

---

**GigaDevice Semiconductor Inc.**

**GD32F3x0**

**Arm<sup>®</sup> Cortex<sup>®</sup>-M4 32-bit MCU**

**固件库  
使用指南**

1.6 版本

(2026 年 1 月)

# 目录

目录.....	2
图索引 .....	5
表索引 .....	6
<b>1. 介绍 .....</b>	<b>22</b>
<b>1.1. 文档和固件库规则 .....</b>	<b>22</b>
1.1.1. 外设缩写 .....	22
1.1.2. 命名规则 .....	23
<b>2. 固件库概述 .....</b>	<b>24</b>
<b>2.1. 文件组织结构.....</b>	<b>24</b>
2.1.1. Examples 文件夹 .....	25
2.1.2. Firmware 文件夹 .....	25
2.1.3. Template 文件夹 .....	25
2.1.4. Utilities 文件夹 .....	28
<b>2.2. 固件库文件描述 .....</b>	<b>29</b>
<b>3. 外设固件库 .....</b>	<b>30</b>
<b>3.1. 外设固件库概述 .....</b>	<b>30</b>
<b>3.2. ADC .....</b>	<b>30</b>
3.2.1. 外设寄存器描述 .....	30
3.2.2. 外设库函数说明 .....	31
<b>3.3. CEC .....</b>	<b>55</b>
3.3.1. 外设寄存器描述 .....	56
3.3.2. 外设库函数说明 .....	56
<b>3.4. CMP.....</b>	<b>73</b>
3.4.1. 外设寄存器说明 .....	73
3.4.2. 外设库函数说明 .....	73
<b>3.5. CRC.....</b>	<b>81</b>
3.5.1. 外设寄存器说明 .....	81
3.5.2. 外设库函数说明 .....	82
<b>3.6. CTC.....</b>	<b>89</b>
3.6.1. 外设寄存器描述 .....	89
3.6.2. 外设库函数说明 .....	90
<b>3.7. DAC .....</b>	<b>102</b>
3.7.1. 外设寄存器说明 .....	102
3.7.2. 外设库函数说明 .....	103

<b>3.8. DBG .....</b>	<b>117</b>
3.8.1. 外设寄存器说明 .....	118
3.8.2. 外设库函数说明 .....	118
<b>3.9. DMA .....</b>	<b>122</b>
3.9.1. 外设寄存器说明 .....	122
3.9.2. 外设库函数说明 .....	123
<b>3.10. EXTI .....</b>	<b>140</b>
3.10.1. 外设寄存器说明 .....	141
3.10.2. 外设库函数说明 .....	141
<b>3.11. FMC .....</b>	<b>149</b>
3.11.1. 外设寄存器说明 .....	149
3.11.2. 外设库函数说明 .....	149
<b>3.12. FWDGT .....</b>	<b>168</b>
3.12.1. 外设寄存器说明 .....	168
3.12.2. 外设库函数说明 .....	168
<b>3.13. GPIO .....</b>	<b>174</b>
3.13.1. 外设寄存器说明 .....	174
3.13.2. 外设库函数说明 .....	174
<b>3.14. I2C .....</b>	<b>185</b>
3.14.1. 外设寄存器说明 .....	185
3.14.2. 外设库函数说明 .....	185
<b>3.15. MISC .....</b>	<b>208</b>
3.15.1. 外设寄存器说明 .....	209
3.15.2. 外设库函数说明 .....	210
<b>3.16. PMU .....</b>	<b>215</b>
3.16.1. 外设寄存器说明 .....	215
3.16.2. 外设库函数说明 .....	216
<b>3.17. RCU .....</b>	<b>228</b>
3.17.1. 外设寄存器说明 .....	228
3.17.2. 外设库函数说明 .....	229
<b>3.18. RTC .....</b>	<b>259</b>
3.18.1. 外设寄存器描述 .....	259
3.18.2. 外设库函数描述 .....	260
<b>3.19. SPI/I2S .....</b>	<b>279</b>
3.19.1. 外设寄存器说明 .....	279
3.19.2. 外设库函数说明 .....	280
<b>3.20. SYSCFG .....</b>	<b>305</b>
3.20.1. 外设寄存器说明 .....	305

3.20.2.	外设库函数说明 .....	306
<b>3.21.</b>	<b>TIMER .....</b>	<b>313</b>
3.21.1.	外设寄存器说明 .....	313
3.21.2.	外设库函数说明 .....	314
<b>3.22.</b>	<b>TSI.....</b>	<b>372</b>
3.22.1.	外设寄存器描述 .....	372
3.22.2.	外设库函数说明 .....	373
<b>3.23.</b>	<b>USART .....</b>	<b>395</b>
3.23.1.	外设寄存器说明 .....	395
3.23.2.	外设库函数说明 .....	395
<b>3.24.</b>	<b>WWDGT .....</b>	<b>441</b>
3.24.1.	外设寄存器说明 .....	441
3.24.2.	外设库函数说明 .....	441
<b>3.25.</b>	<b>USBFS.....</b>	<b>446</b>
<b>4.</b>	<b>版本历史 .....</b>	<b>447</b>

## 图索引

图 2-1. GD32F3x0 固件库文件组织结构.....	24
图 2-2. 选择外设例程文件 .....	26
图 2-3. 拷贝外设例程文件 .....	27
图 2-4. 打开工程文件.....	27
图 2-5. 配置工程文件.....	28
图 2-6. 编译调试下载.....	28

## 表索引

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

表 3-39. 函数 adc_interrupt_flag_clear .....	55
表 3-40. CEC 寄存器 .....	56
表 3-41. CEC 库函数 .....	56
表 3-42. 函数 cec_deinit.....	57
表 3-43. 函数 cec_init.....	57
表 3-44. 函数 cec_error_config .....	58
表 3-45. 函数 cec_enable.....	59
表 3-46. 函数 cec_disable.....	60
表 3-47. 函数 cec_transmission_start.....	60
表 3-48. 函数 cec_transmission_end .....	61
表 3-49. 函数 cec_listen_mode_enable .....	61
表 3-50. 函数 cec_listen_mode_disable .....	62
表 3-51. 函数 cec_own_address_config.....	62
表 3-52. 函数 cec_sft_config.....	63
表 3-53. 函数 cec_generate_errorbit_config.....	64
表 3-54. 函数 cec_stop_receive_bre_config .....	65
表 3-55. 函数 cec_reception_tolerance_enable.....	66
表 3-56. 函数 cec_reception_tolerance_disable .....	66
表 3-57. 函数 cec_data_send .....	67
表 3-58. 函数 cec_data_receive.....	67
表 3-59. 函数 cec_interrupt_enable .....	68
表 3-60. 函数 cec_interrupt_disable.....	68
表 3-61. 函数 cec_flag_get .....	69
表 3-62. 函数 cec_flag_clear .....	70
表 3-63. 函数 cec_interrupt_flag_get .....	71
表 3-64. 函数 cec_interrupt_flag_clear .....	72
表 3-65. CMP 寄存器.....	73
表 3-66. CMP 库函数.....	74
表 3-67. 枚举类型 cmp_enum .....	74
表 3-68. 函数 cmp_deinit .....	74
表 3-69. 函数 cmp_mode_init .....	75
表 3-70. 函数 cmp_output_init.....	76
表 3-71. 函数 cmp_enable .....	77
表 3-72. 函数 cmp_disable .....	78
表 3-73. 函数 cmp_switch_enable .....	78
表 3-74. 函数 cmp_switch_disable .....	79
表 3-75. 函数 cmp_window_enable .....	79
表 3-76. 函数 cmp_window_disable .....	80
表 3-77. 函数 cmp_lock_enable .....	80
表 3-78. 函数 cmp_output_level_get .....	81
表 3-79. CRC 寄存器 .....	81
表 3-80. CRC 库函数 .....	82
表 3-81. 函数 crc_deinit .....	82

