

**GigaDevice Semiconductor Inc.**

**GD32VW55x**  
**RISC-V 32-bit MCU**

**固件库**  
**使用指南**

1.3 版本

(2026 年 2 月)

# 目录

目录 .....	2
图索引 .....	5
表索引 .....	6
<b>1. 介绍 .....</b>	<b>25</b>
<b>1.1. 文档和固件库规则 .....</b>	<b>25</b>
1.1.1. 外设缩写 .....	25
1.1.2. 命名规则 .....	26
<b>2. 固件库概述 .....</b>	<b>27</b>
<b>2.1. 文件组织结构 .....</b>	<b>27</b>
2.1.1. Docs 文件夹 .....	29
2.1.2. Examples 文件夹 .....	29
2.1.3. Firmware 文件夹 .....	29
2.1.4. Template 文件夹 .....	29
2.1.5. Utilities 文件夹 .....	33
<b>2.2. 固件库文件描述 .....</b>	<b>33</b>
<b>3. 外设固件库 .....</b>	<b>35</b>
<b>3.1. 外设固件库概述 .....</b>	<b>35</b>
<b>3.2. ADC .....</b>	<b>35</b>
3.2.1. 外设寄存器描述 .....	35
3.2.2. 外设库函数说明 .....	36
<b>3.3. CAU .....</b>	<b>62</b>
3.3.1. 外设寄存器说明 .....	62
3.3.2. 外设库函数说明 .....	63
<b>3.4. CRC .....</b>	<b>91</b>
3.4.1. 外设寄存器说明 .....	91
3.4.2. 外设库函数说明 .....	91
<b>3.5. DBG .....</b>	<b>95</b>
3.5.1. 外设寄存器说明 .....	95
3.5.2. 外设库函数说明 .....	95
<b>3.6. DMA .....</b>	<b>99</b>
3.6.1. 外设寄存器说明 .....	99
3.6.2. 外设库函数说明 .....	100
<b>3.7. ECLIC .....</b>	<b>125</b>
3.7.1. 外设库函数说明 .....	125

<b>3.8. EFUSE .....</b>	<b>131</b>
3.8.1. 外设寄存器说明 .....	131
3.8.2. 外设库函数说明 .....	132
<b>3.9. EXTI .....</b>	<b>147</b>
3.9.1. 外设寄存器说明 .....	148
3.9.2. 外设库函数说明 .....	148
<b>3.10. FMC .....</b>	<b>155</b>
3.10.1. 外设寄存器说明 .....	155
3.10.2. 外设库函数说明 .....	156
<b>3.11. FWDGT.....</b>	<b>175</b>
3.11.1. 外设寄存器说明 .....	175
3.11.2. 外设库函数说明 .....	176
<b>3.12. GPIO .....</b>	<b>180</b>
3.12.1. 外设寄存器说明 .....	180
3.12.2. 外设库函数说明 .....	181
<b>3.13. HAU .....</b>	<b>191</b>
3.13.1. 外设寄存器说明 .....	192
3.13.2. 外设库函数说明 .....	192
<b>3.14. I2C .....</b>	<b>212</b>
3.14.1. 外设寄存器说明 .....	212
3.14.2. 外设库函数说明 .....	212
<b>3.15. PKCAU.....</b>	<b>251</b>
3.15.1. 外设寄存器说明 .....	251
3.15.2. 外设库函数说明 .....	251
<b>3.16. PMU.....</b>	<b>281</b>
3.16.1. 外设寄存器说明 .....	281
3.16.2. 外设库函数说明 .....	281
<b>3.17. QSPI.....</b>	<b>299</b>
3.17.1. 外设寄存器说明 .....	299
3.17.2. 外设库函数说明 .....	300
<b>3.18. RCU.....</b>	<b>315</b>
3.18.1. 外设寄存器说明 .....	315
3.18.2. 外设库函数说明 .....	316
<b>3.19. RTC .....</b>	<b>348</b>
3.19.1. 外设寄存器描述 .....	348
3.19.2. 外设库函数描述 .....	349
<b>3.20. SPI.....</b>	<b>377</b>
3.20.1. 外设寄存器说明 .....	377
3.20.2. 外设库函数说明 .....	378

<b>3.21.</b>	<b>SYSCFG .....</b>	<b>395</b>
3.21.1.	外设寄存器说明 .....	395
3.21.2.	外设库函数说明 .....	395
<b>3.22.</b>	<b>TIMER .....</b>	<b>400</b>
3.22.1.	外设寄存器说明 .....	400
3.22.2.	外设库函数说明 .....	401
<b>3.23.</b>	<b>TRNG.....</b>	<b>458</b>
3.23.1.	外设寄存器说明 .....	458
3.23.2.	外设库函数说明 .....	459
<b>3.24.</b>	<b>USART .....</b>	<b>464</b>
3.24.1.	外设寄存器说明 .....	464
3.24.2.	外设库函数说明 .....	464
<b>3.25.</b>	<b>WWDGT .....</b>	<b>509</b>
3.25.1.	外设寄存器说明 .....	510
3.25.2.	外设库函数说明 .....	510
<b>4.</b>	<b>版本历史.....</b>	<b>515</b>

## 图索引

图 2-1. GD32VW55x 固件库文件组织结构 .....	28
图 2-2. 选择外设例程文件 .....	30
图 2-3. 拷贝外设例程文件 .....	31
图 2-4. 打开工程文件 .....	31
图 2-5. 配置工程文件 .....	32
图 2-6. 编译 .....	33

## 表索引

表 1-1. 外设缩写.....	25
表 2-1. 固件函数库文件描述.....	33
表 3-1. 外设固件库函数描述格式.....	35
表 3-2. ADC 寄存器.....	35
表 3-3. ADC 库函数.....	36
表 3-4. 函数 adc_deinit .....	37
表 3-5. 函数 adc_clock_config .....	37
表 3-6. 函数 adc_enable.....	38
表 3-7. 函数 adc_disable .....	39
表 3-8. 函数 adc_dma_mode_enable .....	39
表 3-9. 函数 adc_dma_mode_disable .....	40
表 3-10. 函数 adc_dma_request_after_last_enable.....	40
表 3-11. 函数 adc_dma_request_after_last_disable .....	41
表 3-12. 函数 adc_discontinuous_mode_config .....	41
表 3-13. 函数 adc_special_function_config .....	42
表 3-14. 函数 adc_tempsensor_vrefint_enable .....	43
表 3-15. 函数 adc_tempsensor_vrefint_disable .....	43
表 3-16. 函数 adc_data_alignment_config .....	44
表 3-17. 函数 adc_channel_length_config .....	44
表 3-18. 函数 adc_routine_channel_config.....	45
表 3-19. 函数 adc_inserted_channel_config.....	46
表 3-20. 函数 adc_inserted_channel_offset_config .....	47
表 3-21. 函数 adc_external_trigger_config .....	48
表 3-22. 函数 adc_external_trigger_source_config.....	49
表 3-23. 函数 adc_software_trigger_enable.....	50
表 3-24. 函数 adc_end_of_conversion_config .....	51
表 3-25. 函数 adc_resolution_config.....	51
表 3-26. 函数 adc_routine_data_read.....	52
表 3-27. 函数 adc_inserted_data_read.....	53
表 3-28. 函数 adc_watchdog_single_channel_enable .....	53
表 3-29. 函数 adc_watchdog_group_channel_enable .....	54
表 3-30. 函数 adc_watchdog_disable.....	54
表 3-31. 函数 adc_watchdog_threshold_config.....	55
表 3-32. 函数 adc_oversample_mode_config.....	55
表 3-33. 函数 adc_oversample_mode_enable .....	57
表 3-34. 函数 adc_oversample_mode_disable .....	58
表 3-35. 函数 adc_flag_get.....	58
表 3-36. 函数 adc_flag_clear.....	59
表 3-37. 函数 adc_interrupt_enable.....	60
表 3-38. 函数 adc_interrupt_disable.....	60

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

表 3-82. CRC 寄存器.....	91
表 3-83. CRC 库函数.....	91
表 3-84. 函数 crc_deinit.....	91
表 3-85. 函数 crc_data_register_reset.....	92
表 3-86. 函数 crc_data_register_read.....	92
表 3-87. 函数 crc_free_data_register_read.....	93
表 3-88. 函数 crc_free_data_register_write.....	93
表 3-89. 函数 crc_single_data_calculate.....	94
表 3-90. 函数 crc_block_data_calculate.....	94
表 3-91. DBG 寄存器.....	95
表 3-92. DBG 库函数.....	95
表 3-93. 枚举类型 dbg_periph_enum.....	96
表 3-94. 函数 dbg_deinit.....	96
表 3-95. 函数 dbg_id_get.....	97
表 3-96. 函数 dbg_low_power_enable.....	97
表 3-97. 函数 dbg_low_power_disable.....	98
表 3-98. 函数 dbg_periph_enable.....	98
表 3-99. 函数 dbg_periph_disable.....	99
表 3-100. DMA 寄存器.....	100
表 3-101. DMA 库函数.....	100
表 3-102. 枚举类型 dma_channel_enum.....	101
表 3-103. 枚举类型 dma_subperipheral_enum.....	101
表 3-104. 结构体 dma_multi_data_parameter_struct.....	102
表 3-105. 结构体 dma_single_data_parameter_struct.....	102
表 3-106. 函数 dma_deinit.....	102
表 3-107. 函数 dma_single_data_para_struct_init.....	103
表 3-108. 函数 dma_multi_data_para_struct_init.....	104
表 3-109. 函数 dma_single_data_mode_init.....	104
表 3-110. 函数 dma_multi_data_mode_init.....	105
表 3-111. 函数 dma_periph_address_config.....	106
表 3-112. 函数 dma_memory_address_config.....	107
表 3-113. 函数 dma_transfer_number_config.....	108
表 3-114. 函数 dma_transfer_number_get.....	108
表 3-115. 函数 dma_priority_config.....	109
表 3-116. 函数 dma_memory_burst_beats_config.....	109
表 3-117. 函数 dma_periph_burst_beats_config.....	110
表 3-118. 函数 dma_memory_width_config.....	111
表 3-119. 函数 dma_periph_width_config.....	112
表 3-120. 函数 dma_memory_address_generation_config.....	112
表 3-121. 函数 dma_peripheral_address_generation_config.....	113
表 3-122. 函数 dma_circulation_enable.....	114
表 3-123. 函数 dma_circulation_disable.....	114
表 3-124. 函数 dma_channel_enable.....	115



表 3-125. 函数 dma_channel_disable.....	115
表 3-126. 函数 dma_transfer_direction_config .....	116
表 3-127. 函数 dma_switch_buffer_mode_config.....	117
表 3-128. 函数 dma_using_memory_get .....	117
表 3-129. 函数 dma_channel_subperipheral_select.....	118
表 3-130. 函数 dma_flow_controller_config .....	118
表 3-131. 函数 dma_switch_buffer_mode_enable .....	119
表 3-132. 函数 dma_switch_buffer_mode_disable .....	120
表 3-133. 函数 dma_fifo_status_get .....	120
表 3-134. 函数 dma_flag_get.....	121
表 3-135. 函数 dma_flag_clear .....	121
表 3-136. 函数 dma_interrupt_enable .....	122
表 3-137. 函数 dma_interrupt_disable .....	123
表 3-138. 函数 dma_interrupt_flag_get.....	123
表 3-139. 函数 dma_interrupt_flag_clear.....	124
表 3-140. ECLIC 库函数.....	125
表 3-141. 枚举类型 IRQn_Type.....	126
表 3-142. 函数 eclic_global_interrupt_enable.....	128
表 3-143. 函数 eclic_global_interrupt_disable .....	128
表 3-144. 函数 eclic_level_threshold_set .....	129
表 3-145. 函数 eclic_priority_group_set .....	129
表 3-146. 函数 eclic_irq_enable .....	130
表 3-147. 函数 eclic_irq_disable .....	130
表 3-148. 函数 eclic_system_reset .....	131
表 3-149. EFUSE 寄存器 .....	132
表 3-150. EFUSE 库函数 .....	132
表 3-151. 枚举类型 efuse_flag_enum .....	133
表 3-152. 枚举类型 efuse_clear_flag_enum .....	133
表 3-153. 枚举类型 efuse_int_enum.....	133
表 3-154. 枚举类型 efuse_int_flag_enum.....	133
表 3-155. 枚举类型 efuse_clear_int_flag_enum.....	134
表 3-156. 枚举类型 efuse_reg_lock_enum.....	134
表 3-157. 函数 efuse_read .....	134
表 3-158. 函数 efuse_write .....	135
表 3-159. 函数 efuse_boot_config .....	135
表 3-160. 函数 efuse_control1_config.....	136
表 3-161. 函数 efuse_fp_config.....	136
表 3-162. 函数 efuse_user_control_config.....	137
表 3-163. 函数 efuse_res_write .....	138
表 3-164. 函数 efuse_aes_key_write .....	138
表 3-165. 函数 efuse_rotpk_key_write .....	139
表 3-166. 函数 efuse_user_data_write .....	139
表 3-167. 函数 efuse_res_read .....	140

表 3-168. 函数 efuse_aes_key_read .....	141
表 3-169. 函数 efuse_rotpk_key_read .....	141
表 3-170. 函数 efuse_puid_read .....	142
表 3-171. 函数 efuse_huk_key_read .....	142
表 3-172. 函数 efuse_user_data_read .....	143
表 3-173. 函数 efuse_boot_address_get .....	144
表 3-174. 函数 efuse_lock_config .....	144
表 3-175. 函数 efuse_flag_get .....	145
表 3-176. 函数 efuse_flag_clear .....	145
表 3-177. 函数 efuse_interrupt_enable .....	146
表 3-178. 函数 efuse_interrupt_disable .....	146
表 3-179. 函数 efuse_interrupt_flag_get .....	147
表 3-180. 函数 efuse_interrupt_flag_clear .....	147
表 3-181. EXTI 寄存器 .....	148
表 3-182. EXTI 库函数 .....	148
表 3-183. 枚举类型 exti_line_enum .....	148
表 3-184. 枚举类型 exti_mode_enum .....	149
表 3-185. 枚举类型 exti_trig_type_enum .....	149
表 3-186. 函数 exti_deinit .....	149
表 3-187. 函数 exti_init .....	150
表 3-188. 函数 exti_interrupt_enable .....	151
表 3-189. 函数 exti_interrupt_disable .....	151
表 3-190. 函数 exti_event_enable .....	151
表 3-191. 函数 exti_event_disable .....	152
表 3-192. 函数 exti_software_interrupt_enable .....	152
表 3-193. 函数 exti_software_interrupt_disable .....	153
表 3-194. 函数 exti_flag_get .....	153
表 3-195. 函数 exti_flag_clear .....	154
表 3-196. 函数 exti_interrupt_flag_get .....	154
表 3-197. 函数 exti_interrupt_flag_clear .....	155
表 3-198. FMC 寄存器 .....	155
表 3-199. FMC 固件库函数 .....	156
表 3-200. 枚举类型 fmc_state_enum .....	157
表 3-201. 函数 fmc_unlock .....	157
表 3-202. 函数 fmc_lock .....	158
表 3-203. 函数 fmc_page_erase .....	158
表 3-204. 函数 fmc_mass_erase .....	159
表 3-205. 函数 fmc_word_program .....	159
表 3-206. 函数 fmc_continuous_program .....	160
表 3-207. 函数 fmc_obr_function_enable .....	161
表 3-208. 函数 fmc_obr_function_disable .....	161
表 3-209. 函数 ob_unlock .....	162
表 3-210. 函数 ob_lock .....	163

表 3-211. 函数 ob_start .....	163
表 3-212. 函数 ob_reload .....	164
表 3-213. 函数 ob_security_protection_config .....	164
表 3-214. 函数 ob_user_write .....	165
表 3-215. 函数 ob_write_protection_config .....	165
表 3-216. 函数 fmc_no_rtdec_config .....	166
表 3-217. 函数 fmc_offset_region_config .....	167
表 3-218. 函数 fmc_offset_value_config .....	167
表 3-219. 函数 fmc_wifi_trim_cal_get .....	168
表 3-220. 函数 fmc_wifi_trim_pa_get .....	169
表 3-221. 函数 fmc_wifi_trim_get .....	169
表 3-222. 函数 fmc_pid_get .....	170
表 3-223. 函数 ob_write_protection_get .....	170
表 3-224. 函数 ob_user_get .....	171
表 3-225. 函数 ob_security_protection_flag_get .....	171
表 3-226. 函数 fmc_flag_get .....	172
表 3-227. 函数 fmc_flag_clear .....	172
表 3-228. 函数 fmc_interrupt_enable .....	173
表 3-229. 函数 fmc_interrupt_disable .....	173
表 3-230. 函数 fmc_interrupt_flag_get .....	174
表 3-231. 函数 fmc_interrupt_flag_clear .....	175
表 3-232. FWDGT 寄存器 .....	175
表 3-233. FWDGT 库函数 .....	176
表 3-234. 函数 fwdgt_write_enable .....	176
表 3-235. 函数 fwdgt_write_disable .....	176
表 3-236. 函数 fwdgt_enable .....	177
表 3-237. 函数 fwdgt_prescaler_value_config .....	177
表 3-238. 函数 fwdgt_reload_value_config .....	178
表 3-239. 函数 fwdgt_counter_reload .....	179
表 3-240. 函数 fwdgt_config .....	179
表 3-241. 函数 fwdgt_flag_get .....	180
表 3-242. GPIO 寄存器 .....	181
表 3-243. GPIO 库函数 .....	181
表 3-244. 函数 gpio_deinit .....	181
表 3-245. 函数 gpio_mode_set .....	182
表 3-246. 函数 gpio_output_options_set .....	183
表 3-247. 函数 gpio_bit_set .....	184
表 3-248. 函数 gpio_bit_reset .....	184
表 3-249. 函数 gpio_bit_write .....	185
表 3-250. 函数 gpio_port_write .....	186
表 3-251. 函数 gpio_input_bit_get .....	186
表 3-252. 函数 gpio_input_port_get .....	187
表 3-253. 函数 gpio_output_bit_get .....	188

表 3-254. 函数 gpio_output_port_get.....	188
表 3-255. 函数 gpio_af_set.....	189
表 3-256. 函数 gpio_pin_lock .....	190
表 3-257. 函数 gpio_bit_toggle.....	190
表 3-258. 函数 gpio_port_toggle .....	191
表 3-259. HAU 寄存器.....	192
表 3-260. HAU 库函数.....	192
表 3-261. 结构体 hau_init_parameter_struct.....	193
表 3-262. 结构体 hau_digest_parameter_struct .....	193
表 3-263. 结构体 hau_context_parameter_struct.....	193
表 3-264. 函数 hau_deinit.....	194
表 3-265. 函数 hau_init .....	194
表 3-266. 函数 hau_init_struct_para_init .....	195
表 3-267. 函数 hau_reset .....	196
表 3-268. 函数 hau_last_word_validbits_num_config .....	196
表 3-269. 函数 hau_data_write .....	197
表 3-270. 函数 hau_infifo_words_num_get.....	197
表 3-271. 函数 hau_digest_read.....	198
表 3-272. 函数 hau_digest_calculation_enable .....	198
表 3-273. 函数 hau_multiple_single_dma_config .....	199
表 3-274. 函数 hau_dma_enable .....	199
表 3-275. 函数 hau_dma_disable .....	200
表 3-276. 函数 hau_context_struct_para_init.....	200
表 3-277. 函数 hau_context_save .....	201
表 3-278. 函数 hau_context_restore.....	201
表 3-279. 函数 hau_hash_sha_1 .....	202
表 3-280. 函数 hau_hmac_sha_1 .....	202
表 3-281. 函数 hau_hash_sha_224 .....	203
表 3-282. 函数 hau_hmac_sha_224 .....	204
表 3-283. 函数 hau_hash_sha_256 .....	204
表 3-284. 函数 hau_hmac_sha_256 .....	205
表 3-285. 函数 hau_hash_md5 .....	206
表 3-286. 函数 hau_hmac_md5 .....	206
表 3-287. 函数 hau_flag_get.....	207
表 3-288. 函数 hau_flag_clear .....	208
表 3-289. 函数 hau_interrupt_enable .....	208
表 3-290. 函数 hau_interrupt_disable .....	209
表 3-291. 函数 hau_interrupt_flag_get.....	210
表 3-292. 函数 hau_interrupt_flag_clear.....	210
表 3-293. I2C 寄存器 .....	212
表 3-294. I2C 库函数 .....	212
表 3-295. 枚举类型 i2c_interrupt_flag_enum .....	214
表 3-296. 函数 i2c_deinit .....	214

表 3-297. 函数 i2c_timing_config.....	215
表 3-298. 函数 i2c_digital_noise_filter_config.....	215
表 3-299. 函数 i2c_analog_noise_filter_enable .....	216
表 3-300. 函数 i2c_analog_noise_filter_disable .....	217
表 3-301. 函数 i2c_master_clock_config .....	217
表 3-302. 函数 i2c_master_addressing .....	218
3-303. 函数 i2c_address10_header_enable .....	219
表 3-304. 函数 i2c_address10_header_disable .....	219
表 3-305. 函数 i2c_address10_enable.....	220
表 3-306. 函数 i2c_address10_disable.....	220
表 3-307. 函数 i2c_automatic_end_enable.....	221
表 3-308. 函数 i2c_automatic_end_disable.....	221
表 3-309. 函数 i2c_slave_response_to_gcall_enable.....	222
表 3-310. 函数 i2c_slave_response_to_gcall_disable.....	222
表 3-311. 函数 i2c_stretch_scl_low_enable .....	223
表 3-312. 函数 i2c_stretch_scl_low_disable .....	223
表 3-313. 函数 i2c_address_config.....	224
表 3-314. 函数 i2c_address_bit_compare_config .....	225
表 3-315. 函数 i2c_address_disable .....	226
表 3-316. 函数 i2c_second_address_config .....	226
表 3-317. 函数 i2c_second_address_disable.....	227
表 3-318. 函数 i2c_receved_address_get.....	228
表 3-319. 函数 i2c_slave_byte_control_enable .....	228
表 3-320. 函数 i2c_slave_byte_control_disable .....	229
表 3-321. 函数 i2c_nack_enable.....	229
表 3-322. 函数 i2c_wakeup_from_deepsleep_enable .....	230
表 3-323. 函数 i2c_wakeup_from_deepsleep_disable .....	230
表 3-324. 函数 i2c_enable.....	231
表 3-325. 函数 i2c_disable.....	231
表 3-326. 函数 i2c_start_on_bus.....	232
表 3-327. 函数 i2c_stop_on_bus .....	232
表 3-328. 函数 i2c_data_transmit.....	233
表 3-329. 函数 i2c_data_receive.....	233
表 3-330. 函数 i2c_reload_enable .....	234
表 3-331. 函数 i2c_reload_disable .....	234
表 3-332. 函数 i2c_transfer_byte_number_config .....	235
表 3-333. 函数 i2c_dma_enable.....	235
表 3-334. 函数 i2c_dma_disable.....	236
表 3-335. 函数 i2c_pec_transfer.....	236
表 3-336. 函数 i2c_pec_enable .....	237
表 3-337. 函数 i2c_pec_disable .....	238
表 3-338. 函数 i2c_pec_value_get .....	238
表 3-339. 函数 i2c_smbus_alert_enable .....	239

3-340. 函数 i2c_smbus_alert_disable .....	239
表 3-341. 函数 i2c_smbus_default_addr_enable .....	240
表 3-342. 函数 i2c_smbus_default_addr_disable .....	240
表 3-343. 函数 i2c_smbus_host_addr_enable .....	241
表 3-344. 函数 i2c_smbus_host_addr_disable .....	241
表 3-345. 函数 i2c_extented_clock_timeout_enable .....	242
表 3-346. 函数 i2c_extented_clock_timeout_disable .....	242
表 3-347. 函数 i2c_clock_timeout_enable .....	243
表 3-348. 函数 i2c_clock_timeout_disable .....	243
表 3-349. 函数 i2c_bus_timeout_b_config .....	244
表 3-350. 函数 i2c_bus_timeout_a_config .....	244
表 3-351. 函数 i2c_idle_clock_timeout_config .....	245
表 3-352. 函数 i2c_flag_get .....	245
表 3-353. 函数 i2c_flag_clear .....	246
表 3-354. 函数 i2c_interrupt_enable .....	247
表 3-355. 函数 i2c_interrupt_disable .....	248
表 3-356. 函数 i2c_interrupt_flag_get .....	249
表 3-357. 函数 i2c_interrupt_flag_clear .....	250
表 3-358. PKCAU 寄存器 .....	251
表 3-359. PKCAU 库函数 .....	251
表 3-360. 结构体 pkcau_mont_parameter_struct .....	252
表 3-361. 结构体 pkcau_mod_parameter_struct .....	252
表 3-362. 结构体 pkcau_mod_exp_parameter_struct .....	252
表 3-363. 结构体 pkcau_arithmetic_parameter_struct .....	253
表 3-364. 结构体 pkcau_crt_parameter_struct .....	253
表 3-365. 结构体 pkcau_ec_group_parameter_struct .....	253
表 3-366. 结构体 pkcau_point_parameter_struct .....	254
表 3-367. 结构体 pkcau_signature_parameter_struct .....	254
表 3-368. 结构体 pkcau_hash_parameter_struct .....	254
表 3-369. 结构体 pkcau_ecc_out_struct .....	254
表 3-370. 函数 pkcau_deinit .....	255
表 3-371. 函数 pkcau_mont_struct_para_init .....	255
表 3-372. 函数 pkcau_mod_struct_para_init .....	256
表 3-373. 函数 pkcau_mod_exp_struct_para_init .....	256
表 3-374. 函数 pkcau_arithmetic_struct_para_init .....	257
表 3-375. 函数 pkcau_crt_struct_para_init .....	257
表 3-376. 函数 pkcau_ec_group_struct_para_init .....	258
表 3-377. 函数 pkcau_point_struct_para_init .....	258
表 3-378. 函数 pkcau_signature_struct_para_init .....	259
表 3-379. 函数 pkcau_hash_struct_para_init .....	260
表 3-380. 函数 pkcau_ecc_out_struct_para_init .....	260
表 3-381. 函数 pkcau_enable .....	261
表 3-382. 函数 pkcau_disable .....	261

表 3-383. 函数 pkcau_start.....	262
表 3-384. 函数 pkcau_mode_set .....	262
表 3-385. 函数 pkcau_mont_param_operation.....	263
表 3-386. 函数 pkcau_mod_operation.....	264
表 3-387. 函数 pkcau_mod_exp_operation .....	265
表 3-388. 函数 pkcau_mod_inver_operation .....	266
表 3-389. 函数 pkcau_mod_reduc_operation .....	267
表 3-390. 函数 pkcau_arithmetic_operation .....	268
表 3-391. 函数 pkcau crt_exp_operation .....	269
表 3-392. 函数 pkcau_point_check_operation .....	270
表 3-393. 函数 pkcau_point_mul_operation .....	272
表 3-394. 函数 pkcau_ecdsa_sign_operation.....	273
表 3-395. 函数 pkcau_ecdsa_verification_operation .....	275
表 3-396. 函数 pkcau_flag_get .....	277
表 3-397. 函数 pkcau_flag_clear .....	278
表 3-398. 函数 pkcau_interrupt_enable .....	278
表 3-399. 函数 pkcau_interrupt_disable .....	279
表 3-400. 函数 pkcau_interrupt_flag_get.....	279
表 3-401. 函数 pkcau_interrupt_flag_clear.....	280
表 3-402. PMU 寄存器.....	281
表 3-403. PMU 库函数.....	281
表 3-404. 函数 pmu_deinit.....	282
表 3-405. 函数 pmu_lvd_select.....	283
表 3-406. 函数 pmu_lvd_disable .....	283
表 3-407. 函数 pmu_backup_write_enable.....	284
表 3-408. 函数 pmu_backup_write_disable .....	284
表 3-409. 函数 pmu_to_sleepmode .....	285
表 3-410. 函数 pmu_to_deepsleepmode .....	285
表 3-411. 函数 pmu_to_standbymode.....	286
表 3-412. 函数 pmu_wakeup_pin_enable .....	287
表 3-413. 函数 pmu_wakeup_pin_disable .....	287
表 3-414. 函数 pmu_wifi_power_enable .....	288
表 3-415. 函数 pmu_wifi_power_disable .....	289
表 3-416. 函数 pmu_wifi_sram_control .....	289
表 3-417. 函数 pmu_ble_control .....	290
表 3-418. 函数 pmu_ble_wakeup_request_enable .....	290
表 3-419. 函数 pmu_ble_wakeup_request_disable .....	291
表 3-420. 函数 pmu_pll_force_enable .....	291
表 3-421. 函数 pmu_pll_force_disable .....	292
表 3-422. 函数 pmu_ble_rf_config .....	293
表 3-423. 函数 pmu_rf_force_enable.....	293
表 3-424. 函数 pmu_rf_force_disable.....	294
表 3-425. 函数 pmu_rf_sequence_config .....	294



表 3-426. 函数 pmu_flag_get .....	295
表 3-427. 函数 pmu_flag_clear .....	296
表 3-428. 函数 pmu_interrupt_enable .....	297
表 3-429. 函数 pmu_interrupt_disable .....	298
表 3-430. 函数 pmu_interrupt_flag_get .....	298
表 3-431. 函数 pmu_interrupt_flag_clear .....	299
表 3-432. QSPI 寄存器 .....	299
表 3-433. QSPI 库函数 .....	300
表 3-434. 结构体 qspi_init_struct .....	301
表 3-435. 结构体 qspi_command_struct .....	301
表 3-436. 结构体 qspi_polling_struct .....	301
表 3-437. 函数 qspi_deinit .....	302
表 3-438. 函数 qspi_struct_para_init .....	302
表 3-439. 函数 qspi_cmd_struct_para_init .....	303
表 3-440. 函数 qspi_polling_struct_para_init .....	303
表 3-441. 函数 qspi_init .....	304
表 3-442. 函数 qspi_enable .....	304
表 3-443. 函数 qspi_disable .....	305
表 3-444. 函数 qspi_dma_enable .....	305
表 3-445. 函数 qspi_dma_disable .....	306
表 3-445. 函数 qspi_data_length_config .....	306
表 3-446. 函数 qspi_command_config .....	307
表 3-447. 函数 qspi_polling_config .....	308
表 3-448. 函数 qspi_memorymapped_config .....	309
表 3-449. 函数 qspi_data_transmit .....	310
表 3-450. 函数 qspi_data_receive .....	310
表 3-451. 函数 qspi_transmission_abort .....	311
表 3-452. 函数 qspi_flag_get .....	311
表 3-453. 函数 qspi_flag_clear .....	312
表 3-454. 函数 qspi_interrupt_enable .....	313
表 3-455. 函数 qspi_interrupt_disable .....	313
表 3-456. 函数 qspi_interrupt_flag_get .....	314
表 3-457. 函数 qspi_interrupt_flag_clear .....	314
表 3-458. RCU 寄存器 .....	315
表 3-459. RCU 库函数 .....	316
表 3-460. 枚举类型 rcu_periph_enum .....	317
表 3-461. 枚举类型 rcu_periph_reset_enum .....	318
表 3-462. 枚举类型 rcu_unit_enum .....	319
表 3-463. 枚举类型 rcu_flag_enum .....	319
表 3-464. 枚举类型 rcu_int_flag_enum .....	320
表 3-465. 枚举类型 rcu_int_flag_clear_enum .....	320
表 3-466. 枚举类型 rcu_int_enum .....	320
表 3-467. 枚举类型 rcu_osci_type_enum .....	321



表 3-468. 枚举类型 rcu_clock_freq_enum .....	321
表 3-469. 函数 rcu_deinit.....	321
表 3-470. 函数 rcu_irc16m_dfs_to_rf_enable .....	322
表 3-471. 函数 rcu_irc16m_dfs_to_rf_disable .....	322
表 3-472. 函数 rcu_periph_clock_enable .....	323
表 3-473. 函数 rcu_periph_clock_disable .....	323
表 3-474. 函数 rcu_fmc_clock_sleep_enable.....	324
表 3-475. 函数 rcu_fmc_clock_sleep_disable .....	324
表 3-476. 函数 rcu_periph_reset_enable .....	325
表 3-477. 函数 rcu_periph_reset_disable .....	325
表 3-478. 函数 rcu_bkp_reset_enable .....	326
表 3-479. 函数 rcu_bkp_reset_disable .....	326
表 3-480. 函数 rcu_rfppl_cal_enable .....	327
表 3-481. 函数 rcu_rfppl_cal_disable .....	327
表 3-482. 函数 rcu_control_unit_powerup .....	328
表 3-483. 函数 rcu_control_unit_powerdown.....	328
表 3-484. 函数 rcu_system_clock_source_config .....	329
表 3-485. 函数 rcu_system_clock_source_get.....	330
表 3-486. 函数 rcu_ahb_clock_config .....	330
表 3-487. 函数 rcu_apb1_clock_config .....	331
表 3-488. 函数 rcu_apb2_clock_config .....	331
表 3-489. 函数 rcu_ckout0_config .....	332
表 3-490. 函数 rcu_ckout1_config .....	333
表 3-491. 函数 rcu_plldig_config .....	334
表 3-492. 函数 rcu_plldigdiv_sys_config .....	334
表 3-493. 函数 rcu_rtc_clock_config .....	335
表 3-494. 函数 rcu_rtc_div_config .....	336
表 3-495. 函数 rcu_trng_div_config.....	336
表 3-496. 函数 rcu_i2c0_clock_config .....	337
表 3-497. 函数 rcu_usart0_clock_source .....	337
表 3-498. 函数 rcu_irc16m_div_config.....	338
表 3-499. 函数 rcu_timer_clock_prescaler_config .....	338
表 3-500. 函数 rcu_flag_get .....	339
表 3-501. 函数 rcu_all_reset_flag_clear.....	340
表 3-502. 函数 rcu_interrupt_flag_get.....	340
表 3-503. 函数 rcu_interrupt_flag_clear.....	341
表 3-504. 函数 rcu_interrupt_enable .....	341
表 3-505. 函数 rcu_interrupt_disable .....	342
表 3-506. 函数 rcu_lxtal_drive_capability_config .....	342
表 3-507. 函数 rcu_osci_stab_wait.....	343
表 3-508. 函数 rcu_osci_on.....	343
表 3-509. 函数 rcu_osci_off .....	344
表 3-510. 函数 rcu_osci_bypass_mode_enable.....	344

表 3-511. 函数 rcu_osc_bypass_mode_disable .....	345
表 3-512. 函数 rcu_rf_hxtal_clock_monitor_enable .....	345
表 3-513. 函数 rcu_rf_hxtal_clock_monitor_disable .....	346
表 3-514. 函数 rcu_irc16m_adjust_value_set .....	346
表 3-515. 函数 rcu_voltage_key_unlock .....	347
表 3-516. 函数 rcu_deepsleep_voltage_set .....	347
表 3-517. 函数 rcu_clock_freq_get .....	348
表 3-518. RTC 寄存器 .....	349
表 3-519. RTC 库函数 .....	349
表 3-520. 结构体 rtc_parameter_struct .....	350
表 3-521. 结构体 rtc_alarm_struct .....	351
表 3-522. 结构体 rtc_timestamp_struct .....	351
表 3-523. 结构体 rtc_tamper_struct .....	351
表 3-524. 函数 rtc_deinit .....	352
表 3-525. 函数 rtc_init .....	352
表 3-526. 函数 rtc_init_mode_enter .....	353
表 3-527. 函数 rtc_init_mode_exit .....	354
表 3-528. 函数 rtc_register_sync_wait .....	354
表 3-529. 函数 rtc_current_time_get .....	355
表 3-530. 函数 rtc_subsecond_get .....	355
表 3-531. 函数 rtc_alarm_config .....	356
表 3-532. 函数 rtc_alarm_subsecond_config .....	356
表 3-533. 函数 rtc_alarm_get .....	357
表 3-534. 函数 rtc_alarm_subsecond_get .....	358
表 3-535. 函数 rtc_alarm_enable .....	358
表 3-536. 函数 rtc_alarm_disable .....	359
表 3-537. 函数 rtc_timestamp_enable .....	360
表 3-538. 函数 rtc_timestamp_disable .....	360
表 3-539. 函数 rtc_timestamp_get .....	361
表 3-540. 函数 rtc_timestamp_subsecond_get .....	361
表 3-541. 函数 rtc_timestamp_enable .....	362
表 3-542. 函数 rtc_tamper_disable .....	362
表 3-543. 函数 rtc_software_bkp_reset .....	363
表 3-544. 函数 rtc_tamper_without_bkp_reset .....	363
表 3-545. 函数 rtc_output_pad_select .....	364
表 3-546. 函数 rtc_alarm_output_config .....	364
表 3-547. 函数 rtc_calibration_output_config .....	365
表 3-548. 函数 rtc_hour_adjust .....	366
表 3-549. 函数 rtc_second_adjust .....	366
表 3-550. 函数 rtc_bypass_shadow_enable .....	367
表 3-551. 函数 rtc_bypass_shadow_disable .....	367
表 3-552. 函数 rtc_refclock_detection_enable .....	368
表 3-553. 函数 rtc_refclock_detection_disable .....	368

表 3-554. 函数 rtc_wakeup_enable .....	369
表 3-555. 函数 rtc_wakeup_disable .....	369
表 3-556. 函数 rtc_wakeup_clock_set .....	370
表 3-557. 函数 rtc_wakeup_timer_set .....	371
表 3-558. 函数 rtc_wakeup_timer_get .....	371
表 3-559. 函数 rtc_smooth_calibration_config .....	372
表 3-560. 函数 rtc_coarse_calibration_enable .....	373
表 3-561. 函数 rtc_coarse_calibration_disable .....	373
表 3-562. 函数 rtc_coarse_calibration_config .....	374
表 3-563. 函数 rtc_flag_get .....	375
表 3-564. 函数 rtc_flag_clear .....	375
表 3-565. 函数 rtc_interrupt_enable .....	376
表 3-566. 函数 rtc_interrupt_disable .....	377
表 3-567. SPI 寄存器 .....	377
表 3-568. SPI 库函数 .....	378
表 3-569. 结构体 spi_parameter_struct .....	379
表 3-570. 函数 spi_deinit .....	379
表 3-571. 函数 spi_struct_para_init .....	380
表 3-572. 函数 spi_init .....	380
表 3-573. 函数 spi_enable .....	381
表 3-574. 函数 spi_disable .....	381
表 3-575. 函数 spi_nss_output_enable .....	382
表 3-576. 函数 spi_nss_output_disable .....	382
表 3-577. 函数 spi_nss_internal_high .....	383
表 3-578. 函数 spi_nss_internal_low .....	383
表 3-579. 函数 spi_dma_enable .....	384
表 3-580. 函数 spi_dma_disable .....	384
表 3-581. 函数 spi_data_frame_format_config .....	385
表 3-582. 函数 spi_data_transmit .....	386
表 3-583. 函数 spi_data_receive .....	386
表 3-584. 函数 spi_bidirectional_transfer_config .....	387
表 3-585. 函数 spi_format_error_clear .....	387
表 3-586. 函数 spi_crc_polynomial_set .....	388
表 3-587. 函数 spi_crc_polynomial_get .....	388
表 3-588. 函数 spi_crc_on .....	389
表 3-589. 函数 spi_crc_off .....	389
表 3-590. 函数 spi_crc_next .....	390
表 3-591. 函数 spi_crc_get .....	390
表 3-592. 函数 spi_crc_error_clear .....	391
表 3-593. 函数 spi_ti_mode_enable .....	391
表 3-594. 函数 spi_ti_mode_disable .....	392
表 3-595. 函数 spi_interrupt_enable .....	392
表 3-596. 函数 spi_interrupt_disable .....	393

表 3-597. 函数 spi_interrupt_flag_get .....	393
表 3-598. 函数 spi_flag_get .....	394
表 3-599. SYSCFG 寄存器 .....	395
表 3-600. SYSCFG 库函数 .....	395
表 3-601. 枚举类型 syscfg_code_start_enum .....	396
表 3-602. 函数 syscfg_deinit .....	396
表 3-603. 函数 syscfg_exti_line_config .....	396
表 3-604. 函数 syscfg_io_compensation_enable .....	397
表 3-605. 函数 syscfg_io_compensation_disable .....	397
表 3-606. 函数 syscfg_lock_config .....	398
表 3-607. 函数 syscfg_io_compensation_ready_flag_get .....	398
表 3-608. 函数 syscfg_sram_ownership_config .....	399
表 3-609. 函数 syscfg_boot_mode_get .....	400
表 3-610. TIMER 寄存器 .....	400
表 3-611. TIMER 库函数 .....	401
表 3-612. 结构体 timer_parameter_struct .....	403
表 3-613. 结构体 timer_break_parameter_struct .....	404
表 3-614. 结构体 timer_oc_parameter_struct .....	404
表 3-615. 结构体 timer_ic_parameter_struct .....	404
表 3-616. 函数 timer_deinit .....	405
表 3-617. 函数 timer_struct_para_init .....	405
表 3-618. 函数 timer_init .....	406
表 3-619. 函数 timer_enable .....	407
表 3-620. 函数 timer_disable .....	407
表 3-621. 函数 timer_auto_reload_shadow_enable .....	408
表 3-622. 函数 timer_auto_reload_shadow_disable .....	408
表 3-623. 函数 timer_update_event_enable .....	409
表 3-624. 函数 timer_update_event_disable .....	409
表 3-625. 函数 timer_counter_alignment .....	410
表 3-626. 函数 timer_counter_up_direction .....	411
表 3-627. 函数 timer_counter_down_direction .....	411
表 3-628. 函数 timer_prescaler_config .....	412
表 3-629. 函数 timer_repetition_value_config .....	412
表 3-630. 函数 timer_autoreload_value_config .....	413
表 3-631. 函数 timer_counter_value_config .....	414
表 3-632. 函数 timer_counter_read .....	414
表 3-633. 函数 timer_prescaler_read .....	415
表 3-634. 函数 timer_single_pulse_mode_config .....	415
表 3-635. 函数 timer_update_source_config .....	416
表 3-636. 函数 timer_dma_enable .....	417
表 3-637. 函数 timer_dma_disable .....	417
表 3-638. 函数 timer_channel_dma_request_source_select .....	418
表 3-639. 函数 timer_dma_transfer_config .....	419

表 3-640. 函数 timer_event_software_generate .....	421
表 3-641. 函数 timer_break_struct_para_init .....	422
表 3-642. 函数 timer_break_config .....	422
表 3-643. 函数 timer_break_enable .....	423
表 3-644. 函数 timer_break_disable .....	424
表 3-645. 函数 timer_automatic_output_enable .....	424
表 3-646. 函数 timer_automatic_output_disable .....	425
表 3-647. 函数 timer_primary_output_config .....	425
表 3-648. 函数 timer_channel_control_shadow_config .....	426
表 3-649. 函数 timer_channel_control_shadow_update_config .....	426
表 3-650. 函数 timer_channel_output_struct_para_init .....	427
表 3-651. 函数 timer_channel_output_config .....	427
表 3-652. 函数 timer_channel_output_mode_config .....	428
表 3-653. 函数 timer_channel_output_pulse_value_config .....	430
表 3-654. 函数 timer_channel_output_shadow_config .....	430
表 3-655. 函数 timer_channel_output_fast_config .....	431
表 3-656. 函数 timer_channel_output_clear_config .....	432
表 3-657. 函数 timer_channel_output_polarity_config .....	433
表 3-658. 函数 timer_channel_complementary_output_polarity_config .....	434
表 3-659. 函数 timer_channel_output_state_config .....	434
表 3-660. 函数 timer_channel_complementary_output_state_config .....	435
表 3-661. 函数 timer_channel_input_struct_para_init .....	436
表 3-662. 函数 timer_input_capture_config .....	437
表 3-663. 函数 timer_channel_input_capture_prescaler_config .....	437
表 3-664. 函数 timer_channel_capture_value_register_read .....	438
表 3-665. 函数 timer_input_pwm_capture_config .....	439
表 3-666. 函数 timer_hall_mode_config .....	440
表 3-666. 函数 timer_channel_input_remap_config .....	441
表 3-667. 函数 timer_input_trigger_source_select .....	441
表 3-668. 函数 timer_master_output_trigger_source_select .....	442
表 3-669. 函数 timer_slave_mode_select .....	443
表 3-670. 函数 timer_master_slave_mode_config .....	444
表 3-671. 函数 timer_external_trigger_config .....	445
表 3-672. 函数 timer_quadrature_decoder_mode_config .....	446
表 3-673. 函数 timer_internal_clock_config .....	447
表 3-674. 函数 timer_internal_trigger_as_external_clock_config .....	448
表 3-675. 函数 timer_external_trigger_as_external_clock_config .....	448
表 3-676. 函数 timer_external_clock_mode0_config .....	449
表 3-677. 函数 timer_external_clock_mode1_config .....	450
表 3-678. 函数 timer_external_clock_mode1_disable .....	451
表 3-679. 函数 timer_write_chxval_register_config .....	452
表 3-680. 函数 timer_output_value_selection_config .....	452
表 3-681. 函数 timer_flag_get .....	453

表 3-682. 函数 timer_flag_clear .....	454
表 3-683. 函数 timer_interrupt_enable .....	455
表 3-684. 函数 timer_interrupt_disable .....	456
表 3-685. 函数 timer_interrupt_flag_get .....	456
表 3-686. 函数 timer_interrupt_flag_clear .....	457
表 3-687. TRNG 寄存器 .....	459
表 3-688. TRNG 库函数 .....	459
表 3-689. 枚举 trng_flag_enum.....	459
表 3-690. 枚举 trng_int_flag_enum.....	459
表 3-691. 函数 trng_deinit .....	459
表 3-692. 函数 trng_enable .....	460
表 3-693. 函数 trng_disable .....	460
表 3-694. 函数 trng_get_true_random_data.....	461
表 3-695. 函数 trng_interrupt_enable.....	461
表 3-696. 函数 trng_interrupt_disable.....	462
表 3-697. 函数 trng_flag_get.....	462
表 3-698. 函数 trng_interrupt_flag_get .....	463
表 3-699. 函数 trng_interrupt_flag_clear .....	463
表 3-700. USART 寄存器 .....	464
表 3-701. USART 库函数 .....	465
表 3-702. 枚举类型 usart_flag_enum .....	467
表 3-703. 枚举类型 usart_interrupt_flag_enum .....	467
表 3-704. 枚举类型 usart_interrupt_enum .....	468
表 3-705. 枚举类型 usart_invert_enum .....	468
表 3-706. 函数 usart_deinit .....	469
表 3-707. 函数 usart_baudrate_set.....	469
表 3-708. 函数 usart_parity_config.....	470
表 3-709. 函数 usart_word_length_set .....	470
表 3-710. 函数 usart_stop_bit_set .....	471
表 3-711. 函数 usart_enable.....	472
表 3-712. 函数 usart_disable.....	472
表 3-713. 函数 usart_transmit_config .....	473
表 3-714. 函数 usart_receive_config .....	473
表 3-715. 函数 usart_data_first_config .....	474
表 3-716. 函数 usart_invert_config.....	474
表 3-717. 函数 usart_overrun_enable .....	475
表 3-718. 函数 usart_overrun_disable .....	476
表 3-719. 函数 usart_oversample_config .....	476
表 3-720. 函数 usart_sample_bit_config .....	477
表 3-721. 函数 usart_receiver_timeout_enable .....	477
表 3-722. 函数 usart_receiver_timeout_disable .....	478
表 3-723. 函数 usart_receiver_timeout_threshold_config .....	478
表 3-724. 函数 usart_data_transmit.....	479



表 3-725. 函数 usart_data_receive.....	479
表 3-726. 函数 usart_command_enable.....	480
表 3-727. 函数 usart_address_config.....	481
表 3-728. 函数 usart_address_detection_mode_config .....	481
表 3-729. 函数 usart_mute_mode_enable .....	482
表 3-730. 函数 usart_mute_mode_disable .....	483
表 3-731. 函数 usart_mute_mode_wakeup_config.....	483
表 3-732. 函数 usart_lin_mode_enable.....	484
表 3-733. 函数 usart_lin_mode_disable.....	484
表 3-734. 函数 usart_lin_break_dection_length_config .....	485
表 3-735. 函数 usart_halfduplex_enable.....	485
表 3-736. 函数 usart_halfduplex_disable .....	486
表 3-737. 函数 usart_clock_enable.....	486
表 3-738. 函数 usart_clock_disable.....	487
表 3-739. 函数 usart_synchronous_clock_config.....	487
表 3-740. 函数 usart_guard_time_config.....	488
表 3-741. 函数 usart_smartcard_mode_enable .....	489
表 3-742. 函数 usart_smartcard_mode_disable .....	489
表 3-743. 函数 usart_smartcard_mode_nack_enable .....	490
表 3-744. 函数 usart_smartcard_mode_nack_disable .....	490
表 3-745. 函数 usart_smartcard_mode_early_nack_enable .....	491
表 3-746. 函数 usart_smartcard_mode_early_nack_disable .....	491
表 3-747. 函数 usart_smartcard_autoretry_config .....	492
表 3-748. 函数 usart_block_length_config.....	492
表 3-749. 函数 usart_irda_mode_enable .....	493
表 3-750. 函数 usart_irda_mode_disable .....	493
表 3-751. 函数 usart_prescaler_config .....	494
表 3-752. 函数 usart_irda_lowpower_config.....	494
表 3-753. 函数 usart_hardware_flow_rts_config.....	495
表 3-754. 函数 usart_hardware_flow_cts_config .....	496
表 3-755. 函数 usart_hardware_flow_coherence_config .....	496
表 3-756. 函数 usart_rs485_driver_enable .....	497
表 3-757. 函数 usart_rs485_driver_disable .....	498
表 3-758. 函数 usart_driver_asserttime_config.....	498
表 3-759. 函数 usart_driver_deasserttime_config .....	499
表 3-760. 函数 usart_depolarity_config .....	499
表 3-761. 函数 usart_dma_receive_config .....	500
表 3-762. 函数 usart_dma_transmit_config .....	500
表 3-763. 函数 usart_reception_error_dma_disable .....	501
表 3-764. 函数 usart_reception_error_dma_enable.....	502
表 3-765. 函数 usart_wakeup_enable.....	502
表 3-766. 函数 usart_wakeup_disable .....	503
表 3-767. 函数 usart_wakeup_mode_config .....	503

