

**GigaDevice Semiconductor Inc.**

**GD32C2x1**

**Arm<sup>®</sup> Cortex<sup>®</sup>-M23 32-bit MCU**

**固件库  
使用指南**

1.0 版本

(2025 年 5 月)

# 目录

目录 .....	1
图索引 .....	4
表索引 .....	5
<b>1. 介绍 .....</b>	<b>20</b>
<b>1.1. 文档和固件库规则 .....</b>	<b>20</b>
1.1.1. 外设缩写 .....	20
1.1.2. 命名规则 .....	21
<b>2. 固件库概述 .....</b>	<b>22</b>
<b>2.1. 文件组织结构 .....</b>	<b>22</b>
2.1.1. Examples 文件夹 .....	22
2.1.2. Firmware 文件夹 .....	23
2.1.3. Template 文件夹 .....	23
2.1.4. Utilities 文件夹 .....	26
<b>2.2. 固件库文件描述 .....</b>	<b>26</b>
<b>3. 外设固件库 .....</b>	<b>27</b>
<b>3.1. 外设固件库概述 .....</b>	<b>27</b>
<b>3.2. ADC .....</b>	<b>27</b>
3.2.1. 外设寄存器描述 .....	27
3.2.2. 外设库函数说明 .....	28
<b>3.3. CMP .....</b>	<b>54</b>
3.3.1. 外设寄存器说明 .....	55
3.3.2. 外设库函数说明 .....	55
<b>3.4. CRC .....</b>	<b>64</b>
3.4.1. 外设寄存器说明 .....	64
3.4.2. 外设库函数说明 .....	64
<b>3.5. DBG .....</b>	<b>72</b>
3.5.1. 外设寄存器说明 .....	72
3.5.2. 外设库函数说明 .....	72
<b>3.6. DMA/DMAMUX .....</b>	<b>77</b>
3.6.1. 外设寄存器说明 .....	77
3.6.2. 外设库函数说明 .....	78
<b>3.7. EXTI .....</b>	<b>117</b>
3.7.1. 外设寄存器说明 .....	117

3.7.2.	外设库函数说明 .....	117
<b>3.8.</b>	<b>FMC.....</b>	<b>125</b>
3.8.1.	外设寄存器说明 .....	125
3.8.2.	外设库函数说明 .....	125
<b>3.9.</b>	<b>FWDGT .....</b>	<b>153</b>
3.9.1.	外设寄存器说明 .....	154
3.9.2.	外设库函数说明 .....	154
<b>3.10.</b>	<b>GPIO .....</b>	<b>159</b>
3.10.1.	外设寄存器说明 .....	159
3.10.2.	外设库函数说明 .....	160
<b>3.11.</b>	<b>I2C .....</b>	<b>170</b>
3.11.1.	外设寄存器说明 .....	170
3.11.2.	外设库函数说明 .....	171
<b>3.12.</b>	<b>MISC .....</b>	<b>210</b>
3.12.1.	外设寄存器说明 .....	210
3.12.2.	外设库函数说明 .....	210
<b>3.13.</b>	<b>PMU.....</b>	<b>216</b>
3.13.1.	外设寄存器说明 .....	216
3.13.2.	外设库函数说明 .....	216
<b>3.14.</b>	<b>RCU.....</b>	<b>226</b>
3.14.1.	外设寄存器说明 .....	226
3.14.2.	外设库函数说明 .....	227
<b>3.15.</b>	<b>RTC .....</b>	<b>226</b>
3.15.1.	外设寄存器描述 .....	265
3.15.2.	外设库函数描述 .....	266
<b>3.16.</b>	<b>SPI.....</b>	<b>286</b>
3.16.1.	外设寄存器说明 .....	286
3.16.2.	外设库函数说明 .....	286
<b>3.17.</b>	<b>SYSCFG .....</b>	<b>316</b>
3.17.1.	外设寄存器说明 .....	316
3.17.2.	外设库函数说明 .....	316
<b>3.18.</b>	<b>TIMER .....</b>	<b>328</b>
3.18.1.	外设寄存器说明 .....	328
3.18.2.	外设库函数说明 .....	329
<b>3.19.</b>	<b>USART .....</b>	<b>390</b>
3.19.1.	外设寄存器说明 .....	390
3.19.2.	外设库函数说明 .....	390
<b>3.20.</b>	<b>WWDGT .....</b>	<b>439</b>
3.20.1.	外设寄存器说明 .....	439

---

3.20.2. 外设库函数说明 .....	439
<b>4. 版本历史 .....</b>	<b>446</b>

## 图索引

图 2-1. GD32C2x1 固件库文件组织结构 .....	22
图 2-2. 选择外设例程文件 .....	24
图 2-3. 拷贝外设例程文件 .....	24
图 2-4. 打开工程文件.....	25
图 2-5. 配置工程文件.....	25
图 2-6. 编译调试下载.....	26

## 表索引

表 1-1. 外设缩写 .....	20
表 2-1. 固件函数库文件描述 .....	26
表 3-1. 外设固件库函数描述格式 .....	27
表 3-2. ADC 寄存器 .....	27
表 3-3. ADC 库函数 .....	28
表 3-4. 函数 adc_deinit .....	29
表 3-5. 函数 adc_enable .....	29
表 3-6. 函数 adc_disable .....	30
表 3-7. 函数 adc_dma_mode_enable .....	30
表 3-8. 函数 adc_dma_mode_disable .....	31
表 3-9. 函数 adc_discontinuous_mode_config .....	31
表 3-10. 函数 adc_special_function_config .....	32
表 3-11. 函数 adc_channel_13_to_14 .....	33
表 3-12. 函数 adc_data_alignment_config .....	34
表 3-13. 函数 adc_channel_length_config .....	34
表 3-14. 函数 adc_routine_channel_config .....	35
表 3-15. 函数 adc_inserted_channel_config .....	36
表 3-16. 函数 adc_inserted_channel_offset_config .....	37
表 3-17. 函数 adc_external_trigger_config .....	37
表 3-18. 函数 adc_external_trigger_source_config .....	38
表 3-19. 函数 adc_software_trigger_enable .....	39
表 3-20. 函数 adc_routine_data_read .....	40
表 3-21. 函数 adc_inserted_data_read .....	41
表 3-22. 函数 adc_watchdog_single_channel_enable .....	41
表 3-23. 函数 adc_watchdog0_sequence_channel_enable .....	42
表 3-24. 函数 adc_watchdog0_disable .....	42
表 3-25. 函数 adc_watchdog1_channel_config .....	43
表 3-26. 函数 adc_watchdog2_channel_config .....	43
表 3-27. 函数 adc_watchdog1_disable .....	44
表 3-28. 函数 adc_watchdog2_disable .....	45
表 3-29. 函数 adc_watchdog0_threshold_config .....	45
表 3-30. 函数 adc_watchdog1_threshold_config .....	46
表 3-31. 函数 adc_watchdog2_threshold_config .....	46
表 3-32. 函数 adc_resolution_config .....	47
表 3-33. 函数 adc_oversample_mode_config .....	48
表 3-34. 函数 adc_oversample_mode_enable .....	49
表 3-35. 函数 adc_oversample_mode_disable .....	50
表 3-36. 函数 adc_flag_get .....	50
表 3-37. 函数 adc_flag_clear .....	51
表 3-38. 函数 adc_interrupt_enable .....	52

表 3-39. 函数 adc_interrupt_disable.....	52
表 3-40. 函数 adc_interrupt_flag_get .....	53
表 3-41. 函数 adc_interrupt_flag_clear .....	54
表 3-42. CMP 寄存器.....	55
表 3-43. CMP 库函数.....	55
表 3-44. 枚举类型 cmp_enum .....	55
表 3-45. 函数 cmp_deinit .....	55
表 3-46. 函数 cmp_mode_init .....	56
表 3-47. 函数 cmp_output_init.....	57
表 3-48. 函数 cmp_blanking_init.....	58
表 3-49. 函数 cmp_enable .....	59
表 3-50. 函数 cmp_disable .....	59
表 3-51. 函数 cmp_switch_enable .....	60
表 3-52. 函数 cmp_switch_disable .....	60
表 3-53. 函数 cmp_lock_enable .....	61
表 3-54. 函数 cmp_voltage_scaler_enable.....	61
表 3-55. 函数 cmp_voltage_scaler_disable.....	62
表 3-56. 函数 cmp_scaler_bridge_enable .....	62
表 3-57. 函数 cmp_scaler_bridge_disable .....	63
表 3-58. 函数 cmp_output_level_get .....	63
表 3-59. CRC 寄存器.....	64
表 3-60. CRC 库函数.....	64
表 3-61. 函数 crc_deinit .....	65
表 3-62. 函数 crc_init_data_register_write.....	65
表 3-63. 函数 crc_data_register_read .....	66
表 3-64. 函数 crc_free_data_register_read .....	66
表 3-65. 函数 crc_free_data_register_write .....	67
表 3-66. 函数 crc_reverse_output_data_disable.....	67
表 3-67. 函数 crc_reverse_output_data_enable.....	68
表 3-68. 函数 crc_input_data_reverse_config .....	68
表 3-69. 函数 crc_data_register_reset .....	69
表 3-70. 函数 crc_polynomial_size_set .....	69
表 3-71. 函数 crc_polynomial_set.....	70
表 3-72. 函数 crc_single_data_calculate .....	70
表 3-73. 函数 crc_block_data_calculate .....	71
表 3-74. DBG 寄存器.....	72
表 3-75. DBG 库函数.....	72
表 3-76. 枚举类型 dbg_periph_enum.....	73
表 3-77. 函数 dbg_deinit.....	73
表 3-78. 函数 dbg_id_get.....	73
表 3-79. 函数 dbg_low_power_enable .....	74
表 3-80. 函数 dbg_low_power_disable .....	75
表 3-81. 函数 dbg_periph_enable .....	75
表 3-82. 函数 dbg_periph_disable .....	76

表 3-83. DMA 寄存器 .....	77
表 3-84. DMAMUX 寄存器 .....	77
表 3-85. DMA 库函数 .....	78
表 3-86. DMAMUX 库函数 .....	78
表 3-87. 结构体 dma_parameter_struct .....	79
表 3-88. 结构体 dmamux_sync_parameter_struct .....	80
表 3-89. 结构体 dmamux_gen_parameter_struct .....	80
表 3-90. 枚举 dmamux_interrupt_enum .....	80
表 3-91. 枚举 dmamux_flag_enum .....	80
表 3-92. 枚举 dmamux_interrupt_flag_enum .....	81
表 3-93. 枚举 dma_channel_enum .....	81
表 3-94. 枚举 dmamux_multiplexer_channel_enum .....	81
表 3-95. 枚举 dmamux_generator_channel_enum .....	82
表 3-96. 函数 dma_deinit .....	82
表 3-97. 函数 dma_struct_para_init .....	82
表 3-98. 函数 dma_init .....	83
表 3-99. 函数 dma_circulation_enable .....	84
表 3-100. 函数 dma_circulation_disable .....	84
表 3-101. 函数 dma_memory_to_memory_enable .....	85
表 3-102. 函数 dma_memory_to_memory_disable .....	85
表 3-103. 函数 dma_channel_enable .....	86
表 3-104. 函数 dma_channel_disable .....	86
表 3-105. 函数 dma_periph_address_config .....	87
表 3-106. 函数 dma_memory_address_config .....	87
表 3-107. 函数 dma_transfer_number_config .....	88
表 3-108. 函数 dma_transfer_number_get .....	89
表 3-109. 函数 dma_priority_config .....	89
表 3-110. 函数 dma_memory_width_config .....	90
表 3-111. 函数 dma_periph_width_config .....	91
表 3-112. 函数 dma_memory_increase_enable .....	91
表 3-113. 函数 dma_memory_increase_disable .....	92
表 3-114. 函数 dma_periph_increase_enable .....	92
表 3-115. 函数 dma_periph_increase_disable .....	93
表 3-116. 函数 dma_transfer_direction_config .....	93
表 3-117. 函数 dma_flag_get .....	94
表 3-118. 函数 dma_flag_clear .....	95
表 3-119. 函数 dma_interrupt_enable .....	95
表 3-120. 函数 dma_interrupt_disable .....	96
表 3-121. 函数 dma_interrupt_flag_get .....	96
表 3-122. 函数 dma_interrupt_flag_clear .....	97
表 3-123. 函数 dmamux_sync_struct_para_init .....	98
表 3-124. 函数 dmamux_synchronization_init .....	98
表 3-125. 函数 dmamux_synchronization_enable .....	99
表 3-126. 函数 dmamux_synchronization_disable .....	100



表 3-127. 函数 dmamux_event_generation_enable.....	100
表 3-128. 函数 dmamux_event_generation_disable.....	101
表 3-129. 函数 dmamux_gen_struct_para_init.....	101
表 3-130. 函数 dmamux_request_generator_init .....	102
表 3-131. 函数 dmamux_request_generator_channel_enable .....	103
表 3-132. 函数 dmamux_request_generator_channel_disable .....	103
表 3-133. 函数 dmamux_synchronization_polarity_config .....	104
表 3-134. 函数 dmamux_request_forward_number_config .....	105
表 3-135. 函数 dmamux_sync_id_config.....	105
表 3-136. 函数 dmamux_request_id_config.....	107
表 3-137. 函数 dma_interrupt_disable .....	109
表 3-138. 函数 dmamux_request_generate_number_config .....	110
表 3-139. 函数 dmamux_trigger_id_config .....	110
表 3-140. 函数 dmamux_flag_get.....	112
表 3-141. 函数 dmamux_flag_clear.....	113
表 3-142. 函数 dmamux_interrupt_enable.....	114
表 3-143. 函数 dmamux_interrupt_disable.....	114
表 3-144. 函数 dmamux_interrupt_flag_get .....	115
表 3-145. 函数 dmamux_interrupt_flag_clear.....	116
表 3-146. EXTI 寄存器.....	117
表 3-147. EXTI 库函数.....	117
表 3-148. 枚举类型 exti_line_enum.....	118
表 3-149. 枚举类型 exti_mode_enum .....	118
表 3-150. 枚举类型 exti_trig_type_enum .....	119
表 3-151. 函数 exti_deinit .....	119
表 3-152. 函数 exti_init.....	119
表 3-153. 函数 exti_interrupt_enable.....	120
表 3-154. 函数 exti_interrupt_disable.....	120
表 3-155. 函数 exti_event_enable .....	121
表 3-156. 函数 exti_event_disable .....	121
表 3-157. 函数 exti_software_interrupt_enable .....	122
表 3-158. 函数 exti_software_interrupt_disable .....	122
表 3-159. 函数 exti_flag_get.....	123
表 3-160. 函数 exti_flag_clear.....	123
表 3-161. 函数 exti_interrupt_flag_get .....	124
表 3-162. 函数 exti_interrupt_flag_clear .....	124
表 3-163. FMC 寄存器.....	125
表 3-164. FMC 库函数.....	125
表 3-165. 枚举类型 fmc_state_enum .....	127
表 3-166. 枚举类型 ob_user_data_extract_info_enum .....	127
表 3-167. 函数 fmc_unlock.....	127
表 3-168. 函数 fmc_lock.....	128
表 3-169. 函数 fmc_wscnt_set.....	128
表 3-170. 函数 fmc_main_flash_empty_check_stat_modify .....	129

表 3-171. 函数 fmc_wscnt_set.....	130
表 3-172. 函数 fmc_prefetch_enable .....	130
表 3-173. 函数 fmc_prefetch_disable .....	131
表 3-174. 函数 fmc_icache_enable .....	131
表 3-175. 函数 fmc_icache_disable .....	132
表 3-176. 函数 fmc_icache_reset_enable .....	132
表 3-177. 函数 fmc_page_erase .....	133
表 3-178. 函数 fmc_mass_erase .....	133
表 3-179. 函数 fmc_doubleword_program .....	134
表 3-180. 函数 fmc_doubleword_program .....	134
表 3-181. 函数 fmc_debugger_enable.....	135
表 3-182. 函数 fmc_debugger_disable.....	136
表 3-183. 函数 fmc_scr_area_enable .....	136
表 3-184. 函数 ob_unlock.....	137
表 3-185. 函数 ob_lock.....	137
表 3-186. 函数 ob_reload.....	138
表 3-187. 函数 ob_user_write .....	139
表 3-188. 函数 ob_security_protection_level_config .....	141
表 3-189. 函数 ob_dcrp_area_config.....	142
表 3-190. 函数 ob_write_protection_config .....	143
表 3-191. 函数 ob_scr_area_config .....	143
表 3-192. 函数 ob_boot_lock_config.....	144
表 3-193. 函数 ob_user_get .....	145
表 3-194. 函数 ob_security_protection_level_get.....	145
表 3-195. 函数 ob_dcrp_area_get .....	146
表 3-196. 函数 ob_write_protection_area_get .....	147
表 3-197. 函数 ob_scr_area_size_get.....	147
表 3-198. 函数 ob_boot_lock_get.....	148
表 3-199. 函数 fmc_flag_get.....	148
表 3-200. 函数 fmc_flag_clear .....	149
表 3-201. 函数 fmc_interrupt_enable .....	150
表 3-202. 函数 fmc_interrupt_disable .....	150
表 3-203. 函数 fmc_interrupt_flag_get.....	151
表 3-204. 函数 fmc_interrupt_flag_clear.....	152
表 3-205. FWDGT 寄存器.....	154
表 3-206. FWDGT 库函数.....	154
表 3-207. 函数 fwdgt_write_enable.....	154
表 3-208. 函数 fwdgt_write_disable.....	155
表 3-209. 函数 fwdgt_enable.....	155
表 3-210. 函数 fwdgt_prescaler_value_config.....	156
表 3-211. 函数 fwdgt_reload_value_config .....	156
表 3-212. 函数 fwdgt_window_value_config.....	157
表 3-213. 函数 fwdgt_counter_reload .....	157
表 3-214. 函数 fwdgt_config .....	158

表 3-215. 函数 fwdgt_flag_get .....	159
表 3-216. GPIO 寄存器.....	159
表 3-217. GPIO 库函数.....	160
表 3-218. 函数 gpio_deinit.....	160
表 3-219. 函数 gpio_mode_set .....	161
表 3-220. 函数 gpio_output_options_set.....	162
表 3-221. 函数 gpio_bit_set.....	163
表 3-222. 函数 gpio_bit_reset.....	163
表 3-223. 函数 gpio_bit_write .....	164
表 3-224. 函数 gpio_port_write .....	165
表 3-225. 函数 gpio_input_bit_get .....	165
表 3-226. 函数 gpio_input_port_get .....	166
表 3-227. 函数 gpio_output_bit_get.....	166
表 3-228. 函数 gpio_output_port_get.....	167
表 3-229. 函数 gpio_af_set.....	168
表 3-230. 函数 gpio_pin_lock .....	169
表 3-231. 函数 gpio_bit_toggle.....	169
表 3-232. 函数 gpio_port_toggle .....	170
表 3-233. I2C 寄存器 .....	170
表 3-234. I2C 库函数 .....	171
表 3-235. 枚举类型 i2c_interrupt_flag_enum .....	173
表 3-236. 函数 i2c_deinit .....	173
表 3-237. 函数 i2c_timing_config.....	174
表 3-238. 函数 i2c_digital_noise_filter_config.....	174
表 3-239. 函数 i2c_analog_noise_filter_enable .....	175
表 3-240. 函数 i2c_analog_noise_filter_disable.....	176
表 3-241. 函数 i2c_master_clock_config.....	176
表 3-242. 函数 i2c_master_addressing .....	177
表 3-243. 函数 i2c_address10_header_enable .....	178
表 3-244. 函数 i2c_address10_header_disable .....	178
表 3-245. 函数 i2c_address10_enable.....	179
表 3-246. 函数 i2c_address10_disable.....	179
表 3-247. 函数 i2c_automatic_end_enable.....	180
表 3-248. 函数 i2c_automatic_end_disable.....	180
表 3-249. 函数 i2c_slave_response_to_gcall_enable.....	181
表 3-250. 函数 i2c_slave_response_to_gcall_disable.....	181
表 3-251. 函数 i2c_stretch_scl_low_enable .....	182
表 3-252. 函数 i2c_stretch_scl_low_disable .....	182
表 3-253. 函数 i2c_address_config.....	183
表 3-254. 函数 i2c_address_bit_compare_config.....	183
表 3-255. 函数 i2c_address_disable .....	184
表 3-256. 函数 i2c_second_address_config .....	185
表 3-257. 函数 i2c_second_address_disable.....	186
表 3-258. 函数 i2c_receved_address_get.....	186

表 3-259. 函数 i2c_slave_byte_control_enable .....	187
表 3-260. 函数 i2c_slave_byte_control_disable .....	187
表 3-261. 函数 i2c_nack_enable .....	188
表 3-262. 函数 i2c_nack_disable .....	188
表 3-263. 函数 i2c_wakeup_from_deepsleep_enable .....	189
表 3-264. 函数 i2c_wakeup_from_deepsleep_disable .....	189
表 3-265. 函数 i2c_enable .....	190
表 3-266. 函数 i2c_disable .....	190
表 3-267. 函数 i2c_start_on_bus .....	191
表 3-268. 函数 i2c_stop_on_bus .....	191
表 3-269. 函数 i2c_data_transmit .....	192
表 3-270. 函数 i2c_data_receive .....	192
表 3-271. 函数 i2c_reload_enable .....	193
表 3-272. 函数 i2c_reload_disable .....	193
表 3-273. 函数 i2c_transfer_byte_number_config .....	194
表 3-274. 函数 i2c_dma_enable .....	194
表 3-275. 函数 i2c_dma_disable .....	195
表 3-276. 函数 i2c_pec_transfer .....	196
表 3-277. 函数 i2c_pec_enable .....	196
表 3-278. 函数 i2c_pec_disable .....	197
表 3-279. 函数 i2c_pec_value_get .....	197
表 3-280. 函数 i2c_smbus_alert_enable .....	198
表 3-281. 函数 i2c_smbus_alert_disable .....	198
表 3-282. 函数 i2c_smbus_default_addr_enable .....	199
表 3-283. 函数 i2c_smbus_default_addr_disable .....	199
表 3-284. 函数 i2c_smbus_host_addr_enable .....	200
表 3-285. 函数 i2c_smbus_host_addr_disable .....	200
表 3-286. 函数 i2c_extented_clock_timeout_enable .....	201
表 3-287. 函数 i2c_extented_clock_timeout_disable .....	201
表 3-288. 函数 i2c_clock_timeout_enable .....	202
表 3-289. 函数 i2c_clock_timeout_disable .....	202
表 3-290. 函数 i2c_bus_timeout_b_config .....	203
表 3-291. 函数 i2c_bus_timeout_a_config .....	203
表 3-292. 函数 i2c_idle_clock_timeout_config .....	204
表 3-293. 函数 i2c_flag_get .....	204
表 3-294. 函数 i2c_flag_clear .....	205
表 3-295. 函数 i2c_interrupt_enable .....	206
表 3-296. 函数 i2c_interrupt_disable .....	207
表 3-297. 函数 i2c_interrupt_flag_get .....	208
表 3-298. 函数 i2c_interrupt_flag_clear .....	209
表 3-299. NVIC 寄存器 .....	210
表 3-300. SysTick 寄存器 .....	210
表 3-301. MISC 库函数 .....	210
表 3-302. 枚举类型 IRQn_Type .....	211

表 3-303. 函数 nvic_irq_enable .....	212
表 3-304. 函数 nvic_irq_disable .....	212
表 3-305. 函数 nvic_vector_table_set .....	213
表 3-306. 函数 nvic_system_reset .....	213
表 3-307. 函数 system_lowpower_set .....	214
表 3-308. 函数 system_lowpower_reset .....	214
表 3-309. 函数 systick_clksource_set .....	215
表 3-310. PMU 寄存器 .....	216
表 3-311. PMU 库函数 .....	216
表 3-312. 函数 pmu_deinit .....	217
表 3-313. 函数 pmu_deepsleep_voltage_select .....	217
表 3-314. 函数 pmu_low_power_ldo_enable .....	218
表 3-315. 函数 pmu_low_power_ldo_disable .....	218
表 3-316. 函数 pmu_to_sleepmode .....	219
表 3-317. 函数 pmu_to_deepsleepmode .....	219
表 3-318. 函数 pmu_to_standbymode .....	220
表 3-319. 函数 pmu_wakeup_pin_enable .....	220
表 3-320. 函数 pmu_wakeup_pin_disable .....	221
表 3-321. 函数 pmu_backup_write_enable .....	222
表 3-322. 函数 pmu_backup_write_disable .....	222
表 3-323. 函数 pmu_eflash_run_power_config .....	223
表 3-324. 函数 pmu_eflash_deepsleep_power_config .....	223
表 3-325. 函数 pmu_eflash_wakeup_time_config .....	224
表 3-326. 函数 pmu_deepsleep_wait_time_config .....	224
表 3-327. 函数 pmu_flag_get .....	225
表 3-328. 函数 pmu_flag_clear .....	225
表 3-329. RCU 寄存器 .....	226
表 3-330. RCU 库函数 .....	227
表 3-331. 枚举类型 rcu_periph_enum .....	228
表 3-332. 枚举类型 rcu_periph_sleep_enum .....	错误!未定义书签。
表 3-333. 枚举类型 rcu_periph_reset_enum .....	错误!未定义书签。
表 3-334. 枚举类型 rcu_flag_enum .....	错误!未定义书签。
表 3-335. 枚举类型 rcu_int_flag_enum .....	错误!未定义书签。
表 3-336. 枚举类型 rcu_int_flag_clear_enum .....	错误!未定义书签。
表 3-337. 枚举类型 rcu_int_enum .....	错误!未定义书签。
表 3-338. 枚举类型 rcu_osci_type_enum .....	错误!未定义书签。
表 3-339. 枚举类型 rcu_clock_freq_enum .....	错误!未定义书签。
表 3-340. 枚举类型 usart_idx_enum .....	错误!未定义书签。
表 3-341. 枚举类型 i2c_idx_enum .....	错误!未定义书签。
表 3-342. 函数 rcu_deinit .....	错误!未定义书签。
表 3-343. 函数 rcu_periph_clock_enable .....	错误!未定义书签。
表 3-344. 函数 rcu_periph_clock_disable .....	错误!未定义书签。
表 3-345. 函数 rcu_periph_clock_sleep_enable .....	错误!未定义书签。
表 3-346. 函数 rcu_periph_clock_sleep_disable .....	错误!未定义书签。

表 3-347. 函数 rcu_periph_reset_enable .....	错误!未定义书签。
表 3-348. 函数 rcu_periph_reset_disable .....	错误!未定义书签。
表 3-349. 函数 rcu_bkp_reset_enable .....	错误!未定义书签。
表 3-350. 函数 rcu_bkp_reset_disable .....	错误!未定义书签。
表 3-351. 函数 rcu_system_clock_source_config .....	错误!未定义书签。
表 3-352. 函数 rcu_system_clock_source_get.....	错误!未定义书签。
表 3-353. 函数 rcu_ahb_clock_config .....	错误!未定义书签。
表 3-354. 函数 rcu_apb_clock_config .....	错误!未定义书签。
表 3-355. 函数 rcu_adc_clock_config .....	错误!未定义书签。
表 3-356. 函数 rcu_ckout0_config .....	错误!未定义书签。
表 3-357. 函数 rcu_ckout1_config .....	错误!未定义书签。
表 3-358. 函数 rcu_lsckout_enable.....	错误!未定义书签。
表 3-359. 函数 rcu_lsckout_disable.....	错误!未定义书签。
表 3-360. 函数 rcu_lsckout_config .....	错误!未定义书签。
表 3-361. 函数 rcu_usart_clock_config .....	错误!未定义书签。
表 3-362. 函数 rcu_i2c_clock_config .....	错误!未定义书签。
表 3-363. 函数 rcu_irc48mdiv_sys_clock_config.....	错误!未定义书签。
表 3-364. 函数 rcu_irc48mdiv_per_clock_config .....	错误!未定义书签。
表 3-365. 函数 rcu_rtc_clock_config .....	错误!未定义书签。
表 3-366. 函数 rcu_lxtal_drive_capability_config .....	错误!未定义书签。
表 3-367. 函数 rcu_osci_stab_wait.....	错误!未定义书签。
表 3-368. 函数 rcu_osci_on.....	错误!未定义书签。
表 3-369. 函数 rcu_osci_off .....	错误!未定义书签。
表 3-370. 函数 rcu_osci_bypass_mode_enable .....	错误!未定义书签。
表 3-371. 函数 rcu_osci_bypass_mode_disable .....	错误!未定义书签。
表 3-372. 函数 rcu_irc16m_adjust_value_set.....	错误!未定义书签。
表 3-373. 函数 rcu_lxtal_stab_reset_enable .....	错误!未定义书签。
表 3-374. 函数 rcu_lxtal_stab_reset_disable .....	错误!未定义书签。
表 3-375. 函数 rcu_hxtal_clock_monitor_enable .....	错误!未定义书签。
表 3-376. 函数 rcu_hxtal_clock_monitor_disable .....	错误!未定义书签。
表 3-377. 函数 rcu_lxtal_clock_monitor_enable .....	错误!未定义书签。
表 3-378. 函数 rcu_lxtal_clock_monitor_disable .....	错误!未定义书签。
表 3-379. 函数 rcu_clock_freq_get .....	错误!未定义书签。
表 3-380. 函数 rcu_flag_get .....	错误!未定义书签。
表 3-381. 函数 rcu_all_reset_flag_clear.....	错误!未定义书签。
表 3-382. 函数 rcu_interrupt_enable .....	错误!未定义书签。
表 3-383. 函数 rcu_interrupt_disable .....	错误!未定义书签。
表 3-384. 函数 rcu_interrupt_flag_get.....	错误!未定义书签。
表 3-385. 函数 rcu_interrupt_flag_clear.....	错误!未定义书签。
表 3-386. RTC 寄存器 .....	265
表 3-387. RTC 库函数 .....	266
表 3-388. 结构体 rtc_parameter_struct .....	267
表 3-389. 结构体 rtc_alarm_struct .....	267
表 3-390. 结构体 rtc_timestamp_struct.....	268



表 3-391. 函数 rtc_bypass_shadow_enable.....	268
表 3-392. 函数 rtc_bypass_shadow_disable.....	268
表 3-393. 函数 rtc_register_sync_wait.....	269
表 3-394. 函数 rtc_alarm_enable.....	269
表 3-395. 函数 rtc_alarm_disable.....	270
表 3-396. 函数 rtc_alarm_config .....	270
表 3-397. 函数 rtc_alarm_subsecond_config .....	271
表 3-398. 函数 rtc_alarm_subsecond_get.....	272
表 3-399. 函数 rtc_alarm_get .....	272
表 3-400. 函数 rtc_init_mode_enter.....	273
表 3-401. 函数 rtc_init .....	273
表 3-402. 函数 rtc_init_mode_exit .....	274
表 3-403. 函数 rtc_current_time_get .....	275
表 3-404. 函数 rtc_subsecond_get .....	275
表 3-405. 函数 rtc_deinit.....	276
表 3-406. 函数 rtc_hour_adjust .....	276
表 3-407. 函数 rtc_second_adjust .....	277
表 3-408. 函数 rtc_refclock_detection_enable .....	277
表 3-409. 函数 rtc_refclock_detection_disable .....	278
表 3-410. 函数 rtc_smooth_calibration_config.....	278
表 3-411. 函数 rtc_timestamp_enable .....	279
表 3-412. 函数 rtc_timestamp_disable .....	280
表 3-413. 函数 rtc_timestamp_get .....	280
表 3-414. 函数 rtc_timestamp_subsecond_get.....	281
表 3-415. 函数 rtc_calibration_output_config .....	281
表 3-416. 函数 rtc_alarm_output_config.....	282
表 3-417. 函数 rtc_output_pin_select.....	283
表 3-418. 函数 rtc_interrupt_enable .....	283
表 3-419. 函数 rtc_interrupt_disable .....	284
表 3-420. 函数 rtc_flag_get .....	284
表 3-421. 函数 rtc_flag_clear .....	285
表 3-422. SPI/I2S 寄存器 .....	286
表 3-423. SPI/I2S 库函数 .....	286
表 3-424. 结构体 spi_parameter_struct.....	287
表 3-425. 函数 spi_i2s_deinit.....	288
表 3-426. 函数 spi_struct_para_init.....	288
表 3-427. 函数 spi_init.....	289
表 3-428. 函数 spi_enable .....	290
表 3-429. 函数 spi_disable .....	290
表 3-430. 函数 i2s_init.....	291
表 3-431. 函数 i2s_psc_config.....	292
表 3-432. 函数 i2s_enable.....	293
表 3-433. 函数 i2s_disable.....	294
表 3-434. 函数 spi_nss_output_enable .....	294

表 3-435. 函数 spi_nss_output_disable .....	295
表 3-436. 函数 spi_nss_internal_high .....	295
表 3-437. 函数 spi_nss_internal_low .....	296
表 3-438. 函数 spi_dma_enable .....	296
表 3-439. 函数 spi_dma_disable .....	297
表 3-440. 函数 spi_transmit_odd_config .....	298
表 3-441. 函数 spi_receive_odd_config .....	298
表 3-442. 函数 spi_i2s_data_frame_format_config .....	299
表 3-443. 函数 spi_fifo_access_size_config .....	299
表 3-444. 函数 spi_bidirectional_transfer_config .....	300
表 3-445. 函数 spi_i2s_data_transmit .....	301
表 3-446. 函数 spi_i2s_data_receive .....	301
表 3-447. 函数 spi_crc_polynomial_set .....	302
表 3-448. 函数 spi_crc_polynomial_get .....	302
表 3-449. 函数 spi_crc_length_set .....	303
表 3-450. 函数 spi_crc_on .....	304
表 3-451. 函数 spi_crc_off .....	304
表 3-452. 函数 spi_crc_next .....	305
表 3-453. 函数 spi_crc_get .....	305
表 3-454. 函数 spi_crc_error_clear .....	306
表 3-455. 函数 spi_ti_mode_enable .....	306
表 3-456. 函数 spi_ti_mode_disable .....	307
表 3-457. 函数 spi_nssp_mode_enable .....	307
表 3-458. 函数 spi_nssp_mode_disable .....	308
表 3-459. 函数 spi_quad_enable .....	308
表 3-460. 函数 spi_quad_disable .....	309
表 3-461. 函数 spi_quad_write_enable .....	309
表 3-462. 函数 spi_quad_read_enable .....	310
表 3-463. 函数 spi_quad_io23_output_enable .....	310
表 3-464. 函数 spi_quad_io23_output_disable .....	311
表 3-465. 函数 spi_i2s_format_error_clear .....	311
表 3-466. 函数 spi_i2s_flag_get .....	312
表 3-467. 函数 spi_i2s_interrupt_enable .....	313
表 3-468. 函数 spi_i2s_interrupt_disable .....	314
表 3-469. 函数 spi_i2s_interrupt_flag_get .....	315
表 3-470. SYSCFG 寄存器 .....	316
表 3-471. SYSCFG 库函数 .....	316
表 3-472. 函数 syscfg_deinit .....	317
表 3-473. 函数 syscfg_exti_line_config .....	321
表 3-474. 函数 syscfg_pin_remap_enable .....	319
表 3-475. 函数 syscfg_pin_remap_disable .....	320
表 3-476. 函数 syscfg_high_current_enable .....	322
表 3-477. 函数 syscfg_high_current_disable .....	323
表 3-478. 函数 irq_latency_set .....	324



表 3-479. 函数 syscfg_bootmode_get.....	320
表 3-480. TIMER 寄存器 .....	328
表 3-481. TIMER 库函数 .....	329
表 3-482. 结构体 timer_parameter_struct.....	331
表 3-483. 结构体 timer_break_parameter_struct.....	332
表 3-484. 结构体 timer_oc_parameter_struct .....	332
表 3-485. 结构体 timer_ic_parameter_struct.....	333
表 3-486. 函数 timer_deinit .....	333
表 3-487. 函数 timer_struct_para_init .....	333
表 3-488. 函数 timer_init.....	334
表 3-489. 函数 timer_enable.....	335
表 3-490. 函数 timer_disable.....	335
表 3-491. 函数 timer_auto_reload_shadow_enable.....	336
表 3-492. 函数 timer_auto_reload_shadow_disable.....	336
表 3-493. 函数 timer_update_event_enable .....	337
表 3-494. 函数 timer_update_event_disable .....	337
表 3-495. 函数 timer_counter_alignment.....	338
表 3-496. 函数 timer_counter_up_direction .....	339
表 3-497. 函数 timer_counter_down_direction.....	339
表 3-498. 函数 timer_prescaler_config .....	340
表 3-499. 函数 timer_repetition_value_config .....	341
表 3-500. 函数 timer_autoreload_value_config.....	341
表 3-501. 函数 timer_counter_value_config .....	342
表 3-502. 函数 timer_counter_read.....	342
表 3-503. 函数 timer_prescaler_read.....	343
表 3-504. 函数 timer_single_pulse_mode_config .....	343
表 3-505. 函数 timer_update_source_config .....	344
表 3-506. 函数 timer_dma_enable.....	345
表 3-507. 函数 timer_dma_disable.....	346
表 3-508. 函数 timer_channel_dma_request_source_select .....	347
表 3-509. 函数 timer_dma_transfer_config .....	347
表 3-510. 函数 timer_event_software_generate .....	349
表 3-511. 函数 timer_break_struct_para_init .....	350
表 3-512. 函数 timer_break_config.....	350
表 3-513. 函数 timer_break_enable .....	351
表 3-514. 函数 timer_break_disable .....	352
表 3-515. 函数 timer_automatic_output_enable.....	352
表 3-516. 函数 timer_automatic_output_disable.....	353
表 3-517. 函数 timer_primary_output_config .....	354
表 3-518. 函数 timer_channel_control_shadow_config.....	354
表 3-519. 函数 timer_channel_control_shadow_update_config .....	355
表 3-520. 函数 timer_channel_output_struct_para_init.....	356
表 3-521. 函数 timer_channel_output_config.....	356
表 3-522. 函数 timer_channel_output_mode_config.....	357

表 3-523. 函数 timer_channel_combined_3_phase_pwm_config.....	359
表 3-524. 函数 timer_channel_output_pulse_value_config .....	359
表 3-525. 函数 timer_channel_output_shadow_config.....	360
表 3-526. 函数 timer_channel_output_fast_config.....	361
表 3-527. 函数 timer_channel_output_clear_config.....	362
表 3-528. 函数 timer_channel_output_polarity_config .....	363
表 3-529. 函数 timer_channel_complementary_output_polarity_config .....	364
表 3-530. 函数 timer_channel_output_state_config.....	365
表 3-531. 函数 timer_channel_complementary_output_state_config .....	366
表 3-532. 函数 timer_channel_input_struct_para_init .....	367
表 3-533. 函数 timer_input_capture_config .....	367
表 3-534. 函数 timer_channel_input_capture_prescaler_config.....	368
表 3-535. 函数 timer_channel_capture_value_register_read.....	369
表 3-536. 函数 timer_input_pwm_capture_config .....	370
表 3-537. 函数 timer_hall_mode_config .....	371
表 3-538. 函数 timer_input_trigger_source_select.....	372
表 3-539. 函数 timer_master_output_trigger_source_select.....	372
表 3-540. 函数 timer_slave_mode_select.....	373
表 3-541. 函数 timer_master_slave_mode_config.....	374
表 3-542. 函数 timer_external_trigger_config .....	375
表 3-543. 函数 timer_quadrature_decoder_mode_config .....	376
表 3-544. 函数 timer_internal_clock_config.....	377
表 3-545. 函数 timer_external_clock_mode0_config .....	378
表 3-546. 函数 timer_external_clock_mode1_config .....	379
表 3-547. 函数 timer_external_clock_mode1_disable.....	380
表 3-548. 函数 timer_channel_remap_config .....	380
表 3-549. 函数 timer_write_chxval_register_config .....	381
表 3-550. 函数 timer_output_value_selection_config .....	382
表 3-551. 函数 timer_break_external_source_config.....	382
表 3-552. 函数 timer_break_external_polarity_config.....	383
表 3-553. 函数 timer_flag_get.....	384
表 3-554. 函数 timer_flag_clear.....	385
表 3-555. 函数 timer_interrupt_enable.....	386
表 3-556. 函数 timer_interrupt_disable.....	387
表 3-557. 函数 timer_interrupt_flag_get .....	387
表 3-558. 函数 timer_interrupt_flag_clear .....	389
表 3-559. USART 寄存器 .....	390
表 3-560. USART 库函数.....	390
表 3-561. 枚举类型 usart_flag_enum .....	392
表 3-562. 枚举类型 usart_interrupt_flag_enum .....	393
表 3-563. 枚举类型 usart_interrupt_enum .....	393
表 3-564. 枚举类型 usart_invert_enum .....	394
表 3-565. 函数 usart_deinit .....	394
表 3-566. 函数 usart_baudrate_set.....	395

表 3-567. 函数 usart_parity_config.....	395
表 3-568. 函数 usart_word_length_set .....	396
表 3-569. 函数 usart_stop_bit_set .....	396
表 3-570. 函数 usart_enable.....	397
表 3-571. 函数 usart_disable.....	398
表 3-572. 函数 usart_transmit_config .....	398
表 3-573. 函数 usart_receive_config .....	399
表 3-574. 函数 usart_data_first_config .....	400
表 3-575. 函数 usart_invert_config.....	400
表 3-576. 函数 usart_overrun_enable .....	401
表 3-577. 函数 usart_overrun_disable .....	402
表 3-578. 函数 usart_oversample_config .....	402
表 3-579. 函数 usart_sample_bit_config .....	403
表 3-580. 函数 usart_receiver_timeout_enable .....	403
表 3-581. 函数 usart_receiver_timeout_disable .....	404
表 3-582. 函数 usart_receiver_timeout_threshold_config .....	404
表 3-583. 函数 usart_data_transmit.....	405
表 3-584. 函数 usart_data_receive.....	406
表 3-585. 函数 usart_command_enable.....	406
表 3-586. 函数 usart_address_config.....	407
表 3-587. 函数 usart_address_detection_mode_config .....	407
表 3-588. 函数 usart_mute_mode_enable .....	408
表 3-589. 函数 usart_mute_mode_disable .....	409
表 3-590. 函数 usart_mute_mode_wakeup_config.....	409
表 3-591. 函数 usart_lin_mode_enable.....	410
表 3-592. 函数 usart_lin_mode_disable.....	410
表 3-593. 函数 usart_lin_break_dection_length_config .....	411
表 3-594. 函数 usart_halfduplex_enable.....	411
表 3-595. 函数 usart_halfduplex_disable .....	412
表 3-596. 函数 usart_clock_enable.....	412
表 3-597. 函数 usart_clock_disable.....	413
表 3-598. 函数 usart_synchronous_clock_config.....	413
表 3-599. 函数 usart_guard_time_config.....	414
表 3-600. 函数 usart_smartcard_mode_enable .....	415
表 3-601. 函数 usart_smartcard_mode_disable .....	415
表 3-602. 函数 usart_smartcard_mode_nack_enable .....	416
表 3-603. 函数 usart_smartcard_mode_nack_disable .....	416
表 3-604. 函数 usart_smartcard_mode_early_nack_enable .....	417
表 3-605. 函数 usart_smartcard_mode_early_nack_disable .....	417
表 3-606. 函数 usart_smartcard_autoretry_config .....	418
表 3-607. 函数 usart_block_length_config.....	419
表 3-608. 函数 usart_irda_mode_enable .....	419
表 3-609. 函数 usart_irda_mode_disable .....	420
表 3-610. 函数 usart_prescaler_config .....	420

表 3-611. 函数 usart_irda_lowpower_config .....	421
表 3-612. 函数 usart_hardware_flow_rts_config .....	421
表 3-613. 函数 usart_hardware_flow_cts_config .....	422
表 3-614. 函数 usart_hardware_flow_coherence_config .....	423
表 3-615. 函数 usart_rs485_driver_enable .....	423
表 3-616. 函数 usart_rs485_driver_disable .....	424
表 3-617. 函数 usart_driver_asserttime_config .....	424
表 3-618. 函数 usart_driver_deasserttime_config .....	425
表 3-619. 函数 usart_depolarity_config .....	425
表 3-620. 函数 usart_dma_receive_config .....	426
表 3-621. 函数 usart_dma_transmit_config .....	427
表 3-622. 函数 usart_reception_error_dma_disable .....	427
表 3-623. 函数 usart_reception_error_dma_enable .....	428
表 3-624. 函数 usart_wakeup_enable .....	428
表 3-625. 函数 usart_wakeup_disable .....	429
表 3-626. 函数 usart_wakeup_mode_config .....	429
表 3-627. 函数 usart_receive_fifo_enable .....	430
表 3-628. 函数 usart_receive_fifo_disable .....	431
表 3-629. 函数 usart_receive_fifo_counter_number .....	431
表 3-630. 函数 usart_flag_get .....	432
表 3-631. 函数 usart_flag_clear .....	433
表 3-632. 函数 usart_interrupt_enable .....	434
表 3-633. 函数 usart_interrupt_disable .....	435
表 3-634. 函数 usart_interrupt_flag_get .....	436
表 3-635. 函数 usart_interrupt_flag_clear .....	437
表 3-636. WWDGT 寄存器 .....	439
表 3-637. WWDGT 库函数 .....	439
表 3-638. 函数 wwdgt_deinit .....	439
表 3-639. 函数 wwdgt_enable .....	440
表 3-640. 函数 wwdgt_prescaler_value_config .....	440
表 3-641. 函数 wwdgt_window_value_config .....	441
表 3-642. 函数 wwdgt_counter_update .....	442
表 3-643. 函数 wwdgt_config .....	442
表 3-644. 函数 wwdgt_interrupt_enable .....	444
表 3-645. 函数 wwdgt_flag_get .....	444
表 3-646. 函数 wwdgt_flag_clear .....	445
表 4-1. 版本历史 .....	446

## 1. 介绍

本手册介绍了32位基于ARM微控制器GD32C2x1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GD32C2x1所有外设的性能特征。该固件库还包括每一个外设的驱动描述和基于评估板的固件库使用例程。通过使用本固件库，用户无需深入掌握细节，也可以轻松应用每一个外设。使用本固件库可以大大减少用户的编程时间，从而降低开发成本。

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

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

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

此份固件库使用手册的整体架构如下：

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

### 1.1. 文档和固件库规则

#### 1.1.1. 外设缩写

表 1-1. 外设缩写

外设缩写	说明
ADC	模数转换器
CMP	比较器
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
DMAMUX	DMA请求多路复用器
EXTI	外部中断事件控制器
FMC	闪存控制器
FWDGT	独立看门狗
GPIO/AFIO	通用和备用输入/输出接口

外设缩写	说明
I2C	内部集成电路总线接口
MISC	嵌套中断向量列表控制器
PMU	电源管理单元
RCU	复位和时钟单元
RTC	实时时钟
SPI/I2S	串行外设接口/片上音频接口
SYSCFG	系统配置
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：

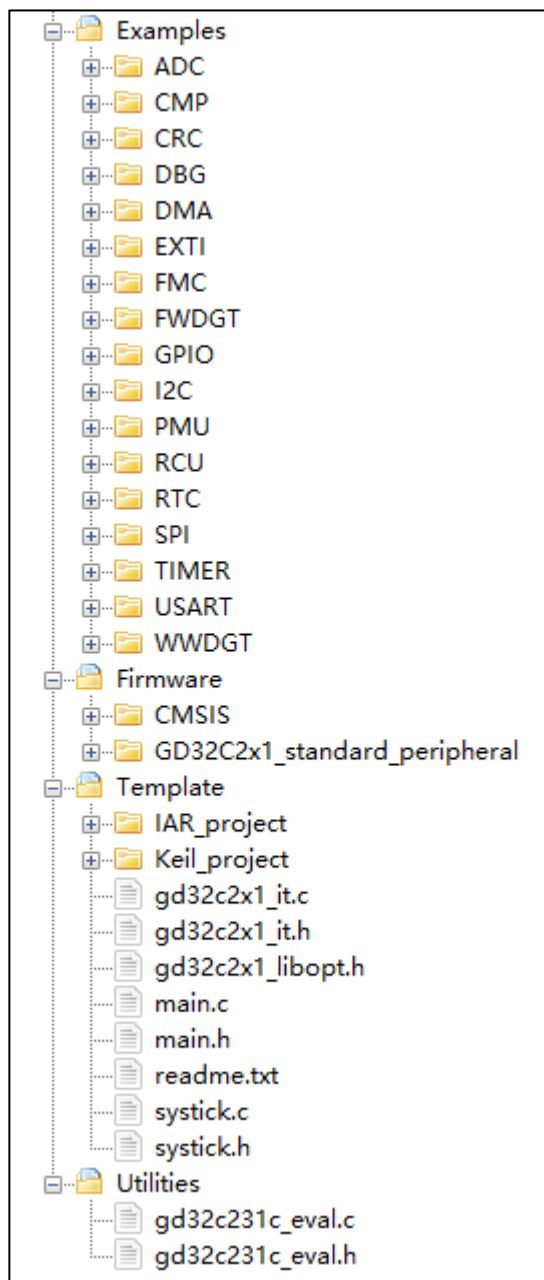
- XXX表示任一外设缩写，例如：ADC。更多缩写相关信息参阅[外设缩写](#)；
- 源文件和头文件命名都以“GD32C2x1\_”作为开头，例如：GD32C2x1\_adc.h；
- 常量仅被应用于一个文件的，定义于该文件中；被应用于多个文件的，在对应头文件中定义。所有常量都由英文字母大写书写；
- 寄存器作为常量处理。他们的命名都由英文字母大写书写。在大多数情况下，寄存器缩写规范与本用户手册一致；
- 变量名采用全部小写，有多个单词组成的，在单词之间以下划线分隔；
- 外设函数的命名以该外设的缩写加下划线为开头，有多个单词组成的，在单词之间以下划线分隔，所有外设函数都由英文字母小写书写。

## 2. 固件库概述

### 2.1. 文件组织结构

GD32C2x1\_Firmware\_Library, 文件组织结构见下图:

图 2-1. GD32C2x1 固件库文件组织结构



#### 2.1.1. Examples 文件夹

文件夹Examples, 对应每一个GD32外设均包含一个子文件夹。每个子文件夹包含了关于本外



设的一个或多个例程，来示范如何使用对应外设。每个例程子文件夹包含如下文件：

- **readme.txt**: 关于本例程的简单描述和使用说明；
- **GD32C2x1\_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）；
- **GD32C2x1\_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）；
- **GD32C2x1\_it.h**: 该头文件包含了所有的中断处理程序的原形；
- **systick.c**: 该源文件包含了使用**systick**的精准延时程序；
- **systick.h**: 该头文件包含了使用**systick**的精准延时程序的原形；
- **main.c**: 例程代码注：所有的例程的使用，都不受不同软件开发环境的影响。

### 2.1.2. Firmware 文件夹

Firmware文件夹包含组成固件库核心的所有子文件夹和文件：

- **CMSIS**子文件夹包含有**Cortex M23**内核的支持文件、基于**Cortex M23**内核处理器的启动代码和库引导文件以及基于**GD32C2x1**的全局头文件和系统配置文件；
- **GD32C2x1\_standard\_peripheral**子文件夹：
  - **Include**子文件夹包含了固件函数库所需的头文件，用户无需修改该文件夹；
  - **Source**子文件夹包含了固件函数库所需的源文件，用户无需修改该文件夹；

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

### 2.1.3. Template 文件夹

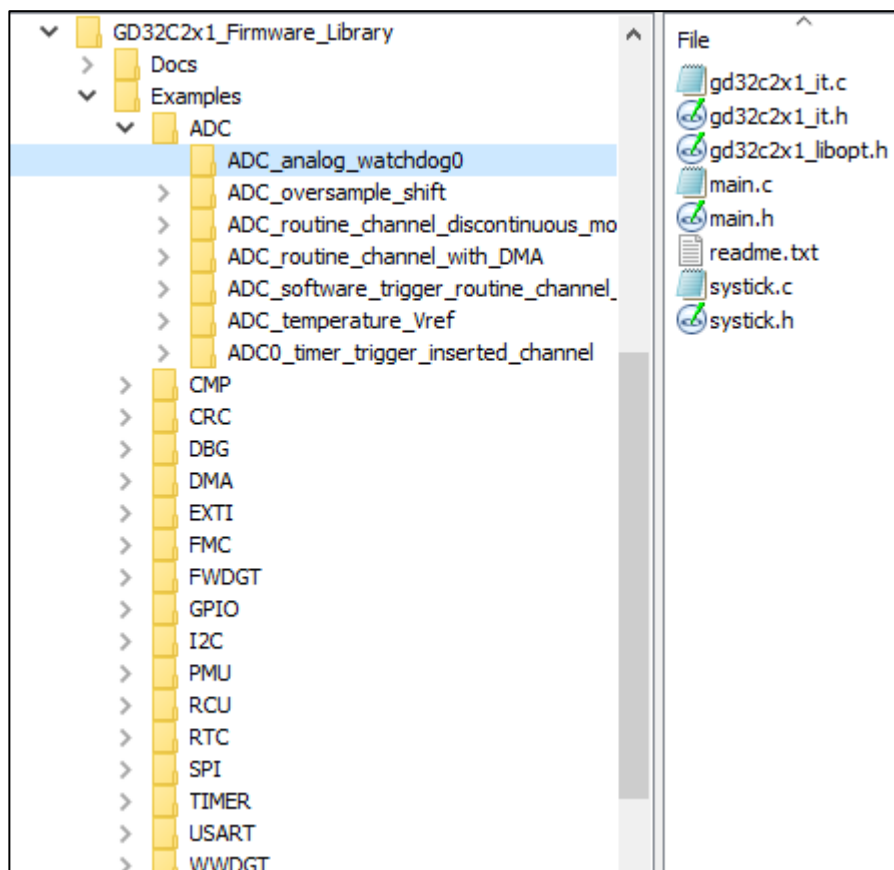
Template文件夹包含一个关于使用**LED**、**USART**打印、按键控制的简单例程，（**IAR\_project**用于**IAR**编译环境，**Keil\_project**用于**Keil5**编译环境）。用户可以使用该工程模板进行固件库例程的移植编译，具体使用方法见下：

#### 选择文件

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



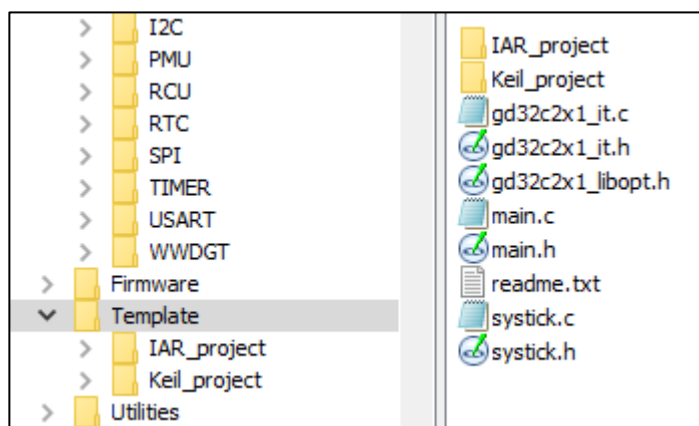
图 2-2. 选择外设例程文件



### 拷贝文件

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

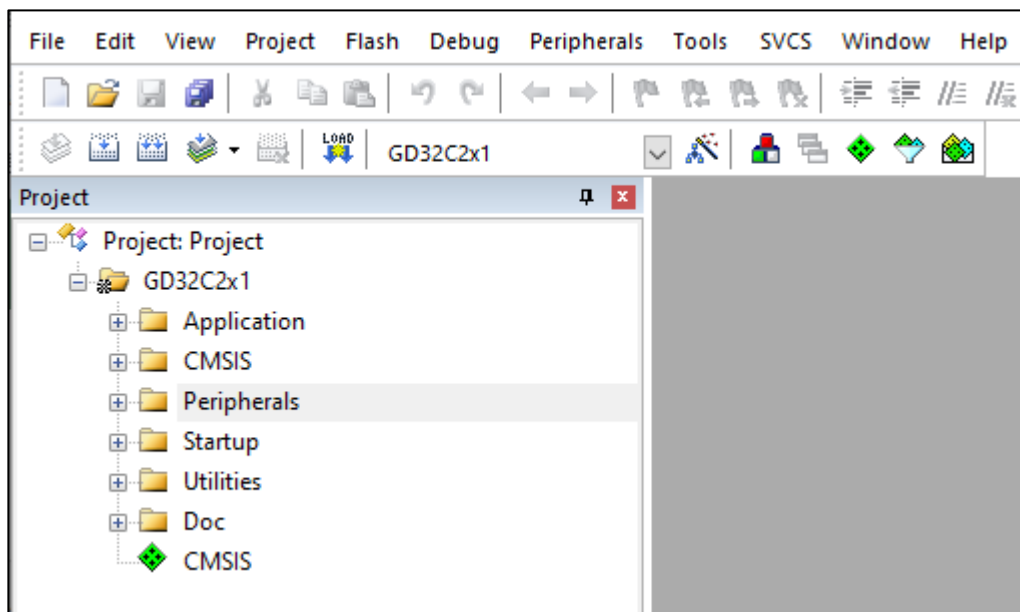
图 2-3. 拷贝外设例程文件



### 打开工程

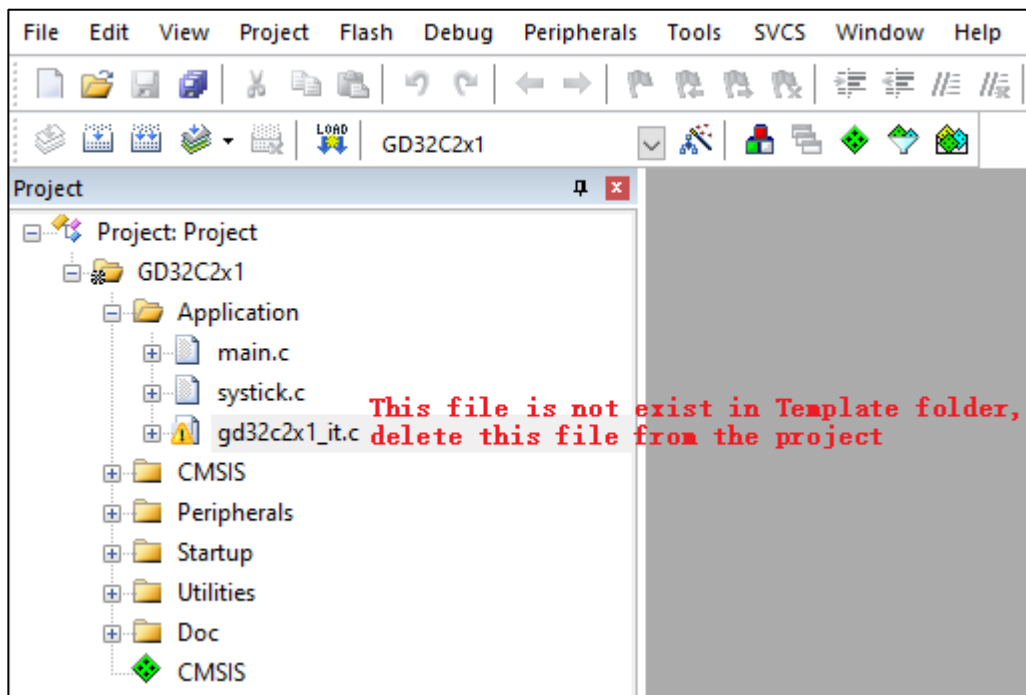
GD 提供 Keil 和 IAR 两种版本的工程，根据客户所安装的软件，打开不同的 project，如“Keil\_project”，打开\Template\Keil\_project\Project.uvprojx，如下图所示：

图 2-4. 打开工程文件



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

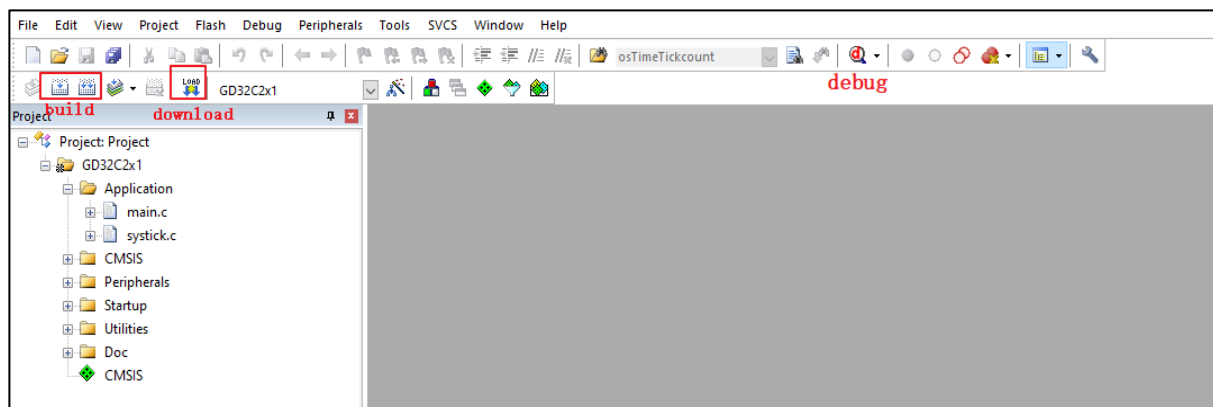
图 2-5. 配置工程文件



## 编译调试下载

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

图 2-6. 编译调试下载



#### 2.1.4. Utilities 文件夹

Utilities文件夹包含运行固件库例程评估板的文件：

- gd32c231c\_eval.c文件是运行固件库例程所需关于评估板的源文件。

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

## 2.2. 固件库文件描述

下表列举和描述了固件库使用的主要文件。

表 2-1. 固件函数库文件描述

文件名	描述
GD32C2x1_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。
GD32C2x1_it.h	头文件，包含所有中断处理函数原形。
GD32C2x1_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。
GD32C2x1_xxx.h	外设xxx的头文件。包含外设xxx函数的定义，以及这些函数使用的变量。
GD32C2x1_xxx.c	由C语言编写的外设xxx的驱动源程序文件。
systick.h	systick.c的头文件。包含systick配置函数的定义，以及外部用延时函数的定义。
systick.c	systick配置与延时函数源文件。
readme.txt	固件库例程使用及配置说明文档。

## 3. 外设固件库

### 3.1. 外设固件库概述

外设固件库函数的描述格式如下表：

表 3-1. 外设固件库函数描述格式

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

### 3.2. ADC

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

#### 3.2.1. 外设寄存器描述

ADC寄存器列表如下表所示：

表 3-2. ADC 寄存器

寄存器名称	寄存器描述
ADC_STAT	状态寄存器
ADC_CTL0	控制寄存器0
ADC_CTL1	控制寄存器1
ADC_SAMPT0	采样时间寄存器0
ADC_SAMPT1	采样时间寄存器1
ADC_IOFFx	注入通道数据偏移寄存器x (x=0..3)
ADC_WD0HT	看门狗0高阈值寄存器
ADC_WD0LT	看门狗0低阈值寄存器
ADC_RSQ0	规则序列寄存器0
ADC_RSQ1	规则序列寄存器1

寄存器名称	寄存器描述
ADC_RSQ2	规则序列寄存器2
ADC_ISQ	注入序列寄存器
ADC_IDATAx	注入数据寄存器x (x=0..3)
ADC_RDATA	规则数据寄存器
ADC_WD1SR	看门狗1通道选择寄存器
ADC_WD2SR	看门狗2通道选择寄存器
ADC_WD1HT	看门狗1高阈值寄存器
ADC_WD1LT	看门狗1低阈值寄存器
ADC_WD2HT	看门狗2高阈值寄存器
ADC_WD2LT	看门狗2低阈值寄存器
ADC_OVSAMPCTL	过采样控制寄存器

### 3.2.2. 外设库函数说明

ADC库函数列表如下表所示：

**表 3-3. ADC 库函数**

库函数名称	库函数描述
adc_deinit	复位ADC外设
adc_enable	使能ADC外设
adc_disable	禁能ADC外设
adc_dma_mode_enable	使能DMA请求
adc_dma_mode_disable	禁能DMA请求
adc_discontinuous_mode_config	配置ADC间断模式
adc_special_function_config	配置ADC特殊功能
adc_internal_channel_config	配置ADC内部通道
adc_data_alignment_config	配置ADC数据对齐方式
adc_channel_length_config	配置常规序列或注入序列的通道长度
adc_routine_channel_config	配置ADC常规序列通道
adc_inserted_channel_config	配置ADC注入序列通道
adc_inserted_channel_offset_config	配置ADC注入序列通道数据偏移值
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	使能ADC软件触发
adc_routine_data_read	读ADC常规序列数据寄存器
adc_inserted_data_read	读ADC注入序列数据寄存器
adc_watchdog0_single_channel_enable	使能ADC模拟看门狗0单通道功能
adc_watchdog0_sequence_channel_enable	使能ADC模拟看门狗0序列通道功能
adc_watchdog0_disable	禁能ADC模拟看门狗0
adc_watchdog1_channel_config	配置ADC模拟看门狗1通道
adc_watchdog2_channel_config	配置ADC模拟看门狗2通道

库函数名称	库函数描述
adc_watchdog1_disable	禁能ADC模拟看门狗1
adc_watchdog2_disable	禁能ADC模拟看门狗2
adc_watchdog0_threshold_config	配置ADC模拟看门狗0阈值
adc_watchdog1_threshold_config	配置ADC模拟看门狗1阈值
adc_watchdog2_threshold_config	配置ADC模拟看门狗2阈值
adc_resolution_config	配置ADC分辨率
adc_oversample_mode_config	配置ADC过采样模式
adc_oversample_mode_enable	使能ADC过采样
adc_oversample_mode_disable	禁能ADC过采样
adc_flag_get	获取ADC标志位
adc_flag_clear	清除ADC标志位
adc_interrupt_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\_enable

函数adc\_enable描述见下表：

表 3-5. 函数 adc\_enable

函数名称	adc_enable
------	------------

函数原形	void adc_enable(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 3-6. 函数 adc\_disable

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

例如：

```
/* disable ADC */
```

```
adc_disable();
```

### 函数 adc\_dma\_mode\_enable

函数 adc\_dma\_mode\_enable 描述见下表：

表 3-7. 函数 adc\_dma\_mode\_enable

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

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

例如：

```
/* enable 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 3-8. 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	禁能DMA请求
先决条件	-
被调用函数	-
输入参数{in}	
-	-
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* disable ADC DMA request */
```

```
adc_dma_mode_disable();
```

### 函数 `adc_discontinuous_mode_config`

函数 `adc_discontinuous_mode_config` 描述见下表：

表 3-9. 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中断模式
先决条件	-
被调用函数	-
输入参数{in}	
<b>adc_channel_group</b>	通道组选择
<i>ADC_ROUTINE_CHANNELNEL</i>	常规序列
<i>ADC_INSERTED_CHANNELNEL</i>	注入序列
<i>ADC_CHANNEL_DISCON_DISABLE</i>	常规序列和注入序列中断模式禁能
输入参数{in}	
<b>length</b>	中断模式下的转换数目，常规序列取值为1..8，注入序列取值无意义
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* configure ADC routine sequence 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 3-10. 函数 **adc\_special\_function\_config**

函数名称	adc_special_function_config
函数原形	void adc_special_function_config(uint32_t function, ControlStatus newvalue);
功能描述	配置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_CHANNELNEL_AUTO</i>	注入序列自动转换
<i>ADC_CONTINUOUS_MODE</i>	连续模式选择
输入参数{in}	
<b>newvalue</b>	控制参数
<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_internal_channel_config`

函数 `adc_internal_channel_config` 描述见下表:

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

函数名称	<code>adc_internal_channel_config</code>
函数原形	<code>void adc_internal_channel_config(uint32_t internal_channel, ControlStatus newvalue)</code>
功能描述	配置ADC内部通道
先决条件	-
被调用函数	-
输入参数{in}	
<code>internal_channel</code>	内部通道
<code>ADC_CHANNEL_INTERNAL_TEMPSENSOR</code>	温度传感器通道
<code>ADC_CHANNEL_INTERNAL_VREFINT</code>	内部参考电压通道
输入参数{in}	
<code>newvalue</code>	控制参数
<code>ENABLE</code>	使能
<code>DISABLE</code>	禁能
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* enable ADC temperature sensor */
```

```
adc_internal_channel_config(ADC_CHANNEL_INTERNAL_TEMPSENSOR, ENABLE);
```

### 函数 `adc_data_alignment_config`

函数 `adc_data_alignment_config` 描述见下表:

表 3-12. 函数 `adc_data_alignment_config`

函数名称	<code>adc_data_alignment_config</code>
函数原形	<code>void adc_data_alignment_config(uint32_t data_alignment);</code>
功能描述	配置ADC数据对齐方式
先决条件	-
被调用函数	-
输入参数{in}	
<b>data_alignment</b>	数据对齐方式选择
<code>ADC_DATAALIGN_RIGHT</code>	右对齐
<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 3-13. 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_sequence, uint32_t length);</code>
功能描述	配置常规序列或注入序列的通道长度
先决条件	-
被调用函数	-
输入参数{in}	
<b>adc_sequence</b>	序列选择
<code>ADC_ROUTINE_CHANNEL</code>	常规序列
<code>ADC_INSERTED_CHANNEL</code>	注入序列
输入参数{in}	
<b>length</b>	序列的通道长度，常规序列为1-16，注入序列为1-4
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* configure the length of ADC routine sequence */
```

```
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 3-14. 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常规序列通道
先决条件	-
被调用函数	-
输入参数{in}	
rank	常规序列通道顺序, 取值范围为0~15
输入参数{in}	
adc_channel	ADC通道选择
ADC_CHANNEL_x	ADC通道x (x=0..15)
输入参数{in}	
sample_time	采样时间
ADC_SAMPLETIME_2POINT5	2.5周期
ADC_SAMPLETIME_3POINT5	3.5周期
ADC_SAMPLETIME_7POINT5	7.5周期
ADC_SAMPLETIME_12POINT5	12.5周期
ADC_SAMPLETIME_19POINT5	19.5周期
ADC_SAMPLETIME_39POINT5	39.5周期
ADC_SAMPLETIME_79POINT5	79.5周期
ADC_SAMPLETIME_160POINT5	160.5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_7POINT5);
```

### 函数 adc\_inserted\_channel\_config

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

表 3-15. 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注入序列通道
先决条件	-
被调用函数	-
输入参数{in}	
rank	注入序列通道顺序, 取值范围为0~3
输入参数{in}	
channel	ADC通道选择
ADC_CHANNEL_x	ADC通道x (x=0..15)
输入参数{in}	
sample_time	采样时间
ADC_SAMPLETIME_2POINT5	2.5周期
ADC_SAMPLETIME_3POINT5	3.5周期
ADC_SAMPLETIME_7POINT5	7.5周期
ADC_SAMPLETIME_12POINT5	12.5周期
ADC_SAMPLETIME_19POINT5	19.5周期
ADC_SAMPLETIME_39POINT5	39.5周期
ADC_SAMPLETIME_79POINT5	79.5周期
ADC_SAMPLETIME_160POINT5	160.5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_7POINT5);
```

### 函数 `adc_inserted_channel_offset_config`

函数 `adc_inserted_channel_offset_config` 描述见下表:

表 3-16. 函数 `adc_inserted_channel_offset_config`

函数名称	<code>adc_inserted_channel_offset_config</code>
函数原形	<code>void adc_inserted_channel_offset_config(uint8_t inserted_channel, uint16_t offset);</code>
功能描述	配置ADC注入通道组数据偏移值
先决条件	-
被调用函数	-
输入参数{in}	
<code>inserted_channel</code>	注入通道选择
<code>ADC_INSERTED_CHANNEL_x</code>	注入通道, x=0,1,2,3
输入参数{in}	
<code>offset</code>	数据偏移值
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 3-17. 函数 `adc_external_trigger_config`

函数名称	<code>adc_external_trigger_config</code>
函数原形	<code>vvoid adc_external_trigger_config(uint8_t adc_sequence, ControlStatus newvalue);</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_sequence</code>	序列选择

ADC_ROUTINE_CHANNEL	常规序列
ADC_INSERTED_CHANNEL	注入序列
输入参数{in}	
newvalue	控制参数
ENABLE	使能
DISABLE	禁能
输出参数{out}	
-	-
返回值	
-	-

例如：

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

```
adc_external_trigger_config(ADC_INSERTED_CHANNEL, ENABLE);
```

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

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

表 3-18. 函数 adc\_external\_trigger\_source\_config

函数名称	adc_external_trigger_source_config
函数原形	void adc_external_trigger_source_config(uint8_t adc_sequence, uint32_t external_trigger_source);
功能描述	配置ADC外部触发源
先决条件	-
被调用函数	-
输入参数{in}	
adc_sequence	序列选择
ADC_ROUTINE_CHANNEL	常规序列
ADC_INSERTED_CHANNEL	注入序列
输入参数{in}	
external_trigger_source	常规序列或注入序列触发源
ADC_EXTTRIG_ROUTINE_T2_CH1	TIMER2 CH1事件（常规序列）
ADC_EXTTRIG_ROUTINE_T0_CH2	TIMER0 CH2事件（常规序列）
ADC_EXTTRIG_ROUTINE_T0_CH1	TIMER0 CH1事件(常规序列)

ADC_EXTTRIG_ROU TINE_T2_TRGO	TIMER2 TRGO事件（常规序列）
ADC_EXTTRIG_ROU TINE_T0_CH0	TIMER0 CH0事件（常规序列）
ADC_EXTTRIG_ROU TINE_T2_CH0	TIMER2 CH0事件（常规序列）
ADC_EXTTRIG_ROU TINE_EXTI_11	外部中断线11（常规序列）
ADC_EXTTRIG_ROU TINE_NONE	软件触发（常规序列）
ADC_EXTTRIG_INSE RTED_T0_TRGO	TIMER0 TRGO事件(注入序列)
ADC_EXTTRIG_INSE RTED_T0_CH3	TIMER0 CH3事件(注入序列)
ADC_EXTTRIG_INSE RTED_T13_CH0	TIMER13 TRGO事件(注入序列)
ADC_EXTTRIG_INSE RTED_T2_CH3	TIMER2 CH3事件（注入序列）
ADC_EXTTRIG_INSE RTED_T2_CH2	TIMER2 CH2事件（注入序列）
ADC_EXTTRIG_INSE RTED_T15_CH0	TIMER15 CH0事件（注入序列）
ADC_EXTTRIG_INSE RTED_EXTI_15	外部中断线15（注入序列）
ADC_EXTTRIG_INSE RTED_NONE	软件触发（注入序列）
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 3-19. 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_sequence);



功能描述	使能ADC软件触发
先决条件	-
被调用函数	-
输入参数{in}	
adc_sequence	序列选择
ADC_ROUTINE_CHANNEL	常规序列
ADC_INSERTED_CHANNEL	注入序列
输出参数{out}	
-	-
返回值	
-	-

例如：

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

### 函数 adc\_routine\_data\_read

函数 adc\_routine\_data\_read 描述见下表：

表 3-20. 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常规序列数据寄存器
先决条件	-
被调用函数	-
输出参数{out}	
-	-
返回值	
uint16_t	ADC转换值：0~0xFFFF

例如：

```
/* read ADC routine sequence 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 3-21. 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注入序列数据寄存器
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</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_watchdog0_single_channel_enable`

函数 `adc_watchdog0_single_channel_enable` 描述见下表:

表 3-22. 函数 `adc_watchdog_single_channel_enable`

函数名称	<code>adc_watchdog0_single_channel_enable</code>
函数原形	<code>void adc_watchdog0_single_channel_enable(uint8_t adc_channel);</code>
功能描述	使能ADC模拟看门狗0单通道功能
先决条件	-
被调用函数	-
输入参数{in}	
<code>adc_channel</code>	选择ADC通道
<code>ADC_CHANNEL_x</code>	ADC Channelx(x=0..15)
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* configure ADC analog watchdog0 single channel */
```

```
adc_watchdog0_single_channel_enable(ADC_CHANNEL_1);
```

## 函数 `adc_watchdog0_sequence_channel_enable`

函数 `adc_watchdog0_sequence_channel_enable` 描述见下表：

**表 3-23. 函数 `adc_watchdog0_sequence_channel_enable`**

函数名称	<code>adc_watchdog0_sequence_channel_enable</code>
函数原形	<code>void adc_watchdog0_sequence_channel_enable(uint8_t adc_sequence);</code>
功能描述	使能ADC模拟看门狗0序列通道功能
先决条件	-
被调用函数	-
输入参数{in}	
<code>adc_sequence</code>	序列选择
<code>ADC_ROUTINE_CHANNELNEL</code>	常规序列
<code>ADC_INSERTED_CHANNELNEL</code>	注入序列
<code>ADC_ROUTINE_INSERTED_CHANNELNEL</code>	常规和注入序列
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* configure ADC analog watchdog 0 sequence channel */
```

```
adc_watchdog0_sequence_channel_enable(ADC_ROUTINE_CHANNELNEL);
```

## 函数 `adc_watchdog0_disable`

函数 `adc_watchdog0_disable` 描述见下表：

**表 3-24. 函数 `adc_watchdog0_disable`**

函数名称	<code>adc_watchdog0_disable</code>
函数原形	<code>void adc_watchdog0_disable(void);</code>
功能描述	禁能ADC模拟看门狗0
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 0*/
```

```
adc_watchdog0_disable();
```

### 函数 `adc_watchdog1_channel_config`

函数 `adc_watchdog1_channel_config` 描述见下表:

**表 3-25. 函数 `adc_watchdog1_channel_config`**

函数名称	<code>adc_watchdog1_channel_config</code>
函数原形	<code>void adc_watchdog1_channel_config(uint32_t selection_channel, ControlStatus newvalue);</code>
功能描述	配置ADC模拟看门狗1单通道有效
先决条件	-
被调用函数	-
输入参数{in}	
<b>selection_channel</b>	选择ADC通道
<code>ADC_AWD1_2_SELECTION_CHANNEL_x (x=0..15), ADC_AWD1_2_SELECTION_CHANNEL_ALL</code>	ADC模拟看门狗1/2通道选择(x=0..15)
输入参数{in}	
<b>newvalue</b>	使能/禁能控制
<code>ENABLE</code>	使能
<code>DISABLE</code>	禁能
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* configure ADC analog watchdog1 channel */
```

```
adc_watchdog1_channel_config(ADC_AWD1_2_SELECTION_CHANNEL_1,ENABLE);
```

### 函数 `adc_watchdog2_channel_config`

函数 `adc_watchdog2_channel_config` 描述见下表:

**表 3-26. 函数 `adc_watchdog2_channel_config`**

函数名称	<code>adc_watchdog2_channel_config</code>
函数原形	<code>void adc_watchdog2_channel_config(uint32_t selection_channel, ControlStatus</code>

	newvalue);
功能描述	配置ADC模拟看门狗2单通道有效
先决条件	-
被调用函数	-
输入参数{in}	
selection_channel	选择ADC通道
ADC_AWD1_2_SELECTION_CHANNEL_x (x=0..15), ADC_AWD1_2_SELECTION_CHANNEL_ALL	ADC模拟看门狗1/2通道选择(x=0..15)
输入参数{in}	
newvalue	使能/禁能控制
ENABLE	使能
DISABLE	禁能
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* configure ADC analog watchdog 2 channel */
```

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

### 函数 adc\_watchdog1\_disable

函数 adc\_watchdog1\_disable 描述见下表:

表 3-27. 函数 adc\_watchdog1\_disable

函数名称	adc_watchdog1_disable
函数原形	void adc_watchdog1_disable(void);
功能描述	禁能ADC模拟看门狗1
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 1*/
```

```
adc_watchdog1_disable();
```

### 函数 `adc_watchdog2_disable`

函数 `adc_watchdog2_disable` 描述见下表：

**表 3-28. 函数 `adc_watchdog2_disable`**

函数名称	adc_watchdog2_disable
函数原形	void adc_watchdog2_disable(void);
功能描述	禁能ADC模拟看门狗2
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 2*/
```

```
adc_watchdog2_disable();
```

### 函数 `adc_watchdog0_threshold_config`

函数 `adc_watchdog0_threshold_config` 描述见下表：

**表 3-29. 函数 `adc_watchdog0_threshold_config`**

函数名称	adc_watchdog0_threshold_config
函数原形	void adc_watchdog0_threshold_config(uint16_t low_threshold, uint16_t high_threshold);
功能描述	配置ADC模拟看门狗0阈值
先决条件	-
被调用函数	-
输入参数{in}	
low_threshold	模拟看门狗0低阈值，范围为0..0xFFF
输入参数{in}	
high_threshold	模拟看门狗0高阈值，范围为0..0xFFF
输出参数{out}	
-	-
返回值	
-	-

例如：

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

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

### 函数 adc\_watchdog1\_threshold\_config

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

表 3-30. 函数 adc\_watchdog1\_threshold\_config

函数名称	adc_watchdog1_threshold_config
函数原形	void adc_watchdog1_threshold_config(uint16_t low_threshold, uint16_t high_threshold);
功能描述	配置ADC模拟看门狗1阈值
先决条件	-
被调用函数	-
输入参数{in}	
low_threshold	模拟看门狗1低阈值，范围为0..0xFFF
输入参数{in}	
high_threshold	模拟看门狗1高阈值，范围为0..0xFFF
输出参数{out}	
-	-
返回值	
-	-

例如：

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

```
adc_watchdog1_threshold_config(0x0400, 0x0A00);
```

### 函数 adc\_watchdog2\_threshold\_config

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

表 3-31. 函数 adc\_watchdog2\_threshold\_config

函数名称	adc_watchdog2_threshold_config
函数原形	void adc_watchdog2_threshold_config(uint16_t low_threshold, uint16_t high_threshold);
功能描述	配置ADC模拟看门狗2阈值
先决条件	-
被调用函数	-
输入参数{in}	
low_threshold	模拟看门狗2低阈值，范围为0..0xFFF
输入参数{in}	
high_threshold	模拟看门狗2高阈值，范围为0..0xFFF

输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* configure ADC analog watchdog 2 threshold */
adc_watchdog2_threshold_config(0x0400, 0x0A00);
```

### 函数 adc\_resolution\_config

函数 adc\_resolution\_config 描述见下表:

表 3-32. 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	ADC分辨率
ADC_RESOLUTION_12B	12位分辨率
ADC_RESOLUTION_10B	10位分辨率
ADC_RESOLUTION_8B	8位分辨率
ADC_RESOLUTION_6B	6位分辨率
输出参数{out}	
-	-
返回值	
-	-

Example:

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

### 函数 adc\_oversample\_mode\_config

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



表 3-33. 函数 `adc_oversample_mode_config`

函数名称	<code>adc_oversample_mode_config</code>
函数原形	<code>void adc_oversample_mode_config(uint32_t mode, uint16_t shift, uint8_t ratio);</code>
功能描述	配置ADC过采样模式
先决条件	-
被调用函数	-
输入参数{in}	
<b>mode</b>	ADC过采样触发模式
<code>ADC_OVERSAMPLING_ALL_CONVERT</code>	在一个触发之后，对一个通道连续进行过采样转换
<code>ADC_OVERSAMPLING_ONE_CONVERT</code>	在一个触发之后，对一个通道只进行一次过采样转换
输入参数{in}	
<b>shift</b>	ADC过滤采样移位
<code>ADC_OVERSAMPLING_SHIFT_NONE</code>	不移位
<code>ADC_OVERSAMPLING_SHIFT_1B</code>	移1位
<code>ADC_OVERSAMPLING_SHIFT_2B</code>	移2位
<code>ADC_OVERSAMPLING_SHIFT_3B</code>	移3位
<code>ADC_OVERSAMPLING_SHIFT_4B</code>	移4位
<code>ADC_OVERSAMPLING_SHIFT_5B</code>	移5位
<code>ADC_OVERSAMPLING_SHIFT_6B</code>	移6位
<code>ADC_OVERSAMPLING_SHIFT_7B</code>	移7位
<code>ADC_OVERSAMPLING_SHIFT</code>	移8位

_8B	
输入参数{in}	
ratio	ADC过采样率
ADC_OVERSAMPLING_RATIO_MUL2	2x
ADC_OVERSAMPLING_RATIO_MUL4	4x
ADC_OVERSAMPLING_RATIO_MUL8	8x
ADC_OVERSAMPLING_RATIO_MUL16	16x
ADC_OVERSAMPLING_RATIO_MUL32	32x
ADC_OVERSAMPLING_RATIO_MUL64	64x
ADC_OVERSAMPLING_RATIO_MUL128	128x
ADC_OVERSAMPLING_RATIO_MUL256	256x
输出参数{out}	
-	-
返回值	
-	-

Example:

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

```
adc_oversample_mode_config(ADC_OVERSAMPLING_ALL_CONVERT,
ADC_OVERSAMPLING_SHIFT_4B, ADC_OVERSAMPLING_RATIO_MUL16);
```

### 函数 adc\_oversample\_mode\_enable

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

表 3-34. 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 3-35. 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 3-36. 函数 adc\_flag\_get

函数名称	adc_flag_get
函数原形	FlagStatus adc_flag_get(uint32_t flag);

功能描述	获取ADC标志位
先决条件	-
被调用函数	-
输入参数{in}	
flag	ADC标志位
ADC_FLAG_WD0E	模拟看门狗0事件标志位
ADC_FLAG_WD1E	模拟看门狗1事件标志位
ADC_FLAG_WD2E	模拟看门狗2事件标志位
ADC_FLAG_EOC	序列转换结束标志位
ADC_FLAG_EOIC	注入序列转换结束标志位
ADC_FLAG_STIC	注入序列转换开始标志位
ADC_FLAG_STRC	常规序列转换开始标志位
输出参数{out}	
-	-
返回值	
FlagStatus	SET或RESET

例如：

```
/* get the ADC analog watchdog 0 flag */
```

```
FlagStatus flag_value;
```

```
flag_value = adc_flag_get(ADC_FLAG_WD0E);
```

### 函数 adc\_flag\_clear

函数 adc\_flag\_clear 描述见下表：

表 3-37. 函数 adc\_flag\_clear

函数名称	adc_flag_clear
函数原形	void adc_flag_clear(uint32_t flag);
功能描述	清除ADC标志位
先决条件	-
被调用函数	-
输入参数{in}	
flag	ADC标志位
ADC_FLAG_WD0E	模拟看门狗0事件标志位
ADC_FLAG_WD1E	模拟看门狗1事件标志位
ADC_FLAG_WD2E	模拟看门狗2事件标志位
ADC_FLAG_EOC	序列转换结束标志位
ADC_FLAG_EOIC	注入序列转换结束标志位
ADC_FLAG_STIC	注入序列转换开始标志位
ADC_FLAG_STRC	常规序列转换开始标志位
输出参数{out}	

-	-
返回值	
-	-

例如：

```
/* clear the ADC analog watchdog 0 flag */
```

```
adc_flag_clear(ADC_FLAG_WD0E);
```

### 函数 `adc_interrupt_enable`

函数 `adc_interrupt_enable` 描述见下表：

表 3-38. 函数 `adc_interrupt_enable`

函数名称	<code>adc_interrupt_enable</code>
函数原形	<code>void adc_interrupt_enable(uint32_t interrupt);</code>
功能描述	使能ADC中断
先决条件	-
被调用函数	-
输入参数{in}	
<b>interrupt</b>	ADC中断
<code>ADC_INT_WD0E</code>	模拟看门狗0中断
<code>ADC_INT_WD1E</code>	模拟看门狗1中断
<code>ADC_INT_WD2E</code>	模拟看门狗2中断
<code>ADC_INT_EOC</code>	序列转换结束中断
<code>ADC_INT_EOIC</code>	注入序列转换结束中断
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* enable ADC analog watchdog 0 interrupt */
```

```
adc_interrupt_enable(ADC_INT_WD0E);
```

### 函数 `adc_interrupt_disable`

函数 `adc_interrupt_disable` 描述见下表：

表 3-39. 函数 `adc_interrupt_disable`

函数名称	<code>adc_interrupt_disable</code>
函数原形	<code>void adc_interrupt_disable(uint32_t interrupt);</code>
功能描述	禁能ADC中断
先决条件	-

被调用函数	-
输入参数{in}	
interrupt	ADC中断
ADC_INT_WD0E	模拟看门狗0中断
ADC_INT_WD1E	模拟看门狗1中断
ADC_INT_WD2E	模拟看门狗2中断
ADC_INT_EOC	序列转换结束中断
ADC_INT_EOIC	注入序列转换结束中断
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* disable ADC analog watchdog 0 interrupt */
```

```
adc_interrupt_disable(ADC_INT_WD0E);
```

### 函数 adc\_interrupt\_flag\_get

函数 adc\_interrupt\_flag\_get 描述见下表：

表 3-40. 函数 adc\_interrupt\_flag\_get

函数名称	adc_interrupt_flag_get
函数原形	FlagStatus adc_interrupt_flag_get(uint32_t 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_WD0E	模拟看门狗0中断标志位
ADC_INT_FLAG_WD1E	模拟看门狗1中断标志位
ADC_INT_FLAG_WD2E	模拟看门狗2中断标志位
ADC_INT_FLAG_EOC	序列转换结束中断标志位
ADC_INT_FLAG_EOIC	注入序列转换结束中断标志位
输出参数{out}	
-	-
返回值	
FlagStatus	SET或RESET

例如：

```
/* get the ADC analog watchdog 0 interrupt flag */
FlagStatus flag_value;

flag_value = adc_interrupt_flag_get(ADC_INT_FLAG_WD0E);
```

### 函数 `adc_interrupt_flag_clear`

函数 `adc_interrupt_flag_clear` 描述见下表：

表 3-41. 函数 `adc_interrupt_flag_clear`

函数名称	<code>adc_interrupt_flag_clear</code>
函数原形	<code>void adc_interrupt_flag_clear(uint32_t int_flag);</code>
功能描述	清除ADC中断标志位
先决条件	-
被调用函数	-
输入参数{in}	
<code>int_flag</code>	ADC中断标志位
<code>ADC_INT_FLAG_WD0E</code>	模拟看门狗0中断标志位
<code>ADC_INT_FLAG_WD1E</code>	模拟看门狗1中断标志位
<code>ADC_INT_FLAG_WD2E</code>	模拟看门狗2中断标志位
<code>ADC_INT_FLAG_EOC</code>	序列转换结束中断标志位
<code>ADC_INT_FLAG_EOIC</code>	注入序列转换结束中断标志位
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* clear the ADC analog watchdog 0 interrupt flag */
adc_interrupt_flag_clear(ADC_INT_FLAG_WD0E);
```

## 3.3. CMP

通用比较器可独立工作，其输出端可用于 I / O 口，也可和定时器结合使用。比较器可通过模拟信号将 MCU 从低功耗模式中唤醒，在一定的条件下，可将模拟信号作为 TIMER 的触发源。

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

### 3.3.1. 外设寄存器说明

CMP寄存器列表如下表所示:

表 3-42. CMP 寄存器

寄存器名称	寄存器描述
CMP0_CS	CMP0控制状态寄存器
CMP1_CS	CMP1控制状态寄存器

### 3.3.2. 外设库函数说明

CMP库函数列表如下表所示:

表 3-43. CMP 库函数

库函数名称	库函数描述
cmp_deinit	复位CMP
cmp_mode_init	CMP工作模式初始化
cmp_output_init	CMP输出初始化
cmp_blanking_init	CMP消隐功能初始化
cmp_enable	使能CMP
cmp_disable	禁能CMP
cmp_switch_enable	使能CMP开关
cmp_switch_disable	禁能CMP开关
cmp_lock_enable	锁定CMP
cmp_voltage_scaler_enable	使能带隙标量
cmp_voltage_scaler_disable	禁能带隙标量
cmp_scaler_bridge_enable	使能标量桥接
cmp_scaler_bridge_disable	禁能标量桥接
cmp_output_level_get	获取CMP输出状态

#### 枚举类型 cmp\_enum

表 3-44. 枚举类型 cmp\_enum

成员名称	功能描述
CMP0	比较器0
CMP1	比较器1

#### 函数 cmp\_deinit

函数cmp\_deinit描述见下表:

表 3-45. 函数 cmp\_deinit

函数名称	cmp_deinit
函数原型	void cmp_deinit(cmp_enum cmp_periph);
功能描述	复位CMP



先决条件	-
被调用函数	-
输入参数{in}	
cmp_periph	参考枚举 <a href="#">表3-44. 枚举类型cmp_enum</a>
输出参数{out}	
-	-
返回值	
-	-

例如：

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

```
cmp_deinit(CMP0);
```

### 函数 cmp\_mode\_init

函数cmp\_mode\_init描述见下表：

表 3-46. 函数 cmp\_mode\_init

函数名称	cmp_mode_init
函数原型	void cmp_mode_init(cmp_enum cmp_periph, uint32_t operating_mode, uint32_t inverting_input, uint32_t output_hysteresis);
功能描述	CMP工作模式初始化
先决条件	-
被调用函数	-
输入参数{in}	
cmp_periph	参考枚举 <a href="#">表3-44. 枚举类型cmp_enum</a>
输入参数{in}	
operating_mode	速度和功耗运行模式
CMP_MODE_HIGHSPEED	高速/全功耗
CMP_MODE_MIDDLEESPEED	中速/中功耗
CMP_MODE_LOWSPEED	低速/低功耗
CMP_MODE_VERYLOWESPEED	超低速/超低功耗
输入参数{in}	
inverting_input	反向输入源选择
CMP_INVERTING_INPUT_1_4VREFINT	VREFINT *1/4作为输入源
CMP_INVERTING_INPUT_1_2VREFINT	VREFINT *1/2作为输入源
CMP_INVERTING_INPUT_3_4VREFINT	VREFINT *3/4作为输入源

PUT_3_4VREFINT	
CMP_INVERTING_IN PUT_VREFINT	VREFINT作为输入源
CMP_INVERTING_IN PUT_PB2_PB6	PB2作为CMP0输入源或者PB6作为CMP1输入源
CMP_INVERTING_IN PUT_PA0_PA2	PA0作为CMP0输入源或者PA2作为CMP1输入源
CMP_INVERTING_IN PUT_PB1_PB3	PB1作为CMP0输入源或者PB3作为CMP1输入源
CMP_INVERTING_IN PUT_VSSA_PB4	VSSA作为CMP0输入源或者PB4作为CMP1输入源
输入参数{in}	
output_hysteresis	迟滞水平
CMP_HYSTERESIS_ NO	无迟滞
CMP_HYSTERESIS_ LOW	低迟滞
CMP_HYSTERESIS_ MIDDLE	中迟滞
CMP_HYSTERESIS_ HIGH	高迟滞
输出参数{out}	
-	-
返回值	
-	-

例如：

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

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

### 函数 cmp\_output\_init

函数cmp\_output\_init描述见下表：

表 3-47. 函数 cmp\_output\_init

函数名称	cmp_output_init
函数原型	void cmp_output_init(cmp_enum cmp_periph, uint32_t output_polarity);
功能描述	CMP输出初始化
先决条件	-
被调用函数	-
输入参数{in}	
cmp_periph	参考枚举 <a href="#">表3-44. 枚举类型cmp_enum</a>

输入参数{in}	
<b>output_polarity</b>	CMP输出极性
<i>CMP_OUTPUT_POLARITY_INVERTED</i>	输出反相
<i>CMP_OUTPUT_POLARITY_NONINVERTED</i>	输出正相
输出参数{out}	
-	-
返回值	
-	-

例如:

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

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

### 函数 cmp\_blanking\_init

函数cmp\_blanking\_init描述见下表:

表 3-48. 函数 cmp\_blanking\_init

函数名称	cmp_blanking_init
函数原型	void cmp_blanking_init(cmp_enum cmp_periph, uint32_t blanking_source_selection);
功能描述	CMP消隐功能初始化
先决条件	-
被调用函数	-
输入参数{in}	
<b>cmp_periph</b>	参考枚举 <a href="#">表3-44. 枚举类型cmp_enum</a>
输入参数{in}	
<b>blanking_source_selection</b>	消隐源选择
<i>CMP_BLANKING_NONE</i>	无消隐
<i>CMP_BLANKING_TIMER0_OC1</i>	TIMER0_CH1输出比较信号作为消隐源
<i>CMP_BLANKING_TIMER2_OC1</i>	TIMER2_CH1输出比较信号作为消隐源
<i>CMP_BLANKING_TIMER13_OC0</i>	TIMER13_CH0输出比较信号作为消隐源
<i>CMP_BLANKING_TIMER15_OC0</i>	TIMER15_CH0输出比较信号作为消隐源
输出参数{out}	

-	-
返回值	
-	-

例如：

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

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

### 函数 cmp\_enable

函数cmp\_enable描述见下表：

表 3-49. 函数 cmp\_enable

函数名称	cmp_enable
函数原型	void cmp_enable(cmp_enum cmp_periph);
功能描述	使能CMP
先决条件	-
被调用函数	-
输入参数{in}	
cmp_periph	参考枚举 <a href="#">表3-44. 枚举类型cmp_enum</a>
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* enable CMP0 */
```

```
cmp_enable(CMP0);
```

### 函数 cmp\_disable

函数cmp\_disable描述见下表：

表 3-50. 函数 cmp\_disable

函数名称	cmp_disable
函数原型	void cmp_disable(cmp_enum cmp_periph);
功能描述	禁能CMP
先决条件	-
被调用函数	-
输入参数{in}	
cmp_periph	参考枚举 <a href="#">表3-44. 枚举类型cmp_enum</a>
输出参数{out}	
-	-

返回值	
-	-

例如：

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

### 函数 cmp\_switch\_enable

函数cmp\_switch\_enable描述见下表：

表 3-51. 函数 cmp\_switch\_enable

函数名称	cmp_switch_enable
函数原型	cmp_switch_enable (void)
功能描述	使能CMP开关模式
先决条件	-
被调用函数	-
输入参数{in}	
-	-
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* enable the CMP0 switch */
cmp_switch_enable(CMP0);
```

### 函数 cmp\_switch\_disable

函数cmp\_switch\_disable描述见下表：

表 3-52. 函数 cmp\_switch\_disable

函数名称	cmp_switch_disable
函数原型	void cmp_switch_disable (void)
功能描述	禁能CMP开关模式
先决条件	-
被调用函数	-
输入参数{in}	
-	-
输出参数{out}	
-	-
返回值	

-	-
---	---

例如:

```
/* disable the CMP0 switch */
```

```
cmp_switch_disable(CMP0);
```

### 函数 cmp\_lock\_enable

函数cmp\_lock\_enable描述见下表:

表 3-53. 函数 cmp\_lock\_enable

函数名称	cmp_lock_enable
函数原型	void cmp_lock_enable(cmp_enum cmp_periph);
功能描述	锁定CMP
先决条件	-
被调用函数	-
输入参数{in}	
cmp_periph	参考枚举 <a href="#">表3-44. 枚举类型cmp_enum</a>
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* lock CMP0 register */
```

```
cmp_lock_enable(CMP0);
```

### 函数 cmp\_voltage\_scaler\_enable

函数cmp\_voltage\_scaler\_enable描述见下表:

表 3-54. 函数 cmp\_voltage\_scaler\_enable

函数名称	cmp_voltage_scaler_enable
函数原型	void cmp_voltage_scaler_enable(cmp_enum cmp_periph);
功能描述	使能CMP带隙标量
先决条件	-
被调用函数	-
输入参数{in}	
cmp_periph	参考枚举 <a href="#">表3-44. 枚举类型cmp_enum</a>
输出参数{out}	
-	-
返回值	
-	-

例如：

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

### 函数 cmp\_voltage\_scaler\_disable

函数cmp\_voltage\_scaler\_disable描述见下表：

**表 3-55. 函数 cmp\_voltage\_scaler\_disable**

函数名称	cmp_voltage_scaler_disable
函数原型	void cmp_voltage_scaler_disable(cmp_enum cmp_periph);
功能描述	禁能CMP带隙标量
先决条件	-
被调用函数	-
输入参数{in}	
cmp_periph	参考枚举 <a href="#">表3-44. 枚举类型cmp_enum</a>
输出参数{out}	
-	-
返回值	
-	-

例如：

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

### 函数 cmp\_scaler\_bridge\_enable

函数cmp\_voltage\_scaler\_bridge\_enable描述见下表：

**表 3-56. 函数 cmp\_scaler\_bridge\_enable**

函数名称	cmp_scaler_bridge_enable
函数原型	void cmp_scaler_bridge_enable(cmp_enum cmp_periph);
功能描述	使能CMP标量桥接
先决条件	-
被调用函数	-
输入参数{in}	
cmp_periph	参考枚举 <a href="#">表3-44. 枚举类型cmp_enum</a>
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* enable CMP0 the scaler bridge */
```

```
cmp_scaler_bridge_enable(CMP0);
```

### 函数 cmp\_scaler\_bridge\_disable

函数cmp\_voltage\_scaler\_bridge\_disable描述见下表：

**表 3-57. 函数 cmp\_scaler\_bridge\_disable**

函数名称	cmp_scaler_bridge_disable
函数原型	void cmp_scaler_bridge_disable(cmp_enum cmp_periph);
功能描述	禁能CMP标量桥接
先决条件	-
被调用函数	-
输入参数{in}	
cmp_periph	参考枚举 <a href="#">表3-44. 枚举类型cmp_enum</a>
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* disable CMP0 the scaler bridge */
```

```
cmp_scaler_bridge_disable(CMP0);
```

### 函数 cmp\_output\_level\_get

函数cmp\_output\_level\_get描述见下表：

**表 3-58. 函数 cmp\_output\_level\_get**

函数名称	cmp_output_level_get
函数原型	uint32_t cmp_output_level_get(cmp_enum cmp_periph);
功能描述	获取CMP输出状态
先决条件	-
被调用函数	-
输入参数{in}	
cmp_periph	参考枚举 <a href="#">表3-44. 枚举类型cmp_enum</a>
输出参数{out}	
-	-
返回值	
uint32_t	输出电平
CMP_OUTPUTLEV_EL_HIGH	比较器输出高电平
CMP_OUTPUTLEV	比较器输出低电平



<i>EL_LOW</i>	
---------------	--

例如:

```
uint32_t level;
```

```
/* get CMP0 output level */
```

```
level = cmp_output_level_get(CMP0);
```

### 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 3-59. CRC 寄存器

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

#### 3.4.2. 外设库函数说明

CRC库函数列表如下表所示:

表 3-60. CRC 库函数

库函数名称	库函数描述
crc_deinit	复位CRC计算单元
crc_init_data_register_write	写初值寄存器
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_reverse_output_data_disable	失能输出数据翻转功能
crc_reverse_output_data_enable	使能输出数据翻转功能
crc_input_data_reverse_config	配置输入数据翻转功能
crc_data_register_reset	根据数据寄存器的复位值复位数据寄存器
crc_polynomial_size_set	配置多项式长度
crc_polynomial_set	设置多项式寄存器数据

库函数名称	库函数描述
crc_single_data_calculate	CRC计算单个数据
crc_block_data_calculate	CRC计算一个数组

### 函数 crc\_deinit

函数crc\_deinit描述见下表：

**表 3-61. 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\_init\_data\_register\_write

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

**表 3-62. 函数 crc\_init\_data\_register\_write**

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

例如：

```
/* write crc initializaiton data register */
```

```
crc_init_data_register_write(0x11223344);
```

### 函数 `crc_data_register_read`

函数 `crc_data_register_read` 描述见下表：

**表 3-63. 函数 `crc_data_register_read`**

函数名称	<code>crc_data_register_read</code>
函数原形	<code>uint32_t crc_data_register_read(void);</code>
功能描述	读数据寄存器
先决条件	-
被调用函数	-
输入参数{in}	
-	-
输出参数{out}	
-	-
返回值	
<code>uint32_t</code>	从数据寄存器读取的32位数据（0-0xFFFFFFFF）

例如：

```
/* read crc data register */

uint32_t crc_value = 0;

crc_value = crc_data_register_read();
```

### 函数 `crc_free_data_register_read`

函数 `crc_free_data_register_read` 描述见下表：

**表 3-64. 函数 `crc_free_data_register_read`**

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

例如：

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

```
uint8_t crc_value = 0;
```

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

### 函数 `crc_free_data_register_write`

函数 `crc_free_data_register_write` 描述见下表：

**表 3-65. 函数 `crc_free_data_register_write`**

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

例如：

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

```
crc_free_data_register_write(0x11);
```

### 函数 `crc_reverse_output_data_disable`

函数 `crc_reverse_output_data_disable` 描述见下表：

**表 3-66. 函数 `crc_reverse_output_data_disable`**

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

例如：

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

```
crc_reverse_output_data_disable();
```

### 函数 `crc_reverse_output_data_enable`

函数 `crc_reverse_output_data_enable` 描述见下表：

**表 3-67. 函数 `crc_reverse_output_data_enable`**

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

例如：

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

```
crc_reverse_output_data_enable();
```

### 函数 `crc_input_data_reverse_config`

函数 `crc_input_data_reverse_config` 描述见下表：

**表 3-68. 函数 `crc_input_data_reverse_config`**

函数名称	<code>crc_input_data_reverse_config</code>
函数原形	<code>void crc_input_data_reverse_config(uint32_t data_reverse)</code>
功能描述	配置输入数据翻转功能
先决条件	-
被调用函数	-
输入参数{in}	
<b>data_reverse</b>	设定的输入数据翻转功能
<code>CRC_INPUT_DATA_NOT</code>	输入数据不翻转
<code>CRC_INPUT_DATA_BYTE</code>	输入数据按字节翻转
<code>CRC_INPUT_DATA_HALFWORD</code>	输入数据按半字翻转
<code>CRC_INPUT_DATA_WORD</code>	输入数据按字翻转
输出参数{out}	

-	-
返回值	
-	-

例如：

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

```
crc_input_data_reverse_config(CRC_INPUT_DATA_WORD);
```

### 函数 `crc_data_register_reset`

函数 `crc_data_register_reset` 描述见下表：

**表 3-69. 函数 `crc_data_register_reset`**

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

例如：

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

```
crc_data_register_reset();
```

### 函数 `crc_polynomial_size_set`

函数 `crc_polynomial_size_set` 描述见下表：

**表 3-70. 函数 `crc_polynomial_size_set`**

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

CRC_CTL_PS_8	8位多项式值用于CRC计算
CRC_CTL_PS_7	7位多项式值用于CRC计算
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* configure the CRC polynomial size*/
crc_polynomial_size_set(CRC_CTL_PS_7);
```

### 函数 crc\_polynomial\_set

函数crc\_polynomial\_set描述见下表：

表 3-71. 函数 crc\_polynomial\_set

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

例如：

```
/* configure the CRC polynomial value */
crc_polynomial_set(0x11223344);
```

### 函数 crc\_single\_data\_calculate

函数crc\_single\_data\_calculate描述见下表：

表 3-72. 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, uint8_t data_format);
功能描述	CRC计算单个数据
先决条件	-
被调用函数	-
输入参数{in}	

<b>sdata</b>	设定的输入数据
<b>输入参数{in}</b>	
<b>data_format</b>	数据格式
<i>INPUT_FORMAT_WORD</i>	输入数据格式为字
<i>INPUT_FORMAT_HALFWORD</i>	输入数据格式为半字
<i>INPUT_FORMAT_BYTE</i>	输入数据格式为字节
<b>输出参数{out}</b>	
-	-
<b>返回值</b>	
<b>uint32_t</b>	CRC计算结果（0-0xFFFFFFFF）

例如：

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

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

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

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

### 函数 **crc\_block\_data\_calculate**

函数 **crc\_block\_data\_calculate** 描述见下表：

**表 3-73. 函数 **crc\_block\_data\_calculate****

<b>函数名称</b>	<b>crc_block_data_calculate</b>
<b>函数原形</b>	<code>uint32_t crc_block_data_calculate(void *array, uint32_t size, uint8_t data_format);</code>
<b>功能描述</b>	CRC计算一个数组
<b>先决条件</b>	-
<b>被调用函数</b>	-
<b>输入参数{in}</b>	
<b>array</b>	指向输入数组
<b>输入参数{in}</b>	
<b>size</b>	数据长度
<b>输入参数{in}</b>	
<b>data_format</b>	数据格式
<i>INPUT_FORMAT_WORD</i>	输入数据格式为字
<i>INPUT_FORMAT_HALFWORD</i>	输入数据格式为半字
<i>INPUT_FORMAT_BYTE</i>	输入数据格式为字节



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

例如：

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

#define BUFFER_SIZE    6

uint32_t valcrc = 0;

static const uint32_t data_buffer[BUFFER_SIZE] = {

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

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

## 3.5. 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 3-74. 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

### 3.5.2. 外设库函数说明

DBG库函数列表如下表所示：

表 3-75. DBG 库函数

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

## 枚举类型 dbg\_periph\_enum

表 3-76. 枚举类型 dbg\_periph\_enum

成员名称	功能描述
DBG_FWDGT_HOLD	当内核停止时，保持FWDGT计数器时钟
DBG_WWDGT_HOLD	当内核停止时，保持WWDGT计数器时钟
DBG_TIMER0_HOLD	当内核停止时，保持TIMER0计数器计数值不变
DBG_TIMER2_HOLD	当内核停止时，保持TIMER2计数器计数值不变
DBG_TIMER13_HOLD	当内核停止时，保持TIMER13计数器计数值不变
DBG_I2C0_HOLD	当内核停止时，保持I2C0的SMBUS状态不变，用于调试
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
DBG_RTC_HOLD	当内核停止时，保持RTC计数器，用于调试

## 函数 dbg\_deinit

函数dbg\_deinit描述见下表：

表 3-77. 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 3-78. 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 3-79. 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 3-80. 函数 dbg\_low\_power\_disable**

函数名称	dbg_low_power_disable
函数原形	void dbg_low_power_disable(uint32_t dbg_low_power);
功能描述	禁能低功耗模式的MCU调试保持功能
先决条件	-
被调用函数	-
<b>输入参数{in}</b>	
dbg_low_power	低功耗模式调试保持
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
<b>输出参数{out}</b>	
-	-
<b>返回值</b>	
-	-

例如：

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

```
dbg_low_power_disable(DBG_LOW_POWER_SLEEP);
```

**函数 dbg\_periph\_enable**

函数dbg\_periph\_enable描述见下表：

**表 3-81. 函数 dbg\_periph\_enable**

函数名称	dbg_periph_enable
函数原形	void dbg_periph_enable(dbg_periph_enum dbg_periph);
功能描述	使能外设的MCU调试保持功能
先决条件	-
被调用函数	-
<b>输入参数{in}</b>	
dbg_periph	参考枚举变量 <a href="#">表3-76. 枚举类型dbg_periph_enum</a>
DBG_FWDGT_HOLD	当内核停止时，保持FWDGT计数器时钟
DBG_WWDGT_HOLD	当内核停止时，保持WWDGT计数器时钟

LD	
DBG_TIMERx_HOLD	当内核停止时，保持TIMERx计数器计数值不变（x=0, 2, 13, 15, 16）
DBG_I2C0_HOLD	当内核停止时，保持I2C0的SMBUS状态不变，用于调试
DBG_RTC_HOLD	当内核停止时，保持RTC计数器，用于调试
输出参数{out}	
-	-
返回值	
-	-

例如：

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

```
dbg_periph_enable(DBG_TIMER2_HOLD);
```

### 函数 dbg\_periph\_disable

函数dbg\_periph\_disable描述见下表：

表 3-82. 函数 dbg\_periph\_disable

函数名称	dbg_periph_disable
函数原形	void dbg_periph_disable(dbg_periph_enum dbg_periph);
功能描述	禁能外设的MCU调试保持功能
先决条件	-
被调用函数	-
输入参数{in}	
dbg_periph	参考枚举变量 <a href="#">表3-76. 枚举类型dbg_periph_enum</a>
DBG_FWDGT_HOLD	当内核停止时，保持FWDGT计数器时钟
DBG_WWDGT_HOLD	当内核停止时，保持WWDGT计数器时钟
DBG_TIMERx_HOLD	当内核停止时，保持TIMERx计数器计数值不变（x=0, 2, 13, 15, 16）
DBG_I2Cx_HOLD	当内核停止时，保持I2C0的SMBUS状态不变，用于调试
DBG_RTC_HOLD	当内核停止时，保持RTC计数器，用于调试
输出参数{out}	
-	-
返回值	
-	-

例如：

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

```
dbg_periph_disable(DBG_TIMER2_HOLD);
```

## 3.6. DMA/DMAMUX

DMA控制器提供了一种硬件的方式在外设和存储器之间或者存储器和存储器之间传输数据，而无需CPU的介入，从而使CPU可以专注在处理其他系统功能上。章节[3.6.1](#)描述了DMA的寄存器列表，章节[3.6.2](#)对DMA库函数进行说明。

DMAMUX是DMA请求的传输调度器。可编程的DMA请求多路复用器DMAMUX，可在外设和DMA控制器之间路由DMA请求线路，或者DMAMUX也可以将可编程事件连入到输入触发信号上，作为一个DMAMUX请求发生器，再由DMAMUX请求路由器在DMAMUX请求生成器产生的DMA请求和DMA控制器之间路由DMA请求线路。章节[3.6.1](#)描述了DMAMUX的寄存器列表，章节[3.6.2](#)对DMAMUX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 3-83. DMA 寄存器

寄存器名称	寄存器描述
DMA_INTF	中断标志位寄存器
DMA_INTC	中断标志位清除寄存器
DMA_CHxCTL (x=0..2)	通道x控制寄存器
DMA_CHxCNT (x=0..2)	通道x计数寄存器
DMA_CHxPADDR (x=0..2)	通道x外设基地址寄存器
DMA_CHxMADDR (x=0..2)	通道x存储器基地址寄存器

DMAMUX寄存器列表如下表所示：

表 3-84. DMAMUX 寄存器

寄存器名称	寄存器描述
DMAMUX_RM_CHx CFG (x=0..2)	请求路由通道x配置寄存器
DMAMUX_RM_INT F	请求路由通道中断标志位寄存器
DMAMUX_RM_INT C	请求路由通道中断标志位清除寄存器
DMAMUX_RG_CHx CFG (x=0..3)	请求生成通道x配置寄存器
DMAMUX_RG_INT F	请求生成通道中断标志位寄存器
DMAMUX_RG_INT	请求生成通道中断标志位清除寄存器

寄存器名称	寄存器描述
C	

### 3.6.2. 外设库函数说明

DMA库函数列表如下表所示：

**表 3-85. DMA 库函数**

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

DMAMUX库函数列表如下表所示：

**表 3-86. DMAMUX 库函数**

库函数名称	库函数描述
dmamux_sync_struct_para_init	将DMAMUX同步结构体中所有参数初始化为默认值
dmamux_synchronization_init	初始化DMAMUX同步结构体通道x
dmamux_synchronization_enable	使能DMAMUX同步模式

库函数名称	库函数描述
dmamux_synchronization_disable	禁能DMAMUX同步模式
dmamux_event_generation_enable	使能DMAMUX事件输出
dmamux_event_generation_disable	禁能DMAMUX事件输出
dmamux_gen_struct_para_init	将DMAMUX请求生成结构体中所有参数初始化为默认值
dmamux_request_generator_init	初始化DMAMUX请求生成结构体通道x
dmamux_request_generator_channel_enable	使能DMAMUX请求生成通道x
dmamux_request_generator_channel_disable	禁能DMAMUX请求生成通道x
dmamux_synchronization_polarity_config	配置DMAMUX同步输入的有效边沿
dmamux_request_forward_number_config	配置DMAMUX通道x要传输多少个DMA请求
dmamux_sync_id_config	配置DMAMUX同步输入标识
dmamux_request_id_config	配置DMAMUX请求路由通道输入标识
dmamux_trigger_polarity_config	配置DMAMUX触发输入的有效边沿
dmamux_request_generate_number_config	配置DMAMUX请求生成器生成请求的数量
dmamux_trigger_id_config	配置DMAMUX触发输入标识
dmamux_flag_get	获取DMAMUX通道x标志位状态
dmamux_flag_clear	清除DMAMUX通道x标志位状态
dmamux_interrupt_enable	使能DMAMUX通道x中断
dmamux_interrupt_disable	禁能DMAMUX通道x中断
dmamux_interrupt_flag_get	获取DMAMUX通道x中断标志位状态
dmamux_interrupt_flag_clear	清除DMAMUX通道x中断标志位状态

## 结构体 dma\_parameter\_struct

表 3-87. 结构体 dma\_parameter\_struct

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



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

表 3-88. 结构体 dmamux\_sync\_parameter\_struct

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

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

表 3-89. 结构体 dmamux\_gen\_parameter\_struct

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

### 枚举 dmamux\_interrupt\_enum

表 3-90. 枚举 dmamux\_interrupt\_enum

成员名称	功能描述
DMAMUX_INT_MU XCH0_SO	DMAMUX请求路由通道0同步溢出中断
DMAMUX_INT_MU XCH1_SO	DMAMUX请求路由通道1同步溢出中断
DMAMUX_INT_MU XCH2_SO	DMAMUX请求路由通道2同步溢出中断
DMAMUX_INT_GE NCH0_TO	DMAMUX请求生成通道0触发溢出中断
DMAMUX_INT_GE NCH1_TO	DMAMUX请求生成通道1触发溢出中断
DMAMUX_INT_GE NCH2_TO	DMAMUX请求生成通道2触发溢出中断
DMAMUX_INT_GE NCH3_TO	DMAMUX请求生成通道3触发溢出中断

### 枚举 dmamux\_flag\_enum

表 3-91. 枚举 dmamux\_flag\_enum

成员名称	功能描述
DMAMUX_FLAG_M UXCH0_SO	DMAMUX请求路由通道0同步溢出标志
DMAMUX_FLAG_M UXCH1_SO	DMAMUX请求路由通道1同步溢出标志
DMAMUX_FLAG_M	DMAMUX请求路由通道2同步溢出标志

UXCH2_SO	
DMAMUX_FLAG_G ENCH0_TO	DMAMUX请求生成通道0触发溢出标志
DMAMUX_FLAG_G ENCH1_TO	DMAMUX请求生成通道1触发溢出标志
DMAMUX_FLAG_G ENCH2_TO	DMAMUX请求生成通道2触发溢出标志
DMAMUX_FLAG_G ENCH3_TO	DMAMUX请求生成通道3触发溢出标志

### 枚举 dmamux\_interrupt\_flag\_enum

表 3-92. 枚举 dmamux\_interrupt\_flag\_enum

成员名称	功能描述
DMAMUX_INT_FL G_MUXCH0_SO	DMAMUX请求路由通道0同步溢出中断标志
DMAMUX_INT_FL G_MUXCH1_SO	DMAMUX请求路由通道1同步溢出中断标志
DMAMUX_INT_FL G_MUXCH2_SO	DMAMUX请求路由通道2同步溢出中断标志
DMAMUX_INT_FL G_GENCH0_TO	DMAMUX请求生成通道0触发溢出中断标志
DMAMUX_INT_FL G_GENCH1_TO	DMAMUX请求生成通道1触发溢出中断标志
DMAMUX_INT_FL G_GENCH2_TO	DMAMUX请求生成通道2触发溢出中断标志
DMAMUX_INT_FL G_GENCH3_TO	DMAMUX请求生成通道3触发溢出中断标志

### 枚举 dma\_channel\_enum

表 3-93. 枚举 dma\_channel\_enum

成员名称	功能描述
DMA_CH0	DMA通道0
DMA_CH1	DMA通道1
DMA_CH2	DMA通道2

### 枚举 dmamux\_multiplexer\_channel\_enum

表 3-94. 枚举 dmamux\_multiplexer\_channel\_enum

成员名称	功能描述
DMAMUX_MUXCH 0	DMAMUX请求路由通道0
DMAMUX_MUXCH	DMAMUX请求路由通道1

1	
DMAMUX_MUXCH	DMAMUX请求路由通道2
2	

### 枚举 dmamux\_generator\_channel\_enum

表 3-95. 枚举 dmamux\_generator\_channel\_enum

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

### 函数 dma\_deinit

函数 dma\_deinit 描述见下表：

表 3-96. 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DMA通道x的所有寄存器
先决条件	无
被调用函数	无
输入参数{in}	
channelx	DMA通道
DMA_CHx( x=0..2)	DMA通道选择, 参考 <a href="#">表3-93. 枚举dma_channel_enum</a>
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* deinitialize DMA channel0 registers */
dma_deinit(DMA_CH0);
```

### 函数 dma\_struct\_para\_init

函数 dma\_struct\_para\_init 描述见下表：

表 3-97. 函数 dma\_struct\_para\_init

函数名称	dma_struct_para_init
函数原型	void dma_struct_para_init(dma_parameter_struct* init_struct);
功能描述	将DMA结构体中所有参数初始化为默认值
先决条件	无
被调用函数	无

输入参数{in}	
-	-
输出参数{out}	
init_struct	一个已经定义的dma_parameter_struct结构体变量地址，参考 <a href="#">表3-87. 结构体 dma_parameter_struct</a>
返回值	
-	-

例如：

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

### 函数 dma\_init

函数 dma\_init 描述见下表：

**表 3-98. 函数 dma\_init**

函数名称	dma_init
函数原型	void dma_init(dma_channel_enum channelx, dma_parameter_struct* init_struct);
功能描述	初始化DMA通道x
先决条件	无
被调用函数	无
输入参数{in}	
channelx	DMA通道
DMA_CHx( x=0..2)	DMA通道选择，参考 <a href="#">表3-93. 枚举dma_channel_enum</a>
输入参数{in}	
init_struct	一个已经定义的dma_parameter_struct结构体变量地址，参考 <a href="#">表3-87. 结构体 dma_parameter_struct</a>
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* DMA channel0 initialize */
dma_parameter_struct dma_init_struct;

dma_struct_para_init(&dma_init_struct);
dma_init_struct.direction = DMA_PERIPHERAL_TO_MEMORY;
dma_init_struct.memory_addr = (uint32_t)g_destbuf;
dma_init_struct.memory_inc = DMA_MEMORY_INCREASE_ENABLE;
dma_init_struct.memory_width = DMA_MEMORY_WIDTH_8BIT;
```

```

dma_init_struct.number = TRANSFER_NUM;
dma_init_struct.periph_addr = (uint32_t)BANK0_WRITE_START_ADDR;
dma_init_struct.periph_inc = DMA_PERIPH_INCREASE_ENABLE;
dma_init_struct.periph_width = DMA_PERIPHERAL_WIDTH_8BIT;
dma_init_struct.priority = DMA_PRIORITY_ULTRA_HIGH;
dma_init(DMA_CH0, &dma_init_struct);

```

### 函数 dma\_circulation\_enable

函数 dma\_circulation\_enable 描述见下表：

**表 3-99. 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循环模式使能
先决条件	相应通道使能位CHEN需为0
被调用函数	无
输入参数{in}	
channelx	DMA通道
DMA_CHx( x=0..2)	DMA通道选择, 参考 <a href="#">表3-93. 枚举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 3-100. 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循环模式禁能
先决条件	相应通道使能位CHEN需为0
被调用函数	无
输入参数{in}	
channelx	DMA通道
DMA_CHx( x=0..2)	DMA通道选择, 参考 <a href="#">表3-93. 枚举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\_memory\_to\_memory\_enable

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

表 3-101. 函数 dma\_memory\_to\_memory\_enable

函数名称	dma_memory_to_memory_enable
函数原型	void dma_memory_to_memory_enable(dma_channel_enum channelx);
功能描述	存储器到存储器DMA传输使能
先决条件	相应通道使能位CHEN需为0
被调用函数	无
输入参数{in}	
channelx	DMA通道
DMA_CHx( x=0..2)	DMA通道选择，参考 <a href="#">表3-93. 枚举dma_channel_enum</a>
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* enable DMA channel0 memory to memory mode */
dma_memory_to_memory_enable(DMA_CH0);
```

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

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

表 3-102. 函数 dma\_memory\_to\_memory\_disable

函数名称	dma_memory_to_memory_disable
函数原形	void dma_memory_to_memory_disable(dma_channel_enum channelx);
功能描述	存储器到存储器DMA传输禁能
先决条件	相应通道使能位CHEN需为0
被调用函数	无
输入参数{in}	
channelx	DMA通道
DMA_CHx( x=0..2)	DMA通道选择，参考 <a href="#">表3-93. 枚举dma_channel_enum</a>
输出参数{out}	
-	-

返回值	
-	-

例如：

```
/* disable DMA channel0 memory to memory mode */
dma_memory_to_memory_disable(DMA_CH0);
```

### 函数 dma\_channel\_enable

函数 dma\_channel\_enable 描述见下表：

表 3-103. 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	DMA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..2)	DMA通道选择，参考 <a href="#">表3-93. 枚举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 3-104. 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	DMA通道x传输禁能
先决条件	无
被调用函数	无
输入参数{in}	
channelx	DMA通道
DMA_CHx( x=0..2)	DMA通道选择，参考 <a href="#">表3-93. 枚举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\_periph\_address\_config

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

表 3-105. 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	相应通道使能位CHEN需为0
被调用函数	无
输入参数{in}	
channelx	DMA通道
DMA_CHx( x=0..2)	DMA通道选择，参考 <a href="#">表3-93. 枚举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 periph address */
#define BANK0_WRITE_START_ADDR ((uint32_t)0x08004000)
dma_periph_address_config(DMA_CH0, BANK0_WRITE_START_ADDR);
```

### 函数 dma\_memory\_address\_config

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

表 3-106. 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, uint32_t address);
功能描述	DMA通道x传输的存储器基地址配置
先决条件	相应通道使能位CHEN需为0
被调用函数	无



输入参数{in}	
<b>channelx</b>	DMA通道
<i>DMA_CHx( x=0..2)</i>	DMA通道选择, 参考 <a href="#">表3-93. 枚举dma_channel_enum</a>
输入参数{in}	
<b>address</b>	存储器基地址, 0 – 0xFFFFFFFF
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* configure DMA channel0 memory address */
uint8_t g_destbuf[TRANSFER_NUM];
dma_memory_address_config(DMA_CH0, (uint32_t) g_destbuf);
```

### 函数 dma\_transfer\_number\_config

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

**表 3-107. 函数 dma\_transfer\_number\_config**

<b>函数名称</b>	dma_transfer_number_config
<b>函数原型</b>	void dma_transfer_number_config(dma_channel_enum channelx, uint32_t number);
<b>功能描述</b>	配置DMA通道x还有多少数据要传输
<b>先决条件</b>	相应通道使能位CHEN需为0
<b>被调用函数</b>	无
输入参数{in}	
<b>channelx</b>	DMA通道
<i>DMA_CHx( x=0..2)</i>	DMA通道选择, 参考 <a href="#">表3-93. 枚举dma_channel_enum</a>
输入参数{in}	
<b>number</b>	数据传输数量 (0x0 – 0xFFFF)
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* configure 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 3-108. 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还有多少数据要传输
先决条件	相应通道使能位CHEN需为0
被调用函数	无
输入参数{in}	
channelx	DMA通道
DMA_CHx( x=0..2)	DMA通道选择, 参考 <a href="#">表3-93. 枚举dma_channel_enum</a>
输出参数{out}	
-	-
返回值	
uint32_t	DMA数据传输剩余数量 (0x0 – 0xFFFF)

例如:

```
/* get DMA channel0 transfer number */
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 3-109. 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	相应通道使能位CHEN需为0
被调用函数	无
输入参数{in}	
channelx	DMA通道
DMA_CHx( x=0..2)	DMA通道选择, 参考 <a href="#">表3-93. 枚举dma_channel_enum</a>
输入参数{in}	
priority	DMA通道软件优先级
DMA_PRIORITY_LOW	低优先级
DMA_PRIORITY_MEDIUM	中优先级
DMA_PRIORITY_HIGH	高优先级
DMA_PRIORITY_ULTRA_HIGH	极高优先级

输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* configure DMA channel0 priority */
dma_priority_config(DMA_CH0, DMA_PRIORITY_ULTRA_HIGH);
```

### 函数 dma\_memory\_width\_config

函数 dma\_memory\_width\_config 描述见下表：

表 3-110. 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 mwidth);
功能描述	DMA通道x传输的存储器数据宽度配置
先决条件	相应通道使能位CHEN需为0
被调用函数	无
输入参数{in}	
channelx	DMA通道
DMA_CHx( x=0..2)	DMA通道选择，参考 <a href="#">表3-93. 枚举dma_channel_enum</a>
输入参数{in}	
mwidth	存储器数据传输宽度
DMA_MEMORY_WIDTH_8BIT	8位数据传输宽度
DMA_MEMORY_WIDTH_16BIT	16位数据传输宽度
DMA_MEMORY_WIDTH_32BIT	32位数据传输宽度
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* configure DMA channel0 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 3-111. 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 pwidth);
功能描述	DMA通道x传输的外设数据宽度配置
先决条件	相应通道使能位CHEN需为0
被调用函数	无
输入参数{in}	
channelx	DMA通道
DMA_CHx( x=0..2)	DMA通道选择, 参考 <a href="#">表3-93. 枚举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 periph width */
dma_periph_width_config(DMA_CH0, DMA_PERIPHERAL_WIDTH_8BIT);
```

### 函数 dma\_memory\_increase\_enable

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

表 3-112. 函数 dma\_memory\_increase\_enable

函数名称	dma_memory_increase_enable
函数原型	void dma_memory_increase_enable(dma_channel_enum channelx);
功能描述	DMA通道x传输的存储器地址生成算法增量模式使能
先决条件	相应通道使能位CHEN需为0
被调用函数	无
输入参数{in}	
channelx	DMA通道
DMA_CHx( x=0..2)	DMA通道选择, 参考 <a href="#">表3-93. 枚举dma_channel_enum</a>
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* enable DMA channel0 memory increase */
dma_memory_increase_enable(DMA_CH0);
```

### 函数 dma\_memory\_increase\_disable

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

表 3-113. 函数 dma\_memory\_increase\_disable

函数名称	dma_memory_increase_disable
函数原型	void dma_memory_increase_disable(dma_channel_enum channelx);
功能描述	DMA通道x传输的存储器地址生成算法增量模式禁能
先决条件	相应通道使能位CHEN需为0
被调用函数	无
输入参数{in}	
channelx	DMA通道
DMA_CHx( x=0..2)	DMA通道选择, 参考 <a href="#">表3-93. 枚举dma_channel_enum</a>
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* disable DMA channel0 memory increase */
dma_memory_increase_disable(DMA_CH0);
```

### 函数 dma\_periph\_increase\_enable

函数 dma\_periph\_increase\_enable 描述见下表：

表 3-114. 函数 dma\_periph\_increase\_enable

函数名称	dma_periph_increase_enable
函数原型	void dma_periph_increase_enable(dma_channel_enum channelx);
功能描述	DMA通道x传输的外设地址生成算法增量模式使能
先决条件	相应通道使能位CHEN需为0
被调用函数	无
输入参数{in}	
channelx	DMA通道
DMA_CHx( x=0..2)	DMA通道选择, 参考 <a href="#">表3-93. 枚举dma_channel_enum</a>
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* enable DMA channel0 periph increase*/
dma_periph_increase_enable(DMA_CH0);
```

### 函数 dma\_periph\_increase\_disable

函数 dma\_periph\_increase\_disable 描述见下表：

表 3-115. 函数 dma\_periph\_increase\_disable

函数名称	dma_periph_increase_disable
函数原型	void dma_periph_increase_disable(dma_channel_enum channelx);
功能描述	DMA通道x传输的外设地址生成算法增量模式禁能
先决条件	相应通道使能位CHEN需为0
被调用函数	无
输入参数{in}	
channelx	DMA通道
DMA_CHx( x=0..2)	DMA通道选择，参考 <a href="#">表3-93. 枚举dma_channel_enum</a>
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* disable DMA channel0 periph increase*/
dma_periph_increase_disable(DMA_CH0);
```

### 函数 dma\_transfer\_direction\_config

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

表 3-116. 函数 dma\_transfer\_direction\_config

函数名称	dma_transfer_direction_config
函数原型	void dma_transfer_direction_config(dma_channel_enum channelx, uint32_t direction);
功能描述	DMA通道x的传输方向配置
先决条件	相应通道使能位CHEN需为0
被调用函数	无
输入参数{in}	
channelx	DMA通道
DMA_CHx( x=0..2)	DMA通道选择，参考 <a href="#">表3-93. 枚举dma_channel_enum</a>
输入参数{in}	
direction	数据传输方向
DMA_PERIPHERAL_TO_MEMORY	读取外设中数据，写入存储器

DMA_MEMORY_TO_PERIPHERAL	读取存储器中数据，写入外设
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* configure DMA channel0 transfer direction*/
dma_transfer_direction_config(DMA_CH0, DMA_PERIPHERAL_TO_MEMORY);
```

### 函数 dma\_flag\_get

函数 dma\_flag\_get 描述见下表：

表 3-117. 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x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..2)	DMA通道选择，参考 <a href="#">表3-93. 枚举dma_channel_enum</a>
输入参数{in}	
flag	DMA标志
DMA_FLAG_G	DMA通道全局中断标志
DMA_FLAG_FTF	DMA通道传输完成标志
DMA_FLAG_HTF	DMA通道半传输完成标志
DMA_FLAG_ERR	DMA通道错误标志
输出参数{out}	
-	-
返回值	
FlagStatus	SET或RESET

例如：

```
/* get DMA channel0 flag*/
FlagStatus flag = RESET;
flag = dma_flag_get(DMA_CH0, DMA_FLAG_FTF);
```

### 函数 dma\_flag\_clear

函数 dma\_flag\_clear 描述见下表：

表 3-118. 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DMA通道x标志位状态
先决条件	无
被调用函数	无
输入参数{in}	
channelx	DMA通道
DMA_CHx( x=0..2)	DMA通道选择, 参考 <a href="#">表3-93. 枚举dma_channel_enum</a>
输入参数{in}	
flag	DMA标志
DMA_FLAG_G	DMA通道全局中断标志
DMA_FLAG_FTF	DMA通道传输完成标志
DMA_FLAG_HTF	DMA通道半传输完成标志
DMA_FLAG_ERR	DMA通道错误标志
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* clear DMA channel0 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 3-119. 函数 dma\_interrupt\_enable

函数名称	dma_interrupt_enable
函数原型	void dma_interrupt_enable(dma_channel_enum channelx, uint32_t source);
功能描述	DMA通道x中断使能
先决条件	无
被调用函数	无
输入参数{in}	
channelx	DMA通道
DMA_CHx( x=0..2)	DMA通道选择, 参考 <a href="#">表3-93. 枚举dma_channel_enum</a>
输入参数{in}	
source	DMA中断源
DMA_INT_FTF	DMA通道传输完成中断
DMA_INT_HTF	DMA通道半传输完成中断
DMA_INT_ERR	DMA通道错误中断
输出参数{out}	



-	-
返回值	
-	-

例如：

```
/* enable DMA channel0 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 3-120. 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 source);
功能描述	DMA通道x中断禁能
先决条件	无
被调用函数	无
输入参数{in}	
channelx	DMA通道
DMA_CHx( x=0..2)	DMA通道选择，参考 <a href="#">表3-93. 枚举dma_channel_enum</a>
输入参数{in}	
source	DMA中断源
DMA_INT_FTF	DMA通道传输完成中断
DMA_INT_HTF	DMA通道半传输完成中断
DMA_INT_ERR	DMA通道错误中断
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* disable DMA channel0 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 3-121. 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 flag);
功能描述	获取DMA通道x中断标志位状态

先决条件	无
被调用函数	无
输入参数{in}	
channelx	DMA通道
DMA_CHx( x=0..2)	DMA通道选择, 参考 <a href="#">表3-93. 枚举dma_channel_enum</a>
输入参数{in}	
flag	DMA标志
DMA_INT_FLAG_F TF	DMA通道传输完成中断标志
DMA_INT_FLAG_H TF	DMA通道半传输完成中断标志
DMA_INT_FLAG_E RR	DMA通道错误中断标志
输出参数{out}	
-	-
返回值	
FlagStatus	SET或RESET

例如:

```
/* get DMA interrupt flag*/
if(dma_interrupt_flag_get(DMA_CH3, DMA_INT_FLAG_FTF)){
    dma_interrupt_flag_clear(DMA_CH3, DMA_INT_FLAG_G);
}
```

### 函数 dma\_interrupt\_flag\_clear

函数 dma\_interrupt\_flag\_clear 描述见下表:

表 3-122. 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 flag);
功能描述	清除DMA通道x中断标志位状态
先决条件	无
被调用函数	无
输入参数{in}	
channelx	DMA通道
DMA_CHx( x=0..2)	DMA通道选择, 参考 <a href="#">表3-93. 枚举dma_channel_enum</a>
输入参数{in}	
flag	DMA标志
DMA_INT_FLAG_G	DMA通道全局中断标志
DMA_INT_FLAG_FTF	DMA通道传输完成中断标志
DMA_INT_FLAG_HTF	DMA通道半传输完成中断标志

DMA_INT_FLAG_ER R	DMA通道错误中断标志
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* clear DMA interrupt flag*/
if(dma_interrupt_flag_get(DMA_CH3, DMA_INT_FLAG_FTF)){
    dma_interrupt_flag_clear(DMA_CH3, DMA_INT_FLAG_G);
}
```

### 函数 dmamux\_sync\_struct\_para\_init

函数 dmamux\_sync\_struct\_para\_init 描述见下表：

表 3-123. 函数 dmamux\_sync\_struct\_para\_init

函数名称	dmamux_sync_struct_para_init
函数原型	void dmamux_sync_struct_para_init(dmamux_sync_parameter_struct *init_struct);
功能描述	将DMAMUX同步结构体中所有参数初始化为默认值
先决条件	无
被调用函数	无
输入参数{in}	
-	-
输出参数{out}	
init_struct	一个已经定义的dmamux_sync_parameter_struct结构体变量地址，参考 <a href="#">表3-88. 结构体dmamux_sync_parameter_struct</a>
返回值	
-	-

例如：

```
/* initialize DMAMUX synchronization mode structure */
dmamux_sync_parameter_struct dmamux_sync_init_struct;
dmamux_sync_struct_para_init(&dmamux_sync_init_struct);
```

### 函数 dmamux\_synchronization\_init

函数 dmamux\_synchronization\_init 描述见下表：

表 3-124. 函数 dmamux\_synchronization\_init

函数名称	dmamux_synchronization_init
函数原型	void dmamux_synchronization_init(dmamux_multiplexer_channel_enum channelx, dmamux_sync_parameter_struct *init_struct);

功能描述	初始化DMAMUX同步结构体通道x
先决条件	无
被调用函数	无
输入参数{in}	
channelx	指定要初始化的DMAMUX请求路由通道
DMAMUX_MUXCHx(x=0..2)	DMAMUX通道选择, 参考 <a href="#">表3-94. 枚举dmamux_mux_channel_enum</a>
输入参数{in}	
init_struct	一个已经定义的dmamux_sync_parameter_struct结构体变量地址, 参考 <a href="#">表3-88. 结构体dmamux_sync_parameter_struct</a>
输出参数{out}	
-	-
返回值	
-	-

例如:

```

/* initialize DMAMUX synchronization mode structure */
dmamux_sync_parameter_struct dmamux_sync_init_struct;
dmamux_sync_struct_para_init(&dmamux_sync_init_struct);
/* initialize DMA request multiplexer channel 0 with synchronization mode */
dmamux_sync_init_struct.sync_id      = DMAMUX_SYNC_EXTI0;
dmamux_sync_init_struct.sync_polarity = DMAMUX_SYNC_RISING;
dmamux_sync_init_struct.request_number = 4;
dmamux_synchronization_init(DMAMUX_MUXCH0, &dmamux_sync_init_struct);

```

### 函数 dmamux\_synchronization\_enable

函数 dmamux\_synchronization\_enable 描述见下表:

**表 3-125. 函数 dmamux\_synchronization\_enable**

函数名称	dmamux_synchronization_enable
函数原型	void dmamux_synchronization_enable(dmamux_mux_channel_enum channelx);
功能描述	使能DMAMUX同步模式
先决条件	无
被调用函数	无
输入参数{in}	
channelx	指定要初始化的DMAMUX请求路由通道
DMAMUX_MUXCHx(x=0..2)	DMAMUX通道选择, 参考 <a href="#">表3-94. 枚举dmamux_mux_channel_enum</a>
输出参数{out}	
-	-
返回值	

例如:

```
/* enable synchronization mode */
dmamux_synchronization_enable(DMAMUX_MUXCH0);
```

### 函数 dmamux\_synchronization\_disable

函数 dmamux\_synchronization\_disable 描述见下表:

表 3-126. 函数 dmamux\_synchronization\_disable

函数名称	dmamux_synchronization_disable
函数原型	void dmamux_synchronization_disable(dmamux_multiplexer_channel_enum channelx);
功能描述	禁能DMAMUX同步模式
先决条件	无
被调用函数	无
输入参数{in}	
channelx	指定要初始化的DMAMUX请求路由通道
DMAMUX_MUXCHx(x=0..2)	DMAMUX通道选择, 参考 <a href="#">表3-94. 枚举dmamux_multiplexer_channel_enum</a>
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* disable synchronization mode */
dmamux_synchronization_disable(DMAMUX_MUXCH0);
```

### 函数 dmamux\_event\_generation\_enable

函数 dmamux\_event\_generation\_enable 描述见下表:

表 3-127. 函数 dmamux\_event\_generation\_enable

函数名称	dmamux_event_generation_enable
函数原型	void dmamux_event_generation_enable(dmamux_multiplexer_channel_enum channelx);
功能描述	使能DMAMUX事件输出
先决条件	无
被调用函数	无
输入参数{in}	
channelx	指定要初始化的DMAMUX请求路由通道
DMAMUX_MUXCHx	DMAMUX通道选择, 参考 <a href="#">表3-94. 枚举dmamux_multiplexer_channel_enum</a>

x(x=0..2)	
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* enable event generation */
dmamux_event_generation_enable(DMAMUX_MUXCH0);
```

### 函数 dmamux\_event\_generation\_disable

函数 dmamux\_event\_generation\_disable 描述见下表:

表 3-128. 函数 dmamux\_event\_generation\_disable

函数名称	dmamux_event_generation_disable
函数原型	void dmamux_event_generation_disable(dmamux_multiplexer_channel_enum channelx);
功能描述	禁能DMAMUX事件输出
先决条件	无
被调用函数	无
输入参数{in}	
channelx	指定要初始化的DMAMUX请求路由通道
DMAMUX_MUXCH x(x=0..2)	DMAMUX通道选择, 参考 <a href="#">表3-94. 枚举dmamux_multiplexer_channel_enum</a>
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* disable event generation */
dmamux_event_generation_disable(DMAMUX_MUXCH0);
```

### 函数 dmamux\_gen\_struct\_para\_init

函数 dmamux\_gen\_struct\_para\_init 描述见下表:

表 3-129. 函数 dmamux\_gen\_struct\_para\_init

函数名称	dmamux_gen_struct_para_init
函数原型	void dmamux_gen_struct_para_init(dmamux_gen_parameter_struct *init_struct);
功能描述	将DMAMUX请求生成结构体中所有参数初始化为默认值
先决条件	无

被调用函数	无
输入参数{in}	
-	-
输出参数{out}	
init_struct	一个已经定义的dmamux_gen_parameter_struct结构体变量地址，参考 <a href="#">表3-89. 结构体dmamux_gen_parameter_struct</a>
返回值	
-	-

例如：

```
/* initialize DMA request generator structure */
dmamux_gen_parameter_struct    dmamux_gen_init_struct;
dmamux_gen_struct_para_init(&dmamux_gen_init_struct);
```

### 函数 dmamux\_request\_generator\_init

函数 dmamux\_request\_generator\_init 描述见下表：

**表 3-130. 函数 dmamux\_request\_generator\_init**

函数名称	dmamux_request_generator_init
函数原型	void dmamux_request_generator_init(dmamux_generator_channel_enum channelx, dmamux_gen_parameter_struct *init_struct);
功能描述	初始化DMAMUX请求生成结构体通道x
先决条件	无
被调用函数	无
输入参数{in}	
channelx	指定要初始化的DMAMUX请求生成通道
DMAMUX_GENCHx (x=0..3)	DMAMUX请求生成通道选择，参考 <a href="#">表3-95. 枚举 dmamux_generator_channel_enum</a>
输入参数{in}	
init_struct	一个已经定义的dmamux_gen_parameter_struct结构体变量地址，参考 <a href="#">表3-89. 结构体dmamux_gen_parameter_struct</a>
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* initialize DMA request generator channel 0 */
dmamux_gen_parameter_struct    dmamux_gen_init_struct;
dmamux_gen_struct_para_init(&dmamux_gen_init_struct);
dmamux_gen_init_struct.trigger_id    = DMAMUX_TRIGGER_EXTI13;
dmamux_gen_init_struct.trigger_polarity = DMAMUX_GEN_RISING;
dmamux_gen_init_struct.request_number = 1;
```

```
dmamux_request_generator_init(DMAMUX_GENCH0, &dmamux_gen_init_struct);
```

### 函数 dmamux\_request\_generator\_channel\_enable

函数 dmamux\_request\_generator\_channel\_enable 描述见下表：

**表 3-131. 函数 dmamux\_request\_generator\_channel\_enable**

函数名称	dmamux_request_generator_channel_enable
函数原型	void dmamux_request_generator_channel_enable(dmamux_generator_channel_enum channelx);
功能描述	使能DMAMUX请求生成通道x
先决条件	无
被调用函数	无
输入参数{in}	
channelx	指定要初始化的DMAMUX请求生成通道
DMAMUX_GENCHx (x=0..3)	DMAMUX请求生成通道选择，参考 <a href="#">表3-95. 枚举 dmamux_generator_channel_enum</a>
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* enable DMAMUX request generator channel */
dmamux_request_generator_channel_enable(DMAMUX_GENCH0);
```

### 函数 dmamux\_request\_generator\_channel\_disable

函数 dmamux\_request\_generator\_channel\_disable 描述见下表：

**表 3-132. 函数 dmamux\_request\_generator\_channel\_disable**

函数名称	dmamux_request_generator_channel_disable
函数原型	void dmamux_request_generator_channel_disable(dmamux_generator_channel_enum channelx);
功能描述	禁能DMAMUX请求生成通道x
先决条件	无
被调用函数	无
输入参数{in}	
channelx	指定要初始化的DMAMUX请求生成通道
DMAMUX_GENCHx (x=0..3)	DMAMUX请求生成通道选择，参考 <a href="#">表3-95. 枚举 dmamux_generator_channel_enum</a>
输出参数{out}	



-	-
返回值	
-	-

例如：

```
/* disable DMAMUX request generator channel */
dmamux_request_generator_channel_disable(DMAMUX_GENCH0);
```

### 函数 dmamux\_synchronization\_polarity\_config

函数 dmamux\_synchronization\_polarity\_config 描述见下表：

**表 3-133. 函数 dmamux\_synchronization\_polarity\_config**

函数名称	dmamux_synchronization_polarity_config
函数原型	void dmamux_synchronization_polarity_config(dmamux_multiplexer_channel_enum channelx, uint32_t polarity);
功能描述	配置DMAMUX同步输入的有效边沿
先决条件	无
被调用函数	无
输入参数{in}	
channelx	指定要初始化的DMAMUX请求路由通道
DMAMUX_MUXCHx(x=0..2)	DMAMUX通道选择，参考 <a href="#">表3-94. 枚举dmamux_multiplexer_channel_enum</a>
输入参数{in}	
polarity	同步输入有效边沿
DMAMUX_SYNC_NO_EVENT	不检测边沿
DMAMUX_SYNC_RISING	上升沿
DMAMUX_SYNC_FALLING	下降沿
DMAMUX_SYNC_RISING_FALLING	上升和下降沿
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* configure synchronization input polarity */
dmamux_synchronization_polarity_config(DMAMUX_MUXCH0, DMAMUX_SYNC_RISING);
```

**函数 dmamux\_request\_forward\_number\_config**

函数 dmamux\_request\_forward\_number\_config 描述见下表:

**表 3-134. 函数 dmamux\_request\_forward\_number\_config**

函数名称	dmamux_request_forward_number_config
函数原型	void dmamux_request_forward_number_config(dmamux_mux_channel_enum channelx, uint32_t number);
功能描述	配置DMAMUX通道x要传输多少个DMA请求
先决条件	无
被调用函数	无
输入参数{in}	
channelx	指定要初始化的DMAMUX请求路由通道
DMAMUX_MUXCHx(x=0..2)	DMAMUX通道选择, 参考 <a href="#">表3-94. 枚举dmamux_mux_channel_enum</a>
输入参数{in}	
number	要传输的DMA请求数量 (1 - 32)
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* configure number of DMA requests to forward */
dmamux_request_forward_number_config(DMAMUX_MUXCH0, 4);
```

**函数 dmamux\_sync\_id\_config**

函数 dmamux\_sync\_id\_config 描述见下表:

**表 3-135. 函数 dmamux\_sync\_id\_config**

函数名称	dmamux_sync_id_config
函数原型	void dmamux_sync_id_config(dmamux_mux_channel_enum channelx, uint32_t id);
功能描述	配置DMAMUX同步输入标识
先决条件	无
被调用函数	无
输入参数{in}	
channelx	指定要初始化的DMAMUX请求路由通道
DMAMUX_MUXCHx(x=0..2)	DMAMUX通道选择, 参考 <a href="#">表3-94. 枚举dmamux_mux_channel_enum</a>
输入参数{in}	
id	同步输入标识

DMAMUX_SYNC_E XTI0	synchronization input is EXTI0
DMAMUX_SYNC_E XTI1	synchronization input is EXTI1
DMAMUX_SYNC_E XTI2	synchronization input is EXTI2
DMAMUX_SYNC_E XTI3	synchronization input is EXTI3
DMAMUX_SYNC_E XTI4	synchronization input is EXTI4
DMAMUX_SYNC_E XTI5	synchronization input is EXTI5
DMAMUX_SYNC_E XTI6	synchronization input is EXTI6
DMAMUX_SYNC_E XTI7	synchronization input is EXTI7
DMAMUX_SYNC_E XTI8	synchronization input is EXTI8
DMAMUX_SYNC_E XTI9	synchronization input is EXTI9
DMAMUX_SYNC_E XTI10	synchronization input is EXTI10
DMAMUX_SYNC_E XTI11	synchronization input is EXTI11
DMAMUX_SYNC_E XTI12	synchronization input is EXTI12
DMAMUX_SYNC_E XTI13	synchronization input is EXTI13
DMAMUX_SYNC_E XTI14	synchronization input is EXTI14
DMAMUX_SYNC_E XTI15	synchronization input is EXTI15
DMAMUX_SYNC_E VT0_OUT	synchronization input is Evt0_out
DMAMUX_SYNC_E VT1_OUT	synchronization input is Evt1_out
DMAMUX_SYNC_E VT2_OUT	synchronization input is Evt2_out
DMAMUX_SYNC_T IMER13_O	synchronization input is DMAMUX_SYNC_TIMER13_O
输出参数{out}	
-	-
返回值	

例如：

```
/* configure synchronization input identification */
dmamux_sync_id_config(DMAMUX_MUXCH0, DMAMUX_SYNC_EXTI0);
```

### 函数 dmamux\_request\_id\_config

函数 dmamux\_request\_id\_config 描述见下表：

表 3-136. 函数 dmamux\_request\_id\_config

函数名称	dmamux_request_id_config
函数原型	void dmamux_request_id_config(dmamux_multiplexer_channel_enum channelx, uint32_t id);
功能描述	配置DMAMUX请求路由通道输入标识
先决条件	无
被调用函数	无
输入参数{in}	
channelx	指定要初始化的DMAMUX请求路由通道
DMAMUX_MUXCHx(x=0..2)	DMAMUX通道选择，参考 <a href="#">表3-94. 枚举dmamux_multiplexer_channel_enum</a>
输入参数{in}	
id	DMA请求输入标识
DMA_REQUEST_MEMORY2MEMORY	memory to memory transfer
DMA_REQUEST_GENERATOR0	DMAMUX request generator 0
DMA_REQUEST_GENERATOR1	DMAMUX request generator 1
DMA_REQUEST_GENERATOR2	DMAMUX request generator 2
DMA_REQUEST_GENERATOR3	DMAMUX request generator 3
DMA_REQUEST_ADC	DMAMUX ADC request
DMA_REQUEST_DAC	DMAMUX DAC request
DMA_REQUEST_I2C0_RX	DMAMUX I2C0 RX request
DMA_REQUEST_I2C0_TX	DMAMUX I2C0 TX request
DMA_REQUEST_I2C1_RX	DMAMUX I2C1 RX request
DMA_REQUEST_I2C1_TX	DMAMUX I2C1 TX request

C1_TX	
DMA_REQUEST_S PI0_RX	DMAMUX SPI0 RX request
DMA_REQUEST_S PI0_TX	DMAMUX SPI0 TX request
DMA_REQUEST_S PI1_RX	DMAMUX SPI1 RX request
DMA_REQUEST_S PI1_TX	DMAMUX SPI1 TX request
DMA_REQUEST_TIMER0_CH	DMAMUX TIMER0 CH0 request
DMA_REQUEST_TIMER0_CH1	DMAMUX TIMER0 CH1 request
DMA_REQUEST_TIMER0_CH2	DMAMUX TIMER0 CH2 request
DMA_REQUEST_TIMER0_CH3	DMAMUX TIMER0 CH3 request
DMA_REQUEST_TIMER0_TRIG	DMAMUX TIMER0 TRIG request
DMA_REQUEST_TIMER0_UP	DMAMUX TIMER0 UP request
DMA_REQUEST_TIMER0_COM	DMAMUX TIMER0 COM request
DMA_REQUEST_TIMER2_CH0	DMAMUX TIMER2 CH0 request
DMA_REQUEST_TIMER2_CH1	DMAMUX TIMER2 CH1 request
DMA_REQUEST_TIMER2_CH2	DMAMUX TIMER2 CH2 request
DMA_REQUEST_TIMER2_CH3	DMAMUX TIMER2 CH3 request
DMA_REQUEST_TIMER2_TRIG	DMAMUX TIMER2 TRIG request
DMA_REQUEST_TIMER2_UP	DMAMUX TIMER2 UP request
DMA_REQUEST_TIMER15_CH0	DMAMUX TIMER15 CH0 request
DMA_REQUEST_TIMER15_UP	DMAMUX TIMER15 UP request
DMA_REQUEST_TIMER16_CH0	DMAMUX TIMER16 CH0 request
DMA_REQUEST_TIMER16_UP	DMAMUX TIMER16 UP request

<code>DMA_REQUEST_USART0_RX</code>	DMAMUX USART0 RX request
<code>DMA_REQUEST_USART0_TX</code>	DMAMUX USART0 TX request
<code>DMA_REQUEST_USART1_RX</code>	DMAMUX USART1 RX request
<code>DMA_REQUEST_USART1_TX</code>	DMAMUX USART1 TX request
<code>DMA_REQUEST_UART2_RX</code>	DMAMUX UART2 RX request
<code>DMA_REQUEST_UART2_TX</code>	DMAMUX UART2 TX request
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* configure multiplexer input identification */
dmamux_request_id_config(DMAMUX_MUXCH0, DMA_REQUEST_GENERATOR0);
```

### 函数 `dmamux_trigger_polarity_config`

函数 `dmamux_trigger_polarity_config` 描述见下表:

表 3-137. 函数 `dma_interrupt_disable`

函数名称	<code>dmamux_trigger_polarity_config</code>
函数原型	<code>void dmamux_trigger_polarity_config(dmamux_generator_channel_enum channelx, uint32_t polarity);</code>
功能描述	配置DMAMUX触发输入的有效边沿
先决条件	无
被调用函数	无
输入参数{in}	
<b>channelx</b>	指定要初始化的DMAMUX请求生成通道
<code>DMAMUX_GENCHx</code> ( $x=0..3$ )	DMAMUX请求生成通道选择, 参考 <a href="#">表3-95. 枚举 <code>dmamux_generator_channel_enum</code></a>
输入参数{in}	
<b>polarity</b>	触发输入信号有效边沿
<code>DMAMUX_GEN_NO_EVENT</code>	不检测边沿
<code>DMAMUX_GEN_RISING</code>	上升沿
<code>DMAMUX_GEN_FALLING</code>	下降沿

DMAMUX_GEN_RISING_FALLING	上升和下降沿
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* configure trigger input polarity */
dmamux_trigger_polarity_config(DMAMUX_GENCH0, DMAMUX_GEN_RISING);
```

### 函数 dmamux\_request\_generate\_number\_config

函数 dmamux\_request\_generate\_number\_config 描述见下表：

表 3-138. 函数 dmamux\_request\_generate\_number\_config

函数名称	dmamux_request_generate_number_config
函数原型	void dmamux_request_generate_number_config(dmamux_generator_channel_enum channelx, uint32_t number);
功能描述	配置DMAMUX请求生成器生成请求的数量
先决条件	无
被调用函数	无
输入参数{in}	
channelx	指定要初始化的DMAMUX请求生成通道
DMAMUX_GENCHx (x=0..3)	DMAMUX请求生成通道选择，参考 <a href="#">表3-95. 枚举 dmamux_generator_channel_enum</a>
输入参数{in}	
number	要生成的DMA请求数量（1 - 32）
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* configure number of DMA requests to be generated */
dmamux_request_generate_number_config(DMAMUX_GENCH0, 1);
```

### 函数 dmamux\_trigger\_id\_config

函数 dmamux\_trigger\_id\_config 描述见下表：

表 3-139. 函数 dmamux\_trigger\_id\_config

函数名称	dmamux_trigger_id_config
------	--------------------------

函数原型	void dmamux_trigger_id_config(dmamux_generator_channel_enum channelx, uint32_t id);
功能描述	配置DMAMUX触发输入标识
先决条件	无
被调用函数	无
输入参数{in}	
channelx	指定要初始化的DMAMUX请求生成通道
DMAMUX_GENCHx (x=0..3)	DMAMUX请求生成通道选择, 参考 <a href="#">表3-95. 枚举 dmamux_generator_channel_enum</a>
输入参数{in}	
id	触发输入标识
DMAMUX_SYNC_EXTI0	synchronization input is EXTI0
DMAMUX_SYNC_EXTI1	synchronization input is EXTI1
DMAMUX_SYNC_EXTI2	synchronization input is EXTI2
DMAMUX_SYNC_EXTI3	synchronization input is EXTI3
DMAMUX_SYNC_EXTI4	synchronization input is EXTI4
DMAMUX_SYNC_EXTI5	synchronization input is EXTI5
DMAMUX_SYNC_EXTI6	synchronization input is EXTI6
DMAMUX_SYNC_EXTI7	synchronization input is EXTI7
DMAMUX_SYNC_EXTI8	synchronization input is EXTI8
DMAMUX_SYNC_EXTI9	synchronization input is EXTI9
DMAMUX_SYNC_EXTI10	synchronization input is EXTI10
DMAMUX_SYNC_EXTI11	synchronization input is EXTI11
DMAMUX_SYNC_EXTI12	synchronization input is EXTI12
DMAMUX_SYNC_EXTI13	synchronization input is EXTI13
DMAMUX_SYNC_EXTI14	synchronization input is EXTI14
DMAMUX_SYNC_EXTI15	synchronization input is EXTI15



DMAMUX_SYNC_EVT0_OUT	synchronization input is Evt0_out
DMAMUX_SYNC_EVT1_OUT	synchronization input is Evt1_out
DMAMUX_SYNC_EVT2_OUT	synchronization input is Evt2_out
DMAMUX_SYNC_TIMER13_O	synchronization input is DMAMUX_SYNC_TIMER13_O
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* configure trigger input identification */
dmamux_trigger_id_config(DMAMUX_GENCH0, DMAMUX_TRIGGER_EXTI13);
```

### 函数 dmamux\_flag\_get

函数 dmamux\_flag\_get 描述见下表：

表 3-140. 函数 dmamux\_flag\_get

函数名称	dmamux_flag_get
函数原型	FlagStatus dmamux_flag_get(dmamux_flag_enum flag);
功能描述	获取DMAMUX通道x标志位状态
先决条件	无
被调用函数	无
输入参数{in}	
flag	标志类型，参考 <a href="#">表3-91. 枚举dmamux_flag_enum</a>
DMAMUX_FLAG_MUXCH0_SO	DMAMUX请求路由通道0同步溢出标志
DMAMUX_FLAG_MUXCH1_SO	DMAMUX请求路由通道1同步溢出标志
DMAMUX_FLAG_MUXCH2_SO	DMAMUX请求路由通道2同步溢出标志
DMAMUX_FLAG_GENCH0_TO	DMAMUX请求生成通道0触发溢出标志
DMAMUX_FLAG_GENCH1_TO	DMAMUX请求生成通道1触发溢出标志
DMAMUX_FLAG_GENCH2_TO	DMAMUX请求生成通道2触发溢出标志
DMAMUX_FLAG_GENCH3_TO	DMAMUX请求生成通道3触发溢出标志
输出参数{out}	

-	-
返回值	
FlagStatus	SET或RESET

例如:

```
FlagStatus flag = RESET;
/* get DMAMUX flag */
flag = dmamux_flag_get(DMAMUX_FLAG_GENCH0_TO);
```

### 函数 dmamux\_flag\_clear

函数 dmamux\_flag\_clear 描述见下表:

表 3-141. 函数 dmamux\_flag\_clear

函数名称	dmamux_flag_clear
函数原型	void dmamux_flag_clear(dmamux_flag_enum flag);
功能描述	清除DMAMUX通道x标志位状态
先决条件	无
被调用函数	无
输入参数{in}	
flag	标志类型, 参考 <a href="#">表3-91. 枚举dmamux_flag_enum</a>
DMAMUX_FLAG_MUXCH0_SO	DMAMUX请求路由通道0同步溢出标志
DMAMUX_FLAG_MUXCH1_SO	DMAMUX请求路由通道1同步溢出标志
DMAMUX_FLAG_MUXCH2_SO	DMAMUX请求路由通道2同步溢出标志
DMAMUX_FLAG_GENCH0_TO	DMAMUX请求生成通道0触发溢出标志
DMAMUX_FLAG_GENCH1_TO	DMAMUX请求生成通道1触发溢出标志
DMAMUX_FLAG_GENCH2_TO	DMAMUX请求生成通道2触发溢出标志
DMAMUX_FLAG_GENCH3_TO	DMAMUX请求生成通道3触发溢出标志
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* clear DMAMUX flag */
dmamux_flag_clear(DMAMUX_FLAG_GENCH0_TO);
```

**函数 dmamux\_interrupt\_enable**

函数 dmamux\_interrupt\_enable 描述见下表：

**表 3-142. 函数 dmamux\_interrupt\_enable**

函数名称	dmamux_interrupt_enable
函数原型	void dmamux_interrupt_enable(dmamux_interrupt_enum interrupt);
功能描述	使能DMAMUX通道x中断
先决条件	无
被调用函数	无
输入参数{in}	
interrupt	中断类型，参考 <a href="#">表3-90. 枚举dmamux_interrupt_enum</a>
DMAMUX_INT_MUXCH0_SO	DMAMUX请求路由通道0同步溢出中断
DMAMUX_INT_MUXCH1_SO	DMAMUX请求路由通道1同步溢出中断
DMAMUX_INT_MUXCH2_SO	DMAMUX请求路由通道2同步溢出中断
DMAMUX_INT_GE_NCH0_TO	DMAMUX请求生成通道0触发溢出中断
DMAMUX_INT_GE_NCH1_TO	DMAMUX请求生成通道1触发溢出中断
DMAMUX_INT_GE_NCH2_TO	DMAMUX请求生成通道2触发溢出中断
DMAMUX_INT_GE_NCH3_TO	DMAMUX请求生成通道3触发溢出中断
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* enable DMAMUX interrupt */
dmamux_interrupt_enable(DMAMUX_INT_MUXCH0_SO);
```

**函数 dmamux\_interrupt\_disable**

函数 dmamux\_interrupt\_disable 描述见下表：

**表 3-143. 函数 dmamux\_interrupt\_disable**

函数名称	dmamux_interrupt_disable
函数原型	void dmamux_interrupt_disable(dmamux_interrupt_enum interrupt);
功能描述	禁能DMAMUX通道x中断
先决条件	无

被调用函数	无
输入参数{in}	
interrupt	中断类型，参考 <a href="#">表3-90. 枚举dmamux_interrupt_enum</a>
DMAMUX_INT_MUXCH0_SO	DMAMUX请求路由通道0同步溢出中断
DMAMUX_INT_MUXCH1_SO	DMAMUX请求路由通道1同步溢出中断
DMAMUX_INT_MUXCH2_SO	DMAMUX请求路由通道2同步溢出中断
DMAMUX_INT_GENERCH0_TO	DMAMUX请求生成通道0触发溢出中断
DMAMUX_INT_GENERCH1_TO	DMAMUX请求生成通道1触发溢出中断
DMAMUX_INT_GENERCH2_TO	DMAMUX请求生成通道2触发溢出中断
DMAMUX_INT_GENERCH3_TO	DMAMUX请求生成通道3触发溢出中断
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* disable DMAMUX interrupt */
dmamux_interrupt_disable(DMAMUX_INT_MUXCH0_SO);
```

### 函数 dmamux\_interrupt\_flag\_get

函数 dmamux\_interrupt\_flag\_get 描述见下表：

表 3-144. 函数 dmamux\_interrupt\_flag\_get

函数名称	dmamux_interrupt_flag_get
函数原型	FlagStatus dmamux_interrupt_flag_get(dmamux_interrupt_flag_enum int_flag);
功能描述	获取DMAMUX通道x中断标志位状态
先决条件	无
被调用函数	无
输入参数{in}	
int_flag	标志类型，参考 <a href="#">表3-92. 枚举dmamux_interrupt_flag_enum</a>
DMAMUX_INT_FLAG_MUXCH0_SO	DMAMUX请求路由通道0同步溢出中断标志
DMAMUX_INT_FLAG_MUXCH1_SO	DMAMUX请求路由通道1同步溢出中断标志
DMAMUX_INT_FLAG_MUXCH2_SO	DMAMUX请求路由通道2同步溢出中断标志

G_MUXCH2_SO	
DMAMUX_INT_FLAG_G_GENCH0_TO	DMAMUX请求生成通道0触发溢出中断标志
DMAMUX_INT_FLAG_G_GENCH1_TO	DMAMUX请求生成通道1触发溢出中断标志
DMAMUX_INT_FLAG_G_GENCH2_TO	DMAMUX请求生成通道2触发溢出中断标志
DMAMUX_INT_FLAG_G_GENCH3_TO	DMAMUX请求生成通道3触发溢出中断标志
输出参数{out}	
-	-
返回值	
FlagStatus	SET或RESET

例如：

```
/* check DMAMUX interrupt flag */
if(dmamux_interrupt_flag_get(DMAMUX_INT_FLAG_GENCH0_TO)) {
    dmamux_interrupt_flag_clear(DMAMUX_INT_FLAG_GENCH0_TO);
}
```

### 函数 dmamux\_interrupt\_flag\_clear

函数 dmamux\_interrupt\_flag\_clear 描述见下表：

表 3-145. 函数 dmamux\_interrupt\_flag\_clear

函数名称	dmamux_interrupt_flag_clear
函数原型	FlagStatus dmamux_interrupt_flag_get(dmamux_interrupt_flag_enum int_flag);
功能描述	清除DMAMUX通道x中断标志位状态
先决条件	无
被调用函数	无
输入参数{in}	
int_flag	标志类型，参考 <a href="#">表3-92. 枚举dmamux_interrupt_flag_enum</a>
DMAMUX_INT_FLAG_MUXCH0_SO	DMAMUX请求路由通道0同步溢出中断标志
DMAMUX_INT_FLAG_MUXCH1_SO	DMAMUX请求路由通道1同步溢出中断标志
DMAMUX_INT_FLAG_MUXCH2_SO	DMAMUX请求路由通道2同步溢出中断标志
DMAMUX_INT_FLAG_GENCH0_TO	DMAMUX请求生成通道0触发溢出中断标志
DMAMUX_INT_FLAG_GENCH1_TO	DMAMUX请求生成通道1触发溢出中断标志
DMAMUX_INT_FLAG_GENCH2_TO	DMAMUX请求生成通道2触发溢出中断标志

DMAMUX_INT_FLAG_GENCH3_TO	DMAMUX请求生成通道3触发溢出中断标志
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* check DMAMUX interrupt flag */
if(dmamux_interrupt_flag_get(DMAMUX_INT_FLAG_GENCH0_TO)) {
    dmamux_interrupt_flag_clear(DMAMUX_INT_FLAG_GENCH0_TO);
}
```

## 3.7. EXTI

EXTI是MCU中的中断/事件控制器,包括24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[3.7.1](#)描述了EXTI的寄存器列表,章节[3.7.2](#)对EXTI库函数进行说明。

### 3.7.1. 外设寄存器说明

EXTI寄存器列表如下表所示:

表 3-146. 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

### 3.7.2. 外设库函数说明

EXTI库函数列表如下表所示:

表 3-147. 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

库函数名称	库函数描述
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 3-148. 枚举类型 exti\_line\_enum

枚举名称	枚举描述
EXTI_0	EXTI线0
EXTI_1	EXTI线1
EXTI_2	EXTI线2
EXTI_3	EXTI线3
EXTI_4	EXTI线4
EXTI_5	EXTI线5
EXTI_6	EXTI线6
EXTI_7	EXTI线7
EXTI_8	EXTI线8
EXTI_9	EXTI线9
EXTI_10	EXTI线10
EXTI_11	EXTI线11
EXTI_12	EXTI线12
EXTI_13	EXTI线13
EXTI_14	EXTI线14
EXTI_15	EXTI线15
EXTI_16	EXTI线16
EXTI_17	EXTI线17
EXTI_18	EXTI线18
EXTI_19	EXTI线19
EXTI_20	EXTI线20
EXTI_21	EXTI线21
EXTI_22	EXTI线22
EXTI_23	EXTI线23

### 枚举类型 exti\_mode\_enum

表 3-149. 枚举类型 exti\_mode\_enum

枚举名称	枚举描述
EXTI_INTERRUPT	EXTI中断模式

枚举名称	枚举描述
EXTI_EVENT	EXTI事件模式

### 枚举类型 `exti_trig_type_enum`

表 3-150. 枚举类型 `exti_trig_type_enum`

枚举名称	枚举描述
EXTI_TRIG_RISING	EXTI上升沿触发
EXTI_TRIG_FALLING	EXTI下降沿触发
EXTI_TRIG_BOTH	EXTI双边沿触发
EXTI_TRIG_NONE	EXTI双边沿均不触发

### 函数 `exti_deinit`

函数`exti_deinit`描述见下表：

表 3-151. 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 3-152. 函数 `exti_init`

函数名称	<code>exti_init</code>
函数原形	<code>void exti_init(exti_line_enum linex, exti_mode_enum mode, exti_trig_type_enum trig_type);</code>
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3-148. 枚举类型exti_line_enum</a>
输入参数{in}	
mode	EXTI模式, 参考 <a href="#">表3-149. 枚举类型exti_mode_enum</a>
输入参数{in}	
trig_type	触发类型, 参考 <a href="#">表3-150. 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 3-153. 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3-148. 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 3-154. 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3-148. 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 3-155. 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3-148. 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 3-156. 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3-148. 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 3-157. 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	使能EXTI线x软件事件中断
先决条件	-
被调用函数	-
输入参数{in}	
linex	EXTI线x, 参考 <a href="#">表3-148. 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 3-158. 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	禁用EXTI线x软件事件中断
先决条件	-

被调用函数	-
输入参数{in}	
linex	EXTI线x, 参考 <a href="#">表3-148. 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 3-159. 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3-148. 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 3-160. 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3-148. 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 3-161. 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3-148. 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 3-162. 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3-148. 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);
```

## 3.8. 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[3.8.1](#)描述了FMC的寄存器列表，章节[3.8.2](#)对FMC库函数进行说明。

### 3.8.1. 外设寄存器说明

FMC寄存器列表如下表所示:

表 3-163. FMC 寄存器

寄存器名称	寄存器描述
FMC_WS	等待状态寄存器
FMC_KEY	解锁寄存器
FMC_OBKEY	选项字节操作解锁寄存器
FMC_STAT	状态寄存器
FMC_CTL	控制寄存器
FMC_OBCTL	选项字节控制寄存器
FMC_DCRP_SADDR0	DCRP0起始地址寄存器
FMC_DCRP_EADDR0	DCRP0结束地址寄存器
FMC_WP0	擦除/编程保护区域0寄存器
FMC_WP1	擦除/编程保护区域1寄存器
FMC_DCRP_SADDR1	DCRP1起始地址寄存器
FMC_DCRP_EADDR1	DCRP1结束地址寄存器
FMC_SCR	安全用户区域寄存器

### 3.8.2. 外设库函数说明

FMC库函数列表如下表所示:

表 3-164. FMC 库函数

库函数名称	库函数描述
fmc_unlock	解锁FMC_CTL寄存器

库函数名称	库函数描述
fmc_lock	锁定FMC_CTL寄存器
fmc_main_flash_empty_check_stat_get	获取主闪存是否为空检查状态
fmc_main_flash_empty_check_stat_modify	修改主闪存空检查状态
fmc_wscnt_set	设置FMC等待状态
fmc_prefetch_enable	使能预取功能
fmc_prefetch_disable	失能预取功能
fmc_icache_enable	使能指令缓存区
fmc_icache_disable	失能指令缓存区
fmc_icache_reset	复位指令缓存区
fmc_page_erase	页擦除
fmc_mass_erase	整片擦除
fmc_doubleword_program	双字编程
fmc_fast_program	快速编程
fmc_debugger_enable	使能调试
fmc_debugger_disable	禁用调试
fmc_scr_area_enable	安全用户区域使能
ob_unlock	解锁FMC_OBCTL寄存器
ob_lock	锁定FMC_OBCTL寄存器
ob_reload	选项字节重加载
ob_user_write	用户选项字节编程
ob_security_protection_level_config	配置安全保护等级
ob_dcrp_area_config	配置DCRP区域
ob_write_protection_area_config	配置写保护区域
ob_scr_area_config	配置安全用户区域
ob_boot_lock_config	配置选项字节boot锁定位
ob_user_get	获取用户选项字节值
ob_security_protection_level_get	获取安全保护等级
ob_dcrp_area_get	获取DCRP区域
ob_write_protection_area_get	获取写保护区域
ob_scr_area_get	获取安全用户区域
ob_boot_lock_get	读取boot锁定配置
fmc_flag_get	获取FMC标志位
fmc_flag_clear	清除FMC标志位
fmc_interrupt_enable	使能FMC中断
fmc_interrupt_disable	失能FMC中断
fmc_interrupt_flag_get	获取FMC中断标志位
fmc_interrupt_flag_clear	清除FMC中断标志位
fmc_state_get	获取FMC状态
fmc_ready_wait	检测FMC等待状态

## 枚举 `fmc_state_enum`

表 3-165. 枚举 `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_OBERR	选项字节读错误
FMC_RPERR	读保护错误
FMC_FSTPRR	快速编程错误
FMC_PGSERR	编程顺序错误
FMC_PGMERR	编程大小错误
FMC_PGAERR	编程对齐错误
FMC_WPERR	擦除/编程保护错误
FMC_PGERR	编程错误标志
FMC_OPRERR	操作失败错误
FMC_TOERR	超时错误
FMC_OB_HSPC	高保护等级标志
FMC_UNDEFINEDERR	未定义初始错误标志

## 枚举 `ob_user_data_extract_info_enum`

表 3-166. 枚举 `ob_user_data_extract_info_enum`

枚举名称	枚举描述
OBCTL_USER_DATA_BORST_EN	BORST_EN位掩码及起始位
OBCTL_USER_DATA_BORR_TH	BORR_TH位掩码及起始位
OBCTL_USER_DATA_BORF_TH	BORF_TH位掩码及起始位
OBCTL_USER_DATA_NRST_DPSLP	nRST_DPSLP位掩码及起始位
OBCTL_USER_DATA_NRST_STDBY	nRST_STDBY位掩码及起始位
OBCTL_USER_DATA_NFWDG_HW	nFWDG_HW位掩码及起始位
OBCTL_USER_DATA_NWWDG_HW	nWWDG位掩码及起始位
OBCTL_USER_DATA_HXTAL_REMAP	HXTAL_REMAP位掩码及起始位
OBCTL_USER_DATA_SRAM_ECC_EN	SRAM_ECC_EN位掩码及起始位
OBCTL_USER_DATA_SWBT0	SWBT0位掩码及起始位
OBCTL_USER_DATA_NBOOT1	nBOOT1位掩码及起始位
OBCTL_USER_DATA_NBOOT0	nBOOT0位掩码及起始位
OBCTL_USER_DATA_NRST_MDSEL	NRST_MDSEL位掩码及起始位

## 函数 `fmc_unlock`

函数 `fmc_unlock` 描述见下表：

表 3-167. 函数 `fmc_unlock`

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



功能描述	解锁FMC_CTL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 FMC_CTL register */
```

```
fmc_unlock();
```

### 函数 fmc\_lock

函数fmc\_lock描述见下表：

表 3-168. 函数 fmc\_lock

函数名称	fmc_lock
函数原型	void fmc_lock(void);
功能描述	锁定FMC_CTL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：

```
/* lock FMC_CTL register */
```

```
fmc_lock();
```

### 函数 fmc\_main\_flash\_empty\_check\_stat\_get

函数fmc\_main\_flash\_empty\_check\_stat\_get描述见下表

表 3-169. 函数 fmc\_wscnt\_set

函数名称	fmc_main_flash_empty_check_stat_get
函数原型	FlagStatus fmc_main_flash_empty_check_stat_get(void);
功能描述	获取主闪存是否为空检查状态

先决条件	-
被调用函数	-
输入参数{in}	
-	-
输出参数{out}	
-	-
返回值	
FlagStatus	主闪存是否为空状态, SET: 空; RESET: 非空

例如:

```
/* get the main flash empty check status */
```

```
FlagStatus status = fmc_main_flash_empty_check_stat_get();
```

### 函数 fmc\_main\_flash\_empty\_check\_stat\_modify

函数fmc\_main\_flash\_empty\_check\_stat\_modify描述见下表

表 3-170. 函数 fmc\_main\_flash\_empty\_check\_stat\_modify

函数名称	fmc_main_flash_empty_check_stat_modify
函数原型	void fmc_main_flash_empty_check_stat_modify(uint32_t empty_check_status)
功能描述	修改主闪存空检查状态
先决条件	-
被调用函数	-
输入参数{in}	
empty_check_status	要设置的主闪存状态
FMC_WS_MFPE_PROGRAMMED	设置主闪存为已编程, 非空
FMC_WS_MFPE_EMPTY	设置主闪存为空
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* set the main flash empty check status to be FMC_WS_MFPE_EMPTY */
```

```
fmc_main_flash_empty_check_stat_modify(FMC_WS_MFPE_EMPTY);
```

### 函数 fmc\_wsctl\_set

函数fmc\_wsctl\_set描述见下表

表 3-171. 函数 `fmc_wscnt_set`

函数名称	<code>fmc_wscnt_set</code>
函数原型	<code>void fmc_wscnt_set(uint32_t wscnt);</code>
功能描述	设置等待状态时间
先决条件	-
被调用函数	-
输入参数{in}	
<b>wscnt</b>	等待状态时间
<code>FMC_WAIT_STATE_0</code>	增加0个等待状态
<code>FMC_WAIT_STATE_1</code>	增加1个等待状态
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* set the wait state 0 */
```

```
fmc_wscnt_set(FMC_WAIT_STATE_0);
```

### 函数 `fmc_prefetch_enable`

函数 `fmc_prefetch_enable` 描述见下表：

表 3-172. 函数 `fmc_prefetch_enable`

函数名称	<code>fmc_prefetch_enable</code>
函数原型	<code>void fmc_prefetch_enable(void);</code>
功能描述	使能预取
先决条件	-
被调用函数	-
输入参数{in}	
-	-
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* enable pre-fetch */
```

```
fmc_prefetch_enable( );
```

### 函数 `fmc_prefetch_disable`

函数 `fmc_prefetch_disable` 描述见下表：

表 3-173. 函数 fmc\_prefetch\_disable

函数名称	fmc_prefetch_disable
函数原型	void fmc_prefetch_disable (void);
功能描述	失能预取
先决条件	-
被调用函数	-
输入参数{in}	
-	-
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* disable pre-fetch */
fmc_prefetch_disable( );
```

### 函数 fmc\_icache\_enable

函数fmc\_icache\_enable描述见下表:

表 3-174. 函数 fmc\_icache\_enable

函数名称	fmc_icache_enable
函数原型	void fmc_icache_enable(void);
功能描述	使能指令cache
先决条件	-
被调用函数	-
输入参数{in}	
-	-
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* enable icache */
fmc_icache_enable( );
```

### 函数 fmc\_icache\_disable

函数fmc\_icache\_disable描述见下表:

表 3-175. 函数 fmc\_icache\_disable

函数名称	fmc_icache_disable
函数原型	void fmc_icache_disable (void);
功能描述	失能指令cache
先决条件	-
被调用函数	-
输入参数{in}	
-	-
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* disable icache */
fmc_icache_disable( );
```

### 函数 fmc\_icache\_reset

函数fmc\_icache\_reset\_enable描述见下表:

表 3-176. 函数 fmc\_icache\_reset\_enable

函数名称	fmc_icache_reset
函数原型	void fmc_icache_reset (void);
功能描述	复位指令cache
先决条件	-
被调用函数	-
输入参数{in}	
-	-
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* icache reset */
fmc_icache_reset( );
```

### 函数 fmc\_page\_erase

函数fmc\_page\_erase描述见下表

表 3-177. 函数 fmc\_page\_erase

函数名称	fmc_page_erase
函数原型	fmc_page_erase(uint32_t page_number)
功能描述	FMC页擦除
先决条件	fmc_unlock
被调用函数	-
输入参数{in}	
page_number	擦除的页码
输出参数{out}	
-	-
返回值	
fmc_state_enum	FMC状态，参考 <a href="#">表3-165. 枚举fmc_state_enum</a>

例如：

```
fmc_unlock();
```

```
/* erase page */
```

```
fmc_state_enum state = fmc_page_erase(0);
```

### 函数 fmc\_mass\_erase

函数fmc\_mass\_erase描述见下表

表 3-178. 函数 fmc\_mass\_erase

函数名称	fmc_mass_erase
函数原型	fmc_state_enum fmc_mass_erase(void);
功能描述	FMC整片擦除
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，参考 <a href="#">表3-165. 枚举fmc_state_enum</a>

例如：

```
fmc_unlock();
```

```
/* erase whole chip */
```

```
fmc_state_enum state = fmc_mass_erase( );
```

函数 **fmc\_doubleword\_program**

函数fmc\_doubleword\_program描述见下表

表 3-179. 函数 **fmc\_doubleword\_program**

函数名称	fmc_doubleword_program
函数原型	fmc_state_enum fmc_doubleword_program(uint32_t address, uint64_t data);
功能描述	在相应地址双字编程
先决条件	fmc_unlock
被调用函数	-
输入参数{in}	
address	编程地址
输入参数{in}	
data	待编程数据
输出参数{out}	
-	-
返回值	
fmc_state_enum	FMC状态, 参考 <a href="#">表3-165. 枚举fmc_state_enum</a>

例如:

```
fmc_unlock();
```

```
fmc_page_erase(0x08004000);
```

```
/* program a double word at the corresponding address in main flash */
```

```
fmc_state_enum fmc_state = fmc_doubleword_program(0x08004000, 0x11223344aabbccdd);
```

函数 **fmc\_fast\_program**

函数fmc\_fast\_program描述见下表

表 3-180. 函数 **fmc\_doubleword\_program**

函数名称	fmc_fast_program
函数原型	fmc_state_enum fmc_fast_program(uint32_t address, uint32_t data_buf)
功能描述	在相应地址快速编程
先决条件	fmc_unlock
被调用函数	-
输入参数{in}	
address	编程地址
输入参数{in}	
data_buf	待编程数据地址
输出参数{out}	
-	-

返回值	
<b>fmc_state_enum</b>	FMC状态，参考 <a href="#">表3-165</a> 。枚举 <b>fmc_state_enum</b>

例如：

```
/* data buffer for fast programming */
```

```
static uint64_t data_buffer[32] = {
```

```
    0x0000000000000000U,    0x1111111111111111U,    0x2222222222222222U,
    0x3333333333333333U,
```

```
    0x4444444444444444U,    0x5555555555555555U,    0x6666666666666666U,
    0x7777777777777777U,
```

```
    0x8888888888888888U,    0x9999999999999999U,    0xAAAAAAAAAAAAAAAAAU,
    0BBBBBBBBBBBBBBBBBU,
```

```
    0CCCCCCCCCCCCCCCCCU,    0DDDDDDDDDDDDDDDDDU,
    0EEEEEEEEEEEEEEEEEU, 0FFFFFFFFFFFFFFFFFU,
```

```
    0x0011001100110011U,    0x2233223322332233U,    0x4455445544554455U,
    0x6677667766776677U,
```

```
    0x8899889988998899U, 0xAABBAABBAABBAABBU, 0xCCDDCCDDCCDDCCDDU,
    0xEEFFEEFFEEFFEEFFU,
```

```
    0x2200220022002200U,    0x3311331133113311U,    0x6644664466446644U,
    0x7755775577557755U,
```

```
    0xAA88AA88AA88AA88U, 0xBB99BB99BB99BB99U, 0xEECCEECCEECCECCU,
    0xFFDDFFDDFFDDFFDDU
```

```
};
```

```
fmc_unlock();
```

```
fmc_page_erase(0x08004000);
```

```
/* program flash */
```

```
fmc_state_enum fmc_state = fmc_fast_program(0x08004000, data_buffer);
```

### 函数 fmc\_debugger\_enable

函数fmc\_debugger\_enable描述见下表

**表 3-181. 函数 fmc\_debugger\_enable**

函数名称	fmc_debugger_enable
函数原型	void fmc_debugger_enable(void);
功能描述	使能debug
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 debugger */
```

```
fmc_debugger_enable( );
```

### 函数 fmc\_debugger\_disable

函数fmc\_debugger\_disable描述见下表

**表 3-182. 函数 fmc\_debugger\_disable**

函数名称	fmc_debugger_disable
函数原型	void fmc_debugger_disable(void);
功能描述	失能debug
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 debugger */
```

```
fmc_debugger_disable( );
```

### 函数 fmc\_scr\_area\_enable

函数fmc\_scr\_area\_enable描述见下表

**表 3-183. 函数 fmc\_scr\_area\_enable**

函数名称	fmc_scr_area_enable
函数原型	void fmc_scr_area_enable();
功能描述	使能SCR区域
先决条件	fmc_unlock
被调用函数	-

输入参数{in}	
-	-
输出参数{out}	
-	-
返回值	
-	-

例如:

```
fmc_unlock();

/* enable secure area protection */

fmc_scr_area_enable( );
```

### 函数 ob\_unlock

函数ob\_unlock描述见下表

表 3-184. 函数 ob\_unlock

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

例如:

```
fmc_unlock();

/* unlock the option bytes operation */

ob_unlock();
```

### 函数 ob\_lock

函数ob\_lock描述见下表

表 3-185. 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:

```
fmc_unlock();
```

```
/*lock the option bytes operation */
```

```
ob_lock();
```

### 函数 ob\_reload

函数ob\_reload描述见下表

表 3-186. 函数 ob\_reload

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

例如:

```
fmc_unlock();
```

```
ob_unlock();
```

```
/* reload the option bytes operation */
```

```
ob_reload( );
```

### 函数 ob\_user\_write

函数ob\_user\_write描述见下表

表 3-187. 函数 `ob_user_write`

函数名称	<code>ob_user_write</code>
函数原型	<code>fmc_state_enum ob_user_write(uint32_t ob_user, uint32_t ob_user_mask);</code>
功能描述	修改用户选项字节值
先决条件	<code>ob_unlock</code>
被调用函数	<code>fmc_ready_wait</code>
输入参数{in}	
<code>ob_user</code>	用户选项字节
<code>OB_NRST_PIN_INPUT_MODE</code>	复位引脚输入复位信号模式
<code>OB_NRST_PIN_NORMAL_GPIO</code>	复位引脚仅为普通GPIO口
<code>OB_NRST_PIN_INPUT_OUTPUT_MODE</code>	复位引脚既输入复位信号也输出复位脉冲模式
<code>OB_NBOOT0_VALUE_0</code>	Boot0为1
<code>OB_NBOOT0_VALUE_1</code>	Boot0为0
<code>OB_NBOOT1_VALUE_0</code>	BOOT1为1
<code>OB_NBOOT1_VALUE_1</code>	BOOT1为0
<code>OB_SWBOOT0_FROM_OB_BOOT0</code>	由选项字节决定Boot0
<code>OB_SWBOOT0_FROM_PIN</code>	由PB8/Boot0决定复位
<code>OB_SRAM_ECC_ENABLE</code>	使能SRAM ECC
<code>OB_SRAM_ECC_DISABLE</code>	失能SRAM ECC
<code>OB_HXTAL_REMAP_ENABLE</code>	使能HXTAL重映射
<code>OB_HXTAL_REMAP_DISABLE</code>	失能HXTAL重映射
<code>OB_WWDGT_HW</code>	硬件窗口看门狗
<code>OB_WWDGT_SW</code>	软件窗口看门狗
<code>OB_FWDGT_HW</code>	硬件独立看门狗
<code>OB_FWDGT_SW</code>	软件独立看门狗
<code>OB_STDBY_RST</code>	当系统进入待机模式时复位
<code>OB_STDBY_NRST</code>	当系统进入待机模式时不复位
<code>OB_DEEPSLEEP_RST</code>	当系统进入深度睡眠模式时复位
<code>OB_DEEPSLEEP_NRS</code>	当系统进入深度睡眠模式时不复位

<i>T</i>	
OB_BORR_TH_VALUE 0	BOR上升等级1，阈值约2.1 V
OB_BORR_TH_VALUE 1	BOR上升等级2，阈值约2.3 V
OB_BORR_TH_VALUE 2	BOR上升等级3，阈值约2.6 V
OB_BORR_TH_VALUE 3	BOR上升等级4，阈值约2.9 V
OB_BORF_TH_VALUE 0	BOR下降等级1，阈值约2.0 V
OB_BORF_TH_VALUE 1	BOR下降等级2，阈值约2.2 V
OB_BORF_TH_VALUE 2	BOR下降等级3，阈值约2.5 V
OB_BORF_TH_VALUE 3	BOR下降等级4，阈值约2.8 V
OB_BORST_DISABLE	电压波动复位失能，上电复位由POR/PDR级别定义
OB_BORST_ENABLE	电压波动复位使能，阈值参考BORR_TH和BORF_TH
输入参数{in}	
ob_user_mask	用户选项字节掩码
FMC_OBCTL_NRST_ MDSEL	复位引脚模式掩码
FMC_OBCTL_NBOOT 0	选项字节NBOOT0掩码
FMC_OBCTL_NBOOT 1	选项字节NBOOT1掩码
FMC_OBCTL_SWBT0	软件BOOT0掩码
FMC_OBCTL_SRAM_ ECC_EN	SRAM_ECC禁能掩码
FMC_OBCTL_HXTAL_ REMAP	HXTAL重映射禁能掩码
FMC_OBCTL_NWWDG _HW	窗口看门狗配置位掩码
FMC_OBCTL_NFWDG _HW	软件独立看门狗配置位掩码
FMC_OBCTL_NRST_S TDBY	系统待机模式时复位掩码
FMC_OBCTL_NRST_D PSLP	系统深度睡眠模式复位掩码
FMC_OBCTL_BORF_T H	VDD供电下降时的BOR阈值掩码
FMC_OBCTL_BORR_T	VDD供电上升时的BOR阈值掩码

<i>H</i>	
<i>FMC_OBCTL_BORST_EN</i>	电压波动复位使能位掩码
输出参数{out}	
-	-
返回值	
<b>fmc_state_enum</b>	FMC状态，参考 <a href="#">表3-165. 枚举fmc_state_enum</a>

例如：

```
fmc_state_enum fmc_state;
```

```
fmc_unlock();
```

```
ob_unlock();
```

```
/* modify option byte */
```

```
fmc_state = ob_user_write (OB_NRST_PIN_INPUT_MODE, FMC_OBCTL_NRST_MDSEL);
```

### 函数 ob\_security\_protection\_level\_config

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

表 3-188. 函数 ob\_security\_protection\_level\_config

函数名称	ob_security_protection_level_config
函数原型	fmc_state_enum ob_security_protection_level_config(uint8_t ob_spc)
功能描述	配置安全保护等级
先决条件	ob_unlock
被调用函数	fmc_ready_wait
输入参数{in}	
<b>ob_spc</b>	安全保护等级
<i>FMC_NSPC</i>	无安全保护
<i>FMC_LSPC</i>	低安全保护
<i>FMC_HSPC</i>	高安全保护
输出参数{out}	
-	-
返回值	
<b>fmc_state_enum</b>	FMC状态，参考 <a href="#">表3-165. 枚举fmc_state_enum</a>

例如：

```
fmc_state_enum fmc_state;
```

```
fmc_unlock();
```

```
ob_unlock();
```

```
/* disable security protection */
```

```
fmc_state = ob_security_protection_config(FMC_NSPPC);
```

### 函数 ob\_dcrp\_area\_config

函数ob\_dcrp\_area\_config描述见下表

表 3-189. 函数 ob\_dcrp\_area\_config

函数名称	ob_dcrp_area_config
函数原型	fmc_state_enum ob_dcrp_area_config(uint32_t dcrp_area, uint32_t dcrp_eren, uint32_t dcrp_start, uint32_t dcrp_end);
功能描述	配置DCRP区域
先决条件	ob_unlock
被调用函数	fmc_ready_wait
输入参数{in}	
dcrp_area	DCRP区域
DCRP_AREA_0	第一个DCRP区域
DCRP_AREA_1	第二个DCRP区域
输入参数{in}	
dcrp_eren	DCRP区域擦除选项
OB_DCRP_AREA_ERASE_DISABLE	当安全保护等级从低降为无保护时DCRP区域擦除失能
OB_DCRP_AREA_ERASE_ENABLE	当安全保护等级从低降为无保护时DCRP区域擦除使能
输入参数{in}	
dcrp_start	DCRP区域第一页
输入参数{in}	
dcrp_end	DCRP区域最后一页
输出参数{out}	
-	-
返回值	
fmc_state_enum	FMC状态，参考 <a href="#">表3-165. 枚举fmc_state_enum</a>

例如：

```
fmc_state_enum fmc_state;
```

```
fmc_unlock();
```

```
ob_unlock();
```

```
/* configure DCRP area */
```

```
fmc_state = fmc_state_enum ob_dcrp_area_config(DCRP_AREA_0, OB_DCRP_AREA_ERASE_ENABLE, 0x05, 0x0B);
```

**函数 ob\_write\_protection\_area\_config**

函数ob\_write\_protection\_area\_config描述见下表

**表 3-190. 函数 ob\_write\_protection\_config**

函数名称	ob_write_protection_area_config
函数原型	fmc_state_enum ob_write_protection_area_config(uint32_t wp_area, uint32_t wp_start, uint32_t wp_end);
功能描述	配置安全访问区域
先决条件	ob_unlock
被调用函数	fmc_ready_wait
输入参数{in}	
wp_area	写保护区域
WP_AREA_0	第一个写保护区域
WP_AREA_1	第二个写保护区域
输入参数{in}	
wp_start	写保护区域的第一页
输入参数{in}	
wp_end	写保护区域的最后一页
输出参数{out}	
-	-
返回值	
fmc_state_enum	FMC状态, 参考 <a href="#">表3-165. 枚举fmc_state_enum</a>

例如:

```
fmc_state_enum fmc_state;
```

```
fmc_unlock();
```

```
ob_unlock();
```

```
/* configure write protection area */
```

```
fmc_state = fmc_state_enum ob_write_area_protection_area_config(WP_AREA0, 0x08, 0x09);
```

**函数 ob\_scr\_area\_config**

函数ob\_scr\_area\_config描述见下表

**表 3-191. 函数 ob\_scr\_area\_config**

函数名称	ob_scr_area_config
函数原型	fmc_state_enum ob_scr_area_config(uint32_t secure_size);
功能描述	配置安全用户区域
先决条件	ob_unlock
被调用函数	fmc_ready_wait



输入参数{in}	
<b>secure_size</b>	以页为单位的安全用户区域大小
输出参数{out}	
-	-
返回值	
<b>fmc_state_enum</b>	FMC状态，参考 <a href="#">表3-165. 枚举fmc_state_enum</a>

例如：

```
fmc_unlock();
```

```
ob_unlock();
```

```
/* configure the option byte secure area */
```

```
fmc_state_enum fmc_state = ob_scr_area_config(0x02);
```

### 函数 ob\_boot\_lock\_config

函数ob\_boot\_lock\_config描述见下表

**表 3-192. 函数 ob\_boot\_lock\_config**

<b>函数名称</b>	ob_boot_lock_config
<b>函数原型</b>	fmc_state_enum ob_boot_lock_config(uint32_t boot_config);
<b>功能描述</b>	配置boot锁定
<b>先决条件</b>	ob_unlock
<b>被调用函数</b>	fmc_ready_wait
输入参数{in}	
<b>boot_config</b>	Boot配置位
OB_BOOT_LOCK_FR OM_MAIN_FLASH	强制从主闪存启动
OB_BOOT_UNLOCK	解除boot锁定
输出参数{out}	
-	-
返回值	
<b>fmc_state_enum</b>	FMC状态，参考 <a href="#">表3-165. 枚举fmc_state_enum</a>

例如：

```
fmc_unlock();
```

```
ob_unlock();
```

```
/* unlock boot */
```

```
fmc_state_enum fmc_state = ob_boot_lock_config(OB_BOOT_UNLOCK);
```

**函数 ob\_user\_get**

函数ob\_user\_get描述见下表

**表 3-193. 函数 ob\_user\_get**

函数名称	ob_user_get
函数原型	uint32_t ob_user_get(ob_user_data_extract_info_enum user_data_extract_info, uint8_t * ob_user_data);
功能描述	获取用户选项字节值
先决条件	-
被调用函数	-
输入参数{in}	
ob_user_data_extract_info_enum	用户选项字节值解析输入信息，参考 <a href="#">表3-166. 枚举 ob_user_data_extract_info_enum</a>
输出参数{out}	
ob_user_data	用户选项字节值
返回值	
uint32_t	返回值是否有效
INVLD_RETURN_VALUE	返回值无效
VLD_RETURN_VALUE	返回值有效

例如：

```
/* get value of option bytes USER */
uint32_t user_value;

ob_user_data_extract_info_enum user_data_extract_info;

uint8_t ob_user_data;

user_value = ob_user_get(OBCTL_USER_DATA_BORST_EN, ob_user_data);
```

**函数 ob\_security\_protection\_level\_get**

函数ob\_security\_protection\_level\_get描述见下表

**表 3-194. 函数 ob\_security\_protection\_level\_get**

函数名称	ob_security_protection_level_get
函数原型	uint8_t ob_security_protection_level_get(void);
功能描述	获取安全保护等级
先决条件	-
被调用函数	-
输入参数{in}	
-	-
输出参数{out}	

-	-
返回值	
uint8_t	安全保护等级
FMC_NSPC	无安全保护
FMC_LSPC	低安全保护
FMC_HSPC	高安全保护

例如：

```
/* get the value of FMC option bytes security protection level */
```

```
uint8_t user = ob_security_protection_level_get();
```

### 函数 ob\_dcrp\_area\_get

函数ob\_dcrp\_area\_get描述见下表

表 3-195. 函数 ob\_dcrp\_area\_get

函数名称	ob_dcrp_area_get
函数原型	uint32_t ob_dcrp_area_get(uint32_t dcrp_area, uint32_t *dcrp_erase_option, uint32_t *dcrp_start, uint32_t *dcrp_end);
功能描述	获取DCRP相关配置
先决条件	-
被调用函数	-
输入参数{in}	
dcrp_area	DCRP区域
DCRP_AREA0	第一个DCRP区域
DCRP_AREA1	第二个DCRP区域
输出参数{out}	
dcrp_erase_option	DCRP擦除选项
输出参数{out}	
dcrp_start	DCRP区域第一页
输出参数{out}	
dcrp_end	DCRP区域最后一页
返回值	
uint32_t	返回值是否有效
INVLD_RETURN_VAL UE	返回值无效
VLD_RETURN_VALUE	返回值有效

例如：

```
/* get configuration of DCRP area */
```

```
uint32_t start, end, erase ,flag;
```

```
flag = ob_dcrp_area_get(DCRP_AREA0, &erase, &start, &end)
```

函数 **ob\_write\_protection\_area\_get**

函数ob\_write\_protection\_area\_get描述见下表

表 3-196. 函数 **ob\_write\_protection\_area\_get**

函数名称	ob_write_protection_area_get
函数原型	uint32_t ob_write_protection_area_get(uint32_t wp_area, uint32_t *wp_start, uint32_t *wp_end);
功能描述	获取写保护地址
先决条件	-
被调用函数	-
输入参数{in}	
wp_area	写保护区域
WP_AREA0	第一个写保护区域
WP_AREA1	第二个写保护区域
输出参数{out}	
wp_start	写保护区域第一页
输出参数{out}	
wp_end	写保护区域最后一页
返回值	
uint32_t	返回值是否有效
INVLD_RETURN_VALUE	返回值无效
VLD_RETURN_VALUE	返回值有效

例如：

```
/* get configuration of WP area */
```

```
uint32_t start, end, flag;
```

```
flag = ob_write_protection_area_get (WP_AREA0, &start, &end);
```

函数 **ob\_scr\_area\_get**

函数ob\_scr\_area\_get描述见下表

表 3-197. 函数 **ob\_scr\_area\_size\_get**

函数名称	ob_scr_area_get
函数原型	uint32_t ob_scr_area_get(uint32_t *scr_area_byte_cnt);
功能描述	获取安全用户区域大小
先决条件	-
被调用函数	-
输入参数{in}	
-	-
输出参数{out}	

<b>scr_area_byte_cnt</b>	以字节为单位的安全用户区域大小
<b>返回值</b>	
<b>uint32_t</b>	返回值是否有效
<b>INVLD_RETURN_VAL UE</b>	返回值无效
<b>VLD_RETURN_VALUE</b>	返回值有效

例如：

```
/* get size of secure area */
```

```
uint32_t size, flag;
```

```
flag = ob_scr_area_get(&size);
```

### 函数 ob\_boot\_lock\_get

函数ob\_boot\_lock\_get描述见下表

**表 3-198. 函数 ob\_boot\_lock\_get**

<b>函数名称</b>	ob_boot_lock_get
<b>函数原型</b>	uint32_t ob_boot_lock_get (void);
<b>功能描述</b>	获取boot锁定配置
<b>先决条件</b>	-
<b>被调用函数</b>	-
<b>输入参数{in}</b>	
-	-
<b>输出参数{out}</b>	
-	-
<b>返回值</b>	
<b>uint32_t</b>	Boot配置

例如：

```
/* get boot value */
```

```
uint32_t boot_value;
```

```
boot_value = ob_boot_lock_get();
```

### 函数 fmc\_flag\_get

函数fmc\_flag\_get描述见下表

**表 3-199. 函数 fmc\_flag\_get**

<b>函数名称</b>	fmc_flag_get
<b>函数原型</b>	FlagStatus fmc_flag_get(uint32_t flag);
<b>功能描述</b>	获取标志位状态
<b>先决条件</b>	-