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



表 3-125. 函数 dac_output_buffer_disable.....	107
表 3-126. 函数 dac_output_value_get.....	108
表 3-127. 函数 dac_data_set.....	108
表 3-128. 函数 dac_trigger_enable .....	109
表 3-129. 函数 dac_trigger_disable .....	110
表 3-130. 函数 dac_trigger_source_config .....	110
表 3-131. 函数 dac_software_trigger_enable .....	111
表 3-132. 函数 dac_wave_mode_config .....	112
表 3-133. 函数 dac_lfsr_noise_config .....	112
表 3-134. 函数 dac_triangle_noise_config .....	113
表 3-135. 函数 dac_flag_get.....	114
表 3-136. 函数 dac_flag_clear.....	115
表 3-137. 函数 dac_interrupt_enable.....	115
表 3-138. 函数 dac_interrupt_disable.....	116
表 3-139. 函数 dac_interrupt_flag_get .....	116
表 3-140. 函数 dac_interrupt_flag_clear .....	117
表 3-141. DBG 寄存器.....	118
表 3-142. DBG 库函数.....	118
表 3-143. 枚举类型 dbg_periph_enum.....	118
表 3-144. 函数 dbg_deinit.....	119
表 3-145. 函数 dbg_id_get.....	119
表 3-146. 函数 dbg_low_power_enable .....	120
表 3-147. 函数 dbg_low_power_disable .....	120
表 3-148. 函数 dbg_periph_enable .....	121
表 3-149. 函数 dbg_periph_disable .....	122
表 3-150. DMA 寄存器 .....	122
表 3-151. DMA 库函数 .....	123
表 3-152. 枚举类型 dma_channel_enum.....	124
表 3-153. 结构体类型 dma_parameter_struct.....	124
表 3-154. 函数 dma_deinit.....	124
表 3-155. 函数 dma_struct_para_init.....	125
表 3-156. 函数 dma_init .....	125
表 3-157. 函数 dma_circulation_enable.....	126
表 3-158. 函数 dma_circulation_disable.....	127
表 3-159. 函数 dma_memory_to_memory_enable.....	127
表 3-160. 函数 dma_memory_to_memory_disable.....	128
表 3-161. 函数 dma_channel_enable.....	128
表 3-162. 函数 dma_channel_disable.....	129
表 3-163. 函数 dma_periph_address_config .....	129
表 3-164. 函数 dma_memory_address_config .....	130
表 3-165. 函数 dma_transfer_number_config .....	130
表 3-166. 函数 dma_transfer_number_get .....	131
表 3-167. 函数 dma_priority_config.....	131

表 3-168. 函数 dma_memory_width_config .....	132
表 3-169. 函数 dma_periph_width_config .....	133
表 3-170. 函数 dma_memory_increase_enable .....	134
表 3-171. 函数 dma_memory_increase_disable .....	134
表 3-172. 函数 dma_periph_increase_enable .....	135
表 3-173. 函数 dma_periph_increase_disable .....	135
表 3-174. 函数 dma_transfer_direction_config .....	136
表 3-175. 函数 dma_flag_get .....	136
表 3-176. 函数 dma_flag_clear .....	137
表 3-177. 函数 dma_interrupt_enable .....	138
表 3-178. 函数 dma_interrupt_disable .....	138
表 3-179. 函数 dma_interrupt_flag_get .....	139
表 3-180. 函数 dma_interrupt_flag_clear .....	140
表 3-181. EXTI 寄存器 .....	141
表 3-182. EXTI 库函数 .....	141
表 3-183. 枚举类型 exti_line_enum .....	141
表 3-184. 枚举类型 exti_mode_enum .....	142
表 3-185. 枚举类型 exti_trig_type_enum .....	142
表 3-186. 函数 exti_deinit .....	142
表 3-187. 函数 exti_init .....	143
表 3-188. 函数 exti_interrupt_enable .....	144
表 3-189. 函数 exti_interrupt_disable .....	144
表 3-190. 函数 exti_event_enable .....	145
表 3-191. 函数 exti_event_disable .....	145
表 3-192. 函数 exti_software_interrupt_enable .....	146
表 3-193. 函数 exti_software_interrupt_disable .....	146
表 3-194. 函数 exti_flag_get .....	147
表 3-195. 函数 exti_flag_clear .....	147
表 3-196. 函数 exti_interrupt_flag_get .....	148
表 3-197. 函数 exti_interrupt_flag_clear .....	148
表 3-198. FMC 寄存器 .....	149
表 3-199. FMC 固件库函数 .....	149
表 3-200. 枚举类型 fmc_state_enum .....	150
表 3-201. 结构体类型 ob_parm_struct .....	150
表 3-202. 函数 fmc_unlock .....	151
表 3-203. 函数 fmc_lock .....	151
表 3-204. 函数 fmc_wscnt_set .....	152
表 3-205. 函数 fmc_wait_state_enable .....	152
表 3-206. 函数 fmc_wait_state_disable .....	153
表 3-207. 函数 fmc_page_erase .....	153
表 3-208. 函数 fmc_mass_erase .....	154
表 3-209. 函数 fmc_word_program .....	154
表 3-210. 函数 fmc_halfword_program .....	155

表 3-211. 函数 fmc_word_reprogram .....	155
表 3-212. 函数 ob_unlock .....	156
表 3-213. 函数 ob_lock .....	156
表 3-214. 函数 ob_reset .....	157
表 3-215. 函数 ob_erase .....	157
表 3-216. 函数 ob_write_protection_enable .....	158
表 3-217. 函数 ob_security_protection_config .....	158
表 3-218. 函数 ob_user_write .....	159
表 3-219. 函数 ob_data_program .....	159
表 3-220. 函数 ob_user_get .....	160
表 3-221. 函数 ob_data_get .....	161
表 3-222. 函数 ob_write_protection_get .....	161
表 3-223. 函数 ob_obstat_plevel_get .....	162
表 3-224. 函数 fmc_interrupt_enable .....	162
表 3-225. 函数 fmc_interrupt_disable .....	163
表 3-226. 函数 fmc_flag_get .....	163
表 3-227. 函数 fmc_flag_clear .....	164
表 3-228. 函数 fmc_interrupt_flag_get .....	164
表 3-229. 函数 fmc_interrupt_flag_clear .....	165
表 3-230. 函数 fmc_state_get .....	166
表 3-231. 函数 fmc_ready_wait .....	166
表 3-232. 函数 ob_parm_get .....	167
表 3-233. 函数 ob_value_modify .....	167
表 3-234. FWDGT 寄存器 .....	168
表 3-235. FWDGT 库函数 .....	168
表 3-236. 函数 fwdgt_write_enable .....	169
表 3-237. 函数 fwdgt_write_disable .....	169
表 3-238. 函数 fwdgt_enable .....	169
表 3-239. 函数 fwdgt_prescaler_value_config .....	170
表 3-240. 函数 fwdgt_reload_value_config .....	171
表 3-241. 函数 fwdgt_window_value_config .....	171
表 3-242. 函数 fwdgt_counter_reload .....	172
表 3-243. 函数 fwdgt_config .....	172
表 3-244. 函数 fwdgt_flag_get .....	173
表 3-245. GPIO 寄存器 .....	174
表 3-246. GPIO 库函数 .....	174
表 3-247. 函数 gpio_deinit .....	175
表 3-248. 函数 gpio_mode_set .....	175
表 3-249. 函数 gpio_output_options_set .....	176
表 3-250. 函数 gpio_bit_set .....	177
表 3-251. 函数 gpio_bit_reset .....	178
表 3-252. 函数 gpio_bit_write .....	178
表 3-253. 函数 gpio_port_write .....	179