表 3-768. 函数 usart_receive_fifo_enable .....	504
表 3-769. 函数 usart_receive_fifo_disable .....	504
表 3-770. 函数 usart_receive_fifo_counter_number .....	505
表 3-771. 函数 usart_flag_get .....	505
表 3-772. 函数 usart_flag_clear .....	506
表 3-773. 函数 usart_interrupt_enable .....	507
表 3-774. 函数 usart_interrupt_disable .....	508
表 3-775. 函数 usart_interrupt_flag_get .....	508
表 3-776. 函数 usart_interrupt_flag_clear .....	509
表 3-777. WWDGT 寄存器 .....	510
表 3-778. WWDGT 库函数 .....	510
表 3-779. 函数 wwdgt_deinit .....	510
表 3-780. 函数 wwdgt_enable .....	511
表 3-781. 函数 wwdgt_counter_update .....	511
表 3-782. 函数 wwdgt_config .....	512
表 3-783. 函数 wwdgt_interrupt_enable .....	512
表 3-784. 函数 wwdgt_flag_get .....	513
表 3-785. 函数 wwdgt_flag_clear .....	513
表 4-1. 版本历史 .....	515



## 1. 介绍

本手册介绍了32位基于RISC-V的微控制器GD32VW55x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GD32VW55x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 应用编程接口）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。

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
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
HAU	哈希处理器
I2C	内部集成电路总线接口
PKCAU	公钥加密处理器
PMU	电源管理单元

外设缩写	说明
QSPI	四线SPI接口
RCU	复位和时钟单元
RTC	实时时钟
SPI	串行外设接口
TRNG	真随机数生成器
TIMER	定时器
USART	通用同步异步收发器
WWDGT	窗口看门狗

### 1.1.2. 命名规则

固件库遵从以下命名规则：

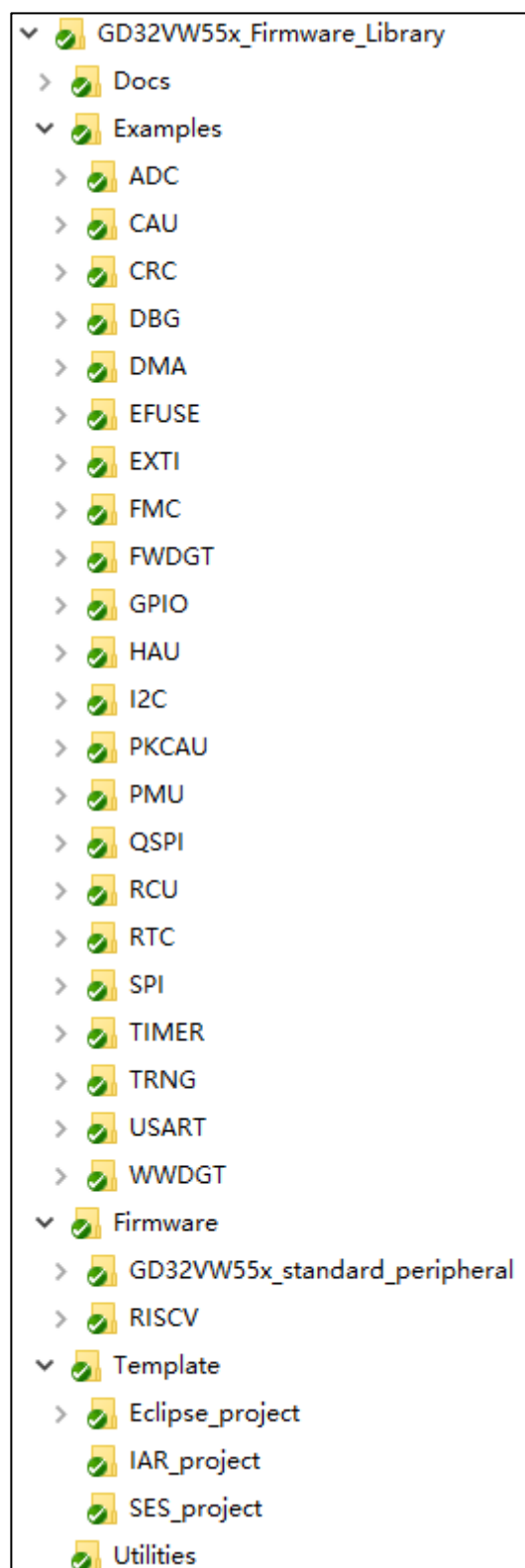
- XXX表示任一外设缩写，例如：ADC。更多缩写相关信息参阅[外设缩写](#)；
- 源文件和头文件命名都以“gd32vw55x\_”作为开头，例如：gd32vw55x\_adc.h；
- 常量仅被应用于一个文件的，定义于该文件中；被应用于多个文件的，在对应头文件中定义。所有常量都由英文字母大写书写；
- 寄存器作为常量处理。他们的命名都由英文字母大写书写。在大多数情况下，寄存器缩写规范与本用户手册一致；
- 变量名采用全部小写，有多个单词组成的，在单词之间以下划线分隔；
- 外设函数的命名以该外设的缩写加下划线为开头，有多个单词组成的，在单词之间以下划线分隔，所有外设函数都由英文字母小写书写。

## **2. 固件库概述**

### **2.1. 文件组织结构**

GD32VW55x\_Firmware\_Library，文件组织结构见下图：

图 2-1. GD32VW55x 固件库文件组织结构



### 2.1.1. Docs 文件夹

文件夹Docs包含固件库使用指南User Guide和开发板原理图Schematic。

### 2.1.2. 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；
- **gd32vw55x\_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）；
- **gd32vw55x\_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）；
- **gd32vw55x\_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

**注意：**所有的例程的使用，都不受不同软件开发环境的影响。

### 2.1.3. 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：

- **RISCV子文件夹**:
  - **Core**子文件夹包含RISC-V内核的支持文件，用户无需修改该文件夹；
  - **env\_Eclipse**子文件夹包含了Eclipse环境中基于RISC-V内核处理器的启动代码、异常服务程序及链接脚本文件，用户无需修改该文件夹；
  - **env\_IAR**子文件夹包含了IAR环境中基于RISC-V内核处理器的启动代码、异常服务程序，用户无需修改该文件夹；
  - **env\_SES**子文件夹包含了SES环境中基于RISC-V内核处理器的启动代码、异常服务程序，用户无需修改该文件夹；
  - **stubs**子文件夹包含了\_write、\_read等桩函数的定义，用户无需修改该文件夹。
- **GD32VW55x\_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；
  - 基于GD32VW55x的全局头文件和系统配置文件，用户无需修改。

### 2.1.4. Template 文件夹

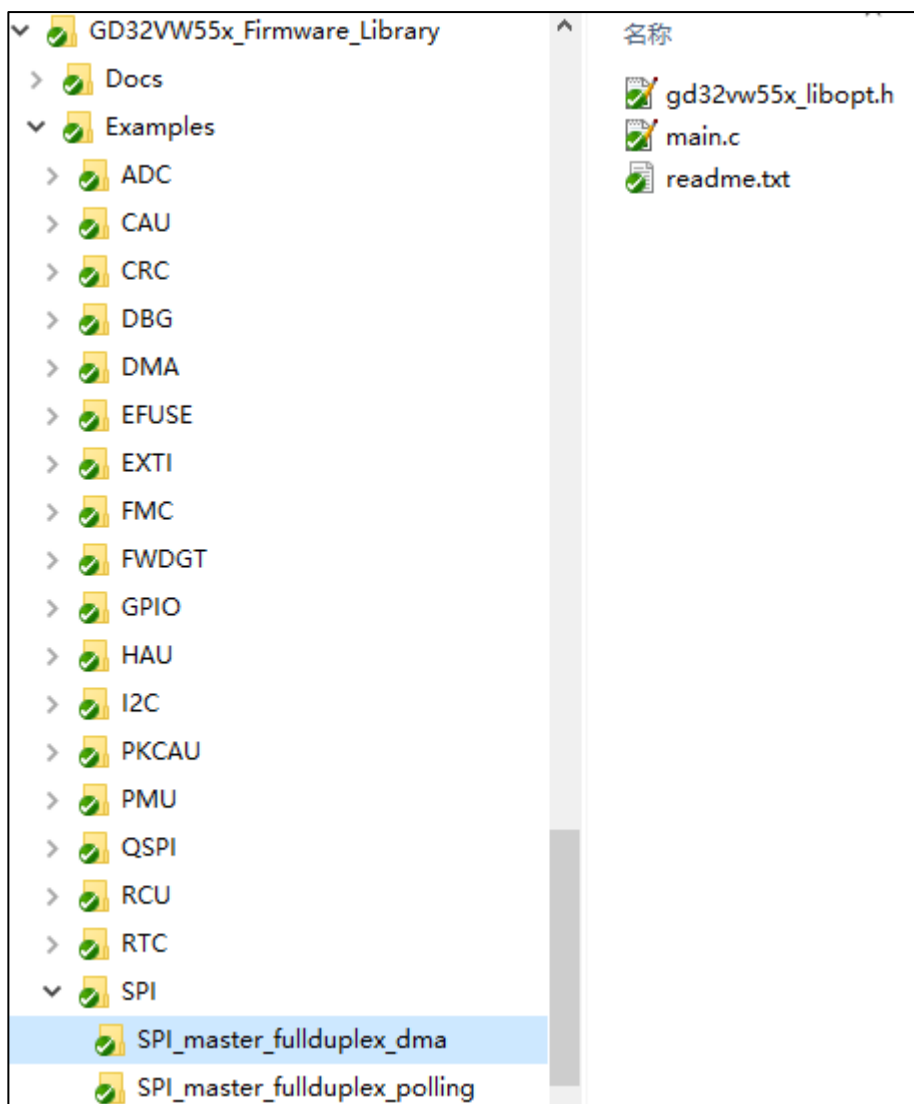
Template文件夹包含一个使用LED、USART打印及按键控制的简单例程（IAR\_project用于EWRISCV-3101版本编译环境，Eclipse\_project用于GD32Eclipse编译环境，SES\_project用于SEGGER Embedded Studio for RISC-V 7.32a编译环境）。用户可以使用该工程模板进行固件

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\_full duplex\_dma”，如下图所示：

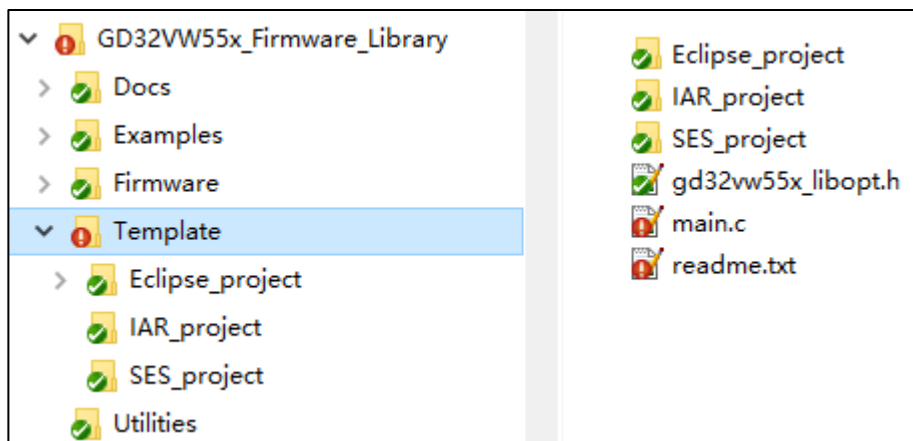
图 2-2. 选择外设例程文件



### 拷贝文件

打开“Template”文件夹，将“IAR\_project”，“Eclipse\_project”和“SES\_project”三个文件夹保留，其他文件都删除，然后将“SPI\_master\_full duplex\_dma”文件夹中的所有文件拷到“Template”文件夹子目录下，如下图所示：

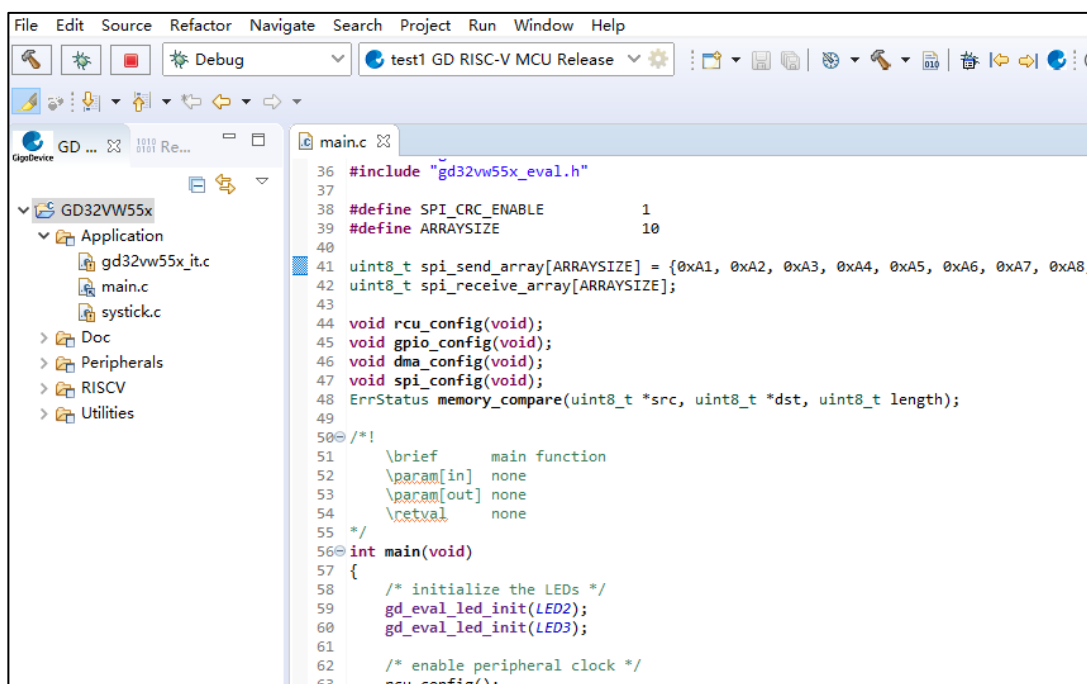
图 2-3. 拷贝外设例程文件



### 打开工程

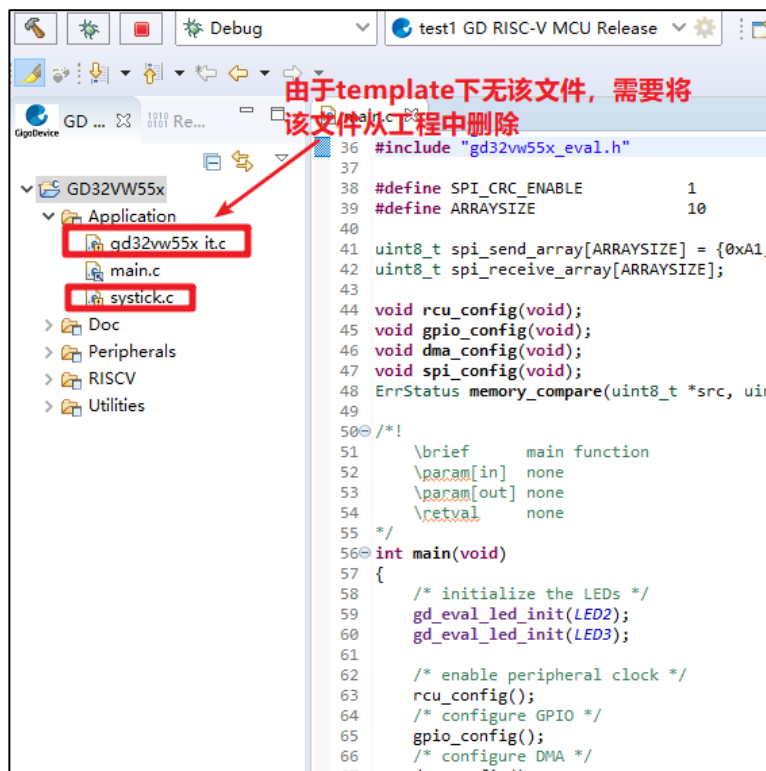
GD提供Eclipse, IAR和SES三种版本的工程, 根据客户所安装的软件, 打开不同的project, 如"Eclipse\_project", 打开GD32Eclipse, 导入Template中工程, 如下图所示:

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

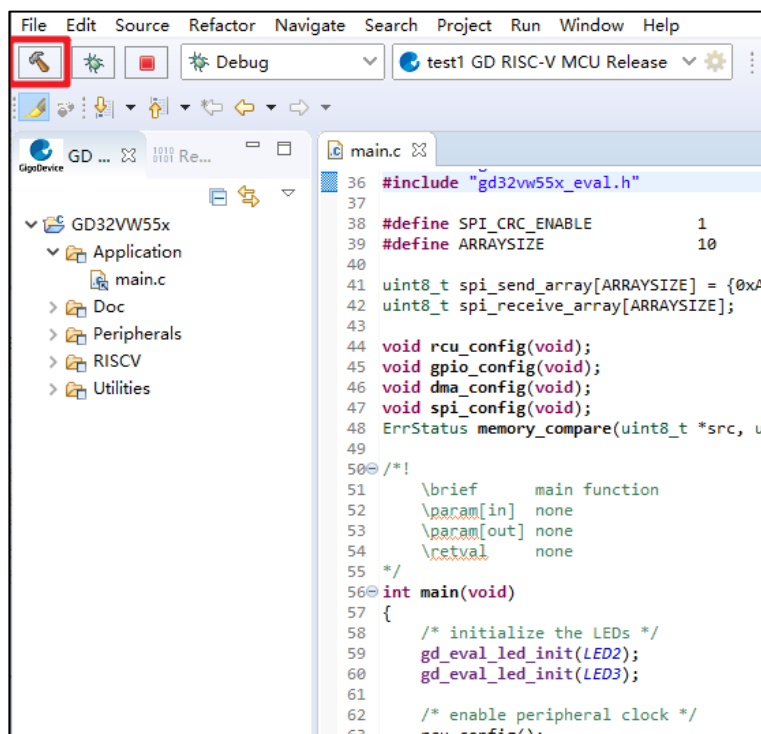


### 编译调试下载

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。编译操作可见下图，调试下载等IDE的具体使用，请参考相应的软件使用说明。



图 2-6. 编译



### 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：

- gd32vw55x\_eval.h文件是运行固件库例程所需关于评估板的头文件；
- gd32vw55x\_eval.c文件是运行固件库例程所需关于评估板的源文件。

## 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
gd32vw55x_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。
gd32vw55x_it.h	头文件，包含所有中断处理函数原形。
gd32vw55x_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。
gd32vw55x_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。
gd32vw55x_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。

文件名	描述
systick.c	systick配置与延时函数源文件。
readme.txt	固件库例程使用及配置说明文档。

## 3. 外设固件库

### 3.1. 外设固件库概述

外设固件库函数的描述格式如下表：

表 3-1. 外设固件库函数描述格式

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

### 3.2. ADC

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

#### 3.2.1. 外设寄存器描述

ADC寄存器列表如下表所示：

表 3-2. ADC 寄存器

寄存器名称	寄存器描述
ADC_STAT	状态寄存器
ADC_CTL0	控制寄存器0
ADC_CTL1	控制寄存器1
ADC_SAMPT0	采样时间寄存器0
ADC_SAMPT1	采样时间寄存器1
ADC_IOFFx(x=0..3)	注入通道数据偏移寄存器x (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	常规序列寄存器0
ADC_RSQ1	常规序列寄存器1

寄存器名称	寄存器描述
ADC_RSQ2	常规序列寄存器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	常规数据寄存器
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，在每个常规组转换结束时，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_tempsensor_vrefint_enable	温度传感器、内部参考电压通道使能
adc_tempsensor_vrefint_disable	温度传感器、内部参考电压通道禁能
adc_data_alignment_config	配置ADC数据对齐方式
adc_channel_length_config	配置常规通道组或注入通道组的长度
adc_routine_channel_config	配置ADC常规通道组
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_resolution_config	配置ADC分辨率
adc_routine_data_read	读ADC常规组数据寄存器
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_	配置ADC模拟看门狗在通道组有效

库函数名称	库函数描述
enable	
adc_watchdog_disable	ADC模拟看门狗禁能
adc_watchdog_threshold_config	配置ADC模拟看门狗阈值
adc_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

### 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	<code>adc_disable</code>
函数原形	<code>void adc_disable(void);</code>
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	<code>adc_dma_mode_enable</code>
函数原形	<code>void adc_dma_mode_enable(void);</code>
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，在每个常规组转换结束时，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 routine 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_ROUTINE_CHANNEL</code>	常规通道组
<code>ADC_INSERTED_CHANNEL</code>	注入通道组
<code>ADC_CHANNEL_</code>	常规通道组和注入通道组间断模式禁能

<i>DISCON_DISABLE</i>	
输入参数{in}	
<b>length</b>	间断模式下的转换数目，常规通道组取值为1..9，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_ROUTINE_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	<code>adc_special_function_config</code>
函数原形	<code>void adc_special_function_config(uint32_t function, ControlStatus newvalue);</code>
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}	
<b>function</b>	功能配置
<i>ADC_SCAN_MODE</i>	扫描模式选择
<i>ADC_INSERTED_CHANNEL_AUTO</i>	注入组自动转换
<i>ADC_CONTINUOUS_MODE</i>	连续模式选择
输入参数{in}	
<b>newvalue</b>	功能使能禁能
<i>ENABLE</i>	使能
<i>DISABLE</i>	禁能
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* enable ADC scan mode */
adc_special_function_config(ADC_SCAN_MODE, ENABLE);
```

## 函数 adc\_tempsensor\_vrefint\_enable

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

表 3-14. 函数 adc\_tempsensor\_vrefint\_enable

函数名称	adc_tempsensor_vrefint_enable
函数原形	void adc_tempsensor_vrefint_enable (void);
功能描述	ADC温度和内部参考电压使能
先决条件	-
被调用函数	-
输入参数{in}	
-	-
输出参数{out}	
-	-
返回值	
-	-

例如:

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

```
adc_tempsensor_vrefint_enable ();
```

## 函数 adc\_tempsensor\_vrefint\_disable

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

表 3-15. 函数 adc\_tempsensor\_vrefint\_disable

函数名称	adc_tempsensor_vrefint_disable
函数原形	void adc_tempsensor_vrefint_disable (void);
功能描述	ADC温度和内部参考电压禁能
先决条件	-
被调用函数	-
输入参数{in}	
-	-
输出参数{out}	
-	-
返回值	
-	-

例如:

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

```
adc_tempsensor_vrefint_disable ();
```

## 函数 `adc_data_alignment_config`

函数`adc_data_alignment_config`描述见下表：

表 3-16. 函数 `adc_data_alignment_config`

函数名称	<code>adc_data_alignment_config</code>
函数原形	<code>void adc_data_alignment_config(uint32_t 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}	
<code>data_alignment</code>	数据对齐方式选择
<code>ADC_DATAALIGN_RIGHT</code>	右对齐
<code>ADC_DATAALIGN_LEFT</code>	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 3-17. 函数 `adc_channel_length_config`

函数名称	<code>adc_channel_length_config</code>
函数原形	<code>void adc_channel_length_config(uint8_t adc_channel_group, uint32_t length);</code>
功能描述	配置常规通道组或注入通道组的长度
先决条件	-
被调用函数	-
输入参数{in}	
<code>adc_channel_group</code>	通道组选择
<code>ADC_ROUTINE_CHANNEL</code>	常规通道组
<code>ADC_INSERTED_CHANNEL</code>	注入通道组
输入参数{in}	
<code>length</code>	通道长度，常规通道组为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 routine channel */
```

```
adc_channel_length_config(ADC_ROUTINE_CHANNEL, 4);
```

### 函数 adc\_routine\_channel\_config

函数adc\_routine\_channel\_config描述见下表:

**表 3-18. 函数 adc\_routine\_channel\_config**

函数名称	adc_routine_channel_config
函数原形	void adc_routine_channel_config(uint8_t rank, uint8_t adc_channel, uint32_t sample_time);
功能描述	配置ADC常规通道组
先决条件	-
被调用函数	-
输入参数{in}	
rank	常规组通道序列，取值范围为0~8
输入参数{in}	
adc_channel	ADC通道选择
ADC_CHANNEL_x (x=0..10)	ADC通道x (x=0..10)
输入参数{in}	
sample_time	采样时间
ADC_SAMPLETIME_1POINT5	1.5周期
ADC_SAMPLETIME_2POINT5	2.5周期
ADC_SAMPLETIME_14POINT5	14.5周期
ADC_SAMPLETIME_27POINT5	27.5周期
ADC_SAMPLETIME_55POINT5	55.5周期
ADC_SAMPLETIME_83POINT5	83.5周期
ADC_SAMPLETIME_111POINT5	111.5周期
ADC_SAMPLETIME	143.5周期

_143POINT5	
ADC_SAMPLETIME _479POINT5	479.5周期
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* configure ADC routine channel */
```

```
adc_routine_channel_config(1, ADC_CHANNEL_0, ADC_SAMPLETIME_14POINT5);
```

### 函数 adc\_inserted\_channel\_config

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

表 3-19. 函数 adc\_inserted\_channel\_config

函数名称	adc_inserted_channel_config
函数原形	void adc_inserted_channel_config(uint8_t rank, uint8_t adc_channel, uint32_t sample_time);
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}	
rank	注入组通道序列, 取值范围为0~3
输入参数{in}	
adc_channel	ADC通道选择
ADC_CHANNEL_x (x=0..10)	ADC通道x (x=0..10)
输入参数{in}	
sample_time	采样时间
ADC_SAMPLETIME _1POINT5	1.5周期
ADC_SAMPLETIME _2POINT5	2.5周期
ADC_SAMPLETIME _14POINT5	14.5周期
ADC_SAMPLETIME _27POINT5	27.5周期
ADC_SAMPLETIME _55POINT5	55.5周期
ADC_SAMPLETIME _83POINT5	83.5周期

ADC_SAMPLETIME_111POINT5	111.5周期
ADC_SAMPLETIME_143POINT5	143.5周期
ADC_SAMPLETIME_479POINT5	479.5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_14POINT5);
```

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

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

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

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

例如：

```
/* configure 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 3-21. 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_ROUTINE_CHANNEL</code>	常规通道组
<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_TRIGGER_DISABLE</code>	外部触发禁能
<code>EXTERNAL_TRIGGER_RISING</code>	外部触发上升沿
<code>EXTERNAL_TRIGGER_FALLING</code>	外部触发下降沿
<code>EXTERNAL_TRIGGER_RISING_FALLING</code>	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 3-22. 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
<code>ADC_ROUTINE_CHANNEL</code>	常规通道组
<code>ADC_INSERTED_CHANNEL</code>	注入通道组
输入参数{in}	
<b>external_trigger_source</b>	常规通道组或注入通道组触发源
<code>ADC_EXTTRIG_ROUTINE_T0_CH0</code>	TIMER0 CH0事件（常规组）
<code>ADC_EXTTRIG_ROUTINE_T0_CH1</code>	TIMER0 CH1事件（常规组）
<code>ADC_EXTTRIG_ROUTINE_T0_CH2</code>	TIMER0 CH2事件（常规组）
<code>ADC_EXTTRIG_ROUTINE_T1_CH1</code>	TIMER1 CH1事件（常规组）
<code>ADC_EXTTRIG_ROUTINE_T1_CH2</code>	TIMER2 CH2事件（常规组）
<code>ADC_EXTTRIG_ROUTINE_T1_CH3</code>	TIMER1 CH3事件（常规组）
<code>ADC_EXTTRIG_ROUTINE_T1_TRGO</code>	TIMER1 TRGO事件（常规组）
<code>ADC_EXTTRIG_ROUTINE_T2_CH0</code>	TIMER2 CH0事件（常规组）
<code>ADC_EXTTRIG_ROUTINE_T2_TRGO</code>	TIMER2 TRGO事件（常规组）
<code>ADC_EXTTRIG_ROUTINE_EXTI_11</code>	外部中断线11（常规组）
<code>ADC_EXTTRIG_INSERTED_T0_CH3</code>	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）
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 routine channel external trigger source */
```

```
adc_external_trigger_source_config (ADC_ROUTINE_CHANNEL,  
ADC_EXTTRIG_ROUTINE_T0_CH0);
```

## 函数 adc\_software\_trigger\_enable

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

表 3-23. 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}	
adc_channel_group	通道组选择
ADC_ROUTINE_CHANNEL	常规通道组

ADC_INSERTED_CHANNEL	注入通道组
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* enable ADC routine channel group software trigger */
adc_software_trigger_enable(ADC_ROUTINE_CHANNEL);
```

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

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

表 3-24. 函数 adc\_end\_of\_conversion\_config

函数名称	adc_end_of_conversion_config
函数原形	void adc_end_of_conversion_config(uint8_t end_selection);
功能描述	转换模式结束配置
先决条件	-
被调用函数	-
输入参数{in}	
end_selection	转换结束模式
ADC_EOC_SET_SEQUENCE	只有在常规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
ADC_EOC_SET_CONVERSION	在每个常规组转换结束时，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\_resolution\_config

函数adc\_resolution\_config描述见下表：

表 3-25. 函数 adc\_resolution\_config

函数名称	adc_resolution_config
函数原形	void adc_resolution_config(uint32_t resolution);

功能描述	配置ADC分辨率
先决条件	-
被调用函数	-
输入参数{in}	
resolution	分辨率
ADC_RESOLUTION_12B	12位分辨率
ADC_RESOLUTION_10B	10位分辨率
ADC_RESOLUTION_8B	8位分辨率
ADC_RESOLUTION_6B	6位分辨率
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* configure the ADC resolution*/
adc_resolution_config(ADC_RESOLUTION_12B);
```

### 函数 adc\_routine\_data\_read

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

表 3-26. 函数 adc\_routine\_data\_read

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

例如：

```
/* read ADC routine group data register */
uint16_t adc_value = 0;
```

```
adc_value = adc_routine_data_read();
```

### 函数 `adc_inserted_data_read`

函数`adc_inserted_data_read`描述见下表：

**表 3-27. 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 3-28. 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(x=0..10)</code>	ADC通道x(x=0..10)
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 3-29. 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}	
<b>adc_channel_group</b>	通道组使用模拟看门狗
<code>ADC_ROUTINE_CHANNEL</code>	常规通道组
<code>ADC_INSERTED_CHANNEL</code>	注入通道组
<code>ADC_ROUTINE_INSERTED_CHANNEL</code>	常规和注入通道组
输出参数{out}	
-	-
返回值	
-	-

例如：

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

```
adc_watchdog_group_channel_enable(ADC_ROUTINE_CHANNEL);
```

### 函数 `adc_watchdog_disable`

函数`adc_watchdog_disable`描述见下表：

**表 3-30. 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 3-31. 函数 adc\_watchdog\_threshold\_config

函数名称	adc_watchdog_threshold_config
函数原形	void adc_watchdog_threshold_config(uint16_t low_threshold, uint16_t high_threshold);
功能描述	配置ADC模拟看门狗阈值
先决条件	-
被调用函数	-
输入参数{in}	
low_threshold	模拟看门狗低阈值，0..4095
输入参数{in}	
high_threshold	模拟看门狗高阈值，0..4095
输出参数{out}	
-	-
返回值	
-	-

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 3-32. 函数 adc\_oversample\_mode\_config

函数名称	adc_oversample_mode_config
函数原形	void adc_oversample_mode_config(uint32_t mode, uint16_t shift, uint8_t ratio);

功能描述	配置ADC过采样模式
先决条件	-
被调用函数	-
输入参数{in}	
mode	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
输入参数{in}	
shift	ADC过滤采样移位
ADC_OVERSAMPLING_SHIFT_NONE	不移位
ADC_OVERSAMPLING_SHIFT_1B	移1位
ADC_OVERSAMPLING_SHIFT_2B	移2位
ADC_OVERSAMPLING_SHIFT_3B	移3位
ADC_OVERSAMPLING_SHIFT_4B	移4位
ADC_OVERSAMPLING_SHIFT_5B	移5位
ADC_OVERSAMPLING_SHIFT_6B	移6位
ADC_OVERSAMPLING_SHIFT_7B	移7位
ADC_OVERSAMPLING_SHIFT_8B	移8位
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 3-33. 函数 adc\_oversample\_mode\_enable

函数名称	adc_oversample_mode_enable
函数原形	void adc_oversample_mode_enable(void);
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 3-34. 函数 adc\_oversample\_mode\_disable

函数名称	adc_oversample_mode_disable
函数原形	void adc_oversample_mode_disable(void);
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 3-35. 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}	
<b>flag</b>	ADC标志位
<code>ADC_FLAG_WDE</code>	模拟看门狗事件标志位
<code>ADC_FLAG_EOC</code>	组转换结束标志位
<code>ADC_FLAG_EOIC</code>	注入通道组转换结束标志位
<code>ADC_FLAG_STIC</code>	注入通道组转换开始标志位
<code>ADC_FLAG_STRC</code>	常规通道组转换开始标志位
<code>ADC_FLAG_ROVF</code>	常规组数据寄存器溢出标志位
输出参数{out}	
-	-
返回值	
<b>FlagStatus</b>	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 3-36. 函数 `adc_flag_clear`**

<b>函数名称</b>	<code>adc_flag_clear</code>
<b>函数原形</b>	<code>void adc_flag_clear(uint32_t flag);</code>
<b>功能描述</b>	清除ADC标志位
<b>先决条件</b>	-
<b>被调用函数</b>	-
输入参数{in}	
<b>flag</b>	ADC标志位
<code>ADC_FLAG_WDE</code>	模拟看门狗事件标志位
<code>ADC_FLAG_EOC</code>	组转换结束标志位
<code>ADC_FLAG_EOIC</code>	注入通道组转换结束标志位
<code>ADC_FLAG_STIC</code>	注入通道组转换开始标志位
<code>ADC_FLAG_STRC</code>	常规通道组转换开始标志位
<code>ADC_FLAG_ROVF</code>	常规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 3-37. 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	常规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 3-38. 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	常规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 3-39. 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_ROVF	常规组数据寄存器溢出中断标志位
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 3-40. 函数 `adc_interrupt_flag_clear`

函数名称	<code>adc_interrupt_flag_clear</code>
函数原形	<code>FlagStatus 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}	
<code>int_flag</code>	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>	常规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);
```

### 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
<code>CAU_CTL</code>	CAU控制寄存器
<code>CAU_STAT0</code>	CAU状态寄存器0
<code>CAU_DI</code>	CAU数据输入寄存器
<code>CAU_DO</code>	CAU数据输出寄存器
<code>CAU_DMAEN</code>	CAU DMA使能寄存器
<code>CAU_INTEN</code>	CAU中断使能寄存器

寄存器名称	寄存器描述
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_GCMCCMCTX Sx (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
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

库函数名称	库函数描述
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_flag_get	获取CAU标志状态
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

### 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	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 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：

```
/* initialize CAU encrypt and decrypt parameter struct */
```

```
cau_parameter_struct text;
```

```
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}	
dma_req	使能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：

```
/* 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}	
<b>dma_req</b>	除能CAU指定的DMA传输请求方向
<i>CAU_DMA_INFIFO</i>	DMA用于接收数据
<i>CAU_DMA_OUTFIFO</i>	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}	
<b>alg_dir</b>	算法方向
<i>CAU_ENCRYPT</i>	加密
<i>CAU_DECRYPT</i>	解密
输入参数{in}	
<b>algo_mode</b>	算法模式选择
<i>CAU_MODE_TDES_ECB</i>	TDES-ECB
<i>CAU_MODE_TDES_CBC</i>	TDES-CBC
<i>CAU_MODE_DES_ECB</i>	DES-ECB
<i>CAU_MODE_DES_CBC</i>	DES-CBC
<i>CAU_MODE_AES_ECB</i>	AES-ECB

CAU_MODE_AES_CB C	AES-CBC
CAU_MODE_AES_CT R	AES-CTR
CAU_MODE_AES_KEY Y	AES解密密钥准备模式
CAU_MODE_AES_GCM M	AES-GCM
CAU_MODE_AES_CCM M	AES-CCM
CAU_MODE_AES_CFB B	AES-CFB
CAU_MODE_AES_OFB B	AES-OFB
输入参数{in}	
swapping	数据交换选择
CAU_SWAPPING_32BIT T	无交换
CAU_SWAPPING_16BIT T	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;
```

```
key_initpara->key_0_high = 0x12345678;
```

```
key_initpara->key_0_low = 0x12345678;
```

```
key_initpara->key_1_high = 0x12345678;
```

```
key_initpara->key_1_low = 0x12345678;
```

```
key_initpara->key_2_high = 0x12345678;
```

```
key_initpara->key_3_low = 0x12345678;
```

```
key_initpara->key_3_high = 0x12345678;
```



```
key_initpara->key_3_low = 0x12345678;
```

```
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;
iv_initpara->iv_0_high = 0x12345678;
iv_initpara->iv_0_low = 0x12345678;
iv_initpara->iv_1_high = 0x12345678;
iv_initpara->iv_1_low = 0x12345678;
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 = DISABLE;
```

```
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 = 0U;
```

```
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}	
<b>cau_context</b>	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}	
<b>algo_dir</b>	算法方向
<i>CAU_ENCRYPT</i>	加密
<i>CAU_DECRYPT</i>	解密
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[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}	
<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 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}	
<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
<b>返回值</b>	
<b>ErrStatus</b>	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**

<b>函数名称</b>	cau_aes_gcm
<b>函数原形</b>	ErrStatus cau_aes_gcm(cau_parameter_struct *cau_parameter, uint8_t *output, uint8_t *tag);
<b>功能描述</b>	GCM模式下使用AES算法加密和解密
<b>先决条件</b>	-
<b>被调用函数</b>	-
<b>输入参数{in}</b>	
<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>
<b>输出参数{out}</b>	
<b>output</b>	指针指向返回数组
<b>输出参数{out}</b>	
<b>tag</b>	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}	

<b>tag_size</b>	标签字节长度
<b>输出参数{out}</b>	
<b>output</b>	指针指向返回数组
<b>输出参数{out}</b>	
<b>tag</b>	指针指向返回标签数组
<b>输出参数{out}</b>	
<b>aad_buf</b>	指针指向用户定义的用于填充附加身份验证数据的数组
<b>返回值</b>	
<b>ErrStatus</b>	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}	
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.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}	

<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>
<b>输出参数{out}</b>	
<b>output</b>	指针指向返回数组
<b>返回值</b>	
<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.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**

<b>函数名称</b>	cau_des_ecb
<b>函数原形</b>	ErrStatus cau_des_ecb(cau_parameter_struct *cau_parameter, uint8_t *output);
<b>功能描述</b>	ECB模式下使用DES算法加密和解密
<b>先决条件</b>	-
<b>被调用函数</b>	-
<b>输入参数{in}</b>	
<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>
<b>输出参数{out}</b>	
<b>output</b>	指针指向返回数组
<b>返回值</b>	
<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        = 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\_flag\_get

函数cau\_flag\_get描述见下表:

**表 3-78. 函数 cau\_flag\_get**

函数名称	cau_flag_get
函数原形	FlagStatus cau_flag_get(uint32_t flag);
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}	
flag	CAU标志状态
CAU_FLAG_INFIFO_EMPTY	输入FIFO空标志
CAU_FLAG_INFIFO_NO_FULL	输入FIFO未滿标志
CAU_FLAG_OUTFIFO_NO_EMPTY	输出FIFO非空标志
CAU_FLAG_OUTFIFO_FULL	输出FIFO滿标志
CAU_FLAG_BUSY	CAU内核忙标志
CAU_FLAG_INFIFO	输入FIFO标志状态
CAU_FLAG_OUTFIFO	输出FIFO标志状态
输出参数{out}	
-	-
返回值	
FlagStatus	SET或RESET

例如:

```

/* get the CAU flag status */

FlagStatus status = RESET;

status = cau_flag_get(CAU_FLAG_INFIFO_EMPTY);

```



## 函数 cau\_interrupt\_enable

函数cau\_interrupt\_enable描述见下表:

表 3-79. 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 3-80. 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 3-81. 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或RESET

例如：

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

```
FlagStatus status = RESET;
```

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

## 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	根据数据寄存器的复位值（0xFFFFFFFF）复位数据寄存器
crc_data_register_read	读数据寄存器
crc_free_data_register_read	读独立数据寄存器
crc_free_data_register_write	写独立数据寄存器
crc_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(CRC_DATA)为复位值 (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	
<b>uint32_t</b>	从数据寄存器读取的32位数据(0-0xFFFFFFFF)

例如:

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

```
uint32_t crc_value = 0;
```

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

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

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

表 3-87. 函数 **crc\_free\_data\_register\_read**

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

例如:

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

```
uint8_t crc_value = 0;
```

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

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

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

表 3-88. 函数 **crc\_free\_data\_register\_write**

函数名称	crc_free_data_register_write
函数原形	void crc_free_data_register_write(uint8_t free_data);
功能描述	写独立数据寄存器
先决条件	-
被调用函数	-
输入参数{in}	
<b>free_data</b>	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	crc_single_data_calculate
函数原形	uint32_t crc_single_data_calculate(uint32_t sdata);
功能描述	CRC计算一个32位数据
先决条件	-
被调用函数	-
输入参数{in}	
sdata	设定的32位数据
输出参数{out}	
-	-
返回值	
uint32_t	32位CRC计算结果(0-0xFFFFFFFF)

例如：

```
/* CRC calculate a 32-bit data */
uint32_t val = 0, valcrc = 0;
val = (uint32_t) 0xabcd1234;
valcrc = crc_single_data_calculate(val);
```

### 函数 crc\_block\_data\_calculate

函数crc\_block\_data\_calculate描述见下表：

表 3-90. 函数 crc\_block\_data\_calculate

函数名称	crc_block_data_calculate
函数原形	uint32_t crc_block_data_calculate(uint32_t array[], uint32_t size);
功能描述	CRC计算一个32位数组
先决条件	-
被调用函数	-

输入参数{in}	
array	32位数据数组的指针
输入参数{in}	
size	数组长度
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DBG寄存器
dbg_id_get	读DBG_ID寄存器

库函数名称	库函数描述
dbg_low_power_enable	使能低功耗模式的MCU调试保持功能
dbg_low_power_disable	禁能低功耗模式的MCU调试保持功能
dbg_periph_enable	使能外设的MCU调试保持功能
dbg_periph_disable	禁能外设的MCU调试保持功能

## 枚举类型 dbg\_periph\_enum

表 3-93. 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_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，用于调试
DBG_WWDGT_HOLD	当内核停止时，保持WWDGT计数器时钟
DBG_FWDGT_HOLD	当内核停止时，保持FWDGT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	dbg_deinit
函数原形	void dbg_deinit(void);
功能描述	复位DBG寄存器
先决条件	-
被调用函数	-
输入参数{in}	
-	-
输出参数{out}	
-	-
返回值	
-	-

例如：

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

```
dbg_deinit();
```



## 函数 dbg\_id\_get

函数dbg\_id\_get描述见下表：

表 3-95. 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：

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

```
dbg_low_power_enable(DBG_LOW_POWER_SLEEP);
```

### 函数 dbg\_low\_power\_disable

函数dbg\_low\_power\_disable描述见下表：

**表 3-97. 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：

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

```
dbg_low_power_disable(DBG_LOW_POWER_SLEEP);
```

### 函数 dbg\_periph\_enable

函数dbg\_periph\_enable描述见下表：

**表 3-98. 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}	
dbg_periph	外设参考 <a href="#">表3-93. 枚举类型dbg_periph_enum</a>
输出参数{out}	
-	-
返回值	
-	-

例如：

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

```
dbg_periph_enable(DBG_TIMER0_HOLD);
```

### 函数 dbg\_periph\_disable

函数dbg\_periph\_disable描述见下表：

**表 3-99. 函数 dbg\_periph\_disable**

函数名称	dbg_periph_disable
函数原形	void dbg_periph_disable(dbg_periph_enum dbg_periph);
功能描述	禁能外设的MCU调试保持功能
先决条件	-
被调用函数	-
输入参数{in}	
dbg_periph	外设参考 <a href="#">表3-93. 枚举类型dbg_periph_enum</a>
输出参数{out}	
-	-
返回值	
-	-

例如：

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

```
dbg_periph_disable(DBG_TIMER0_HOLD);
```

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

### 3.6.1. 外设寄存器说明

DMA寄存器列表如下表所示：

表 3-100. 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	中断标志位清除寄存器0
DMA_INTC1	中断标志位清除寄存器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
DMA_CHxM1ADDR (x=0..7)	通道x存储器1基地址寄存器
DMA_CHxFCTL (x=0..7)	通道x FIFO控制寄存器

### 3.6.2. 外设库函数说明

DMA库函数列表如下表所示：

表 3-101. DMA 库函数

库函数名称	库函数描述
dma_deinit	复位外设DMA通道x的所有寄存器
dma_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	DMA的通道x单数据传输模式初始化
dma_multi_data_mode_init	DMA的通道x突发传输模式初始化
dma_periph_address_config	DMA通道x传输的外设基地址配置
dma_memory_address_config	DMA通道x传输的存储器基地址配置
dma_transfer_number_config	配置DMA通道x需要传输的数据量
dma_transfer_number_get	获取DMA通道x剩余需要传输的数据量
dma_priority_config	DMA通道x的传输软件优先级配置
dma_memory_burst_beats_config	DMA通道x突发传输存储器节拍数配置
dma_periph_burst_beats_config	DMA通道x突发传输外设节拍数配置
dma_memory_width_config	DMA通道x传输的存储器数据宽度配置
dma_periph_width_config	DMA通道x传输的外设数据宽度配置
dma_memory_address_generation_config	DMA通道x存储器地址生成配置
dma_peripheral_address_generation_config	DMA通道x外设地址生成配置

库函数名称	库函数描述
<code>dma_circulation_enable</code>	DMA的通道x循环模式使能
<code>dma_circulation_disable</code>	DMA的通道x循环模式禁能
<code>dma_channel_enable</code>	外设DMA的通道x传输使能
<code>dma_channel_disable</code>	外设DMA的通道x传输禁能
<code>dma_transfer_direction_config</code>	DMA通道x的传输方向配置
<code>dma_switch_buffer_mode_config</code>	DMA通道x的存储切换模式配置
<code>dma_using_memory_get</code>	获取当前正在使用的存储地址
<code>dma_channel_subperipheral_select</code>	DMA通道x的外设请求选择
<code>dma_flow_controller_config</code>	DMA通道x的传输控制器选择
<code>dma_switch_buffer_mode_enable</code>	DMA通道x存储切换模式使能
<code>dma_switch_buffer_mode_disable</code>	DMA通道x存储切换模式禁能
<code>dma_fifo_status_get</code>	获取DMA通道x的FIFO状态
<code>dma_flag_get</code>	获取DMA通道x的标志位
<code>dma_flag_clear</code>	清除DMA通道x的标志位
<code>dma_interrupt_enable</code>	DMA通道x中断使能
<code>dma_interrupt_disable</code>	DMA通道x中断禁能
<code>dma_interrupt_flag_get</code>	获取DMA通道x的中断标志位
<code>dma_interrupt_flag_clear</code>	清除DMA通道x的中断标志位

### 枚举类型 `dma_channel_enum`

表 3-102. 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 3-103. 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

DMA_SUBPERI6	DMA外设6
DMA_SUBPERI7	DMA外设7

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

表 3-104. 结构体 dma\_multi\_data\_parameter\_struct

成员名称	功能描述
periph_addr	外设基地址
periph_width	外设数据传输宽度
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 3-105. 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 3-106. 函数 dma\_deinit

函数名称	dma_deinit
------	------------

函数原型	void dma_deinit(dma_channel_enum channelx);
功能描述	复位外设 DMA 的通道 x 的所有寄存器
先决条件	无
被调用函数	无
输入参数{in}	
channelx	DMA 通道
DMA_CHx(x=0..7)	DMA 通道选择, 参考 <a href="#">表 3-102. 枚举类型 dma_channel_enum</a>
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* DMA channel0 initialize */
dma_deinit(DMA_CH0);
```

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

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

表 3-107. 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 3-105. 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 3-108. 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 3-104. 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 3-109. 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(dma_channel_enum channelx, dma_single_data_parameter_struct* init_struct);
功能描述	DMA 的通道 x 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}	
channelx	DMA 通道
DMA_CHx(x=0..7)	DMA 通道选择，参考 <a href="#">表 3-102. 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 3-105. 结构体 dma_single_data_parameter_struct</a>
输出参数{out}	
-	-
返回值	
-	-

例如：



```

/* DMA channel0 single data mode initialize */

dma_single_data_parameter_struct dma_init_struct;

dma_deinit(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(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 3-110. 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(dma_channel_enum channelx, dma_multi_data_parameter_struct* init_struct);
功能描述	DMA 的通道 x 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}	
channelx	DMA 通道
DMA_CHx(x=0..7)	DMA 通道选择, 参考 <a href="#">表 3-102. 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 3-104. 结构体 dma_multi_data_parameter_struct</a>
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* DMA channel0 multi data mode initialize */

dma_multi_data_parameter_struct dma_init_parameter;

dma_multi_data_para_struct_init(&dma_init_parameter);

dma_deinit(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(DMA_CH0, &dma_init_parameter);
```

## 函数 dma\_periph\_address\_config

函数dma\_periph\_address\_config描述见下表：

**表 3-111. 函数 dma\_periph\_address\_config**

函数名称	dma_periph_address_config
函数原型	void dma_periph_address_config(dma_channel_enum channelx, uint32_t address);
功能描述	DMA 通道 x 传输的外设基地址配置
先决条件	无
被调用函数	无
输入参数{in}	
channelx	DMA 通道
DMA_CHx(x=0..7)	DMA 通道选择，参考 <a href="#">表 3-102. 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 DMA channel0 peripheral address */

