize the input point parameter */

pkcau_point_check_parameter.point_x = pcheck_x;

pkcau_point_check_parameter.point_y = pcheck_y;

/* initialize the input ECC curve parameter */

pkcau_curve_group.modulus_p      = (uint8_t *)brainpoolp256r1_p;

pkcau_curve_group.coff_a         = (uint8_t *)brainpoolp256r1_a;

pkcau_curve_group.coff_b         = (uint8_t *)brainpoolp256r1_b;

pkcau_curve_group.a_sign         = 0;

pkcau_curve_group.modulus_p_len = sizeof(brainpoolp256r1_p);

/* execute point check operation */

pkcau_point_check_operation(&pkcau_point_parameter, &pkcau_curve_group);
```

### 函数 pkcau\_point\_mul\_operation

函数pkcau\_point\_mul\_operation描述见下表：

表 3-558. 函数 pkcau\_point\_mul\_operation

函数名称	pkcau_point_mul_operation
函数原型	void pkcau_point_mul_operation(pkcau_point_parameter_struct *point_para, const pkcau_ec_group_parameter_struct* curve_group_para, uint32_t mode);
功能描述	执行点乘运算
先决条件	-
被调用函数	-
输入参数{in}	
point_para	点参数初始化结构体，结构体成员参考 <a href="#">表3-531. 结构体 pkcau_point_parameter_struct</a> 。
输入参数{in}	
curve_group_para	ECC椭圆参数初始化结构体，结构体成员参考 <a href="#">表3-530. 结构体 pkcau_ec_group_parameter_struct</a> 。

输入参数{in}	
<b>mode</b>	点乘运算模式
<i>PKCAU_MODE_ECC_MUL</i>	先计算蒙哥马利参数，然后进行ECC标量乘法运算
<i>PKCAU_MODE_ECC_MUL_FAST</i>	仅进行ECC标量乘法运算
输出参数{out}	
-	-
返回值	
-	-

例如：

```

/* initialize point parameter and ECC curve parameter structure */

pkcau_point_parameter_struct pkcau_point_parameter;

pkcau_ec_group_parameter_struct pkcau_curve_group;

pkcau_point_struct_para_init(&pkcau_point_parameter);

pkcau_ec_group_struct_para_init(&pkcau_curve_group);

/* initialize the input point parameter */

pkcau_point_parameter.point_x = ec_pmul_x;

pkcau_point_parameter.point_y = ec_pmul_y;

/* initialize the input ECC curve parameter */

pkcau_curve_group.modulus_p      = (uint8_t *)brainpoolp256r1_p;

pkcau_curve_group.coff_a         = (uint8_t *)brainpoolp256r1_a;

pkcau_curve_group.modulus_p_len = sizeof(brainpoolp256r1_p);

pkcau_curve_group.a_sign         = 0;

pkcau_curve_group.multi_k        = ec_pmul_k;

pkcau_curve_group.k_len          = PMUL_K_SIZE;

/* execute scalar multiplication operation */

pkcau_point_mul_operation(&pkcau_point_parameter, &pkcau_curve_group,
PKCAU_MODE_ECC_MUL);

```

### 函数 pkcau\_ecdsa\_sign\_operation

函数pkcau\_ecdsa\_sign\_operation描述见下表：

表 3-559. 函数 pkcau\_ecdsa\_sign\_operation

函数名称	pkcau_ecdsa_sign_operation
函数原型	void pkcau_ecdsa_sign_operation(const uint8_t* p_key_d, const uint8_t* k, pkcau_hash_parameter_struct* hash_para, const pkcau_ec_group_parameter_struct* curve_group_para);
功能描述	执行ECDSA签名运算
先决条件	-
被调用函数	-
输入参数{in}	
p_key_d	私钥d
输入参数{in}	
k	整数k
输入参数{in}	
hash_para	哈希参数初始化结构体，结构体成员参考 <a href="#">表3-533. 结构体 pkcau_hash_parameter_struct</a> 。
输入参数{in}	
curve_group_para	ECC椭圆参数初始化结构体，结构体成员参考 <a href="#">表3-530. 结构体 pkcau_ec_group_parameter_struct</a> 。
输出参数{out}	
-	-
返回值	
-	-

例如：

```

/* initialize hash parameter and ECC curve parameter structure */

pkcau_hash_parameter_struct pkcau_hash_parameter;

pkcau_ec_group_parameter_struct pkcau_curve_group;

pkcau_hash_struct_para_init(&pkcau_hash_parameter);

pkcau_ec_group_struct_para_init(&pkcau_curve_group);

/* initialize the input hash parameter */

pkcau_hash_parameter.hash_z      = hash;

pkcau_hash_parameter.hash_z_len = DATA_SIZE;

/* initialize the input ECC curve parameter */

pkcau_curve_group.modulus_p      = secp112r2_p;

pkcau_curve_group.coff_a         = secp112r2_a;

pkcau_curve_group.a_sign         = 0;

pkcau_curve_group.base_point_x   = secp112r2_gx;

```

```

pkcau_curve_group.base_point_y = secp112r2_gy;

pkcau_curve_group.order_n      = secp112r2_n,

pkcau_curve_group.modulus_p_len = sizeof(secp112r2_p);

pkcau_curve_group.order_n_len  = sizeof(secp112r2_n);

/* execute ECDSA sign operation */

pkcau_ecdsa_sign_operation(d, k, &pkcau_hash_parameter, &pkcau_curve_group);

```

### 函数 pkcau\_ecdsa\_verification\_operation

函数pkcau\_ecdsa\_verification\_operation描述见下表:

**表 3-560. 函数 pkcau\_ecdsa\_verification\_operation**

函数名称	pkcau_ecdsa_verification_operation
函数原型	void pkcau_ecdsa_verification_operation(pkcau_point_parameter_struct *point_para, pkcau_hash_parameter_struct *hash_para, pkcau_signature_parameter_struct *signature_para, const pkcau_ec_group_parameter_struct* curve_group_para);
功能描述	执行ECDSA验证运算
先决条件	-
被调用函数	-
输入参数{in}	
point_para	点参数初始化结构体, 结构体成员参考 <a href="#">表3-531. 结构体 pkcau_point_parameter_struct</a> 。
输入参数{in}	
hash_para	哈希参数初始化结构体, 结构体成员参考 <a href="#">表3-533. 结构体 pkcau_hash_parameter_struct</a> 。
输入参数{in}	
signature_para	签名参数初始化结构体, 结构体成员参考 <a href="#">表3-532. 结构体 pkcau_signature_parameter_struct</a> 。
输入参数{in}	
curve_group_para	ECC椭圆参数初始化结构体, 结构体成员参考 <a href="#">表3-530. 结构体 pkcau_ec_group_parameter_struct</a> 。
输出参数{out}	
-	-
返回值	
-	-

例如:

```

/* initialize point parameter, hash parameter and ECC curve parameter structure */

pkcau_point_parameter_struct pkcau_point_parameter;

pkcau_hash_parameter_struct pkcau_hash_parameter;

```

```

pkcau_signature_parameter_struct pkcau_signature_parameter;

pkcau_ec_group_parameter_struct pkcau_curve_group;

pkcau_point_struct_para_init(&pkcau_point_parameter);

pkcau_hash_struct_para_init(&pkcau_hash_parameter);

pkcau_signature_struct_para_init(&pkcau_signature_parameter);

pkcau_ec_group_struct_para_init(&pkcau_curve_group);

/* initialize the input point parameters */

pkcau_point_parameter.point_x = ecc_verify_x;

pkcau_point_parameter.point_y = ecc_verify_y;

/* initialize the input hash parameters */

pkcau_hash_parameter.hash_z      = (uint8_t *)ecc_verify_hash;

pkcau_hash_parameter.hash_z_len = sizeof(ecc_verify_hash);

/* initialize the input ECC signature parameters */

pkcau_signature_parameter.sign_r = (uint8_t *)ecc_verify_r;

pkcau_signature_parameter.sign_s = (uint8_t *)ecc_verify_s;

/* initialize the input ECC curve parameters */

pkcau_curve_group.modulus_p      = (uint8_t *)brainpoolp256r1_p;

pkcau_curve_group.coff_a         = (uint8_t *)brainpoolp256r1_a;

pkcau_curve_group.a_sign         = 0;

pkcau_curve_group.base_point_x   = (uint8_t *)brainpoolp256r1_gx;

pkcau_curve_group.base_point_y   = (uint8_t *)brainpoolp256r1_gy;

pkcau_curve_group.order_n        = (uint8_t *)brainpoolp256r1_n;

pkcau_curve_group.modulus_p_len  = sizeof(brainpoolp256r1_p);

pkcau_curve_group.order_n_len    = sizeof(brainpoolp256r1_n);

/* execute ECDSA verification operation */

pkcau_ecdsa_verification_operation(&pkcau_point_parameter, &pkcau_hash_parameter,
&pkcau_signature_parameter, &pkcau_curve_group);

```

## 函数 pkcau\_memread

函数pkcau\_memread描述见下表:

表 3-561. 函数 `pkcau_memread`

函数名称	<code>pkcau_memread</code>
函数原型	<code>void pkcau_memread(uint32_t offset, uint8_t buf[], uint32_t size);</code>
功能描述	从PKCAU RAM读取结果
先决条件	-
被调用函数	-
输入参数{in}	
<b>offset</b>	基于PKCAU基地址的偏移地址
输入参数{in}	
<b>size</b>	要读取的字节数
输出参数{out}	
<b>buf</b>	大端结果缓冲区，PKCAU的最端左字节应该在缓冲区的第一个元素中
返回值	
-	-

例如：

```
/* read results from RAM address */
uint8_t verify_res = 0;
pkcau_memread(0x5B0, &verify_res, 1);
```

### 函数 `pkcau_flag_get`

函数`pkcau_flag_get`描述见下表：

表 3-562. 函数 `pkcau_flag_get`

函数名称	<code>pkcau_flag_get</code>
函数原型	<code>FlagStatus pkcau_flag_get(uint32_t flag);</code>
功能描述	获取 PKCAU 标志位
先决条件	-
被调用函数	-
输入参数{in}	
<b>flag</b>	PKCAU 标志位
<code>PKCAU_FLAG_ADDRE RR</code>	地址错误标志
<code>PKCAU_FLAG_RAME RR</code>	PKCAU RAM 错误标志
<code>PKCAU_FLAG_END</code>	PKCAU 运算结束标志
<code>PKCAU_FLAG_BUSY</code>	忙标志
输出参数{out}	
-	-
返回值	
<b>FlagStatus</b>	SET / RESET

例如:

```
/* get PKCAU flag status */

FlagStatus flag_state = RESET;

flag_state = pkcau_flag_get(PKCAU_FLAG_ADDRERR);
```

### 函数 pkcau\_flag\_clear

函数pkcau\_flag\_clear描述见下表:

表 3-563. 函数 pkcau\_flag\_clear

函数名称	pkcau_flag_clear
函数原型	void pkcau_flag_clear(uint32_t flag);
功能描述	清除 PKCAU 标志位
先决条件	-
被调用函数	-
输入参数{in}	
flag	PKCAU 标志位
PKCAU_FLAG_ADDRERR	地址错误标志
PKCAU_FLAG_RAMERR	PKCAU RAM 错误标志
PKCAU_FLAG_END	PKCAU 运算结束标志
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* clear address error flag*/

pkcau_flag_clear(PKCAU_FLAG_ADDRERR);
```

### 函数 pkcau\_interrupt\_enable

函数pkcau\_interrupt\_enable描述见下表:

表 3-564. 函数 pkcau\_interrupt\_enable

函数名称	pkcau_interrupt_enable
函数原型	void pkcau_enable(void);
功能描述	中断使能
先决条件	-
被调用函数	-
输入参数{in}	
interrupt	中断类型

<i>PKCAU_INT_ADDRER R</i>	地址错误中断
<i>PKCAU_INT_RAMERR</i>	PKCAU RAM错误中断
<i>PKCAU_INT_END</i>	PKCAU运算结束中断
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* enable PKCAU address error interrupt */
pkcau_interrupt_enable(PKCAU_INT_ADDRERR);
```

### 函数 pkcau\_interrupt\_disable

函数pkcau\_interrupt\_disable描述见下表：

表 3-565. 函数 pkcau\_interrupt\_disable

函数名称	pkcau_interrupt_disable
函数原型	void pkcau_interrupt_disable(void);
功能描述	中断禁能
先决条件	-
被调用函数	-
输入参数{in}	
<b>interrupt</b>	中断类型
<i>PKCAU_INT_ADDRER R</i>	地址错误中断
<i>PKCAU_INT_RAMERR</i>	PKCAU RAM错误中断
<i>PKCAU_INT_END</i>	PKCAU运算结束中断
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* disable PKCAU address error interrupt */
pkcau_interrupt_disable(PKCAU_INT_ADDRERR);
```

### 函数 pkcau\_interrupt\_flag\_get

函数pkcau\_interrupt\_flag\_get描述见下表：

表 3-566. 函数 pkcau\_interrupt\_flag\_get

函数名称	pkcau_interrupt_flag_get
------	--------------------------



函数原型	FlagStatus pkcau_interrupt_flag_get(uint32_t int_flag);
功能描述	获取 PKCAU 中断标志位
先决条件	-
被调用函数	-
输入参数{in}	
int_flag	PKCAU 中断标志
PKCAU_INT_FLAG_ADDRERR	地址错误中断标志
PKCAU_INT_FLAG_RAMERR	PKCAU RAM 错误中断标志
PKCAU_INT_FLAG_END	PKCAU 运算结束中断标志
输出参数{out}	
-	-
返回值	
FlagStatus	SET / RESET

例如：

```
/* get PKCAU interrupt flag status */
```

```
FlagStatus flag_state = RESET;
```

```
flag_state = pkcau_interrupt_flag_get(PKCAU_INT_FLAG_ADDRERR);
```

### 函数 pkcau\_interrupt\_flag\_clear

函数pkcau\_interrupt\_flag\_clear描述见下表：

表 3-567. 函数 pkcau\_interrupt\_flag\_clear

函数名称	pkcau_interrupt_flag_clear
函数原型	void pkcau_interrupt_flag_clear(uint32_t int_flag);
功能描述	清除 PKCAU 中断标志位
先决条件	-
被调用函数	-
输入参数{in}	
int_flag	PKCAU 中断标志
PKCAU_INT_FLAG_ADDRERR	地址错误中断标志
PKCAU_INT_FLAG_RAMERR	PKCAU RAM 错误中断标志
PKCAU_INT_FLAG_END	PKCAU 运算结束中断标志
输出参数{out}	
-	-
返回值	

-	-
---	---

例如:

```
/* clear address error interrupt flag*/
```

```
pkcau_interrupt_flag_clear(PKCAU_INT_FLAG_ADDRERR);
```

## 3.19. PMU

电源管理单元提供了五种省电模式，包括睡眠模式，深度睡眠模式，SRAM\_sleep 模式，WIFI\_sleep 模式和待机模式。章节 [3.19.1](#) 描述了 PMU 的寄存器列表，章节 [3.19.2](#) 对 PMU 库函数进行说明。

### 3.19.1. 外设寄存器说明

PMU 寄存器列表如下表所示:

**表 3-568. PMU 寄存器**

寄存器名称	寄存器描述
PMU_CTL0	控制寄存器0
PMU_CS0	电源控制和状态寄存器0
PMU_CTL1	控制寄存器1
PMU_CS1	电源控制和状态寄存器1
PMU_RFCTL	RF控制寄存器
PMU_SECCFG	安全配置寄存器
PMU_PRICFG	权限配置寄存器

### 3.19.2. 外设库函数说明

PMU 库函数列表如下表所示:

**表 3-569. PMU 库函数**

库函数名称	库函数描述
pmu_deinit	复位外设PMU
pmu_lvd_select	选择低压检测阈值
pmu_lvd_disable	关闭低压检测器
pmu_vlvd_enable	使能PMU VLVD
pmu_vlvd_disable	禁能PMU VLVD
pmu_ldo_output_select	选择LDO输出电压
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

库函数名称	库函数描述
pmu_backup_write_enable	备份域写使能
pmu_backup_write_disable	备份域写失能
pmu_wifi_power_enable	使能WIFI电源
pmu_wifi_power_disable	禁能WIFI电源
pmu_wifi_sram_control	WIFI & SRAM省电控制
pmu_rf_force_enable	使能RF时序强制开启/关闭
pmu_rf_force_disable	禁能RF时序强制开启/关闭
pmu_rf_sequence_config	RF时序配置
pmu_security_enable	使能安全属性配置
pmu_security_disable	禁能安全属性配置
pmu_privilege_enable	使能特权访问
pmu_privilege_disable	禁能特权访问
pmu_flag_clear	清除标志位
pmu_flag_get	获取标志位

### 函数 pmu\_deinit

函数pmu\_deinit描述见下表:

表 3-570. 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 3-571. 函数 pmu\_lvd\_select

函数名称	pmu_lvd_select
函数原型	void pmu_lvd_select(uint32_t lvdn);

功能描述	选择低压检测阈值
先决条件	-
被调用函数	-
输入参数{in}	
lvdt_n	电压阈值
PMU_LVDT_0	电压阈值为2.1V
PMU_LVDT_1	电压阈值为2.3V
PMU_LVDT_2	电压阈值为2.4V
PMU_LVDT_3	电压阈值为2.6V
PMU_LVDT_4	电压阈值为2.7V
PMU_LVDT_5	电压阈值为2.9V
PMU_LVDT_6	电压阈值为3.0V
PMU_LVDT_7	电压阈值为3.1V
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* select low voltage detector threshold as 3.1V */
```

```
pmu_lvd_select(PMU_LVDT_7);
```

### 函数 pmu\_lvd\_disable

函数pmu\_lvd\_disable描述见下表：

表 3-572. 函数 pmu\_lvd\_disable

函数名称	pmu_lvd_disable
函数原型	void pmu_lvd_disable (void);
功能描述	关闭低压检测器
先决条件	-
被调用函数	-
输入参数{in}	
-	-
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* disable PMU lvd */
```

```
pmu_lvd_disable();
```

## 函数 pmu\_vlvd\_enable

函数pmu\_vlvd\_enable描述见下表：

**表 3-573. 函数 pmu\_vlvd\_enable**

函数名称	pmu_vlvd_enable
函数原型	void pmu_vlvd_enable(void);
功能描述	使能PMU VLVD
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 PMU VLVD */
pmu_vlvd_enable();
```

## 函数 pmu\_vlvd\_disable

函数pmu\_vlvd\_disable描述见下表：

**表 3-574. 函数 pmu\_vlvd\_disable**

函数名称	pmu_vlvd_disable
函数原型	void pmu_vlvd_disable(void);
功能描述	禁能PMU VLVD
先决条件	-
被调用函数	-
输入参数{in}	
-	-
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* disable PMU VLVD */
pmu_vlvd_disable();
```

函数 **pmu\_ldo\_output\_select**

函数pmu\_ldo\_output\_select描述见下表：

表 3-575. 函数 **pmu\_ldo\_output\_select**

函数名称	pmu_ldo_output_select
函数原型	void pmu_ldo_output_select(uint32_t ldo_output);
功能描述	选择LDO输出电压
先决条件	-
被调用函数	-
输入参数{in}	
ldo_output	LDO输出电压模式
PMU_LDOVS_LOW	LDO输出低电压模式
PMU_LDOVS_HIGH	LDO输出高电压模式
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* LDO output low voltage */
pmu_ldo_output_select(PMU_LDOVS_LOW);
```

函数 **pmu\_to\_sleepmode**

函数pmu\_to\_sleepmode描述见下表：

表 3-576. 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 3-577. 函数 pmu\_to\_deepsleepmode

函数名称	pmu_to_deepsleepmode
函数原型	void pmu_to_deepsleepmode(uint32_t ldo, uint32_t lowdrive, uint8_t deepsleepmodecmd);
功能描述	进入深度睡眠模式
先决条件	-
被调用函数	-
输入参数{in}	
ldo	LDO工作模式
PMU_LDO_NORMAL	当系统进入深度睡眠模式时，LDO仍正常工作
PMU_LDO_LOWPOWER	当系统进入深度睡眠模式时，LDO进入低功耗模式
输入参数{in}	
lowdrive	LDO低驱动模式
PMU_LOWDRIVER_DISABLE	当系统进入深度睡眠模式时，LDO禁能低驱动模式
PMU_LOWDRIVER_ENABLE	当系统进入深度睡眠模式时，LDO使能低驱动模式
输入参数{in}	
deepsleepmodecmd	进入深度睡眠模式命令
WFI_CMD	WFI命令
WFE_CMD	WFE命令
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* PMU work at deepsleep mode */
```

```
pmu_to_deepsleepmode(PMU_LDO_NORMAL, PMU_LOWDRIVER_ENABLE, WFI_CMD);
```

### 函数 pmu\_to\_standbymode

函数pmu\_to\_standbymode描述见下表：

表 3-578. 函数 pmu\_to\_standbymode

函数名称	pmu_to_standbymode
函数原型	void pmu_to_standbymode(void);
功能描述	进入待机模式
先决条件	-
被调用函数	-
输入参数{in}	
-	-
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* PMU work at standby mode */
pmu_to_standbymode();
```

### 函数 pmu\_wakeup\_pin\_enable

函数pmu\_wakeup\_pin\_enable描述见下表：

表 3-579. 函数 pmu\_wakeup\_pin\_enable

函数名称	pmu_wakeup_pin_enable
函数原型	void pmu_wakeup_pin_enable(uint32_t wakeup_pin);
功能描述	WKUP引脚唤醒使能
先决条件	-
被调用函数	-
输入参数{in}	
wakeup_pin	唤醒系统的管脚
PMU_WAKEUP_PIN0	唤醒脚0 (PA2)
PMU_WAKEUP_PIN1	唤醒脚1 (PA15)
PMU_WAKEUP_PIN2	唤醒脚2 (PB2)
PMU_WAKEUP_PIN3	唤醒脚3 (PA12)
输出参数{out}	
-	-
返回值	
-	-

例如：



```
/* enable wakeup pin 0 */
```

```
pmu_wakeup_pin_enable(PMU_WAKEUP_PIN0);
```

### 函数 pmu\_wakeup\_pin\_disable

函数pmu\_wakeup\_pin\_disable描述见下表：

**表 3-580. 函数 pmu\_wakeup\_pin\_disable**

函数名称	pmu_wakeup_pin_disable
函数原型	void pmu_wakeup_pin_disable (uint32_t wakeup_pin);
功能描述	WKUP引脚唤醒失能
先决条件	-
被调用函数	-
输入参数{in}	
wakeup_pin	唤醒系统的管脚
PMU_WAKEUP_PIN0	唤醒脚0 (PA2)
PMU_WAKEUP_PIN1	唤醒脚1 (PA15)
PMU_WAKEUP_PIN2	唤醒脚2 (PB2)
PMU_WAKEUP_PIN3	唤醒脚3 (PA12)
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* disable wakeup pin 0 */
```

```
pmu_wakeup_pin_disable(PMU_WAKEUP_PIN0);
```

### 函数 pmu\_backup\_write\_enable

函数pmu\_backup\_write\_enable描述见下表：

**表 3-581. 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 3-582. 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\_wifi\_power\_enable

函数pmu\_wifi\_power\_enable描述见下表：

表 3-583. 函数 pmu\_wifi\_power\_enable

函数名称	pmu_wifi_power_enable
函数原型	void pmu_wifi_power_enable(void);
功能描述	使能WIFI电源
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 WIFI power */
```

```
pmu_wifi_power_enable();
```

### 函数 pmu\_wifi\_power\_disable

函数pmu\_wifi\_power\_disable描述见下表：

**表 3-584. 函数 pmu\_wifi\_power\_disable**

函数名称	pmu_wifi_power_disable
函数原型	void pmu_wifi_power_disable(void);
功能描述	禁能WIFI电源
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 WIFI power */
```

```
pmu_wifi_power_disable();
```

### 函数 pmu\_wifi\_sram\_control

函数pmu\_wifi\_sram\_control描述见下表：

**表 3-585. 函数 pmu\_wifi\_sram\_control**

函数名称	pmu_wifi_sram_control
函数原型	void pmu_wifi_sram_control(uint32_t wifi_sram);
功能描述	WIFI & SRAM省电控制
先决条件	-
被调用函数	-
输入参数{in}	
wifi_sram	WIFI或SRAM睡眠控制
PMU_WIFI_SLEEP	WIFI进入睡眠
PMU_WIFI_WAKE	WIFI唤醒
PMU_SRAM1_SLEEP	SRAM1进入睡眠

<i>PMU_SRAM1_WAKE</i>	SRAM1唤醒
<i>PMU_SRAM2_SLEEP</i>	SRAM2进入睡眠
<i>PMU_SRAM2_WAKE</i>	SRAM2唤醒
<i>PMU_SRAM3_SLEEP</i>	SRAM3进入睡眠
<i>PMU_SRAM3_WAKE</i>	SRAM3唤醒
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* SRAM2 go to sleep */
```

```
pmu_wifi_sram_control(PMU_SRAM2_SLEEP);
```

### 函数 pmu\_rf\_force\_enable

函数pmu\_rf\_force\_enable描述见下表：

表 3-586. 函数 pmu\_rf\_force\_enable

函数名称	pmu_rf_force_enable
函数原型	void pmu_rf_force_enable(uint32_t force);
功能描述	使能RF时序强制开启/关闭
先决条件	-
被调用函数	-
输入参数{in}	
<b>force</b>	使能RF电源控制
<i>PMU_RF_FORCE_OPEN</i>	软件强制启动，打开RF电源
<i>PMU_RF_FORCE_CLOSE</i>	软件强制关闭RF电源
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* enable open RF power */
```

```
pmu_rf_force_enable(PMU_RF_FORCE_OPEN);
```

## 函数 pmu\_rf\_force\_disable

函数pmu\_rf\_force\_disable描述见下表：

**表 3-587. 函数 pmu\_rf\_force\_disable**

函数名称	pmu_rf_force_disable
函数原型	void pmu_rf_force_disable(uint32_t force);
功能描述	禁能RF时序强制开启/关闭
先决条件	-
被调用函数	-
输入参数{in}	
force	禁能RF电源控制
PMU_RF_FORCE_OPEN	软件强制启动，打开RF电源
PMU_RF_FORCE_CLOSE	软件强制关闭RF电源
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* disable open RF power */
```

```
pmu_rf_force_disable(PMU_RF_FORCE_OPEN);
```

## 函数 pmu\_rf\_sequence\_config

函数pmu\_rf\_sequence\_config描述见下表：

**表 3-588. 函数 pmu\_rf\_sequence\_config**

函数名称	pmu_rf_sequence_config
函数原型	void pmu_rf_sequence_config(uint32_t mode);
功能描述	RF时序配置
先决条件	-
被调用函数	-
输入参数{in}	
mode	RF时序模式
PMU_RF_SOFTWARE	RF时序软件控制使能
PMU_RF_HARDWARE	RF时序硬件控制使能
输出参数{out}	
-	-
返回值	

-	-
---	---

例如:

```
/* enable RF hardware sequence */
```

```
pmu_rf_sequence_config(PMU_RF_HARDWARE);
```

### 函数 pmu\_security\_enable

函数pmu\_security\_enable描述见下表:

表 3-589. 函数 pmu\_security\_enable

函数名称	pmu_security_enable
函数原型	void pmu_security_enable(uint32_t security);
功能描述	使能安全属性配置
先决条件	-
被调用函数	-
输入参数{in}	
security	安全属性配置
PMU_SEC_WAKEUP_P_PIN0	WKUP脚0安全
PMU_SEC_WAKEUP_P_PIN1	WKUP脚1安全
PMU_SEC_WAKEUP_P_PIN2	WKUP脚2安全
PMU_SEC_WAKEUP_P_PIN3	WKUP脚3安全
PMU_SEC_LPLDO_DPSLP_STB	省电模式功耗
PMU_SEC_LVD_VLVD	电压监测安全
PMU_SEC_BACKUP_P_WRITE	备份域写保护安全
PMU_SEC_WIFI_SRAM_CONTROL	WIFI_sleep和SRAM_sleep模式安全
PMU_SEC_RF_SEQUENCE	RF安全
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* enable WKUP pin 0 security */
```

```
pmu_security_enable(PMU_SEC_WAKEUP_PIN0);
```

### 函数 pmu\_security\_disable

函数pmu\_security\_disable描述见下表:

表 3-590. 函数 pmu\_security\_disable

函数名称	pmu_security_disable
函数原型	void pmu_security_disable(uint32_t security);
功能描述	禁能安全属性配置
先决条件	-
被调用函数	-
输入参数{in}	
security	安全属性配置
PMU_SEC_WAKEUP_PIN0	WKUP脚0安全
PMU_SEC_WAKEUP_PIN1	WKUP脚1安全
PMU_SEC_WAKEUP_PIN2	WKUP脚2安全
PMU_SEC_WAKEUP_PIN3	WKUP脚3安全
PMU_SEC_LPLDO_DPSLP_STB	省电模式功耗
PMU_SEC_LVD_VLVD	电压监测安全
PMU_SEC_BACKUP_WRITE	备份域写保护安全
PMU_SEC_WIFI_SRAM_CONTROL	WIFI_sleep和SRAM_sleep模式安全
PMU_SEC_RF_SEQUENCE	RF安全
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* disable WKUP pin 0 security */
```

```
pmu_security_disable(PMU_SEC_WAKEUP_PIN0);
```

### 函数 pmu\_privilege\_enable

函数pmu\_privilege\_enable描述见下表:

表 3-591. 函数 pmu\_privilege\_enable

函数名称	pmu_privilege_enable
函数原型	void pmu_privilege_enable(void);
功能描述	使能特权访问
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 privileged access only */
pmu_privilege_enable();
```

### 函数 pmu\_privilege\_disable

函数pmu\_privilege\_disable描述见下表:

表 3-592. 函数 pmu\_privilege\_disable

函数名称	pmu_privilege_disable
函数原型	void pmu_privilege_disable(void);
功能描述	禁能特权访问
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 privileged access only */
pmu_privilege_disable();
```

### 函数 pmu\_flag\_reset

函数pmu\_flag\_reset描述见下表:

表 3-593. 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	清除标志位
先决条件	-
被调用函数	-
输入参数{in}	
flag_reset	标志位
PMU_FLAG_RESE T_WAKEUP	清除唤醒标志
PMU_FLAG_RESE T_STANDBY	清除待机标志
输出参数{out}	
-	
返回值	
-	

例如：

```
/* clear flag bit */
```

```
pmu_flag_reset(PMU_FLAG_RESET_WAKEUP);
```

### 函数 pmu\_flag\_get

函数pmu\_flag\_get描述见下表：

表 3-594. 函数 pmu\_flag\_get

函数名称	pmu_flag_get
函数原型	FlagStatus pmu_flag_get(uint32_t flag_reset);
功能描述	获取标志位
先决条件	-
被调用函数	-
输入参数{in}	
flag	标志位
PMU_FLAG_RESE T_WAKEUP	唤醒标志
PMU_FLAG_RESE T_STANDBY	待机标志
PMU_FLAG_LVD	低电压检测标志
PMU_FLAG_VLVD	VDDA低电压检测标志
PMU_FLAG_LDOV SRF	LDO电压选择就绪标志
PMU_FLAG_LDRF	低驱动就绪标志
PMU_FLAG_WIFI_ SLEEP	WIFI睡眠状态标志
PMU_FLAG_WIFI_	WIFI工作状态标志

<i>ACTIVE</i>	
<i>PMU_FLAG_SRAM1_SLEEP</i>	SRAM1睡眠状态标志
<i>PMU_FLAG_SRAM1_ACTIVE</i>	SRAM1工作状态标志
<i>PMU_FLAG_SRAM2_SLEEP</i>	SRAM2睡眠状态标志
<i>PMU_FLAG_SRAM2_ACTIVE</i>	SRAM2工作状态标志
<i>PMU_FLAG_SRAM3_SLEEP</i>	SRAM3睡眠状态标志
<i>PMU_FLAG_SRAM3_ACTIVE</i>	SRAM3工作状态标志
输出参数{out}	
-	-
返回值	
<b>FlagStatus</b>	SET或RESET

例如：

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

```
FlagStatus status;
```

```
status = pmu_flag_get(PMU_FLAG_RESET_STANDBY);
```

## 3.20. QSPI

QSPI是一种专用于和Flash存储器通信的接口，可以支持单线，双线，四线SPI FLASH。章节[3.20.1](#)描述了QSPI的寄存器列表，章节[3.20.2](#)对QSPI库函数进行说明。

### 3.20.1. 外设寄存器说明

QSPI寄存器列表如下表所示：

**表 3-595. SPI/I2S 寄存器**

寄存器名称	寄存器描述
QSPI_CTL	控制寄存器
QSPI_DCFG	设备配置寄存器
QSPI_STAT	状态寄存器
QSPI_STATC	状态清除寄存器
QSPI_DTLEN	数据长度寄存器
QSPI_TCFG	传输配置寄存器
QSPI_ADDR	地址寄存器
QSPI_ALTE	交替字节寄存器
QSPI_DATA	数据寄存器
QSPI_STATMK	状态屏蔽寄存器
QSPI_STATMATCH	状态匹配寄存器
QSPI_INTERVAL	间隔寄存器
QSPI_TMOUT	超时寄存器
QSPI_FLUSH	FIFO刷新寄存器
QSPI_WTCNT	间接写模式等待计数器寄存器
QSPI_SPTMOUT	状态轮询模式超时寄存器
QSPI_FMC_SECCFG	FMC模式安全配置寄存器
QSPI_CTLF	FMC模式控制寄存器
QSPI_TCFGF	FMC模式传输配置寄存器
QSPI_ALTEF	FMC模式交替字节寄存器
QSPI_BYTE_CNT	字节计数器寄存器
QSPI_PRIVCFG	特权配置寄存器
QSPI_STAT_SEC	FMC模式下安全状态寄存器
QSPI_STATC_SEC	FMC模式下安全状态清除寄存器
QSPI_DTLEN_SEC	FMC模式下安全数据长度寄存器
QSPI_TCFG_SEC	FMC模式下安全传输配置寄存器
QSPI_ADDR_SEC	FMC模式下安全地址寄存器
QSPI_ALTE_SEC	FMC模式下安全交替字节寄存器
QSPI_DATA_SEC	FMC模式下安全数据寄存器

## 3.20.2. 外设库函数说明

QSPI库函数列表如下表所示：

表 3-596. QSPI 库函数

库函数名称	库函数描述
qspi_deinit	复位外设QSPI
qspi_init_struct_para_init	将QSPI初始化结构体中所有参数初始化为默认值
qspi_init	初始化外设QSPI
qspi_enable	使能外设QSPI
qspi_disable	失能外设QSPI
qspi_dma_enable	使能QSPI DMA功能
qspi_dma_disable	失能QSPI DMA功能
qspi_command	QSPI命令参数配置
qspi_transmit	QSPI数据发送
qspi_receive	QSPI数据接收
qspi_autopolling	QSPI轮询模式配置
qspi_memorymapped	QSPI内存映射模式配置
qspi_abort	终止当前传输
qspi_command_fmc_s	FMC模式下QSPI命令参数配置
qspi_transmit_fmc_s	FMC模式下QSPI数据发送
qspi_receive_fmc_s	FMC模式下QSPI数据接收
qspi_autopolling_fmc_s	FMC模式下QSPI轮询模式配置
qspi_memorymapped_fmc_s	FMC模式下QSPI内存映射模式配置
qspi_interrupt_enable	QSPI中断使能
qspi_interrupt_disable	QSPI中断失能
qspi_flag_get	QSPI状态获取
qspi_flag_clear	QSPI状态清除

## 结构体 qspi\_init\_struct

表 3-597. 结构体 qspi\_init\_struct

成员名称	功能描述
prescaler	QSPI分频 (0x00 - 0xFF)
fifothreshold	QSPI FIFO阈值 (0x01 - 0x1F)
sampleshift	QSPI采样移位 (QSPI_SAMPLE_SHIFTING_NONE, QSPI_SAMPLE_SHIFTING_HALFCYCLE, QSPI_SAMPLE_SHIFTING_ONECYCLE)
flashsize	外部flash大小 (0x00 - 0x1F)

成员名称	功能描述
chipselecthightime	两个命令序列之间cs信号保持高电平的周期数 (QSPI_CS_HIGH_TIME_n_CYCLE (n=1,2,...,8))
clockmode	QSPI时钟模式 (QSPI_CLOCK_MODE_0, QSPI_CLOCK_MODE_3)

### 结构体 `qspi_command_struct`

表 3-598. 结构体 `qspi_command_struct`

成员名称	功能描述
instructionmode	命令模式 (QSPI_INSTRUCTION_NONE, QSPI_INSTRUCTION_1_LINE, QSPI_INSTRUCTION_2_LINES, QSPI_INSTRUCTION_4_LINES)
instruction	8 位命令 (0x00-0xFF)
addressmode	地址模式 (QSPI_ADDRESS_NONE, QSPI_ADDRESS_1_LINE, QSPI_ADDRESS_2_LINES, QSPI_ADDRESS_4_LINES)
addresssize	地址大小 (QSPI_ADDRESS_8_BITS, QSPI_ADDRESS_16_BITS, QSPI_ADDRESS_24_BITS, QSPI_ADDRESS_32_BITS)
address	外部存储地址
alternatebytemode	交替字节模式 (QSPI_ALTERNATE_BYTES_NONE, QSPI_ALTERNATE_BYTES_1_LINE, QSPI_ALTERNATE_BYTES_2_LINES, QSPI_ALTERNATE_BYTES_4_LINES)
addresssize	交替字节大小 (QSPI_ADDRESS_8_BITS, QSPI_ADDRESS_16_BITS, QSPI_ADDRESS_24_BITS, QSPI_ADDRESS_32_BITS)
alternatebytes	交替字节信息
dummycycles	空闲周期 (0x00 - 0x1F)
datamode	数据模式 (QSPI_DATA_NONE, QSPI_DATA_1_LINE, QSPI_DATA_2_LINES, QSPI_DATA_4_LINES)
nbddata	32 位数据长度
sioomode	只发送一次指令模式 (QSPI_SIOO_INST_EVERY_CMD, QSPI_SIOO_INST_ONLY_FIRST_CMD)

结构体 `qspi_autopolling_struct`表 3-599. 结构体 `qspi_autopolling_struct`

成员名称	功能描述
match	匹配值 (0x0-0xFFFFFFFF)
mask	屏蔽值 (0x0-0xFFFFFFFF)
interval	两次状态轮询之间的时钟周期数 (0x0-0xFFFF)
stautsbytessize	接收状态字节大小 (0x01-0x04)
matchmode	匹配方法 (QSPI_MATCH_MODE_AND, QSPI_MATCH_MODE_OR)
automaticstop	匹配后是否自动终止状态轮询 (QSPI_AUTOMATIC_STOP_DISABLE, QSPI_AUTOMATIC_STOP_ENABLE)

函数 `qspi_deinit`

函数`qspi_deinit`描述见下表：

表 3-600. 函数 `qspi_deinit`

函数名称	<code>qspi_deinit</code>
函数原形	<code>void qspi_deinit(void)</code>
功能描述	复位外设QSPI
先决条件	-
被调用函数	<code>rcu_periph_reset_enable</code> / <code>rcu_periph_reset_disable</code>
输入参数{in}	
-	-
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* reset QSPI */
```

```
qspi_deinit();
```

函数 `qspi_init_struct_para_init`

函数`qspi_init_struct_para_init`描述见下表：

表 3-601. 函数 `qspi_init_struct_para_init`

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

函数原形	void qspi_init_struct_para_init(qspi_init_struct* qspi_struct);
功能描述	将QSPI结构体参数初始化为默认值
先决条件	-
被调用函数	-
输入参数{in}	
qspi_struct	QSPI初始化结构体，结构体成员参考 <a href="#">表3-597. 结构体qspi_init_struct</a> 。
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* initialize the parameters of QSPI */
qspi_parameter_struct qspi_init_struct;
qspi_struct_para_init(&qspi_init_struct);
```

### 函数 qspi\_init

函数qspi\_init描述见下表：

表 3-602. 函数 qspi\_init

函数名称	qspi_init
函数原形	void qspi_init(qspi_init_struct* qspi_struct);
功能描述	初始化外设QSPI
先决条件	-
被调用函数	-
输入参数{in}	
qspi_struct	QSPI初始化结构体，结构体成员参考 <a href="#">表3-597. 结构体qspi_init_struct</a> 。
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* initialize QSPI */
qspi_init_struct qspi_struct;
qspi_struct.clockmode = QSPI_CLOCK_MODE_0;
qspi_struct.prescaler = 3;
qspi_struct.fifothreshold = 16;
qspi_struct.sampleshift = QSPI_SAMPLE_SHIFTING_NONE;
```

```
qspi_struct.flashsize = 0x1F;
```

```
qspi_struct.chipselecthightime = QSPI_CS_HIGH_TIME_1_CYCLE;
```

```
qspi_init(&qspi_struct);
```

### 函数 qspi\_enable

函数qspi\_enable描述见下表:

表 3-603. 函数 qspi\_enable

函数名称	qspi_enable
函数原形	void qspi_enable(void);
功能描述	使能外设QSPI
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 QSPI */  
qspi_enable();
```

### 函数 qspi\_disable

函数qspi\_disable描述见下表:

表 3-604. 函数 qspi\_disable

函数名称	qspi_disable
函数原形	void qspi_disable(void);
功能描述	失能外设QSPI
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 QSPI */
```

```
qspi_disable();
```

### 函数 qspi\_dma\_enable

函数qspi\_dma\_enable描述见下表:

表 3-605. 函数 qspi\_dma\_enable

函数名称	qspi_dma_enable
函数原形	void qspi_dma_enable(void);
功能描述	使能QSPI DMA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 QSPI DMA function */
```

```
qspi_dma_enable();
```

### 函数 qspi\_dma\_disable

函数qspi\_dma\_disable描述见下表:

表 3-606. 函数 qspi\_dma\_disable

函数名称	qspi_dma_disable
函数原形	void qspi_dma_disable(void);
功能描述	失能QSPI DMA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 QSPI DMA function */
```

```
qspi_dma_disable();
```

### 函数 `qspi_command`

函数`qspi_command`描述见下表：

表 3-607. 函数 `qspi_command`

函数名称	<code>qspi_command</code>
函数原形	<code>void qspi_command(qspi_command_struct* command_struct);</code>
功能描述	发送QSPI命令
先决条件	-
被调用函数	-
输入参数{in}	
<code>command_struct</code>	QSPI命令结构体，结构体成员参考 <a href="#">表3-598. 结构体 <code>qspi_command_struct</code></a> 。
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* send QSPI command */

qspi_command_struct command;

command.instruction_mode = QSPI_INSTRUCTION_1_LINE;

command.instruction = RDID;

command.addressmode = QSPI_ADDRESS_NONE;

command.addr_size = QSPI_ADDRESS_SIZE_8_BITS;

command.address = 0;

command.alternatebytemode = QSPI_ALTERNATE_BYTES_1_LINE;

command.alternatebytessize = QSPI_ALTERNATE_BYTES_24_BITS;

command.alter = 0;

command.dummy_cycles = 0;

command.data_mode = QSPI_DATA_1_LINE;

command.data_length = 3;

command.Sioomode = QSPI_SIOO_INST_EVERY_CMD;

qspi_command(&command);
```

**函数 qspi\_transmit**

函数qspi\_transmit描述见下表:

**表 3-608. 函数 qspi\_transmit**

函数名称	qspi_transmit
函数原形	void qspi_transmit(uint8_t *tdata);
功能描述	QSPI发送数据
先决条件	-
被调用函数	-
输入参数{in}	
tdata	发送数据
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* QSPI transmit data */
qspi_transmit(&tdata);
```

**函数 qspi\_receive**

函数qspi\_receive描述见下表:

**表 3-609. 函数 qspi\_receive**

函数名称	qspi_receive
函数原形	void qspi_receive(uint8_t *rdata)
功能描述	QSPI接收数据
先决条件	-
被调用函数	-
输入参数{in}	
rdata	接收数据
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* QSPI receive data */
qspi_receive(&rdata);
```

## 函数 qspi\_autopolling

函数qspi\_autopolling描述见下表：

表 3-610. 函数 qspi\_autopolling

函数名称	qspi_autopolling
函数原形	void qspi_autopolling(qspi_command_struct *cmd, qspi_autopolling_struct *cfg);
功能描述	配置QSPI状态轮询模式
先决条件	-
被调用函数	-
输入参数{in}	
cmd	QSPI命令结构体，结构体成员参考 <a href="#">表3-598. 结构体 qspi_command_struct</a> 。
输入参数{in}	
cfg	QSPI命令结构体，结构体成员参考 <a href="#">表3-599. 结构体 qspi_autopolling_struct</a> 。
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* config QSPI autopolling */
qspi_command_struct command;
qspi_autopolling_struct autopolling_cmd;
command.instruction_mode = QSPI_INSTRUCTION_1_LINE;
command.instruction = RDID;
command.addressmode = QSPI_ADDRESS_NONE;
command.addr_size = QSPI_ADDRESS_SIZE_8_BITS;
command.address = 0;
command.alternatebytemode = QSPI_ALTERNATE_BYTES_1_LINE;
command.alternatebytessize = QSPI_ALTERNATE_BYTES_24_BITS;
command.alter = 0;
command.dummy_cycles = 0;
command.data_mode = QSPI_DATA_1_LINE;
command.data_length = 3;
```

```

command.Sioomode = QSPI_SIOO_INST_EVERY_CMD;

autopolling_cmd.match          = 0x02;

autopolling_cmd.mask           = 0x02;

autopolling_cmd.matchmode      = QSPI_MATCH_MODE_AND;

autopolling_cmd.statusbytesize = 1;

autopolling_cmd.interval       = 0x10;

autopolling_cmd.automaticstop  = QSPI_AUTOMATIC_STOP_ENABLE;

qspi_autopolling(&command, &autopolling_cmd);

```

### 函数 qspi\_memorymapped

函数qspi\_memorymapped描述见下表：

表 3-611. 函数 qspi\_memorymapped

函数名称	qspi_memorymapped
函数原形	void qspi_memorymapped(qspi_command_struct *cmd, uint16_t timeout, uint32_t toen);
功能描述	配置QSPI内存映射模式
先决条件	-
被调用函数	-
输入参数{in}	
cmd	QSPI命令结构体，结构体成员参考 <a href="#">结构体qspi_command_struct</a> 。
输入参数{in}	
timeout	命令超时值 (0-0xFFFF)
输入参数{in}	
toen	超时计数使能。
QSPI_TOEN_DISABLE	超时计数失能
QSPI_TOEN_ENABLE	超时计数使能
输出参数{out}	
-	-
返回值	
-	-

例如：

```

/* config QSPI memorymapped */
qspi_command_struct command;

command.instruction_mode = QSPI_INSTRUCTION_1_LINE;

```

```

command.instruction = RDID;

command.addressmode = QSPI_ADDRESS_NONE;

command.addr_size = QSPI_ADDRESS_SIZE_8_BITS;

command.address = 0;

command.alternatebytemode = QSPI_ALTERNATE_BYTES_1_LINE;

command.alternatebytessize = QSPI_ALTERNATE_BYTES_24_BITS;

command.alter = 0;

command.dummy_cycles = 0;

command.data_mode = QSPI_DATA_1_LINE;

command.data_length = 3;

command.Sioomode = QSPI_SIOO_INST_EVERY_CMD;

qspi_memorymapped (&command,0xff, QSPI_TOEN_ENABLE);

```

### 函数 qspi\_abort

函数qspi\_abort描述见下表:

表 3-612. 函数 qspi\_abort

函数名称	qspi_abort
函数原形	void qspi_abort(void);
功能描述	终止当前传输
先决条件	-
被调用函数	-
输入参数{in}	
-	-
输出参数{out}	
-	-
返回值	
-	-

例如:

```

/* abort QSPI */

qspi_abort();

```

### 函数 qspi\_command\_fmc\_s

函数qspi\_command\_fmc\_s描述见下表:

表 3-613. 函数 `qspi_command_fmc_s`

函数名称	<code>qspi_command_fmc_s</code>
函数原形	<code>void qspi_command_fmc_s(qspi_command_struct* command_struct);</code>
功能描述	FMC模式下配置QSPI命令
先决条件	-
被调用函数	-
输入参数{in}	
<code>command_struct</code>	QSPI命令结构体，结构体成员参考 <a href="#">表3-598. 结构体 <code>qspi_command_struct</code></a> 。
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* config QSPI command */
qspi_command_struct command;

command.instruction_mode = QSPI_INSTRUCTION_1_LINE;
command.instruction = RDID;
command.addressmode = QSPI_ADDRESS_NONE;
command.addr_size = QSPI_ADDRESS_SIZE_8_BITS;
command.address = 0;
command.alternatebytemode = QSPI_ALTERNATE_BYTES_1_LINE;
command.alternatebytessize = QSPI_ALTERNATE_BYTES_24_BITS;
command.alter = 0;
command.dummy_cycles = 0;
command.data_mode = QSPI_DATA_1_LINE;
command.data_length = 3;
command.Sioomode = QSPI_SIOO_INST_EVERY_CMD;
qspi_command_fmc_s(&command);
```

### 函数 `qspi_transmit_fmc_s`

函数`qspi_transmit_fmc_s`描述见下表：

表 3-614. 函数 `qspi_transmit_fmc_s`

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

函数原形	void qspi_transmit_fmc_s(uint8_t *tdata);
功能描述	FMC模式下QSPI发送数据
先决条件	-
被调用函数	-
输入参数{in}	
tdata	发送数据
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* QSPI transmit data */
qspi_transmit_fmc_s(&tdata);
```

### 函数 qspi\_receive\_fmc\_s

函数qspi\_receive\_fmc\_s描述见下表:

表 3-615. 函数 qspi\_receive\_fmc\_s

函数名称	qspi_receive_fmc_s
函数原形	void qspi_receive_fmc_s(uint8_t *rdata)
功能描述	FMC模式下QSPI接收数据
先决条件	-
被调用函数	-
输入参数{in}	
rdata	接收数据
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* QSPI receive data */
qspi_receive_fmc_s(&rdata);
```

### 函数 qspi\_autopolling\_fmc\_s

函数qspi\_autopolling\_fmc\_s描述见下表:

表 3-616. 函数 qspi\_autopolling\_fmc\_s

函数名称	qspi_autopolling_fmc_s
函数原形	void qspi_autopolling_fmc_s(qspi_command_struct *cmd, qspi_autopolling_struct *cfg);



功能描述	FMC模式下配置QSPI状态轮询模式
先决条件	-
被调用函数	-
输入参数{in}	
cmd	QSPI命令结构体，结构体成员参考 <a href="#">表3-598. 结构体 <i>qspi_command_struct</i></a> 。
输入参数{in}	
cfg	QSPI命令结构体，结构体成员参考 <a href="#">表3-599. 结构体 <i>qspi_autopolling_struct</i></a> 。
输出参数{out}	
-	-
返回值	
-	-

例如：

```

/* config QSPI autopolling */
qspi_command_struct command;
qspi_autopolling_struct autopolling_cmd;
command.instruction_mode = QSPI_INSTRUCTION_1_LINE;
command.instruction = RDID;
command.addressmode = QSPI_ADDRESS_NONE;
command.addr_size = QSPI_ADDRESS_SIZE_8_BITS;
command.address = 0;
command.alternatebytemode = QSPI_ALTERNATE_BYTES_1_LINE;
command.alternatebytessize = QSPI_ALTERNATE_BYTES_24_BITS;
command.alter = 0;
command.dummy_cycles = 0;
command.data_mode = QSPI_DATA_1_LINE;
command.data_length = 3;
command.Sioomode = QSPI_SIOO_INST_EVERY_CMD;
autopolling_cmd.match          = 0x02;
autopolling_cmd.mask           = 0x02;
autopolling_cmd.matchmode      = QSPI_MATCH_MODE_AND;
autopolling_cmd.statusbytessize = 1;

```

```

autopolling_cmd.interval      = 0x10;

autopolling_cmd.automaticstop  = QSPI_AUTOMATIC_STOP_ENABLE;

qspi_autopolling_fmc_s (&command, &autopolling_cmd);

```

### 函数 qspi\_memorymapped\_fmc\_s

函数qspi\_memorymapped\_fmc\_s描述见下表：

表 3-617. 函数 qspi\_memorymapped\_fmc\_s

函数名称	qspi_memorymapped_fmc_s
函数原形	void qspi_memorymapped_fmc_s(qspi_command_struct *cmd, uint16_t timeout, uint32_t toen);
功能描述	配置QSPI内存映射模式
先决条件	-
被调用函数	-
输入参数{in}	
cmd	QSPI命令结构体，结构体成员参考 <a href="#">表3-598. 结构体 qspi_command_struct</a> 。
输入参数{in}	
timeout	命令超时值 (0-0xFFFF)
输入参数{in}	
toen	超时计数使能。
QSPI_TOEN_DISABLE	超时计数失能
QSPI_TOEN_ENABLE	超时计数使能
输出参数{out}	
-	-
返回值	
-	-

例如：

```

/* config QSPI memorymapped */

qspi_command_struct command;

command.instruction_mode = QSPI_INSTRUCTION_1_LINE;

command.instruction = RDID;

command.addressmode = QSPI_ADDRESS_NONE;

command.addr_size = QSPI_ADDRESS_SIZE_8_BITS;

command.address = 0;

```

```

command.alternatebytemode = QSPI_ALTERNATE_BYTES_1_LINE;

command.alternatebytessize = QSPI_ALTERNATE_BYTES_24_BITS;

command.alter = 0;

command.dummy_cycles = 0;

command.data_mode = QSPI_DATA_1_LINE;

command.data_length = 3;

command.sioomode = QSPI_SIOO_INST_EVERY_CMD;

qspi_memorymapped (&command,0xff, QSPI_TOEN_ENABLE);

```

### 函数 qspi\_interrupt\_enable

函数qspi\_interrupt\_enable描述见下表：

**表 3-618. 函数 qspi\_interrupt\_enable**

函数名称	qspi_interrupt_enable
函数原形	void qspi_interrupt_enable( uint8_t interrupt);
功能描述	使能外设QSPI中断
先决条件	-
被调用函数	-
输入参数{in}	
interrupt	QSPI中断
QSPI_INT_TC	传输完成中断
QSPI_INT_FT	FIFO阈值中断
QSPI_INT_TERR	传输错误中断
QSPI_INT_SM	状态匹配中断
QSPI_INT_TMOUT	超时中断
QSPI_INT_WS	开始序列错误中断
输出参数{out}	
-	-
返回值	
-	-

例如：

```

/* enable QSPI transfer complete interrupt */

qspi_interrupt_enable (QSPI_INT_TC);

```

### 函数 qspi\_interrupt\_disable

函数qspi\_interrupt\_disable描述见下表：

表 3-619. 函数 `qspi_interrupt_disable`

函数名称	<code>qspi_interrupt_disable</code>
函数原形	<code>void qspi_interrupt_disable( uint8_t interrupt);</code>
功能描述	失能外设QSPI中断
先决条件	-
被调用函数	-
输入参数{in}	
<b>interrupt</b>	QSPI中断
<code>QSPI_INT_TC</code>	传输完成中断
<code>QSPI_INT_FT</code>	FIFO阈值中断
<code>QSPI_INT_TERR</code>	传输错误中断
<code>QSPI_INT_SM</code>	状态匹配中断
<code>QSPI_INT_TMOUT</code>	超时中断
<code>QSPI_INT_WS</code>	开始序列错误中断
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* disable QSPI transfer complete interrupt */
```

```
qspi_interrupt_disable (QSPI_INT_TC);
```

### 函数 `qspi_flag_get`

函数`qspi_flag_get`描述见下表：

表 3-620. 函数 `qspi_flag_get`

函数名称	<code>qspi_flag_get</code>
函数原形	<code>FlagStatus qspi_flag_get(uint32_t flag);</code>
功能描述	获取外设QSPI中断状态
先决条件	-
被调用函数	-
输入参数{in}	
<b>flag</b>	QSPI标志位状态
<code>QSPI_FLAG_TERR</code>	传输错误标志
<code>QSPI_FLAG_TC</code>	传输完成标志
<code>QSPI_FLAG_SM</code>	状态匹配标志
<code>QSPI_FLAG_TMOU T</code>	超时标志
<code>QSPI_FLAG_WS</code>	开始序列错误标志
输出参数{out}	
-	-

返回值	
FlagStatus	SET 或 RESET

例如:

```
/* get QSPI busy flag status */
while(RESET == qspi_flag_get (QSPI_FLAG_TC));
```

### 函数 qspi\_flag\_clear

函数qspi\_flag\_clear描述见下表:

表 3-621. 函数 qspi\_flag\_clear

函数名称	qspi_flag_clear
函数原形	void qspi_flag_clear( uint32_t flag);
功能描述	清除QSPI标志状态
先决条件	-
被调用函数	-
输入参数{in}	
flag	QSPI标志位状态
QSPI_FLAG_TERR	传输错误标志
QSPI_FLAG_TC	传输完成标志
QSPI_FLAG_SM	状态匹配标志
QSPI_FLAG_TMOU T	超时标志
QSPI_FLAG_WS	开始序列错误中断
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* clear QSPI transfer complete flag status */
qspi_flag_clear(QSPI_FLAG_TC);
```

## 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 3-622. RCU 寄存器**

寄存器名称	寄存器描述
RCU_CTL	控制寄存器
RCU_PLL	PLL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
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_PLLCFG	PLL时钟配置寄存器
RCU_CFG1	时钟配置寄存器1
RCU_ADDCTL	附加时钟控制寄存器
RCU_SECP_CFG	安全保护配置寄存器
RCU_SECP_STAT	安全保护状态寄存器
RCU_AHB1SECP_STAT	AHB1安全保护状态寄存器
RCU_AHB2SECP_STAT	AHB2安全保护状态寄存器
RCU_AHB3SECP_STAT	AHB3安全保护状态寄存器
RCU_APB1SECP_STAT	APB1安全保护状态寄存器
RCU_APB2SECP_STAT	APB2安全保护状态寄存器

寄存器名称	寄存器描述
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 3-623. RCU 库函数**

库函数名称	库函数描述
rcu_deinit	复位RCU
rcu_periph_clock_enable	使能外设时钟
rcu_periph_clock_disable	禁能外设时钟
rcu_periph_clock_sleep_enable	在睡眠模式下，使能外设时钟
rcu_periph_clock_sleep_disable	在睡眠模式下，禁能外设时钟
rcu_periph_reset_enable	使能外设复位
rcu_periph_reset_disable	禁能外设复位
rcu_bkp_reset_enable	使能BKP复位
rcu_bkp_reset_disable	禁能BKP复位
rcu_hxtal_plli2s_enable	使能PLLI2S高速晶体振荡器（HXTAL）
rcu_hxtal_plli2s_disable	禁能PLLI2S高速晶体振荡器（HXTAL）
rcu_hxtal_pll_p_enable	使能PLL高速晶体振荡器（HXTAL）
rcu_hxtal_pll_p_disable	禁能PLL高速晶体振荡器（HXTAL）
rcu_control_unit_powerup	时钟上电
rcu_control_unit_powerdown	时钟掉电
rcu_rfpll_cal_enable	使能RFPLL计算
rcu_rfpll_cal_disable	禁能RFPLL计算
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CKOUT0时钟源选择
rcu_pll_config	配置主PLL时钟
rcu_plli2s_config	配置主PLLI2S时钟
rcu_plldig_config	配置主PLLDIG时钟
rcu_plldig_div_sys_config	配置PLLDIG时钟作为系统时钟的预分频系数
rcu_rtc_clock_config	配置RTC的时钟源选择
rcu_rtc_div_config	配置RTC时钟的预分频系数
rcu_i2s_clock_config	配置I2S的时钟源选择
rcu_pll_div_i2s_config	配置PLL时钟的分频因子用于I2S时钟

库函数名称	库函数描述
rcu_hpdf_clock_config	配置HPDF的时钟源选择（GD32W515Tx系列设备上不支持）
rcu_hpdf_audio_clock_config	配置HPDF_AUDIO的时钟源选择（GD32W515Tx系列设备上不支持）
rcu_sdio_clock_config	配置SDIO的时钟源选择
rcu_sdio_div_config	SDIO时钟分频因子
rcu_usbfs_clock_config	配置USBFS的时钟源选择
rcu_usbfs_div_config	配置USBFS时钟源的分频因子
rcu_i2c0_clock_config	配置I2C0的时钟源选择
rcu_usart0_clock_config	配置USART0的时钟源选择
rcu_usart2_clock_config	配置USART2的时钟源选择
rcu_irc16m_div_config	配置用于IRC16M时钟的分频因子用于系统时钟
rcu_timer_clock_prescaler_config	配置TIMER时钟预分频
rcu_lxtal_drive_capability_config	配置LXTAL驱动能力
rcu_osci_stab_wait	等待振荡器稳定标志位置位或振荡器起振超时
rcu_osci_on	打开振荡器
rcu_osci_off	关闭振荡器
rcu_osci_bypass_mode_enable	使能振荡器时钟旁路模式
rcu_osci_bypass_mode_disable	禁能振荡器时钟旁路模式
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_hxtal_clock_monitor_enable	使能HXTAL时钟监视器
rcu_hxtal_clock_monitor_disable	禁能HXTAL时钟监视器
rcu_rf_hxtal_clock_monitor_enable	使能RF HXTAL时钟监视器
rcu_rf_hxtal_clock_monitor_disable	禁能RF HXTAL时钟监视器
rcu_voltage_key_unlock	解锁电源寄存器
rcu_deepsleep_voltage_set	设置深度睡眠模式电压值
rcu_clock_freq_get	获取系统时钟、总线频率
rcu_security_enable	使能RCU寄存器安全属性
rcu_security_disable	禁能寄存器安全属性
rcu_privilege_enable	使能寄存器特权属性
rcu_privilege_disable	禁能寄存器特权属性
rcu_flag_get	获取时钟稳定和外设复位标志
rcu_all_reset_flag_clear	清除所有复位标志位
rcu_interrupt_flag_get	获取时钟稳定中断和时钟阻塞中断标志
rcu_interrupt_flag_clear	清除中断标志
rcu_security_flag_get	获取寄存器安全属性标志
rcu_interrupt_enable	使能时钟稳定中断
rcu_interrupt_disable	禁能时钟稳定中断



## 枚举类型 rcu\_periph\_enum

表 3-624. 枚举类型 rcu\_periph\_enum

成员名称	功能描述
RCU_GPIOA	GPIOA时钟
RCU_GPIOB	GPIOB时钟
RCU_GPIOC	GPIOC时钟
RCU_TZPCU	TZPCU时钟
RCU_TSI	TSI时钟
RCU_CRC	CRC时钟
RCU_WIFI	WIFI时钟
RCU_WIFIRUN	WIFIRUN时钟
RCU_SRAM0	SRAM0时钟
RCU_SRAM1	SRAM1时钟
RCU_SRAM2	SRAM2时钟
RCU_SRAM3	SRAM3时钟
RCU_DMA0	DMA0时钟
RCU_DMA1	DMA1时钟
RCU_USBFS	USBFS时钟
RCU_DCI	DCI时钟（DCI在GD32W515Tx系列设备上不支持）
RCU_PKCAU	PKCAU时钟
RCU_CAU	CAU时钟
RCU_HAU	HAU时钟
RCU_TRNG	TRNG时钟
RCU_SQPI	SQPI时钟
RCU_QSPI	QSPI时钟
RCU_TIMER1	TIMER1时钟
RCU_TIMER2	TIMER2时钟
RCU_TIMER3	TIMER3时钟（TIMER3在GD32W515Tx系列设备上不支持）
RCU_TIMER4	TIMER4时钟
RCU_TIMER5	TIMER5时钟
RCU_WWDGT	WWDGT时钟
RCU_SPI1	SPI1时钟
RCU_USART1	USART1时钟
RCU_USART0	USART0时钟
RCU_I2C0	I2C0时钟
RCU_I2C1	I2C1时钟
RCU_PMU	PMU时钟
RCU_RTC	RTC时钟
RCU_TIMER0	TIMER0时钟
RCU_USART2	USART0时钟
RCU_ADC	ADC时钟

成员名称	功能描述
RCU_SDIO	SDIO时钟
RCU_SPI0	SPI0时钟
RCU_SYSCFG	SYSCFG时钟
RCU_TIMER15	TIMER15时钟
RCU_TIMER16	TIMER16时钟
RCU_HPDF	HPDF时钟（HPDF在GD32W515Tx系列设备上不支持）
RCU_RF	RF时钟

### 枚举类型 `rcu_periph_sleep_enum`

表 3-625. 枚举类型 `rcu_periph_sleep_enum`

成员名称	功能描述
RCU_GPIOA_SLP	GPIOA时钟
RCU_GPIOB_SLP	GPIOB时钟
RCU_GPIOC_SLP	GPIOC时钟
RCU_TZPCU_SLP	TZPCU时钟
RCU_TSI_SLP	TSI时钟
RCU_CRC_SLP	CRC时钟
RCU_WIFI_SLP	WIFI时钟
RCU_WIFIRUN_SLP	WIFIRUN时钟
RCU_SRAM0_SLP	SRAM0时钟
RCU_SRAM1_SLP	SRAM1时钟
RCU_SRAM2_SLP	SRAM2时钟
RCU_SRAM3_SLP	SRAM3时钟
RCU_DMA0_SLP	DMA0时钟
RCU_DMA1_SLP	DMA1时钟
RCU_USBFS_SLP	USBFS时钟
RCU_DCI_SLP	DCI时钟（DCI在GD32W515Tx系列设备上不支持）
RCU_PKCAU_SLP	PKCAU时钟
RCU_CAU_SLP	CAU时钟
RCU_HAU_SLP	HAU时钟
RCU_TRNG_SLP	TRNG时钟
RCU_SQPI_SLP	SQPI时钟
RCU_QSPI_SLP	QSPI时钟
RCU_TIMER1_SLP	TIMER1时钟
RCU_TIMER2_SLP	TIMER2时钟
RCU_TIMER3_SLP	TIMER3时钟
RCU_TIMER4_SLP	TIMER4时钟
RCU_TIMER5_SLP	TIMER5时钟
RCU_WWDGT_SLP	WWDGT时钟
RCU_SPI1_SLP	SPI1时钟
RCU_USART1_SLP	USART1时钟

成员名称	功能描述
RCU_USART0_SLP	USART0时钟
RCU_I2C0_SLP	I2C0时钟
RCU_I2C1_SLP	I2C1时钟
RCU_PMU_SLP	PMU时钟
RCU_RTC_SLP	RTC时钟
RCU_TIMER0_SLP	TIMER0时钟
RCU_USART2_SLP	USART0时钟
RCU_ADC_SLP	ADC时钟
RCU_SDIO_SLP	SDIO时钟
RCU_SPI0_SLP	SPI0时钟
RCU_SYSCFG_SLP	SYSCFG时钟
RCU_TIMER15_SLP	TIMER15时钟
RCU_TIMER16_SLP	TIMER16时钟
RCU_HPDCF_SLP	HPDCF时钟（HPDCF在GD32W515Tx系列设备上不支持）
RCU_RF_SLP	RF时钟

#### 枚举类型 `rcu_periph_reset_enum`

表 3-626. 枚举类型 `rcu_periph_reset_enum`

成员名称	功能描述
RCU_GPIOARST	GPIOA时钟
RCU_GPIOBRST	GPIOB时钟
RCU_GPIOCRST	GPIOC时钟
RCU_TZPCURST	TZPCU时钟
RCU_TSIRST	TSI时钟
RCU_CRCRST	CRC时钟
RCU_WIFIRST	WIFI时钟
RCU_DMA0RST	DMA0时钟
RCU_DMA1RST	DMA1时钟
RCU_USBFSRST	USBFS时钟
RCU_DCIRST	DCI时钟（DCI在GD32W515Tx系列设备上不支持）
RCU_PKCAURST	PKCAU时钟
RCU_CAURST	CAU时钟
RCU_HAURST	HAU时钟
RCU_TRNGRST	TRNG时钟
RCU_SQPIRST	SQPI时钟
RCU_QSPIRST	QSPI时钟
RCU_TIMER1RST	TIMER1时钟
RCU_TIMER2RST	TIMER2时钟
RCU_TIMER3RST	TIMER3时钟（TIMER3在GD32W515Tx系列设备上不支持）
RCU_TIMER4RST	TIMER4时钟
RCU_TIMER5RST	TIMER5时钟

成员名称	功能描述
RCU_WWDGTRST	WWDGT时钟
RCU_SPI1RST	SPI1时钟
RCU_USART1RST	USART1时钟
RCU_USART0RST	USART0时钟
RCU_I2C0RST	I2C0时钟
RCU_I2C1RST	I2C1时钟
RCU_PMURST	PMU时钟
RCU_RTCCRST	RTC时钟
RCU_TIMER0RST	TIMER0时钟
RCU_USART2RST	USART0时钟
RCU_ADCRST	ADC时钟
RCU_SDIORST	SDIO时钟
RCU_SPI0RST	SPI0时钟
RCU_SYSCFGRST	SYSCFG时钟
RCU_TIMER15RST	TIMER15时钟
RCU_TIMER16RST	TIMER16时钟
RCU_HPDRST	HPDF时钟（HPDF在GD32W515Tx系列设备上不支持）
RCU_RFRST	RF时钟

### 枚举类型 `rcu_flag_enum`

表 3-627. 枚举类型 `rcu_flag_enum`

成员名称	功能描述
RCU_FLAG_IRC16MSTB	IRC16M振荡器稳定标志
RCU_FLAG_HXTALSTB	外部高速晶振稳定标志
RCU_FLAG_PLLDIGSTB	PLLDIG稳定标志
RCU_FLAG_PLLSTB	PLL稳定标志
RCU_FLAG_PLLI2SSTB	PLLI2S稳定标志
RCU_FLAG_LXTALSTB	LXTAL稳定标志
RCU_FLAG_IRC32KSTB	IRC32K稳定标志
RCU_FLAG_OBLRST	选项字节重载复位标志
RCU_FLAG_EPRST	外部引脚复位标志
RCU_FLAG_PORRST	电源复位标志
RCU_FLAG_SWRST	软件复位标志
RCU_FLAG_FWDGTRST	独立看门狗复位标志
RCU_FLAG_WWDGTRST	窗口看门狗复位标志
RCU_FLAG_LPRST	低功耗复位标志

### 枚举类型 rcu\_int\_flag\_enum

表 3-628. 枚举类型 rcu\_int\_flag\_enum

成员名称	功能描述
RCU_INT_FLAG_IRC32KSTB	IRC32K时钟稳定中断标志
RCU_INT_FLAG_LXTALSTB	外部低速晶振时钟稳定中断标志
RCU_INT_FLAG_IRC16MSTB	IRC16M时钟稳定中断标志
RCU_INT_FLAG_HXTALSTB	外部高速晶振时钟稳定中断标志
RCU_INT_FLAG_PLLSTB	PLL时钟稳定中断标志
RCU_INT_FLAG_PLLI2SSTB	PLLI2S时钟稳定中断标志
RCU_INT_FLAG_PLL2STB	PLL2时钟稳定中断标志
RCU_INT_FLAG_PLLDIGSTB	PLLDIG时钟稳定中断标志
RCU_INT_FLAG_CKM	外部高速晶振时钟阻塞中断标志

### 枚举类型 rcu\_int\_flag\_clear\_enum

表 3-629. 枚举类型 rcu\_int\_flag\_clear\_enum

成员名称	功能描述
RCU_INT_FLAG_IRC32KSTB_CLR	IRC32K时钟稳定中断清除标志
RCU_INT_FLAG_LXTALSTB_CLR	外部低速晶振时钟稳定中断清除标志
RCU_INT_FLAG_IRC16MSTB_CLR	IRC16M时钟稳定中断清除标志
RCU_INT_FLAG_HXTALSTB_CLR	外部高速晶振时钟稳定中断清除标志
RCU_INT_FLAG_PLLSTB_CLR	PLL时钟稳定中断清除标志
RCU_INT_FLAG_PLLI2SSTB_CLR	PLLI2S时钟稳定中断清除标志
RCU_INT_FLAG_PLLDIGSTB_CLR	PLLDIG时钟稳定中断清除标志
RCU_INT_FLAG_CKM_CLR	外部高速晶振时钟阻塞中断清除标志

### 枚举类型 rcu\_int\_enum

表 3-630. 枚举类型 rcu\_int\_enum

成员名称	功能描述
RCU_INT_IRC32KSTB	IRC32K时钟稳定中断
RCU_INT_LXTALSTB	外部低速晶振时钟稳定中断
RCU_INT_IRC16MSTB	IRC16M时钟稳定中断
RCU_INT_HXTALSTB	外部高速晶振时钟稳定中断
RCU_INT_PLLSTB	PLL时钟稳定中断
RCU_INT_PLLI2SSTB	PLLI2S时钟稳定中断
RCU_INT_PLLDIGSTB	PLLDIG时钟稳定中断

### 枚举类型 `rcu_osci_type_enum`

表 3-631. 枚举类型 `rcu_osci_type_enum`

成员名称	功能描述
RCU_HXTAL	外部高速振荡器
RCU_LXTAL	外部低速振荡器
RCU_IRC16M	IRC16M振荡器
RCU_IRC32K	IRC32K振荡器
RCU_PLLDIG_CK	PLLDIG时钟
RCU_PLL_CK	锁相环时钟
RCU_PLLI2S_CK	PLLI2S时钟

### 枚举类型 `rcu_clock_freq_enum`

表 3-632. 枚举类型 `rcu_clock_freq_enum`

成员名称	功能描述
CK_SYS	系统时钟
CK_AHB	AHB时钟
CK_APB1	APB1时钟
CK_APB2	APB2时钟
CK_USART	USART0时钟
CK_USART2	USART2时钟
CK_I2C0	I2C0时钟

### 枚举类型 `rcu_sec_enum`

表 3-633. 枚举类型 `rcu_sec_enum`

成员名称	功能描述
RCU_SEC_IRC16MSEC	IRC16M时钟配置和状态位安全
RCU_SEC_HXTALSEC	HXTAL时钟配置和状态位安全
RCU_SEC_IRC32KSEC	IRC32K时钟配置和状态位安全
RCU_SEC_LXTALSEC	LXTAL时钟配置和状态位安全
RCU_SEC_SYSCCLKSEC	SYSCLK时钟选择, STOPWUCK位, MCO配置上的时钟输出
RCU_SEC_PRESCSEC	AHBx / APBx预分频器配置位安全性
RCU_SEC_PLLSEC	PLL时钟配置和状态位安全
RCU_SEC_PLLDIGSEC	PLLDIG时钟配置和状态位安全
RCU_SEC_PLLI2SSEC	PLLI2S时钟配置和状态位安全
RCU_SEC_RMVRSTFSEC	删除复位标志安全
RCU_SEC_BKPSEC	BKP安全

## 枚举类型 `rcu_sec_flag_enum`

表 3-634. 枚举类型 `rcu_sec_flag_enum`

成员名称	功能描述
<code>RCU_SEC_FLAG_IRC16MSEC</code>	IRC16M时钟配置和状态位安全
<code>RCU_SEC_FLAG_HXTALSEC</code>	HXTAL时钟配置和状态位安全
<code>RCU_SEC_FLAG_IRC32KSEC</code>	IRC32K时钟配置和状态位安全
<code>RCU_SEC_FLAG_LXTALSEC</code>	LXTAL时钟配置和状态位安全
<code>RCU_SEC_FLAG_SYSCCLKSEC</code>	SYSCLK时钟选择, STOPWUCK位, MCO配置上的时钟输出
<code>RCU_SEC_FLAG_PRESCSEC</code>	AHBx / APBx预分频器配置位安全性
<code>RCU_SEC_FLAG_PLLSEC</code>	PLL时钟配置和状态位安全
<code>RCU_SEC_FLAG_PLLDIGSEC</code>	PLLDIG时钟配置和状态位安全
<code>RCU_SEC_FLAG_PLI2SSEC</code>	PLI2S时钟配置和状态位安全
<code>RCU_SEC_FLAG_RMVRSTFSEC</code>	删除复位标志安全
<code>RCU_SEC_FLAG_BKPSEC</code>	BKP安全
<code>RCU_SEC_FLAG_GPIOA</code>	GPIOA时钟
<code>RCU_SEC_FLAG_GPIOB</code>	GPIOB时钟
<code>RCU_SEC_FLAG_GPIOC</code>	GPIOC时钟
<code>RCU_SEC_FLAG_CRC</code>	CRC时钟
<code>RCU_SEC_FLAG_WIFI</code>	WiFi时钟
<code>RCU_SEC_FLAG_FMC</code>	FMC时钟
<code>RCU_SEC_FLAG_SRAM0</code>	SRAM0时钟
<code>RCU_SEC_FLAG_SRAM1</code>	SRAM1时钟
<code>RCU_SEC_FLAG_SRAM2</code>	SRAM2时钟
<code>RCU_SEC_FLAG_SRAM3</code>	SRAM3时钟
<code>RCU_SEC_FLAG_DMA0</code>	DMA0时钟
<code>RCU_SEC_FLAG_DMA1</code>	DMA1时钟
<code>RCU_SEC_FLAG_USBFS</code>	USBFS时钟
<code>RCU_SEC_FLAG_DCI</code>	DCI时钟 (DCI在GD32W515Tx系列设备上不支持)
<code>RCU_SEC_FLAG_PKCAU</code>	PKCAU时钟
<code>RCU_SEC_FLAG_CAU</code>	CAU时钟
<code>RCU_SEC_FLAG_HAU</code>	HAU时钟
<code>RCU_SEC_FLAG_TRNG</code>	TRNG时钟
<code>RCU_SEC_FLAG_SQPI</code>	SQPI时钟
<code>RCU_SEC_FLAG_QSPI</code>	QSPI时钟
<code>RCU_SEC_FLAG_TIMER1</code>	TIMER1时钟
<code>RCU_SEC_FLAG_TIMER2</code>	TIMER2时钟
<code>RCU_SEC_FLAG_TIMER3</code>	TIMER3时钟
<code>RCU_SEC_FLAG_TIMER4</code>	TIMER4时钟
<code>RCU_SEC_FLAG_TIMER5</code>	TIMER5时钟
<code>RCU_SEC_FLAG_WWDGT</code>	WWDGT时钟
<code>RCU_SEC_FLAG_SPI1</code>	SPI1时钟

成员名称	功能描述
RCU_SEC_FLAG_USART1	USART1时钟
RCU_SEC_FLAG_USART0	USART0时钟
RCU_SEC_FLAG_I2C0	I2C0时钟
RCU_SEC_FLAG_I2C1	I2C1时钟
RCU_SEC_FLAG_PMU	PMU时钟
RCU_SEC_FLAG_TIMER0	TIMER0时钟
RCU_SEC_FLAG_USART2	USART2时钟
RCU_SEC_FLAG_ADC	ADC时钟
RCU_SEC_FLAG_SDIO	SDIO时钟
RCU_SEC_FLAG_SPI0	SPI0时钟
RCU_SEC_FLAG_SYSCFG	SYSCFG时钟
RCU_SEC_FLAG_TIMER15	TIMER15时钟
RCU_SEC_FLAG_TIMER16	TIMER16时钟
RCU_SEC_FLAG_HPDCF	HPDCF时钟（HPDCF在GD32W515Tx系列设备上不支持）
RCU_SEC_FLAG_RF	RF时钟

### 枚举类型 `rcu_unit_enum`

表 3-635. 枚举类型 `rcu_unit_enum`

成员名称	功能描述
RCU_UNIT_HXTAL	HXTAL
RCU_UNIT_PLLDIG	PLLDIG
RCU_UNIT_RFPLL	RFPLL
RCU_UNIT_LDOANA	LDOANA
RCU_UNIT_LDOCLK	LDOCLK
RCU_UNIT_BANDGAP	BANDGAP

### 函数 `rcu_deinit`

函数`rcu_deinit`描述见下表：

表 3-636. 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	<code>rcu_osc_stab_wait</code>
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 3-637. 函数 rcu\_periph\_clock\_enable**

函数名称	rcu_periph_clock_enable
函数原形	void rcu_periph_clock_enable(rcu_periph_enum periph);
功能描述	使能外设时钟
先决条件	-
被调用函数	-
输入参数{in}	
periph	RCU外设，具体参考 <a href="#">表3-624. 枚举类型rcu_periph_enum</a>
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* enable the USART0 clock */
```

```
rcu_periph_clock_enable(RCU_USART0);
```

### 函数 rcu\_periph\_clock\_disable

函数rcu\_periph\_clock\_disable描述见下表：

**表 3-638. 函数 rcu\_periph\_clock\_disable**

函数名称	rcu_periph_clock_disable
函数原形	void rcu_periph_clock_disable(rcu_periph_enum periph);
功能描述	禁能外设时钟
先决条件	-
被调用函数	-
输入参数{in}	
periph	RCU外设，具体参考 <a href="#">表3-624. 枚举类型rcu_periph_enum</a>
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* disable the USART0 clock */
```

```
rcu_periph_clock_disable(RCU_USART0);
```

### 函数 rcu\_periph\_clock\_sleep\_enable

函数rcu\_periph\_clock\_sleep\_enable描述见下表：

**表 3-639. 函数 rcu\_periph\_clock\_sleep\_enable**

函数名称	rcu_periph_clock_sleep_enable
函数原形	void rcu_periph_clock_sleep_enable(rcu_periph_sleep_enum periph);
功能描述	在睡眠模式下，使能外设时钟
先决条件	-
被调用函数	-
输入参数{in}	
periph	RCU外设，参考 <a href="#">表3-625. 枚举类型rcu_periph_sleep_enum</a>
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* enable the FMC clock when in sleep mode */
```

```
rcu_periph_clock_sleep_enable(RCU_FMC_SLP);
```

### 函数 rcu\_periph\_clock\_sleep\_disable

函数rcu\_periph\_clock\_sleep\_disable描述见下表：

**表 3-640. 函数 rcu\_periph\_clock\_sleep\_disable**

函数名称	rcu_periph_clock_sleep_disable
函数原形	void rcu_periph_clock_sleep_disable(rcu_periph_sleep_enum periph);
功能描述	在睡眠模式下，禁能外设时钟
先决条件	-
被调用函数	-
输入参数{in}	
periph	RCU外设，参考 <a href="#">表3-625. 枚举类型rcu_periph_sleep_enum</a>
RCU_HPDI_SLP	
-	-
返回值	
-	-

例如：

```
/* disable the FMC clock when in sleep mode */
```

```
rcu_periph_clock_sleep_disable(RCU_FMC_SLP);
```

### 函数 rcu\_periph\_reset\_enable

函数rcu\_periph\_reset\_enable描述见下表:

表 3-641. 函数 rcu\_periph\_reset\_enable

函数名称	rcu_periph_reset_enable
函数原形	void rcu_periph_reset_enable(rcu_periph_reset_enum periph_reset);
功能描述	使能外设复位
先决条件	-
被调用函数	-
输入参数{in}	
periph_reset	RCU外设复位, 参考 <a href="#">表3-626. 枚举类型rcu_periph_reset_enum</a>
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* enable SPI0 reset */  
  
rcu_periph_reset_enable(RCU_SPI0RST);
```

### 函数 rcu\_periph\_reset\_disable

函数rcu\_periph\_reset\_disable描述见下表:

表 3-642. 函数 rcu\_periph\_reset\_disable

函数名称	rcu_periph_reset_disable
函数原形	void rcu_periph_reset_disable(rcu_periph_reset_enum periph_reset);
功能描述	禁能外设复位
先决条件	-
被调用函数	-
输入参数{in}	
periph_reset	RCU外设复位, 参考 <a href="#">表3-626. 枚举类型rcu_periph_reset_enum</a>
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* disable SPI0 reset */  
  
rcu_periph_reset_disable(RCU_SPI0RST);
```

## 函数 rcu\_bkp\_reset\_enable

函数rcu\_bkp\_reset\_enable描述见下表:

**表 3-643. 函数 rcu\_bkp\_reset\_enable**

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

例如:

```
/* reset the BKP domain */
rcu_bkp_reset_enable();
```

## 函数 rcu\_bkp\_reset\_disable

函数rcu\_bkp\_reset\_disable描述见下表:

**表 3-644. 函数 rcu\_bkp\_reset\_disable**

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

例如:

```
/* disable the BKP domain reset */
rcu_bkp_reset_disable();
```

## 函数 rcu\_hxtal\_plli2s\_enable

函数rcu\_hxtal\_plli2s\_enable描述见下表：

**表 3-645. 函数 rcu\_hxtal\_plli2s\_enable**

函数名称	rcu_hxtal_plli2s_enable
函数原形	void rcu_hxtal_plli2s_enable(void);
功能描述	PLLI2S高速晶体振荡器使能
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 HXTAL for PLLI2S */
rcu_hxtal_plli2s_enable();
```

## 函数 rcu\_hxtal\_plli2s\_disable

函数rcu\_hxtal\_plli2s\_disable描述见下表：

**表 3-646. 函数 rcu\_hxtal\_plli2s\_disable**

函数名称	rcu_hxtal_plli2s_disable
函数原形	void rcu_hxtal_plli2s_disable(void);
功能描述	PLLI2S高速晶体振荡器禁能
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 HXTAL for PLLI2S */
rcu_hxtal_plli2s_disable();
```

**函数 rcu\_control\_unit\_powerup**

函数rcu\_control\_unit\_powerup描述见下表：

**表 3-647. 函数 rcu\_control\_unit\_powerup**

函数名称	rcu_control_unit_powerup
函数原形	void rcu_control_unit_powerup(rcu_unit_enum rcu_unit);
功能描述	时钟上电
先决条件	-
被调用函数	-
输入参数{in}	
rcu_unit	
RCU_UNIT_HXTAL	高速晶体振荡器掉电
RCU_UNIT_PLLDIG	PLLDIG上电
RCU_UNIT_RFPLL	上电RFPLL时钟
RCU_UNIT_LDOAN A	LDO模拟域上电
RCU_UNIT_LDOCL K	LDO时钟上电
RCU_UNIT_BANDG AP	BandGap上电
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* power on the HXTAL */
```

```
rcu_control_unit_powerup(RCU_UNIT_HXTAL);
```

**函数 rcu\_control\_unit\_powerdown**

函数rcu\_control\_unit\_powerdown描述见下表：

**表 3-648. 函数 rcu\_control\_unit\_powerdown**

函数名称	rcu_control_unit_powerdown
函数原形	void rcu_control_unit_powerdown(rcu_unit_enum rcu_unit);
功能描述	时钟掉电
先决条件	-
被调用函数	-
输入参数{in}	
rcu_unit	
RCU_UNIT_HXTAL	高速晶体振荡器掉电
RCU_UNIT_PLLDIG	PLLDIG掉电

<i>RCU_UNIT_RFPLL</i>	掉电RFPLL时钟
<i>RCU_UNIT_LDOAN A</i>	LDO模拟域掉电
<i>RCU_UNIT_LDOCL K</i>	LDO时钟掉电
<i>RCU_UNIT_BANDG AP</i>	BandGap掉电
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* power on the HXTAL */
```

```
rcu_control_unit_powerdown(RCU_UNIT_HXTAL);
```

### 函数 `rcu_rfppl_cal_enable`

函数`rcu_rfppl_cal_enable`描述见下表：

**表 3-649. 函数 `rcu_rfppl_cal_enable`**

函数名称	<code>rcu_rfppl_cal_enable</code>
函数原形	<code>void rcu_rfppl_cal_enable(void);</code>
功能描述	使能RFPLL计算
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 RF PLL calculation */
```

```
rcu_rfppl_cal_enable();
```

### 函数 `rcu_rfppl_cal_disable`

函数`rcu_rfppl_cal_disable`描述见下表：

**表 3-650. 函数 `rcu_rfppl_cal_disable`**

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

功能描述	禁能RFPLL计算
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 RF PLL calculation */
```

```
rcu_rfppl_cal_disable ();
```

### 函数 rcu\_system\_clock\_source\_config

函数rcu\_system\_clock\_source\_config描述见下表：

表 3-651. 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}	
<b>ck_sys</b>	系统时钟源选择
<i>RCU_CKSYSSRC_IRC16M</i>	选择CK_IRC16M时钟作为CK_SYS时钟源
<i>RCU_CKSYSSRC_HXTAL</i>	选择CK_HXTAL时钟作为CK_SYS时钟源
<i>RCU_CKSYSSRC_PLL</i>	选择CK_PLL时钟作为CK_SYS时钟源
<i>RCU_CKSYSSRC_PLLDIG</i>	选择CK_PLLDIG时钟作为CK_SYS时钟源
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* configure the CK_HXTAL as the CK_SYS source */
```

```
rcu_system_clock_source_config(RCU_CKSYSSRC_HXTAL);
```



**函数 rcu\_system\_clock\_source\_get**

函数rcu\_system\_clock\_source\_get描述见下表：

**表 3-652. 函数 rcu\_system\_clock\_source\_get**

函数名称	rcu_system_clock_source_get
函数原形	uint32_t rcu_system_clock_source_get(void);
功能描述	获取系统时钟源选择状态
先决条件	-
被调用函数	-
输入参数{in}	
-	-
输出参数{out}	
-	-
返回值	
uint32_t	RCU_SCSS_IRC16M/RCU_SCSS_HXTAL/RCU_SCSS_PLL/RCU_CKSYSSR C_PLLDIG

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 3-653. 函数 rcu\_ahb\_clock\_config**

函数名称	rcu_ahb_clock_config
函数原形	void rcu_ahb_clock_config(uint32_t ck_ahb);
功能描述	配置AHB时钟预分频选择
先决条件	-
被调用函数	-
输入参数{in}	
ck_ahb	AHB预分频选择
RCU_AHB_CKSYS _DIVx	选择CK_SYS时钟x分频（x=1, 2, 4, 8, 16, 64, 128, 256, 512）
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* configure CK_SYS/128 */
```

```
rcu_ahb_clock_config(RCU_AHB_CKSYS_DIV128);
```

### 函数 rcu\_apb1\_clock\_config

函数rcu\_apb1\_clock\_config描述见下表:

表 3-654. 函数 rcu\_apb1\_clock\_config

函数名称	rcu_apb1_clock_config
函数原形	void rcu_apb1_clock_config(uint32_t ck_apb1);
功能描述	配置APB1时钟预分频选择
先决条件	-
被调用函数	-
输入参数{in}	
ck_apb1	APB1预分频选择
RCU_APB1_CKAHB_DIV4	选择CK_AHB/4为CK_APB1
RCU_APB1_CKAHB_DIV8	选择CK_AHB/8为CK_APB1
RCU_APB1_CKAHB_DIV16	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 3-655. 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_CKAHB_DIV1	选择CK_AHB为CK_APB2
RCU_APB2_CKAHB_DIV2	选择CK_AHB/2为CK_APB2

<i>B_DIV2</i>	
<i>RCU_APB2_CKAH</i> <i>B_DIV4</i>	选择CK_AHB/4为CK_APB2
<i>RCU_APB2_CKAH</i> <i>B_DIV8</i>	选择CK_AHB/8为CK_APB2
<i>RCU_APB2_CKAH</i> <i>B_DIV16</i>	选择CK_AHB/16为CK_APB2
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* configure CK_AHB/8 as CK_APB2 */
```

```
rcu_apb2_clock_config(RCU_APB2_CKAHB_DIV8);
```

### 函数 rcu\_ckout0\_config

函数rcu\_ckout0\_config描述见下表：

表 3-656. 函数 rcu\_ckout0\_config

函数名称	rcu_ckout0_config
函数原形	void rcu_ckout0_config(uint32_t ckout0_src);
功能描述	配置CKOUT0时钟源选择
先决条件	-
被调用函数	-
输入参数{in}	
<b>ckout0_src</b>	CKOUT0时钟源选择
<i>RCU_CKOUT0SRC</i> <i>_IRC16M</i>	选择内部16M RC振荡器时钟
<i>RCU_CKOUT0SRC</i> <i>_LXTAL</i>	选择低速晶体振荡器时钟（LXTAL）
<i>RCU_CKOUT0SRC</i> <i>_HXTAL</i>	选择高速晶体振荡器时钟（HXTAL）
<i>RCU_CKOUT0SRC</i> <i>_PLL</i>	选择CK_PLLP时钟
输入参数{in}	
<b>ckout0_div</b>	CK_OUT0分频选择
<i>RCU_CKOUT0_DIV</i> <i>x</i>	选择CK_OUT0/x为输出频率（x=1, 2, 3, 4, 5）
输出参数{out}	
-	-
返回值	

-	-
---	---

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 3-657. 函数 rcu\_ckout1\_config

函数名称	rcu_ckout1_config
函数原形	void rcu_ckout1_config(uint32_t ckout0_src);
功能描述	配置CKOUT1时钟源选择
先决条件	-
被调用函数	-
输入参数{in}	
ckout1_src	CKOUT1时钟源选择
RCU_CKOUT1SRC_CKSYS	选择系统时钟
RCU_CKOUT1SRC_PLLI2S	选择PLLI2S时钟
RCU_CKOUT1SRC_HXTAL	选择高速晶体振荡器时钟（HXTAL）
RCU_CKOUT1SRC_PLLDIG	选择PLLDIG时钟
输入参数{in}	
ckout1_div	CK_OUT1分频选择
RCU_CKOUT1_DIVx	选择CK_OUT1/x为输出频率（x=1，2，3，4，5）
输出参数{out}	
-	-
返回值	
-	-

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 3-658. 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);
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_IRC16M	IRC16M被选择为PLL时钟的时钟源
RCU_PLLSRC_HXTAL	HXTAL时钟被选择为PLL时钟的时钟源
输入参数{in}	
pll_psc	PLL VCO时钟源分频因子
uint32_t	2~63
输入参数{in}	
pll_p	PLL VCO时钟源分频因子
uint32_t	2,4,6,8
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* Configure the main PLL, PSC = 8, PLL_N = 240, PLL_P = 2 */
```

```
rcu_pll_config(RCU_PLLSRC_HXTAL,8,240,2);
```

### 函数 rcu\_plli2s\_config

函数rcu\_plli2s\_config描述见下表:

表 3-659. 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_psc, uint32_t plli2s_div);
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	PLLI2S VCO时钟倍频因子
uint32_t	8~127
输入参数{in}	

<b>plli2s_psc</b>	PLLI2S VCO时钟源分频因子
<i>RCU_PLLI2SSRC_DIVx</i>	2~63
输入参数{in}	
<b>plli2s_div</b>	PLLI2SDIV时钟源分频因子
<i>RCU_PLLI2S_DIVx</i>	PLLI2SDIV时钟/ x(x=1.5,2,2.5,3,...,32)
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* Configure the I2S PLL, PLLI2S_PSC = 5, PLL_N = 10, PLLI2S_DIV = 2 */
rcu_plli2s_config (10, RCU_PLLI2SSRC_DIV5, RCU_PLLI2S_DIV2);
```

### 函数 rcu\_plldig\_config

函数rcu\_plldig\_config描述见下表:

表 3-660. 函数 rcu\_plldig\_config

函数名称	rcu_plldig_config
函数原形	void rcu_plldig_config(uint32_t plldig_clk);
功能描述	配置PLLDIG时钟
先决条件	-
被调用函数	-
输入参数{in}	
<b>plldig_clk</b>	PLLDIG 时钟输出选择
<i>RCU_PLLDIG_192_M</i>	选择192Mhz为PLLDIG时钟输出
<i>RCU_PLLDIG_240_M</i>	选择240Mhz为PLLDIG时钟输出
<i>RCU_PLLDIG_320_M</i>	选择320Mhz为PLLDIG时钟输出
<i>RCU_PLLDIG_480_M</i>	选择480Mhz为PLLDIG时钟输出
输出参数{out}	
-	-
返回值	
<b>ErrStatus</b>	SUCCESS or ERROR

例如:

```
/* Configure the I2S PLL, PLLI2S_PSC = 5, PLL_N = 10, PLLI2S_DIV = 2 */
rcu_plli2s_config (10, RCU_PLLI2SSRC_DIV5, RCU_PLLI2S_DIV2);
```

```
/* configure the PLLDIG output 192Mhz clock frequency */
```

```
rcu_plldig_config(RCU_PLLDIG_192M);
```

### 函数 rcu\_plldig\_div\_sys\_config

函数rcu\_plldig\_div\_sys\_config描述见下表:

表 3-661. 函数 rcu\_plldig\_div\_sys\_config

函数名称	rcu_plldig_div_sys_config
函数原形	void rcu_plldig_div_sys_config(uint32_t plldig_div_sys);
功能描述	配置PLLDIG时钟分频系数输出给系统时钟
先决条件	-
被调用函数	-
输入参数{in}	
plldig_div_sys	PLLDIG时钟分频系数输出给系统时钟
RCU_PLLDIG_SYS _DIVx	PLLDIG时钟/x作为系统时钟(x=1,2,3,...,64)
输出参数{out}	
-	-
返回值	
ErrStatus	SUCCESS or ERROR

例如:

```
/* configure PLLDIG clock divider 2 for system clock */
```

```
rcu_plldig_div_sys_config (RCU_PLLDIGFSYS_DIV2);
```

### 函数 rcu\_rtc\_clock\_config

函数rcu\_rtc\_clock\_config描述见下表:

表 3-662. 函数 rcu\_rtc\_clock\_config

函数名称	rcu_rtc_clock_config
函数原形	void rcu_rtc_clock_config(uint32_t rtc_clock_source);
功能描述	配置RTC的时钟源选择
先决条件	-
被调用函数	-
输入参数{in}	
rtc_clock_source	RTC时钟源选择
RCU_RTCSRC_NO NE	没有时钟
RCU_RTCSRC_LX TAL	选择CK_LXTAL时钟作为RTC的时钟源
RCU_RTCSRC_IRC 32K	选择CK_IRC40K时钟作为RTC的时钟源

<i>RCU_RTC_SRC_HXTAL_DIV_RTC_DIV</i>	选择CK_HXTAL / RTCDIV时钟作为RTC的时钟源
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* configure the RTC clock source selection */
rcu_rtc_clock_config(RCU_RTC_SRC_IRC32K);
```

### 函数 `rcu_rtc_div_config`

函数`rcu_rtc_div_config`描述见下表：

表 3-663. 函数 `rcu_rtc_div_config`

函数名称	<code>rcu_rtc_div_config</code>
函数原形	<code>void rcu_rtc_div_config(uint32_t rtc_div);</code>
功能描述	当HXTAL为RTC时钟源时，配置RTC的时钟源分频系数
先决条件	-
被调用函数	-
输入参数{in}	
<b>rtc_div</b>	RTC时钟源分频系数
<i>RCU_RTC_HXTAL_NONE</i>	没有时钟
<i>RCU_RTC_HXTAL_DIVx</i>	CK_HXTAL/x作为RTCDIV时钟（x = 2...31）
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* RTCDIV clock select CK_HXTAL/2 */
rcu_rtc_div_config(RCU_RTC_HXTAL_DIV2);
```

### 函数 `rcu_i2s_clock_config`

函数`rcu_i2s_clock_config`描述见下表：

表 3-664. 函数 `rcu_i2s_clock_config`

函数名称	<code>rcu_i2s_clock_config</code>
函数原形	<code>void rcu_i2s_clock_config(uint32_t i2s_clock_source);</code>
功能描述	配置I2S时钟源选择



先决条件	-
被调用函数	-
输入参数{in}	
<b>i2s_clock_source</b>	I2S时钟源选择
<i>RCU_I2SSRC_PLLI2S</i>	选择CK_PLLI2S作为I2S时钟源
<i>RCU_I2SSRC_I2S_CKIN</i>	选择外部i2s_ckin引脚作为I2S时钟源
<i>RCU_I2SSRC_I2S_PLLDIV</i>	选择PLL分频作为I2S时钟源
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* select CK_PLLI2S as I2S source clock */
rcu_i2s_clock_config(RCU_I2SSRC_PLLI2S);
```

### 函数 rcu\_pll\_div\_i2s\_config

函数rcu\_pll\_div\_i2s\_config描述见下表：

表 3-665. 函数 rcu\_pllfi2s\_clock\_div\_config

函数名称	rcu_pll_div_i2s_config
函数原形	void rcu_pll_div_i2s_config(uint32_t pll_div_i2s);
功能描述	配置PLL提供给I2S时钟分频因子
先决条件	-
被调用函数	-
输入参数{in}	
<b>pll_div_i2s</b>	PLL提供给I2S时钟分频因子
<i>RCU_PLLDIV_I2S_DIVx</i>	PLL时钟/x(x=1,2,...,64)
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* PLL clock divided by 8 for I2S clock */
rcu_pll_div_i2s_config (RCU_PLLFI2SDIV_DIV8);
```

函数 `rcu_hpdf_clock_config`

函数 `rcu_hpdf_clock_config` 描述见下表:

表 3-666. 函数 `rcu_pllfi2s_clock_div_config`

函数名称	<code>rcu_hpdf_clock_config</code>
函数原形	<code>void rcu_hpdf_clock_config(uint32_t hpdf_clock_source);</code>
功能描述	配置HPDF时钟源选择
先决条件	-
被调用函数	-
输入参数{in}	
<code>hpdf_clock_source</code>	PLL提供给I2S时钟分频因子
<code>RCU_HPDFSRC_PCLK2</code>	PCLK2时钟选择为HPDF时钟
<code>RCU_HPDFSRC_CKSYS</code>	系统时钟选择为HPDF时钟
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* select system clock as hpdf clock source */
rcu_hpdf_clock_config(RCU_HPDFSRC_CKSYS);
```

函数 `rcu_hpdf_audio_clock_config`

函数 `rcu_hpdf_audio_clock_config` 描述见下表:

表 3-667. 函数 `rcu_hpdf_audio_clock_config`

函数名称	<code>rcu_hpdf_audio_clock_config</code>
函数原形	<code>void rcu_hpdf_audio_clock_config(uint32_t hpdfaudio_clock_source);</code>
功能描述	配置HPDF_AUDIO时钟源选择
先决条件	-
被调用函数	-
输入参数{in}	
<code>hpdf_clock_source</code>	HPDF_AUDIO时钟源选择
<code>RCU_HPDDAUDIO_SRC_PLLI2S</code>	PCLK2时钟选择为HPDF_AUDIO时钟源
<code>RCU_HPDDAUDIO_SRC_I2S_CKIN</code>	外部I2S_CKIN选择为HPDF_AUDIO时钟源
<code>RCU_HPDDAUDIO_SRC_PLL</code>	PLL时钟作为HPDF_AUDIO时钟源
<code>RCU_HPDDAUDIO</code>	IRC16M选择为HPDF_AUDIO时钟源

<i>SRC_IRC16M</i>	
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* select PLLI2S clock as hpdf audio clock source */
```

```
rcu_hpdf_audio_clock_config(RCU_HPDAUDIOSRC_PLLI2S);
```

### 函数 **rcu\_sdio\_clock\_config**

函数rcu\_sdio\_clock\_config描述见下表：

**表 3-668. 函数 rcu\_sdio\_clock\_config**

函数名称	rcu_sdio_clock_config
函数原形	void rcu_sdio_clock_config(uint32_t sdio_clock_source);
功能描述	配置SDIO时钟源选择
先决条件	-
被调用函数	-
输入参数{in}	
<b>sdio_clock_source</b>	SDIO时钟源选择
<i>RCU_SDIOSRC_PL</i> <i>L</i>	PLL时钟选择为SDIO时钟源
<i>RCU_SDIOSRC_PL</i> <i>LDIG</i>	PLLDIG选择为SDIO时钟源
<i>RCU_SDIOSRC_IR</i> <i>C16M</i>	IRC16M时钟作为SDIO时钟源
<i>RCU_SDIOSRC_H</i> <i>XTAL</i>	HXTAL选择为SDIO时钟源
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* select PLL clock as sdio clock source */
```

```
rcu_sdio_clock_config(RCU_SDIOSRC_PLL);
```

### 函数 **rcu\_sdio\_div\_config**

函数rcu\_sdio\_div\_config描述见下表：

表 3-669. 函数 rcu\_sdio\_div\_config

函数名称	rcu_sdio_div_config
函数原形	void rcu_sdio_div_config(uint32_t sdio_div);
功能描述	配置SDIO时钟源分频系数
先决条件	-
被调用函数	-
输入参数{in}	
sdio_div	SDIO时钟源分频系数
RCU_SDIODIV_DIV x	SDIO时钟输入源/x,(x = 1....32)
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* SDIODIV input source clock divided by 4 */
```

```
rcu_sdio_div_config(RCU_SDIODIV_DIV4);
```

### 函数 rcu\_usbfs\_clock\_config

函数rcu\_usbfs\_clock\_config描述见下表:

表 3-670. 函数 rcu\_usbfs\_clock\_config

函数名称	rcu_usbfs_clock_config
函数原形	void rcu_usbfs_clock_config(uint32_t usbfs_clock_source);
功能描述	配置USBFS时钟源选择
先决条件	-
被调用函数	-
输入参数{in}	
usbfs_clock_source	USBFS时钟源选择
RCU_USBFSRC_SRC_PLL	选择PLL时钟作为USB时钟源
RCU_USBFSRC_SRC_PLLDIG	选择PLLDIG时钟作为USB时钟源
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* PLL clock selected as USB source clock */
```

```
rcu_usbfs_clock_config(RCU_USBFS_SRC_PLL);
```

### 函数 rcu\_usbfs\_div\_config

函数rcu\_usbfs\_div\_config描述见下表:

表 3-671. 函数 rcu\_usbfs\_div\_config

函数名称	rcu_usbfs_div_config
函数原形	void rcu_usbfs_div_config(uint32_t usbfs_div);
功能描述	配置USBFS时钟源分频系数
先决条件	-
被调用函数	-
输入参数{in}	
usbfs_div	USBFS时钟源分频系数
RCU_USBFS_DIVx	USB时钟源/x(x = 1,...,32)
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* USBFSDIV input source clock divided by 4 */
```

```
rcu_usbfs_div_config(RCU_USBFS_DIV4);
```

### 函数 rcu\_i2c0\_clock\_config

函数rcu\_i2c0\_clock\_config描述见下表:

表 3-672. 函数 rcu\_i2c0\_clock\_config

函数名称	rcu_i2c0_clock_config
函数原形	void rcu_i2c0_clock_config(uint32_t i2c0_clock_source);
功能描述	配置I2C0时钟源
先决条件	-
被调用函数	-
输入参数{in}	
i2c0_clock_source	USBFS时钟源分频系数
RCU_I2C0SRC_CK APB1	选择CK_APB1为I2C0时钟源
RCU_I2C0SRC_CK SYS	选择CK_SYS为I2C0时钟源
RCU_I2C0SRC_IR C16M	选择CK_IRC16M为I2C0时钟源
输出参数{out}	
-	-

返回值	
-	-

例如：

```
/* select IRC16M as I2C0 source clock */
```

```
rcu_i2c0_clock_config(RCU_I2C0SRC_IRC16M);
```

### 函数 usart0\_clock\_source

函数usart0\_clock\_source描述见下表：

**表 3-673. 函数 usart0\_clock\_source**

函数名称	usart0_clock_source
函数原形	void rcu_usart0_clock_config(uint32_t usart0_clock_source);
功能描述	配置USART0时钟源
先决条件	-
被调用函数	-
输入参数{in}	
usart0_clock_source	USART0时钟源选择
RCU_USART0SRC_CKAPB1	选择CK_APB1为USART0时钟源
RCU_USART0SRC_CKSYS	选择CK_SYS为USART0时钟源
RCU_USART0SRC_LXTAL	选择LXTAL为USART0时钟源
RCU_USART0SRC_IRC16M	选择IRC16M为USART0时钟源
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* select APB1 clock as USART0 clock */
```

```
rcu_usart0_clock_config(RCU_USART0SRC_CKAPB1);
```

### 函数 usart2\_clock\_source

函数usart2\_clock\_source描述见下表：

**表 3-674. 函数 usart2\_clock\_source**

函数名称	usart2_clock_source
函数原形	void rcu_usart2_clock_config(uint32_t usart2_clock_source);

功能描述	配置USART2时钟源
先决条件	-
被调用函数	-
输入参数{in}	
<b>usart2_clock_source</b>	USART2时钟源选择
<i>RCU_USART2SRC_CKAPB1</i>	选择CK_APB1为USART2时钟源
<i>RCU_USART2SRC_CKSYS</i>	选择CK_SYS为USART2时钟源
<i>RCU_USART2SRC_LXTAL</i>	选择LXTAL为USART2时钟源
<i>RCU_USART2SRC_IRC16M</i>	选择IRC16M为USART2时钟源
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* select CK_APB1 as USART2 source clock */
```

```
rcu_usart2_clock_config(RCU_USART2SRC_CKAPB1);
```

### 函数 `rcu_irc16m_div_config`

函数`rcu_irc16m_div_config`描述见下表：

**表 3-675. 函数 `rcu_irc16m_div_config`**

函数名称	<code>rcu_irc16m_div_config</code>
函数原形	<code>void rcu_irc16m_div_config(uint32_t irc16m_div);</code>
功能描述	配置提供给系统时钟的IRC16M时钟分频系数
先决条件	-
被调用函数	-
输入参数{in}	
<b>irc16m_div</b>	提供给系统时钟的IRC16M时钟分频系数
<i>RCU_IRC16M_DIVx</i>	IRC16M/x提供给系统时钟(x=1,2,3,...,512)
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* IRC16M clock divided by 4 for system clock */
```

```
rcu_irc16m_div_config(RCU_IRC16M_DIV4);
```

### 函数 rcu\_timer\_clock\_prescaler\_config

函数rcu\_timer\_clock\_prescaler\_config描述见下表：

表 3-676. 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\_lxtal\_drive\_capability\_config

函数rcu\_lxtal\_drive\_capability\_config描述见下表：

表 3-677. 函数 rcu\_lxtal\_drive\_capability\_config

函数名称	rcu_lxtal_drive_capability_config
函数原形	void rcu_lxtal_drive_capability_config(uint32_t lxtal_dricap);
功能描述	配置LXTAL驱动能力
先决条件	-
被调用函数	-
输入参数{in}	
lxtal_dricap	LXTAL驱动能力
RCU_LXTALDRIVER_LOWER_DRIVE	低驱动能力
RCU_LXTALDRIVER_HIGH_DRIVE	中高驱动能力



RCU_LXTALDRI_HI GHER_DRIVE	高驱动能力
RCU_LXTALDRI_HI GHEST_DRIVEI	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\_osci\_stab\_wait

函数rcu\_osci\_stab\_wait描述见下表：

表 3-678. 函数 rcu\_osci\_stab\_wait

函数名称	rcu_osci_stab_wait
函数原形	ErrStatus rcu_osci_stab_wait(rcu_osci_type_enum osci);
功能描述	等待振荡器稳定标志位置位或振荡器起振超时
先决条件	-
被调用函数	rcu_flag_get
输入参数{in}	
osci	振荡器类型，参考 <a href="#">表3-631. 枚举类型rcu_osci_type_enum</a>
输出参数{out}	
-	-
返回值	
ErrStatus	SUCCESS 或 ERROR

例如：

```
/* wait for oscillator stabilization flag */
```

```
if(SUCCESS == rcu_osci_stab_wait(RCU_HXTAL)){
}
```

### 函数 rcu\_osci\_on

函数rcu\_osci\_on描述见下表：

表 3-679. 函数 rcu\_osci\_on

函数名称	rcu_osci_on
函数原形	void rcu_osci_on(rcu_osci_type_enum osci);
功能描述	打开振荡器

先决条件	-
被调用函数	-
输入参数{in}	
osci	振荡器类型，参考 <a href="#">表3-631. 枚举类型rcu_osci_type_enum</a>
-	-
返回值	
-	-

例如：

```
/* turn on the high speed crystal oscillator */
```

```
rcu_osci_on(RCU_HXTAL);
```

### 函数 rcu\_osci\_off

函数rcu\_osci\_off描述见下表：

表 3-680. 函数 rcu\_osci\_off

函数名称	rcu_osci_off
函数原形	void rcu_osci_off(rcu_osci_type_enum osci);
功能描述	关闭振荡器
先决条件	-
被调用函数	-
输入参数{in}	
osci	振荡器类型，参考 <a href="#">表3-631. 枚举类型rcu_osci_type_enum</a>
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* turn off the high speed crystal oscillator */
```

```
rcu_osci_off(RCU_HXTAL);
```

### 函数 rcu\_osci\_bypass\_mode\_enable

函数rcu\_osci\_bypass\_mode\_enable描述见下表：

表 3-681. 函数 rcu\_osci\_bypass\_mode\_enable

函数名称	rcu_osci_bypass_mode_enable
函数原形	void rcu_osci_bypass_mode_enable(rcu_osci_type_enum osci);
功能描述	使能振荡器时钟旁路模式
先决条件	HXTALEN或LXTALEN应在使能振荡器时钟旁路模式前先复位
被调用函数	-
输入参数{in}	

<b>osci</b>	振荡器类型，参考 <a href="#">表3-631. 枚举类型rcu_osci_type_enum</a>
<i>RCU_HXTAL</i>	高速晶体振荡器
<i>RCU_LXTAL</i>	低速晶体振荡器
<b>输出参数{out}</b>	
-	-
<b>返回值</b>	
-	-

例如：

```
/* enable the high speed crystal oscillator bypass mode */
```

```
rcu_osci_bypass_mode_enable(RCU_HXTAL);
```

### 函数 rcu\_osci\_bypass\_mode\_disable

函数rcu\_osci\_bypass\_mode\_disable描述见下表：

**表 3-682. 函数 rcu\_osci\_bypass\_mode\_disable**

<b>函数名称</b>	rcu_osci_bypass_mode_disable
<b>函数原形</b>	void rcu_osci_bypass_mode_disable(rcu_osci_type_enum osci);
<b>功能描述</b>	禁能振荡器时钟旁路模式
<b>先决条件</b>	HXTALEN或LXTALEN应在使能振荡器时钟旁路模式前先复位
<b>被调用函数</b>	-
<b>输入参数{in}</b>	
<b>osci</b>	振荡器类型，参考 <a href="#">表3-631. 枚举类型rcu_osci_type_enum</a>
<i>RCU_HXTAL</i>	高速晶体振荡器
<i>RCU_LXTAL</i>	低速晶体振荡器
<b>输出参数{out}</b>	
-	-
<b>返回值</b>	
-	-

例如：

```
/* disable the high speed crystal oscillator bypass mode */
```

```
rcu_osci_bypass_mode_disable(RCU_HXTAL);
```

### 函数 rcu\_irc16m\_adjust\_value\_set

函数rcu\_irc16m\_adjust\_value\_set描述见下表：

**表 3-683. 函数 rcu\_irc16m\_adjust\_value\_set**

<b>函数名称</b>	rcu_irc16m_adjust_value_set
<b>函数原形</b>	void rcu_irc16m_adjust_value_set(uint32_t irc16m_adjval);
<b>功能描述</b>	设置内部8MHz RC振荡器时钟调整值
<b>先决条件</b>	-

被调用函数	-
输入参数{in}	
lrc16m_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 3-684. 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}	
spread_spectrum_type	PLL扩频调制类型选择
RCU_SS_TYPE_CENTRAL	选择中心扩频
RCU_SS_TYPE_DOWN	选择向下扩频
输入参数{in}	
modstep	这些位配置PLL扩频调制曲线振幅
uint32_t	0 ~ 0x7FFF，满足modstep * modcnt <= 2 <sup>15</sup> -1
输入参数{in}	
modcnt	这些位配置PLL扩频调制曲线频率
uint32_t	0 ~ 0x1FFF，满足modstep * modcnt <= 2 <sup>15</sup> -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 3-685. 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 3-686. 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\_hxtal\_clock\_monitor\_enable

函数rcu\_hxtal\_clock\_monitor\_enable描述见下表：

**表 3-687. 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 3-688. 函数 rcu\_hxtal\_clock\_monitor\_disable**

函数名称	rcu_hxtal_clock_monitor_disable
函数原形	void rcu_hxtal_clock_monitor_disable(void);
功能描述	禁能HXTAL时钟监视器
先决条件	-
被调用函数	-
输入参数{in}	
-	-
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* disable the HXTAL clock monitor */
rcu_hxtal_clock_monitor_disable();
```