表 3-254. 函数 gpio_input_bit_get .....	180
表 3-255. 函数 gpio_input_port_get .....	180
表 3-256. 函数 gpio_output_bit_get.....	181
表 3-257. 函数 gpio_output_port_get.....	182
表 3-258. 函数 gpio_af_set.....	182
表 3-259. 函数 gpio_pin_lock .....	183
表 3-260. 函数 gpio_bit_toggle.....	184
表 3-261. 函数 gpio_port_toggle .....	184
表 3-262. I2C 寄存器 .....	185
表 3-263. I2C 库函数 .....	185
表 3-264. 枚举类型 i2c_flag_enum .....	186
表 3-265. 枚举类型 i2c_interrupt_flag_enum .....	187
表 3-266. 枚举类型 i2c_interrupt_enum .....	187
表 3-267. 函数 i2c_deinit .....	188
表 3-268. 函数 i2c_clock_config .....	188
表 3-269. 函数 i2c_mode_addr_config.....	189
表 3-270. 函数 i2c_smbus_type_config .....	190
表 3-271. 函数 i2c_ack_config.....	190
表 3-272. 函数 i2c_ackpos_config .....	191
表 3-273. 函数 i2c_master_addressing.....	192
表 3-274. 函数 i2c_dualaddr_enable .....	192
表 3-275. 函数 i2c_dualaddr_disable .....	193
表 3-276. 函数 i2c_enable.....	193
表 3-277. 函数 i2c_disable.....	194
表 3-278. 函数 i2c_start_on_bus.....	194
表 3-279. 函数 i2c_stop_on_bus .....	195
表 3-280. 函数 i2c_data_transmit.....	195
表 3-281. 函数 i2c_data_receive.....	196
表 3-282. 函数 i2c_dma_config .....	196
表 3-283. 函数 i2c_dma_last_transfer_config .....	197
表 3-284. 函数 i2c_stretch_scl_low_config.....	198
表 3-285. 函数 i2c_slave_response_to_gcall_config .....	198
表 3-286. 函数 i2c_software_reset_config .....	199
表 3-287. 函数 i2c_pec_config.....	200
表 3-288. 函数 i2c_pec_transfer_config .....	200
表 3-289. 函数 i2c_pec_value_get .....	201
表 3-290. 函数 i2c_smbus_alert_config.....	201
表 3-291. 函数 i2c_smbus_arp_config.....	202
表 3-292. 函数 i2c_flag_get .....	203
表 3-293. 函数 i2c_flag_clear .....	204
表 3-294. 函数 i2c_interrupt_enable .....	205
表 3-295. 函数 i2c_interrupt_disable .....	205
表 3-296. 函数 i2c_interrupt_flag_get .....	206

表 3-297. 函数 i2c_interrupt_flag_clear .....	207
表 3-298. NVIC 寄存器.....	209
表 3-299. SysTick 寄存器.....	209
表 3-300. MISC 库函数.....	210
表 3-301. 枚举类型 IRQn_Type.....	210
表 3-302. 函数 nvic_priority_group_set.....	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 system_lowpower_set.....	213
表 3-307. 函数 system_lowpower_reset .....	214
表 3-308. 函数 systick_clksource_set.....	215
表 3-309. PMU 寄存器.....	215
表 3-310. PMU 库函数.....	216
表 3-311. 函数 pmu_deinit .....	216
表 3-312. 函数 pmu_lvd_select.....	217
表 3-313. 函数 pmu_ldo_output_select .....	217
表 3-314. 函数 pmu_lvd_disable .....	218
表 3-315. 函数 pmu_lowdriver_mode_enable.....	218
表 3-316. 函数 pmu_lowdriver_mode_disable.....	219
表 3-317. 函数 pmu_highdriver_mode_enable .....	219
表 3-318. 函数 pmu_highdriver_mode_disable .....	220
表 3-319. 函数 pmu_highdriver_switch_select.....	220
表 3-320. 函数 pmu_lowpower_driver_config .....	221
表 3-321. 函数 pmu_normalpower_driver_config .....	222
表 3-322. 函数 pmu_to_sleepmode .....	222
表 3-323. 函数 pmu_to_deepsleepmode .....	223
表 3-324. 函数 pmu_to_standbymode.....	224
表 3-325. 函数 pmu_wakeup_pin_enable .....	224
表 3-326. 函数 pmu_wakeup_pin_disable .....	225
表 3-327. 函数 pmu_backup_write_enable.....	226
表 3-328. 函数 pmu_backup_write_disable .....	226
表 3-329. 函数 pmu_flag_get .....	227
表 3-330. 函数 pmu_flag_clear .....	227
表 3-331. RCU 寄存器.....	228
表 3-332. RCU 库函数.....	229
表 3-333. 枚举类型 rcu_periph_enum.....	230
表 3-334. 枚举类型 rcu_periph_sleep_enum.....	231
表 3-335. 枚举类型 rcu_periph_reset_enum .....	231
表 3-336. 枚举类型 rcu_flag_enum .....	232
表 3-337. 枚举类型 rcu_int_flag_enum.....	233
表 3-338. 枚举类型 rcu_int_flag_clear_enum.....	233
表 3-339. 枚举类型 rcu_int_enum .....	234

表 3-340. 枚举类型 rcu_adc_clock_enum.....	234
表 3-341. 枚举类型 rcu_osci_type_enum.....	235
表 3-342. 枚举类型 rcu_clock_freq_enum .....	235
表 3-343. 函数 rcu_deinit.....	235
表 3-344. 函数 rcu_periph_clock_enable .....	236
表 3-345. 函数 rcu_periph_clock_disable .....	236
表 3-346. 函数 rcu_periph_clock_sleep_enable.....	237
表 3-347. 函数 rcu_periph_clock_sleep_disable.....	237
表 3-348. 函数 rcu_periph_reset_enable .....	238
表 3-349. 函数 rcu_periph_reset_disable .....	238
表 3-350. 函数 rcu_bkp_reset_enable .....	239
表 3-351. 函数 rcu_bkp_reset_disable .....	239
表 3-352. 函数 rcu_system_clock_source_config .....	240
表 3-353. 函数 rcu_system_clock_source_get.....	240
表 3-354. 函数 rcu_ahb_clock_config .....	241
表 3-355. 函数 rcu_apb1_clock_config .....	241
表 3-356. 函数 rcu_apb2_clock_config .....	242
表 3-357. 函数 rcu_adc_clock_config .....	242
表 3-358. 函数 rcu_usbfs_clock_config.....	243
表 3-359. 函数 rcu_ckout_config .....	244
表 3-360. 函数 rcu_pll_preselection_config.....	245
表 3-361. 函数 rcu_pll_config.....	245
表 3-362. 函数 rcu_usart_clock_config .....	246
表 3-363. 函数 rcu_cec_clock_config .....	247
表 3-364. 函数 rcu_rtc_clock_config .....	247
表 3-365. 函数 rcu_ck48m_clock_config .....	248
表 3-366. 函数 rcu_hxtal_prediv_config .....	248
表 3-367. 函数 rcu_lxtal_drive_capability_config .....	249
表 3-368. 函数 rcu_flag_get .....	250
表 3-369. 函数 rcu_all_reset_flag_clear.....	250
表 3-370. 函数 rcu_interrupt_flag_get.....	251
表 3-371. 函数 rcu_interrupt_flag_clear.....	251
表 3-372. 函数 rcu_interrupt_enable .....	252
表 3-373. 函数 rcu_interrupt_disable .....	252
表 3-374. 函数 rcu_osci_stab_wait.....	253
表 3-375. 函数 rcu_osci_on.....	253
表 3-376. 函数 rcu_osci_off .....	254
表 3-377. 函数 rcu_osci_bypass_mode_enable.....	254
表 3-378. 函数 rcu_osci_bypass_mode_disable .....	255
表 3-379. 函数 rcu_hxtal_clock_monitor_enable .....	255
表 3-380. 函数 rcu_hxtal_clock_monitor_disable .....	256
表 3-381. 函数 rcu_irc8m_adjust_value_set .....	256
表 3-382. 函数 rcu_irc28m_adjust_value_set .....	257