#define USART_DATA_ADDR (USART0 + 0x00000024U)

dma_periph_address_config(DMA_CH0, USART_DATA_ADDR);
```

### 函数 dma\_memory\_address\_config

函数dma\_memory\_address\_config描述见下表：

表 3-112. 函数 dma\_memory\_address\_config

函数名称	dma_memory_address_config
函数原型	void dma_memory_address_config(dma_channel_enum channelx, uint8_t memory_flag, uint32_t address);
功能描述	DMA 通道 x 传输的存储器基地址配置
先决条件	无
被调用函数	无
输入参数{in}	
channelx	DMA 通道
DMA_CHx(x=0..7)	DMA 通道选择，参考 <a href="#">表 3-102. 枚举类型 dma_channel_enum</a>
输入参数{in}	
address	存储器基地址，0~0xFFFFFFFF
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* configure DMA channel0 memory0 address */

uint32_t array[10] = {0};

dma_memory_address_config(DMA_CH0, array);
```

### 函数 dma\_transfer\_number\_config

函数dma\_transfer\_number\_config描述见下表：

表 3-113. 函数 dma\_transfer\_number\_config

函数名称	dma_transfer_number_config
函数原型	void dma_transfer_number_config(dma_channel_enum channelx, uint32_t number);
功能描述	配置 DMA 通道 x 需要传输的数据量
先决条件	无
被调用函数	无
输入参数{in}	
channelx	DMA 通道
DMA_CHx(x=0..7)	DMA 通道选择, 参考 <a href="#">表 3-102. 枚举类型 dma_channel_enum</a>
输入参数{in}	
number	数据传输数量 (0x00000000 – 0x0000FFFF)
输出参数{out}	
-	-
返回值	
-	-

例如:

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

### 函数 dma\_transfer\_number\_get

函数dma\_transfer\_number\_get描述见下表:

表 3-114. 函数 dma\_transfer\_number\_get

函数名称	dma_transfer_number_get
函数原型	uint32_t dma_transfer_number_get(dma_channel_enum channelx);
功能描述	获取 DMA 通道 x 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}	
channelx	DMA 通道
DMA_CHx(x=0..7)	DMA 通道选择, 参考 <a href="#">表 3-102. 枚举类型 dma_channel_enum</a>
输出参数{out}	
-	-
返回值	
uint32_t	DMA 数据传输剩余数量 (0x00000000 – 0x0000FFFF)

例如:

```
/* get channel0 memory0 address */
```

```
uint32_t number = 0;
```

```
number = dma_transfer_number_get(DMA_CH0);
```

### 函数 dma\_priority\_config

函数dma\_priority\_config描述见下表：

**表 3-115. 函数 dma\_priority\_config**

函数名称	dma_priority_config
函数原型	void dma_priority_config(dma_channel_enum channelx, uint32_t priority);
功能描述	DMA 通道 x 的传输软件优先级配置
先决条件	无
被调用函数	无
输入参数{in}	
channelx	DMA 通道
DMA_CHx(x=0..7)	DMA 通道选择，参考 <a href="#">表 3-102. 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 DMA channel0 priority */
```

```
dma_priority_config(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 3-116. 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(dma_channel_enum channelx, uint32_t mbeat);

功能描述	DMA 通道 x 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}	
channelx	DMA 通道
DMA_CHx(x=0..7)	DMA 通道选择, 参考 <a href="#">表 3-102. 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 DMA channel0 transfer burst beats of memory */
dma_memory_burst_beats_config(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 3-117. 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 (dma_channel_enum channelx, uint32_t pbeat);
功能描述	DMA 通道 x 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}	
channelx	DMA 通道
DMA_CHx(x=0..7)	DMA 通道选择, 参考 <a href="#">表 3-102. 枚举类型 dma_channel_enum</a>
输入参数{in}	
pbeat	突发传输节拍数
DMA_PERIPH_BURST_SINGLE	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
输出参数{out}	
-	-
返回值	
-	-

例如：

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

```
dma_periph_burst_beats_config(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 3-118. 函数 dma\_memory\_width\_config

函数名称	dma_memory_width_config
函数原型	void dma_memory_width_config(dma_channel_enum channelx, uint32_t msize);
功能描述	DMA 通道 x 传输的存储器数据宽度配置
先决条件	无
被调用函数	无
输入参数{in}	
channelx	DMA 通道
<i>DMA_CHx(x=0..7)</i>	DMA 通道选择，参考 <a href="#">表 3-102. 枚举类型 dma_channel_enum</a>
输入参数{in}	
mwidth	存储器数据传输宽度
<i>DMA_MEMORY_WIDTH_8BIT</i>	8 位数据传输宽度
<i>DMA_MEMORY_WIDTH_16BIT</i>	16 位数据传输宽度
<i>DMA_MEMORY_WIDTH_32BIT</i>	32 位数据传输宽度
输出参数{out}	
-	-
返回值	
-	-

例如：

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

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

### 函数 dma\_periph\_width\_config

函数dma\_periph\_width\_config描述见下表：

**表 3-119. 函数 dma\_periph\_width\_config**

函数名称	dma_periph_width_config
函数原型	void dma_periph_width_config (dma_channel_enum channelx, uint32_t psize);
功能描述	DMA 通道 x 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}	
channelx	DMA 通道
DMA_CHx(x=0..7)	DMA 通道选择，参考 <a href="#">表 3-102. 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 DMA channel0 transfer peripheral width */
```

```
dma_periph_width_config(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 3-120. 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(dma_channel_enum channelx, uint8_t generation_algorithm);
功能描述	DMA 通道 x 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}	
channelx	DMA 通道
DMA_CHx(x=0..7)	DMA 通道选择, 参考 <a href="#">表 3-102. 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 DMA channel0 memory address generation algorithm as memory increase*/
dma_memory_address_generation_config(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 3-121. 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(dma_channel_enum channelx, uint8_t generation_algorithm);
功能描述	DMA 通道 x 外设地址生成配置
先决条件	无
被调用函数	无
输入参数{in}	
channelx	DMA 通道
DMA_CHx(x=0..7)	DMA 通道选择, 参考 <a href="#">表 3-102. 枚举类型 dma_channel_enum</a>
输入参数{in}	
generation_algorithm	地址生成模式
DMA_PERIPH_INCREASE_ENABLE	地址增长模式
DMA_PERIPH_INCREASE_DISABLE	固定地址模式
DMA_PERIPH_INCREMENT_FIX	外设地址固定增量 (4)

输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* configure DMA channel0 memory address generation algorithm as memory increase*/
dma_memory_address_generation_config(DMA_CH0,DMA_MEMORY_INCREASE_ENABLE);
```

### 函数 dma\_circulation\_enable

函数dma\_circulation\_enable描述见下表：

**表 3-122. 函数 dma\_circulation\_enable**

函数名称	dma_circulation_enable
函数原型	void dma_circulation_enable(dma_channel_enum channelx);
功能描述	DMA 的通道 x 循环模式使能
先决条件	无
被调用函数	无
输入参数{in}	
channelx	DMA 通道
DMA_CHx(x=0..7)	DMA 通道选择，参考 <a href="#">表 3-102. 枚举类型 dma_channel_enum</a>
输出参数{out}	
-	-
返回值	
-	-

例如：

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

### 函数 dma\_circulation\_disable

函数dma\_circulation\_disable描述见下表：

**表 3-123. 函数 dma\_circulation\_disable**

函数名称	dma_circulation_disable
函数原型	void dma_circulation_disable(dma_channel_enum channelx);
功能描述	DMA 的通道 x 循环模式禁能
先决条件	无
被调用函数	无
输入参数{in}	
channelx	DMA 通道

DMA_CHx(x=0..7)	DMA 通道选择, 参考 <a href="#">表 3-102. 枚举类型 dma_channel_enum</a>
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* disable DMA channel0 circulation mode */
```

```
dma_circulation_disable(DMA_CH0);
```

### 函数 dma\_channel\_enable

函数dma\_channel\_enable描述见下表:

**表 3-124. 函数 dma\_channel\_enable**

函数名称	dma_channel_enable
函数原型	void dma_channel_enable(dma_channel_enum channelx);
功能描述	外设 DMA 的通道 x 传输使能
先决条件	无
被调用函数	无
输入参数{in}	
channelx	DMA 通道
DMA_CHx(x=0..7)	DMA 通道选择, 参考 <a href="#">表 3-102. 枚举类型 dma_channel_enum</a>
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* enable DMA channel0 */
```

```
dma_channel_enable(DMA_CH0);
```

### 函数 dma\_channel\_disable

函数dma\_channel\_disable描述见下表:

**表 3-125. 函数 dma\_channel\_disable**

函数名称	dma_channel_disable
函数原型	void dma_channel_disable(dma_channel_enum channelx);
功能描述	外设 DMA 的通道 x 传输禁能
先决条件	无
被调用函数	无
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 3-102. 枚举类型 dma_channel_enum</a>
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* disable DMA channel0 */
```

```
dma_channel_disable(DMA_CH0);
```

### 函数 dma\_transfer\_direction\_config

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

**表 3-126. 函数 dma\_transfer\_direction\_config**

<b>函数名称</b>	dma_transfer_direction_config
<b>函数原型</b>	void dma_transfer_direction_config(dma_channel_enum channelx, uint8_t direction);
<b>功能描述</b>	DMA 通道 x 的传输方向配置
<b>先决条件</b>	无
<b>被调用函数</b>	无
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 3-102. 枚举类型 dma_channel_enum</a>
输入参数{in}	
<b>direction</b>	数据传输方向
<i>DMA_PERIPHERAL_TO_MEMORY</i>	读取外设中数据, 写入存储器
<i>DMA_MEMORY_TO_PERIPHERAL</i>	读取存储器中数据, 写入外设
<i>DMA_MEMORY_TO_MEMORY</i>	读取存储器中数据, 写入存储器
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* configure DMA channel0 transfer direction as peripheral to memory mode*/
```

```
dma_transfer_direction_config(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 3-127. 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(dma_channel_enum channelx, uint32_t memory1_addr, uint32_t memory_select);
功能描述	DMA 通道 x 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}	
channelx	DMA 通道
DMA_CHx(x=0..7)	DMA 通道选择，参考 <a href="#">表 3-102. 枚举类型 dma_channel_enum</a>
输入参数{in}	
memory1_addr	存储器 1 基地址
输入参数{in}	
memory_select	选择存储器缓冲区
DMA_MEMORY_0	选择存储器 0 作为传输区域
DMA_MEMORY_1	选择存储器 1 作为传输区域
输出参数{out}	
-	-
返回值	
-	-

例如：

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

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

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

## 函数 dma\_using\_memory\_get

函数dma\_using\_memory\_get描述见下表：

表 3-128. 函数 dma\_using\_memory\_get

函数名称	dma_using_memory_get
函数原型	uint32_t dma_using_memory_get(dma_channel_enum channelx);
功能描述	获取当前正在使用的存储地址
先决条件	无
被调用函数	无
输入参数{in}	
channelx	DMA 通道
DMA_CHx(x=0..7)	DMA 通道选择，参考 <a href="#">表 3-102. 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 DMA channel0 transfer buffer currently used */
```

```
uint32_t buf_addr = DMA_MEMORY_0;
```

```
buf_addr = dma_using_memory_get(DMA_CH0);
```

### 函数 dma\_channel\_subperipheral\_select

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

表 3-129. 函数 dma\_channel\_subperipheral\_select

函数名称	dma_channel_subperipheral_select
函数原型	void dma_channel_subperipheral_select(dma_channel_enum channelx, dma_subperipheral_enum sub_periph);
功能描述	DMA 通道 x 的外设请求选择
先决条件	无
被调用函数	无
输入参数{in}	
channelx	DMA 通道
DMA_CHx(x=0..7)	DMA 通道选择，参考 <a href="#">表 3-102. 枚举类型 dma_channel_enum</a>
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* configure DMA channel0 peripheral */
```

```
dma_channel_subperipheral_select(DMA_CH0, DMA_SUBPERI1);
```

### 函数 dma\_flow\_controller\_config

函数dma\_flow\_controller\_config描述见下表：

表 3-130. 函数 dma\_flow\_controller\_config

函数名称	dma_flow_controller_config
函数原型	void dma_flow_controller_config(dma_channel_enum channelx, uint32_t controller);

功能描述	DMA 通道 x 的传输控制器选择
先决条件	无
被调用函数	无
输入参数{in}	
channelx	DMA 通道
DMA_CHx(x=0..7)	DMA 通道选择, 参考 <a href="#">表 3-102. 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 DMA channel0 flow controller */
```

```
dma_flow_controller_config(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 3-131. 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(dma_channel_enum channelx);
功能描述	DMA 通道 x 存储切换模式使能
先决条件	无
被调用函数	无
输入参数{in}	
channelx	DMA 通道
DMA_CHx(x=0..7)	DMA 通道选择, 参考 <a href="#">表 3-102. 枚举类型 dma_channel_enum</a>
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* enable DMA channel0 switch buffer mode */
```

```
dma_switch_buffer_mode_enable(DMA_CH0);
```

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

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

表 3-132. 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(dma_channel_enum channelx);
功能描述	DMA 通道 x 存储切换模式禁能
先决条件	无
被调用函数	无
输入参数{in}	
channelx	DMA 通道
DMA_CHx(x=0..7)	DMA 通道选择，参考 <a href="#">表 3-102. 枚举类型 dma_channel_enum</a>
输出参数{out}	
-	-
返回值	
-	-

例如：

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

## 函数 dma\_fifo\_status\_get

函数dma\_fifo\_status\_get描述见下表：

表 3-133. 函数 dma\_fifo\_status\_get

函数名称	dma_fifo_status_get
函数原型	uint32_t dma_fifo_status_get(dma_channel_enum channelx);
功能描述	获取 DMA 通道 x 的 FIFO 状态
先决条件	无
被调用函数	无
输入参数{in}	
channelx	DMA 通道
DMA_CHx(x=0..7)	DMA 通道选择，参考 <a href="#">表 3-102. 枚举类型 dma_channel_enum</a>
输出参数{out}	
-	-
返回值	
uint32_t	当前 FIFO 使用情况

例如：

```
/* get DMA channel0 FIFO status */
uint32_t fifo_state = 0;
```



```
fifo_state = dma_fifo_status_get(DMA_CH0);
```

### 函数 dma\_flag\_get

函数dma\_flag\_get描述见下表:

**表 3-134. 函数 dma\_flag\_get**

函数名称	dma_flag_get
函数原型	FlagStatus dma_flag_get(dma_channel_enum channelx, uint32_t flag);
功能描述	获取 DMA 通道的标志位
先决条件	无
被调用函数	无
输入参数{in}	
channelx	DMA 通道
DMA_CHx(x=0..7)	DMA 通道选择, 参考 <a href="#">表 3-102. 枚举类型 dma_channel_enum</a>
输入参数{in}	
flag	DMA 标志
DMA_FLAG_FEE	FIFO 异常标志位
DMA_FLAG_SDE	单数据传输异常标志位
DMA_FLAG_TAE	传输错误标志位
DMA_FLAG_HTF	半传输完成标志位
DMA_FLAG_FTF	传输完成标志位
输出参数{out}	
-	-
返回值	
FlagStatus	SET 或 RESET

例如:

```
/* check DMA channel0 full transfer finish flag set or not */
FlagStatus state = RESET;
state = dma_flag_get(DMA_CH0, DMA_FLAG_FTF);
```

### 函数 dma\_flag\_clear

函数dma\_flag\_clear描述见下表:

**表 3-135. 函数 dma\_flag\_clear**

函数名称	dma_flag_clear
函数原型	void dma_flag_clear(dma_channel_enum channelx, uint32_t flag);
功能描述	清除 DMAx 通道 y 标志位状态
先决条件	无
被调用函数	无
输入参数{in}	

<b>channelx</b>	DMA 通道
<i>DMA_CHx(x=0..7)</i>	DMA 通道选择, 参考 <a href="#">表 3-102. 枚举类型 dma_channel_enum</a>
输入参数{in}	
<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
输出参数{out}	
-	-
返回值	
-	-

例如:

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

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

## 函数 dma\_interrupt\_enable

函数dma\_interrupt\_enable描述见下表:

表 3-136. 函数 dma\_interrupt\_enable

<b>函数名称</b>	dma_interrupt_enable
<b>函数原型</b>	void dma_interrupt_enable(dma_channel_enum channelx, uint32_t interrupt);
<b>功能描述</b>	DMA 通道 x 中断使能
<b>先决条件</b>	无
<b>被调用函数</b>	无
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 3-102. 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 DMA channel0 full transfer finish interrupt */
dma_interrupt_enable(DMA_CH0, DMA_INT_FTF);
```

### 函数 dma\_interrupt\_disable

函数dma\_interrupt\_disable描述见下表：

**表 3-137. 函数 dma\_interrupt\_disable**

函数名称	dma_interrupt_disable
函数原型	void dma_interrupt_disable(dma_channel_enum channelx,uint32_t interrupt);
功能描述	DMA 通道 x 中断禁能
先决条件	无
被调用函数	无
输入参数{in}	
channelx	DMA 通道
DMA_CHx(x=0..7)	DMA 通道选择，参考 <a href="#">表 3-102. 枚举类型 dma_channel_enum</a>
输入参数{in}	
source	DMA 中断源
DMA_INT_FEE	FIFO 异常中断
DMA_INT_SDE	单数据传输异常中断
DMA_INT_TAE	传输错误中断
DMA_INT_HTF	半传输完成中断
DMA_INT_FTF	传输完成中断
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* disable DMA channel0 full transfer finish interrupt */
dma_interrupt_disable(DMA_CH0, DMA_INT_FTF);
```

### 函数 dma\_interrupt\_flag\_get

函数dma\_interrupt\_flag\_get描述见下表：

**表 3-138. 函数 dma\_interrupt\_flag\_get**

函数名称	dma_interrupt_flag_get
函数原型	FlagStatus dma_interrupt_flag_get(dma_channel_enum channelx, uint32_t int_flag);
功能描述	获取 DMA 通道 x 的中断标志位
先决条件	无

被调用函数	无
输入参数{in}	
channelx	DMA 通道
DMA_CHx(x=0..7)	DMA 通道选择, 参考 <a href="#">表 3-102. 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 DMA channel0 full transfer finish interrupt flag set or not */
```

```
FlagStatus state = RESET;
```

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

### 函数 dma\_interrupt\_flag\_clear

函数dma\_interrupt\_flag\_clear描述见下表:

表 3-139. 函数 dma\_interrupt\_flag\_clear

函数名称	dma_interrupt_flag_clear
函数原型	void dma_interrupt_flag_clear(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}	
channelx	DMA 通道
DMA_CHx(x=0..7)	DMA 通道选择, 参考 <a href="#">表 3-102. 枚举类型 dma_channel_enum</a>
输入参数{in}	
int_flag	DMA 标志
DMA_INT_FLAG_F	FIFO 异常中断标志位

<i>EE</i>	
<i>DMA_INT_FLAG_S</i> <i>DE</i>	单数据传输异常中断标志位
<i>DMA_INT_FLAG_T</i> <i>AE</i>	传输错误中断标志位
<i>DMA_INT_FLAG_H</i> <i>TF</i>	半传输完成中断标志位
<i>DMA_INT_FLAG_F</i> <i>TF</i>	传输完成中断标志位
输出参数{out}	
-	-
返回值	
-	-

例如：

```

/* clear DMA channel0 full transfer finish interrupt flag */
if(dma_interrupt_flag_get(DMA_CH3, DMA_INT_FLAG_FTF)){
    dma_interrupt_flag_clear(DMA_CH3, DMA_INT_FLAG_FTF);
}

```

## 3.7. ECLIC

RISC-V集成了改进型内核中断控制器(ECLIC)来实现高效的异常和中断处理。ECLIC旨在为RISC-V系统提供低延迟、矢量化、抢占式的中断。章节[3.7.1](#)对ECLIC库函数进行说明。

### 3.7.1. 外设库函数说明

ECLIC库函数列表如下表所示：

表 3-140. ECLIC 库函数

库函数名称	库函数描述
eclic_global_interrupt_enable	使能全局中断
eclic_global_interrupt_disable	禁能全局中断
eclic_level_threshold_set	设置机器模式下中断等级阈值
eclic_priority_group_set	设置优先级组
eclic_irq_enable	使能ECLIC的中断
eclic_irq_disable	禁能ECLIC的中断
eclic_system_reset	系统复位

## 枚举类型 IRQn\_Type

表 3-141. 枚举类型 IRQn\_Type

成员名称	功能描述
CLIC_INT_SFT	Software interrupt
CLIC_INT_TMR	CPU Timer interrupt
WWDGT_IRQn	窗口看门狗中断
LVD_IRQn	连接到 EXTI 线的 LVD 中断
TAMPER_STAMP_IRQn	RTC 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
DMA_Channel0_IRQn	DMA 通道 0 全局中断
DMA_Channel1_IRQn	DMA 通道 1 全局中断
DMA_Channel2_IRQn	DMA 通道 2 全局中断
DMA_Channel3_IRQn	DMA 通道 3 全局中断
DMA_Channel4_IRQn	DMA 通道 4 全局中断
DMA_Channel5_IRQn	DMA 通道 5 全局中断
DMA_Channel6_IRQn	DMA 通道 6 全局中断
DMA_Channel7_IRQn	DMA 通道 7 全局中断
ADC_IRQn	ADC 中断
EXTI5_9_IRQn	EXTI 线 5-9 中断
TIMER0_BRK_IRQn	TIMER0 中止中断
TIMER0_UP_IRQn	TIMER0 更新中断
TIMER0_CMT_IRQn	TIMER0 触发与通道换相中断
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
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
SPI_IRQn	SPI 全局中断
USART0_IRQn	USART0 全局中断
USART1_IRQn	UART1 全局中断
USART2_IRQn	UART2 全局中断
EXTI10_15_IRQn	EXTI 线 10-15 中断

成员名称	功能描述
CLIC_INT_SFT	Software interrupt
RTC_Alarm_IRQn	RTC 闹钟中断
VLVDF_IRQn	VLVD 中断
TIMER15_IRQn	TIMER15 全局中断
TIMER16_IRQn	TIMER16 全局中断
I2C0_WKUP_IRQn	I2C0 唤醒中断
USART0_WKUP_IRQn	USART0 唤醒中断
TIMER5_IRQn	TIMER5 全局中断
WIFI_TRIGGER_IRQn	WIFI 协议触发中断
WIFI_MAC_IRQn	WIFI MAC 中断
WIFI_TX_IRQn	WIFI 发送中断
WIFI_RX_IRQn	WIFI 接收中断
LA_IRQn	LA 中断
WIFI_WKUP_IRQn	WIFI 唤醒中断
BLE_WKUP_IRQn	BLE 唤醒中断
PLATFORM_WAKE_IRQn	Platform (PLF) 唤醒中断
ISO_BT_STAMP0_IRQn	ISO 蓝牙时间戳中断 0
ISO_BT_STAMP1_IRQn	ISO 蓝牙时间戳中断 1
ISO_BT_STAMP2_IRQn	ISO 蓝牙时间戳中断 2
ISO_BT_STAMP3_IRQn	ISO 蓝牙时间戳中断 3
ISO_BT_STAMP4_IRQn	ISO 蓝牙时间戳中断 4
ISO_BT_STAMP5_IRQn	ISO 蓝牙时间戳中断 5
ISO_BT_STAMP6_IRQn	ISO 蓝牙时间戳中断 6
ISO_BT_STAMP7_IRQn	ISO 蓝牙时间戳中断 7
PMU_IRQn	PMU 中断
CAU_IRQn	CAU 全局中断
HAU_TRNG_IRQn	HAU / TRNG 全局中断
WIFI_INT_IRQn	WIFI 中断
WIFI_SW_TRIG_IRQn	SW 触发中断
WIFI_FINE_TIMER_TARGET_IRQn	Fine 定时器目标中断
WIFI_STAMP_TARGET1_IRQn	时间戳目标 1 中断
WIFI_STAMP_TARGET2_IRQn	时间戳目标 2 中断
WIFI_STAMP_TARGET3_IRQn	时间戳目标 3 中断
WIFI_ENCRYPTION_ENGINE_IRQn	加密引擎中断
WIFI_SLEEP_MODE_IRQn	睡眠模式中断
WIFI_HALF_SLOT_IRQn	Half slot 中断
WIFI_FIFO_ACTIVITY_IRQn	FIFO 活动中断
WIFI_ERROR_IRQn	错误中断
WIFI_FREQ_SELECT_IRQn	频率选择中断
EFUSE_IRQn	EFUSE 全局中断
QSPI_IRQn	QSPI 全局中断

成员名称	功能描述
CLIC_INT_SFT	Software interrupt
PKCAU_IRQn	PKCAU 全局中断

### 函数 eclic\_global\_interrupt\_enable

函数eclic\_global\_interrupt\_enable描述见下表：

表 3-142. 函数 eclic\_global\_interrupt\_enable

函数名称	eclic_global_interrupt_enable
函数原形	void eclic_global_interrupt_enable(void);
功能描述	使能全局中断
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 the global interrupt */
eclic_global_interrupt_enable();
```

### 函数 eclic\_global\_interrupt\_disable

函数eclic\_global\_interrupt\_disable描述见下表：

表 3-143. 函数 eclic\_global\_interrupt\_disable

函数名称	eclic_global_interrupt_disable
函数原形	void eclic_global_interrupt_disable(void);
功能描述	禁能全局中断
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 the global interrupt */
```



```
eclic_global_interrupt_disable();
```

## 函数 eclic\_level\_threshold\_set

函数eclic\_level\_threshold\_set描述见下表：

**表 3-144. 函数 eclic\_level\_threshold\_set**

函数名称	eclic_level_threshold_set
函数原形	void eclic_level_threshold_set(uint8_t threshold);
功能描述	设置机器模式下中断等级阈值
先决条件	-
被调用函数	-
输入参数{in}	
threshold	阈值等级（0 ~ 15）
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* set machine mode interrupt level threshold */
eclic_level_threshold_set(0);
```

## 函数 eclic\_priority\_group\_set

函数eclic\_priority\_group\_set描述见下表：

**表 3-145. 函数 eclic\_priority\_group\_set**

函数名称	eclic_priority_group_set
函数原形	void eclic_priority_group_set(uint8_t prigroup);
功能描述	设置优先级组
先决条件	-
被调用函数	-
输入参数{in}	
prigroup	特定的优先级组
ECLIC_PRIGROUP_LEVEL0_PRIO4	0位用于级别（Level），4位用于优先级（Priority）
ECLIC_PRIGROUP_LEVEL1_PRIO3	1位用于级别（Level），3位用于优先级（Priority）
ECLIC_PRIGROUP_LEVEL2_PRIO2	2位用于级别（Level），2位用于优先级（Priority）
ECLIC_PRIGROUP_LEVEL3_PRIO1	3位用于级别（Level），1位用于优先级（Priority）
ECLIC_PRIGROUP	4位用于级别（Level），0位用于优先级（Priority）

_LEVEL4_PRI00	
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* configure the priority group */
eclic_priority_group_set(ECLIC_PRIGROUP_LEVEL0_PRI04);
```

### 函数 eclic\_irq\_enable

函数eclic\_irq\_enable描述见下表：

表 3-146. 函数 eclic\_irq\_enable

函数名称	eclic_irq_enable
函数原形	void eclic_irq_enable(IRQn_Type source, uint8_t level, uint8_t priority);
功能描述	使能ECLIC的中断
先决条件	-
被调用函数	-
输入参数{in}	
source	中断请求，详见 <a href="#">表 3-141. 枚举类型IRQn_Type</a>
输入参数{in}	
level	需要设置的级别值（最大为15，具体取决于优先级组）
输入参数{in}	
priority	需要设置的优先级值（最大为15，具体取决于优先级组）
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* enable and set key EXTI interrupt to the specified priority */
eclic_global_interrupt_enable();
eclic_priority_group_set(ECLIC_PRIGROUP_LEVEL3_PRI01);
eclic_irq_enable(EXTI10_15_IRQn, 1, 1);
```

### 函数 eclic\_irq\_disable

函数eclic\_irq\_disable描述见下表：

表 3-147. 函数 eclic\_irq\_disable

函数名称	eclic_irq_disable
函数原形	void eclic_irq_disable(IRQn_Type source);

功能描述	禁能ECLIC的中断
先决条件	-
被调用函数	-
输入参数{in}	
source	中断请求, 详见 <a href="#">表 3-141. 枚举类型IRQn_Type</a>
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* disable the interrupt request */
eclic_irq_disable(EXTI10_15_IRQn);
```

### 函数 eclic\_system\_reset

函数eclic\_system\_reset描述见下表:

表 3-148. 函数 eclic\_system\_reset

函数名称	eclic_system_reset
函数原形	void eclic_system_reset(void);
功能描述	系统复位
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 system */
eclic_system_reset();
```

## 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 3-149. 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_CTL0	控制寄存器 0
EFUSE_CTL1	控制寄存器 1
EFUSE_FPCTL	安全保护控制寄存器
EFUSE_USERCTL	用户控制寄存器
EFUSE_RESx(x = 0...2)	保留寄存器 x (x = 0...2)
EFUSE_AESKEYx(x = 0...3)	固件 AES 密钥寄存器 x (x = 0...3)
EFUSE_ROTpkKEYx(x = 0...7)	RoTPK 密钥寄存器 x (x = 0...7)
EFUSE_PUIDx(x = 0...3)	产品 UID 寄存器 x (x = 0...3)
EFUSE_HUKKEYx(x = 0...3)	HUK 密钥寄存器 x (x = 0...3)
EFUSE_USER_DATAx(x = 0...7)	用户数据寄存器 x(x = 0...7)
EFUSE_BOOTADDR	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 3-150. 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_control1_config	配置 EFUSE 中 control1 控制字段
efuse_fp_config	配置 EFUSE 中安全保护控制字段
efuse_user_control_config	写 EFUSE 中用户控制字段
efuse_res_write	写 EFUSE 中 MCU 保留字段
efuse_aes_key_write	写 EFUSE 中 AES 密钥字段
efuse_rotpk_key_write	写 EFUSE 中 RoTPK 密钥字段
efuse_user_data_write	写 EFUSE 中用户数据字段
efuse_res_read	读 EFUSE 中 MCU 保留字段
efuse_aes_key_read	读 EFUSE 中 AES 密钥字段
efuse_rotpk_key_read	读 EFUSE 中 RoTPK 密钥字段
efuse_puid_read	读 EFUSE 中产品 UID 字段

库函数名称	库函数描述
efuse_huk_key_read	读 EFUSE 中 HUK 秘钥字段
efuse_user_data_read	读 EFUSE 中用户数据字段
efuse_boot_address_get	获取启动地址
efuse_lock_config	锁定 EFUSE 字段
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 3-151. 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_OVBERIF	越界错误标志位

### 枚举类型 efuse\_clear\_flag\_enum

表 3-152. 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_OVBERIC	清除越界错误标志位

### 枚举类型 efuse\_int\_enum

表 3-153. 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_OVBER	使能越界错误中断

### 枚举类型 efuse\_int\_flag\_enum

表 3-154. 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_OVBERIF	越界错误中断标志

### 枚举类型 `efuse_clear_int_flag_enum`

表 3-155. 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_OBVERIC	清除越界错误中断标志

### 枚举类型 `efuse_reg_lock_enum`

表 3-156. 枚举类型 `efuse_reg_lock_enum`

成员名称	功能描述
EFUSE_BOOT_LOCK	EFUSE boot字段锁定位
EFUSE_ROTPKKEY_LOCK	EFUSE_ROTPKKEY寄存器锁定位
EFUSE_USER_DATA_LOCK	EFUSE_USER_DATA寄存器锁定位
EFUSE_AESKEY_LOCK	EFUSE_AESKEY寄存器锁定位
EFUSE_FPCTL_USERCTL_LOCK	EFUSE_FPCTL及EFUSE_USERCTL寄存器锁定位

### 函数 `efuse_read`

函数`efuse_read`描述见下表：

表 3-157. 函数 `efuse_read`

函数名称	<code>efuse_read</code>
函数原形	<code>ErrStatus efuse_read(uint32_t ef_addr, uint32_t size, uint32_t buf[]);</code>
功能描述	读 EFUSE 值
先决条件	-
被调用函数	-
输入参数{in}	
<b>ef_addr</b>	EFUSE 地址编号（0x00 – 0x60）
输入参数{in}	
<b>size</b>	要读出的 EFUSE 字段大小（0x01 – 0x20 字节）
输出参数{out}	
<b>buf</b>	存储读回 EFUSE 字段数据的数组
返回值	
<b>ErrStatus</b>	ERROR 或 SUCCESS

例如：

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

uint32_t buffer[8] = {0};

ErrStatus flag = ERROR;

flag = efuse_read(0x60, 32, buffer);
```

## 函数 efuse\_write

函数efuse\_write描述见下表:

表 3-158. 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 – 0x60)
输入参数{in}	
size	要写入的 EFUSE 字段大小 (0x01 – 0x20 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(0x60, 8, buffer);
```

## 函数 efuse\_boot\_config

函数efuse\_boot\_config描述见下表:

表 3-159. 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	
<b>ErrStatus</b>	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\_control1\_config

函数efuse\_control1\_config描述见下表：

**表 3-160. 函数 efuse\_control1\_config**

函数名称	efuse_control1_config
函数原形	ErrStatus efuse_control1_config(uint32_t size, uint8_t ctl[]);
功能描述	配置 EFUSE 中 control1 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>ctl</b>	要写入的 control1 控制字段值
输出参数{out}	
-	-
返回值	
<b>ErrStatus</b>	ERROR 或 SUCCESS

例如：

```
/* write control1 value */

uint8_t ctl[1] = 0x02;

ErrStatus flag = ERROR;

flag = efuse_control1_config(1, ctl);
```

### 函数 efuse\_fp\_config

函数efuse\_fp\_config描述见下表：

**表 3-161. 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}	
size	传入的数组大小（字节），必须为 1
输入参数{in}	
fp_value	要写入的存储保护字段值
输出参数{out}	
-	-
返回值	
ErrStatus	ERROR或SUCCESS

例如：

```
/* write Flash protection value */
uint8_t fp_value[1] = 0x01;
ErrStatus flag = ERROR;
flag = efuse_fp_config(1, fp_value);
```

### 函数 efuse\_user\_control\_config

函数efuse\_user\_control\_config描述见下表：

表 3-162. 函数 efuse\_user\_control\_config

函数名称	efuse_user_control_config
函数原形	ErrStatus efuse_user_control_config(uint32_t size, uint8_t user_ctl[]);
功能描述	写 EFUSE 中用户控制字段
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}	
user_ctl	用户控制字段
输出参数{out}	
-	-
返回值	
ErrStatus	ERROR 或 SUCCESS

例如：

```
/* configure user control */
uint8_t user_ctl[1] = 0x02;
ErrStatus flag = ERROR;
flag = efuse_user_control_config(1, user_ctl);
```

## 函数 efuse\_res\_write

函数efuse\_res\_write描述见下表:

表 3-163. 函数 efuse\_res\_write

函数名称	efuse_res_write
函数原形	ErrStatus efuse_res_write(uint32_t size, uint8_t buf[]);
功能描述	写 EFUSE 中 MCU 保留字段
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 12
输入参数{in}	
buf	MCU 保留字段
输出参数{out}	
-	-
返回值	
ErrStatus	ERROR 或 SUCCESS

例如:

```
/* configure user control */

/* write MCU reserved value */

uint32_t buffer[3] = {0x11223344, 0x55667788, 0x9900aabb};

ErrStatus flag = ERROR;

flag = efuse_res_write(12, (uint8_t *)buffer);
```

## 函数 efuse\_aes\_key\_write

函数efuse\_aes\_key\_write描述见下表:

表 3-164. 函数 efuse\_aes\_key\_write

函数名称	efuse_aes_key_write
函数原形	ErrStatus efuse_aes_key_write(uint32_t size, uint8_t buf[]);
功能描述	写 EFUSE 中固件 AES 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 16
输入参数{in}	
buf	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 3-165. 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}	
<b>size</b>	传入的数组大小（字节），必须为 32
输入参数{in}	
<b>buf</b>	要写入的 RoTPK 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[8] = {0x11223344, 0x55667788, 0x9900aabb, 0xccddeeff,
                      0x11223344, 0x55667788, 0x9900aabb, 0xccddeeff};

ErrStatus flag = ERROR;

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

### 函数 efuse\_user\_data\_write

函数efuse\_user\_data\_write描述见下表：

**表 3-166. 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\_res\_read

函数efuse\_res\_read描述见下表：

表 3-167. 函数 efuse\_res\_read

函数名称	efuse_res_read
函数原形	void efuse_res_read(uint32_t buf[]);
功能描述	读 EFUSE 中 MCU 保留字段
先决条件	-
被调用函数	-
输入参数{in}	
-	-
输入参数{in}	
buf	系统复位后 MCU 保留字段
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* read MCU reserved value */
```

```
uint32_t buffer[3];
```

```
efuse_res_read(*buffer);
```

### 函数 efuse\_aes\_key\_read

函数efuse\_aes\_key\_read描述见下表：

**表 3-168. 函数 efuse\_aes\_key\_read**

函数名称	efuse_aes_key_read
函数原形	void efuse_aes_key_read(uint32_t buf[]);
功能描述	读 EFUSE 中 AES 密钥字段
先决条件	-
被调用函数	-
输入参数{in}	
-	-
输入参数{in}	
buf	系统复位后 AES 密钥字段
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* read AES key */
```

```
uint32_t buffer[4];
```

```
efuse_aes_key_read(*buffer);
```

### 函数 efuse\_rotpk\_key\_read

函数efuse\_rotpk\_key\_read描述见下表：

**表 3-169. 函数 efuse\_rotpk\_key\_read**

函数名称	efuse_rotpk_key_read
函数原形	void efuse_rotpk_key_read(uint32_t buf[]);
功能描述	读 EFUSE 中 ROTPK 密钥字段
先决条件	-
被调用函数	-
输入参数{in}	
-	-
输入参数{in}	
buf	系统复位后 ROTPK 密钥字段
输出参数{out}	
-	-

返回值	
-	-

例如：

```
/* read ROTPK key */
uint32_t buffer[8];
efuse_rotpk_key_read(*buffer);
```

### 函数 efuse\_puid\_read

函数efuse\_puid\_read描述见下表：

表 3-170. 函数 efuse\_puid\_read

函数名称	efuse_puid_read
函数原形	void efuse_puid_read(uint32_t buf[]);
功能描述	读 EFUSE 中产品 UID 字段
先决条件	-
被调用函数	-
输入参数{in}	
-	-
输入参数{in}	
buf	系统复位后产品 UID 字段
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* read puid */
uint32_t buffer[4];
efuse_puid_read(*buffer);
```

### 函数 efuse\_huk\_key\_read

函数efuse\_huk\_key\_read描述见下表：

表 3-171. 函数 efuse\_huk\_key\_read

函数名称	efuse_huk_key_read
函数原形	void efuse_huk_key_read(uint32_t buf[]);
功能描述	读 EFUSE 中 HUK 密钥字段
先决条件	-
被调用函数	-

输入参数{in}	
-	-
输入参数{in}	
buf	系统复位后 HUK 密钥字段
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* read huk key */
uint32_t buffer[4];
efuse_huk_key_read(*buffer);
```

### 函数 efuse\_user\_data\_read

函数efuse\_user\_data\_read描述见下表：

表 3-172. 函数 efuse\_user\_data\_read

函数名称	efuse_user_data_read
函数原形	void efuse_user_data_read(uint32_t buf[]);
功能描述	读 EFUSE 中用户数据字段
先决条件	-
被调用函数	-
输入参数{in}	
-	-
输入参数{in}	
buf	系统复位后用户数据字段
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* read user data */
uint32_t buffer[8];
efuse_user_data_read(*buffer);
```

### 函数 efuse\_boot\_address\_get

函数efuse\_boot\_address\_get描述见下表：

表 3-173. 函数 efuse\_boot\_address\_get

函数名称	efuse_boot_address_get
函数原形	uint32_t efuse_boot_address_get(void);
功能描述	获取启动地址
先决条件	-
被调用函数	-
输入参数{in}	
-	-
输出参数{out}	
-	-
返回值	
uint32_t	当前启动地址（0x0 – 0xFFFFFFFF）

例如：

```
/* get boot address */
uint32_t addr = 0;
addr = efuse_boot_address_get();
```

### 函数 efuse\_lock\_config

函数efuse\_lock\_config描述见下表：

表 3-174. 函数 efuse\_lock\_config

函数名称	efuse_lock_config
函数原形	void efuse_lock_config(efuse_reg_lock_enum source);
功能描述	锁定 EFUSE 字段
先决条件	-
被调用函数	-
输入参数{in}	
source	指定锁定的字段，参考 <a href="#">表 3-156. 枚举类型 efuse_reg_lock_enum</a>
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* lock EFUSE boot bits */
efuse_lock_config(EFUSE_BOOT_LOCK);
```

### 函数 efuse\_flag\_get

函数efuse\_flag\_get描述见下表：



表 3-175. 函数 efuse\_flag\_get

函数名称	efuse_flag_get
函数原形	FlagStatus efuse_flag_get(efuse_flag_enum 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}	
flag	EFUSE 状态标志, 参考 <a href="#">表 3-151. 枚举类型 efuse_flag_enum</a>
输出参数{out}	
-	-
返回值	
FlagStatus	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 3-176. 函数 efuse\_flag\_clear

函数名称	efuse_flag_clear
函数原形	void efuse_flag_clear(efuse_clear_flag_enum flag);
功能描述	清除 EFUSE 标志位
先决条件	-
被调用函数	-
输入参数{in}	
flag	EFUSE 状态标志, 参考 <a href="#">表 3-152. 枚举类型 efuse_clear_flag_enum</a>
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 3-177. 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}	
source	要使能的中断源，参考 <a href="#">表 3-153. 枚举类型 efuse_int_enum</a>
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 3-178. 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 3-154. 枚举类型 efuse_int_flag_enum</a>
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 3-179. 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 3-155. 枚举类型 efuse_clear_int_flag_enum</a>
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 3-180. 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_flag);
功能描述	清除 EFUSE 中断标志位
先决条件	-
被调用函数	-
输入参数{in}	
int_flag	中断标志清除位, 参考 <a href="#">表 3-155. 枚举类型 efuse_clear_int_flag_enum</a>
输出参数{out}	
-	-
返回值	
-	-

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 26 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 3-181. EXTI 寄存器**

寄存器名称	寄存器描述
EXTI_INTEN	中断使能寄存器
EXTI_EVEN	事件使能寄存器
EXTI_RTEN	上升沿触发使能寄存器
EXTI_FTEN	下降沿触发使能寄存器
EXTI_SWIEV	软件中断事件寄存器
EXTI_PD	挂起寄存器

### 3.9.2. 外设库函数说明

EXTI库函数列表如下表所示：

**表 3-182. EXTI 库函数**

库函数名称	库函数描述
exti_deinit	复位EXTI
exti_init	初始化EXTI线x
exti_interrupt_enable	EXTI线x中断使能
exti_interrupt_disable	EXTI线x中断禁能
exti_event_enable	EXTI线x事件使能
exti_event_disable	EXTI线x事件禁能
exti_software_interrupt_enable	EXTI线x软件中断事件使能
exti_software_interrupt_disable	EXTI线x软件中断事件禁能
exti_flag_get	获取EXTI线x中断标志位
exti_flag_clear	清除EXTI线x中断标志位
exti_interrupt_flag_get	获取EXTI线x中断标志位
exti_interrupt_flag_clear	清除EXTI线x中断标志位

#### 枚举类型 exti\_line\_enum

**表 3-183. 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

成员名称	功能描述
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_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

#### 枚举类型 `exti_mode_enum`

表 3-184. 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 3-185. 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	无EXIT边沿触发

#### 函数 `exti_deinit`

函数`exti_deinit`描述见下表:

表 3-186. 函数 `exti_deinit`

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

功能描述	复位EXTI
先决条件	-
被调用函数	-
输入参数{in}	
-	-
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* deinitialize the EXTI */
exti_deinit();
```

### 函数 exti\_init

函数exti\_init描述见下表：

表 3-187. 函数 exti\_init

函数名称	exti_init
函数原形	void exti_init(exti_line_enum linex, exti_mode_enum mode, exti_trig_type_enum trig_type);
功能描述	初始化EXTI线x
先决条件	-
被调用函数	-
输入参数{in}	
linex	EXTI线x，参考 <a href="#">表 3-183. 枚举类型exti_line_enum</a>
输入参数{in}	
mode	EXTI模式，参考 <a href="#">表 3-184. 枚举类型exti_mode_enum</a>
输入参数{in}	
trig_type	触发类型，参考 <a href="#">表 3-185. 枚举类型exti_trig_type_enum</a>
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* configure EXTI_0 */
exti_init(EXTI_0, EXTI_INTERRUPT, EXTI_TRIG_BOTH);
```

### 函数 exti\_interrupt\_enable

函数exti\_interrupt\_enable描述见下表：

表 3-188. 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 3-183. 枚举类型exti_line_enum</a>
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* enable the interrupts from EXTI line 0 */
exti_interrupt_enable(EXTI_0);
```

### 函数 exti\_interrupt\_disable

函数exti\_interrupt\_disable描述见下表:

表 3-189. 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 3-183. 枚举类型exti_line_enum</a>
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* disable the interrupts from EXTI line 0 */
exti_interrupt_disable(EXTI_0);
```

### 函数 exti\_event\_enable

函数exti\_event\_enable描述见下表:

表 3-190. 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 3-183. 枚举类型exti_line_enum</a>
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* enable the events from EXTI line 0 */
exti_event_enable(EXTI_0);
```

### 函数 exti\_event\_disable

函数exti\_event\_disable描述见下表:

表 3-191. 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 3-183. 枚举类型exti_line_enum</a>
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* disable the events from EXTI line 0 */
exti_event_disable(EXTI_0);
```

### 函数 exti\_software\_interrupt\_enable

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

表 3-192. 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 3-183. 枚举类型exti_line_enum</a>
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* enable EXTI line 0 software interrupt */
exti_software_interrupt_enable(EXTI_0);
```

### 函数 exti\_software\_interrupt\_disable

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

表 3-193. 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 3-183. 枚举类型exti_line_enum</a>
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* disable EXTI line 0 software interrupt */
exti_software_interrupt_disable(EXTI_0);
```

### 函数 exti\_flag\_get

函数exti\_flag\_get描述见下表:

表 3-194. 函数 exti\_flag\_get

函数名称	exti_flag_get
函数原形	FlagStatus exti_flag_get(exti_line_enum linex);
功能描述	获取EXTI线x中断标志位
先决条件	-
被调用函数	-

输入参数{in}	
linex	EXTI线x, 参考 <a href="#">表 3-183. 枚举类型exti_line_enum</a>
输出参数{out}	
-	-
返回值	
FlagStatus	SET或RESET

例如:

```
/* get EXTI line 0 flag status */
FlagStatus state = exti_flag_get(EXTI_0);
```

### 函数 exti\_flag\_clear

函数exti\_flag\_clear描述见下表:

表 3-195. 函数 exti\_flag\_clear

函数名称	exti_flag_clear
函数原形	void exti_flag_clear(exti_line_enum linex);
功能描述	清除EXTI线x中断标志位
先决条件	-
被调用函数	-
输入参数{in}	
linex	EXTI线x, 参考 <a href="#">表 3-183. 枚举类型exti_line_enum</a>
输出参数{out}	
-	-
返回值	
-	-

例如:

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

### 函数 exti\_interrupt\_flag\_get

函数exti\_interrupt\_flag\_get描述见下表:

表 3-196. 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 3-183. 枚举类型exti_line_enum</a>

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

例如：

```
/* get EXTI line 0 interrupt flag status */
FlagStatus state = exti_interrupt_flag_get(EXTI_0);
```

### 函数 exti\_interrupt\_flag\_clear

函数exti\_interrupt\_flag\_clear描述见下表：

表 3-197. 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 3-183. 枚举类型exti_line_enum</a>
输出参数{out}	
-	-
返回值	
-	-

例如：

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

## 3.10. 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[3.10.2](#)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 3-198. 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

寄存器	描述
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_OBR	选项字节寄存器
FMC_OBUSER	选项字节用户寄存器
FMC_OBWRP0	选项字节写保护/擦除保护寄存器0
FMC_OBWRP1	选项字节写保护/擦除保护寄存器1
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_PID0	产品ID0寄存器
FMC_PID1	产品ID1寄存器
FMC_RFT0	RF微调寄存器0
FMC_RFT1	RF微调寄存器1
FMC_WFTx (x=0..15)	WIFI调整寄存器x (x=0..15)

### 3.10.2. 外设库函数说明

FMC固件库函数列举如下表：

**表 3-199. 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	在相应地址全字编程
fmc_continuous_program	在相应地址连续字编程
fmc_obr_function_enable	使能FMC选项字节功能
fmc_obr_function_disable	禁能FMC选项字节功能
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
ob_security_protection_config	配置选项字节安全保护
ob_user_write	用户选项字节编程
ob_write_protection_config	配置擦写保护区域

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
fmc_wifi_trim_cal_get	获取校准值
fmc_wifi_trim_pa_get	获取功率放大器偏置调整值
fmc_wifi_trim_get	获取WIFI调整值
fmc_pid_get	获取产品ID
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
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 3-200. 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	擦除/编程保护错误
FMC_TOERR	超时错误
FMC_ERR	参数错误

### 函数 fmc\_unlock

函数fmc\_unlock描述见下表：

表 3-201. 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 3-202. 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 3-203. 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 3-200. 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 3-204. 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 3-200. 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 3-205. 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	
<b>fmc_state_enum</b>	FMC状态，详细请参考 <a href="#">表 3-200. 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 3-206. 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}	
<b>address</b>	编程地址，必须4字节对齐
输入参数{in}	
<b>data[]</b>	编程数据
输入参数{in}	
<b>size</b>	编程数据个数
输出参数{out}	
-	-
返回值	
<b>fmc_state_enum</b>	FMC状态，详细请参考 <a href="#">表 3-200. 枚举类型fmc_state_enum</a>