## 函数 rcu\_rf\_hxtal\_clock\_monitor\_enable

函数rcu\_rf\_hxtal\_clock\_monitor\_enable描述见下表：

**表 3-689. 函数 rcu\_rf\_hxtal\_clock\_monitor\_enable**

函数名称	rcu_rf_hxtal_clock_monitor_enable
函数原形	void rcu_rf_hxtal_clock_monitor_enable(void);
功能描述	使能RF 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 RF HXTAL clock monitor */
```

```
rcu_rf_hxtal_clock_monitor_enable();
```

## 函数 rcu\_rf\_hxtal\_clock\_monitor\_disable

函数rcu\_rf\_hxtal\_clock\_monitor\_disable描述见下表：

**表 3-690. 函数 rcu\_rf\_hxtal\_clock\_monitor\_disable**

函数名称	rcu_rf_hxtal_clock_monitor_disable
函数原形	void rcu_rf_hxtal_clock_monitor_disable(void);
功能描述	禁能RF 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 RF HXTAL clock monitor */
```

```
rcu_rf_hxtal_clock_monitor_disable();
```

## 函数 rcu\_voltage\_key\_unlock

函数rcu\_voltage\_key\_unlock描述见下表：

表 3-691. 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 3-692. 函数 rcu\_deepsleep\_voltage\_set

函数名称	rcu_deepsleep_voltage_set
函数原形	void rcu_deepsleep_voltage_set(uint32_t dsvol);
功能描述	设置深度睡眠模式电压值
先决条件	-
被调用函数	-
输入参数{in}	
dsvol	深度睡眠模式电压值
RCU_DEEPSLEEP_V_1_1	在深度睡眠模式下内核电压为1.1V
RCU_DEEPSLEEP_V_1_0	在深度睡眠模式下内核电压为1.0V
RCU_DEEPSLEEP_V_0_9	在深度睡眠模式下内核电压为0.9V
RCU_DEEPSLEEP_V_0_8	在深度睡眠模式下内核电压为0.8V
输出参数{out}	
-	-
返回值	



-	-
---	---

例如：

```
/* set the deep-sleep mode voltage */
```

```
rcu_deepsleep_voltage_set(RCU_DEEPSLEEP_V_1_0);
```

### 函数 rcu\_clock\_freq\_get

函数rcu\_clock\_freq\_get描述见下表：

表 3-693. 函数 rcu\_clock\_freq\_get

函数名称	rcu_clock_freq_get
函数原形	uint32_t rcu_clock_freq_get(rcu_clock_freq_enum clock);
功能描述	获取系统时钟、总线频率
先决条件	-
被调用函数	-
输入参数{in}	
clock	要获取的时钟频率，参考 <a href="#">表3-632. 枚举类型rcu_clock_freq_enum</a>
输出参数{out}	
-	-
返回值	
ck_freq	系统时钟/AHB时钟/APB1时钟/APB2时钟频率

例如：

```
uint32_t temp_freq;
```

```
/* get the system clock frequency */
```

```
temp_freq = rcu_clock_freq_get(CK_SYS);
```

### 函数 rcu\_security\_enable

函数rcu\_security\_enable描述见下表：

表 3-694. 函数 rcu\_security\_enable

函数名称	rcu_security_enable
函数原形	void rcu_security_enable(rcu_sec_enum security);
功能描述	使能RCU寄存器安全属性
先决条件	-
被调用函数	-
输入参数{in}	
security	时钟安全属性，参考- <a href="#">表3-633. 枚举类型rcu_sec_enum</a>
输出参数{out}	
-	-
返回值	

-	-
---	---

例如：

```
/* enable the PLLI2S configuration and status bits security attribution */
```

```
rcu_security_enable (RCU_SEC_PLLI2SSEC);
```

### 函数 rcu\_security\_disable

函数rcu\_security\_disable描述见下表：

**表 3-695. 函数 rcu\_security\_disable**

函数名称	rcu_security_disable
函数原形	void rcu_security_disable(rcu_sec_enum security);
功能描述	禁能RCU寄存器安全属性
先决条件	-
被调用函数	-
输入参数{in}	
security	时钟安全属性，参考 <a href="#">表3-633. 枚举类型rcu_sec_enum</a>
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* enable the PLLI2S configuration and status bits security attribution */
```

```
rcu_security_disable (RCU_SEC_PLLI2SSEC);
```

### 函数 rcu\_privilege\_enable

函数rcu\_privilege\_enable描述见下表：

**表 3-696. 函数 rcu\_privilege\_enable**

函数名称	rcu_privilege_enable
函数原形	void rcu_privilege_enable(void);
功能描述	使能RCU寄存器特权保护
先决条件	-
被调用函数	-
输入参数{in}	
-	-
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* privileged access enable */
```

```
rcu_privilege_enable ();
```

### 函数 rcu\_privilege\_disable

函数rcu\_privilege\_disable描述见下表：

表 3-697. 函数 rcu\_privilege\_disable

函数名称	rcu_privilege_disable
函数原形	void rcu_privilege_disable(void);
功能描述	禁能RCU寄存器特权保护
先决条件	-
被调用函数	-
输入参数{in}	
-	-
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* privileged access disable */
```

```
rcu_privilege_disable ();
```

### 函数 rcu\_flag\_get

函数rcu\_flag\_get描述见下表：

表 3-698. 函数 rcu\_flag\_get

函数名称	rcu_flag_get
函数原形	FlagStatus rcu_flag_get(rcu_flag_enum flag);
功能描述	获取时钟稳定和外设复位标志
先决条件	-
被调用函数	-
输入参数{in}	
flag	时钟稳定和外设复位标志，参考 <a href="#">表3-628. 枚举类型rcu_int_flag_enum</a>
输出参数{out}	
-	-
返回值	
FlagStatus	SET 或 RESET

例如：

```

/* get the clock stabilization 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 3-699. 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 3-700. 函数 rcu\_interrupt\_flag\_get**

函数名称	rcu_interrupt_flag_get
函数原形	FlagStatus rcu_interrupt_flag_get(rcu_int_flag_enum int_flag);
功能描述	获取时钟稳定中断和时钟阻塞中断标志
先决条件	-
被调用函数	-
输入参数{in}	
int_flag	中断以及CKM标志，参考 <a href="#">表3-628. 枚举类型rcu_int_flag_enum</a>
输出参数{out}	
-	-
返回值	
FlagStatus	SET 或 RESET

例如：

```

/* get the clock stabilization interrupt flag */

if(SET == rcu_interrupt_flag_get(RCU_INT_FLAG_HXTALSTB)){

}

```

### 函数 rcu\_interrupt\_flag\_clear

函数rcu\_interrupt\_flag\_clear描述见下表：

表 3-701. 函数 rcu\_interrupt\_flag\_clear

函数名称	rcu_interrupt_flag_clear
函数原形	void rcu_interrupt_flag_clear(rcu_int_flag_clear_enum int_flag);
功能描述	清除中断标志和时钟阻塞中断标志
先决条件	-
被调用函数	-
输入参数{in}	
int_flag	时钟稳定和阻塞中断标志清除，参考 <a href="#">表3-629. 枚举类型 rcu_int_flag_clear_enum</a>
输出参数{out}	
-	-
返回值	
-	-

例如：

```

/* clear the interrupt HXTAL stabilization interrupt flag */

rcu_interrupt_flag_clear(RCU_INT_FLAG_HXTALSTB_CLR);

```

### 函数 rcu\_security\_flag\_get

函数rcu\_security\_flag\_get描述见下表：

表 3-702. 函数 rcu\_security\_flag\_get

函数名称	rcu_security_flag_get
函数原形	FlagStatus rcu_security_flag_get(rcu_sec_flag_enum sec_flag);
功能描述	获取外设时钟的安全属性标志
先决条件	-
被调用函数	-
输入参数{in}	
sec_flag	安全标志，参考 <a href="#">表3-634. 枚举类型rcu_sec_flag_enum</a>
输出参数{out}	
-	-
返回值	
FlagStatus	SET 或 RESET

例如：

```

/* get the peripherals SPI0 clock secure flag */

If(SET == rcu_security_flag_get(RCU_SEC_FLAG_APB2_SPI0)){

};

```

### 函数 rcu\_interrupt\_enable

函数rcu\_interrupt\_enable描述见下表：

**表 3-703. 函数 rcu\_interrupt\_enable**

函数名称	rcu_interrupt_enable
函数原形	void rcu_interrupt_enable(rcu_int_enum interrupt);
功能描述	使能时钟稳定中断
先决条件	-
被调用函数	-
输入参数{in}	
interrupt	时钟稳定中断,参考 <a href="#">表3-630. 枚举类型rcu_int_enum</a>
输出参数{out}	
-	-
返回值	
-	-

例如：

```

/* enable the HXTAL stabilization interrupt */

rcu_interrupt_enable(RCU_INT_HXTALSTB);

```

### 函数 rcu\_interrupt\_disable

函数rcu\_interrupt\_disable描述见下表：

**表 3-704. 函数 rcu\_interrupt\_disable**

函数名称	rcu_interrupt_disable
函数原形	void rcu_interrupt_disable(rcu_int_enum interrupt);
功能描述	禁能时钟稳定中断
先决条件	-
被调用函数	-
输入参数{in}	
interrupt	时钟稳定中断,参考 <a href="#">表3-630. 枚举类型rcu_int_enum</a>
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* disable the HXTAL stabilization interrupt */
```

```
rcu_interrupt_disable(RCU_INT_HXTALSTB);
```

## 3.22. RTC

实时时钟RTC通常被用作时钟日历。位于备份域中的RTC电路，包含一个32位的累加计数器、两个闹钟、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 3-705. 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_ICS	初始化状态和控制寄存器
RTC_PSC	预分频寄存器
RTC_WUT	唤醒定时器寄存器
RTC_COSC	粗校准寄存器
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
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_PPM_CTL	特权保护模式控制寄存器
RTC_SPM_CTL	安全保护模式控制寄存器
RTC_STAT	状态寄存器
RTC_NSMI_STAT	非安全屏蔽中断状态寄存器
RTC_SMI_STAT	安全屏蔽中断状态寄存器
RTC_STATC	状态标志清除寄存器
RTC_BKPx(x = 0, 1,	备份寄存器

寄存器名称	寄存器描述
2, ..., 18, 19)	

### 3.22.2. 外设库函数描述

RTC库函数列表如下表所示：

**表 3-706. 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
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_software_bkp_reset	软件复位RTC_BKP寄存器
rtc_tamper_without_bkp_seset	配置侵入事件擦除RTC_BKP寄存器
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_nsec_interrupt_flag_get	获取非安全中断标志位
rtc_nsec_interrupt_flag_clear	清除非安全中断标志位
rtc_sec_interrupt_flag_get	获取安全中断标志位
rtc_sec_interrupt_flag_clear	清除安全中断标志位
rtc_output_pad_select	配置RTC输出引脚
rtc_alarm_output_config	配置RTC闹钟输出
rtc_calibration_output_config	配置RTC校准输出选择



库函数名称	库函数描述
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
rtc_pri_pro_enable	使能RTC特权保护模式
rtc_pri_pro_disable	失能RTC特权保护模式
rtc_sec_pro_enable	使能RTC安全保护模式
rtc_sec_pro_disable	失能RTC安全保护模式
rtc_bkp_zonea_sec_pro_set	设置RTC_BKP安全保护区域a结束值
rtc_bkp_zoneb_sec_pro_set	设置RTC_BKP安全保护区域b结束值
rtc_bkp_zoneb_sec_pro_check	检查RTC_BKP安全保护区域b是否有效

### 结构体 rtc\_parameter\_struct

表 3-707. 结构体 rtc\_parameter\_struct

成员名称	功能描述
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 3-708. 结构体 `rtc_alarm_struct`

成员名称	功能描述
<code>alarm_mask</code>	RTC闹钟屏蔽
<code>weekday_or_date</code>	指定RTC闹钟是日期还是星期几
<code>alarm_day</code>	RTC闹钟日期或者星期几的值
<code>alarm_hour</code>	RTC闹钟小时值
<code>alarm_minute</code>	RTC闹钟分钟值: 0x0 - 0x59(BCD格式)
<code>alarm_second</code>	RTC闹钟秒数值: 0x0 - 0x59(BCD格式)
<code>am_pm</code>	RTC闹钟AM/PM数值

## 结构体 `rtc_timestamp_struct`

表 3-709. 结构体 `rtc_timestamp_struct`

成员名称	功能描述
<code>timestamp_month</code>	RTC时间戳月份值
<code>timestamp_date</code>	RTC 时间戳日期值: 0x1 - 0x31(BCD 格式)
<code>timestamp_day</code>	RTC时间戳星期值
<code>timestamp_hour</code>	RTC时间戳小时值
<code>timestamp_minute</code>	RTC时间戳分钟值: 0x0 - 0x59(BCD格式)
<code>timestamp_second</code>	RTC时间戳秒数值: 0x0 - 0x59(BC 格式)
<code>am_pm</code>	RTC时间戳AM/PM数值

## 结构体 `rtc_tamper_struct`

表 3-710. 结构体 `rtc_tamper_struct`

成员名称	功能描述
<code>tamper_source</code>	RTC侵入检测源
<code>tamper_trigger</code>	RTC侵入事件检测触发沿
<code>tamper_filter</code>	RTC 侵入事件检测在电平检测期间需要的连续采样次数
<code>tamper_sample_frequency</code>	RTC侵入事件电平模式检测的采样频率
<code>tamper_precharge_enable</code>	RTC在电压电平检测期间的预充电功能
<code>tamper_precharge_time</code>	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
<code>tamper_with_timestamp</code>	RTC侵入事件触发时间戳

## 函数 `rtc_deinit`

函数`rtc_deinit`描述见下表:

表 3-711. 函数 rtc\_deinit

函数名称	rtc_deinit
函数原型	ErrStatus rtc_deinit(void);
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	
ErrStatus	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 3-712. 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}	
rtc_initpara_struct	初始化结构体，结构体成员参考 <a href="#">表3-707. 结构体rtc_parameter_struct</a>
输出参数{out}	
-	-
返回值	
ErrStatus	ERROR 或 SUCCESS

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 3-713. 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 或 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 3-714. 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 3-715. 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	
ErrStatus	ERROR 或 SUCCESS

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 3-716. 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}	
rtc_initpara_struct	初始化结构体，结构体成员参考 <a href="#">表3-707. 结构体rtc_parameter_struct</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 3-717. 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	-
输入参数{in}	
-	-
输出参数{out}	
-	-
返回值	
uint32_t	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 3-718. 函数 rtc\_alarm\_config

函数名称	rtc_alarm_config
函数原型	void rtc_alarm_config(uint8_t rtc_alarm, rtc_alarm_struct* rtc_alarm_time);
功能描述	配置RTC闹钟
先决条件	-
被调用函数	-
输入参数{in}	
rtc_alarm	闹钟选择
RTC_ALARM0	闹钟0
RTC_ALARM1	闹钟1

输入参数{in}	
<b>rtc_alarm_time</b>	闹钟结构体，结构体成员参考 <a href="#">表3-708. 结构体rtc_alarm_struct</a>
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/*rtc_alarm_config*/
```

```
rtc_alarm_struct rtc_alarm_time;
```

```
rtc_alarm_config (RTC_ALARM0, &rtc_alarm_time);
```

### 函数 **rtc\_alarm\_subsecond\_config**

函数rtc\_alarm\_subsecond\_config描述见下表：

**表 3-719. 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	-
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
输入参数{in}	
<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_1</i> 4	屏蔽RTC_ALARM0SS_SSC[14:11], SSC[10:0]位用于时间匹配
<i>RTC_MASKSSC_12_1</i> 4	屏蔽RTC_ALARM0SS_SSC[14:12], SSC[11:0]位用于时间匹配
<i>RTC_MASKSSC_13_1</i> 4	屏蔽RTC_ALARM0SS_SSC[14:13], SSC[12:0]位用于时间匹配
<i>RTC_MASKSSC_14</i>	屏蔽RTC_ALARM0SS_SSC[14], SSC[13:0]位用于时间匹配
<i>RTC_MASKSSC_NONE</i>	无屏蔽, SSC[14:0]位用于时间匹配
输入参数{in}	
<b>subsecond</b>	闹钟亚秒值(0x000 - 0x7FFF)
输出参数{out}	
-	-
返回值	
-	-

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 3-720. 函数 **rtc\_alarm\_get****

<b>函数名称</b>	rtc_alarm_get
<b>函数原型</b>	void rtc_alarm_get(uint8_t rtc_alarm, rtc_alarm_struct* rtc_alarm_time);
<b>功能描述</b>	获取RTC闹钟
<b>先决条件</b>	-
<b>被调用函数</b>	-
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3-708. 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 3-721. 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}	
rtc_alarm	闹钟选择
RTC_ALARM0	闹钟0
RTC_ALARM1	闹钟1
输出参数{out}	
-	-
返回值	
uint32_t	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 3-722. 函数 rtc\_alarm\_enable**

函数名称	rtc_alarm_enable
函数原型	void rtc_alarm_enable(uint8_t rtc_alarm);
功能描述	使能RTC闹钟
先决条件	-
被调用函数	-
输入参数{in}	
rtc_alarm	闹钟选择
RTC_ALARM0	闹钟0
RTC_ALARM1	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 3-723. 函数 rtc\_alarm\_disable

函数名称	rtc_alarm_disable
函数原型	ErrStatus rtc_alarm_disable(uint8_t rtc_alarm);
功能描述	失能RTC闹钟
先决条件	-
被调用函数	-
输入参数{in}	
rtc_alarm	闹钟选择
RTC_ALARM0	闹钟0
RTC_ALARM1	闹钟1
输出参数{out}	
-	-
返回值	
ErrStatus	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 3-724. 函数 rtc\_timestamp\_enable

函数名称	rtc_timestamp_enable
函数原型	void rtc_timestamp_enable(uint32_t edge);
功能描述	使能RTC时间戳
先决条件	-
被调用函数	-
输入参数{in}	
edge	选定哪种边沿触发时间戳检测
RTC_TIMESTAMP_RISING_EDGE	上升沿是时间戳事件有效检测沿
RTC_TIMESTAMP_FALLING_EDGE	下降沿是时间戳事件有效检测沿
输出参数{out}	
-	-
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 3-725. 函数 rtc\_timestamp\_disable**

函数名称	rtc_timestamp_disable
函数原型	void rtc_timestamp_disable(void);
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 3-726. 函数 rtc\_timestamp\_get**

函数名称	rtc_timestamp_get
函数原型	void rtc_timestamp_get(rtc_timestamp_struct* rtc_timestamp);
功能描述	获取RTC时间戳时间和日期
先决条件	-
被调用函数	-
输入参数{in}	
-	-
输出参数{out}	
rtc_timestamp	时间戳结构体, 结构体成员参考 <a href="#">表3-709. 结构体 rtc_timestamp_struct</a>
返回值	
-	-

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 3-727. 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	-
输入参数{in}	
-	-
输出参数{out}	
-	-
返回值	
uint32_t	RTC时间戳亚秒值

例如：

```
/* get RTC time-stamp subsecond */

uint32_t subsecond = rtc_timestamp_subsecond_get();
```

### 函数 rtc\_tamper\_enable

函数rtc\_tamper\_enable描述见下表：

表 3-728. 函数 rtc\_tamper\_enable

函数名称	rtc_tamper_enable
函数原型	void rtc_tamper_enable(rtc_tamper_struct* rtc_tamper);
功能描述	使能RTC侵入检测
先决条件	-
被调用函数	-
输入参数{in}	
rtc_tamper	tamper化结构体，结构体成员参考 <a href="#">表3-710. 结构体rtc_tamper_struct</a>
输出参数{out}	
-	-
返回值	
-	-

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 3-729. 函数 rtc\_tamper\_disable

函数名称	rtc_tamper_disable
函数原型	void rtc_tamper_disable(uint32_t source);
功能描述	失能RTC侵入检测
先决条件	-
被调用函数	-
输入参数{in}	
source	选定被失能的侵入检测来源
RTC_TAMPER0	RTC tamper0
RTC_TAMPER1	RTC tamper1
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* disable RTC tamper */

rtc_tamper_disable(RTC_TAMPER0);
```

### 函数 rtc\_software\_bkp\_reset

函数rtc\_software\_bkp\_reset描述见下表：

表 3-730. 函数 rtc\_software\_bkp\_reset

函数名称	rtc_software_bkp_reset
函数原型	void rtc_software_bkp_reset(void);
功能描述	软件复位RTC_BKP寄存器
先决条件	-
被调用函数	-
输入参数{in}	
-	-
输出参数{out}	
-	-
返回值	

-	-
---	---

例如:

```
/* reset the RTC_BKP registers by software */
```

```
rtc_software_bkp_reset ();
```

### 函数 `rtc_tamper_without_bkp_seset`

函数 `rtc_tamper_without_bkp_seset` 描述见下表:

**表 3-731. 函数 `rtc_tamper_without_bkp_seset`**

函数名称	<code>rtc_tamper_without_bkp_seset</code>
函数原型	<code>void rtc_tamper_without_bkp_seset(uint32_t ne_source);</code>
功能描述	配置侵入事件擦除RTC_BKP寄存器
先决条件	-
被调用函数	-
输入参数{in}	
<b>ne_source</b>	配置侵入事件源触发
<code>RTC_TAMPXNOER_NONE</code>	tamper0和tamper 1不会擦除bkp寄存器
<code>RTC_TAMPXNOER_TP0</code>	tamper0不会擦除bkp寄存器
<code>RTC_TAMPXNOER_TP1</code>	tamper1不会擦除bkp寄存器
<code>RTC_TAMPXNOER_TP0_TP1</code>	tamper0和tamper 1会擦除bkp寄存器
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* set tamper0 will not trigger RTC_BKP registers */
```

```
rtc_tamper_without_bkp_seset(RTC_TAMPXNOER_TP0);
```

### 函数 `rtc_interrupt_enable`

函数 `rtc_interrupt_enable` 描述见下表:

**表 3-732. 函数 `rtc_interrupt_enable`**

函数名称	<code>rtc_interrupt_enable</code>
函数原型	<code>void rtc_interrupt_enable(uint32_t interrupt);</code>
功能描述	使能RTC指定的中断
先决条件	-

被调用函数	-
输入参数{in}	
<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_WAKEUP</i>	唤醒定时器中断
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* enable specified RTC interrupt*/
rtc_interrupt_enable(RTC_INT_ALARM0);
```

### 函数 **rtc\_interrupt\_disable**

函数rtc\_interrupt\_disable描述见下表：

**表 3-733. 函数 rtc\_interrupt\_disable**

函数名称	rtc_interrupt_disable
函数原型	void rtc_interrupt_disable(uint32_t interrupt);
功能描述	失能RTC指定中断
先决条件	-
被调用函数	-
输入参数{in}	
<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_WAKEUP</i>	唤醒定时器中断
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* disble specified RTC interrupt */
rtc_interrupt_disable(RTC_INT_ALARM0);
```

**函数 rtc\_flag\_get**

函数rtc\_flag\_get描述见下表:

**表 3-734. 函数 rtc\_flag\_get**

函数名称	rtc_flag_get
函数原型	FlagStatus rtc_flag_get(uint32_t flag);
功能描述	获取指定中断标志位
先决条件	-
被调用函数	-
<b>输入参数{in}</b>	
flag	选定被获取的中断标志
RTC_FLAG_SCP	平滑校准挂起标志
RTC_FLAG_TP1	tamper 1 事件标志
RTC_FLAG_TP0	tamper 0 事件标志
RTC_FLAG_TSOVR	时间戳事件溢出标志
RTC_FLAG_TS	时间戳事件标志
RTC_FLAG_ALARM0	Alarm0 发生标志
RTC_FLAG_ALARM1	Alarm1 发生标志
RTC_FLAG_WT	唤醒定时器发生标志
RTC_FLAG_INIT	进入初始化模式
RTC_FLAG_RSYN	寄存器同步标志
RTC_FLAG_YCM	年份配置标志
RTC_FLAG_SOP	移位功能操作挂起标志
RTC_FLAG_ALARM0W	Alarm0 配置可写标志
RTC_FLAG_ALARM1W	Alarm1 配置可写标志
RTC_FLAG_WTW	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_TS);
```

**函数 rtc\_flag\_clear**

函数rtc\_flag\_clear描述见下表:

**表 3-735. 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	-
输入参数{in}	
<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_TP0</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_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
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* clear time-stamp event flag */
```

```
rtc_flag_clear (RTC_FLAG_TS);
```

### 函数 `rtc_nsec_interrupt_flag_get`

函数 `rtc_nsec_interrupt_flag_get` 描述见下表：

表 3-736. 函数 `rtc_nsec_interrupt_flag_get`

函数名称	<code>rtc_nsec_interrupt_flag_get</code>
函数原型	<code>FlagStatus rtc_nsec_interrupt_flag_get(uint32_t int_flag);</code>
功能描述	获取特定的非安全中断标志位
先决条件	-
被调用函数	-
输入参数{in}	
<b>int_flag</b>	非安全中断标志
<i>RTC_NSMI_STAT_ALRMONSMF</i>	alarm0 非安全中断屏蔽标志
<i>RTC_NSMI_STAT_ALRM1NSMF</i>	Alarm1 非安全中断屏蔽标志
<i>RTC_NSMI_STAT_WTNSMF</i>	唤醒定时器非安全中断屏蔽标志
<i>RTC_NSMI_STAT_TSNSMF</i>	时间戳非安全中断屏蔽标志
<i>RTC_NSMI_STAT_TSOVRNSMF</i>	时间戳事件非安全中断屏蔽标志

<i>RTC_NSMI_STAT_TP0NSMF</i>	tamper 0事件非安全中断屏蔽标志
<i>RTC_NSMI_STAT_TP1NSMF</i>	tamper 1事件非安全中断屏蔽标志
输出参数{out}	
-	-
返回值	
<b>FlagStatus</b>	SET 或 RESET

例如:

```
/* get alarm0 non-secure interrupt flag */
```

```
rtc_nsec_interrupt_flag_get(RTC_NSMI_STAT_ALRM0NSMF);
```

### 函数 `rtc_nsec_interrupt_flag_clear`

函数`rtc_nsec_interrupt_flag_clear`描述见下表:

**表 3-737. 函数 `rtc_nsec_interrupt_flag_clear`**

函数名称	<code>rtc_nsec_interrupt_flag_clear</code>
函数原型	<code>FlagStatus rtc_nsec_interrupt_flag_clear(uint32_t int_flag);</code>
功能描述	清除特定的非安全中断标志位
先决条件	-
被调用函数	-
输入参数{in}	
<b>int_flag</b>	非安全中断标志
<i>RTC_NSMI_STAT_ALRM0NSMF</i>	alarm0 非安全中断屏蔽标志
<i>RTC_NSMI_STAT_ALRM1NSMF</i>	Alarm1 非安全中断屏蔽标志
<i>RTC_NSMI_STAT_WTNSMF</i>	唤醒定时器非安全中断屏蔽标志
<i>RTC_NSMI_STAT_TSNSMF</i>	时间戳非安全中断屏蔽标志
<i>RTC_NSMI_STAT_TSOVRNSMF</i>	时间戳事件非安全中断屏蔽标志
<i>RTC_NSMI_STAT_TP0NSMF</i>	tamper 0事件非安全中断屏蔽标志
<i>RTC_NSMI_STAT_TP1NSMF</i>	tamper 1事件非安全中断屏蔽标志
输出参数{out}	
-	-
返回值	
<b>FlagStatus</b>	SET 或 RESET

例如：

```
/* clear alarm0 non-secure interrupt flag */
```

```
rtc_nsec_interrupt_flag_clear(RTC_NSMI_STAT_ALRM0NSMF);
```

### 函数 rtc\_sec\_interrupt\_flag\_get

函数rtc\_sec\_interrupt\_flag\_get描述见下表：

表 3-738. 函数 rtc\_sec\_interrupt\_flag\_get

函数名称	rtc_sec_interrupt_flag_get
函数原型	FlagStatus rtc_sec_interrupt_flag_get(uint32_t int_flag);
功能描述	获取特定的安全中断标志位
先决条件	-
被调用函数	-
输入参数{in}	
int_flag	安全中断标志
RTC_SMI_STAT_ALR M0SMF	alarm0 安全中断屏蔽标志
RTC_SMI_STAT_ALR M1SMF	Alarm1 安全中断屏蔽标志
RTC_SMI_STAT_WTS MF	唤醒定时器安全中断屏蔽标志
RTC_SMI_STAT_TSS MF	时间戳安全中断屏蔽标志
RTC_SMI_STAT_TSO VRSMF	时间戳事件安全中断屏蔽标志
RTC_SMI_STAT_TP0S MF	tamper 0事件安全中断屏蔽标志
RTC_SMI_STAT_TP1S MF	tamper 1事件安全中断屏蔽标志
输出参数{out}	
-	-
返回值	
FlagStatus	SET 或 RESET

例如：

```
/* get alarm0 secure interrupt flag */
```

```
rtc_nsec_interrupt_flag_get(RTC_SMI_STAT_ALRM0SMF);
```

### 函数 rtc\_sec\_interrupt\_flag\_clear

函数rtc\_sec\_interrupt\_flag\_clear描述见下表：

表 3-739. 函数 `rtc_sec_interrupt_flag_clear`

函数名称	<code>rtc_sec_interrupt_flag_clear</code>
函数原型	<code>FlagStatus rtc_sec_interrupt_flag_clear(uint32_t int_flag);</code>
功能描述	清除特定的非安全中断标志位
先决条件	-
被调用函数	-
输入参数{in}	
<b>flag</b>	安全中断标志
<code>RTC_SMI_STAT_ALR M0SMF</code>	alarm0 安全中断屏蔽标志
<code>RTC_SMI_STAT_ALR M1SMF</code>	Alarm1 安全中断屏蔽标志
<code>RTC_SMI_STAT_WTS MF</code>	唤醒定时器安全中断屏蔽标志
<code>RTC_SMI_STAT_TSS MF</code>	时间戳安全中断屏蔽标志
<code>RTC_SMI_STAT_TSO VRSMF</code>	时间戳事件安全中断屏蔽标志
<code>RTC_SMI_STAT_TP0S MF</code>	tamper 0事件安全中断屏蔽标志
<code>RTC_SMI_STAT_TP1S MF</code>	tamper 1事件安全中断屏蔽标志
输出参数{out}	
-	-
返回值	
<b>FlagStatus</b>	SET 或 RESET

例如：

```
/* clear alarm0 secure interrupt flag */
```

```
rtc_nsec_interrupt_flag_clear(RTC_SMI_STAT_ALRM0SMF);
```

### 函数 `rtc_output_pad_select`

函数`rtc_output_pad_select`描述见下表：

表 3-740. 函数 `rtc_output_pad_select`

函数名称	<code>rtc_output_pad_select</code>
函数原型	<code>void rtc_output_pad_select(uint32_t pad);</code>
功能描述	选择RTC输出引脚
先决条件	-
被调用函数	-
输入参数{in}	
<b>pad</b>	指定RTC输出引脚

<i>RTC_OUT_PC15</i>	RTC输出引脚为PC15
<i>RTC_OUT_PA3_PA8</i>	RTC输出引脚为PA3或PA8
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* select the rtc output pad is PC15 */
```

```
rtc_output_pad_select(RTC_OUT_PC15);
```

### 函数 `rtc_alarm_output_config`

函数`rtc_alarm_output_config`描述见下表：

表 3-741. 函数 `rtc_alarm_output_config`

函数名称	<code>rtc_alarm_output_config</code>
函数原型	<code>void rtc_alarm_output_config(uint32_t source, uint32_t mode);</code>
功能描述	配置RTC闹钟输出源
先决条件	-
被调用函数	-
输入参数{in}	
<b>source</b>	特定信号输出
<i>RTC_ALARM0_HIGH</i>	当alarm0标志位置位时，输出引脚为高电平
<i>RTC_ALARM0_LOW</i>	当alarm0标志位置位时，输出引脚为低电平
<i>RTC_ALARM1_HIGH</i>	当alarm1标志位置位时，输出引脚为高电平
<i>RTC_ALARM1_LOW</i>	当alarm1标志位置位时，输出引脚为低电平
<i>RTC_WAKEUP_HIGH</i>	当唤醒标志位置位时，输出引脚为高电平
<i>RTC_WAKEUP_LOW</i>	当唤醒标志位置位时，输出引脚为低电平
输入参数{in}	
<b>mode</b>	当输出闹钟信号时指定输出引脚的模式
<i>RTC_ALARM_OUTPUT_OD</i>	开漏输出
<i>RTC_ALARM_OUTPUT_PP</i>	推挽输出
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* configure rtc alternate output source */
```

```
rtc_alter_output_config(RTC_ALARM0_LOW, RTC_ALARM_OUTPUT_PP);
```

函数 `rtc_calibration_output_config`

函数`rtc_calibration_output_config`描述见下表:

表 3-742. 函数 `rtc_calibration_output_config`

函数名称	<code>rtc_calibration_output_config</code>
函数原型	<code>void rtc_calibration_output_config(uint32_t source);</code>
功能描述	配置RTC校准输出源
先决条件	-
被调用函数	-
输入参数{in}	
<b>source</b>	指定输出信号
<code>RTC_CALIBRATION_512HZ</code>	当外部低速时钟频率为32768Hz并且RTC_PSC为默认值，输出512Hz信号
<code>RTC_CALIBRATION_1HZ</code>	当外部低速时钟频率为32768Hz并且RTC_PSC为默认值，输出1Hz信号
输出参数{out}	
-	-
返回值	
-	-

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 3-743. 函数 `rtc_hour_adjust`

函数名称	<code>rtc_hour_adjust</code>
函数原型	<code>void rtc_hour_adjust(uint32_t operation);</code>
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	-
输入参数{in}	
<b>operation</b>	小时调整操作
<code>RTC_CTL_A1H</code>	增加一个小时
<code>RTC_CTL_S1H</code>	减少一个小时
输出参数{out}	
-	-
返回值	
-	-

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 3-744. 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}	
add	在当前时间上增加1S或者不增加
RTC_SHIFT_ADD1S_R ESET	无影响
RTC_SHIFT_ADD1S_S ET	在当前时间增加1秒
输入参数{in}	
minus	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 3-745. 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 3-746. 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 3-747. 函数 rtc\_refclock\_detection\_enable

函数名称	rtc_refclock_detection_enable
函数原型	ErrStatus rtc_refclock_detection_enable(void);
功能描述	使能RTC参考时钟检测功能
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 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 3-748. 函数 rtc\_refclock\_detection\_disable**

函数名称	rtc_refclock_detection_disable
函数原型	ErrStatus rtc_refclock_detection_disable(void);
功能描述	失能RTC参考时钟检测功能
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或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 3-749. 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 3-750. 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 或 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 3-751. 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_RTCK_DIV 16	RTC时钟的16分频
WAKEUP_RTCK_DIV 8	RTC时钟的8分频
WAKEUP_RTCK_DIV 4	RTC时钟的4分频

WAKEUP_RTCK_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	
ErrStatus	ERROR 或 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 3-752. 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	-
输入参数{in}	
wakeup_timer	定时器选择
uint16_t	0x0000-0xffff
输出参数{out}	
-	-
返回值	
ErrStatus	ERROR 或 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 3-753. 函数 rtc\_wakeup\_timer\_get

函数名称	rtc_wakeup_timer_get
函数原型	uint16_t rtc_wakeup_timer_get(void);
功能描述	获取唤醒定时器值

先决条件	-
被调用函数	-
输入参数{in}	
-	-
输出参数{out}	
-	-
返回值	
uint16_t	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 3-754. 函数 rtc\_smooth\_calibration\_config

函数名称	rtc_smooth_calibration_config
函数原型	ErrStatus rtc_smooth_calibration_config(uint32_t window, uint32_t plus, uint32_t minus);
功能描述	配置RTC平滑校准
先决条件	-
被调用函数	-
输入参数{in}	
window	校准窗口选择
RTC_CALIBRATION_WINDOW_32S	采用32S校准周期
RTC_CALIBRATION_WINDOW_16S	采用16S校准周期
RTC_CALIBRATION_WINDOW_8S	采用8S校准周期
输入参数{in}	
plus	增加脉冲
RTC_CALIBRATION_PLUS_SET	每2048个脉冲增加一个RTCCLK脉冲
RTC_CALIBRATION_PLUS_RESET	无影响
输入参数{in}	
minus	校准窗口校准周期RTCCLK脉冲屏蔽数（0x0-0x1FF）
输出参数{out}	

-	-
返回值	
ErrStatus	ERROR 或 SUCCESS

例如:

```
/* configure RTC smooth calibration*/
```

```
ErrStatus error_status;
```

```
error_status = rtc_smooth_calibration_config(RTC_CALIBRATION_WINDOW_32S, RTC_
CALIBRATION_PLUS_SET, 0x10);
```

### 函数 rtc\_coarse\_calibration\_enable

函数rtc\_coarse\_calibration\_enable描述见下表:

表 3-755. 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
输入参数{in}	
-	-
输出参数{out}	
-	-
返回值	
ErrStatus	ERROR 或 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 3-756. 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	
ErrStatus	ERROR 或 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 3-757. 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
输入参数{in}	
direction	粗校准方向
CALIB_INCREASE	增加日历更新频率
CALIB_DECREASE	降低日历更新频率
step	
0x00-0x1F	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)
输出参数{out}	
-	-
返回值	
ErrStatus	ERROR 或 SUCCESS

例如:

```
/* config coarse calibration direction and step */
```

```
ErrStatus error_status = rtc_coarse_calibration_config (CALIB_INCREASE, 0x01);
```

### 函数 rtc\_pri\_pro\_enable

函数rtc\_pri\_pro\_enable描述见下表:

**表 3-758. 函数 rtc\_pri\_pro\_enable**

函数名称	rtc_pri_pro_enable
函数原型	void rtc_pri_pro_enable(uint32_t sub_area);
功能描述	使能RTC特权保护
先决条件	-
被调用函数	-
输入参数{in}	
sub_area	指定RTC模块寄存器为特权保护模式
RTC_PPM_CTL_ALRM0PRIP	alarm0特权保护
RTC_PPM_CTL_ALRM1PRIP	alarm1特权保护
RTC_PPM_CTL_WUTPRIP	唤醒定时器特权保护
RTC_PPM_CTL_TSPRIP	时间戳特权保护
RTC_PPM_CTL_TAMPPRIP	tamper事件特权保护(除了备份域寄存器)
RTC_PPM_CTL_CALCPRIP	移位寄存器, 省时, 校准和参考时钟特权保护
RTC_PPM_CTL_INITPRIP	RTC初始化特权保护
RTC_PPM_CTL_RTCPRIP	RTC全局特权保护
RTC_PPM_CTL_BKPRWPRIP	备份域区域a特权保护
RTC_PPM_CTL_BKPWPRIP	备份域区域b特权保护
输出参数{out}	
-	-
返回值	
ErrStatus	ERROR 或 SUCCESS

例如:

```
/* enable RTC alarm 0 privilege protection mode */
```

```
rtc_pri_pro_enable(RTC_PPM_CTL_ALRM0PRIP);
```

函数 **rtc\_pri\_pro\_disable**

函数rtc\_pri\_pro\_disable描述见下表：

表 3-759. 函数 **rtc\_pri\_pro\_disable**

函数名称	rtc_pri_pro_disable
函数原型	void rtc_pri_pro_disable(uint32_t sub_area);
功能描述	失能RTC特权保护
先决条件	-
被调用函数	-
输入参数{in}	
sub_area	指定RTC模块寄存器为特权保护模式
RTC_PPM_CTL_ALRM0PRIP	alarm0特权保护
RTC_PPM_CTL_ALRM1PRIP	alarm1特权保护
RTC_PPM_CTL_WUTPRIP	唤醒定时器特权保护
RTC_PPM_CTL_TSPRIP	时间戳特权保护
RTC_PPM_CTL_TAMPPRIP	tamper事件特权保护(除了备份域寄存器)
RTC_PPM_CTL_CALCPRIP	移位寄存器, 省时, 校准和参考时钟特权保护
RTC_PPM_CTL_INITPRIP	RTC初始化特权保护
RTC_PPM_CTL_RTCPRIP	RTC全局特权保护
RTC_PPM_CTL_BKPRWPRIP	备份域区域a特权保护
RTC_PPM_CTL_BKPRIP	备份域区域b特权保护
输出参数{out}	
-	-
返回值	
ErrStatus	ERROR 或 SUCCESS

例如：

```
/* disable RTC alarm 0 privilege protection mode */
```

```
rtc_pri_pro_disable(RTC_PPM_CTL_ALRM0PRIP);
```

函数 **rtc\_sec\_pro\_enable**

函数rtc\_sec\_pro\_enable描述见下表：



表 3-760. 函数 rtc\_sec\_pro\_enable

函数名称	rtc_sec_pro_enable
函数原型	void rtc_sec_pro_enable(uint32_t sub_area);
功能描述	使能RTC安全保护
先决条件	-
被调用函数	-
输入参数{in}	
sub_area	指定RTC模块寄存器为安全保护模式
RTC_SPM_CTL_ALRM0SECP	alarm0安全保护
RTC_SPM_CTL_ALRM1SECP	alarm1安全保护
RTC_SPM_CTL_WUTSECP	唤醒定时器安全保护
RTC_SPM_CTL_TSSECP	时间戳安全保护
RTC_SPM_CTL_TAMPSECP	tamper事件安全保护(除了备份域寄存器)
RTC_SPM_CTL_CALCSECP	移位寄存器, 省时, 校准和参考时钟安全保护
RTC_SPM_CTL_INITSECP	RTC初始化安全保护
RTC_SPM_CTL_RTCSSECP	RTC全局安全保护
输出参数{out}	
-	-
返回值	
ErrStatus	ERROR 或 SUCCESS

例如:

```
/* enable RTC alarm 0 secure protection mode */
```

```
rtc_sec_pro_enable (RTC_SPM_CTL_ALRM0SECP);
```

### 函数 rtc\_sec\_pro\_disable

函数rtc\_sec\_pro\_disable描述见下表:

表 3-761. 函数 rtc\_sec\_pro\_disable

函数名称	rtc_sec_pro_disable
函数原型	void rtc_sec_pro_disable(uint32_t sub_area);
功能描述	失能RTC安全保护
先决条件	-
被调用函数	-

输入参数{in}	
sub_area	指定RTC模块寄存器为安全保护模式
RTC_SPM_CTL_ALRM0SECP	alarm0安全保护
RTC_SPM_CTL_ALRM1SECP	alarm1安全保护
RTC_SPM_CTL_WUTSECP	唤醒定时器安全保护
RTC_SPM_CTL_TSSECP	时间戳安全保护
RTC_SPM_CTL_TAMPSECP	tamper事件安全保护(除了备份域寄存器)
RTC_SPM_CTL_CALCSECP	移位寄存器, 省时, 校准和参考时钟安全保护
RTC_SPM_CTL_INITSECP	RTC初始化安全保护
RTC_SPM_CTL_RTCSSECP	RTC全局安全保护
输出参数{out}	
-	-
返回值	
ErrStatus	ERROR 或 SUCCESS

例如:

```
/* disable RTC alarm 0 secure protection mode */
```

```
rtc_sec_pro_disable (RTC_SPM_CTL_ALRM0SECP);
```

### 函数 rtc\_bkp\_zone\_a\_sec\_pro\_set

函数rtc\_bkp\_zone\_a\_sec\_pro\_set描述见下表:

表 3-762. 函数 rtc\_bkp\_zone\_a\_sec\_pro\_set

函数名称	rtc_bkp_zone_a_sec_pro_set
函数原型	void rtc_bkp_zone_a_sec_pro_set(uint32_t zone_a_tail);
功能描述	配置RTC_BKP寄存器安全保护区域a
先决条件	-
被调用函数	-
输入参数{in}	
zone_a_tail	RTC_BKP寄存器安全保护区域a
RTC_BKP_PRO_ZONE_A_TAIL_NONE	没有RTC_BKP寄存器安全保护区域a
RTC_BKP_PRO_ZONE_A_TAIL_x	安全保护区域a, RTC_BKP0到RTC_BKPx只能通过APB在安全模式下读写

输出参数{out}	
-	-
返回值	
ErrStatus	ERROR 或 SUCCESS

例如：

```
/* set the RTC_BKP secure protection zonea tail value */
rtc_bkp_zoneb_sec_pro_set (RTC_BKP_PRO_ZONEA_TAIL_3);
```

### 函数 rtc\_bkp\_zoneb\_sec\_pro\_set

函数rtc\_bkp\_zoneb\_sec\_pro\_set描述见下表：

表 3-763. 函数 rtc\_bkp\_zoneb\_sec\_pro\_set

函数名称	rtc_bkp_zoneb_sec_pro_set
函数原型	void rtc_bkp_zoneb_sec_pro_set(uint32_t zoneb_tail);
功能描述	配置RTC_BKP寄存器安全保护区域b
先决条件	-
被调用函数	-
输入参数{in}	
zoneb_tail	RTC_BKP寄存器安全保护区域a
RTC_BKP_PRO_ZONEB_TAIL_NONE	没有RTC_BKP寄存器安全保护区域b
RTC_BKP_PRO_ZONEB_TAIL_x	安全保护区域b,RTC_BKP0到RTC_BKPx只能通过APB在安全模式下读写
输出参数{out}	
-	-
返回值	
ErrStatus	ERROR 或 SUCCESS

例如：

```
/* set the RTC_BKP secure protection zonea tail value */
rtc_bkp_zoneb_sec_pro_set (RTC_BKP_PRO_ZONEA_TAIL_3);
```

### 函数 rtc\_bkp\_zoneb\_sec\_pro\_check

函数rtc\_bkp\_zoneb\_sec\_pro\_check描述见下表：

表 3-764. 函数 rtc\_bkp\_zoneb\_sec\_pro\_check

函数名称	rtc_bkp_zoneb_sec_pro_ckeck
函数原型	ErrStatus rtc_bkp_zoneb_sec_pro_check(void);
功能描述	检查RTC_BKP寄存器安全保护区域b是否有效
先决条件	-
被调用函数	-

输入参数{in}	
-	-
输出参数{out}	
-	-
返回值	
ErrStatus	ERROR 或 SUCCESS

例如：

```
/* check the RTC_BKP secure protection zoneb is valid or not */
```

```
ErrStatus error_states = rtc_bkp_zoneb_sec_pro_check ();
```

## 3.23. SDIO

安全的数字输入/输出接口（SDIO）定义了SD卡、SD I/O卡、多媒体卡（MMC）和CE-ATA卡主机接口，提供AHB系统总线与SD存储卡、SD I/O卡、MMC和CE-ATA设备之间的数据传输。

章节[3.23.1](#)描述了SDIO的寄存器列表，章节[3.23.2](#)对SDIO库函数进行说明。

### 3.23.1. 外设寄存器说明

SDIO寄存器列表如下表所示：

**表 3-765. 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 3-766. SDIO 库函数**

库函数名称	库函数描述
sdio_deinit	复位SDIO
sdio_clock_config	配置SDIO时钟
sdio_hardware_clock_enable	使能硬件时钟控制
sdio_hardware_clock_disable	禁能硬件时钟控制
sdio_bus_mode_set	设置多种SDIO卡总线模式
sdio_power_state_set	设置SDIO电源状态
sdio_power_state_get	获取SDIO电源状态
sdio_clock_enable	使能SDIO_CLK时钟
sdio_clock_disable	禁能SDIO_CLK时钟
sdio_command_response_config	配置命令和响应
sdio_wait_type_set	设置命令状态机等待类型
sdio_csm_enable	使能命令状态机
sdio_csm_disable	禁能命令状态机
sdio_command_index_get	获取上一次响应的命令索引
sdio_response_get	获取上一次响应的接收命令
sdio_data_config	配置数据超时、数据长度和数据块大小
sdio_data_transfer_config	配置数据传输模式和方向
sdio_dsm_enable	使能数据传输的数据状态机
sdio_dsm_disable	禁能数据传输的数据状态机
sdio_data_write	在发送FIFO里写入数据（一个字）
sdio_data_read	在接收FIFO里读取数据（一个字）
sdio_data_counter_get	获取要传输到卡的剩余数据字节的数目
sdio_fifo_counter_get	从FIFO中获取要写入或读取的字数
sdio_dma_enable	使能SDIO的DMA请求
sdio_dma_disable	禁能SDIO的DMA请求
sdio_flag_get	获取SDIO的标志位状态
sdio_flag_clear	清除SDIO的标志位状态
sdio_interrupt_enable	使能SDIO中断
sdio_interrupt_disable	禁能SDIO中断
sdio_interrupt_flag_get	获取SDIO的中断标志位状态
sdio_interrupt_flag_clear	清除SDIO的中断标志位状态
sdio_readwait_enable	使能读等待模式（仅限SD I/O模式）
sdio_readwait_disable	禁能读等待模式（仅限SD I/O模式）
sdio_stop_readwait_enable	使能停止读等待过程的功能（仅限SD I/O模式）
sdio_stop_readwait_disable	禁能停止读等待过程的功能（仅限SD I/O模式）
sdio_readwait_type_set	设置读等待类型（仅限SD I/O模式）
sdio_operation_enable	使能SD I/O模式特定操作（仅限SD I/O模式）

库函数名称	库函数描述
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 3-767. 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 3-768. 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}	
clock_edge	SDIO_CLK时钟边沿选择
SDIO_SDIOLCKED GE_RISING	选择SDIOCLK的上升沿产生SDIO_CLK
SDIO_SDIOLCKED GE_FALLING	选择SDIOCLK的下降沿产生SDIO_CLK
输入参数{in}	
clock_bypass	旁路时钟使能
SDIO_CLOCKBYPASS SS_ENABLE	使能旁路时钟
SDIO_CLOCKBYPASS SS_DISABLE	失能旁路时钟
输入参数{in}	
clock_powersave	SDIO_CLK时钟动态开启/关闭以节省功耗
SDIO_CLOCKPWR SAVE_ENABLE	SDIO_CLK时钟在总线空闲时关闭
SDIO_CLOCKPWR SAVE_DISABLE	SDIO_CLK时钟总是开启
输入参数{in}	
clock_division	时钟分频，小于512
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* configure the SDIO clock */
```

```
sdio_clock_config(SDIO_SDIOLCKEDGE_RISING, SDIO_CLOCKBYPASS_DISABLE,  
SDIO_CLOCKPWRSAVE_DISABLE, SD_CLK_DIV_TRANS);
```

### 函数 sdio\_hardware\_clock\_enable

函数sdio\_hardware\_clock\_enable描述见下表：

表 3-769. 函数 sdio\_hardware\_clock\_enable

函数名称	sdio_hardware_clock_enable
函数原形	void sdio_hardware_clock_enable(void);
功能描述	使能硬件时钟控制
先决条件	-
被调用函数	-
输入参数{in}	
-	-

输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* enable hardware clock control */
sdio_hardware_clock_enable();
```

### 函数 **sdio\_hardware\_clock\_disable**

函数sdio\_hardware\_clock\_disable描述见下表：

**表 3-770. 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 3-771. 函数 sdio\_bus\_mode\_set**

函数名称	sdio_bus_mode_set
函数原形	void sdio_bus_mode_set(uint32_t bus_mode);
功能描述	设置多种SDIO卡总线模式
先决条件	-
被调用函数	-
输入参数{in}	
bus_mode	SDIO卡总线模式
SDIO_BUSMODE_1 BIT	1位SDIO卡总线模式



<i>SDIO_BUSMODE_4 BIT</i>	4位SDIO卡总线模式
<i>SDIO_BUSMODE_8 BIT</i>	8位SDIO卡总线模式
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* set SDIO bus mode */
sdio_bus_mode_set(SDIO_BUSMODE_1BIT);
```

### 函数 **sdio\_power\_state\_set**

函数sdio\_power\_state\_set描述见下表：

**表 3-772. 函数 sdio\_power\_state\_set**

函数名称	sdio_power_state_set
函数原形	void sdio_power_state_set(uint32_t power_state);
功能描述	设置SDIO电源状态
先决条件	-
被调用函数	-
输入参数{in}	
<b>power_state</b>	SDIO电源状态
<i>SDIO_POWER_ON</i>	SDIO上电
<i>SDIO_POWER_OFF</i>	SDIO断电
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* set SDIO power state */
sdio_power_state_set(SDIO_POWER_ON);
```

### 函数 **sdio\_power\_state\_get**

函数sdio\_power\_state\_get描述见下表：

**表 3-773. 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 3-774. 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 3-775. 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 3-776. 函数 sdio\_command\_response\_config**

函数名称	sdio_command_response_config
函数原形	void sdio_command_response_config(uint32_t cmd_index, uint32_t cmd_argument, uint32_t response_type);
功能描述	配置命令和响应
先决条件	-
被调用函数	-
输入参数{in}	
cmd_index	命令索引，请参阅相关规范
输入参数{in}	
cmd_argument	命令参数，请参阅相关规范
输入参数{in}	
response_type	命令响应类型
SDIO_RESPONSETYPE_NO	无响应
SDIO_RESPONSETYPE_SHORT	短响应
SDIO_RESPONSETYPE_LONG	长响应
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* CMD2(SD_CMD_ALL_SEND_CID) command response config*/
```

```
sdio_command_response_config(SD_CMD_ALL_SEND_CID, (uint32_t)0x0,
SDIO_RESPONSETYPE_LONG);
```

### 函数 **sdio\_wait\_type\_set**

函数sdio\_wait\_type\_set描述见下表：

**表 3-777. 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}	
wait_type	等待类型
SDIO_WAITTYPE_NO	不等待中断
SDIO_WAITTYPE_INTERRUPT	等待中断
SDIO_WAITTYPE_DATAEND	等待数据传输结束
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* set the command state machine wait type */
```

```
sdio_wait_type_set(SDIO_WAITTYPE_NO);
```

### 函数 **sdio\_csm\_enable**

函数sdio\_csm\_enable描述见下表：

**表 3-778. 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 3-779. 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 3-780. 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_commond_value;

sdio_commond_value = sdio_command_index_get();
```

### 函数 **sdio\_response\_get**

函数sdio\_response\_get描述见下表：

**表 3-781. 函数 sdio\_response\_get**

<b>函数名称</b>	sdio_response_get
<b>函数原形</b>	uint32_t sdio_response_get(uint32_t responsex);
<b>功能描述</b>	获取上一次响应的接收命令
<b>先决条件</b>	-
<b>被调用函数</b>	-
<b>输入参数{in}</b>	
<b>responsex</b>	SDIO响应
SDIO_RESPONSE0	卡响应 [31:0]/卡响应 [127:96]
SDIO_RESPONSE1	卡响应 [95:64]
SDIO_RESPONSE2	卡响应 [63:32]
SDIO_RESPONSE3	卡响应 [31:1]，加上位0
<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[4] = {0, 0, 0, 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 3-782. 函数 sdio\_data\_config**

<b>函数名称</b>	sdio_data_config
<b>函数原形</b>	void sdio_data_config(uint32_t data_timeout, uint32_t data_length, uint32_t data_blocksize);
<b>功能描述</b>	配置数据超时、数据长度和数据块大小
<b>先决条件</b>	-

被调用函数	-
输入参数{in}	
data_timeout	卡总线时钟周期中的数据超时周期
输入参数{in}	
data_length	要传输的数据字节数
输入参数{in}	
data_blocksize	块传输中数据块的大小
SDIO_DATABLOCK SIZE_1BYTE	块大小 = 1字节
SDIO_DATABLOCK SIZE_2BYTES	块大小 = 2字节
SDIO_DATABLOCK SIZE_4BYTES	块大小 = 4字节
SDIO_DATABLOCK SIZE_8BYTES	块大小 = 8字节
SDIO_DATABLOCK SIZE_16BYTES	块大小 = 16字节
SDIO_DATABLOCK SIZE_32BYTES	块大小 = 32字节
SDIO_DATABLOCK SIZE_64BYTES	块大小 = 64字节
SDIO_DATABLOCK SIZE_128BYTES	块大小 = 128字节
SDIO_DATABLOCK SIZE_256BYTES	块大小 = 256字节
SDIO_DATABLOCK SIZE_512BYTES	块大小 = 512字节
SDIO_DATABLOCK SIZE_1024BYTES	块大小 = 1024字节
SDIO_DATABLOCK SIZE_2048BYTES	块大小 = 2048字节
SDIO_DATABLOCK SIZE_4096BYTES	块大小 = 4096字节
SDIO_DATABLOCK SIZE_8192BYTES	块大小 = 8192字节
SDIO_DATABLOCK SIZE_16384BYTES	块大小 = 16384字节
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* configure SDIO data */
```

```
sdio_data_config(0, 0, SDIO_DATABLOCKSIZE_1BYTE);
```

### 函数 **sdio\_data\_transfer\_config**

函数sdio\_data\_transfer\_config描述见下表:

**表 3-783. 函数 sdio\_data\_transfer\_config**

函数名称	sdio_data_transfer_config
函数原形	void sdio_data_transfer_config(uint32_t transfer_mode, uint32_t transfer_direction);
功能描述	配置数据传输模式和方向
先决条件	-
被调用函数	-
输入参数{in}	
transfer_mode	数据传输模式
SDIO_TRANSMOD E_BLOCK	块传输模式
SDIO_TRANSMOD E_STREAM	流传输或SDIO多字节传输模式
输入参数{in}	
transfer_direction	数据传输方向
SDIO_TRANSDI RECTION_TO CARD	写数据到卡上
SDIO_TRANSDI RECTION_TO SDIO	从卡中读取数据
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* configure SDIO data transmisson */
```