表 3-383. 函数 rcu_voltage_key_unlock .....	257
表 3-384. 函数 rcu_deepsleep_voltage_set .....	258
表 3-385. 函数 rcu_clock_freq_get .....	258
表 3-386. RTC 寄存器 .....	259
表 3-387. RTC 库函数 .....	260
表 3-388. 结构体类型 rtc_parameter_struct .....	261
表 3-389. 结构体类型 rtc_alarm_struct .....	261
表 3-390. 结构体 rtc_timestamp_struct .....	261
表 3-391. 结构体 rtc_tamper_struct .....	262
表 3-392. 函数 rtc_deinit .....	262
表 3-393. 函数 rtc_init .....	262
表 3-394. 函数 rtc_init_mode_enter .....	263
表 3-395. 函数 rtc_init_mode_exit .....	264
表 3-396. 函数 rtc_register_sync_wait .....	264
表 3-397. 函数 rtc_current_time_get .....	265
表 3-398. 函数 rtc_subsecond_get .....	265
表 3-399. 函数 rtc_alarm_config .....	266
表 3-400. 函数 rtc_alarm_subsecond_config .....	266
表 3-401. 函数 rtc_alarm_get .....	267
表 3-402. 函数 rtc_alarm_subsecond_get .....	268
表 3-403. 函数 rtc_alarm_enable .....	268
表 3-404. 函数 rtc_alarm_disable .....	269
表 3-405. 函数 rtc_timestamp_enable .....	269
表 3-406. 函数 rtc_timestamp_disable .....	270
表 3-407. 函数 rtc_timestamp_get .....	270
表 3-408. 函数 rtc_timestamp_subsecond_get .....	271
表 3-409. 函数 rtc_tamper_enable .....	271
表 3-410. 函数 rtc_tamper_disable .....	272
表 3-411. 函数 rtc_interrupt_enable .....	272
表 3-412. 函数 rtc_interrupt_disable .....	273
表 3-413. 函数 rtc_flag_get .....	273
表 3-414. 函数 rtc_flag_clear .....	274
表 3-415. 函数 rtc_alter_output_config .....	275
表 3-416. 函数 rtc_calibration_config .....	275
表 3-417. 函数 rtc_hour_adjust .....	276
表 3-418. 函数 rtc_second_adjust .....	277
表 3-419. 函数 rtc_bypass_shadow_enable .....	277
表 3-420. 函数 rtc_bypass_shadow_disable .....	278
表 3-421. 函数 rtc_refclock_detection_enable .....	278
表 3-422. 函数 rtc_refclock_detection_disable .....	279
表 3-423. SPI/I2S 寄存器 .....	280
表 3-424. SPI/I2S 库函数 .....	280
表 3-425. 结构体类型 spi_parameter_struct .....	281



表 3-426. 函数 spi_i2s_deinit.....	282
表 3-427. 函数 spi_struct_para_init.....	282
表 3-428. 函数 spi_init.....	283
表 3-429. 函数 spi_enable .....	283
表 3-430. 函数 spi_disable .....	284
表 3-431. 函数 i2s_init.....	284
表 3-432. 函数 i2s_psc_config.....	286
表 3-433. 函数 i2s_enable.....	287
表 3-434. 函数 i2s_disable.....	288
表 3-435. 函数 spi_nss_output_enable .....	288
表 3-436. 函数 spi_nss_output_disable.....	289
表 3-437. 函数 spi_nss_internal_high .....	289
表 3-438. 函数 spi_nss_internal_low.....	290
表 3-439. 函数 spi_dma_enable .....	290
表 3-440. 函数 spi_dma_disable .....	291
表 3-441. 函数 spi_i2s_data_frame_format_config .....	291
表 3-442. 函数 spi_bidirectional_transfer_config .....	292
表 3-443. 函数 spi_i2s_data_transmit .....	293
表 3-444. 函数 spi_i2s_data_receive .....	293
表 3-445. 函数 spi_i2s_format_error_clear .....	294
表 3-446. 函数 spi_crc_polynomial_set .....	294
表 3-447. 函数 spi_crc_polynomial_get.....	295
表 3-448. 函数 spi_crc_on.....	295
表 3-449. 函数 spi_crc_off.....	296
表 3-450. 函数 spi_crc_next.....	296
表 3-451. 函数 spi_crc_get.....	297
表 3-452. 函数 spi_crc_error_clear.....	298
表 3-453. 函数 spi_ti_mode_enable.....	298
表 3-454. 函数 spi_ti_mode_disable.....	299
表 3-455. 函数 spi_nssp_mode_enable .....	299
表 3-456. 函数 spi_nssp_mode_disable .....	300
表 3-457. 函数 spi_quad_enable .....	300
表 3-458. 函数 spi_quad_disable .....	301
表 3-459. 函数 spi_quad_write_enable .....	301
表 3-460. 函数 spi_quad_read_enable .....	302
表 3-461. 函数 spi_i2s_flag_get .....	302
表 3-462. 函数 spi_i2s_interrupt_enable .....	303
表 3-463. 函数 spi_i2s_interrupt_disable .....	304
表 3-464. 函数 spi_i2s_interrupt_flag_get.....	304
表 3-465. SYSCFG 寄存器 .....	306
表 3-466. SYSCFG 库函数 .....	306
表 3-467. 函数 syscfg_deinit.....	306
表 3-468. 函数 syscfg_dma_remap_enable .....	307



表 3-469. 函数 syscfg_dma_remap_disable .....	307
表 3-470. 函数 syscfg_high_current_enable.....	308
表 3-471. 函数 syscfg_high_current_disable.....	309
表 3-472. 函数 syscfg_exti_line_config .....	309
表 3-473. 函数 syscfg_lock_config.....	310
表 3-474. 函数 syscfg_flag_get .....	310
表 3-475. 函数 syscfg_flag_clear .....	311
表 3-476. 函数 syscfg_compensation_config.....	312
表 3-477. 函数 syscfg_cps_rdy_flag_get.....	312
表 3-478. TIMER 寄存器 .....	313
表 3-479. TIMER 库函数 .....	314
表 3-480. 结构体类型 timer_parameter_struct .....	316
表 3-481. 结构体类型 timer_break_parameter_struct.....	316
表 3-482. 结构体类型 timer_oc_parameter_struct.....	317
表 3-483. 结构体类型 timer_ic_parameter_struct.....	317
表 3-484. 函数 timer_deinit .....	317
表 3-485. 函数 timer_struct_para_init .....	318
表 3-486. 函数 timer_init.....	318
表 3-487. 函数 timer_enable.....	319
表 3-488. 函数 timer_disable.....	320
表 3-489. 函数 timer_auto_reload_shadow_enable.....	320
表 3-490. 函数 timer_auto_reload_shadow_disable.....	321
表 3-491. 函数 timer_update_event_enable .....	321
表 3-492. 函数 timer_update_event_disable .....	322
表 3-493. 函数 timer_counter_alignment.....	322
表 3-494. 函数 timer_counter_up_direction .....	323
表 3-495. 函数 timer_counter_down_direction.....	324
表 3-496. 函数 timer_prescaler_config .....	324
表 3-497. 函数 timer_repetition_value_config .....	325
表 3-498. 函数 timer_autoreload_value_config .....	326
表 3-499. 函数 timer_counter_value_config .....	326
表 3-500. 函数 timer_counter_read.....	327
表 3-501. 函数 timer_prescaler_read.....	327
表 3-502. 函数 timer_single_pulse_mode_config .....	328
表 3-503. 函数 timer_update_source_config .....	328
表 3-504. 函数 timer_ocpre_clear_source_config .....	329
表 3-505. 函数 timer_interrupt_enable .....	330
表 3-506. 函数 timer_interrupt_disable.....	331
表 3-507. 函数 timer_interrupt_flag_get .....	331
表 3-508. 函数 timer_interrupt_flag_clear .....	332
表 3-509. 函数 timer_flag_get.....	333
表 3-510. 函数 timer_flag_clear .....	334
表 3-511. 函数 timer_dma_enable .....	335