例如：

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

uint32_t data[8]= {0x01234567, 0x01234567, 0x01234567, 0x01234567, 0x01234567,
0x01234567, 0x01234567, 0x01234567};

fmc_state_enum state;

fmc_unlock();
```



```
state = fmc_word_program(0x08004000, data[], 8);
```

### 函数 fmc\_obr\_function\_enable

函数fmc\_obr\_function\_enable描述见下表：

表 3-207. 函数 fmc\_obr\_function\_enable

函数名称	fmc_obr_function_enable
函数原型	void fmc_obr_function_enable(uint32_t obr_function);
功能描述	使能FMC选项字节功能
先决条件	fmc_unlock
被调用函数	-
输入参数{in}	
obr_function	FMC选项字节功能
FMC_OBR_NWDG_HW	看门狗状态功能
FMC_OBR_NRST_STDBY	当进入Standby模式时不产生复位功能
FMC_OBR_NSRT_DPSLP	当进入Deepsleep模式时不产生复位功能
FMC_OBR_SRAM1_RST	系统复位后自动清除SRAM1功能
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* enable software watchdog function */
ob_unlock();
fmc_obr_function_enable(FMC_OBR_NWDG_HW);
```

### 函数 fmc\_obr\_function\_disable

函数fmc\_obr\_function\_disable描述见下表：

表 3-208. 函数 fmc\_obr\_function\_disable

函数名称	fmc_obr_function_disable
函数原型	void fmc_obr_function_disable(uint32_t obr_function);
功能描述	禁能FMC选项字节功能
先决条件	fmc_unlock
被调用函数	-
输入参数{in}	

obr_function	FMC选项字节功能
FMC_OBR_NWDG_HW	看门狗状态功能
FMC_OBR_NRST_STDBY	当进入Standby模式时不产生复位功能
FMC_OBR_NSRT_DPSLP	当进入Deepsleep模式时不产生复位功能
FMC_OBR_SRAM1_RST	系统复位后自动清除SRAM1功能
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* enable hardware watchdog function */
```

```
ob_unlock();
```

```
fmc_obr_function_disable(FMC_OBR_NWDG_HW);
```

### 函数 ob\_unlock

函数ob\_unlock描述见下表：

表 3-209. 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 3-210. 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 3-211. 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 3-212. 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 3-213. 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	配置选项字节安全保护
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_P1	保护等级1
输出参数{out}	
-	-
返回值	
fmc_state_enum	FMC状态，详细请参考 <a href="#">表 3-200. 枚举类型fmc_state_enum</a>

例如：

```
/* configure no security protection */

ob_unlock();

ob_security_protection_config (FMC_NSPC);

ob_start();
```

### 函数 ob\_user\_write

函数ob\_user\_write描述见下表：

表 3-214. 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 3-200. 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 3-215. 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	ob_unlock
被调用函数	-
输入参数{in}	

<b>wrp_spage</b>	擦写保护区域起始页，0~1023
输入参数{in}	
<b>wrp_epage</b>	擦写保护区域末尾页，0~1023
输入参数{in}	
<b>wrp_register_index</b>	FMC_OBWRP <sub>x</sub> register index
<b>OBWRP_INDEX0</b>	option byte write protection area register 0
<b>OBWRP_INDEX1</b>	option byte write protection area register 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 3-200. 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 fmc\_no\_rtdec\_config

函数fmc\_no\_rtdec\_config描述见下表：

表 3-216. 函数 fmc\_no\_rtdec\_config

<b>函数名称</b>	fmc_no_rtdec_config
<b>函数原型</b>	void fmc_no_rtdec_config(uint32_t nodec_spage, uint32_t nodec_epage, uint32_t nodec_register_index);
<b>功能描述</b>	配置不即时解密区域
<b>先决条件</b>	ob_unlock
<b>被调用函数</b>	-
输入参数{in}	
<b>nodec_spage</b>	NO-RTDEC区域起始页，0~0x03FF
输入参数{in}	
<b>nodec_epage</b>	NO-RTDEC区域末尾页，0~0x03FF
输入参数{in}	
<b>nodec_register_index</b>	NO-RTDEC区域配置寄存器
<b>NODEC_INDEX0</b>	NO-RTDEC区域配置寄存器0
<b>NODEC_INDEX1</b>	NO-RTDEC区域配置寄存器1
<b>NODEC_INDEX2</b>	NO-RTDEC区域配置寄存器2
<b>NODEC_INDEX3</b>	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 3-217. 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 3-218. 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 fmc\_wifi\_trim\_cal\_get

函数fmc\_wifi\_trim\_cal\_get描述见下表：

表 3-219. 函数 fmc\_wifi\_trim\_cal\_get

函数名称	fmc_wifi_trim_cal_get
函数原型	void fmc_wifi_trim_cal_get(uint32_t *rft0_bletxcal, uint32_t *rft0_wftxcal, uint32_t *rft0_thecal, uint32_t *rft1_wfrxcal);
功能描述	获取校准值
先决条件	-
被调用函数	-
输入参数{in}	
-	-
输出参数{out}	
rft0_bletxcal	BLE 发射功率校准值
rft0_wftxcal	WIFI 发射功率校准值
rft0_thecal	温度值校准值
rft1_wfrxcal	WIFI 接收增益校准值
返回值	
-	-

例如：

```
/* get calibration value */
```

```
uint32_t bletxcal, wftxcal, thecal, wfrxcal;
```

```
fmc_wifi_trim_cal_get(&bletxcal, &wftxcal, &thecal, &wfrxcal);
```

### 函数 fmc\_wifi\_trim\_pa\_get

函数fmc\_wifi\_trim\_pa\_get描述见下表：



表 3-220. 函数 fmc\_wifi\_trim\_pa\_get

函数名称	fmc_wifi_trim_pa_get
函数原型	void fmc_wifi_trim_pa_get(uint32_t *rft0_pa_tune0, uint32_t *rft0_pa_tune1);
功能描述	获取功率放大器偏置调整值
先决条件	-
被调用函数	-
输入参数{in}	
-	-
输出参数{out}	
rft0_pa_tune0	功率放大器微调值
rft0_pa_tune1	功率放大器粗调值
返回值	
-	-

例如：

```
/* get Power Amplifier bias tune value */
uint32_t pa_tune0, pa_tune1;
fmc_wifi_trim_pa_get(&pa_tune0, &pa_tune1);
```

### 函数 fmc\_wifi\_trim\_get

函数fmc\_wifi\_trim\_get描述见下表：

表 3-221. 函数 fmc\_wifi\_trim\_get

函数名称	fmc_wifi_trim_get
函数原型	fmc_state_enum fmc_wifi_trim_get(uint32_t size, uint32_t buf[]);
功能描述	获取WIFI调整寄存器
先决条件	-
被调用函数	-
输入参数{in}	
size	读取 WIFI 调整寄存器的数据个数，必须是 16U
输出参数{out}	
buf[ ]	读取 WIFI 调整寄存器的数据
返回值	
fmc_state_enum	FMC状态，详细请参考 <a href="#">表 3-200. 枚举类型fmc_state_enum</a>

例如：

```
/* get the FMC WIFI trim register */
fmc_state_enum fmc_state = FMC_ERR;
uint32_t size = 16U;
uint32_t data[16];
```

```
fmc_state = fmc_wifi_trim_get(size, &data);
```

## 函数 fmc\_pid\_get

函数fmc\_pid\_get描述见下表

**表 3-222. 函数 fmc\_pid\_get**

函数名称	fmc_pid_get
函数原型	void fmc_pid_get(uint32_t *pid);
功能描述	get product ID
先决条件	-
被调用函数	-
输入参数{in}	
-	-
输出参数{out}	
pid	产品ID
返回值	
-	-

例如:

```
/* get product ID */
```

```
uint32_t pid[2];
```

```
fmc_pid_get(&pid);
```

## 函数 ob\_write\_protection\_get

函数ob\_write\_protection\_get描述见下表:

**表 3-223. 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 3-224. 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 3-225. 函数 ob\_security\_protection\_flag\_get**

函数名称	ob_security_protection_flag_get
函数原型	FlagStatus ob_security_protection_flag_get(void);
功能描述	获取安全保护状态选项字节
先决条件	-
被调用函数	-
输入参数{in}	
-	-
输出参数{out}	
-	-
返回值	
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();
```

### 函数 fmc\_flag\_get

函数fmc\_flag\_get描述见下表:

表 3-226. 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_WPER R	FMC写保护编程/擦除错误标志
FMC_FLAG_END	FMC操作完成标志
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 3-227. 函数 fmc\_flag\_clear

函数名称	fmc_flag_clear
函数原型	void fmc_flag_clear(uint32_t flag);
功能描述	清除FMC标志
先决条件	-
被调用函数	-
输入参数{in}	
flag	清除FMC标志

<i>FMC_FLAG_WPER</i> <i>R</i>	FMC写保护编程/擦除错误标志
<i>FMC_FLAG_END</i>	FMC操作完成标志
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 3-228. 函数 **fmc\_interrupt\_enable**

函数名称	fmc_interrupt_enable
函数原型	void fmc_interrupt_enable(uint32_t interrupt);
功能描述	使能FMC中断
先决条件	fmc_unlock
被调用函数	-
输入参数{in}	
<b>interrupt</b>	FMC中断
<i>FMC_INT_END</i>	FMC编程完成中断
<i>FMC_INT_ERR</i>	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 3-229. 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 3-230. 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_WPERR	FMC擦写保护错误中断标志
FMC_INT_FLAG_END	FMC操作完成中断标志
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 3-231. 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}	
<b>flag</b>	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
输出参数{out}	
-	-
返回值	
<b>FlagStatus</b>	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 3-232. 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 3-233. 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重载值
fwdgt_counter_reload	重载FWDGT计数器
fwdgt_config	设置FWDGT重载值、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 3-234. 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 3-235. 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 3-236. 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 3-237. 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 3-238. 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}	
<b>reload_value</b>	指定FWDGT重装载值(0x0000-0x0FFF)
输出参数{out}	
-	-
返回值	
<b>ErrStatus</b>	ERROR或SUCCESS

例如:

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

```
fwdgt_reload_value_config(625);
```

### 函数 fwdgt\_counter\_reload

函数fwdgt\_counter\_reload描述见下表:

表 3-239. 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 3-240. 函数 fwdgt\_config

函数名称	fwdgt_config
函数原型	ErrStatus fwdgt_config(uint16_t reload_value, uint8_t prescaler_value);
功能描述	设置FWDGT重装载值、预分频值
先决条件	-
被调用函数	-
输入参数{in}	
reload_value	重装载值(0x0000 - 0x0FFF)
输入参数{in}	
prescaler_value	FWDGT预分频值
FWDGT_PSC_DIV4	FWDGT预分频值设为4
FWDGT_PSC_DIV8	FWDGT预分频值设为8
FWDGT_PSC_DIV16	FWDGT预分频值设为16
6	
FWDGT_PSC_DIV32	FWDGT预分频值设为32
2	
FWDGT_PSC_DIV64	FWDGT预分频值设为64
4	
FWDGT_PSC_DIV128	FWDGT预分频值设为128
28	
FWDGT_PSC_DIV256	FWDGT预分频值设为256
56	

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

例如：

```
/* configure FWDGT counter clock: 40KHz(IRC40K) / 64 = 0.625 KHz */
```

```
fwdgt_config(2*500, FWDGT_PSC_DIV64);
```

### 函数 fwdgt\_flag\_get

函数fwdgt\_flag\_get描述见下表：

表 3-241. 函数 fwdgt\_flag\_get

函数名称	fwdgt_flag_get
函数原型	FlagStatus fwdgt_flag_get(uint16_t flag);
功能描述	获取FWDGT标志位状态
先决条件	-
被调用函数	-
输入参数{in}	
flag	需要获取状态的FWDGT标志位
FWDGT_FLAG_PUD	预分频值更新进行中
FWDGT_FLAG_RUD	重装载值更新进行中
输出参数{out}	
-	-
返回值	
FlagStatus	SET或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 3-242. GPIO 寄存器

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

### 3.12.2. 外设库函数说明

GPIO库函数列表如下表所示：

表 3-243. GPIO 库函数

库函数名称	库函数描述
gpio_deinit	复位外设GPIOx
gpio_mode_set	设置GPIO模式
gpio_output_options_set	设置GPIO输出模式和速度
gpio_bit_set	置位引脚值
gpio_bit_reset	复位引脚值
gpio_bit_write	将特定的值写入引脚
gpio_port_write	将特定的值写入一组端口
gpio_input_bit_get	获取引脚的输入值
gpio_input_port_get	获取一组端口的输入值
gpio_output_bit_get	获取引脚的输出值
gpio_output_port_get	获取一组端口的输出值
gpio_af_set	设置GPIO复用功能
gpio_pin_lock	相应的引脚配置被锁定
gpio_bit_toggle	翻转GPIO引脚状态
gpio_port_toggle	翻转一组GPIO状态

#### 函数 gpio\_deinit

函数gpio\_deinit描述见下表：

表 3-244. 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 3-245. 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}	
<b>pin</b>	GPIO pin
<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 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 3-246. 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}	
<b>gpio_periph</b>	GPIO端口
<i>GPIOx</i>	端口选择 (x = A,B,C)
输入参数{in}	
<b>otype</b>	GPIO引脚输出模式
<i>GPIO_OTYPE_PP</i>	推挽输出模式
<i>GPIO_OTYPE_OD</i>	开漏输出模式
输入参数{in}	
<b>speed</b>	GPIO引脚输出最大速度
<i>GPIO_OSPEED_2MHZ</i>	最大输出速度为2MHz
<i>GPIO_OSPEED_10MHZ</i>	最大输出速度为10MHz
<i>GPIO_OSPEED_25MHZ</i>	最大输出速度为25MHz
<i>GPIO_OSPEED_MAX</i>	最大输出速度
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 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 3-247. 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 3-248. 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 3-249. 函数 gpio\_bit\_write

函数名称	gpio_bit_write
函数原型	void gpio_bit_write(uint32_t gpio_periph,uint32_t pin,bit_status bit_value);
功能描述	将特定的值写入引脚
先决条件	-
被调用函数	-
输入参数{in}	
gpio_periph	GPIO端口
GPIOx	端口选择 (x = A,B,C)
输入参数{in}	
pin	GPIO引脚
GPIO_PIN_x	引脚选择 (x = 0..15)
GPIO_PIN_ALL	所有引脚
输入参数{in}	
bit_value	设置或清除
RESET	清除引脚值
SET	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 3-250. 函数 gpio\_port\_write

函数名称	gpio_port_write
函数原型	void gpio_port_write(uint32_t gpio_periph,uint16_t data);
功能描述	将特定的值写入端口
先决条件	-
被调用函数	-
输入参数{in}	
gpio_periph	GPIO端口
GPIOx	端口选择 (x = A,B,C)
输入参数{in}	
data	将要写入的具体值
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* write 1010 0101 1010 0101 to Port A */
```

```
gpio_port_write(GPIOA, 0xA5A5);
```

### 函数 gpio\_input\_bit\_get

函数gpio\_input\_bit\_get描述见下表:

表 3-251. 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}	
<b>pin</b>	GPIO引脚
<i>GPIO_PIN_x</i>	引脚选择 (x = 0..15)
<i>GPIO_PIN_ALL</i>	所有引脚
输出参数{out}	
-	-
返回值	
<b>FlagStatus</b>	SET或RESET

例如:

```
/* get output status of PA0 */

FlagStatus bit_state;

bit_state = gpio_output_bit_get(GPIOA, GPIO_PIN_0);
```

### 函数 gpio\_input\_port\_get

函数gpio\_input\_port\_get描述见下表:

表 3-252. 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 input value of Port A */

uint16_t port_state;

port_state = gpio_input_port_get(GPIOA);
```

### 函数 gpio\_output\_bit\_get

函数gpio\_output\_bit\_get描述见下表:

表 3-253. 函数 gpio\_output\_bit\_get

函数名称	gpio_output_bit_get
函数原型	FlagStatus gpio_output_bit_get(uint32_t gpio_periph,uint32_t pin);
功能描述	获取引脚的输出值
先决条件	-
被调用函数	-
输入参数{in}	
gpio_periph	GPIO端口
GPIOx	端口选择 (x = A,B,C)
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 output status of PA0 */
FlagStatus bit_state;

bit_state = gpio_output_bit_get(GPIOA, GPIO_PIN_0);
```

### 函数 gpio\_output\_port\_get

函数gpio\_output\_port\_get描述见下表:

表 3-254. 函数 gpio\_output\_port\_get

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

例如:

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

```
uint16_t port_state;
```

```
port_state = gpio_output_port_get(GPIOA);
```

## 函数 gpio\_af\_set

函数gpio\_af\_set描述见下表:

表 3-255. 函数 gpio\_af\_set

函数名称	gpio_af_set
函数原型	void gpio_af_set(uint32_t gpio_periph, uint32_t alt_func_num, uint32_t pin);
功能描述	设置GPIO的备用功能
先决条件	-
被调用函数	-
输入参数{in}	
gpio_periph	GPIO端口
GPIOx	GPIOx (x = A,B,C)
输入参数{in}	
alt_func_num	GPIO引脚备用功能, 请参见特定设备的数据手册
GPIO_AF_0	USART0, UART1, I2C1, CK_OUT0, SPI, JTAG, CK_OUT1, RTC_REF, IFRP_OUT
GPIO_AF_1	TIMER0, TIMER1
GPIO_AF_2	USART0, TIMER0, TIMER2, SPI
GPIO_AF_3	QSPI, USART1, TIMER0, TIMER1, TIMER2
GPIO_AF_4	UART1, SPI, I2C0, QSPI, I2C1
GPIO_AF_5	SPI, I2C0
GPIO_AF_6	TIMER0, I2C0, SPI, I2C1
GPIO_AF_7	USART0, UART1, TIMER16, SPI
GPIO_AF_8	USART0, UART1, UART2, TIMER0, TIMER2, TIMER15
GPIO_AF_9	RTC, TIMER0, TIMER1, TIMER16, IFRP_OUT
GPIO_AF_10	UART2, TIMER2, TIMER16
GPIO_AF_11	TIMER0, TIMER15
GPIO_AF_12	-
GPIO_AF_13	-
GPIO_AF_14	-
GPIO_AF_15	EVENTOUT
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 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 3-256. 函数 gpio\_pin\_lock

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

例如:

```
/* lock PA0 */
```

```
gpio_pin_lock(GPIOA, GPIO_PIN_0);
```

### 函数 gpio\_bit\_toggle

函数gpio\_bit\_toggle描述见下表:

表 3-257. 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}	
<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:

```
/* 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 3-258. 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}	
<b>gpio_periph</b>	GPIO端口
<i>GPIOx</i>	GPIOx (x = A,B,C)
输出参数{out}	
-	-
返回值	
-	-

例如:

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

```
gpio_port_toggle(GPIOA);
```

## 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 (HAMC)。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 3-259. 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 3-260. 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
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



库函数名称	库函数描述
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 3-261. 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 3-262. 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 3-263. 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 3-264. 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 3-265. 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3-261. 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 3-266. 函数 `hau_init_struct_para_init`**

函数名称	<code>hau_init_struct_para_init</code>
函数原形	<code>void hau_init_struct_para_init(hau_init_parameter_struct* initpara)</code>
功能描述	初始化结构体 <code>hau_initpara</code>
先决条件	-
被调用函数	-
输入参数{in}	
-	-
输出参数{out}	
<b>initpara</b>	HAU外设参数，参考结构体 <a href="#">表3-261. 结构体 <code>hau_init_parameter_struct</code></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 3-267. 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 3-268. 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 3-269. 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 3-270. 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 3-271. 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3-262. 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 3-272. 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 3-273. 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	多DMA或单DMA
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 3-274. 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 3-275. 函数 hau\_dma\_disable**

函数名称	hau_dma_disable
函数原形	void hau_dma_disable(void);
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 3-276. 函数 hau\_context\_struct\_para\_init**

函数名称	hau_context_struct_para_init
函数原形	void hau_context_struct_para_init(hau_context_parameter_struct* 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}	
context	上下文结构体, 参考结构体 <a href="#">表3-263. 结构体 hau_context_parameter_struct</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 3-277. 函数 hau\_context\_save**

函数名称	hau_context_save
函数原形	void hau_context_save(hau_context_parameter_struct* context_save)
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}	
context	上下文结构体，参考结构体 <a href="#">表3-263. 结构体 hau_context_parameter_struct</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 3-278. 函数 hau\_context\_restore**

函数名称	hau_context_restore
函数原形	void hau_context_restore(hau_context_parameter_struct* context_restore)
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}	
context	上下文结构体，参考结构体 <a href="#">表3-263. 结构体 hau_context_parameter_struct</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 3-279. 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 */

static uint8_t output[20];

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 3-280. 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 3-281. 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 */
```

```
static uint8_t output[20];
```

```
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 3-282. 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 */
```

```
static uint8_t output[20];
```

```
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 3-283. 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 */

static uint8_t output[20];

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 3-284. 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 */

static uint8_t output[20];

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 3-285. 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 */

static uint8_t output[16];

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 3-286. 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}	
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 HMAC mode */
```

```
static uint8_t output[16];
```

```
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 3-287. 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}	
flag	HAU标志状态
HAU_FLAG_DATA_INP UT	输入FIFO有足够空间
HAU_FLAG_CALCULA TION_COMPLETE	摘要计算完成
HAU_FLAG_DMA	DMA被使能或传输正在处理
HAU_FLAG_BUSY	数据块正在处理
HAU_FLAG_INFIFO_N O_EMPTY	输入FIFO非空
输出参数{out}	

-	-
返回值	
FlagStatus	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 3-288. 函数 hau\_flag\_clear

函数名称	hau_flag_clear
函数原形	void hau_flag_clear(uint32_t flag);
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}	
flag	HAU标志状态
HAU_FLAG_DATA_INP UT	输入FIFO有足够空间
HAU_FLAG_CALCULA TION_COMPLETE	摘要计算完成
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 3-289. 函数 hau\_interrupt\_enable

函数名称	hau_interrupt_enable
函数原形	void hau_interrupt_enable(uint32_t interrupt);
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
<i>HAU_INT_DATA_INPUT</i>	新数据块进入IN缓存区
<i>HAU_INT_CALCULATION_COMPLETE</i>	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 3-290. 函数 hau\_interrupt\_disable**

函数名称	hau_interrupt_disable
函数原形	void hau_interrupt_disable(uint32_t interrupt);
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
<i>HAU_INT_DATA_INPUT</i>	新数据块进入IN缓存区
<i>HAU_INT_CALCULATION_COMPLETE</i>	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 3-291. 函数 hau\_interrupt\_flag\_get

函数名称	hau_interrupt_flag_get
函数原形	FlagStatus hau_interrupt_flag_get(uint32_t int_flag)
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}	
int_flag	HAU标志状态
HAU_INT_FLAG_DATA_INPUT	输入FIFO有足够空间
HAU_INT_FLAG_CALCULATION_COMPLETE	摘要计算完整
输出参数{out}	
-	-
返回值	
FlagStatus	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 3-292. 函数 hau\_interrupt\_flag\_clear

函数名称	hau_interrupt_flag_clear
函数原形	void hau_interrupt_flag_clear(uint32_t int_flag)
功能描述	清除HAU中断标志状态
先决条件	-
被调用函数	-
输入参数{in}	
int_flag	HAU标志状态
HAU_INT_FLAG_DATA_INPUT	输入FIFO有足够空间
HAU_INT_FLAG_CALCULATION_COMPLETE	摘要计算完整
输出参数{out}	

-	-
返回值	
-	-

例如：

```
/* clear the HAU interrupt flag status */
```

```
hau_interrupt_flag_clear(HAU_INT_FLAG_DATA_INPUT);
```

## 3.14. I2C

I2C（内部集成电路总线）模块提供了符合工业标准的两线串行制接口，可用于MCU和外部I2C设备的通讯。章节[3.14.1](#)描述了I2C的寄存器列表，章节[3.14.2](#)对I2C库函数进行说明。

### 3.14.1. 外设寄存器说明

I2C寄存器列表如下表所示：

**表 3-293. I2C 寄存器**

寄存器名称	寄存器描述
I2C_CTL0	控制寄存器 0
I2C_CTL1	控制寄存器 1
I2C_SADDR0	从机地址寄存器 0
I2C_SADDR1	从机地址寄存器 1
I2C_TIMING	时序寄存器
I2C_TIMEOUT	超时寄存器
I2C_STAT	状态寄存器
I2C_STATC	状态清除寄存器
I2C_PEC	PEC 寄存器
I2C_RDATA	接收数据寄存器
I2C_TDATA	发送数据寄存器
I2C_CTL2	控制寄存器 2

### 3.14.2. 外设库函数说明

I2C库函数列表如下表所示：

**表 3-294. 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

库函数名称	库函数描述
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
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_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

库函数名称	库函数描述
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 3-295. 枚举类型 i2c\_interrupt\_flag\_enum

成员名称	功能描述
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_ADDSEND	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_STPDET	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_LOSTARB	仲裁丢失中断标志
I2C_INT_FLAG_OUERR	从机模式下，过载/欠载错误中断标志
I2C_INT_FLAG_PECERR	PEC错误中断标志
I2C_INT_FLAG_TIMEOUT	超时中断标志
I2C_INT_FLAG_SMBALT	SMBus报警中断标志

### 函数 i2c\_deinit

函数i2c\_deinit描述见下表：

表 3-296. 函数 i2c\_deinit

函数名称	i2c_deinit
函数原型	void i2c_deinit(uint32_t i2c_periph);
功能描述	复位外设 I2C
先决条件	-
被调用函数	rcu_periph_reset_enable / rcu_periph_reset_disable
输入参数{in}	
i2c_periph	I2C 外设
I2Cx	(x=0,1)
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 3-297. 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 3-298. 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>I2CCLK</sub> 的尖峰
FILTER_LENGTH_2	数字噪声滤波使能并且可以滤除脉宽宽度不大于 2 t <sub>I2CCLK</sub> 的尖峰
FILTER_LENGTH_3	数字噪声滤波使能并且可以滤除脉宽宽度不大于 3 t <sub>I2CCLK</sub> 的尖峰
FILTER_LENGTH_4	数字噪声滤波使能并且可以滤除脉宽宽度不大于 4 t <sub>I2CCLK</sub> 的尖峰
FILTER_LENGTH_5	数字噪声滤波使能并且可以滤除脉宽宽度不大于 5 t <sub>I2CCLK</sub> 的尖峰
FILTER_LENGTH_6	数字噪声滤波使能并且可以滤除脉宽宽度不大于 6 t <sub>I2CCLK</sub> 的尖峰
FILTER_LENGTH_7	数字噪声滤波使能并且可以滤除脉宽宽度不大于 7 t <sub>I2CCLK</sub> 的尖峰
FILTER_LENGTH_8	数字噪声滤波使能并且可以滤除脉宽宽度不大于 8 t <sub>I2CCLK</sub> 的尖峰
FILTER_LENGTH_9	数字噪声滤波使能并且可以滤除脉宽宽度不大于 9 t <sub>I2CCLK</sub> 的尖峰
FILTER_LENGTH_10	数字噪声滤波使能并且可以滤除脉宽宽度不大于 10 t <sub>I2CCLK</sub> 的尖峰
FILTER_LENGTH_11	数字噪声滤波使能并且可以滤除脉宽宽度不大于 11 t <sub>I2CCLK</sub> 的尖峰
FILTER_LENGTH_12	数字噪声滤波使能并且可以滤除脉宽宽度不大于 12 t <sub>I2CCLK</sub> 的尖峰
FILTER_LENGTH_13	数字噪声滤波使能并且可以滤除脉宽宽度不大于 13 t <sub>I2CCLK</sub> 的尖峰
FILTER_LENGTH_14	数字噪声滤波使能并且可以滤除脉宽宽度不大于 14 t <sub>I2CCLK</sub> 的尖峰
FILTER_LENGTH_15	数字噪声滤波使能并且可以滤除脉宽宽度不大于 15 t <sub>I2CCLK</sub> 的尖峰
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* 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 3-299. 函数 i2c\_analog\_noise\_filter\_enable

函数名称	i2c_analog_noise_filter_enable
函数原型	void i2c_analog_noise_filter_enable(uint32_t i2c_periph);
功能描述	使能模拟噪声滤波器



先决条件	-
被调用函数	-
输入参数{in}	
i2c_periph	I2C外设
I2Cx	(x=0,1)
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 3-300. 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 3-301. 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 3-302. 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_TRANS MIT	主机发送
I2C_MASTER_RECEIV E	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

#### 3-303. 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 3-304. 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 3-305. 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 3-306. 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}	

<b>i2c_periph</b>	I2C 外设
<i>I2Cx</i>	(x=0,1)
<b>输出参数{out}</b>	
-	-
<b>返回值</b>	
-	-

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 3-307. 函数 i2c\_automatic\_end\_enable**

<b>函数名称</b>	i2c_automatic_end_enable
<b>函数原型</b>	void i2c_automatic_end_enable(uint32_t i2c_periph);
<b>功能描述</b>	使能主机模式下 I2C 自动结束模式
<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
<i>I2Cx</i>	(x=0,1)
<b>输出参数{out}</b>	
-	-
<b>返回值</b>	
-	-

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 3-308. 函数 i2c\_automatic\_end\_disable**

<b>函数名称</b>	i2c_automatic_end_disable
<b>函数原型</b>	void i2c_automatic_end_disable(uint32_t i2c_periph);
<b>功能描述</b>	禁能主机模式下 I2C 自动结束模式
<b>先决条件</b>	-
<b>被调用函数</b>	-

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 3-309. 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 3-310. 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 3-311. 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 3-312. 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}	
<b>i2c_periph</b>	I2C 外设
<i>I2Cx</i>	(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 3-313. 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}	
<b>i2c_periph</b>	I2C 外设
<i>I2Cx</i>	(x=0,1)
输入参数{in}	
<b>address</b>	I2C 地址
输入参数{in}	
<b>addr_format</b>	7 位地址或 10 位地址
<i>I2C_ADDFORMAT_7BITS</i>	7 位地址
<i>I2C_ADDFORMAT_10BITS</i>	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 3-314. 函数 i2c\_address\_bit\_compare\_config**

函数名称	i2c_address_bit_compare_config
函数原型	void i2c_address_bit_compare_config(uint32_t i2c_periph, uint32_t compare_bits);
功能描述	定义 ADDRESS[7:1]的哪些位和接收到的地址进行比较
先决条件	-
被调用函数	-
输入参数{in}	
i2c_periph	I2C peripheral
I2Cx	(x=0,1)
输入参数{in}	
compare_bits	需要进行比较的位
ADDRESS_BIT1_COMPARE	地址的第 1 位需要进行比较
ADDRESS_BIT2_COMPARE	地址的第 2 位需要进行比较
ADDRESS_BIT3_COMPARE	地址的第 3 位需要进行比较
ADDRESS_BIT4_COMPARE	地址的第 4 位需要进行比较
ADDRESS_BIT5_COMPARE	地址的第 5 位需要进行比较
ADDRESS_BIT6_COMPARE	地址的第 6 位需要进行比较
ADDRESS_BIT7_COMPARE	地址的第 7 位需要进行比较
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* define which bits of ADDRESS[7:1] need to compare with the incoming address byte */
```

```
i2c_address_bit_compare_config(I2C0, ADDRESS_BIT1_COMPARE);
```

### 函数 i2c\_address\_disable

函数i2c\_address\_disable描述见下表:

表 3-315. 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 3-316. 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

<i>T1_3</i>	
<i>ADDRESS2_MASK_BIT1_4</i>	ADDRESS2[4:1]屏蔽，ADDRESS2[7:5]进行比较
<i>ADDRESS2_MASK_BIT1_5</i>	ADDRESS2[5:1]屏蔽，ADDRESS2[7:6]进行比较
<i>ADDRESS2_MASK_BIT1_6</i>	ADDRESS2[6:1]屏蔽，ADDRESS2[7]进行比较
<i>ADDRESS2_MASK_ALL</i>	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 3-317. 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}	
<i>i2c_periph</i>	I2C 外设
<i>I2Cx</i>	(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\_received\_address\_get

函数i2c\_received\_address\_get描述见下表：

表 3-318. 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_receved_address_get(I2C0);
```

### 函数 i2c\_slave\_byte\_control\_enable

函数i2c\_slave\_byte\_control\_enable描述见下表:

表 3-319. 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 3-320. 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 3-321. 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\_wakeup\_from\_deepsleep\_enable

函数i2c\_wakeup\_from\_deepsleep\_enable描述见下表：

表 3-322. 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 3-323. 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 3-324. 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 3-325. 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 3-326. 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 3-327. 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 3-328. 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 3-329. 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 3-330. 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 3-331. 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	禁用 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 3-332. 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, uint8_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 3-333. 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}	
dma	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 3-334. 函数 i2c\_dma\_disable

函数名称	i2c_dma_disable
函数原型	void i2c_dma_disable(uint32_t i2c_periph, uint8_t dma);
功能描述	禁能发送/接收模式下 DMA
先决条件	-
被调用函数	-
输入参数{in}	
<b>i2c_periph</b>	I2C 外设
<i>I2Cx</i>	(x=0,1)
输入参数{in}	
<b>dma</b>	I2C DMA
<i>I2C_DMA_TRANSMIT</i>	采用 DMA 方式发送数据
<i>I2C_DMA_RECEIVE</i>	采用 DMA 方式接收数据
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* disable I2C DMA for transmission or reception */
```

```
i2c_dma_disable(I2C0, I2C_DMA_RECEIVE);
```

### 函数 i2c\_pec\_transfer

函数i2c\_pec\_transfer描述见下表：

表 3-335. 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 3-336. 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 3-337. 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 3-338. 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 3-339. 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

3-340. 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 3-341. 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 3-342. 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 3-343. 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 3-344. 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 3-345. 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 3-346. 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 3-347. 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 3-348. 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 3-349. 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 3-350. 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 3-351. 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 3-352. 函数 i2c\_flag\_get**

函数名称	i2c_flag_get
函数原型	FlagStatus i2c_flag_get(uint32_t i2c_periph, uint32_t flag);
功能描述	获取 I2C 标志位
先决条件	-

被调用函数	-
输入参数{in}	
i2c_periph	I2C 外设
I2Cx	(x=0,1)
输入参数{in}	
flag	I2C 标志位
I2C_FLAG_TBE	发送期间 I2C_TDATA 寄存器空标志
I2C_FLAG_TI	发送中断标志
I2C_FLAG_RBNE	接收期间 I2C_RDATA 非空标志
I2C_FLAG_ADDSEND	从机模式下，接收到的地址与自身地址匹配
I2C_FLAG_NACK	NACK 标志
I2C_FLAG_STPDET	从机模式下检测到 STOP 信号
I2C_FLAG_TC	主机模式下传输完成标志
I2C_FLAG_TCR	传输完成重载标志
I2C_FLAG_BERR	总线错误标志
I2C_FLAG_LOSTARB	仲裁丢失标志
I2C_FLAG_OUERR	从机模式下，过载/欠载错误标志
I2C_FLAG_PECERR	PEC 错误标志
I2C_FLAG_TIMEOUT	超时标志
I2C_FLAG_SMBALT	SMBus 报警标志
I2C_FLAG_I2CBSY	忙标志
I2C_FLAG_TR	从机模式下，I2C 作为发送器还是接收器标志
输出参数{out}	
-	-
返回值	
FlagStatus	SET / RESET

例如：

```
/* get I2C flag status */
```

```
FlagStatus flag_state = RESET;
```

```
flag_state = i2c_flag_get(I2C0, I2C_FLAG_TBE);
```

### 函数 i2c\_flag\_clear

函数i2c\_flag\_clear描述见下表：

表 3-353. 函数 i2c\_flag\_clear

函数名称	i2c_flag_clear
函数原型	void i2c_flag_clear(uint32_t i2c_periph, uint32_t flag);
功能描述	清除 I2C 标志位
先决条件	-
被调用函数	-

输入参数{in}	
<b>i2c_periph</b>	I2C 外设
<i>I2Cx</i>	(x=0,1)
输入参数{in}	
<b>flag</b>	I2C 标志位
<i>I2C_FLAG_ADDSEND</i>	从机模式下, 接收到的地址与自身地址匹配
<i>I2C_FLAG_NACK</i>	NACK 标志
<i>I2C_FLAG_STPDET</i>	从机模式下检测到 STOP 信号
<i>I2C_FLAG_BERR</i>	总线错误标志
<i>I2C_FLAG_LOSTARB</i>	仲裁丢失标志
<i>I2C_FLAG_OUERR</i>	从机模式下, 过载/欠载错误标志
<i>I2C_FLAG_PECERR</i>	PEC 错误标志
<i>I2C_FLAG_TIMEOUT</i>	超时标志
<i>I2C_FLAG_SMBALT</i>	SMBus 报警标志
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* clear a bus error flag*/
```

```
i2c_flag_clear(I2C0, I2C_FLAG_BERR);
```

## 函数 i2c\_interrupt\_enable

函数i2c\_interrupt\_enable描述见下表:

表 3-354. 函数 i2c\_interrupt\_enable

函数名称	i2c_interrupt_enable
函数原型	void i2c_interrupt_enable(uint32_t i2c_periph, uint32_t interrupt);
功能描述	中断使能
先决条件	-
被调用函数	-
输入参数{in}	
<b>i2c_periph</b>	I2C 外设
<i>I2Cx</i>	(x=0,1)
输入参数{in}	
<b>interrupt</b>	I2C 中断
<i>I2C_INT_ERR</i>	错误中断
<i>I2C_INT_TC</i>	发送完成中断
<i>I2C_INT_STPDET</i>	检测到 STOP 中断
<i>I2C_INT_NACK</i>	接收到 NACK 中断
<i>I2C_INT_ADDM</i>	地址匹配中断

<i>I2C_INT_RBNE</i>	接收中断
<i>I2C_INT_TI</i>	发送中断
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* enable I2C0 transmit interrupt */
```

```
i2c_interrupt_enable(I2C0, I2C_INT_TI);
```

### 函数 i2c\_interrupt\_disable

函数i2c\_interrupt\_disable描述见下表：

表 3-355. 函数 i2c\_interrupt\_disable

函数名称	i2c_interrupt_disable
函数原型	void i2c_interrupt_disable(uint32_t i2c_periph, uint32_t interrupt);
功能描述	中断除能
先决条件	-
被调用函数	-
输入参数{in}	
<b>i2c_periph</b>	I2C 外设
<i>I2Cx</i>	(x=0,1)
输入参数{in}	
<b>interrupt</b>	I2C 中断
<i>I2C_INT_ERR</i>	错误中断
<i>I2C_INT_TC</i>	发送完成中断
<i>I2C_INT_STPDET</i>	检测到 STOP 中断
<i>I2C_INT_NACK</i>	接收到 NACK 中断
<i>I2C_INT_ADDM</i>	地址匹配中断
<i>I2C_INT_RBNE</i>	接收中断
<i>I2C_INT_TI</i>	发送中断
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* disable I2C0 transmit interrupt */
```

```
i2c_interrupt_disable(I2C0, I2C_INT_TI);
```



## 函数 i2c\_interrupt\_flag\_get

函数i2c\_interrupt\_flag\_get描述见下表:

表 3-356. 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 3-295. 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 3-357. 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 3-295. 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_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	
-	-

例如：

```
/* clear a bus error flag */
```

```
i2c_interrupt_flag_clear(I2C0, I2C_INT_FLAG_BERR);
```

## 3.15. 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[3.15.1](#)描述了PKCAU的寄存器列表，章节[3.15.2](#)对PKCAU库函数进行说明。

### 3.15.1. 外设寄存器说明

PKCAU寄存器列表如下表所示：

**表 3-358. 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

### 3.15.2. 外设库函数说明

PKCAU库函数列表如下表所示：

**表 3-359. 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_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_ecc_out_struct_para_init	初始化ECC输出参数结构体为默认值
pkcau_enable	使能PKCAU
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

库函数名称	库函数描述
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_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 3-360. 结构体 pkcau\_mont\_parameter\_struct

成员名称	功能描述
modulus_n	模数n
modulus_n_len	模长（字节长度）

### 结构体 pkcau\_mod\_parameter\_struct

表 3-361. 结构体 pkcau\_mod\_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长度（字节长度）
opr_d_b	操作数B
opr_d_b_len	操作数B长度（字节长度）
modulus_n	模数n
modulus_n_len	模长（字节长度）

### 结构体 pkcau\_mod\_exp\_parameter\_struct

表 3-362. 结构体 pkcau\_mod\_exp\_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长度（字节长度）
exp_e	指数e
e_len	指数e长度（字节长度）
modulus_n	模数n
modulus_n_len	模长（字节长度）
mont_para	蒙哥马利参数R2 mod n
mont_para_len	蒙哥马利参数长度（字节长度）

## 结构体 `pkcau_arithmetic_parameter_struct`

表 3-363. 结构体 `pkcau_arithmetic_parameter_struct`

成员名称	功能描述
<code>oprnd_a</code>	操作数A
<code>oprnd_a_len</code>	操作数A的长度（字节长度）
<code>oprnd_b</code>	操作数B
<code>oprnd_b_len</code>	操作数B长度（字节长度）

## 结构体 `pkcau_crt_parameter_struct`

表 3-364. 结构体 `pkcau_crt_parameter_struct`

成员名称	功能描述
<code>oprnd_a</code>	操作数A
<code>oprnd_a_len</code>	操作数A的长度（字节长度）
<code>oprnd_dp</code>	操作数dp
<code>oprnd_dp_len</code>	操作数dp的长度（字节长度）
<code>oprnd_dq</code>	操作数dq
<code>oprnd_dq_len</code>	操作数dq的长度（字节长度）
<code>oprnd_qinv</code>	操作数qinv
<code>oprnd_qinv_len</code>	操作数qinv的长度（字节长度）
<code>oprnd_p</code>	质数操作数p
<code>oprnd_p_len</code>	质数操作数p的长度（字节长度）
<code>oprnd_q</code>	质数操作数q
<code>oprnd_q_len</code>	质数操作数q的长度（字节长度）

## 结构体 `pkcau_ec_group_parameter_struct`

表 3-365. 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>modulus_p_len</code>	曲线模长p的长度（字节长度）
<code>coff_a</code>	曲线系数a
<code>coff_a_len</code>	曲线系数a的长度（字节长度）
<code>coff_b</code>	曲线系数b
<code>coff_b_len</code>	曲线系数b的长度（字节长度）
<code>base_point_x</code>	曲线基点坐标x
<code>base_point_x_len</code>	曲线基点坐标x的长度（字节长度）
<code>base_point_y</code>	曲线基点坐标y
<code>base_point_y_len</code>	曲线基点坐标y的长度（字节长度）
<code>order_n</code>	曲线质数阶n
<code>order_n_len</code>	曲线质数阶n的长度（字节长度）
<code>a_sign</code>	曲线系数a的符号

成员名称	功能描述
multi_k	标量乘数k
multi_k_len	标量乘数k的长度
integer_k	整数k
integer_k_len	整数k的长度（字节长度）
private_key_d	私钥d
private_key_d_len	私钥d的长度（字节长度）
mont_para	蒙哥马利参数R2 mod n
mont_para_len	蒙哥马利参数的长度（字节长度）

### 结构体 pkcau\_point\_parameter\_struct

表 3-366. 结构体 pkcau\_point\_parameter\_struct

成员名称	功能描述
point_x	点的坐标x
point_x_len	点的坐标x的长度（字节长度）
point_y	点的坐标y
point_y_len	点的坐标y的长度（字节长度）

### 结构体 pkcau\_signature\_parameter\_struct

表 3-367. 结构体 pkcau\_signature\_parameter\_struct

成员名称	功能描述
sign_r	签名r部分
sign_r_len	签名r部分的长度（字节长度）
sign_s	签名s部分
sign_s_len	签名s部分的长度（字节长度）

### 结构体 pkcau\_hash\_parameter\_struct

表 3-368. 结构体 pkcau\_hash\_parameter\_struct

成员名称	功能描述
hash_z	哈希值z
hash_z_len	哈希值z的长度（字节长度）

### 结构体 pkcau\_ecc\_out\_struct

表 3-369. 结构体 pkcau\_ecc\_out\_struct

成员名称	功能描述
sign_extra	扩展ECDSA签名标志
sign_r	签名r部分的长度）
sign_s	签名s部分
point_x	点kP的坐标x
point_y	点kP的坐标y

## 函数 pkcau\_deinit

函数pkcau\_deinit描述见下表：

**表 3-370. 函数 pkcau\_deinit**

函数名称	pkcau_deinit
函数原型	void pkcau_deinit(void);
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 3-371. 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}	
-	-
输出参数{out}	
init_para	蒙哥马利参数初始化结构体，结构体成员参考 <a href="#">表3-360. 结构体 pkcau_mont_parameter_struct</a> 。
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 3-372. 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}	
-	-
输出参数{out}	
init_para	模参数初始化结构体，结构体成员参考 <a href="#">表3-361. 结构体 pkcau_mod_parameter_struct</a> 。
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 3-373. 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}	
-	-
输出参数{out}	
init_para	模幂参数初始化结构体，结构体成员参考 <a href="#">表3-362. 结构体 pkcau_mod_exp_parameter_struct</a> 。
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\_arithmetic\_struct\_para\_init

函数pkcau\_arithmetic\_struct\_para\_init描述见下表：

**表 3-374. 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}	
-	-
输出参数{out}	
init_para	算术参数初始化结构体，结构体成员参考 <a href="#">表3-363. 结构体 pkcau_arithmetic_parameter_struct</a> 。
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 3-375. 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}	
-	-
输出参数{out}	

init_para	CRT参数初始化结构体，结构体成员参考 <a href="#">表3-364. 结构体 pkcau crt parameter struct</a> 。
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 3-376. 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}	
-	-
输出参数{out}	
init_para	ECC椭圆参数初始化结构体，结构体成员参考 <a href="#">表3-365. 结构体 pkcau ec group parameter struct</a> 。
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 3-377. 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}	
-	-
输出参数{out}	
init_para	点参数初始化结构体，结构体成员参考 <a href="#">表3-366. 结构体 pkcau_point_parameter_struct</a>
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 3-378. 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}	
-	-
输出参数{out}	
init_para	签名参数初始化结构体，结构体成员参考 <a href="#">表3-367. 结构体 pkcau_signature_parameter_struct</a> 。
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 3-379. 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}	
-	-
输出参数{out}	
init_para	哈希参数初始化结构体，结构体成员参考 <a href="#">表3-368. 结构体 pkcau_hash_parameter_struct</a> 。
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\_ecc\_out\_struct\_para\_init

函数pkcau\_ecc\_out\_struct\_para\_init描述见下表：

表 3-380. 函数 pkcau\_ecc\_out\_struct\_para\_init

函数名称	pkcau_ecc_out_struct_para_init
函数原型	void pkcau_ecc_out_struct_para_init(pkcau_ecc_out_struct* init_para)
功能描述	初始化ECC输出参数结构体为默认值
先决条件	-
被调用函数	-
输入参数{in}	
-	-
输出参数{out}	
init_para	ecdsa 签名，ECC 标量乘法输出结果结构体结构体成员参考 <a href="#">表 3-369. 结构体 pkcau_ecc_out_struct</a> 。
返回值	
-	-

例如：

```
/* initialize PKCAU modular reduction parameter struct with a default value */
pkcau_ecc_out_struct_para_init pkcau_ecc_out_parameter;
```

pkcau\_ecc\_out\_struct\_para\_init (&pkcau\_ecc\_out\_parameter);

## 函数 pkcau\_enable

函数pkcau\_enable描述见下表:

表 3-381. 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 3-382. 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 3-383. 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 3-384. 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_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
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_ SCALAR_MUL	先计算蒙哥马利参数，然后进行ECC标量乘法运算
PKCAU_MODE_ECC_ SCALAR_MUL_FAST	仅进行ECC标量乘法运算
PKCAU_MODE_ECDS A_SIGN	ECDSA签名
PKCAU_MODE_ECDS A_VERIFICATION	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 3-385. 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, uint8_t *results);
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3-360. 结构体 pkcau_mont_parameter_struct</a> 。
输出参数{out}	
results	输出结果
返回值	
-	-

例如：

```
uint8_t results[ME_MOD_SIZE];

/* 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_n_len = ME_MOD_SIZE;

pkcau_mont_parameter.modulus_n = (uint8_t *)modulus_n;

/* execute montgomery parameter operation */

pkcau_mont_param_operation(&pkcau_mont_parameter, results);
```

## 函数 pkcau\_mod\_operation

函数pkcau\_mod\_operation描述见下表：

表 3-386. 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, uint8_t *results)
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3-361. 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_	模减运算



<i>SUB</i>	
<i>PKCAU_MODE_MONT</i> <i>_MUL</i>	蒙哥马利乘法运算
输出参数{out}	
<b>results</b>	输出结果
返回值	
-	-

例如：

```
uint8_t results[MA_MOD_SIZE];

/* 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_n_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, results);
```

### 函数 pkcau\_mod\_exp\_operation

函数pkcau\_mod\_exp\_operation描述见下表：

**表 3-387. 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, uint8_t *results);
功能描述	执行模幂运算
先决条件	-
被调用函数	-
输入参数{in}	
<b>mod_exp_para</b>	模幂参数初始化结构体，结构体成员参考 <a href="#">表3-362. 结构体 pkcau_mod_exp_parameter_struct</a> 。
输入参数{in}	
<b>mode</b>	模幂运算模式
<i>PKCAU_MODE_MOD_EXP</i>	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}	
results	输出结果
返回值	
-	-