```
sdio_data_transfer_config(SDIO_TRANSDIRECTION_TOSDIO,  
SDIO_TRANSMODE_BLOCK);
```

### 函数 **sdio\_dsm\_enable**

函数sdio\_dsm\_enable描述见下表:

**表 3-784. 函数 sdio\_dsm\_enable**

函数名称	sdio_dsm_enable
函数原形	void sdio_dsm_enable(void);
功能描述	使能数据传输的数据状态机



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

例如：

```
/* enable the DSM(data state machine) */
```

```
sdio_dsm_enable();
```

### 函数 sdio\_dsm\_disable

函数sdio\_dsm\_disable描述见下表：

**表 3-785. 函数 sdio\_dsm\_disable**

函数名称	sdio_dsm_disable
函数原形	void sdio_dsm_disable(void);
功能描述	禁能数据传输的数据状态机
先决条件	-
被调用函数	-
输入参数{in}	
-	-
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* disable the DSM(data state machine) */
```

```
sdio_dsm_disable();
```

### 函数 sdio\_data\_write

函数sdio\_data\_write描述见下表：

**表 3-786. 函数 sdio\_data\_write**

函数名称	sdio_data_write
函数原形	void sdio_data_write(uint32_t data);
功能描述	在发送FIFO里写入数据（一个字）
先决条件	-
被调用函数	-

输入参数{in}	
<b>data</b>	往卡里写入32位数据
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* write data(one word) to the transmit FIFO */
```

```
sdio_data_write(0x0000 0001);
```

### 函数 sdio\_data\_read

函数sdio\_data\_read描述见下表：

表 3-787. 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 3-788. 函数 sdio\_data\_counter\_get

函数名称	sdio_data_counter_get
函数原形	uint32_t sdio_data_counter_get(void);
功能描述	获取要传输到卡的剩余数据字节的数目
先决条件	-
被调用函数	-
输入参数{in}	
-	-

输出参数{out}	
-	-
返回值	
uint32_t	要传输的剩余数据字节数

例如：

```
/* get the number of remaining data bytes to be transferred to card */
```

```
uint32_t sdio_data_value;
```

```
sdio_data_value = sdio_data_counter_get();
```

### 函数 sdio\_fifo\_counter\_get

函数sdio\_fifo\_counter\_get描述见下表：

表 3-789. 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 3-790. 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 3-791. 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 3-792. 函数 sdio\_flag\_get**

函数名称	sdio_flag_get
函数原形	FlagStatus sdio_flag_get(uint32_t flag);
功能描述	获取SDIO的标志位状态
先决条件	-
被调用函数	-
输入参数{in}	
flag	SDIO标志位状态
SDIO_FLAG_CCRC	命令响应已接收（CRC检测失败）

<i>ERR</i>	
<i>SDIO_FLAG_DTCRCERR</i>	数据块已发送/已接收（CRC检测失败）
<i>SDIO_FLAG_CMDTMOU</i>	命令响应超时
<i>SDIO_FLAG_DTTMOU</i>	数据超时
<i>SDIO_FLAG_TXUR</i> <i>E</i>	发送FIFO下溢错误发生
<i>SDIO_FLAG_RXOR</i> <i>E</i>	接收FIFO上溢错误发生
<i>SDIO_FLAG_CMDRECV</i>	命令响应已接收（CRC检测通过）
<i>SDIO_FLAG_CMDS</i> <i>END</i>	命令已发送（不需响应）
<i>SDIO_FLAG_DTEND</i> <i>D</i>	数据结束（数据计数器，SDIO_DATACNT为零）
<i>SDIO_FLAG_STBITE</i>	总线上起始位错误
<i>SDIO_FLAG_DTBLKEND</i>	数据块已发送/已接收（CRC检测通过）
<i>SDIO_FLAG_CMDRUN</i>	正在传输命令
<i>SDIO_FLAG_TXRUN</i> <i>N</i>	正在传输数据
<i>SDIO_FLAG_RXRUN</i> <i>N</i>	正在接收数据
<i>SDIO_FLAG_TFH</i>	发送FIFO半空：至少还有8个字可被写入到FIFO中
<i>SDIO_FLAG_RFH</i>	接收FIFO半满：FIFO中至少还有8个字可被读取
<i>SDIO_FLAG_TFF</i>	发送FIFO为满
<i>SDIO_FLAG_RFF</i>	接收FIFO为满
<i>SDIO_FLAG_TFE</i>	发送FIFO为空
<i>SDIO_FLAG_RFE</i>	接收FIFO为空
<i>SDIO_FLAG_TXDTVAL</i>	发送FIFO中的数据有效
<i>SDIO_FLAG_RXDTVAL</i>	接收FIFO中的数据有效
<i>SDIO_FLAG_SDIOINT</i>	SD I/O中断已接收
<i>SDIO_FLAG_ATAEND</i>	CE-ATA命令完成信号已接收（仅用于CMD61）
输出参数{out}	
-	-

返回值	
FlagStatus	SET 或 RESET

例如：

```
/* get the flags state of SDIO */
```

```
FlagStatus flag_value;
```

```
flag_value = sdio_flag_get(SDIO_FLAG_RFH);
```

### 函数 sdio\_flag\_clear

函数sdio\_flag\_clear描述见下表：

表 3-793. 函数 sdio\_flag\_clear

函数名称	sdio_flag_clear
函数原形	void sdio_flag_clear(uint32_t flag);
功能描述	清除SDIO的标志位状态
先决条件	-
被调用函数	-
输入参数{in}	
flag	SDIO标志位状态
SDIO_FLAG_CCRC ERR	命令响应已接收（CRC检测失败）
SDIO_FLAG_DTCR CERR	数据块已发送/已接收（CRC检测失败）
SDIO_FLAG_CMDT MOUT	命令响应超时
SDIO_FLAG_DTTM OUT	数据超时
SDIO_FLAG_TXUR E	发送FIFO下溢错误发生
SDIO_FLAG_RXOR E	接收FIFO上溢错误发生
SDIO_FLAG_CMDR ECV	命令响应已接收（CRC检测通过）
SDIO_FLAG_CMDS END	命令已发送（不需响应）
SDIO_FLAG_DTEN D	数据结束（数据计数器，SDIO_DATACNT为零）
SDIO_FLAG_STBIT E	总线上起始位错误
SDIO_FLAG_DTBL KEND	数据块已发送/已接收（CRC检测通过）
SDIO_FLAG_SDIOI	SD I/O中断已接收

NT	
SDIO_FLAG_ATAE ND	CE-ATA命令完成信号已接收（仅用于CMD61）
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* clear the pending flags of SDIO */
```

```
sdio_flag_clear(SDIO_FLAG_DTCRCERR);
```

### 函数 sdio\_interrupt\_enable

函数sdio\_interrupt\_enable描述见下表：

表 3-794. 函数 sdio\_interrupt\_enable

函数名称	sdio_interrupt_enable
函数原形	void sdio_interrupt_enable(uint32_t int_flag);
功能描述	使能SDIO中断
先决条件	-
被调用函数	-
输入参数{in}	
int_flag	SDIO中断标志位状态
SDIO_INT_CCRCE RR	命令响应CRC错误中断
SDIO_INT_DTCRC 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	正在传输命令中断

<i>N</i>	
<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_RXDTV</i> <i>AL</i>	接收FIFO中的数据有效中断
<i>SDIO_INT_SDIOINT</i> <i>T</i>	SD I/O中断已接收中断
<i>SDIO_INT_ATAEND</i> <i>D</i>	CE-ATA命令完成信号已接收中断
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* enable the SDIO corresponding interrupts */
```

```
sdio_interrupt_enable(SDIO_INT_CCRCE | 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 3-795. 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}	
<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



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
SDIO_INT_TXRUN	正在传输数据中断
SDIO_INT_RXRUN	正在接收数据中断
SDIO_INT_TFH	发送FIFO半满中断
SDIO_INT_RFH	接收FIFO半满中断
SDIO_INT_TFF	发送FIFO满中断
SDIO_INT_RFF	接收FIFO满中断
SDIO_INT_TFE	发送FIFO空中断
SDIO_INT_RFE	接收FIFO空中断
SDIO_INT_TXDTVA L	发送FIFO中的数据有效中断
SDIO_INT_RXDTV AL	接收FIFO中的数据有效中断
SDIO_INT_SDIOIN T	SD I/O中断已接收中断
SDIO_INT_ATAEN D	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_DTCCRERR);
```

## 函数 `sdio_interrupt_flag_get`

函数 `sdio_interrupt_flag_get` 描述见下表：

**表 3-796. 函数 `sdio_interrupt_flag_get`**

函数名称	<code>sdio_interrupt_flag_get</code>
函数原形	<code>FlagStatus sdio_interrupt_flag_get(uint32_t int_flag);</code>
功能描述	获取SDIO的中断标志位状态
先决条件	-
被调用函数	-
输入参数{in}	
<code>int_flag</code>	SDIO中断标志位状态
<code>SDIO_INT_FLAG_C_CRCERR</code>	命令响应CRC错误中断
<code>SDIO_INT_FLAG_D_TCRCERR</code>	数据CRC错误中断
<code>SDIO_INT_FLAG_C_MDTMOUT</code>	命令响应超时中断
<code>SDIO_INT_FLAG_D_TTMOUT</code>	数据超时中断
<code>SDIO_INT_FLAG_TXURE</code>	发送FIFO下溢错误中断
<code>SDIO_INT_FLAG_RXORE</code>	接收FIFO上溢错误中断
<code>SDIO_INT_FLAG_C_MDRECV</code>	命令响应已接收中断
<code>SDIO_INT_FLAG_C_MDSEND</code>	命令已发送中断
<code>SDIO_INT_FLAG_D_TEND</code>	数据结束中断
<code>SDIO_INT_FLAG_S_TBITE</code>	起始位错误中断
<code>SDIO_INT_FLAG_D_TBLKEND</code>	数据块已发送/已接收中断
<code>SDIO_INT_FLAG_C_MDRUN</code>	正在传输命令中断
<code>SDIO_INT_FLAG_TXRUN</code>	正在传输数据中断
<code>SDIO_INT_FLAG_RXRUN</code>	正在接收数据中断
<code>SDIO_INT_FLAG_TXFH</code>	发送FIFO半满中断
<code>SDIO_INT_FLAG_RXFH</code>	接收FIFO半满中断

SDIO_INT_FLAG_T FF	发送FIFO满中断
SDIO_INT_FLAG_R FF	接收FIFO满中断
SDIO_INT_FLAG_T FE	发送FIFO空中断
SDIO_INT_FLAG_R FE	接收FIFO空中断
SDIO_INT_FLAG_T XDTVAL	发送FIFO中的数据有效中断
SDIO_INT_FLAG_R XDTVAL	接收FIFO中的数据有效中断
SDIO_INT_FLAG_S DIOINT	SD I/O中断已接收中断
SDIO_INT_FLAG_A TAEND	CE-ATA命令完成信号已接收中断
输出参数{out}	
-	-
返回值	
FlagStatus	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 3-797. 函数 sdio\_interrupt\_flag\_clear

函数名称	sdio_interrupt_flag_clear
函数原形	void sdio_interrupt_flag_clear(uint32_t int_flag);
功能描述	清除SDIO的中断标志位状态
先决条件	-
被调用函数	-
输入参数{in}	
int_flag	SDIO中断标志位状态
SDIO_INT_FLAG_C CRCERR	命令响应CRC错误中断
SDIO_INT_FLAG_D TCRCERR	数据CRC错误中断
SDIO_INT_FLAG_C	命令响应超时中断

MDTMOUT	
SDIO_INT_FLAG_D TTMOUT	数据超时中断
SDIO_INT_FLAG_T XURE	发送FIFO下溢错误中断
SDIO_INT_FLAG_R XORE	接收FIFO上溢错误中断
SDIO_INT_FLAG_C MDRECV	命令响应已接收中断
SDIO_INT_FLAG_C MDSEND	命令已发送中断
SDIO_INT_FLAG_D TEND	数据结束中断
SDIO_INT_FLAG_S TBITE	起始位错误中断
SDIO_INT_FLAG_D TBLKEND	数据块已发送/已接收中断
SDIO_INT_FLAG_S DIOINT	SD I/O中断已接收中断
SDIO_INT_FLAG_A TAEND	CE-ATA命令完成信号已接收中断
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* clear the interrupt pending flags of SDIO */
```

```
sdio_interrupt_flag_clear(SDIO_INT_FLAG_DTEND);
```

### 函数 sdio\_readwait\_enable

函数sdio\_readwait\_enable描述见下表：

表 3-798. 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 3-799. 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 3-800. 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 3-801. 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 3-802. 函数 sdio\_readwait\_type\_set**

函数名称	sdio_readwait_type_set
函数原形	void sdio_readwait_type_set(uint32_t readwait_type);
功能描述	设置读等待类型（仅限SD I/O模式）
先决条件	-
被调用函数	-
输入参数{in}	
readwait_type	SD I/O 读等待模式
SDIO_READWAITTYPE_CLK	通过停止SDIO_CLK控制读等待
SDIO_READWAITTYPE_DAT2	使用SDIO_DAT[2] 控制读等待
输出参数{out}	

-	-
返回值	
-	-

例如：

```
/* set the read wait type(SD I/O only) */
sdio_readwait_type_set(uint32_t readwait_type);
```

### 函数 sdio\_operation\_enable

函数sdio\_operation\_enable描述见下表：

表 3-803. 函数 sdio\_operation\_enable

函数名称	sdio_operation_enable
函数原形	void sdio_operation_enable(void);
功能描述	使能SD I/O模式特定操作（仅限SD I/O模式）
先决条件	-
被调用函数	-
输入参数{in}	
-	-
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* enable the SD I/O mode specific operation(SD I/O only) */
sdio_operation_enable();
```

### 函数 sdio\_operation\_disable

函数sdio\_operation\_disable描述见下表：

表 3-804. 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 3-805. 函数 sdio\_suspend\_enable**

函数名称	sdio_suspend_enable
函数原形	void sdio_suspend_enable(void);
功能描述	使能SD I/O暂停模式（仅限SD I/O模式）
先决条件	-
被调用函数	-
输入参数{in}	
-	-
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* enable the SD I/O suspend operation(SD I/O only) */
sdio_suspend_enable();
```

### 函数 **sdio\_suspend\_disable**

函数sdio\_suspend\_disable描述见下表：

**表 3-806. 函数 sdio\_suspend\_disable**

函数名称	sdio_suspend_disable
函数原形	void sdio_suspend_disable(void);
功能描述	禁能SD I/O暂停模式（仅限SD I/O模式）
先决条件	-
被调用函数	-
输入参数{in}	
-	-
输出参数{out}	
-	-
返回值	
-	-



例如：

```
/* disable the SD I/O suspend operation(SD I/O only) */
```

```
sdio_suspend_disable();
```

### 函数 **sdio\_ceata\_command\_enable**

函数sdio\_ceata\_command\_enable描述见下表：

**表 3-807. 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 3-808. 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 3-809. 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 3-810. 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 3-811. 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 3-812. 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 3-813. 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

### 3.24.2. 外设库函数说明

SPI/I2S库函数列表如下表所示：

**表 3-814. 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_ckin_psc_config	配置I2S输入时钟分频
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

库函数名称	库函数描述
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
i2s_full_duplex_mode_config	配置I2S全双工模式
spi_i2s_format_error_clear	清除SPI/I2S帧格式错误标志
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_crc_error_clear	清除SPIx CRC错误标志状态
spi_ti_mode_enable	使能SPI TI模式
spi_ti_mode_disable	禁能SPI TI模式
spi_quad_enable	使能四线SPI模式
spi_quad_disable	禁能四线SPI模式
spi_quad_write_enable	使能四线SPI写
spi_quad_read_enable	使能四线SPI读
spi_quad_io23_output_enable	使能SPI_IO2和SPI_IO3输出
spi_quad_io23_output_disable	禁能SPI_IO2和SPI_IO3输出
spi_i2s_flag_get	获取外设SPIx/I2Sx标志状态
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

### 结构体 spi\_parameter\_struct

表 3-815. 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_BDRCEIVE, SPI_TRANSMODE_BDTRANSMIT)
frame_size	数据帧格式配置 (SPI_FRAME_SIZE_16BIT, SPI_FRAME_SIZE_8BIT)
nss	配置NSS由软件或硬件控制 (SPI_NSS_SOFT, SPI_NSS_HARD)
endian	大端或小端模式配置 (SPI_ENDIAN_MSB, SPI_ENDIAN_LSB)

成员名称	功能描述
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 3-816. 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
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 3-817. 函数 spi\_struct\_para\_init

函数名称	spi_struct_para_init
函数原形	void spi_struct_para_init(spi_parameter_struct* spi_struct);
功能描述	将SPI结构体中所有参数初始化为默认值
先决条件	-
被调用函数	-
输入参数{in}	
*spi_struct	一个已经定义的spi_parameter_struct结构体变量地址
输出参数{out}	
-	-
返回值	

-	-
---	---

例如:

```
/* initialize the parameters of SPI */

spi_parameter_struct spi_init_struct;

spi_struct_para_init(&spi_init_struct);
```

### 函数 spi\_init

函数spi\_init描述见下表:

表 3-818. 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
输入参数{in}	
spi_struct	初始化结构体, 结构体成员参考 <a href="#">表3-815. 结构体spi_parameter_struct</a>
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* initialize SPI0 */

spi_parameter_struct spi_init_struct;

spi_init_struct.trans_mode      = SPI_TRANSMODE_BDTRANSMIT;
spi_init_struct.device_mode     = SPI_MASTER;
spi_init_struct.frame_size      = SPI_FRAME_SIZE_8BIT;
spi_init_struct.clock_polarity_phase = SPI_CLK_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 3-819. 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
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 3-820. 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
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 3-821. 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
输入参数{in}	
i2s_mode	I2S运行模式
I2S_MODE_SLAVE_TX	I2S从机发送模式
I2S_MODE_SLAVE_RX	I2S从机接收模式
I2S_MODE_MASTERTX	I2S主机发送模式
I2S_MODE_MASTERRX	I2S主机接收模式
输入参数{in}	
i2s_standard	I2S标准选择
I2S_STD_PHILLIPS	I2S飞利浦标准
I2S_STD_MSB	I2S MSB对齐标准
I2S_STD_LSB	I2S LSB对齐标准
I2S_STD_PCMSHORT	I2S PCM短帧标准
I2S_STD_PCMLONG	I2S PCM长帧标准
输入参数{in}	
i2s_ckpl	I2S空闲状态时钟极性
I2S_CKPL_LOW	I2S_CK空闲状态为低电平
I2S_CKPL_HIGH	I2S_CK空闲状态为高电平
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* initialize I2S1 */
```

```
i2s_init(SPI1, I2S_MODE_MASTERTX, I2S_STD_PHILLIPS, I2S_CKPL_LOW);
```

### 函数 i2s\_psc\_config

函数i2s\_psc\_config描述见下表:

表 3-822. 函数 i2s\_psc\_config

函数名称	i2s_psc_config
函数原形	void i2s_psc_config(uint32_t spi_periph, uint32_t i2s_audiosample, uint32_t i2s_frameformat, uint32_t i2s_mckout);
功能描述	配置I2Sx预分频器
先决条件	-
被调用函数	rcu_clock_freq_get
输入参数{in}	
spi_periph	外设I2Sx
SPIx	x=1
输入参数{in}	
i2s_audiosample	I2S音频采样频率
I2S_AUDIOSAMPL E_8K	音频采样频率为8KHz
I2S_AUDIOSAMPL E_11K	音频采样频率为11KHz
I2S_AUDIOSAMPL E_16K	音频采样频率为16KHz
I2S_AUDIOSAMPL E_22K	音频采样频率为22KHz
I2S_AUDIOSAMPL E_32K	音频采样频率为32KHz
I2S_AUDIOSAMPL E_44K	音频采样频率为44KHz
I2S_AUDIOSAMPL E_48K	音频采样频率为48KHz
I2S_AUDIOSAMPL E_96K	音频采样频率为96KHz
I2S_AUDIOSAMPL E_192K	音频采样频率为192KHz
输入参数{in}	
i2s_frameformat	I2S数据长度和通道长度
I2S_FRAMEFORMA T_DT16B_CH16B	I2S数据长度为16位，通道长度为16位
I2S_FRAMEFORMA T_DT16B_CH32B	I2S数据长度为16位，通道长度为32位

<i>I2S_FRAMEFORMA T_DT24B_CH32B</i>	I2S数据长度为24位，通道长度为32位
<i>I2S_FRAMEFORMA T_DT32B_CH32B</i>	I2S数据长度为32位，通道长度为32位
输入参数{in}	
<b>i2s_mckout</b>	I2S_MCK输出使能
<i>I2S_MCKOUT_ENA BLE</i>	I2S_MCK输出使能
<i>I2S_MCKOUT_DIS ABLE</i>	I2S_MCK输出禁止
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\_ckin\_psc\_config

函数i2s\_ckin\_psc\_config描述见下表：

表 3-823. 函数 i2s\_ckin\_psc\_config

函数名称	i2s_ckin_psc_config
函数原形	void i2s_ckin_psc_config(uint32_t input_clock, uint32_t i2s_audiosample, uint32_t i2s_frameformat, uint32_t i2s_mckout);
功能描述	配置I2S输入时钟分频
先决条件	-
被调用函数	-
输入参数{in}	
<b>input_clock</b>	I2S输入时钟
输入参数{in}	
<b>i2s_audiosample</b>	I2S音频采样频率
<i>I2S_AUDIOSAMPL E_8K</i>	音频采样频率为8KHz
<i>I2S_AUDIOSAMPL E_11K</i>	音频采样频率为11KHz
<i>I2S_AUDIOSAMPL E_16K</i>	音频采样频率为16KHz
<i>I2S_AUDIOSAMPL E_22K</i>	音频采样频率为22KHz

<i>I2S_AUDIOSAMPL E_32K</i>	音频采样频率为32KHz
<i>I2S_AUDIOSAMPL E_44K</i>	音频采样频率为44KHz
<i>I2S_AUDIOSAMPL E_48K</i>	音频采样频率为48KHz
<i>I2S_AUDIOSAMPL E_96K</i>	音频采样频率为96KHz
<i>I2S_AUDIOSAMPL E_192K</i>	音频采样频率为192KHz
输入参数{in}	
<b>i2s_frameformat</b>	I2S数据长度和通道长度
<i>I2S_FRAMEFORMA T_DT16B_CH16B</i>	I2S数据长度为16位，通道长度为16位
<i>I2S_FRAMEFORMA T_DT16B_CH32B</i>	I2S数据长度为16位，通道长度为32位
<i>I2S_FRAMEFORMA T_DT24B_CH32B</i>	I2S数据长度为24位，通道长度为32位
<i>I2S_FRAMEFORMA T_DT32B_CH32B</i>	I2S数据长度为32位，通道长度为32位
输入参数{in}	
<b>i2s_mckout</b>	2S_MCK输出使能
<i>I2S_MCKOUT_ENA BLE</i>	I2S_MCK输出使能
<i>I2S_MCKOUT_DIS ABLE</i>	I2S_MCK输出禁止
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* configure I2S1 input clock prescaler */
```

```
#define I2S1_INPUT_CK 50000000
```

```
i2s_psc_config(I2S1_INPUT_CK, I2S_AUDIOSAMPLE_8K, I2S_FRAMEFORMAT_DT16B  
_CH16B, I2S_MCKOUT_DISABLE);
```

### 函数 i2s\_enable

函数i2s\_enable描述见下表：

表 3-824. 函数 i2s\_enable

函数名称	i2s_enable
------	------------

函数原形	void i2s_enable(uint32_t spi_periph);
功能描述	使能外设I2Sx
先决条件	-
被调用函数	-
输入参数{in}	
spi_periph	外设SPI/I2Sx
SPI1	I2S1外设
I2S1_ADD	I2S1_ADD外设
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 3-825. 函数 i2s\_disable

函数名称	i2s_disable
函数原形	void i2s_disable(uint32_t spi_periph);
功能描述	禁能外设I2Sx
先决条件	-
被调用函数	-
输入参数{in}	
spi_periph	外设SPI/I2Sx
SPI1	I2S1外设
I2S1_ADD	I2S1_ADD外设
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 3-826. 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
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 3-827. 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
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 3-828. 函数 `spi_nss_internal_high`

函数名称	<code>spi_nss_internal_high</code>
函数原形	<code>void spi_nss_internal_high(uint32_t spi_periph);</code>
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}	
<code>spi_periph</code>	外设SPIx
<code>SPIx</code>	x=0,1
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* SPI0 NSS pin is pulled high level in software mode */
```

```
spi_nss_internal_high(SPI0);
```

### 函数 `spi_nss_internal_low`

函数`spi_nss_internal_low`描述见下表：

表 3-829. 函数 `spi_nss_internal_low`

函数名称	<code>spi_nss_internal_low</code>
函数原形	<code>void spi_nss_internal_low(uint32_t spi_periph);</code>
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}	
<code>spi_periph</code>	外设SPIx
<code>SPIx</code>	x=0,1
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* SPI0 NSS pin is pulled low level in software mode */
```

```
spi_nss_internal_low(SPI0);
```

### 函数 `spi_dma_enable`

函数`spi_dma_enable`描述见下表：

表 3-830. 函数 `spi_dma_enable`

函数名称	<code>spi_dma_enable</code>
函数原形	<code>void spi_dma_enable(uint32_t spi_periph, uint8_t dma);</code>
功能描述	使能外设SPIx的DMA功能
先决条件	-
被调用函数	-
输入参数{in}	
<code>spi_periph</code>	外设SPIx
<code>SPIx</code>	x=0,1
输入参数{in}	
<code>dma</code>	SPI DMA模式
<code>SPI_DMA_TRANSMIT</code>	SPI发送DMA使能
<code>SPI_DMA_RECEIVE</code>	SPI接收DMA使能
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* enable SPI0 transmit data DMA function */
```

```
spi_dma_enable(SPI0, SPI_DMA_TRANSMIT);
```

### 函数 `spi_dma_disable`

函数`spi_dma_disable`描述见下表：

表 3-831. 函数 `spi_dma_disable`

函数名称	<code>spi_dma_disable</code>
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
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 3-832. 函数 spi\_i2s\_data\_frame\_format\_config

函数名称	spi_i2s_data_frame_format_config
函数原形	void spi_i2s_data_frame_format_config(uint32_t spi_periph, uint16_t frame_format);
功能描述	配置外设SPIx/I2Sx数据帧格式
先决条件	-
被调用函数	-
输入参数{in}	
spi_periph	外设SPIx
SPIx	x=0,1
输入参数{in}	
frame_format	SPI帧大小
SPI_FRAME_SIZE_16BIT	SPI 16位数据帧格式
SPI_FRAME_SIZE_8BIT	SPI 8位数据帧格式
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* configure SPI1/I2S1 data frame format size is 16 bits */
```

```
spi_i2s_data_frame_format_config(SPI1, SPI_FRAME_SIZE_16BIT);
```

### 函数 spi\_i2s\_data\_transmit

函数spi\_i2s\_data\_transmit描述见下表：

表 3-833. 函数 spi\_i2s\_data\_transmit

函数名称	spi_i2s_data_transmit
------	-----------------------

函数原形	void spi_i2s_data_transmit(uint32_t spi_periph, uint16_t data);
功能描述	SPI发送数据
先决条件	-
被调用函数	-
输入参数{in}	
spi_periph	外设SPIx
SPIx	x=0,1
输入参数{in}	
data	16位数据
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* SPI0 transmit data */
uint16_t spi0_send_array[] = {0x5050, 0xA0A0};
spi_i2s_data_transmit(SPI0, spi0_send_array[0]);
```

### 函数 spi\_i2s\_data\_receive

函数spi\_i2s\_data\_receive描述见下表：

表 3-834. 函数 spi\_i2s\_data\_receive

函数名称	spi_i2s_data_receive
函数原形	uint16_t spi_i2s_data_receive(uint32_t spi_periph);
功能描述	SPI接收数据
先决条件	-
被调用函数	-
输入参数{in}	
spi_periph	外设SPIx
SPIx	x=0,1
输出参数{out}	
-	-
返回值	
-	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 3-835. 函数 `spi_bidirectional_transfer_config`

函数名称	<code>spi_bidirectional_transfer_config</code>
函数原形	<code>void spi_bidirectional_transfer_config(uint32_t spi_periph, uint32_t transfer_direction);</code>
功能描述	配置外设SPIx的数据传输方向
先决条件	-
被调用函数	-
输入参数{in}	
<code>spi_periph</code>	外设SPIx
<code>SPIx</code>	x=0,1
输入参数{in}	
<code>transfer_direction</code>	SPI双向传输输出使能
<code>SPI_BIDIRECTIONAL_TRANSMIT</code>	SPI工作在只发送模式
<code>SPI_BIDIRECTIONAL_RECEIVE</code>	SPI工作在只接收模式
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* SPI0 works in transmit-only mode */
```

```
spi_bidirectional_transfer_config(SPI0, SPI_BIDIRECTIONAL_TRANSMIT);
```

函数 `i2s_full_duplex_mode_config`

函数 `i2s_full_duplex_mode_config` 描述见下表：

表 3-836. 函数 `i2s_full_duplex_mode_config`

函数名称	<code>i2s_full_duplex_mode_config</code>
函数原形	<code>void i2s_full_duplex_mode_config(uint32_t i2s_add_periph, uint32_t i2s_mode, uint32_t i2s_standard, uint32_t i2s_ckpl, uint32_t i2s_frameformat);</code>
功能描述	配置I2S全双工模式
先决条件	-
被调用函数	-
输入参数{in}	
<code>i2s_add_periph</code>	外设I2Sx_ADD
<code>I2Sx_ADD</code>	x=1
输入参数{in}	

<b>i2s_mode</b>	I2S运行模式
<i>I2S_MODE_SLAVE_TX</i>	I2S从机发送模式
<i>I2S_MODE_SLAVE_RX</i>	I2S从机接收模式
<i>I2S_MODE_MASTERTX</i>	I2S主机发送模式
<i>I2S_MODE_MASTERRX</i>	I2S主机接收模式
输入参数{in}	
<b>i2s_standard</b>	I2S标准选择
<i>I2S_STD_PHILLIPS</i>	I2S 飞利浦标准
<i>I2S_STD_MSB</i>	I2S MSB对齐标准
<i>I2S_STD_LSB</i>	I2S LSB对齐标准
<i>I2S_STD_PCMSHORT</i>	I2S PCM短帧标准
<i>I2S_STD_PCMLONG</i>	I2S PCM长帧标准
输入参数{in}	
<b>i2s_ckpl</b>	I2S空闲状态时钟极性
<i>I2S_CKPL_LOW</i>	I2S_CK空闲状态为低电平
<i>I2S_CKPL_HIGH</i>	I2S_CK空闲状态为高电平
输入参数{in}	
<b>i2s_frameformat</b>	I2S数据长度和通道长度
<i>I2S_FRAMEFORMAT_DT16B_CH16B</i>	I2S数据长度为16位，通道长度为16位
<i>I2S_FRAMEFORMAT_DT16B_CH32B</i>	I2S数据长度为16位，通道长度为32位
<i>I2S_FRAMEFORMAT_DT24B_CH32B</i>	I2S数据长度为24位，通道长度为32位
<i>I2S_FRAMEFORMAT_DT32B_CH32B</i>	I2S数据长度为32位，通道长度为32位
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* configure I2S1_ADD */
```

```
i2s_full_duplex_mode_config(I2S1_ADD,I2S_MODE_MASTERTX,I2S_STD_PHILLIPS,
I2S_CKPL_HIGH, I2S_FRAMEFORMAT_DT16B_CH16B);
```

函数 **spi\_i2s\_format\_error\_clear**

函数spi\_i2s\_format\_error\_clear描述见下表：

表 3-837. 函数 **spi\_i2s\_format\_error\_clear**

函数名称	spi_i2s_format_error_clear
函数原形	void spi_i2s_format_error_clear(uint32_t spi_periph, uint32_t flag);
功能描述	清除SPI/I2S帧格式错误标志
先决条件	-
被调用函数	-
输入参数{in}	
spi_periph	外设SPIx
SPIx	x=0,1
输入参数{in}	
flag	SPI/I2S帧格式错误标志
SPI_FLAG_FERR	SPI TI模式下帧格式错误
I2S_FLAG_FERR	I2S帧格式错误
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* clear SPI0 TI mode format error flag */
spi_i2s_format_error_clear(SPI0, SPI_FLAG_FERR);
```

函数 **spi\_crc\_polynomial\_set**

函数spi\_crc\_polynomial\_set描述见下表：

表 3-838. 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
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_t CRC_VALUE = 0x5050;

spi_crc_polynomial_set(SPI0,CRC_VALUE);
```

### 函数 spi\_crc\_polynomial\_get

函数spi\_crc\_polynomial\_get描述见下表：

表 3-839. 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
输出参数{out}	
-	-
返回值	
uint16_t	16位CRC多项式值（0-0xFFFF）

例如：

```
/* get SPI0 CRC polynomial */

uint16_t crc_val;

crc_val = spi_crc_polynomial_get(SPI0);
```

### 函数 spi\_crc\_on

函数spi\_crc\_on描述见下表：

表 3-840. 函数 spi\_crc\_on

函数名称	spi_crc_on
函数原形	void spi_crc_on(uint32_t spi_periph);
功能描述	打开外设SPIx的CRC功能
先决条件	-
被调用函数	-
输入参数{in}	
spi_periph	外设SPIx
SPIx	x=0,1

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 3-841. 函数 spi\_crc\_off

函数名称	spi_crc_off
函数原形	void spi_crc_off(uint32_t spi_periph);
功能描述	关闭外设SPIx的CRC功能
先决条件	-
被调用函数	-
输入参数{in}	
spi_periph	外设SPIx
SPIx	x=0,1
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 3-842. 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

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 3-843. 函数 spi\_crc\_get

函数名称	spi_crc_get
函数原形	uint16_t spi_crc_get(uint32_t spi_periph, uint8_t crc);
功能描述	外设SPIx获取CRC值
先决条件	-
被调用函数	-
输入参数{in}	
spi_periph	外设SPIx
SPIx	x=0,1
输入参数{in}	
crc	SPI crc值
SPI_CRC_TX	获取发送CRC寄存器值
SPI_CRC_RX	获取接收CRC寄存器值
输出参数{out}	
-	-
返回值	
uint16_t	16位CRC值（0-0xFFFF）

例如：

```
/* get SPI0 CRC send value */
```

```
uint16_t crc_val;
```

```
crc_val = spi_crc_get(SPI0, SPI_CRC_TX);
```

### 函数 spi\_crc\_error\_clear

函数spi\_crc\_error\_clear描述见下表：

表 3-844. 函数 spi\_crc\_error\_clear

函数名称	spi_crc_error_clear
函数原形	void spi_crc_error_clear(uint32_t spi_periph);



功能描述	清除SPIx CRC错误标志状态
先决条件	-
被调用函数	-
输入参数{in}	
spi_periph	外设SPIx
SPIx	x=0,1
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* clear SPI0 CRC error flag status */
spi_crc_error_clear(SPI0);
```

### 函数 spi\_ti\_mode\_enable

函数spi\_ti\_mode\_enable描述见下表：

表 3-845. 函数 spi\_ti\_mode\_enable

函数名称	spi_ti_mode_enable
函数原形	void spi_ti_mode_enable(uint32_t spi_periph);
功能描述	使能SPI TI模式
先决条件	-
被调用函数	-
输入参数{in}	
spi_periph	外设SPIx
SPIx	x=0,1
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* enable SPI0 TI mode */
spi_ti_mode_enable(SPI0);
```

### 函数 spi\_ti\_mode\_disable

函数spi\_ti\_mode\_disable描述见下表：

表 3-846. 函数 spi\_ti\_mode\_disable

函数名称	spi_ti_mode_disable
函数原形	void spi_ti_mode_disable(uint32_t spi_periph);

功能描述	禁能SPI TI模式
先决条件	-
被调用函数	-
输入参数{in}	
spi_periph	外设SPIx
SPIx	x=0,1
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* disable SPI0 TI mode */
spi_ti_mode_disable(SPI0);
```

### 函数 spi\_quad\_enable

函数spi\_quad\_enable描述见下表：

表 3-847. 函数 spi\_quad\_enable

函数名称	spi_quad_enable
函数原形	void spi_quad_enable(uint32_t spi_periph);
功能描述	使能四线SPI模式
先决条件	-
被调用函数	-
输入参数{in}	
spi_periph	外设SPIx
SPIx	x=0
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* enable SPI0 quad wire mode */
spi_quad_enable(SPI0);
```

### 函数 spi\_quad\_disable

函数spi\_quad\_disable描述见下表：

表 3-848. 函数 spi\_quad\_disable

函数名称	spi_quad_disable
函数原形	spi_quad_disable(uint32_t spi_periph);

功能描述	禁能四线SPI模式
先决条件	-
被调用函数	-
输入参数{in}	
spi_periph	外设SPIx
SPIx	x=0
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* disable SPI0 quad wire mode */
```

```
spi_quad_disable(SPI0);
```

### 函数 spi\_quad\_write\_enable

函数spi\_quad\_write\_enable描述见下表：

表 3-849. 函数 spi\_quad\_write\_enable

函数名称	spi_quad_write_enable
函数原形	void spi_quad_write_enable(uint32_t spi_periph);
功能描述	使能四线SPI写
先决条件	-
被调用函数	-
输入参数{in}	
spi_periph	外设SPIx
SPIx	x=0
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* enable SPI0 quad wire write */
```

```
spi_quad_write_enable(SPI0);
```

### 函数 spi\_quad\_read\_enable

函数spi\_quad\_read\_enable描述见下表：

表 3-850. 函数 spi\_quad\_read\_enable

函数名称	spi_quad_read_enable
函数原形	void spi_quad_read_enable(uint32_t spi_periph);

功能描述	使能四线SPI读
先决条件	-
被调用函数	-
输入参数{in}	
spi_periph	外设SPIx
SPIx	x=0
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* enable SPI0 quad wire read */
spi_quad_read_enable(SPI0);
```

### 函数 spi\_quad\_io23\_output\_enable

函数spi\_quad\_io23\_output\_enable描述见下表：

表 3-851. 函数 spi\_quad\_io23\_output\_enable

函数名称	spi_quad_io23_output_enable
函数原形	void spi_quad_io23_output_enable(uint32_t spi_periph);
功能描述	使能SPI_IO2和SPI_IO3输出
先决条件	-
被调用函数	-
输入参数{in}	
spi_periph	外设SPIx
SPIx	x=0
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* enable SPI0 SPI_IO2 and SPI_IO3 pin output */
spi_quad_io23_output_enable(SPI0);
```

### 函数 spi\_quad\_io23\_output\_disable

函数spi\_quad\_io23\_output\_disable描述见下表：

表 3-852. 函数 spi\_quad\_io23\_output\_disable

函数名称	spi_quad_io23_output_disable
函数原形	void spi_quad_io23_output_disable(uint32_t spi_periph);

功能描述	禁能SPI_IO2和SPI_IO3输出
先决条件	-
被调用函数	-
输入参数{in}	
spi_periph	外设SPIx
SPIx	x=0
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* disable SPI0 SPI_IO2 and SPI_IO3 pin output */
```

```
spi_quad_io23_output_disable(SPI0);
```

### 函数 spi\_i2s\_flag\_get

函数spi\_i2s\_flag\_get描述见下表：

表 3-853. 函数 spi\_i2s\_flag\_get

函数名称	spi_i2s_flag_get
函数原形	FlagStatus spi_i2s_flag_get(uint32_t spi_periph, uint32_t flag);
功能描述	获取外设SPIx/I2Sx标志状态
先决条件	-
被调用函数	-
输入参数{in}	
spi_periph	外设SPIx
SPIx	x=0,1
输入参数{in}	
flag	SPI/I2S标志状态
SPI_FLAG_TBE	发送缓冲区空标志
SPI_FLAG_RBNE	接收缓冲区非空标志
SPI_FLAG_TRANS	通信进行中标志
SPI_I2S_INT_FLAG_RXORERR	接收过载错误标志
SPI_FLAG_CONFERR	配置错误标志
SPI_FLAG_CRCERR	CRC错误标志
SPI_FLAG_FERR	帧错误标志
I2S_FLAG_TBE	发送缓冲区空标志
I2S_FLAG_RBNE	接收缓冲区非空标志
I2S_FLAG_TRANS	通信进行中标志

<i>I2S_FLAG_RXORE</i> <i>RR</i>	接收过载错误标志
<i>I2S_FLAG_TXURE</i> <i>RR</i>	发送欠载错误标志
<i>I2S_FLAG_CH</i>	通道标志
<i>I2S_FLAG_FERR</i>	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\_i2s\_interrupt\_enable

函数spi\_i2s\_interrupt\_enable描述见下表:

表 3-854. 函数 spi\_i2s\_interrupt\_enable

函数名称	spi_i2s_interrupt_enable
函数原形	void spi_i2s_interrupt_enable(uint32_t spi_periph, uint8_t interrupt);
功能描述	使能外设SPIx/I2Sx中断
先决条件	-
被调用函数	-
输入参数{in}	
<b>spi_periph</b>	外设SPIx
<i>SPIx</i>	x=0,1
输入参数{in}	
<b>interrupt</b>	SPI/I2S中断
<i>SPI_I2S_INT_TBE</i>	发送缓冲区空中断使能
<i>SPI_I2S_INT_RBNE</i>	接收缓冲区非空中断使能
<i>SPI_I2S_INT_ERR</i>	错误中断使能
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* enable SPI0 transmit buffer empty interrupt */
spi_i2s_interrupt_enable(SPI0, SPI_I2S_INT_TBE);
```

**函数 spi\_i2s\_interrupt\_disable**

函数spi\_i2s\_interrupt\_disable描述见下表:

**表 3-855. 函数 spi\_i2s\_interrupt\_disable**

函数名称	spi_i2s_interrupt_disable
函数原形	void spi_i2s_interrupt_disable(uint32_t spi_periph, uint8_t interrupt);
功能描述	禁能外设SPIx/I2Sx中断
先决条件	-
被调用函数	-
输入参数{in}	
spi_periph	外设SPIx
SPIx	x=0,1
输入参数{in}	
interrupt	SPI/I2S中断
SPI_I2S_INT_TBE	发送缓冲区空中断使能
SPI_I2S_INT_RBNE	接收缓冲区非空中断使能
SPI_I2S_INT_ERR	错误中断使能
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* disable SPI0 transmit buffer empty interrupt */
```

```
spi_i2s_interrupt_disable(SPI0, SPI_I2S_INT_TBE);
```

**函数 spi\_i2s\_interrupt\_flag\_get**

函数spi\_i2s\_interrupt\_flag\_get描述见下表:

**表 3-856. 函数 spi\_i2s\_interrupt\_flag\_get**

函数名称	spi_i2s_interrupt_flag_get
函数原形	FlagStatus spi_i2s_interrupt_flag_get(uint32_t spi_periph, uint8_t interrupt);
功能描述	获取外设SPIx/I2Sx中断状态
先决条件	-
被调用函数	-
输入参数{in}	
spi_periph	外设SPIx
SPIx	x=0,1
输入参数{in}	
interrupt	SPI/I2S中断状态
SPI_I2S_INT_FLAG_TBE	发送缓冲区空中断

<code>SPI_I2S_INT_FLAG_RBNE</code>	接收缓冲区非空中断
<code>SPI_I2S_INT_FLAG_RXOERR</code>	接收过载错误中断
<code>SPI_INT_FLAG_CONFERR</code>	配置错误中断
<code>SPI_INT_FLAG_CRCERR</code>	CRC错误中断
<code>I2S_INT_FLAG_TXURERR</code>	发送欠载错误中断
<code>SPI_I2S_INT_FLAG_FERR</code>	帧错误中断
输出参数{out}	
-	-
返回值	
<b>FlagStatus</b>	SET 或 RESET

例如：

```

/* get SPI0 transmit buffer empty interrupt status */
if(RESET != spi_i2s_interrupt_flag_get(SPI0, SPI_I2S_INT_FLAG_TBE)){
    while(RESET == spi_i2s_flag_get(SPI0, SPI_FLAG_TBE));
    spi_i2s_data_transmit(SPI0, spi0_send_array[send_n++]);
}

```

## 3.25. SQPI

SQPI接口是一个用于串行、双线、四线接口存储设备的控制器。章节[3.25.1](#)描述了SQPI的寄存器列表，章节[3.25.2](#)对SQPI库函数进行说明。

### 3.25.1. 外设寄存器说明

SQPI寄存器列表如下表所示：

**表 3-857. SQPI 寄存器**

寄存器名称	寄存器描述
SQPI_INIT	SQPI初始化寄存器
SQPI_RCMD	SQPI读命令寄存器
SQPI_WCMD	SQPI写命令寄存器
SQPI_IDL	SQPI ID低位寄存器
SQPI_IDH	SQPI ID 高位寄存器



## 3.25.2. 外设库函数说明

SQPI库函数列表如下表所示：

表 3-858. SQPI 库函数

库函数名称	库函数描述
sqpi_deinit	复位外设SQPI
sqpi_struct_para_init	将SQPI结构体中所有参数初始化为默认值
sqpi_init	初始化外设SQPI
sqpi_read_id_command	SQPI发送读ID命令
sqpi_special_command	SQPI发送特殊命令
sqpi_read_command_config	SQPI读命令配置
sqpi_write_command_config	SQPI写命令配置
sqpi_low_id_receive	SQPI获取ID低位数据
sqpi_high_id_receive	SQPI获取ID高位数据

## 结构体 sqpi\_parameter\_struct

表 3-859. 结构体 sqpi\_parameter\_struct

成员名称	功能描述
polarity	SQPI采样极性 (SQPI_SAMPLE_POLARITY_RISING, SQPI_SAMPLE_POLARITY_FALLING)
id_length	外部存储器ID长度 (QSPI_ID_LENGTH_n_BITS (n=8,16,32,64))
addr_bit	地址阶段的地址位数 (0x00 - 0x1F)
clk_div	SQPI时钟分频 (0x01 - 0x3F)
cmd_bit	命令阶段的命令位数 (QSPI_CMDBIT_n_BITS (n=4,8,16))

## 函数 sqpi\_deinit

函数sqpi\_deinit描述见下表：

表 3-860. 函数 sqpi\_deinit

函数名称	sqpi_deinit
函数原形	void sqpi_deinit(void);
功能描述	复位外设SQPI
先决条件	-
被调用函数	rcu_periph_reset_enable / rcu_periph_reset_disable
输入参数{in}	
-	-
输出参数{out}	

-	-
返回值	
-	-

例如：

```
/* reset SQPI */
```

```
sqpi_deinit();
```

### 函数 sqpi\_struct\_para\_init

函数sqpi\_struct\_para\_init描述见下表：

**表 3-861. 函数 sqpi\_struct\_para\_init**

函数名称	sqpi_struct_para_init
函数原形	void sqpi_struct_para_init(sqpi_parameter_struct* sqpi_struct);
功能描述	将SQPI结构体中所有参数初始化为默认值
先决条件	-
被调用函数	-
输入参数{in}	
*sqpi_struct	一个已经定义的sqpi_parameter_struct结构体变量地址
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* initialize the parameters of SQPI */
```

```
sqpi_parameter_struct sqpi_init_struct;
```

```
sqpi_struct_para_init(&sqpi_init_struct);
```

### 函数 sqpi\_init

函数sqpi\_init描述见下表：

**表 3-862. 函数 sqpi\_init**

函数名称	sqpi_init
函数原形	void sqpi_init(sqpi_parameter_struct* sqpi_struct);
功能描述	初始化外设SQPI
先决条件	-
被调用函数	-
输入参数{in}	
sqi_struct	初始化结构体，结构体成员参考 <a href="#">表3-859. 结构体sqpi_parameter_struct</a>
输出参数{out}	

-	-
返回值	
-	-

例如:

```
/* initialize SQPI */
```

```
sqpi_parameter_struct sqpi_init_struct;
```

```
sqpi_struct->polarity = SQPI_SAMPLE_POLARITY_RISING;
```

```
sqpi_struct->id_length = QSPI_ID_LENGTH_32_BITS;
```

```
sqpi_struct->addr_bit = 24U;
```

```
sqpi_struct->clk_div = 2U;
```

```
sqpi_struct->cmd_bit = QSPI_CMDBIT_8_BITS;
```

```
sqpi_init(&sqpi_init_struct);
```

### 函数 sqpi\_read\_id\_command

函数sqpi\_read\_id\_command描述见下表:

表 3-863. 函数 sqpi\_read\_id\_command

函数名称	sqpi_read_id_command
函数原形	void sqpi_read_id_command (void);
功能描述	发送读ID命令
先决条件	-
被调用函数	-
输入参数{in}	
-	-
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* send SQPI read ID command */
```

```
sqpi_read_id_command ();
```

### 函数 sqpi\_special\_command

函数sqpi\_special\_command描述见下表:

表 3-864. 函数 sqpi\_special\_command

函数名称	sqpi_special_command
------	----------------------

函数原形	void sqpi_special_command(void);
功能描述	SQPI发送特殊命令
先决条件	-
被调用函数	-
输入参数{in}	
-	-
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* send SQPI special command */
```

```
sqpi_special_command ();
```

### 函数 sqpi\_read\_command\_config

函数sqpi\_read\_command\_config描述见下表：

表 3-865. 函数 sqpi\_read\_command\_config

函数名称	sqpi_read_command_config
函数原形	void sqpi_read_command_config(uint32_t rmode, uint32_t rwaitcycle, uint32_t rcmd);
功能描述	配置SQPI读命令
先决条件	-
被调用函数	-
输入参数{in}	
rmode	SQPI读命令模式
QSPI_MODE_SSQ	SSQ模式
QSPI_MODE_SSS	SSS模式
QSPI_MODE_SQQ	SQQ模式
QSPI_MODE_QQQ	QQQ模式
QSPI_MODE_SSD	SSD模式
QSPI_MODE_SDD	SDD模式
输入参数{in}	
rwaitcycle	SQPI read wait cycle (0x00 – 0x1F)
输入参数{in}	
rcmd	SQPI read command(0x00 – 0xFF)
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* send SQPI special command */
```

```
sqpi_special_command ();
```

### 函数 sqpi\_write\_command\_config

函数sqpi\_write\_command\_config描述见下表:

表 3-866. 函数 sqpi\_write\_command\_config

函数名称	sqpi_write_command_config
函数原形	void sqpi_write_command_config(uint32_t wmode, uint32_t wwaitcycle, uint32_t wcmd);
功能描述	配置SQPI写命令
先决条件	-
被调用函数	-
输入参数{in}	
wmode	SQPI写命令模式
QSPI_MODE_SSQ	SSQ模式
QSPI_MODE_SSS	SSS模式
QSPI_MODE_SQQ	SQQ模式
QSPI_MODE_QQQ	QQQ模式
QSPI_MODE_SSD	SSD模式
QSPI_MODE_SDD	SDD模式
输入参数{in}	
wwaitcycle	SQPI write wait cycle (0x00 – 0x1F)
输入参数{in}	
wcmd	SQPI write command(0x00 – 0xFF)
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* configure SQPI write command */
```

```
sqpi_write_command_config(QSPI_MODE_SSS,0x00,0x9f);
```

### 函数 sqpi\_low\_id\_receive

函数sqpi\_low\_id\_receive描述见下表:

表 3-867. 函数 sqpi\_low\_id\_receive

函数名称	sqpi_low_id_receive
函数原形	uint32_t sqpi_low_id_receive(void);
功能描述	获取低位ID值
先决条件	-

被调用函数	-
输入参数{in}	
-	-
输出参数{out}	
-	-
返回值	
uint32_t	32位ID值(0-0xFFFF)

例如:

```
/* get SQPI low ID value */
```

```
uint32_t val;
```

```
val = sqpi_low_id_receive ();
```

### 函数 sqpi\_high\_id\_receive

函数sqpi\_high\_id\_receive描述见下表:

表 3-868. 函数 sqpi\_high\_id\_receive

函数名称	sqpi_high_id_receive
函数原形	uint32_t sqpi_high_id_receive(void);
功能描述	获取高位ID值
先决条件	-
被调用函数	-
输入参数{in}	
-	-
输出参数{out}	
-	-
返回值	
uint32_t	32位ID值(0-0xFFFF)

例如:

```
/* get SQPI high ID value */
```

```
uint32_t val;
```

```
val = sqpi_high_id_receive ();
```

## 3.26. SYSCFG

章节 [错误!未找到引用源。](#) 描述了SYSCFG的寄存器列表, 章节 [错误!未找到引用源。](#) 对SYSCFG库函数进行说明。

### 3.26.1. 外设寄存器说明

SYSCFG寄存器列表如下所示：

**表 3-869. SYSCFG 寄存器**

寄存器名称	寄存器描述
SYSCFG_CFG0	SYSCFG配置寄存器0
SYSCFG_EXTISS0	EXTI源选择寄存器0
SYSCFG_EXTISS1	EXTI源选择寄存器1
SYSCFG_EXTISS2	EXTI源选择寄存器2
SYSCFG_EXTISS3	EXTI源选择寄存器3
SYSCFG_CPCTL	I/O补偿控制寄存器
SYSCFG_SECCFG	SYSCFG安全配置寄存器
SYSCFG_FPUINTMSK	FPU中断屏蔽寄存器
SYSCFG_CNSLOCK	SYSCFG CPU非安全锁定寄存器
SYSCFG_CSLOCK	SYSCFG CPU安全锁定寄存器
SYSCFG_CFG1	SYSCFG配置寄存器1
SYSCFG_SCS	SYSCFG SRAM1控制和状态寄存器
SYSCFG_SKEY	SYSCFG SRAM1解锁寄存器
SYSCFG_SWP0	SYSCFG SRAM1写保护寄存器0
SYSCFG_SWP1	SYSCFG SRAM1写保护寄存器1
SYSCFG_GSSACMD	SYSCFG GSSA命令寄存器

### 3.26.2. 外设库函数说明

库函数列表如下所示：

**表 3-870.SYSCFG 库函数**

库函数名称	库函数说明
syscfg_deinit	复位SYSCFG寄存器
syscfg_exti_line_config	配置GPIO EXTI线
compensation_pwdn_mode_enable	使能补偿单元
compensation_pwdn_mode_disable	失能补偿单元
syscfg_clock_access_security_config	配置SYSCFG时钟控制安全性
classb_access_security_config	配置ClassB安全性
sram1_access_security_config	配置SRAM1安全性
fpu_access_security_config	配置FPU安全性
vtor_ns_write_disable	VTOR_NS寄存器写失能
mpu_ns_write_disable	非安全MPU寄存器写失能
vtors_aircr_write_disable	VTOR_S和AIRCR寄存器中PRIS和BFHFNMISS位域写失能

库函数名称	库函数说明
mpu_s_write_disable	安全MPU寄存器写失能
sau_write_disable	SAU寄存器写失能
syscfg_lock_config	选择与TIMER0/15/16的break输入端连接参数
gssacmd_write_data	定义都GSSA执行的命令
sram1_erase	使能SRAM1擦除
sram1_unlock	解锁SYSCFG_SCS寄存器中SRAM1ERS位的写保护
sram1_lock	使能SYSCFG_SCS寄存器中SRAM1ERS位的写保护
sram1_write_protect_0_31	使能SRAM1 0-31页写保护
sram1_write_protect_32_63	使能SRAM1 32-63页写保护
compensation_ready_flag_get	获取补偿单元准备就绪标志
sram1_bsy_flag_get	获取SRAM1擦除忙标志
fpu_interrupt_enable	使能浮点单元中断
fpu_interrupt_disable	失能浮点单元中断

### 函数 syscfg\_deinit

函数syscfg\_deinit描述见下表:

表 3-871. 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寄存器
先决条件	-
被调用函数	rcu_periph_reset_enable / rcu_periph_reset_disable
输入参数{in}	
-	-
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* reset SYSCFG */
```

```
syscfg_deinit();
```

### 函数 syscfg\_exti\_line\_config

函数syscfg\_exti\_line\_config描述见下表

表 3-872. 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GPIO EXTI线



先决条件	-
被调用函数	-
输入参数{in}	
exti_port	指定EXTI GPIO端口
EXTI_SOURCE_GPIOx	EXTI GPIO端口(x = A, B, C)
输入参数{in}	
exti_pin	指定EXTI线
EXTI_SOURCE_PINx	EXTI GPIO引脚(GPIOA x = 0..15, GPIOB x = 0..15, GPIOC x = 0..8)
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* configure the GPIO pin as EXTI Line */
```

```
syscfg_exti_line_config(EXTI_SOURCE_GPIOA, EXTI_SOURCE_PIN1);
```

### 函数 compensation\_pwdn\_mode\_enable

函数compensation\_pwdn\_mode\_enable描述见下表

表 3-873. 函数 compensation\_pwdn\_mode\_enable

函数名称	compensation_pwdn_mode_enable
函数原型	void compensation_pwdn_mode_enable(void);
功能描述	使能补偿单元
先决条件	-
被调用函数	-
输入参数{in}	
-	-
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* enable the compensation cell */
```

```
compensation_pwdn_mode_enable();
```