被调用函数	-
输入参数{in}	
<b>flag</b>	FMC标志位
<i>FMC_FLAG_BUSY</i>	操作进行中标志
<i>FMC_FLAG_OBERR</i>	选项字节读错误标志
<i>FMC_FLAG_WPERR</i>	擦除/编程保护错误标志
<i>FMC_FLAG_PGSERR</i>	编程顺序错误标志
<i>FMC_FLAG_PGMERR</i>	编程大小错误标志
<i>FMC_FLAG_PGAERR</i>	编程对齐错误标志
<i>FMC_FLAG_FSTPERR</i>	快速编程错误标志
<i>FMC_FLAG_RPERR</i>	读保护错误标志
<i>FMC_FLAG_PGERR</i>	编程错误标志
<i>FMC_FLAG_OPRERR</i>	操作失败错误标志
<i>FMC_FLAG_ENDF</i>	操作结束标志
输出参数{out}	
-	-
返回值	
<b>FlagStatus</b>	标志状态
<i>RESET</i>	复位
<i>SET</i>	置位

例如:

```
/* get FMC flag status */
```

```
FlagStatus flag = fmc_flag_get(FMC_FLAG_ENDF);
```

### 函数 fmc\_flag\_clear

函数fmc\_flag\_clear描述见下表

表 3-200. 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}	
<b>flag</b>	FMC标志位
<i>FMC_FLAG_OBERR</i>	选项字节读错误标志
<i>FMC_FLAG_WPERR</i>	擦除/编程保护错误标志
<i>FMC_FLAG_PGSERR</i>	编程顺序错误标志
<i>FMC_FLAG_PGMERR</i>	编程大小错误标志
<i>FMC_FLAG_PGAERR</i>	编程对齐错误标志
<i>FMC_FLAG_FSTPERR</i>	快速编程错误标志

<i>FMC_FLAG_RPERR</i>	读保护错误标志
<i>FMC_FLAG_PGERR</i>	编程错误标志
<i>FMC_FLAG_OPRERR</i>	操作失败错误标志
<i>FMC_FLAG_ENDF</i>	操作结束标志
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* clear FMC ENDF flag */
```

```
fmc_flag_clear(FMC_FLAG_ENDF);
```

### 函数 fmc\_interrupt\_enable

函数fmc\_interrupt\_enable描述见下表

表 3-201. 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	-
被调用函数	-
输入参数{in}	
interrupt	FMC中断使能选项
<i>FMC_INT_END</i>	操作结束中断使能
<i>FMC_INT_ERR</i>	操作失败中断使能
<i>FMC_INT_RPERR</i>	读保护错误中断使能
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* enable FMC end interrupt */
```

```
fmc_interrupt_enable(FMC_INT_END);
```

### 函数 fmc\_interrupt\_disable

函数fmc\_interrupt\_disable描述见下表

表 3-202. 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	-
被调用函数	-
输入参数{in}	
<b>interrupt</b>	FMC中断失能选项
<i>FMC_INT_END</i>	操作结束中断失能
<i>FMC_INT_ERR</i>	操作失败中断失能
<i>FMC_INT_RPERR</i>	读保护错误中断失能
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* disable FMC end interrupt */
```

```
fmc_interrupt_disable(FMC_INT_END);
```

### 函数 fmc\_interrupt\_flag\_get

函数fmc\_interrupt\_flag\_get描述见下表

表 3-203. 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}	
<b>interrupt</b>	FMC中断标志
<i>FMC_INT_END</i>	操作结束中断标志
<i>FMC_INT_OPRERR</i>	操作失败中断标志
<i>FMC_INT_RPERR</i>	读保护错误中断标志
输出参数{out}	
-	-
返回值	
<b>FlagStatus</b>	标志状态
<i>RESET</i>	复位
<i>SET</i>	置位

例如：

```
/* get FMC RPERR error flag */
```



```
FlagStatus flag = fmc_interrupt_flag_get(FMC_INT_FLAG_RPERR);
```

### 函数 fmc\_interrupt\_flag\_clear

函数fmc\_interrupt\_flag\_clear描述见下表

表 3-204. 函数 fmc\_interrupt\_flag\_clear

函数名称	fmc_interrupt_flag_clear
函数原型	void fmc_interrupt_flag_clear(uint32_t flag);
功能描述	清除FMC中断标志
先决条件	-
被调用函数	-
输入参数{in}	
interrupt	FMC中断标志
FMC_INT_END	操作结束中断标志
FMC_INT_OPRERR	操作失败中断标志
FMC_INT_RPERR	读保护错误中断标志
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* clear FMC RPERR error flag */
```

```
fmc_interrupt_flag_clear(FMC_INT_FLAG_RPERR);
```

### 函数 fmc\_state\_get

函数fmc\_state\_get描述见下表:

表 3-205. 函数 fmc\_state\_get

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

-	-
返回值	
fmc_state_enum	FMC状态值，详情参考枚举变量 <a href="#">表3-165. 枚举fmc_state_enum</a>

例如：

```
/* get the FMC state */
fmc_state_enum state = fmc_state_get( );
```

### 函数 fmc\_ready\_wait

函数 fmc\_ready\_wait描述见下表：

表 3-206. 函数 fmc\_ready\_wait

函数名称	fmc_ready_wait
函数原型	fmc_state_enum fmc_ready_wait(uint32_t timeout);
功能描述	检查FMC是否准备好
先决条件	-
被调用函数	fmc_state_get();
输入参数{in}	
timeout	循环计数次数
输出参数{out}	
-	-
返回值	
fmc_state_enum	FMC状态值，详情参考枚举变量 <a href="#">表3-165. 枚举fmc_state_enum</a>

例如：

```
/* check whether FMC is ready or not */
fmc_state_enum state = fmc_ready_wait (0x00001000 );
```

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

### 3.9.1. 外设寄存器说明

FWDGT寄存器列表如下表所示：

表 3-207. FWDGT 寄存器

寄存器名称	寄存器描述
FWDGT_CTL	控制寄存器
FWDGT_PSC	预分频寄存器
FWDGT_RLD	重装载寄存器
FWDGT_STAT	状态寄存器
FWDGT_WND	窗口寄存器

### 3.9.2. 外设库函数说明

FWDGT库函数列表如下表所示：

表 3-208. FWDGT 库函数

库函数名称	库函数描述
fdwgt_write_enable	使能对寄存器FWDGT_PSC, FWDGT_RLD和FWDGT_WND的写操作
fdwgt_write_disable	失能对寄存器FWDGT_PSC, FWDGT_RLD和FWDGT_WND的写操作
fdwgt_enable	使能FWDGT
fdwgt_prescaler_value_config	配置独立看门狗定时器时钟预分频数
fdwgt_reload_value_config	配置独立看门狗定时器计数器重装载值
fdwgt_window_value_config	配置独立看门狗定时器计数窗口值
fdwgt_counter_reload	按照FWDGT_RLD寄存器的值重装载FWDGT计数器
fdwgt_config	设置FWDGT重装载值、预分频值
fdwgt_flag_get	获取FWDGT标志位状态

#### 函数 fdwgt\_write\_enable

函数fdwgt\_write\_enable描述见下表：

表 3-209. 函数 fdwgt\_write\_enable

函数名称	fdwgt_write_enable
函数原型	void fdwgt_write_enable(void);
功能描述	使能对寄存器FWDGT_PSC, FWDGT_RLD和FWDGT_WND的写操作
先决条件	-
被调用函数	-
输入参数{in}	
-	-
输出参数{out}	
-	-

返回值	
-	-

例如：

```
/* enable write access to FWDGT_PSC and FWDGT_RLD and FWDGT_WND */
```

```
fwdgt_write_enable();
```

### 函数 fwdgt\_write\_disable

函数fwdgt\_write\_disable描述见下表：

表 3-210. 函数 fwdgt\_write\_disable

函数名称	fwdgt_write_disable
函数原型	void fwdgt_write_disable(void);
功能描述	除能对寄存器FWDGT_PSC, FWDGT_RLD和FWDGT_WND的写操作
先决条件	-
被调用函数	-
输入参数{in}	
-	-
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* disable write access to FWDGT_PSC,FWDGT_RLD and FWDGT_WND */
```

```
fwdgt_write_disable();
```

### 函数 fwdgt\_enable

函数fwdgt\_enable描述见下表：

表 3-211. 函数 fwdgt\_enable

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

-	-
---	---

例如:

```
/* start the FWDGT counter */
```

```
fwdgt_enable();
```

### 函数 fwdgt\_prescaler\_value\_config

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

表 3-212. 函数 fwdgt\_prescaler\_value\_config

函数名称	fwdgt_prescaler_value_config
函数原型	ErrStatus fwdgt_prescaler_value_config(uint16_t prescaler_value);
功能描述	配置独立看门狗定时器时钟预分频数
先决条件	-
被调用函数	-
输入参数{in}	
prescaler_value	预分频值
FWDGT_PSC_DIVx	FWDGT预分频值设为x (x=4,8,16,32,64,128,256)
输出参数{out}	
-	-
返回值	
ErrStatus	ERROR / SUCCESS

例如:

```
/* set FWDGT prescaler to 4 */
```

```
ErrStatus flag;
```

```
flag = fwdgt_prescaler_value_config(FWDGT_PSC_DIV4);
```

### 函数 fwdgt\_reload\_value\_config

函数fwdgt\_reload\_value\_config描述见下表:

表 3-213. 函数 fwdgt\_reload\_value\_config

函数名称	fwdgt_reload_value_config
函数原型	ErrStatus fwdgt_reload_value_config(uint16_t reload_value);
功能描述	配置独立看门狗定时器计数器重装载值
先决条件	-
被调用函数	-
输入参数{in}	
reload_value	重装载值, 数值范围为0x0000 - 0x0FFF
输出参数{out}	
-	-

返回值	
ErrStatus	ERROR / SUCCESS

例如:

```
/* set FWDGT reload value to 0xFFFF */
```

```
ErrStatus flag;
```

```
flag = fwdgt_reloadr_value_config(0xFFFF);
```

### 函数 fwdgt\_window\_value\_reload

函数fwdgt\_window\_value\_config描述见下表:

表 3-214. 函数 fwdgt\_window\_value\_config

函数名称	fwdgt_window_value_config
函数原型	ErrStatus fwdgt_window_value_config(uint16_t window_value);
功能描述	配置独立看门狗定时器计数器窗口值
先决条件	-
被调用函数	-
输入参数{in}	
window_value	窗口值,数值范围为0x0000 – 0xFFFF
输出参数{out}	
-	-
返回值	
ErrStatus	ERROR / SUCCESS

例如:

```
/* set FWDGT window value to 0xFFFF */
```

```
ErrStatus flag;
```

```
flag = fwdgt_window_value_config(0xFFFF);
```

### 函数 fwdgt\_counter\_reload

函数fwdgt\_counter\_reload描述见下表:

表 3-215. 函数 fwdgt\_counter\_reload

函数名称	fwdgt_counter_reload
函数原型	void fwdgt_counter_reload(void);
功能描述	按照FWDGT_RLD寄存器的值重载FWDG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 3-216. 函数 fwdgt\_config

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

例如:

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

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

**函数 fwdgt\_flag\_get**

函数fwdgt\_flag\_get描述见下表:

**表 3-217. 函数 fwdgt\_flag\_get**

函数名称	fwdgt_flag_get
函数原型	FlagStatus fwdgt_flag_get(uint16_t flag);
功能描述	获取FWDGT标志位状态
先决条件	-
被调用函数	-
输入参数{in}	
flag	需要获取状态的FWDGT标志位
FWDGT_FLAG_PUD	预分频值更新进行中
FWDGT_FLAG_RU D	重装载值更新进行中
FWDGT_FLAG_WU D	窗口值更新进行中
输出参数{out}	
-	-
返回值	
FlagStatus	SET / RESET

例如:

```
/* test if a prescaler value update is on going */
```

```
FlagStatus status;
```

```
status = fwdgt_flag_get(FWDGT_FLAG_PUD);
```

**3.10. GPIO**

GPIO用来实现各片上设备的逻辑输入/输出功能。章节[3.10.1](#)描述了GPIO的寄存器列表，章节[3.10.2](#)对GPIO库函数进行说明。

**3.10.1. 外设寄存器说明**

GPIO寄存器列表如下表所示:

**表 3-218. 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



寄存器名称	寄存器描述
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

### 3.10.2. 外设库函数说明

GPIO库函数列表如下表所示：

**表 3-219. 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 3-220. 函数 gpio\_deinit**

函数名称	gpio_deinit
函数原型	void gpio_deinit(uint32_t gpio_periph);
功能描述	复位外设GPIOx
先决条件	-
被调用函数	rcu_periph_reset_enable / rcu_periph_reset_disable
输入参数{in}	
gpio_periph	GPIO端口

<i>GPIOx</i>	端口选择(x = A,B,C,D,F)
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* reset GPIOA */
```

```
gpio_deinit(GPIOA);
```

### 函数 **gpio\_mode\_set**

函数gpio\_mode\_set描述见下表:

**表 3-221. 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}	
<b>gpio_periph</b>	GPIO端口
<i>GPIOx</i>	GPIOx(x = A,B,C,D,F)
输入参数{in}	
<b>mode</b>	GPIO引脚模式
<i>GPIO_MODE_INPUT</i>	输入模式
<i>GPIO_MODE_OUTPUT</i>	输出模式
<i>GPIO_MODE_AF</i>	备用功能模式
<i>GPIO_MODE_ANALOG</i>	模拟模式
输入参数{in}	
<b>pull_up_down</b>	GPIO引脚上拉下拉电阻设置
<i>GPIO_PUPD_NONE</i>	悬空模式, 无上拉和下拉
<i>GPIO_PUPD_PULLUP</i>	带上拉电阻
<i>GPIO_PUPD_PULLDOWN</i>	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:

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

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

### 函数 **gpio\_output\_options\_set**

函数gpio\_output\_options\_set描述见下表:

**表 3-222. 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,D,F)
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_LE</i> <i>VEL0</i>	AF最大输出速度等级0
<i>GPIO_OSPEED_LE</i> <i>VEL1</i>	AF最大输出速度等级1
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:

```
/* config 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 3-223. 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,D,F)
输入参数{in}	
pin	GPIO引脚
GPIO_PIN_x	引脚选择(x=0..15)
GPIO_PIN_ALL	所有引脚
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* 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 3-224. 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

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

例如:

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

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

### 函数 gpio\_bit\_write

函数gpio\_bit\_write描述见下表:

表 3-225. 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}	
<b>gpio_periph</b>	GPIO端口
<i>GPIOx</i>	端口选择(x = A,B,C,D,F)
输入参数{in}	
<b>pin</b>	GPIO引脚
<i>GPIO_PIN_x</i>	引脚选择(x=0..15)
<i>GPIO_PIN_ALL</i>	所有引脚
输入参数{in}	
<b>bit_value</b>	设置或清除
<i>RESET</i>	清除引脚值
<i>SET</i>	设置引脚值
输出参数{out}	
-	-
返回值	
-	-

例如:

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

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

### 函数 gpio\_port\_write

函数gpio\_port\_write描述见下表:

表 3-226. 函数 gpio\_port\_write

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

例如:

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

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

### 函数 gpio\_input\_bit\_get

函数gpio\_input\_bit\_get描述见下表:

表 3-227. 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,D,F)
输入参数{in}	
pin	GPIO引脚
GPIO_PIN_x	引脚选择(x=0..15)
GPIO_PIN_ALL	所有引脚
输出参数{out}	

-	-
返回值	
FlagStatus	SET / RESET

例如：

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

```
FlagStatus bit_state;
```

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

### 函数 gpio\_input\_port\_get

函数gpio\_input\_port\_get描述见下表：

表 3-228. 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}	
gpio_periph	GPIO端口
GPIOx	端口选择(x = A,B,C,D,F)
输出参数{out}	
-	-
返回值	
uint16_t	0x0000-0xFFFF

例如：

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

```
uint16_t port_state;
```

```
port_state = gpio_input_port_get(GPIOA);
```

### 函数 gpio\_output\_bit\_get

函数gpio\_output\_bit\_get描述见下表：

表 3-229. 函数 gpio\_output\_bit\_get

函数名称	gpio_output_bit_get
函数原型	FlagStatus gpio_output_bit_get(uint32_t gpio_periph, uint32_t pin);
功能描述	获取端口所有引脚的输出值
先决条件	-
被调用函数	-

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

例如:

```
/* get 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 3-230. 函数 gpio\_output\_port\_get

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

例如:

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

```
uint16_t port_state;
```

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



## 函数 gpio\_af\_set

函数gpio\_af\_set描述见下表:

表 3-231. 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,D,F)
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	RTC_OUT, CK_OUT0, SWDIO, SWCLK, SPI0, SPI1, USART0, EVENTOUT
GPIO_AF_1	USART0, USART1, USART2, TIMER0, TIMER2, SPI1
GPIO_AF_2	TIMER0, TIMER13, TIMER15, TIMER16
GPIO_AF_3	SPI1, TIMER2, CK_OUT1, USART2
GPIO_AF_4	USART0, USART1, SPI1, TIMER13
GPIO_AF_5	TIMER0, TIMER16, I2S, USART0, SPI1
GPIO_AF_6	I2C0, I2C1, USART1, USART2
GPIO_AF_7	CMP0, CMP1, EVENTOUT, I2C0
GPIO_AF_8	SPI0
GPIO_AF_9	TIMER0, USART1
GPIO_AF_10	SPI0, TIMER0, TIMER15, TIMER16
GPIO_AF_11	CK_OUT1, TIMER0, TIMER2
GPIO_AF_12	USART0, TIMER0, TIMER2
GPIO_AF_13	TIMER2, TIMER13
GPIO_AF_14	USART0, TIMER15, I2C0
GPIO_AF_15	CK_OUT1, TIMER16, 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 3-232. 函数 **gpio\_pin\_lock**

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

例如:

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

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

函数 **gpio\_bit\_toggle**

函数gpio\_bit\_toggle描述见下表:

表 3-233. 函数 **gpio\_bit\_toggle**

函数名称	gpio_bit_toggle
函数原型	void gpio_bit_toggle(uint32_t gpio_periph, uint32_t pin);
功能描述	翻转GPIO引脚状态
先决条件	-
被调用函数	-
输入参数{in}	
gpio_periph	GPIO端口
GPIOx	GPIOx(x = A,B,C,D,F)
输入参数{in}	
pin	GPIO引脚
GPIO_PIN_x	引脚选择(x=0..15)
GPIO_PIN_ALL	所有引脚
输出参数{out}	

-	-
返回值	
-	-

例如：

```
/* 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 3-234. 函数 gpio\_port\_toggle

函数名称	gpio_port_toggle
函数原型	void gpio_port_toggle(uint32_t gpio_periph);
功能描述	翻转一组GPIO状态
先决条件	-
被调用函数	-
输入参数{in}	
gpio_periph	GPIO端口
GPIOx	GPIOx(x = A,B,C,D,F)
输出参数{out}	
-	-
返回值	
-	-

例如：

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

```
gpio_port_toggle(GPIOA);
```

## 3.11. 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[3.11.1](#)描述了I2C的寄存器列表，章节[3.11.2](#)对I2C库函数进行说明。

### 3.11.1. 外设寄存器说明

I2C寄存器列表如下表所示：

表 3-235. I2C 寄存器

寄存器名称	寄存器描述
I2C_CTL0	控制寄存器 0

寄存器名称	寄存器描述
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

### 3.11.2. 外设库函数说明

I2C库函数列表如下表所示：

**表 3-236. I2C 库函数**

库函数名称	库函数描述
i2c_deinit	复位外设 I2C
i2c_timing_config	配置时序参数
i2c_digital_noise_filter_config	配置数字噪声过滤器
i2c_analog_noise_filter_enable	使能数字噪声过滤器
i2c_analog_noise_filter_disable	禁能数字噪声过滤器
i2c_master_clock_config	配置主机模式下 SCL 高低电平时钟周期
i2c_master_addressing	配置 I2C 从机地址以及数据传输方向
i2c_address10_header_enable	主机接收模式下，10 位地址头只执行读操作
i2c_address10_header_disable	主机接收模式下，10 位地址头执行完整的读操作序列
i2c_address10_enable	使能主机模式下 10 位地址寻址模式
i2c_address10_disable	禁能主机模式下 10 位地址寻址模式
i2c_automatic_end_enable	使能主机模式下 I2C 自动结束模式
i2c_automatic_end_disable	禁能主机模式下 I2C 自动结束模式
i2c_slave_response_to_gcall_enable	使能从机响应广播呼叫
i2c_slave_response_to_gcall_disable	禁能从机响应广播呼叫
i2c_stretch_scl_low_enable	当从机数据没有准备好时拉低 SCL
i2c_stretch_scl_low_disable	当从机数据没有准备好时不拉低 SCL
i2c_address_config	配置 I2C 从机地址
i2c_address_bit_compare_config	定义 ADDRESS[7:1]的哪些位和接收到的地址进行比较
i2c_address_disable	禁能从机模式下 I2C 地址
i2c_second_address_config	配置 I2C 从机第二个地址
i2c_second_address_disable	禁能 I2C 从机第二个地址
i2c_received_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

库函数名称	库函数描述
i2c_slave_byte_control_disable	禁能从机字节控制
i2c_nack_enable	从机模式下产生 NACK
i2c_nack_disable	从机模式下产生 ACK
i2c_wakeup_from_deepsleep_enable	使能从 Deep-sleep 模式中唤醒
i2c_wakeup_from_deepsleep_disable	禁止从 Deep-sleep 模式中唤醒
i2c_enable	使能 I2C
i2c_disable	禁能 I2C
i2c_start_on_bus	在 I2C 总线上生成起始位
i2c_stop_on_bus	在 I2C 总线上生成停止位
i2c_data_transmit	发送数据
i2c_data_receive	接收数据
i2c_reload_enable	使能 I2C 重载模式
i2c_reload_disable	禁能 I2C 重载模式
i2c_transfer_byte_number_config	配置待发送字节数
i2c_dma_enable	使能发送/接收模式下 DMA
i2c_dma_disable	禁能发送/接收模式下 DMA
i2c_pec_transfer	I2C 传输 PEC 值
i2c_pec_enable	使能报文错误校验
i2c_pec_disable	禁能报文错误校验
i2c_pec_value_get	获取报文错误校验值
i2c_smbus_alert_enable	使能 SMBus 报警
i2c_smbus_alert_disable	禁能 SMBus 报警
i2c_smbus_default_addr_enable	使能 SMBus 设备默认地址
i2c_smbus_default_addr_disable	禁能 SMBus 设备默认地址
i2c_smbus_host_addr_enable	使能 SMBus 主机地址
i2c_smbus_host_addr_disable	禁能 SMBus 主机地址
i2c_extented_clock_timeout_enable	使能时钟信号延展超时检测
i2c_extented_clock_timeout_disable	禁能时钟信号延展超时检测
i2c_clock_timeout_enable	使能时钟超时检测
i2c_clock_timeout_disable	禁能时钟超时检测
i2c_bus_timeout_b_config	配置总线超时 B
i2c_bus_timeout_a_config	配置总线超时 A
i2c_idle_clock_timeout_config	配置空闲时钟超时检测
i2c_flag_get	获取 I2C 标志位
i2c_flag_clear	清除 I2C 标志位
i2c_interrupt_enable	中断使能
i2c_interrupt_disable	中断禁能
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 3-237. 枚举类型 `i2c_interrupt_flag_enum`

枚举名称	枚举描述
<code>I2C_INT_FLAG_TI</code>	发送中断标志
<code>I2C_INT_FLAG_RBNE</code>	接收期间I2C_RDATA非空中断标志
<code>I2C_INT_FLAG_ADDSEND</code>	从机模式下，接收到的地址与自身地址匹配中断标志
<code>I2C_INT_FLAG_NACK</code>	NACK中断标志
<code>I2C_INT_FLAG_STPDET</code>	从机模式下检测到STOP信号中断标志
<code>I2C_INT_FLAG_TC</code>	主机模式下传输完成中断标志
<code>I2C_INT_FLAG_TCR</code>	传输完成重载中断标志
<code>I2C_INT_FLAG_BERR</code>	总线错误中断标志
<code>I2C_INT_FLAG_LOSTARB</code>	仲裁丢失中断标志
<code>I2C_INT_FLAG_OUERR</code>	从机模式下，过载/欠载错误中断标志
<code>I2C_INT_FLAG_PECERR</code>	PEC错误中断标志
<code>I2C_INT_FLAG_TIMEOUT</code>	超时中断标志
<code>I2C_INT_FLAG_SMBALT</code>	SMBus报警中断标志

函数 `i2c_deinit`

函数*i2c\_deinit*描述见下表：

表 3-238. 函数 `i2c_deinit`

函数名称	<code>i2c_deinit</code>
函数原型	<code>void i2c_deinit(uint32_t i2c_periph);</code>
功能描述	复位外设 I2C
先决条件	-
被调用函数	<code>rcu_periph_reset_enable</code> / <code>rcu_periph_reset_disable</code>
输入参数{in}	
<code>i2c_periph</code>	I2C 外设
<code>I2Cx</code>	(x=0,1)
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* reset I2C0 */
```

```
i2c_deinit(I2C0);
```

函数 `i2c_timing_config`

函数*i2c\_timing\_config*描述见下表：

表 3-239. 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 3-240. 函数 i2c\_digital\_noise\_filter\_config

函数名称	i2c_digital_noise_filter_config
函数原型	void i2c_digital_noise_filter_config(uint32_t i2c_periph, uint32_t filter_length);
功能描述	配置数字噪声过滤器
先决条件	-
被调用函数	-
输入参数{in}	
i2c_periph	I2C 外设
I2Cx	(x=0,1)
输入参数{in}	
filter_length	过滤长度
FILTER_DISABLE	数字噪声过滤器禁能
FILTER_LENGTH_1	数字噪声滤波使能并且可以滤除脉宽宽度不大于 1 t <sub>12CCLK</sub> 的尖峰

<i>FILTER_LENGTH_2</i>	数字噪声滤波使能并且可以滤除脉宽宽度不大于 2 $t_{I2CCLK}$ 的尖峰
<i>FILTER_LENGTH_3</i>	数字噪声滤波使能并且可以滤除脉宽宽度不大于 3 $t_{I2CCLK}$ 的尖峰
<i>FILTER_LENGTH_4</i>	数字噪声滤波使能并且可以滤除脉宽宽度不大于 4 $t_{I2CCLK}$ 的尖峰
<i>FILTER_LENGTH_5</i>	数字噪声滤波使能并且可以滤除脉宽宽度不大于 5 $t_{I2CCLK}$ 的尖峰
<i>FILTER_LENGTH_6</i>	数字噪声滤波使能并且可以滤除脉宽宽度不大于 6 $t_{I2CCLK}$ 的尖峰
<i>FILTER_LENGTH_7</i>	数字噪声滤波使能并且可以滤除脉宽宽度不大于 7 $t_{I2CCLK}$ 的尖峰
<i>FILTER_LENGTH_8</i>	数字噪声滤波使能并且可以滤除脉宽宽度不大于 8 $t_{I2CCLK}$ 的尖峰
<i>FILTER_LENGTH_9</i>	数字噪声滤波使能并且可以滤除脉宽宽度不大于 9 $t_{I2CCLK}$ 的尖峰
<i>FILTER_LENGTH_10</i>	数字噪声滤波使能并且可以滤除脉宽宽度不大于 10 $t_{I2CCLK}$ 的尖峰
<i>FILTER_LENGTH_11</i>	数字噪声滤波使能并且可以滤除脉宽宽度不大于 11 $t_{I2CCLK}$ 的尖峰
<i>FILTER_LENGTH_12</i>	数字噪声滤波使能并且可以滤除脉宽宽度不大于 12 $t_{I2CCLK}$ 的尖峰
<i>FILTER_LENGTH_13</i>	数字噪声滤波使能并且可以滤除脉宽宽度不大于 13 $t_{I2CCLK}$ 的尖峰
<i>FILTER_LENGTH_14</i>	数字噪声滤波使能并且可以滤除脉宽宽度不大于 14 $t_{I2CCLK}$ 的尖峰
<i>FILTER_LENGTH_15</i>	数字噪声滤波使能并且可以滤除脉宽宽度不大于 15 $t_{I2CCLK}$ 的尖峰
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* I2C0 digital filter filters spikes with a length of up to 1 tI2CCLK */
```

```
i2c_digital_noise_filter_config(I2C0, FILTER_LENGTH_1);
```

### 函数 i2c\_analog\_noise\_filter\_enable

函数 i2c\_analog\_noise\_filter\_enable 描述见下表：

表 3-241. 函数 i2c\_analog\_noise\_filter\_enable

函数名称	i2c_analog_noise_filter_enable
函数原型	void i2c_analog_noise_filter_enable(uint32_t i2c_periph);
功能描述	使能模拟噪声滤波器
先决条件	-
被调用函数	-
输入参数{in}	
<b>i2c_periph</b>	I2C 外设
<i>I2Cx</i>	(x=0,1)
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* enable analog noise filter */
```



```
i2c_analog_noise_filter_enable(I2C0);
```

### 函数 i2c\_analog\_noise\_filter\_disable

函数i2c\_analog\_noise\_filter\_disable描述见下表：

**表 3-242. 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 3-243. 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 3-244. 函数 i2c\_master\_addressing

函数名称	i2c_master_addressing
函数原型	void i2c_master_addressing(uint32_t i2c_periph, uint32_t address, uint32_t trans_direction);
功能描述	配置 I2C 从机地址以及数据传输方向
先决条件	-
被调用函数	-
输入参数{in}	
i2c_periph	I2C 外设
I2Cx	(x=0,1)
输入参数{in}	
address	除保留地址外的地址，0-0x3FF，由主机发送给从机的地址
输入参数{in}	
trans_direction	主机模式下，I2C 传输方向
I2C_MASTER_TRANSMIT	主机发送
I2C_MASTER_RECEIVE	主机接收
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* send slave address to I2C bus */
```

```
i2c_master_addressing(I2C0, 0x82, I2C_MASTER_TRANSMIT);
```

### 函数 i2c\_address10\_header\_enable

函数i2c\_address10\_header\_enable描述见下表：

表 3-245. 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 3-246. 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 3-247. 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 3-248. 函数 i2c\_address10\_disable

函数名称	i2c_address10_disable
函数原型	void i2c_address10_disable(uint32_t i2c_periph);
功能描述	禁能主机模式下 10 位地址寻址模式
先决条件	-
被调用函数	-
输入参数{in}	
i2c_periph	I2C 外设
I2Cx	(x=0,1)
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* disable 10-bit addressing mode in master mode */
```

```
i2c_address10_disable(I2C0);
```

### 函数 i2c\_automatic\_end\_enable

函数i2c\_automatic\_end\_enable描述见下表：

表 3-249. 函数 i2c\_automatic\_end\_enable

函数名称	i2c_automatic_end_enable
函数原型	void i2c_automatic_end_enable(uint32_t i2c_periph);
功能描述	使能主机模式下 I2C 自动结束模式
先决条件	-
被调用函数	-
输入参数{in}	
i2c_periph	I2C 外设
I2Cx	(x=0,1)
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* enable I2C automatic end mode in master mode */
```

```
i2c_automatic_end_enable(I2C0);
```

### 函数 i2c\_automatic\_end\_disable

函数i2c\_automatic\_end\_disable描述见下表：

表 3-250. 函数 i2c\_automatic\_end\_disable

函数名称	i2c_automatic_end_disable
函数原型	void i2c_automatic_end_disable(uint32_t i2c_periph);
功能描述	禁能主机模式下 I2C 自动结束模式
先决条件	-
被调用函数	-
输入参数{in}	
i2c_periph	I2C 外设
I2Cx	(x=0,1)
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* disable I2C automatic end mode in master mode */
```

```
i2c_automatic_end_disable(I2C0);
```

### 函数 i2c\_slave\_response\_to\_gcall\_enable

函数i2c\_slave\_response\_to\_gcall\_enable描述见下表：

表 3-251. 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 3-252. 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 3-253. 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 3-254. 函数 i2c\_stretch\_scl\_low\_disable

函数名称	i2c_stretch_scl_low_disable
函数原型	void i2c_stretch_scl_low_disable(uint32_t i2c_periph);
功能描述	当从机数据没有准备好时不拉低 SCL
先决条件	-
被调用函数	-
输入参数{in}	
i2c_periph	I2C 外设
I2Cx	(x=0,1)
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* disable to stretch SCL low when data is not ready in slave mode */
```

```
i2c_stretch_scl_low_disable(I2C0);
```

### 函数 i2c\_address\_config

函数i2c\_address\_config描述见下表:

表 3-255. 函数 i2c\_address\_config

函数名称	i2c_address_config
函数原型	void i2c_address_config(uint32_t i2c_periph, uint32_t address, uint32_t addr_format);
功能描述	配置 I2C 从机地址
先决条件	-
被调用函数	-
输入参数{in}	
i2c_periph	I2C 外设
I2Cx	(x=0,1)
输入参数{in}	
address	I2C 地址
输入参数{in}	
addr_format	7 位地址或 10 位地址
I2C_ADDFORMAT_7BITS	7 位地址
I2C_ADDFORMAT_10BITS	10 位地址
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* configure i2c slave address */
```

```
i2c_address_config(I2C0, 0x82, I2C_ADDFORMAT_7BITS);
```

### 函数 i2c\_address\_bit\_compare\_config

函数 i2c\_address\_bit\_compare\_config 描述见下表:

表 3-256. 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_COM PARE	地址的第 1 位需要进行比较
ADDRESS_BIT2_COM PARE	地址的第 2 位需要进行比较
ADDRESS_BIT3_COM PARE	地址的第 3 位需要进行比较
ADDRESS_BIT4_COM PARE	地址的第 4 位需要进行比较
ADDRESS_BIT5_COM PARE	地址的第 5 位需要进行比较
ADDRESS_BIT6_COM PARE	地址的第 6 位需要进行比较
ADDRESS_BIT7_COM PARE	地址的第 7 位需要进行比较
输出参数{out}	
-	-
返回值	
-	-

例如：

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

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

### 函数 i2c\_address\_disable

函数i2c\_address\_disable描述见下表：

表 3-257. 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 3-258. 函数 i2c\_second\_address\_config**

函数名称	i2c_second_address_config
函数原型	void i2c_second_address_config(uint32_t i2c_periph, uint32_t address, uint32_t addr_mask);
功能描述	配置 I2C 从机第二个地址
先决条件	-
被调用函数	-
输入参数{in}	
i2c_periph	I2C 外设
I2Cx	(x=0,1)
输入参数{in}	
address	I2C 地址
输入参数{in}	
addr_mask	不需要进行比较的地址
ADDRESS2_NO_MASK	无屏蔽，全部都需要进行比较
ADDRESS2_MASK_BIT1	ADDRESS2[1]屏蔽， ADDRESS2[7:2]进行比较
ADDRESS2_MASK_BIT1_2	ADDRESS2[2:1]屏蔽， ADDRESS2[7:3]进行比较
ADDRESS2_MASK_BIT1_3	ADDRESS2[3:1]屏蔽， ADDRESS2[7:4]进行比较
ADDRESS2_MASK_BIT1_4	ADDRESS2[4:1]屏蔽， ADDRESS2[7:5]进行比较
ADDRESS2_MASK_BIT1_5	ADDRESS2[5:1]屏蔽， ADDRESS2[7:6]进行比较
ADDRESS2_MASK_BIT1_6	ADDRESS2[6:1]屏蔽， ADDRESS2[7]进行比较
ADDRESS2_MASK_ALL	ADDRESS2[7:1]屏蔽
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* configure i2c second slave address */
```

```
i2c_second_address_config(I2C0, 0x82, ADDRESS2_MASK_BIT1_2);
```

### 函数 i2c\_second\_address\_disable

函数i2c\_second\_address\_disable描述见下表:

**表 3-259. 函数 i2c\_second\_address\_disable**

函数名称	i2c_second_address_disable
函数原型	void i2c_second_address_disable(uint32_t i2c_periph);
功能描述	禁能 I2C 从机第二个地址
先决条件	-
被调用函数	-
输入参数{in}	
i2c_periph	I2C 外设
I2Cx	(x=0,1)
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* disable i2c second address in slave mode */
```

```
i2c_second_address_disable(I2C0);
```

### 函数 i2c\_receved\_address\_get

函数i2c\_receved\_address\_get描述见下表:

**表 3-260. 函数 i2c\_receved\_address\_get**

函数名称	i2c_receved_address_get
函数原型	uint32_t i2c_receved_address_get(uint32_t i2c_periph);
功能描述	获取从机模式下匹配成功的地址
先决条件	-
被调用函数	-
输入参数{in}	
i2c_periph	I2C 外设
I2Cx	(x=0,1)
输出参数{out}	
-	-
返回值	
uint32_t	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 3-261. 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 3-262. 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 3-263. 函数 i2c\_nack\_enable

函数名称	i2c_nack_enable
函数原型	void i2c_nack_enable(uint32_t i2c_periph);
功能描述	从机模式下产生 NACK
先决条件	-
被调用函数	-
输入参数{in}	
i2c_periph	I2C 外设
I2Cx	(x=0,1)
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* generate a NACK in slave mode */
```

```
i2c_nack_enable(I2C0);
```

### 函数 i2c\_nack\_disable

函数i2c\_nack\_disable描述见下表:

表 3-264. 函数 i2c\_nack\_disable

函数名称	i2c_nack_disable
函数原型	void i2c_nack_disable(uint32_t i2c_periph);
功能描述	从机模式下产生 ACK
先决条件	-
被调用函数	-
输入参数{in}	
i2c_periph	I2C 外设
I2Cx	(x=0,1)
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* generate a ACK in slave mode */
```

```
i2c_nack_disable(I2C0);
```

### 函数 i2c\_wakeup\_from\_deepsleep\_enable

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

**表 3-265. 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=0,1)
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 3-266. 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=0,1)
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 3-267. 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 3-268. 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 3-269. 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 3-270. 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 3-271. 函数 i2c\_data\_transmit**

函数名称	i2c_data_transmit
函数原型	void i2c_data_transmit(uint32_t i2c_periph, uint8_t data);
功能描述	发送数据
先决条件	-
被调用函数	-
输入参数{in}	
i2c_periph	I2C 外设
I2Cx	(x=0,1)
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 3-272. 函数 i2c\_data\_receive**

函数名称	i2c_data_receive
函数原型	Uin8_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	

uint8_t	0x00..0xFF
---------	------------

例如:

```
/* I2C0 receive data */
```

```
uint8_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 3-273. 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 3-274. 函数 i2c\_reload\_disable

函数名称	i2c_reload_disable
函数原型	void i2c_reload_disable(uint32_t i2c_periph);
功能描述	禁能 I2C 重载模式
先决条件	-
被调用函数	-
输入参数{in}	
i2c_periph	I2C 外设
I2Cx	(x=0,1)
输出参数{out}	

-	-
返回值	
-	-

例如：

```
/* disable I2C reload mode */
```

```
i2c_reload_disable(I2C0);
```

### 函数 i2c\_transfer\_byte\_number\_config

函数i2c\_transfer\_byte\_number\_config描述见下表：

表 3-275. 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	0x00-0xFF，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 3-276. 函数 i2c\_dma\_enable

函数名称	i2c_dma_enable
函数原型	void i2c_dma_enable(uint32_t i2c_periph, uint8_t dma);
功能描述	使能发送/接收模式下 DMA
先决条件	-
被调用函数	-

输入参数{in}	
<b>i2c_periph</b>	I2C 外设
<i>I2Cx</i>	(x=0,1)
输入参数{in}	
<b>dma</b>	I2C DMA
<i>I2C_DMA_TRANSMIT</i>	采用 DMA 方式发送数据
<i>I2C_DMA_RECEIVE</i>	采用 DMA 方式接收数据
输出参数{out}	
-	-
返回值	
-	-

例如：

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

```
i2c_dma_enable(I2C0, I2C_DMA_RECEIVE);
```

### 函数 i2c\_dma\_disable

函数i2c\_dma\_disable描述见下表：

表 3-277. 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 3-278. 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)
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 3-279. 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)
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 3-280. 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)
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 3-281. 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)
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 3-282. 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)
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 3-283. 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)
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 3-284. 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)
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 3-285. 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)
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 3-286. 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)
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 3-287. 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)
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 3-288. 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 3-289. 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 3-290. 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 3-291. 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 3-292. 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 3-293. 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 3-294. 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 3-295. 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 3-296. 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 3-297. 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 3-298. 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 3-299. 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 3-237. 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 3-300. 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 3-237. 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);
```

## 3.12. MISC

MISC 是对嵌套向量中断控制器（NVIC）和系统定时器（SysTick）操作的软件包。章节 [3.12.1](#) 描述了 NVIC 和 SysTick 的寄存器列表，章节 [3.12.2](#) 对 MISC 库函数进行说明。

### 3.12.1. 外设寄存器说明

表 3-301. NVIC 寄存器

寄存器名称	寄存器描述
ISER <sup>(1)</sup>	中断使能寄存器
ICER <sup>(1)</sup>	中断禁能寄存器
ISPR <sup>(1)</sup>	中断挂起寄存器
ICPR <sup>(1)</sup>	中断清除寄存器
IABR <sup>(1)</sup>	中断活动状态寄存器
ITNS <sup>(1)</sup>	中断不安全状态寄存器
IPR <sup>(1)</sup>	中断优先级寄存器
CPUID <sup>(2)</sup>	CPUID寄存器
ICSR <sup>(2)</sup>	中断控制及状态寄存器
VTOR <sup>(2)</sup>	向量表偏移量寄存器
AIRCR <sup>(2)</sup>	应用程序中断及复位控制寄存器
SCR <sup>(2)</sup>	系统控制寄存器
CCR <sup>(2)</sup>	配置与控制寄存器
SHPR <sup>(2)</sup>	系统异常优先级寄存器
SHCSR <sup>(2)</sup>	系统异常控制及状态寄存器

1. 参考 core\_cm23.h 文件中定义的结构体类型 NVIC\_Type
2. 参考 core\_cm23.h 文件中定义的结构体类型 SCB\_Type

表 3-302. SysTick 寄存器

寄存器名称	寄存器描述
CTRL <sup>(1)</sup>	SysTick控制和状态寄存器
LOAD <sup>(1)</sup>	SysTick重载值寄存器
VAL <sup>(1)</sup>	SysTick当前值寄存器
CALIB <sup>(1)</sup>	SysTick校准寄存器

1. 参考 core\_cm23.h 文件中定义的结构体类型 SysTick\_Type

### 3.12.2. 外设库函数说明

MISC库函数列表如下表所示：

表 3-303. MISC 库函数

库函数名称	库函数描述
nvic_irq_enable	使能NVIC的中断
nvic_irq_disable	禁能NVIC的中断

库函数名称	库函数描述
nvic_vector_table_set	设置向量表基地址
nvic_system_reset	复位MCU
system_lowpower_set	设置系统低功耗模式状态
system_lowpower_reset	复位系统低功耗模式状态
systick_clksource_set	设置系统定时器时钟源

## 枚举类型 IRQn\_Type

表 3-304. 枚举类型 IRQn\_Type

成员名称	功能描述
WWDGT_IRQn	窗口看门狗中断
TIMESTAMP_IRQn	RTC 时间戳中断
FMC_IRQn	FMC 中断
RCU_IRQn	RCU 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	DMA0 通道 0 全局中断
DMA_Channel1_IRQn	DMA0 通道 1 全局中断
DMA_Channel2_IRQn	DMA0 通道 2 全局中断
ADC_IRQn	ADC 全局中断
USART0_IRQn	USART0 全局中断
USART1_IRQn	USART1 全局中断
USART2_IRQn	USART2 全局中断
I2C0_EV_IRQn	I2C0 事件中断
I2C0_ER_IRQn	I2C0 错误中断
I2C1_EV_IRQn	I2C1 事件中断
I2C1_ER_IRQn	I2C1 错误中断
SPI0_IRQn	SPI0 全局中断
SPI1_IRQn	SPI1 全局中断
RTC_Alarm_IRQn	RTC 闹钟中断
EXTI5_9_IRQn	EXTI 线[9:5]中断
TIMER0_TRG_CMT_UP_BRK_IRQn	TIMER0 触发, 通道换相, 更新, 中止中断
TIMER0_Channel_IRQn	TIMER0 捕获比较中断
TIMER2_IRQn	TIMER2 中断
TIMER13_IRQn	TIMER13 中断
TIMER15_IRQn	TIMER15 中断
TIMER16_IRQn	TIMER16 中断
EXTI10_15_IRQn	EXTI 线[15:10]中断
DMAMUX_IRQn	DMAMUX 全局中断

CMP0_IRQn	CMP0 全局中断
CMP1_IRQn	CMP1 全局中断
I2C0_WKUP_IRQn	I2C0 唤醒中断
I2C1_WKUP_IRQn	I2C1 唤醒中断
USART0_WKUP_IRQn	USART0 唤醒中断

### 函数 nvic\_irq\_enable

函数nvic\_irq\_enable描述见下表：

**表 3-305. 函数 nvic\_irq\_enable**

函数名称	nvic_irq_enable
函数原形	void nvic_irq_enable(uint8_t nvic_irq, uint8_t nvic_irq_priority);
功能描述	使能NVIC中断
先决条件	-
被调用函数	NVIC_SetPriority、NVIC_EnableIRQ
输入参数{in}	
nvic_irq	NVIC中断，参考枚举类型 <a href="#">表3-304. 枚举类型IRQn_Type</a>
输入参数{in}	
nvic_irq_priority	优先级（0~3）
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* enable window watchDog timer interrupt, priority is 1 */
```

```
nvic_irq_enable(WWDGT_IRQn, 1);
```

### 函数 nvic\_irq\_disable

函数nvic\_irq\_disable描述见下表：

**表 3-306. 函数 nvic\_irq\_disable**

函数名称	nvic_irq_disable
函数原形	void nvic_irq_disable(uint8_t nvic_irq);
功能描述	禁能NVIC中断
先决条件	-
被调用函数	NVIC_DisableIRQ
输入参数{in}	
nvic_irq	NVIC中断，参考枚举类型 <a href="#">表3-304. 枚举类型IRQn_Type</a>
输出参数{out}	
-	-
返回值	

-	-
---	---

例如：

```
/* disable window watchDog timer interrupt */
```

```
nvic_irq_disable(WWDGT_IRQn);
```

### 函数 nvic\_vector\_table\_set

函数nvic\_vector\_table\_set描述见下表：

表 3-307. 函数 nvic\_vector\_table\_set

函数名称	nvic_vector_table_set
函数原形	void nvic_vector_table_set(uint32_t nvic_vect_tab, uint32_t offset);
功能描述	设置向量表基地址
先决条件	-
被调用函数	-
输入参数{in}	
nvic_vect_tab	RAM或者FLASH基地址
NVIC_VECTTAB_RAM	RAM基地址
NVIC_VECTTAB_FLASH	FLASH基地址
输入参数{in}	
offset	向量表偏移量（向量表地址=基地址+偏移量）
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* set vector table address = NVIC_VECTTAB_FLASH + 0x200 */
```

```
nvic_vector_table_set(NVIC_VECTTAB_FLASH, 0x200);
```

表 3-308. 函数 nvic\_system\_reset

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

返回值	
-	-

例如：

```
/* reset the MCU */
nvic_system_reset();
```

### 函数 **system\_lowpower\_set**

函数system\_lowpower\_set描述见下表：

**表 3-309. 函数 system\_lowpower\_set**

函数名称	system_lowpower_set
函数原形	void system_lowpower_set(uint8_t lowpower_mode);
功能描述	设置系统低功耗模式状态
先决条件	-
被调用函数	-
输入参数{in}	
lowpower_mode	系统低功耗模式的状态
SCB_LPM_SLEEP_EXIT_ISR	该位为1时，退出ISR时一直处于低功耗模式
SCB_LPM_DEEPSLEEP	该位为1时，系统处于deep sleep模式
SCB_LPM_WAKE_BY_ALL_INT	该位为1时，低功耗模式可以被所有中断唤醒（无论中断是否被使能）
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* the system always enter low power mode by exiting from ISR */
system_lowpower_set(SCB_LPM_SLEEP_EXIT_ISR);
```

### 函数 **system\_lowpower\_reset**

函数system\_lowpower\_reset描述见下表：

**表 3-310. 函数 system\_lowpower\_reset**

函数名称	system_lowpower_reset
函数原形	void system_lowpower_reset(uint8_t lowpower_mode);
功能描述	复位系统低功耗模式状态
先决条件	-

被调用函数	-
输入参数{in}	
lowpower_mode	系统低功耗模式的状态
SCB_LPM_SLEEP_EXIT_ISR	该位为0时，系统将通过退出ISR退出低功耗模式
SCB_LPM_DEEPSLEEP	该位为0时，系统进入sleep模式
SCB_LPM_WAKEUP_BY_ALL_INT	该位为0时，系统只能被使能的中断唤醒
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* the system will exit low power mode by exiting from ISR */
system_lowpower_reset(SCB_LPM_SLEEP_EXIT_ISR);
```

### 函数 systick\_clksource\_set

函数systick\_clksource\_set描述见下表：

表 3-311. 函数 systick\_clksource\_set

函数名称	systick_clksource_set
函数原形	void systick_clksource_set(uint32_t systick_clksource);
功能描述	设置SysTick时钟源
先决条件	-
被调用函数	-
输入参数{in}	
systick_clksource	SysTick时钟源
SYSTICK_CLKSOURCE_HCLK	SysTick时钟源为AHB时钟
SYSTICK_CLKSOURCE_HCLK_DIV8	SysTick时钟源为AHB时钟的8分频
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* systick clock source is HCLK/8 */
systick_clksource_set(SYSTICK_CLKSOURCE_HCLK_DIV8);
```



### 3.13. PMU

电源管理单元提供了六种省电模式，包括运行模式 1、睡眠模式、睡眠模式 1、深度睡眠模式、深度睡眠模式 1 和待机模式。章节 [3.13.1](#) 描述了 PMU 的寄存器列表，章节 [3.13.2](#) 对 PMU 库函数进行说明。

#### 3.13.1. 外设寄存器说明

PMU 寄存器列表如下表所示：

表 3-312. PMU 寄存器

寄存器名称	寄存器描述
PMU_CTL0	控制寄存器0
PMU_CS	电源控制和状态寄存器
PMU_CTL1	控制寄存器1
PMU_STAT	状态寄存器
PMU_PAR	参数寄存器

#### 3.13.2. 外设库函数说明

PMU 库函数列表如下表所示：

表 3-313. PMU 库函数

库函数名称	库函数描述
pmu_deinit	复位PMU寄存器
pmu_deepsleep_voltage_select	选择低压检测阈值
pmu_low_power_enable	在Run/Sleep模式下使能低驱
pmu_low_power_disable	在Run/Sleep模式下禁能低驱
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	使能PMU WKUP引脚唤醒
pmu_wakeup_pin_disable	禁能PMU WKUP引脚唤醒
pmu_backup_write_enable	备份域写使能
pmu_backup_write_disable	备份域写禁能
pmu_eflash_run_power_config	配置进入运行/运行1模式后FLASH电源状态
pmu_eflash_deepsleep_power_config	配置进入深度睡眠/深度睡眠1模式后FLASH电源状态
pmu_eflash_wakeup_time_config	配置FLASH唤醒时间，使用IRC48M计数
pmu_deepsleep_wait_time_config	配置在进入深度睡眠/深度睡眠1之前IRC48M计数
pmu_flag_get	获取PMU标志位状态
pmu_flag_clear	清除PMU标志位状态

**函数 pmu\_deinit**

函数 pmu\_deinit 描述见下表：

**表 3-314. 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\_deepsleep\_voltage\_select**

函数 pmu\_deepsleep\_voltage\_select 描述见下表：

**表 3-315. 函数 pmu\_deepsleep\_voltage\_select**

函数名称	pmu_deepsleep_voltage_select
函数原型	void pmu_deepsleep_voltage_select(uint32_t dsv_n);
功能描述	选择深度睡眠模式下的电压
先决条件	-
被调用函数	-
输入参数{in}	
deepsleepvol	深度睡眠电
PMU_DSV_0	0.9V (仅在NPLDO关闭下有效)
PMU_DSV_1	1.0V
PMU_DSV_2	1.1V
PMU_DSV_3	1.2V
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* select deepsleep mode voltage as 1.0V */
```

```
pmu_deepsleep_voltage_select(PMU_DSV_1);
```

### 函数 pmu\_low\_power\_ldo\_enable

函数pmu\_low\_power\_ldo\_enable描述见下表:

表 3-316. 函数 pmu\_low\_power\_ldo\_enable

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

例如:

```
/* enable low power LDO */  
  
pmu_low_power_ldo_enable();
```

### 函数 pmu\_low\_power\_ldo\_disable

函数pmu\_low\_power\_ldo\_disable描述见下表:

表 3-317. 函数 pmu\_low\_power\_ldo\_disable

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

例如:

```
/* disable low power LDO */  
  
pmu_low_power_ldo_disable();
```

**函数 pmu\_to\_sleepmode**

函数 pmu\_to\_sleepmode 描述见下表:

**表 3-318. 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 3-319. 函数 pmu\_to\_deepsleepmode**

函数名称	pmu_to_deepsleepmode
函数原型	void pmu_to_deepsleepmode(uint8_t deepsleepmodecmd, uint8_t deepsleepmode);
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}	
deepsleepmodecmd	进入深度睡眠模式命令
WFI_CMD	WFI命令
WFE_CMD	WFE命令
输入参数{in}	
deepsleepmode	深度睡眠模式
PMU_DEEPSLEEP	深度睡眠模式
PMU_DEEPSLEEP1	深度睡眠模式1
输出参数{out}	
-	-

返回值	
-	-

例如：

```
/* PMU work in Deep-sleep mode */
```

```
pmu_to_deepsleepmode(WFI_CMD, PMU_DEEPSLEEP);
```

### 函数 pmu\_to\_standbymode

函数 pmu\_to\_standbymode 描述见下表：

表 3-320. 函数 pmu\_to\_standbymode

函数名称	pmu_to_standbymode
函数原型	void pmu_to_standbymode();
功能描述	进入待机模式
先决条件	-
被调用函数	-
输入参数{in}	
-	-
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* PMU work in standby mode */
```

```
pmu_to_standby(WFI_CMD);
```

### 函数 pmu\_wakeup\_pin\_enable

函数 pmu\_wakeup\_pin\_enable 描述见下表：

表 3-321. 函数 pmu\_wakeup\_pin\_enable

函数名称	pmu_wakeup_pin_enable
函数原型	void pmu_wakeup_pin_enable(uint32_t wakeup_pin);
功能描述	使能PMU WKUP引脚唤醒
先决条件	-
被调用函数	-
输入参数{in}	
wakeup_pin	唤醒引脚
PMU_WAKEUP_PIN0	WKUP Pin 0 (PA0)
PMU_WAKEUP_PIN1	WKUP Pin 1 (PC13 / PA4)

N1	
PMU_WAKEUP_PIN2	WKUP Pin 2 (PB6)
PMU_WAKEUP_PIN3	WKUP Pin 3 (PA2)
PMU_WAKEUP_PIN5	WKUP Pin 5 (PB5)
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* enable wakeup pin0 */
```

```
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 3-322. 函数 pmu\_wakeup\_pin\_disable

函数名称	pmu_wakeup_pin_disable
函数原型	void pmu_wakeup_pin_disable(uint32_t wakeup_pin);
功能描述	禁能PMU WKUP引脚唤醒
先决条件	-
被调用函数	-
输入参数{in}	
wakeup_pin	唤醒引脚
PMU_WAKEUP_PIN0	WKUP Pin 0 (PA0)
PMU_WAKEUP_PIN1	WKUP Pin 1 (PC13 / PA4)
PMU_WAKEUP_PIN2	WKUP Pin 2 (PB6)
PMU_WAKEUP_PIN3	WKUP Pin 3 (PA2)
PMU_WAKEUP_PIN5	WKUP Pin 5 (PB5)
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* disable wakeup pin0 */
```

```
pmu_wakeup_pin_disable(PMU_WAKEUP_PIN0);
```

### 函数 pmu\_backup\_write\_enable

函数 pmu\_backup\_write\_enable 描述见下表：

表 3-323. 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 3-324. 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\_eflash\_run\_power\_config

函数pmu\_eflash\_run\_power\_config描述见下表：

**表 3-325. 函数 pmu\_eflash\_run\_power\_config**

函数名称	pmu_eflash_run_power_config
函数原型	void pmu_eflash_run_power_config(ControlStatus state);
功能描述	配置EFLASH在运行/运行1模式下的电源状态
先决条件	-
被调用函数	-
输入参数{in}	
state	电源状态
ENABLE	eflash域上电
DISABLE	eflash域掉电
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* configure power state of EFLASH */
pmu_eflash_run_power_config(ENABLE);
```

### 函数 pmu\_eflash\_deepsleep\_power\_config

函数pmu\_eflash\_deepsleep\_power\_config描述见下表：

**表 3-326. 函数 pmu\_eflash\_deepsleep\_power\_config**

函数名称	pmu_eflash_deepsleep_power_config
函数原型	void pmu_eflash_deepsleep_power_config(ControlStatus state);
功能描述	配置EFLASH在深度睡眠/深度睡眠1模式下的电源状态
先决条件	-
被调用函数	-
输入参数{in}	
state	电源状态
ENABLE	eflash域上电
DISABLE	eflash域掉电
输出参数{out}	
-	-
返回值	
-	-



例如:

```
/* configure power state of EFLASH */  
  
pmu_eflash_deepsleep_power_config(DISABLE);
```

### 函数 pmu\_eflash\_wakeup\_time\_config

函数pmu\_eflash\_wakeup\_time\_config描述见下表:

表 3-327. 函数 pmu\_eflash\_wakeup\_time\_config

函数名称	pmu_eflash_wakeup_time_config
函数原型	void pmu_eflash_wakeup_time_config(uint32_t wakeup_time);
功能描述	EFLASH唤醒时间, 使用IRC48M计数
先决条件	-
被调用函数	-
输入参数{in}	
wakeup_time	IRC48M counter for eflash wakeup time
uint32_t	0x0~0xFF
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* configure IRC48M counter for eflash wakeup time */  
  
pmu_eflash_wakeup_time_config(100);
```

### 函数 pmu\_deepsleep\_wait\_time\_config

函数pmu\_deepsleep\_wait\_time\_config描述见下表:

表 3-328. 函数 pmu\_deepsleep\_wait\_time\_config

函数名称	pmu_deepsleep_wait_time_config
函数原型	void pmu_deepsleep_wait_time_config(uint32_t wait_time);
功能描述	配置进入深度睡眠/深度睡眠1前的等待时间, 使用IRC48M计数
先决条件	-
被调用函数	-
输入参数{in}	
wait_time	等待时间, 以IRC48M计数
uint32_t	0x0~0x1F
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* configure IRC48M counter before enter Deepsleep/Deepsleep1 mode */
```

```
pmu_deepsleep_wait_time_config(20);
```

```
pmu_to_deepsleepmode(WFI_CMD, PMU_DEEPSLEEP);
```

### 函数 pmu\_flag\_get

函数pmu\_flag\_get描述见下表：

表 3-329. 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PMU标志位状态
先决条件	-
被调用函数	-
输入参数{in}	
flag	标志位
PMU_FLAG_WAKEUP	唤醒标志
PMU_FLAG_STANDBY	待机标志
PMU_FLAG_LDOVSRF	LDO电压选择准备就绪标志
PMU_FLAG_NPRDY	正常功耗LDO准备就绪标志
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 3-330. 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PMU标志位状态
先决条件	-
被调用函数	-
输入参数{in}	

flag	标志位
PMU_FLAG_WAKEUP	唤醒标志
PMU_FLAG_STANDBY	待机标志
输出参数{out}	
-	-
返回值	
-	-

例如:

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

```
pmu_flag_clear(PMU_FLAG_STANDBY);
```

## 3.14. 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 [3.14.1](#) 描述了 RCU 的寄存器列表，章节 [3.14.2](#) 对 RCU 库函数进行说明。

### 3.14.1. 外设寄存器说明

RCU寄存器列表如下表所示：

表 3-331. RCU 寄存器

寄存器名称	寄存器描述
RCU_CTL0	控制 0 寄存器
RCU_CFG0	配置寄存器 0
RCU_INT	中断寄存器
RCU_AHB1RST	AHB1 复位寄存器
RCU_AHB2RST	AHB1 复位寄存器
RCU_APBRSR	APB 复位寄存器
RCU_AHB1EN	AHB1 使能寄存器
RCU_AHB2EN	AHB2 使能寄存器
RCU_APBEN	APB 使能寄存器
RCU_AHB1SPDPEN	AHB1 睡眠和深度睡眠模式使能寄存器
RCU_AHB2SPDPEN	AHB2 睡眠和深度睡眠模式使能寄存器
RCU_APBSPDPEN	APB 睡眠和深度睡眠模式使能寄存器
RCU_CTL1	控制 1 寄存器
RCU_RSTSCK	复位源/时钟寄存器
RCU_CFG1	配置寄存器 1

### 3.14.2. 外设库函数说明

RCU 库函数列表如下表所示：

**表 3-332. RCU 库函数**

库函数名称	库函数描述
rcu_deinit	失能 RCU
rcu_periph_clock_enable	使能外设时钟
rcu_periph_clock_disable	失能外设时钟
rcu_periph_clock_sleep_enable	在睡眠模式下使能外设时钟
rcu_periph_clock_sleep_disable	失能外设时钟当睡眠模式时
rcu_periph_reset_enable	复位外设
rcu_periph_reset_disable	失能复位该外设
rcu_bkp_reset_enable	复位 BKP（适用于 GD32C231 和 GD32C221）
rcu_bkp_reset_disable	失能 BKP 复位（适用于 GD32C231 和 GD32C221）
rcu_rtc_reset_enable	复位 RTC（适用于 GD32C211）
rcu_rtc_reset_disable	失能 RTC 复位（适用于 GD32C211）
rcu_system_clock_source_config	配置系统时钟源
rcu_system_clock_source_get	获取系统时钟源
rcu_ahb_clock_config	配置 AHB 时钟分频器选择
rcu_apb_clock_config	配置 APB 时钟分频器选择
rcu_adc_clock_config	配置 ADC 时钟源和预分频器选择（适用于 GD32C231 和 GD32C221）
rcu_adc_clock_config	配置 ADC 时钟源和分频器选择（适用于 GD32C211）
rcu_ckout0_config	配置 CK_OUT0 时钟源和分频器
rcu_ckout1_config	配置 CK_OUT1 时钟源和分频器
rcu_lscout_enable	使能低速时钟输出
rcu_lscout_disable	失能低速时钟输出
rcu_lscout_config	配置 LSCOUT 时钟源
rcu_usart_clock_config	配置 USART 时钟源选择
rcu_i2c_clock_config	配置 I2Cx(x=0,1)时钟源选择
rcu_i2s_clock_config	配置 I2S 时钟源选择
rcu_irc48mdiv_sys_clock_config	配置 IRC48MDIV_SYS 时钟选择
rcu_irc48mdiv_per_clock_config	配置 IRC48MDIV 时钟选择

rcu_rtc_clock_config	配置 RTC 时钟源选择
rcu_lxtal_drive_capability_config	配置 LXTAL 驱动能力
rcu_osc_i_stab_wait	等待直到振荡器稳定标志被置位
rcu_osc_i_on	使能振荡器
rcu_osc_i_start_enable	当 osci 准备好时，启动电路使能（适用于 GD32C211）
rcu_osc_i_start_disable	启动电路在 osci 准备好时失能（适用于 GD32C211）
rcu_osc_i_off	关闭振荡器
rcu_osc_i_bypass_mode_enable	使能振荡器旁路模式之前，必须复位 HXTALEN 或 LXTALEN
rcu_osc_i_bypass_mode_disable	失能振荡器旁路模式，需先复位 HXTALEN 或 LXTALEN
rcu_irc48m_adjust_value_set	设置 IRC48M 调整值
rcu_lxtal_stab_reset_enable	使能 LXTAL 稳定复位
rcu_lxtal_stab_reset_disable	失能 LXTAL 稳定复位
rcu_hxtal_clock_monitor_enable	使能 HXTAL 时钟监视器
rcu_hxtal_clock_monitor_disable	失能 HXTAL 时钟监测器
rcu_lxtal_clock_monitor_enable	使能 LXTAL 时钟监测器
rcu_lxtal_clock_monitor_disable	失能 LXTAL 时钟监测器
rcu_clock_freq_get	获取系统时钟、总线和外设时钟频率
rcu_flag_get	获取时钟稳定和复位标志
rcu_all_reset_flag_clear	清除复位标志
rcu_interrupt_enable	使能稳定中断
rcu_interrupt_disable	失能稳定中断
rcu_interrupt_flag_get	获取时钟稳定中断和 CKM 标志
rcu_interrupt_flag_clear	清除中断标志

### 枚举类型 rcu\_periph\_enum

表 3-333. 枚举类型 rcu\_periph\_enum

Member name	Descriptions
RCU_FMC	FMC 时钟
RCU_CRC	CRC 时钟
RCU_DMA	DMA 时钟
RCU_DMAMUX	DMAMUX 时钟
RCU_GPIOA	GPIOA 时钟

RCU_GPIOB	GPIOB 时钟
RCU_GPIOC	GPIOC 时钟
RCU_GPIOD	GPIOD 时钟
RCU_GPIOF	GPIOF 时钟
RCU_SYSCFG	SYSCFG 时钟
RCU_CMP	CMP 时钟
RCU_WWDGT	WWDGT 时钟
RCU_ADC	ADC 时钟
RCU_TIMER0	TIMER0 时钟
RCU_TIMER2	TIMER2 时钟
RCU_SPI0	SPI0 时钟
RCU_SPI1	SPI1 时钟
RCU_USART0	USART0 时钟
RCU_USART1	USART1 时钟
RCU_TIMER13	TIMER13 时钟
RCU_TIMER15	TIMER15 时钟
RCU_TIMER16	TIMER16 时钟
RCU_USART2	USART2 时钟
RCU_I2C0	I2C0 时钟
RCU_I2C1	I2C1 时钟
RCU_DBGMCU	DBGMCU 时钟
RCU_PMU	PMU 时钟
RCU_RTC	RTC 时钟

### 枚举类型 rcu\_periph\_sleep\_enum

表 3-334. 枚举类型 rcu\_periph\_sleep\_enum

Member name	Descriptions
RCU_SRAM_SLP	SRAM 时钟
RCU_FMC_SLP	FMC 时钟
RCU_CRC_SLP	CRC 时钟
RCU_DMA_SLP	DMA 时钟
RCU_DMAMUX_SLP	DMAMUX 时钟
RCU_GPIOA_SLP	GPIOA 时钟
RCU_GPIOB_SLP	GPIOB 时钟
RCU_GPIOC_SLP	GPIOC 时钟
RCU_GPIOD_SLP	GPIOD 时钟
RCU_GPIOF_SLP	GPIOF 时钟
RCU_SYSCFG_SLP	SYSCFG 时钟
RCU_CMP_SLP	CMP 时钟
RCU_WWDGT_SLP	WWDGT 时钟
RCU_ADC_SLP	ADC 时钟
RCU_TIMER0_SLP	TIMER0 时钟

RCU_TIMER2_SLP	TIMER2 时钟
RCU_SPI0_SLP	SPI0 时钟
RCU_SPI1_SLP	SPI1 时钟
RCU_USART0_SLP	USART0 时钟
RCU_USART1_SLP	USART1 时钟
RCU_TIMER13_SLP	TIMER13 时钟
RCU_TIMER15_SLP	TIMER15 时钟
RCU_TIMER16_SLP	TIMER16 时钟
RCU_USART2_SLP	USART2 时钟
RCU_I2C0_SLP	I2C0 时钟
RCU_I2C1_SLP	I2C1 时钟
RCU_PMU_SLP	PMU 时钟

### 枚举类型 `rcu_periph_reset_enum`

表 3-335. 枚举类型 `rcu_periph_reset_enum`

Member name	Descriptions
RCU_CRCRST	CRC 时钟
RCU_DMARST	DMA 时钟
RCU_DMAMUXRST	DMA 时钟
RCU_GPIOARST	GPIOA 时钟
RCU_GPIOBRST	GPIOB 时钟
RCU_GPIOCRST	GPIOC 时钟
RCU_GPIODRST	GPIOD 时钟
RCU_GPIOFRST	GPIOF 时钟
RCU_SYSCFGRST	SYSCFG 时钟
RCU_CMPRST	CMP 时钟
RCU_WWDGTRST	WWDGT 时钟
RCU_ADCRST	ADC 时钟
RCU_TIMER0RST	TIMER0 时钟
RCU_TIMER2RST	TIMER2 时钟
RCU_SPI0RST	SPI0 时钟
RCU_SPI1RST	SPI1 时钟
RCU_USART0RST	USART0 时钟
RCU_USART1RST	USART1 时钟
RCU_TIMER13RST	TIMER13 时钟
RCU_TIMER15RST	TIMER15 时钟
RCU_TIMER16RST	TIMER16 时钟
RCU_USART2RST	USART2 时钟
RCU_I2C0RST	I2C0 时钟
RCU_I2C1RST	I2C1 时钟
RCU_PMURST	PMU 时钟

### 枚举类型 rcu\_flag\_enum

表 3-336. 枚举类型 rcu\_flag\_enum

Member name	Descriptions
RCU_FLAG_IRC32KST B	IRC32K 稳定标志
RCU_FLAG_LXTALST B	LXTAL 稳定标志
RCU_FLAG_IRC48MS TB	IRC48M 稳定标志
RCU_FLAG_HXTALST B	HXTAL 稳定标志
RCU_FLAG_OBLRST	选项字节加载器复位标志
RCU_FLAG_BORRST	BOR 复位标志
RCU_FLAG_EPRST	外部 PIN 复位标志
RCU_FLAG_PORRST	电源复位标志
RCU_FLAG_SWRST	SW 复位标志
RCU_FLAG_FWDGTR ST	FWDGT 复位标志
RCU_FLAG_WWDGTR ST	WWDGT 复位标志
RCU_FLAG_LPRST	低功耗复位标志
RCU_FLAG_LCKCMD	LXTAL 时钟失效检测标志

### 枚举类型 rcu\_int\_flag\_enum

表 3-337. 枚举类型 rcu\_int\_flag\_enum

Member name	Descriptions
RCU_INT_FLAG_IRC3 2KSTB	IRC32K 稳定中断标志
RCU_INT_FLAG_LXTA LSTB	LXTAL 稳定中断标志
RCU_INT_FLAG_IRC4 8MSTB	IRC48M 稳定中断标志
RCU_INT_FLAG_LXTA LCKM	LXTAL 时钟卡死中断标志
RCU_INT_FLAG_HXTA LSTB	HXTAL 稳定中断标志
RCU_INT_FLAG_CKM	CKM 中断标志

### 枚举类型 rcu\_int\_flag\_clear\_enum

表 3-338. 枚举类型 rcu\_int\_flag\_clear\_enum

Member name	Descriptions
-------------	--------------



RCU_INT_FLAG_IRC3 2KSTB_CLR	IRC32K 稳定中断标志清零
RCU_INT_FLAG_LXTA LSTB_CLR	LXTAL 稳定中断标志清除
RCU_INT_FLAG_IRC4 8MSTB_CLR	IRC48M 稳定中断标志清除
RCU_INT_FLAG_LXTA LCKM_CLR	LXTAL 时钟卡死中断标志清除
RCU_INT_FLAG_HXTA LSTB_CLR	HXTAL 稳定中断标志清除
RCU_INT_FLAG_CKM _CLR	CKM 中断标志清除

### 枚举类型 `rcu_int_enum`

表 3-339. 枚举类型 `rcu_int_enum`

Member name	Descriptions
RCU_INT_IRC32KSTB	IRC32K 稳定中断
RCU_INT_LXTALSTB	LXTAL 稳定中断
RCU_INT_IRC48MSTB	IRC48M 稳定中断
RCU_INT_HXTALSTB	HXTAL 稳定中断

### 枚举类型 `rcu_osci_type_enum`

表 3-340. 枚举类型 `rcu_osci_type_enum`

Member name	Descriptions
RCU_HXTAL	HXTAL
RCU_LXTAL	LXTAL
RCU_IRC48M	IRC48M
RCU_IRC32K	IRC32K

### 枚举类型 `rcu_clock_freq_enum`

表 3-341. 枚举类型 `rcu_clock_freq_enum`

Member name	Descriptions
CK_SYS	系统时钟

### 枚举类型 `usart_idx_enum`

表 3-342. 枚举类型 `usart_idx_enum`

Member name	Descriptions
IDX_USART0	USART0 的索引
IDX_USART1	USART1 的索引

枚举类型 `i2c_idx_enum`表 3-343. 枚举类型 `i2c_idx_enum`

Member name	Descriptions
IDX_I2C0	I2C0 索引

函数 `rcu_deinit`

函数 `rcu_deinit` 描述见下表：

表 3-344. 函数 `rcu_deinit`

函数名称	<code>rcu_deinit</code>
函数原型	<code>void rcu_deinit(void)</code>
功能描述	失能 RCU
先决条件	-
被调用函数	-
输入参数 {in}	
-	-
输出参数 {out}	
-	-
返回值	
-	-

Example:

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

函数 `rcu_periph_clock_enable`

函数 `rcu_periph_clock_enable` 描述见下表：

表 3-345. 函数 `rcu_periph_clock_enable`

函数名称	<code>rcu_periph_clock_enable</code>
函数原型	<code>void rcu_periph_clock_enable(rcu_periph_enum periph)</code>
功能描述	使能外设时钟
先决条件	-
被调用函数	-
输入参数 {in}	
<b>periph</b>	RCU 外设, 请参考 <a href="#">表 3-333. 枚举类型 <code>rcu_periph_enum</code></a>
<code>RCU_FMC</code>	FMC 时钟
<code>RCU_CRC</code>	CRC 时钟
<code>RCU_DMA</code>	DMA 时钟
<code>RCU_DMAMUX</code>	DMAMUX 时钟
<code>RCU_GPIOx</code>	GPIO 端口时钟

$(x=A,B,C,D,F)$	
<i>RCU_SYSCFG</i>	SYSCFG 时钟
<i>RCU_CMP</i>	CMP 时钟
<i>RCU_WWDGT</i>	WWDGT 时钟
<i>RCU_ADC</i>	ADC 时钟
<i>RCU_TIMERx</i> $(x=0,2,13,15,16)$	定时器时钟
<i>RCU_SPIx</i> $(x=0,1)$	SPI 时钟
<i>RCU_USARTx</i> $(x=0,1,2)$	USART 时钟
<i>RCU_I2Cx</i> $(x=0,1)$	I2C 时钟
<i>RCU_DBGMCU</i>	DBGMCU 时钟
<i>RCU_PMU</i>	PMU 时钟
<i>RCU_RTC</i>	RTC 时钟
输出参数 {out}	
-	-
返回值	
-	-

Example:

```
/* CMP peripherals clock enable*/
```

```
rcu_periph_clock_enable(RCU_CMP);
```

### 函数 `rcu_periph_clock_disable`

函数 `rcu_periph_clock_disable` 描述见下表:

表 3-346. 函数 `rcu_periph_clock_disable`

函数名称	<code>rcu_periph_clock_disable</code>
函数原型	<code>void rcu_periph_clock_disable(rcu_periph_enum periph)</code>
功能描述	失能外设时钟
先决条件	-
被调用函数	-
输入参数 {in}	
<b>periph</b>	RCU 外设, 请参考 <a href="#">表 3-333. 枚举类型 <code>rcu_periph_enum</code></a>
<i>RCU_CRC</i>	CRC 时钟
<i>RCU_DMA</i>	DMA 时钟
<i>RCU_DMAMUX</i>	DMAMUX 时钟
<i>RCU_GPIOx</i> $(x=A,B,C,D,F)$	GPIO 端口时钟
<i>RCU_SYSCFG</i>	SYSCFG 时钟
<i>RCU_CMP</i>	CMP 时钟
<i>RCU_WWDGT</i>	WWDGT 时钟

<i>RCU_ADC</i>	ADC 时钟
<i>RCU_TIMERx</i> ( <i>x</i> =0,1,2,5,13,15,16)	定时器时钟
<i>RCU_SPIx</i> ( <i>x</i> =0,1)	SPI 时钟
<i>RCU_USARTx</i> ( <i>x</i> =0,1,2)	USART 时钟
<i>RCU_I2Cx</i> ( <i>x</i> =0,1)	I2C 时钟
<i>RCU_DBGMCU</i>	DBGMCU 时钟
<i>RCU_PMU</i>	PMU 时钟
<i>RCU_RTC</i>	RTC 时钟
输出参数 {out}	
-	-
返回值	
-	-

Example:

```
/* CMP peripherals clock disable*/
```

```
rcu_periph_clock_disable(RCU_CMP);
```

### 函数 `rcu_periph_clock_sleep_enable`

函数 `rcu_periph_clock_sleep_enable` 描述见下表:

表 3-347. 函数 `rcu_periph_clock_sleep_enable`

函数名称	<code>rcu_periph_clock_sleep_enable</code>
函数原型	<code>void rcu_periph_clock_sleep_enable(rcu_periph_sleep_enum periph)</code>
功能描述	在睡眠模式下使能外设时钟
先决条件	-
被调用函数	-
输入参数 {in}	
<b>periph</b>	RCU 外设, 请参阅 <a href="#">表 3-334. 枚举类型 <code>rcu_periph_sleep_enum</code></a>
<i>RCU_SRAM_SLP</i>	SRAM 时钟
<i>RCU_FMC_SLP</i>	FMC 时钟
<i>RCU_CRC_SLP</i>	CRC 时钟
<i>RCU_DMA_SLP</i>	DMA 时钟
<i>RCU_DMAMUX_SLP</i>	DMAMUX 时钟
<i>RCU_GPIOx_SLP</i> ( <i>x</i> = A,B,C,D,F)	GPIO 端口时钟
<i>RCU_SYSCFG_SLP</i>	SYSCFG 时钟
<i>RCU_CMP_SLP</i>	CMP 时钟
<i>RCU_WWDGT_SLP</i>	WWDGT 时钟
<i>RCU_ADC_SLP</i>	ADC 时钟
<i>RCU_TIMERx_SLP</i>	定时器时钟

$(x=0,2,13,15,16)$	
$RCU\_SPIx\_SLP$ ( $x = 0,1$ )	SPI 时钟
$RCU\_USARTx\_SLP$ ( $x = 0,1,2$ )	USART 时钟
$RCU\_I2Cx\_SLP$ ( $x = 0,1$ )	I2C 时钟
$RCU\_PMU\_SLP$	PMU 时钟
输出参数 {out}	
-	-
返回值	
-	-

Example:

```
/* CMP peripherals clock enable when sleep mode */
```

```
rcu_periph_clock_sleep_enable(RCU_CMP_SLP);
```

### 函数 rcu\_periph\_clock\_sleep\_disable

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

表 3-348. 函数 rcu\_periph\_clock\_sleep\_disable

函数名称	rcu_periph_clock_sleep_disable
函数原型	void rcu_periph_clock_sleep_disable(rcu_periph_sleep_enum periph)
功能描述	失能外设时钟当睡眠模式时
先决条件	-
被调用函数	-
输入参数 {in}	
periph	RCU 外设, 请参考 <a href="#">表 3-334. 枚举类型 rcu_periph_sleep_enum</a>
$RCU\_SRAM\_SLP$	SRAM 时钟
$RCU\_FMC\_SLP$	FMC 时钟
$RCU\_CRC\_SLP$	CRC 时钟
$RCU\_DMA\_SLP$	DMA 时钟
$RCU\_DMAMUX\_SLP$	DMAMUX 时钟
$RCU\_GPIOx\_SLP$ ( $x = A,B,C,D,F$ )	GPIO 端口时钟
$RCU\_SYSCFG\_SLP$	SYSCFG 时钟
$RCU\_CMP\_SLP$	CMP 时钟
$RCU\_WWDGT\_SLP$	WWDGT 时钟
$RCU\_ADC\_SLP$	ADC 时钟
$RCU\_TIMERx\_SLP$ ( $x=0,2,13,15,16$ )	定时器时钟
$RCU\_SPIx\_SLP$ ( $x =$	SPI 时钟

0,1)	
<i>RCU_USARTx_SLP</i> ( $x = 0, 1, 2$ )	USART 时钟
<i>RCU_I2Cx_SLP</i> ( $x = 0, 1$ )	I2C 时钟
<i>RCU_PMU_SLP</i>	PMU 时钟
输出参数 {out}	
-	-
返回值	
-	-

Example:

```
/* CMP peripherals clock disable when sleep mode */
rcu_periph_clock_sleep_disable(RCU_CMP_SLP);
```

### 函数 rcu\_periph\_reset\_enable

函数 rcu\_periph\_reset\_enable 描述见下表:

表 3-349. 函数 rcu\_periph\_reset\_enable

函数名称	rcu_periph_reset_enable
函数原型	void rcu_periph_reset_enable(rcu_periph_reset_enum periph_reset)
功能描述	复位外设
先决条件	-
被调用函数	-
输入参数 {in}	
periph_reset	RCU 外设复位, 请参阅 <a href="#">表 3-335. 枚举类型 rcu_periph_reset_enum</a>
<i>RCU_CRCRST</i>	复位 CRC
<i>RCU_DMARST</i>	复位 DMA
<i>RCU_DMAMUXRST</i>	复位 DMAMUX
<i>RCU_GPIOxRST</i> ( $x = A, B, C, D, F$ )	复位 GPIO 端口
<i>RCU_SYSCFGRST</i>	复位 SYSCFG
<i>RCU_CMPRST</i>	复位 CMP
<i>RCU_WWDGTRST</i>	复位 WWDGT
<i>RCU_ADCRST</i>	复位 ADC
<i>RCU_TIMERxRST</i> ( $x=0, 2, 13, 15, 16$ )	复位 TIMER
<i>RCU_SPlxRST</i> ( $x = 0, 1$ )	复位 SPI
<i>RCU_USARTxRST</i> ( $x = 0, 1, 2$ )	复位 USART
<i>RCU_I2CxRST</i> ( $x = 0, 1$ )	复位 I2C

<i>RCU_PMURST</i>	复位 PMU
输出参数 {out}	
-	-
返回值	
-	-

Example:

```
/* CMP peripherals reset enable */
rcu_periph_reset_enable(RCU_CMPRST);
```

### 函数 `rcu_periph_reset_disable`

函数 `rcu_periph_reset_disable` 描述见下表:

**表 3-350. 函数 `rcu_periph_reset_disable`**

函数名称	<code>rcu_periph_reset_disable</code>
函数原型	<code>void rcu_periph_reset_disable(rcu_periph_reset_enum periph_reset)</code>
功能描述	失能复位该外设
先决条件	-
被调用函数	-
输入参数 {in}	
<b>periph_reset</b>	RCU 外设复位, 请参阅 <a href="#">表 3-335. 枚举类型 <code>rcu_periph_reset_enum</code></a>
<i>RCU_CRCSRST</i>	复位 CRC
<i>RCU_DMARST</i>	复位 DMA
<i>RCU_DMAMUXRST</i>	复位 DMAMUX
<i>RCU_GPIOxRST</i> ( $x = A, B, C, D, F$ )	复位 GPIO 端口
<i>RCU_SYSCFGRST</i>	复位 SYSCFG
<i>RCU_CMPRST</i>	复位 CMP
<i>RCU_WWDGTRST</i>	复位 WWDGT
<i>RCU_ADCRST</i>	复位 ADC
<i>RCU_TIMERxRST</i> ( $x=0, 2, 13, 15, 16$ )	复位 TIMER
<i>RCU_SPIxRST</i> ( $x = 0, 1$ )	复位 SPI
<i>RCU_USARTxRST</i> ( $x = 0, 1, 2$ )	复位 USART
<i>RCU_I2CxRST</i> ( $x = 0, 1$ )	复位 I2C
<i>RCU_PMURST</i>	复位 PMU
输出参数 {out}	
-	-
返回值	
-	-

Example:

```
/* CMP peripherals reset disable */
rcu_periph_reset_enable(RCU_CMPRST);
```

**函数 rcu\_bkp\_reset\_enable**（适用于 GD32C231 和 GD32C221）

函数 rcu\_bkp\_reset\_enable 描述见下表：

**表 3-351. 函数 rcu\_bkp\_reset\_enable**

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

Example:

```
/* reset RTC enable */
rcu_rtc_reset_enable();
```

**函数 rcu\_bkp\_reset\_disable**（适用于 GD32C231 和 GD32C221）

函数 rcu\_bkp\_reset\_disable 描述见下表：

**表 3-352. 函数 rcu\_bkp\_reset\_disable**

函数名称	rcu_bkp_reset_disable
函数原型	void rcu_bkp_reset_disable(void)
功能描述	失能 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	
-	-

Example:

```
/* reset RTC disable */
```



```
rcu_rtc_reset_disable();
```

### 函数 rcu\_rtc\_reset\_enable（适用于 GD32C211）

函数 rcu\_rtc\_reset\_enable 描述见下表：

**表 3-353. 函数 rcu\_rtc\_reset\_enable**

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

Example:

```
/* reset RTC enable */
rcu_rtc_reset_enable();
```

### 函数 rcu\_rtc\_reset\_disable（适用于 GD32C211）

函数 rcu\_rtc\_reset\_disable 描述见下表：

**表 3-354. 函数 rcu\_rtc\_reset\_disable**

函数名称	rcu_rtc_reset_disable
函数原型	void rcu_rtc_reset_disable(void)
功能描述	失能 RTC 复位
先决条件	-
被调用函数	-
输入参数 {in}	
-	-
输出参数 {out}	
-	-
返回值	
-	-

Example:

```
/* reset RTC disable */
rcu_rtc_reset_disable();
```

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

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

**表 3-355. 函数 rcu\_system\_clock\_source\_config**

函数名称	rcu_system_clock_source_config
函数原型	void rcu_system_clock_source_config(uint32_t ck_sys)
功能描述	配置系统时钟源
先决条件	-
被调用函数	-
输入参数 {in}	
ck_sys	系统时钟源选择
RCU_CKSYSSRC_IRC48MDIV_SYS	选择 IRC48MDIV_SYS 作为 CK_SYS 的时钟源
RCU_CKSYSSRC_HXTAL	选择 CK_HXTAL 作为 CK_SYS 的时钟源
RCU_CKSYSSRC_IRC32K	选择 IRC32K 作为 CK_SYS 来源
RCU_CKSYSSRC_LXTAL	选择 LXTLA 作为 CK_SYS 源
输出参数 {out}	
-	-
返回值	
-	-

Example:

```
/* select 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 3-356. 函数 rcu\_system\_clock\_source\_get**

函数名称	rcu_system_clock_source_get
函数原型	uint32_t rcu_system_clock_source_get(void)
功能描述	获取系统时钟源
先决条件	-
被调用函数	-
输入参数 {in}	
-	-
输出参数 {out}	
-	-
返回值	

uint32_t	描述
<i>RCU_CKSYSSRC_IRC48MDIV_SYS</i>	选择 IRC48MDIV_SYS 作为 CK_SYS 的源
<i>RCU_CKSYSSRC_HXTAL</i>	选择 CK_HXTAL 作为 CK_SYS 的源
<i>RCU_CKSYSSRC_IRC32K</i>	选择 IRC32K 作为 CK_SYS 的时钟源
<i>RCU_CKSYSSRC_LXTAL</i>	选择 LXTLA 作为 CK_SYS 时钟源

Example:

```
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 3-357. 函数 rcu\_ahb\_clock\_config

函数名称	rcu_ahb_clock_config
函数原型	void rcu_ahb_clock_config(uint32_t ck_ahb)
功能描述	配置 AHB 时钟分频器选择
先决条件	-
被调用函数	-
输入参数 {in}	
ck_ahb	AHB 时钟分频器选择
<i>RCU_AHB_CKSYS_DIVx</i> , x=1, 2, 4, 8, 16, 64, 128, 256, 512	选择 CK_SYS/x 作为 CK_AHB
输出参数 {out}	
-	-
返回值	
-	-

Example:

```
/* configure CK_SYS / 128 */

rcu_ahb_clock_config(RCU_AHB_CKSYS_DIV128);
```

### 函数 rcu\_apb\_clock\_config

函数 rcu\_apb\_clock\_config 描述见下表:

表 3-358. 函数 rcu\_apb\_clock\_config

函数名称	rcu_apb_clock_config
函数原型	void rcu_apb_clock_config(uint32_t ck_apb)
功能描述	配置 APB 时钟分频器选择
先决条件	-
被调用函数	-
输入参数 {in}	
ck_apb	APB 时钟预分频选择
RCU_APB_CK_AHB_DIV1	选择 CK_AHB 作为 CK_APB
RCU_APB_CK_AHB_DIV2	选择 CK_AHB/2 作为 CK_APB
RCU_APB_CK_AHB_DIV4	选择 CK_AHB/4 作为 CK_APB
RCU_APB_CK_AHB_DIV8	选择 CK_AHB/8 作为 CK_APB
RCU_APB_CK_AHB_DIV16	选择 CK_AHB/16 作为 CK_APB
输出参数 {out}	
-	-
返回值	
-	-

Example:

```
/* configure CK_AHB / 16 as CK_APB1 */
rcu_apb1_clock_config(RCU_APB1_CK_AHB_DIV16);
```

函数 rcu\_adc\_clock\_config (适用于 GD32C231 和 GD32C221)

函数 rcu\_adc\_clock\_config 描述见下表:

表 3-359. 函数 rcu\_adc\_clock\_config

函数名称	rcu_adc_clock_config
函数原型	void rcu_adc_clock_config(uint32_t adc_clock_source, uint32_t ck_adc)
功能描述	配置 ADC 时钟源和预分频器选择
先决条件	-
被调用函数	-
输入参数 {in}	
adc_clock_source	ADC 时钟源选择
RCU_ADCSRC_CKSYS	ADC 时钟源选择 CK_SYS
RCU_ADCSRC_IRC48M_PER	ADC 时钟源选择 CK_IRC48MDIV_PER

输入参数 {in}	
<b>ck_adc</b>	ADC 时钟预分频选择
<i>RCU_ADCCCK_DIV1</i>	ADC 时钟预分频选择未分频
<i>RCU_ADCCCK_DIV2</i>	ADC 时钟预分频选择除以 2
<i>RCU_ADCCCK_DIV4</i>	ADC 时钟预分频选择除以 4
<i>RCU_ADCCCK_DIV6</i>	ADC 时钟预分频选择除以 6
<i>RCU_ADCCCK_DIV8</i>	ADC 时钟预分频选择除以 8
<i>RCU_ADCCCK_DIV10</i>	ADC 时钟预分频选择除以 10
<i>RCU_ADCCCK_DIV12</i>	ADC 时钟预分频选择除以 12
<i>RCU_ADCCCK_DIV16</i>	ADC 时钟预分频选择除以 16
<i>RCU_ADCCCK_DIV32</i>	ADC 时钟预分频选择除以 32
<i>RCU_ADCCCK_DIV64</i>	ADC 时钟预分频选择除以 64
<i>RCU_ADCCCK_DIV128</i>	ADC 时钟预分频选择除以 128
<i>RCU_ADCCCK_DIV256</i>	ADC 时钟预分频选择除以 256
输出参数 {out}	
-	-
返回值	
-	-

Example:

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

```
rcu_adc_clock_config(RCU_ADCSRC_CKSYS, RCU_ADCCCK_DIV8);
```

### 函数 rcu\_adc\_clock\_config（适用于 GD32C211）

函数 rcu\_adc\_clock\_config 描述见下表：

表 3-360. 函数 rcu\_adc\_clock\_config

函数名称	rcu_adc_clock_config
函数原型	void rcu_adc_clock_config(uint32_t adc_clock_source, uint32_t ck_adc)
功能描述	配置 ADC 时钟源和分频器选择
先决条件	-
被调用函数	-
输入参数 {in}	
<b>adc_clock_source</b>	ADC 时钟源选择
<i>RCU_ADCSRC_CKSYS</i>	ADC 时钟源选择 CK_SYS
<i>RCU_ADCSRC_IRC48M_PER</i>	ADC 时钟源选择 CK_IRC48MDIV_PER
<i>RCU_ADCSRC_IRC48M_PER</i>	ADC 时钟源选择 CK_IRC48MDIV_PER
输入参数 {in}	
<b>ck_adc</b>	ADC 时钟分频器选择

<i>RCU_ADCCCK_DIV2</i>	ADC 时钟预分频选择除以 2
<i>RCU_ADCCCK_DIV3</i>	ADC 时钟预分频选择除以 3
<i>RCU_ADCCCK_DIV4</i>	ADC 时钟分频器选择为 4 分频
<i>RCU_ADCCCK_DIV5</i>	ADC 时钟预分频选择除以 5
<i>RCU_ADCCCK_DIV6</i>	ADC 时钟预分频选择为 6 分频
<i>RCU_ADCCCK_DIV7</i>	ADC 时钟预分频器选择除以 7
<i>RCU_ADCCCK_DIV8</i>	ADC 时钟分频器选择除以 8
<i>RCU_ADCCCK_DIV9</i>	ADC 时钟预分频器选择为 9 分频
<i>RCU_ADCCCK_DIV10</i>	ADC 时钟预分频器选择除以 10
<i>RCU_ADCCCK_DIV11</i>	ADC 时钟预分频选择除以 11
<i>RCU_ADCCCK_DIV12</i>	ADC 时钟预分频选择除以 12
<i>RCU_ADCCCK_DIV13</i>	ADC 时钟分频器选择除以 13
<i>RCU_ADCCCK_DIV14</i>	ADC 时钟分频器选择为 14 分频
<i>RCU_ADCCCK_DIV15</i>	ADC 时钟预分频选择除以 15
<i>RCU_ADCCCK_DIV16</i>	ADC 时钟预分频选择除以 16
输出参数 {out}	
-	-
返回值	
-	-

Example:

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

```
rcu_adc_clock_config(RCU_ADCSRC_CKSYS, RCU_ADCCCK_DIV8);
```

### 函数 `rcu_ckout0_config`

函数 `rcu_ckout0_config` 描述见下表:

表 3-361. 函数 `rcu_ckout0_config`

函数名称	<code>rcu_ckout0_config</code>
函数原型	<code>void rcu_ckout0_config(uint32_t ckout0_src, uint32_t ckout0_div)</code>
功能描述	配置 CK_OUT0 时钟源和分频器
先决条件	-
被调用函数	-
输入参数 {in}	
<b>ckout_src</b>	CK_OUT0 时钟源选择
<i>RCU_CKOUT0SRC_NONE</i>	未选择时钟
<i>RCU_CKOUT0SRC_CKSYS</i>	CK_OUT0 时钟源选择 CKSYS
<i>RCU_CKOUT0SRC_IRC48M</i>	CK_OUT0 时钟源选择 IRC48M
<i>RCU_CKOUT0SRC_HXTAL</i>	CK_OUT0 时钟源选择 HXTAL

XTAL	
RCU_CKOUT0SRC_IRC32K	CK_OUT0 时钟源选择 IRC32K
RCU_CKOUT0SRC_LXTAL	CK_OUT0 时钟源选择 LXTAL
输入参数 {in}	
ckout_div	CK_OUT0 分频器
RCU_CKOUT0_DIVx(x = 1, 2, 4, 8, 16, 32, 64, 128)	CK_OUT is divided by x
输出参数 {out}	
-	-
返回值	
-	-

Example:

```
/* 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 3-362. 函数 rcu\_ckout1\_config

函数名称	rcu_ckout1_config
函数原型	void rcu_ckout1_config(uint32_t ckout1_src, uint32_t ckout1_div)
功能描述	配置 CK_OUT1 时钟源和分频器
先决条件	-
被调用函数	-
输入参数 {in}	
ckout_src	CK_OUT1 时钟源选择
RCU_CKOUT1SRC_NONE	未选择时钟
RCU_CKOUT1SRC_CKSYS	CK_OUT1 时钟源选择 CKSYS
RCU_CKOUT1SRC_IRC48M	CK_OUT1 时钟源选择 IRC48M
RCU_CKOUT1SRC_HXTAL	CK_OUT1 时钟源选择 HXTAL
RCU_CKOUT1SRC_IRC32K	CK_OUT1 时钟源选择 IRC32K
RCU_CKOUT1SRC_LXTAL	CK_OUT1 时钟源选择 LXTAL
输入参数 {in}	

<b>ckout_div</b>	CK_OUT 分频器
<i>RCU_CKOUT1_DIVx(x=1,2,4,8,16,32,64,128)</i>	CK_OUT is divided by x
输出参数 {out}	
-	-
返回值	
-	-

Example:

```
/* configure the HXTAL as CK_OUT1 clock source */
rcu_ckout1_config(RCU_CKOUT1SRC_HXTAL, RCU_CKOUT1_DIV1);
```

### 函数 rcu\_lsckout\_enable

函数 rcu\_lsckout\_enable 描述见下表:

表 3-363. 函数 rcu\_lsckout\_enable

函数名称	rcu_lsckout_enable
函数原型	void rcu_lsckout_enable(void)
功能描述	使能低速时钟输出
先决条件	-
被调用函数	-
输入参数 {in}	
-	-
输出参数 {out}	
-	-
返回值	
-	-

Example:

```
/* enable the low speed clock output */
rcu_lsckout_enable ();
```

### 函数 rcu\_lsckout\_disable

函数 rcu\_lsckout\_disable 描述见下表:

表 3-364. 函数 rcu\_lsckout\_disable

函数名称	rcu_lsckout_disable
函数原型	void rcu_lsckout_disable(void)
功能描述	失能低速时钟输出
先决条件	-
被调用函数	-
输入参数 {in}	



-	-
输出参数 {out}	
-	-
返回值	
-	-

Example:

```
/* disable the low speed clock output */
```

```
rcu_lsckout_disable ();
```

### 函数 rcu\_lsckout\_config

函数 rcu\_lsckout\_config 描述见下表:

表 3-365. 函数 rcu\_lsckout\_config

函数名称	rcu_lsckout_config
函数原型	void rcu_lsckout_config(uint32_t lsckout_src)
功能描述	配置 LSCKOUT 时钟源
先决条件	-
被调用函数	-
输入参数 {in}	
lsckout_src	LSCKOUT 时钟源选择
RCU_LSCKOUTSRC_IRC32K	IRC32K 时钟被选中
RCU_LSCKOUTSRC_LXTAL	选择 LXTAL
输出参数 {out}	
-	-
返回值	
-	-

Example:

```
/* configure IRC32K as LSCKOUT clock source */
```

```
rcu_lsckout_config (RCU_LSCKOUTSRC_IRC32K);
```

### 函数 rcu\_usart\_clock\_config

函数 rcu\_usart\_clock\_config 描述见下表:

表 3-366. 函数 rcu\_usart\_clock\_config

函数名称	rcu_usart_clock_config
函数原型	void rcu_usart_clock_config(usart_idx_enum usart_idx, uint32_t ck_usart)
功能描述	配置 USART 时钟源选择
先决条件	-

被调用函数	-
输入参数 {in}	
usart_idx	IDX_USART0
输入参数 {in}	
ck_usart	USART 时钟源选择
RCU_USART0SRC_CK_APB	CK_USART 选择 CK_APB
RCU_USART0SRC_CK_SYS	CK_USART 选择 CK_SYS
RCU_USART0SRC_IRC48MDIV_PER	选择 CK_USART 为 CK_IRC48MDIV_PER
RCU_USART0SRC_LXTAL	CK_USART 选择 LXTAL
输出参数 {out}	
-	-
返回值	
-	-

Example:

```
/* configure the LXTAL as USART0 clock */
```

```
rcu_usart_clock_config(IDX_USART0,RCU_USART0SRC_LXTAL);
```

### 函数 rcu\_i2c\_clock\_config

函数 rcu\_i2c\_clock\_config 描述见下表:

表 3-367. 函数 rcu\_i2c\_clock\_config

函数名称	rcu_i2c_clock_config
函数原型	void rcu_i2c_clock_config(i2c_idx_enum i2c_idx, uint32_t ck_i2c)
功能描述	配置 I2Cx(x=0,1)时钟源选择
先决条件	-
被调用函数	-
输入参数 {in}	
i2c_idx	IDX_I2Cx(x=0,1)
输入参数 {in}	
ck_i2c	I2C 时钟源选择
RCU_I2CSRC_CK_APB	CK_I2C 选择 CK_APB
RCU_I2CSRC_CKSYS	CK_I2C 选择 CK_SYS
RCU_I2CSRC_IRC48MDIV_PER	CK_I2C 选择 CK_IRC48MDIV_PER
输出参数 {out}	
-	-
返回值	

-	-
---	---

Example:

```
/* configure the CKAPB as I2C0 clock */
rcu_usart_clock_config(IDX_I2C0, RCU_I2CSRC_CKAPB);
```

### 函数 rcu\_i2s\_clock\_config

函数 rcu\_i2s\_clock\_config 描述见下表:

表 3-368. 函数 rcu\_i2s\_clock\_config

函数名称	rcu_i2s_clock_config
函数原型	void rcu_i2s_clock_config(uint32_t ck_i2s)
功能描述	配置 I2S 时钟源选择
先决条件	-
被调用函数	-
输入参数 {in}	
ck_i2s	I2S 时钟源选择
RCU_I2SSRC_CKSYS	I2S 时钟选择 CK_SYS
RCU_I2SSRC_IRC48M DIV_PER	CK_I2S 选择 IRC48MDIV_PER
RCU_I2SSRC_CKIN	CK_I2S 选择 I2S_CKIN
输出参数 {out}	
-	-
返回值	
-	-

Example:

```
/* configure the CKSYS as I2S clock */
rcu_i2s_clock_config (RCU_I2SSRC_CKSYS);
```

### 函数 rcu\_irc48mdiv\_sys\_clock\_config

函数 rcu\_irc48mdiv\_sys\_clock\_config 描述见下表:

表 3-369. 函数 rcu\_irc48mdiv\_sys\_clock\_config

函数名称	rcu_irc48mdiv_sys_clock_config
函数原型	void rcu_irc48mdiv_sys_clock_config(uint32_t ck_irc48mdiv_sys)
功能描述	配置 IRC48MDIV_SYS 时钟选择
先决条件	-
被调用函数	-
输入参数 {in}	
ck_irc48mdiv_sys	IRC48MDIV 时钟选择
RCU_IRC48MDIV_SYS	CK_IRC48MDIV_SYS 选择 CK_IRC48M

<code>_1</code>	
<code>RCU_IRC48MDIV_SYS</code> <code>_2</code>	CK_IRC48MDIV_SYS 选择 CK_IRC48M 除以 2
<code>RCU_IRC48MDIV_SYS</code> <code>_4</code>	CK_IRC48MDIV_SYS 选择 CK_IRC48M 除以 4
<code>RCU_IRC48MDIV_SYS</code> <code>_8</code>	CK_IRC48MDIV_SYS 选择 CK_IRC48M 除以 8
<code>RCU_IRC48MDIV_SYS</code> <code>_16</code>	CK_IRC48MDIV_SYS 选择 CK_IRC48M 除以 16
<code>RCU_IRC48MDIV_SYS</code> <code>_32</code>	CK_IRC48MDIV_SYS 选择 CK_IRC48M 除以 32
<code>RCU_IRC48MDIV_SYS</code> <code>_64</code>	CK_IRC48MDIV_SYS 选择 CK_IRC48M 除以 64
<code>RCU_IRC48MDIV_SYS</code> <code>_128</code>	CK_IRC48MDIV_SYS 选择 CK_IRC48M 被 128 分频
输出参数 {out}	
-	-
返回值	
-	-

Example:

```
/* configure the IRC48MDIV_SYS clock selection */
```

```
rcu_irc48mdiv_sys_clock_config (RCU_IRC48MDIV_SYS_2);
```

### 函数 `rcu_irc48mdiv_per_clock_config`

函数 `rcu_irc48mdiv_per_clock_config` 描述见下表:

表 3-370. 函数 `rcu_irc48mdiv_per_clock_config`

函数名称	<code>rcu_irc48mdiv_per_clock_config</code>
函数原型	<code>void rcu_irc48mdiv_per_clock_config(uint32_t ck_irc48mdiv_per)</code>
功能描述	配置 IRC48MDIV 时钟选择
先决条件	-
被调用函数	-
输入参数 {in}	
<code>ck_irc48mdiv_per</code>	IRC48MDIV 时钟选择
<code>RCU_IRC48MDIV_PER</code> <code>_1</code>	CK_IRC48MDIV_PER 选择 CK_IRC48M
<code>RCU_IRC48MDIV_PER</code> <code>_2</code>	选择 CK_IRC48M 除以 2
<code>RCU_IRC48MDIV_PER</code> <code>_3</code>	CK_IRC48MDIV_PER 选择 CK_IRC48M 除以 3
<code>RCU_IRC48MDIV_PER</code>	CK_IRC48MDIV_PER 选择 CK_IRC48M 除以 4

<code>_4</code>	
<code>RCU_IRC48MDIV_PER</code> <code>_5</code>	CK_IRC48MDIV_PER 选择 CK_IRC48M 除以 5
<code>RCU_IRC48MDIV_PER</code> <code>_6</code>	CK_IRC48MDIV_PER 选择 CK_IRC48M 除以 6
<code>RCU_IRC48MDIV_PER</code> <code>_7</code>	CK_IRC48MDIV_PER 选择 CK_IRC48M 除以 7
<code>RCU_IRC48MDIV_PER</code> <code>_8</code>	CK_IRC48MDIV_PER 选择 CK_IRC48M 除以 8
输出参数 {out}	
-	-
返回值	
-	-

Example:

```
/* configure the IRC48MDIV clock selection */
```

```
rcu_irc48mdiv_per_clock_config(RCU_IRC48MDIV_PER_8);
```

### 函数 `rcu_rtc_clock_config`

函数 `rcu_rtc_clock_config` 描述见下表:

表 3-371. 函数 `rcu_rtc_clock_config`

函数名称	<code>rcu_rtc_clock_config</code>
函数原型	<code>void rcu_rtc_clock_config(uint32_t rtc_clock_source)</code>
功能描述	配置 RTC 时钟源选择
先决条件	-
被调用函数	-
输入参数 {in}	
<code>rtc_clock_source</code>	RTC 时钟源选择
<code>RCU_RTCSRC_NONE</code>	未选择时钟
<code>RCU_RTCSRC_LXTAL</code>	CK_LXTAL 选作 RTC 源时钟
<code>RCU_RTCSRC_IRC32</code> <code>K</code>	CK_IRC32K 被选作 RTC 时钟源
<code>RCU_RTCSRC_HXTAL</code> <code>_DIV32</code>	CK_HXTAL/32 选作 RTC 源时钟
输出参数 {out}	
-	-
返回值	
-	-

Example:

```
/* configure the RTC clock source selection */
```

```
rcu_rtc_clock_config(RCU_RTCSRC_IRC32K);
```

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

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

表 3-372. 函数 rcu\_lxtal\_drive\_capability\_config

函数名称	rcu_lxtal_drive_capability_config
函数原型	void rcu_lxtal_drive_capability_config(uint32_t lxtal_dricap)
功能描述	配置 LXTAL 驱动能力
先决条件	-
被调用函数	-
输入参数 {in}	
lxtal_dricap	LXTAL 驱动能力
RCU_LXTAL_LOWDRI	较低驱动能力
RCU_LXTAL_HIGHDRI	更高驱动能力
输出参数 {out}	
-	-
返回值	
-	-

Example:

```
/* set the LXTAL lower driving capability */
```

```
rcu_lxtal_drive_capability_config(RCU_LXTAL_LOWDRI);
```

### 函数 rcu\_osci\_stab\_wait

函数 rcu\_osci\_stab\_wait 描述见下表：

表 3-373. 函数 rcu\_osci\_stab\_wait

函数名称	rcu_osci_stab_wait
函数原型	ErrStatus rcu_osci_stab_wait(rcu_osci_type_enum osci)
功能描述	等待直到振荡器稳定标志被置位
先决条件	-
被调用函数	-
输入参数 {in}	
osci	振荡器类型，参见 <a href="#">表 3-340. 枚举类型 rcu_osci_type_enum</a>
RCU_HXTAL	HXTAL
RCU_LXTAL	LXTAL
RCU_IRC48M	IRC48M
RCU_IRC32K	IRC32K
输出参数 {out}	
-	-
返回值	

ErrStatus	成功或错误
-----------	-------

Example:

```
/* wait for oscillator stabilization flag */

if(SUCCESS == rcu_osc_stab_wait(RCU_HXTAL)){

};
```

### 函数 rcu\_osc\_on

函数 rcu\_osc\_on 描述见下表:

表 3-374. 函数 rcu\_osc\_on

函数名称	rcu_osc_on
函数原型	void rcu_osc_on(rcu_osc_type_enum osci)
功能描述	使能振荡器 (API_ID(0x001FU))
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 3-340. 枚举类型 rcu_osc_type_enum</a>
RCU_HXTAL	HXTAL
RCU_LXTAL	LXTAL
RCU_IRC48M	IRC48M
RCU_IRC32K	IRC32K
输出参数 {out}	
-	-
返回值	
-	-

Example:

```
/* turn on the high speed crystal oscillator */

rcu_osc_on(RCU_HXTAL);
```

### 函数 rcu\_osc\_off

函数 rcu\_osc\_off 描述见下表:

表 3-375. 函数 rcu\_osc\_off

函数名称	rcu_osc_off
函数原型	void rcu_osc_off(rcu_osc_type_enum osci)
功能描述	关闭振荡器
先决条件	-
被调用函数	-
输入参数 {in}	
osci	振荡器类型, 参阅 <a href="#">表 3-340. 枚举类型 rcu_osc_type_enum</a>

<i>RCU_HXTAL</i>	HXTAL
<i>RCU_LXTAL</i>	LXTAL
<i>RCU_IRC48M</i>	IRC48M
<i>RCU_IRC32K</i>	IRC32K
输出参数 {out}	
-	-
返回值	
-	-

Example:

```
/* turn off the high speed crystal oscillator */
```

```
rcu_osc_off(RCU_HXTAL);
```

### 函数 **rcu\_osc\_start\_enable**

函数 **rcu\_osc\_start\_enable** 描述见下表:

表 3-376. 函数 **rcu\_osc\_start\_enable**

函数名称	rcu_osc_start_enable
函数原型	void rcu_osc_start_enable(rcu_osc_type_enum osci)
功能描述	当 osci 准备好时，启动电路使能
先决条件	-
被调用函数	-
输入参数 {in}	
<b>osci</b>	振荡器类型，请参考 <a href="#">表 3-340. 枚举类型 rcu_osc_type_enum</a>
<i>RCU_LXTAL</i>	LXTAL
<i>RCU_IRC32K</i>	IRC32K
输出参数 {out}	
-	-
返回值	
-	-

Example:

```
/* The startup circuit enable when IRC32K is ready */
```

```
rcu_osc_start_enable(RCU_IRC32K);
```

### 函数 **rcu\_osc\_start\_disable**

函数 **rcu\_osc\_start\_disable** 描述见下表:

表 3-377. 函数 **rcu\_osc\_start\_disable**

函数名称	rcu_osc_start_disable
函数原型	void rcu_osc_start_disable(rcu_osc_type_enum osci)
功能描述	启动电路在 osci 准备好时失能



先决条件	-
被调用函数	-
输入参数 {in}	
<b>osci</b>	振荡器类型, 参见 <a href="#">表 3-340. 枚举类型 rcu_osci_type_enum</a>
<i>RCU_LXTAL</i>	LXTAL
<i>RCU_IRC32K</i>	IRC32K
输出参数 {out}	
-	-
返回值	
-	-

Example:

```
/* The startup circuit disable when IRC32K is ready */
rcu_osci_start_disable(RCU_IRC32K);
```

### 函数 rcu\_osci\_bypass\_mode\_enable

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

表 3-378. 函数 rcu\_osci\_bypass\_mode\_enable

函数名称	rcu_osci_bypass_mode_enable
函数原型	void rcu_osci_bypass_mode_enable(rcu_osci_type_enum osci)
功能描述	使能振荡器旁路模式之前, 必须复位 HXTALEN 或 LXTALEN
先决条件	-
被调用函数	-
输入参数 {in}	
<b>osci</b>	振荡器类型, 参见 <a href="#">表 3-340. 枚举类型 rcu_osci_type_enum</a>
<i>RCU_HXTAL</i>	HXTAL
<i>RCU_LXTAL</i>	LXTAL
输出参数 {out}	
-	-
返回值	
-	-

Example:

```
/* 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 3-379. 函数 rcu\_osci\_bypass\_mode\_disable

函数名称	rcu_osci_bypass_mode_disable
------	------------------------------

函数原型	void rcu_osci_bypass_mode_disable(rcu_osci_type_enum osci)
功能描述	失能振荡器旁路模式，需先复位 HXTALEN 或 LXTALEN
先决条件	-
被调用函数	-
输入参数 {in}	
osci	振荡器类型，请参考 <a href="#">表 3-340. 枚举类型 rcu_osci_type_enum</a>
RCU_HXTAL	HXTAL
RCU_LXTAL	LXTAL
输出参数 {out}	
-	-
返回值	
-	-

Example:

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

```
rcu_osci_bypass_mode_disable(RCU_HXTAL);
```

### 函数 rcu\_irc48m\_adjust\_value\_set

函数 rcu\_irc48m\_adjust\_value\_set 描述见下表：

表 3-380. 函数 rcu\_irc48m\_adjust\_value\_set

函数名称	rcu_irc48m_adjust_value_set
函数原型	void rcu_irc48m_adjust_value_set(uint8_t irc48m_adjval)
功能描述	设置 IRC48M 调整值
先决条件	-
被调用函数	-
输入参数 {in}	
irc48m_adjval	IRC48M 调节值，必须介于 0 到 0x3F 之间。
输出参数 {out}	
-	-
返回值	
-	-

Example:

```
/* clear the interrupt flags (API_ID(0x0032U)) */
```

```
rcu_irc48m_adjust_value_set();
```

### 函数 rcu\_lxtal\_stab\_reset\_enable

函数 rcu\_lxtal\_stab\_reset\_enable 描述见下表：

表 3-381. 函数 rcu\_lxtal\_stab\_reset\_enable

函数名称	rcu_lxtal_stab_reset_enable
------	-----------------------------

函数原型	void rcu_lxtal_stab_reset_enable(void)
功能描述	使能 LXTAL 稳定复位
先决条件	-
被调用函数	-
输入参数 {in}	
-	-
输出参数 {out}	
-	-
返回值	
-	-

Example:

```
/* enable LXTAL stabilization reset */
rcu_lxtal_stab_reset_enable();
```

### 函数 rcu\_lxtal\_stab\_reset\_disable

函数 rcu\_lxtal\_stab\_reset\_disable 描述见下表:

表 3-382. 函数 rcu\_lxtal\_stab\_reset\_disable

函数名称	rcu_lxtal_stab_reset_disable
函数原型	void rcu_lxtal_stab_reset_disable(void)
功能描述	失能 LXTAL 稳定复位
先决条件	-
被调用函数	-
输入参数 {in}	
-	-
输出参数 {out}	
-	-
返回值	
-	-

Example:

```
/* disable LXTAL stabilization reset */
rcu_lxtal_stab_reset_disable();
```

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

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

表 3-383. 函数 rcu\_hxtal\_clock\_monitor\_enable

函数名称	rcu_hxtal_clock_monitor_enable
函数原型	void rcu_hxtal_clock_monitor_enable(void)
功能描述	使能 HXTAL 时钟监视器

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

Example:

```
/* enable the HXTAL clock monitor */
rcu_hxtal_clock_monitor_enable();
```

### 函数 rcu\_hxtal\_clock\_monitor\_disable

函数 rcu\_hxtal\_clock\_monitor\_disable 描述见下表:

表 3-384. 函数 rcu\_hxtal\_clock\_monitor\_disable

函数名称	rcu_hxtal_clock_monitor_disable
函数原型	void rcu_hxtal_clock_monitor_disable(void)
功能描述	失能 HXTAL 时钟监测器
先决条件	-
被调用函数	-
输入参数 {in}	
-	-
输出参数 {out}	
-	-
返回值	
-	-

Example:

```
/* disable the HXTAL clock monitor */
rcu_hxtal_clock_monitor_disable();
```

### 函数 rcu\_lxtal\_clock\_monitor\_enable

函数 rcu\_lxtal\_clock\_monitor\_enable 描述见下表:

表 3-385. 函数 rcu\_lxtal\_clock\_monitor\_enable

函数名称	rcu_lxtal_clock_monitor_enable
函数原型	void rcu_lxtal_clock_monitor_enable(void)
功能描述	使能 LXTAL 时钟监测器
先决条件	-
被调用函数	-

输入参数 {in}	
-	-
输出参数 {out}	
-	-
返回值	
-	-

Example:

```
/* enable the LXTAL clock monitor */
```

```
rcu_lxtal_clock_monitor_enable();
```

### 函数 rcu\_lxtal\_clock\_monitor\_disable

函数 rcu\_lxtal\_clock\_monitor\_disable 描述见下表:

表 3-386. 函数 rcu\_lxtal\_clock\_monitor\_disable

函数名称	rcu_lxtal_clock_monitor_disable
函数原型	void rcu_lxtal_clock_monitor_disable(void)
功能描述	失能 LXTAL 时钟监测器
先决条件	-
被调用函数	-
输入参数 {in}	
-	-
输出参数 {out}	
-	-
返回值	
-	-

Example:

```
/* disable the LXTAL clock monitor */
```

```
rcu_lxtal_clock_monitor_disable();
```

### 函数 rcu\_clock\_freq\_get

函数 rcu\_clock\_freq\_get 描述见下表:

表 3-387. 函数 rcu\_clock\_freq\_get

函数名称	rcu_clock_freq_get
函数原型	uint32_t rcu_clock_freq_get(rcu_clock_freq_enum clock)
功能描述	获取系统时钟、总线和外设时钟频率
先决条件	-
被调用函数	-
输入参数 {in}	
clock	获取的时钟频率

CK_SYS	系统时钟频率
CK_AHB	AHB 时钟频率
CK_APB	APB 时钟频率
CK_ADC	ADC 时钟频率
CK_USART0	USART0 时钟频率
输出参数 {out}	
-	-
返回值	
clock frequency of system, AHB, APB, ADC or USART0	系统, AHB, APB, ADC 或者 USART0 时钟

Example:

```
uint32_t temp_freq;

/* get the system clock frequency */

temp_freq = rcu_clock_freq_get(CK_SYS);
```

### 函数 rcu\_flag\_get

函数 rcu\_flag\_get 描述见下表:

表 3-388. 函数 rcu\_flag\_get

函数名称	rcu_flag_get
函数原型	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	时钟稳定性和外设复位标志, 请参考 <a href="#">表 3-336. 枚举类型 rcu_flag_enum</a>
RCU_FLAG_IRC32KST B	IRC32K 稳定标志
RCU_FLAG_LXTALST B	LXTAL 稳定标志
RCU_FLAG_IRC48MS TB	IRC48M 稳定标志
RCU_FLAG_HXTALST B	HXTAL 稳定标志
RCU_FLAG_OBLRST	选项字节加载器复位标志
RCU_FLAG_BORRST	BOR 复位标志
RCU_FLAG_EPRST	外部引脚复位标志
RCU_FLAG_PORRST	电源复位标志
RCU_FLAG_SWRST	软件复位标志
RCU_FLAG_FWDGTR	FWDGT 复位标志

ST	
RCU_FLAG_WWDGTR	WWDGT 复位标志
ST	
RCU_FLAG_LPRST	低功耗复位标志
RCU_FLAG_LCKCMD	LXTAL 时钟故障检测标志
输出参数 {out}	
-	-
返回值	
FlagStatus	设置或复位

Example:

```
/* 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 3-389. 函数 rcu\_all\_reset\_flag\_clear

函数名称	rcu_all_reset_flag_clear
函数原型	void rcu_all_reset_flag_clear(void)
功能描述	清除复位标志
先决条件	-
被调用函数	-
输入参数 {in}	
-	-
输出参数 {out}	
-	-
返回值	
-	-

Example:

```
/* clear all the reset flag */
rcu_all_reset_flag_clear();
```

### 函数 rcu\_interrupt\_enable

函数 rcu\_interrupt\_enable 描述见下表:

表 3-390. 函数 rcu\_interrupt\_enable

函数名称	rcu_interrupt_enable
函数原型	void rcu_interrupt_enable(rcu_int_enum stab_int)
功能描述	使能稳定中断

先决条件	-
被调用函数	-
输入参数 {in}	
<b>stab_int</b>	时钟稳定中断, 请参阅 <a href="#">表 3-339. 枚举类型 rcu_int_enum</a>
<i>RCU_INT_IRC32KSTB</i>	IRC32K 稳态中断使能
<i>RCU_INT_LXTALSTB</i>	LXTAL 稳定中断使能
<i>RCU_INT_IRC48MSTB</i>	IRC48M 稳定中断使能
<i>RCU_INT_HXTALSTB</i>	HXTAL 稳定中断使能
输出参数 {out}	
-	-
返回值	
-	-

Example:

```
/* 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 3-391. 函数 rcu\_interrupt\_disable

函数名称	rcu_interrupt_disable
函数原型	void rcu_interrupt_disable(rcu_int_enum stab_int)
功能描述	失能稳定中断
先决条件	-
被调用函数	-
输入参数 {in}	
<b>stab_int</b>	时钟稳定中断, 请参考 <a href="#">表 3-339. 枚举类型 rcu_int_enum</a>
<i>RCU_INT_IRC32KSTB</i>	IRC32K 稳定中断失能
<i>RCU_INT_LXTALSTB</i>	LXTAL 稳定中断失能
<i>RCU_INT_IRC48MSTB</i>	IRC48M 稳定中断失能
<i>RCU_INT_HXTALSTB</i>	HXTAL 稳定中断失能
输出参数 {out}	
-	-
返回值	
-	-

Example:

```
/* disable the HXTAL stabilization interrupt */
rcu_interrupt_disable(RCU_INT_HXTALSTB);
```



**函数 rcu\_interrupt\_flag\_get**

函数 rcu\_interrupt\_flag\_get 描述见下表:

**表 3-392. 函数 rcu\_interrupt\_flag\_get**

函数名称	rcu_interrupt_flag_get
函数原型	FlagStatus rcu_interrupt_flag_get(rcu_int_flag_enum int_flag)
功能描述	获取时钟稳定中断和 CKM 标志
先决条件	-
被调用函数	-
输入参数 {in}	
int_flag	中断和 ckm 标志, 请参阅 <a href="#">表 3-337. 枚举类型 rcu_int_flag_enum</a>
RCU_INT_FLAG_IRC3 2KSTB	IRC32K 稳定中断标志
RCU_INT_FLAG_LXTA LSTB	LXTAL 稳定中断标志
RCU_INT_FLAG_IRC4 8MSTB	IRC48M 稳定中断标志
RCU_INT_FLAG_HXTA LSTB	HXTAL 稳定中断标志
RCU_INT_FLAG_LXTA LCKM	LXTAL 时钟卡死中断标志
RCU_INT_FLAG_CKM	HXTAL 时钟卡死中断标志
输出参数 {out}	
-	-
返回值	
FlagStatus	设置或复位

Example:

```
/* 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 3-393. 函数 rcu\_interrupt\_flag\_clear**

函数名称	rcu_interrupt_flag_clear
函数原型	void rcu_interrupt_flag_clear(rcu_int_flag_clear_enum int_flag_clear)
功能描述	清除中断标志
先决条件	-
被调用函数	-
输入参数 {in}	

<b>int_flag_clear</b>	时钟稳定及卡死中断标志清除，请参考 <a href="#">表 3-338. 枚举类型</a> <a href="#">rcu_int_flag_clear_enum</a>
<i>RCU_INT_FLAG_IRC3 2KSTB_CLR</i>	IRC32K 稳定中断标志清除
<i>RCU_INT_FLAG_LXTA LSTB_CLR</i>	LXTAL 稳定中断标志清除
<i>RCU_INT_FLAG_IRC4 8MSTB_CLR</i>	IRC48M 稳定中断标志清除
<i>RCU_INT_FLAG_HXTA LSTB_CLR</i>	HXTAL 稳定中断标志清除
<i>RCU_INT_FLAG_LXTA LCKM_CLR</i>	LXTAL 时钟卡死中断标志清除
<i>RCU_INT_FLAG_CKM _CLR</i>	时钟卡死中断标志清除
输出参数 {out}	
-	-
返回值	
-	-

Example:

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

## 3.15. RTC

实时时钟RTC通常被用作时钟日历。位于备份域中的RTC电路，包含一个32位的累加计数器、一个闹钟、一个预分频器、一个分频器以及RTC时钟配置寄存器。章节[3.15.1](#)描述了RTC的寄存器列表，章节[3.15.2](#)对RTC库函数进行说明。

### 3.15.1. 外设寄存器描述

RTC寄存器列表如下表所示：

表 3-394. RTC 寄存器

寄存器名称	寄存器描述
RTC_TIME	RTC时间寄存器
RTC_DATE	RTC日期寄存器
RTC_CTL	RTC控制寄存器
RTC_STAT	RTC状态寄存器
RTC_PSC	RTC预分频寄存器
RTC_ALRM0TD	RTC闹钟0时间日期寄存器

寄存器名称	寄存器描述
RTC_WPK	RTC写保护钥匙寄存器
RTC_SS	RTC亚秒寄存器
RTC_SHIFTCTL	RTC移位控制寄存器
RTC_TTS	RTC时间戳时间寄存器
RTC_DTS	RTC时间戳日期寄存器
RTC_SSTS	RTC时间戳亚秒寄存器
RTC_HRFC	RTC高精度频率补偿寄存器
RTC_TYPE	RTC类型寄存器
RTC_ALRM0SS	RTC闹钟0亚秒寄存器
RTC_BKP0	RTC备份域寄存器0
RTC_BKP1	RTC备份域寄存器1
RTC_BKP2	RTC备份域寄存器2
RTC_BKP3	RTC备份域寄存器3

### 3.15.2. 外设库函数描述

RTC库函数列表如下表所示：

**表 3-395. RTC 库函数**

库函数名称	库函数描述
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_register_sync_wait	等待直到RTC_TIME和RTC_DATE寄存器与APB时钟同步，并且阴影寄存器被更新
rtc_alarm_enable	使能RTC 闹钟
rtc_alarm_disable	失能RTC 闹钟
rtc_alarm_config	配置RTC闹钟
rtc_alarm_subsecond_config	配置RTC闹钟的亚秒值
rtc_alarm_get	获取RTC闹钟
rtc_alarm_subsecond_get	获取RTC闹钟亚秒值
rtc_init_mode_enter	进入RTC初始化模式
rtc_init	初始化RTC寄存器
rtc_init_mode_exit	退出RTC初始化模式
rtc_current_time_get	获取当前的时间和日期
rtc_subsecond_get	获取当前的亚秒值
rtc_deinit	复位大多数RTC寄存器
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_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_smooth_calibration_config	配置RTC平滑校准

库函数名称	库函数描述
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时间戳亚秒值
rtc_calibration_output_config	配置RTC校准输出
rtc_alarm_output_config	配置RTC闹钟输出
rtc_output_pin_select	配置RTC输出引脚
rtc_interrupt_enable	使能RTC指定的中断
rtc_interrupt_disable	失能RTC指定中断
rtc_flag_get	获取指定中断标志位
rtc_flag_clear	清除指定中断标志位

### 结构体 rtc\_parameter\_struct

表 3-396. 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（BCD格式）
date	RTC日期值：0x1 - 0x31（BCD格式）
day_of_week	RTC星期值（BCD格式）
hour	RTC 小时值：0x1 - 0x12（BCD格式）or 0x0 - 0x23（BCD格式）
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 3-397. 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（BCD格式）
alarm_hour	RTC闹钟小时值：0x1 - 0x12（BCD格式）或0x0 - 0x23（BCD格式）
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 3-398. 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（BCD格式）
timestamp_hour	RTC 时间戳小时值：0x1 - 0x12（BCD格式）或0x0 - 0x23（BCD格式）
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（BCD格式）
am_pm	RTC时间戳AM/PM数值

## 函数 rtc\_bypass\_shadow\_enable

函数rtc\_bypass\_shadow\_enable描述见下表：

表 3-399. 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 3-400. 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\_register\_sync\_wait

函数rtc\_register\_sync\_wait描述见下表：

表 3-401. 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	等待直到RTC_TIME和RTC_DATE寄存器与APB时钟同步，并且阴影寄存器被更新
先决条件	-
被调用函数	-
输入参数{in}	
-	-
输入参数{in}	
-	-
返回值	
ErrStatus	ERROR或SUCCESS

例如：

```
/*wait until RTC_TIME and RTC_DATE registers are synchronized with APB clock, and the shadow registers are updated*/
```

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

### 函数 rtc\_alarm\_enable

函数rtc\_alarm\_enable描述见下表：

表 3-402. 函数 rtc\_alarm\_enable

函数名称	rtc_alarm_enable
函数原型	void rtc_alarm_enable();
功能描述	使能RTC闹钟
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 RTC alarm*/
```

```
rtc_alarm_enable();
```

### 函数 rtc\_alarm\_disable

函数rtc\_alarm\_disable描述见下表：

表 3-403. 函数 rtc\_alarm\_disable

函数名称	rtc_alarm_disable
函数原型	ErrStatus rtc_alarm_disable();
功能描述	失能RTC闹钟
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 alarm*/
```

```
ErrStatus error_status = rtc_alarm_disable();
```

### 函数 rtc\_alarm\_config

函数rtc\_alarm\_config描述见下表：

表 3-404. 函数 rtc\_alarm\_config

函数名称	rtc_alarm_config
函数原型	void rtc_alarm_config(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}	

<b>rtc_alarm_time</b>	闹钟结构体，结构体成员参考 <a href="#">表3-397. 结构体rtc_alarm_struct</a>
<b>输出参数{out}</b>	
-	-
<b>返回值</b>	
-	-

例如：

```
/*rtc_alarm_config*/
```

```
rtc_alarm_struct rtc_alarm_time;
```

```
rtc_alarm_config(&rtc_alarm_time);
```

### 函数 rtc\_alarm\_subsecond\_config

函数rtc\_alarm\_subsecond\_config描述见下表：

**表 3-405. 函数 rtc\_alarm\_subsecond\_config**

<b>函数名称</b>	rtc_alarm_subsecond_config
<b>函数原型</b>	void rtc_alarm_subsecond_config(uint32_t mask_subsecond, uint32_t subsecond)
<b>功能描述</b>	配置RTC闹钟的亚秒值
<b>先决条件</b>	-
<b>被调用函数</b>	-
<b>输入参数{in}</b>	
<b>mask_subsecond</b>	闹钟亚秒屏蔽位
<i>RTC_MASKSSC_0_14</i>	屏蔽闹钟亚秒设置
<i>RTC_MASKSSC_1_14</i>	屏蔽RTC_ALRM0SS_SSC[14:1]，SSC[0]位用于时间匹配
<i>RTC_MASKSSC_2_14</i>	屏蔽RTC_ALRM0SS_SSC[14:2]，SSC[1:0]位用于时间匹配
<i>RTC_MASKSSC_3_14</i>	屏蔽RTC_ALRM0SS_SSC[14:3]，SSC[2:0]位用于时间匹配
<i>RTC_MASKSSC_4_14</i>	屏蔽RTC_ALRM0SS_SSC[14:4]，SSC[3:0]位用于时间匹配
<i>RTC_MASKSSC_5_14</i>	屏蔽RTC_ALRM0SS_SSC[14:5]，SSC[4:0]位用于时间匹配
<i>RTC_MASKSSC_6_14</i>	屏蔽RTC_ALRM0SS_SSC[14:6]，SSC[5:0]位用于时间匹配
<i>RTC_MASKSSC_7_14</i>	屏蔽RTC_ALRM0SS_SSC[14:7]，SSC[6:0]位用于时间匹配
<i>RTC_MASKSSC_8_14</i>	屏蔽RTC_ALRM0SS_SSC[14:8]，SSC[7:0]位用于时间匹配
<i>RTC_MASKSSC_9_14</i>	屏蔽RTC_ALRM0SS_SSC[14:9]，SSC[8:0]位用于时间匹配
<i>RTC_MASKSSC_10_14</i>	屏蔽RTC_ALRM0SS_SSC[14:10]，SSC[9:0]位用于时间匹配
<i>RTC_MASKSSC_11_14</i>	屏蔽RTC_ALRM0SS_SSC[14:11]，SSC[10:0]位用于时间匹配
<i>RTC_MASKSSC_12_14</i>	屏蔽RTC_ALRM0SS_SSC[14:12]，SSC[11:0]位用于时间匹配
<i>RTC_MASKSSC_13_14</i>	屏蔽RTC_ALRM0SS_SSC[14:13]，SSC[12:0]位用于时间匹配



<i>RTC_MASKSSC_14</i>	屏蔽RTC_ALRM0SS_SSC[14], SSC[13:0]位用于时间匹配
<i>RTC_MASKSSC_NON E</i>	无屏蔽, SSC[14:0]位用于时间匹配
输入参数{in}	
<b>subsecond</b>	闹钟亚秒值(0x000 - 0x7FFF)
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/*configure subsecond of RTC alarm*/
```

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

### 函数 **rtc\_alarm\_subsecond\_get**

函数rtc\_alarm\_subsecond\_get描述见下表:

表 3-406. 函数 **rtc\_alarm\_subsecond\_get**

函数名称	rtc_alarm_subsecond_get
函数原型	uint32_t rtc_alarm_subsecond_get();
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}	
-	-
输出参数{out}	
-	-
返回值	
<b>uint32_t</b>	RTC 闹钟亚秒值(0x0-0x3FFF)

例如:

```
/*get RTC alarm subsecond*/
```

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

### 函数 **rtc\_alarm\_get**

函数rtc\_alarm\_get描述见下表:

表 3-407. 函数 **rtc\_alarm\_get**

函数名称	rtc_alarm_get
函数原型	void rtc_alarm_get(rtc_alarm_struct* rtc_alarm_time);
功能描述	获取RTC闹钟

先决条件	-
被调用函数	-
输入参数{in}	
-	-
输出参数{out}	
rtc_alarm_time	闹钟结构体，结构体成员参考 <a href="#">表3-397. 结构体rtc_alarm_struct</a>
返回值	
-	-

例如：

```
/*disable RTC alarm*/

rtc_alarm_struct rtc_alarm_time;

rtc_alarm_get(&rtc_alarm_time);
```

### 函数 rtc\_init\_mode\_enter

函数rtc\_init\_mode\_enter描述见下表：

表 3-408. 函数 rtc\_init\_mode\_enter

函数名称	rtc_init_mode_enter
函数原型	ErrStatus rtc_init_mode_enter(void);
功能描述	进入RTC初始化模式
先决条件	-
被调用函数	-
输入参数{in}	
-	-
输出参数{out}	
-	-
返回值	
ErrStatus	ERROR或SUCCESS

例如：

```
/*enter RTC init mode*/

ErrStatus error_status = rtc_init_mode_enter();
```

### 函数 rtc\_init

函数rtc\_init描述见下表：

表 3-409. 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3-396. 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\_exit

函数rtc\_init\_mode\_exit描述见下表:

表 3-410. 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_current_time_get`

函数`rtc_current_time_get`描述见下表:

**表 3-411. 函数 `rtc_current_time_get`**

函数名称	<code>rtc_current_time_get</code>
函数原型	<code>void rtc_current_time_get(rtc_parameter_struct* rtc_initpara_struct);</code>
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}	
-	-
输出参数{out}	
<code>rtc_initpara_struct</code>	初始化结构体, 结构体成员参考 <a href="#">表3-396. 结构体<code>rtc_parameter_struct</code></a>
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 3-412. 函数 `rtc_subsecond_get`**

函数名称	<code>rtc_subsecond_get</code>
函数原型	<code>uint32_t rtc_subsecond_get(void);</code>
功能描述	获取当前的亚秒值
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\_deinit

函数rtc\_deinit描述见下表:

表 3-413. 函数 rtc\_deinit

函数名称	rtc_deinit
函数原型	ErrStatus rtc_deinit(void);
功能描述	复位大多数RTC寄存器
先决条件	-
被调用函数	rcu_periph_reset_enable/ rcu_periph_reset_disable
输入参数{in}	
-	-
输出参数{out}	
-	-
返回值	
ErrStatus	ERROR或SUCCESS

例如:

```
/* reset most of the RTC registers*/
```

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

### 函数 rtc\_hour\_adjust

函数rtc\_hour\_adjust描述见下表:

表 3-414. 函数 rtc\_hour\_adjust

函数名称	rtc_hour_adjust
函数原型	void rtc_hour_adjust(uint32_t operation);
功能描述	通过在当前时间上增加或者减少一个小时来适应夏令时和冬令时
先决条件	-
被调用函数	-
输入参数{in}	
operation	小时调整操作
RTC_CTL_A1H	增加一个小时
RTC_CTL_S1H	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 3-415. 函数 rtc\_second\_adjust

函数名称	rtc_second_adjust
函数原型	ErrStatus rtc_second_adjust(uint32_t add, uint32_t minus);
功能描述	调整RTC当前时间的秒或亚秒值
先决条件	-
被调用函数	-
输入参数{in}	
add	在当前时间上增加1S或者不增加
RTC_SHIFT_ADD1S_RESET	无影响
RTC_SHIFT_ADD1S_SET	在当前时间增加1秒
输入参数{in}	
minus	在当前是时间上减少的亚秒值(0x0 - 0x7FFF)
输出参数{out}	
-	-
返回值	
ErrStatus	ERROR或 SUCCESS

例如：

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

```
ErrStatus error_status = rtc_second_adjust(RTC_SHIFT_ADD1S_SET, 0);
```

### 函数 rtc\_refclock\_detection\_enable

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

表 3-416. 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 = rtc_refclock_detection_enable();
```

### 函数 rtc\_refclock\_detection\_disable

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

表 3-417. 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 or SUCCESS

例如：

```
/* disableRTC reference clock detection function*/
```

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

### 函数 rtc\_smooth\_calibration\_config

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

表 3-418. 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
<i>RTC_CALIBRATION_WINDOW_32S</i>	采用32S校准周期
<i>RTC_CALIBRATION_WINDOW_16S</i>	采用16S校准周期
<i>RTC_CALIBRATION_WINDOW_8S</i>	采用8S校准周期
输入参数{in}	
<b>plus</b>	增加脉冲
<i>RTC_CALIBRATION_PLUS_SET</i>	每2048个脉冲增加一个RTCCLK脉冲
<i>RTC_CALIBRATION_PLUS_RESET</i>	无影响
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\_timestamp\_enable**

函数rtc\_timestamp\_enable描述见下表：

**表 3-419. 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}	
<b>edge</b>	选定哪种边沿触发时间戳检测
<i>RTC_TIMESTAMP_RISING_EDGE</i>	上升沿是时间戳事件有效检测沿



<i>RTC_TIMESTAMP_FALLING_EDGE</i>	下降沿是时间戳事件有效检测沿
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 3-420. 函数 `rtc_timestamp_disable`

函数名称	<code>rtc_timestamp_disable</code>
函数原型	<code>void rtc_timestamp_disable(void);</code>
功能描述	失能RTC时间戳
先决条件	-
被调用函数	-
输入参数{in}	
-	-
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/*disable RTC time-stamp*/
```

```
rtc_timestamp_disable();
```

### 函数 `rtc_timestamp_get`

函数`rtc_timestamp_get`描述见下表：

表 3-421. 函数 `rtc_timestamp_get`

函数名称	<code>rtc_timestamp_get</code>
函数原型	<code>void rtc_timestamp_get(rtc_timestamp_struct* rtc_timestamp);</code>
功能描述	获取RTC时间戳时间和日期
先决条件	-
被调用函数	-
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3-398. 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 3-422. 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\_calibration\_output\_config

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

**表 3-423. 函数 rtc\_calibration\_output\_config**

函数名称	rtc_calibration_output_config
函数原型	void rtc_calibration_output_config(uint32_t source);
功能描述	RTC校准输出配置
先决条件	-
被调用函数	-
输入参数{in}	

source	配置输出信号
<i>RTC_CALIBRATION_5</i> <i>12HZ</i>	当LSE频率为32768Hz并且RTC_PSC 为默认值, 输出512Hz信号
<i>RTC_CALIBRATION_1</i> <i>HZ</i>	当LSE频率为32768Hz并且RTC_PSC 为默认值, 输出1Hz信号
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* when the LSE frequency is 32768Hz and the RTC_PSC
```

```
is the default value, output 1Hz signal */
```

```
rtc_calibration_output_config(RTC_CALIBRATION_1HZ);
```

### 函数 rtc\_alarm\_output\_config

函数rtc\_alarm\_output\_config描述见下表:

表 3-424. 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}	
source	specify signal to output
<i>RTC_ALARM0_HIGH</i>	当alarm0标志置位, 输出引脚为高电平
<i>RTC_ALARM0_LOW</i>	当alarm0标志置位, 输出引脚为低电平
输入参数{in}	
mode	当输出闹钟信号或者唤醒信号时指定输出引脚的模式
<i>RTC_ALARM_OUTPU</i> <i>T_OD</i>	开漏输出
<i>RTC_ALARM_OUTPU</i> <i>T_PP</i>	推挽输出
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* configure rtc alarm0 output source */
```

```
rtc_alarm_output_config(RTC_ALARM0_LOW, RTC_ALARM_OUTPUT_PP);
```

### 函数 rtc\_output\_pin\_select

函数rtc\_output\_pin\_select描述见下表:

表 3-425. 函数 rtc\_output\_pin\_select

函数名称	rtc_output_pin_select
函数原型	void rtc_output_pin_select(uint32_t outputpin);
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}	
outputpin	选择RTC输出引脚
RTC_OUT_PC13	RTC输出引脚为PC13
RTC_OUT_PB2_PB14	RTC输出引脚为PB2 或者PB14
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* specify the rtc output pin is PC13 */
```

```
rtc_output_pin_select(RTC_OUT_PC13);
```

### 函数 rtc\_interrupt\_enable

函数rtc\_interrupt\_enable描述见下表:

表 3-426. 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
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 3-427. 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}	
<b>interrupt</b>	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
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* disble RTC ALARM interrupt */
```

```
rtc_interrupt_disable(RTC_INT_ALARM0);
```

### 函数 `rtc_flag_get`

函数`rtc_flag_get`描述见下表：

**表 3-428. 函数 `rtc_flag_get`**

函数名称	<code>rtc_flag_get</code>
函数原型	<code>FlagStatus rtc_flag_get(uint32_t flag);</code>
功能描述	获取指定中断标志位
先决条件	-
被调用函数	-
输入参数{in}	
<b>flag</b>	选定被获取的中断标志
<code>RTC_FLAG_SCP</code>	平滑校准挂起标志
<code>RTC_FLAG_TSOVR</code>	时间戳事件溢出标志
<code>RTC_FLAG_TS</code>	时间戳事件标志
<code>RTC_FLAG_ALARM0</code>	Alarm0发生标志

<i>RTC_FLAG_INIT</i>	进入初始化模式
<i>RTC_FLAG_RSYN</i>	寄存器同步标志
<i>RTC_FLAG_YCM</i>	年份配置标志
<i>RTC_FLAG_SOP</i>	移位功能操作挂起标志
<i>RTC_FLAG_ALARM0</i> <i>W</i>	Alarm0配置可写标志
输出参数{out}	
-	-
返回值	
FlagStatus	SET 或 RESET

例如:

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

```
FlagStatus = rtc_flag_get(RTC_FLAG_TS);
```

### 函数 `rtc_flag_clear`

函数`rtc_flag_clear`描述见下表:

表 3-429. 函数 `rtc_flag_clear`

函数名称	<code>rtc_flag_clear</code>
函数原型	<code>void rtc_flag_clear(uint32_t flag);</code>
功能描述	清除指定中断标志位
先决条件	-
被调用函数	-
输入参数{in}	
<b>flag</b>	要清除的中断标志位
<i>RTC_FLAG_TSOVR</i>	时间戳事件溢出标志
<i>RTC_FLAG_TS</i>	时间戳事件标志
<i>RTC_FLAG_ALARM0</i>	Alarm0发生标志
<i>RTC_FLAG_RSYN</i>	寄存器同步标志
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* cleartime-stamp event flag */
```

```
rtc_flag_clear(RTC_FLAG_TS);
```

## 3.16. SPI

SPI/I2S模块可以通过SPI协议或I2S音频协议与外部设备进行通信。章节[3.16.1](#)描述了SPI/I2S的寄存器列表，章节[3.16.2](#)对SPI/I2S库函数进行说明。

### 3.16.1. 外设寄存器说明

SPI/I2S寄存器列表如下表所示：

**表 3-430. SPI/I2S 寄存器**

寄存器名称	寄存器描述
SPI_CTL0	控制寄存器0
SPI_CTL1	控制寄存器1
SPI_STAT	状态寄存器
SPI_DATA	数据寄存器
SPI_CRCPOLY	CRC多项式寄存器
SPI_RCRC	接收CRC寄存器
SPI_TCRC	发送CRC寄存器
SPI_I2SCTL	I2S控制寄存器
SPI_I2SPSC	I2S时钟预分频寄存器
SPI_QCTL	四线SPI控制寄存器

### 3.16.2. 外设库函数说明

SPI/I2S库函数列表如下表所示：

**表 3-431. SPI/I2S 库函数**

库函数名称	库函数描述
spi_i2s_deinit	复位外设SPI/I2S
spi_struct_para_init	初始化SPI结构体中所有参数为默认值
spi_init	初始化外设SPI
spi_enable	使能外设SPI
spi_disable	失能外设SPI
i2s_init	初始化外设I2S
i2s_psc_config	配置I2S预分频器
i2s_enable	使能外设I2S
i2s_disable	失能外设I2S
spi_nss_output_enable	使能外设SPI NSS输出
spi_nss_output_disable	失能外设SPI NSS输出
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

库函数名称	库函数描述
spi_transmit_odd_config	配置SPI通过DMA发送的数据总数是否为奇数
spi_receive_odd_config	配置SPI通过DMA接收的数据总数是否为奇数
spi_i2s_data_frame_format_config	配置外设SPI/I2S数据帧格式
spi_fifo_access_size_config	配置SPI访问FIFO的大小（8位或16位）
spi_bidirectional_transfer_config	配置外设SPI的数据传输方向
spi_i2s_data_transmit	发送数据
spi_i2s_data_receive	接收数据
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_length_set	设置CRC长度
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_nssp_mode_enable	使能SPI NSS脉冲模式
spi_nssp_mode_disable	禁能SPI NSS脉冲模式
spi_quad_enable	使能四线SPI模式
spi_quad_disable	禁能四线SPI模式
spi_quad_write_enable	使能四线SPI写
spi_quad_read_enable	使能四线SPI读
spi_quad_io23_output_enable	使能SPI_IO2和SPI_IO3输出
spi_quad_io23_output_disable	禁能SPI_IO2和SPI_IO3输出
spi_i2s_format_error_clear	清除SPI/I2S格式错误标志
spi_i2s_flag_get	获取外设SPI/I2S标志状态
spi_i2s_interrupt_enable	使能外设SPI/I2S中断
spi_i2s_interrupt_disable	失能外设SPI/I2S中断
spi_i2s_interrupt_flag_get	获取外设SPI/I2S中断状态

### 结构体 spi\_parameter\_struct

表 3-432. 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=4,5,..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\_i2s\_deinit

函数spi\_i2s\_deinit描述见下表:

表 3-433. 函数 spi\_i2s\_deinit

函数名称	spi_i2s_deinit
函数原形	void spi_i2s_deinit(uint32_t spi_periph);
功能描述	复位外设SPI/I2S
先决条件	-
被调用函数	rcu_periph_reset_enable / rcu_periph_reset_disable
输入参数{in}	
spi_periph	外设SPIx
SPIx (x=0,1)	SPI外设选择
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* reset SPI0 */
```

```
spi_i2s_deinit(SPI0);
```

### 函数 spi\_struct\_para\_init

函数spi\_struct\_para\_init描述见下表:

表 3-434. 函数 spi\_struct\_para\_init

函数名称	spi_struct_para_init
函数原形	void spi_struct_para_init(spi_parameter_struct* spi_struct);
功能描述	初始化SPI结构体中所有参数为默认值
先决条件	-
被调用函数	-
输入参数{in}	
spi_struct	SPI初始化结构体, 结构体成员参考 <a href="#">表3-432. 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 3-435. 函数 spi\_init**

函数名称	spi_init
函数原形	ErrStatus spi_init(uint32_t spi_periph, spi_parameter_struct* spi_struct);
功能描述	初始化外设SPI
先决条件	-
被调用函数	-
输入参数{in}	
spi_periph	外设SPI
SPIx (x=0,1)	SPI外设选择
输入参数{in}	
spi_struct	初始化结构体，结构体成员参考 <a href="#">表3-432. 结构体spi_parameter_struct</a>
输出参数{out}	
-	-
返回值	
ErrStatus	ERROR或者SUCCESS

例如：

```
/* initialize SPI0 */
spi_parameter_struct spi_init_struct;
ErrStatus errstatus = ERROR;
spi_init_struct.trans_mode      = SPI_TRANSMODE_BDTRANSMIT;
spi_init_struct.device_mode     = SPI_MASTER;
spi_init_struct.frame_size      = SPI_FRAME_SIZE_8BIT;
spi_init_struct.clock_polarity_phase = SPI_CK_PL_HIGH_PH_2EDGE;
spi_init_struct.nss              = SPI_NSS_SOFT;
```

```

spi_init_struct.prescale      = SPI_PSC_8;

spi_init_struct.endian        = SPI_ENDIAN_MSB;

errorstatus = spi_init(SPI0, &spi_init_struct);

```

### 函数 spi\_enable

函数spi\_enable描述见下表：

表 3-436. 函数 spi\_enable

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

例如：

```

/* enable SPI0 */

spi_enable(SPI0);

```

### 函数 spi\_disable

函数spi\_disable描述见下表：

表 3-437. 函数 spi\_disable

函数名称	spi_disable
函数原形	void spi_disable(uint32_t spi_periph);
功能描述	失能外设SPI
先决条件	-
被调用函数	-
输入参数{in}	
spi_periph	外设SPI
SPIx (x=0,1)	SPI外设选择
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* disable SPI0 */
```

```
spl_disable(SPI0);
```

### 函数 i2s\_init

函数i2s\_init描述见下表：

表 3-438. 函数 i2s\_init

函数名称	i2s_init
函数原形	void i2s_init(uint32_t spi_periph, uint32_t mode, uint32_t standard, uint32_t ckpl);
功能描述	初始化外设I2S
先决条件	-
被调用函数	-
输入参数{in}	
spi_periph	外设I2S
SPIx (x=0)	I2S外设选择
输入参数{in}	
mode	I2S运行模式
I2S_MODE_SLAVE_TX	I2S从机发送模式
I2S_MODE_SLAVE_RX	I2S从机接收模式
I2S_MODE_MASTER_TX	I2S主机发送模式
I2S_MODE_MASTER_RX	I2S主机接收模式
输入参数{in}	
standard	I2S标准选择
I2S_STD_PHILIPS	I2S飞利浦标准
I2S_STD_MSB	I2S MSB对齐标准
I2S_STD_LSB	I2S LSB对齐标准
I2S_STD_PCMSHORT	I2S PCM短帧标准
I2S_STD_PCMLONG	I2S PCM长帧标准
输入参数{in}	
ckpl	I2S空闲状态时钟极性
I2S_CKPL_LOW	I2S_CK空闲状态为低电平
I2S_CKPL_HIGH	I2S_CK空闲状态为高电平
输出参数{out}	

-	-
返回值	
-	-

例如：

```
/* initialize I2S0 */
```

```
i2s_init(SPI0, I2S_MODE_MASTERTX, I2S_STD_PHILIPS, I2S_CKPL_LOW);
```

### 函数 i2s\_psc\_config

函数i2s\_psc\_config描述见下表：

表 3-439. 函数 i2s\_psc\_config

函数名称	i2s_psc_config
函数原形	void i2s_psc_config(uint32_t spi_periph, uint32_t audiosample, uint32_t frameformat, uint32_t mckout);
功能描述	配置I2S预分频器
先决条件	-
被调用函数	rcu_clock_freq_get
输入参数{in}	
spi_periph	外设I2S
SPIx (x=0)	I2S外设选择
输入参数{in}	
audiosample	I2S音频采样频率
I2S_AUDIOSAMPL E_8K	音频采样频率为8KHz
I2S_AUDIOSAMPL E_11K	音频采样频率为11KHz
I2S_AUDIOSAMPL E_16K	音频采样频率为16KHz
I2S_AUDIOSAMPL E_22K	音频采样频率为22KHz
I2S_AUDIOSAMPL E_32K	音频采样频率为32KHz
I2S_AUDIOSAMPL E_44K	音频采样频率为44KHz
I2S_AUDIOSAMPL E_48K	音频采样频率为48KHz
I2S_AUDIOSAMPL E_96K	音频采样频率为96KHz
I2S_AUDIOSAMPL E_192K	音频采样频率为192KHz
输入参数{in}	

<b>frameformat</b>	I2S数据长度和通道长度
<i>I2S_FRAMEFORMAT_DT16B_CH16B</i>	I2S数据长度为16位，通道长度为16位
<i>I2S_FRAMEFORMAT_DT16B_CH32B</i>	I2S数据长度为16位，通道长度为32位
<i>I2S_FRAMEFORMAT_DT24B_CH32B</i>	I2S数据长度为24位，通道长度为32位
<i>I2S_FRAMEFORMAT_DT32B_CH32B</i>	I2S数据长度为32位，通道长度为32位
<b>输入参数{in}</b>	
<b>mckout</b>	I2S_MCK输出
<i>I2S_MCKOUT_ENABLE</i>	I2S_MCK输出使能
<i>I2S_MCKOUT_DISABLE</i>	I2S_MCK输出禁止
<b>输出参数{out}</b>	
-	-
<b>返回值</b>	
-	-

例如：

```
/* configure I2S0 prescaler */
```

```
i2s_psc_config(SPI0, I2S_AUDIOSAMPLE_44K, I2S_FRAMEFORMAT_DT16B_CH16B,
I2S_MCKOUT_DISABLE);
```

## 函数 i2s\_enable

函数i2s\_enable描述见下表：

**表 3-440. 函数 i2s\_enable**

<b>函数名称</b>	i2s_enable
<b>函数原形</b>	void i2s_enable(uint32_t spi_periph);
<b>功能描述</b>	使能外设I2S
<b>先决条件</b>	-
<b>被调用函数</b>	-
<b>输入参数{in}</b>	
<b>spi_periph</b>	外设I2S0
<i>SPIx (x=0)</i>	I2S外设选择
<b>输出参数{out}</b>	
-	-
<b>返回值</b>	
-	-

例如：

```
/* enable I2S0 */
```

```
i2s_enable(SPI0);
```

### 函数 i2s\_disable

函数i2s\_disable描述见下表:

表 3-441. 函数 i2s\_disable

函数名称	i2s_disable
函数原形	void i2s_disable(uint32_t spi_periph);
功能描述	失能外设I2S
先决条件	-
被调用函数	-
输入参数{in}	
spi_periph	外设I2S
SPIx (x=1)	I2S外设选择
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* disable I2S0 */
```

```
i2s_disable(SPI0);
```

### 函数 spi\_nss\_output\_enable

函数spi\_nss\_output\_enable描述见下表:

表 3-442. 函数 spi\_nss\_output\_enable

函数名称	spi_nss_output_enable
函数原形	void spi_nss_output_enable(uint32_t spi_periph);
功能描述	使能外设SPI NSS输出
先决条件	-
被调用函数	-
输入参数{in}	
spi_periph	外设SPI
SPIx (x=0,1)	SPI外设选择
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* enable SPI0 NSS output */
```

```
spi_nss_output_enable(SPI0);
```

### 函数 spi\_nss\_output\_disable

函数spi\_nss\_output\_disable描述见下表:

表 3-443. 函数 spi\_nss\_output\_disable

函数名称	spi_nss_output_disable
函数原形	void spi_nss_output_disable(uint32_t spi_periph);
功能描述	失能外设SPI NSS输出
先决条件	-
被调用函数	-
输入参数{in}	
spi_periph	外设SPI
SPIx (x=0,1)	SPI外设选择
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* disable SPI0 NSS output */
```

```
spi_nss_output_disable(SPI0);
```

### 函数 spi\_nss\_internal\_high

函数spi\_nss\_internal\_high描述见下表:

表 3-444. 函数 spi\_nss\_internal\_high

函数名称	spi_nss_internal_high
函数原形	void spi_nss_internal_high(uint32_t spi_periph);
功能描述	NSS软件模式下NSS引脚拉高
先决条件	-
被调用函数	-
输入参数{in}	
spi_periph	外设SPI
SPIx (x=0,1)	SPI外设选择
输出参数{out}	
-	-
返回值	
-	-

例如:



```
/* SPI0 NSS pin is pulled high level in software mode */
```

```
spi_nss_internal_high(SPI0);
```

### 函数 spi\_nss\_internal\_low

函数spi\_nss\_internal\_low描述见下表：

**表 3-445. 函数 spi\_nss\_internal\_low**

函数名称	spi_nss_internal_low
函数原形	void spi_nss_internal_low(uint32_t spi_periph);
功能描述	NSS软件模式下NSS引脚拉低
先决条件	-
被调用函数	-
输入参数{in}	
spi_periph	外设SPI
SPIx (x=0,1)	SPI外设选择
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* SPI0 NSS pin is pulled low level in software mode */
```

```
spi_nss_internal_low(SPI0);
```

### 函数 spi\_dma\_enable

函数spi\_dma\_enable描述见下表：

**表 3-446. 函数 spi\_dma\_enable**

函数名称	spi_dma_enable
函数原形	void spi_dma_enable(uint32_t spi_periph, uint8_t dma);
功能描述	使能外设SPI的DMA功能
先决条件	-
被调用函数	-
输入参数{in}	
spi_periph	外设SPI
SPIx (x=0,1)	SPI外设选择
输入参数{in}	
dma	SPI DMA模式
SPI_DMA_TRANSMIT	SPI发送缓冲区DMA使能
SPI_DMA_RECEIVE	SPI接收缓冲区DMA使能

<i>E</i>	
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* enable SPI0 transmit data DMA function */
```

```
spi_dma_enable(SPI0, SPI_DMA_TRANSMIT);
```

### 函数 spi\_dma\_disable

函数spi\_dma\_disable描述见下表:

表 3-447. 函数 spi\_dma\_disable

函数名称	spi_dma_disable
函数原形	void spi_dma_disable(uint32_t spi_periph, uint8_t dma);
功能描述	失能外设SPI的DMA功能
先决条件	-
被调用函数	-
输入参数{in}	
spi_periph	外设SPI
SPIx (x=0,1)	SPI外设选择
输入参数{in}	
dma	SPI DMA模式
SPI_DMA_TRANSMIT	SPI发送缓冲区DMA失能
SPI_DMA_RECEIVE	SPI接收缓冲区DMA失能
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* disable SPI0 transmit data DMA function */
```

```
spi_dma_disable(SPI0, SPI_DMA_TRANSMIT);
```

### 函数 spi\_transmit\_odd\_config

函数spi\_transmit\_odd\_config描述见下表:

表 3-448. 函数 `spi_transmit_odd_config`

函数名称	<code>spi_transmit_odd_config</code>
函数原形	<code>void spi_transmit_odd_config(uint32_t spi_periph, uint16_t odd);</code>
功能描述	配置SPI0通过DMA发送的数据总数是否为奇数
先决条件	-
被调用函数	-
输入参数{in}	
<code>spi_periph</code>	外设SPI
<code>SPIx (x=1)</code>	SPI外设选择
输入参数{in}	
<code>odd</code>	DMA通道发送的字节数是奇数还是偶数
<code>SPI_TXDMA_EVEN</code>	DMA发送的字节数是偶数
<code>SPI_TXDMA_ODD</code>	DMA发送的字节数是奇数
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* configure SPI1 total number of data to transmit by DMA is odd */
```

```
spi_transmit_odd_config(SPI1, SPI_TXDMA_ODD);
```

### 函数 `spi_receive_odd_config`

函数`spi_receive_odd_config`描述见下表:

表 3-449. 函数 `spi_receive_odd_config`

函数名称	<code>spi_receive_odd_config</code>
函数原形	<code>void spi_receive_odd_config(uint32_t spi_periph, uint16_t odd);</code>
功能描述	配置SPI0通过DMA接收到的数据总数是否为奇数
先决条件	-
被调用函数	-
输入参数{in}	
<code>spi_periph</code>	外设SPI
<code>SPIx (x=1)</code>	SPI外设选择
输入参数{in}	
<code>odd</code>	DMA通道接收的字节数时奇数还是偶数
<code>SPI_RXDMA_EVEN</code>	DMA接收的字节数是偶数
<code>SPI_RXDMA_ODD</code>	DMA接收的字节数是奇数
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* configure SPI1 total number of data to receive by DMA is odd */
```

```
spi_receive_odd_config(SPI1, SPI_RXDMA_ODD);
```

### 函数 spi\_i2s\_data\_frame\_format\_config

函数spi\_i2s\_data\_frame\_format\_config描述见下表：

表 3-450. 函数 spi\_i2s\_data\_frame\_format\_config

函数名称	spi_i2s_data_frame_format_config
函数原形	ErrStatus spi_i2s_data_frame_format_config(uint32_t spi_periph, uint16_t frame_format);
功能描述	配置外设SPI/I2S数据帧格式
先决条件	-
被调用函数	-
输入参数{in}	
spi_periph	外设SPI
SPIx (x=0,1)	SPI外设选择
输入参数{in}	
frame_format	SPI帧大小
SPI_FRAME_SIZE_x BIT	SPI x位数据帧格式，x=4,5,..16
输出参数{out}	
-	-
返回值	
ErrStatus	ERROR或者SUCCESS-

例如：

```
/* configure SPI1 data frame format size is 16 bits */
```

```
spi_i2s_data_frame_format_config(SPI1, SPI_FRAME_SIZE_16BIT);
```

### 函数 spi\_fifo\_access\_size\_config

函数spi\_fifo\_access\_size\_config描述见下表：

表 3-451. 函数 spi\_fifo\_access\_size\_config

函数名称	spi_fifo_access_size_config
函数原形	void spi_fifo_access_size_config(uint32_t spi_periph, uint16_t fifo_access_size);
功能描述	配置SPI0的FIFO访问大小
先决条件	-
被调用函数	-
输入参数{in}	

<b>spi_periph</b>	外设SPI
<b>SPIx (x=1)</b>	SPI外设选择
<b>输入参数{in}</b>	
<b>fifo_access_size</b>	fifo访问大小
<b>SPI_HALFWORD_ACCESS</b>	半字访问
<b>SPI_BYTE_ACCESS</b>	字节访问
<b>输出参数{out}</b>	
-	-
<b>返回值</b>	
-	-

例如：

```
/* configure SPI1 access size half word */
```

```
spi_fifo_access_size_config(SPI1, SPI_HALFWORD_ACCESS);
```

### 函数 spi\_bidirectional\_transfer\_config

函数spi\_bidirectional\_transfer\_config描述见下表：

**表 3-452. 函数 spi\_bidirectional\_transfer\_config**

<b>函数名称</b>	spi_bidirectional_transfer_config
<b>函数原形</b>	void spi_bidirectional_transfer_config(uint32_t spi_periph, uint32_t transfer_direction);
<b>功能描述</b>	配置外设SPI的数据传输方向
<b>先决条件</b>	-
<b>被调用函数</b>	-
<b>输入参数{in}</b>	
<b>spi_periph</b>	外设SPI
<b>SPIx (x=0,1)</b>	SPI外设选择
<b>输入参数{in}</b>	
<b>transfer_direction</b>	SPI双向传输输出使能
<b>SPI_BIDIRECTIONAL_TRANSMIT</b>	SPI工作在只发送模式
<b>SPI_BIDIRECTIONAL_RECEIVE</b>	SPI工作在只接收模式
<b>输出参数{out}</b>	
-	-
<b>返回值</b>	
-	-

例如：

```
/* SPI0 works in transmit-only mode */
```

```
spi_bidirectional_transfer_config(SPI0, SPI_BIDIRECTIONAL_TRANSMIT);
```

### 函数 spi\_i2s\_data\_transmit

函数spi\_i2s\_data\_transmit描述见下表：

**表 3-453. 函数 spi\_i2s\_data\_transmit**

函数名称	spi_i2s_data_transmit
函数原形	void spi_i2s_data_transmit(uint32_t spi_periph, uint16_t data);
功能描述	SPI发送数据
先决条件	-
被调用函数	-
输入参数{in}	
spi_periph	外设SPI
SPIx (x=0,1)	SPI外设选择
输入参数{in}	
data	16位数据
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* SPI0 transmit data */
```

```
uint16_t spi0_send_array[10];
```

```
uint8_t send_n = 1;
```

```
spi_i2s_data_transmit(SPI0, spi0_send_array[send_n]);
```

### 函数 spi\_i2s\_data\_receive

函数spi\_i2s\_data\_receive描述见下表：

**表 3-454. 函数 spi\_i2s\_data\_receive**

函数名称	spi_i2s_data_receive
函数原形	uint16_t spi_i2s_data_receive(uint32_t spi_periph);
功能描述	SPI接收数据
先决条件	-
被调用函数	-
输入参数{in}	
spi_periph	外设SPI
SPIx (x=0,1)	SPI外设选择

输出参数{out}	
-	-
返回值	
uint16_t	16位数据

例如:

```
/* SPI0 receive data */
uint16_t spi0_receive_array[10];
uint8_t receive_n = 1;
spi0_receive_array[receive_n] = spi_i2s_data_receive(SPI0);
```

### 函数 spi\_crc\_polynomial\_set

函数spi\_crc\_polynomial\_set描述见下表:

表 3-455. 函数 spi\_crc\_polynomial\_set

函数名称	spi_crc_polynomial_set
函数原形	void spi_crc_polynomial_set(uint32_t spi_periph, uint16_t crc_poly);
功能描述	设置外设SPI的CRC多项式值
先决条件	-
被调用函数	-
输入参数{in}	
spi_periph	外设SPI
SPIx (x=0,1)	SPI外设选择
输入参数{in}	
crc_poly	CRC多项式值
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* set SPI0 CRC polynomial */
uint16_t CRC_VALUE = 0x8;
spi_crc_polynomial_set(SPI0,CRC_VALUE);
```

### 函数 spi\_crc\_polynomial\_get

函数spi\_crc\_polynomial\_get描述见下表:

表 3-456. 函数 spi\_crc\_polynomial\_get

函数名称	spi_crc_polynomial_get
------	------------------------

函数原形	uint16_t spi_crc_polynomial_get(uint32_t spi_periph);
功能描述	获取外设SPI的CRC多项式值
先决条件	-
被调用函数	-
输入参数{in}	
spi_periph	外设SPI
SPIx (x=0,1)	SPI外设选择
输出参数{out}	
-	-
返回值	
uint16_t	16位CRC多项式值 (0-0xFFFF)

例如:

```
/* get SPI0 CRC polynomial */
uint16_t crc_val;
crc_val = spi_crc_polynomial_get(SPI0);
```

### 函数 spi\_crc\_length\_set

函数spi\_crc\_length\_set描述见下表:

表 3-457. 函数 spi\_crc\_length\_set

函数名称	spi_crc_length_set
函数原形	void spi_crc_length_set(uint32_t spi_periph, uint16_t crc_length);
功能描述	设置CRC长度
先决条件	-
被调用函数	-
输入参数{in}	
spi_periph	外设SPI
SPIx (x=1)	SPI外设选择
输入参数{in}	
crc_length	crc长度
SPI_CRC_8BIT	CRC长度为8位数据
SPI_CRC_16BIT	CRC长度为16位数据
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/*set SPI0 CRC length 16 bits */
spi_crc_length_set(SPI1,SPI_CRC_16BIT);
```



## 函数 spi\_crc\_on

函数spi\_crc\_on描述见下表：

表 3-458. 函数 spi\_crc\_on

函数名称	spi_crc_on
函数原形	void spi_crc_on(uint32_t spi_periph);
功能描述	打开外设SPI的CRC功能
先决条件	-
被调用函数	-
输入参数{in}	
spi_periph	外设SPI
SPIx (x=0,1)	SPI外设选择
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* turn on SPI0 CRC function */
```

```
spi_crc_on(SPI0);
```

## 函数 spi\_crc\_off

函数spi\_crc\_off描述见下表：

表 3-459. 函数 spi\_crc\_off

函数名称	spi_crc_off
函数原形	void spi_crc_off(uint32_t spi_periph);
功能描述	关闭外设SPI的CRC功能
先决条件	-
被调用函数	-
输入参数{in}	
spi_periph	外设SPI
SPIx (x=0,1)	SPI外设选择
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* turn off SPI0 CRC function */
```

```
spi_crc_off(SPI0);
```

函数 **spi\_crc\_next**

函数spi\_crc\_next描述见下表:

表 3-460. 函数 **spi\_crc\_next**

函数名称	spi_crc_next
函数原形	void spi_crc_next(uint32_t spi_periph);
功能描述	设置外设SPI下一次传输数据为CRC值
先决条件	-
被调用函数	-
输入参数{in}	
spi_periph	外设SPI
SPIx (x=0,1)	SPI外设选择
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* SPI0 next data is CRC value */
```

```
spi_crc_next(SPI0);
```

函数 **spi\_crc\_get**

函数spi\_crc\_get描述见下表:

表 3-461. 函数 **spi\_crc\_get**

函数名称	spi_crc_get
函数原形	uint16_t spi_crc_get(uint32_t spi_periph, uint8_t crc);
功能描述	外设SPI获取CRC值
先决条件	-
被调用函数	-
输入参数{in}	
spi_periph	外设SPI
SPIx (x=0,1)	SPI外设选择
输入参数{in}	
crc	SPI crc值
SPI_CRC_TX	获取发送CRC寄存器值
SPI_CRC_RX	获取接收CRC寄存器值
输出参数{out}	
-	-
返回值	
uint16_t	16位CRC值 (0-0xFFFF)

例如：

```
/* get SPI0 CRC send value */
uint16_t crc_val;

crc_val = spi_crc_get(SPI0, SPI_CRC_TX);
```

### 函数 spi\_crc\_error\_clear

函数spi\_crc\_error\_clear描述见下表：

表 3-462. 函数 spi\_crc\_error\_clear

函数名称	spi_crc_error_clear
函数原形	void spi_crc_error_clear(uint32_t spi_periph);
功能描述	清除SPI CRC错误标志状态
先决条件	-
被调用函数	-
输入参数{in}	
spi_periph	外设SPI
SPIx (x=0,1)	SPI外设选择
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* clear SPI0 CRC error flag status */

spi_crc_error_clear(SPI0);
```

### 函数 spi\_ti\_mode\_enable

函数spi\_ti\_mode\_enable描述见下表：

表 3-463. 函数 spi\_ti\_mode\_enable

函数名称	spi_ti_mode_enable
函数原形	void spi_ti_mode_enable(uint32_t spi_periph);
功能描述	使能SPI TI模式
先决条件	-
被调用函数	-
输入参数{in}	
spi_periph	外设SPI
SPIx (x=0,1)	SPI外设选择
输出参数{out}	
-	-

返回值	
-	-

例如：

```
/* enable SPI0 TI mode */
spi_ti_mode_enable(SPI0);
```

### 函数 spi\_ti\_mode\_disable

函数spi\_ti\_mode\_disable描述见下表：

表 3-464. 函数 spi\_ti\_mode\_disable

函数名称	spi_ti_mode_disable
函数原形	void spi_ti_mode_disable(uint32_t spi_periph);
功能描述	失能SPI TI模式
先决条件	-
被调用函数	-
输入参数{in}	
spi_periph	外设SPIx
SPIx (x=0,1)	SPI外设选择
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* disable SPI0 TI mode */
spi_ti_mode_disable(SPI0);
```

### 函数 spi\_nssp\_mode\_enable

函数spi\_nssp\_mode\_enable描述见下表：

表 3-465. 函数 spi\_nssp\_mode\_enable

函数名称	spi_nssp_mode_enable
函数原形	void spi_nssp_mode_enable(uint32_t spi_periph);
功能描述	使能SPI NSS脉冲模式
先决条件	-
被调用函数	-
输入参数{in}	
spi_periph	外设SPI
SPIx (x=0,1)	SPI外设选择
输出参数{out}	

-	-
返回值	
-	-

例如：

```
/* enable SPI0 NSS pulse mode */
```

```
spi_nssp_mode_enable(SPI0);
```

### 函数 spi\_nssp\_mode\_disable

函数spi\_nssp\_mode\_disable描述见下表：

表 3-466. 函数 spi\_nssp\_mode\_disable

函数名称	spi_nssp_mode_disable
函数原形	void spi_nssp_mode_disable(uint32_t spi_periph);
功能描述	失能SPI NSS脉冲模式
先决条件	-
被调用函数	-
输入参数{in}	
spi_periph	外设SPI
SPIx (x=0,1)	SPI外设选择
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* disable SPI0 NSS pulse mode */
```

```
spi_nssp_mode_disable(SPI0);
```

### 函数 spi\_quad\_enable

函数spi\_quad\_enable描述见下表：

表 3-467. 函数 spi\_quad\_enable

函数名称	spi_quad_enable
函数原形	void spi_quad_enable (uint32_t spi_periph);
功能描述	使能四线SPI模式
先决条件	-
被调用函数	-
输入参数{in}	
spi_periph	外设SPI
SPIx (x=1)	SPI外设选择

输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* enable SPI1 quad wire mode */
```

```
spi_quad_enable(SPI1);
```

### 函数 spi\_quad\_disable

函数spi\_quad\_disable描述见下表：

表 3-468. 函数 spi\_quad\_disable

函数名称	spi_quad_disable
函数原形	void spi_quad_disable(uint32_t spi_periph);
功能描述	失能四线SPI模式
先决条件	-
被调用函数	-
输入参数{in}	
spi_periph	外设SPI
SPIx (x=1)	SPI外设选择
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* disable SPI1 quad wire mode */
```

```
spi_quad_disable(SPI1);
```

### 函数 spi\_quad\_write\_enable

函数spi\_quad\_write\_enable描述见下表：

表 3-469. 函数 spi\_quad\_write\_enable

函数名称	spi_quad_write_enable
函数原形	void spi_quad_write_enable(uint32_t spi_periph);
功能描述	使能四线SPI写
先决条件	-
被调用函数	-
输入参数{in}	
spi_periph	外设SPI

<i>SPIx (x=1)</i>	SPI外设选择
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* enable SPI1 quad wire write */
```

```
spi_quad_write_enable(SPI1);
```

### 函数 spi\_quad\_read\_enable

函数spi\_quad\_read\_enable描述见下表：

表 3-470. 函数 spi\_quad\_read\_enable

函数名称	spi_quad_read_enable
函数原形	void spi_quad_read_enable(uint32_t spi_periph);
功能描述	使能四线SPI读
先决条件	-
被调用函数	-
输入参数{in}	
spi_periph	外设SPI
<i>SPIx (x=1)</i>	SPI外设选择
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* enable SPI1 quad wire read */
```

```
spi_quad_read_enable(SPI1);
```

### 函数 spi\_quad\_io23\_output\_enable

函数spi\_quad\_io23\_output\_enable描述见下表：

表 3-471. 函数 spi\_quad\_io23\_output\_enable

函数名称	spi_quad_io23_output_enable
函数原形	void spi_quad_io23_output_enable(uint32_t spi_periph);
功能描述	使能SPI_IO2和SPI_IO3输出
先决条件	-
被调用函数	-
输入参数{in}	

<b>spi_periph</b>	外设SPI
<i>SPIx (x=1)</i>	SPI外设选择
<b>输出参数{out}</b>	
-	-
<b>返回值</b>	
-	-

例如：

```
/* enable SPI1 SPI_IO2 and SPI_IO3 pin output */
```

```
spi_quad_io23_output_enable(SPI1);
```

### 函数 spi\_quad\_io23\_output\_disable

函数spi\_quad\_io23\_output\_disable描述见下表：

**表 3-472. 函数 spi\_quad\_io23\_output\_disable**

<b>函数名称</b>	spi_quad_io23_output_disable
<b>函数原形</b>	void spi_quad_io23_output_disable(uint32_t spi_periph);
<b>功能描述</b>	失能SPI_IO2和SPI_IO3输出
<b>先决条件</b>	-
<b>被调用函数</b>	-
<b>输入参数{in}</b>	
<b>spi_periph</b>	外设SPI
<i>SPIx (x=1)</i>	SPI外设选择
<b>输出参数{out}</b>	
-	-
<b>返回值</b>	
-	-

例如：

```
/* disable SPI1 SPI_IO2 and SPI_IO3 pin output */
```

```
spi_quad_io23_output_disable(SPI1);
```

### 函数 spi\_i2s\_format\_error\_clear

函数spi\_i2s\_format\_error\_clear描述见下表：

**表 3-473. 函数 spi\_i2s\_format\_error\_clear**

<b>函数名称</b>	spi_i2s_format_error_clear
<b>函数原形</b>	void spi_i2s_format_error_clear(uint32_t spi_periph, uint32_t flag);
<b>功能描述</b>	清除SPI/I2S格式错误标志
<b>先决条件</b>	-
<b>被调用函数</b>	-



输入参数{in}	
<b>spi_periph</b>	外设SPI
<i>SPIx (x=0,1)</i>	SPI外设选择
输入参数{in}	
<b>flag</b>	SPI/I2S格式错误标志
<i>SPI_FLAG_FERR</i>	SPI格式错误标志
<i>I2S_FLAG_FERR</i>	I2S错误标志
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* clear SPI format error flag status */
```

```
spi_i2s_format_error_clear(SPI1, SPI_FLAG_FERR);
```

### 函数 spi\_i2s\_flag\_get

函数spi\_i2s\_flag\_get描述见下表：

表 3-474. 函数 spi\_i2s\_flag\_get

函数名称	spi_i2s_flag_get
函数原形	FlagStatus spi_i2s_flag_get(uint32_t spi_periph, uint32_t flag);
功能描述	获取外设SPI/I2S标志状态
先决条件	-
被调用函数	-
输入参数{in}	
<b>spi_periph</b>	外设SPI
<i>SPIx (x=0,1)</i>	SPI外设选择
输入参数{in}	
<b>flag</b>	SPI/I2S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_RXORERR</i>	SPI接收过载错误标志
<i>SPI_FLAG_CONFERR</i>	SPI配置错误标志
<i>SPI_FLAG_CRCERR</i>	SPI CRC错误标志
<i>SPI_FLAG_FERR</i>	SPI格式错误标志
<i>I2S_FLAG_TBE</i>	I2S发送缓冲区空标志
<i>I2S_FLAG_RBNE</i>	I2S接收缓冲区非空标志

<i>I2S_FLAG_TRANS</i>	I2S通信进行中标志
<i>I2S_FLAG_RXORERR</i>	I2S接收过载错误标志
<i>I2S_FLAG_TXURERR</i>	I2S发送欠载错误标志
<i>I2S_FLAG_CH</i>	I2S通道标志
<i>I2S_FLAG_FERR</i>	I2S格式错误标志
以下参数只适用于SPI0	
<i>SPI_TXLVL_EMPTY</i>	SPI TXFIFO空
<i>SPI_TXLVL_QUARTER_FULL</i>	SPI TXFIFO四分之一满
<i>SPI_TXLVL_HALF_FULL</i>	SPI TXFIFO半满
<i>SPI_TXLVL_FULL</i>	TXFIFO全满
<i>SPI_RXLVL_EMPTY</i>	SPI RXFIFO空
<i>SPI_RXLVL_QUARTER_FULL</i>	SPI RXFIFO四分之一满
<i>SPI_RXLVL_HALF_FULL</i>	SPI RXFIFO半满
<i>SPI_RXLVL_FULL</i>	RXFIFO全满
输出参数{out}	
-	-
返回值	
<b>FlagStatus</b>	SET 或 RESET

例如：

```
/* get SPI0 transmit buffer empty flag status */
```

```
FlagStatus Flag = RESET;
```

```
Flag = spi_i2s_flag_get(SPI0, SPI_FLAG_TBE);
```

### 函数 **spi\_i2s\_interrupt\_enable**

函数spi\_i2s\_interrupt\_enable描述见下表：

**表 3-475. 函数 spi\_i2s\_interrupt\_enable**

函数名称	spi_i2s_interrupt_enable
函数原形	void spi_i2s_interrupt_enable(uint32_t spi_periph, uint8_t interrupt);
功能描述	使能外设SPI/I2S中断
先决条件	-
被调用函数	-

输入参数{in}	
<b>spi_periph</b>	外设SPI
<i>SPIx</i> ( <i>x</i> =0,1)	SPI外设选择
输入参数{in}	
<b>interrupt</b>	SPI/I2S中断
<i>SPI_I2S_INT_TBE</i>	发送缓冲区空中断
<i>SPI_I2S_INT_RBNE</i>	接收缓冲区非空中断
<i>SPI_I2S_INT_ERR</i>	错误中断使能
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* enable SPI0 transmit buffer empty interrupt */
```

```
spi_i2s_interrupt_enable(SPI0, SPI_I2S_INT_TBE);
```

### 函数 spi\_i2s\_interrupt\_disable

函数spi\_i2s\_interrupt\_disable描述见下表：

**表 3-476. 函数 spi\_i2s\_interrupt\_disable**

<b>函数名称</b>	spi_i2s_interrupt_disable
<b>函数原形</b>	void spi_i2s_interrupt_disable(uint32_t spi_periph, uint8_t interrupt);
<b>功能描述</b>	失能外设SPI/I2S中断
<b>先决条件</b>	-
<b>被调用函数</b>	-
输入参数{in}	
<b>spi_periph</b>	外设SPI
<i>SPIx</i> ( <i>x</i> =0,1)	SPI外设选择
输入参数{in}	
<b>interrupt</b>	SPI/I2S中断
<i>SPI_I2S_INT_TBE</i>	发送缓冲区空中断
<i>SPI_I2S_INT_RBNE</i>	接收缓冲区非空中断
<i>SPI_I2S_INT_ERR</i>	错误中断使能
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* disable SPI0 transmit buffer empty interrupt */
```

`spi_i2s_interrupt_disable(SPI0, SPI_I2S_INT_TBE);`

### 函数 `spi_i2s_interrupt_flag_get`

函数 `spi_i2s_interrupt_flag_get` 描述见下表:

**表 3-477. 函数 `spi_i2s_interrupt_flag_get`**

函数名称	<code>spi_i2s_interrupt_flag_get</code>
函数原形	<code>FlagStatus spi_i2s_interrupt_flag_get(uint32_t spi_periph, uint8_t interrupt);</code>
功能描述	获取外设SPI/I2S中断状态
先决条件	-
被调用函数	-
输入参数{in}	
<b>spi_periph</b>	外设SPI
<i>SPIx</i>	x=0,1
输入参数{in}	
<b>interrupt</b>	SPI/I2S中断状态
<i>SPI_I2S_INT_FLAG_TBE</i>	发送缓冲区空中断
<i>SPI_I2S_INT_FLAG_RBNE</i>	接收缓冲区非空中断
<i>SPI_I2S_INT_FLAG_RXORERR</i>	接收过载错误中断
<i>SPI_INT_FLAG_CONFERR</i>	配置错误中断
<i>SPI_INT_FLAG_CRCERR</i>	CRC错误中断
<i>I2S_INT_FLAG_TXURERR</i>	发送欠载错误中断
<i>SPI_I2S_INT_FLAG_FERR</i>	格式错误中断
输出参数{out}	
-	-
返回值	
<b>FlagStatus</b>	SET 或 RESET

例如:

```
/* get SPI0 transmit buffer empty interrupt status */
```

```
FlagStatus Flag_interrupt = RESET;
```

```
Flag_interrupt = spi_i2s_interrupt_flag_get(SPI0, SPI_I2S_INT_FLAG_TBE);
```

### 3.17. SYSCFG

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

#### 3.17.1. 外设寄存器说明

SYSCFG寄存器列表如下表所示：

**表 3-478. 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_CFG1	配置寄存器1
SYSCFG_STAT	状态寄存器
SYSCFG_CFG2	配置寄存器2
SYSCFG_CFG3	配置寄存器3
SYSCFG_CPU_IRQ_LAT	IRQ延迟寄存器
SYSCFG_TIMER0CFG0	TIMER0配置寄存器
SYSCFG_TIMER0CFG1	TIMER0配置寄存器
SYSCFG_TIMER2CFG0	TIMER2配置寄存器
SYSCFG_TIMER2CFG1	TIMER2配置寄存器

#### 3.17.2. 外设库函数说明

SYSCFG库函数列表如下表所示：

**表 3-479. SYSCFG 库函数**

库函数名称	库函数描述
syscfg_deinit	复位SYSCFG寄存器
syscfg_i2c_fast_mode_plus_enable	使能I2C Fm+模式
syscfg_i2c_fast_mode_plus_disable	失能I2C Fm+模式
syscfg_pin_remap_enable	使能引脚重映射
syscfg_pin_remap_disable	失能引脚重映射
syscfg_bootmode_get	获取BOOT启动方式
syscfg_exti_line_config	配置GPIO引脚作为EXTI

库函数名称	库函数描述
syscfg_lockup_enable	使能LOCKUP
syscfg_lockup_disable	失能LOCKUP
syscfg_sram_ecc_single_correctable_bit_get	获取单比特可纠错错误位
syscfg_sram_ecc_error_address_get	获取单比特可纠错错误地址
syscfg_irq_latency_set	设置中断延时
syscfg_interrupt_enable	使能中断
syscfg_interrupt_disable	失能中断
syscfg_interrupt_flag_get	获取中断标志
syscfg_interrupt_flag_clear	清除中断标志

### 函数 syscfg\_deinit

函数syscfg\_deinit描述见下表：

表 3-480. 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\_i2c\_fast\_mode\_plus\_enable

函数syscfg\_i2c\_fast\_mode\_plus\_enable描述见下表：

表 3-481. 函数 syscfg\_i2c\_fast\_mode\_plus\_enable

函数名称	syscfg_i2c_fast_mode_plus_enable
函数原形	void syscfg_i2c_fast_mode_plus_enable(uint32_t syscfg_gpio);
功能描述	使能I2C Fm+模式
先决条件	-
被调用函数	-
输入参数{in}	

<b>syscfg_gpio</b>	I2C I2C Fm+模式
<i>SYSCFG_PB6_FM</i> <i>PEN</i>	PB6 I2C Fm+模式
<i>SYSCFG_PB7_FM</i> <i>PEN</i>	PB7 I2C Fm+模式
<i>SYSCFG_PB8_FM</i> <i>PEN</i>	PB8 I2C Fm+模式
<i>SYSCFG_PB9_FM</i> <i>PEN</i>	PB9 I2C Fm+模式
<i>SYSCFG_I2C0_FM</i> <i>PEN</i>	I2C0 I2C Fm+模式
<i>SYSCFG_PA9_FM</i> <i>PEN</i>	PA9 I2C Fm+模式
<i>SYSCFG_PA10_FM</i> <i>PEN</i>	PA10 I2C Fm+模式
<i>SYSCFG_PC14_FM</i> <i>PEN</i>	PC14 I2C Fm+模式
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* enable PB6 I2C Fm+ mode */
```

```
syscfg_i2c_fast_mode_plus_enable(SYSCFG_PB6_FMPEN);
```

### 函数 syscfg\_i2c\_fast\_mode\_plus\_disable

函数syscfg\_i2c\_fast\_mode\_plus\_disable描述见下表:

**表 3-482. 函数 syscfg\_i2c\_fast\_mode\_plus\_disable**

<b>函数名称</b>	syscfg_i2c_fast_mode_plus_disable
<b>函数原形</b>	void syscfg_i2c_fast_mode_plus_disable(uint32_t syscfg_gpio);
<b>功能描述</b>	失能I2C Fm+模式
<b>先决条件</b>	-
<b>被调用函数</b>	-
输入参数{in}	
<b>syscfg_gpio</b>	I2C I2C Fm+模式
<i>SYSCFG_PB6_FM</i> <i>PEN</i>	PB6 I2C Fm+模式
<i>SYSCFG_PB7_FM</i> <i>PEN</i>	PB7 I2C Fm+模式
<i>SYSCFG_PB8_FM</i>	PB8 I2C Fm+模式

<i>PEN</i>	
<i>SYSCFG_PB9_FM</i> <i>PEN</i>	PB9 I2C Fm+模式
<i>SYSCFG_I2C0_FM</i> <i>PEN</i>	I2C0 I2C Fm+模式
<i>SYSCFG_PA9_FM</i> <i>PEN</i>	PA9 I2C Fm+模式
<i>SYSCFG_PA10_FM</i> <i>PEN</i>	PA10 I2C Fm+模式
<i>SYSCFG_PC14_FM</i> <i>PEN</i>	PC14 I2C Fm+模式
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* disable PB6 I2C Fm+ mode */
```

```
syscfg_i2c_fast_mode_plus_disable(SYSCFG_PB6_FMPEN);
```

### 函数 **syscfg\_pin\_remap\_enable**

函数syscfg\_pin\_remap\_enable描述见下表：

**表 3-483. 函数 syscfg\_pin\_remap\_enable**

函数名称	syscfg_pin_remap_enable
函数原形	void syscfg_pin_remap_enable(uint32_t remap_pin);
功能描述	对于小封装芯片，使能引脚重映射功能
先决条件	-
被调用函数	-
输入参数{in}	
<b>remap_pin</b>	引脚重映射
<i>SYSCFG_CFG0_P</i> <i>A11_RMP</i>	PA11重新映射PA9
<i>SYSCFG_CFG0_P</i> <i>A12_RMP</i>	PA12重新映射PA10
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* PA11 remap to PA9 */
```



```
syscfg_pin_remap_enable(SYSCFG_CFG0_PA11_RMP);
```

### 函数 syscfg\_pin\_remap\_disable

函数syscfg\_pin\_remap\_disable描述见下表：

**表 3-484. 函数 syscfg\_pin\_remap\_disable**

函数名称	syscfg_pin_remap_disable
函数原形	void syscfg_pin_remap_disable(uint32_t remap_pin);
功能描述	对于小封装芯片，失能引脚重映射功能
先决条件	-
被调用函数	-
输入参数{in}	
remap_pin	引脚重映射
SYSCFG_PA11_PA12_REMAP	PA11重新映射PA9
SYSCFG_BOOT0_PD3_REMAP	PA12重新映射PA10
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* disable PA11 remap to PA9 */
```

```
syscfg_pin_remap_disable(SYSCFG_CFG0_PA11_RMP);
```

### 函数 syscfg\_bootmode\_get

函数syscfg\_bootmode\_get描述见下表：

**表 3-485. 函数 syscfg\_bootmode\_get**

函数名称	syscfg_bootmode_get
函数原形	uint8_t syscfg_bootmode_get(void);
功能描述	获取BOOT启动方式
先决条件	-
被调用函数	-
输入参数{in}	
-	-
输出参数{out}	
-	-
返回值	
uint8_t	启动方式
SYSCFG_BOOTMODE	从片上闪存的主存引导启动

DE_FLASH	
SYSCFG_BOOTMODE_SYSTEM	从片上闪存的系统存储器引导启动
SYSCFG_BOOTMODE_SRAM	从片上SRAM引导启动

例如:

```
/* get the current boot mode */
```

```
uint8_t boot_mode;
```

```
boot_mode = syscfg_bootmode_get();
```

### 函数 syscfg\_exti\_line\_config

函数syscfg\_exti\_line\_config描述见下表:

**表 3-486. 函数 syscfg\_exti\_line\_config**

函数名称	syscfg_exti_line_config
函数原形	void syscfg_exti_line_config(uint8_t exti_port, uint8_t exti_pin);
功能描述	配置GPIO引脚作为EXTI
先决条件	-
被调用函数	-
输入参数{in}	
exti_port	指定EXTI使用的GPIO端口
EXTI_SOURCE_GPIOx	x=A,B,C,D,F
输入参数{in}	
exti_pin	EXTI引脚
EXTI_SOURCE_PINx	GPIOAx = 0..15, GPIOBx = 0..15, GPIOCx = 0..15, GPIODx = 0..6,8,9, GPIOFx = 0,1
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\_lockup\_enable

函数syscfg\_lockup\_enable描述见下表:

表 3-487. 函数 syscfg\_lockup\_enable

函数名称	syscfg_lockup_enable
函数原形	void syscfg_lockup_enable(uint32_t lockup);
功能描述	使能模块lockup
先决条件	-
被调用函数	-
输入参数{in}	
lockup	模块lockup
SYSCFG_LOCKUP_LOCK	CPU lockup信号
SYSCFG_SRAM_LOCKUP	SRAM ECC错误信号
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* enable SRAM ECC lockup signal */
```

```
syscfg_lockup_enable(SYSCFG_SRAM_LOCKUP);
```

### 函数 syscfg\_lockup\_disable

函数syscfg\_lockup\_disable描述见下表:

表 3-488. 函数 syscfg\_lockup\_disable

函数名称	syscfg_lockup_disable
函数原形	void syscfg_lockup_disable(uint32_t lockup);
功能描述	失能模块lockup
先决条件	-
被调用函数	-
输入参数{in}	
lockup	模块lockup
SYSCFG_LOCKUP_LOCK	CPU lockup信号
SYSCFG_SRAM_LOCKUP	SRAM ECC错误信号
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* disable SRAM ECC lockup signal */
```

```
syscfg_lockup_disable(SYSCFG_SRAM_LOCKUP);
```

### 函数 syscfg\_sram\_ecc\_single\_correctable\_bit\_get

函数syscfg\_sram\_ecc\_single\_correctable\_bit\_get描述见下表：

**表 3-489. 函数 syscfg\_sram\_ecc\_single\_correctable\_bit\_get**

函数名称	syscfg_sram_ecc_single_correctable_bit_get
函数原形	uint32_t syscfg_sram_ecc_single_correctable_bit_get(void);
功能描述	获取SRAM ECC单比特可纠正错误位
先决条件	-
被调用函数	-
输入参数{in}	
-	-
输出参数{out}	
-	-
返回值	
error_bits	单比特可纠正错误位

例如：

```
/* SRAM ECC single correctable bit get */
```

```
Uint32_t error_bits;
```

```
error_bits = syscfg_sram_ecc_single_correctable_bit_get();
```

### 函数 syscfg\_sram\_ecc\_error\_address\_get

函数syscfg\_sram\_ecc\_error\_address\_get描述见下表：

**表 3-490. 函数 syscfg\_sram\_ecc\_error\_address\_get**

函数名称	syscfg_sram_ecc_error_address_get
函数原形	uint32_t syscfg_sram_ecc_error_address_get(void);
功能描述	获取SRAM ECC单比特可纠正错误地址
先决条件	-
被调用函数	-
输入参数{in}	
-	-
输出参数{out}	
-	-
返回值	
addr	单比特可纠正错误地址

例如：

```
/* SRAM ECC single correctable bit get */
```

```
Uint32_t addr;
```

```
addr = syscfg_sram_ecc_error_address_get();
```

### 函数 irq\_latency\_set

函数irq\_latency\_set描述见下表：

表 3-491. 函数 irq\_latency\_set

函数名称	irq_latency_set
函数原形	void irq_latency_set(uint8_t irq_latency);
功能描述	设置延迟时间值
先决条件	-
被调用函数	-
输入参数{in}	
irq_latency	延迟时间值
0x00-0xFF	延迟时间值
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* set the wait state counter value */
```

```
Irq_latency_set(0xFF);
```

### 函数 syscfg\_pinmux\_config

函数syscfg\_pinmux\_config描述见下表：

表 3-492. 函数 syscfg\_pinmux\_config

函数名称	syscfg_pinmux_config
函数原形	void syscfg_pinmux_config(syscfg_pinmux_enum syscfg_pinmux_pin, syscfg_pin_enum syscfg_pin);
功能描述	配置 GPIO 引脚多路复用器
先决条件	-
被调用函数	-
输入参数{in}	
syscfg_pinmux_pin	指定引脚所映射的GPIO
SYSCFG_PINMUX0_PB7	PB7映射到引脚1
SYSCFG_PINMUX0	PC14映射到引脚1

_PC14	
SYSCFG_PINMUX1 _PC2	PC2映射到引脚4
SYSCFG_PINMUX1 _PA0	PA0映射到引脚4
SYSCFG_PINMUX1 _PA1	PA1映射到引脚4
SYSCFG_PINMUX1 _PA2	PA2映射到引脚4
SYSCFG_PINMUX2 _PA8	PA8映射到引脚5
SYSCFG_PINMUX2 _PA11	PA11映射到引脚5
SYSCFG_PINMUX3 _PA14	PA14映射到引脚8
SYSCFG_PINMUX3 _PB6	PB6映射到引脚8
SYSCFG_PINMUX3 _PC15	PC15映射到引脚8
输入参数{in}	
syscfg_pin	映射引脚
SYSCFG_PINx(x=1, 4,5,8)	引脚（1,4,5,8）映射
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* configure the pin 1 as PB7 */
```

```
syscfg_pinmux_config(SYSCFG_PINMUX0_PB7, SYSCFG_PIN1);
```

### 函数 syscfg\_interrupt\_enable

函数syscfg\_interrupt\_enable描述见下表：

表 3-493. 函数 syscfg\_interrupt\_enable

函数名称	syscfg_interrupt_enable
函数原形	void syscfg_interrupt_enable(uint32_t interrupt);
功能描述	中断使能
先决条件	-
被调用函数	-
输入参数{in}	

interrupt	中断类型
SYSCFG_CFG2_LXTA LCSS_IE	LXTAL监控中断
SYSCFG_CFG2_HXT ALCSS_IE	HXTAL监控中断
SYSCFG_CFG2_ECC MEIE	多比特（两比特）无纠正错误NMI中断
SYSCFG_CFG2_ECCS EIE	单比特可纠正错误中断
输出参数{out}	
-	-
返回值	
-	-

例如：

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

```
syscfg_interrupt_enable(SYSCFG_CFG2_LXTALCSS_IE);
```

### 函数 syscfg\_interrupt\_disable

函数syscfg\_interrupt\_disable描述见下表：

表 3-494. 函数 syscfg\_interrupt\_disable

函数名称	syscfg_interrupt_disable
函数原形	void syscfg_interrupt_disable(uint32_t interrupt);
功能描述	中断失能
先决条件	-
被调用函数	-
输入参数{in}	
interrupt	中断类型
SYSCFG_CFG2_LXTA LCSS_IE	LXTAL监控中断
SYSCFG_CFG2_HXT ALCSS_IE	HXTAL监控中断
SYSCFG_CFG2_ECC MEIE	多比特（两比特）无纠正错误NMI中断
SYSCFG_CFG2_ECCS EIE	单比特可纠正错误中断
输出参数{out}	
-	-
返回值	
-	-

例如：

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

```
syscfg_interrupt_disable(SYSCFG_CFG2_LXTALCSS_IE);
```

### 函数 syscfg\_interrupt\_flag\_get

函数syscfg\_interrupt\_flag\_get描述见下表：

**表 3-495. 函数 syscfg\_interrupt\_flag\_get**

函数名称	syscfg_interrupt_flag_get
函数原形	FlagStatus syscfg_interrupt_flag_get(uint32_t flag);
功能描述	获取中断标志
先决条件	-
被调用函数	-
输入参数{in}	
interrupt	中断类型
SYSCFG_FLAG_ECCME	两比特无纠正事件标志
SYSCFG_FLAG_ECCSE	单比特可纠正事件标志
输出参数{out}	
-	-
返回值	
FlagStatus	SET 或者 RESET

例如：

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

```
FlagStatus Flag = RESET;
```

```
Flag = syscfg_interrupt_flag_get(SYSCFG_FLAG_ECCME);
```

### 函数 syscfg\_interrupt\_flag\_clear

函数syscfg\_interrupt\_flag\_clear描述见下表：

**表 3-496. 函数 syscfg\_interrupt\_flag\_clear**

函数名称	syscfg_interrupt_flag_clear
函数原形	void syscfg_interrupt_flag_clear(uint32_t flag);
功能描述	清除中断标志
先决条件	-
被调用函数	-
输入参数{in}	
interrupt	中断类型
SYSCFG_FLAG_ECCME	两比特无纠正事件标志



SYSCFG_FLAG_ECCS E	单比特可纠正事件标志
输出参数{out}	
-	-
返回值	
-	-

例如:

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

```
syscfg_interrupt_flag_clear(SYSCFG_FLAG_ECCME);
```

## 3.18. TIMER

定时器含有可编程的一个无符号计数器，支持输入捕获和输出比较，分为五种类型：高级定时器(TIMER0)，通用定时器L0(TIMER2)，通用定时器L2(TIMER13)，通用定时器L4(TIMER15, TIMER16)，不同类型的定时器具体功能有所差别。章节[3.18.1](#)描述了TIMER的寄存器列表，章节[3.18.2](#)对TIMER库函数进行说明。

### 3.18.1. 外设寄存器说明

TIMER寄存器列表如下表所示:

表 3-497. TIMER 寄存器

寄存器名称	寄存器描述
TIMER_CTL0	控制寄存器0
TIMERx_CTL1	控制寄存器1
TIMERx_SMCFG	从模式配置寄存器
TIMERx_DMAINTEN	DMA和中断使能寄存器
TIMERx_INTF	中断标志寄存器
TIMERx_SWEVG	软件事件产生寄存器
TIMERx_CHCTL0	通道控制寄存器0
TIMERx_CHCTL1	通道控制寄存器1
TIMERx_CHCTL2	通道控制寄存器2
TIMERx_CNT	计数器寄存器
TIMERx_PSC	预分频寄存器
TIMERx_CAR	计数器自动重载寄存器
TIMERx_CREP	重复计数寄存器
TIMERx_CH0CV	通道0捕获/比较寄存器
TIMERx_CH1CV	通道1捕获/比较寄存器
TIMERx_CH2CV	通道2捕获/比较寄存器
TIMERx_CH3CV	通道3捕获/比较寄存器
TIMERx_CCHP	互补通道保护寄存器

寄存器名称	寄存器描述
TIMERx_DMACHCFG	DMA配置寄存器
TIMERx_DMATB	DMA发送缓冲区寄存器
TIMERx_CHCTL3	通道控制寄存器3
TIMERx_CH4CV	通道3捕获/比较寄存器
TIMERx_AFCTL0	附加控制寄存器0
TIMERx_AFCTL1	附加控制寄存器1
TIMERx_INSEL	输入选择寄存器
TIMERx_CFG	配置寄存器

### 3.18.2. 外设库函数说明

TIMER库函数列表如下表所示：

**表 3-498. 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更新事件使能
timer_update_event_disable	TIMERx更新事件禁能
timer_counter_alignment	设置外设TIMERx的对齐模式
timer_counter_up_direction	设置外设TIMERx向上计数
timer_counter_down_direction	设置外设TIMERx向下计数
timer_prescaler_config	配置外设TIMERx预分频器
timer_repetition_value_config	配置外设TIMERx的重复计数器
timer_autoreload_value_config	配置外设TIMERx的自动重载寄存器
timer_counter_value_config	配置外设TIMERx的计数器值
timer_counter_read	读取外设TIMERx的计数器值
timer_prescaler_read	读取外设TIMERx的预分频器值
timer_single_pulse_mode_config	配置外设TIMERx的单脉冲模式
timer_update_source_config	配置外设TIMERx的更新源
timer_dma_enable	使能TIMERx的DMA功能
timer_dma_disable	禁能TIMERx的DMA功能
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