例如：

```
uint8_t mod_exp_res[ME_MOD_SIZE];

/* 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.oprd_a_len = sizeof(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_n_len = ME_MOD_SIZE;
pkcau_mod_exp_parameter.modulus_n = (uint8_t *)modulus_n;
pkcau_mod_exp_parameter.mont_para = (uint8_t *)mont_para;
pkcau_mod_exp_parameter.mont_para_len = sizeof(mont_para);

/* execute modular exponentiation fast operation */
pkcau_mod_exp_operation(&pkcau_mod_exp_parameter,
PKCAU_MODE_MOD_EXP_FAST, mod_exp_res);
```

## 函数 pkcau\_mod\_inver\_operation

函数pkcau\_mod\_inver\_operation描述见下表：

表 3-388. 函数 pkcau\_mod\_inver\_operation

函数名称	pkcau_mod_inver_operation
函数原型	void pkcau_mod_inver_operation(pkcau_mod_parameter_struct *mod_inver_para, uint8_t *results);
功能描述	执行模逆运算
先决条件	-
被调用函数	-
输入参数{in}	

<b>mod_inver_para</b>	模逆参数初始化结构体，结构体成员参考 <a href="#">表3-361. 结构体 pkcau_mod_parameter_struct</a> 。
<b>输出参数{out}</b>	
<b>results</b>	输出结果
<b>返回值</b>	
-	-

例如：

```
uint8_t mod_inver_res[ME_MOD_SIZE];

/* initialize the modular inversion parameter structure */

pkcau_mod_parameter_struct pkcau_mod_inver_parameter;

pkcau_mod_inver_struct_para_init(&pkcau_mod_inver_parameter);

/* initialize the modular parameters */

pkcau_mod_inver_parameter.oprd_a = (uint8_t *)oprd_a;

pkcau_mod_inver_parameter.oprd_a = sizeof(oprd_a);

pkcau_mod_inver_parameter.modulus_n_len = ME_MOD_SIZE;

pkcau_mod_inver_parameter.modulus_n = (uint8_t *)modulus_n;

/* execute modular inversion operation */

pkcau_mod_inver_operation(&pkcau_mod_inver_parameter, mod_inver_res);
```

## 函数 pkcau\_mod\_reduc\_operation

函数pkcau\_mod\_reduc\_operation描述见下表：

**表 3-389. 函数 pkcau\_mod\_reduc\_operation**

<b>函数名称</b>	pkcau_mod_reduc_operation
<b>函数原型</b>	void pkcau_mod_reduc_operation(pkcau_mod_parameter_struct *mod_reduc_para, uint8_t *results);
<b>功能描述</b>	执行取模运算
<b>先决条件</b>	-
<b>被调用函数</b>	-
<b>输入参数{in}</b>	
<b>mod_reduc_para</b>	取模参数初始化结构体，结构体成员参考 <a href="#">表3-361. 结构体 pkcau_mod_parameter_struct</a> 。
<b>输出参数{out}</b>	
<b>results</b>	输出结果
<b>返回值</b>	
-	-

例如：

```
uint8_t mod_reduc_res[MA_MOD_SIZE];

/* initialize the modular inversion parameter structure */

pkcau_mod_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_n_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, mod_reduc_res);
```

## 函数 pkcau\_arithmetic\_operation

函数pkcau\_arithmetic\_operation描述见下表：

**表 3-390. 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, uint8_t* results);
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3-363. 结构体 pkcau_arithmetic_parameter_struct</a> 。
输入参数{in}	
mode	算术运算模式
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
输出参数{out}	

results	输出结果
返回值	
-	-

例如：

```
uint8_t ari_multi_res[AM_SIZE];

/* 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_a_len = AM_A_SIZE;

pkcau_ari_multi_parameter.oprd_b = (uint8_t *)oprd_b;

pkcau_ari_multi_parameter.oprd_b_len = AM_B_SIZE;

/* execute arithmetic addition operation */

pkcau_arithmetic_operation(&pkcau_arithmetic_parameter,
PKCAU_MODE_ARITHMETIC_ADD, ari_multi_res);
```

### 函数 pkcau\_crt\_exp\_operation

函数pkcau\_crt\_exp\_operation描述见下表：

**表 3-391. 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, uint8_t* results);
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3-364. 结构体 pkcau_crt_parameter_struct</a> 。
输出参数{out}	
results	输出结果
返回值	
-	-

例如：

```
uint8_t crt_res[sizeof(rsa_crt_a)];
```

```

/* 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_a_len = sizeof(rsa_crt_a);
pkcau_crt_parameter.oprd_dp = (uint8_t *)rsa_crt_dp;
pkcau_crt_parameter.oprd_dp_len = sizeof(rsa_crt_dp);
pkcau_crt_parameter.oprd_dq = (uint8_t *)rsa_crt_dq;
pkcau_crt_parameter.oprd_dq_len = sizeof(rsa_crt_dq);
pkcau_crt_parameter.oprd_qinv = (uint8_t *)rsa_crt_qinv;
pkcau_crt_parameter.oprd_qinv_len = sizeof(rsa_crt_qinv);
pkcau_crt_parameter.oprd_p = (uint8_t *)rsa_crt_p;
pkcau_crt_parameter.oprd_p_len = sizeof(rsa_crt_p);
pkcau_crt_parameter.oprd_q = (uint8_t *)rsa_crt_q;
pkcau_crt_parameter.oprd_q_len = sizeof(rsa_crt_q);

/* execute RSA CRT exponentiation operation */

pkcau_crt_exp_operation(&pkcau_crt_parameter, crt_res);

```

## 函数 pkcau\_point\_check\_operation

函数pkcau\_point\_check\_operation描述见下表：

**表 3-392. 函数 pkcau\_point\_check\_operation**

函数名称	pkcau_point_check_operation
函数原型	uint8_t 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3-366. 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3-365. 结构体</a>

	<a href="#"><u>pkcau_ec_group_parameter_struct</u></a>
输出参数{out}	
-	-
返回值	
uint8_t	表示该点是否在椭圆曲线上

例如:

```
uint8_t res = 1;

/* 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_x_len = sizeof(pcheck_x);
pkcau_point_check_parameter.point_y = pcheck_y;
pkcau_point_check_parameter.point_y_len = sizeof(pcheck_y);

/* initialize the input ECC curve parameter */
pkcau_curve_group.modulus_p = (uint8_t *)brainpoolp256r1_p;
pkcau_curve_group.modulus_p_len = sizeof(brainpoolp256r1_p);
pkcau_curve_group.coff_a = (uint8_t *)brainpoolp256r1_a;
pkcau_curve_group.coff_a_len = sizeof(brainpoolp256r1_a);
pkcau_curve_group.coff_b = (uint8_t *)brainpoolp256r1_b;
pkcau_curve_group.coff_b_len = sizeof(brainpoolp256r1_b);
pkcau_curve_group.a_sign = 0;

/* execute point check operation */
res = 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 3-393. 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, pkcau_ecc_out_struct* result);
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3-366. 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3-365. 结构体 pkcau_ec_group_parameter_struct</a> 。
输入参数{in}	
mode	点乘运算模式
PKCAU_MODE_ECC_SCALAR_MUL	先计算蒙哥马利参数，然后进行ECC标量乘法运算
PKCAU_MODE_ECC_SCALAR_MUL_FAST	仅进行ECC标量乘法运算
输出参数{out}	
result	ecdsa签名，ECC标量乘法输出结果结构体，结构体成员参考 <a href="#">表3-369. 结构体 pkcau_ecc_out_struct</a> 。
返回值	
-	-

例如：

```
pkcau_ecc_out_struct pkcau_ecc_out_result;

/* initialize the ECC out parameter */

pkcau_ecc_out_struct_para_init(&pkcau_ecc_out_result);

pkcau_ecc_out_result.point_x = res_x;

pkcau_ecc_out_result.point_y = res_y;

/* 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_x_len = sizeof(ec_pmul_x);

pkcau_point_parameter.point_y = ec_pmul_y;

pkcau_point_parameter.point_x_len = sizeof(ec_pmul_x);

/* initialize the input ECC curve parameter */

pkcau_curve_group.modulus_p      = (uint8_t *)brainpoolp256r1_p;

pkcau_curve_group.modulus_p_len = sizeof(brainpoolp256r1_p);

pkcau_curve_group.coff_a         = (uint8_t *)brainpoolp256r1_a;

pkcau_curve_group.coff_a_len    = sizeof(brainpoolp256r1_a);

pkcau_curve_group.a_sign        = 0;

pkcau_curve_group.multi_k       = ec_pmul_k;

pkcau_curve_group.multi_k_len   = PMUL_K_SIZE;

/* execute scalar multiplication operation */

pkcau_point_mul_operation(&pkcau_point_parameter, &pkcau_curve_group,
PKCAU_MODE_ECC_MUL, pkcau_ecc_out_result);

```

## 函数 pkcau\_ecdsa\_sign\_operation

函数pkcau\_ecdsa\_sign\_operation描述见下表：

**表 3-394. 函数 pkcau\_ecdsa\_sign\_operation**

函数名称	pkcau_ecdsa_sign_operation
函数原型	uint8_t pkcau_ecdsa_sign_operation(pkcau_hash_parameter_struct *hash_para, const pkcau_ec_group_parameter_struct* curve_group_para, pkcau_ecc_out_struct* result);
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}	
hash_para	哈希参数初始化结构体，结构体成员参考 <a href="#">表3-368. 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3-365. 结构体 pkcau_ec_group_parameter_struct</a> 。
输出参数{out}	
results	ecdsa签名，ECC标量乘法输出结果结构体，结构体成员参考 <a href="#">表3-369. 结构体 pkcau_ecc_out_struct</a> 。

返回值	
uint8_t	表示签名操作是否成功

例如：

```
pkcau_ecc_out_struct pkcau_ecc_out_result;

/* initialize the ECC out parameter */

pkcau_ecc_out_struct_para_init(&pkcau_ecc_out_result);

pkcau_ecc_out_result.sign_r = r;

pkcau_ecc_out_result.sign_s = s;

/* 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.modulus_p_len = sizeof(secp112r2_p);

pkcau_curve_group.coff_a         = secp112r2_a;

pkcau_curve_group.coff_a_len     = sizeof(secp112r2_a);

pkcau_curve_group.a_sign         = 0;

pkcau_curve_group.base_point_x   = secp112r2_gx;

pkcau_curve_group.base_point_x_len = sizeof(secp112r2_gx);

pkcau_curve_group.base_point_y   = secp112r2_gy;

pkcau_curve_group.base_point_y_len = sizeof(secp112r2_gy);

pkcau_curve_group.order_n        = secp112r2_n,

pkcau_curve_group.order_n_len    = sizeof(secp112r2_n);

pkcau_curve_group.integer_k      = k;
```

```
pkcau_curve_group.integer_k_len    = DATA_SIZE;

pkcau_curve_group.private_key_d    = d;

pkcau_curve_group.private_key_d_len = DATA_SIZE;

/* execute ECDSA sign operation */

pkcau_ecdsa_sign_operation(&pkcau_hash_parameter, &pkcau_curve_group, results);
```

## 函数 pkcau\_ecdsa\_verification\_operation

函数pkcau\_ecdsa\_verification\_operation描述见下表：

**表 3-395. 函数 pkcau\_ecdsa\_verification\_operation**

函数名称	pkcau_ecdsa_verification_operation
函数原型	uint8_t 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3-366. 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3-368. 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3-367. 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3-365. 结构体 pkcau_ec_group_parameter_struct</a> 。
输出参数{out}	
-	-
返回值	
uint8_t	表示签名验证操作是否成功

例如：

```
uint8_t verify_res = 1;

/* ECC curve parameter structure */

pkcau_ec_group_parameter_struct pkcau_curve_group;
```

```
/* hash parameter structure */

pkcau_hash_parameter_struct pkcau_hash_parameter;

/* signature parameter structure */

pkcau_signature_parameter_struct pkcau_signature_parameter;

/* point parameter structure */

pkcau_point_parameter_struct pkcau_point_parameter;

/* initialize the ECC curve parameter, hash parameter, point parameter and signature
parameter structure */

pkcau_ec_group_struct_para_init(&pkcau_curve_group);

pkcau_hash_struct_para_init(&pkcau_hash_parameter);

pkcau_point_struct_para_init(&pkcau_point_parameter);

pkcau_signature_struct_para_init(&pkcau_signature_parameter);

/* initialize the input ECC signature parameters */

pkcau_signature_parameter.sign_r      = (uint8_t *)ecc_verify_r;
pkcau_signature_parameter.sign_r_len = sizeof(ecc_verify_r);

pkcau_signature_parameter.sign_s      = (uint8_t *)ecc_verify_s;
pkcau_signature_parameter.sign_s_len = sizeof(ecc_verify_s);

/* initialize the input point parameters */

pkcau_point_parameter.point_x      = ecc_verify_x;
pkcau_point_parameter.point_x_len = sizeof(ecc_verify_x);

pkcau_point_parameter.point_y      = ecc_verify_y;
pkcau_point_parameter.point_y_len = sizeof(ecc_verify_y);

/* initialize the input ECC curve parameters */

pkcau_curve_group.modulus_p      = (uint8_t *)brainpoolp256r1_p;
pkcau_curve_group.modulus_p_len = sizeof(brainpoolp256r1_p);

pkcau_curve_group.coff_a      = (uint8_t *)brainpoolp256r1_a;
pkcau_curve_group.coff_a_len = sizeof(brainpoolp256r1_a);

pkcau_curve_group.a_sign      = 0;

pkcau_curve_group.base_point_x      = (uint8_t *)brainpoolp256r1_gx;
pkcau_curve_group.base_point_x_len = sizeof(brainpoolp256r1_gx);
```

```

pkcau_curve_group.base_point_y      = (uint8_t *)brainpoolp256r1_gy;

pkcau_curve_group.base_point_y_len = sizeof(brainpoolp256r1_gy);

pkcau_curve_group.order_n           = (uint8_t *)brainpoolp256r1_n,

pkcau_curve_group.order_n_len      = sizeof(brainpoolp256r1_n);

/* 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);

/* execute ECDSA verification operation */

verify_res = pkcau_ecdsa_verification_operation(&pkcau_point_parameter,
&pkcau_hash_parameter, &pkcau_signature_parameter, &pkcau_curve_group);

```

## 函数 pkcau\_flag\_get

函数pkcau\_flag\_get描述见下表：

**表 3-396. 函数 pkcau\_flag\_get**

函数名称	pkcau_flag_get
函数原型	FlagStatus pkcau_flag_get(uint32_t flag);
功能描述	获取 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
PKCAU_FLAG_BUSY	忙标志
输出参数{out}	
-	-
返回值	
FlagStatus	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 3-397. 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 3-398. 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
PKCAU_INT_ADDRERR	地址错误中断
PKCAU_INT_RAMERR	PKCAU RAM错误中断
PKCAU_INT_END	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 3-399. 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}	
interrupt	中断类型
PKCAU_INT_ADDRERR	地址错误中断
PKCAU_INT_RAMERR	PKCAU RAM错误中断
PKCAU_INT_END	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 3-400. 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}	

<b>int_flag</b>	PKCAU 中断标志
<i>PKCAU_INT_FLAG_ADDRERR</i>	地址错误中断标志
<i>PKCAU_INT_FLAG_RAMERR</i>	PKCAU RAM 错误中断标志
<i>PKCAU_INT_FLAG_END</i>	PKCAU 运算结束中断标志
<b>输出参数{out}</b>	
-	-
<b>返回值</b>	
<b>FlagStatus</b>	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 3-401. 函数 pkcau\_interrupt\_flag\_clear**

<b>函数名称</b>	pkcau_interrupt_flag_clear
<b>函数原型</b>	void pkcau_interrupt_flag_clear(uint32_t int_flag);
<b>功能描述</b>	清除 PKCAU 中断标志位
<b>先决条件</b>	-
<b>被调用函数</b>	-
<b>输入参数{in}</b>	
<b>int_flag</b>	PKCAU 中断标志
<i>PKCAU_INT_FLAG_ADDRERR</i>	地址错误中断标志
<i>PKCAU_INT_FLAG_RAMERR</i>	PKCAU RAM 错误中断标志
<i>PKCAU_INT_FLAG_END</i>	PKCAU 运算结束中断标志
<b>输出参数{out}</b>	
-	-
<b>返回值</b>	
-	-

例如:

```
/* clear address error interrupt flag*/
```



```
pkcau_interrupt_flag_clear(PKCAU_INT_FLAG_ADDRERR);
```

### 3.16. PMU

电源管理单元提供了六种省电模式，包括睡眠模式，深度睡眠模式，待机模式，SRAM\_sleep 模式，Wi-Fi\_sleep 模式和 BLE\_sleep 模式。章节 [3.16.1](#) 描述了 PMU 的寄存器列表，章节 [3.16.2](#) 对 PMU 库函数进行说明。

#### 3.16.1. 外设寄存器说明

PMU 寄存器列表如下表所示：

表 3-402. PMU 寄存器

寄存器名称	寄存器描述
PMU_CTL0	PMU控制寄存器0
PMU_CS0	PMU控制和状态寄存器0
PMU_CTL1	PMU控制寄存器1
PMU_CS1	PMU控制和状态寄存器1
PMU_PAR0	PMU参数寄存器0
PMU_PAR1	PMU参数寄存器1
PMU_PAR2	PMU参数寄存器2
PMU_RFCTL	PMU RF控制寄存器
PMU_RFPAR	PMU RF时序参数寄存器
PMU_INTF	PMU BLE中断标志寄存器
PMU_INTEN	PMU BLE中断使能寄存器
PMU_INTC	PMU BLE中断清除寄存器

#### 3.16.2. 外设库函数说明

PMU 库函数列表如下表所示：

表 3-403. 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_backup_write_enable	备份域写使能
pmu_backup_write_disable	备份域写失能
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_wifi_power_enable	WIFI电源使能
pmu_wifi_power_disable	WIFI电源失能
pmu_wifi_sram_control	WIFI & SRAM低功耗控制
pmu_ble_control	BLE低功耗控制
pmu_ble_wakeup_request_enable	使能BLE唤醒请求
pmu_ble_wakeup_request_disable	失能BLE唤醒请求
pmu_pll_force_enable	强制打开/关闭MCU PLL电源使能
pmu_pll_force_disable	强制打开/关闭MCU PLL电源失能
pmu_ble_rf_config	配置BLE RF序列
pmu_rf_force_enable	强制打开/关闭RF电源使能
pmu_rf_force_disable	强制打开/关闭RF电源失能
pmu_rf_sequence_config	配置RF时序
pmu_flag_get	获取状态标志
pmu_flag_clear	清除标志
pmu_interrupt_enable	使能PMU中断
pmu_interrupt_disable	失能PMU中断
pmu_interrupt_flag_get	获取PMU中断标志
pmu_interrupt_flag_clear	清除PMU中断标志

## 函数 pmu\_deinit

函数pmu\_deinit描述见下表：

**表 3-404. 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 3-405. 函数 pmu\_lvd\_select**

函数名称	pmu_lvd_select
函数原型	void pmu_lvd_select(uint32_t lvd_t_n);
功能描述	选择低压检测阈值
先决条件	-
被调用函数	-
输入参数{in}	
lvd_t_n	电压阈值
PMU_LVDT_0	电压阈值为2.1V
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 2.9V */
```

```
pmu_lvd_select (PMU_LVDT_5);
```

## 函数 pmu\_lvd\_disable

函数pmu\_lvd\_disable描述见下表:

**表 3-406. 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\_backup\_write\_enable

函数pmu\_backup\_write\_enable描述见下表：

表 3-407. 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 3-408. 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\_to\_sleepmode

函数pmu\_to\_sleepmode描述见下表:

**表 3-409. 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 3-410. 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_NORMA	当系统进入深度睡眠模式时, LDO仍正常工作

<i>L</i>	
<i>PMU_LDO_LOWPOWER</i>	当系统进入深度睡眠模式时，LDO进入低功耗模式
输入参数{in}	
<i>lowdrive</i>	低驱动模式
<i>PMU_LOWDRIVER_DISABLE</i>	深度睡眠模式下低驱动模式失能
<i>PMU_LOWDRIVER_ENABLE</i>	深度睡眠模式下低驱动模式使能
输入参数{in}	
<i>deepsleepmodecmd</i>	进入深度睡眠模式命令
<i>WFI_CMD</i>	WFI命令
<i>WFE_CMD</i>	WFE命令
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* PMU work at deepsleep mode */
```

```
pmu_to_deepsleepmode(PMU_LDO_NORMAL, PMU_LOWDRIVER_DISABLE, WFI_CMD);
```

### 函数 `pmu_to_standbymode`

函数 `pmu_to_standbymode` 描述见下表：

**表 3-411. 函数 `pmu_to_standbymode`**

函数名称	<code>pmu_to_standbymode</code>
函数原型	<code>void pmu_to_standbymode(uint8_t standbymodecmd);</code>
功能描述	进入待机模式
先决条件	-
被调用函数	-
输入参数{in}	
<i>standbymodecmd</i>	进入待机模式命令
<i>WFI_CMD</i>	WFI命令
<i>WFE_CMD</i>	WFE命令
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* PMU work at standby mode */
```

```
pmu_to_standby(WFI_CMD);
```

## 函数 pmu\_wakeup\_pin\_enable

函数pmu\_wakeup\_pin\_enable描述见下表：

**表 3-412. 函数 pmu\_wakeup\_pin\_enable**

函数名称	pmu_wakeup_pin_enable
函数原型	void pmu_wakeup_pin_enable(void);
功能描述	WKUP引脚唤醒使能
先决条件	-
被调用函数	-
输入参数{in}	
wakeup_pin	WKUP引脚
PMU_WAKEUP_PIN0	WKUP引脚0（PA0）
PMU_WAKEUP_PIN1	WKUP引脚1（PA15）
PMU_WAKEUP_PIN2	WKUP引脚2（PA7）
PMU_WAKEUP_PIN3	WKUP引脚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 3-413. 函数 pmu\_wakeup\_pin\_disable**

函数名称	pmu_wakeup_pin_disable
函数原型	void pmu_wakeup_pin_disable(void);
功能描述	WKUP引脚唤醒失能
先决条件	-
被调用函数	-
输入参数{in}	

wakeup_pin	WKUP引脚
PMU_WAKEUP_PIN0	WKUP引脚0 (PA0)
PMU_WAKEUP_PIN1	WKUP引脚1 (PA15)
PMU_WAKEUP_PIN2	WKUP引脚2 (PA7)
PMU_WAKEUP_PIN3	WKUP引脚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\_wifi\_power\_enable

函数pmu\_wifi\_power\_enable描述见下表:

表 3-414. 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 3-415. 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	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：

```
/* 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 3-416. 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 低功耗控制
先决条件	-
被调用函数	-
输入参数{in}	
wakeup_sram	低功耗控制
PMU_WIFI_SLEEP	WIFI进入睡眠模式
PMU_WIFI_WAKE	WIFI唤醒
PMU_SRAM1_SLEEP	SRAM1进入睡眠模式
PMU_SRAM1_WAKE	SRAM1唤醒
PMU_SRAM2_SLEEP	SRAM2进入睡眠模式
PMU_SRAM2_WAKE	SRAM2唤醒
PMU_SRAM3_SLEEP	SRAM3进入睡眠模式
PMU_SRAM3_WAKE	SRAM3唤醒

<i>PMU_SRAM0_SLE EP</i>	SRAM0进入睡眠模式
<i>PMU_SRAM0_WAK E</i>	SRAM0唤醒
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* WIFI & SRAM low power control */
pmu_wifi_sram_control(PMU_WIFI_SLEEP);
```

### 函数 **pmu\_ble\_control**

函数pmu\_ble\_control描述见下表：

表 3-417. 函数 pmu\_ble\_control

函数名称	pmu_ble_control
函数原型	void pmu_ble_control(uint32_t wifi_sram);
功能描述	BLE低功耗控制
先决条件	-
被调用函数	-
输入参数{in}	
<b>ble</b>	BLE低功耗控制
<i>PMU_BLE_SLEEP</i>	BLE进入睡眠
<i>PMU_BLE_WAKE</i>	BLE唤醒
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* BLE low power control */
pmu_ble_control(PMU_BLE_SLEEP);
```

### 函数 **pmu\_ble\_wakeup\_request\_enable**

函数pmu\_ble\_wakeup\_request\_enable描述见下表：

表 3-418. 函数 pmu\_ble\_wakeup\_request\_enable

函数名称	pmu_ble_wakeup_request_enable
函数原型	void pmu_ble_wakeup_request_enable(void);

功能描述	使能BLE唤醒请求
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 BLE wakeup request */
pmu_ble_wakeup_request_enable();
```

### 函数 pmu\_ble\_wakeup\_request\_disable

函数pmu\_ble\_wakeup\_request\_disable描述见下表：

表 3-419. 函数 pmu\_ble\_wakeup\_request\_disable

函数名称	pmu_ble_wakeup_request_disable
函数原型	void pmu_ble_wakeup_request_disable(void);
功能描述	失能BLE唤醒请求
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 BLE wakeup request */
pmu_ble_wakeup_request_disable();
```

### 函数 pmu\_pll\_force\_enable

函数pmu\_pll\_force\_enable描述见下表：

表 3-420. 函数 pmu\_pll\_force\_enable

函数名称	pmu_pll_force_enable
函数原型	void pmu_pll_force_enable(uint32_t force);
功能描述	强制打开/关闭MCU PLL电源使能

先决条件	-
被调用函数	-
输入参数{in}	
<b>force</b>	强制打开/关闭PLL电源
PMU_PLL_FORCE_OPEN	软件强制打开PLL电源
PMU_PLL_FORCE_CLOSE	软件强制关闭PLL电源
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* enable MCU PLL power force open/close */
pmu_pll_force_enable(PMU_PLL_FORCE_OPEN);
```

### 函数 pmu\_pll\_force\_disable

函数pmu\_pll\_force\_disable描述见下表：

表 3-421. 函数 pmu\_pll\_force\_disable

函数名称	pmu_pll_force_disable
函数原型	void pmu_pll_force_disable(uint32_t force);
功能描述	强制打开/关闭MCU PLL电源失能
先决条件	-
被调用函数	-
输入参数{in}	
<b>force</b>	强制打开/关闭PLL电源
PMU_PLL_FORCE_OPEN	软件强制打开PLL电源
PMU_PLL_FORCE_CLOSE	软件强制关闭PLL电源
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* disable MCU PLL power force open/close */
pmu_pll_force_disable(PMU_PLL_FORCE_CLOSE);
```

## 函数 pmu\_ble\_rf\_config

函数pmu\_ble\_rf\_config描述见下表:

表 3-422. 函数 pmu\_ble\_rf\_config

函数名称	pmu_ble_rf_config
函数原型	void pmu_ble_rf_config(uint32_t mode);
功能描述	配置BLE RF序列
先决条件	-
被调用函数	-
输入参数{in}	
mode	使能模式
PMU_BLE_RF_SOFTWARE	通过软件使能RF
PMU_BLE_RF_HARDWARE	通过BLE硬件使能RF
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* enable RF by software */
```

```
pmu_ble_rf_config(PMU_BLE_RF_SOFTWARE);
```

## 函数 pmu\_rf\_force\_enable

函数pmu\_rf\_force\_enable描述见下表:

表 3-423. 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	强制打开/关闭RF电源使能
先决条件	-
被调用函数	-
输入参数{in}	
force	强制打开/关闭RF电源
PMU_RF_FORCE_OPEN	软件强制打开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:

```
/* enable RF sequence force open/close */
```

```
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 3-424. 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	强制打开/关闭RF电源失能
先决条件	-
被调用函数	-
输入参数{in}	
force	强制打开/关闭RF电源
PMU_RF_FORCE_OPEN	软件强制打开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 RF sequence open/close force */
```

```
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 3-425. 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	配置RF时序
先决条件	-
被调用函数	-
输入参数{in}	
mode	RF序列模式

<i>PMU_RF_SOFTWARE</i>	使能RF软件序列
<i>PMU_RF_HARDWARE</i>	失能RF软件序列
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* configure RF sequence */
```

```
pmu_rf_sequence_config(PMU_RF_SOFTWARE);
```

### 函数 pmu\_flag\_get

函数pmu\_flag\_get描述见下表:

表 3-426. 函数 pmu\_flag\_get

函数名称	pmu_flag_get
函数原型	FlagStatus pmu_flag_get(uint32_t flag);
功能描述	获取标志位
先决条件	-
被调用函数	-
输入参数{in}	
flag	标志位
<i>PMU_FLAG_WAKEUP</i>	唤醒标志
<i>PMU_FLAG_STANDBY</i>	待机标志
<i>PMU_FLAG_LVD</i>	低电压状态标志
<i>PMU_FLAG_LDOVSOF</i>	LDO电压选择就绪标志
<i>PMU_FLAG_LDRF</i>	low-driver mode ready flag 低驱动模式就绪标志
<i>PMU_FLAG_WIFI_SLEEP</i>	WIFI is in sleep state WIFI处于睡眠状态
<i>PMU_FLAG_WIFI_ACTIVE</i>	WIFI is in active state WIFI处于运行状态
<i>PMU_FLAG_BLE_POWER</i>	BLE power status BLE电源状态
<i>PMU_FLAG_SRAM1_SLEEP</i>	SRAM1 is in sleep state SRAM1处于睡眠状态
<i>PMU_FLAG_SRAM1</i>	SRAM1 is in active state

1_ACTIVE	SRAM1处于运行状态
PMU_FLAG_SRAM0_SLEEP	SRAM0 is in sleep state SRAM0处于睡眠状态
PMU_FLAG_SRAM0_ACTIVE	SRAM0 is in active state SRAM0处于运行状态
PMU_FLAG_SRAM2_SLEEP	SRAM2处于睡眠状态
PMU_FLAG_SRAM2_ACTIVE	SRAM2处于运行状态
PMU_FLAG_SRAM3_SLEEP	SRAM3处于睡眠状态
PMU_FLAG_SRAM3_ACTIVE	SRAM3处于运行状态
PMU_FLAG_BLE_SLEEP	BLE处于睡眠状态
PMU_FLAG_BLE_ACTIVE	BLE处于运行状态
PMU_FLAG_BLE_POWER_FALL	BLE电源状态下降沿标志
PMU_FLAG_BLE_POWER_RISE	BLE电源状态上升沿标志
输出参数{out}	
-	-
返回值	
FlagStatus	SET或RESET

例如:

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

```
FlagStatus status;
```

```
status = pmu_flag_get(PMU_FLAG_WAKEUP);
```

### 函数 pmu\_flag\_clear

函数pmu\_flag\_clear描述见下表:

表 3-427. 函数 pmu\_flag\_clear

函数名称	pmu_flag_clear
函数原型	void pmu_flag_clear(uint32_t flag);
功能描述	清除标志位
先决条件	-
被调用函数	-
输入参数{in}	
flag	标志位



<i>PMU_FLAG_RESE T_WAKEUP</i>	复位唤醒标志
<i>PMU_FLAG_RESE T_STANDBY</i>	复位待机标志
<i>PMU_FLAG_RESE T_LDRF</i>	复位低驱动模式就绪标志
输出参数{out}	
-	
返回值	
-	

例如：

```
/* clear flag bit */
```

```
pmu_flag_clear(PMU_FLAG_RESET_WAKEUP);
```

### 函数 pmu\_interrupt\_enable

函数pmu\_interrupt\_enable描述见下表：

**表 3-428. 函数 pmu\_interrupt\_enable**

函数名称	pmu_interrupt_enable
函数原型	void pmu_interrupt_enable(uint32_t interrupt);
功能描述	使能PMU中断
先决条件	-
被调用函数	-
输入参数{in}	
interrupt	PMU中断
<i>PMU_INT_BLE_PO WER_FALL</i>	BLE电源状态下降沿中断
<i>PMU_INT_BLE_PO WER_RISE</i>	BLE电源状态上升沿中断
输出参数{out}	
-	
返回值	
-	

例如：

```
/* enable PMU interrupt */
```

```
pmu_interrupt_enable(PMU_INT_BLE_POWER_FALL);
```

### 函数 pmu\_interrupt\_disable

函数pmu\_interrupt\_disable描述见下表：

表 3-429. 函数 pmu\_interrupt\_disable

函数名称	pmu_interrupt_disable
函数原型	void pmu_interrupt_disable(uint32_t interrupt);
功能描述	失能PMU中断
先决条件	-
被调用函数	-
输入参数{in}	
interrupt	PMU中断
PMU_INT_BLE_POWER_FALL	BLE电源状态下降沿中断
PMU_INT_BLE_POWER_RISE	BLE电源状态上升沿中断
输出参数{out}	
-	
返回值	
-	

例如：

```
/* disable PMU interrupt */
pmu_interrupt_disable(PMU_INT_BLE_POWER_FALL);
```

### 函数 pmu\_interrupt\_flag\_get

函数pmu\_interrupt\_flag\_get描述见下表：

表 3-430. 函数 pmu\_interrupt\_flag\_get

函数名称	pmu_interrupt_flag_get
函数原型	FlagStatus pmu_interrupt_flag_get(uint32_t int_flag);
功能描述	获取PMU中断标志
先决条件	-
被调用函数	-
输入参数{in}	
int_flag	PMU中断标志
PMU_INT_FLAG_BLE_POWER_FALL	BLE电源状态下降沿中断标志
PMU_INT_FLAG_BLE_POWER_RISE	BLE电源状态上升沿中断标志
输出参数{out}	
-	
返回值	
-	

例如：

```
/* get PMU interrupt flag */
```

```
pmu_interrupt_flag_get(PMU_INT_FLAG_BLE_POWER_FALL);
```

### 函数 pmu\_interrupt\_flag\_clear

函数pmu\_interrupt\_flag\_clear描述见下表：

**表 3-431. 函数 pmu\_interrupt\_flag\_clear**

函数名称	pmu_interrupt_flag_clear
函数原型	FlagStatus pmu_interrupt_flag_clear(uint32_t int_flag);
功能描述	清除PMU中断标志
先决条件	-
被调用函数	-
输入参数{in}	
int_flag	PMU中断标志
PMU_INT_FLAG_BLE_POWER_FALL	BLE电源状态下降沿中断标志
PMU_INT_FLAG_BLE_POWER_RISE	BLE电源状态上升沿中断标志
输出参数{out}	
-	
返回值	
-	

例如：

```
/* clear PMU interrupt flag */
```

```
pmu_interrupt_flag_get(PMU_INT_FLAG_RESET_BLE_POWER_FALL);
```

## 3.17. 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[3.17.1](#)描述了QSPI的寄存器列表，章节[3.17.2](#)对QSPI库函数进行说明。

### 3.17.1. 外设寄存器说明

QSPI寄存器列表如下表所示：

**表 3-432. QSPI 寄存器**

寄存器名称	寄存器描述
QSPI_CTL	QSPI控制寄存器
QSPI_DCFG	QSPI设备配置寄存器
QSPI_STAT	QSPI状态寄存器
QSPI_STATC	QSPI状态清除寄存器

寄存器名称	寄存器描述
QSPI_DTLEN	QSPI数据长度寄存器
QSPI_TCFG	QSPI传输配置寄存器
QSPI_ADDR	QSPI地址寄存器
QSPI_ALTE	QSPI交替字节寄存器
QSPI_DATA	QSPI数据寄存器
QSPI_STATMK	QSPI状态屏蔽寄存器
QSPI_STATMATCH	QSPI状态匹配寄存器
QSPI_INTERVAL	QSPI间隔寄存器
QSPI_TMOUT	QSPI超时寄存器
QSPI_FLUSH	QSPI FIFO刷新寄存器

### 3.17.2. 外设库函数说明

QSPI库函数列表如下表所示：

**表 3-433. QSPI 库函数**

库函数名称	库函数描述
qspi_deinit	复位外设QSPI
qspi_struct_para_init	将QSPI初始化结构体中所有参数初始化为默认值
qspi_cmd_struct_para_init	将QSPI命令结构体中所有参数初始化为默认值
qspi_polling_struct_para_init	将QSPI读轮询结构体中所有参数初始化为默认值
qspi_init	初始化QSPI
qspi_enable	使能QSPI
qspi_disable	禁能QSPI
qspi_dma_enable	使能QSPI DMA
qspi_dma_disable	禁能QSPI DMA
qspi_command_config	配置QSPI命令参数
qspi_polling_config	配置QSPI读轮询模式
qspi_memorymapped_config	配置QSPI存储器映射模式
qspi_data_transmit	QSPI发送数据
qspi_data_receive	QSPI接收数据
qspi_transmission_abort	终止当前传输
qspi_flag_get	获取QSPI标志状态
qspi_flag_clear	清除QSPI标志状态
qspi_interrupt_enable	使能QSPI中断
qspi_interrupt_disable	禁能QSPI中断
qspi_interrupt_flag_get	获取QSPI中断标志状态
qspi_interrupt_flag_clear	清除QSPI中断标志状态

## 结构体 `qspi_init_struct`

表 3-434. 结构体 `qspi_init_struct`

成员名称	功能描述
<code>prescaler</code>	QSPI分频
<code>fifo_threshold</code>	QSPI FIFO 阈值
<code>sample_shift</code>	QSPI采样移位
<code>flash_size</code>	外部flash大小
<code>cs_high_time</code>	两个命令序列之间cs信号保持高电平的周期数
<code>clock_mode</code>	QSPI时钟模式

## 结构体 `qspi_command_struct`

表 3-435. 结构体 `qspi_command_struct`

成员名称	功能描述
<code>instruction_mode</code>	命令模式
<code>instruction</code>	8位命令
<code>addr_mode</code>	地址模式
<code>addr_size</code>	地址大小
<code>addr</code>	外部存储地址
<code>altebytes_mode</code>	交替字节模式
<code>altebytes_size</code>	交替字节大小
<code>altebytes</code>	交替字节信息
<code>dummycycles</code>	空闲周期
<code>data_mode</code>	数据模式
<code>data_length</code>	32位数据长度
<code>sioo_mode</code>	只发送一次指令模式

## 结构体 `qspi_polling_struct`

表 3-436. 结构体 `qspi_polling_struct`

成员名称	功能描述
<code>match</code>	匹配值
<code>mask</code>	屏蔽值
<code>interval</code>	两次状态轮询之间的时钟周期数
<code>statusbytes_size</code>	接收状态字节大小
<code>match_mode</code>	匹配方法
<code>polling_stop</code>	匹配后是否自动终止状态轮询

## 函数 `qspi_deinit`

函数`qspi_deinit`描述见下表：

表 3-437. 函数 qspi\_deinit

函数名称	qspi_deinit
函数原形	void qspi_deinit(void);
功能描述	复位外设QSPI
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 QSPI */
qspi_deinit();
```

### 函数 qspi\_struct\_para\_init

函数qspi\_struct\_para\_init描述见下表：

表 3-438. 函数 qspi\_struct\_para\_init

函数名称	qspi_struct_para_init
函数原形	void qspi_struct_para_init(qspi_init_struct *init_para);
功能描述	将QSPI初始化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}	
-	-
输出参数{out}	
init_para	QSPI初始化参数结构体，结构体成员可参考 <a href="#">表3-434. 结构体 qspi_init_struct</a> 。
返回值	
-	-

例如：

```
/* initialize the parameters of QSPI */
qspi_init_struct qspi_init_para;
qspi_struct_para_init(&qspi_init_para);
```

### 函数 qspi\_cmd\_struct\_para\_init

函数qspi\_cmd\_struct\_para\_init描述见下表：

表 3-439. 函数 `qspi_cmd_struct_para_init`

函数名称	<code>qspi_cmd_struct_para_init</code>
函数原形	<code>void qspi_cmd_struct_para_init(qspi_command_struct *init_para);</code>
功能描述	将QSPI命令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}	
-	-
输出参数{out}	
<code>init_para</code>	QSPI命令结构体，结构体成员可参考 <a href="#">表3-435. 结构体 <code>qspi_command_struct</code></a>
返回值	
-	-

例如：

```
/* initialize the parameters of QSPI command structtrue */
qspi_command_struct qspi_cmd_para;
qspi_cmd_struct_para_init(&qspi_cmd_para);
```

### 函数 `qspi_polling_struct_para_init`

函数`qspi_polling_struct_para_init`描述见下表：

表 3-440. 函数 `qspi_polling_struct_para_init`

函数名称	<code>qspi_polling_struct_para_init</code>
函数原形	<code>void qspi_polling_struct_para_init(qspi_polling_struct *init_para);</code>
功能描述	将QSPI读轮询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}	
-	-
输出参数{out}	
<code>init_para</code>	QSPI读轮询结构体，结构体成员可参考 <a href="#">表3-436. 结构体 <code>qspi_polling_struct</code></a>
返回值	
-	-

例如：

```
/* initialize the parameters of QSPI read polling structtrue */
qspi_polling_struct qspi_polling_para;
qspi_polling_struct_para_init(&qspi_polling_para);
```

## 函数 qspi\_init

函数qspi\_init描述见下表：

**表 3-441. 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QSPI
先决条件	-
被调用函数	-
输入参数{in}	
qspi_struct	QSPI初始化参数结构体，结构体成员可参考 <a href="#">表3-434. 结构体 qspi_init_struct</a> 。
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* initialize the init parameter structure */
qspi_struct_para_init(&qspi_init_para);
qspi_init_para.clock_mode = QSPI_CLOCK_MODE_0;
qspi_init_para.fifo_threshold = 4;
qspi_init_para.sample_shift = QSPI_SAMPLE_SHIFTING_HALFCYCLE;
qspi_init_para.cs_high_time = QSPI_CS_HIGH_TIME_2_CYCLE;
qspi_init_para.flash_size = 27;
qspi_init_para.prescaler = 4;
qspi_init(&qspi_init_para);
```

## 函数 qspi\_enable

函数qspi\_enable描述见下表：

**表 3-442. 函数 qspi\_enable**

函数名称	qspi_enable
函数原形	void qspi_enable(void);
功能描述	使能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 3-443. 函数 qspi\_disable**

函数名称	qspi_disable
函数原形	void qspi_disable(void);
功能描述	禁能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 3-444. 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
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 */
```

```
qspi_dma_enable();
```

### 函数 qspi\_dma\_disable

函数qspi\_dma\_disable描述见下表：

**表 3-445. 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	禁能QSPI DMA
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 */
```

```
qspi_dma_disable();
```

### 函数 qspi\_data\_length\_config

函数qspi\_data\_length\_config描述见下表：

**表 3-446. 函数 qspi\_data\_length\_config**

函数名称	qspi_data_length_config
函数原形	void qspi_data_length_config(uint32_t dtlen);
功能描述	配置QSPI数据长度
先决条件	-
被调用函数	-
输入参数{in}	
<b>dtlen</b>	QSPI数据长度(1 ~ 4294967295 (4G-1))
输出参数{out}	

-	-
返回值	
-	-

例如：

```
/* configure QSPI data length */
```

```
qspi_data_length_config(8);
```

### 函数 **qspi\_command\_config**

函数qspi\_command\_config描述见下表：

**表 3-447. 函数 qspi\_command\_config**

函数名称	qspi_command_config
函数原形	void qspi_command_config(qspi_command_struct *cmd);
功能描述	配置QSPI命令参数
先决条件	-
被调用函数	-
输入参数{in}	
cmd	QSPI命令参数结构体，结构体成员可参考 <a href="#">表3-435. 结构体 qspi_command_struct</a> 。
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* initialize the QSPI command parameter structure */
```

```
qspi_cmd_struct_para_init(&qspi_cmd);
```

```
/* write enable */
```

```
qspi_cmd.instruction_mode = QSPI_INSTRUCTION_1_LINE;
```

```
qspi_cmd.instruction = WRITE_ENABLE_CMD;
```

```
qspi_cmd.addr_mode = QSPI_ADDR_NONE;
```

```
qspi_cmd.altebytes_mode = QSPI_ALTE_BYTES_NONE;
```

```
qspi_cmd.data_mode = QSPI_DATA_NONE;
```

```
qspi_cmd.dummycycles = 0;
```

```
qspi_cmd.sioo_mode = QSPI_SIOO_INST_EVERY_CMD;
```

```
qspi_command_config(&qspi_cmd);
```

## 函数 qspi\_polling\_config

函数qspi\_polling\_config描述见下表：

**表 3-448. 函数 qspi\_polling\_config**

函数名称	qspi_polling_config
函数原形	void qspi_polling_config(qspi_command_struct *cmd, qspi_polling_struct *cfg);
功能描述	配置QSPI读轮询模式
先决条件	-
被调用函数	-
输入参数{in}	
cmd	QSPI命令参数结构体，结构体成员可参考 <a href="#">表3-435. 结构体 qspi_command_struct</a> 。
输入参数{in}	
cfg	QSPI读轮询参数结构体，结构体成员可参考 <a href="#">表3-436. 结构体 qspi_polling_struct</a> 。
输出参数{out}	
-	-
返回值	
-	-

例如：

```

/* initialize the QSPI command parameter structure */
qspi_cmd_struct_para_init(&qspi_cmd);

/* initialize the QSPI read polling parameter structure */
qspi_polling_struct_para_init(&polling_cmd);

/* read enable */

polling_cmd.match = 0x02;

polling_cmd.mask = 0x02;

polling_cmd.match_mode = QSPI_MATCH_MODE_AND;

polling_cmd.statusbytes_size = 1;

polling_cmd.interval = 0x10;

polling_cmd.polling_stop = QSPI_POLLING_STOP_ENABLE;

qspi_cmd.instruction = READ_STATUS_REG1_CMD;

qspi_cmd.data_mode = QSPI_DATA_1_LINE;

qspi_polling_config(&qspi_cmd, &polling_cmd);

```

函数 **qspi\_memorymapped\_config**

函数qspi\_memorymapped\_config描述见下表：

表 3-449. 函数 **qspi\_memorymapped\_config**

函数名称	qspi_memorymapped_config
函数原形	void qspi_memorymapped_config(qspi_command_struct *cmd, uint16_t timeout, uint32_t toen);
功能描述	配置QSPI存储器映射模式
先决条件	-
被调用函数	-
输入参数{in}	
cmd	QSPI命令参数结构体，结构体成员可参考 <a href="#">表3-435. 结构体 qspi_command_struct</a> 。
输入参数{in}	
timeout	0-0xFFFF
输入参数{in}	
toen	超时计数
QSPI_TMOUT_DISABLE	禁能超时计数
QSPI_TMOUT_ENABLE	使能超时计数
输出参数{out}	
-	-
返回值	
-	-

例如：

```

/* initialize the QSPI command parameter structure */

qspi_cmd_struct_para_init(&qspi_cmd);

qspi_cmd.instruction_mode = QSPI_INSTRUCTION_1_LINE;

qspi_cmd.instruction = RDID;

qspi_cmd.addr_mode = QSPI_ADDR_NONE;

qspi_cmd.addr_size = QSPI_ADDR_8_BITS;

qspi_cmd.addr = 0;

qspi_cmd.altebytes_mode = QSPI_ALTE_BYTES_1_LINE;

qspi_cmd.altebytes_size = QSPI_ALTE_BYTES_24_BITS;

qspi_cmd.altebytes = 0;

qspi_cmd.dummycycles = 0;

```

```
qspi_cmd.data_mode = QSPI_DATA_1_LINE;  
  
qspi_cmd.data_length = 3;  
  
qspi_cmd.sioo_mode = QSPI_SIOO_INST_EVERY_CMD;  
  
qspi_memorymapped_config(&qspi_cmd, 0xf, QSPI_TMOUT_ENABLE);
```

### 函数 qspi\_data\_transmit

函数qspi\_data\_transmit描述见下表:

表 3-450. 函数 qspi\_data\_transmit

函数名称	qspi_data_transmit
函数原形	void qspi_data_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	指向待发送数据的指针
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* QSPI transmit data */  
  
uint8_t data[32];  
  
qspi_data_transmit(&data);
```

### 函数 qspi\_data\_receive

函数qspi\_data\_receive描述见下表:

表 3-451. 函数 qspi\_data\_receive

函数名称	qspi_data_receive
函数原形	void qspi_data_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	指向待接收数据的指针
输出参数{out}	
-	-
返回值	

-	-
---	---

例如:

```
/* QSPI receive data */
```

```
uint8_t data[32];
```

```
qspi_data_receive(&data);
```

## 函数 qspi\_transmission\_abort

函数qspi\_transmission\_abort描述见下表:

**表 3-452. 函数 qspi\_transmission\_abort**

函数名称	qspi_transmission_abort
函数原形	void qspi_transmission_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 transmission */
```

```
qspi_transmission_abort();
```

## 函数 qspi\_flag\_get

函数qspi\_flag\_get描述见下表:

**表 3-453. 函数 qspi\_flag\_get**

函数名称	qspi_flag_get
函数原形	FlagStatus qspi_flag_get(uint32_t flag);
功能描述	获取QSPI标志状态
先决条件	-
被调用函数	-
输入参数{in}	
flag	QSPI标志
QSPI_FLAG_BUSY	忙标志
QSPI_FLAG_TERR	传输错误标志
QSPI_FLAG_TC	传输完成标志

QSPI_FLAG_FT	FIFO阈值标志
QSPI_FLAG_RPMF	读轮询匹配标志
QSPI_FLAG_TMOUT	超时标志
输出参数{out}	
-	-
返回值	
FlagStatus	SET或RESET

例如：

```
/* get QSPI transfer complete flag */
```

```
FlagStatus status;
```

```
status = qspi_flag_get(QSPI_FLAG_TC);
```

### 函数 qspi\_flag\_clear

函数qspi\_flag\_clear描述见下表：

表 3-454. 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
QSPI_FLAG_TERR	传输错误标志
QSPI_FLAG_TC	传输完成标志
QSPI_FLAG_RPMF	FIFO阈值标志
QSPI_FLAG_TMOUT	读轮询匹配标志
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* clear QSPI transfer complete flag status */
```

```
qspi_flag_clear(QSPI_FLAG_TC);
```

### 函数 qspi\_interrupt\_enable

函数qspi\_interrupt\_enable描述见下表：



表 3-455. 函数 `qspi_interrupt_enable`

函数名称	<code>qspi_interrupt_enable</code>
函数原形	<code>void qspi_interrupt_enable(uint32_t interrupt);</code>
功能描述	使能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_RPMF</code>	读轮询匹配中断
<code>QSPI_INT_TMOUT</code>	超时中断
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 3-456. 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	禁能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_RPMF</code>	读轮询匹配中断
<code>QSPI_INT_TMOUT</code>	超时中断
输出参数{out}	
-	-
返回值	