表 3-512. 函数 timer_dma_disable.....	336
表 3-513. 函数 timer_channel_dma_request_source_select .....	337
表 3-514. 函数 timer_dma_transfer_config .....	338
表 3-515. 函数 timer_event_software_generate .....	339
表 3-516. 函数 timer_break_struct_para_init .....	340
表 3-517. 函数 timer_break_config .....	341
表 3-518. 函数 timer_break_enable .....	342
表 3-519. 函数 timer_break_disable .....	342
表 3-520. 函数 timer_automatic_output_enable.....	343
表 3-521. 函数 timer_automatic_output_disable.....	343
表 3-522. 函数 timer_primary_output_config .....	344
表 3-523. 函数 timer_channel_control_shadow_config.....	345
表 3-524. 函数 timer_channel_control_shadow_update_config .....	345
表 3-525. 函数 timer_channel_output_struct_para_init.....	346
表 3-526. 函数 timer_channel_output_config.....	346
表 3-527. 函数 timer_channel_output_mode_config.....	347
表 3-528. 函数 timer_channel_output_pulse_value_config .....	349
表 3-529. 函数 timer_channel_output_shadow_config.....	349
表 3-530. 函数 timer_channel_output_fast_config .....	350
表 3-531. 函数 timer_channel_output_clear_config .....	351
表 3-532. 函数 timer_channel_output_polarity_config .....	352
表 3-533. 函数 timer_channel_complementary_output_polarity_config .....	353
表 3-534. 函数 timer_channel_output_state_config .....	353
表 3-535. 函数 timer_channel_complementary_output_state_config .....	354
表 3-536. 函数 timer_channel_input_struct_para_init .....	355
表 3-537. 函数 timer_input_capture_config .....	356
表 3-538. 函数 timer_channel_input_capture_prescaler_config.....	357
表 3-539. 函数 timer_channel_capture_value_register_read.....	357
表 3-540. 函数 timer_input_pwm_capture_config .....	358
表 3-541. 函数 timer_hall_mode_config .....	359
表 3-542. 函数 timer_input_trigger_source_select.....	360
表 3-543. 函数 timer_master_output_trigger_source_select.....	361
表 3-544. 函数 timer_slave_mode_select.....	362
表 3-545. 函数 timer_master_slave_mode_config .....	363
表 3-546. 函数 timer_external_trigger_config .....	363
表 3-547. 函数 timer_quadrature_decoder_mode_config .....	364
表 3-548. 函数 timer_internal_clock_config.....	365
表 3-549. 函数 timer_internal_trigger_as_external_clock_config.....	366
表 3-550. 函数 timer_external_trigger_as_external_clock_config.....	367
表 3-551. 函数 timer_external_clock_mode0_config .....	368
表 3-552. 函数 timer_external_clock_mode1_config .....	369
表 3-553. 函数 timer_external_clock_mode1_disable .....	370
表 3-554. 函数 timer_channel_remap_config .....	370

表 3-555. 函数 timer_write_chxval_register_config .....	371
表 3-556. 函数 timer_output_value_selection_config .....	372
表 3-557. TSI 寄存器 .....	372
表 3-558. TSI 库函数 .....	373
表 3-559. 函数 tsi_deinit .....	374
表 3-560. 函数 tsi_init.....	374
表 3-561. 函数 cec_enable.....	376
表 3-562. 函数 tsi_disable .....	376
表 3-563. 函数 tsi_sample_pin_enable .....	377
表 3-564. 函数 tsi_sample_pin_disable .....	377
表 3-565. 函数 tsi_channel_pin_enable .....	378
表 3-566. 函数 tsi_channel_pin_disable .....	379
表 3-567. 函数 tsi_sofeware_mode_config .....	379
表 3-568. 函数 tsi_software_start.....	380
表 3-569. 函数 tsi_software_stop.....	380
表 3-570. 函数 tsi_hardware_mode_config .....	381
表 3-571. 函数 tsi_pin_mode_config .....	381
表 3-572. 函数 tsi_extend_charge_config .....	382
表 3-573. 函数 tsi_pulse_config .....	383
表 3-574. 函数 tsi_max_number_config .....	384
表 3-575. 函数 tsi_hysteresis_on .....	385
表 3-576. 函数 tsi_hysteresis_off.....	385
表 3-577. 函数 tsi_analog_on .....	386
表 3-578. 函数 tsi_analog_off.....	386
表 3-579. 函数 tsi_group_enable.....	387
表 3-580. 函数 tsi_group_disable.....	387
表 3-581. 函数 tsi_group_status_get.....	388
表 3-582. 函数 tsi_group0_cycle_get .....	388
表 3-583. 函数 tsi_group1_cycle_get .....	389
表 3-584. 函数 tsi_group2_cycle_get .....	389
表 3-585. 函数 tsi_group3_cycle_get .....	390
表 3-586. 函数 tsi_group4_cycle_get .....	390
表 3-587. 函数 tsi_group5_cycle_get .....	391
表 3-588. 函数 tsi_flag_get.....	391
表 3-589. 函数 tsi_flag_clear.....	392
表 3-590. 函数 tsi_interrupt_enable.....	393
表 3-591. 函数 tsi_interrupt_disable.....	393
表 3-592. 函数 tsi_interrupt_flag_get .....	394
表 3-593. 函数 tsi_interrupt_flag_clear .....	394
表 3-594. USART 寄存器 .....	395
表 3-595. USART 库函数 .....	395
表 3-596. 枚举类型 usart_flag_enum .....	397
表 3-597. 枚举类型 usart_interrupt_flag_enum .....	398

表 3-598. 枚举类型 usart_interrupt_enum .....	399
表 3-599. 枚举类型 usart_invert_enum .....	399
表 3-600. 函数 usart_deinit .....	399
表 3-601. 函数 usart_baudrate_set .....	400
表 3-602. 函数 usart_parity_config .....	400
表 3-603. 函数 usart_word_length_set .....	401
表 3-604. 函数 usart_stop_bit_set .....	402
表 3-605. 函数 usart_enable .....	402
表 3-606. 函数 usart_disable .....	403
表 3-607. 函数 usart_transmit_config .....	403
表 3-608. 函数 usart_receive_config .....	404
表 3-609. 函数 usart_data_first_config .....	405
表 3-610. 函数 usart_invert_config .....	405
表 3-611. 函数 usart_overrun_enable .....	406
表 3-612. 函数 usart_overrun_disable .....	407
表 3-613. 函数 usart_oversample_config .....	407
表 3-614. 函数 usart_sample_bit_config .....	408
表 3-615. 函数 usart_receiver_timeout_enable .....	409
表 3-616. 函数 usart_receiver_timeout_disable .....	409
表 3-617. 函数 usart_receiver_timeout_threshold_config .....	410
表 3-618. 函数 usart_data_transmit .....	410
表 3-619. 函数 usart_data_receive .....	411
表 3-620. 函数 usart_address_config .....	411
表 3-621. 函数 usart_address_detection_mode_config .....	412
表 3-622. 函数 usart_mute_mode_enable .....	413
表 3-623. 函数 usart_mute_mode_disable .....	413
表 3-624. 函数 usart_mute_mode_wakeup_config .....	414
表 3-625. 函数 usart_lin_mode_enable .....	414
表 3-626. 函数 usart_lin_mode_disable .....	415
表 3-627. 函数 usart_lin_break_detection_length_config .....	415
表 3-628. 函数 usart_halfduplex_enable .....	416
表 3-629. 函数 usart_halfduplex_disable .....	416
表 3-630. 函数 usart_clock_enable .....	417
表 3-631. 函数 usart_clock_disable .....	417
表 3-632. 函数 usart_synchronous_clock_config .....	418
表 3-633. 函数 usart_guard_time_config .....	419
表 3-634. 函数 usart_smartcard_mode_enable .....	419
表 3-635. 函数 usart_smartcard_mode_disable .....	420
表 3-636. 函数 usart_smartcard_mode_nack_enable .....	420
表 3-637. 函数 usart_smartcard_mode_nack_disable .....	421
表 3-638. 函数 usart_smartcard_mode_early_nack_enable .....	422
表 3-639. 函数 usart_smartcard_mode_early_nack_disable .....	422
表 3-640. 函数 usart_smartcard_autoretry_config .....	423