### 函数 compensation\_pwdn\_mode\_disable

函数compensation\_pwdn\_mode\_disable描述见下表

表 3-874. 函数 compensation\_pwdn\_mode\_disable

函数名称	compensation_pwdn_mode_disable
------	--------------------------------

函数原型	void compensation_pwdn_mode_disable(void)
功能描述	失能补偿单元
先决条件	-
被调用函数	-
输入参数{in}	
-	-
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* disable the compensation cell */
compensation_pwdn_mode_disable();
```

### 函数 syscfg\_clock\_access\_security\_config

函数syscfg\_clock\_access\_security\_config描述见下表

表 3-875. 函数 syscfg\_clock\_access\_security\_config

函数名称	syscfg_clock_access_security_config
函数原型	void syscfg_clock_access_security_config(uint32_t access_mode);
功能描述	配置SYSCFG时钟控制安全
先决条件	-
被调用函数	-
输入参数{in}	
<b>access_mode</b>	安全访问或安全和非安全访问
SYSCFG_SECURE_ACCESS	RCU寄存器中的SYSCFG配置时钟只能通过安全访问写入
SYSCFG_NONSECURE_ACCESS	RCU寄存器中的SYSCFG配置时钟可以通过安全和非安全访问写入
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* configure SYSCFG clock control security */
syscfg_clock_access_security_config(SYSCFG_SECURE_ACCESS);
```

### 函数 classb\_access\_security\_config

函数classb\_access\_security\_config描述见下表：

表 3-876. 函数 `classb_access_security_config`

函数名称	<code>classb_access_security_config</code>
函数原型	<code>void classb_access_security_config(uint32_t access_mode);</code>
功能描述	配置ClassB安全性
先决条件	-
被调用函数	-
输入参数{in}	
<b>access_mode</b>	安全访问或安全和非安全访问
<code>CLASSB_SECURE_ACCESS</code>	SYSCFG_CFG1只能通过安全访问写入
<code>CLASSB_SECURE_NONSECURE_ACCESS</code>	SYSCFG_CFG1可以通过安全和非安全访问写入
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* configure ClassB security */
```

```
classb_access_security_config(CLASSB_SECURE_ACCESS);
```

### 函数 `sram1_access_security_config`

函数 `sram1_access_security_config` 描述见下表：

表 3-877. 函数 `sram1_access_security_config`

函数名称	<code>sram1_access_security_config</code>
函数原型	<code>void sram1_access_security_config(uint32_t access_mode)</code>
功能描述	配置SRAM1安全性
先决条件	-
被调用函数	-
输入参数{in}	
<b>access_mode</b>	安全访问或安全和非安全访问
<code>SRAM1_SECURE_ACCESS</code>	SYSCFG_SKEY, SYSCFG_SCS和SYSCFG_SWPx只能通过安全访问写入
<code>SRAM1_SECURE_NONSECURE_ACCESS</code>	SYSCFG_SKEY, SYSCFG_SCS和SYSCFG_SWPx可以通过安全访问和非安全访问写入
输出参数{out}	
-	-
返回值	
-	-

例如

```
/* configure SRAM1 security */
```

```
sram1_access_security_config(SRAM1_SECURE_ACCESS);
```

### 函数 fpu\_access\_security\_config

函数fpu\_access\_security\_config描述见下表：

表 3-878.函数 fpu\_access\_security\_config

函数名称	fpu_access_security_config
函数原型	void fpu_access_security_config(uint32_t access_mode);
功能描述	配置FPU安全性
先决条件	-
被调用函数	-
输入参数{in}	
access_mode	安全访问或安全和非安全访问
FPU_SECURE_ACCESS	SYSCFG_FPUINTMSK只能通过安全访问写入
FPU_SECURE_NONSECURE_ACCESS	SYSCFG_FPUINTMSK可以通过安全访问和非安全访问写入
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* configure FPU security */
```

```
fpu_access_security_config(FPU_SECURE_ACCESS);
```

### 函数 vtor\_ns\_write\_disable

函数vtor\_ns\_write\_disable描述见下表

表 3-879. 函数 vtor\_ns\_write\_disable

函数名称	vtor_ns_write_disable
函数原型	void vtor_ns_write_disable(void);
功能描述	disable VTOR_NS register write
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 VTOR_NS register write */  
  
vtor_ns_write_disable();
```

### 函数 mpu\_ns\_write\_disable

函数mpu\_ns\_write\_disable描述见下表

表 3-880. 函数 mpu\_ns\_write\_disable

函数名称	mpu_ns_write_disable
函数原型	void mpu_ns_write_disable(void)
功能描述	非安全MPU寄存器写失能
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 Non-secure MPU registers write */  
  
mpu_ns_write_disable();
```

### 函数 vtors\_aicr\_write\_disable

函数vtors\_aicr\_write\_disable描述见下表

表 3-881. 函数 vtors\_aicr\_write\_disable

函数名称	vtors_aicr_write_disable
函数原型	void vtors_aicr_write_disable(void)
功能描述	VTOR_S和AICR寄存器中PRIS和BFHFNMINs位域写失能
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 VTOR_S register PRIS and BFHFNMINS bits in the AIRCR register write */
```

```
vtors_aicr_write_disable();
```

### 函数 vtors\_aicr\_write\_disable

函数vtors\_aicr\_write\_disable描述见下表：

**表 3-882. 函数 vtors\_aicr\_write\_disable**

函数名称	vtors_aicr_write_disable
函数原型	void mpu_s_write_disable(void);
功能描述	安全MPU寄存器写失能
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 secure MPU registers writes */
```

```
mpu_s_write_disable();
```

### 函数 sau\_write\_disable

函数sau\_write\_disable描述见下表：

**表 3-883. 函数 sau\_write\_disable**

函数名称	sau_write_disable
函数原型	void sau_write_disable(void);
功能描述	SAU寄存器写失能
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 SAU registers write */
```

sau\_write\_disable();

### 函数 syscfg\_lock\_config

函数syscfg\_lock\_config描述见下表：

表 3-884. 函数 syscfg\_lock\_config

函数名称	syscfg_lock_config
函数原型	void syscfg_lock_config(uint32_t syscfg_lock);
功能描述	选择与TIMER0/15/16的break输入端连接参数
先决条件	-
被调用函数	-
输入参数{in}	
syscfg_lock	指定连接的参数
SYSCFG_LOCK_LOCKUP	LOCKUP输出与TIMER0/15/16的break输入端连接
SYSCFG_LOCK_LVD	LVD中断与TIMER0/TIMER15/TIMER16的break输入端连接
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* connect TIMER0/15/16 break input to the selected parameter */
```

```
syscfg_lock_config(SYSCFG_LOCK_LOCKUP);
```

### 函数 gssacmd\_write\_data

函数gssacmd\_write\_data描述见下表：

表 3-885. 函数 gssacmd\_write\_data

函数名称	gssacmd_write_data
函数原型	void gssacmd_write_data(uint32_t data);
功能描述	定义GSSA执行的命令
先决条件	-
被调用函数	-
输入参数{in}	
data	写入到GSSA的数据(0x00-0xFF)
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* write data to GSSA */
```

```
gssacmd_write_data(0x0C);
```

### 函数 sram1\_erase

函数sram1\_erase描述见下表:

表 3-886. 函数 sram1\_erase

函数名称	sram1_erase
函数原型	ErrStatus sram1_erase(void);
功能描述	使能SRAM1擦除
先决条件	-
被调用函数	-
输入参数{in}	
-	-
输出参数{out}	
-	-
返回值	
ErrStatus	SUCCESS或ERROR

例如:

```
/* enable sram1 erase */
```

```
ErrStatus state;
```

```
state = sram1_erase();
```

### 函数 sram1\_unlock

函数sram1\_unlock描述见下表:

表 3-887. 函数 sram1\_unlock

函数名称	sram1_unlock
函数原型	void sram1_unlock(void);
功能描述	解锁SYSCFG_SCS寄存器中SRAM1ERS位的写保护
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 write protection of the SRAM1ERS bit in the SYSCFG_SCS register */
```

```
sram1_unlock();
```

### 函数 sram1\_lock

函数sram1\_lock描述见下表:

表 3-888. 函数 sram1\_lock

函数名称	sram1_lock
函数原型	void sram1_lock(void);
功能描述	使能SYSCFG_SCS寄存器中SRAM1ERS位的写保护
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 write protection of the SRAM1ERS bit in the SYSCFG_SCS register */
```

```
sram1_lock();
```

### 函数 sram1\_write\_protect\_0\_31

函数sram1\_write\_protect\_0\_31描述见下表:

表 3-889. 函数 sram1\_write\_protect\_0\_31

函数名称	sram1_write_protect_0_31
函数原型	void sram1_write_protect_0_31(uint32_t page);
功能描述	使能SRAM1 0-31页写保护
先决条件	-
被调用函数	-
输入参数{in}	
page	指定被保护的SRAM1页数
SRAM1_PAGEx_WRIT E_PROTECT	使能SRAM1被保护的页x (x=0,1,...31)
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* SRAM1 write protect page1 */
```

```
sram1_write_protect_0_31(SRAM1_PAGE1_WRITE_PROTECT);
```

### 函数 sram1\_write\_protect\_32\_63

函数sram1\_write\_protect\_32\_63描述见下表：

表 3-890. 函数 sram1\_write\_protect\_32\_63

函数名称	sram1_write_protect_32_63
函数原型	void sram1_write_protect_32_63(uint32_t page);
功能描述	使能SRAM1 32-63页写保护
先决条件	-
被调用函数	-
输入参数{in}	
page	指定被保护的SRAM1页数
SRAM1_PAGEx_WRITE_PROTECT	使能SRAM1被保护的页x (x=32,33,...63)
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* SRAM1 write protect page32 */
```

```
sram1_write_protect_32_63(SRAM1_PAGE32_WRITE_PROTECT);
```

### 函数 compensation\_ready\_flag\_get

函数compensation\_ready\_flag\_get描述见下表：

表 3-891. 函数 compensation\_ready\_flag\_get

函数名称	compensation_ready_flag_get
函数原型	FlagStatus compensation_ready_flag_get(void);
功能描述	获取补偿单元准备就绪标志
先决条件	-
被调用函数	-
输入参数{in}	
-	-
输出参数{out}	
-	-
返回值	
FlagStatus	RESET或SET

例如：

```
/* get the compensation ready flag */
```

```
compensation_ready_flag_get();
```

### 函数 sram1\_bsy\_flag\_get

函数sram1\_bsy\_flag\_get描述见下表：

表 3-892. 函数 sram1\_bsy\_flag\_get

函数名称	sram1_bsy_flag_get
函数原型	FlagStatus sram1_bsy_flag_get(void);
功能描述	获取SRAM1擦除忙标志
先决条件	-
被调用函数	-
输入参数{in}	
输出参数{out}	
-	-
返回值	
FlagStatus	SET或RESET

例如：

```
/* get SRAM1 erase busy flag */
```

```
FlagStatus state;
```

```
State = sram1_bsy_flag_get();
```

### 函数 fpu\_interrupt\_enable

函数fpu\_interrupt\_enable描述见下表：

表 3-893. 函数 fpu\_interrupt\_enable

函数名称	fpu_interrupt_enable
函数原型	void fpu_interrupt_enable(uint32_t interrupt);
功能描述	使能浮点单元中断
先决条件	-
被调用函数	-
输入参数{in}	
interrupt	中断类型
INVALID_OPERATION_INT	无效操作中断
DEVIDE_BY_ZERO_INT	除数为零中断
UNDERFLOW_INT	下溢中断
OVERFLOW_INT	溢出中断

<i>INPUT_ABNORMAL_INT</i>	输入异常中断
<i>INEXACT_INT</i>	不精确中断（复位时禁止中断）
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* enable floating point unit interrupt */
fpu_interrupt_enable(INVALID_OPERATION_INT);
```

### 函数 fpu\_interrupt\_disable

函数fpu\_interrupt\_disable描述见下表：

表 3-894. 函数 fpu\_interrupt\_disable

函数名称	fpu_interrupt_disable
函数原型	void fpu_interrupt_disable(uint32_t interrupt);
功能描述	失能浮点单元中断
先决条件	-
被调用函数	-
输入中断{in}	
<b>interrupt</b>	中断类型
<i>INVALID_OPERATION_INT</i>	无效操作中断
<i>DEVIDE_BY_ZERO_INT</i>	除数为零中断
<i>UNDERFLOW_INT</i>	下溢中断
<i>OVERFLOW_INT</i>	溢出中断
<i>INPUT_ABNORMAL_INT</i>	输入异常中断
<i>INEXACT_INT</i>	不精确中断（复位时禁止中断）
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* disable floating point unit interrupt */
fpu_interrupt_disable(INVALID_OPERATION_INT);
```

### 3.27. TIMER

定时器含有可编程的一个16位无符号计数器，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)，通用定时器L0(TIMERx, x=1, 2, 3, 4)，通用定时器L4(TIMERx, x=15, 16)，基本定时器(TIMERx, x=5)，不同类型的定时器具体功能有所差别。章节[3.27.1](#)描述了TIMER的寄存器列表，章节[3.27.2](#)对TIMER库函数进行说明。

#### 3.27.1. 外设寄存器说明

TIMER寄存器列表如下表所示：

表 3-895. TIMER 寄存器

寄存器名称	寄存器描述
TIMER_CTL0	控制寄存器0
TIMER_CTL1	控制寄存器1
TIMER_SMCFG	从模式配置寄存器
TIMER_DMAINTEN	DMA和中断使能寄存器
TIMER_INTF	中断标志寄存器
TIMER_SWEVG	软件事件产生寄存器
TIMER_CHCTL0	通道控制寄存器0
TIMER_CHCTL1	通道控制寄存器1
TIMER_CHCTL2	通道控制寄存器2
TIMER_CNT	计数器寄存器
TIMER_PSC	预分频寄存器
TIMER_CAR	计数器自动重载寄存器
TIMER_CREP	重复计数寄存器
TIMER_CH0CV	通道0捕获/比较寄存器
TIMER_CH1CV	通道1捕获/比较寄存器
TIMER_CH2CV	通道2捕获/比较寄存器
TIMER_CH3CV	通道3捕获/比较寄存器
TIMER_CCHP	互补通道保护寄存器
TIMER_DMACFG	DMA配置寄存器
TIMER_DMATB	DMA发送缓冲区寄存器
TIMER_CFG	配置寄存器

#### 3.27.2. 外设库函数说明

TIMER库函数列表如下表所示：

表 3-896. TIMER 库函数

库函数名称	库函数描述
timer_deinit	复位外设TIMERx
timer_struct_para_init	将TIMER初始化结构体中所有参数初始化为默认值

库函数名称	库函数描述
timer_init	初始化外设TIMERx
timer_enable	使能外设TIMERx
timer_disable	禁能外设TIMERx
timer_auto_reload_shadow_enable	TIMERx自动重载影子使能
timer_auto_reload_shadow_disable	TIMERx自动重载影子禁能
timer_update_event_enable	TIMERx更新使能
timer_update_event_disable	TIMERx更新禁能
timer_counter_alignment	设置外设TIMERx的对齐模式
timer_counter_up_direction	设置外设TIMERx向上计数
timer_counter_down_direction	设置外设TIMERx向下计数
timer_prescaler_config	配置外设TIMERx预分频器
timer_repetition_value_config	配置外设TIMERx的重复计数器
timer_autoreload_value_config	配置外设TIMERx的自动重载寄存器
timer_counter_value_config	配置外设TIMERx的计数器值
timer_counter_read	读取外设TIMERx的计数器值
timer_prescaler_read	读取外设TIMERx的预分频器值
timer_single_pulse_mode_config	配置外设TIMERx的单脉冲模式
timer_update_source_config	配置外设TIMERx的更新源
timer_dma_enable	外设TIMERx的DMA使能
timer_dma_disable	外设TIMERx的DMA禁能
timer_channel_dma_request_source_select	外设TIMERx的通道DMA请求源选择
timer_dma_transfer_config	配置外设TIMERx的DMA模式
timer_event_software_generate	软件产生事件
timer_break_struct_para_init	将TIMER中止功能参数结构体中所有参数初始化为默认值
timer_break_config	配置中止功能
timer_break_enable	使能TIMERx的中止功能
timer_break_disable	禁能TIMERx的中止功能
timer_automatic_output_enable	自动输出使能
timer_automatic_output_disable	自动输出禁能
timer_primary_output_config	所有的通道输出使能
timer_channel_control_shadow_config	通道换相控制影子寄存器配置
timer_channel_control_shadow_update_config	通道换相控制影子寄存器更新控制
timer_channel_output_struct_para_init	将TIMER通道输出参数结构体中所有参数初始化为默认值
timer_channel_output_config	外设TIMERx的通道输出配置
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

库函数名称	库函数描述
timer_channel_output_shadow_config	配置TIMERx通道输出比较影子寄存器功能
timer_channel_output_fast_config	配置TIMERx通道输出比较快速功能
timer_channel_output_clear_config	配置TIMERx的通道输出比较清0功能
timer_channel_output_polarity_config	通道输出极性配置
timer_channel_complementary_output_polarity_config	互补通道输出极性配置
timer_channel_output_state_config	配置通道状态
timer_channel_complementary_output_state_config	配置互补通道输出状态
timer_channel_input_struct_para_init	将TIMER通道输入参数结构体中所有参数初始化为默认值
timer_input_capture_config	配置TIMERx输入捕获参数
timer_channel_input_capture_prescaler_config	配置TIMERx通道输入捕获预分频值
timer_channel_capture_value_register_read	读取通道输入捕获值
timer_input_pwm_capture_config	配置TIMERx捕获PWM输入参数
timer_hall_mode_config	配置TIMERx的HALL接口功能
timer_input_trigger_source_select	TIMERx的输入触发源选择
timer_master_output_trigger_source_select	选择TIMERx主模式输出触发
timer_slave_mode_select	TIMERx从模式配置
timer_master_slave_mode_config	TIMERx主从模式配置
timer_external_trigger_config	配置TIMERx外部触发输入
timer_quadrature_decoder_mode_config	TIMERx配置为正交译码器模式
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_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

库函数名称	库函数描述
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 3-897. 结构体 timer\_parameter\_struct

成员名称	功能描述
prescaler	预分频值（0~65535）
alignedmode	对齐模式（TIMER_COUNTER_EDGE, TIMER_COUNTER_CENTER_DOWN, TIMER_COUNTER_CENTER_UP, TIMER_COUNTER_CENTER_BOTH）
counterdirection	计数方向（TIMER_COUNTER_UP, TIMER_COUNTER_DOWN）
period	周期
clockdivision	时钟分频因子（TIMER_CKDIV_DIV1, TIMER_CKDIV_DIV2, TIMER_CKDIV_DIV4）
repetitioncounter	重复计数器值（0~255）

### 结构体 timer\_break\_parameter\_struct

表 3-898. 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 3-899. 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）



成员名称	功能描述
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 3-900. 结构体 timer\_ic\_parameter\_struct

成员名称	功能描述
icpolarity	通道输入极性 (TIMER_IC_POLARITY_RISING, TIMER_IC_POLARITY_FALLING, TIMER_IC_POLARITY_BOTH_EDGE)
icselection	通道输入模式选择 (TIMER_IC_SELECTION_DIRECTTI, TIMER_IC_SELECTION_INDIRECTTI, TIMER_IC_SELECTION_ITS)
icprescaler	通道输入捕获预分频 (TIMER_IC_PSC_DIV1, TIMER_IC_PSC_DIV2, TIMER_IC_PSC_DIV4, TIMER_IC_PSC_DIV8)
icfilter	通道输入捕获滤波 (0~15)

### 函数 timer\_deinit

函数timer\_deinit描述见下表:

表 3-901. 函数 timer\_deinit

函数名称	timer_deinit
函数原型	void timer_deinit(uint32_t timer_periph);
功能描述	复位外设TIMER
先决条件	-
被调用函数	rcu_periph_reset_enable / rcu_periph_reset_disable
输入参数{in}	
timer_periph	TIMER外设
TIMEx(x=0..5, 16)	TIMER外设选择
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* reset TIMER0 */
timer_deinit (TIMER0);
```

## 函数 timer\_struct\_para\_init

函数timer\_struct\_para\_init描述见下表：

表 3-902. 函数 timer\_struct\_para\_init

函数名称	timer_struct_para_init
函数原型	void timer_struct_para_init(timer_parameter_struct* initpara);
功能描述	将TIMER初始化参数结构体中所有参数初始化为默认值
先决条件	-
被调用函数	-
输入参数{in}	
initpara	TIMER初始化结构体，结构体成员参考 <a href="#">表3-897. 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 3-903. 函数 timer\_init

函数名称	timer_init
函数原型	void timer_init(uint32_t timer_periph, timer_parameter_struct* initpara);
功能描述	初始化外设TIMER
先决条件	-
被调用函数	-
输入参数{in}	
timer_periph	TIMER外设
TIMERx(x=0..5, 15, 16)	TIMER外设选择
输入参数{in}	
initpara	TIMER初始化结构体，结构体成员参考 <a href="#">表3-897. 结构体 timer_parameter_struct</a> 。
输出参数{out}	
-	-
返回值	

-	-
---	---

例如:

```
/* initialize TIMERO */

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(TIMERO, &timer_initpara);
```

### 函数 timer\_enable

函数timer\_enable描述见下表:

表 3-904. 函数 timer\_enable

函数名称	timer_enable
函数原型	void timer_enable(uint32_t timer_periph);
功能描述	使能外设TIMER
先决条件	-
被调用函数	-
输入参数{in}	
timer_periph	TIMER外设
TIMEx(x=0..5, 15, 16)	TIMER外设选择
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* enable TIMERO */

timer_enable(TIMERO);
```

### 函数 timer\_disable

函数timer\_disable描述见下表:

表 3-905. 函数 timer\_disable

函数名称	timer_disable
函数原型	void timer_disable(uint32_t timer_periph);
功能描述	禁能外设TIMER
先决条件	-
被调用函数	-
输入参数{in}	
timer_periph	TIMER外设
TIMERx(x=0..5, 15, 16)	TIMER外设选择
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* disable TIMER0 */
```

```
timer_disable(TIMER0);
```

### 函数 timer\_auto\_reload\_shadow\_enable

函数timer\_auto\_reload\_shadow\_enable描述见下表:

表 3-906. 函数 timer\_auto\_reload\_shadow\_enable

函数名称	timer_auto_reload_shadow_enable
函数原型	void timer_auto_reload_shadow_enable(uint32_t timer_periph);
功能描述	TIMER自动重载影子使能
先决条件	-
被调用函数	-
输入参数{in}	
timer_periph	TIMER外设
TIMERx(x=0..5, 15, 16)	TIMER外设选择
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* enable the TIMER0 auto reload shadow function */
```

```
timer_auto_reload_shadow_enable(TIMER0);
```

**函数 timer\_auto\_reload\_shadow\_disable**

函数timer\_auto\_reload\_shadow\_disable描述见下表:

**表 3-907. 函数 timer\_auto\_reload\_shadow\_disable**

函数名称	timer_auto_reload_shadow_disable
函数原型	void timer_auto_reload_shadow_disable (uint32_t timer_periph);
功能描述	TIMER自动重载影子禁能
先决条件	-
被调用函数	-
输入参数{in}	
timer_periph	TIMER外设
TIMERx(x=0..5, 15, 16)	TIMER外设选择
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* disable the TIMER0 auto reload shadow function */
```

```
timer_auto_reload_shadow_disable(TIMER0);
```

**函数 timer\_update\_event\_enable**

函数timer\_update\_event\_enable描述见下表:

**表 3-908. 函数 timer\_update\_event\_enable**

函数名称	timer_update_event_enable
函数原型	void timer_update_event_enable(uint32_t timer_periph);
功能描述	TIMER更新使能
先决条件	-
被调用函数	-
输入参数{in}	
timer_periph	TIMER外设
TIMERx(x=0..5, 15, 16)	TIMER外设选择
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* enable TIMER0 the update event */
```

```
timer_update_event_enable(TIMERO);
```

### 函数 timer\_update\_event\_disable

函数timer\_update\_event\_disable描述见下表：

表 3-909. 函数 timer\_update\_event\_disable

函数名称	timer_update_event_disable
函数原型	void timer_update_event_disable (uint32_t timer_periph);
功能描述	TIMER更新禁能
先决条件	-
被调用函数	-
输入参数{in}	
timer_periph	TIMER外设
TIMERx(x=0..5, 15, 16)	TIMER外设选择
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* disable TIMERO the update event */
```

```
timer_update_event_disable(TIMERO);
```

### 函数 timer\_counter\_alignment

函数timer\_counter\_alignment描述见下表：

表 3-910. 函数 timer\_counter\_alignment

函数名称	timer_counter_alignment
函数原型	void timer_counter_alignment(uint32_t timer_periph, uint16_t aligned);
功能描述	设置外设TIMER的对齐模式
先决条件	-
被调用函数	-
输入参数{in}	
timer_periph	TIMER外设
TIMERx(x=0..4)	TIMER外设选择
输入参数{in}	
aligned	对齐模式
TIMER_COUNTER_EDGE	无中央对齐计数模式(边沿对齐模式)，DIR位指定了计数方向
TIMER_COUNTER_CENTER_DOWN	中央对齐向下计数置1模式。计数器在中央计数模式计数，通道被配置在输出模式（TIMERx_CHCTL0寄存器中CHxMS=00），只有在向下计数时，通道的比较

	中断标志置1
<i>TIMER_COUNTER_CENTER_UP</i>	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（ <i>TIMERx_CHCTL0</i> 寄存器中 <i>CHxMS=00</i> ），只有在向上计数时，通道的比较中断标志置1
<i>TIMER_COUNTER_CENTER_BOTH</i>	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（ <i>TIMERx_CHCTL0</i> 寄存器中 <i>CHxMS=00</i> ），在向上和向下计数时，通道的比较中断标志都会置1
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* set TIMER0 counter center-aligned and counting up assert mode */
timer_counter_alignment(TIMER0, TIMER_COUNTER_CENTER_UP);
```

### 函数 timer\_counter\_up\_direction

函数timer\_counter\_up\_direction描述见下表：

表 3-911. 函数 timer\_counter\_up\_direction

函数名称	timer_counter_up_direction
函数原型	void timer_counter_up_direction(uint32_t timer_periph);
功能描述	设置外设TIMER向上计数
先决条件	计数器设置为无中央对齐计数模式（边沿对齐模式）
被调用函数	-
输入参数{in}	
timer_periph	TIMER外设
<i>TIMERx(x=0..4)</i>	TIMER外设选择
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* set TIMER0 counter up direction */
timer_counter_up_direction(TIMER0);
```

### 函数 timer\_counter\_down\_direction

函数timer\_counter\_down\_direction描述见下表：

表 3-912. 函数 timer\_counter\_down\_direction

函数名称	timer_counter_down_direction
------	------------------------------

函数原型	void timer_counter_down_direction(uint32_t timer_periph);
功能描述	设置外设TIMER向下计数
先决条件	计数器设置为无中央对齐计数模式（边沿对齐模式）
被调用函数	-
输入参数{in}	
timer_periph	TIMER外设
TIMERx(x=0..4)	TIMER外设选择
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* set TIMER0 counter down direction */
timer_counter_down_direction(TIMER0);
```

### 函数 timer\_prescaler\_config

函数timer\_prescaler\_config描述见下表：

表 3-913. 函数 timer\_prescaler\_config

函数名称	timer_prescaler_config
函数原型	void timer_prescaler_config(uint32_t timer_periph, uint16_t prescaler, uint32_t pscreload);
功能描述	配置外设TIMER预分频器
先决条件	-
被调用函数	-
输入参数{in}	
timer_periph	TIMER外设
TIMERx(x=0..5, 15, 16)	TIMER外设选择
输入参数{in}	
prescaler	预分频值，0~65535
输入参数{in}	
pscreload	预分频值加载模式
TIMER_PSC_RELO AD_NOW	预分频值立即加载
TIMER_PSC_RELO AD_UPDATE	预分频值在下次更新事件发生时加载
输出参数{out}	
-	-
返回值	
-	-



例如：

```
/* configure TIMER0 prescaler */
```

```
timer_prescaler_config(TIMER0, 3000, TIMER_PSC_RELOAD_NOW);
```

### 函数 timer\_repetition\_value\_config

函数timer\_repetition\_value\_config描述见下表：

表 3-914. 函数 timer\_repetition\_value\_config

函数名称	timer_repetition_value_config
函数原型	void timer_repetition_value_config(uint32_t timer_periph, uint16_t repetition);
功能描述	配置外设TIMER的重复计数器
先决条件	-
被调用函数	-
输入参数{in}	
timer_periph	TIMER外设
TIMERx(x=0, 15, 16)	TIMER外设选择
输入参数{in}	
repetition	重复计数器值，取值范围0~255
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* configure TIMER0 repetition register value */
```

```
timer_repetition_value_config(TIMER0, 98);
```

### 函数 timer\_autoreload\_value\_config

函数timer\_autoreload\_value\_config描述见下表：

表 3-915. 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TIMER的自动重载寄存器
先决条件	-
被调用函数	-
输入参数{in}	
timer_periph	TIMER外设
TIMERx(x=0..5, 15, 16)	TIMER外设选择
输入参数{in}	

<b>autoreload</b>	计数器自动重载值（0-0xFFFFFFFF）
<b>输出参数{out}</b>	
-	-
<b>返回值</b>	
-	-

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 3-916. 函数 timer\_counter\_value\_config**

<b>函数名称</b>	timer_counter_value_config
<b>函数原型</b>	void timer_counter_value_config(uint32_t timer_periph, uint32_t counter);
<b>功能描述</b>	配置外设TIMER的计数器值
<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
TIMERx(x=0..5, 15, 16)	TIMER外设选择
<b>输入参数{in}</b>	
<b>counter</b>	计数器值（0-0xFFFFFFFF）
<b>输出参数{out}</b>	
-	-
<b>返回值</b>	
-	-

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 3-917. 函数 timer\_counter\_read**

<b>函数名称</b>	timer_counter_read
<b>函数原型</b>	uint32_t timer_counter_read(uint32_t timer_periph);
<b>功能描述</b>	读取外设TIMER的计数器值
<b>先决条件</b>	-

被调用函数	-
输入参数{in}	
timer_periph	TIMER外设
TIMERx(x=0..5, 15, 16)	TIMER外设选择
输出参数{out}	
-	-
返回值	
uint32_t	外设TIMER的计数器值（0~0xFFFFFFFF）

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 3-918. 函数 timer\_prescaler\_read

函数名称	timer_prescaler_read
函数原型	uint16_t timer_prescaler_read(uint32_t timer_periph);
功能描述	读取外设TIMER的预分频器值
先决条件	-
被调用函数	-
输入参数{in}	
timer_periph	TIMER外设
TIMERx(x=0..5, 15, 16)	TIMER外设选择
输出参数{out}	
-	-
返回值	
uint16_t	外设TIMER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 3-919. 函数 timer\_single\_pulse\_mode\_config

函数名称	timer_single_pulse_mode_config
函数原型	void timer_single_pulse_mode_config(uint32_t timer_periph, uint8_t spmode);
功能描述	配置外设TIMER的单脉冲模式
先决条件	-
被调用函数	-
输入参数{in}	
timer_periph	TIMER外设
TIMERx(x=0..5, 15, 16)	TIMER外设选择
输入参数{in}	
spmode	脉冲模式
TIMER_SP_MODE_SINGLE	单脉冲模式计数
TIMER_SP_MODE_REPETITIVE	重复模式计数
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* configure TIMER0 single pulse mode */
```

```
timer_single_pulse_mode_config(TIMER0, TIMER_SP_MODE_SINGLE);
```

### 函数 timer\_update\_source\_config

函数timer\_update\_source\_config描述见下表:

表 3-920. 函数 timer\_update\_source\_config

函数名称	timer_update_source_config
函数原型	void timer_update_source_config(uint32_t timer_periph, uint32_t update);
功能描述	配置外设TIMER的更新源
先决条件	-
被调用函数	-
输入参数{in}	
timer_periph	TIMER外设
TIMERx(x=0..5, 15, 16)	TIMER外设选择
输入参数{in}	
update	更新源
TIMER_UPDATE_SRC_GLOBAL	下述任一事件产生更新中断或DMA请求: - UPG位被置1

	- 计数器溢出/下溢 - 从模式控制器产生的更新
<i>TIMER_UPDATE_SRC_REGULAR</i>	只有计数器溢出/下溢才产生更新中断或DMA请求
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* configure TIMER update only by counter overflow/underflow */
```

```
timer_update_source_config(TIMER0, TIMER_UPDATE_SRC_REGULAR);
```

### 函数 timer\_dma\_enable

函数timer\_dma\_enable描述见下表:

表 3-921. 函数 timer\_dma\_enable

函数名称	timer_dma_enable
函数原型	void timer_dma_enable(uint32_t timer_periph, uint16_t dma);
功能描述	外设TIMER的DMA使能
先决条件	-
被调用函数	-
输入参数{in}	
timer_periph	TIMER外设
<i>TIMERx</i>	参考具体参数
输入参数{in}	
dma	DMA源
<i>TIMER_DMA_UPD</i>	更新DMA请求, <i>TIMERx</i> ( <i>x</i> =0..5,15,16)
<i>TIMER_DMA_CH0D</i>	通道0比较/捕获 DMA请求, <i>TIMERx</i> ( <i>x</i> =0..4,15,16)
<i>TIMER_DMA_CH1D</i>	通道1比较/捕获 DMA请求, <i>TIMERx</i> ( <i>x</i> =0..4)
<i>TIMER_DMA_CH2D</i>	通道2比较/捕获 DMA请求, <i>TIMERx</i> ( <i>x</i> =0..4)
<i>TIMER_DMA_CH3D</i>	通道3比较/捕获 DMA请求, <i>TIMERx</i> ( <i>x</i> =0..4)
<i>TIMER_DMA_CMTD</i>	换相DMA更新请求, <i>TIMERx</i> ( <i>x</i> =0,15,16)
<i>TIMER_DMA_TRGD</i>	触发DMA请求使能, <i>TIMERx</i> ( <i>x</i> =0..4)
输出参数{out}	
-	-

返回值	
-	-

例如：

```
/* enable the TIMER0 update DMA */
```

```
timer_dma_enable(TIMER0, TIMER_DMA_UPD);
```

### 函数 timer\_dma\_disable

函数timer\_dma\_disable描述见下表：

表 3-922. 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TIMER的DMA禁能
先决条件	-
被调用函数	-
输入参数{in}	
timer_periph	TIMER外设
TIMERx	参考具体参数
输入参数{in}	
dma	DMA源
TIMER_DMA_UPD	更新DMA请求, TIMERx(x=0..5,15,16)
TIMER_DMA_CH0 D	通道0比较/捕获 DMA请求, TIMERx(x=0..4,15,16)
TIMER_DMA_CH1 D	通道1比较/捕获 DMA请求, TIMERx(x=0..4)
TIMER_DMA_CH2 D	通道2比较/捕获 DMA请求, TIMERx(x=0..4)
TIMER_DMA_CH3 D	通道3比较/捕获 DMA请求, TIMERx(x=0..4)
TIMER_DMA_CMT D	换相DMA更新请求, TIMERx(x=0,15,16)
TIMER_DMA_TRG D	触发DMA请求使能, TIMERx(x=0..4)
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* disable the TIMER0 update DMA */
```

```
timer_dma_disable(TIMER0, TIMER_DMA_UPD);
```

## 函数 timer\_channel\_dma\_request\_source\_select

函数timer\_channel\_dma\_request\_source\_select描述见下表:

表 3-923. 函数 timer\_channel\_dma\_request\_source\_select

函数名称	timer_channel_dma_request_source_select
函数原型	void timer_channel_dma_request_source_select(uint32_t timer_periph, uint32_t dma_request);
功能描述	外设TIMER的通道DMA请求源选择
先决条件	-
被调用函数	-
输入参数{in}	
timer_periph	TIMER外设
TIMEx(x=0..4)	TIMER外设选择
输入参数{in}	
dma_request	通道的DMA请求源选择
TIMER_DMAREQUEST_CHANNELEVENT	当通道捕获/比较事件发生时, 发送通道n的DMA请求
TIMER_DMAREQUEST_UPDATEEVENT	当更新事件发生, 发送通道n的DMA请求
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* TIMER0 channel DMA request of channel n is sent when channel 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 3-924. 函数 timer\_dma\_transfer\_config

函数名称	timer_dma_transfer_config
函数原型	void timer_dma_transfer_config(uint32_t timer_periph, uint32_t dma_baseaddr, uint32_t dma_lenth);
功能描述	配置外设TIMER的DMA模式
先决条件	-
被调用函数	-
输入参数{in}	

<b>timer_periph</b>	TIMER外设
<i>TIMERx</i>	参考具体参数
<b>输入参数{in}</b>	
<b>dma_baseaddr</b>	DMA传输起始地址
<i>TIMER_DMACFG_DMATA_CTL0</i>	DMA传输起始地址: TIMER_DMACFG_DMATA_CTL0, TIMERx(x=0..4)
<i>TIMER_DMACFG_DMATA_CTL1</i>	DMA传输起始地址: TIMER_DMACFG_DMATA_CTL1, TIMERx(x=0..4)
<i>TIMER_DMACFG_DMATA_SMCFG</i>	DMA传输起始地址: TIMER_DMACFG_DMATA_SMCFG, TIMERx(x=0..4)
<i>TIMER_DMACFG_DMATA_DMAINTEN</i>	DMA传输起始地址: TIMER_DMACFG_DMATA_DMAINTEN, TIMERx(x=0..4)
<i>TIMER_DMACFG_DMATA_INTF</i>	DMA传输起始地址: TIMER_DMACFG_DMATA_INTF, TIMERx(x=0..4)
<i>TIMER_DMACFG_DMATA_SWEVG</i>	DMA传输起始地址: TIMER_DMACFG_DMATA_SWEVG, TIMERx(x=0..4)
<i>TIMER_DMACFG_DMATA_CHCTL0</i>	DMA传输起始地址: TIMER_DMACFG_DMATA_CHCTL0, TIMERx(x=0..4)
<i>TIMER_DMACFG_DMATA_CHCTL1</i>	DMA传输起始地址: TIMER_DMACFG_DMATA_CHCTL1, TIMERx(x=0..4)
<i>TIMER_DMACFG_DMATA_CHCTL2</i>	DMA传输起始地址: TIMER_DMACFG_DMATA_CHCTL2, TIMERx(x=0..4)
<i>TIMER_DMACFG_DMATA_CNT</i>	DMA传输起始地址: TIMER_DMACFG_DMATA_CNT, TIMERx(x=0..4)
<i>TIMER_DMACFG_DMATA_PSC</i>	DMA传输起始地址: TIMER_DMACFG_DMATA_PSC, TIMERx(x=0..4)
<i>TIMER_DMACFG_DMATA_CAR</i>	DMA传输起始地址: TIMER_DMACFG_DMATA_CAR, TIMERx(x=0..4)
<i>TIMER_DMACFG_DMATA_CREP</i>	DMA传输起始地址: TIMER_DMACFG_DMATA_CREP, TIMERx(x=0,15,16)
<i>TIMER_DMACFG_DMATA_CH0CV</i>	DMA传输起始地址: TIMER_DMACFG_DMATA_CH0CV, TIMERx(x=0..4,15,16)
<i>TIMER_DMACFG_DMATA_CH1CV</i>	DMA传输起始地址: TIMER_DMACFG_DMATA_CH1CV, TIMERx(x=0..4)
<i>TIMER_DMACFG_DMATA_CH2CV</i>	DMA传输起始地址: TIMER_DMACFG_DMATA_CH2CV, TIMERx(x=0..4)
<i>TIMER_DMACFG_DMATA_CH3CV</i>	DMA传输起始地址: TIMER_DMACFG_DMATA_CH3CV, TIMERx(x=0..4)
<i>TIMER_DMACFG_DMATA_CCHP</i>	DMA传输起始地址: TIMER_DMACFG_DMATA_CCHP, TIMERx(x=0,15,16)
<i>TIMER_DMACFG_DMATA_DMACFG</i>	DMA传输起始地址: TIMER_DMACFG_DMATA_DMACFG, TIMERx(x=0..4)



<i>TIMER_DMACFG_DMATA_DMATB</i>	DMA传输起始地址: TIMER_DMACFG_DMATA_DMATB, TIMERx(x=0..4)
输入参数{in}	
<b>dma_lenth</b>	DMA传输长度
<i>TIMER_DMACFG_DMATC_xTRANSFER</i>	x=1..18, DMA传输 x 次
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* configure the TIMER0 DMA transfer */
```

```
timer_dma_transfer_config(TIMER0, TIMER_DMACFG_DMATA_CTL0,
TIMER_DMACFG_DMATC_5TRANSFER);
```

### 函数 timer\_event\_software\_generate

函数timer\_event\_software\_generate描述见下表:

表 3-925. 函数 timer\_event\_software\_generate

函数名称	timer_event_software_generate
函数原型	void timer_event_software_generate(uint32_t timer_periph, uint16_t event);
功能描述	软件产生事件
先决条件	-
被调用函数	-
输入参数{in}	
<b>timer_periph</b>	TIMER外设
<i>TIMERx</i>	参考具体参数
输入参数{in}	
<b>event</b>	事件源
<i>TIMER_EVENT_SRC_UPG</i>	更新事件产生, TIMERx(x=0..5,15,16)
<i>TIMER_EVENT_SRC_CH0G</i>	通道0捕获或比较事件发生, TIMERx(x=0..4,15,16)
<i>TIMER_EVENT_SRC_CH1G</i>	通道1捕获或比较事件发生, TIMERx(x=0..4)
<i>TIMER_EVENT_SRC_CH2G</i>	通道2捕获或比较事件发生, TIMERx(x=0..4)
<i>TIMER_EVENT_SRC_CH3G</i>	通道3捕获或比较事件发生, TIMERx(x=0..4)
<i>TIMER_EVENT_SRC</i>	通道换相更新事件发生, TIMERx(x=0,15,16)

C_CMTG	
TIMER_EVENT_SR C_TRGG	触发事件产生, TIMERx(x=0..4)
TIMER_EVENT_SR C_BRKG	产生中止事件, TIMERx(x=0,15,16)
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* software generate update event*/
```

```
timer_event_software_generate(TIMER0, TIMER_EVENT_SRC_UPG);
```

### 函数 timer\_break\_struct\_para\_init

函数timer\_break\_struct\_para\_init描述见下表:

表 3-926. 函数 timer\_break\_struct\_para\_init

函数名称	timer_break_struct_para_init
函数原型	void timer_break_struct_para_init(timer_break_parameter_struct* breakpara);
功能描述	将TIMER中止功能参数结构体中所有参数初始化为默认值
先决条件	-
被调用函数	-
输入参数{in}	
breakpara	中止功能配置结构体, 详见 <a href="#">表3-898. 结构体timer_break_parameter_struct.</a>
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* initialize TIMER break parameter struct with a default value */
```

```
timer_break_parameter_struct timer_breakpara;
```

```
timer_break_struct_para_init(&timer_breakpara);
```

### 函数 timer\_break\_config

函数timer\_break\_config描述见下表:

表 3-927. 函数 timer\_break\_config

函数名称	timer_break_config
函数原型	void timer_break_config(uint32_t timer_periph, timer_break_parameter_struct*

	breakpara);
功能描述	配置中止功能
先决条件	-
被调用函数	-
输入参数{in}	
timer_periph	TIMER外设
TIMERx(x=0, 15, 16)	TIMER外设选择
输入参数{in}	
breakpara	中止功能配置结构体, 详见 <a href="#">表3-898. 结构体timer_break_parameter_struct</a> 。
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* configure TIMER0 break function */
timer_break_parameter_struct timer_breakpara;

timer_breakpara.runoffstate      = TIMER_ROS_STATE_DISABLE;
timer_breakpara.ideloffstate     = TIMER_IOS_STATE_DISABLE ;
timer_breakpara.deadtime         = 255;
timer_breakpara.breakpolarity    = TIMER_BREAK_POLARITY_LOW;
timer_breakpara.outputautostate  = TIMER_OUTAUTO_ENABLE;
timer_breakpara.protectmode      = TIMER_CCHP_PROT_0;
timer_breakpara.breakstate       = TIMER_BREAK_ENABLE;

timer_break_config(TIMER0,&timer_breakpara);
```

### 函数 timer\_break\_enable

函数timer\_break\_enable描述见下表:

表 3-928. 函数 timer\_break\_enable

函数名称	timer_break_enable
函数原型	void timer_break_enable(uint32_t timer_periph);
功能描述	使能TIMER的中止功能
先决条件	只有在TIMERx_CCHP寄存器的PROT [1:0] =00 时, 才可修改
被调用函数	-
输入参数{in}	
timer_periph	TIMER外设
TIMERx(x=0, 15, 16)	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 3-929. 函数 timer\_break\_disable

函数名称	timer_break_disable
函数原型	void timer_break_disable (uint32_t timer_periph);
功能描述	禁能TIMER的中止功能
先决条件	只有在TIMERx_CCHP寄存器的PROT [1:0] =00 时，才可修改
被调用函数	-
输入参数{in}	
timer_periph	TIMER外设
TIMERx(x=0,15,16)	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 3-930. 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=0,15,16)	TIMER外设选择

输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* enable TIMER0 output automatic function */
```

```
timer_automatic_output_enable(TIMER0);
```

### 函数 timer\_automatic\_output\_disable

函数timer\_automatic\_output\_disable描述见下表：

表 3-931. 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=0,15,16)	TIMER外设选择
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* disable TIMER0 output automatic function */
```

```
timer_automatic_output_disable(TIMER0);
```

### 函数 timer\_primary\_output\_config

函数timer\_primary\_output\_config描述见下表：

表 3-932. 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, 15, 16)	TIMER外设选择
输入参数{in}	
newvalue	控制状态
ENABLE	使能
DISABLE	禁能
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* enable TIMER0 primary output function */
```

```
timer_primary_output_config(TIMER0, ENABLE);
```

### 函数 timer\_channel\_control\_shadow\_config

函数timer\_channel\_control\_shadow\_config描述见下表:

表 3-933. 函数 timer\_channel\_control\_shadow\_config

函数名称	timer_channel_control_shadow_config
函数原型	void timer_channel_control_shadow_config(uint32_t timer_periph, ControlStatus newvalue);
功能描述	通道换相控制影子配置
先决条件	-
被调用函数	-
输入参数{in}	
timer_periph	TIMER外设
TIMERx(x=0, 15, 16)	TIMER外设选择
输入参数{in}	
newvalue	控制状态
ENABLE	使能
DISABLE	禁能
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* channel capture/compare control shadow register enable */
```

```
timer_channel_control_shadow_config(TIMER0, ENABLE);
```

## 函数 timer\_channel\_control\_shadow\_update\_config

函数timer\_channel\_control\_shadow\_update\_config描述见下表：

表 3-934. 函数 timer\_channel\_control\_shadow\_update\_config

函数名称	timer_channel_control_shadow_update_config
函数原型	void timer_channel_control_shadow_update_config(uint32_t timer_periph, uint32_t ccuctl);
功能描述	通道换相控制影子寄存器更新控制
先决条件	-
被调用函数	-
输入参数{in}	
timer_periph	TIMER外设
TIMERx(x=0,15,16)	TIMER外设选择
输入参数{in}	
ccuctl	通道换相控制影子寄存器更新控制
TIMER_UPDATECTL_CCUC	CMTG位被置1时更新影子寄存器
TIMER_UPDATECTL_CCUTRI	当CMTG位被置1或检测到TRIGI上升沿时，影子寄存器更新
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* configure TIMER0 channel control shadow register update when CMTG bit is set */
timer_channel_control_shadow_update_config(TIMER0, TIMER_UPDATECTL_CCUC);
```

## 函数 timer\_channel\_output\_struct\_para\_init

函数timer\_channel\_output\_struct\_para\_init描述见下表：

表 3-935. 函数 timer\_channel\_output\_struct\_para\_init

函数名称	timer_channel_output_struct_para_init
函数原型	void timer_channel_output_struct_para_init(timer_oc_parameter_struct* ocpa);
功能描述	将TIMER通道输出参数结构体中所有参数初始化为默认值
先决条件	-
被调用函数	-
输入参数{in}	
ocpa	输出通道结构体，详见 <a href="#">表3-899. 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 3-936. 函数 timer\_channel\_output\_config**

函数名称	timer_channel_output_config
函数原型	void timer_channel_output_config(uint32_t timer_periph, uint16_t channel, timer_oc_parameter_struct* ocpara);
功能描述	外设TIMER的通道输出配置
先决条件	-
被调用函数	-
输入参数{in}	
timer_periph	TIMER外设
TIMERx	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..4,15,16)
TIMER_CH_1	通道1, TIMERx(x=0..4)
TIMER_CH_2	通道2, TIMERx(x=0..4)
TIMER_CH_3	通道3, TIMERx(x=0..4)
输入参数{in}	
ocpara	输出通道结构体，详见 <a href="#">表3-899. 结构体timer_oc_parameter_struct</a>
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* configure TIMER0 channel 0 output function */
```

```
timer_oc_parameter_struct timer_ocintpara;
```

```
timer_ocintpara.outputstate = TIMER_CCX_ENABLE;
```

```
timer_ocintpara.outputnstate = TIMER_CCXN_ENABLE;
```

```
timer_ocintpara.ocpolarity = TIMER_OC_POLARITY_HIGH;
```



```

timer_ocintpara.ocnpolarity = TIMER_OCN_POLARITY_HIGH;

timer_ocintpara.ocidlestate = TIMER_OC_IDLE_STATE_HIGH;

timer_ocintpara.ocidlestate = 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 3-937. 函数 timer\_channel\_output\_mode\_config**

函数名称	timer_channel_output_mode_config
函数原型	void timer_channel_output_mode_config(uint32_t timer_periph, uint16_t channel, uint16_t ocmode);
功能描述	配置外设TIMER通道输出比较模式
先决条件	-
被调用函数	-
输入参数{in}	
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..4,15,16)
TIMER_CH_1	通道1, TIMERx (x=0..4)
TIMER_CH_2	通道2, TIMERx (x=0..4)
TIMER_CH_3	通道3, TIMERx (x=0..4)
输入参数{in}	
ocmode	通道输出比较模式
TIMER_OC_MODE_TIMING	冻结模式
TIMER_OC_MODE_ACTIVE	匹配时设置为高
TIMER_OC_MODE_INACTIVE	匹配时设置为低
TIMER_OC_MODE_TOGGLE	匹配时翻转
TIMER_OC_MODE_LOW	强制为低
TIMER_OC_MODE_HIGH	强制为高
TIMER_OC_MODE_PWM0	PWM模式0
TIMER_OC_MODE	PWM模式1

<code>_PWM1</code>	
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* configure TIMER0 channel PWM 0 mode */
```

```
timer_channel_output_mode_config(TIMER0,TIMER_CH_0,TIMER_OC_MODE_PWM0);
```

### 函数 timer\_channel\_output\_pulse\_value\_config

函数timer\_channel\_output\_pulse\_value\_config描述见下表：

**表 3-938. 函数 timer\_channel\_output\_pulse\_value\_config**

函数名称	timer_channel_output_pulse_value_config
函数原型	void timer_channel_output_pulse_value_config(uint32_t timer_periph, uint16_t channel, uint32_t pulse);
功能描述	配置外设TIMER的通道输出比较值
先决条件	-
被调用函数	-
输入参数{in}	
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..4,15,16)
TIMER_CH_1	通道1, TIMERx (x=0..4)
TIMER_CH_2	通道2, TIMERx (x=0..4)
TIMER_CH_3	通道3, TIMERx (x=0..4)
输入参数{in}	
pulse	通道输出比较值 (0~0xFFFFFFFF)
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* configure TIMER0 channel 0 output pulse value */
```

```
timer_channel_output_pulse_value_config(TIMER0, TIMER_CH_0, 399);
```

**函数 timer\_channel\_output\_shadow\_config**

函数timer\_channel\_output\_shadow\_config描述见下表：

**表 3-939. 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TIMER通道输出比较影子寄存器功能
先决条件	-
被调用函数	-
输入参数{in}	
timer_periph	TIMER外设
TIMERx	参考具体参数
输入参数{in}	
channel	待配置通道
TIMER_CH_0	通道0, TIMERx (x=0..4,15,16)
TIMER_CH_1	通道1, TIMERx (x=0..4)
TIMER_CH_2	通道2, TIMERx (x=0..4)
TIMER_CH_3	通道3, TIMERx (x=0..4)
输入参数{in}	
ocshadow	输出比较影子寄存器功能状态
TIMER_OC_SHADOW_ENABLE	使能输出比较影子寄存器
TIMER_OC_SHADOW_DISABLE	禁能输出比较影子寄存器
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/*configure TIMER0 channel 0 output shadow function */
```

```
timer_channel_output_shadow_config(TIMER0, TIMER_CH_0,  
TIMER_OC_SHADOW_ENABLE);
```

**函数 timer\_channel\_output\_fast\_config**

函数timer\_channel\_output\_fast\_config描述见下表：

**表 3-940. 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TIMER通道输出比较快速功能
先决条件	-
被调用函数	-
输入参数{in}	
timer_periph	TIMER外设
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..4,15,16)
TIMER_CH_1	通道1, TIMERx (x=0..4)
TIMER_CH_2	通道2, TIMERx (x=0..4)
TIMER_CH_3	通道3, TIMERx (x=0..4)
输入参数{in}	
ocfast	通道输出比较快速功能状态
TIMER_OC_FAST_ENABLE	通道输出比较快速功能使能
TIMER_OC_FAST_DISABLE	通道输出比较快速功能禁能
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* configure TIMER0 channel 0 output fast function */
```

```
timer_channel_output_fast_config(TIMER0, TIMER_CH_0, TIMER_OC_FAST_ENABLE);
```

### 函数 timer\_channel\_output\_clear\_config

函数timer\_channel\_output\_clear\_config描述见下表:

表 3-941. 函数 timer\_channel\_output\_clear\_config

函数名称	timer_channel_output_clear_config
函数原型	void timer_channel_output_clear_config(uint32_t timer_periph, uint16_t channel, uint16_t occlear);
功能描述	配置TIMER的通道输出比较清0功能
先决条件	-
被调用函数	-
输入参数{in}	
timer_periph	TIMER外设
TIMERx	参考具体参数
输入参数{in}	
channel	待配置通道

<i>TIMER_CH_0</i>	通道0, <i>TIMERx</i> ( <i>x</i> =0..4)
<i>TIMER_CH_1</i>	通道1, <i>TIMERx</i> ( <i>x</i> =0..4)
<i>TIMER_CH_2</i>	通道2, <i>TIMERx</i> ( <i>x</i> =0..4)
<i>TIMER_CH_3</i>	通道3, <i>TIMERx</i> ( <i>x</i> =0..4)
输入参数{in}	
<b>occlear</b>	通道比较输出清0功能状态
<i>TIMER_OC_CLEAR_ENABLE</i>	通道比较输出清0功能使能
<i>TIMER_OC_CLEAR_DISABLE</i>	通道比较输出清0功能禁能
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* configure TIMER0 channel 0 output clear function */
```

```
timer_channel_output_clear_config(TIMER0, TIMER_CH_0,  
TIMER_OC_CLEAR_ENABLE);
```

### 函数 timer\_channel\_output\_polarity\_config

函数timer\_channel\_output\_polarity\_config描述见下表:

表 3-942. 函数 timer\_channel\_output\_polarity\_config

函数名称	timer_channel_output_polarity_config
函数原型	void timer_channel_output_polarity_config(uint32_t timer_periph, uint16_t channel, uint16_t ocpolarity);
功能描述	通道输出极性配置
先决条件	-
被调用函数	-
输入参数{in}	
<b>timer_periph</b>	TIMER外设
<i>TIMERx</i>	参考具体参数
输入参数{in}	
<b>channel</b>	待配置通道
<i>TIMER_CH_0</i>	通道0, <i>TIMERx</i> ( <i>x</i> =0..4,15,16)
<i>TIMER_CH_1</i>	通道1, <i>TIMERx</i> ( <i>x</i> =0..4)
<i>TIMER_CH_2</i>	通道2, <i>TIMERx</i> ( <i>x</i> =0..4)
<i>TIMER_CH_3</i>	通道3, <i>TIMERx</i> ( <i>x</i> =0..4)
输入参数{in}	
<b>ocpolarity</b>	通道输出极性
<i>TIMER_OC_POLAR</i>	通道输出极性高电平有效

<i>ITY_HIGH</i>	
<i>TIMER_OC_POLARITY_LOW</i>	通道输出极性低电平有效
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* configure TIMER0 channel 0 output polarity */
```

```
timer_channel_output_polarity_config(TIMER0, TIMER_CH_0,  
TIMER_OC_POLARITY_HIGH);
```

### 函数 timer\_channel\_complementary\_output\_polarity\_config

函数timer\_channel\_complementary\_output\_polarity\_config描述见下表：

**表 3-943. 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 ocpolarity);
功能描述	互补通道输出极性配置
先决条件	-
被调用函数	-
输入参数{in}	
timer_periph	TIMER外设
<i>TIMERx</i>	参考具体参数
输入参数{in}	
channel	待配置通道
<i>TIMER_CH_0</i>	通道0, <i>TIMERx</i> (x=0,15,16)
<i>TIMER_CH_1</i>	通道1, <i>TIMERx</i> (x=0)
<i>TIMER_CH_2</i>	通道2, <i>TIMERx</i> (x=0)
输入参数{in}	
ocpolarity	互补通道输出极性
<i>TIMER_OCN_POLARITY_HIGH</i>	互补通道输出极性高电平有效
<i>TIMER_OCN_POLARITY_LOW</i>	互补通道输出极性低电平有效
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* configure TIMER0 channel 0 complementary output polarity */
```

```
timer_channel_complementary_output_polarity_config(TIMER0, TIMER_CH_0,  
TIMER_OCN_POLARITY_HIGH);
```

### 函数 timer\_channel\_output\_state\_config

函数timer\_channel\_output\_state\_config描述见下表：

**表 3-944. 函数 timer\_channel\_output\_state\_config**

函数名称	timer_channel_output_state_config
函数原型	void timer_channel_output_state_config(uint32_t timer_periph, uint16_t channel, uint32_t state);
功能描述	配置通道状态
先决条件	-
被调用函数	-
输入参数{in}	
timer_periph	TIMER外设
TIMERx	参考具体参数
输入参数{in}	
channel	待配置通道
TIMER_CH_0	通道0, TIMERx (x=0..4,15,16)
TIMER_CH_1	通道1, TIMERx (x=0..4)
TIMER_CH_2	通道2, TIMERx (x=0..4)
TIMER_CH_3	通道3, TIMERx (x=0..4)
输入参数{in}	
state	通道状态
TIMER_CCX_ENABLE	通道使能
TIMER_CCX_DISABLE	通道禁能
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* configure TIMER0 channel 0 enable state */
```

```
timer_channel_output_state_config(TIMER0, TIMER_CH_0, TIMER_CCX_ENABLE);
```