例如：

```
/* disable QSPI transfer complete interrupt */
```

```
qspi_interrupt_disable(QSPI_INT_TC);
```

### 函数 qspi\_interrupt\_flag\_get

函数qspi\_interrupt\_flag\_get描述见下表：

**表 3-457. 函数 qspi\_interrupt\_flag\_get**

函数名称	qspi_interrupt_flag_get
函数原形	FlagStatus qspi_interrupt_flag_get(uint32_t int_flag);
功能描述	获取QSPI中断标志状态
先决条件	-
被调用函数	-
输入参数{in}	
int_flag	QSPI中断标志
QSPI_INT_FLAG_TERR	传输错误中断标志
QSPI_INT_FLAG_TC	传输完成中断标志
QSPI_INT_FLAG_FT	FIFO阈值中断标志
QSPI_INT_FLAG_RPMF	读轮询匹配中断标志
QSPI_INT_FLAG_TMOU	超时中断标志
输出参数{out}	
-	-
返回值	
FlagStatus	SET或RESET

例如：

```
/* get QSPI transfer complete flag */
```

```
FlagStatus status;
```

```
status = qspi_flag_get(QSPI_FLAG_TC);
```

### 函数 qspi\_interrupt\_flag\_clear

函数qspi\_interrupt\_flag\_clear描述见下表：

**表 3-458. 函数 qspi\_interrupt\_flag\_clear**

函数名称	qspi_interrupt_flag_clear
函数原形	void qspi_interrupt_flag_clear(uint32_t int_flag);
功能描述	清除QSPI中断标志状态
先决条件	-

被调用函数	-
输入参数{in}	
int_flag	QSPI中断标志
QSPI_INT_FLAG_TERR	传输错误中断标志
QSPI_INT_FLAG_TC	传输完成中断标志
QSPI_INT_FLAG_RPMF	读轮询匹配中断标志
QSPI_INT_FLAG_TMOU	超时中断标志
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* clear QSPI transfer complete flag status */
qspi_flag_clear(QSPI_FLAG_TC);
```

## 3.18. RCU

RCU 是复位和时钟单元，复位控制包括三种控制方式：电源复位、系统复位和备份域复位。时钟控制单元提供了一系列频率的时钟功能。章节 [3.18.1](#) 描述了 RCU 的寄存器列表，章节 [3.18.2](#) 对 RCU 库函数进行说明。

### 3.18.1. 外设寄存器说明

RCU寄存器列表如下表所示：

表 3-459. 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

寄存器名称	寄存器描述
RCU_AHB3EN	AHB3使能寄存器
RCU_APB1EN	APB1使能寄存器
RCU_APB2EN	APB2使能寄存器
RCU_AHB1SPEN	AHB1睡眠模式使能寄存器
RCU_BDCTL	备份域控制寄存器
RCU_RSTSCK	复位源/时钟寄存器
RCU_PLLDIGCFG0	PLLDIG时钟配置寄存器0
RCU_CFG1	配置寄存器1
RCU_ADDCTL	附加时钟控制寄存器
RCU_PLLDIGCFG1	PLLDIG时钟配置寄存器1
RCU_VKEY	电源解锁寄存器
RCU_DSV	深度睡眠模式电压寄存器

### 3.18.2. 外设库函数说明

RCU库函数列表如下表所示：

**表 3-460. RCU 库函数**

库函数名称	库函数描述
rcu_deinit	复位RCU
rcu_irc16m_dfs_to_rf_enable	使能内部16 MHz RC RF差分时钟信号
rcu_irc16m_dfs_to_rf_disable	禁能内部16 MHz RC RF差分时钟信号
rcu_periph_clock_enable	使能外设时钟
rcu_periph_clock_disable	禁能外设时钟
rcu_fmc_clock_sleep_enable	在睡眠模式下，使能FMC时钟
rcu_fmc_clock_sleep_disable	在睡眠模式下，禁能FMC时钟
rcu_periph_reset_enable	使能外设时钟复位
rcu_periph_reset_disable	禁能外设时钟复位
rcu_bkp_reset_enable	使能备份域时钟复位
rcu_bkp_reset_disable	禁能备份域时钟复位
rcu_rfpll_cal_enable	使能RF PLL计算
rcu_rfpll_cal_disable	禁能RF PLL计算
rcu_control_unit_powerup	时钟上电
rcu_control_unit_powerdown	时钟断电
rcu_system_clock_source_config	配置选择系统时钟源
rcu_system_clock_source_get	获取系统时钟源选择状态
rcu_ahb_clock_config	配置AHB时钟预分频选择
rcu_apb1_clock_config	配置APB1时钟预分频选择
rcu_apb2_clock_config	配置APB2时钟预分频选择
rcu_ckout0_config	配置CKOUT0时钟源选择
rcu_ckout1_config	配置CKOUT1时钟源选择
rcu_plldig_config	配置PLLDIG输出时钟

库函数名称	库函数描述
rcu_plldigdiv_sys_config	配置PLLDIG时钟的分频因子用于系统时钟
rcu_rtc_clock_config	配置RTC时钟源选择
rcu_rtc_div_config	配置RTC时钟的预分频系数
rcu_trng_div_config	配置TRNG时钟的预分频系数
rcu_i2c0_clock_config	配置I2C0时钟源选择
rcu_usart0_clock_config	配置USART0时钟源选择
rcu_irc16m_div_config	配置内部16 MHz RC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_flag_get	获取时钟稳定状态和外设复位标志
rcu_all_reset_flag_clear	清除所有复位标志位
rcu_interrupt_flag_get	获取时钟中断和CKM中断标志
rcu_interrupt_flag_clear	清除中断标志
rcu_interrupt_enable	使能时钟稳定中断
rcu_interrupt_disable	禁能时钟稳定中断
rcu_lxtal_drive_capability_config	配置LXTAL驱动能力
rcu_osci_stab_wait	等待振荡器稳定标志位置位或振荡器起振超时
rcu_osci_on	打开振荡器
rcu_osci_off	关闭振荡器
rcu_osci_bypass_mode_enable	使能时钟旁路模式
rcu_osci_bypass_mode_disable	禁能时钟旁路模式
rcu_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_irc16m_adjust_value_set	设置内部16MHz RC振荡器时钟调整值
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、总线或外设时钟频率

## 枚举类型 rcu\_periph\_enum

表 3-461. 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_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

成员名称	功能描述
RCU_DMA	DMA时钟
RCU_BLE	BLE时钟
RCU_PKCAU	PKCAU时钟
RCU_CAU	CAU时钟
RCU_HAU	HAU时钟
RCU_TRNG	TRNG时钟
RCU_QSPI	QSPI时钟
RCU_TIMER1	TIMER1时钟
RCU_TIMER2	TIMER2时钟
RCU_TIMER5	TIMER5时钟
RCU_WWDGT	WWDGT时钟
RCU_RFI	RFI时钟
RCU_UART1	UART1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_UART2	UART2时钟
RCU_ADC	ADC时钟
RCU_SPI	SPI时钟
RCU_SYSCFG	SYSCFG时钟
RCU_TIMER15	TIMER15时钟
RCU_TIMER16	TIMER16时钟
RCU_RF	RF时钟

### 枚举类型 `rcu_periph_reset_enum`

表 3-462. 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_CRCRST	CRC时钟
RCU_WIFIRST	WIFI时钟
RCU_DMARST	DMA时钟
RCU_BLERST	BLE时钟
RCU_PKCAURST	PKCAU时钟
RCU_CAURST	CAU时钟
RCU_HAURST	HAU时钟

成员名称	功能描述
RCU_TRNGRST	TRNG时钟
RCU_QSPIRST	QSPI时钟
RCU_TIMER1RST	TIMER1时钟
RCU_TIMER2RST	TIMER2时钟
RCU_TIMER5RST	TIMER5时钟
RCU_WWDGTRST	WWDGT时钟
RCU_RFIRST	RFI时钟
RCU_UART1RST	UART1时钟
RCU_USART0RST	USART0时钟
RCU_I2C0RST	I2C0时钟
RCU_I2C1RST	I2C1时钟
RCU_PMURST	PMU时钟
RCU_TIMER0RST	TIMER0时钟
RCU_UART2RST	UART2时钟
RCU_ADCRST	ADC时钟
RCU_SPIRST	SPI时钟
RCU_SYSCFGRST	SYSCFG时钟
RCU_TIMER15RST	TIMER15时钟
RCU_TIMER16RST	TIMER16时钟
RCU_RFRST	RF时钟

### 枚举类型 `rcu_unit_enum`

表 3-463. 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

### 枚举类型 `rcu_flag_enum`

表 3-464. 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_LXTALSTB	LXTAL稳定标志
RCU_FLAG_IRC32KSTB	IRC32K稳定标志
RCU_FLAG_EPRST	外部引脚复位标志

成员名称	功能描述
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 3-465. 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_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 3-466. 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_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 3-467. 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_PLLDIGSTB	PLLDIG时钟稳定中断



## 枚举类型 `rcu_osc_type_enum`

表 3-468. 枚举类型 `rcu_osc_type_enum`

成员名称	功能描述
RCU_HXTAL	外部高速振荡器
RCU_LXTAL	外部低速振荡器
RCU_IRC16M	IRC16M振荡器
RCU_IRC32K	IRC32K振荡器
RCU_PLLDIG_CK	PLLDIG时钟

## 枚举类型 `rcu_clock_freq_enum`

表 3-469. 枚举类型 `rcu_clock_freq_enum`

成员名称	功能描述
CK_SYS	系统时钟
CK_AHB	AHB时钟
CK_APB1	APB1时钟
CK_APB2	APB2时钟
CK_USART0	USART0时钟
CK_I2C0	I2C0时钟

## 函数 `rcu_deinit`

函数`rcu_deinit`描述见下表：

表 3-470. 函数 `rcu_deinit`

函数名称	<code>rcu_deinit</code>
函数原形	<code>void rcu_deinit(void);</code>
功能描述	复位RCU，将RCU所有寄存器的值复位成初始值
先决条件	-
被调用函数	-
输入参数{in}	
-	-
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* deinitialize the RCU */
rcu_deinit();
```

## 函数 rcu\_irc16m\_dfs\_to\_rf\_enable

函数rcu\_irc16m\_dfs\_to\_rf\_enable描述见下表:

表 3-471. 函数 rcu\_irc16m\_dfs\_to\_rf\_enable

函数名称	rcu_irc16m_dfs_to_rf_enable
函数原形	void rcu_irc16m_dfs_to_rf_enable(void);
功能描述	使能内部16 MHz RC RF差分时钟信号
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 differential signal of IRC16M to RF module */
```

```
rcu_irc16m_dfs_to_rf_enable();
```

## 函数 rcu\_irc16m\_dfs\_to\_rf\_disable

函数rcu\_irc16m\_dfs\_to\_rf\_disable描述见下表:

表 3-472. 函数 rcu\_irc16m\_dfs\_to\_rf\_disable

函数名称	rcu_irc16m_dfs_to_rf_disable
函数原形	void rcu_irc16m_dfs_to_rf_disable(void);
功能描述	禁能内部16 MHz RC RF差分时钟信号
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 differential signal of IRC16M to RF module */
```

```
rcu_irc16m_dfs_to_rf_disable();
```

## 函数 rcu\_periph\_clock\_enable

函数rcu\_periph\_clock\_enable描述见下表：

表 3-473. 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3-461. 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 3-474. 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3-461. 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\_fmc\_clock\_sleep\_enable

函数rcu\_fmc\_clock\_sleep\_enable描述见下表：

**表 3-475. 函数 rcu\_fmc\_clock\_sleep\_enable**

函数名称	rcu_fmc_clock_sleep_enable
函数原形	void rcu_fmc_clock_sleep_enable(void);
功能描述	在睡眠模式下，使能FMC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 the FMC clock when in sleep mode */
```

```
rcu_fmc_clock_sleep_enable();
```

## 函数 rcu\_fmc\_clock\_sleep\_disable

函数rcu\_fmc\_clock\_sleep\_disable描述见下表：

**表 3-476. 函数 rcu\_fmc\_clock\_sleep\_disable**

函数名称	rcu_fmc_clock_sleep_disable
函数原形	void rcu_fmc_clock_sleep_disable(void);
功能描述	在睡眠模式下，禁能FMC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 the FMC clock when in sleep mode */
```

```
rcu_fmc_clock_sleep_disable();
```

## 函数 rcu\_periph\_reset\_enable

函数rcu\_periph\_reset\_enable描述见下表:

表 3-477. 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3-462. 枚举类型rcu_periph_reset_enum</a>
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* enable SPI reset */
```

```
rcu_periph_reset_enable(RCU_SPIRST);
```

## 函数 rcu\_periph\_reset\_disable

函数rcu\_periph\_reset\_disable描述见下表:

表 3-478. 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3-462. 枚举类型rcu_periph_reset_enum</a>
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* disable SPI reset */
```

```
rcu_periph_reset_disable(RCU_SPIRST);
```

## 函数 rcu\_bkp\_reset\_enable

函数rcu\_bkp\_reset\_enable描述见下表：

**表 3-479. 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 3-480. 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\_rfppl\_cal\_enable

函数rcu\_rfppl\_cal\_enable描述见下表：

**表 3-481. 函数 rcu\_rfppl\_cal\_enable**

函数名称	rcu_rfppl_cal_enable
函数原形	void rcu_rfppl_cal_enable(void);
功能描述	使能RF PLL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 3-482. 函数 rcu\_rfppl\_cal\_disable**

函数名称	rcu_rfppl_cal_disable
函数原形	void rcu_rfppl_cal_disable(void);
功能描述	禁能RF PLL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\_control\_unit\_powerup**

函数rcu\_control\_unit\_powerup描述见下表:

**表 3-483. 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	时钟单元
RCU_UNIT_HXTAL	高速晶体振荡器上电
RCU_UNIT_PLLDIG	PLLDIG上电
RCU_UNIT_RFPLL	上电RFPLL时钟
RCU_UNIT_LDOANA	LDO模拟域上电
RCU_UNIT_LDOCLK	LDO时钟上电
RCU_UNIT_BANDGAP	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 3-484. 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	时钟单元
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</i> <i>A</i>	LDO模拟域掉电
<i>RCU_UNIT_LDOCL</i> <i>K</i>	LDO时钟掉电
<i>RCU_UNIT_BANDG</i> <i>AP</i>	BandGap掉电
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* power down the HXTAL */
```

```
rcu_control_unit_powerdown(RCU_UNIT_HXTAL);
```

### 函数 **rcu\_system\_clock\_source\_config**

函数rcu\_system\_clock\_source\_config描述见下表:

**表 3-485. 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_I</i> <i>RC16M</i>	选择CK_IRC16M时钟作为CK_SYS时钟源
<i>RCU_CKSYSSRC_</i> <i>HXTAL</i>	选择CK_HXTAL时钟作为CK_SYS时钟源
<i>RCU_CKSYSSRC_</i> <i>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 3-486. 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_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 3-487. 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 3-488. 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_DIV1	选择CK_AHB为CK_APB1
RCU_APB1_CKAHB_DIV2	选择CK_AHB/2为CK_APB1
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 3-489. 函数 rcu\_apb2\_clock\_config

函数名称	rcu_apb2_clock_config
函数原形	void rcu_apb2_clock_config(uint32_t ck_apb2);
功能描述	配置APB2时钟预分频选择
先决条件	-
被调用函数	-
输入参数{in}	

<b>ck_apb2</b>	APB2预分频选择
<i>RCU_APB2_CKAHB_DIV1</i>	选择CK_AHB为CK_APB2
<i>RCU_APB2_CKAHB_DIV2</i>	选择CK_AHB/2为CK_APB2
<i>RCU_APB2_CKAHB_DIV4</i>	选择CK_AHB/4为CK_APB2
<i>RCU_APB2_CKAHB_DIV8</i>	选择CK_AHB/8为CK_APB2
<i>RCU_APB2_CKAHB_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 3-490. 函数 rcu\_ckout0\_config**

函数名称	rcu_ckout0_config
函数原形	void rcu_ckout0_config(uint32_t ckout0_src, uint32_t ckout0_div);
功能描述	配置CKOUT0时钟源选择
先决条件	-
被调用函数	-
输入参数{in}	
<b>ckout0_src</b>	CKOUT0时钟源选择
<i>RCU_CKOUT0SRC_IRC16M</i>	选择内部16M RC振荡器时钟
<i>RCU_CKOUT0SRC_LXTAL</i>	选择低速晶体振荡器时钟（LXTAL）
<i>RCU_CKOUT0SRC_HXTAL</i>	选择高速晶体振荡器时钟（HXTAL）
<i>RCU_CKOUT0SRC_PLLDIG</i>	选择CK_PLLDIG时钟
<i>RCU_CKOUT0SRC_IRC32K</i>	选择内部32K RC振荡器时钟
<i>RCU_CKOUT0SRC</i>	选择系统时钟

<code>_CKSYS</code>	
输入参数{in}	
<code>ckout0_div</code>	CK_OUT0分频选择
<code>RCU_CKOUT0_DIV</code> <code>x</code>	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 3-491. 函数 `rcu_ckout1_config`**

函数名称	<code>rcu_ckout1_config</code>
函数原形	<code>void rcu_ckout1_config(uint32_t ckout1_src, uint32_t ckout1_div);</code>
功能描述	配置CKOUT1时钟源选择
先决条件	-
被调用函数	-
输入参数{in}	
<code>ckout1_src</code>	CKOUT1时钟源选择
<code>RCU_CKOUT1SRC</code> <code>_CKSYS</code>	选择系统时钟
<code>RCU_CKOUT1SRC</code> <code>_IRC16M</code>	选择内部16M RC振荡器时钟
<code>RCU_CKOUT1SRC</code> <code>_HXTAL</code>	选择高速晶体振荡器时钟（HXTAL）
<code>RCU_CKOUT1SRC</code> <code>_PLLDIG</code>	选择PLLDIG时钟
输入参数{in}	
<code>ckout1_div</code>	CK_OUT1分频选择
<code>RCU_CKOUT1_DIV</code> <code>x</code>	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\_plldig\_config

函数rcu\_plldig\_config描述见下表：

**表 3-492. 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输出时钟
先决条件	-
被调用函数	-
输入参数{in}	
plldig_clk	PLLDIG输出时钟选择
RCU_PLLDIG_192M	选择192Mhz为PLLDIG时钟输出
RCU_PLLDIG_240M	选择240Mhz为PLLDIG时钟输出
RCU_PLLDIG_320M	选择320Mhz为PLLDIG时钟输出
RCU_PLLDIG_480M	选择480Mhz为PLLDIG时钟输出
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* configure the PLLDIG output 192Mhz clock frequency */
rcu_plldig_config(RCU_PLLDIG_192M);
```

### 函数 rcu\_plldigdiv\_sys\_config

函数rcu\_plldigdiv\_sys\_config描述见下表：

**表 3-493. 函数 rcu\_plldigdiv\_sys\_config**

函数名称	rcu_plldig_div_sys_config
函数原形	void rcu_plldigdiv_sys_config(uint32_t plldigdiv_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}	
<b>plldig_div_sys</b>	PLLDIG时钟分频系数输出给系统时钟
<i>RCU_PLLDIG_SYS</i> <i>_DIVx</i>	PLLDIG时钟/x作为系统时钟(x=1,2,3,...,64)
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* configure PLLDIG clock divider 2 for system clock */
rcu_plldigdiv_sys_config(RCU_PLLDIG_SYS_DIV2);
```

### 函数 rcu\_rtc\_clock\_config

函数rcu\_rtc\_clock\_config描述见下表:

表 3-494. 函数 rcu\_rtc\_clock\_config

<b>函数名称</b>	rcu_rtc_clock_config
<b>函数原形</b>	void rcu_rtc_clock_config(uint32_t rtc_clock_source);
<b>功能描述</b>	配置RTC的时钟源选择
<b>先决条件</b>	-
<b>被调用函数</b>	-
输入参数{in}	
<b>rtc_clock_source</b>	RTC时钟源选择
<i>RCU_RTCSRC_NO</i> <i>NE</i>	没有时钟
<i>RCU_RTCSRC_LX</i> <i>TAL</i>	选择CK_LXTAL时钟作为RTC的时钟源
<i>RCU_RTCSRC_IRC</i> <i>32K</i>	选择CK_IRC32K时钟作为RTC的时钟源
<i>RCU_RTCSRC_HX</i> <i>TAL_DIV_RTCDIV</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_RTCSRC_IRC32K);
```

## 函数 rcu\_rtc\_div\_config

函数rcu\_rtc\_div\_config描述见下表:

**表 3-495. 函数 rcu\_rtc\_div\_config**

函数名称	rcu_rtc_div_config
函数原形	void rcu_rtc_div_config(uint32_t rtc_div);
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}	
rtc_div	RTC时钟源分频系数
RCU_RTC_HXTAL_DIVx	CK_HXTAL/x作为RTCDIV时钟 (x = 1....32)
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\_trng\_div\_config

函数rcu\_trng\_div\_config描述见下表:

**表 3-496. 函数 rcu\_trng\_div\_config**

函数名称	rcu_trng_div_config
函数原形	void rcu_trng_div_config(uint32_t trng_div);
功能描述	配置TRNG时钟的预分频系数
先决条件	-
被调用函数	-
输入参数{in}	
trng_div	TRNG时钟的预分频系数
RCU_TRNG_DIVx	TRNG输入时钟/x (x = 1....32)
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* TRNG clock divide 2 */
```



rcu\_trng\_div\_config (RCU\_TRNG\_DIV2);

### 函数 rcu\_i2c0\_clock\_config

函数rcu\_i2c0\_clock\_config描述见下表:

表 3-497. 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 rcu\_usart0\_clock\_source

函数rcu\_usart0\_clock\_source描述见下表:

表 3-498. 函数 rcu\_usart0\_clock\_source

函数名称	rcu_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_CK APB1	选择CK_APB1为USART0时钟源

<code>RCU_USART0SRC_CKSYS</code>	选择CK_SYS为USART0时钟源
<code>RCU_USART0SRC_LXTAL</code>	选择LXTAL为USART0时钟源
<code>RCU_USART0SRC_IRC16M</code>	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 `rcu_irc16m_div_config`

函数`rcu_irc16m_div_config`描述见下表:

表 3-499. 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}	
<code>irc16m_div</code>	提供给系统时钟的IRC16M时钟分频系数
<code>RCU_IRC16M_DIVx</code>	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 3-500. 函数 `rcu_timer_clock_prescaler_config`

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

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\_flag\_get

函数rcu\_flag\_get描述见下表:

表 3-501. 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3-464. 枚举类型rcu_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 3-502. 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 3-503. 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3-465. 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 3-504. 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3-466. 枚举类型 rcu_int_flag_clear_enum</a>
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* clear the interrupt HXTAL stabilization interrupt flag */
rcu_interrupt_flag_clear(RCU_INT_FLAG_HXTALSTB_CLR);
```

## 函数 rcu\_interrupt\_enable

函数rcu\_interrupt\_enable描述见下表：

**表 3-505. 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3-467. 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 3-506. 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3-467. 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);
```

## 函数 rcu\_lxtal\_drive\_capability\_config

函数rcu\_lxtal\_drive\_capability\_config描述见下表:

**表 3-507. 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_LXTALDRI_LOWER_DRIVE	低驱动能力
RCU_LXTALDRI_HIGH_DRIVE	中高驱动能力
RCU_LXTALDRI_HIGHER_DRIVE	高驱动能力
RCU_LXTALDRI_HIGHEST_DRIVE	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_LXTALDRI_LOWER_DRIVE);
```

### 函数 rcu\_osci\_stab\_wait

函数rcu\_osci\_stab\_wait描述见下表：

**表 3-508. 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3-468. 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 3-509. 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3-468. 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 3-510. 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3-468. 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 3-511. 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}	
osci	振荡器类型，参考 <a href="#">表3-468. 枚举类型rcu_osci_type_enum</a>
RCU_HXTAL	高速晶体振荡器
RCU_LXTAL	低速晶体振荡器
输出参数{out}	
-	-
返回值	
-	-



例如：

```
/* enable the high speed crystal oscillator bypass mode */
```

```
rcu_osci_bypass_mode_enable(RCU_HXTAL);
```

### 函数 rcu\_osci\_bypass\_mode\_disable

函数rcu\_osci\_bypass\_mode\_disable描述见下表：

**表 3-512. 函数 rcu\_osci\_bypass\_mode\_disable**

函数名称	rcu_osci_bypass_mode_disable
函数原形	void rcu_osci_bypass_mode_disable(rcu_osci_type_enum osci);
功能描述	禁能振荡器时钟旁路模式
先决条件	HXTALEN或LXTALEN应在使能振荡器时钟旁路模式前先复位
被调用函数	-
输入参数{in}	
osci	振荡器类型，参考 <a href="#">表3-468. 枚举类型rcu_osci_type_enum</a>
RCU_HXTAL	高速晶体振荡器
RCU_LXTAL	低速晶体振荡器
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* disable the high speed crystal oscillator bypass mode */
```

```
rcu_osci_bypass_mode_disable(RCU_HXTAL);
```

### 函数 rcu\_rf\_hxtal\_clock\_monitor\_enable

函数rcu\_rf\_hxtal\_clock\_monitor\_enable描述见下表：

**表 3-513. 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 3-514. 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\_irc16m\_adjust\_value\_set

函数rcu\_irc16m\_adjust\_value\_set描述见下表：

**表 3-515. 函数 rcu\_irc16m\_adjust\_value\_set**

函数名称	rcu_irc16m_adjust_value_set
函数原形	void rcu_irc16m_adjust_value_set(uint32_t irc16m_adjval);
功能描述	设置内部16MHz RC振荡器时钟调整值
先决条件	-
被调用函数	-
输入参数{in}	
irc16m_adjval	IRC16M调整值（0到0x1F之间）
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* set the IRC16M adjust value */
```

```
rcu_irc16m_adjust_value_set(0x10);
```

## 函数 rcu\_voltage\_key\_unlock

函数rcu\_voltage\_key\_unlock描述见下表：

**表 3-516. 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 3-517. 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
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 3-518. 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3-469. 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);
```

## 3.19. RTC

实时时钟RTC通常被用作时钟日历。位于备份域中的RTC电路，包含一个32位的累加计数器、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[3.19.1](#)描述了RTC的寄存器列表，章节[3.19.2](#)对RTC库函数进行说明。

### 3.19.1. 外设寄存器描述

RTC寄存器列表如下表所示：

表 3-519. RTC 寄存器

寄存器名称	寄存器描述
RTC_TIME	时间寄存器
RTC_DATE	日期寄存器
RTC_CTL	控制寄存器
RTC_STAT	状态寄存器
RTC_PSC	预分频寄存器
RTC_WUT	唤醒定时器寄存器
RTC_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_BKPx(x = 0, 1, 2, ..., 18, 19)	备份寄存器

### 3.19.2. 外设库函数描述

RTC库函数列表如下表所示：

表 3-520. 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

库函数名称	库函数描述
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_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_output_pin_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
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_flag_get	获取RTC标志位
rtc_flag_clear	清除RTC标志位
rtc_interrupt_enable	使能RTC中断
rtc_interrupt_disable	失能RTC中断

## 结构体 rtc\_parameter\_struct

表 3-521. 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 3-522. 结构体 rtc\_alarm\_struct

成员名称	功能描述
alarm_mask	RTC闹钟屏蔽
weekday_or_date	指定RTC闹钟是日期还是星期几
alarm_day	RTC闹钟日期或者星期几的值
alarm_hour	RTC闹钟小时值
alarm_minute	RTC闹钟分钟值: 0x0 - 0x59(BCD格式)
alarm_second	RTC闹钟秒数值: 0x0 - 0x59(BCD格式)
am_pm	RTC闹钟AM/PM数值

### 结构体 rtc\_timestamp\_struct

表 3-523. 结构体 rtc\_timestamp\_struct

成员名称	功能描述
timestamp_month	RTC时间戳月份值
timestamp_date	RTC 时间戳日期值: 0x1 - 0x31(BCD格式)
timestamp_day	RTC时间戳星期值
timestamp_hour	RTC时间戳小时值
timestamp_minute	RTC时间戳分钟值: 0x0 - 0x59(BCD格式)
timestamp_second	RTC时间戳秒数值: 0x0 - 0x59(BC格式)
am_pm	RTC时间戳AM/PM数值

### 结构体 rtc\_tamper\_struct

表 3-524. 结构体 rtc\_tamper\_struct

成员名称	功能描述
tamper_source	RTC侵入检测源
tamper_trigger	RTC侵入事件检测触发沿
tamper_filter	RTC 侵入事件检测在电平检测期间需要的连续采样次数
tamper_sample_frequency	RTC侵入事件电平模式检测的采样频率
tamper_precharge_enable	RTC在电压电平检测期间的预充电功能
tamper_precharge_time	RTC侵入事件电平检测采样预充电时间, 如果预充电功能使能

tamper_with_timest amp	RTC侵入事件触发时间戳
---------------------------	--------------

### 函数 rtc\_deinit

函数rtc\_deinit描述见下表:

表 3-525. 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;
```

```
error_status = rtc_deinit();
```

### 函数 rtc\_init

函数rtc\_init描述见下表:

表 3-526. 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3-521. 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 3-527. 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;

error_status = rtc_init_mode_enter ();

```

### 函数 rtc\_init\_mode\_exit

函数rtc\_init\_mode\_exit描述见下表：

表 3-528. 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 3-529. 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;
```

```
error_status = rtc_register_sync_wait ( );
```

## 函数 rtc\_current\_time\_get

函数rtc\_current\_time\_get描述见下表：

表 3-530. 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3-521. 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 3-531. 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 3-532. 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}	
rtc_alarm_time	闹钟结构体，结构体成员参考 <a href="#">表3-522. 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 3-533. 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}	
rtc_alarm	闹钟选择
RTC_ALARM0	闹钟0
RTC_ALARM1	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_ALARM0SS_SSC[14:1], SSC[0]位用于时间匹配
<i>RTC_MASKSSC_2_14</i>	屏蔽RTC_ALARM0SS_SSC[14:2], SSC[1:0]位用于时间匹配
<i>RTC_MASKSSC_3_14</i>	屏蔽RTC_ALARM0SS_SSC[14:3], SSC[2:0]位用于时间匹配
<i>RTC_MASKSSC_4_14</i>	屏蔽RTC_ALARM0SS_SSC[14:4], SSC[3:0]位用于时间匹配
<i>RTC_MASKSSC_5_14</i>	屏蔽RTC_ALARM0SS_SSC[14:5], SSC[4:0]位用于时间匹配
<i>RTC_MASKSSC_6_14</i>	屏蔽RTC_ALARM0SS_SSC[14:6], SSC[5:0]位用于时间匹配
<i>RTC_MASKSSC_7_14</i>	屏蔽RTC_ALARM0SS_SSC[14:7], SSC[6:0]位用于时间匹配
<i>RTC_MASKSSC_8_14</i>	屏蔽RTC_ALARM0SS_SSC[14:8], SSC[7:0]位用于时间匹配
<i>RTC_MASKSSC_9_14</i>	屏蔽RTC_ALARM0SS_SSC[14:9], SSC[8:0]位用于时间匹配
<i>RTC_MASKSSC_10_14</i>	屏蔽RTC_ALARM0SS_SSC[14:10], SSC[9:0]位用于时间匹配
<i>RTC_MASKSSC_11_14</i>	屏蔽RTC_ALARM0SS_SSC[14:11], SSC[10:0]位用于时间匹配
<i>RTC_MASKSSC_12_14</i>	屏蔽RTC_ALARM0SS_SSC[14:12], SSC[11:0]位用于时间匹配
<i>RTC_MASKSSC_13_14</i>	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
<b>输入参数{in}</b>	
<b>subsecond</b>	闹钟亚秒值(0x000 - 0x7FFF)
<b>输出参数{out}</b>	
-	-
<b>返回值</b>	
-	-

例如:

```
/*configure subsecond of RTC alarm0*/
```

```
rtc_subsecond_config(RTC_ALARM0, RTC_MASKSSC_9_14, 0x7FFF);
```

### 函数 rtc\_alarm\_get

函数rtc\_alarm\_get描述见下表:

**表 3-534. 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3-522. 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 3-535. 函数 rtc\_alarm\_subsecond\_get**

函数名称	rtc_alarm_subsecond_get
函数原型	uint32_t rtc_alarm_subsecond_get(uint8_t rtc_alarm);
功能描述	获取RTC闹钟亚秒值
先决条件	-
被调用函数	-
输入参数{in}	
<b>rtc_alarm</b>	闹钟选择
<i>RTC_ALARM0</i>	闹钟0
<i>RTC_ALARM1</i>	闹钟1
输出参数{out}	
-	-
返回值	
<b>uint32_t</b>	RTC 闹钟亚秒值(0x0-0x3FFF)

例如：

```
/*get RTC alarm0 subsecond*/
```

```
uint32_t subsecond = rtc_alarm_subsecond_get(RTC_ALARM0);
```

### 函数 **rtc\_alarm\_enable**

函数rtc\_alarm\_enable描述见下表：

**表 3-536. 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}	
<b>rtc_alarm</b>	闹钟选择
<i>RTC_ALARM0</i>	闹钟0
<i>RTC_ALARM1</i>	闹钟1
输出参数{out}	
-	-
返回值	
-	-

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 3-537. 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}	
<b>rtc_alarm</b>	闹钟选择
<i>RTC_ALARM0</i>	闹钟0
<i>RTC_ALARM1</i>	闹钟1
输出参数{out}	
-	-
返回值	
<b>ErrStatus</b>	ERROR或SUCCESS

例如：

```
/*disable RTC alarm1*/
```

```
ErrStatus error_status;
```

```
error_status = rtc_alarm_disable(RTC_ALARM0);
```

## 函数 rtc\_timestamp\_enable

函数can\_init描述见下表:

表 3-538. 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 3-539. 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 3-540. 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3-523. 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 3-541. 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 3-542. 函数 rtc\_timestamp\_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3-524. 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 3-543. 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 3-544. 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 3-545. 函数 rtc\_tamper\_without\_bkp\_seset**

函数名称	rtc_tamper_without_bkp_seset
函数原型	void rtc_tamper_without_bkp_seset(uint32_t ne_source);
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}	
ne_source	配置侵入事件源触发
RTC_TAMPXNOER_NONE	tamper0和tamper 1不会擦除bkp寄存器
RTC_TAMPXNOER_TP0	tamper0不会擦除bkp寄存器
RTC_TAMPXNOER_TP1	tamper1不会擦除bkp寄存器
RTC_TAMPXNOER_T	tamper0和tamper 1会擦除bkp寄存器

P0_TP1	
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\_output\_pin\_select

函数rtc\_output\_pin\_select描述见下表:

表 3-546. 函数 rtc\_output\_pad\_select

函数名称	rtc_output_pin_select
函数原型	void rtc_output_pin_select(uint32_t pin);
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}	
pad	指定RTC输出引脚
RTC_OUT_PC15	RTC输出引脚为PC15
RTC_OUT_PA3_PA8	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 3-547. 函数 rtc\_alarm\_output\_config

函数名称	rtc_alarm_output_config
函数原型	void rtc_alarm_output_config(uint32_t source, uint32_t mode);
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 3-548. 函数 rtc\_calibration\_output\_config

<b>函数名称</b>	rtc_calibration_output_config
<b>函数原型</b>	void rtc_calibration_output_config(uint32_t source);
<b>功能描述</b>	配置RTC校准输出源
<b>先决条件</b>	-
<b>被调用函数</b>	-
输入参数{in}	
<b>source</b>	指定输出信号
<i>RTC_CALIBRATION_512HZ</i>	当外部低速时钟频率为32768Hz并且RTC_PSC为默认值，输出512Hz信号
<i>RTC_CALIBRATION_1HZ</i>	当外部低速时钟频率为32768Hz并且RRTC_PSC为默认值，输出1Hz信号
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 3-549. 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 3-550. 函数 `rtc_second_adjust`**

函数名称	<code>rtc_second_adjust</code>
函数原型	<code>ErrStatus rtc_second_adjust(uint32_t add, uint32_t minus);</code>
功能描述	调整RTC当前时间的秒或亚秒值
先决条件	-
被调用函数	-
输入参数{in}	
<b>add</b>	在当前时间上增加1S或者不增加
<code>RTC_SHIFT_ADD1S_R</code> <code>ESET</code>	无影响
<code>RTC_SHIFT_ADD1S_S</code>	在当前时间增加1秒

<i>ET</i>	
输入参数{in}	
<b>minus</b>	在当前是时间上减少的亚秒值(0x0 - 0x7FFF)
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* adjust RTC second or subsecond value of current time */
```

```
ErrStatus error_status;
```

```
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 3-551. 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 3-552. 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 3-553. 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	
ErrStatus	ERROR 或 SUCCESS

例如：

```
/* enable RTC reference clock detection function*/
```

```
ErrStatus error_status;
```

```
error_status = rtc_refclock_detection_enable();
```

### 函数 rtc\_refclock\_detection\_disable

函数rtc\_refclock\_detection\_disable描述见下表：

表 3-554. 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	
ErrStatus	ERROR或SUCCESS

例如：

```
/* disable RTC reference clock detection function*/
```

```
ErrStatus error_status;
```

```
error_status = rtc_refclock_detection_disable ();
```

### 函数 rtc\_wakeup\_enable

函数rtc\_wakeup\_enable描述见下表：

表 3-555. 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 3-556. 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;
```

```
error_status = rtc_wakeup_disable( );
```

### 函数 rtc\_wakeup\_clock\_set

函数rtc\_wakeup\_clock\_set描述见下表：

表 3-557. 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;
```

```
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 3-558. 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;
```

```
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 3-559. 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 3-560. 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}	
<b>window</b>	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}	
<b>plus</b>	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}	
<b>minus</b>	校准窗口校准周期RTCCLK脉冲屏蔽数（0x0-0x1FF）
输出参数{out}	
-	-
返回值	
<b>ErrStatus</b>	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 3-561. 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;
```

```
error_status = rtc_coarse_calibration_enable ( );
```

## 函数 rtc\_coarse\_calibration\_disable

函数rtc\_coarse\_calibration\_disable描述见下表：

**表 3-562. 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;
```

```
error_status = rtc_coarse_calibration_disable ( );
```

### 函数 rtc\_coarse\_calibration\_config

函数rtc\_coarse\_calibration\_config描述见下表:

**表 3-563. 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;
```

```
error_status = rtc_coarse_calibration_config (CALIB_INCREASE, 0x01);
```

### 函数 rtc\_flag\_get

函数rtc\_flag\_get描述见下表:

表 3-564. 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	-
输入参数{in}	
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
输出参数{out}	
-	-
返回值	
FlagStatus	SET 或 RESET

例如:

```
/* check time-stamp event flag */
```

```
FlagStatus flag_status;
```

```
flag_status = rtc_flag_get(RTC_FLAG_TS);
```

### 函数 rtc\_flag\_clear

函数rtc\_flag\_clear描述见下表:

表 3-565. 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}	
flag	要清除的中断标志位
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_WT	唤醒定时器可写标志
RTC_FLAG_ALARM0	闹钟0发生标志
RTC_FLAG_ALARM1	闹钟1发生标志
RTC_FLAG_RSYN	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\_interrupt\_enable

函数rtc\_interrupt\_enable描述见下表:

表 3-566. 函数 rtc\_interrupt\_enable

函数名称	rtc_interrupt_enable
函数原型	void rtc_interrupt_enable(uint32_t interrupt);
功能描述	使能RTC指定的中断
先决条件	-
被调用函数	-
输入参数{in}	
interrupt	选定被使能的中断源
RTC_INT_TIMESTAMP	时间戳中断
RTC_INT_ALARM0	闹钟0中断
RTC_INT_ALARM1	闹钟1中断
RTC_INT_WAKEUP	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 3-567. 函数 `rtc_interrupt_disable`**

函数名称	<code>rtc_interrupt_disable</code>
函数原型	<code>void rtc_interrupt_disable(uint32_t interrupt);</code>
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}	
<code>interrupt</code>	选定被失能的RTC中断
<code>RTC_INT_TIMESTAMP</code>	时间戳中断
<code>RTC_INT_ALARM0</code>	闹钟0中断
<code>RTC_INT_ALARM1</code>	闹钟1中断
<code>RTC_INT_WAKEUP</code>	唤醒定时器中断
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* disable specified RTC interrupt */
```

```
rtc_interrupt_disable(RTC_INT_ALARM0);
```

## 3.20. SPI

SPI模块可以通过SPI协议与外部设备进行通信。章节 [3.20.1](#) 描述了SPI的寄存器列表，章节 [3.20.2](#) 对SPI库函数进行说明。

### 3.20.1. 外设寄存器说明

SPI寄存器列表如下表所示：

**表 3-568. SPI 寄存器**

寄存器名称	寄存器描述
<code>SPI_CTL0</code>	控制寄存器0
<code>SPI_CTL1</code>	控制寄存器1
<code>SPI_STAT</code>	状态寄存器
<code>SPI_DATA</code>	数据寄存器
<code>SPI_CRCPOLY</code>	CRC多项式寄存器

寄存器名称	寄存器描述
SPI_RCR	接收CRC寄存器
SPI_TCR	发送CRC寄存器

### 3.20.2. 外设库函数说明

SPI库函数列表如下表所示：

**表 3-569. SPI 库函数**

库函数名称	库函数描述
spi_deinit	复位外设SPI
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SPI
spi_enable	使能外设SPI
spi_disable	失能外设SPI
spi_nss_output_enable	使能外设SPI NSS输出
spi_nss_output_disable	失能外设SPI 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SPI的DMA功能
spi_dma_disable	失能外设SPI的DMA功能
spi_data_frame_format_config	配置外设SPI数据帧格式
spi_data_transmit	发送数据
spi_data_receive	接收数据
spi_bidirectional_transfer_config	配置外设SPI的数据传输方向
spi_format_error_clear	清除SPI TI模式格式错误标志状态
spi_crc_polynomial_set	设置外设SPI的CRC多项式值
spi_crc_polynomial_get	获取外设SPI的CRC多项式值
spi_crc_on	打开外设SPI的CRC功能
spi_crc_off	关闭外设SPI的CRC功能
spi_crc_next	设置外设SPI下一次传输数据为CRC值
spi_crc_get	外设SPI获取CRC值
spi_crc_error_clear	清除SPI CRC错误标志状态
spi_ti_mode_enable	使能SPI TI模式
spi_ti_mode_disable	禁能SPI TI模式
spi_interrupt_enable	使能外设SPI中断
spi_interrupt_disable	失能外设SPI中断
spi_interrupt_flag_get	获取外设SPI中断状态
spi_flag_get	获取外设SPI标志状态

## 结构体 spi\_parameter\_struct

表 3-570. 结构体 spi\_parameter\_struct

成员名称	功能描述
device_mode	主机或设备模式配置 (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_xBIT, x=8/16)
nss	NSS由软件或硬件控制配置 (SPI_NSS_SOFT, SPI_NSS_HARD)
endian	大端或小端模式配置 (SPI_ENDIAN_MSB, SPI_ENDIAN_LSB)
clock_polarity_phase	相位和极性配置 (SPI_CK_PL_LOW_PH_1EDGE, SPI_CK_PL_HIGH_PH_1EDGE, SPI_CK_PL_LOW_PH_2EDGE, SPI_CK_PL_HIGH_PH_2EDGE)
prescale	预分频器配置 (SPI_PSC_n (n=2,4,8,16,32,64,128,256))

## 函数 spi\_deinit

函数spi\_deinit描述见下表:

表 3-571. 函数 spi\_deinit

函数名称	spi_deinit
函数原形	void spi_deinit(void);
功能描述	复位外设SPI
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 SPI */
spi_deinit();
```

## 函数 spi\_struct\_para\_init

函数spi\_struct\_para\_init描述见下表：

表 3-572. 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	SPI初始化结构体，结构体成员参考 <a href="#">表 3-570. 结构体spi_parameter_struct</a> 。
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 3-573. 函数 spi\_init

函数名称	spi_init
函数原形	void spi_init( spi_parameter_struct* spi_struct);
功能描述	初始化外设SPI
先决条件	-
被调用函数	-
输入参数{in}	
spi_struct	初始化结构体，结构体成员参考 <a href="#">表 3-570. 结构体spi_parameter_struct</a>
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* initialize SPI */
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_CLOCK_POLARITY_HIGH_PHASE_2EDGE;

spi_init_struct.nss             = SPI_NSS_SOFT;

spi_init_struct.prescale        = SPI_PSC_8;

spi_init_struct.endian          = SPI_ENDIAN_MSB;

spi_init(&spi_init_struct);

```

### 函数 spi\_enable

函数spi\_enable描述见下表：

表 3-574. 函数 spi\_enable

函数名称	spi_enable
函数原形	void spi_enable(void);
功能描述	使能外设SPI
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 SPI */
```

```
spi_enable();
```

### 函数 spi\_disable

函数spi\_disable描述见下表：

表 3-575. 函数 spi\_disable

函数名称	spi_disable
函数原形	void spi_disable(void);
功能描述	失能外设SPI
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 SPI */
```

```
spi_disable();
```

### 函数 spi\_nss\_output\_enable

函数spi\_nss\_output\_enable描述见下表：

表 3-576. 函数 spi\_nss\_output\_enable

函数名称	spi_nss_output_enable
函数原形	void spi_nss_output_enable(void);
功能描述	使能外设SPI NSS输出
先决条件	-
被调用函数	-
输入参数{in}	
-	-
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* enable SPI NSS output */
```

```
spi_nss_output_enable();
```

### 函数 spi\_nss\_output\_disable

函数spi\_nss\_output\_disable描述见下表：

表 3-577. 函数 spi\_nss\_output\_disable

函数名称	spi_nss_output_disable
函数原形	void spi_nss_output_disable(void);
功能描述	失能外设SPI NSS输出
先决条件	-
被调用函数	-
输入参数{in}	
-	-

输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* disable SPI NSS output */
```

```
spi_nss_output_disable();
```

### 函数 spi\_nss\_internal\_high

函数spi\_nss\_internal\_high描述见下表:

表 3-578. 函数 spi\_nss\_internal\_high

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

例如:

```
/* SPI NSS pin is pulled high level in software mode */
```

```
spi_nss_internal_high();
```

### 函数 spi\_nss\_internal\_low

函数spi\_nss\_internal\_low描述见下表:

表 3-579. 函数 spi\_nss\_internal\_low

函数名称	spi_nss_internal_low
函数原形	void spi_nss_internal_low(void);
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}	
-	-
输出参数{out}	

-	-
返回值	
-	-

例如：

```
/* SPI NSS pin is pulled low level in software mode */
```

```
spi_nss_internal_low();
```

### 函数 spi\_dma\_enable

函数spi\_dma\_enable描述见下表：

表 3-580. 函数 spi\_dma\_enable

函数名称	spi_dma_enable
函数原形	void spi_dma_enable( uint8_t dma);
功能描述	使能外设SPI的DMA功能
先决条件	-
被调用函数	-
输入参数{in}	
dma	SPI DMA模式
SPI_DMA_TRANSMIT	SPI发送缓冲区DMA使能
SPI_DMA_RECEIVE	SPI接收缓冲区DMA使能
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* enable SPI transmit data DMA function */
```

```
spi_dma_enable(SPI_DMA_TRANSMIT);
```

### 函数 spi\_dma\_disable

函数spi\_dma\_disable描述见下表：

表 3-581. 函数 spi\_dma\_disable

函数名称	spi_dma_disable
函数原形	void spi_dma_disable( uint8_t dma);
功能描述	失能外设SPI的DMA功能
先决条件	-
被调用函数	-



输入参数{in}	
<b>dma</b>	SPI DMA模式
<i>SPI_DMA_TRANSMIT</i>	SPI发送缓冲区DMA禁能
<i>SPI_DMA_RECEIVE</i>	SPI接收缓冲区DMA禁能
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* disable SPI transmit data DMA function */
```

```
spi_dma_disable(SPI_DMA_TRANSMIT);
```

### 函数 spi\_data\_frame\_format\_config

函数spi\_data\_frame\_format\_config描述见下表：

表 3-582. 函数 spi\_data\_frame\_format\_config

函数名称	spi_data_frame_format_config
函数原形	void spi_data_frame_format_config( uint16_t frame_format);
功能描述	配置外设SPI数据帧格式
先决条件	-
被调用函数	-
输入参数{in}	
<b>frame_format</b>	SPI帧大小
<i>SPI_FRAME_SIZE_x</i> <i>BIT</i>	SPI x位数据帧格式，x=8/16
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* configure SPI data frame format size is 16 bits */
```

```
spi_data_frame_format_config(SPI_FRAME_SIZE_16BIT);
```

### 函数 spi\_data\_transmit

函数spi\_data\_transmit描述见下表：

表 3-583. 函数 spi\_data\_transmit

函数名称	spi_data_transmit
函数原形	void spi_data_transmit( 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}	
data	16位数据
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* SPI transmit data */
spi_data_transmit(spi_send_array[send_n]);
```

### 函数 spi\_data\_receive

函数spi\_data\_receive描述见下表:

表 3-584. 函数 spi\_data\_receive

函数名称	spi_data_receive
函数原形	uint16_t spi_data_receive(void);
功能描述	SPI接收数据
先决条件	-
被调用函数	-
输入参数{in}	
-	-
输出参数{out}	
-	-
返回值	
uint16_t	16位数据

例如:

```
/* SPI receive data */
spi_receive_array[receive_n] = spi_data_receive();
```

### 函数 spi\_bidirectional\_transfer\_config

函数spi\_bidirectional\_transfer\_config描述见下表:

表 3-585. 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 transfer_direction);</code>
功能描述	配置外设SPI的数据传输方向
先决条件	-
被调用函数	-
输入参数{in}	
<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：