表 3-641. 函数 usart_block_length_config .....	423
表 3-642. 函数 usart_irda_mode_enable .....	424
表 3-643. 函数 usart_irda_mode_disable .....	424
表 3-644. 函数 usart_prescaler_config .....	425
表 3-645. 函数 usart_irda_lowpower_config .....	425
表 3-646. 函数 usart_hardware_flow_rts_config .....	426
表 3-647. 函数 usart_hardware_flow_cts_config .....	427
表 3-648. 函数 usart_rs485_driver_enable .....	427
表 3-649. 函数 usart_rs485_driver_disable .....	428
表 3-650. 函数 usart_driver_asserttime_config .....	428
表 3-651. 函数 usart_driver_deasserttime_config .....	429
表 3-652. 函数 usart_depolarity_config .....	429
表 3-653. 函数 usart_dma_receive_config .....	430
表 3-654. 函数 usart_dma_transmit_config .....	431
表 3-655. 函数 usart_reception_error_dma_enable .....	431
表 3-656. 函数 usart_reception_error_dma_disable .....	432
表 3-657. 函数 usart_wakeup_enable .....	432
表 3-658. 函数 usart_wakeup_disable .....	433
表 3-659. 函数 usart_wakeup_mode_config .....	433
表 3-660. 函数 usart_command_enable .....	434
表 3-661. 函数 usart_receive_fifo_enable .....	435
表 3-662. 函数 usart_receive_fifo_disable .....	435
表 3-663. 函数 usart_receive_fifo_counter_number .....	436
表 3-664. 函数 usart_flag_get .....	436
表 3-665. 函数 usart_flag_clear .....	437
表 3-666. 函数 usart_interrupt_enable .....	438
表 3-667. 函数 usart_interrupt_disable .....	439
表 3-668. 函数 usart_interrupt_flag_get .....	439
表 3-669. 函数 usart_interrupt_flag_clear .....	440
表 3-670. WWDGT 寄存器 .....	441
表 3-671. WWDGT 库函数 .....	442
表 3-672. 函数 wwdgt_deinit .....	442
表 3-673. 函数 wwdgt_enable .....	442
表 3-674. 函数 wwdgt_counter_update .....	443
表 3-675. 函数 wwdgt_config .....	443
表 3-676. 函数 wwdgt_interrupt_enable .....	444
表 3-677. 函数 wwdgt_flag_get .....	444
表 3-678. 函数 wwdgt_flag_clear .....	445
表 4-1. 版本历史 .....	447

## 1. 介绍

本手册介绍了基于ARM处理器的32位微控制器GD32F3x0的固件库。

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GD32F3x0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。

该固件库是通用库，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，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
CEC	HDMI-CEC控制器
CMP	比较器
CRC	循环冗余校验计算单元
CTC	时钟校准控制器
DAC	数模转换器
DBG	调试模块
DMA	直接存储器访问控制器
EXTI	外部中断事件控制器
FMC	闪存控制器
FWDGT	独立看门狗

外设缩写	说明
GPIO/AFIO	通用和备用输入/输出接口
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
TSI	触摸传感控制器
USART	通用同步异步收发器
WWDGT	窗口看门狗
USBFS	通用串行总线全速接口

### 1.1.2. 命名规则

固件库遵从以下命名规则：

- XXX表示任一外设缩写，例如：ADC。更多缩写相关信息参阅[外设缩写](#)；
- 源文件和头文件命名都以“GD32f3x0\_”作为开头，例如：GD32f3x0\_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。

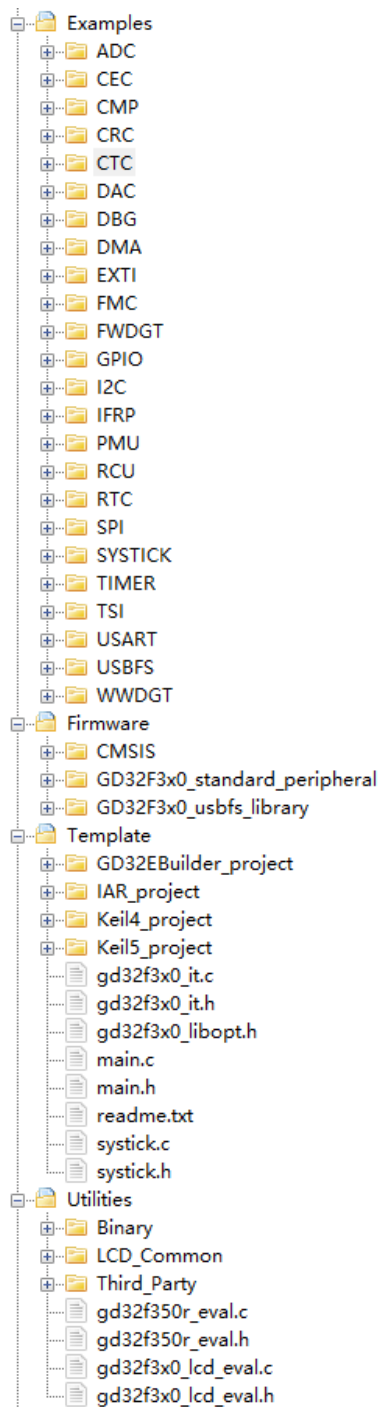


## 2. 固件库概述

### 2.1. 文件组织结构

GD32F3x0\_Firmware\_Library，文件组织结构见下图：

图 2-1. GD32F3x0 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；
- **GD32f3x0\_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）；
- **GD32f3x0\_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）；
- **GD32f3x0.it.h**: 该头文件包含了所有的中断处理程序的原形；
- **systick.c**: 该源文件包含了使用**systick**的精准延时程序；
- **systick.h**: 该头文件包含了使用**systick**的精准延时程序的原形；
- **main.c**: 例程代码注：所有的例程的使用，都不受不同软件开发环境的影响。

### 2.1.2. Firmware 文件夹

**Firmware**文件夹包含组成固件库核心的所有子文件夹和文件：

- **CMSIS**子文件夹包含有**Cortex M4**内核的支持文件、基于**Cortex M4**内核处理器的启动代码和库引导文件以及基于**GD32F3x0**的全局头文件和系统配置文件；
- **GD32F3x0\_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。