### 函数 timer\_channel\_complementary\_output\_state\_config

函数timer\_channel\_complementary\_output\_state\_config描述见下表：

表 3-945. 函数 timer\_channel\_complementary\_output\_state\_config

函数名称	timer_channel_complementary_output_state_config
函数原型	void timer_channel_complementary_output_state_config(uint32_t timer_periph, uint16_t channel, uint16_t ocnstate);
功能描述	配置互补通道输出状态
先决条件	-
被调用函数	-
输入参数{in}	
timer_periph	TIMER外设
TIMERx	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=0,15,16)
TIMER_CH_1	通道1, TIMERx (x=0)
TIMER_CH_2	通道2, TIMERx (x=0)
输入参数{in}	
state	互补通道状态
TIMER_CCXN_ENABLE	互补通道使能
TIMER_CCXN_DISABLE	互补通道禁能
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* configure TIMER0 channel 0 complementary output enable state */
timer_channel_complementary_output_state_config(TIMER0, TIMER_CH_0,
TIMER_CCXN_ENABLE);
```

### 函数 timer\_channel\_input\_struct\_para\_init

函数timer\_channel\_input\_struct\_para\_init描述见下表:

表 3-946. 函数 timer\_channel\_input\_struct\_para\_init

函数名称	timer_channel_input_struct_para_init
函数原型	void timer_channel_input_struct_para_init(timer_ic_parameter_struct* icpara);
功能描述	将TIMER通道输入参数结构体中所有参数初始化为默认值
先决条件	-
被调用函数	-
输入参数{in}	
icpara	通道输入结构体, 详见 <a href="#">表3-900. 结构体timer_ic_parameter_struct</a> 。



输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* initialize TIMER channel input parameter struct with a default value */
```

```
timer_ic_parameter_struct timer_icinitpara;
```

```
timer_channel_input_struct_para_init(&timer_icinitpara);
```

### 函数 timer\_input\_capture\_config

函数timer\_input\_capture\_config描述见下表：

表 3-947. 函数 timer\_input\_capture\_config

函数名称	timer_input_capture_config
函数原型	void timer_input_capture_config(uint32_t timer_periph, uint16_t channel, timer_ic_parameter_struct* icpara);
功能描述	配置TIMER输入捕获参数
先决条件	-
被调用函数	timer_channel_input_capture_prescaler_config
输入参数{in}	
timer_periph	TIMER外设
TIMEx	参考具体参数
输入参数{in}	
channel	待配置通道
TIMER_CH_0	通道0, TIMEx (x=0..4,15,16)
TIMER_CH_1	通道1, TIMEx (x=0..4)
TIMER_CH_2	通道2, TIMEx (x=0..4)
TIMER_CH_3	通道3, TIMEx (x=0..4)
输入参数{in}	
icpara	输入捕获结构体, 详见 <a href="#">表3-900. 结构体timer_ic_parameter_struct</a> 。
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* configure TIMER0 input capture parameter */
```

```
timer_ic_parameter_struct timer_icinitpara;
```

```
timer_icinitpara.icpolarity = TIMER_IC_POLARITY_RISING;
```

```
timer_icinitpara.icselection = TIMER_IC_SELECTION_DIRECTTTI;
```

```
timer_icinitpara.icprescaler = TIMER_IC_PSC_DIV1;
```

```
timer_icinitpara.icfilter      = 0x0;
```

```
timer_input_capture_config(TIMER0, TIMER_CH_0, &timer_icinitpara);
```

### 函数 timer\_channel\_input\_capture\_prescaler\_config

函数timer\_channel\_input\_capture\_prescaler\_config描述见下表:

**表 3-948. 函数 timer\_channel\_input\_capture\_prescaler\_config**

函数名称	timer_channel_input_capture_prescaler_config
函数原型	void timer_channel_input_capture_prescaler_config(uint32_t timer_periph, uint16_t channel, uint16_t prescaler);
功能描述	配置TIMER通道输入捕获预分频值
先决条件	-
被调用函数	-
输入参数{in}	
timer_periph	TIMER外设
TIMERx	参考具体参数
输入参数{in}	
channel	待配置通道
TIMER_CH_0	通道0, TIMERx (x=0..4,15,16)
TIMER_CH_1	通道1, TIMERx (x=0..4)
TIMER_CH_2	通道2, TIMERx (x=0..4)
TIMER_CH_3	通道3, TIMERx (x=0..4)
输入参数{in}	
prescaler	通道输入捕获预分频值
TIMER_IC_PSC_DIV1	不分频
TIMER_IC_PSC_DIV2	2分频
TIMER_IC_PSC_DIV4	4分频
TIMER_IC_PSC_DIV8	8分频
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* configure TIMER0 channel 0 input capture prescaler value */
```

timer\_channel\_input\_capture\_prescaler\_config(TIMERO, TIMER\_CH\_0,  
TIMER\_IC\_PSC\_DIV2);

### 函数 timer\_channel\_capture\_value\_register\_read

函数timer\_channel\_capture\_value\_register\_read描述见下表:

**表 3-949. 函数 timer\_channel\_capture\_value\_register\_read**

函数名称	timer_channel_capture_value_register_read
函数原型	uint32_t timer_channel_capture_value_register_read(uint32_t timer_periph, uint16_t channel);
功能描述	读取通道捕获值
先决条件	-
被调用函数	-
输入参数{in}	
timer_periph	TIMER外设
TIMERx	参考具体参数
输入参数{in}	
channel	待配置通道
TIMER_CH_0	通道0, TIMERx (x=0..4,15,16)
TIMER_CH_1	通道1, TIMERx (x=0..4)
TIMER_CH_2	通道2, TIMERx (x=0..4)
TIMER_CH_3	通道3, TIMERx (x=0..4)
输出参数{out}	
-	-
返回值	
uint32_t	通道输入捕获值, (0~0xFFFFFFFF)

例如:

```
/* read TIMERO channel 0 capture compare register value */
```

```
uint32_t ch0_value = 0;
```

```
ch0_value = timer_channel_capture_value_register_read(TIMERO, TIMER_CH_0);
```

### 函数 timer\_input\_pwm\_capture\_config

函数timer\_input\_pwm\_capture\_config描述见下表:

**表 3-950. 函数 timer\_input\_pwm\_capture\_config**

函数名称	timer_input_pwm_capture_config
函数原型	void timer_input_pwm_capture_config(uint32_t timer_periph, uint16_t channel, timer_ic_parameter_struct* icpwm);
功能描述	配置TIMER捕获PWM输入参数
先决条件	-
被调用函数	timer_channel_input_capture_prescaler_config

输入参数{in}	
<b>timer_periph</b>	TIMER外设
<i>TIMERx(x=0..4)</i>	TIMER外设选择
输入参数{in}	
<b>channel</b>	待配置通道
<i>TIMER_CH_0</i>	通道0
<i>TIMER_CH_1</i>	通道1
输入参数{in}	
<b>icpwm</b>	输入捕获结构体, 详见 <a href="#">表3-900. 结构体timer_ic_parameter_struct</a>
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* configure TIMER0 input pwm capture parameter */
timer_ic_parameter_struct timer_icinitpara;

timer_icinitpara.icpolarity = TIMER_IC_POLARITY_RISING;

timer_icinitpara.icselection = TIMER_IC_SELECTION_DIRECTTTI;

timer_icinitpara.icprescaler = TIMER_IC_PSC_DIV1;

timer_icinitpara.icfilter = 0x0;

timer_input_pwm_capture_config(TIMER0, TIMER_CH_0, &timer_icinitpara);
```

### 函数 timer\_hall\_mode\_config

函数timer\_hall\_mode\_config描述见下表:

表 3-951. 函数 timer\_hall\_mode\_config

<b>函数名称</b>	timer_hall_mode_config
<b>函数原型</b>	void timer_hall_mode_config(uint32_t timer_periph, uint32_t hallmode);
<b>功能描述</b>	配置TIMER的HALL接口功能
<b>先决条件</b>	-
<b>被调用函数</b>	-
输入参数{in}	
<b>timer_periph</b>	TIMER外设
<i>TIMERx(x=0..4)</i>	TIMER外设选择
输入参数{in}	
<b>hallmode</b>	HALL接口功能状态
<i>TIMER_HALLINTE</i> <i>RFACE_ENABLE</i>	HALL接口使能
<i>TIMER_HALLINTE</i>	HALL接口禁能

RFACE_DISABLE	
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 3-952. 函数 timer\_input\_trigger\_source\_select**

函数名称	timer_input_trigger_source_select
函数原型	void timer_input_trigger_source_select(uint32_t timer_periph, uint32_t intrigger);
功能描述	TIMER的输入触发源选择
先决条件	SMC[2:0] = 000
被调用函数	-
输入参数{in}	
timer_periph	TIMER外设
TIMERx(x=0..4)	TIMER外设选择
输入参数{in}	
intrigger	待选择的触发源
TIMER_SMCFG_T RGSEL_ITI0	内部触发输入0
TIMER_SMCFG_T RGSEL_ITI1	内部触发输入1
TIMER_SMCFG_T RGSEL_ITI2	内部触发输入2
TIMER_SMCFG_T RGSEL_ITI3	内部触发输入3
TIMER_SMCFG_T RGSEL_CIOF_ED	CIO的边沿标志位
TIMER_SMCFG_T RGSEL_CIOFE0	滤波后的通道0输入
TIMER_SMCFG_T RGSEL_CIOFE1	滤波后的通道1输入
TIMER_SMCFG_T RGSEL_ETIFP	滤波后的外部触发输入
输出参数{out}	

-	-
返回值	
-	-

例如:

```
/* select TIMER0 input trigger source */
```

```
timer_input_trigger_source_select (TIMER0, TIMER_SMCFG_TRGSEL_ITI0);
```

### 函数 timer\_master\_output\_trigger\_source\_select

函数timer\_master\_output\_trigger\_source\_select描述见下表:

**表 3-953. 函数 timer\_master\_output\_trigger\_source\_select**

函数名称	timer_master_output_trigger_source_select
函数原型	void timer_master_output_trigger_source_select(uint32_t timer_periph, uint32_t outrigger);
功能描述	选择TIMER主模式输出触发
先决条件	-
被调用函数	-
输入参数{in}	
timer_periph	TIMER外设
TIMERx(x=0..5)	TIMER外设选择
输入参数{in}	
outrigger	主模式输出触发
TIMER_TRI_OUT_SRC_RESET	复位。TIMERx_SWEVG寄存器的UPG位被置1或从模式控制器产生复位触发一次TRGO脉冲，后一种情况下，TRGO上的信号相对实际的复位会有一个延迟。
TIMER_TRI_OUT_SRC_ENABLE	使能。此模式可用于同时启动多个定时器或控制在一段时间内使能从定时器。主模式控制器选择计数器使能信号作为触发输出TRGO。当CEN控制位被置1或者暂停模式下触发输入为高电平时，计数器使能信号被置1。在暂停模式下，计数器使能信号受控于触发输入，在触发输入和TRGO上会有一个延迟，除非选择了主/从模式。
TIMER_TRI_OUT_SRC_UPDATE	更新。主模式控制器选择更新事件作为TRGO。
TIMER_TRI_OUT_SRC_CC0	捕获/比较脉冲.通道0在发生一次捕获或一次比较成功时，主模式控制器产生一个TRGO脉冲
TIMER_TRI_OUT_SRC_O0CPRE	比较。在这种模式下主模式控制器选择O0CPRE信号被用于作为触发输出TRGO
TIMER_TRI_OUT_SRC_O1CPRE	比较。在这种模式下主模式控制器选择O1CPRE信号被用于作为触发输出TRGO
TIMER_TRI_OUT_SRC_O2CPRE	比较。在这种模式下主模式控制器选择O2CPRE信号被用于作为触发输出TRGO
TIMER_TRI_OUT_SRC_O3CPRE	比较。在这种模式下主模式控制器选择O3CPRE信号被用于作为触发输出TRGO

输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* select TIMER0 master mode output trigger source */
```

```
timer_master_output_trigger_source_select(TIMER0, TIMER_TRI_OUT_SRC_RESET);
```

### 函数 timer\_slave\_mode\_select

函数timer\_slave\_mode\_select描述见下表:

表 3-954. 函数 timer\_slave\_mode\_select

函数名称	timer_slave_mode_select
函数原型	void timer_slave_mode_select(uint32_t timer_periph, uint32_t slavemode);
功能描述	TIMER从模式配置
先决条件	-
被调用函数	-
输入参数{in}	
timer_periph	TIMER外设
TIMERx(x=0..4)	TIMER外设选择
输入参数{in}	
slavemode	从模式
TIMER_SLAVE_MODE_DISABLE	关闭从模式
TIMER_QUAD_DECODER_MODE0	正交译码器模式0
TIMER_QUAD_DECODER_MODE1	正交译码器模式1
TIMER_QUAD_DECODER_MODE2	正交译码器模式2
TIMER_SLAVE_MODE_RESTART	复位模式
TIMER_SLAVE_MODE_PAUSE	暂停模式
TIMER_SLAVE_MODE_EVENT	事件模式
TIMER_SLAVE_MODE_EXTERNAL0	外部时钟模式0
输出参数{out}	
-	-
返回值	

-	-
---	---

例如:

```
/* select TIMER0 slave mode */
```

```
timer_slave_mode_select(TIMER0, TIMER_QUAD_DECODER_MODE0);
```

### 函数 timer\_master\_slave\_mode\_config

函数timer\_master\_slave\_mode\_config描述见下表:

表 3-955. 函数 timer\_master\_slave\_mode\_config

函数名称	timer_master_slave_mode_config
函数原型	void timer_master_slave_mode_config(uint32_t timer_periph, uint8_t masterslave);
功能描述	TIMER主从模式配置
先决条件	-
被调用函数	-
输入参数{in}	
timer_periph	TIMER外设
TIMERx(x=0..4)	TIMER外设选择
输入参数{in}	
masterslave	主从模式使能状态
TIMER_MASTER_SLAVE_MODE_ENABLE	主从模式使能
TIMER_MASTER_SLAVE_MODE_DISABLE	主从模式禁能
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* configure TIMER0 master slave mode */
```

```
timer_master_slave_mode_config(TIMER0, TIMER_MASTER_SLAVE_MODE_ENABLE);
```

### 函数 timer\_external\_trigger\_config

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

表 3-956. 函数 timer\_external\_trigger\_config

函数名称	timer_external_trigger_config
函数原型	void timer_external_trigger_config(uint32_t timer_periph, uint32_t extprescaler,



	uint32_t expolarity, uint32_t extfilter);
功能描述	配置TIMER外部触发输入
先决条件	-
被调用函数	-
输入参数{in}	
timer_periph	TIMER外设
TIMERx(x=0..4)	TIMER外设选择
输入参数{in}	
extprescaler	外部触发预分频
TIMER_EXT_TRI_P SC_OFF	不分频
TIMER_EXT_TRI_P SC_DIV2	2分频
TIMER_EXT_TRI_P SC_DIV4	4分频
TIMER_EXT_TRI_P SC_DIV8	8分频
输入参数{in}	
expolarity	外部触发输入极性
TIMER_ETP_FALLI NG	低电平或者下降沿有效
TIMER_ETP_RISIN G	高电平或者上升沿有效
输入参数{in}	
extfilter	外部触发滤波控制（0~15）
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* configure TIMER0 external trigger input */
timer_external_trigger_config(TIMER0, TIMER_EXT_TRI_PSC_DIV2,
TIMER_ETP_FALLING, 10);
```

### 函数 timer\_quadrature\_decoder\_mode\_config

函数timer\_quadrature\_decoder\_mode\_config描述见下表：

表 3-957. 函数 timer\_quadrature\_decoder\_mode\_config

函数名称	timer_quadrature_decoder_mode_config
函数原型	void timer_quadrature_decoder_mode_config(uint32_t timer_periph, uint32_t decomode, uint16_t ic0polarity, uint16_t ic1polarity);

功能描述	TIMER配置为正交译码器模式
先决条件	-
被调用函数	-
输入参数{in}	
timer_periph	TIMER外设
TIMERx(x=0..4)	TIMER外设选择
输入参数{in}	
decomode	正交译码器模式
TIMER_QUAD_DECODER_MODE0	根据CI0FE0的电平，计数器在CI1FE1的边沿向上/下计数
TIMER_QUAD_DECODER_MODE1	根据CI1FE1的电平，计数器在CI0FE0的边沿向上/下计数
TIMER_QUAD_DECODER_MODE2	根据另一个信号的输入电平，计数器在CI0FE0和CI1FE1的边沿向上/下计数
输入参数{in}	
ic0polarity	IC0极性
TIMER_IC_POLARITY_RISING	捕获上升边沿
TIMER_IC_POLARITY_FALLING	捕获下降边沿
TIMER_IC_POLARITY_BOTH_EDGE	双边沿有效
输入参数{in}	
ic1polarity	IC1极性
TIMER_IC_POLARITY_RISING	捕获上升边沿
TIMER_IC_POLARITY_FALLING	捕获下降边沿
TIMER_IC_POLARITY_BOTH_EDGE	双边沿有效
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* configure TIMER0 quadrature decoder mode */
```

```
timer_quadrature_decoder_mode_config(TIMER0, TIMER_QUAD_DECODER_MODE0,  
TIMER_IC_POLARITY_RISING, TIMER_IC_POLARITY_RISING);
```

### 函数 timer\_internal\_clock\_config

函数timer\_internal\_clock\_config描述见下表：

表 3-958. 函数 timer\_internal\_clock\_config

函数名称	timer_internal_clock_config
函数原型	void timer_internal_clock_config(uint32_t timer_periph);
功能描述	TIMER配置为内部时钟模式
先决条件	-
被调用函数	-
输入参数{in}	
timer_periph	TIMER外设
TIMERx(x=0..4)	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 3-959. 函数 timer\_internal\_trigger\_as\_external\_clock\_config

函数名称	timer_internal_trigger_as_external_clock_config
函数原型	void timer_internal_trigger_as_external_clock_config(uint32_t timer_periph, uint32_t intrigger);
功能描述	配置TIMER的内部触发为时钟源
先决条件	-
被调用函数	timer_input_trigger_source_select
输入参数{in}	
timer_periph	TIMER外设
TIMERx(x=0..4)	TIMER外设选择
输入参数{in}	
intrigger	被选择的内部触发源
TIMER_SMCFG_T RGSEL_ITI0	选择内部触发0 (ITI0)为时钟源
TIMER_SMCFG_T RGSEL_ITI1	选择内部触发1 (ITI1)为时钟源
TIMER_SMCFG_T RGSEL_ITI2	选择内部触发2 (ITI2)为时钟源
TIMER_SMCFG_T RGSEL_ITI3	选择内部触发3 (ITI3)为时钟源
输出参数{out}	

-	-
返回值	
-	-

例如:

```
/* configure TIMER0 the internal trigger ITIO as external clock input */
```

```
timer_internal_trigger_as_external_clock_config(TIMER0, TIMER_SMC_CFG_TRGSEL_ITIO);
```

### 函数 timer\_external\_trigger\_as\_external\_clock\_config

函数timer\_external\_trigger\_as\_external\_clock\_config描述见下表:

表 3-960. 函数 timer\_external\_trigger\_as\_external\_clock\_config

函数名称	timer_external_trigger_as_external_clock_config
函数原型	void timer_external_trigger_as_external_clock_config(uint32_t timer_periph, uint32_t extrigger, uint16_t expolarity, uint32_t extfilter);
功能描述	配置TIMER的外部触发作为时钟源
先决条件	-
被调用函数	timer_input_trigger_source_select
输入参数{in}	
timer_periph	TIMER外设
TIMERx(x=0..4)	TIMER外设选择
输入参数{in}	
extrigger	外部触发源
TIMER_SMC_CFG_TRGSEL_CIOF_ED	CIO的边沿标志(CIOF_ED)
TIMER_SMC_CFG_TRGSEL_CIOFE0	滤波后的通道0输入(CIOFE0)
TIMER_SMC_CFG_TRGSEL_C1FE1	滤波后的通道1输入(C1FE1)
输入参数{in}	
expolarity	外部触发源极性
TIMER_IC_POLARITY_RISING	外部触发源高电平或者上升沿有效
TIMER_IC_POLARITY_FALLING	外部触发源低电平或者下降沿有效
TIMER_IC_POLARITY_BOTH_EDGE	外部触发双边沿有效
输入参数{in}	
extfilter	滤波参数 (0~15)
输出参数{out}	
-	-
返回值	

-	-
---	---

例如:

```
/* configure TIMER0 the external trigger CI0FE0 as external clock input */
```

```
timer_external_trigger_as_external_clock_config(TIMER0,  
TIMER_SMCFG_TRGSEL_CI0FE0, TIMER_IC_POLARITY_RISING, 0);
```

### 函数 timer\_external\_clock\_mode0\_config

函数timer\_external\_clock\_mode0\_config描述见下表:

表 3-961. 函数 timer\_external\_clock\_mode0\_config

函数名称	timer_external_clock_mode0_config
函数原型	void timer_external_clock_mode0_config(uint32_t timer_periph, uint32_t extprescaler, uint32_t expolarity, uint32_t extfilter);
功能描述	配置TIMER外部时钟模式0, ETI作为时钟源
先决条件	-
被调用函数	timer_external_trigger_config
输入参数{in}	
timer_periph	TIMER外设
TIMERx(x=0..4)	TIMER外设选择
输入参数{in}	
extprescaler	ETI触发源预分频值
TIMER_EXT_TRI_P SC_OFF	不分频
TIMER_EXT_TRI_P SC_DIV2	2分频
TIMER_EXT_TRI_P SC_DIV4	4分频
TIMER_EXT_TRI_P SC_DIV8	8分频
输入参数{in}	
expolarity	ETI触发源极性
TIMER_ETP_FALLI NG	下降沿或者低电平有效
TIMER_ETP_RISIN G	上升沿或者高电平有效
输入参数{in}	
extfilter	ETI触发源滤波参数 (0~15)
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* configure TIMER0 the external clock mode0 */
```

```
timer_external_clock_mode0_config(TIMER0, TIMER_EXT_TRI_PSC_DIV2,  
TIMER_ETP_FALLING, 0);
```

### 函数 timer\_external\_clock\_mode1\_config

函数timer\_external\_clock\_mode1\_config描述见下表：

表 3-962. 函数 timer\_external\_clock\_mode1\_config

函数名称	timer_external_clock_mode1_config
函数原型	void timer_external_clock_mode1_config(uint32_t timer_periph, uint32_t extprescaler, uint32_t expolarity, uint32_t extfilter);
功能描述	配置TIMER外部时钟模式1
先决条件	-
被调用函数	timer_external_trigger_config
输入参数{in}	
timer_periph	TIMER外设
TIMERx(x=0..4)	TIMER外设选择
输入参数{in}	
extprescaler	ETI触发源预分频值
TIMER_EXT_TRI_PSC_OFF	不分频
TIMER_EXT_TRI_PSC_DIV2	2分频
TIMER_EXT_TRI_PSC_DIV4	4分频
TIMER_EXT_TRI_PSC_DIV8	8分频
输入参数{in}	
expolarity	ETI触发源极性
TIMER_ETP_FALLING	下降沿或者低电平有效
TIMER_ETP_RISING	上升沿或者高电平有效
输入参数{in}	
extfilter	ETI触发源滤波参数（0~15）
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* configure TIMER0 the external clock mode1 */
```

```
timer_external_clock_mode1_config(TIMER0, TIMER_EXT_TRI_PSC_DIV2,  
TIMER_ETP_FALLING, 0);
```

### 函数 timer\_external\_clock\_mode1\_disable

函数timer\_external\_clock\_mode1\_disable描述见下表:

**表 3-963. 函数 timer\_external\_clock\_mode1\_disable**

函数名称	timer_external_clock_mode1_disable
函数原型	void timer_external_clock_mode1_disable(uint32_t timer_periph);
功能描述	TIMER外部时钟模式1禁能
先决条件	-
被调用函数	-
输入参数{in}	
timer_periph	TIMER外设
TIMERx(x=0..4)	TIMER外设选择
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* disable TIMER0 the external clock mode1 */
```

```
timer_external_clock_mode1_disable(TIMER0);
```

### 函数 timer\_write\_chxval\_register\_config

函数timer\_write\_chxval\_register\_config描述见下表:

**表 3-964. 函数 timer\_write\_chxval\_register\_config**

函数名称	timer_write_chxval_register_config
函数原型	void timer_write_chxval_register_config(uint32_t timer_periph, uint16_t ccsel);
功能描述	配置TIMER写CHxVAL选择位
先决条件	-
被调用函数	-
输入参数{in}	
timer_periph	TIMER外设
TIMERx(x=0..4, 15, 16)	TIMER外设选择
输入参数{in}	
ccsel	写CHxVAL寄存器选择位
TIMER_CHVSEL_DISABLE	无影响

<i>TIMER_CHVSEL_ENABLE</i>	当写入捕获比较寄存器的值与寄存器当前值相等时，写入操作无效。
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* configure TIMER0 write CHxVAL register selection */
```

```
timer_write_chxval_register_config(TIMER0, TIMER_CHVSEL_ENABLE);
```

### 函数 timer\_output\_value\_selection\_config

函数timer\_output\_value\_selection\_config描述见下表：

表 3-965. 函数 timer\_output\_value\_selection\_config

函数名称	timer_output_value_selection_config
函数原型	void timer_output_value_selection_config(uint32_t timer_periph, uint16_t outsel);
功能描述	配置TIMER输出值选择位
先决条件	-
被调用函数	-
输入参数{in}	
timer_periph	TIMER外设
<i>TIMERx</i> (x=0, 15, 16)	TIMER外设选择
输入参数{in}	
ccsel	输出值选择位
<i>TIMER_OUTSEL_DISABLE</i>	无影响
<i>TIMER_OUTSEL_ENABLE</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 3-966. 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TIMER的状态标志
先决条件	-
被调用函数	-
输入参数{in}	
timer_periph	TIMER外设
TIMERx	参考具体参数
输入参数{in}	
flag	状态标志
TIMER_FLAG_UP	更新标志, TIMERx(x=0..5,15,16)
TIMER_FLAG_CH0	通道0比较/捕获标志, TIMERx(x=0..4,15,16)
TIMER_FLAG_CH1	通道1比较/捕获标志, TIMERx(x=0..4)
TIMER_FLAG_CH2	通道2比较/捕获标志, TIMERx(x=0..4)
TIMER_FLAG_CH3	通道3比较/捕获标志, TIMERx(x=0..4)
TIMER_FLAG_CMT	通道换相更新标志, TIMERx(x=0,15,16)
TIMER_FLAG_TRG	触发标志, TIMERx(x=0..4)
TIMER_FLAG_BRK	中止标志位, TIMERx(x=0,15,16)
TIMER_FLAG_CH0 O	通道0捕获溢出标志, TIMERx(x=0..4,15,16)
TIMER_FLAG_CH1 O	通道1捕获溢出标志, TIMERx(x=0..4)
TIMER_FLAG_CH2 O	通道2捕获溢出标志, TIMERx(x=0..4)
TIMER_FLAG_CH3 O	通道3捕获溢出标志, TIMERx(x=0..4)
输出参数{out}	
-	-
返回值	
FlagStatus	SET或者RESET

例如:

```
/* get TIMERO0 update flags */
```

```
FlagStatus Flag_status = RESET;
```

```
Flag_status = timer_flag_get(TIMERO0, TIMER_FLAG_UP);
```

### 函数 timer\_flag\_clear

函数timer\_flag\_clear描述见下表:

表 3-967. 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TIMER状态标志
先决条件	-
被调用函数	-
输入参数{in}	
timer_periph	TIMER外设
TIMEx	参考具体参数
输入参数{in}	
flag	状态标志
TIMER_FLAG_UP	更新标志, TIMEx(x=0..5,15,16)
TIMER_FLAG_CH0	通道0比较/捕获标志, TIMEx(x=0..4,15,16)
TIMER_FLAG_CH1	通道1比较/捕获标志, TIMEx(x=0..4)
TIMER_FLAG_CH2	通道2比较/捕获标志, TIMEx(x=0..4)
TIMER_FLAG_CH3	通道3比较/捕获标志, TIMEx(x=0..4)
TIMER_FLAG_CMT	通道换相更新标志, TIMEx(x=0,15,16)
TIMER_FLAG_TRG	触发标志, TIMEx(x=0..4)
TIMER_FLAG_BRK	中止标志位, TIMEx(x=0,15,16)
TIMER_FLAG_CH0 O	通道0捕获溢出标志, TIMEx(x=0..4,15,16)
TIMER_FLAG_CH1 O	通道1捕获溢出标志, TIMEx(x=0..4)
TIMER_FLAG_CH2 O	通道2捕获溢出标志, TIMEx(x=0..4)
TIMER_FLAG_CH3 O	通道3捕获溢出标志, TIMEx(x=0..4)
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* clear TIMER0 update flags */
```

```
timer_flag_clear(TIMER0, TIMER_FLAG_UP);
```

### 函数 timer\_interrupt\_enable

函数timer\_interrupt\_enable描述见下表:

表 3-968. 函数 timer\_interrupt\_enable

函数名称	timer_interrupt_enable
函数原型	void timer_interrupt_enable(uint32_t timer_periph, uint32_t interrupt);
功能描述	外设TIMER中断使能
先决条件	-

被调用函数	-
输入参数{in}	
timer_periph	TIMER外设
TIMERx	参考具体参数
输入参数{in}	
interrupt	中断源
TIMER_INT_UP	更新中断, TIMERx(x=0..5,15,16)
TIMER_INT_CH0	通道0比较/捕获中断, TIMERx(x=0..4,15,16)
TIMER_INT_CH1	通道1比较/捕获中断, TIMERx(x=0..4)
TIMER_INT_CH2	通道2比较/捕获中断, TIMERx(x=0..4)
TIMER_INT_CH3	通道3比较/捕获中断, TIMERx(x=0..4)
TIMER_INT_CMT	换相更新中断, TIMERx(x=0,15,16)
TIMER_INT_TRG	触发中断, TIMERx(x=0..4)
TIMER_INT_BRK	中止中断, TIMERx(x=0,15,16)
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* enable the TIMER0 update interrupt */
```

```
timer_interrupt_enable (TIMER0, TIMER_INT_UP);
```

### 函数 timer\_interrupt\_disable

函数timer\_interrupt\_disable描述见下表:

表 3-969. 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TIMER中断禁能
先决条件	-
被调用函数	-
输入参数{in}	
timer_periph	TIMER外设
TIMERx	参考具体参数
输入参数{in}	
interrupt	中断源
TIMER_INT_UP	更新中断, TIMERx(x=0..5,15,16)
TIMER_INT_CH0	通道0比较/捕获中断, TIMERx(x=0..4,15,16)
TIMER_INT_CH1	通道1比较/捕获中断, TIMERx(x=0..4)
TIMER_INT_CH2	通道2比较/捕获中断, TIMERx(x=0..4)
TIMER_INT_CH3	通道3比较/捕获中断, TIMERx(x=0..4)

<i>TIMER_INT_CMT</i>	换相更新中断, TIMERx(x=0,15,16)
<i>TIMER_INT_TRG</i>	触发中断, TIMERx(x=0..4)
<i>TIMER_INT_BRK</i>	中止中断, TIMERx(x=0,15,16)
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* disable the TIMER0 update interrupt */
```

```
timer_interrupt_disable(TIMER0, TIMER_INT_UP);
```

### 函数 timer\_interrupt\_flag\_get

函数timer\_interrupt\_flag\_get描述见下表:

表 3-970. 函数 timer\_interrupt\_flag\_get

函数名称	timer_interrupt_flag_get
函数原型	FlagStatus timer_interrupt_flag_get(uint32_t timer_periph, uint32_t int_flag);
功能描述	获取外设TIMER中断标志
先决条件	-
被调用函数	-
输入参数{in}	
timer_periph	TIMER外设
TIMERx	参考具体参数
输入参数{in}	
int_flag	中断源
<i>TIMER_INT_FLAG_UP</i>	更新中断, TIMERx(x=0..5,15,16)
<i>TIMER_INT_FLAG_CH0</i>	通道0比较/捕获中断, TIMERx(x=0..4,15,16)
<i>TIMER_INT_FLAG_CH1</i>	通道1比较/捕获中断, TIMERx(x=0..4)
<i>TIMER_INT_FLAG_CH2</i>	通道2比较/捕获中断, TIMERx(x=0..4)
<i>TIMER_INT_FLAG_CH3</i>	通道3比较/捕获中断, TIMERx(x=0..4)
<i>TIMER_INT_FLAG_CMT</i>	换相更新中断, TIMERx(x=0,15,16)
<i>TIMER_INT_FLAG_TRG</i>	触发中断, TIMERx(x=0..4)
<i>TIMER_INT_FLAG_BRK</i>	中止中断, TIMERx(x=0,15,16)

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 3-971. 函数 timer\_interrupt\_flag\_clear

函数名称	timer_interrupt_flag_clear
函数原型	void timer_interrupt_flag_clear(uint32_t timer_periph, uint32_t int_flag);
功能描述	清除外设TIMER的中断标志
先决条件	-
被调用函数	-
输入参数{in}	
timer_periph	TIMER外设
TIMERx	参考具体参数
输入参数{in}	
int_flag	中断源
TIMER_INT_FLAG_UP	更新中断, TIMERx(x=0..5,15,16)
TIMER_INT_FLAG_CH0	通道0比较/捕获中断, TIMERx(x=0..4,15,16)
TIMER_INT_FLAG_CH1	通道1比较/捕获中断, TIMERx(x=0..4)
TIMER_INT_FLAG_CH2	通道2比较/捕获中断, TIMERx(x=0..4)
TIMER_INT_FLAG_CH3	通道3比较/捕获中断, TIMERx(x=0..4)
TIMER_INT_FLAG_CMT	换相更新中断, TIMERx(x=0,15,16)
TIMER_INT_FLAG_TRG	触发中断, TIMERx(x=0..4)
TIMER_INT_FLAG_BRK	中止中断, TIMERx(x=0,15,16)
输出参数{out}	
-	-

返回值	
-	-

例如：

```
/* clear TIMER0 update interrupt flag */
```

```
timer_interrupt_flag_clear(TIMER0, TIMER_INT_FLAG_UP);
```

## 3.28. TRNG

真随机数发生器模块（TRNG）能够通过连续模拟噪声生成一个32位的随机数值。TRNG寄存器列举在章节[3.28.1](#)，TRNG固件库函数介绍在章节[3.28.2](#)。

### 3.28.1. 外设寄存器说明

TRNG寄存器列表如下表所示：

表 3-972. 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 3-973. TRNG 库函数

库函数名称	库函数描述
trng_deinit	复位TRNG
trng_enable	使能TRNG接口
trng_disable	禁能TRNG接口
trng_get_true_random_data	获取真随机值
trng_interrupt_enable	使能TRNG中断
trng_interrupt_disable	禁能TRNG中断
trng_flag_get	获取TRNG状态标志
trng_interrupt_flag_get	获取TRNG中断标志
trng_interrupt_flag_clear	清除TRNG中断标志

#### 枚举 trng\_flag\_enum

表 3-974. 枚举 trng\_flag\_enum

成员名称	功能描述
TRNG_FLAG_DRDY	随机数据就绪状态

成员名称	功能描述
TRNG_FLAG_CECS	时钟错误目前状态
TRNG_FLAG_SECS	种子错误目前状态

### 枚举 trng\_int\_flag\_enum

表 3-975. 枚举 trng\_int\_flag\_enum

成员名称	功能描述
TRNG_INT_FLAG_CEIF	时钟错误中断标志
TRNG_INT_FLAG_SEIF	种子错误中断标志

### 函数 trng\_deinit

函数trng\_deinit描述见下表:

表 3-976. 函数 trng\_deinit

函数名称	trng_deinit
函数原形	void trng_deinit (void);
功能描述	复位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 3-977. 函数 trng\_enable

函数名称	trng_enable
函数原形	void trng_enable(void);
功能描述	使能TRNG接口
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 3-978. 函数 trng\_disable

函数名称	trng_disable
函数原形	void trng_disable(void);
功能描述	禁能TRNG接口
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 3-979. 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	-
输入参数{in}	
-	-
输出参数{out}	
-	-



返回值	
uint32_t	0x0 – 0xFFFFFFFF

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 3-980. 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 3-981. 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	禁能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 3-982. 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状态标志
先决条件	-
被调用函数	-
输入参数{in}	
flag	TRNG状态标志，参阅 <a href="#">表3-974. 枚举trng_flag_enum</a>
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 3-983. 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中断标志
先决条件	-
被调用函数	-
输入参数{in}	
int_flag	TRNG中断标志，参阅 <a href="#">表3-975. 枚举trng_int_flag_enum</a>
输出参数{out}	
-	-

返回值	
FlagStatus	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 3-984. 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中断标志
先决条件	-
被调用函数	-
输入参数{in}	
int_flag	TRNG中断标志，参阅 <a href="#">表3-975. 枚举trng_int_flag_enum</a>
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. TSI

触摸传感控制器（TSI）为触摸按键、滑块、电容近距感测等应用提供了简易的解决方案。章节[3.29.1](#)描述了TSI的寄存器列表，章节[3.29.2](#)对TSI库函数进行说明。

### 3.29.1. 外设寄存器描述

TSI寄存器列表如下表所示：

表 3-985. TSI 寄存器

寄存器名称	寄存器描述
TSI_CTL0	控制寄存器 0
TSI_INTEN	中断使能寄存器

寄存器名称	寄存器描述
TSI_INTC	中断标志位清除寄存器
TSI_INTF	中断标志位寄存器
TSI_PHM	引脚迟滞模式寄存器
TSI_ASW	模拟开关寄存器
TSI_SAMPCFG	采样配置寄存器
TSI_CHCFG	通道配置寄存器
TSI_GCTL	组控制寄存器
TSI_GxCYCN (x=0..2)	组 x 周期数寄存器
TSI_CTL1	控制寄存器 1

### 3.29.2. 外设库函数说明

TSI库函数列表如下表所示：

**表 3-986. TSI 库函数**

库函数名称	库函数描述
tsi_deinit	复位TSI外设
tsi_init	TSI初始化
tsi_enable	使能TSI模块
tsi_disable	禁用TSI模块
tsi_sample_pin_enable	使能TSI采样引脚
tsi_sample_pin_disable	禁用TSI采样引脚
tsi_channel_pin_enable	使能TSI通道引脚
tsi_channel_pin_disable	禁用TSI通道引脚
tsi_software_mode_config	配置TSI为软件触发模式
tsi_software_start	软件启动一次电荷转移序列
tsi_software_stop	软件停止已启动的电荷转移序列
tsi_hardware_mode_config	配置TSI为硬件触发模式
tsi_pin_mode_config	状态机为空闲状态时TSI引脚的模式
tsi_extend_charge_config	配置TSI扩展充电状态最大持续时间
tsi_plus_config	配置TSI的电荷转移状态和充电状态的持续时间
tsi_max_number_config	配置TSI电荷转移序列的最大充电、转移周期数
tsi_hysteresis_on	引脚施密特触发迟滞模式使能
tsi_hysteresis_off	引脚施密特触发迟滞模式禁用
tsi_analog_on	模拟开关闭合
tsi_analog_off	模拟开关断开
tsi_interrupt_enable	TSI中断使能
tsi_interrupt_disable	TSI中断禁止
tsi_interrupt_flag_clear	TSI中断标志位清除
tsi_interrupt_flag_get	TSI中断标志位获取
tsi_flag_clear	TSI标志位清除

库函数名称	库函数描述
tsi_flag_get	TSI标志位获取
tsi_group_enable	TSI引脚组使能
tsi_group_disable	TSI引脚组禁止
tsi_group_status_get	电荷转移完成状态获取
tsi_group0_cycle_get	电荷转移序列完成时组0执行的周期数
tsi_group1_cycle_get	电荷转移序列完成时组1执行的周期数
tsi_group2_cycle_get	电荷转移序列完成时组2执行的周期数

### 函数 tsi\_deinit

函数tsi\_deinit描述见下表：

表 3-987. 函数 tsi\_deinit

函数名称	tsi_deinit
函数原形	void tsi_deinit(void);
功能描述	复位TSI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 TSI*/
```

```
tsi_deinit();
```

### 函数 tsi\_init

函数tsi\_init描述见下表：

表 3-988. 函数 tsi\_init

函数名称	tsi_init
函数原形	void tsi_init(uint32_t prescaler,uint32_t charge_duration,uint32_t transfer_duration,uint32_t max_number);
功能描述	TSI初始化各个参数
先决条件	-
被调用函数	-
输入参数{in}	
prescaler	CTCLK时钟分频系数
TSI_CTCDIV_DIV1	CTCLK等于HCLK

<i>TSI_CTCDIV_DIV2</i>	HCLK 2分频
<i>TSI_CTCDIV_DIV4</i>	HCLK 4分频
<i>TSI_CTCDIV_DIV8</i>	HCLK 8分频
<i>TSI_CTCDIV_DIV16</i>	HCLK 16分频
<i>TSI_CTCDIV_DIV32</i>	HCLK 32分频
<i>TSI_CTCDIV_DIV64</i>	HCLK 64分频
<i>TSI_CTCDIV_DIV12</i> 8	HCLK 128分频
<i>TSI_CTCDIV_DIV25</i> 6	HCLK 256分频
<i>TSI_CTCDIV_DIV51</i> 2	HCLK 512分频
<i>TSI_CTCDIV_DIV10</i> 24	HCLK 1024分频
<i>TSI_CTCDIV_DIV20</i> 48	HCLK 2048分频
<i>TSI_CTCDIV_DIV40</i> 96	HCLK 4096分频
<i>TSI_CTCDIV_DIV81</i> 92	HCLK 8192分频
<i>TSI_CTCDIV_DIV16</i> 384	HCLK 16384分频
<i>TSI_CTCDIV_DIV32</i> 768	HCLK 32768分频
输入参数{in}	
<b>charge_duration</b>	充电状态持续时间
<i>TSI_CHARGE_1CT</i> <i>CLK(x=1..16)</i>	时间位 $X \cdot t_{CTCLK}(x=1..16)$
输入参数{in}	
<b>transfer_duration</b>	电荷转移状态持续时间
<i>TSI_TRANSFER_x</i> <i>CTCLK(x=1..16)SS_</i> <i>CLEAR</i>	时间位 $X \cdot t_{CTCLK}(x=1..16)$
输入参数{in}	
<b>max_number</b>	电荷转移序列的最大充电、转移周期数
<i>TSI_MAXNUM255</i>	255个周期
<i>TSI_MAXNUM511</i>	511个周期
<i>TSI_MAXNUM1023</i>	1023个周期
<i>TSI_MAXNUM2047</i>	2047个周期
<i>TSI_MAXNUM4095</i>	4095个周期
<i>TSI_MAXNUM8191</i>	8191个周期
<i>TSI_MAXNUM1638</i> 3	16383个周期

输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* init TSI*/
```

```
tsi_init (TSI_CTCDIV_DIV4096, TSI_CHARGE_10CTCLK, TSI_TRANSFER_8CTCLK,  
TSI_MAXNUM511);
```

### 函数 tsi\_enable

函数tsi\_enable描述见下表:

**表 3-989. 函数 cec\_enable**

函数名称	tsi_enable
函数原形	void tsi_enable (void);
功能描述	使能TSI外设
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 TSI module */
```

```
tsi_enable();
```

### 函数 tsi\_disable

函数tsi\_disable描述见下表:

**表 3-990. 函数 tsi\_disable**

函数名称	tsi_disable
函数原形	void tsi_disable (void);
功能描述	禁用TSI外设
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 TSI module */
```

```
tsi_disable ();
```

### 函数 tsi\_sample\_pin\_enable

函数tsi\_sample\_pin\_enable描述见下表：

表 3-991. 函数 tsi\_sample\_pin\_enable

函数名称	tsi_sample_pin_enable
函数原形	void tsi_sample_pin_enable(uint32_t sample);
功能描述	使能TSI采样引脚
先决条件	-
被调用函数	-
输入参数{in}	
sample	采样引脚
TSI_SAMPCFG_Gx Py( x=0..2,y=0..3)	GxPy引脚是采样引脚
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* enable G1P2 sample pin */
```

```
tsi_sample_pin_enable (TSI_SAMPCFG_G1P2);
```

### 函数 tsi\_sample\_pin\_disable

函数tsi\_sample\_pin\_disable描述见下表：

表 3-992. 函数 tsi\_sample\_pin\_disable

函数名称	tsi_sample_pin_disable
函数原形	void tsi_sample_pin_disable(uint32_t sample);
功能描述	禁用TSI采样引脚
先决条件	-
被调用函数	-
输入参数{in}	
sample	采样引脚
TSI_SAMPCFG_Gx	GxPy引脚不是采样引脚



<i>Py( x=0..2,y=0..3)</i>	
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* disable G1P2 sample pin */
```

```
tsi_sample_pin_disable (TSI_SAMPCFG_G1P2);
```

### 函数 tsi\_channel\_pin\_enable

函数tsi\_channel\_pin\_enable描述见下表：

**表 3-993. 函数 tsi\_channel\_pin\_enable**

函数名称	tsi_channel_pin_enable
函数原形	void tsi_channel_pin_enable(uint32_t channel);
功能描述	使能TSI通道引脚
先决条件	-
被调用函数	-
输入参数{in}	
channel	通道引脚
<i>TSI_CHCFG_GxPy(</i> <i>x=0..2,y=0..3)</i>	GxPy引脚是通道引脚
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* enable G1P2 channel pin */
```

```
tsi_channel_pin_enable (TSI_CHCFG_G1P2);
```

### 函数 tsi\_channel\_pin\_disable

函数tsi\_channel\_pin\_disable描述见下表：

**表 3-994. 函数 tsi\_channel\_pin\_disable**

函数名称	tsi_channel_pin_disable
函数原形	void tsi_channel_pin_disable(uint32_t channel);
功能描述	禁用TSI通道引脚
先决条件	-
被调用函数	-
输入参数{in}	

<b>channel</b>	通道引脚
<i>TSI_CHCFG_GxPy</i> ( <i>x=0..2,y=0..3</i> )	GxPy引脚不是通道引脚
<b>输出参数{out}</b>	
-	-
<b>返回值</b>	
-	-

例如：

```
/* disable G1P2 channel pin */
```

```
tsi_channel_pin_disable (TSI_CHCFG_G1P2);
```

### 函数 **tsi\_software\_mode\_config**

函数tsi\_software\_mode\_config描述见下表：

**表 3-995. 函数 tsi\_software\_mode\_config**

<b>函数名称</b>	tsi_software_mode_config
<b>函数原形</b>	void tsi_software_mode_config (void);
<b>功能描述</b>	配置TSI为软件触发模式
<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：

```
/* configure TSI triggering by software */
```

```
tsi_software_mode_config ();
```

### 函数 **tsi\_software\_start**

函数tsi\_software\_start描述见下表：

**表 3-996. 函数 tsi\_software\_start**

<b>函数名称</b>	tsi_software_start
<b>函数原形</b>	void tsi_software_start (void);
<b>功能描述</b>	软件启动一次电荷转移序列
<b>先决条件</b>	-
<b>被调用函数</b>	-
<b>输入参数{in}</b>	

-	-
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* start a charge-transfer sequence when TSI is in software trigger mode */
```

```
tsi_software_start ();
```

### 函数 tsi\_software\_stop

函数tsi\_software\_stop描述见下表：

**表 3-997. 函数 tsi\_software\_stop**

函数名称	tsi_software_stop
函数原形	void tsi_software_stop (void);
功能描述	软件停止已启动的电荷转移序列
先决条件	-
被调用函数	-
输入参数{in}	
-	-
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* stop a charge-transfer sequence when TSI is in software trigger mode */
```

```
tsi_software_stop ();
```

### 函数 tsi\_hardware\_mode\_config

函数tsi\_hardware\_mode\_config描述见下表：

**表 3-998. 函数 tsi\_hardware\_mode\_config**

函数名称	tsi_hardware_mode_config
函数原形	void tsi_hardware_mode_config(uint8_t trigger_edge);
功能描述	配置TSI为硬件触发模式
先决条件	-
被调用函数	-
输入参数{in}	
trigger_edge	触发边沿
TSI_FALLING_TRI	下降沿触发

gger	
TSI_RISING_TRIG GER	上升沿触发
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* configure TSI triggering by hardware */
```

```
tsi_hardware_mode_config (TSI_FALLING_TRIGGER);
```

### 函数 tsi\_pin\_mode\_config

函数tsi\_pin\_mode\_config描述见下表：

表 3-999. 函数 tsi\_pin\_mode\_config

函数名称	tsi_pin_mode_config
函数原形	void tsi_pin_mode_config(uint8_t pin_mode);
功能描述	状态机为空闲状态时TSI引脚的模式
先决条件	-
被调用函数	-
输入参数{in}	
pin_mode	引脚模式
TSI_OUTPUT_LOW	TSI引脚输出低电平
TSI_INPUT_FLOATING	TSI引脚保持悬空输入模式
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* configure TSI pin mode when charge-transfer sequence is IDLE */
```

```
tsi_pin_mode_config (TSI_OUTPUT_LOW);
```

### 函数 tsi\_extend\_charge\_config

函数tsi\_extend\_charge\_config描述见下表：

表 3-1000. 函数 tsi\_extend\_charge\_config

函数名称	tsi_extend_charge_config
函数原形	void tsi_extend_charge_config(ControlStatus extend,uint8_t prescaler,uint32_t max_duration);

功能描述	配置TSI扩展充电状态最大持续时间
先决条件	-
被调用函数	-
输入参数{in}	
<b>extend</b>	扩展充电状态是否使能
ENABLE	使能
DISABLE	不使能
输入参数{in}	
<b>prescaler</b>	ECCLK时钟分频系数
TSI_EXTEND_DIV1	HCLK不分频
TSI_EXTEND_DIV2	HCLK2分频
TSI_EXTEND_DIV3	HCLK3分频
TSI_EXTEND_DIV4	HCLK4分频
TSI_EXTEND_DIV5	HCLK5分频
TSI_EXTEND_DIV6	HCLK6分频
TSI_EXTEND_DIV7	HCLK7分频
TSI_EXTEND_DIV8	HCLK8分频
输入参数{in}	
<b>max_duration</b>	扩展充电状态最大持续时间
value range 1...128	时间位 $X \times t_{ECCLK}$ ( $X=1..128$ )
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* configure extend charge state */
```

```
tsi_extend_charge_config (ENABLE, TSI_EXTEND_DIV2, 10);
```

### 函数 tsi\_plus\_config

函数tsi\_plus\_config描述见下表:

表 3-1001. 函数 tsi\_plus\_config

函数名称	tsi_plus_config
函数原形	void tsi_plus_config(uint32_t prescaler,uint32_t charge_duration,uint32_t transfer_duration);
功能描述	配置TSI的电荷转移状态和充电状态的持续时间
先决条件	-
被调用函数	-
输入参数{in}	
<b>prescaler</b>	ECCLK时钟分频系数
TSI_CTCDIV_DIV1	HCLK不分频

<i>TSI_CTCDIV_DIV2</i>	HCLK2分频
<i>TSI_CTCDIV_DIV4</i>	HCLK4分频
<i>TSI_CTCDIV_DIV8</i>	HCLK8分频
<i>TSI_CTCDIV_DIV16</i>	HCLK16分频
<i>TSI_CTCDIV_DIV32</i>	HCLK32分频
<i>TSI_CTCDIV_DIV64</i>	HCLK64分频
<i>TSI_CTCDIV_DIV128</i>	HCLK128分频
<i>TSI_CTCDIV_DIV256</i>	HCLK256分频
<i>TSI_CTCDIV_DIV512</i>	HCLK512分频
<i>TSI_CTCDIV_DIV1024</i>	HCLK1024分频
<i>TSI_CTCDIV_DIV2048</i>	HCLK2048分频
<i>TSI_CTCDIV_DIV4096</i>	HCLK4096分频
<i>TSI_CTCDIV_DIV8192</i>	HCLK8192分频
<i>TSI_CTCDIV_DIV16384</i>	HCLK16384分频
<i>TSI_CTCDIV_DIV32768</i>	HCLK32768分频
输入参数{in}	
<b>charge_duration</b>	充电状态持续时间
<i>TSI_CHARGE_1CTCLK(x=1..16)</i>	时间位X*t <sub>CTCLK</sub> (x=1..16)
输入参数{in}	
<b>transfer_duration</b>	电荷转移状态持续时间
<i>TSI_TRANSFER_xCTCLK(x=1..16)</i>	时间位X*t <sub>CTCLK</sub> (x=1..16)
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* configure charge plus and transfer plus */
```

```
tsi_plus_config (TSI_CTCDIV_DIV1024, TSI_CHARGE_10CTCLK,  
TSI_TRANSFER_8CTCLK);
```

函数 **tsi\_max\_number\_config**

函数tsi\_max\_number\_config描述见下表:

表 3-1002. 函数 **tsi\_max\_number\_config**

函数名称	tsi_max_number_config
函数原形	void tsi_max_number_config(uint32_t max_number);
功能描述	配置TSI电荷转移序列的最大充电、转移周期数
先决条件	-
被调用函数	-
输入参数{in}	
<b>max_number</b>	电荷转移序列的最大充电、转移周期数
<i>TSI_MAXNUM255</i>	255个周期
<i>TSI_MAXNUM511</i>	511个周期
<i>TSI_MAXNUM1023</i>	1023个周期
<i>TSI_MAXNUM2047</i>	2047个周期
<i>TSI_MAXNUM4095</i>	4095个周期
<i>TSI_MAXNUM8191</i>	8191个周期
<i>TSI_MAXNUM16383</i>	16383个周期
3	
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* configure the max cycle number of a charge-transfer sequence */
```

```
tsi_max_number_config (TSI_MAXNUM1023);
```

函数 **tsi\_hysteresis\_on**

函数tsi\_hysteresis\_on描述见下表:

表 3-1003. 函数 **tsi\_hysteresis\_on**

函数名称	tsi_hysteresis_on
函数原形	void tsi_hysteresis_on(uint32_t group_pin);
功能描述	引脚施密特触发迟滞模式使能
先决条件	-
被调用函数	-
输入参数{in}	
<b>group_pin</b>	引脚迟滞模式使能
<i>TSI_PHM_GxPy</i> (x=0..2,y=0..3)	引脚GxPy施密特触发迟滞模式使能(x=0..2,y=0..3)
输出参数{out}	

-	-
返回值	
-	-

例如：

```
/* switch on hysteresis pin */
```

```
tsi_hysteresis_on (TSI_PHM_G1P2);
```

### 函数 tsi\_hysteresis\_off

函数tsi\_hysteresis\_off描述见下表：

**表 3-1004. 函数 tsi\_hysteresis\_off**

函数名称	tsi_hysteresis_off
函数原形	void tsi_hysteresis_off(uint32_t group_pin);
功能描述	引脚施密特触发迟滞模式禁用
先决条件	-
被调用函数	-
输入参数{in}	
group_pin	引脚迟滞模式禁用
TSI_PHM_GxPy(x=0..2,y=0..3)	引脚GxPy施密特触发迟滞模式禁用(x=0..2,y=0..3)
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* switch off hysteresis pin */
```

```
tsi_hysteresis_off (TSI_PHM_G1P2);
```

### 函数 tsi\_analog\_on

函数tsi\_analog\_on描述见下表：

**表 3-1005. 函数 tsi\_analog\_on**

函数名称	tsi_analog_on
函数原形	void tsi_analog_on(uint32_t group_pin);
功能描述	模拟开关闭合
先决条件	-
被调用函数	-
输入参数{in}	
group_pin	模拟开关状态
TSI_ASW_GxPy	引脚GxPy 的模拟开关闭合(x=0..2,y=0..3)



(x=0..2,y=0..3)	
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* switch on analog pin */
```

```
tsi_analog_on (TSI_ASW_G1P2);
```

### 函数 tsi\_analog\_off

函数tsi\_analog\_off描述见下表：

表 3-1006. 函数 tsi\_analog\_off

函数名称	tsi_analog_off
函数原形	void tsi_analog_off(uint32_t group_pin);
功能描述	模拟开关断开
先决条件	-
被调用函数	-
输入参数{in}	
group_pin	模拟开关状态
TSI_ASW_GxPy (x=0..2,y=0..3)	引脚GxPy 的模拟开关断开(x=0..2,y=0..3)
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* switch off analog pin */
```

```
tsi_analog_off (TSI_ASW_G1P2);
```

### 函数 tsi\_group\_enable

函数tsi\_group\_enable描述见下表：

表 3-1007. 函数 tsi\_group\_enable

函数名称	tsi_group_enable
函数原形	void tsi_group_enable(uint32_t group);
功能描述	TSI引脚组使能
先决条件	-
被调用函数	-
输入参数{in}	

<b>group</b>	引脚组
<i>TSI_GCTL_GEx(x=0..2)</i>	引脚组x使能
<b>输出参数{out}</b>	
-	-
<b>返回值</b>	
-	-

例如：

```
/* enable group 2 */
```

```
tsi_group_enable (TSI_GCTL_GE2);
```

### 函数 tsi\_group\_disable

函数tsi\_group\_disable描述见下表：

**表 3-1008. 函数 tsi\_group\_disable**

<b>函数名称</b>	tsi_group_disable
<b>函数原形</b>	void tsi_group_disable(uint32_t group);
<b>功能描述</b>	TSI引脚组禁用
<b>先决条件</b>	-
<b>被调用函数</b>	-
<b>输入参数{in}</b>	
<b>group</b>	引脚组
<i>TSI_GCTL_GEx(x=0..2)</i>	引脚组x禁用
<b>输出参数{out}</b>	
-	-
<b>返回值</b>	
-	-

例如：