```
/* SPI works in transmit-only mode */
```

```
spi_bidirectional_transfer_config(SPI_BIDIRECTIONAL_TRANSMIT);
```

### 函数 `spi_format_error_clear`

函数`spi_format_error_clear`描述见下表：

表 3-586. 函数 `spi_format_error_clear`

函数名称	<code>spi_format_error_clear</code>
函数原形	<code>void spi_format_error_clear(void);</code>
功能描述	清除SPI TI模式格式错误标志状态
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 TI mode format error flag status */
```

```
spi_format_error_clear();
```

## 函数 spi\_crc\_polynomial\_set

函数spi\_crc\_polynomial\_set描述见下表：

表 3-587. 函数 spi\_crc\_polynomial\_set

函数名称	spi_crc_polynomial_set
函数原形	void spi_crc_polynomial_set( uint16_t crc_poly);
功能描述	设置外设SPI的CRC多项式值
先决条件	-
被调用函数	-
输入参数{in}	
crc_poly	CRC多项式值
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* set SPI CRC polynomial */
spi_crc_polynomial_set(CRC_VALUE);
```

## 函数 spi\_crc\_polynomial\_get

函数spi\_crc\_polynomial\_get描述见下表：

表 3-588. 函数 spi\_crc\_polynomial\_get

函数名称	spi_crc_polynomial_get
函数原形	uint16_t spi_crc_polynomial_get(void);
功能描述	获取外设SPI的CRC多项式值
先决条件	-
被调用函数	-
输入参数{in}	
-	-
输出参数{out}	
-	-
返回值	
uint16_t	16位CRC多项式值（0-0xFFFF）

例如：

```
/* get SPI CRC polynomial */
uint16_t crc_val;
crc_val = spi_crc_polynomial_get();
```

## 函数 spi\_crc\_on

函数spi\_crc\_on描述见下表：

表 3-589. 函数 spi\_crc\_on

函数名称	spi_crc_on
函数原形	void spi_crc_on(void);
功能描述	打开外设SPI的CRC功能
先决条件	-
被调用函数	-
输入参数{in}	
-	-
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* turn on SPI CRC function */
```

```
spi_crc_on();
```

## 函数 spi\_crc\_off

函数spi\_crc\_off描述见下表：

表 3-590. 函数 spi\_crc\_off

函数名称	spi_crc_off
函数原形	void spi_crc_off(void);
功能描述	关闭外设SPI的CRC功能
先决条件	-
被调用函数	-
输入参数{in}	
-	-
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* turn off SPI CRC function */
```

```
spi_crc_off();
```

## 函数 spi\_crc\_next

函数spi\_crc\_next描述见下表：

表 3-591. 函数 spi\_crc\_next

函数名称	spi_crc_next
函数原形	void spi_crc_next(void);
功能描述	设置外设SPI下一次传输数据为CRC值
先决条件	-
被调用函数	-
输入参数{in}	
-	-
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* SPI next data is CRC value */
```

```
spi_crc_next();
```

## 函数 spi\_crc\_get

函数spi\_crc\_get描述见下表：

表 3-592. 函数 spi\_crc\_get

函数名称	spi_crc_get
函数原形	uint16_t spi_crc_get( uint8_t crc);
功能描述	外设SPI获取CRC值
先决条件	-
被调用函数	-
输入参数{in}	
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 SPI CRC send value */
```

```
uint16_t crc_val;
```

```
crc_val = spi_crc_get(SPI_CRC_TX);
```

### 函数 spi\_crc\_error\_clear

函数spi\_crc\_error\_clear描述见下表：

**表 3-593. 函数 spi\_crc\_error\_clear**

函数名称	spi_crc_error_clear
函数原形	void spi_crc_error_clear(void);
功能描述	清除SPI CRC错误标志状态
先决条件	-
被调用函数	-
输入参数{in}	
-	-
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* clear SPI CRC error flag status */
spi_crc_error_clear();
```

### 函数 spi\_ti\_mode\_enable

函数spi\_ti\_mode\_enable描述见下表：

**表 3-594. 函数 spi\_ti\_mode\_enable**

函数名称	spi_ti_mode_enable
函数原形	void spi_ti_mode_enable(void);
功能描述	使能SPI TI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 SPI TI mode */
spi_ti_mode_enable();
```

## 函数 spi\_ti\_mode\_disable

函数spi\_ti\_mode\_disable描述见下表：

表 3-595. 函数 spi\_ti\_mode\_disable

函数名称	spi_ti_mode_disable
函数原形	void spi_ti_mode_disable(void);
功能描述	失能SPI TI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 SPI TI mode */
```

```
spi_ti_mode_disable();
```

## 函数 spi\_interrupt\_enable

函数spi\_interrupt\_enable描述见下表：

表 3-596. 函数 spi\_interrupt\_enable

函数名称	spi_interrupt_enable
函数原形	void spi_interrupt_enable( uint8_t interrupt);
功能描述	使能外设SPI中断
先决条件	-
被调用函数	-
输入参数{in}	
interrupt	SPI中断
SPI_INT_TBE	发送缓冲区空中断使能
SPI_INT_RBNE	接收缓冲区非空中断使能
SPI_INT_ERR	错误中断使能
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* enable SPI transmit buffer empty interrupt */
```



```
spi_interrupt_enable(SPI_INT_TBE);
```

### 函数 spi\_interrupt\_disable

函数spi\_interrupt\_disable描述见下表：

表 3-597. 函数 spi\_interrupt\_disable

函数名称	spi_interrupt_disable
函数原形	void spi_interrupt_disable( uint8_t interrupt);
功能描述	失能外设SPI中断
先决条件	-
被调用函数	-
输入参数{in}	
interrupt	SPI中断
SPI_INT_TBE	发送缓冲区空中断使能
SPI_INT_RBNE	接收缓冲区非空中断使能
SPI_INT_ERR	错误中断使能
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* disable SPI transmit buffer empty interrupt */
```

```
spi_interrupt_disable(SPI_INT_TBE);
```

### 函数 spi\_interrupt\_flag\_get

函数spi\_interrupt\_flag\_get描述见下表：

表 3-598. 函数 spi\_interrupt\_flag\_get

函数名称	spi_interrupt_flag_get
函数原形	FlagStatus spi_interrupt_flag_get( uint8_t interrupt);
功能描述	获取外设SPI中断状态
先决条件	-
被调用函数	-
输入参数{in}	
interrupt	SPI中断状态
SPI_INT_FLAG_TB E	发送缓冲区空中断
SPI_INT_FLAG_RB NE	接收缓冲区非空中断
SPI_INT_FLAG_RX ORERR	接收过载错误中断

<i>SPI_INT_FLAG_CO NFERR</i>	配置错误中断
<i>SPI_INT_FLAG_CR CERR</i>	CRC错误中断
<i>SPI_INT_FLAG_FE RR</i>	格式错误中断
输出参数{out}	
-	-
返回值	
<b>FlagStatus</b>	SET 或 RESET

例如:

```
/* get SPI transmit buffer empty interrupt status */
if(RESET != spi_interrupt_flag_get(SPI_INT_FLAG_TBE)){
    while(RESET == spi_flag_get(SPI_FLAG_TBE));
    spi_data_transmit(spi_send_array[send_n++]);
}
```

## 函数 spi\_flag\_get

函数spi\_flag\_get描述见下表:

表 3-599. 函数 spi\_flag\_get

函数名称	spi_flag_get
函数原形	FlagStatus spi_flag_get( uint32_t flag);
功能描述	获取外设SPI标志状态
先决条件	-
被调用函数	-
输入参数{in}	
<b>flag</b>	SPI标志状态
<i>SPI_FLAG_TBE</i>	SPI发送缓冲区空标志
<i>SPI_FLAG_RBNE</i>	SPI接收缓冲区非空标志
<i>SPI_FLAG_TRANS</i>	SPI通信进行中标志
<i>SPI_FLAG_RXORE RR</i>	SPI接收过载错误标志
<i>SPI_FLAG_CONFE RR</i>	SPI配置错误标志
<i>SPI_FLAG_CRCER R</i>	SPI CRC错误标志
<i>SPI_FLAG_FERR</i>	SPI格式错误标志
输出参数{out}	
-	-

返回值	
FlagStatus	SET 或 RESET

例如：

```
/* get SPI transmit buffer empty flag status */
while(RESET == spi_flag_get(SPI_FLAG_TBE));
spi_data_transmit(spi_send_array[send_n++]);
```

## 3.21. SYSCFG

章节[3.21.1](#)描述了SYSCFG的寄存器列表，章节[3.21.2](#)对SYSCFG库函数进行说明。

### 3.21.1. 外设寄存器说明

SYSCFG寄存器列表如下表所示：

表 3-600. SYSCFG 寄存器

寄存器名称	寄存器描述
SYSCFG_CFG0	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_CFG1	系统配置寄存器1
SYSCFG_SCFG	SYSCFG共享SRAM配置寄存器
SYSCFG_TIMERxCFG	TIMER触发选择寄存器（x = 0..2）

### 3.21.2. 外设库函数说明

SYSCFG库函数列表如下表所示：

表 3-601. SYSCFG 库函数

库函数名称	库函数描述
syscfg_deinit	复位SYSCFG寄存器
syscfg_exti_line_config	配置GPIO引脚作为EXTI
syscfg_io_compensation_enable	I/O补偿单元使能
syscfg_io_compensation_disable	I/O补偿单元掉电
syscfg_lock_config	将TIMER 0/15/16中断输入连接到所选参数
syscfg_io_compensation_ready_flag_get	获取I/O补偿单元准备就绪标志
syscfg_sram_ownership_config	配置共享SRAM所有权

库函数名称	库函数描述
syscfg_boot_mode_get	获取代码启动模式

### 枚举类型 **syscfg\_code\_start\_enum**

表 3-602. 枚举类型 **syscfg\_code\_start\_enum**

成员名称	功能描述
FLASH_START	flash启动模式
SRAM_START	sram启动模式
SYSTEM_START	系统启动模式

### 函数 **syscfg\_deinit**

函数syscfg\_deinit描述见下表：

表 3-603. 函数 **syscfg\_deinit**

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

例如：

```
/* reset SYSCFG registers */
```

```
syscfg_deinit();
```

### 函数 **syscfg\_exti\_line\_config**

函数syscfg\_exti\_line\_config描述见下表：

表 3-604. 函数 **syscfg\_exti\_line\_config**

函数名称	syscfg_exti_line_config
函数原形	void syscfg_exti_line_config(uint8_t exti_port, uint8_t exti_pin);
功能描述	配置GPIO引脚作为EXTI
先决条件	-
被调用函数	-
输入参数{in}	
exti_port	指定EXTI使用的GPIO端口

<i>EXTI_SOURCE_GPIOx</i>	x=A,B,C
输入参数{in}	
<b>exti_pin</b>	EXTI引脚
<i>EXTI_SOURCE_PINx</i>	x=0..15(GPIOA), x=0..4,11,12,13,15(GPIOB) , x=8,13,14,15 (GPIOC)
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_PIN0);
```

### 函数 syscfg\_io\_compensation\_enable

函数syscfg\_io\_compensation\_enable描述见下表:

**表 3-605. 函数 syscfg\_io\_compensation\_enable**

函数名称	syscfg_io_compensation_enable
函数原形	void syscfg_io_compensation_enable(void);
功能描述	I/O补偿单元使能
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 I/O compensation cell */
```

```
syscfg_io_compensation_enable();
```

### 函数 syscfg\_io\_compensation\_disable

函数syscfg\_io\_compensation\_disable描述见下表:

**表 3-606. 函数 syscfg\_io\_compensation\_disable**

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

功能描述	I/O补偿单元掉电
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 I/O compensation cell */
syscfg_io_compensation_disable();
```

### 函数 syscfg\_lock\_config

函数syscfg\_lock\_config描述见下表：

表 3-607. 函数 syscfg\_lock\_config

函数名称	syscfg_lock_config
函数原形	void syscfg_lock_config(void);
功能描述	将TIMER 0/15/16中断输入连接到所选参数
先决条件	-
被调用函数	-
输入参数{in}	
-	-
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* connect TIMER0/15/16 break input to the selected parameter */
syscfg_lock_config();
```

### 函数 syscfg\_io\_compensation\_ready\_flag\_get

函数syscfg\_io\_compensation\_ready\_flag\_get描述见下表：

表 3-608. 函数 syscfg\_io\_compensation\_ready\_flag\_get

函数名称	syscfg_io_compensation_ready_flag_get
函数原形	FlagStatus syscfg_io_compensation_ready_flag_get(void);
功能描述	获取I/O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 / SET

例如：

```
/* get the compensation ready flag */
while(RESET != syscfg_io_compensation_ready_flag_get()){
}
```

### 函数 syscfg\_sram\_ownership\_config

函数syscfg\_sram\_ownership\_config描述见下表：

表 3-609. 函数 syscfg\_sram\_ownership\_config

函数名称	syscfg_sram_ownership_config
函数原形	void syscfg_sram_ownership_config(uint32_t sram_owner);
功能描述	配置共享SRAM所有权
先决条件	-
被调用函数	-
输入参数{in}	
sram_owner	共享SRAM所有权
SRAMEN_WIRELESS	wireless
SRAMEN_CORE	core
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* configure the ownership */
syscfg_sram_ownership_config(SRAMEN_WIRELESS);
```

### 函数 syscfg\_boot\_mode\_get

函数syscfg\_boot\_mode\_get描述见下表：

表 3-610. 函数 syscfg\_boot\_mode\_get

函数名称	syscfg_boot_mode_get
函数原形	syscfg_code_start_enum syscfg_boot_mode_get(void);
功能描述	获取代码启动模式
先决条件	-
被调用函数	-
输入参数{in}	
-	-
输出参数{out}	
-	-
返回值	
syscfg_code_start_enum	代码启动模式
FLASH_START	flash启动模式
SRAM_START	sram启动模式
SYSTEM_START	system启动模式

例如:

```
/* get code start mode */
if(FLASH_START==syscfg_boot_mode_get()){
}
```

## 3.22. 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)，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[3.22.1](#)描述了TIMER的寄存器列表，章节[3.22.2](#)对TIMER库函数进行说明。

### 3.22.1. 外设寄存器说明

TIMER寄存器列表如下表所示:

表 3-611. TIMER 寄存器

寄存器名称	寄存器描述
TIMER_CTL0	控制寄存器0
TIMER_CTL1	控制寄存器1
TIMER_SMCFG	从模式配置寄存器
TIMER_DMAINTEN	DMA和中断使能寄存器
TIMER_INTF	中断标志寄存器
TIMER_SWEVG	软件事件产生寄存器



寄存器名称	寄存器描述
TIMER_CHCTL0	通道控制寄存器0
TIMER_CHCTL1	通道控制寄存器1
TIMER_CHCTL2	通道控制寄存器2
TIMER_CNT	计数器寄存器
TIMER_PSC	预分频寄存器
TIMER_CAR	计数器自动重载寄存器
TIMER_CREP	重复计数寄存器
TIMER_CH0CV	通道0捕获/比较寄存器
TIMER_CH1CV	通道1捕获/比较寄存器
TIMER_CH2CV	通道2捕获/比较寄存器
TIMER_CH3CV	通道3捕获/比较寄存器
TIMER_CCHP	互补通道保护寄存器
TIMER_DMACFG	DMA配置寄存器
TIMER_DMATB	DMA发送缓冲区寄存器
TIMER_IRMP	TIMER通道输入映射寄存器
TIMER_CFG	配置寄存器

### 3.22.2. 外设库函数说明

TIMER库函数列表如下表所示：

**表 3-612. TIMER 库函数**

库函数名称	库函数描述
timer_deinit	复位外设TIMERx
timer_struct_para_init	将TIMER初始化结构体中所有参数初始化为默认值
timer_init	初始化外设TIMERx
timer_enable	使能外设TIMERx
timer_disable	禁能外设TIMERx
timer_auto_reload_shadow_enable	TIMERx自动重载影子使能
timer_auto_reload_shadow_disable	TIMERx自动重载影子禁能
timer_update_event_enable	TIMERx更新使能
timer_update_event_disable	TIMERx更新禁能
timer_counter_alignment	设置外设TIMERx的对齐模式
timer_counter_up_direction	设置外设TIMERx向上计数
timer_counter_down_direction	设置外设TIMERx向下计数
timer_prescaler_config	配置外设TIMERx预分频器
timer_repetition_value_config	配置外设TIMERx的重复计数器
timer_autoreload_value_config	配置外设TIMERx的自动重载寄存器
timer_counter_value_config	配置外设TIMERx的计数器值
timer_counter_read	读取外设TIMERx的计数器值
timer_prescaler_read	读取外设TIMERx的预分频器值
timer_single_pulse_mode_config	配置外设TIMERx的单脉冲模式

库函数名称	库函数描述
timer_update_source_config	配置外设TIMERx的更新源
timer_dma_enable	外设TIMERx的DMA使能
timer_dma_disable	外设TIMERx的DMA禁能
timer_channel_dma_request_source_select	外设TIMERx的通道DMA请求源选择
timer_dma_transfer_config	配置外设TIMERx的DMA模式
timer_event_software_generate	软件产生事件
timer_break_struct_para_init	将TIMER中止功能参数结构体中所有参数初始化为默认值
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
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	读取通道输入捕获值

库函数名称	库函数描述
register_read	
timer_input_pwm_capture_config	配置TIMERx捕获PWM输入参数
timer_hall_mode_config	配置TIMERx的HALL接口功能
timer_channel_input_remap_config	配置TIMERx通道输入映射
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
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 3-613. 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）

成员名称	功能描述
repetitioncounter	重复计数器值（0~255）

### 结构体 timer\_break\_parameter\_struct

表 3-614. 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）
idleoffstate	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 3-615. 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_CCN_ENABLE, TIMER_CCN_DISABLE）
ocpolarity	通道输出极性（TIMER_OC_POLARITY_HIGH, TIMER_OC_POLARITY_LOW）
ocnpolarity	互补通道输出极性（TIMER_OCN_POLARITY_HIGH, TIMER_OCN_POLARITY_LOW）
ocidlestate	空闲状态下通道输出（TIMER_OC_IDLE_STATE_LOW, TIMER_OC_IDLE_STATE_HIGH）
ocnidlestate	空闲状态下互补通道输出（TIMER_OCN_IDLE_STATE_LOW, TIMER_OCN_IDLE_STATE_HIGH）

### 结构体 timer\_ic\_parameter\_struct

表 3-616. 结构体 timer\_ic\_parameter\_struct

成员名称	功能描述
icpolarity	通道输入极性（TIMER_IC_POLARITY_RISING, TIMER_IC_POLARITY_FALLING, TIMER_IC_POLARITY_BOTH_EDGE）
icselection	通道输入模式选择（TIMER_IC_SELECTION_DIRECTTI, TIMER_IC_SELECTION_INDIRECTTI, TIMER_IC_SELECTION_ITS）
icprescaler	通道输入捕获预分频（TIMER_IC_PSC_DIV1, TIMER_IC_PSC_DIV2, ...）

成员名称	功能描述
	TIMER_IC_PSC_DIV4, TIMER_IC_PSC_DIV8)
icfilter	通道输入捕获滤波 (0~15)

## 函数 timer\_deinit

函数timer\_deinit描述见下表:

表 3-617. 函数 timer\_deinit

函数名称	timer_deinit
函数原型	void timer_deinit(uint32_t timer_periph);
功能描述	复位外设TIMER
先决条件	-
被调用函数	rcu_periph_reset_enable / rcu_periph_reset_disable
输入参数{in}	
timer_periph	TIMER外设
TIMERx(x=0,1,2,5,15,16)	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 3-618. 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3-613. 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 3-619. 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=0,1,2,5,15,16)	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3-613. 结构体 timer_parameter_struct</a> 。
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* initialize TIMER0 */
```

```
timer_parameter_struct timer_initpara;
```

```
timer_initpara.prescaler = 107;
```

```
timer_initpara.alignedmode = TIMER_COUNTER_EDGE;
```

```
timer_initpara.counterdirection = TIMER_COUNTER_UP;
```

```
timer_initpara.period = 999;
```

```
timer_initpara.clockdivision = TIMER_CKDIV_DIV1;
```

```
timer_initpara.repetitioncounter = 1;
```

```
timer_init(TIMER0, &timer_initpara);
```

## 函数 timer\_enable

函数timer\_enable描述见下表:

表 3-620. 函数 timer\_enable

函数名称	timer_enable
函数原型	void timer_enable(uint32_t timer_periph);
功能描述	使能外设TIMER
先决条件	-
被调用函数	-
输入参数{in}	
timer_periph	TIMER外设
TIMEx(x=0,1,2,5,15,16)	TIMER外设选择
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* enable TIMER0 */
timer_enable(TIMER0);
```

## 函数 timer\_disable

函数timer\_disable描述见下表:

表 3-621. 函数 timer\_disable

函数名称	timer_disable
函数原型	void timer_disable(uint32_t timer_periph);
功能描述	禁能外设TIMER
先决条件	-
被调用函数	-
输入参数{in}	
timer_periph	TIMER外设
TIMEx(x=0,1,2,5,15,16)	TIMER外设选择
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* disable TIMER0 */
```

timer\_disable(TIMERO);

### 函数 timer\_auto\_reload\_shadow\_enable

函数timer\_auto\_reload\_shadow\_enable描述见下表:

表 3-622. 函数 timer\_auto\_reload\_shadow\_enable

函数名称	timer_auto_reload_shadow_enable
函数原型	void timer_auto_reload_shadow_enable(uint32_t timer_periph);
功能描述	TIMER自动重载影子使能
先决条件	-
被调用函数	-
输入参数{in}	
timer_periph	TIMER外设
TIMERx(x=0,1,2,5,15,16)	TIMER外设选择
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* enable the TIMERO auto reload shadow function */
```

```
timer_auto_reload_shadow_enable(TIMERO);
```

### 函数 timer\_auto\_reload\_shadow\_disable

函数timer\_auto\_reload\_shadow\_disable描述见下表:

表 3-623. 函数 timer\_auto\_reload\_shadow\_disable

函数名称	timer_auto_reload_shadow_disable
函数原型	void timer_auto_reload_shadow_disable (uint32_t timer_periph);
功能描述	TIMER自动重载影子禁能
先决条件	-
被调用函数	-
输入参数{in}	
timer_periph	TIMER外设
TIMERx(x=0,1,2,5,15,16)	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 3-624. 函数 timer\_update\_event\_enable**

函数名称	timer_update_event_enable
函数原型	void timer_update_event_enable(uint32_t timer_periph);
功能描述	TIMER更新使能
先决条件	-
被调用函数	-
输入参数{in}	
timer_periph	TIMER外设
TIMERx(x=0,1,2,5,15,16)	TIMER外设选择
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* enable TIMER0 the update event */
```

```
timer_update_event_enable(TIMER0);
```

### 函数 timer\_update\_event\_disable

函数timer\_update\_event\_disable描述见下表：

**表 3-625. 函数 timer\_update\_event\_disable**

函数名称	timer_update_event_disable
函数原型	void timer_update_event_disable (uint32_t timer_periph);
功能描述	TIMER更新禁能
先决条件	-
被调用函数	-
输入参数{in}	
timer_periph	TIMER外设
TIMERx(x=0,1,2,5,15,16)	TIMER外设选择
输出参数{out}	
-	-

返回值	
-	-

例如：

```
/* disable TIMER0 the update event */
```

```
timer_update_event_disable(TIMER0);
```

### 函数 timer\_counter\_alignment

函数timer\_counter\_alignment描述见下表：

**表 3-626. 函数 timer\_counter\_alignment**

函数名称	timer_counter_alignment
函数原型	void timer_counter_alignment(uint32_t timer_periph, uint16_t aligned);
功能描述	设置外设TIMER的对齐模式
先决条件	-
被调用函数	-
输入参数{in}	
timer_periph	TIMER外设
TIMERx(x=0,1,2)	TIMER外设选择
输入参数{in}	
aligned	对齐模式
TIMER_COUNTER_EDGE	无中央对齐计数模式(边沿对齐模式)，DIR位指定了计数方向
TIMER_COUNTER_CENTER_DOWN	中央对齐向下计数置1模式。计数器在中央计数模式计数，通道被配置在输出模式（TIMERx_CHCTL0寄存器中CHxMS=00），只有在向下计数时，通道的比较中断标志置1
TIMER_COUNTER_CENTER_UP	中央对齐向上计数置1模式。计数器在中央计数模式计数，通道被配置在输出模式（TIMERx_CHCTL0寄存器中CHxMS=00），只有在向上计数时，通道的比较中断标志置1
TIMER_COUNTER_CENTER_BOTH	中央对齐上下计数置1模式。计数器在中央计数模式计数，通道被配置在输出模式（TIMERx_CHCTL0寄存器中CHxMS=00），在向上和向下计数时，通道的比较中断标志都会置1
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* set TIMER0 counter center-aligned and counting up assert mode */
```

```
timer_counter_alignment(TIMER0, TIMER_COUNTER_CENTER_UP);
```

## 函数 timer\_counter\_up\_direction

函数timer\_counter\_up\_direction描述见下表：

表 3-627. 函数 timer\_counter\_up\_direction

函数名称	timer_counter_up_direction
函数原型	void timer_counter_up_direction(uint32_t timer_periph);
功能描述	设置外设TIMER向上计数
先决条件	计数器设置为无中央对齐计数模式（边沿对齐模式）
被调用函数	-
输入参数{in}	
timer_periph	TIMER外设
TIMERx(x=0,1,2)	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 3-628. 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=0,1,2)	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 3-629. 函数 timer\_prescaler\_config**

函数名称	timer_prescaler_config
函数原型	void timer_prescaler_config(uint32_t timer_periph, uint16_t prescaler, uint32_t pscreload);
功能描述	配置外设TIMER预分频器
先决条件	-
被调用函数	-
输入参数{in}	
timer_periph	TIMER外设
TIMERx(x=0,1,2,5,15,16)	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_RELOAD_NOW	预分频值立即加载
TIMER_PSC_RELOAD_UPDATE	预分频值在下次更新事件发生时加载
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* configure TIMER0 prescaler */
```

```
timer_prescaler_config(TIMER0, 3000, TIMER_PSC_RELOAD_NOW);
```

**函数 timer\_repetition\_value\_config**

函数timer\_repetition\_value\_config描述见下表:

**表 3-630. 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

<i>TIMERx</i> ( <i>x</i> =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 3-631. 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
<i>TIMERx</i> ( <i>x</i> =0,1,2,5,15,16)	TIMER外设选择
输入参数{in}	
autoreload	计数器自动重载值 0~65535( <i>TIMERx</i> ( <i>x</i> =0,5,15,16)),0~4294967295( <i>TIMERx</i> ( <i>x</i> =1,2))
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* configure TIMER autoreload register value */
```

```
timer_autoreload_value_config(TIMER0, 3000);
```

### 函数 timer\_counter\_value\_config

函数timer\_counter\_value\_config描述见下表：

表 3-632. 函数 timer\_counter\_value\_config

函数名称	timer_counter_value_config
函数原型	void timer_counter_value_config(uint32_t timer_periph, uint32_t counter);
功能描述	配置外设TIMER的计数器值
先决条件	-
被调用函数	-
输入参数{in}	
timer_periph	TIMER外设
TIMERx(x=0,1,2,5,15,16)	TIMER外设选择
输入参数{in}	
counter	计数器值0~65535(TIMERx(x=0,5,15,16)),0~4294967295(TIMERx(x=1,2))
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* configure TIMER0 counter register value */
```

```
timer_counter_value_config(TIMER0);
```

### 函数 timer\_counter\_read

函数timer\_counter\_read描述见下表:

表 3-633. 函数 timer\_counter\_read

函数名称	timer_counter_read
函数原型	uint32_t timer_counter_read(uint32_t timer_periph);
功能描述	读取外设TIMER的计数器值
先决条件	-
被调用函数	-
输入参数{in}	
timer_periph	TIMER外设
TIMERx(x=0,1,2,5,15,16)	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(TIMERO);
```

## 函数 timer\_prescaler\_read

函数timer\_prescaler\_read描述见下表:

**表 3-634. 函数 timer\_prescaler\_read**

函数名称	timer_prescaler_read
函数原型	uint16_t timer_prescaler_read(uint32_t timer_periph);
功能描述	读取外设TIMER的预分频器值
先决条件	-
被调用函数	-
输入参数{in}	
timer_periph	TIMER外设
TIMERx(x=0,1,2,5,15,16)	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 TIMERO prescaler value */
```

```
uint16_t i = 0;
```

```
i = timer_prescaler_read(TIMERO);
```

## 函数 timer\_single\_pulse\_mode\_config

函数timer\_single\_pulse\_mode\_config描述见下表:

**表 3-635. 函数 timer\_single\_pulse\_mode\_config**

函数名称	timer_single_pulse_mode_config
函数原型	void timer_single_pulse_mode_config(uint32_t timer_periph, uint32_t spmode);
功能描述	配置外设TIMER的单脉冲模式
先决条件	-
被调用函数	-
输入参数{in}	
timer_periph	TIMER外设
TIMERx(x=0,1,2,5,15,16)	TIMER外设选择
输入参数{in}	
spmode	脉冲模式
TIMER_SP_MODE_*	单脉冲模式计数

<i>SINGLE</i>	
<i>TIMER_SP_MODE_</i> <i>REPETITIVE</i>	重复模式计数
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* configure TIMER0 single pulse mode */
timer_single_pulse_mode_config(TIMER0, TIMER_SP_MODE_SINGLE);
```

### 函数 timer\_update\_source\_config

函数timer\_update\_source\_config描述见下表：

**表 3-636. 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=0,1,2,5,15,16)	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
TIMER_UPDATE_SRC_REGULAR	只有计数器溢出/ 下溢才产生更新中断或DMA请求
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* configure TIMER update only by counter overflow/underflow */
timer_update_source_config(TIMER0, TIMER_UPDATE_SRC_REGULAR);
```



## 函数 timer\_dma\_enable

函数timer\_dma\_enable描述见下表:

表 3-637. 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
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=0,1,2,5,15,16)
TIMER_DMA_CH0 D	通道0比较/捕获 DMA请求, TIMERx(x=0,1,2,15,16)
TIMER_DMA_CH1 D	通道1比较/捕获 DMA请求, TIMERx(x=0,1,2)
TIMER_DMA_CH2 D	通道2比较/捕获 DMA请求, TIMERx(x=0,1,2)
TIMER_DMA_CH3 D	通道3比较/捕获 DMA请求, TIMERx(x=0,1,2)
TIMER_DMA_CMT D	换相DMA更新请求, TIMERx(x=0)
TIMER_DMA_TRG D	触发DMA请求使能, TIMERx(x=0,1,2)
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 3-638. 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=0,1,2,5,15,16)
TIMER_DMA_CH0 D	通道0比较/捕获 DMA请求, TIMERx(x=0,1,2,15,16)
TIMER_DMA_CH1 D	通道1比较/捕获 DMA请求, TIMERx(x=0,1,2)
TIMER_DMA_CH2 D	通道2比较/捕获 DMA请求, TIMERx(x=0,1,2)
TIMER_DMA_CH3 D	通道3比较/捕获 DMA请求, TIMERx(x=0,1,2)
TIMER_DMA_CMT D	换相DMA更新请求, TIMERx(x=0)
TIMER_DMA_TRG D	触发DMA请求使能, TIMERx(x=0,1,2)
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 3-639. 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

<i>TIMERx</i> ( <i>x</i> =0,1,2,15,16)	TIMER外设选择
输入参数{in}	
<b>dma_request</b>	通道的DMA请求源选择
<i>TIMER_DMAREQUEST_CHANNELEVENT</i>	当通道捕获/比较事件发生时，发送通道 <i>y</i> 的DMA请求
<i>TIMER_DMAREQUEST_UPDATEEVENT</i>	当更新事件发生，发送通道 <i>y</i> 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 3-640. 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
输入参数{in}	
<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, <i>TIMERx</i> ( <i>x</i> =0,1,2,15,16)
<i>TIMER_DMACFG_DMATA_CTL1</i>	DMA传输起始地址：TIMER_DMACFG_DMATA_CTL1, <i>TIMERx</i> ( <i>x</i> =0,1,2,15,16)
<i>TIMER_DMACFG_DMATA_SMCFG</i>	DMA传输起始地址：TIMER_DMACFG_DMATA_SMCFG, <i>TIMERx</i> ( <i>x</i> =0,1,2)
<i>TIMER_DMACFG_</i>	DMA传输起始地址：TIMER_DMACFG_DMATA_DMAINTEN,

<i>DMATA_DMAINTEN</i>	TIMERx(x=0,1,2,15,16)
<i>TIMER_DMACFG_DMATA_INTF</i>	DMA传输起始地址: TIMER_DMACFG_DMATA_INTF, TIMERx(x=0,1,2,15,16)
<i>TIMER_DMACFG_DMATA_SWEVG</i>	DMA传输起始地址: TIMER_DMACFG_DMATA_SWEVG, TIMERx(x=0,1,2,15,16)
<i>TIMER_DMACFG_DMATA_CHCTL0</i>	DMA传输起始地址: TIMER_DMACFG_DMATA_CHCTL0, TIMERx(x=0,1,2,15,16)
<i>TIMER_DMACFG_DMATA_CHCTL1</i>	DMA传输起始地址: TIMER_DMACFG_DMATA_CHCTL1, TIMERx(x=0,1,2)
<i>TIMER_DMACFG_DMATA_CHCTL2</i>	DMA传输起始地址: TIMER_DMACFG_DMATA_CHCTL2, TIMERx(x=0,1,2,15,16)
<i>TIMER_DMACFG_DMATA_CNT</i>	DMA传输起始地址: TIMER_DMACFG_DMATA_CNT, TIMERx(x=0,1,2,15,16)
<i>TIMER_DMACFG_DMATA_PSC</i>	DMA传输起始地址: TIMER_DMACFG_DMATA_PSC, TIMERx(x=0,1,2,15,16)
<i>TIMER_DMACFG_DMATA_CAR</i>	DMA传输起始地址: TIMER_DMACFG_DMATA_CAR, TIMERx(x=0,1,2,15,16)
<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=0,1,2,15,16)
<i>TIMER_DMACFG_DMATA_CH1CV</i>	DMA传输起始地址: TIMER_DMACFG_DMATA_CH1CV, TIMERx(x=0,1,2)
<i>TIMER_DMACFG_DMATA_CH2CV</i>	DMA传输起始地址: TIMER_DMACFG_DMATA_CH2CV, TIMERx(x=0,1,2)
<i>TIMER_DMACFG_DMATA_CH3CV</i>	DMA传输起始地址: TIMER_DMACFG_DMATA_CH3CV, TIMERx(x=0,1,2)
<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=0,1,2,15,16)
<i>TIMER_DMACFG_DMATA_DMATB</i>	DMA传输起始地址: TIMER_DMACFG_DMATA_DMATB, TIMERx(x=0,1,2,15,16)
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 3-641. 函数 timer\_event\_software\_generate

函数名称	timer_event_software_generate
函数原型	void timer_event_software_generate(uint32_t timer_periph, uint16_t event);
功能描述	软件产生事件
先决条件	-
被调用函数	-
输入参数{in}	
timer_periph	TIMER外设
TIMERx	参考具体参数
输入参数{in}	
event	事件源
TIMER_EVENT_SR_C_UPG	更新事件产生，TIMERx(x=0,1,2,5,15,16)
TIMER_EVENT_SR_C_CH0G	通道0捕获或比较事件发生，TIMERx(x=0,1,2,15,16)
TIMER_EVENT_SR_C_CH1G	通道1捕获或比较事件发生，TIMERx(x=0,1,2)
TIMER_EVENT_SR_C_CH2G	通道2捕获或比较事件发生，TIMERx(x=0,1,2)
TIMER_EVENT_SR_C_CH3G	通道3捕获或比较事件发生，TIMERx(x=0,1,2)
TIMER_EVENT_SR_C_CMTG	通道换相更新事件发生，TIMERx(x=0,15,16)
TIMER_EVENT_SR_C_TRGG	触发事件产生，TIMERx(x=0,1,2)
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 3-642. 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3-614. 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 3-643. 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3-614. 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.idleoffstate     = 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 3-644. 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 3-645. 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 TIMERO break function*/
```

```
timer_break_disable(TIMERO);
```

### 函数 timer\_automatic\_output\_enable

函数timer\_automatic\_output\_enable描述见下表：

表 3-646. 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 TIMERO output automatic function */
```

```
timer_automatic_output_enable(TIMERO);
```

### 函数 timer\_automatic\_output\_disable

函数timer\_automatic\_output\_disable描述见下表：



表 3-647. 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 3-648. 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 3-649. 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 3-650. 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}	

<b>ccuctl</b>	通道换相控制影子寄存器更新控制
<i>TIMER_UPDATECTL_CCUC</i>	CMTG位被置1时更新影子寄存器
<i>TIMER_UPDATECTL_CCUTRI</i>	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 3-651. 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3-615. 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 3-652. 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=0,1,2,15,16)
TIMER_CH_1	通道1, TIMERx(x=0,1,2)
TIMER_CH_2	通道2, TIMERx(x=0,1,2)
TIMER_CH_3	通道3, TIMERx(x=0,1,2)
输入参数{in}	
ocpara	输出通道结构体, 详见 <a href="#">表3-615. 结构体timer_oc_parameter_struct</a>
输出参数{out}	
-	-
返回值	
-	-

例如:

```

/* configure TIMER0 channel 0 output function */

timer_oc_parameter_struct timer_ocintpara;

timer_ocintpara.outputstate = TIMER_CCX_ENABLE;

timer_ocintpara.outputnstate = TIMER_CCXN_ENABLE;

timer_ocintpara.ocpolarity = TIMER_OC_POLARITY_HIGH;

timer_ocintpara.ocnpolarity = TIMER_OCN_POLARITY_HIGH;

timer_ocintpara.ocidlestate = TIMER_OC_IDLE_STATE_HIGH;

timer_ocintpara.ocnidlestate = TIMER_OCN_IDLE_STATE_LOW;

timer_channel_output_config(TIMER0, TIMER_CH_0, &timer_ocintpara);

```

### 函数 timer\_channel\_output\_mode\_config

函数timer\_channel\_output\_mode\_config描述见下表:

**表 3-653. 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=0,1,2,15,16)
TIMER_CH_1	通道1, TIMERx (x=0,1,2)
TIMER_CH_2	通道2, TIMERx (x=0,1,2)
TIMER_CH_3	通道3, TIMERx (x=0,1,2)
输入参数{in}	
ocmode	通道输出比较模式
TIMER_OC_MODE_TIMING	冻结模式
TIMER_OC_MODE_ACTIVE	匹配时设置为高
TIMER_OC_MODE_INACTIVE	匹配时设置为低
TIMER_OC_MODE_TOGGLE	匹配时翻转
TIMER_OC_MODE_LOW	强制为低
TIMER_OC_MODE_HIGH	强制为高
TIMER_OC_MODE_PWM0	PWM模式0
TIMER_OC_MODE_PWM1	PWM模式1
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* configure TIMER0 channel PWM 0 mode */
```

```
timer_channel_output_mode_config(TIMER0,TIMER_CH_0,TIMER_OC_MODE_PWM0);
```

### 函数 timer\_channel\_output\_pulse\_value\_config

函数timer\_channel\_output\_pulse\_value\_config描述见下表:

表 3-654. 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=0,1,2,15,16)
TIMER_CH_1	通道1, TIMERx (x=0,1,2)
TIMER_CH_2	通道2, TIMERx (x=0,1,2)
TIMER_CH_3	通道3, TIMERx (x=0,1,2)
输入参数{in}	
pulse	通道输出比较值
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 3-655. 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

<i>TIMER_CH_0</i>	通道0, TIMERx (x=0,1,2,15,16)
<i>TIMER_CH_1</i>	通道1, TIMERx (x=0,1,2)
<i>TIMER_CH_2</i>	通道2, TIMERx (x=0,1,2)
<i>TIMER_CH_3</i>	通道3, TIMERx (x=0,1,2)
输入参数{in}	
<b>ocshadow</b>	输出比较影子寄存器功能状态
<i>TIMER_OC_SHADOW_ENABLE</i>	使能输出比较影子寄存器
<i>TIMER_OC_SHADOW_DISABLE</i>	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 3-656. 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}	
<b>timer_periph</b>	TIMER外设
<i>TIMERx</i>	参考具体参数
输入参数{in}	
<b>channel</b>	待配置通道
<i>TIMER_CH_0</i>	通道0, TIMERx (x=0,1,2,15,16)
<i>TIMER_CH_1</i>	通道1, TIMERx (x=0,1,2)
<i>TIMER_CH_2</i>	通道2, TIMERx (x=0,1,2)
<i>TIMER_CH_3</i>	通道3, TIMERx (x=0,1,2)
输入参数{in}	
<b>ocfast</b>	通道输出比较快速功能状态
<i>TIMER_OC_FAST_</i>	通道输出比较快速功能使能

ENABLE	
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 3-657. 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
TIMER_CH_0	通道0, TIMERx (x=0,1,2)
TIMER_CH_1	通道1, TIMERx (x=0,1,2)
TIMER_CH_2	通道2, TIMERx (x=0,1,2)
TIMER_CH_3	通道3, TIMERx (x=0,1,2)
输入参数{in}	
occlear	通道比较输出清0功能状态
TIMER_OC_CLEAR_ENABLE	通道比较输出清0功能使能
TIMER_OC_CLEAR_DISABLE	通道比较输出清0功能禁能
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 3-658. 函数 timer\_channel\_output\_polarity\_config

函数名称	timer_channel_output_polarity_config
函数原型	void timer_channel_output_polarity_config(uint32_t timer_periph, uint16_t channel, uint16_t ocpolarity);
功能描述	通道输出极性配置
先决条件	-
被调用函数	-
输入参数{in}	
timer_periph	TIMER外设
TIMERx	参考具体参数
输入参数{in}	
channel	待配置通道
TIMER_CH_0	通道0，TIMERx (x=0,1,2,15,16)
TIMER_CH_1	通道1，TIMERx (x=0,1,2)
TIMER_CH_2	通道2，TIMERx (x=0,1,2)
TIMER_CH_3	通道3，TIMERx (x=0,1,2)
输入参数{in}	
ocpolarity	通道输出极性
TIMER_OC_POLARITY_HIGH	通道输出极性高电平有效
TIMER_OC_POLARITY_LOW	通道输出极性低电平有效
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* configure TIMER0 channel 0 output polarity */

timer_channel_output_polarity_config(TIMER0, TIMER_CH_0,
TIMER_OC_POLARITY_HIGH);
```

## 函数 timer\_channel\_complementary\_output\_polarity\_config

函数timer\_channel\_complementary\_output\_polarity\_config描述见下表:

表 3-659. 函数 timer\_channel\_complementary\_output\_polarity\_config

函数名称	timer_channel_complementary_output_polarity_config
函数原型	void timer_channel_complementary_output_polarity_config(uint32_t timer_periph, uint16_t channel, uint16_t ocnpolarity);
功能描述	互补通道输出极性配置
先决条件	-
被调用函数	-
输入参数{in}	
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}	
ocnpolarity	互补通道输出极性
TIMER_OCN_POLARITY_HIGH	互补通道输出极性高电平有效
TIMER_OCN_POLARITY_LOW	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 3-660. 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}	
<b>timer_periph</b>	TIMER外设
<i>TIMERx</i>	参考具体参数
输入参数{in}	
<b>channel</b>	待配置通道
<i>TIMER_CH_0</i>	通道0, <i>TIMERx</i> (x=0,1,2,15,16)
<i>TIMER_CH_1</i>	通道1, <i>TIMERx</i> (x=0,1,2)
<i>TIMER_CH_2</i>	通道2, <i>TIMERx</i> (x=0,1,2)
<i>TIMER_CH_3</i>	通道3, <i>TIMERx</i> (x=0,1,2)
输入参数{in}	
<b>state</b>	通道状态
<i>TIMER_CCX_ENABLE</i>	通道使能
<i>TIMER_CCX_DISABLE</i>	通道禁能
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 3-661. 函数 timer\_channel\_complementary\_output\_state\_config

函数名称	timer_channel_complementary_output_state_config
函数原型	void timer_channel_complementary_output_state_config(uint32_t timer_periph, uint16_t channel, uint16_t ocnstate);
功能描述	配置互补通道输出状态
先决条件	-
被调用函数	-
输入参数{in}	
<b>timer_periph</b>	TIMER外设
<i>TIMERx</i>	参考具体参数
输入参数{in}	
<b>channel</b>	待配置通道
<i>TIMER_CH_0</i>	通道0, <i>TIMERx</i> (x=0,15,16)

<i>TIMER_CH_1</i>	通道1, TIMERx (x=0)
<i>TIMER_CH_2</i>	通道2, TIMERx (x=0)
输入参数{in}	
<b>ocnstate</b>	互补通道状态
<i>TIMER_CCXN_ENA</i> <i>BLE</i>	互补通道使能
<i>TIMER_CCXN_DIS</i> <i>ABLE</i>	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 3-662. 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}	
<b>icpara</b>	通道输入结构体, 详见 <a href="#">表3-616. 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 3-663. 函数 timer\_input\_capture\_config

函数名称	timer_input_capture_config
函数原型	void timer_input_capture_config(uint32_t timer_periph, uint16_t channel, timer_ic_parameter_struct* icpara);
功能描述	配置TIMER输入捕获参数
先决条件	-
被调用函数	timer_channel_input_capture_prescaler_config
输入参数{in}	
timer_periph	TIMER外设
TIMERx	参考具体参数
输入参数{in}	
channel	待配置通道
TIMER_CH_0	通道0, TIMERx (x=0,1,2,15,16)
TIMER_CH_1	通道1, TIMERx (x=0,1,2)
TIMER_CH_2	通道2, TIMERx (x=0,1,2)
TIMER_CH_3	通道3, TIMERx (x=0,1,2)
输入参数{in}	
icpara	输入捕获结构体, 详见 <a href="#">表3-616. 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 3-664. 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=0,1,2,15,16)
TIMER_CH_1	通道1, TIMERx (x=0,1,2)
TIMER_CH_2	通道2, TIMERx (x=0,1,2)
TIMER_CH_3	通道3, TIMERx (x=0,1,2)
输入参数{in}	
prescaler	通道输入捕获预分频值
TIMER_IC_PSC_DIV1	不分频
TIMER_IC_PSC_DIV2	2分频
TIMER_IC_PSC_DIV4	4分频
TIMER_IC_PSC_DIV8	8分频
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* configure TIMER0 channel 0 input capture prescaler value */
timer_channel_input_capture_prescaler_config(TIMER0, TIMER_CH_0,
TIMER_IC_PSC_DIV2);
```

### 函数 timer\_channel\_capture\_value\_register\_read

函数timer\_channel\_capture\_value\_register\_read描述见下表:

表 3-665. 函数 timer\_channel\_capture\_value\_register\_read

函数名称	timer_channel_capture_value_register_read
函数原型	uint32_t timer_channel_capture_value_register_read(uint32_t timer_periph, uint16_t channel);
功能描述	读取通道捕获值
先决条件	-
被调用函数	-
输入参数{in}	

<b>timer_periph</b>	TIMER外设
<i>TIMERx</i>	参考具体参数
<b>输入参数{in}</b>	
<b>channel</b>	待配置通道
<i>TIMER_CH_0</i>	通道0, TIMERx (x=0,1,2,15,16)
<i>TIMER_CH_1</i>	通道1, TIMERx (x=0,1,2)
<i>TIMER_CH_2</i>	通道2, TIMERx (x=0,1,2)
<i>TIMER_CH_3</i>	通道3, TIMERx (x=0,1,2)
<b>输出参数{out}</b>	
-	-
<b>返回值</b>	
<b>uint32_t</b>	通道输入捕获值, (0~0xFFFFFFFF)

例如:

```
/* read TIMER0 channel 0 capture compare register value */
```

```
uint32_t ch0_value = 0;
```

```
ch0_value = timer_channel_capture_value_register_read(TIMER0, TIMER_CH_0);
```

## 函数 timer\_input\_pwm\_capture\_config

函数timer\_input\_pwm\_capture\_config描述见下表:

**表 3-666. 函数 timer\_input\_pwm\_capture\_config**

<b>函数名称</b>	timer_input_pwm_capture_config
<b>函数原型</b>	void timer_input_pwm_capture_config(uint32_t timer_periph, uint16_t channel, timer_ic_parameter_struct* icpwm);
<b>功能描述</b>	配置TIMER捕获PWM输入参数
<b>先决条件</b>	-
<b>被调用函数</b>	timer_channel_input_capture_prescaler_config
<b>输入参数{in}</b>	
<b>timer_periph</b>	TIMER外设
<i>TIMERx(x=0,1,2)</i>	TIMER外设选择
<b>输入参数{in}</b>	
<b>channel</b>	待配置通道
<i>TIMER_CH_0</i>	通道0
<i>TIMER_CH_1</i>	通道1
<b>输入参数{in}</b>	
<b>icpwm</b>	输入捕获结构体, 详见 <a href="#">表3-616. 结构体timer ic parameter struct</a>
<b>输出参数{out}</b>	
-	-
<b>返回值</b>	
-	-

例如：

```
/* configure TIMER0 input pwm capture parameter */

timer_ic_parameter_struct timer_icinitpara;

timer_icinitpara.icpolarity = TIMER_IC_POLARITY_RISING;

timer_icinitpara.icselection = TIMER_IC_SELECTION_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 3-667. 函数 timer\_hall\_mode\_config