库函数名称	库函数描述
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_combined_3_phase_pwm_config	配置PWM组合模式
timer_channel_output_pulse_value_config	配置外设TIMERx的通道输出比较值
timer_channel_output_shadow_config	配置TIMERx通道输出比较影子寄存器功能
timer_channel_output_fast_config	配置TIMERx通道输出比较快速功能
timer_channel_output_clear_config	配置TIMERx的通道输出比较清0功能
timer_channel_output_polarity_config	通道输出极性配置
timer_channel_complementary_output_polarity_config	互补通道输出极性配置
timer_channel_output_state_config	配置通道状态
timer_channel_complementary_output_state_config	配置互补通道输出状态
timer_channel_input_struct_para_init	将TIMER通道输入参数结构体中所有参数初始化为默认值
timer_input_capture_config	配置TIMERx输入捕获参数
timer_channel_input_capture_prescaler_config	配置TIMERx通道输入捕获预分频值
timer_channel_capture_value_register_read	读取通道输入捕获值
timer_input_pwm_capture_config	配置TIMERx捕获PWM输入参数
timer_hall_mode_config	配置TIMERx的HALL接口功能
timer_input_trigger_source_select	TIMERx的输入触发源选择
timer_master_output_trigger_source_select	选择TIMERx主模式输出触发源

库函数名称	库函数描述
timer_slave_mode_select	TIMERx从模式配置
timer_master_slave_mode_config	TIMERx主从模式配置
timer_external_trigger_config	配置TIMERx外部触发输入
timer_quadrature_decoder_mode_config	TIMERx配置为编码器模式
timer_internal_clock_config	TIMERx配置为内部时钟
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	禁能TIMERx外部时钟模式1
timer_input_selection_config	配置TIMERx输入选择
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
timer_break_external_source_config	配置TIMERx中止外部源
timer_break_external_polarity_config	配置TIMERx中止极性
timer_flag_get	外设TIMERx标志位获取
timer_flag_clear	外设TIMERx标志位清除
timer_interrupt_enable	外设TIMERx的中断使能
timer_interrupt_disable	外设TIMERx的中断禁能
timer_interrupt_flag_get	外设TIMERx中断标志获取
timer_interrupt_flag_clear	外设TIMERx中断标志清除

## 结构体 timer\_parameter\_struct

表 3-499. 结构体 timer\_parameter\_struct

成员名称	功能描述
prescaler	预分频值（0~65535）
alignedmode	对齐模式（TIMER_COUNTER_EDGE, TIMER_COUNTER_CENTER_DOWN, TIMER_COUNTER_CENTER_UP, TIMER_COUNTER_CENTER_BOTH）
counterdirection	计数方向（TIMER_COUNTER_UP, TIMER_COUNTER_DOWN）
clockdivision	时钟分频因子（TIMER_CKDIV_DIV1, TIMER_CKDIV_DIV2, TIMER_CKDIV_DIV4）
period	周期（0~65535）
repetitioncounter	重复计数器值（0~255） 只对GD32L235有效

## 结构体 timer\_break\_parameter\_struct

表 3-500. 结构体 timer\_break\_parameter\_struct

成员名称	功能描述
runoffstate	运行模式下“关闭状态”配置（TIMER_ROS_STATE_ENABLE, TIMER_ROS_STATE_DISABLE）
ideloffstate	空闲模式下“关闭状态”配置（TIMER_IOS_STATE_ENABLE, TIMER_IOS_STATE_DISABLE）
deadtime	死区时间（0~255）
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）
break0state	中止使能（TIMER_BREAK0_ENABLE, TIMER_BREAK0_DISABLE）
break0filter	中止输入滤波(0-15)
break0polarity	中止信号1极性（TIMER_BREAK1_POLARITY_LOW, TIMER_BREAK1_POLARITY_HIGH）
break1state	中止1使能（TIMER_BREAK1_ENABLE, TIMER_BREAK1_DISABLE）
break1filter	中止1输入滤波(0-15)
break1polarity	中止信号1极性（TIMER_BREAK1_POLARITY_LOW, TIMER_BREAK1_POLARITY_HIGH）

## 结构体 timer\_oc\_parameter\_struct

表 3-501. 结构体 timer\_oc\_parameter\_struct

成员名称	功能描述
outputstate	通道输出状态（TIMER_CCX_ENABLE, TIMER_CCX_DISABLE）
outputnstate	互补通道输出状态（TIMER_CCXN_ENABLE, TIMER_CCXN_DISABLE） 只对GD32L235有效
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） 只对GD32L235有效
ocidlestate	空闲状态下通道输出（TIMER_OC_IDLE_STATE_LOW, TIMER_OC_IDLE_STATE_HIGH） 只对GD32L235有效
ocnidlestate	空闲状态下互补通道输出（TIMER_OCN_IDLE_STATE_LOW, TIMER_OCN_IDLE_STATE_HIGH） 只对GD32L235有效

结构体 `timer_ic_parameter_struct`表 3-502. 结构体 `timer_ic_parameter_struct`

成员名称	功能描述
icpolarity	通道输入极性 (TIMER_IC_POLARITY_RISING, TIMER_IC_POLARITY_FALLING, TIMER_IC_POLARITY_BOTH_EDGE)
icselection	通道输入模式选择 (TIMER_IC_SELECTION_DIRECTTI, TIMER_IC_SELECTION_INDIRECTTI, TIMER_IC_SELECTION_ITS)
icprescaler	通道输入捕获预分频 (TIMER_IC_PSC_DIV1, TIMER_IC_PSC_DIV2, TIMER_IC_PSC_DIV4, TIMER_IC_PSC_DIV8)
icfilter	通道输入捕获滤波 (0~15)

函数 `timer_deinit`

函数 `timer_deinit` 描述见下表:

表 3-503. 函数 `timer_deinit`

函数名称	<code>timer_deinit</code>
函数原型	<code>void timer_deinit(uint32_t timer_periph);</code>
功能描述	复位外设 <code>TIMERx</code>
先决条件	-
被调用函数	<code>rcu_periph_reset_enable</code> / <code>rcu_periph_reset_disable</code>
输入参数{in}	
<code>timer_periph</code>	<code>TIMER</code> 外设
<code>TIMERx</code>	<code>TIMER</code> 外设选择 ( <code>x=0,2,13,15,16</code> )
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* reset TIMER2 */
timer_deinit(TIMER2);
```

函数 `timer_struct_para_init`

函数 `timer_struct_para_init` 描述见下表:

表 3-504. 函数 `timer_struct_para_init`

函数名称	<code>timer_struct_para_init</code>
函数原型	<code>void timer_struct_para_init(timer_parameter_struct* initpara);</code>
功能描述	将 <code>TIMER</code> 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3-499. 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 3-505. 函数 timer\_init**

函数名称	timer_init
函数原型	void timer_init(uint32_t timer_periph, timer_parameter_struct* initpara);
功能描述	初始化外设TIMERx
先决条件	-
被调用函数	-
输入参数{in}	
timer_periph	TIMER外设
TIMERx	TIMER外设选择 (x=0,2,13,15,16)
输入参数{in}	
initpara	TIMER初始化结构体，结构体成员参考 <a href="#">表3-499. 结构体 timer_parameter_struct</a>
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* initialize TIMER2 */
```

```
timer_parameter_struct timer_initpara;
```

```
timer_initpara.prescaler = 47;
```

```
timer_initpara.alignedmode = TIMER_COUNTER_EDGE;
```

```
timer_initpara.counterdirection = TIMER_COUNTER_UP;
```

```
timer_initpara.period           = 999;
```

```
timer_initpara.clockdivision    = TIMER_CKDIV_DIV1;
```

```
timer_init(TIMER1, &timer_initpara);
```

### 函数 timer\_enable

函数timer\_enable描述见下表:

表 3-506. 函数 timer\_enable

函数名称	timer_enable
函数原型	void timer_enable(uint32_t timer_periph);
功能描述	使能外设TIMERx
先决条件	-
被调用函数	-
输入参数{in}	
timer_periph	TIMER外设
TIMERx	TIMER外设选择 (x=0,2,13,15,16)
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* enable TIMER2 */
```

```
timer_enable(TIMER2);
```

### 函数 timer\_disable

函数timer\_disable描述见下表:

表 3-507. 函数 timer\_disable

函数名称	timer_disable
函数原型	void timer_disable(uint32_t timer_periph);
功能描述	禁能外设TIMERx
先决条件	-
被调用函数	-
输入参数{in}	
timer_periph	TIMER外设
TIMERx	TIMER外设选择 (x=0,2,13,15,16)



输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* disable TIMER2 */
```

```
timer_disable(TIMER2);
```

### 函数 timer\_auto\_reload\_shadow\_enable

函数timer\_auto\_reload\_shadow\_enable描述见下表：

**表 3-508. 函数 timer\_auto\_reload\_shadow\_enable**

函数名称	timer_auto_reload_shadow_enable
函数原型	void timer_auto_reload_shadow_enable(uint32_t timer_periph);
功能描述	TIMERx自动重载影子使能
先决条件	-
被调用函数	-
输入参数{in}	
timer_periph	TIMER外设
TIMERx	TIMER外设选择 (x=0,2,13,15,16)
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* enable the TIMER2 auto reload shadow function */
```

```
timer_auto_reload_shadow_enable(TIMER2);
```

### 函数 timer\_auto\_reload\_shadow\_disable

函数timer\_auto\_reload\_shadow\_disable描述见下表：

**表 3-509. 函数 timer\_auto\_reload\_shadow\_disable**

函数名称	timer_auto_reload_shadow_disable
函数原型	void timer_auto_reload_shadow_disable(uint32_t timer_periph);
功能描述	TIMERx自动重载影子禁能
先决条件	-
被调用函数	-
输入参数{in}	

<b>timer_periph</b>	TIMER外设
<i>TIMERx</i>	TIMER外设选择 (x=0,2,13,15,16)
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* disable the TIMER2 auto reload shadow function */
```

```
timer_auto_reload_shadow_disable(TIMER2);
```

### 函数 timer\_update\_event\_enable

函数timer\_update\_event\_enable描述见下表：

表 3-510. 函数 timer\_update\_event\_enable

函数名称	timer_update_event_enable
函数原型	void timer_update_event_enable(uint32_t timer_periph);
功能描述	TIMERx更新事件使能
先决条件	-
被调用函数	-
输入参数{in}	
<b>timer_periph</b>	TIMER外设
<i>TIMERx</i>	TIMER外设选择 (x=0,2,13,15,16)
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* enable TIMER2 the update event */
```

```
timer_update_event_enable(TIMER2);
```

### 函数 timer\_update\_event\_disable

函数timer\_update\_event\_disable描述见下表：

表 3-511. 函数 timer\_update\_event\_disable

函数名称	timer_update_event_disable
函数原型	void timer_update_event_disable(uint32_t timer_periph);
功能描述	TIMERx更新事件禁能

先决条件	-
被调用函数	-
输入参数{in}	
timer_periph	TIMER外设
TIMERx	TIMER外设选择 (x=0,2,13,15,16)
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* disable TIMER2 the update event */
timer_update_event_disable(TIMER2);
```

### 函数 timer\_counter\_alignment

函数timer\_counter\_alignment描述见下表:

表 3-512. 函数 timer\_counter\_alignment

函数名称	timer_counter_alignment
函数原型	void timer_counter_alignment(uint32_t timer_periph, uint16_t aligned);
功能描述	设置外设TIMERx的对齐模式
先决条件	-
被调用函数	-
输入参数{in}	
timer_periph	TIMER外设
TIMERx	TIMER外设选择 (x=0,2)
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 TIMER2 counter center-aligned and counting up assert mode */
```

```
timer_counter_alignment(TIMER2, TIMER_COUNTER_CENTER_UP);
```

### 函数 timer\_counter\_up\_direction

函数timer\_counter\_up\_direction描述见下表：

表 3-513. 函数 timer\_counter\_up\_direction

函数名称	timer_counter_up_direction
函数原型	void timer_counter_up_direction(uint32_t timer_periph);
功能描述	设置外设TIMERx向上计数
先决条件	计数器设置为无中央对齐计数模式（边沿对齐模式）
被调用函数	-
输入参数{in}	
timer_periph	TIMER外设
TIMERx	TIMER外设选择 (x=0,2)
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* set TIMER1 counter up direction */
```

```
timer_counter_up_direction(TIMER1);
```

### 函数 timer\_counter\_down\_direction

函数timer\_counter\_down\_direction描述见下表：

表 3-514. 函数 timer\_counter\_down\_direction

函数名称	timer_counter_down_direction
函数原型	void timer_counter_down_direction(uint32_t timer_periph);
功能描述	设置外设TIMERx向下计数
先决条件	计数器设置为无中央对齐计数模式（边沿对齐模式）
被调用函数	-
输入参数{in}	
timer_periph	TIMER外设

<i>TIMERx</i>	TIMER外设选择 (x=0,2)
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* set TIMER2 counter down direction */
timer_counter_down_direction(TIMER2);
```

### 函数 timer\_prescaler\_config

函数timer\_prescaler\_config描述见下表:

表 3-515. 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, uint8_t pscreload);
功能描述	配置外设TIMERx预分频器
先决条件	-
被调用函数	-
输入参数{in}	
timer_periph	TIMER外设
<i>TIMERx</i>	TIMER外设选择 (x=0,2,13,15,16)
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 TIMER1 prescaler */
timer_prescaler_config(TIMER2, 3000, TIMER_PSC_RELOAD_NOW);
```

## 函数 timer\_repetition\_value\_config

函数timer\_repetition\_value\_config描述见下表:

表 3-516. 函数 timer\_repetition\_value\_config

函数名称	timer_repetition_value_config
函数原型	void timer_repetition_value_config(uint32_t timer_periph, uint16_t repetition);
功能描述	配置外设TIMERx的重复计数器
先决条件	-
被调用函数	-
输入参数{in}	
timer_periph	TIMER外设
TIMERx	TIMER外设选择 (x=0,15,16)
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 3-517. 函数 timer\_autoreload\_value\_config

函数名称	timer_autoreload_value_config
函数原型	void timer_autoreload_value_config(uint32_t timer_periph, uint16_t autoreload);
功能描述	配置外设TIMERx的自动重载寄存器
先决条件	-
被调用函数	-
输入参数{in}	
timer_periph	TIMER外设
TIMERx	TIMER外设选择 (x=0,2,13,15,16)
输入参数{in}	
autoreload	计数器自动重载值 (0-65535)
输出参数{out}	
-	-
返回值	

-	-
---	---

例如:

```
/* configure TIMER1 autoreload register value */
```

```
timer_autoreload_value_config(TIMER1, 3000);
```

### 函数 timer\_counter\_value\_config

函数timer\_counter\_value\_config描述见下表:

表 3-518. 函数 timer\_counter\_value\_config

函数名称	timer_counter_value_config
函数原型	void timer_counter_value_config(uint32_t timer_periph, uint16_t counter);
功能描述	配置外设TIMERx的计数器值
先决条件	-
被调用函数	-
输入参数{in}	
timer_periph	TIMER外设
TIMERx	TIMER外设选择 (x=0,2,13,15,16)
输入参数{in}	
counter	计数器值 (0-65535)
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* configure TIMER2 counter register value */
```

```
timer_counter_value_config(TIMER2, 3000);
```

### 函数 timer\_counter\_read

函数timer\_counter\_read描述见下表:

表 3-519. 函数 timer\_counter\_read

函数名称	timer_counter_read
函数原型	uint32_t timer_counter_read(uint32_t timer_periph);
功能描述	读取外设TIMERx的计数器值
先决条件	-
被调用函数	-
输入参数{in}	
timer_periph	TIMER外设

<i>TIMERx</i>	TIMER外设选择 (x=0,2,13,15,16)
输出参数{out}	
-	-
返回值	
uint32_t	外设TIMERx的计数器值（0~65535）

例如：

```
/* read TIMER2 counter value */

uint32_t i = 0;

i = timer_counter_read(TIMER2);
```

### 函数 timer\_prescaler\_read

函数timer\_prescaler\_read描述见下表：

表 3-520. 函数 timer\_prescaler\_read

函数名称	timer_prescaler_read
函数原型	uint16_t timer_prescaler_read(uint32_t timer_periph);
功能描述	读取外设TIMERx的预分频器值
先决条件	-
被调用函数	-
输入参数{in}	
timer_periph	TIMER外设
<i>TIMERx</i>	TIMER外设选择 (x=0,2,13,15,16)
输出参数{out}	
-	-
返回值	
uint16_t	外设TIMERx的预分频器值（0~65535）

例如：

```
/* read TIMER2 prescaler value */

uint16_t i = 0;

i = timer_prescaler_read(TIMER2);
```

### 函数 timer\_single\_pulse\_mode\_config

函数timer\_single\_pulse\_mode\_config描述见下表：

表 3-521. 函数 timer\_single\_pulse\_mode\_config

函数名称	timer_single_pulse_mode_config
函数原型	void timer_single_pulse_mode_config(uint32_t timer_periph, uint8_t spmode);



功能描述	配置外设TIMERx的单脉冲模式
先决条件	-
被调用函数	-
输入参数{in}	
timer_periph	TIMER外设
TIMERx	TIMER外设选择 (x=0,2,13,15,16)
输入参数{in}	
spmode	脉冲模式
TIMER_SP_MODE_SINGLE	单脉冲模式计数
TIMER_SP_MODE_REPETITIVE	重复模式计数
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* configure TIMER2 single pulse mode */
```

```
timer_single_pulse_mode_config(TIMER2, TIMER_SP_MODE_SINGLE);
```

### 函数 timer\_update\_source\_config

函数timer\_update\_source\_config描述见下表:

表 3-522. 函数 timer\_update\_source\_config

函数名称	timer_update_source_config
函数原型	void timer_update_source_config(uint32_t timer_periph, uint32_t update);
功能描述	配置外设TIMERx的更新源
先决条件	-
被调用函数	-
输入参数{in}	
timer_periph	TIMER外设
TIMERx	TIMER外设选择 (x=0,2,13,15,16)
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	只有计数器溢出/下溢才产生更新中断或DMA请求

RC_REGULAR	
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* configure TIMER2 update only by counter overflow/underflow */
```

```
timer_update_source_config(TIMER2, TIMER_UPDATE_SRC_REGULAR);
```

### 函数 timer\_dma\_enable

函数timer\_dma\_enable描述见下表:

表 3-523. 函数 timer\_dma\_enable

函数名称	timer_dma_enable
函数原型	void timer_dma_enable(uint32_t timer_periph, uint16_t dma);
功能描述	外设TIMERx的DMA使能
先决条件	-
被调用函数	-
输入参数{in}	
timer_periph	TIMER外设
TIMERx	参考具体参数
输入参数{in}	
dma	DMA源
TIMER_DMA_UPD	更新DMA请求 TIMERx(x=0,2,13,15,16)
TIMER_DMA_CH0 D	通道0比较/捕获DMA请求 TIMERx(x=0,2,13,15,16)
TIMER_DMA_CH1 D	通道1比较/捕获 DMA请求 TIMERx(x=0,2)
TIMER_DMA_CH2 D	通道2比较/捕获DMA请求 TIMERx(x=0,2)
TIMER_DMA_CH3 D	通道3比较/捕获DMA请求 TIMERx(x=0,2)
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=0,2)
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* enable the TIMER2 update DMA */
timer_dma_enable(TIMER2, TIMER_DMA_UPD);
```

### 函数 timer\_dma\_disable

函数timer\_dma\_disable描述见下表:

表 3-524. 函数 timer\_dma\_disable

函数名称	timer_dma_disable
函数原型	void timer_dma_disable(uint32_t timer_periph, uint16_t dma);
功能描述	外设TIMERx的DMA禁能
先决条件	-
被调用函数	-
输入参数{in}	
timer_periph	TIMER外设
TIMERx	参考具体参数
输入参数{in}	
dma	DMA源
TIMER_DMA_UPD	更新DMA请求 TIMERx(x=0,2,13,15,16)
TIMER_DMA_CH0 D	通道0比较/捕获DMA请求 TIMERx(x=0,2,13,15,16)
TIMER_DMA_CH1 D	通道1比较/捕获 DMA请求 TIMERx(x=0,2)
TIMER_DMA_CH2 D	通道2比较/捕获DMA请求 TIMERx(x=0,2)
TIMER_DMA_CH3 D	通道3比较/捕获DMA请求 TIMERx(x=0,2)
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=0,2)
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* disable the TIMER2 update DMA */
timer_dma_disable(TIMER2, 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 3-525. 函数 timer\_channel\_dma\_request\_source\_select

函数名称	timer_channel_dma_request_source_select
函数原型	void timer_channel_dma_request_source_select(uint32_t timer_periph, uint32_t dma_request);
功能描述	外设TIMERx的通道DMA请求源选择
先决条件	-
被调用函数	-
输入参数{in}	
timer_periph	TIMER外设
TIMERx	TIMER外设选择 TIMERx(x=0,2,15,16)
输入参数{in}	
dma_request	通道的DMA请求源选择
TIMER_DMAREQUEST_CHANNELEVENT	当通道捕获/比较事件发生时, 发送通道n的DMA请求
TIMER_DMAREQUEST_UPDATEEVENT	当更新事件发生, 发送通道n的DMA请求
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* TIMER2 channel DMA request of channel n is sent when channel y event occurs */
```

```
timer_channel_dma_request_source_select(TIMER2,  
TIMER_DMAREQUEST_CHANNELEVENT);
```

## 函数 timer\_dma\_transfer\_config

函数timer\_dma\_transfer\_config描述见下表:

表 3-526. 函数 timer\_dma\_transfer\_config

函数名称	timer_dma_transfer_config
函数原型	void timer_dma_transfer_config(uint32_t timer_periph, uint32_t dma_baseaddr, uint32_t dma_lenth);
功能描述	配置外设TIMERx的DMA模式
先决条件	-
被调用函数	-

输入参数{in}	
<b>timer_periph</b>	TIMER外设
<i>TIMERx</i>	定时器外设选择 (x=0,2,15,16)
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>TIMER_DMACFG_DMATA_CTL1</i>	DMA传输起始地址: TIMER_DMACFG_DMATA_CTL1
<i>TIMER_DMACFG_DMATA_SMCFG</i>	DMA传输起始地址: TIMER_DMACFG_DMATA_SMCFG
<i>TIMER_DMACFG_DMATA_DMAINTEN</i>	DMA传输起始地址: TIMER_DMACFG_DMATA_DMAINTEN
<i>TIMER_DMACFG_DMATA_INTF</i>	DMA传输起始地址: TIMER_DMACFG_DMATA_INTF
<i>TIMER_DMACFG_DMATA_SWEVG</i>	DMA传输起始地址: TIMER_DMACFG_DMATA_SWEVG
<i>TIMER_DMACFG_DMATA_CHCTL0</i>	DMA传输起始地址: TIMER_DMACFG_DMATA_CHCTL0
<i>TIMER_DMACFG_DMATA_CHCTL1</i>	DMA传输起始地址: TIMER_DMACFG_DMATA_CHCTL1
<i>TIMER_DMACFG_DMATA_CHCTL2</i>	DMA传输起始地址: TIMER_DMACFG_DMATA_CHCTL2
<i>TIMER_DMACFG_DMATA_CNT</i>	DMA传输起始地址: TIMER_DMACFG_DMATA_CNT
<i>TIMER_DMACFG_DMATA_PSC</i>	DMA传输起始地址: TIMER_DMACFG_DMATA_PSC
<i>TIMER_DMACFG_DMATA_CAR</i>	DMA传输起始地址: TIMER_DMACFG_DMATA_CAR
<i>TIMER_DMACFG_DMATA_CH0CV</i>	DMA传输起始地址: TIMER_DMACFG_DMATA_CH0CV
<i>TIMER_DMACFG_DMATA_CH1CV</i>	DMA传输起始地址: TIMER_DMACFG_DMATA_CH1CV
<i>TIMER_DMACFG_DMATA_CH2CV</i>	DMA传输起始地址: TIMER_DMACFG_DMATA_CH2CV
<i>TIMER_DMACFG_DMATA_CH3CV</i>	DMA传输起始地址: TIMER_DMACFG_DMATA_CH3CV
<i>TIMER_DMACFG_DMATA_CCHP</i>	DMA传输起始地址: TIMER_DMACFG_DMATA_CCHP
<i>TIMER_DMACFG_DMATA_DMACFG</i>	DMA传输起始地址: TIMER_DMACFG_DMATA_DMACFG

输入参数{in}	
<b>dma_lenth</b>	DMA传输长度
<i>TIMER_DMACFG_DMATC_xTRANSFER</i>	x=1..17, DMA传输 x 次
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* configure the TIMER2 DMA transfer */
```

```
timer_dma_transfer_config(TIMER2, 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 3-527. 函数 timer\_event\_software\_generate

<b>函数名称</b>	timer_event_software_generate
<b>函数原型</b>	void timer_event_software_generate(uint32_t timer_periph, uint16_t event);
<b>功能描述</b>	软件产生事件
<b>先决条件</b>	-
<b>被调用函数</b>	-
输入参数{in}	
<b>timer_periph</b>	TIMER外设
<i>TIMERx</i>	参考具体参数
输入参数{in}	
<b>event</b>	事件源
<i>TIMER_EVENT_SRC_UPG</i>	更新事件产生 TIMERx(x=0,2,13,15,16)
<i>TIMER_EVENT_SRC_CH0G</i>	通道0捕获或比较事件发生, TIMERx(x=0,2,13,15,16)
<i>TIMER_EVENT_SRC_CH1G</i>	通道1捕获或比较事件发生, TIMERx(x=0,2)
<i>TIMER_EVENT_SRC_CH2G</i>	通道2捕获或比较事件发生 TIMERx(x=0,2)
<i>TIMER_EVENT_SRC_CH3G</i>	通道3捕获或比较事件发生 TIMERx(x=0,2)
<i>TIMER_EVENT_SRC_CMTG</i>	通道换相更新事件发生 TIMERx(x=0,15,16)
<i>TIMER_EVENT_SRC</i>	触发事件产生

<i>C_TRGG</i>	TIMERx(x=0,15,16)
<i>TIMER_EVENT_SR</i> <i>C_BRK0G</i>	产生中止0事件 TIMERx(x=0,15,16)
<i>TIMER_EVENT_SR</i> <i>C_BRK1G</i>	产生中止1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(TIMER2, TIMER_EVENT_SRC_UPG);
```

### 函数 timer\_break\_struct\_para\_init

函数timer\_break\_struct\_para\_init描述见下表:

表 3-528. 函数 timer\_break\_struct\_para\_init

函数名称	timer_break_struct_para_init
函数原型	void timer_break_struct_para_init(timer_break_parameter_struct* breakpara);
功能描述	将TIMER中止功能参数结构体中所有参数初始化为默认值
先决条件	-
被调用函数	-
输入参数{in}	
breakpara	中止功能配置结构体, 详见 <a href="#">结构体timer_break_parameter_struct</a>
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* initialize TIMER break parameter struct with a default value */
```

```
timer_break_parameter_struct timer_breakpara;
```

```
timer_break_struct_para_init(timer_breakpara);
```

### 函数 timer\_break\_config

函数timer\_break\_config描述见下表:

表 3-529. 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="#">结构体timer_break_parameter_struct</a>
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* configure TIMER0 break function */
timer_break_parameter_struct timer_breakpara;

timer_breakpara.runoffstate      = TIMER_ROS_STATE_DISABLE;
timer_breakpara.ideloffstate     = TIMER_IOS_STATE_DISABLE ;
timer_breakpara.deadtime         = 255;
timer_breakpara.break0polarity   = TIMER_BREAK0_POLARITY_LOW;
timer_breakpara.outputautostate  = TIMER_OUTAUTO_ENABLE;
timer_breakpara.protectmode      = TIMER_CCHP_PROT_0;
timer_breakpara.break0state      = TIMER_BREAK0_ENABLE;
timer_break_config(TIMER0, &timer_breakpara);
```

### 函数 timer\_break\_enable

函数timer\_break\_enable描述见下表:

表 3-530. 函数 timer\_break\_enable

函数名称	timer_break_enable
函数原型	void timer_break_enable(uint32_t timer_periph);
功能描述	使能TIMERx的中止功能
先决条件	只有在TIMERx_CCHP寄存器的PROT [1:0] =00 时, 才可修改
被调用函数	-
输入参数{in}	
timer_periph	TIMER外设



<i>TIMERx</i> ( <i>x</i> =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 3-531. 函数 timer\_break\_disable

函数名称	timer_break_disable
函数原型	void timer_break_disable (uint32_t timer_periph);
功能描述	禁能TIMERx的中止功能
先决条件	只有在TIMERx_CCHP寄存器的PROT [1:0] =00 时, 才可修改
被调用函数	-
输入参数{in}	
timer_periph	TIMER外设
<i>TIMERx</i> <i>TIMERx</i> ( <i>x</i> =0, 15, 16)	TIMER外设选择
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* disable TIMER0 break function*/
```

```
timer_break_disable (TIMER0);
```

### 函数 timer\_automatic\_output\_enable

函数timer\_automatic\_output\_enable描述见下表:

表 3-532. 函数 timer\_automatic\_output\_enable

函数名称	timer_automatic_output_enable
函数原型	void timer_automatic_output_enable(uint32_t timer_periph);
功能描述	自动输出使能

先决条件	只有在TIMERx_CCHP寄存器的PROT [1:0] =00 时，才可修改
被调用函数	-
输入参数{in}	
timer_periph	TIMER外设
TIMERx(x=0, 15, 16)	TIMER外设选择
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* enable TIMER0 output automatic function */
```

```
timer_automatic_output_enable (TIMER0);
```

### 函数 timer\_automatic\_output\_disable

函数timer\_automatic\_output\_disable描述见下表：

表 3-533. 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 3-534. 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 3-535. 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 3-536. 函数 timer\_channel\_control\_shadow\_update\_config**

函数名称	timer_channel_control_shadow_update_config
函数原型	void timer_channel_control_shadow_update_config(uint32_t timer_periph, uint8_t ccuctl);
功能描述	通道换相控制影子寄存器更新控制
先决条件	-
被调用函数	-
输入参数{in}	
timer_periph	TIMER外设
TIMERx(x=0, 15, 16)	TIMER外设选择
输入参数{in}	
ccuctl	通道换相控制影子寄存器更新控制
TIMER_UPDATECTL_CCUC	CMTG位被置1时更新影子寄存器
TIMER_UPDATECTL_CCUTRI	当CMTG位被置1或检测到TRIGI上升沿时，影子寄存器更新
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* configure TIMER0 channel control shadow register update when CMTG bit is set */
```

```
timer_channel_control_shadow_update_config (TIMER0, TIMER_UPDATECTL_CCUC);
```

### 函数 timer\_channel\_output\_struct\_para\_init

函数timer\_channel\_output\_struct\_para\_init描述见下表：

表 3-537. 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}	
ocpara	输出通道结构体, 详见 <a href="#">结构体timer_oc_parameter_struct</a>
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* initialize TIMER channel output parameter struct with a default value */
timer_oc_parameter_struct timer_ocinitpara;
timer_channel_output_struct_para_init(timer_ocinitpara);
```

### 函数 timer\_channel\_output\_config

函数timer\_channel\_output\_config描述见下表:

表 3-538. 函数 timer\_channel\_output\_config

函数名称	timer_channel_output_config
函数原型	void timer_channel_output_config(uint32_t timer_periph, uint16_t channel, timer_oc_parameter_struct* ocpa);
功能描述	外设TIMERx的通道输出配置
先决条件	-
被调用函数	-
输入参数{in}	
timer_periph	TIMER外设
TIMERx	参考具体参数
输入参数{in}	
channel	待配置通道
TIMER_CH_0	通道0 TIMERx(x=0,2,13,15,16)
TIMER_CH_1	通道1 TIMERx(x=0,2)
TIMER_CH_2	通道2 TIMERx(x=0,2)
TIMER_CH_3	通道3

	TIMERx(x=0,2)
<i>TIMER_CH_4</i>	通道4 TIMERx(x=0)
输入参数{in}	
<b>ocpara</b>	输出通道结构体, 详见结构体timer_oc_parameter_struct
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* configure TIMER2 channel 0 output function */

timer_oc_parameter_struct timer_ocintpara;

timer_ocintpara.outputstate = TIMER_CCX_ENABLE;

timer_ocintpara.ocpolarity = TIMER_OC_POLARITY_HIGH;

timer_channel_output_config(TIMER2, 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 3-539. 函数 timer\_channel\_output\_mode\_config

函数名称	timer_channel_output_mode_config
函数原型	void timer_channel_output_mode_config(uint32_t timer_periph, uint16_t channel, uint16_t ocmode);
功能描述	配置外设TIMERx通道输出比较模式
先决条件	-
被调用函数	-
输入参数{in}	
<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=0,2,13,15,16)
<i>TIMER_CH_1</i>	通道1 TIMERx(x=0,2)
<i>TIMER_CH_2</i>	通道2 TIMERx(x=0,2)
<i>TIMER_CH_3</i>	通道3 TIMERx(x=0,2)
<i>TIMER_CH_4</i>	通道4

	TIMERx(x=0)
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
TIMER_OC_MODE _DELAYABLE_SPM 0	延时SPM模式0
TIMER_OC_MODE _DELAYABLE_SPM 1	延时SPM模式1
TIMER_OC_MODE _COMBINED_PWM 0	复合PWM模式0
TIMER_OC_MODE _COMBINED_PWM 1	复合PWM模式1
TIMER_OC_MODE _ASYMMETRIC_P WM0	非对称PWM模式0
TIMER_OC_MODE _ASYMMETRIC_P WM1	非对称PWM模式1
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* configure TIMER2 channel PWM 0 mode */
```

```
timer_channel_output_mode_config(TIMER2, TIMER_CH_0, TIMER_OC_MODE_PWM0);
```

### 函数 timer\_channel\_combined\_3\_phase\_pwm\_config

函数timer\_channel\_combined\_3\_phase\_pwm\_config描述见下表：

**表 3-540. 函数 timer\_channel\_combined\_3\_phase\_pwm\_config**

函数名称	timer_channel_output_mode_config
函数原型	void timer_channel_combined_3_phase_pwm_config(uint32_t timer_periph, uint16_t channel, uint32_t state);
功能描述	配置外设TIMERx通道输出组合模式
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=0,2,13,15,16)
TIMER_CH_1	通道1 TIMERx(x=0,2)
TIMER_CH_2	通道2 TIMERx(x=0,2)
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* configure TIMER0 channel output combine mode */
```

```
timer_channel_combined_3_phase_pwm_config (TIMER0, TIMER_CH_0);
```

### 函数 timer\_channel\_output\_pulse\_value\_config

函数timer\_channel\_output\_pulse\_value\_config描述见下表：

**表 3-541. 函数 timer\_channel\_output\_pulse\_value\_config**

函数名称	timer_channel_output_pulse_value_config
函数原型	void timer_channel_output_pulse_value_config(uint32_t timer_periph, uint16_t channel, uint32_t pulse);
功能描述	配置外设TIMERx的通道输出比较值



先决条件	-
被调用函数	-
输入参数{in}	
timer_periph	TIMER外设
TIMERx	参考具体参数
输入参数{in}	
channel	待配置通道
TIMER_CH_0	通道0 TIMERx(x=0,2,13,15,16)
TIMER_CH_1	通道1 TIMERx(x=0,2)
TIMER_CH_2	通道2 TIMERx(x=0,2)
TIMER_CH_3	通道3 TIMERx(x=0,2)
输入参数{in}	
pulse	通道输出比较值 (0~65535)
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* configure TIMER2 channel 0 output pulse value */
```

```
timer_channel_output_pulse_value_config(TIMER2, TIMER_CH_0, 399);
```

### 函数 timer\_channel\_output\_shadow\_config

函数timer\_channel\_output\_shadow\_config描述见下表:

表 3-542. 函数 timer\_channel\_output\_shadow\_config

函数名称	timer_channel_output_shadow_config
函数原型	void timer_channel_output_shadow_config(uint32_t timer_periph, uint16_t channel, uint16_t ocshadow);
功能描述	配置TIMERx通道输出比较影子寄存器功能
先决条件	-
被调用函数	-
输入参数{in}	
timer_periph	TIMER外设
TIMERx	参考具体参数
输入参数{in}	
channel	待配置通道
TIMER_CH_0	通道0

	TIMERx(x=0,2,13,15,16)
TIMER_CH_1	通道1 TIMERx(x=0,2)
TIMER_CH_2	通道2 TIMERx(x=0,2)
TIMER_CH_3	通道3 TIMERx(x=0,2)
TIMER_CH_4	通道4 TIMERx(x=0)
输入参数{in}	
ocshadow	输出比较影子寄存器功能状态
TIMER_OC_SHADOW_ENABLE	使能输出比较影子寄存器
TIMER_OC_SHADOW_DISABLE	禁能输出比较影子寄存器
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/*configure TIMER2 channel 0 output shadow function */
```

```
timer_channel_output_shadow_config(TIMER2, 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 3-543. 函数 timer\_channel\_output\_fast\_config

函数名称	timer_channel_output_fast_config
函数原型	void timer_channel_output_fast_config(uint32_t timer_periph, uint16_t channel, uint16_t ocfast);
功能描述	配置TIMERx通道输出比较快速功能
先决条件	-
被调用函数	-
输入参数{in}	
timer_periph	TIMER外设
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=0,2,13,15,16)

<i>TIMER_CH_1</i>	通道1 TIMERx(x=0,2)
<i>TIMER_CH_2</i>	通道2 TIMERx(x=0,2)
<i>TIMER_CH_3</i>	通道3 TIMERx(x=0,2)
<i>TIMER_CH_4</i>	通道4 TIMERx(x=0)
输入参数{in}	
<b>ocfast</b>	通道输出比较快速功能状态
<i>TIMER_OC_FAST_ENABLE</i>	通道输出比较快速功能使能
<i>TIMER_OC_FAST_DISABLE</i>	通道输出比较快速功能禁能
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* configure TIMER2 channel 0 output fast function */
```

```
timer_channel_output_fast_config(TIMER2, 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 3-544. 函数 timer\_channel\_output\_clear\_config

函数名称	timer_channel_output_clear_config
函数原型	void timer_channel_output_clear_config(uint32_t timer_periph, uint16_t channel, uint16_t occlear);
功能描述	配置TIMERx的通道输出比较清0功能
先决条件	-
被调用函数	-
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=0,2,13,15,16)
<i>TIMER_CH_1</i>	通道1 TIMERx(x=0,2)

<i>TIMER_CH_2</i>	通道2 TIMERx(x=0,2)
<i>TIMER_CH_3</i>	通道3 TIMERx(x=0,2)
<i>TIMER_CH_4</i>	通道4 TIMERx(x=0)
输入参数{in}	
<b>occlear</b>	通道比较输出清0功能状态
<i>TIMER_OC_CLEAR_ENABLE</i>	通道比较输出清0功能使能
<i>TIMER_OC_CLEAR_DISABLE</i>	通道比较输出清0功能禁能
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* configure TIMER2 channel 0 output clear function */
```

```
timer_channel_output_clear_config(TIMER2, 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 3-545. 函数 timer\_channel\_output\_polarity\_config

函数名称	timer_channel_output_polarity_config
函数原型	void timer_channel_output_polarity_config(uint32_t timer_periph, uint16_t channel, uint16_t ocpolarity);
功能描述	通道输出极性配置
先决条件	-
被调用函数	-
输入参数{in}	
<b>timer_periph</b>	TIMER外设
<i>TIMERx</i>	参考具体参数
输入参数{in}	
<b>channel</b>	待配置通道
<i>TIMER_CH_0</i>	通道0 TIMERx(x=0,2,13,15,16)
<i>TIMER_CH_1</i>	通道1 TIMERx(x=0,2)
<i>TIMER_CH_2</i>	通道2

	TIMERx(x=0,2)
<i>TIMER_CH_3</i>	通道3 TIMERx(x=0,2)
<i>TIMER_CH_4</i>	通道4 TIMERx(x=0)
输入参数{in}	
<b>ocpolarity</b>	通道输出极性
<i>TIMER_OC_POLARITY_HIGH</i>	通道输出极性高电平有效
<i>TIMER_OC_POLARITY_LOW</i>	通道输出极性低电平有效
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* configure TIMER2 channel 0 output polarity */
```

```
timer_channel_output_polarity_config(TIMER2, 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 3-546. 函数 timer\_channel\_complementary\_output\_polarity\_config

函数名称	timer_channel_complementary_output_polarity_config
函数原型	void timer_channel_complementary_output_polarity_config(uint32_t timer_periph, uint16_t channel, uint16_t ocpolarity);
功能描述	互补通道输出极性配置
先决条件	-
被调用函数	-
输入参数{in}	
<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, 14, 40)
<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>ocpolarity</b>	互补通道输出极性
<i>TIMER_OCN_POLA</i>	互补通道输出极性高电平有效

<i>RITY_HIGH</i>	
<i>TIMER_OCN_POLA</i> <i>RITY_LOW</i>	互补通道输出极性低电平有效
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* configure TIMER0 channel 0 complementary output polarity */
```

```
timer_channel_complementary_output_polarity_config (TIMER0, TIMER_CH_0,  
TIMER_OCN_POLARITY_HIGH);
```

### 函数 timer\_channel\_output\_state\_config

函数timer\_channel\_output\_state\_config描述见下表：

表 3-547. 函数 timer\_channel\_output\_state\_config

函数名称	timer_channel_output_state_config
函数原型	void timer_channel_output_state_config(uint32_t timer_periph, uint16_t channel, uint32_t state);
功能描述	配置通道状态
先决条件	-
被调用函数	-
输入参数{in}	
timer_periph	TIMER外设
TIMERx	参考具体参数
输入参数{in}	
channel	待配置通道
TIMER_CH_0	通道0 TIMERx(x=0,2,13,15,16)
TIMER_CH_1	通道1 TIMERx(x=0,2)
TIMER_CH_2	通道2 TIMERx(x=0,2)
TIMER_CH_3	通道3 TIMERx(x=0,2)
TIMER_CH_4	通道4 TIMERx(x=0)
输入参数{in}	
state	通道状态
TIMER_CCX_ENAB LE	通道使能

<i>TIMER_CCX_DISABLE</i>	通道禁能
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* configure TIMER2 channel 0 enable state */
```

```
timer_channel_output_state_config(TIMER2, 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 3-548. 函数 timer\_channel\_complementary\_output\_state\_config**

函数名称	timer_channel_complementary_output_state_config
函数原型	void timer_channel_complementary_output_state_config(uint32_t timer_periph, uint16_t channel, uint16_t ocnstate);
功能描述	配置互补通道输出状态
先决条件	-
被调用函数	-
输入参数{in}	
timer_periph	TIMER外设
<i>TIMERx</i>	参考具体参数
输入参数{in}	
channel	待配置通道
<i>TIMER_CH_0</i>	通道0, <i>TIMERx</i> ( <i>x</i> =0,15,16)
<i>TIMER_CH_1</i>	通道1, <i>TIMERx</i> ( <i>x</i> =0)
<i>TIMER_CH_2</i>	通道2, <i>TIMERx</i> ( <i>x</i> =0)
输入参数{in}	
state	互补通道状态
<i>TIMER_CCXN_ENABLE</i>	互补通道使能
<i>TIMER_CCXN_DISABLE</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 3-549. 函数 timer\_channel\_input\_struct\_para\_init

函数名称	timer_channel_input_struct_para_init
函数原型	void timer_channel_input_struct_para_init(timer_ic_parameter_struct* icpara);
功能描述	将TIMER通道输入参数结构体中所有参数初始化为默认值
先决条件	-
被调用函数	-
输入参数{in}	
icpara	通道输入结构体, 详见 <a href="#">表3-502. 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 3-550. 函数 timer\_input\_capture\_config

函数名称	timer_input_capture_config
函数原型	void timer_input_capture_config(uint32_t timer_periph, uint16_t channel, timer_ic_parameter_struct* icpara);
功能描述	配置TIMERx输入捕获参数
先决条件	-
被调用函数	timer_channel_input_capture_prescaler_config
输入参数{in}	
timer_periph	TIMER外设
TIMERx	参考具体参数
输入参数{in}	
channel	待配置通道
TIMER_CH_0	通道0 TIMERx(x=0,2,13,15,16)



<i>TIMER_CH_1</i>	通道1 TIMERx(x=0,2)
<i>TIMER_CH_2</i>	通道2 TIMERx(x=0,2)
<i>TIMER_CH_3</i>	通道3 TIMERx(x=0,2)
输入参数{in}	
<b>icpara</b>	输入捕获结构体, 详见 <a href="#">表3-502. 结构体timer_ic_parameter_struct</a>
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* configure TIMER2 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(TIMER2, 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 3-551. 函数 timer\_channel\_input\_capture\_prescaler\_config

函数名称	timer_channel_input_capture_prescaler_config
函数原型	void timer_channel_input_capture_prescaler_config(uint32_t timer_periph, uint16_t channel, uint16_t prescaler);
功能描述	配置TIMERx通道输入捕获预分频值
先决条件	-
被调用函数	-
输入参数{in}	
<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=0,2,13,15,16)
<i>TIMER_CH_1</i>	通道1

	TIMERx(x=0,2)
TIMER_CH_2	通道2 TIMERx(x=0,2)
TIMER_CH_3	通道3 TIMERx(x=0,2)
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 TIMER2 channel 0 input capture prescaler value */
```

```
timer_channel_input_capture_prescaler_config(TIMER2, 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 3-552. 函数 timer\_channel\_capture\_value\_register\_read

函数名称	timer_channel_capture_value_register_read
函数原型	uint32_t timer_channel_capture_value_register_read(uint32_t timer_periph, uint16_t channel);
功能描述	读取通道捕获值
先决条件	-
被调用函数	-
输入参数{in}	
timer_periph	TIMER外设
TIMERx	参考具体参数
输入参数{in}	
channel	待配置通道
TIMER_CH_0	通道0 TIMERx(x=0,2,13,15,16)

<i>TIMER_CH_1</i>	通道1 TIMERx(x=0,2)
<i>TIMER_CH_2</i>	通道2 TIMERx(x=0,2)
<i>TIMER_CH_3</i>	通道3 TIMERx(x=0,2)
输出参数{out}	
-	-
返回值	
uint32_t	通道输入捕获值，（0~65535）

例如：

```
/* read TIMER1 channel 0 capture compare register value */
```

```
uint32_t ch0_value = 0;
```

```
ch0_value = timer_channel_capture_value_register_read(TIMER1, TIMER_CH_0);
```

### 函数 timer\_input\_pwm\_capture\_config

函数timer\_input\_pwm\_capture\_config描述见下表：

表 3-553. 函数 timer\_input\_pwm\_capture\_config

函数名称	timer_input_pwm_capture_config
函数原型	void timer_input_pwm_capture_config(uint32_t timer_periph, uint16_t channel, timer_ic_parameter_struct* icpwm);
功能描述	配置TIMERx捕获PWM输入参数
先决条件	-
被调用函数	timer_channel_input_capture_prescaler_config
输入参数{in}	
timer_periph	TIMER外设
TIMERx	TIMER外设选择 TIMERx(x=0,2)
输入参数{in}	
channel	待配置通道
TIMER_CH_0	通道0
TIMER_CH_1	通道1
输入参数{in}	
icpwm	输入捕获结构体，详见 <a href="#">表3-502. 结构体timer_ic_parameter_struct</a>
输出参数{out}	
-	-
返回值	
-	-

例如：

```

/* configure TIMER2 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(TIMER2, TIMER_CH_0, &timer_icinitpara);

```

### 函数 timer\_hall\_mode\_config

函数timer\_hall\_mode\_config描述见下表:

表 3-554. 函数 timer\_hall\_mode\_config

函数名称	timer_hall_mode_config
函数原型	void timer_hall_mode_config(uint32_t timer_periph, uint8_t hallmode);
功能描述	配置TIMERx的HALL接口功能
先决条件	-
被调用函数	-
输入参数{in}	
timer_periph	TIMER外设
TIMERx	TIMER外设选择 TIMERx(x=0,2)
输入参数{in}	
hallmode	HALL接口功能状态
TIMER_HALLINTE RFACE_ENABLE	HALL接口使能
TIMER_HALLINTE RFACE_DISABLE	HALL接口禁能
输出参数{out}	
-	-
返回值	
-	-

例如:

```

/* configure TIMER2 hall sensor mode */

timer_hall_mode_config(TIMER2, TIMER_HALLINTERFACE_ENABLE);

```

### 函数 timer\_input\_trigger\_source\_select

函数timer\_input\_trigger\_source\_select描述见下表:

表 3-555. 函数 timer\_input\_trigger\_source\_select

函数名称	timer_input_trigger_source_select
函数原型	void timer_input_trigger_source_select(uint32_t timer_periph, uint32_t intrigger);
功能描述	TIMERx的输入触发源选择
先决条件	SMC[2:0] = 000
被调用函数	-
输入参数{in}	
timer_periph	TIMER外设
TIMERx	TIMER外设选择
输入参数{in}	
intrigger	待选择的触发源
TIMER_SMCFG_TRGSEL_ITI0	内部触发输入0
TIMER_SMCFG_TRGSEL_ITI2	内部触发输入2
TIMER_SMCFG_TRGSEL_ITI3	内部触发输入3
TIMER_SMCFG_TRGSEL_CIOF_ED	CIO的边沿标志位
TIMER_SMCFG_TRGSEL_CIOFE0	滤波后的通道0输入
TIMER_SMCFG_TRGSEL_CIOFE1	滤波后的通道1输入
TIMER_SMCFG_TRGSEL_ETIFP	滤波后的外部触发输入
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* select TIMER2 input trigger source */
```

```
timer_input_trigger_source_select(TIMER2, 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 3-556. 函数 timer\_master\_output\_trigger\_source\_select

函数名称	timer_master_output_trigger_source_select
函数原型	void timer_master_output_trigger_source_select(uint32_t timer_periph, uint32_t outrigger);

功能描述	选择TIMERx主模式输出触发
先决条件	-
被调用函数	-
输入参数{in}	
timer_periph	TIMER外设
TIMERx	TIMER外设选择
输入参数{in}	
outrigger	主模式输出触发
TIMER_TRI_OUT_SRC_RESET	复位。TIMERx_SWEVG寄存器的UPG位被置1或从模式控制器产生复位触发一次TRGO脉冲，后一种情况下，TRGO上的信号相对实际的复位会有一个延迟。
TIMER_TRI_OUT_SRC_ENABLE	使能。此模式可用于同时启动多个定时器或控制在一段时间内使能从定时器。主模式控制器选择计数器使能信号作为触发输出TRGO。当CEN控制位被置1或者暂停模式下触发输入为高电平时，计数器使能信号被置1。在暂停模式下，计数器使能信号受控于触发输入，在触发输入和TRGO上会有一个延迟，除非选择了主/从模式。
TIMER_TRI_OUT_SRC_UPDATE	更新。主模式控制器选择更新事件作为TRGO。
TIMER_TRI_OUT_SRC_CH0	捕获/比较脉冲。通道0在发生一次捕获或一次比较成功时，主模式控制器产生一个TRGO脉冲
TIMER_TRI_OUT_SRC_O0CPRE	比较。在这种模式下主模式控制器选择O0CPRE信号被用于作为触发输出TRGO
TIMER_TRI_OUT_SRC_O1CPRE	比较。在这种模式下主模式控制器选择O1CPRE信号被用于作为触发输出TRGO
TIMER_TRI_OUT_SRC_O2CPRE	比较。在这种模式下主模式控制器选择O2CPRE信号被用于作为触发输出TRGO
TIMER_TRI_OUT_SRC_O3CPRE	比较。在这种模式下主模式控制器选择O3CPRE信号被用于作为触发输出TRGO
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* select TIMER2 master mode output trigger source */
```

```
timer_master_output_trigger_source_select(TIMER2, TIMER_TRI_OUT_SRC_RESET);
```

### 函数 timer\_slave\_mode\_select

函数timer\_slave\_mode\_select描述见下表：

表 3-557. 函数 timer\_slave\_mode\_select

函数名称	timer_slave_mode_select
函数原型	void timer_slave_mode_select(uint32_t timer_periph, uint32_t slavemode);

功能描述	TIMERx从模式配置
先决条件	-
被调用函数	-
输入参数{in}	
timer_periph	TIMER外设
TIMERx	TIMER外设选择
输入参数{in}	
slavemode	从模式
TIMER_SLAVE_MODE_DISABLE	关闭从模式
TIMER_QUAD_DECODER_MODE0	正交译码器模式0
TIMER_QUAD_DECODER_MODE1	正交译码器模式1
TIMER_QUAD_DECODER_MODE2	正交译码器模式2
TIMER_SLAVE_MODE_RESTART	复位模式
TIMER_SLAVE_MODE_PAUSE	暂停模式
TIMER_SLAVE_MODE_EVENT	事件模式
TIMER_SLAVE_MODE_EXTERNAL0	外部时钟模式0
TIMER_SLAVE_MODE_RESTART_EVENT	复位+暂停模式
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* select TIMER2 slave mode */
```

```
timer_slave_mode_select(TIMER2, TIMER_QUAD_DECODER_MODE0);
```

### 函数 timer\_master\_slave\_mode\_config

函数timer\_master\_slave\_mode\_config描述见下表：

表 3-558. 函数 timer\_master\_slave\_mode\_config

函数名称	timer_master_slave_mode_config
函数原型	void timer_master_slave_mode_config(uint32_t timer_periph, uint8_t

	masterslave);
功能描述	TIMERx主从模式配置
先决条件	-
被调用函数	-
输入参数{in}	
timer_periph	TIMER外设
TIMERx	TIMER外设选择
输入参数{in}	
masterslave	主从模式使能状态
TIMER_MASTER_SLAVE_MODE_ENABLE	主从模式使能
TIMER_MASTER_SLAVE_MODE_DISABLE	主从模式禁能
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* configure TIMER2 master slave mode */
```

```
timer_master_slave_mode_config(TIMER2, TIMER_MASTER_SLAVE_MODE_ENABLE);
```

### 函数 timer\_external\_trigger\_config

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

表 3-559. 函数 timer\_external\_trigger\_config

函数名称	timer_external_trigger_config
函数原型	void timer_external_trigger_config(uint32_t timer_periph, uint32_t extprescaler, uint32_t expolarity, uint32_t extfilter);
功能描述	配置TIMERx外部触发输入
先决条件	-
被调用函数	-
输入参数{in}	
timer_periph	TIMER外设
TIMERx	TIMER外设选择 TIMERx(x=0,2)
输入参数{in}	
extprescaler	外部触发预分频
TIMER_EXT_TRIP_SC_OFF	不分频



<i>TIMER_EXT_TRI_P</i> <i>SC_DIV2</i>	2分频
<i>TIMER_EXT_TRI_P</i> <i>SC_DIV4</i>	4分频
<i>TIMER_EXT_TRI_P</i> <i>SC_DIV8</i>	8分频
输入参数{in}	
<b>expolarity</b>	外部触发输入极性
<i>TIMER_ETP_FALLI</i> <i>NG</i>	低电平或者下降沿有效
<i>TIMER_ETP_RISIN</i> <i>G</i>	高电平或者上升沿有效
输入参数{in}	
<b>extfilter</b>	外部触发滤波控制（0~15）
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* configure TIMER2 external trigger input */
```

```
timer_external_trigger_config(TIMER2,                                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 3-560. 函数 timer\_quadrature\_decoder\_mode\_config

函数名称	timer_quadrature_decoder_mode_config
函数原型	void timer_quadrature_decoder_mode_config(uint32_t timer_periph, uint32_t decomode, uint16_t ic0polarity, uint16_t ic1polarity);
功能描述	TIMERx配置为编码器模式
先决条件	-
被调用函数	-
输入参数{in}	
<b>timer_periph</b>	TIMER外设
<i>TIMERx</i>	TIMER外设选择 TIMERx(x=0,2)
输入参数{in}	
<b>decomode</b>	编码器模式
<i>TIMER_QUAD_DE</i> <i>CODER_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>	捕获双边沿
输入参数{in}	
<b>ic1polarity</b>	IC1极性
<i>TIMER_IC_POLARITY_RISING</i>	捕获上升边沿
<i>TIMER_IC_POLARITY_FALLING</i>	捕获下降边沿
<i>TIMER_IC_POLARITY_BOTH_EDGE</i>	捕获双边沿
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* configure TIMER1 quadrature decoder mode */
```

```
timer_quadrature_decoder_mode_config(TIMER1,    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 3-561. 函数 timer\_internal\_clock\_config

函数名称	timer_internal_clock_config
函数原型	void timer_internal_clock_config(uint32_t timer_periph);
功能描述	TIMERx配置为内部时钟模式
先决条件	-
被调用函数	-
输入参数{in}	
<b>timer_periph</b>	TIMER外设
<i>TIMERx</i>	TIMER外设选择

	TIMERx(x=0,2)
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* configure TIMER0 internal clock mode */
```

```
timer_internal_clock_config (TIMER0);
```

### 函数 timer\_external\_clock\_mode0\_config

函数timer\_external\_clock\_mode0\_config描述见下表:

表 3-562. 函数 timer\_external\_clock\_mode0\_config

函数名称	timer_external_clock_mode0_config
函数原型	void timer_external_clock_mode0_config(uint32_t timer_periph, uint32_t extprescaler, uint32_t expolarity, uint32_t extfilter);
功能描述	配置TIMERx外部时钟模式0，ETI作为时钟源
先决条件	-
被调用函数	timer_external_trigger_config
输入参数{in}	
timer_periph	TIMER外设
TIMERx	TIMER外设选择 TIMERx(x=0,2,13,15,16)
输入参数{in}	
extprescaler	ETI触发源预分频值
TIMER_EXT_TRI_P SC_OFF	不分频
TIMER_EXT_TRI_P SC_DIV2	2分频
TIMER_EXT_TRI_P SC_DIV4	4分频
TIMER_EXT_TRI_P SC_DIV8	8分频
输入参数{in}	
expolarity	ETI触发源极性
TIMER_ETP_FALLI NG	下降沿或者低电平有效
TIMER_ETP_RISIN G	上升沿或者高电平有效
输入参数{in}	
extfilter	ETI触发源滤波参数（0~15）

输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* configure TIMER2 the external clock mode0 */
```

```
timer_external_clock_mode0_config(TIMER2, 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 3-563. 函数 timer\_external\_clock\_mode1\_config

函数名称	timer_external_clock_mode1_config
函数原型	void timer_external_clock_mode1_config(uint32_t timer_periph, uint32_t extprescaler, uint32_t expolarity, uint32_t extfilter);
功能描述	配置TIMERx外部时钟模式1
先决条件	-
被调用函数	timer_external_trigger_config
输入参数{in}	
timer_periph	TIMER外设
TIMERx	TIMER外设选择 TIMERx(x=0,2)
输入参数{in}	
extprescaler	ETI触发源预分频值
TIMER_EXT_TRI_PSC_OFF	不分频
TIMER_EXT_TRI_PSC_DIV2	2分频
TIMER_EXT_TRI_PSC_DIV4	4分频
TIMER_EXT_TRI_PSC_DIV8	8分频
输入参数{in}	
expolarity	ETI触发源极性
TIMER_ETP_FALLING	下降沿或者低电平有效
TIMER_ETP_RISING	上升沿或者高电平有效
输入参数{in}	
extfilter	ETI触发源滤波参数 (0~15)

输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* configure TIMER2 the external clock mode1 */
```

```
timer_external_clock_mode1_config(TIMER2, 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 3-564. 函数 timer\_external\_clock\_mode1\_disable

函数名称	timer_external_clock_mode1_disable
函数原型	void timer_external_clock_mode1_disable(uint32_t timer_periph);
功能描述	TIMERx外部时钟模式1禁能
先决条件	-
被调用函数	-
输入参数{in}	
timer_periph	TIMER外设
TIMERx	TIMER外设选择 TIMERx(x=0,2)
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* disable TIMER2 the external clock mode1 */
```

```
timer_external_clock_mode1_disable(TIMER2);
```

### 函数 timer\_input\_selection\_config

函数timer\_input\_selection\_config描述见下表：

表 3-565. 函数 timer\_channel\_remap\_config

函数名称	timer_input_selection_config
函数原型	void timer_input_selection_config(uint32_t timer_periph, uint16_t channel, uint16_t insel);
功能描述	配置 TIMERx 通道输入选择
先决条件	-

被调用函数	-
输入参数{in}	
timer_periph	TIMER 外设
TIMERx(x=0,2,13,15,16)	TIMER 外设选择
输入参数{in}	
remap	重映射功能选择
TIMER_INSEL_CHx	连接到 CHx
TIMER_INSEL_CMPx	连接到 CMPx
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* configure TIMER0 input selection */
```

```
timer_input_selection_config (TIMER0, TIMER_INSEL_CHx);
```

### 函数 timer\_write\_chxval\_register\_config

函数timer\_write\_chxval\_register\_config描述见下表：

表 3-566. 函数 timer\_write\_chxval\_register\_config

函数名称	timer_write_chxval_register_config
函数原型	void timer_write_chxval_register_config(uint32_t timer_periph, uint16_t ccsel);
功能描述	配置TIMERx写CHxVAL选择位
先决条件	-
被调用函数	-
输入参数{in}	
timer_periph	TIMER外设
TIMERx	TIMER外设选择 TIMERx(x=0,2,13,15,16)
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 TIMER1 write CHxVAL register selection */
timer_write_chxval_register_config(TIMER1, TIMER_CHVSEL_ENABLE);
```

### 函数 timer\_output\_value\_selection\_config

函数timer\_output\_value\_selection\_config描述见下表：

表 3-567. 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输出选择
先决条件	-
被调用函数	-
输入参数{in}	
timer_periph	TIMER外设
TIMERx (x=0,2,13,15,16)	TIMER外设选择
输入参数{in}	
newvalue	控制状态
TIMER_OUTSEL_DISABLE	无效
TIMER_OUTSEL_ENABLE	如果OPEN清0，输出禁能
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\_break\_external\_source\_config

函数timer\_break\_external\_source\_config描述见下表：

表 3-568. 函数 timer\_break\_external\_source\_config

函数名称	timer_break_external_source_config
函数原型	void timer_break_external_source_config(uint32_t timer_periph, uint16_t break_num, ControlStatus newvalue);
功能描述	配置定时器中止源

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}	
break_num	TIMER外设中止源
TIMER_BREAK0	BREAK0作为输入信号
TIMER_BREAK1	BREAK2作为输入信号
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 break0 source */
```

```
timer_break_external_source_config(TIMER0, TIMER_BREAK0, ENABLE);
```

### 函数 timer\_break\_external\_polarity\_config

函数timer\_break\_external\_polarity\_config描述见下表:

表 3-569. 函数 timer\_break\_external\_polarity\_config

函数名称	timer_break_external_polarity_config
函数原型	void timer_break_external_polarity_config(uint32_t timer_periph, uint16_t break_num, uint16_t bkinpolarity);
功能描述	配置定时器中止极性
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}	
break_num	TIMER外设中止源
TIMER_BREAK0	BREAK0作为输入信号
TIMER_BREAK1	BREAK2作为输入信号
输入参数{in}	
bkinpolarity	控制状态
TIMER_BRKIN_PO	中止输入极性低电平



LARITY_LOW	
TIMER_BRKIN_PO LARITY_HIGH	中止输入极性高电平
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* configure TIMER break polarity */
```

```
timer_break_external_polarity_config          (TIMER0,          TIMER_BREAK0,  
TIMER_BRKIN_POLARITY_HIGH);
```

### 函数 timer\_flag\_get

函数timer\_flag\_get描述见下表:

表 3-570. 函数 timer\_flag\_get

函数名称	timer_flag_get
函数原型	FlagStatus timer_flag_get(uint32_t timer_periph, uint32_t flag);
功能描述	获取外设TIMERx的状态标志
先决条件	-
被调用函数	-
输入参数{in}	
timer_periph	TIMER外设
TIMERx	参考具体参数
输入参数{in}	
flag	状态标志
TIMER_FLAG_UP	更新标志
TIMER_FLAG_CH0	通道0比较/捕获标志
TIMER_FLAG_CH1	通道1比较/捕获标志
TIMER_FLAG_CH2	通道2比较/捕获标志
TIMER_FLAG_CH3	通道3比较/捕获标志
TIMER_FLAG_CMT	通道换相更新标志
TIMER_FLAG_TRG	触发标志
TIMER_FLAG_BRK 0	中止0标志位
TIMER_FLAG_BRK 1	中止1标志位
TIMER_FLAG_CH0 0	通道0捕获溢出标志
TIMER_FLAG_CH1 0	通道1捕获溢出标志

<i>TIMER_FLAG_CH2</i> 0	通道2捕获溢出标志
<i>TIMER_FLAG_CH3</i> 0	通道3捕获溢出标志
输出参数{out}	
-	-
返回值	
FlagStatus	SET或者RESET

例如:

```
/* get TIMER2 update flags */
```

```
FlagStatus Flag_status = RESET;
```

```
Flag_status = timer_flag_get(TIMER2, TIMER_FLAG_UP);
```

### 函数 timer\_flag\_clear

函数timer\_flag\_clear描述见下表:

表 3-571. 函数 timer\_flag\_clear

函数名称	timer_flag_clear
函数原型	void timer_flag_clear(uint32_t timer_periph, uint32_t flag);
功能描述	清除外设TIMERx状态标志
先决条件	-
被调用函数	-
输入参数{in}	
timer_periph	TIMER外设
TIMERx	参考具体参数
输入参数{in}	
flag	状态标志
TIMER_FLAG_UP	update flag
TIMER_FLAG_CH0	channel 0 flag
TIMER_FLAG_CH1	channel 1 flag
TIMER_FLAG_CH2	channel 2 flag
TIMER_FLAG_CH3	channel 3 flag
TIMER_FLAG_CMT	channel commutation flag
TIMER_FLAG_TRG	trigger flag
TIMER_FLAG_BRK 0	Break0 flag
TIMER_FLAG_BRK 1	Break1 flag
TIMER_FLAG_CH0 0	channel 0 overcapture flag

<i>TIMER_FLAG_CH1</i> 0	channel 1 overcapture flag
<i>TIMER_FLAG_CH2</i> 0	channel 2 overcapture flag
<i>TIMER_FLAG_CH3</i> 0	channel 3 overcapture flag
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* clear TIMER2 update flags */
```

```
timer_flag_clear(TIMER2 TIMER_FLAG_UP);
```

### 函数 timer\_interrupt\_enable

函数timer\_interrupt\_enable描述见下表:

表 3-572. 函数 timer\_interrupt\_enable

函数名称	timer_interrupt_enable
函数原型	void timer_interrupt_enable(uint32_t timer_periph, uint32_t interrupt);
功能描述	外设TIMERx中断使能
先决条件	-
被调用函数	-
输入参数{in}	
timer_periph	TIMER外设
TIMERx	参考具体参数
输入参数{in}	
interrupt	中断源
TIMER_INT_UP	更新中断
TIMER_INT_CH0	通道0比较/捕获中断
TIMER_INT_CH1	通道1比较/捕获中断
TIMER_INT_CH2	通道2比较/捕获中断
TIMER_INT_CH3	通道3比较/捕获中断
TIMER_INT_CMT	换相更新中断
TIMER_INT_TRG	触发中断
TIMER_INT_BRK	中止中断
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* enable the TIMER2 update interrupt */
timer_interrupt_enable(TIMER2, TIMER_INT_UP);
```

### 函数 timer\_interrupt\_disable

函数timer\_interrupt\_disable描述见下表：

表 3-573. 函数 timer\_interrupt\_disable

函数名称	timer_interrupt_disable
函数原型	void timer_interrupt_disable(uint32_t timer_periph, uint32_t interrupt);
功能描述	外设TIMERx中断禁能
先决条件	-
被调用函数	-
输入参数{in}	
timer_periph	TIMER外设
TIMERx	参考具体参数
输入参数{in}	
interrupt	中断源
TIMER_INT_UP	update interrupt enable
TIMER_INT_CH0	channel 0 interrupt enable
TIMER_INT_CH1	channel 1 interrupt enable
TIMER_INT_CH2	channel 2 interrupt enable
TIMER_INT_CH3	channel 3 interrupt enable
TIMER_INT_CMT	commutation interrupt enable
TIMER_INT_TRG	trigger interrupt enable
TIMER_INT_BRK	break interrupt enable
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* disable the TIMER2 update interrupt */
timer_interrupt_disable(TIMER2, TIMER_INT_UP);
```