```
/* disable group 2 */
```

```
tsi_group_disable (TSI_GCTL_GE2);
```

### 函数 tsi\_group\_status\_get

函数tsi\_group\_status\_get描述见下表：

**表 3-1009. 函数 tsi\_group\_status\_get**

<b>函数名称</b>	tsi_group_status_get
<b>函数原形</b>	FlagStatus tsi_group_status_get(uint32_t group);
<b>功能描述</b>	电荷转移完成状态获取
<b>先决条件</b>	-

被调用函数	-
输入参数{in}	
group	引脚组
TSI_GCTL_GCx(x=0..2)	组x电荷转移完成
输出参数{out}	
-	-
返回值	
FlagStatus	SET或RESET

例如:

```
/* get group complete status */
```

```
FlagStatus flag = RESET;
```

```
flag = tsi_flag_get (TSI_GCTL_GC2);
```

### 函数 tsi\_group0\_cycle\_get

函数tsi\_group0\_cycle\_get描述见下表:

表 3-1010. 函数 tsi\_group0\_cycle\_get

函数名称	tsi_group0_cycle_get
函数原形	uint16_t tsi_group0_cycle_get(void);
功能描述	电荷转移序列完成时组0执行的周期数
先决条件	-
被调用函数	-
输入参数{in}	
-	-
输出参数{out}	
-	-
返回值	
uint16_t	0-8192

例如:

```
/* get the cycle number for group0 as soon as a charge-transfer sequence completes */
```

```
uint16_t flag = 0;
```

```
flag = tsi_group0_cycle_get ();
```

### 函数 tsi\_group1\_cycle\_get

函数tsi\_group1\_cycle\_get描述见下表:

表 3-1011. 函数 tsi\_group1\_cycle\_get

函数名称	tsi_group1_cycle_get
------	----------------------

函数原形	uint16_t tsi_group1_cycle_get(void);
功能描述	电荷转移序列完成时组1执行的周期数
先决条件	-
被调用函数	-
输入参数{in}	
-	-
输出参数{out}	
-	-
返回值	
uint16_t	0-8192

例如：

```
/* get the cycle number for group1 as soon as a charge-transfer sequence completes */
uint16_t flag = 0;

flag = tsi_group1_cycle_get ();
```

### 函数 tsi\_group2\_cycle\_get

函数tsi\_group2\_cycle\_get描述见下表：

表 3-1012. 函数 tsi\_group2\_cycle\_get

函数名称	tsi_group2_cycle_get
函数原形	uint16_t tsi_group2_cycle_get(void);
功能描述	电荷转移序列完成时组2执行的周期数
先决条件	-
被调用函数	-
输入参数{in}	
-	-
输出参数{out}	
-	-
返回值	
uint16_t	0-8192

例如：

```
/* get the cycle number for group2 as soon as a charge-transfer sequence completes */
uint16_t flag = 0;

flag = tsi_group2_cycle_get ();
```

### 函数 tsi\_flag\_clear

函数tsi\_flag\_clear描述见下表：

表 3-1013. 函数 tsi\_flag\_clear

函数名称	tsi_flag_clear
函数原形	void tsi_flag_clear(uint32_t flag);
功能描述	TSI标志位清除
先决条件	-
被调用函数	-
输入参数{in}	
flag	标志位
TSI_FLAG_CTCF	电荷转移完成标志
TSI_FLAG_MNERR	最大循环次数错误标志
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* clear TSI_FLAG_CTCF_CLR flag */
tsi_flag_clear (TSI_FLAG_CTCF_CLR);
```

### 函数 tsi\_flag\_get

函数tsi\_flag\_get描述见下表：

表 3-1014. 函数 tsi\_flag\_get

函数名称	tsi_flag_get
函数原形	FlagStatus tsi_flag_get(uint32_t flag);
功能描述	TSI标志位获取
先决条件	-
被调用函数	-
输入参数{in}	
flag	标志位
TSI_FLAG_CTCF	电荷转移完成标志
TSI_FLAG_MNERR	最大循环次数错误标志
输出参数{out}	
-	-
返回值	
FlagStatus	SET或RESET

例如：

```
/* get TSI_FLAG_CTCF flag */
FlagStatus flag = RESET;
flag = tsi_flag_get (TSI_FLAG_CTCF);
```

**函数 tsi\_interrupt\_enable**

函数tsi\_interrupt\_enable描述见下表：

**表 3-1015. 函数 tsi\_interrupt\_enable**

函数名称	tsi_interrupt_enable
函数原形	void tsi_interrupt_enable(uint32_t source);
功能描述	TSI中断使能
先决条件	-
被调用函数	-
输入参数{in}	
source	中断使能位
TSI_INT_CCTCF	电荷转移完成标志中断使能
TSI_INT_MNERR	最大循环次数错误中断使能
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* enable TSI_INT_CCTCF interrupt */
tsi_interrupt_enable (TSI_INT_CCTCF);
```

**函数 tsi\_interrupt\_disable**

函数tsi\_interrupt\_disable描述见下表：

**表 3-1016. 函数 tsi\_interrupt\_disable**

函数名称	tsi_interrupt_disable
函数原形	void tsi_interrupt_disable(uint32_t source);
功能描述	TSI中断禁止
先决条件	-
被调用函数	-
输入参数{in}	
source	中断使能位
TSI_INT_CCTCF	电荷转移完成标志中断禁止
TSI_INT_MNERR	最大循环次数错误中断禁止
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* disable TSI_INT_CCTCF interrupt */
```

tsi\_interrupt\_disable (TSI\_INT\_CCTCF);

### 函数 tsi\_interrupt\_flag\_clear

函数tsi\_interrupt\_flag\_clear描述见下表:

表 3-1017. 函数 tsi\_interrupt\_flag\_clear

函数名称	tsi_interrupt_flag_clear
函数原形	void tsi_interrupt_flag_clear(uint32_t flag);
功能描述	TSI中断标志位清除
先决条件	-
被调用函数	-
输入参数{in}	
flag	中断标志位
TSI_INT_FLAG_CTCF	电荷转移完成标志中断
TSI_INT_FLAG_MNERR	最大循环次数错误标志中断
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* clear TSI_INT_FLAG_CTCF interrupt flag */
tsi_interrupt_flag_clear (TSI_INT_FLAG_CTCF);
```

### 函数 tsi\_interrupt\_flag\_get

函数tsi\_interrupt\_flag\_get描述见下表:

表 3-1018. 函数 tsi\_interrupt\_flag\_get

函数名称	tsi_interrupt_flag_get
函数原形	FlagStatus tsi_interrupt_flag_get(uint32_t flag);
功能描述	TSI中断标志位获取
先决条件	-
被调用函数	-
输入参数{in}	
flag	中断标志位
TSI_INT_FLAG_CTCF	电荷转移完成标志中断
TSI_INT_FLAG_MNERR	最大循环次数错误标志中断
输出参数{out}	

-	-
返回值	
FlagStatus	SET或RESET

例如:

```
/* get TSI_INT_FLAG_CTCF interrupt flag */
```

```
FlagStatus flag = RESET;
```

```
flag = tsi_interrupt_flag_get (TSI_INT_FLAG_CTCF);
```

### 3.30. TZPCU

TZPCU用来实现片上设备的TrustZone保护功能。章节[3.30.1](#)描述了TZPCU的寄存器列表，章节[3.30.2](#)对TZPCU库函数进行说明。

#### 3.30.1. 外设寄存器说明

寄存器列表如下表所示:

表 3-1019. TZPCU 寄存器

寄存器名称	寄存器描述
TZSPC寄存器	
TZPCU_TZSPC_CTL	TZSPC控制寄存器
TZPCU_TZSPC_SAM_CFG0	TZSPC安全访问模式配置寄存器0
TZPCU_TZSPC_SAM_CFG1	TZSPC安全访问模式配置寄存器1
TZPCU_TZSPC_SAM_CFG2	TZSPC安全访问模式配置寄存器2
TZPCU_TZSPC_PAM_CFG0	TZSPC特权访问模式配置寄存器0
TZPCU_TZSPC_PAM_CFG1	TZSPC特权访问模式配置寄存器1
TZPCU_TZSPC_PAM_CFG2	TZSPC特权访问模式配置寄存器2
TZPCU_TZSPC_TZMMPC0_NSM0	TZSPC外部存储0非安全标记寄存器0
TZPCU_TZSPC_TZMMPC0_NSM1	TZSPC外部存储0非安全标记寄存器1
TZPCU_TZSPC_TZMMPC0_NSM2	TZSPC外部存储0非安全标记寄存器2
TZPCU_TZSPC_TZMMPC0_NSM3	TZSPC外部存储0非安全标记寄存器3
TZPCU_TZSPC_TZMMPC1_NSM0	TZSPC外部存储1非安全标记寄存器0
TZPCU_TZSPC_TZMMPC1_NSM1	TZSPC外部存储1非安全标记寄存器1
TZPCU_TZSPC_DBG_CFG	TZSPC调试配置寄存器
TZBMPC寄存器	
TZPCU_TZBMPC_CTL	TZBMPC控制寄存器
TZPCU_TZBMPC_VEC	TZBMPC矢量寄存器
TZPCU_TZBMPC_LOCK0	TZBMPC锁定寄存器0
TZIAC寄存器	
TZPCU_TZIAC_INTEN0	TZIAC失能寄存器0
TZPCU_TZIAC_INTEN1	TZIAC失能寄存器1



寄存器名称	寄存器描述
TZPCU_TZIAC_INTEN2	TZIAC失能寄存器2
TZPCU_TZIAC_STAT0	TZIAC状态寄存器0
TZPCU_TZIAC_STAT1	TZIAC状态寄存器1
TZPCU_TZIAC_STAT2	TZIAC状态寄存器2
TZPCU_TZIAC_STATC0	TZIAC标志清除寄存器0
TZPCU_TZIAC_STATC1	TZIAC标志清除寄存器1
TZPCU_TZIAC_STATC2	TZIAC标志清除寄存器2

### 3.30.2. 外设库函数说明

TZPCU 库函数列表如下表所示：

**表 3-1020. TZPCU 库函数**

库函数名称	库函数描述
TZSPC函数	
tzpcu_tzspc_peripheral_attributes_config	配置外设安全属性
tzpcu_tzspc_peripheral_attributes_get	获取外设的安全属性
tzpcu_non_secure_mark_struct_parameter_init	将TZPCU非安全标记结构体初始化为默认值
tzpcu_tzspc_emnsm_config	配置外部存储器的非安全标记区域
tzpcu_tzspc_items_lock	锁定TZPCU各个项
tzpcu_tzspc_dbg_config	配置调试类型
TZBMPC函数	
tzpcu_tzbmpc_lock	锁定TZBMPC控制寄存器的子模块
tzpcu_tzbmpc_security_state_config	配置TZBMPC时钟源的默认安全状态
tzpcu_tzbmpc_secure_access_config	配置安全读写访问非安全SRAM区域的访问状态
tzpcu_tzbmpc_block_secure_access_mode_config	配置块的安全访问模式
tzpcu_tzbmpc_union_block_lock	锁定组合块的安全访问模式
TZIAC函数	
tzpcu_tziac_interrupt_enable	使能非法访问中断
tzpcu_tziac_interrupt_disable	失能非法访问中断
tzpcu_tziac_flag_get	获取非法访问中断标志
tzpcu_tziac_flag_clear	清除非法访问中断标志

#### 枚举类型 `tzpcu_mem`

**表 3-1021. 枚举类型 `tzpcu_mem`**

成员名称	功能描述
QSPI_FLASH_MEM	QSPI flash存储器
SQPI_PSRAM_MEM	SQPI PSRAM存储器

枚举类型 `tzpcu_non_secure_mark_region`表 3-1022. 枚举类型 `tzpcu_non_secure_mark_region`

成员名称	功能描述
NON_SECURE_MARK_REGION0	非安全标记区域0
NON_SECURE_MARK_REGION1	非安全标记区域1
NON_SECURE_MARK_REGION2	非安全标记区域2（仅适用于QSPI flash）
NON_SECURE_MARK_REGION3	非安全标记区域3（仅适用于QSPI flash）

结构体 `tzpcu_non_secure_mark_struct`表 3-1023. 结构体 `tzpcu_non_secure_mark_struct`

成员名称	功能描述
memory_type	存储器类型，具体存储器类型可参 <a href="#">表 3-1021. 枚举类型 <code>tzpcu_mem</code></a>
region_number	非安全标记区域编号，可参考 <a href="#">表 3-1022. 枚举类型 <code>tzpcu_non_secure_mark_region</code></a>
start_address	外部存储器非安全区域开始地址
length	外部存储器非安全区域长度

函数 `tzpcu_tzspc_peripheral_attributes_config`

函数 `tzpcu_tzspc_peripheral_attributes_config` 描述见下表：

表 3-1024. 函数 `tzpcu_tzspc_peripheral_attributes_config`

函数名称	<code>tzpcu_tzspc_peripheral_attributes_config</code>
函数原形	<code>void tzpcu_tzspc_peripheral_attributes_config(uint32_t periph, uint32_t attributes);</code>
功能描述	配置外设安全属性
先决条件	-
被调用函数	-
输入参数{in}	
periph	外设
<code>TZPCU_PERIPH_TIMER1</code>	TIMER1外设
<code>TZPCU_PERIPH_TIMER2</code>	TIMER2外设
<code>TZPCU_PERIPH_TIMER3</code>	TIMER3外设
<code>TZPCU_PERIPH_TIMER4</code>	TIMER4外设
<code>TZPCU_PERIPH_TIMER5</code>	TIMER5外设
<code>TZPCU_PERIPH_WWDGT</code>	WWDGT外设

TZPCU_PERIPH_FWD GT	FWDGT外设
TZPCU_PERIPH_SPI1	SPI1外设
TZPCU_PERIPH_USA RT1	USART1外设
TZPCU_PERIPH_USA RT2	USART2外设
TZPCU_PERIPH_I2C0	I2C0外设
TZPCU_PERIPH_I2C1	I2C1外设
TZPCU_PERIPH_USB FS	USBFS外设
TZPCU_PERIPH_TIME R0	TIMER0外设
TZPCU_PERIPH_SPI0	SPI0外设
TZPCU_PERIPH_USA RT0	USART0外设
TZPCU_PERIPH_TIME R15	TIMER15外设
TZPCU_PERIPH_TIME R16	TIMER16外设
TZPCU_PERIPH_HPD F	HPDF外设（DCI在GD32W515Tx系列设备上不支持）
TZPCU_PERIPH_CRC	CRC外设
TZPCU_PERIPH_TSI	TSI外设
TZPCU_PERIPH_ICAC HE	ICACHE外设
TZPCU_PERIPH_ADC	ADC外设
TZPCU_PERIPH_CAU	CAU外设
TZPCU_PERIPH_HAU	HAU外设
TZPCU_PERIPH_TRN G	TRNG外设
TZPCU_PERIPH_PKC AU	PKCAU外设
TZPCU_PERIPH_SDIO	SDIO外设
TZPCU_PERIPH_EFU SE	EFUSE外设
TZPCU_PERIPH_DBG	DBG外设(仅适用于特权属性)
TZPCU_PERIPH_SQPI _PSRAMREG	SQPI PSRAMREG外设
TZPCU_PERIPH_QSPI _FLASHREG	QSPI FLASHREG外设
TZPCU_PERIPH_WIFI _RF	WIFI RF外设

<i>TZPCU_PERIPH_I2S1_ADD</i>	I2S1_ADD外设
<i>TZPCU_PERIPH_DCI</i>	DCI外设（DCI在GD32W515Tx系列设备上不支持）
<i>TZPCU_PERIPH_WIFI</i>	WIFI外设
<i>TZPCU_PERIPH_ALL</i>	全部外设
输入参数{in}	
<b>attributes</b>	安全和特权属性
<i>TZPCU_SEC</i>	外设的安全属性是安全的，不能和TZPCU_NSEC同时作为输入参数
<i>TZPCU_NSEC</i>	外设的安全属性是非安全的，不能和TZPCU_NSEC同时作为输入参数
<i>TZPCU_PRIV</i>	外设的特权属性是特权级别，不能和TZPCU_NPRIV同时作为输入参数
<i>TZPCU_NPRIV</i>	外设的特权属性是特权级别，不能和TZPCU_PRIV同时作为输入参数
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* configure TIMER1 secure attributes to secure and non-privilege */
```

```
tzpcu_tzspc_peripheral_attributes_config(TZPCU_PERIPH_TIMER1, TZPCU_SEC | TZPCU_NPRIV);
```

### 函数 `tzpcu_tzspc_peripheral_attributes_get`

函数 `tzpcu_tzspc_peripheral_attributes_get` 描述见下表：

表 3-1025. 函数 `tzpcu_tzspc_peripheral_attributes_get`

函数名称	<code>tzpcu_tzspc_peripheral_attributes_get</code>
函数原形	<code>uint32_t tzpcu_tzspc_peripheral_attributes_get(uint32_t periph);</code>
功能描述	获取外设的安全属性
先决条件	-
被调用函数	-
输入参数{in}	
<b>periph</b>	外设
<i>TZPCU_PERIPH_TIMER1</i>	TIMER1外设
<i>TZPCU_PERIPH_TIMER2</i>	TIMER2外设
<i>TZPCU_PERIPH_TIMER3</i>	TIMER3外设
<i>TZPCU_PERIPH_TIMER4</i>	TIMER4外设
<i>TZPCU_PERIPH_TIMER5</i>	TIMER5外设

TZPCU_PERIPH_WW DGT	WWDGT外设
TZPCU_PERIPH_FWD GT	FWDGT外设
TZPCU_PERIPH_SPI1	SPI1外设
TZPCU_PERIPH_USA RT1	USART1外设
TZPCU_PERIPH_USA RT2	USART2外设
TZPCU_PERIPH_I2C0	I2C0外设
TZPCU_PERIPH_I2C1	I2C1外设
TZPCU_PERIPH_USB FS	USBFS外设
TZPCU_PERIPH_TIME R0	TIMER0外设
TZPCU_PERIPH_SPI0	SPI0外设
TZPCU_PERIPH_USA RT0	USART0外设
TZPCU_PERIPH_TIME R15	TIMER15外设
TZPCU_PERIPH_TIME R16	TIMER16外设
TZPCU_PERIPH_HPD F	HPDF外设（DCI在GD32W515Tx系列设备上不支持）
TZPCU_PERIPH_CRC	CRC外设
TZPCU_PERIPH_TSI	TSI外设
TZPCU_PERIPH_ICAC HE	ICACHE外设
TZPCU_PERIPH_ADC	ADC外设
TZPCU_PERIPH_CAU	CAU外设
TZPCU_PERIPH_HAU	HAU外设
TZPCU_PERIPH_TRN G	TRNG外设
TZPCU_PERIPH_PKC AU	PKCAU外设
TZPCU_PERIPH_SDIO	SDIO外设
TZPCU_PERIPH_EFU SE	EFUSE外设
TZPCU_PERIPH_DBG	DBG外设(仅适用于特权属性)
TZPCU_PERIPH_SQPI _PSRAMREG	SQPI PSRAMREG外设
TZPCU_PERIPH_QSPI _FLASHREG	QSPI FLASHREG外设

<code>TZPCU_PERIPH_WIFI_RF</code>	WIFI RF外设
<code>TZPCU_PERIPH_I2S1_ADD</code>	I2S1_ADD外设
<code>TZPCU_PERIPH_DCI</code>	DCI外设（DCI在GD32W515Tx系列设备上不支持）
<code>TZPCU_PERIPH_WIFI</code>	WIFI外设
<code>TZPCU_PERIPH_ALL</code>	全部外设
输出参数{out}	
-	-
返回值	
<code>uint32_t</code>	TZPCU_SEC/TZPCU_NSEC或TZPCU_PRIV/TZPCU_NPRIV

例如：

```
/* get TIMER1 secure attributes */
```

```
uint32_t sec_attr = tzpcu_tzspc_peripheral_attributes_get(TZPCU_PERIPH_TIMER1);
```

## 函数 `tzpcu_non_secure_mark_struct_para_init`

函数`tzpcu_non_secure_mark_struct_para_init`描述见下表：

**表 3-3. 函数 `tzpcu_non_secure_mark_struct_para_init`**

函数名称	<code>tzpcu_non_secure_mark_struct_para_init</code>
函数原形	<code>void tzpcu_non_secure_mark_struct_para_init(tzpcu_non_secure_mark_struct* init_struct);</code>
功能描述	将TZPCU非安全标记结构体初始化为默认值
先决条件	-
被调用函数	-
输入参数{in}	
<code>init_struct</code>	TZPCU非安全标记初始化结构体，结构体成员可参考 <a href="#">表3-1023. 结构体 <code>tzpcu_non_secure_mark_struct</code></a>
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* initialize the TZPCU non-secure mark struct with the default values */
```

```
tzpcu_non_secure_mark_struct init_struct;
```

```
tzpcu_non_secure_mark_struct_para_init(&init_struct);
```

**函数 tzpcu\_tzspc\_emnsm\_config**

函数tzpcu\_tzspc\_emnsm\_config描述见下表:

**表 3-1026. 函数 tzpcu\_tzspc\_emnsm\_config**

函数名称	tzpcu_tzspc_emnsm_config
函数原形	ErrStatus tzpcu_tzspc_emnsm_config(tzpcu_non_secure_mark_struct *p_non_secure_mark);
功能描述	配置外部存储器的非安全标记区域
先决条件	-
被调用函数	-
输入参数{in}	
p_non_secure_mark	TZPCU非安全标记结构体，结构体成员可参考 <a href="#">表3-1023. 结构体 tzpcu_non_secure_mark_struct</a>
输出参数{out}	
p_non_secure_mark	TZPCU非安全标记结构体，结构体成员可参考 <a href="#">表3-1023. 结构体 tzpcu_non_secure_mark_struct</a>
返回值	
ErrStatus	ERROR或SUCCESS

例如:

```
/* configure external memory QSPI FLASH first 8KB space to non-secure */
tzpcu_non_secure_mark_struct non_secure_mark;

ErrStatus status = ERROR;

tzpcu_non_secure_mark_struct_para_init(&non_secure_mark);

non_secure_mark.memory_type = QSPI_FLASH_MEM;

non_secure_mark.region_number = NON_SECURE_MARK_REGION0;

non_secure_mark.start_address = 0U;

non_secure_mark.length = 0x00002000;

status = tzpcu_tzspc_emnsm_config(&non_secure_mark);
```

**函数 tzpcu\_tzspc\_items\_lock**

函数tzpcu\_tzspc\_items\_lock描述见下表:

**表 3-1027. 函数 tzpcu\_tzspc\_items\_lock**

函数名称	tzpcu_tzspc_items_lock
函数原形	void tzpcu_tzspc_items_lock(void);
功能描述	锁定tzspc各个项
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 tzspc items */
```

```
tzpcu_tzspc_items_lock();
```

### 函数 tzpcu\_tzspc\_dbg\_config

函数tzpcu\_tzspc\_dbg\_config描述见下表：

表 3-1028. 函数 tzpcu\_tzspc\_dbg\_config

函数名称	tzpcu_tzspc_dbg_config
函数原形	void tzpcu_tzspc_dbg_config(uint32_t dbg_type, ControlStatus config_value);
功能描述	配置调试类型
先决条件	-
被调用函数	-
输入参数{in}	
dbg_type	调试类型
INVASIVE_DEBUG	侵入调试
NON_INVASIVE_DEBUG	非侵入调试
SECURE_INVASIVE_DEBUG	安全侵入调试
SECURE_NON_INVASIVE_DEBUG	安全非侵入调试
输入参数{in}	
config_value	ENABLE或DISABLE
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* disable invasive debug */
```

```
tzpcu_tzspc_dbg_config(INVASIVE_DEBUG, DISABLE);
```



函数 `tzpcu_tzbmpc_lock`

函数 `tzpcu_tzbmpc_lock` 描述见下表：

表 3-1029. 函数 `tzpcu_tzbmpc_lock`

函数名称	<code>tzpcu_tzbmpc_lock</code>
函数原形	<code>void tzpcu_tzbmpc_lock(uint32_t tzbmpx);</code>
功能描述	锁定TZBMPC控制寄存器的子模块
先决条件	-
被调用函数	-
输入参数{in}	
<b>tzbmp</b>	TZBMPC
<i>TZBMPCx</i>	TZBMPCx(x=0,1,2,3)
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* lock the control register of the TZBMPC0 sub-block */
```

```
tzpcu_tzbmpc_lock(TZBMPC0);
```

函数 `tzpcu_tzbmpc_security_state_config`

函数 `tzpcu_tzbmpc_security_state_config` 描述见下表：

表 3-1030. 函数 `tzpcu_tzbmpc_security_state_config`

函数名称	<code>tzpcu_tzbmpc_security_state_config</code>
函数原形	<code>void tzpcu_tzbmpc_security_state_config(uint32_t tzbmpx, uint32_t sec_state);</code>
功能描述	配置TZBMPC时钟源的默认安全状态
先决条件	-
被调用函数	-
输入参数{in}	
<b>tzbmp</b>	TZBMPC
<i>TZBMPCx</i>	TZBMPCx(x=0,1,2,3)
输入参数{in}	
<b>sec_state</b>	TZBMPC时钟源的默认安全状态
<i>DEFAULT_STATE</i>	默认状态，如果不存在安全区域，TZBMPC时钟源为非安全状态
<i>INVERT_STATE</i>	翻转的状态，即便不存在非安全区域，TZBMPC时钟源任然为安全状态
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* configure TZBMPC0 default security state to invert state */
```

```
tzpcu_tzbmpc_security_state_config(TZBMPC0, INVERT_STATE);
```

### 函数 tzpcu\_tzbmpc\_secure\_access\_config

函数tzpcu\_tzbmpc\_secure\_access\_config描述见下表:

表 3-1031. 函数 tzpcu\_tzbmpc\_secure\_access\_config

函数名称	tzpcu_tzbmpc_secure_access_config
函数原形	void tzpcu_tzbmpc_secure_access_config(uint32_t tzbmpx, uint32_t sec_illaccess_state);
功能描述	配置安全读写访问非安全SRAM区域的访问状态
先决条件	-
被调用函数	-
输入参数{in}	
tzbmp	TZBMPC
TZBMPCx	TZBMPCx(x=0,1,2,3)
输入参数{in}	
sec_illaccess_state	安全非法访问非安全区域状态
SECURE_ILLEGAL_ACCESS_ENABLE	安全代码读写访问非安全SRAM是非法的
SECURE_ILLEGAL_ACCESS_DISABLE	安全代码读写访问非安全SRAM是合法的
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* configure TZBMPC0 secure access state to enabled , secure read/write access non-secure SRAM is illegal */
```

```
tzpcu_tzbmpc_secure_access_config(TZBMPC0, SECURE_ILLEGAL_ACCESS_ENABLE);
```

### 函数 tzpcu\_tzbmpc\_block\_secure\_access\_mode\_config

函数tzpcu\_tzbmpc\_block\_secure\_access\_mode\_config描述见下表:

表 3-1032. 函数 tzpcu\_tzbmpc\_block\_secure\_access\_mode\_config

函数名称	tzpcu_tzbmpc_block_secure_access_mode_config
函数原形	void tzpcu_tzbmpc_block_secure_access_mode_config(uint32_t tzbmpx, uint32_t block_pos_num, uint32_t access_mode);
功能描述	配置块的安全访问模式

先决条件	-
被调用函数	-
输入参数{in}	
<b>tzbmp</b>	TZBMPC
<i>TZBMPCx</i>	TZBMPCx(x=0,1,2,3)
输入参数{in}	
<b>block_pos_num</b>	块位置编号(对于TZBMPC0和TZBMPC1块位置编号范围为0-255; 对于TZBMPC2块位置标号范围为0-511; 对于TZBMPC3块位置标号范围为0-799)
输入参数{in}	
<b>access_mode</b>	块安全访问模式
<i>BLOCK_SECURE_ACCESS_MODE_SEC</i>	块安全访问模式为安全
<i>BLOCK_SECURE_ACCESS_MODE_NSEC</i>	块安全访问模式为非安全
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* configure TZBMPC0 block 0 secure access mode to secure */
```

```
tzpcu_tzbmpc_block_secure_access_mode_config(TZBMPC0, 0,  
BLOCK_SECURE_ACCESS_MODE_SEC);
```

### 函数 tzpcu\_tzbmpc\_union\_block\_lock

函数tzpcu\_tzbmpc\_union\_block\_lock描述见下表:

表 3-1033. 函数 tzpcu\_tzbmpc\_union\_block\_lock

函数名称	tzpcu_tzbmpc_union_block_lock
函数原形	void tzpcu_tzbmpc_union_block_lock(uint32_t tzbmpx, uint32_t union_block_position_num);
功能描述	锁定组合块的安全访问模式
先决条件	-
被调用函数	-
输入参数{in}	
<b>tzbmp</b>	TZBMPC
<i>TZBMPCx</i>	TZBMPCx(x=0,1,2,3)
输入参数{in}	
<b>union_block_position_num</b>	组合块位置编号(对于TZBMPC0和TZBMPC1组合块位置编号范围为0-7; 对于TZBMPC2组合块位置编号范围为0-15; 对于TZBMPC3组合块位置编号范围为0-23)

输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* lock configure union block secure access mode of TZBMPC0 union block 0 */
```

```
tzpcu_tzmpc_union_block_lock(TZBMPC0, 0);
```

### 函数 tzpcu\_tziac\_interrupt\_enable

函数tzpcu\_tziac\_interrupt\_enable描述见下表：

**表 3-1034. 函数 tzpcu\_tziac\_interrupt\_enable**

函数名称	tzpcu_tziac_interrupt_enable
函数原形	void tzpcu_tziac_interrupt_enable(uint32_t periph);
功能描述	使能非法访问中断
先决条件	-
被调用函数	-
输入参数{in}	
periph	外设
TZPCU_PERIPH_TIME R1	TIMER1外设
TZPCU_PERIPH_TIME R2	TIMER2外设
TZPCU_PERIPH_TIME R3	TIMER3外设
TZPCU_PERIPH_TIME R4	TIMER4外设
TZPCU_PERIPH_TIME R5	TIMER5外设
TZPCU_PERIPH_WW DGT	WWDGT外设
TZPCU_PERIPH_FWD GT	FWDGT外设
TZPCU_PERIPH_SPI1	SPI1外设
TZPCU_PERIPH_USA RT1	USART1外设
TZPCU_PERIPH_USA RT2	USART2外设
TZPCU_PERIPH_I2C0	I2C0外设
TZPCU_PERIPH_I2C1	I2C1外设
TZPCU_PERIPH_USB	USBFS外设

FS	
TZPCU_PERIPH_TIMER0	TIMER0外设
TZPCU_PERIPH_SPI0	SPI0外设
TZPCU_PERIPH_USART0	USART0外设
TZPCU_PERIPH_TIMER15	TIMER15外设
TZPCU_PERIPH_TIMER16	TIMER16外设
TZPCU_PERIPH_HPDAF	HPDAF外设（DCI在GD32W515Tx系列设备上不支持）
TZPCU_PERIPH_CRC	CRC外设
TZPCU_PERIPH_TSI	TSI外设
TZPCU_PERIPH_ICACHE	ICACHE外设
TZPCU_PERIPH_ADC	ADC外设
TZPCU_PERIPH_CAU	CAU外设
TZPCU_PERIPH_HAU	HAU外设
TZPCU_PERIPH_TRNG	TRNG外设
TZPCU_PERIPH_PKCAU	PKCAU外设
TZPCU_PERIPH_SDIO	SDIO外设
TZPCU_PERIPH_RTC	RTC外设
TZPCU_PERIPH_PMU	PMU外设
TZPCU_PERIPH_SYSCFG	SYSCFG外设
TZPCU_PERIPH_DMA0	DMA0外设
TZPCU_PERIPH_DMA1	DMA1外设
TZPCU_PERIPH_RCU	RCU外设
TZPCU_PERIPH_FLASH	FLASH外设
TZPCU_PERIPH_FLASH_REG	FLASH REG外设
TZPCU_PERIPH_EXTI	EXTI外设
TZPCU_PERIPH_TZSPC	TZSPC外设
TZPCU_PERIPH_TZIAC	TZIAC外设
TZPCU_PERIPH_SRAM0	SRAM0外设

TZPCU_PERIPH_TZB MPC0_REG	ZBMPC0寄存器
TZPCU_PERIPH_SRA M1	SRAM1外设
TZPCU_PERIPH_TZB MPC1_REG	TZBMPC1寄存器
TZPCU_PERIPH_SRA M2	SRAM2外设
TZPCU_PERIPH_TZB MPC2_REG	TZBMPC2寄存器
TZPCU_PERIPH_SRA M3	SRAM3外设
TZPCU_PERIPH_TZB MPC3_REG	TZBMPC3 register
TZPCU_PERIPH_EFU SE	EFUSE外设
TZPCU_PERIPH_QSPI _PSRAM	SQPI_PSRAM外设
TZPCU_PERIPH_QSPI _FLASH	QSPI_FLASH外设
TZPCU_PERIPH_QSPI _PSRAMREG	SQPI PSRAMREG外设
TZPCU_PERIPH_QSPI _FLASHREG	QSPI FLASHREG外设
TZPCU_PERIPH_WIFI _RF	WIFI RF外设
TZPCU_PERIPH_I2S1 _ADD	I2S1_ADD外设
TZPCU_PERIPH_DCI	DCI外设（DCI在GD32W515Tx系列设备上不支持）
TZPCU_PERIPH_WIFI	WIFI外设
TZPCU_PERIPH_ALL	全部外设
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* enable TIMER1 llegal access interrupt */
```

```
tzpcu_tziac_interrupt_enable(TZPCU_PERIPH_TIMER1);
```

### 函数 tzpcu\_tziac\_interrupt\_disable

函数tzpcu\_tziac\_interrupt\_disable描述见下表：

表 3-1035. 函数 tzpcu\_tziac\_interrupt\_disable

函数名称	tzpcu_tziac_interrupt_disable
函数原形	void tzpcu_tziac_interrupt_disable (uint32_t periph);
功能描述	失能非法访问中断
先决条件	-
被调用函数	-
输入参数{in}	
periph	外设
TZPCU_PERIPH_TIME R1	TIMER1外设
TZPCU_PERIPH_TIME R2	TIMER2外设
TZPCU_PERIPH_TIME R3	TIMER3外设
TZPCU_PERIPH_TIME R4	TIMER4外设
TZPCU_PERIPH_TIME R5	TIMER5外设
TZPCU_PERIPH_WW DGT	WWDGT外设
TZPCU_PERIPH_FWD GT	FWDGT外设
TZPCU_PERIPH_SPI1	SPI1外设
TZPCU_PERIPH_USA RT1	USART1外设
TZPCU_PERIPH_USA RT2	USART2外设
TZPCU_PERIPH_I2C0	I2C0外设
TZPCU_PERIPH_I2C1	I2C1外设
TZPCU_PERIPH_USB FS	USBFS外设
TZPCU_PERIPH_TIME R0	TIMER0外设
TZPCU_PERIPH_SPI0	SPI0外设
TZPCU_PERIPH_USA RT0	USART0外设
TZPCU_PERIPH_TIME R15	TIMER15外设
TZPCU_PERIPH_TIME R16	TIMER16外设
TZPCU_PERIPH_HPD F	HPDF外设（DCI在GD32W515Tx系列设备上不支持）
TZPCU_PERIPH_CRC	CRC外设

TZPCU_PERIPH_TSI	TSI外设
TZPCU_PERIPH_ICACHE	ICACHE外设
TZPCU_PERIPH_ADC	ADC外设
TZPCU_PERIPH_CAU	CAU外设
TZPCU_PERIPH_HAU	HAU外设
TZPCU_PERIPH_TRNG	TRNG外设
TZPCU_PERIPH_PKCAU	PKCAU外设
TZPCU_PERIPH_SDIO	SDIO外设
TZPCU_PERIPH_RTC	RTC外设
TZPCU_PERIPH_PMU	PMU外设
TZPCU_PERIPH_SYSCFG	SYSCFG外设
TZPCU_PERIPH_DMA0	DMA0外设
TZPCU_PERIPH_DMA1	DMA1外设
TZPCU_PERIPH_RCU	RCU外设
TZPCU_PERIPH_FLASH	FLASH外设
TZPCU_PERIPH_FMC	FLASH REG外设
TZPCU_PERIPH_EXTI	EXTI外设
TZPCU_PERIPH_TZSPC	TZSPC外设
TZPCU_PERIPH_TZIAC	TZIAC外设
TZPCU_PERIPH_SRAM0	SRAM0外设
TZPCU_PERIPH_TZBMPC0_REG	ZBMPC0寄存器
TZPCU_PERIPH_SRAM1	SRAM1外设
TZPCU_PERIPH_TZBMPC1_REG	TZBMPC1寄存器
TZPCU_PERIPH_SRAM2	SRAM2外设
TZPCU_PERIPH_TZBMPC2_REG	TZBMPC2寄存器
TZPCU_PERIPH_SRAM3	SRAM3外设
TZPCU_PERIPH_TZBMPC3	TZBMPC3寄存器



MPC3_REG	
TZPCU_PERIPH_EFUSE	EFUSE外设
TZPCU_PERIPH_SQPI_PSRAM	SQPI_PSRAM外设
TZPCU_PERIPH_QSPI_FLASH	QSPI_FLASH外设
TZPCU_PERIPH_SQPI_PSRAMREG	SQPI PSRAMREG外设
TZPCU_PERIPH_QSPI_FLASHREG	QSPI FLASHREG外设
TZPCU_PERIPH_WIFI_RF	WIFI RF外设
TZPCU_PERIPH_I2S1_ADD	I2S1_ADD外设
TZPCU_PERIPH_DCI	DCI外设（DCI在GD32W515Tx系列设备上不支持）
TZPCU_PERIPH_WIFI	WIFI外设
TZPCU_PERIPH_ALL	全部外设
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* disable TIMER1 llegal access interrupt */
```

```
tzpcu_tziac_interrupt_disable(TZPCU_PERIPH_TIMER1);
```

### 函数 tzpcu\_tziac\_flag\_get

函数tzpcu\_tziac\_flag\_get描述见下表：

表 3-1036. 函数 tzpcu\_tziac\_flag\_get

函数名称	tzpcu_tziac_flag_get
函数原形	uint32_t tzpcu_tziac_flag_get (uint32_t periph_flag);
功能描述	获取非法访问中断标志
先决条件	-
被调用函数	-
输入参数{in}	
periph_flag	外设中断标志
TZPCU_TZIAC_FLAG_TIMER1	TIMER1外设
TZPCU_TZIAC_FLAG_TIMER2	TIMER2外设

TZPCU_TZIAC_FLAG_ TIMER3	TIMER3外设
TZPCU_TZIAC_FLAG_ TIMER4	TIMER4外设
TZPCU_TZIAC_FLAG_ TIMER5	TIMER5外设
TZPCU_TZIAC_FLAG_ WWDGT	WWDGT外设
TZPCU_TZIAC_FLAG_ FWDGT	FWDGT外设
TZPCU_TZIAC_FLAG_ SPI1	SPI1外设
TZPCU_TZIAC_FLAG_ USART1	USART1外设
TZPCU_TZIAC_FLAG_ USART2	USART2外设
TZPCU_TZIAC_FLAG_ I2C0	I2C0外设
TZPCU_TZIAC_FLAG_ I2C1	I2C1外设
TZPCU_TZIAC_FLAG_ USBFS	USBFS外设
TZPCU_TZIAC_FLAG_ TIMER0	TIMER0外设
TZPCU_TZIAC_FLAG_ SPI0	SPI0外设
TZPCU_TZIAC_FLAG_ USART0	USART0外设
TZPCU_TZIAC_FLAG_ TIMER15	TIMER15外设
TZPCU_TZIAC_FLAG_ TIMER16	TIMER16外设
TZPCU_TZIAC_FLAG_ HPDF	HPDF外设（DCI在GD32W515Tx系列设备上不支持）
TZPCU_TZIAC_FLAG_ CRC	CRC外设
TZPCU_TZIAC_FLAG_ TSI	TSI外设
TZPCU_TZIAC_FLAG_ ICACHE	ICACHE外设
TZPCU_TZIAC_FLAG_ ADC	ADC外设
TZPCU_TZIAC_FLAG_ CAU	CAU外设

CAU	
TZPCU_TZIAC_FLAG_HAU	HAU外设
TZPCU_TZIAC_FLAG_TRNG	TRNG外设
TZPCU_TZIAC_FLAG_PKCAU	PKCAU外设
TZPCU_TZIAC_FLAG_SDIO	SDIO外设
TZPCU_TZIAC_FLAG_RTC	RTC外设
TZPCU_TZIAC_FLAG_PMU	PMU外设
TZPCU_TZIAC_FLAG_SYSCFG	SYSCFG外设
TZPCU_TZIAC_FLAG_DMA0	DMA0外设
TZPCU_TZIAC_FLAG_DMA1	DMA1外设
TZPCU_TZIAC_FLAG_RCU	RCU外设
TZPCU_TZIAC_FLAG_FLASH	FLASH外设
TZPCU_TZIAC_FLAG_FMC	FLASH REG外设
TZPCU_TZIAC_FLAG_EXTI	EXTI外设
TZPCU_TZIAC_FLAG_TZSPC	TZSPC外设
TZPCU_TZIAC_FLAG_TZIAC	TZIAC外设
TZPCU_TZIAC_FLAG_SRAM0	SRAM0外设
TZPCU_TZIAC_FLAG_TZBMPC0_REG	ZBMPC0寄存器
TZPCU_TZIAC_FLAG_SRAM1	SRAM1外设
TZPCU_TZIAC_FLAG_TZBMPC1_REG	TZBMPC1寄存器
TZPCU_TZIAC_FLAG_SRAM2	SRAM2外设
TZPCU_TZIAC_FLAG_TZBMPC2_REG	TZBMPC2寄存器

<code>TZPCU_TZIAC_FLAG_</code> <code>SRAM3</code>	SRAM3外设
<code>TZPCU_TZIAC_FLAG_</code> <code>TZBMPC3_REG</code>	TZBMPC3寄存器
<code>TZPCU_TZIAC_FLAG_</code> <code>EFUSE</code>	EFUSE外设
<code>TZPCU_TZIAC_FLAG_</code> <code>SQPI_PSRAM</code>	SQPI_PSRAM外设
<code>TZPCU_TZIAC_FLAG_</code> <code>QSPI_FLASH</code>	QSPI_FLASH外设
<code>TZPCU_TZIAC_FLAG_</code> <code>SQPI_PSRAMREG</code>	SQPI PSRAMREG外设
<code>TZPCU_TZIAC_FLAG_</code> <code>QSPI_FLASHREG</code>	QSPI FLASHREG外设
<code>TZPCU_TZIAC_FLAG_</code> <code>WIFI_RF</code>	WIFI RF外设
<code>TZPCU_TZIAC_FLAG_</code> <code>I2S1_ADD</code>	I2S1_ADD外设
<code>TZPCU_TZIAC_FLAG_</code> <code>DCI</code>	DCI外设（DCI在GD32W515Tx系列设备上不支持）
<code>TZPCU_TZIAC_FLAG_</code> <code>WIFI</code>	WIFI外设
输出参数{out}	
-	-
返回值	
<code>uint32_t</code>	NO_ILLEGAL_ACCESS_PENDING或ILLEGAL_ACCESS_PENDING

例如：

```
/* get TIMER1 llegal access interrupt flag */
```

```
uint32_t tziac_flag = tzpcu_tziac_flag_get(TZPCU_TZIAC_FLAG_TIMER1);
```

### 函数 `tzpcu_tziac_flag_clear`

函数`tzpcu_tziac_flag_clear`描述见下表：

表 3-1037. 函数 `tzpcu_tziac_flag_clear`

函数名称	<code>tzpcu_tziac_flag_clear</code>
函数原形	<code>void tzpcu_tziac_flag_clear (uint32_t periph_flag);</code>
功能描述	清除非法访问中断标志
先决条件	-
被调用函数	-
输入参数{in}	
<code>periph_flag</code>	外设中断标志

TZPCU_TZIAC_FLAG_ TIMER1	TIMER1外设
TZPCU_TZIAC_FLAG_ TIMER2	TIMER2外设
TZPCU_TZIAC_FLAG_ TIMER3	TIMER3外设
TZPCU_TZIAC_FLAG_ TIMER4	TIMER4外设
TZPCU_TZIAC_FLAG_ TIMER5	TIMER5外设
TZPCU_TZIAC_FLAG_ WWDGT	WWDGT外设
TZPCU_TZIAC_FLAG_ FWDGT	FWDGT外设
TZPCU_TZIAC_FLAG_ SPI1	SPI1外设
TZPCU_TZIAC_FLAG_ USART1	USART1外设
TZPCU_TZIAC_FLAG_ USART2	USART2外设
TZPCU_TZIAC_FLAG_ I2C0	I2C0外设
TZPCU_TZIAC_FLAG_ I2C1	I2C1外设
TZPCU_TZIAC_FLAG_ USBFS	USBFS外设
TZPCU_TZIAC_FLAG_ TIMER0	TIMER0外设
TZPCU_TZIAC_FLAG_ SPI0	SPI0外设
TZPCU_TZIAC_FLAG_ USART0	USART0外设
TZPCU_TZIAC_FLAG_ TIMER15	TIMER15外设
TZPCU_TZIAC_FLAG_ TIMER16	TIMER16外设
TZPCU_TZIAC_FLAG_ HPDF	HPDF外设（DCI在GD32W515Tx系列设备上不支持）
TZPCU_TZIAC_FLAG_ CRC	CRC外设
TZPCU_TZIAC_FLAG_ TSI	TSI外设
TZPCU_TZIAC_FLAG_ ICACHE	ICACHE外设

ICACHE	
TZPCU_TZIAC_FLAG_ ADC	ADC外设
TZPCU_TZIAC_FLAG_ CAU	CAU外设
TZPCU_TZIAC_FLAG_ HAU	HAU外设
TZPCU_TZIAC_FLAG_ TRNG	TRNG外设
TZPCU_TZIAC_FLAG_ PKCAU	PKCAU外设
TZPCU_TZIAC_FLAG_ SDIO	SDIO外设
TZPCU_TZIAC_FLAG_ RTC	RTC外设
TZPCU_TZIAC_FLAG_ PMU	PMU外设
TZPCU_TZIAC_FLAG_ SYSCFG	SYSCFG外设
TZPCU_TZIAC_FLAG_ DMA0	DMA0外设
TZPCU_TZIAC_FLAG_ DMA1	DMA1外设
TZPCU_TZIAC_FLAG_ RCU	RCU外设
TZPCU_TZIAC_FLAG_ FLASH	FLASH外设
TZPCU_TZIAC_FLAG_ FMC	FLASH REG外设
TZPCU_TZIAC_FLAG_ EXTI	EXTI外设
TZPCU_TZIAC_FLAG_ TZSPC	TZSPC外设
TZPCU_TZIAC_FLAG_ TZIAC	TZIAC外设
TZPCU_TZIAC_FLAG_ SRAM0	SRAM0外设
TZPCU_TZIAC_FLAG_ TZBMPC0_REG	ZBMPC0寄存器
TZPCU_TZIAC_FLAG_ SRAM1	SRAM1外设
TZPCU_TZIAC_FLAG_ TZBMPC1_REG	TZBMPC1寄存器

TZPCU_TZIAC_FLAG_ SRAM2	SRAM2外设
TZPCU_TZIAC_FLAG_ TZBMPC2_REG	TZBMPC2寄存器
TZPCU_TZIAC_FLAG_ SRAM3	SRAM3外设
TZPCU_TZIAC_FLAG_ TZBMPC3_REG	TZBMPC3寄存器
TZPCU_TZIAC_FLAG_ EFUSE	EFUSE外设
TZPCU_TZIAC_FLAG_ SQPI_PSRAM	SQPI_PSRAM外设
TZPCU_TZIAC_FLAG_ QSPI_FLASH	QSPI_FLASH外设
TZPCU_TZIAC_FLAG_ SQPI_PSRAMREG	SQPI PSRAMREG外设
TZPCU_TZIAC_FLAG_ QSPI_FLASHREG	QSPI FLASHREG外设
TZPCU_TZIAC_FLAG_ WIFI_RF	WIFI RF外设
TZPCU_TZIAC_FLAG_ I2S1_ADD	I2S1_ADD外设
TZPCU_TZIAC_FLAG_ DCI	DCI外设（DCI在GD32W515Tx系列设备上不支持）
TZPCU_TZIAC_FLAG_ WIFI	WIFI外设
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* clear TIMER1 llegal access interrupt flag */
```

```
tzpcu_tziac_flag_clear(TZPCU_TZIAC_FLAG_TIMER1);
```

### 3.31. 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[3.31.1](#)描述了USART的寄存器列表，章节[3.31.2](#)对USART库函数进行说明。

### 3.31.1. 外设寄存器说明

USART 寄存器列表如下表所示：

表 3-1038. USART 寄存器

寄存器名称	寄存器描述
USART_CTL0	控制寄存器 0
USART_CTL1	控制寄存器 1
USART_CTL2	控制寄存器 2
USART_BAUD	波特率寄存器
USART_GP	保护时间和预分频器寄存器
USART_RT	接收超时寄存器
USART_CMD	请求寄存器
USART_STAT	状态寄存器
USART_INTC	中断标志清除寄存器
USART_RDATA	接收数据寄存器
USART_TDATA	发送数据寄存器
USART_CHC	兼容性控制寄存器
USART_RFCS	接收 FIFO 控制和状态寄存器

### 3.31.2. 外设库函数说明

USART 库函数列表如下表所示：

表 3-1039. USART 库函数

库函数名称	库函数描述
usart_deinit	复位 USART
usart_baudrate_set	配置 USART 波特率
usart_parity_config	配置 USART 奇偶校验
usart_word_length_set	配置 USART 字长
usart_stop_bit_set	配置 USART 停止位
usart_enable	使能 USART
usart_disable	失能 USART
usart_transmit_config	USART 发送配置
usart_receive_config	USART 接收配置
usart_data_first_config	配置数据传输时低位在前或高位在前
usart_invert_config	配置 USART 反转功能
usart_oversample_enable	使能 USART 溢出禁止功能
usart_oversample_disable	失能 USART 溢出禁止功能
usart_oversample_config	配置 USART 过采样模式
usart_sample_bit_config	配置 USART 过采样方法
usart_receiver_timeout_enable	使能 USART 接收超时
usart_receiver_timeout_disable	失能 USART 接收超时



库函数名称	库函数描述
usart_receiver_timeout_threshold_config	设置 USART 接收超时阈值
usart_data_transmit	USART 发送数据功能
usart_data_receive	USART 接收数据功能
usart_address_config	在地址掩码唤醒模式下配置 USART 地址
usart_address_detection_mode_config	配置 USART 地址检测模式
usart_mute_mode_enable	使能 USART 静默模式
usart_mute_mode_disable	失能 USART 静默模式
usart_mute_mode_wakeup_config	配置 USART 静默模式唤醒方式
usart_lin_mode_enable	使能 USART LIN 模式
usart_lin_mode_disable	失能 USART LIN 模式
usart_lin_break_detection_length_config	配置 USART LIN 模式中断帧长度
usart_halfduplex_enable	使能 USART 半双工模式
usart_halfduplex_disable	失能 USART 半双工模式
usart_clock_enable	使能 USART CK 引脚时钟
usart_clock_disable	失能 USART CK 引脚时钟
usart_synchronous_clock_config	配置 USART 同步通讯模式参数
usart_guard_time_config	在 USART 智能卡模式下配置保护时间值
usart_smartcard_mode_enable	使能 USART 智能卡模式
usart_smartcard_mode_disable	失能 USART 智能卡模式
usart_smartcard_mode_nack_enable	在 USART 智能卡模式下使能 NACK
usart_smartcard_mode_nack_disable	在 USART 智能卡模式下失能 NACK
usart_smartcard_mode_early_nack_enable	使能 USART 智能卡模式提前 NACK
usart_smartcard_mode_early_nack_disable	失能 USART 智能卡模式提前 NACK
usart_smartcard_autoretry_config	配置智能卡自动重试次数
usart_block_length_config	配置智能卡 T=1 的接收时块的长度
usart_irda_mode_enable	使能 USART 串行红外编解码功能模块
usart_irda_mode_disable	失能 USART 串行红外编解码功能模块
usart_prescaler_config	在 USART IrDA 低功耗模式下配置外设时钟分频系数
usart_irda_lowpower_config	配置 USART IrDA 低功耗模式
usart_hardware_flow_rts_config	配置 USART RTS 硬件控制流
usart_hardware_flow_cts_config	配置 USART CTS 硬件控制流
usart_hardware_flow_coherence_config	配置硬件流控兼容性模式
usart_rs485_driver_enable	使能 USART rs485 驱动
usart_rs485_driver_disable	失能 USART rs485 驱动
usart_driver_assertime_config	配置 USART 驱动使能置位时间

库函数名称	库函数描述
usart_driver_deasserttime_config	配置 USART 驱动使能置低时间
usart_depolarity_config	配置 USART 驱动使能极性模式
usart_dma_receive_config	配置 USART DMA 接收
usart_dma_transmit_config	配置 USART DMA 发送
usart_reception_error_dma_disable	USART 接收错误时失能 DMA
usart_reception_error_dma_enable	USART 接收错误时使能 DMA
usart_wakeup_enable	使能 USART 唤醒
usart_wakeup_disable	失能 USART 唤醒
usart_wakeup_mode_config	配置 USART 唤醒模式
usart_receive_fifo_enable	使能接收 FIFO
usart_receive_fifo_disable	失能接收 FIFO
usart_receive_fifo_counter_number	读取接收 FIFO 计数器的值
usart_command_enable	使能 USART 命令
usart_flag_get	获取 USART 状态寄存器标志位
usart_flag_clear	清除 USART 状态寄存器标志位
usart_interrupt_enable	使能 USART 中断
usart_interrupt_disable	失能 USART 中断
usart_interrupt_flag_get	获取 USART 中断标志位状态
usart_interrupt_flag_clear	清除 USART 中断标志位状态

### 枚举类型 usart\_flag\_enum

表 3-1040. 枚举类型 usart\_flag\_enum

成员名称	功能描述
USART_FLAG_REA	接收使能通知标志
USART_FLAG_TEA	发送使能通知标志
USART_FLAG_WU	从深度睡眠模式唤醒标志
USART_FLAG_RWU	接收器从静默模式唤醒
USART_FLAG_SB	断开信号发送标志
USART_FLAG_AM	地址匹配标志
USART_FLAG_BSY	忙标志
USART_FLAG_EB	块结束标志
USART_FLAG_RT	接收超时标志
USART_FLAG_CTS	CTS电平
USART_FLAG_CTSF	CTS变化标志
USART_FLAG_LBD	LIN断开检测标志
USART_FLAG_TBE	发送数据寄存器空
USART_FLAG_TC	发送完成
USART_FLAG_RBNE	读数据缓冲区非空
USART_FLAG_IDLE	空闲线检测标志
USART_FLAG_ORERR	溢出错误
USART_FLAG_NERR	噪声错误标志

成员名称	功能描述
USART_FLAG_FERR	帧错误
USART_FLAG_PERR	校验错误
USART_FLAG_EPERR	校验错误超前检测标志
USART_FLAG_RFFINT	接收FIFO满中断标志
USART_FLAG_RFF	接收FIFO满标志
USART_FLAG_RFE	接收FIFO空标志

### 枚举类型 `usart_interrupt_flag_enum`

表 3-1041. 枚举类型 `usart_interrupt_flag_enum`

成员名称	功能描述
USART_INT_FLAG_EB	块结束中断标志
USART_INT_FLAG_RT	接收超时中断标志
USART_INT_FLAG_AM	地址匹配中断标志
USART_INT_FLAG_PERR	奇偶校验错误中断标志
USART_INT_FLAG_TBE	发送寄存器空中断标志
USART_INT_FLAG_TC	发送完成中断标志
USART_INT_FLAG_RBNE	读缓冲区非空中断标志
USART_INT_FLAG_RBNE_ORE RR	读缓冲区非空和溢出中断标志
USART_INT_FLAG_IDLE	空闲线检测中断标志
USART_INT_FLAG_LBD	LIN断开检测中断标志
USART_INT_FLAG_WU	从深度睡眠模式唤醒中断标志
USART_INT_FLAG_CTS	CTS中断标志
USART_INT_FLAG_ERR_NERR	噪声错误中断标志
USART_INT_FLAG_ERR_ORER R	溢出错误中断标志
USART_INT_FLAG_ERR_FERR	帧错误中断标志
USART_INT_FLAG_RFF	接收FIFO满中断标志

### 枚举类型 `usart_interrupt_enum`

表 3-1042. 枚举类型 `usart_interrupt_enum`

成员名称	功能描述
USART_INT_EB	块结束中断使能
USART_INT_RT	接收超时中断使能
USART_INT_AM	地址匹配中断使能
USART_INT_PERR	奇偶校验错误中断使能
USART_INT_TBE	发送寄存器空中断使能
USART_INT_TC	发送完成中断使能
USART_INT_RBNE	读缓冲区非空中断和溢出错误中断使能
USART_INT_IDLE	空闲线检测中断使能

成员名称	功能描述
USART_INT_LBD	LIN断开检测中断使能
USART_INT_WU	从深度睡眠模式唤醒中断使能
USART_INT_CTS	CTS中断使能
USART_INT_ERR	错误中断使能
USART_INT_RFF	接收FIFO满中断使能

枚举类型 **usart\_invert\_enum**

表 3-1043. 枚举类型 **usart\_invert\_enum**

成员名称	功能描述
USART_DINV_ENABLE	数据位反转
USART_DINV_DISABLE	数据位不反转
USART_TXPIN_ENABLE	TX管脚电平反转
USART_TXPIN_DISABLE	TX管脚电平不反转
USART_RXPIN_ENABLE	RX管脚电平反转
USART_RXPIN_DISABLE	RX管脚电平不反转
USART_SWAP_ENABLE	交换TX/RX管脚
USART_SWAP_DISABLE	不交换TX/RX管脚

函数 **usart\_deinit**

函数usart\_deinit描述见下表：

表 3-1044. 函数 **usart\_deinit**

函数名称	usart_deinit
函数原型	void usart_deinit(uint32_t usart_periph);
功能描述	复位外设USARTx
先决条件	-
被调用函数	rcu_periph_reset_enable / rcu_periph_reset_disable
输入参数{in}	
<b>usart_periph</b>	外设USARTx
USARTx	x=0,1,2
输出参数{out}	
-	-
返回值	
-	-

例如：