函数名称	timer_hall_mode_config
函数原型	void timer_hall_mode_config(uint32_t timer_periph, uint32_t hallmode);
功能描述	配置TIMER的HALL接口功能
先决条件	-
被调用函数	-
输入参数{in}	
<b>timer_periph</b>	TIMER外设
<i>TIMERx(x=0,1,2)</i>	TIMER外设选择
输入参数{in}	
<b>hallmode</b>	HALL接口功能状态
<i>TIMER_HALLINTERFACE_ENABLE</i>	HALL接口使能
<i>TIMER_HALLINTERFACE_DISABLE</i>	HALL接口禁能
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* configure TIMER0 hall sensor mode */

timer_hall_mode_config(TIMER0, TIMER_HALLINTERFACE_ENABLE);
```

### 函数 timer\_channel\_input\_remap\_config

函数timer\_channel\_input\_remap\_config描述见下表：



表 3-668. 函数 timer\_channel\_input\_remap\_config

函数名称	timer_channel_input_remap_config
函数原型	void timer_channel_input_remap_config(uint32_t timer_periph, uint32_t remap);
功能描述	配置TIMER的通道输入映射
先决条件	-
被调用函数	-
输入参数{in}	
timer_periph	TIMER外设
TIMERx(x=2)	TIMER外设选择
输入参数{in}	
remap	通道映射选择
TIMER_IRMP_GPIO	通道3输入连接到GPIO
TIMER_IRMP_IRC32K	通道3输入连接到IRC32K
TIMER_IRMP_LXTAL	通道3输入连接到LXTAL
TIMER_IRMP_CKOUT	通道3输入连接到CKOUT
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* configure TIMER channel input remap */
```

```
timer_channel_input_remap_config(TIMER2, TIMER_IRMP_GPIO);
```

### 函数 timer\_input\_trigger\_source\_select

函数timer\_input\_trigger\_source\_select描述见下表：

表 3-669. 函数 timer\_input\_trigger\_source\_select

函数名称	timer_input_trigger_source_select
函数原型	void timer_input_trigger_source_select(uint32_t timer_periph, uint32_t intrigger);
功能描述	TIMER的输入触发源选择
先决条件	SMC[2:0] = 000
被调用函数	-
输入参数{in}	
timer_periph	TIMER外设
TIMERx(x=0,1,2)	TIMER外设选择

输入参数{in}	
<b>intrigger</b>	待选择的触发源
<i>TIMER_SMCFG_TRGSEL_ITI0</i>	内部触发输入0
<i>TIMER_SMCFG_TRGSEL_ITI1</i>	内部触发输入1
<i>TIMER_SMCFG_TRGSEL_ITI2</i>	内部触发输入2
<i>TIMER_SMCFG_TRGSEL_ITI3</i>	内部触发输入3
<i>TIMER_SMCFG_TRGSEL_CIOF_ED</i>	CIO的边沿标志位
<i>TIMER_SMCFG_TRGSEL_CIOFE0</i>	滤波后的通道0输入
<i>TIMER_SMCFG_TRGSEL_CIOFE1</i>	滤波后的通道1输入
<i>TIMER_SMCFG_TRGSEL_ETIFP</i>	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 3-670. 函数 timer\_master\_output\_trigger\_source\_select**

<b>函数名称</b>	timer_master_output_trigger_source_select
<b>函数原型</b>	void timer_master_output_trigger_source_select(uint32_t timer_periph, uint32_t outtrigger);
<b>功能描述</b>	选择TIMER主模式输出触发
<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=0,1,2)</i>	TIMER外设选择
输入参数{in}	
<b>outtrigger</b>	主模式输出触发

<code>TIMER_TRI_OUT_SRC_RESET</code>	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。
<code>TIMER_TRI_OUT_SRC_ENABLE</code>	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。
<code>TIMER_TRI_OUT_SRC_UPDATE</code>	更新。主模式控制器选择更新事件作为TRGO。
<code>TIMER_TRI_OUT_SRC_CC0</code>	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
<code>TIMER_TRI_OUT_SRC_O0CPRE</code>	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
<code>TIMER_TRI_OUT_SRC_O1CPRE</code>	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
<code>TIMER_TRI_OUT_SRC_O2CPRE</code>	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
<code>TIMER_TRI_OUT_SRC_O3CPRE</code>	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 3-671. 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=0,1,2)	TIMER外设选择
输入参数{in}	
slavemode	从模式

<code>TIMER_SLAVE_MODE_DISABLE</code>	关闭从模式
<code>TIMER_QUAD_DECODER_MODE0</code>	正交译码器模式0
<code>TIMER_QUAD_DECODER_MODE1</code>	正交译码器模式1
<code>TIMER_QUAD_DECODER_MODE2</code>	正交译码器模式2
<code>TIMER_SLAVE_MODE_RESTART</code>	复位模式
<code>TIMER_SLAVE_MODE_PAUSE</code>	暂停模式
<code>TIMER_SLAVE_MODE_EVENT</code>	事件模式
<code>TIMER_SLAVE_MODE_EXTERNAL0</code>	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 3-672. 函数 timer\_master\_slave\_mode\_config**

函数名称	timer_master_slave_mode_config
函数原型	void timer_master_slave_mode_config(uint32_t timer_periph, uint32_t masterslave);
功能描述	TIMER主从模式配置
先决条件	-
被调用函数	-
输入参数{in}	
timer_periph	TIMER外设
<code>TIMERx(x=0,1,2)</code>	TIMER外设选择
输入参数{in}	
masterslave	主从模式使能状态
<code>TIMER_MASTER_SLAVE_MODE_ENA</code>	主从模式使能

BLE	
TIMER_MASTER_SLAVE_MODE_DISABLE	主从模式禁能
BLE	
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 3-673. 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=0,1,2)	TIMER外设选择
输入参数{in}	
extprescaler	外部触发预分频
TIMER_EXT_TRIP_SC_OFF	不分频
TIMER_EXT_TRIP_SC_DIV2	2分频
TIMER_EXT_TRIP_SC_DIV4	4分频
TIMER_EXT_TRIP_SC_DIV8	8分频
输入参数{in}	
expolarity	外部触发输入极性
TIMER_ETP_FALLING	低电平或者下降沿有效
TIMER_ETP_RISING	高电平或者上升沿有效

输入参数{in}	
<b>extfilter</b>	外部触发滤波控制（0~15）
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* configure TIMER0 external trigger input */
```

```
timer_external_trigger_config(TIMER0, TIMER_EXT_TRI_PSC_DIV2,
TIMER_ETP_FALLING, 10);
```

### 函数 timer\_quadrature\_decoder\_mode\_config

函数timer\_quadrature\_decoder\_mode\_config描述见下表：

**表 3-674. 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}	
<b>timer_periph</b>	TIMER外设
<i>TIMERx(x=0,1,2)</i>	TIMER外设选择
输入参数{in}	
<b>decomode</b>	正交译码器模式
<i>TIMER_QUAD_DECODER_MODE0</i>	根据CI0FE0的电平，计数器在CI1FE1的边沿向上/下计数
<i>TIMER_QUAD_DECODER_MODE1</i>	根据CI1FE1的电平，计数器在CI0FE0的边沿向上/下计数
<i>TIMER_QUAD_DECODER_MODE2</i>	根据另一个信号的输入电平，计数器在CI0FE0和CI1FE1的边沿向上/下计数
输入参数{in}	
<b>ic0polarity</b>	IC0极性
<i>TIMER_IC_POLARITY_RISING</i>	捕获上升边沿
<i>TIMER_IC_POLARITY_FALLING</i>	捕获下降边沿
<i>TIMER_IC_POLARITY_BOTH_EDGE</i>	双边沿有效
输入参数{in}	

<b>ic1polarity</b>	IC1极性
<i>TIMER_IC_POLARITY_RISING</i>	捕获上升边沿
<i>TIMER_IC_POLARITY_FALLING</i>	捕获下降边沿
<i>TIMER_IC_POLARITY_BOTH_EDGE</i>	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 3-675. 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}	
<b>timer_periph</b>	TIMER外设
<i>TIMERx(x=0,1,2)</i>	TIMER外设选择
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* configure TIMER0 internal clock mode */
```

```
timer_internal_clock_config (TIMER0);
```

### 函数 timer\_internal\_trigger\_as\_external\_clock\_config

函数timer\_internal\_trigger\_as\_external\_clock\_config描述见下表：

表 3-676. 函数 timer\_internal\_trigger\_as\_external\_clock\_config

函数名称	timer_internal_trigger_as_external_clock_config
函数原型	void timer_internal_trigger_as_external_clock_config(uint32_t timer_periph, uint32_t intrigger);
功能描述	配置TIMER的内部触发为时钟源
先决条件	-
被调用函数	timer_input_trigger_source_select
输入参数{in}	
timer_periph	TIMER外设
TIMERx(x=0,1,2)	TIMER外设选择
输入参数{in}	
intrigger	被选择的内部触发源
TIMER_SMCFG_TRGSEL_ITI0	选择内部触发0 (ITI0)为时钟源
TIMER_SMCFG_TRGSEL_ITI1	选择内部触发1 (ITI1)为时钟源
TIMER_SMCFG_TRGSEL_ITI2	选择内部触发2 (ITI2)为时钟源
TIMER_SMCFG_TRGSEL_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 ITI0 as external clock input */
```

```
timer_internal_trigger_as_external_clock_config(TIMER0, TIMER_SMCFG_TRGSEL_ITI0);
```

### 函数 timer\_external\_trigger\_as\_external\_clock\_config

函数timer\_external\_trigger\_as\_external\_clock\_config描述见下表:

表 3-677. 函数 timer\_external\_trigger\_as\_external\_clock\_config

函数名称	timer_external_trigger_as_external_clock_config
函数原型	void timer_external_trigger_as_external_clock_config(uint32_t timer_periph, uint32_t extrigger, uint16_t expolarity, uint32_t extfilter);
功能描述	配置TIMER的外部触发作为时钟源
先决条件	-
被调用函数	timer_input_trigger_source_select
输入参数{in}	
timer_periph	TIMER外设
TIMERx(x=0,1,2)	TIMER外设选择



输入参数{in}	
<b>extrigger</b>	外部触发源
<i>TIMER_SMCFG_T RGSEL_CIOF_ED</i>	CI0的边沿标志(CIOF_ED)
<i>TIMER_SMCFG_T RGSEL_CIOFE0</i>	滤波后的通道0输入(CIOFE0)
<i>TIMER_SMCFG_T RGSEL_CII1FE1</i>	滤波后的通道1输入(CII1FE1)
输入参数{in}	
<b>expolarity</b>	外部触发源极性
<i>TIMER_IC_POLARI TY_RISING</i>	外部触发源高电平或者上升沿有效
<i>TIMER_IC_POLARI TY_FALLING</i>	外部触发源低电平或者下降沿有效
<i>TIMER_IC_POLARI TY_BOTH_EDGE</i>	外部触发双边沿有效
输入参数{in}	
<b>extfilter</b>	滤波参数 (0~15)
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* configure TIMER0 the external trigger CIOFE0 as external clock input */
timer_external_trigger_as_external_clock_config(TIMER0,
TIMER_SMCFG_TRGSEL_CIOFE0, 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 3-678. 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}	
<b>timer_periph</b>	TIMER外设
<i>TIMERx(x=0,1,2)</i>	TIMER外设选择
输入参数{in}	

<b>extprescaler</b>	ETI触发源预分频值
<i>TIMER_EXT_TRI_P</i> <i>SC_OFF</i>	不分频
<i>TIMER_EXT_TRI_P</i> <i>SC_DIV2</i>	2分频
<i>TIMER_EXT_TRI_P</i> <i>SC_DIV4</i>	4分频
<i>TIMER_EXT_TRI_P</i> <i>SC_DIV8</i>	8分频
<b>输入参数{in}</b>	
<b>expolarity</b>	ETI触发源极性
<i>TIMER_ETP_FALLI</i> <i>NG</i>	下降沿或者低电平有效
<i>TIMER_ETP_RISIN</i> <i>G</i>	上升沿或者高电平有效
<b>输入参数{in}</b>	
<b>extfilter</b>	ETI触发源滤波参数（0~15）
<b>输出参数{out}</b>	
-	-
<b>返回值</b>	
-	-

例如：

```
/* configure TIMER0 the external clock mode0 */
```

```
timer_external_clock_mode0_config(TIMER0, TIMER_EXT_TRI_PSC_DIV2,  
TIMER_ETP_FALLING, 0);
```

### 函数 timer\_external\_clock\_mode1\_config

函数timer\_external\_clock\_mode1\_config描述见下表：

**表 3-679. 函数 timer\_external\_clock\_mode1\_config**

<b>函数名称</b>	timer_external_clock_mode1_config
<b>函数原型</b>	void timer_external_clock_mode1_config(uint32_t timer_periph, uint32_t extprescaler, uint32_t extpolarity, uint32_t extfilter);
<b>功能描述</b>	配置TIMER外部时钟模式1
<b>先决条件</b>	-
<b>被调用函数</b>	timer_external_trigger_config
<b>输入参数{in}</b>	
<b>timer_periph</b>	TIMER外设
<i>TIMERx(x=0,1,2)</i>	TIMER外设选择
<b>输入参数{in}</b>	
<b>extprescaler</b>	ETI触发源预分频值

<i>TIMER_EXT_TRI_P</i> <i>SC_OFF</i>	不分频
<i>TIMER_EXT_TRI_P</i> <i>SC_DIV2</i>	2分频
<i>TIMER_EXT_TRI_P</i> <i>SC_DIV4</i>	4分频
<i>TIMER_EXT_TRI_P</i> <i>SC_DIV8</i>	8分频
输入参数{in}	
<b>extpolarity</b>	ETI触发源极性
<i>TIMER_ETP_FALLI</i> <i>NG</i>	下降沿或者低电平有效
<i>TIMER_ETP_RISIN</i> <i>G</i>	上升沿或者高电平有效
输入参数{in}	
<b>extfilter</b>	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 3-680. 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}	
<b>timer_periph</b>	TIMER外设
<i>TIMERx(x=0,1,2)</i>	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 3-681. 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=0,1,2,15,16)	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
TIMER_CHVSEL_ENABLE	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 3-682. 函数 timer\_output\_value\_selection\_config**

函数名称	timer_output_value_selection_config
函数原型	void timer_output_value_selection_config(uint32_t timer_periph, uint16_t outsel);
功能描述	配置TIMER输出值选择位
先决条件	-

被调用函数	-
输入参数{in}	
timer_periph	TIMER外设
TIMERx (x=0,15,16)	TIMER外设选择
输入参数{in}	
ccsel	输出值选择位
TIMER_OUTSEL_DISABLE	无影响
TIMER_OUTSEL_ENABLE	如果POEN位与IOS位均为0，则输出无效。
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* configure TIMER output value selection */
```

```
timer_output_value_selection_config(TIMER0, TIMER_OUTSEL_ENABLE);
```

### 函数 timer\_flag\_get

函数timer\_flag\_get描述见下表：

表 3-683. 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=0,1,2,5,15,16)
TIMER_FLAG_CH0	通道0比较/捕获标志，TIMERx(x=0,1,2,15,16)
TIMER_FLAG_CH1	通道1比较/捕获标志，TIMERx(x=0,1,2)
TIMER_FLAG_CH2	通道2比较/捕获标志，TIMERx(x=0,1,2)
TIMER_FLAG_CH3	通道3比较/捕获标志，TIMERx(x=0,1,2)
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=0,1,2)
TIMER_FLAG_BRK	中止标志位，TIMERx(x=0,15,16)
TIMER_FLAG_CH0	通道0捕获溢出标志，TIMERx(x=0,1,2,15,16)

0	
TIMER_FLAG_CH1	通道1捕获溢出标志, TIMERx(x=0,1,2)
0	
TIMER_FLAG_CH2	通道2捕获溢出标志, TIMERx(x=0,1,2)
0	
TIMER_FLAG_CH3	通道3捕获溢出标志, TIMERx(x=0,1,2)
0	
输出参数{out}	
-	-
返回值	
FlagStatus	SET或者RESET

例如:

```
/* get TIMER0 update flags */
```

```
FlagStatus Flag_status = RESET;
```

```
Flag_status = timer_flag_get(TIMER0, TIMER_FLAG_UP);
```

### 函数 timer\_flag\_clear

函数timer\_flag\_clear描述见下表:

表 3-684. 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
TIMERx	参考具体参数
输入参数{in}	
flag	状态标志
TIMER_FLAG_UP	更新标志, TIMERx(x=0,1,2,5,15,16)
TIMER_FLAG_CH0	通道0比较/捕获标志, TIMERx(x=0,1,2,15,16)
TIMER_FLAG_CH1	通道1比较/捕获标志, TIMERx(x=0,1,2)
TIMER_FLAG_CH2	通道2比较/捕获标志, TIMERx(x=0,1,2)
TIMER_FLAG_CH3	通道3比较/捕获标志, TIMERx(x=0,1,2)
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=0,1,2)
TIMER_FLAG_BRK	中止标志位, TIMERx(x=0,15,16)
TIMER_FLAG_CH0	通道0捕获溢出标志, TIMERx(x=0,1,2,15,16)
0	

<i>TIMER_FLAG_CH1</i> 0	通道1捕获溢出标志, <i>TIMERx</i> ( <i>x</i> =0,1,2)
<i>TIMER_FLAG_CH2</i> 0	通道2捕获溢出标志, <i>TIMERx</i> ( <i>x</i> =0,1,2)
<i>TIMER_FLAG_CH3</i> 0	通道3捕获溢出标志, <i>TIMERx</i> ( <i>x</i> =0,1,2)
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 3-685. 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
<i>TIMERx</i>	参考具体参数
输入参数{in}	
interrupt	中断源
<i>TIMER_INT_UP</i>	更新中断, <i>TIMERx</i> ( <i>x</i> =0,1,2,5,15,16)
<i>TIMER_INT_CH0</i>	通道0比较/捕获中断, <i>TIMERx</i> ( <i>x</i> =0,1,2,15,16)
<i>TIMER_INT_CH1</i>	通道1比较/捕获中断, <i>TIMERx</i> ( <i>x</i> =0,1,2)
<i>TIMER_INT_CH2</i>	通道2比较/捕获中断, <i>TIMERx</i> ( <i>x</i> =0,1,2)
<i>TIMER_INT_CH3</i>	通道3比较/捕获中断, <i>TIMERx</i> ( <i>x</i> =0,1,2)
<i>TIMER_INT_CMT</i>	换相更新中断, <i>TIMERx</i> ( <i>x</i> =0,15,16)
<i>TIMER_INT_TRG</i>	触发中断, <i>TIMERx</i> ( <i>x</i> =0,1,2)
<i>TIMER_INT_BRK</i>	中止中断, <i>TIMERx</i> ( <i>x</i> =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 3-686. 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=0,1,2,5,15,16)
TIMER_INT_CH0	通道0比较/捕获中断, TIMERx(x=0,1,2,15,16)
TIMER_INT_CH1	通道1比较/捕获中断, TIMERx(x=0,1,2)
TIMER_INT_CH2	通道2比较/捕获中断, TIMERx(x=0,1,2)
TIMER_INT_CH3	通道3比较/捕获中断, TIMERx(x=0,1,2)
TIMER_INT_CMT	换相更新中断, TIMERx(x=0,15,16)
TIMER_INT_TRG	触发中断, TIMERx(x=0,1,2)
TIMER_INT_BRK	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 3-687. 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
TIMER_INT_FLAG_UP	更新中断, TIMERx(x=0,1,2,5,15,16)
TIMER_INT_FLAG_CH0	通道0比较/捕获中断, TIMERx(x=0,1,2,15,16)
TIMER_INT_FLAG_CH1	通道1比较/捕获中断, TIMERx(x=0,1,2)
TIMER_INT_FLAG_CH2	通道2比较/捕获中断, TIMERx(x=0,1,2)
TIMER_INT_FLAG_CH3	通道3比较/捕获中断, TIMERx(x=0,1,2)
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=0,1,2)
TIMER_INT_FLAG_BRK	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 3-688. 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=0,1,2,5,15,16)
TIMER_INT_FLAG_CH0	通道0比较/捕获中断, TIMERx(x=0,1,2,15,16)
TIMER_INT_FLAG_CH1	通道1比较/捕获中断, TIMERx(x=0,1,2)
TIMER_INT_FLAG_CH2	通道2比较/捕获中断, TIMERx(x=0,1,2)
TIMER_INT_FLAG_CH3	通道3比较/捕获中断, TIMERx(x=0,1,2)
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=0,1,2)
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);
```

## 3.23. 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[3.23.1](#)，TRNG固件库函数介绍在章节[3.23.2](#)。

### 3.23.1. 外设寄存器说明

TRNG寄存器列表如下表所示:

表 3-689. 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

### 3.23.2. 外设库函数说明

TRNG库函数列表如下表所示：

表 3-690. 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 3-691. 枚举 trng\_flag\_enum

成员名称	功能描述
TRNG_FLAG_DRDY	随机数据就绪状态
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 3-692. 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 3-693. 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 3-694. 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 3-695. 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 3-696. 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 3-697. 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 3-698. 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 3-699. 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3-691. 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 3-700. 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3-692. 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 3-701. 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3-692. 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);
```

## 3.24. USART

通用同步异步收发器 (USART) 提供了一个灵活方便的串行数据交换接口, 章节 [3.24.1](#) 描述了 USART 的寄存器列表, 章 [3.24.2](#) 对 USART 库函数进行说明。

### 3.24.1. 外设寄存器说明

USART 寄存器列表如下表所示:

**表 3-702. 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

### 3.24.2. 外设库函数说明

USART 库函数列表如下表所示:



表 3-703. 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_overrun_enable	使能 USART 溢出禁止功能
usart_overrun_disable	失能 USART 溢出禁止功能
usart_oversample_config	配置 USART 过采样模式
usart_sample_bit_config	配置 USART 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
usart_receiver_timeout_threshold_config	设置 USART 接收超时阈值
usart_data_transmit	USART 发送数据功能
usart_data_receive	USART 接收数据功能
usart_command_enable	使能 USART 命令
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

库函数名称	库函数描述
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_asserttime_config	配置 USART 驱动使能置位时间
usart_driver_deasserttime_config	配置 USART 驱动使能置低时间
usart_depolarity_config	配置 USART 驱动使能极性模式
usart_dma_receive_config	配置 USART DMA 接收
usart_dma_transmit_config	配置 USART DMA 发送
usart_reception_error_dma_disable	USART 接收错误时失能 DMA
usart_reception_error_dma_enable	USART 接收错误时使能 DMA
usart_wakeup_enable	使能 USART 唤醒
usart_wakeup_disable	失能 USART 唤醒
usart_wakeup_mode_config	配置 USART 唤醒模式
usart_receive_fifo_enable	使能接收 FIFO
usart_receive_fifo_disable	失能接收 FIFO
usart_receive_fifo_counter_number	读取接收 FIFO 计数器的值
usart_flag_get	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 3-704. 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
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 3-705. 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

成员名称	功能描述
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 3-706. 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
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 3-707. 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 3-708. 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USART0/UARTx
先决条件	-
被调用函数	rcu_periph_reset_enable / rcu_periph_reset_disable
输入参数{in}	
usart_periph	外设USART0/UARTx
USART0/UARTx	x=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 3-709. 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USART0/UARTx
USART0/UARTx	x=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 3-710. 函数 `usart_parity_config`**

函数名称	<code>usart_parity_config</code>
函数原型	<code>void usart_parity_config(uint32_t usart_periph, uint32_t paritycfg);</code>
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}	
<code>usart_periph</code>	外设USART0/UARTx
<code>USART0/UARTx</code>	x=1,2
输入参数{in}	
<code>paritycfg</code>	配置USART奇偶校验
<code>USART_PM_NONE</code>	无校验
<code>USART_PM_ODD</code>	奇校验
<code>USART_PM_EVEN</code>	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 3-711. 函数 `usart_word_length_set`**

函数名称	<code>usart_word_length_set</code>
函数原型	<code>void usart_word_length_set(uint32_t usart_periph, uint32_t wlen);</code>
功能描述	配置USART字长
先决条件	-
被调用函数	-
输入参数{in}	
<code>usart_periph</code>	外设USART0/UARTx
<code>USART0/UARTx</code>	x=1,2
输入参数{in}	
<code>wlen</code>	配置USART字长
<code>USART_WL_8BIT</code>	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 3-712. 函数 usart\_stop\_bit\_set

函数名称	usart_stop_bit_set
函数原型	void usart_stop_bit_set(uint32_t usart_periph, uint32_t stblen);
功能描述	配置USART停止位
先决条件	-
被调用函数	-
输入参数{in}	
usart_periph	外设USART0/UARTx
USART0/UARTx	x=1,2
输入参数{in}	
stblen	配置USART停止位
USART_STB_1BIT	1停止位
USART_STB_0_5BIT	0.5停止位
USART_STB_2BIT	2停止位
USART_STB_1_5BIT	1.5停止位
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* configure USART0 stop bit length */
```

```
usart_stop_bit_set(USART0, USART_STB_1_5BIT);
```

### 函数 usart\_enable

函数usart\_enable描述见下表:

表 3-713. 函数 usart\_enable

函数名称	usart_enable
函数原型	void usart_enable(uint32_t usart_periph);
功能描述	使能USART
先决条件	-
被调用函数	-
输入参数{in}	
usart_periph	外设USART0/UARTx
USART0/UARTx	x=1,2
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* enable USART0 */
usart_enable(USART0);
```

### 函数 usart\_disable

函数usart\_disable描述见下表:

表 3-714. 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USART0/UARTx
USART0/UARTx	x=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 3-715. 函数 `usart_transmit_config`

函数名称	<code>usart_transmit_config</code>
函数原型	<code>void usart_transmit_config(uint32_t usart_periph, uint32_t txconfig);</code>
功能描述	USART发送器配置
先决条件	-
被调用函数	-
输入参数{in}	
<code>usart_periph</code>	外设USART0/UARTx
<code>USART0/UARTx</code>	x=1,2
输入参数{in}	
<code>txconfig</code>	使能/失能USART发送器
<code>USART_TRANSMIT_ENABLE</code>	使能USART发送
<code>USART_TRANSMIT_DISABLE</code>	失能USART发送
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* configure USART0 transmitter */
```

```
usart_transmit_config(USART0,USART_TRANSMIT_ENABLE);
```

### 函数 `usart_receive_config`

函数`usart_receive_config`描述见下表:

表 3-716. 函数 `usart_receive_config`

函数名称	<code>usart_receive_config</code>
函数原型	<code>void usart_receive_config(uint32_t usart_periph, uint32_t rxconfig);</code>
功能描述	USART接收器配置
先决条件	-
被调用函数	-
输入参数{in}	
<code>usart_periph</code>	外设USART0/UARTx
<code>USART0/UARTx</code>	x=1,2
输入参数{in}	
<code>rxconfig</code>	使能/失能USART接收器
<code>USART_RECEIVE_ENABLE</code>	使能USART接收
<code>USART_RECEIVE_DISABLE</code>	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 3-717. 函数 usart\_data\_first\_config**

函数名称	usart_data_first_config
函数原型	void usart_data_first_config(uint32_t usart_periph, uint32_t msbf);
功能描述	配置数据传输时低位在前或高位在前
先决条件	-
被调用函数	-
输入参数{in}	
usart_periph	外设USART0/UARTx
USART0/UARTx	x=1,2
输入参数{in}	
msbf	数据传输时低位在前/高位在前
USART_MSBF_LSB 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 3-718. 函数 usart\_invert\_config**

函数名称	usart_invert_config
------	---------------------

函数原型	void usart_invert_config(uint32_t usart_periph, usart_invert_enum invertpara);
功能描述	配置USART反转功能
先决条件	-
被调用函数	-
输入参数{in}	
usart_periph	外设USART0/UARTx
USART0/UARTx	x=1,2
输入参数{in}	
invertpara	参考 <a href="#">表3-707. 枚举类型usart_invert_enum</a>
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* configure USART0 inversion */
```

```
usart_invert_config(USART0, USART_DINV_ENABLE);
```

### 函数 usart\_overrun\_enable

函数usart\_overrun\_enable描述见下表:

**表 3-719. 函数 usart\_overrun\_enable**

函数名称	usart_overrun_enable
函数原型	void usart_overrun_enable (uint32_t usart_periph);
功能描述	使能USART溢出禁止功能
先决条件	-
被调用函数	-
输入参数{in}	
usart_periph	外设USART0/UARTx
USART0/UARTx	x=1,2
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* enable USART0 overrun */
```

```
usart_overrun_enable (USART0);
```

### 函数 usart\_overrun\_disable

函数usart\_overrun\_disable描述见下表:

表 3-720. 函数 usart\_oversample\_disable

函数名称	usart_oversample_disable
函数原型	void usart_oversample_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USART0/UARTx
USART0/UARTx	x=1,2
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* disable USART0 overrun */
usart_oversample_disable (USART0);
```

### 函数 usart\_oversample\_config

函数usart\_oversample\_config描述见下表:

表 3-721. 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USART0/UARTx
USART0/UARTx	x=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 3-722. 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USART0/UARTx
USART0/UARTx	x=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 3-723. 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USART0/UARTx
USART0	USART0
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 3-724. 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USART0/UARTx
USART0	USART0
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 3-725. 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USART0/UARTx

USART0	USART0
输入参数{in}	
rtimetypeout	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 3-726. 函数 usart\_data\_transmit

函数名称	usart_data_transmit
函数原型	void usart_data_transmit(uint32_t usart_periph, uint32_t data);
功能描述	USART发送数据功能
先决条件	-
被调用函数	-
输入参数{in}	
usart_periph	外设USART0/UARTx
USART0/UARTx	x=1,2
输入参数{in}	
data	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 3-727. 函数 usart\_data\_receive

函数名称	usart_data_receive
------	--------------------

函数原型	uint16_t usart_data_receive(uint32_t usart_periph);
功能描述	USART接收数据功能
先决条件	-
被调用函数	-
输入参数{in}	
usart_periph	外设USART0/UARTx
USART0/UARTx	x=1,2
输出参数{out}	
-	-
返回值	
uint32_t	接收到的数据

例如:

```
/* USART0 receive data */
```

```
uint16_t temp;
```

```
temp = usart_data_receive(USART0);
```

### 函数 usart\_command\_enable

函数usart\_command\_enable描述见下表:

表 3-728. 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USART0/UARTx
USART0/UARTx	x=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	静默模式请求
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\_address\_config

函数usart\_address\_config描述见下表：

**表 3-729. 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}	
usart_periph	外设USART0/UARTx
USART0/UARTx	x=1,2
输入参数{in}	
addr	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 3-730. 函数 usart\_address\_detection\_mode\_config**

函数名称	usart_address_detection_mode_config
函数原型	void usart_address_detection_mode_config(uint32_t usart_periph, uint32_t addmod);
功能描述	配置USART地址检测模式
先决条件	-
被调用函数	-
输入参数{in}	

<b>usart_periph</b>	外设USART0/UARTx
<i>USART0/UARTx</i>	x=1,2
<b>输入参数{in}</b>	
<b>addmod</b>	地址检测模式
<i>USART_ADDM_4BIT</i>	4位地址检测
<i>USART_ADDM_FULLBIT</i>	全位地址检测
<b>输出参数{out}</b>	
-	-
<b>返回值</b>	
-	-

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 3-731. 函数 usart\_mute\_mode\_enable**

<b>函数名称</b>	usart_mute_mode_enable
<b>函数原型</b>	void usart_mute_mode_enable(uint32_t usart_periph);
<b>功能描述</b>	使能USART静默模式
<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USART0/UARTx
<i>USART0/UARTx</i>	x=1,2
<b>输出参数{out}</b>	
-	-
<b>返回值</b>	
-	-

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 3-732. 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USART0/UARTx
<code>USART0/UARTx</code>	x=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 3-733. 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USART0/UARTx
<code>USART0/UARTx</code>	x=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 3-734. 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USART0/UARTx
USART0	USART0
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 3-735. 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USART0/UARTx
USART0	USART0
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 3-736. 函数 `usart_lin_break_dection_length_config`

函数名称	<code>usart_lin_break_dection_length_config</code>
函数原型	<code>void usart_lin_break_dection_length_config(uint32_t usart_periph, uint32_t lblen);</code>
功能描述	配置USART LIN模式中断帧长度
先决条件	-
被调用函数	-
输入参数{in}	
<code>usart_periph</code>	外设USART0/UARTx
<code>USART0</code>	USART0
输入参数{in}	
<code>lblen</code>	LIN模式中断帧长度
<code>USART_LBLEN_10</code> <code>B</code>	断开帧长度为10 bits
<code>USART_LBLEN_11</code> <code>B</code>	断开帧长度为11 bits
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* configure LIN break frame length */
```

```
usart_lin_break_dection_length_config(USART0, USART_LBLEN_10B);
```

## 函数 `usart_halfduplex_enable`

函数`usart_halfduplex_enable`描述见下表:

表 3-737. 函数 `usart_halfduplex_enable`

函数名称	<code>usart_halfduplex_enable</code>
函数原型	<code>void usart_halfduplex_enable(uint32_t usart_periph);</code>
功能描述	使能USART半双工模式
先决条件	-
被调用函数	-
输入参数{in}	
<code>usart_periph</code>	外设USART0/UARTx
<code>USART0/UARTx</code>	x=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 3-738. 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USART0/UARTx
USART0/UARTx	x=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 3-739. 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USART0/UARTx
USART0	USART0
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 3-740. 函数 usart\_clock\_disable

函数名称	usart_clock_disable
函数原型	void usart_clock_disable(uint32_t usart_periph);
功能描述	失能USART CK引脚
先决条件	-
被调用函数	-
输入参数{in}	
usart_periph	外设USART0/UARTx
USART0	USART0
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 3-741. 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USART0/UARTx

USART0	USART0
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 3-742. 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USART0/UARTx
USART0	USART0
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 3-743. 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USART0/UARTx
USART0	USART0
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 3-744. 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USART0/UARTx
USART0	USART0
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 3-745. 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USART0/UARTx
USART0	USART0
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 3-746. 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USART0/UARTx
USART0	USART0
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 3-747. 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USART0/UARTx
USART0	USART0
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 3-748. 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USART0/UARTx
USART0	USART0
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 3-749. 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USART0/UARTx
USART0	USART0
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 3-750. 函数 usart\_block\_length\_config**

函数名称	usart_block_length_config
函数原型	void usart_block_length_config(uint32_t usart_periph, uint32_t bl);
功能描述	配置智能卡T=1的接收时块的长度
先决条件	-
被调用函数	-
输入参数{in}	
usart_periph	外设USART0/UARTx
USART0	USART0
输入参数{in}	

<b>bl</b>	块长度（0x00-0xFF）
<b>输出参数{out}</b>	
-	-
<b>返回值</b>	
-	-

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 3-751. 函数 usart\_irda\_mode\_enable**

<b>函数名称</b>	usart_irda_mode_enable
<b>函数原型</b>	void usart_irda_mode_enable(uint32_t usart_periph);
<b>功能描述</b>	使能USART串行红外编解码功能模块
<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USART0/UARTx
USART0	USART0
<b>输出参数{out}</b>	
-	-
<b>返回值</b>	
-	-

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 3-752. 函数 usart\_irda\_mode\_disable**

<b>函数名称</b>	usart_irda_mode_disable
<b>函数原型</b>	void usart_irda_mode_disable(uint32_t usart_periph);
<b>功能描述</b>	失能USART串行红外编解码功能模块
<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USART0/UARTx
<i>USART0</i>	USART0
<b>输出参数{out}</b>	
-	-
<b>返回值</b>	
-	-

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 3-753. 函数 usart\_prescaler\_config**

<b>函数名称</b>	usart_prescaler_config
<b>函数原型</b>	void usart_prescaler_config(uint32_t usart_periph, uint32_t psc);
<b>功能描述</b>	在USART IrDA低功耗模式下配置外设时钟分频系数
<b>先决条件</b>	-
<b>被调用函数</b>	-
<b>输入参数{in}</b>	
<b>usart_periph</b>	外设USART0/UARTx
<i>USART0</i>	USART0
<b>输入参数{in}</b>	
<b>psc</b>	时钟分频系数（0x00-0xFF）
<b>输出参数{out}</b>	
-	-
<b>返回值</b>	
-	-

例如：

```
/* configure the USART0 peripheral clock prescaler in USART IrDA low-power mode */
```

```
usart_prescaler_config(USART0, 0x00);
```

### 函数 usart\_irda\_lowpower\_config

函数usart\_irda\_lowpower\_config描述见下表：

**表 3-754. 函数 usart\_irda\_lowpower\_config**

<b>函数名称</b>	usart_irda_lowpower_config
<b>函数原型</b>	void usart_irda_lowpower_config(uint32_t usart_periph, uint32_t irlp);
<b>功能描述</b>	配置 USART IrDA 低功耗模式

先决条件	-
被调用函数	-
输入参数{in}	
<b>usart_periph</b>	外设 USART0/UARTx
USART0	USART0
输入参数{in}	
<b>irlp</b>	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 3-755. 函数 usart\_hardware\_flow\_rts\_config

函数名称	usart_hardware_flow_rts_config
函数原型	void usart_hardware_flow_rts_config(uint32_t usart_periph, uint32_t rtsconfig);
功能描述	配置USART RTS硬件控制流
先决条件	-
被调用函数	-
输入参数{in}	
<b>usart_periph</b>	外设USART0/UARTx
USART0/UARTx	x=1,2
输入参数{in}	
<b>rtsconfig</b>	使能/失能RTS
USART_RTS_ENABLE	使能RTS
USART_RTS_DISABLE	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 3-756. 函数 usart\_hardware\_flow\_cts\_config**

函数名称	usart_hardware_flow_cts_config
函数原型	void usart_hardware_flow_cts_config(uint32_t usart_periph, uint32_t ctsconfig);
功能描述	配置USART CTS硬件控制流
先决条件	-
被调用函数	-
输入参数{in}	
usart_periph	外设USART0/UARTx
USART0/UARTx	x=1,2
输入参数{in}	
ctsconfig	使能/失能CTS
USART_CTS_ENABLE	使能CTS
USART_CTS_DISABLE	失能CTS
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* configure USART0 hardware flow control CTS */

usart_hardware_flow_cts_config(USART0, USART_CTS_ENABLE);
```

### 函数 usart\_hardware\_flow\_coherence\_config

函数usart\_hardware\_flow\_coherence\_config描述见下表：

**表 3-757. 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}	
<b>usart_periph</b>	外设USART0/UARTx
<i>USART0/UARTx</i>	x=1,2
输入参数{in}	
<b>hcm</b>	硬件流控制兼容模式
<i>USART_HCM_NONE</i>	nRTS信号与USART_STAT0寄存器中RBNE位相同
<i>USART_HCM_EN</i>	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 3-758. 函数 usart\_rs485\_driver\_enable**

<b>函数名称</b>	usart_rs485_driver_enable
<b>函数原型</b>	void usart_rs485_driver_enable (uint32_t usart_periph);
<b>功能描述</b>	使能USART rs485驱动
<b>先决条件</b>	-
<b>被调用函数</b>	-
输入参数{in}	
<b>usart_periph</b>	外设USART0/UARTx
<i>USART0/UARTx</i>	x=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 3-759. 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USART0/UARTx
USART0/UARTx	x=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 3-760. 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USART0/UARTx
USART0/UARTx	x=1,2
输入参数{in}	
deatime	驱动使能置位时间 (0x00-0x1F)
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* set USART0 driver assertime */
usart_driver_assertime_config(USART0, 0x1F);
```

## 函数 usart\_driver\_deasserttime\_config

函数usart\_driver\_deasserttime\_config描述见下表：

表 3-761. 函数 usart\_driver\_deasserttime\_config

函数名称	usart_driver_deasserttime_config
函数原型	void usart_driver_deasserttime_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USART0/UARTx
USART0/UARTx	x=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 deasserttime */
```

```
usart_driver_deasserttime_config(USART0, 0x1F);
```

## 函数 usart\_depolarity\_config

函数usart\_depolarity\_config描述见下表：

表 3-762. 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USART0/UARTx
USART0/UARTx	x=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 3-763. 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USART0/UARTx
USART0/UARTx	x=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 3-764. 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}	
<b>usart_periph</b>	外设USART0/UARTx
USART0/UARTx	x=1,2
输入参数{in}	
<b>dmacmd</b>	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 3-765. 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}	
<b>usart_periph</b>	外设USART0/UARTx
USART0/UARTx	x=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 3-766. 函数 `usart_reception_error_dma_enable`**

函数名称	<code>usart_reception_error_dma_enable</code>
函数原型	<code>void usart_reception_error_dma_enable(uint32_t usart_periph);</code>
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}	
<code>usart_periph</code>	外设USART0/UARTx
<code>USART0/UARTx</code>	x=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 3-767. 函数 `usart_wakeup_enable`**

函数名称	<code>usart_wakeup_enable</code>
函数原型	<code>void usart_wakeup_enable(uint32_t usart_periph);</code>
功能描述	使能USART唤醒
先决条件	-
被调用函数	-
输入参数{in}	
<code>usart_periph</code>	外设USART0/UARTx
<code>USART0</code>	USART0
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 3-768. 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USART0/UARTx
USART0	USART0
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 3-769. 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USART0/UARTx
USART0	USART0
输入参数{in}	
wum	唤醒模式
USART_WUM_ADD R	WUF在地址匹配时置位
USART_WUM_STA RTB	WUF在检测到起始位时置位
USART_WUM_RBN E	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 3-770. 函数 usart\_receive\_fifo\_enable

函数名称	usart_receive_fifo_enable
函数原型	void usart_receive_fifo_enable(uint32_t usart_periph);
功能描述	使能接收FIFO
先决条件	-
被调用函数	-
输入参数{in}	
usart_periph	外设USART0/UARTx
USART0/UARTx	x=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 3-771. 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USART0/UARTx
USART0/UARTx	x=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 3-772. 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USART0/UARTx
USART0/UARTx	x=1,2
输出参数{out}	
-	-
返回值	
uint8_t	接收FIFO计数器的值

例如：

```
/* read receive FIFO counter number */
```

```
uint8_t temp;
```

```
temp = usart_receive_fifo_counter_number(USART0);
```

### 函数 usart\_flag\_get

函数usart\_flag\_get描述见下表：

**表 3-773. 函数 usart\_flag\_get**

函数名称	usart_flag_get
函数原型	FlagStatus usart_flag_get(uint32_t usart_periph, usart_flag_enum flag);
功能描述	获取USART STAT/CHC/RFCS寄存器标志位
先决条件	-
被调用函数	-
输入参数{in}	

<b>usart_periph</b>	外设USART0/UARTx
USART0/UARTx	x=1,2
<b>输入参数{in}</b>	
<b>flag</b>	USART标志位，参考 <a href="#">表3-704. 枚举类型usart_flag_enum</a> 只能选择一个参数
<b>输出参数{out}</b>	
-	-
<b>返回值</b>	
<b>FlagStatus</b>	SET或RESET

例如：

```
/* get flag USART0 state */
```

```
FlagStatus status;
```

```
status = usart_flag_get(USART0, USART_FLAG_TBE);
```

### 函数 usart\_flag\_clear

函数usart\_flag\_clear描述见下表：

**表 3-774. 函数 usart\_flag\_clear**

<b>函数名称</b>	usart_flag_clear
<b>函数原型</b>	void usart_flag_clear(uint32_t usart_periph, usart_flag_enum flag);
<b>功能描述</b>	清除USART状态寄存器标志位
<b>先决条件</b>	-
<b>被调用函数</b>	-
<b>输入参数{in}</b>	
<b>usart_periph</b>	外设USART0/UARTx
USART0/UARTx	x=1,2
<b>输入参数{in}</b>	
<b>flag</b>	USART标志位，参考 <a href="#">表3-704. 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_CTS F	CTS变化标志
USART_FLAG_RT	接收超时标志
USART_FLAG_EB	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_EPE RR	校验错误超前检测标志
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* clear USART0 flag */
usart_flag_clear(USART0,USART_FLAG_TC);
```

### 函数 usart\_interrupt\_enable

函数usart\_interrupt\_enable描述见下表:

表 3-775. 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USART0/UARTx
USART0/UARTx	x=1,2
输入参数{in}	
interrupt	USART中断USART标志位, 参考 <a href="#">表3-706. 枚举类型usart_interrupt_enum</a> 只能选择一个参数
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* enable USART0 TBE interrupt */
usart_interrupt_enable(USART0, USART_INT_TBE);
```

## 函数 usart\_interrupt\_disable

函数usart\_interrupt\_disable描述见下表：

表 3-776. 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USART0/UARTx
USART0/UARTx	x=1,2
输入参数{in}	
interrupt	USART中断USART标志位，参考 <a href="#">表3-706. 枚举类型usart_interrupt_enum</a> 只能选择一个参数
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* disable USART0 TBE interrupt */
```

```
usart_interrupt_disable(USART0, USART_INT_TBE);
```

## 函数 usart\_interrupt\_flag\_get

函数usart\_interrupt\_flag\_get描述见下表：

表 3-777. 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USART0/UARTx
USART0/UARTx	x=1,2
输入参数{in}	
int_flag	USART中断标志，参考 <a href="#">表3-705. 枚举类型usart_interrupt_flag_enum</a> 只能选择一个参数
输出参数{out}	

-	-
返回值	
FlagStatus	SET或RESET

例如：

```
/* get the USART0 interrupt flag status */
```

```
FlagStatus status;
```

```
status = usart_interrupt_flag_get(USART0, USART_INT_FLAG_RBNE);
```

### 函数 usart\_interrupt\_flag\_clear

函数usart\_interrupt\_flag\_clear描述见下表：

表 3-778. 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USART0/UARTx
USART0/UARTx	x=1,2
输入参数{in}	
Int_flag	USART中断标志，参考 <a href="#">表3-705. 枚举类型usart_interrupt_flag_enum</a> 只能选择一个参数
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* clear the USART0 interrupt flag */
```

```
usart_interrupt_flag_clear(USART0, USART_INT_FLAG_TC);
```

## 3.25. WWDGT

窗口看门狗定时器（WWDGT）用来监测由软件故障导致的系统故障。章节[3.25.1](#)描述了WWDGT的寄存器列表，章节[3.25.2](#)对WWDGT库函数进行说明。

### 3.25.1. 外设寄存器说明

WWDGT寄存器列表如下表所示：

**表 3-779. WWDGT 寄存器**

寄存器名称	寄存器描述
WWDGT_CTL	控制寄存器
WWDGT_CFG	配置寄存器
WWDGT_STAT	状态寄存器

### 3.25.2. 外设库函数说明

WWDGT库函数列表如下表所示：

**表 3-780. 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 3-781. 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 3-782. 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 3-783. 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(0x7F);
```

## 函数 wwdgt\_config

函数wwdgt\_config描述见下表:

表 3-784. 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 3-785. 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 3-786. 函数 wwdgt\_flag\_get**

函数名称	wwdgt_flag_get
函数原型	FlagStatus wwdgt_flag_get(void);
功能描述	检查WWDGT提前唤醒中断标志位是否置位
先决条件	-
被调用函数	-
输入参数{in}	
-	-
输出参数{out}	
-	-
返回值	
<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 3-787. 函数 wwdgt\_flag\_clear**

函数名称	wwdgt_flag_clear
函数原型	void wwdgt_flag_clear(void);
功能描述	清除WWDGT提前唤醒中断标志位状态
先决条件	-
被调用函数	-

输入参数{in}	
-	-
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* clear early wakeup interrupt state of WWDGT */
```

```
wwdgt_flag_clear();
```

## 4. 版本历史

表 4-1. 版本历史

版本号.	说明	日期
1.0	初稿发布	2023 年 7 月 20 日
1.1	添加 SES 工程相关描述	2024 年 1 月 16 日
1.2	1. 修改 <b>3.14. I2C</b> 章节 i2c_transfer_byte_number_config 形参。 2. 删除 <b>3.14. I2C</b> 章节 void i2c_nack_disable(uint32_t i2c_periph)接口。	2025 年 8 月 9 日
1.3	1. <b>3.11. FWDGT</b> 章节增加 fwdgt_config 形参。 2. <b>3.17. QSPI</b> 章节增加函数 qspi_data_length_config。 3. <b>3.22. TIMER</b> 章节增加函数 timer_channel_input_remap_config。	2026 年 2 月 4 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 according to the laws of the People's Republic of China and other applicable laws. The Company reserves all rights under such laws and no Intellectual Property Rights are transferred (either wholly or partially) or licensed by the Company (either expressly or impliedly) herein. 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.

To the maximum extent permitted by applicable law, the Company makes no representations or warranties of any kind, express or implied, with regard to the merchantability and the fitness for a particular purpose of the Product, nor does the Company assume any liability arising out of the application or use of any Product. Any information provided in this document is provided only for reference purposes. It is the sole responsibility of the user of this document to determine whether the Product is suitable and fit for its applications and products planned, and properly design, program, and test the functionality and safety of its applications and products planned using the Product. The Product is designed, developed, and/or manufactured for ordinary business, industrial, personal, and/or household applications only, and the Product is not designed or intended for use in (i) safety critical applications such as weapons systems, nuclear facilities, atomic energy controller, combustion controller, aeronautic or aerospace applications, traffic signal instruments, pollution control or hazardous substance management; (ii) life-support systems, other medical equipment or systems (including life support equipment and surgical implants); (iii) automotive applications or environments, including but not limited to applications for active and passive safety of automobiles (regardless of front market or aftermarket), for example, EPS, braking, ADAS (camera/fusion), EMS, TCU, BMS, BSG, TPMS, Airbag, Suspension, DMS, ICMS, Domain, ESC, DCDC, e-clutch, advanced-lighting, etc.. Automobile herein means a vehicle propelled by a self-contained motor, engine or the like, such as, without limitation, cars, trucks, motorcycles, electric cars, and other transportation devices; and/or (iv) other uses where the failure of the device or the Product can reasonably be expected to result in personal injury, death, or severe property or environmental damage (collectively "Unintended Uses"). Customers shall take any and all actions to ensure the Product meets the applicable laws and regulations. The Company is not liable for, in whole or in part, and customers shall hereby 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 Product. Customers shall indemnify and hold the Company, and its officers, employees, subsidiaries, affiliates 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 Product.

Information in this document is provided solely in connection with the Product. The Company reserves the right to make changes, corrections, modifications or improvements to this document and the Product described herein at any time without notice. The Company shall have no responsibility whatsoever for conflicts or incompatibilities arising from future changes to them. Information in this document supersedes and replaces information previously supplied in any prior versions of this document.