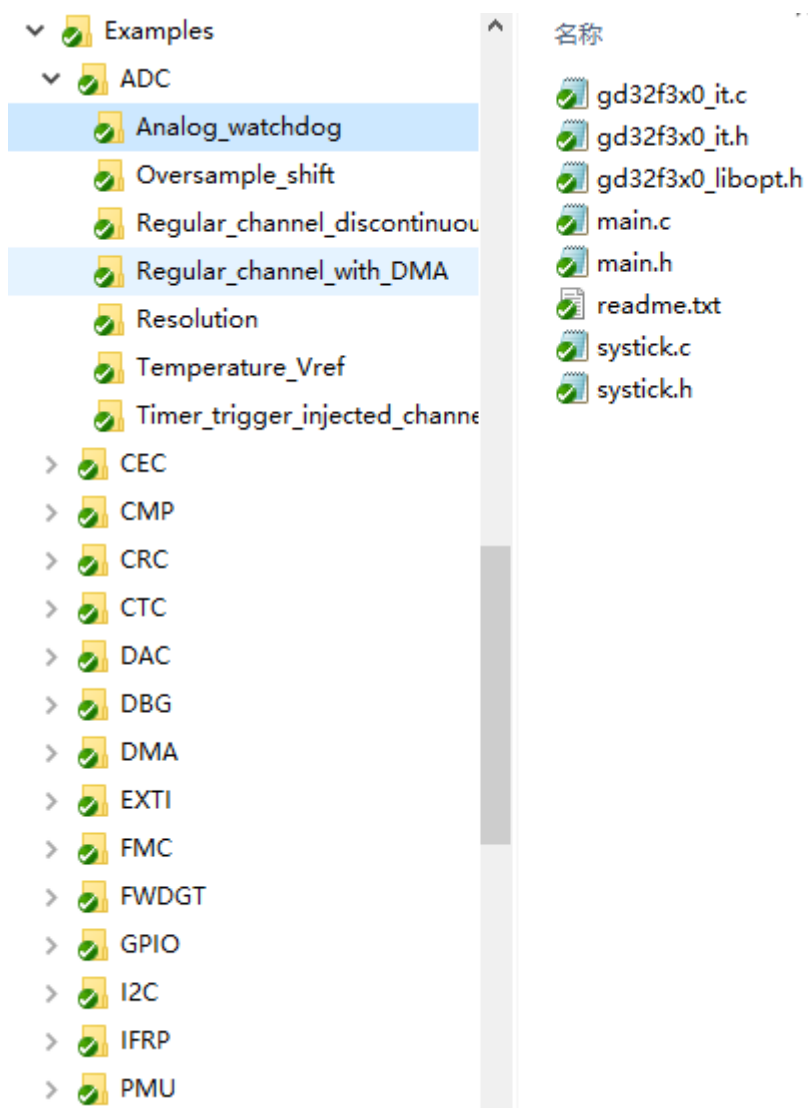
### 2.1.3. Template 文件夹

**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，**Keil4\_project**用于**Keil4**编译环境，**Keil5\_project**用于**Keil5**编译环境，**GD32EBuilder\_project**用于**GD32EmbeddedBuilder**编译环境）。用户可以使用该工程模板进行固件库例程的移植编译，具体使用方法见下：

#### 选择文件

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

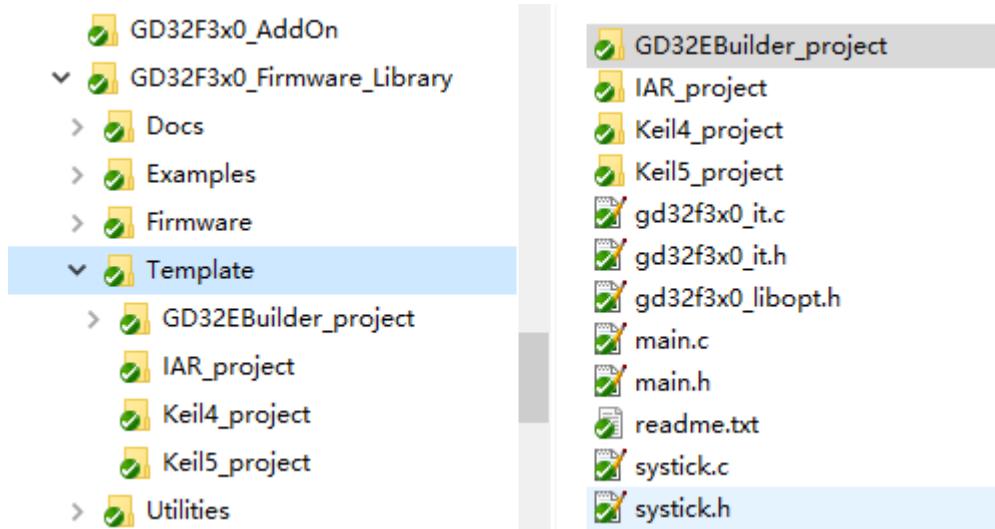
图 2-2. 选择外设例程文件



### 拷贝文件

打开“Template”文件夹，保留“ IAR\_project ”、“ Keil4\_project ”、“ Keil5\_project ”和“ GD32EBuilder\_project ”文件夹，其他文件都删除，然后将“SPI\_master\_transmit\_slave\_receive\_interrupt”文件夹中的所有文件拷到“Template”文件夹子目录下，如下图所示：

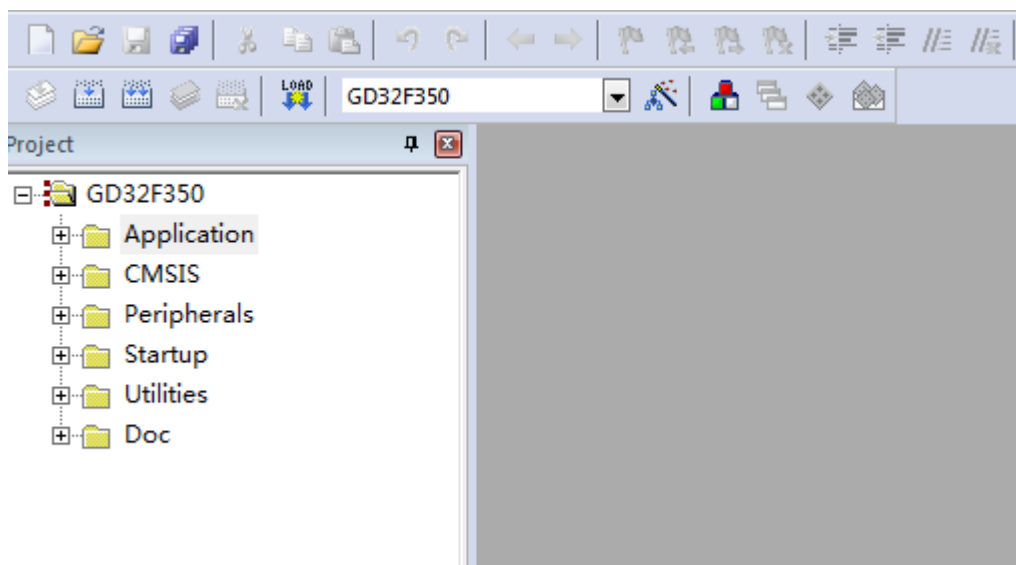
图 2-3. 拷贝外设例程文件



### 打开工程

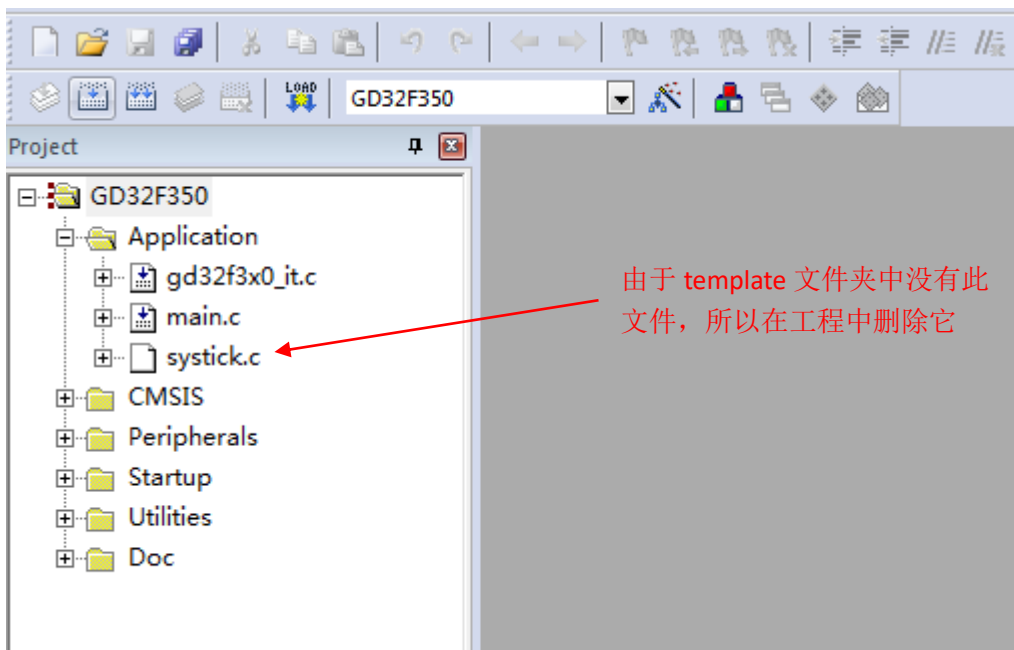
GD提供IAR、Keil4、Keil5和GD32EBuilder四种版本的工程，根据客户所安装的软件，打开不同的project，如“Keil4\_project”，打开\Template\Keil4\_project\Project.uvproj，如下图所示：