### 函数 timer\_interrupt\_flag\_get

函数timer\_interrupt\_flag\_get描述见下表：

表 3-574. 函数 timer\_interrupt\_flag\_get

函数名称	timer_interrupt_flag_get
函数原型	FlagStatus timer_interrupt_flag_get(uint32_t timer_periph, uint32_t interrupt);

功能描述	获取外设TIMERx中断标志
先决条件	-
被调用函数	-
输入参数{in}	
timer_periph	TIMER外设
TIMERx	参考具体参数
输入参数{in}	
interrupt	中断源
TIMER_INT_FLAG_UP	更新中断
TIMER_INT_FLAG_CH0	通道0比较/捕获中断
TIMER_INT_FLAG_CH1	通道1比较/捕获中断
TIMER_INT_FLAG_CH2	通道2比较/捕获中断
TIMER_INT_FLAG_CH3	通道3比较/捕获中断
TIMER_INT_FLAG_CMT	换相更新中断
TIMER_INT_FLAG_TRG	触发中断
TIMER_INT_FLAG_BRK0	中止0中断
TIMER_INT_FLAG_BRK1	中止1中断
TIMER_INT_FLAG_SYSB	触发中断
输出参数{out}	
-	-
返回值	
FlagStatus	SET或者RESET

例如：

```
/* get TIMER2 update interrupt flag */
```

```
FlagStatus Flag_interrupt = RESET;
```

```
Flag_interrupt = timer_interrupt_flag_get(TIMER2, TIMER_INT_FLAG_UP);
```

### 函数 timer\_interrupt\_flag\_clear

函数timer\_interrupt\_flag\_clear描述见下表：

表 3-575. 函数 timer\_interrupt\_flag\_clear

函数名称	timer_interrupt_flag_clear
函数原型	void timer_interrupt_flag_clear(uint32_t timer_periph, uint32_t interrupt);
功能描述	清除外设TIMERx的中断标志
先决条件	-
被调用函数	-
输入参数{in}	
timer_periph	TIMER外设
TIMERx	参考具体参数
输入参数{in}	
interrupt	中断源
TIMER_INT_FLAG_UP	更新中断
TIMER_INT_FLAG_CH0	通道0比较/捕获中断
TIMER_INT_FLAG_CH1	通道1比较/捕获中断
TIMER_INT_FLAG_CH2	通道2比较/捕获中断
TIMER_INT_FLAG_CH3	通道3比较/捕获中断
TIMER_INT_FLAG_CMT	换相更新中断
TIMER_INT_FLAG_TRG	触发中断
TIMER_INT_FLAG_BRK0	中止0中断
TIMER_INT_FLAG_BRK1	中止1中断
TIMER_INT_FLAG_SYSB	触发中断
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* clear TIMER2 update interrupt flag */
```

```
timer_interrupt_flag_clear(TIMER2 TIMER_INT_FLAG_UP);
```

### 3.19. 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[3.19.1](#)描述了USART的寄存器列表，章节[3.19.2](#)对USART库函数进行说明。

#### 3.19.1. 外设寄存器说明

USART寄存器列表如下表所示：

**表 3-576. USART 寄存器**

寄存器名称	寄存器描述
USART_CTL0	控制寄存器0
USART_CTL1	控制寄存器1
USART_CTL2	控制寄存器2
USART_BAUD	波特率寄存器
USART_GP	保护时间和预分频器寄存器
USART_RT	接收超时寄存器
USART_CMD	请求寄存器
USART_STAT	状态寄存器
USART_INTC	中断标志清除寄存器
USART_RDATA	数据接收寄存器
USART_TDATA	数据发送寄存器
USART_CHC	兼容性控制寄存器
USART_RFCS	接收FIFO控制和状态寄存器

#### 3.19.2. 外设库函数说明

USART库函数列表如下表所示：

**表 3-577. USART 库函数**

库函数名称	库函数描述
usart_deinit	复位外设USART
usart_baudrate_set	配置USART波特率
usart_parity_config	配置USART奇偶校验
usart_word_length_set	配置USART字长
usart_stop_bit_set	配置USART停止位
usart_enable	使能USART
usart_disable	失能USART
usart_transmit_config	USART发送配置
usart_receive_config	USART接收配置
usart_data_first_config	配置数据传输时低位在前或高位在前
usart_invert_config	配置USART反转功能
usart_overrun_enable	使能USART溢出禁止功能

库函数名称	库函数描述
usart_overrun_disable	失能USART溢出禁止功能
usart_oversample_config	配置USART过采样模式
usart_sample_bit_config	配置USART单次采样方式
usart_receiver_timeout_enable	使能USART接收超时
usart_receiver_timeout_disable	失能USART接收超时
usart_receiver_timeout_threshold_config	设置USART接收超时阈值
usart_data_transmit	USART发送数据功能
usart_data_receive	USART接收数据功能
usart_command_enable	使能USART请求
usart_address_config	配置USART地址
usart_address_detection_mode_config	配置USART地址检测模式
usart_mute_mode_enable	使能USART静默模式
usart_mute_mode_disable	失能USART静默模式
usart_mute_mode_wakeup_config	配置USART静默模式唤醒方式
usart_lin_mode_enable	使能USART LIN模式
usart_lin_mode_disable	失能USART LIN模式
usart_lin_break_detection_length_config	配置USART LIN模式中断帧长度
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
usart_clock_disable	失能USART CK引脚
usart_synchronous_clock_config	配置USART同步通讯模式参数
usart_guard_time_config	在USART智能卡模式下配置保护时间值
usart_smartcard_mode_enable	使能USART智能卡模式
usart_smartcard_mode_disable	失能USART智能卡模式
usart_smartcard_mode_nack_enable	在USART智能卡模式下使能NACK
usart_smartcard_mode_nack_disable	在USART智能卡模式下失能NACK
usart_smartcard_mode_early_nack_enable	使能USART智能卡模式提前NACK
usart_smartcard_mode_early_nack_disable	失能USART智能卡模式提前NACK
usart_smartcard_autoretry_config	配置智能卡自动重试次数
usart_block_length_config	配置智能卡T=1的接收时块的长度
usart_irda_mode_enable	使能USART串行红外编解码功能模块
usart_irda_mode_disable	失能USART串行红外编解码功能模块
usart_prescaler_config	在USART IrDA低功耗模式下配置外设时钟分频系数
usart_irda_lowpower_config	配置USART IrDA低功耗模式
usart_hardware_flow_rts_config	配置USART RTS硬件控制流



库函数名称	库函数描述
usart_hardware_flow_cts_config	配置USART CTS硬件控制流
usart_hardware_flow_coherence_config	配置硬件流控兼容模式
usart_rs485_driver_enable	使能USART rs485驱动
usart_rs485_driver_disable	失能USART rs485驱动
usart_driver_asserttime_config	配置USART驱动使能置位时间
usart_driver_deasserttime_config	配置USART驱动使能置低时间
usart_depolarity_config	配置USART驱动使能极性模式
usart_dma_receive_config	配置USART DMA接收
usart_dma_transmit_config	配置USART DMA发送
usart_reception_error_dma_disable	USART接收错误时禁能DMA
usart_reception_error_dma_enable	USART接收错误时使能DMA
usart_wakeup_enable	使能USART唤醒
usart_wakeup_disable	失能USART唤醒
usart_wakeup_mode_config	配置USART唤醒模式
usart_receive_fifo_enable	使能接收FIFO
usart_receive_fifo_disable	失能接收FIFO
usart_receive_fifo_counter_number	读取接收FIFO计数器的值
usart_flag_get	得到STAT/RFCFS寄存器中的标志
usart_flag_clear	清除USART状态
usart_interrupt_enable	使能USART中断
usart_interrupt_disable	失能USART中断
usart_interrupt_flag_get	得到USART中断和标志状态
usart_interrupt_flag_clear	清除USART中断标志位

### 枚举类型 usart\_flag\_enum

表 3-578. 枚举类型 usart\_flag\_enum

成员名称	功能描述
USART_FLAG_REA	接收使能通知标志
USART_FLAG_TEA	发送使能通知标志
USART_FLAG_WU	从深度睡眠模式唤醒标志
USART_FLAG_RWU	接收器从静默模式唤醒
USART_FLAG_SB	断开信号发送标志
USART_FLAG_AM	地址匹配标志
USART_FLAG_BSY	忙标志
USART_FLAG_EB	块结束标志
USART_FLAG_RT	接收超时标志
USART_FLAG_CTS	CTS电平
USART_FLAG_CTSF	CTS变化标志
USART_FLAG_LBD	LIN断开检测标志
USART_FLAG_TBE	发送数据寄存器空

成员名称	功能描述
USART_FLAG_TC	发送完成
USART_FLAG_RBNE	读数据缓冲区非空
USART_FLAG_IDLE	空闲线检测标志
USART_FLAG_ORERR	溢出错误
USART_FLAG_NERR	噪声错误标志
USART_FLAG_FERR	帧错误
USART_FLAG_PERR	校验错误
USART_FLAG_EPERR	校验错误超前检测标志
USART_FLAG_RFFINT	接收FIFO满中断标志
USART_FLAG_RFF	接收FIFO满标志
USART_FLAG_RFE	接收FIFO空标志

### 枚举类型 `usart_interrupt_flag_enum`

表 3-579. 枚举类型 `usart_interrupt_flag_enum`

成员名称	功能描述
USART_INT_FLAG_EB	块结束中断标志
USART_INT_FLAG_RT	接收超时中断标志
USART_INT_FLAG_AM	地址匹配中断标志
USART_INT_FLAG_PERR	奇偶校验错误中断标志
USART_INT_FLAG_TBE	发送寄存器空中断标志
USART_INT_FLAG_TC	发送完成中断标志
USART_INT_FLAG_RBNE	读缓冲区非空中断标志
USART_INT_FLAG_RBNE_ORERR	读缓冲区非空和溢出中断标志
USART_INT_FLAG_IDLE	空闲线检测中断标志
USART_INT_FLAG_LBD	LIN断开检测中断标志
USART_INT_FLAG_WU	从深度睡眠模式唤醒中断标志
USART_INT_FLAG_CTS	CTS中断标志
USART_INT_FLAG_ERR_NERR	噪声错误中断标志
USART_INT_FLAG_ERR_ORERR	溢出错误中断标志
USART_INT_FLAG_ERR_FERR	帧错误中断标志
USART_INT_FLAG_RFF	接收FIFO满中断标志

### 枚举类型 `usart_interrupt_enum`

表 3-580. 枚举类型 `usart_interrupt_enum`

成员名称	功能描述
USART_INT_EB	块结束中断使能
USART_INT_RT	接收超时中断使能
USART_INT_AM	地址匹配中断使能

成员名称	功能描述
USART_INT_PERR	奇偶校验错误中断使能
USART_INT_TBE	发送寄存器空中断使能
USART_INT_TC	发送完成中断使能
USART_INT_RBNE	读缓冲区非空中断和溢出错误中断使能
USART_INT_IDLE	空闲线检测中断使能
USART_INT_LBD	LIN断开检测中断使能
USART_INT_WU	从深度睡眠模式唤醒中断使能
USART_INT_CTS	CTS中断使能
USART_INT_ERR	错误中断使能
USART_INT_RFF	接收FIFO满中断使能

### 枚举类型 `usart_invert_enum`

表 3-581. 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 3-582. 函数 `usart_deinit`

函数名称	<code>usart_deinit</code>
函数原型	<code>void usart_deinit(uint32_t usart_periph);</code>
功能描述	复位外设USARTx
先决条件	-
被调用函数	<code>rcu_periph_reset_enable</code> / <code>rcu_periph_reset_disable</code>
输入参数{in}	
<b>usart_periph</b>	外设USARTx
USARTx	x=0,1,2
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* reset USART0 */
```

```
usart_deinit(USART0);
```

### 函数 usart\_baudrate\_set

函数usart\_baudrate\_set描述见下表：

表 3-583. 函数 usart\_baudrate\_set

函数名称	usart_baudrate_set
函数原型	void usart_baudrate_set(uint32_t usart_periph, uint32_t baudval);
功能描述	配置USART波特率
先决条件	-
被调用函数	rcu_clock_freq_get
输入参数{in}	
usart_periph	外设USARTx
USARTx	x=0,1,2
输入参数{in}	
baudval	波特率值
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* configure USART0 baud rate value */
```

```
usart_baudrate_set(USART0, 115200);
```

### 函数 usart\_parity\_config

函数usart\_parity\_config描述见下表：

表 3-584. 函数 usart\_parity\_config

函数名称	usart_parity_config
函数原型	void usart_parity_config(uint32_t usart_periph, uint32_t paritycfg);
功能描述	配置USART奇偶校验
先决条件	-
被调用函数	-
输入参数{in}	
usart_periph	外设USARTx
USARTx	x=0,1,2
输入参数{in}	
paritycfg	配置USART奇偶校验
USART_PM_NONE	无校验

USART_PM_ODD	奇校验
USART_PM_EVEN	偶校验
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* configure USART0 parity */
```

```
usart_parity_config(USART0, USART_PM_EVEN);
```

### 函数 usart\_word\_length\_set

函数usart\_word\_length\_set描述见下表：

表 3-585. 函数 usart\_word\_length\_set

函数名称	usart_word_length_set
函数原型	void usart_word_length_set(uint32_t usart_periph, uint32_t wlen);
功能描述	配置USART字长
先决条件	-
被调用函数	-
输入参数{in}	
usart_periph	外设USARTx
USARTx	x=0,1,2
输入参数{in}	
wlen	配置USART字长
USART_WL_8BIT	8 bits
USART_WL_9BIT	9 bits
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* configure USART0 word length */
```

```
usart_word_length_set(USART0, USART_WL_9BIT);
```

### 函数 usart\_stop\_bit\_set

函数usart\_stop\_bit\_set描述见下表：

表 3-586. 函数 usart\_stop\_bit\_set

函数名称	usart_stop_bit_set
------	--------------------

函数原型	void usart_stop_bit_set(uint32_t usart_periph, uint32_t stblen);
功能描述	配置USART停止位
先决条件	-
被调用函数	-
输入参数{in}	
usart_periph	外设USARTx
USARTx	x=0,1,2
输入参数{in}	
stblen	配置USART停止位
USART_STB_1BIT	1 bit
USART_STB_0_5BIT	0.5 bit
USART_STB_2BIT	2 bit
USART_STB_1_5BIT	1.5 bit
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 3-587. 函数 usart\_enable

函数名称	usart_enable
函数原型	void usart_enable(uint32_t usart_periph);
功能描述	使能USART
先决条件	-
被调用函数	-
输入参数{in}	
usart_periph	外设USARTx
USARTx	x=0,1,2
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* enable USART0 */
```

```
usart_enable(USART0);
```

### 函数 usart\_disable

函数usart\_disable描述见下表：

**表 3-588. 函数 usart\_disable**

函数名称	usart_disable
函数原型	void usart_disable(uint32_t usart_periph);
功能描述	失能USART
先决条件	-
被调用函数	-
输入参数{in}	
usart_periph	外设USARTx
USARTx	x=0,1,2
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* disable USART0 */
```

```
usart_disable(USART0);
```

### 函数 usart\_transmit\_config

函数usart\_transmit\_config描述见下表：

**表 3-589. 函数 usart\_transmit\_config**

函数名称	usart_transmit_config
函数原型	void usart_transmit_config(uint32_t usart_periph, uint32_t txconfig);
功能描述	USART发送器配置
先决条件	-
被调用函数	-
输入参数{in}	
usart_periph	外设USARTx
USARTx	x=0,1,2
输入参数{in}	
txconfig	使能/失能USART发送器
USART_TRANSMIT_ENABLE	使能USART发送
USART_TRANSMIT_DISABLE	失能USART发送

<code>_DISABLE</code>	
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* configure USART0 transmitter */

usart_transmit_config(USART0,USART_TRANSMIT_ENABLE);
```

### 函数 usart\_receive\_config

函数usart\_receive\_config描述见下表:

表 3-590. 函数 usart\_receive\_config

函数名称	usart_receive_config
函数原型	void usart_receive_config(uint32_t usart_periph, uint32_t rxconfig);
功能描述	USART接收器配置
先决条件	-
被调用函数	-
输入参数{in}	
usart_periph	外设USARTx
USARTx	x=0,1,2
输入参数{in}	
rxconfig	使能/失能USART接收器
USART_RECEIVE_ENABLE	使能USART接收
USART_RECEIVE_DISABLE	失能USART接收
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* configure USART0 receiver */

usart_receive_config(USART0, USART_RECEIVE_ENABLE);
```

### 函数 usart\_data\_first\_config

函数usart\_data\_first\_config描述见下表:



表 3-591. 函数 `usart_data_first_config`

函数名称	<code>usart_data_first_config</code>
函数原型	<code>void usart_data_first_config(uint32_t usart_periph, uint32_t msbf);</code>
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}	
<code>usart_periph</code>	外设USARTx
<code>USARTx</code>	x=0,1,2
输入参数{in}	
<code>msbf</code>	数据传输时低位在前/高位在前
<code>USART_MSBF_LSB</code> <code>B</code>	数据传输时低位在前
<code>USART_MSBF_MS</code> <code>B</code>	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 3-592. 函数 `usart_invert_config`

函数名称	<code>usart_invert_config</code>
函数原型	<code>void usart_invert_config(uint32_t usart_periph, usart_invert_enum invertpara);</code>
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}	
<code>usart_periph</code>	外设USARTx
<code>USARTx</code>	x=0,1,2
输入参数{in}	
<code>invertpara</code>	参考 <a href="#">表3-581. 枚举类型usart_invert_enum</a>
<code>USART_DINV_ENA</code> <code>BLE</code>	数据位电平反转
<code>USART_DINV_DIS</code> <code>ABLE</code>	数据位电平不反转

USART_TXPIN_ENABLE	TX引脚电平反转
USART_TXPIN_DISABLE	TX引脚电平不反转
USART_RXPIN_ENABLE	RX引脚电平反转
USART_RXPIN_DISABLE	RX引脚电平不反转
USART_SWAP_ENABLE	TX和RX管脚功能被交换
USART_SWAP_DISABLE	TX和RX管脚功能不被交换
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* configure USART0 inversion */
usart_invert_config(USART0, USART_DINV_ENABLE);
```

### 函数 usart\_overrun\_enable

函数usart\_overrun\_enable描述见下表:

表 3-593. 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USARTx
USARTx	x=0,1,2
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* enable USART0 overrun */
usart_overrun_enable(USART0);
```

**函数 usart\_oversample\_disable**

函数usart\_oversample\_disable描述见下表:

**表 3-594. 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USARTx
USARTx	x=0,1,2
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* disable USART0 overrun */
usart_oversample_disable(USART0);
```

**函数 usart\_oversample\_config**

函数usart\_oversample\_config描述见下表:

**表 3-595. 函数 usart\_oversample\_config**

函数名称	usart_oversample_config
函数原型	void usart_oversample_config(uint32_t usart_periph, uint32_t oversamp);
功能描述	配置USART过采样模式
先决条件	-
被调用函数	-
输入参数{in}	
usart_periph	外设USARTx
USARTx	x=0,1,2
输入参数{in}	
oversamp	过采样值
USART_OVSMOD_8	8倍过采样
USART_OVSMOD_16	16倍过采样
输出参数{out}	
-	-
返回值	

-	-
---	---

例如：

```
/* config USART0 oversampling by 8 */
```

```
usart_oversample_config(USART0, USART_OVSMOD_8);
```

### 函数 usart\_sample\_bit\_config

函数usart\_sample\_bit\_config描述见下表：

表 3-596. 函数 usart\_sample\_bit\_config

函数名称	usart_sample_bit_config
函数原型	void usart_sample_bit_config(uint32_t usart_periph, uint32_t osb);
功能描述	配置USART单次采样方式
先决条件	-
被调用函数	-
输入参数{in}	
usart_periph	外设USARTx
USARTx	x=0,1,2
输入参数{in}	
osb	单次采样方式
USART_OSB_1BIT	1次采样方法
USART_OSB_3BIT	3次采样方法
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* config USART0 1 bit sample mode */
```

```
usart_sample_bit_config(USART0, USART_OSB_1BIT);
```

### 函数 usart\_receiver\_timeout\_enable

函数usart\_receiver\_timeout\_enable描述见下表：

表 3-597. 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}	

<b>usart_periph</b>	外设USARTx
<i>USARTx</i>	x=0
<b>输出参数{out}</b>	
-	-
<b>返回值</b>	
-	-

例如：

```
/* enable USART0 receiver timeout */
```

```
usart_receiver_timeout_enable(USART0);
```

### 函数 usart\_receiver\_timeout\_disable

函数usart\_receiver\_timeout\_disable描述见下表：

**表 3-598. 函数 usart\_receiver\_timeout\_disable**

<b>函数名称</b>	usart_receiver_timeout_disable
<b>函数原型</b>	void usart_receiver_timeout_disable(uint32_t usart_periph);
<b>功能描述</b>	失能USART接收超时
<b>先决条件</b>	-
<b>被调用函数</b>	-
<b>输入参数{in}</b>	
<b>usart_periph</b>	外设USARTx
<i>USARTx</i>	x=0
<b>输出参数{out}</b>	
-	-
<b>返回值</b>	
-	-

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 3-599. 函数 usart\_receiver\_timeout\_threshold\_config**

<b>函数名称</b>	usart_receiver_timeout_threshold_config
<b>函数原型</b>	void usart_receiver_timeout_threshold_config(uint32_t usart_periph, uint32_t rtimeout);
<b>功能描述</b>	设置USART接收超时阈值
<b>先决条件</b>	-

被调用函数	-
输入参数{in}	
usart_periph	外设USARTx
USARTx	x=0
输入参数{in}	
rtimeout	超时时间（0x00000000-0x00FFFFFF）
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* set the receiver timeout threshold of USART0*/
```

```
usart_receiver_timeout_threshold_config(USART0, 115200*3);
```

### 函数 usart\_data\_transmit

函数usart\_data\_transmit描述见下表：

表 3-600. 函数 usart\_data\_transmit

函数名称	usart_data_transmit
函数原型	void usart_data_transmit(uint32_t usart_periph, uint16_t data);
功能描述	USART发送数据功能
先决条件	-
被调用函数	-
输入参数{in}	
usart_periph	外设USARTx
USARTx	x=0,1,2
输入参数{in}	
data	发送的数据（0x00-0x1FF）
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 3-601. 函数 `usart_data_receive`

函数名称	<code>usart_data_receive</code>
函数原型	<code>uint16_t usart_data_receive(uint32_t usart_periph);</code>
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}	
<code>usart_periph</code>	外设USARTx
<code>USARTx</code>	x=0,1,2
输出参数{out}	
-	-
返回值	
<code>uint16_t</code>	接收到的数据（0x00-0x1FF）

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 3-602. 函数 `usart_command_enable`

函数名称	<code>usart_command_enable</code>
函数原型	<code>void usart_command_enable(uint32_t usart_periph, uint32_t cmdtype);</code>
功能描述	使能USART请求
先决条件	-
被调用函数	-
输入参数{in}	
<code>usart_periph</code>	外设USARTx
<code>USARTx</code>	x=0,1,2
输入参数{in}	
<code>cmdtype</code>	请求类型
<code>USART_CMD_SBK_CMD</code>	发送断开帧请求
<code>USART_CMD_MM_CMD</code>	静模式请求
<code>USART_CMD_RXF_CMD</code>	接收数据清空请求
<code>USART_CMD_TXF_CMD</code>	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 3-603. 函数 usart\_address\_config

函数名称	usart_address_config
函数原型	void usart_address_config(uint32_t usart_periph, uint8_t addr);
功能描述	配置USART地址
先决条件	-
被调用函数	-
输入参数{in}	
usart_periph	外设USARTx
USARTx	x=0,1,2
输入参数{in}	
addr	USART地址（0x00-0xFF）
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* configure address of the USART0 */
```

```
usart_address_config(USART0, 0x00);
```

### 函数 usart\_address\_detection\_mode\_config

函数usart\_address\_detection\_mode\_config描述见下表：

表 3-604. 函数 usart\_address\_detection\_mode\_config

函数名称	usart_address_detection_mode_config
函数原型	void usart_address_detection_mode_config(uint32_t usart_periph, uint32_t addmod);
功能描述	配置USART地址检测模式
先决条件	-



被调用函数	-
输入参数{in}	
usart_periph	外设USARTx
USARTx	x=0,1,2
输入参数{in}	
addmod	地址检测模式
USART_ADDDM_4BIT	4位地址检测
USART_ADDDM_FULLBIT	全位地址检测
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/*configure address detection mode */
```

```
usart_address_config(USART0, USART_ADDDM_4BIT);
```

### 函数 usart\_mute\_mode\_enable

函数usart\_mute\_mode\_enable描述见下表：

表 3-605. 函数 usart\_mute\_mode\_enable

函数名称	usart_mute_mode_enable
函数原型	void usart_mute_mode_enable(uint32_t usart_periph);
功能描述	使能USART静默模式
先决条件	-
被调用函数	-
输入参数{in}	
usart_periph	外设USARTx
USARTx	x=0,1,2
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* enable USART0 receiver in mute mode */
```

```
usart_mute_mode_enable(USART0);
```

**函数 usart\_mute\_mode\_disable**

函数usart\_mute\_mode\_disable描述见下表:

**表 3-606. 函数 usart\_mute\_mode\_disable**

函数名称	usart_mute_mode_disable
函数原型	void usart_mute_mode_disable(uint32_t usart_periph);
功能描述	失能USART静默模式
先决条件	-
被调用函数	-
输入参数{in}	
usart_periph	外设USARTx
USARTx	x=0,1,2
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* disable USART0 receiver in mute mode */
```

```
usart_mute_mode_disable(USART0);
```

**函数 usart\_mute\_mode\_wakeup\_config**

函数usart\_mute\_mode\_wakeup\_config描述见下表:

**表 3-607. 函数 usart\_mute\_mode\_wakeup\_config**

函数名称	usart_mute_mode_wakeup_config
函数原型	void usart_mute_mode_wakeup_config(uint32_t usart_periph, uint32_t wmethod);
功能描述	配置USART静默模式唤醒方式
先决条件	-
被调用函数	-
输入参数{in}	
usart_periph	外设USARTx
USARTx	x=0,1,2
输入参数{in}	
wmethod	两种方法用于进入或退出静默模式
USART_WM_IDLE	空闲线唤醒
USART_WM_ADDR	地址匹配唤醒
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* configure USART0 wakeup method in mute mode */  
  
usart_mute_mode_wakeup_config(USART0, USART_WM_IDLE);
```

### 函数 usart\_lin\_mode\_enable

函数usart\_lin\_mode\_enable描述见下表:

表 3-608. 函数 usart\_lin\_mode\_enable

函数名称	usart_lin_mode_enable
函数原型	void usart_lin_mode_enable(uint32_t usart_periph);
功能描述	使能USART LIN模式
先决条件	-
被调用函数	-
输入参数{in}	
usart_periph	外设USARTx
USARTx	x=0
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 3-609. 函数 usart\_lin\_mode\_disable

函数名称	usart_lin_mode_disable
函数原型	void usart_lin_mode_disable(uint32_t usart_periph);
功能描述	失能USART LIN模式
先决条件	-
被调用函数	-
输入参数{in}	
usart_periph	外设USARTx
USARTx	x=0
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 3-610. 函数 usart\_lin\_break\_dection\_length\_config

函数名称	usart_lin_break_dection_length_config
函数原型	void usart_lin_break_dection_length_config(uint32_t usart_periph, uint32_t lblen);
功能描述	配置USART LIN模式中断帧长度
先决条件	-
被调用函数	-
输入参数{in}	
usart_periph	外设USARTx
USARTx	x=0
输入参数{in}	
lblen	LIN模式中断帧长度
USART_LBLEN_10 B	断开帧长度为10 bits
USART_LBLEN_11 B	断开帧长度为11 bits
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* configure LIN break frame length */
```

```
usart_lin_break_dection_length_config(USART0, USART_LBLEN_10B);
```

### 函数 usart\_halfduplex\_enable

函数usart\_halfduplex\_enable描述见下表：

表 3-611. 函数 usart\_halfduplex\_enable

函数名称	usart_halfduplex_enable
函数原型	void usart_halfduplex_enable(uint32_t usart_periph);
功能描述	使能USART半双工模式
先决条件	-
被调用函数	-

输入参数{in}	
<b>usart_periph</b>	外设USARTx
<i>USARTx</i>	x=0,1,2
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* enable USART0 half duplex mode */
```

```
usart_halfduplex_enable(USART0);
```

### 函数 **usart\_halfduplex\_disable**

函数usart\_halfduplex\_disable描述见下表：

**表 3-612. 函数 usart\_halfduplex\_disable**

<b>函数名称</b>	usart_halfduplex_disable
<b>函数原型</b>	void usart_halfduplex_disable(uint32_t usart_periph);
<b>功能描述</b>	失能USART半双工模式
<b>先决条件</b>	-
<b>被调用函数</b>	-
输入参数{in}	
<b>usart_periph</b>	外设USARTx
<i>USARTx</i>	x=0,1,2
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* 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 3-613. 函数 usart\_clock\_enable**

<b>函数名称</b>	usart_clock_enable
<b>函数原型</b>	void usart_clock_enable(uint32_t usart_periph);
<b>功能描述</b>	使能USART CK引脚
<b>先决条件</b>	-

被调用函数	-
输入参数{in}	
usart_periph	外设USARTx
USARTx	x=0,1,2
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* enable clock */
```

```
usart_clock_enable(USART0);
```

### 函数 usart\_clock\_disable

函数usart\_clock\_disable描述见下表：

表 3-614. 函数 usart\_clock\_disable

函数名称	usart_clock_disable
函数原型	void usart_clock_disable(uint32_t usart_periph);
功能描述	失能USART CK引脚
先决条件	-
被调用函数	-
输入参数{in}	
usart_periph	外设USARTx
USARTx	x=0,1,2
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* disable clock */
```

```
usart_clock_disable(USART0);
```

### 函数 usart\_synchronous\_clock\_config

函数usart\_synchronous\_clock\_config描述见下表：

表 3-615. 函数 usart\_synchronous\_clock\_config

函数名称	usart_synchronous_clock_config
函数原型	void usart_synchronous_clock_config(uint32_t usart_periph, uint32_t clen, uint32_t cph, uint32_t cpl);

功能描述	配置USART同步通讯模式参数
先决条件	-
被调用函数	-
输入参数{in}	
usart_periph	外设USARTx
USARTx	x=0,1,2
输入参数{in}	
clen	CK信号长度
USART_CLEN_NO NE	8位数据帧中有7个CK脉冲，9位数据帧中有8个CK脉冲
USART_CLEN_EN	8位数据帧中有8个CK脉冲，9位数据帧中有9个CK脉冲
输入参数{in}	
cph	时钟相位
USART_CPH_1CK	在首个时钟边沿采样第一个数据
USART_CPH_2CK	在第二个时钟边沿采样第一个数据
输入参数{in}	
cpl	时钟极性
USART_CPL_LOW	CK引脚不对外发送时保持为低电平
USART_CPL_HIGH	CK引脚不对外发送时保持为高电平
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* configure USART0 synchronous mode parameters */
```

```
usart_synchronous_clock_config(USART0,          USART_CLEN_EN,USART_CPH_2CK,
USART_CPL_HIGH);
```

### 函数 usart\_guard\_time\_config

函数usart\_guard\_time\_config描述见下表：

表 3-616. 函数 usart\_guard\_time\_config

函数名称	usart_guard_time_config
函数原型	void usart_guard_time_config(uint32_t usart_periph,uint32_t guat);
功能描述	在USART智能卡模式下配置保护时间值
先决条件	-
被调用函数	-
输入参数{in}	
usart_periph	外设USARTx
USARTx	x=0
输入参数{in}	

<b>guat</b>	保护时间值 (0x00-0x000000FF)
<b>输出参数{out}</b>	
-	-
<b>返回值</b>	
-	-

例如:

```
/* configure USART0 guard time value in smartcard mode */
usart_guard_time_config(USART0, 0x0000 0055);
```

### 函数 usart\_smartcard\_mode\_enable

函数usart\_smartcard\_mode\_enable描述见下表:

表 3-617. 函数 usart\_smartcard\_mode\_enable

<b>函数名称</b>	usart_smartcard_mode_enable
<b>函数原型</b>	void usart_smartcard_mode_enable(uint32_t usart_periph);
<b>功能描述</b>	使能USART智能卡模式
<b>先决条件</b>	-
<b>被调用函数</b>	-
<b>输入参数{in}</b>	
<b>usart_periph</b>	外设USARTx
USARTx	x=0
<b>输出参数{out}</b>	
-	-
<b>返回值</b>	
-	-

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 3-618. 函数 usart\_smartcard\_mode\_disable

<b>函数名称</b>	usart_smartcard_mode_disable
<b>函数原型</b>	void usart_smartcard_mode_disable(uint32_t usart_periph);
<b>功能描述</b>	失能USART智能卡模式
<b>先决条件</b>	-
<b>被调用函数</b>	-
<b>输入参数{in}</b>	



<b>usart_periph</b>	外设USARTx
<i>USARTx</i>	x=0
<b>输出参数{out}</b>	
-	-
<b>返回值</b>	
-	-

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 3-619. 函数 usart\_smartcard\_mode\_nack\_enable**

<b>函数名称</b>	usart_smartcard_mode_nack_enable
<b>函数原型</b>	void usart_smartcard_mode_nack_enable(uint32_t usart_periph);
<b>功能描述</b>	在USART智能卡模式下使能NACK
<b>先决条件</b>	-
<b>被调用函数</b>	-
<b>输入参数{in}</b>	
<b>usart_periph</b>	外设USARTx
<i>USARTx</i>	x=0
<b>输出参数{out}</b>	
-	-
<b>返回值</b>	
-	-

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 3-620. 函数 usart\_smartcard\_mode\_nack\_disable**

<b>函数名称</b>	usart_smartcard_mode_nack_disable
<b>函数原型</b>	void usart_smartcard_mode_nack_disable(uint32_t usart_periph);
<b>功能描述</b>	在USART智能卡模式下失能NACK
<b>先决条件</b>	-
<b>被调用函数</b>	-

输入参数{in}	
usart_periph	外设USARTx
USARTx	x=0
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 3-621. 函数 usart\_smartcard\_mode\_early\_nack\_enable

函数名称	usart_smartcard_mode_early_nack_enable
函数原型	void usart_smartcard_mode_early_nack_enable(uint32_t usart_periph);
功能描述	使能USART智能卡模式提前NACK
先决条件	-
被调用函数	-
输入参数{in}	
usart_periph	外设USARTx
USARTx	x=0
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 3-622. 函数 usart\_smartcard\_mode\_early\_nack\_disable

函数名称	usart_smartcard_mode_early_nack_disable
函数原型	void usart_smartcard_mode_early_nack_disable(uint32_t usart_periph);
功能描述	失能USART智能卡模式提前NACK
先决条件	-

被调用函数	-
输入参数{in}	
usart_periph	外设USARTx
USARTx	x=0
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 3-623. 函数 usart\_smartcard\_autoretry\_config

函数名称	usart_smartcard_autoretry_config
函数原型	void usart_smartcard_autoretry_config(uint32_t usart_periph, uint32_t scrtnum);
功能描述	配置智能卡自动重试次数
先决条件	-
被调用函数	-
输入参数{in}	
usart_periph	外设USARTx
USARTx	x=0
输入参数{in}	
scrtnum	智能卡自动重试次数（0x00-0x00000007）
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 3-624. 函数 `usart_block_length_config`

函数名称	<code>usart_block_length_config</code>
函数原型	<code>void usart_block_length_config(uint32_t usart_periph, uint32_t bl);</code>
功能描述	配置智能卡T=1的接收时块的长度
先决条件	-
被调用函数	-
输入参数{in}	
<code>usart_periph</code>	外设USARTx
<code>USARTx</code>	x=0
输入参数{in}	
<code>bl</code>	块长度（0x00-0x000000FF）
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* configure block length in Smartcard T=1 reception */
```

```
usart_block_length_config(USART0, 0x000000FF);
```

### 函数 `usart_irda_mode_enable`

函数`usart_irda_mode_enable`描述见下表：

表 3-625. 函数 `usart_irda_mode_enable`

函数名称	<code>usart_irda_mode_enable</code>
函数原型	<code>void usart_irda_mode_enable(uint32_t usart_periph);</code>
功能描述	使能USART串行红外编解码功能模块
先决条件	-
被调用函数	-
输入参数{in}	
<code>usart_periph</code>	外设USARTx
<code>USARTx</code>	x=0
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* enable USART0 IrDA mode */
```

```
usart_irda_mode_enable(USART0);
```

**函数 usart\_irda\_mode\_disable**

函数usart\_irda\_mode\_disable描述见下表:

**表 3-626. 函数 usart\_irda\_mode\_disable**

函数名称	usart_irda_mode_disable
函数原型	void usart_irda_mode_disable(uint32_t usart_periph);
功能描述	失能USART串行红外编解码功能模块
先决条件	-
被调用函数	-
输入参数{in}	
usart_periph	外设USARTx
USARTx	x=0
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* disable USART0 IrDA mode */  
usart_irda_mode_disable(USART0);
```

**函数 usart\_prescaler\_config**

函数usart\_prescaler\_config描述见下表:

**表 3-627. 函数 usart\_prescaler\_config**

函数名称	usart_prescaler_config
函数原型	void usart_prescaler_config(uint32_t usart_periph, uint32_t psc);
功能描述	在USART IrDA低功耗模式下配置外设时钟分频系数
先决条件	-
被调用函数	-
输入参数{in}	
usart_periph	外设USARTx
USARTx	x=0
输入参数{in}	
psc	时钟分频系数 (0x00-0xFF)
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* configure the USART0 peripheral clock prescaler in USART IrDA low-power mode */
```

```
usart_prescaler_config(USART0, 0x00);
```

### 函数 usart\_irda\_lowpower\_config

函数usart\_irda\_lowpower\_config描述见下表:

表 3-628. 函数 usart\_irda\_lowpower\_config

函数名称	usart_irda_lowpower_config
函数原型	void usart_irda_lowpower_config(uint32_t usart_periph, uint32_t irlp);
功能描述	配置USART IrDA低功耗模式
先决条件	-
被调用函数	-
输入参数{in}	
usart_periph	外设USARTx
USARTx	x=0
输入参数{in}	
irlp	IrDA低功耗模式或正常模式
USART_IRLP_LOW	低功耗模式
USART_IRLP_NORMAL	正常模式
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* configure USART0 IrDA low-power */
```

```
usart_irda_lowpower_config(USART0, USART_IRLP_LOW);
```

### 函数 usart\_hardware\_flow\_rts\_config

函数usart\_hardware\_flow\_rts\_config描述见下表:

表 3-629. 函数 usart\_hardware\_flow\_rts\_config

函数名称	usart_hardware_flow_rts_config
函数原型	void usart_hardware_flow_rts_config(uint32_t usart_periph, uint32_t rtsconfig);
功能描述	配置USART RTS硬件控制流
先决条件	-
被调用函数	-
输入参数{in}	
usart_periph	外设USARTx
USARTx	x=0,1,2
输入参数{in}	
rtsconfig	使能/失能RTS

<i>USART_RTS_ENA</i> <i>BLE</i>	使能RTS
<i>USART_RTS_DISA</i> <i>BLE</i>	失能RTS
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* configure USART0 hardware flow control RTS */
```

```
usart_hardware_flow_rts_config(USART0, USART_RTS_ENABLE);
```

### 函数 usart\_hardware\_flow\_cts\_config

函数usart\_hardware\_flow\_cts\_config描述见下表：

表 3-630. 函数 usart\_hardware\_flow\_cts\_config

函数名称	usart_hardware_flow_cts_config
函数原型	void usart_hardware_flow_cts_config(uint32_t usart_periph, uint32_t ctsconfig);
功能描述	配置USART CTS硬件控制流
先决条件	-
被调用函数	-
输入参数{in}	
<b>usart_periph</b>	外设USARTx
<i>USARTx</i>	x=0,1,2
输入参数{in}	
<b>ctsconfig</b>	使能/失能CTS
<i>USART_CTS_ENA</i> <i>BLE</i>	使能CTS
<i>USART_CTS_DISA</i> <i>BLE</i>	失能CTS
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* configure USART0 hardware flow control CTS */
```

```
usart_hardware_flow_cts_config(USART0, USART_CTS_ENABLE);
```

**函数 usart\_hardware\_flow\_coherence\_config**

函数usart\_hardware\_flow\_coherence\_config描述见下表：

**表 3-631. 函数 usart\_hardware\_flow\_coherence\_config**

函数名称	usart_hardware_flow_coherence_config
函数原型	void usart_hardware_flow_coherence_config(uint32_t usart_periph, uint32_t hcm);
功能描述	配置硬件流控兼容模式
先决条件	-
被调用函数	-
输入参数{in}	
usart_periph	外设USARTx
USARTx	x=0,1,2
输入参数{in}	
hcm	硬件流控制兼容模式
USART_HCM_NONE	nRTS信号与USART_STAT0寄存器中RBNE位相同
USART_HCM_EN	nRTS信号在最后一个数据位被采样后被置位
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* configure hardware flow control coherence mode */
```

```
usart_hardware_flow_coherence_config(USART0, USART_HCM_NONE);
```

**函数 usart\_rs485\_driver\_enable**

函数usart\_rs485\_driver\_enable描述见下表：

**表 3-632. 函数 usart\_rs485\_driver\_enable**

函数名称	usart_rs485_driver_enable
函数原型	void usart_rs485_driver_enable(uint32_t usart_periph);
功能描述	使能USART rs485驱动
先决条件	-
被调用函数	-
输入参数{in}	
usart_periph	外设USARTx
USARTx	x=0,1,2
输出参数{out}	
-	-
返回值	



-	-
---	---

例如:

```
/* enable USART0 RS485 driver */
usart_rs485_driver_enable(USART0);
```

### 函数 usart\_rs485\_driver\_disable

函数usart\_rs485\_driver\_disable描述见下表:

表 3-633. 函数 usart\_rs485\_driver\_disable

函数名称	usart_rs485_driver_disable
函数原型	void usart_rs485_driver_disable(uint32_t usart_periph);
功能描述	失能USART rs485驱动
先决条件	-
被调用函数	-
输入参数{in}	
usart_periph	外设USARTx
USARTx	x=0,1,2
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* disable USART0 RS485 driver */
usart_rs485_driver_disable(USART0);
```

### 函数 usart\_driver\_assertime\_config

函数usart\_driver\_assertime\_config描述见下表:

表 3-634. 函数 usart\_driver\_assertime\_config

函数名称	usart_driver_assertime_config
函数原型	void usart_driver_assertime_config(uint32_t usart_periph, uint32_t deatime);
功能描述	配置USART驱动使能置位时间
先决条件	-
被调用函数	-
输入参数{in}	
usart_periph	外设USARTx
USARTx	x=0,1,2
输入参数{in}	
deatime	驱动使能置位时间 (0x00-0x0000001F)

输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* set USART0 driver assertime */
```

```
usart_driver_assertime_config(USART0, 0x0000001F);
```

### 函数 usart\_driver\_deassertime\_config

函数usart\_driver\_deassertime\_config描述见下表：

**表 3-635. 函数 usart\_driver\_deassertime\_config**

函数名称	usart_driver_deassertime_config
函数原型	void usart_driver_deassertime_config(uint32_t usart_periph, uint32_t dedtime);
功能描述	配置USART驱动使能置低时间
先决条件	-
被调用函数	-
输入参数{in}	
usart_periph	外设USARTx
USARTx	x=0,1,2
输入参数{in}	
dedtime	驱动使能置低时间（0x00-0x0000001F）
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* set USART0 driver deassertime */
```

```
usart_driver_deassertime_config(USART0, 0x0000001F);
```

### 函数 usart\_depolarity\_config

函数usart\_depolarity\_config描述见下表：

**表 3-636. 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}	
<b>usart_periph</b>	外设USARTx
<i>USARTx</i>	x=0,1,2
输入参数{in}	
<b>dep</b>	驱动使能的极性选择模式
<i>USART_DEP_HIGH</i>	DE信号高有效
<i>USART_DEP_LOW</i>	DE信号低有效
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\_receive\_config描述见下表:

表 3-637. 函数 usart\_dma\_receive\_config

<b>函数名称</b>	usart_dma_receive_config
<b>函数原型</b>	void usart_dma_receive_config(uint32_t usart_periph, uint32_t dmacmd);
<b>功能描述</b>	配置USART DMA接收功能
<b>先决条件</b>	-
<b>被调用函数</b>	-
输入参数{in}	
<b>usart_periph</b>	外设USARTx
<i>USARTx</i>	x=0,1,2
输入参数{in}	
<b>dmacmd</b>	DMA使能/失能DMA接收功能
<i>USART_RECEIVE_DMA_ENABLE</i>	使能DMA接收功能
<i>USART_RECEIVE_DMA_DISABLE</i>	失能DMA接收功能
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* USART0 DMA enable for reception */
```

```
usart_dma_receive_config(USART0, USART_RECEIVE_DMA_ENABLE);
```

**函数 usart\_dma\_transmit\_config**

函数usart\_dma\_transmit\_config描述见下表:

**表 3-638. 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, uint32_t dmacmd);
功能描述	配置USART DMA发送功能
先决条件	-
被调用函数	-
输入参数{in}	
usart_periph	外设USARTx
USARTx	x=0,1,2
输入参数{in}	
dmacmd	使能/失能DMA发送功能
USART_TRANSMIT_DMA_ENABLE	使能DMA发送功能
USART_TRANSMIT_DMA_DISABLE	失能DMA发送功能
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* USART0 DMA enable for transmission */
```

```
usart_dma_transmit_config(USART0, USART_TRANSMIT_DMA_ENABLE);
```

**函数 usart\_reception\_error\_dma\_disable**

函数usart\_reception\_error\_dma\_disable描述见下表:

**表 3-639. 函数 usart\_reception\_error\_dma\_disable**

函数名称	usart_reception_error_dma_disable
函数原型	void usart_reception_error_dma_disable(uint32_t usart_periph);
功能描述	USART接收错误时失能DMA
先决条件	-
被调用函数	-
输入参数{in}	
usart_periph	外设USARTx
USARTx	x=0,1,2
输出参数{out}	
-	-
返回值	

-	-
---	---

例如:

```
/* disable DMA on reception error */
```

```
usart_reception_error_dma_disable(USART0);
```

### 函数 usart\_reception\_error\_dma\_enable

函数usart\_reception\_error\_dma\_enable描述见下表:

表 3-640. 函数 usart\_reception\_error\_dma\_enable

函数名称	usart_reception_error_dma_enable
函数原型	void usart_reception_error_dma_enable(uint32_t usart_periph);
功能描述	USART接收错误时使能DMA
先决条件	-
被调用函数	-
输入参数{in}	
usart_periph	外设USARTx
USARTx	x=0,1,2
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* enable DMA on reception error */
```

```
usart_reception_error_dma_enable(USART0);
```

### 函数 usart\_wakeup\_enable

函数usart\_wakeup\_enable描述见下表:

表 3-641. 函数 usart\_wakeup\_enable

函数名称	usart_wakeup_enable
函数原型	void usart_wakeup_enable(uint32_t usart_periph);
功能描述	使能USART唤醒
先决条件	-
被调用函数	-
输入参数{in}	
usart_periph	外设USARTx
USARTx	x=0
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 3-642. 函数 usart\_wakeup\_disable

函数名称	usart_wakeup_disable
函数原型	void usart_wakeup_disable(uint32_t usart_periph);
功能描述	失能USART唤醒
先决条件	-
被调用函数	-
输入参数{in}	
usart_periph	外设USARTx
USARTx	x=0
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 3-643. 函数 usart\_wakeup\_mode\_config

函数名称	usart_wakeup_mode_config
函数原型	void usart_wakeup_mode_config(uint32_t usart_periph, uint32_t wum);
功能描述	配置USART唤醒模式
先决条件	-
被调用函数	-
输入参数{in}	
usart_periph	外设USARTx
USARTx	x=0
输入参数{in}	

<b>wum</b>	唤醒模式
<i>USART_WUM_ADD R</i>	WUF在地址匹配时置位
<i>USART_WUM_STA RTB</i>	WUF在检测到起始位时置位
<i>USART_WUM_RBN E</i>	WUF在检测到RBNE时置位
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* configure USART0 wake up mode */
usart_wakeup_mode_config(USART0, USART_WUM_ADDR);
```

### 函数 usart\_receive\_fifo\_enable

函数usart\_receive\_fifo\_enable描述见下表：

表 3-644. 函数 usart\_receive\_fifo\_enable

函数名称	usart_receive_fifo_enable
函数原型	void usart_receive_fifo_enable(uint32_t usart_periph);
功能描述	使能接收FIFO
先决条件	-
被调用函数	-
输入参数{in}	
<b>usart_periph</b>	外设USARTx
<i>USARTx</i>	x=0
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 3-645. 函数 `usart_receive_fifo_disable`

函数名称	<code>usart_receive_fifo_disable</code>
函数原型	<code>void usart_receive_fifo_disable(uint32_t usart_periph);</code>
功能描述	失能接收FIFO
先决条件	-
被调用函数	-
输入参数{in}	
<code>usart_periph</code>	外设USARTx
<code>USARTx</code>	x=0
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 3-646. 函数 `usart_receive_fifo_counter_number`

函数名称	<code>usart_receive_fifo_counter_number</code>
函数原型	<code>uint8_t usart_receive_fifo_counter_number(uint32_t usart_periph);</code>
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}	
<code>usart_periph</code>	外设USARTx
<code>USARTx</code>	x=0
输出参数{out}	
-	-
返回值	
<code>uint8_t</code>	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 3-647. 函数 `usart_flag_get`

函数名称	<code>usart_flag_get</code>
函数原型	<code>FlagStatus usart_flag_get(uint32_t usart_periph, usart_flag_enum flag);</code>
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}	
<code>usart_periph</code>	外设USARTx
<code>USARTx</code>	x=0,1,2
输入参数{in}	
<code>flag</code>	USART标志位，参考 <a href="#">表3-578. 枚举类型usart_flag_enum</a> 只能选择一个参数
<code>USART_FLAG_PERR</code>	校验错误标志
<code>USART_FLAG_FERR</code>	帧错误标志
<code>USART_FLAG_NERR</code>	噪声错误标志
<code>USART_FLAG_ORERR</code>	溢出错误标志
<code>USART_FLAG_IDLE</code>	空闲线检测标志
<code>USART_FLAG_RBNE</code>	读数据缓冲区非空标志
<code>USART_FLAG_TC</code>	发送完成标志
<code>USART_FLAG_TBE</code>	发送数据缓冲区空标志
<code>USART_FLAG_LBD</code>	LIN断开检测标志（仅USART0支持）
<code>USART_FLAG_CTSF</code>	CTS变化标志
<code>USART_FLAG_CTS</code>	CTS电平
<code>USART_FLAG_RT</code>	接收超时标志（仅USART0支持）
<code>USART_FLAG_EB</code>	块结束标志（仅USART0支持）
<code>USART_FLAG_BSY</code>	忙状态标志
<code>USART_FLAG_AM</code>	ADDR匹配标志
<code>USART_FLAG_SB</code>	断开信号发送标识
<code>USART_FLAG_RWU</code>	接收器从静默模式唤醒
<code>USART_FLAG_WU</code>	从深度睡眠模式唤醒标志（仅USART0支持）
<code>USART_FLAG_TEA</code>	发送使能通知标志

USART_FLAG_REA	接收使能通知标志
USART_FLAG_EPERR	校验错误超前检测标志
USART_FLAG_RFE	接收FIFO空标志（仅USART0支持）
USART_FLAG_RFF	接收FIFO满标志（仅USART0支持）
USART_FLAG_RFFINT	接收FIFO满中断标志（仅USART0支持）
输出参数{out}	
-	-
返回值	
FlagStatus	SET或RESET

例如：

```
/* get flag USART0 state */
```

```
FlagStatus status;
```

```
status = usart_flag_get(USART0, USART_FLAG_TBE);
```

### 函数 usart\_flag\_clear

函数usart\_flag\_clear描述见下表：

表 3-648. 函数 usart\_flag\_clear

函数名称	usart_flag_clear
函数原型	void usart_flag_clear(uint32_t usart_periph, usart_flag_enum flag);
功能描述	清除USART状态寄存器标志位
先决条件	-
被调用函数	-
输入参数{in}	
usart_periph	外设USARTx
USARTx	x=0,1,2
输入参数{in}	
flag	USART标志位，参考 <a href="#">表3-578. 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_IDL	空闲线检测标志

<i>E</i>	
USART_FLAG_TC	发送完成标志
USART_FLAG_LBD	LIN断开检测标志（仅USART0支持）
USART_FLAG_CTS <i>F</i>	CTS变化标志
USART_FLAG_RT	接收超时标志（仅USART0支持）
USART_FLAG_EB	块结束标志（仅USART0支持）
USART_FLAG_AM	ADDR匹配标志
USART_FLAG_WU	从深度睡眠模式唤醒标志（仅USART0支持）
USART_FLAG_EPE <i>RR</i>	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 3-649. 函数 usart\_interrupt\_enable

函数名称	usart_interrupt_enable
函数原型	void usart_interrupt_enable(uint32_t usart_periph, usart_interrupt_enum interrupt);
功能描述	使能USART中断
先决条件	-
被调用函数	-
输入参数{in}	
usart_periph	外设USARTx
USARTx	x=0,1,2
UARTx	x=3,4
输入参数{in}	
interrupt	USART中断USART标志位，参考 <a href="#">表3-580. 枚举类型usart_interrupt_enum</a> 只能选择一个参数
USART_INT_IDLE	IDLE线检测中断
USART_INT_RBNE	读数据缓冲区非空中断和过载错误中断
USART_INT_TC	发送完成中断
USART_INT_TBE	发送缓冲区空中断
USART_INT_PERR	校验错误中断

USART_INT_AM	ADDR匹配中断
USART_INT_RT	接收超时事件中断（仅USART0支持）
USART_INT_EB	块结束事件中断（仅USART0支持）
USART_INT_LBD	LIN断开信号检测中断（仅USART0支持）
USART_INT_ERR	错误中断
USART_INT_CTS	CTS中断
USART_INT_WU	从深度睡眠模式唤醒中断（仅USART0支持）
USART_INT_RFF	接收FIFO满中断（仅USART0支持）
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 3-650. 函数 usart\_interrupt\_disable

函数名称	usart_interrupt_disable
函数原型	void usart_interrupt_disable(uint32_t usart_periph, usart_interrupt_enum interrupt);
功能描述	失能USART中断
先决条件	-
被调用函数	-
输入参数{in}	
usart_periph	外设USARTx
USARTx	x=0,1,2
输入参数{in}	
interrupt	USART中断USART标志位，参考 <a href="#">表3-580. 枚举类型usart_interrupt_enum</a> 只能选择一个参数
USART_INT_IDLE	IDLE线检测中断
USART_INT_RBNE	读数据缓冲区非空中断和过载错误中断
USART_INT_TC	发送完成中断
USART_INT_TBE	发送缓冲区空中断
USART_INT_PERR	校验错误中断
USART_INT_AM	ADDR匹配中断
USART_INT_RT	接收超时事件中断（仅USART0支持）
USART_INT_EB	块结束事件中断（仅USART0支持）
USART_INT_LBD	LIN断开信号检测中断（仅USART0支持）

USART_INT_ERR	错误中断
USART_INT_CTS	CTS中断
USART_INT_WU	从深度睡眠模式唤醒中断（仅USART0支持）
USART_INT_RFF	接收FIFO满中断（仅USART0支持）
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 3-651. 函数 usart\_interrupt\_flag\_get

函数名称	usart_interrupt_flag_get
函数原型	FlagStatus usart_interrupt_flag_get(uint32_t usart_periph, usart_interrupt_flag_enum int_flag);
功能描述	获取USART中断标志位状态
先决条件	-
被调用函数	-
输入参数{in}	
usart_periph	外设USARTx
USARTx	x=0,1,2
输入参数{in}	
int_flag	USART中断标志，参考 <a href="#">表3-579. 枚举类型usart_interrupt_flag_enum</a> 只能选择一个参数
USART_INT_FLAG _EB	块结束事件中断标志（仅USART0支持）
USART_INT_FLAG _RT	超时事件中断标志（仅USART0支持）
USART_INT_FLAG _AM	ADDR匹配中断标志
USART_INT_FLAG _PERR	校验错误中断标志
USART_INT_FLAG _TBE	发送缓冲区空中断标志
USART_INT_FLAG _TC	发送完成中断标志
USART_INT_FLAG	读数据缓冲区非空中断标志

<code>_RBNE</code>	
<code>USART_INT_FLAG_RBNE_ORERR</code>	读数据缓冲区非空中断和溢出错误中断标志
<code>USART_INT_FLAG_IDLE</code>	IDLE线检测中断标志
<code>USART_INT_FLAG_LBD</code>	LIN断开检测中断标志（仅USART0支持）
<code>USART_INT_FLAG_WU</code>	从深度睡眠模式唤醒中断标志（仅USART0支持）
<code>USART_INT_FLAG_CTS</code>	CTS中断标志
<code>USART_INT_FLAG_ERR_NERR</code>	噪声错误中断标志
<code>USART_INT_FLAG_ERR_ORERR</code>	过载错误中断标志
<code>USART_INT_FLAG_ERR_FERR</code>	帧错误中断标志
<code>USART_INT_FLAG_RFF</code>	接收FIFO满中断标志（仅USART0支持）
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 3-652. 函数 usart\_interrupt\_flag\_clear

函数名称	usart_interrupt_flag_clear
函数原型	void usart_interrupt_flag_clear(uint32_t usart_periph, usart_interrupt_flag_enum int_flag);
功能描述	清除USART中断标志位状态
先决条件	-
被调用函数	-
输入参数{in}	
usart_periph	外设USARTx
USARTx	x=0,1,2

输入参数{in}	
int_flag	USART中断标志，参考 <a href="#">表3-579. 枚举类型usart_interrupt_flag_enum</a> 只能选择一个参数
USART_INT_FLAG_PERR	校验错误中断标志
USART_INT_FLAG_ERR_FERR	帧错误中断标志
USART_INT_FLAG_ERR_NERR	噪声错误中断标志
USART_INT_FLAG_RBNE_ORERR	读数据缓冲区非空中断和溢出错误中断标志
USART_INT_FLAG_ERR_ORERR	过载错误中断标志
USART_INT_FLAG_IDLE	IDLE线检测中断标志
USART_INT_FLAG_TC	发送完成中断标志
USART_INT_FLAG_LBD	LIN断开检测中断标志（仅USART0支持）
USART_INT_FLAG_CTS	CTS变化中断标志
USART_INT_FLAG_RT	接收超时事件中断标志（仅USART0支持）
USART_INT_FLAG_EB	块结束事件中断标志（仅USART0支持）
USART_INT_FLAG_AM	ADDR匹配中断标志
USART_INT_FLAG_WU	从深度睡眠模式唤醒中断标志（仅USART0支持）
USART_INT_FLAG_RFF	接收FIFO满中断标志（仅USART0支持）
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* clear the USART0 interrupt flag */
```

```
usart_interrupt_flag_clear(USART0, USART_INT_FLAG_TC);
```

## 3.20. 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 [3.20.1](#) 描述了 WWDGT 的寄存器列表，章节 [3.20.2](#) 对 WWDGT 库函数进行说明。

### 3.20.1. 外设寄存器说明

WWDGT 寄存器列表如下表所示：

**表 3-653. 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

### 3.20.2. 外设库函数说明

WWDGT 库函数列表如下表所示：

**表 3-654. WWDGT 库函数**

库函数名称	库函数说明
wwdgt_deinit	将 WWDGT 寄存器重设为缺省值
wwdgt_enable	使能 WWDGT
wwdgt_prescaler_value_config	配置 WWDGT 预分频值
wwdgt_window_value_config	配置 WWDGT 窗口计数器的值
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 3-655. 函数 wwdgt\_deinit**

函数名称	wwdgt_deinit
函数原型	void wwdgt_deinit(void);
功能描述	将 WWDGT 寄存器重设为缺省值
先决条件	-
被调用函数	-
输入参数{in}	
-	-
输出参数{out}	



-	-
返回值	
-	-

例如：

```
/* reset the WWDGT configuration */
```

```
wwdgt_deinit();
```

### 函数 wwdgt\_enable

函数wwdgt\_enable描述见下表：

表 3-656. 函数 wwdgt\_enable

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

例如：

```
/* start the WWDGT counter */
```

```
wwdgt_enable();
```

### 函数 wwdgt\_prescaler\_value\_config

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

表 3-657. 函数 wwdgt\_prescaler\_value\_config

函数名称	wwdgt_prescaler_value_config
函数原型	void wwdgt_prescaler_value_config(uint16_t prescaler);
功能描述	配置WWDGT预分频值
先决条件	-
被调用函数	-
输入参数{in}	
prescaler	预分频值
WWDGT_CFG_PS C_DIV1	WWDGT计数器时钟为(WWDGT_CLK/4096)/1

WWDGT_CFG_PS C_DIV2	WWDGT计数器时钟为(WWDGT_CLK/4096)/2
WWDGT_CFG_PS C_DIV4	WWDGT计数器时钟为(WWDGT_CLK/4096)/4
WWDGT_CFG_PS C_DIV8	WWDGT计数器时钟为(WWDGT_CLK/4096)/8
WWDGT_CFG_PS C_DIV16	WWDGT计数器时钟为(WWDGT_CLK/4096)/16
WWDGT_CFG_PS C_DIV32	WWDGT计数器时钟为(WWDGT_CLK/4096)/32
WWDGT_CFG_PS C_DIV64	WWDGT计数器时钟为(WWDGT_CLK/4096)/64
WWDGT_CFG_PS C_DIV128	WWDGT计数器时钟为(WWDGT_CLK/4096)/128
WWDGT_CFG_PS C_DIV256	WWDGT计数器时钟为(WWDGT_CLK/4096)/256
WWDGT_CFG_PS C_DIV512	WWDGT计数器时钟为(WWDGT_CLK/4096)/512
WWDGT_CFG_PS C_DIV1024	WWDGT计数器时钟为(WWDGT_CLK/4096)/1024
WWDGT_CFG_PS C_DIV2048	WWDGT计数器时钟为(WWDGT_CLK/4096)/2048
WWDGT_CFG_PS C_DIV4096	WWDGT计数器时钟为(WWDGT_CLK/4096)/4096
WWDGT_CFG_PS C_DIV8192	WWDGT计数器时钟为(WWDGT_CLK/4096)/8192
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* update WWDGT prescaler value to 8192 */
```

```
wwdgt_prescaler_value_config (WWDGT_CFG_PSC_DIV8192);
```

### 函数 wwdgt\_window\_value\_config

函数wwdgt\_window\_value\_config描述见下表:

表 3-658. 函数 wwdgt\_window\_value\_config

函数名称	wwdgt_window_value_config
函数原型	void wwdgt_window_value_config(uint16_t window);
功能描述	配置WWDGT窗口计数器的值

先决条件	-
被调用函数	-
输入参数{in}	
window	WWDGT窗口计数器 (0x0000 - 0x007F)
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* update WWDGT window value to 0x50 */
```

```
wwdgt_window_value_config (80);
```

### 函数 wwdgt\_counter\_update

函数wwdgt\_counter\_update描述见下表：

表 3-659. 函数 wwdgt\_counter\_update

函数名称	wwdgt_counter_update
函数原型	void wwdgt_counter_update(uint16_t counter_value);
功能描述	设置WWDGT计数器更新值
先决条件	-
被调用函数	-
输入参数{in}	
counter_value	计数器值，数值范围为0x00000000 - 0x0000007F
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* update WWDGT counter to 0x7F */
```

```
wwdgt_counter_update(127);
```

### 函数 wwdgt\_config

函数wwdgt\_config描述见下表：

表 3-660. 函数 wwdgt\_config

函数名称	wwdgt_config
函数原型	void wwdgt_config(uint16_t counter, uint16_t window, uint32_t prescaler);
功能描述	设置WWDGT计数器值、窗口值和预分频值
先决条件	-

被调用函数	-
输入参数{in}	
counter	定时器计数值，数值范围0x00000000 - 0x0000007F
输入参数{in}	
window	窗口值，数值范围0x00000000 - 0x0000007F
输入参数{in}	
prescaler	WWDGT预分频值
WWDGT_CFG_PSC_DIV1	WWDGT计数器时钟为 (PCLK/4096) /1
WWDGT_CFG_PSC_DIV2	WWDGT计数器时钟为 (PCLK/4096) /2
WWDGT_CFG_PSC_DIV4	WWDGT计数器时钟为 (PCLK/4096) /4
WWDGT_CFG_PSC_DIV8	WWDGT计数器时钟为 (PCLK/4096) /8
WWDGT_CFG_PSC_DIV16	WWDGT计数器时钟为 (PCLK/4096) /16
WWDGT_CFG_PSC_DIV32	WWDGT计数器时钟为 (PCLK/4096) /32
WWDGT_CFG_PSC_DIV64	WWDGT计数器时钟为 (PCLK/4096) /64
WWDGT_CFG_PSC_DIV128	WWDGT计数器时钟为 (PCLK/4096) /128
WWDGT_CFG_PSC_DIV256	WWDGT计数器时钟为 (PCLK/4096) /256
WWDGT_CFG_PSC_DIV512	WWDGT计数器时钟为 (PCLK/4096) /512
WWDGT_CFG_PSC_DIV1024	WWDGT计数器时钟为 (PCLK/4096) /1024
WWDGT_CFG_PSC_DIV2048	WWDGT计数器时钟为 (PCLK/4096) /2048
WWDGT_CFG_PSC_DIV4096	WWDGT计数器时钟为 (PCLK/4096) /4096
WWDGT_CFG_PSC_DIV8192	WWDGT计数器时钟为 (PCLK/4096) /8192
输出参数{out}	
-	-
Return value	
-	-

例如：

```
/* configure WWDGT counter value to 0x7F, window value to 0x50, prescaler divider value to 8 */
```

```
wwdgt_config(127, 80, WWDGT_CFG_PSC_DIV8);
```

### 函数 `wwdgt_interrupt_enable`

函数 `wwdgt_interrupt_enable` 描述见下表:

表 3-661. 函数 `wwdgt_interrupt_enable`

函数名称	<code>wwdgt_interrupt_enable</code>
函数原型	<code>void wwdgt_interrupt_enable(void);</code>
功能描述	使能WWDGT提前唤醒中断
先决条件	-
被调用函数	-
输入参数{in}	
-	-
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* enable early wakeup interrupt of WWDGT */
```

```
wwdgt_interrupt_enable();
```

### 函数 `wwdgt_flag_get`

函数 `wwdgt_flag_get` 描述见下表:

表 3-662. 函数 `wwdgt_flag_get`

函数名称	<code>wwdgt_flag_get</code>
函数原型	<code>FlagStatus wwdgt_flag_get(void);</code>
功能描述	检查WWDGT提前唤醒中断标志位是否置位
先决条件	-
被调用函数	-
输入参数{in}	
-	-
输出参数{out}	
-	-
返回值	
<b>FlagStatus</b>	SET or RESET

例如:

```
/* test if the counter value update has reached the 0x40 */
```

```
FlagStatus status;
```

```
status = wwdgt_flag_get();
```

## 函数 wwdgt\_flag\_clear

函数wwdgt\_flag\_clear描述见下表：

**表 3-663. 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	2025 年 5 月 30 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.