```

/* reset USART0 */
usart_deinit(USART0);

```

## 函数 usart\_baudrate\_set

函数usart\_baudrate\_set描述见下表：

表 3-1045. 函数 usart\_baudrate\_set

函数名称	usart_baudrate_set
函数原型	void usart_baudrate_set(uint32_t usart_periph, uint32_t baudval);
功能描述	配置USART波特率
先决条件	-
被调用函数	rcu_clock_freq_get
输入参数{in}	
usart_periph	外设USARTx
USARTx	x=0,1,2
输入参数{in}	
baudval	波特率值
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* configure USART0 baud rate value */
usart_baudrate_set(USART0, 115200);
```

## 函数 usart\_parity\_config

函数usart\_parity\_config描述见下表：

表 3-1046. 函数 usart\_parity\_config

函数名称	usart_parity_config
函数原型	void usart_parity_config(uint32_t usart_periph, uint32_t paritycfg);
功能描述	配置USART奇偶校验
先决条件	-
被调用函数	-
输入参数{in}	
usart_periph	外设USARTx
USARTx	x=0,1,2
输入参数{in}	
paritycfg	配置USART奇偶校验
USART_PM_NONE	无校验
USART_PM_ODD	奇校验
USART_PM_EVEN	偶校验
输出参数{out}	
-	-

返回值	
-	-

例如：

```
/* configure USART0 parity */
```

```
usart_parity_config(USART0, USART_PM_EVEN);
```

### 函数 usart\_word\_length\_set

函数usart\_word\_length\_set描述见下表：

表 3-1047. 函数 usart\_word\_length\_set

函数名称	usart_word_length_set
函数原型	void usart_word_length_set(uint32_t usart_periph, uint32_t wlen);
功能描述	配置USART字长
先决条件	-
被调用函数	-
输入参数{in}	
usart_periph	外设USARTx
USARTx	x=0,1,2
输入参数{in}	
wlen	配置USART字长
USART_WL_8BIT	8 bits
USART_WL_9BIT	9 bits
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* configure USART0 word length */
```

```
usart_word_length_set(USART0, USART_WL_9BIT);
```

### 函数 usart\_stop\_bit\_set

函数usart\_stop\_bit\_set描述见下表：

表 3-1048. 函数 usart\_stop\_bit\_set

函数名称	usart_stop_bit_set
函数原型	void usart_stop_bit_set(uint32_t usart_periph, uint32_t stblen);
功能描述	配置USART停止位
先决条件	-
被调用函数	-
输入参数{in}	

<b>usart_periph</b>	外设USARTx
<i>USARTx</i>	x=0,1,2
<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
<i>USART_STB_2BIT</i>	2 bit
<i>USART_STB_1_5BIT</i> <i>T</i>	1.5 bit
<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 3-1049. 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
<i>USARTx</i>	x=0,1,2
<b>输出参数{out}</b>	
-	-
<b>返回值</b>	
-	-

例如：

```
/* enable USART0 */
usart_enable(USART0);
```

**函数 usart\_disable**

函数usart\_disable描述见下表：

**表 3-1050. 函数 usart\_disable**

函数名称	usart_disable
函数原型	void usart_disable(uint32_t usart_periph);
功能描述	失能USART
先决条件	-
被调用函数	-
输入参数{in}	
usart_periph	外设USARTx
USARTx	x=0,1,2
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* disable USART0 */
```

```
usart_disable(USART0);
```

**函数 usart\_transmit\_config**

函数usart\_transmit\_config描述见下表：

**表 3-1051. 函数 usart\_transmit\_config**

函数名称	usart_transmit_config
函数原型	void usart_transmit_config(uint32_t usart_periph, uint32_t txconfig);
功能描述	USART发送器配置
先决条件	-
被调用函数	-
输入参数{in}	
usart_periph	外设USARTx
USARTx	x=0,1,2
输入参数{in}	
txconfig	使能/失能USART发送器
USART_TRANSMIT_ENABLE	使能USART发送
USART_TRANSMIT_DISABLE	失能USART发送
输出参数{out}	
-	-
返回值	



-	-
---	---

例如：

```
/* configure USART0 transmitter */
```

```
usart_transmit_config(USART0,USART_TRANSMIT_ENABLE);
```

### 函数 usart\_receive\_config

函数usart\_receive\_config描述见下表：

表 3-1052. 函数 usart\_receive\_config

函数名称	usart_receive_config
函数原型	void usart_receive_config(uint32_t usart_periph, uint32_t rxconfig);
功能描述	USART接收器配置
先决条件	-
被调用函数	-
输入参数{in}	
usart_periph	外设USARTx
USARTx	x=0,1,2
输入参数{in}	
rxconfig	使能/失能USART接收器
USART_RECEIVE_ENABLE	使能USART接收
USART_RECEIVE_DISABLE	失能USART接收
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* configure USART0 receiver */
```

```
usart_receive_config(USART0, USART_RECEIVE_ENABLE);
```

### 函数 usart\_data\_first\_config

函数usart\_data\_first\_config描述见下表：

表 3-1053. 函数 usart\_data\_first\_config

函数名称	usart_data_first_config
函数原型	void usart_data_first_config(uint32_t usart_periph, uint32_t msbf);
功能描述	配置数据传输时低位在前或高位在前
先决条件	-
被调用函数	-

输入参数{in}	
<b>usart_periph</b>	外设USARTx
USARTx	x=0,1,2
输入参数{in}	
<b>msbf</b>	数据传输时低位在前/高位在前
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
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* configure LSB of data first */
```

```
usart_data_first_config(USART0, USART_MSBF_LSB);
```

### 函数 usart\_invert\_config

函数usart\_invert\_config描述见下表：

表 3-1054. 函数 usart\_invert\_config

函数名称	usart_invert_config
函数原型	void usart_invert_config(uint32_t usart_periph, usart_invert_enum invertpara);
功能描述	配置USART反转功能
先决条件	-
被调用函数	-
输入参数{in}	
<b>usart_periph</b>	外设USARTx
USARTx	x=0,1,2
输入参数{in}	
invertpara	参考 <a href="#">表3-1043. 枚举类型usart_invert_enum</a>
USART_DINV_ENA BLE	数据位电平反转
USART_DINV_DIS ABLE	数据位电平不反转
USART_TXPIN_EN ABLE	TX引脚电平反转
USART_TXPIN_DIS ABLE	TX引脚电平不反转
USART_RXPIN_EN ABLE	RX引脚电平反转

USART_RXPIN_DISABLE	RX引脚电平不反转
USART_SWAP_ENABLE	TX和RX管脚功能被交换
USART_SWAP_DISABLE	TX和RX管脚功能不被交换
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* configure USART0 inversion */
usart_invert_config(USART0, USART_DINV_ENABLE);
```

### 函数 usart\_oversize\_enable

函数usart\_oversize\_enable描述见下表：

表 3-1055. 函数 usart\_oversize\_enable

函数名称	usart_oversize_enable
函数原型	void usart_oversize_enable (uint32_t usart_periph);
功能描述	使能USART溢出禁止功能
先决条件	-
被调用函数	-
输入参数{in}	
usart_periph	外设USARTx
USARTx	x=0,1,2
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* enable USART0 overrun */
usart_oversize_enable (USART0);
```

### 函数 usart\_oversize\_disable

函数usart\_oversize\_disable描述见下表：

表 3-1056. 函数 usart\_oversize\_disable

函数名称	usart_oversize_disable
函数原型	void usart_oversize_disable (uint32_t usart_periph);

功能描述	失能USART溢出禁止功能
先决条件	-
被调用函数	-
输入参数{in}	
usart_periph	外设USARTx
USARTx	x=0,1,2
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* disable USART0 overrun */
usart_overrun_disable (USART0);
```

### 函数 usart\_oversample\_config

函数usart\_oversample\_config描述见下表:

表 3-1057. 函数 usart\_oversample\_config

函数名称	usart_oversample_config
函数原型	void usart_oversample_config(uint32_t usart_periph,uint32_t oversamp);
功能描述	配置USART过采样模式
先决条件	-
被调用函数	-
输入参数{in}	
usart_periph	外设USARTx
USARTx	x=0,1,2
输入参数{in}	
oversamp	过采样值
USART_OVSMOD_8	8倍过采样
USART_OVSMOD_16	16倍过采样
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* config USART0 oversampling by 8 */
usart_oversample_config(USART0,USART_OVSMOD_8);
```

## 函数 usart\_sample\_bit\_config

函数usart\_sample\_bit\_config描述见下表：

**表 3-1058. 函数 usart\_sample\_bit\_config**

函数名称	usart_sample_bit_config
函数原型	void usart_sample_bit_config(uint32_t usart_periph,uint32_t osb);
功能描述	配置USART单次采样方式
先决条件	-
被调用函数	-
输入参数{in}	
usart_periph	外设USARTx
USARTx	x=0,1,2
输入参数{in}	
osb	单次采样方式
USART_OSB_1BIT	1次采样方法
USART_OSB_3BIT	3次采样方法
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* config USART0 1 bit sample mode */
usart_sample_bit_config(USART0,USART_OSB_1BIT);
```

## 函数 usart\_receiver\_timeout\_enable

函数usart\_receiver\_timeout\_enable描述见下表：

**表 3-1059. 函数 usart\_receiver\_timeout\_enable**

函数名称	usart_receiver_timeout_enable
函数原型	void usart_receiver_timeout_enable(uint32_t usart_periph);
功能描述	使能USART接收超时
先决条件	-
被调用函数	-
输入参数{in}	
usart_periph	外设USARTx
USARTx	x=0,2
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* enable USART0 receiver timeout */

usart_receiver_timeout_enable(USART0);
```

### 函数 usart\_receiver\_timeout\_disable

函数usart\_receiver\_timeout\_disable描述见下表:

表 3-1060. 函数 usart\_receiver\_timeout\_disable

函数名称	usart_receiver_timeout_disable
函数原型	void usart_receiver_timeout_disable(uint32_t usart_periph);
功能描述	失能USART接收超时
先决条件	-
被调用函数	-
输入参数{in}	
usart_periph	外设USARTx
USARTx	x=0,2
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* disable USART0 receiver timeout */

usart_receiver_timeout_disable(USART0);
```

### 函数 usart\_receiver\_timeout\_threshold\_config

函数usart\_receiver\_timeout\_threshold\_config描述见下表:

表 3-1061. 函数 usart\_receiver\_timeout\_threshold\_config

函数名称	usart_receiver_timeout_threshold_config
函数原型	void usart_receiver_timeout_threshold_config(uint32_t usart_periph, uint32_t rtimeout);
功能描述	设置USART接收超时阈值
先决条件	-
被调用函数	-
输入参数{in}	
usart_periph	外设USARTx
USARTx	x=0,2
输入参数{in}	
rtimeout	超时时间 (0x00000000-0x00FFFFFF)

输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* set the receiver timeout threshold of USART0*/
```

```
usart_receiver_timeout_threshold_config(USART0,115200*3);
```

### 函数 usart\_data\_transmit

函数usart\_data\_transmit描述见下表:

表 3-1062. 函数 usart\_data\_transmit

函数名称	usart_data_transmit
函数原型	void usart_data_transmit(uint32_t usart_periph, uint32_t data);
功能描述	USART发送数据功能
先决条件	-
被调用函数	-
输入参数{in}	
<b>usart_periph</b>	外设USARTx
USARTx	x=0,1,2
输入参数{in}	
<b>data</b>	发送的数据
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* USART0 transmit data */
```

```
usart_data_transmit(USART0, 0xAA);
```

### 函数 usart\_data\_receive

函数usart\_data\_receive描述见下表:

表 3-1063. 函数 usart\_data\_receive

函数名称	usart_data_receive
函数原型	void usart_data_receive(uint32_t usart_periph);
功能描述	USART接收数据功能
先决条件	-
被调用函数	-
输入参数{in}	

<b>usart_periph</b>	外设USARTx
<i>USARTx</i>	x=0,1,2
输出参数{out}	
-	-
返回值	
<b>uint32_t</b>	接收到的数据

例如：

```
/* USART0 receive data */
```

```
uint16_t temp;
```

```
temp = usart_data_receive(USART0);
```

### 函数 usart\_address\_config

函数usart\_address\_config描述见下表：

表 3-1064. 函数 usart\_address\_config

函数名称	usart_address_config
函数原型	void usart_address_config(uint32_t usart_periph, uint8_t addr);
功能描述	在地址匹配唤醒模式下配置USART地址
先决条件	-
被调用函数	-
输入参数{in}	
<b>usart_periph</b>	外设USARTx
<i>USARTx</i>	x=0,1,2
输入参数{in}	
<b>addr</b>	USART地址（0-0xFF）
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* configure address of the USART0 */
```

```
usart_address_config(USART0, 0x00);
```

### 函数 usart\_address\_detection\_mode\_config

函数usart\_address\_detection\_mode\_config描述见下表：

表 3-1065. 函数 usart\_address\_detection\_mode\_config

函数名称	usart_address_detection_mode_config
函数原型	void usart_address_detection_mode_config(uint32_t usart_periph, uint32_t



	addmod);
功能描述	配置USART地址检测模式
先决条件	-
被调用函数	-
输入参数{in}	
usart_periph	外设USARTx
USARTx	x=0,1,2
输入参数{in}	
addmod	地址检测模式
USART_ADDM_4BIT	4位地址检测
USART_ADDM_FULLBIT	全位地址检测
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* configure address detection mode */
usart_address_config(USART0, USART_ADDM_4BIT);
```

### 函数 usart\_mute\_mode\_enable

函数usart\_mute\_mode\_enable描述见下表:

表 3-1066. 函数 usart\_mute\_mode\_enable

函数名称	usart_mute_mode_enable
函数原型	void usart_mute_mode_enable(uint32_t usart_periph);
功能描述	使能USART静默模式
先决条件	-
被调用函数	-
输入参数{in}	
usart_periph	外设USARTx
USARTx	x=0,1,2
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* enable USART0 receiver in mute mode */
usart_mute_mode_enable(USART0);
```

函数 `usart_mute_mode_disable`

函数 `usart_mute_mode_disable` 描述见下表：

表 3-1067. 函数 `usart_mute_mode_disable`

函数名称	<code>usart_mute_mode_disable</code>
函数原型	<code>void usart_mute_mode_disable(uint32_t usart_periph);</code>
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}	
<code>usart_periph</code>	外设USARTx
<code>USARTx</code>	x=0,1,2
-	-
返回值	
-	-

例如：

```
/* disable USART0 receiver in mute mode */
```

```
usart_mute_mode_disable(USART0);
```

函数 `usart_mute_mode_wakeup_config`

函数 `usart_mute_mode_wakeup_config` 描述见下表：

表 3-1068. 函数 `usart_mute_mode_wakeup_config`

函数名称	<code>usart_mute_mode_wakeup_config</code>
函数原型	<code>void usart_mute_mode_wakeup_config(uint32_t usart_periph, uint32_t wmethod);</code>
功能描述	配置USART静默模式唤醒方式
先决条件	-
被调用函数	-
输入参数{in}	
<code>usart_periph</code>	外设USARTx
<code>USARTx</code>	x=0,1,2
输入参数{in}	
<code>wmethod</code>	两种方法用于进入或退出静默模式
<code>USART_WM_IDLE</code>	空闲线唤醒
<code>USART_WM_ADDR</code>	地址匹配唤醒
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* configure USART0 wakeup method in mute mode */  
  
usart_mute_mode_wakeup_config(USART0, USART_WM_IDLE);
```

### 函数 usart\_lin\_mode\_enable

函数usart\_lin\_mode\_enable描述见下表：

表 3-1069. 函数 usart\_lin\_mode\_enable

函数名称	usart_lin_mode_enable
函数原型	void usart_lin_mode_enable(uint32_t usart_periph);
功能描述	使能USART LIN模式
先决条件	-
被调用函数	-
输入参数{in}	
usart_periph	外设USARTx
USARTx	x=0,2
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* USART0 LIN mode enable */  
  
usart_lin_mode_enable(USART0);
```

### 函数 usart\_lin\_mode\_disable

函数usart\_lin\_mode\_disable描述见下表：

表 3-1070. 函数 usart\_lin\_mode\_disable

函数名称	usart_lin_mode_disable
函数原型	void usart_lin_mode_disable(uint32_t usart_periph);
功能描述	失能USART LIN模式
先决条件	-
被调用函数	-
输入参数{in}	
usart_periph	外设USARTx
USARTx	x=0,2
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* USART0 LIN mode disable */
```

```
usart_lin_mode_disable(USART0);
```

### 函数 usart\_lin\_break\_dection\_length\_config

函数usart\_lin\_break\_dection\_length\_config描述见下表:

表 3-1071. 函数 usart\_lin\_break\_dection\_length\_config

函数名称	usart_lin_break_dection_length_config
函数原型	void usart_lin_break_dection_length_config(uint32_t usart_periph, uint32_t lblen);
功能描述	配置USART LIN模式中断帧长度
先决条件	-
被调用函数	-
输入参数{in}	
usart_periph	外设USARTx
USARTx	x=0,2
输入参数{in}	
lblen	LIN模式中断帧长度
USART_LBLEN_10 B	断开帧长度为10 bits
USART_LBLEN_11 B	断开帧长度为11 bits
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* configure LIN break frame length */
```

```
usart_lin_break_dection_length_config(USART0, USART_LBLEN_10B);
```

### 函数 usart\_halfduplex\_enable

函数usart\_halfduplex\_enable描述见下表:

表 3-1072. 函数 usart\_halfduplex\_enable

函数名称	usart_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}	
<b>usart_periph</b>	外设USARTx
USARTx	x=0,1,2
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* enable USART0 half duplex mode*/
```

```
usart_halfduplex_enable(USART0);
```

### 函数 usart\_halfduplex\_disable

函数usart\_halfduplex\_disable描述见下表：

表 3-1073. 函数 usart\_halfduplex\_disable

函数名称	usart_halfduplex_disable
函数原型	void usart_halfduplex_disable(uint32_t usart_periph);
功能描述	失能USART半双工模式
先决条件	-
被调用函数	-
输入参数{in}	
<b>usart_periph</b>	外设USARTx
USARTx	x=0,1,2
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* disable USART0 half duplex mode*/
```

```
usart_halfduplex_disable(USART0);
```

### 函数 usart\_clock\_enable

函数usart\_clock\_enable描述见下表：

表 3-1074. 函数 usart\_clock\_enable

函数名称	usart_clock_enable
函数原型	void usart_clock_enable(uint32_t usart_periph);
功能描述	使能USART CK引脚
先决条件	-
被调用函数	-

输入参数{in}	
<b>usart_periph</b>	外设USARTx
USARTx	x=0,2
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* enable USART0 CK pin */
```

```
usart_synchronous_clock_enable(USART0);
```

### 函数 usart\_clock\_disable

函数usart\_clock\_disable描述见下表：

表 3-1075. 函数 usart\_clock\_disable

函数名称	usart_clock_disable
函数原型	void usart_clock_disable(uint32_t usart_periph);
功能描述	失能USART CK引脚
先决条件	-
被调用函数	-
输入参数{in}	
<b>usart_periph</b>	外设USARTx
USARTx	x=0,2
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* disable USART0 CK pin */
```

```
usart_clock_disable(USART0);
```

### 函数 usart\_synchronous\_clock\_config

函数usart\_synchronous\_clock\_config描述见下表：

表 3-1076. 函数 usart\_synchronous\_clock\_config

函数名称	usart_synchronous_clock_config
函数原型	void usart_synchronous_clock_config(uint32_t usart_periph, uint32_t clen, uint32_t cph, uint32_t cpl);
功能描述	配置USART同步通讯模式参数
先决条件	-

被调用函数	-
输入参数{in}	
usart_periph	外设USARTx
USARTx	x=0,1,2
输入参数{in}	
clen	CK信号长度
USART_CLEN_NO NE	8位数据帧中有7个CK脉冲, 9位数据帧中有8个CK脉冲
USART_CLEN_EN	8位数据帧中有8个CK脉冲, 9位数据帧中有9个CK脉冲
输入参数{in}	
cph	时钟相位
USART_CPH_1CK	在首个时钟边沿采样第一个数据
USART_CPH_2CK	在第二个时钟边沿采样第一个数据
输入参数{in}	
cpl	时钟极性
USART_CPL_LOW	CK引脚不对外发送时保持为低电平
USART_CPL_HIGH	CK引脚不对外发送时保持为高电平
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* configure USART0 synchronous mode parameters */
```

```
usart_synchronous_clock_config(USART0,    USART_CLEN_EN,    USART_CPH_2CK,  
USART_CPL_HIGH);
```

### 函数 usart\_guard\_time\_config

函数usart\_guard\_time\_config描述见下表:

表 3-1077. 函数 usart\_guard\_time\_config

函数名称	usart_guard_time_config
函数原型	void usart_guard_time_config(uint32_t usart_periph,uint32_t guat);
功能描述	在USART智能卡模式下配置保护时间值
先决条件	-
被调用函数	-
输入参数{in}	
usart_periph	外设USARTx
USARTx	x=0,2
输入参数{in}	
guat	保护时间值 (0-0xFF)
输出参数{out}	

-	-
返回值	
-	-

例如：

```
/* configure USART0 guard time value in smartcard mode */
```

```
usart_guard_time_config(USART0, 0x55);
```

### 函数 usart\_smartcard\_mode\_enable

函数usart\_smartcard\_mode\_enable描述见下表：

表 3-1078. 函数 usart\_smartcard\_mode\_enable

函数名称	usart_smartcard_mode_enable
函数原型	void usart_smartcard_mode_enable(uint32_t usart_periph);
功能描述	使能USART智能卡模式
先决条件	-
被调用函数	-
输入参数{in}	
usart_periph	外设USARTx
USARTx	x=0,2
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* USART0 smartcard mode enable */
```

```
usart_smartcard_mode_enable(USART0);
```

### 函数 usart\_smartcard\_mode\_disable

函数usart\_smartcard\_mode\_disable描述见下表：

表 3-1079. 函数 usart\_smartcard\_mode\_disable

函数名称	usart_smartcard_mode_disable
函数原型	void usart_smartcard_mode_disable(uint32_t usart_periph);
功能描述	失能USART智能卡模式
先决条件	-
被调用函数	-
输入参数{in}	
usart_periph	外设USARTx
USARTx	x=0,2
输出参数{out}	



-	-
返回值	
-	-

例如：

```
/* USART0 smartcard mode disable */
```

```
usart_smartcard_mode_disable(USART0);
```

### 函数 usart\_smartcard\_mode\_nack\_enable

函数usart\_smartcard\_mode\_nack\_enable描述见下表：

**表 3-1080. 函数 usart\_smartcard\_mode\_nack\_enable**

函数名称	usart_smartcard_mode_nack_enable
函数原型	void usart_smartcard_mode_nack_enable(uint32_t usart_periph);
功能描述	在USART智能卡模式下使能NACK
先决条件	-
被调用函数	-
输入参数{in}	
usart_periph	外设USARTx
USARTx	x=0,2
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* enable USART0 NACK in smartcard mode */
```

```
usart_smartcard_mode_nack_enable(USART0);
```

### 函数 usart\_smartcard\_mode\_nack\_disable

函数usart\_smartcard\_mode\_nack\_disable描述见下表：

**表 3-1081. 函数 usart\_smartcard\_mode\_nack\_disable**

函数名称	usart_smartcard_mode_nack_disable
函数原型	void usart_smartcard_mode_nack_disable(uint32_t usart_periph);
功能描述	在USART智能卡模式下失能NACK
先决条件	-
被调用函数	-
输入参数{in}	
usart_periph	外设USARTx
USARTx	x=0,2
输出参数{out}	

-	-
返回值	
-	-

例如：

```
/* disable USART0 NACK in smartcard mode */
```

```
usart_smartcard_mode_nack_disable(USART0);
```

### 函数 usart\_smartcard\_mode\_early\_nack\_enable

函数usart\_smartcard\_mode\_early\_nack\_enable描述见下表：

**表 3-1082. 函数 usart\_smartcard\_mode\_early\_nack\_enable**

函数名称	usart_smartcard_mode_early_nack_enable
函数原型	void usart_smartcard_mode_early_nack_enable (uint32_t usart_periph);
功能描述	使能USART智能卡模式提前NACK
先决条件	-
被调用函数	-
输入参数{in}	
usart_periph	外设USARTx
USARTx	x=0,2
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* enable USART0 early NACK in smartcard mode */
```

```
usart_smartcard_mode_early_nack_enable(USART0);
```

### 函数 usart\_smartcard\_mode\_early\_nack\_disable

函数usart\_smartcard\_mode\_early\_nack\_disable描述见下表：

**表 3-1083. 函数 usart\_smartcard\_mode\_early\_nack\_disable**

函数名称	usart_smartcard_mode_early_nack_disable
函数原型	void usart_smartcard_mode_early_nack_disable(uint32_t usart_periph);
功能描述	失能USART智能卡模式提前NACK
先决条件	-
被调用函数	-
输入参数{in}	
usart_periph	外设USARTx
USARTx	x=0,2
输出参数{out}	

-	-
返回值	
-	-

例如:

```
/* disable USART0 early NACK in smartcard mode */
```

```
usart_smartcard_mode_early_nack_disable(USART0);
```

### 函数 usart\_smartcard\_autoretry\_config

函数usart\_smartcard\_autoretry\_config描述见下表:

表 3-1084. 函数 usart\_smartcard\_autoretry\_config

函数名称	usart_smartcard_autoretry_config
函数原型	void usart_smartcard_autoretry_config(uint32_t usart_periph, uint32_t scrtnum);
功能描述	配置智能卡自动重试次数
先决条件	-
被调用函数	-
输入参数{in}	
usart_periph	外设USARTx
USARTx	x=0,2
输入参数{in}	
scrtnum	智能卡自动重试次数 (0-0x00000007)
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* configure smartcard auto-retry number */
```

```
usart_smartcard_autoretry_config (USART0, 0x00000007);
```

### 函数 usart\_block\_length\_config

函数usart\_block\_length\_config描述见下表:

表 3-1085. 函数 usart\_block\_length\_config

函数名称	usart_block_length_config
函数原型	void usart_block_length_config(uint32_t usart_periph, uint32_t bl);
功能描述	配置智能卡T=1的接收时块的长度
先决条件	-
被调用函数	-
输入参数{in}	

<b>usart_periph</b>	外设USARTx
<i>USARTx</i>	x=0,2
输入参数{in}	
<b>bl</b>	块长度（0x00-0xFF）
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* configure block length in Smartcard T=1 reception */
```

```
usart_block_length_config(USART0, 0x000000FF);
```

### 函数 usart\_irda\_mode\_enable

函数usart\_irda\_mode\_enable描述见下表：

表 3-1086. 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}	
<b>usart_periph</b>	外设USARTx
<i>USARTx</i>	x=0,2
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* enable USART0 IrDA mode */
```

```
usart_irda_mode_enable(USART0);
```

### 函数 usart\_irda\_mode\_disable

函数usart\_irda\_mode\_disable描述见下表：

表 3-1087. 函数 usart\_irda\_mode\_disable

函数名称	usart_irda_mode_disable
函数原型	void usart_irda_mode_disable(uint32_t usart_periph);
功能描述	失能USART串行红外编解码功能模块
先决条件	-

被调用函数	-
输入参数{in}	
usart_periph	外设USARTx
USARTx	x=0,2
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* disable USART0 IrDA mode */
```

```
usart_irda_mode_disable(USART0);
```

### 函数 usart\_prescaler\_config

函数usart\_prescaler\_config描述见下表：

表 3-1088. 函数 usart\_prescaler\_config

函数名称	usart_prescaler_config
函数原型	void usart_prescaler_config(uint32_t usart_periph, uint32_t psc);
功能描述	在USART IrDA低功耗模式下配置外设时钟分频系数
先决条件	-
被调用函数	-
输入参数{in}	
usart_periph	外设USARTx
USARTx	x=0,2
输入参数{in}	
psc	时钟分频系数（0x00-0xFF）
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* configure the USART0 peripheral clock prescaler in USART IrDA low-power mode */
```

```
usart_prescaler_config(USART0, 0x00);
```

### 函数 usart\_irda\_lowpower\_config

函数usart\_irda\_lowpower\_config描述见下表：

表 3-1089. 函数 usart\_irda\_lowpower\_config

函数名称	usart_irda_lowpower_config
------	----------------------------

函数原型	void usart_irda_lowpower_config(uint32_t usart_periph, uint32_t irlp);
功能描述	配置USART IrDA低功耗模式
先决条件	-
被调用函数	-
输入参数{in}	
usart_periph	外设USARTx
USARTx	x=0,2
输入参数{in}	
irlp	IrDA低功耗模式或正常模式
USART_IRLP_LOW	低功耗模式
USART_IRLP_NORMAL	正常模式
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* configure USART0 IrDA low-power */
```

```
usart_irda_lowpower_config(USART0, USART_IRLP_LOW);
```

### 函数 usart\_hardware\_flow\_rts\_config

函数usart\_hardware\_flow\_rts\_config描述见下表:

表 3-1090. 函数 usart\_hardware\_flow\_rts\_config

函数名称	usart_hardware_flow_rts_config
函数原型	void usart_hardware_flow_rts_config(uint32_t usart_periph, uint32_t rtsconfig);
功能描述	配置USART RTS硬件控制流
先决条件	-
被调用函数	-
输入参数{in}	
usart_periph	外设USARTx
USARTx	x=0,1,2
输入参数{in}	
rtsconfig	使能/失能RTS

<i>USART_RTS_ENA</i> <i>BLE</i>	使能RTS
<i>USART_RTS_DISA</i> <i>BLE</i>	失能RTS
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* configure USART0 hardware flow control RTS */
```

```
usart_hardware_flow_rts_config(USART0, USART_RTS_ENABLE);
```

### 函数 usart\_hardware\_flow\_cts\_config

函数usart\_hardware\_flow\_cts\_config描述见下表：

表 3-1091. 函数 usart\_hardware\_flow\_cts\_config

函数名称	usart_hardware_flow_cts_config
函数原型	void usart_hardware_flow_cts_config(uint32_t usart_periph, uint32_t ctsconfig);
功能描述	配置USART CTS硬件控制流
先决条件	-
被调用函数	-
输入参数{in}	
<b>usart_periph</b>	外设USARTx
<i>USARTx</i>	x=0,1,2
输入参数{in}	
<b>ctsconfig</b>	使能/失能CTS
<i>USART_CTS_ENA</i> <i>BLE</i>	使能CTS
<i>USART_CTS_DISA</i> <i>BLE</i>	失能CTS
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* configure USART0 hardware flow control CTS */
```

```
usart_hardware_flow_cts_config(USART0, USART_CTS_ENABLE);
```

**函数 usart hardware\_flow\_coherence\_config**

函数usart hardware\_flow\_coherence\_config描述见下表：

**表 3-1092. 函数 usart hardware\_flow\_coherence\_config**

函数名称	usart hardware_flow_coherence_config
函数原型	void usart hardware_flow_coherence_config(uint32_t usart_periph, uint32_t hcm);
功能描述	配置硬件流控兼容模式
先决条件	-
被调用函数	-
输入参数{in}	
usart_periph	外设USARTx
USARTx	x=0,1,2
输入参数{in}	
hcm	硬件流控制兼容模式
USART_HCM_NONE	nRTS信号与USART_STAT0寄存器中RBNE位相同
USART_HCM_EN	nRTS信号在最后一个数据位被采样后被置位
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* configure hardware flow control coherence mode */
```

```
usart hardware_flow_coherence_config(USART0, USART_HCM_NONE);
```

**函数 usart\_rs485\_driver\_enable**

函数usart\_rs485\_driver\_enable描述见下表：

**表 3-1093. 函数 usart\_rs485\_driver\_enable**

函数名称	usart_rs485_driver_enable
函数原型	void usart_rs485_driver_enable (uint32_t usart_periph);
功能描述	使能USART rs485驱动
先决条件	-
被调用函数	-
输入参数{in}	
usart_periph	外设USARTx
USARTx	x=0,1,2
输出参数{out}	
-	-
返回值	



-	-
---	---

例如：

```
/* enable USART0 RS485 driver */

usart_rs485_driver_enable(USART0);
```

### 函数 usart\_rs485\_driver\_disable

函数usart\_rs485\_driver\_disable描述见下表：

表 3-1094. 函数 usart\_rs485\_driver\_disable

函数名称	usart_rs485_driver_disable
函数原型	void usart_rs485_driver_disable(uint32_t usart_periph);
功能描述	失能USART rs485驱动
先决条件	-
被调用函数	-
输入参数{in}	
usart_periph	外设USARTx
USARTx	x=0,1,2
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* disable USART0 RS485 driver */

usart_rs485_driver_disable(USART0);
```

### 函数 usart\_driver\_assertime\_config

函数usart\_driver\_assertime\_config描述见下表：

表 3-1095. 函数 usart\_driver\_assertime\_config

函数名称	usart_driver_assertime_config
函数原型	void usart_driver_assertime_config(uint32_t usart_periph, uint32_t deatime);
功能描述	配置USART驱动使能置位时间
先决条件	-
被调用函数	-
输入参数{in}	
usart_periph	外设USARTx
USARTx	x=0,1,2
输入参数{in}	
deatime	驱动使能置位时间（00xx-0x1F）
输出参数{out}	

-	-
返回值	
-	-

例如：

```
/* set USART0 driver assertime */
```

```
usart_driver_assertime_config(USART0, 0x1F);
```

### 函数 usart\_driver\_deassertime\_config

函数usart\_driver\_deassertime\_config描述见下表：

表 3-1096. 函数 usart\_driver\_deassertime\_config

函数名称	usart_driver_deassertime_config
函数原型	void usart_driver_deassertime_config(uint32_t usart_periph, uint32_t dedtime);
功能描述	配置USART驱动使能置低时间
先决条件	-
被调用函数	-
输入参数{in}	
usart_periph	外设USARTx
USARTx	x=0,1,2
输入参数{in}	
dedtime	驱动使能置低时间（0x00-0x1F）
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* set USART0 driver deassertime */
```

```
usart_driver_deassertime_config(USART0, 0x1F);
```

### 函数 usart\_depolarity\_config

函数usart\_depolarity\_config描述见下表：

表 3-1097. 函数 usart\_depolarity\_config

函数名称	usart_depolarity_config
函数原型	void usart_depolarity_config(uint32_t usart_periph, uint32_t dep);
功能描述	配置USART驱动使能极性模式
先决条件	-
被调用函数	-
输入参数{in}	
usart_periph	外设USARTx

USARTx	x=0,1,2
输入参数{in}	
dep	驱动使能的极性选择模式
USART_DEP_HIGH	DE信号高有效
USART_DEP_LOW	DE信号低有效
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* configure driver enable polarity mode */
```

```
usart_driver_depolarity_config(USART0, USART_DEP_HIGH);
```

### 函数 usart\_dma\_receive\_config

函数usart\_dma\_enable描述见下表:

表 3-1098. 函数 usart\_dma\_receive\_config

函数名称	usart_dma_receive_configusart_dma_enable
函数原型	void usart_dma_receive_config(uint32_t usart_periph, uint8_t dmacmd);
功能描述	配置USART DMA接收
先决条件	-
被调用函数	-
输入参数{in}	
usart_periph	外设USARTx
USARTx	x=0,1,2
输入参数{in}	
dmacmd	USART DMA模式
USART_RECEIVE_DMA_ENABLE	使能USART DMA接收
USART_RECEIVE_DMA_DISABLE	失能USART DMA接收
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* configure USART DMA reception */
```

```
usart_dma_receive_config(USART0, USART_RECEIVE_DMA_ENABLE);
```

## 函数 usart\_dma\_transmit\_config

函数usart\_dma\_transmit\_config描述见下表：

**表 3-1099. 函数 usart\_dma\_transmit\_config**

函数名称	usart_dma_transmit_config
函数原型	void usart_dma_transmit_config(uint32_t usart_periph, uint8_t dmacmd);
功能描述	失能USART DMA发送或接收
先决条件	-
被调用函数	-
输入参数{in}	
usart_periph	外设USARTx
USARTx	x=0,1,2
输入参数{in}	
dmacmd	USART DMA模式
USART_TRANSMIT_DMA_ENABLE	使能USART DMA发送
USART_TRANSMIT_DMA_DISABLE	失能USART DMA发送
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* configure USART DMA transmission */
```

```
usart_dma_transmit_config(USART0, USART_TRANSMIT_DMA_ENABLE);
```

## 函数 usart\_reception\_error\_dma\_disable

函数usart\_reception\_error\_dma\_disable描述见下表：

**表 3-1100. 函数 usart\_reception\_error\_dma\_disable**

函数名称	usart_reception_error_dma_disable
函数原型	void usart_reception_error_dma_disable (uint32_t usart_periph);
功能描述	USART接收错误时失能DMA
先决条件	-
被调用函数	-
输入参数{in}	
usart_periph	外设USARTx
USARTx	x=0,1,2
输出参数{out}	
-	-
返回值	

-	-
---	---

例如：

```
/* disable DMA on reception error */
```

```
usart_reception_error_dma_disable(USART0);
```

### 函数 usart\_reception\_error\_dma\_enable

函数usart\_reception\_error\_dma\_enable描述见下表：

表 3-1101. 函数 usart\_reception\_error\_dma\_enable

函数名称	usart_reception_error_dma_enable
函数原型	void usart_reception_error_dma_enable(uint32_t usart_periph);
功能描述	USART接收错误时使能DMA
先决条件	-
被调用函数	-
输入参数{in}	
usart_periph	外设USARTx
USARTx	x=0,1,2
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* enable DMA on reception error */
```

```
usart_reception_error_dma_enable(USART0);
```

### 函数 usart\_wakeup\_enable

函数usart\_wakeup\_enable描述见下表：

表 3-1102. 函数 usart\_wakeup\_enable

函数名称	usart_wakeup_enable
函数原型	void usart_wakeup_enable(uint32_t usart_periph);
功能描述	使能USART唤醒
先决条件	-
被调用函数	-
输入参数{in}	
usart_periph	外设USARTx
USARTx	x=0,2
输出参数{out}	
-	-
返回值	

-	-
---	---

例如:

```
/* USART0 wake up enable */
usart_wakeup_enable(USART0);
```

### 函数 usart\_wakeup\_disable

函数usart\_wakeup\_disable描述见下表:

表 3-1103. 函数 usart\_wakeup\_disable

函数名称	usart_wakeup_disable
函数原型	void usart_wakeup_disable(uint32_t usart_periph);
功能描述	失能USART唤醒
先决条件	-
被调用函数	-
输入参数{in}	
usart_periph	外设USARTx
USARTx	x=0,2
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* USART0 wake up disable */
usart_wakeup_disable(USART0);
```

### 函数 usart\_wakeup\_mode\_config

函数usart\_wakeup\_mode\_config描述见下表:

表 3-1104. 函数 usart\_wakeup\_mode\_config

函数名称	usart_wakeup_mode_config
函数原型	void usart_wakeup_mode_config(uint32_t usart_periph, uint32_t wum);
功能描述	配置USART唤醒模式
先决条件	-
被调用函数	-
输入参数{in}	
usart_periph	外设USARTx
USARTx	x=0,2
输入参数{in}	
wum	唤醒模式
USART_WUM_ADD	WUF在地址匹配时置位

<i>R</i>	
<i>USART_WUM_STA</i> <i>RTB</i>	WUF在检测到起始位时置位
<i>USART_WUM_RBN</i> <i>E</i>	WUF在检测到RBNE时置位
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* configure USART0 wake up mode */
usart_wakeup_mode_config(USART0, USART_WUM_ADDR);
```

### 函数 usart\_receive\_fifo\_enable

函数usart\_receive\_fifo\_enable描述见下表：

表 3-1105. 函数 usart\_receive\_fifo\_enable

函数名称	usart_receive_fifo_enable
函数原型	void usart_receive_fifo_enable(uint32_t usart_periph);
功能描述	使能接收FIFO
先决条件	-
被调用函数	-
输入参数{in}	
<b>usart_periph</b>	外设USARTx
<i>USARTx</i>	x=0,1,2
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* enable receive FIFO */
usart_receive_fifo_enable (USART0);
```

### 函数 usart\_receive\_fifo\_disable

函数usart\_receive\_fifo\_disable描述见下表：

表 3-1106. 函数 usart\_receive\_fifo\_disable

函数名称	usart_receive_fifo_disable
函数原型	void usart_receive_fifo_disable(uint32_t usart_periph);
功能描述	失能接收FIFO

先决条件	-
被调用函数	-
输入参数{in}	
usart_periph	外设USARTx
USARTx	x=0,1,2
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* disable receive FIFO */
usart_receive_fifo_disable(USART0);
```

### 函数 usart\_receive\_fifo\_counter\_number

函数usart\_receive\_fifo\_counter\_number描述见下表：

表 3-1107. 函数 usart\_receive\_fifo\_counter\_number

函数名称	usart_receive_fifo_counter_number
函数原型	uint8_t usart_receive_fifo_counter_number(uint32_t usart_periph);
功能描述	读取接收FIFO计数器的值
先决条件	-
被调用函数	-
输入参数{in}	
usart_periph	外设USARTx
USARTx	x=0,1,2
输出参数{out}	
-	-
返回值	
uint8_t	接收FIFO计数器的值

例如：

```
/* read receive FIFO counter number */
uint8_t temp;
temp = usart_receive_fifo_counter_number(USART0);
```

### 函数 usart\_command\_enable

函数usart\_command\_enable描述见下表：

表 3-1108. 函数 usart\_command\_enable

函数名称	usart_command_enable
------	----------------------



函数原型	void usart_command_enable(uint32_t usart_periph, uint32_t cmdtype);
功能描述	使能USART请求
先决条件	-
被调用函数	-
输入参数{in}	
usart_periph	外设USARTx
USARTx	x=0,1,2
输入参数{in}	
cmdtype	请求类型
USART_CMD_SBK CMD	发送断开帧请求
USART_CMD_MM CMD	静模式请求
USART_CMD_RXF CMD	接收数据清空请求
USART_CMD_TXF CMD	发送数据清空请求
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* enable USART0 command */
```

```
usart_command_enable(USART0, USART_CMD_SBKCMD);
```

### 函数 usart\_flag\_get

函数usart\_flag\_get描述见下表：

表 3-1109. 函数 usart\_flag\_get

函数名称	usart_flag_get
函数原型	FlagStatus usart_flag_get(uint32_t usart_periph, usart_flag_enum flag);
功能描述	获取USART STAT/CHC/RFCSC寄存器标志位
先决条件	-
被调用函数	-
输入参数{in}	
usart_periph	外设USARTx
USARTx	x=0,1,2
输入参数{in}	
flag	USART标志位，参考 <a href="#">表3-1040. 枚举类型usart_flag_enum</a> 只能选择一个参数
USART_FLAG_PE	校验错误标志

RR	
USART_FLAG_FERR	帧错误标志
USART_FLAG_NERR	噪声错误标志
USART_FLAG_ORERR	溢出错误标志
USART_FLAG_IDLE	空闲线检测标志
USART_FLAG_RBNE	读数据缓冲区非空标志
USART_FLAG_TC	发送完成标志
USART_FLAG_TBE	发送数据缓冲区空标志
USART_FLAG_LBD	LIN断开检测标志
USART_FLAG_CTSF	CTS变化标志
USART_FLAG_CTS	CTS电平
USART_FLAG_RT	接收超时标志
USART_FLAG_EB	块结束标志
USART_FLAG_BSY	忙状态标志
USART_FLAG_AM	ADDR匹配标志
USART_FLAG_SB	断开信号发送标识
USART_FLAG_RWU	接收器从静默模式唤醒
USART_FLAG_WU	从深度睡眠模式唤醒标志
USART_FLAG_TEA	发送使能通知标志
USART_FLAG_REA	接收使能通知标志
USART_FLAG_EPERR	校验错误超前检测标志
USART_FLAG_RFE	接收FIFO空标志
USART_FLAG_RFF	接收FIFO满标志
USART_FLAG_RFFINT	接收FIFO满中断标志
输出参数{out}	
-	-
返回值	
FlagStatus	SET或RESET

例如：

```
/* get flag USART0 state */
```

```
FlagStatus status;
```

```
status = usart_flag_get(USART0,USART_FLAG_TBE);
```

### 函数 usart\_flag\_clear

函数usart\_flag\_clear描述见下表:

表 3-1110. 函数 usart\_flag\_clear

函数名称	usart_flag_clear
函数原型	void usart_flag_clear(uint32_t usart_periph, usart_flag_enum flag);
功能描述	清除USART状态寄存器标志位
先决条件	-
被调用函数	-
输入参数{in}	
usart_periph	外设USARTx
USARTx	x=0,1,2
输入参数{in}	
flag	USART标志位, 参考 <a href="#">表3-1040. 枚举类型usart_flag_enum</a> 只能选择一个参数
USART_FLAG_PERR	校验错误标志
USART_FLAG_FERR	帧错误标志
USART_FLAG_NERR	噪声错误标志
USART_FLAG_ORERR	溢出错误标志
USART_FLAG_IDLE	空闲线检测标志
USART_FLAG_TC	发送完成标志
USART_FLAG_LBD	LIN断开检测标志
USART_FLAG_CTSF	CTS变化标志
USART_FLAG_RTF	接收超时标志
USART_FLAG_EBF	块结束标志
USART_FLAG_AM	ADDR匹配标志
USART_FLAG_WU	从深度睡眠模式唤醒标志
USART_FLAG_EPERR	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 3-1111. 函数 usart\_interrupt\_enable

函数名称	usart_interrupt_enable
函数原型	void usart_interrupt_enable(uint32_t usart_periph, usart_interrupt_enum interrupt);
功能描述	使能USART中断
先决条件	-
被调用函数	-
输入参数{in}	
usart_periph	外设USARTx
USARTx	x=0,1,2
输入参数{in}	
interrupt	USART中断USART标志位, 参考 <a href="#">表3-1042. 枚举类型usart_interrupt_enum</a> 只能选择一个参数
USART_INT_IDLE	IDLE线检测中断
USART_INT_RBNE	读数据缓冲区非空中断和过载错误中断
USART_INT_TC	发送完成中断
USART_INT_TBE	发送缓冲区空中断
USART_INT_PERR	校验错误中断
USART_INT_AM	ADDR匹配中断
USART_INT_RT	接收超时事件中断
USART_INT_EB	块结束事件中断
USART_INT_LBD	LIN断开信号检测中断
USART_INT_ERR	错误中断
USART_INT_CTS	CTS中断
USART_INT_WU	从深度睡眠模式唤醒中断
USART_INT_RFF	接收FIFO满中断
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* enable USART0 TBE interrupt */
```

```
usart_interrupt_enable(USART0, USART_INT_TBE);
```

**函数 usart\_interrupt\_disable**

函数usart\_interrupt\_disable描述见下表:

**表 3-1112. 函数 usart\_interrupt\_disable**

函数名称	usart_interrupt_disable
函数原型	void usart_interrupt_disable(uint32_t usart_periph, usart_interrupt_enum interrupt);
功能描述	失能USART中断
先决条件	-
被调用函数	-
输入参数{in}	
usart_periph	外设USARTx
USARTx	x=0,1,2
输入参数{in}	
interrupt	USART中断USART标志位, 参考 <a href="#">表3-1042. 枚举类型usart_interrupt_enum</a> 只能选择一个参数
USART_INT_IDLE	IDLE线检测中断
USART_INT_RBNE	读数据缓冲区非空中断和过载错误中断
USART_INT_TC	发送完成中断
USART_INT_TBE	发送缓冲区空中断
USART_INT_PERR	校验错误中断
USART_INT_AM	ADDR匹配中断
USART_INT_RT	接收超时事件中断
USART_INT_EB	块结束事件中断
USART_INT_LBD	LIN断开信号检测中断
USART_INT_ERR	错误中断
USART_INT_CTS	CTS中断
USART_INT_WU	从深度睡眠模式唤醒中断
USART_INT_RFF	接收FIFO满中断
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* disable USART0 TBE interrupt */
usart_interrupt_disable(USART0, USART_INT_TBE);
```

**函数 usart\_interrupt\_flag\_get**

函数usart\_interrupt\_flag\_get描述见下表:

表 3-1113. 函数 usart\_interrupt\_flag\_get

函数名称	usart_interrupt_flag_get
函数原型	FlagStatus usart_interrupt_flag_get(uint32_t usart_periph, usart_interrupt_flag_enum int_flag);
功能描述	获取USART中断标志位状态
先决条件	-
被调用函数	-
输入参数{in}	
usart_periph	外设USARTx
USARTx	x=0,1,2
输入参数{in}	
int_flag	USART中断标志, 参考 <a href="#">表3-1041. 枚举类型usart_interrupt_flag_enum</a> 只能选择一个参数
USART_INT_FLAG _EB	块结束事件中断标志
USART_INT_FLAG _RT	超时事件中断标志
USART_INT_FLAG _AM	ADDR匹配中断标志
USART_INT_FLAG _PERR	校验错误中断标志
USART_INT_FLAG _TBE	发送缓冲区空中断标志
USART_INT_FLAG _TC	发送完成中断标志
USART_INT_FLAG _RBNE	读数据缓冲区非空中断标志
USART_INT_FLAG _RBNE_ORERR	读数据缓冲区非空中断和溢出错误中断标志
USART_INT_FLAG _IDLE	IDLE线检测中断标志
USART_INT_FLAG _LBD	LIN断开检测中断标志
USART_INT_FLAG _WU	从深度睡眠模式唤醒中断标志
USART_INT_FLAG _CTS	CTS中断标志
USART_INT_FLAG _ERR_NERR	噪声错误中断标志
USART_INT_FLAG _ERR_ORERR	过载错误中断标志
USART_INT_FLAG _ERR_FERR	帧错误中断标志

USART_INT_FLAG _RFF	接收FIFO满中断标志
输出参数{out}	
-	-
返回值	
FlagStatus	SET或RESET

例如：

```
/* get the USART0 interrupt flag status */
```

```
FlagStatus status;
```

```
status = usart_interrupt_flag_get(USART0, USART_INT_FLAG_RBNE);
```

### 函数 usart\_interrupt\_flag\_clear

函数usart\_interrupt\_flag\_clear描述见下表：

表 3-1114. 函数 usart\_interrupt\_flag\_clear

函数名称	usart_interrupt_flag_clear
函数原型	void usart_interrupt_flag_clear(uint32_t usart_periph, usart_interrupt_flag_enum int_flag);
功能描述	清除USART中断标志位状态
先决条件	-
被调用函数	-
输入参数{in}	
usart_periph	外设USARTx
USARTx	x=0,1,2
输入参数{in}	
Int_flag	USART中断标志，参考 <a href="#">表3-1041. 枚举类型usart_interrupt_flag_enum</a> 只能选择一个参数
USART_INT_FLAG _PERR	校验错误中断标志
USART_INT_FLAG _ERR_FERR	帧错误中断标志
USART_INT_FLAG _ERR_NERR	噪声错误中断标志
USART_INT_FLAG _RBNE_ORERR	读数据缓冲区非空中断和溢出错误中断标志
USART_INT_FLAG _ERR_ORERR	过载错误中断标志
USART_INT_FLAG _IDLE	IDLE线检测中断标志
USART_INT_FLAG _TC	发送完成中断标志

USART_INT_FLAG _LBD	LIN断开检测中断标志
USART_INT_FLAG _CTS	CTS变化中断标志
USART_INT_FLAG _RT	接收超时事件中断标志
USART_INT_FLAG _EB	块结束事件中断标志
USART_INT_FLAG _AM	ADDR匹配中断标志
USART_INT_FLAG _WU	从深度睡眠模式唤醒中断标志
USART_INT_FLAG _RFF	接收FIFO满中断标志
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* clear the USART0 interrupt flag */
```

```
usart_interrupt_flag_clear(USART0, USART_INT_FLAG_TC);
```

## 3.32. 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[3.32.1](#)描述了WWDGT的寄存器列表，章节[3.32.2](#)对WWDGT库函数进行说明。

### 3.32.1. 外设寄存器说明

WWDGT寄存器列表如下表所示：

表 3-1115. 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

### 3.32.2. 外设库函数说明

WWDGT库函数列表如下表所示：



表 3-1116. WWDGT 库函数

库函数名称	库函数描述
wwdgt_deinit	将WWDGT寄存器设为缺省值
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 3-1117. 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 3-1118. 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 3-1119. 函数 wwdgt\_counter\_update

函数名称	wwdgt_counter_update
函数原型	void wwdgt_counter_update(uint16_t counter_value);
功能描述	设置WWDGT计数器更新值
先决条件	-
被调用函数	-
输入参数{in}	
counter_value	计数器更新值（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 3-1120. 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	计数器值（0x00 - 0x7F）
输入参数{in}	
window	窗口值（0x00 - 0x7F）
输入参数{in}	

prescaler	WWDGT预分频值
WWDGT_CFG_PSC_DIV1	WWDGT计数器时钟为 (PCLK/4096) /1
WWDGT_CFG_PSC_DIV2	WWDGT计数器时钟为 (PCLK/4096) /2
WWDGT_CFG_PSC_DIV4	WWDGT计数器时钟为 (PCLK/4096) /4
WWDGT_CFG_PSC_DIV8	WWDGT计数器时钟为 (PCLK/4096) /8
输出参数{out}	
-	-
Return value	
-	-

例如:

```
/* configure WWDGT counter value to 0x7F, window value to 0x50, prescaler divider value to 8 */
```

```
wwdgt_config(0x7F, 0x50, WWDGT_CFG_PSC_DIV8);
```

### 函数 wwdgt\_interrupt\_enable

函数wwdgt\_interrupt\_enable描述见下表:

表 3-1121. 函数 wwdgt\_interrupt\_enable

函数名称	wwdgt_interrupt_enable
函数原型	void wwdgt_interrupt_enable(void);
功能描述	使能WWDGT提前唤醒中断
先决条件	-
被调用函数	-
输入参数{in}	
-	-
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* enable early wakeup interrupt of WWDGT */
```

```
wwdgt_interrupt_enable ( );
```

### 函数 wwdgt\_flag\_get

函数wwdgt\_flag\_get描述见下表:

表 3-1122. 函数 `wwdgt_flag_get`

函数名称	<code>wwdgt_flag_get</code>
函数原型	<code>FlagStatus wwdgt_flag_get(void);</code>
功能描述	检查WWDGT提前唤醒中断标志位是否置位
先决条件	-
被调用函数	-
输入参数{in}	
-	-
输出参数{out}	
-	-
返回值	
<b>FlagStatus</b>	SET或RESET

例如:

```
/* test if the counter value update has reached the 0x40 */
```

```
FlagStatus status;
```

```
status = wwdgt_flag_get ( );
```

### 函数 `wwdgt_flag_clear`

函数`wwdgt_flag_clear`描述见下表:

表 3-1123. 函数 `wwdgt_flag_clear`

函数名称	<code>wwdgt_flag_clear</code>
函数原型	<code>void wwdgt_flag_clear(void);</code>
功能描述	清除WWDGT提前唤醒中断标志位状态
先决条件	-
被调用函数	-
输入参数{in}	
-	-
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* clear early wakeup interrupt state of WWDGT */
```

```
wwdgt_flag_clear( );
```

## 4. 版本历史

表 4-1. 版本历史

版本号.	说明	日期
1.0	初稿发布	2021 年 11 月 23 日
1.1	<ol style="list-style-type: none"> <li>1. 修改 <b>3.8.2.</b> 函数接口 efuse_mcu_init_data_write / efuse_aes_key_write / efuse_rotpk_key_write / efuse_dp_write / efuse_iak_write / efuse_user_data_write</li> <li>2. 修改 <b>3.24.2.</b> 函数接口 qspi_enable to spi_quad_enable / qspi_disable to spi_quad_disable / qspi_write_enable to spi_quad_write_enable / qspi_read_enable 为 spi_quad_read_enable / qspi_io23_output_enable to spi_quad_io23_output_enable / qspi_io23_output_disable to spi_quad_io23_output_disable</li> <li>3. 修改 <b>3.20.2.</b> 函数接口 quad_spi_enable 为 qspi_enable / quad_spi_disable 为 qspi_disable</li> <li>4. 修改 <b>3.21.2.</b> 中函数顺序</li> </ol>	2022 年 7 月 5 日
1.2	<ol style="list-style-type: none"> <li>1. <b>3.24.2.</b> 增加函数接口 spi_i2s_format_error_clear</li> <li>2. 修改 <b>3.31.2</b> 函数接口 usart_dma_enable / usart_dma_disable 为 usart_dma_receive_config /usart_dma_transmit_config</li> <li>3. <b>3.22.2</b> 中增加枚举 rcu_unit_enum, 修 改函数接口 rcu_xxx_poweron / rcu_xxx_powerdown 为 rcu_control_unit_powerup /rcu_control_unit_powerdown, 修改函 数接口 rcu_plldigsys_div_config 为 rcu_plldig_div_sys_config, 修改函数接 口 rcu_pllfi2s_clock_div_config 为 rcu_pll_div_i2s_config, 删除函数接口 rcu_plldig_enable/disable</li> </ol>	2022 年 12 月 25 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.