图 2-4. 打开工程文件



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：

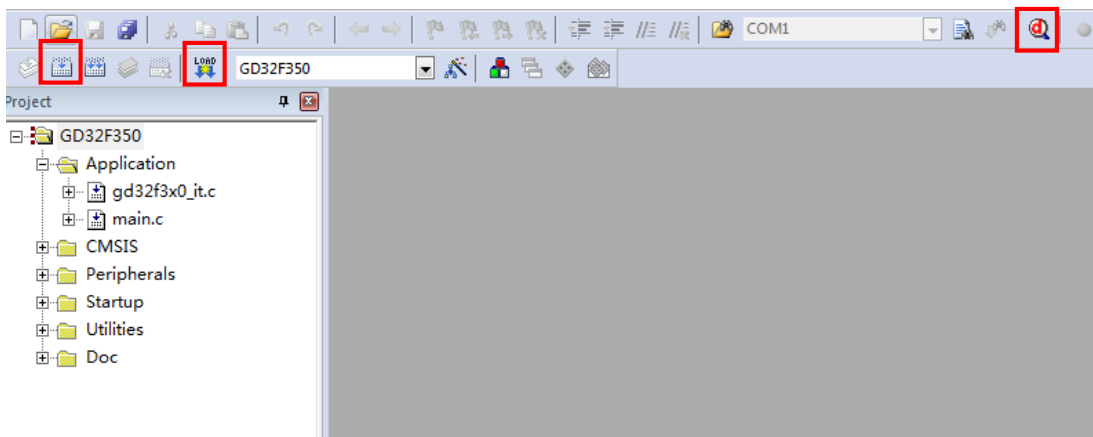
图 2-5. 配置工程文件



### 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Keil4，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：

- Binary、LCD\_Commom及Third\_Party子文件夹；
- GD32f3x0r\_eval.h及GD32f3x0\_lcd\_eval.h文件是运行固件库例程所需关于评估板的头文件；
- GD32f3x0r\_eval.c及GD32f3x0\_lcd\_eval.c文件是运行固件库例程所需关于评估板的源文

件。

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

## 2.2. 固件库文件描述

下表列举和描述了固件库使用的主要文件。

**表 2-1. 固件函数库文件描述**

文件名	描述
gd32f3x0_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。
gd32f3x0_it.h	头文件，包含所有中断处理函数原形。
gd32f3x0_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。
gd32f3x0_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。
gd32f3x0_xxx.c	由C语言编写的外设PPP的驱动源程序文件。
systick.h	systick.c的头文件。包含systick配置函数的定义，以及外部用延时函数的定义。
systick.c	systick配置与延时函数源文件。
readme.txt	固件库例程使用及配置说明文档。

### 3. 外设固件库

#### 3.1. 外设固件库概述

外设固件库函数的描述格式如下表：

表 3-1. 外设固件库函数描述格式

函数名称	外设函数的名称
函数原型	原型声明
功能描述	简要解释函数是如何执行的
先决条件	调用函数前应满足的要求
被调用函数	其他被该函数调用的库函数
输入参数{in}	
XXX	输入参数描述
XXX	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_WDHT	看门狗高阈值寄存器
ADC_WDLT	看门狗低阈值寄存器
ADC_RSQ0	规则序列寄存器0
ADC_RSQ1	规则序列寄存器1

寄存器名称	寄存器描述
ADC_RSQ2	规则序列寄存器2
ADC_ISQ	注入序列寄存器
ADC_IDATAx	注入数据寄存器x (x=0..3)
ADC_RDATA	规则数据寄存器
ADC_OVSAMPCTL	过采样控制寄存器

### 3.2.2. 外设库函数说明

ADC库函数列表如下表所示：

**表 3-3. ADC 库函数**

库函数名称	库函数描述
adc_deinit	复位ADC外设
adc_enable	使能ADC外设
adc_disable	禁能ADC外设
adc_calibration_enable	ADC校准复位
adc_dma_mode_enable	ADC DMA请求使能
adc_dma_mode_disable	ADC DMA请求禁能
adc_tempsensor_vrefint_enable	温度传感器和Vrefint通道使能
adc_tempsensor_vrefint_disable	温度传感器和Vrefint通道禁能
adc_vbat_enable	Vbat通道使能
adc_vbat_disable	Vbat通道禁能
adc_discontinuous_mode_config	配置ADC间断模式
adc_special_function_config	使能或禁能ADC特殊功能
adc_data_alignment_config	配置ADC数据对齐方式
adc_channel_length_config	配置规则通道组或注入通道组的长度
adc_regular_channel_config	配置ADC规则通道组
adc_inserted_channel_config	配置ADC注入通道组
adc_inserted_channel_offset_config	配置ADC注入通道组数据偏移值
adc_external_trigger_config	配置ADC外部触发
adc_external_trigger_source_config	配置ADC外部触发源
adc_software_trigger_enable	ADC软件触发使能
adc_regular_data_read	读ADC规则组数据寄存器
adc_inserted_data_read	读ADC注入组数据寄存器
adc_watchdog_single_channel_enable	配置ADC模拟看门狗单通道有效
adc_watchdog_group_channel_enable	配置ADC模拟看门狗在通道组有效
adc_watchdog_disable	ADC模拟看门狗禁能
adc_watchdog_threshold_config	配置ADC模拟看门狗阈值
adc_resolution_config	配置ADC分辨率

库函数名称	库函数描述
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\_calibration\_enable**

函数adc\_calibration\_enable描述见下表：

**表 3-7. 函数 adc\_calibration\_enable**

函数名称	adc_calibration_enable
函数原形	void adc_calibration_enable(void);
功能描述	ADC校准复位
先决条件	-
被调用函数	-
输入参数{in}	
-	-

输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* ADC calibration and reset calibration */
```

```
adc_calibration_enable();
```

### 函数 `adc_dma_mode_enable`

函数`adc_dma_mode_enable`描述见下表：

**表 3-8. 函数 `adc_dma_mode_enable`**

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

例如：

```
/* enable ADC DMA request */
```

```
adc_dma_mode_enable();
```

### 函数 `adc_dma_mode_disable`

函数`adc_dma_mode_disable`描述见下表：

**表 3-9. 函数 `adc_dma_mode_disable`**

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

-	-
返回值	
-	-

例如：

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

```
adc_dma_mode_disable();
```

### 函数 adc\_tempsensor\_vrefint\_enable

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

表 3-10. 函数 adc\_tempsensor\_vrefint\_enable

函数名称	adc_tempsensor_vrefint_enable
函数原形	void adc_tempsensor_vrefint_enable(void);
功能描述	温度传感器和Vrefint通道使能
先决条件	-
被调用函数	-
输入参数{in}	
-	-
输出参数{out}	
-	-
返回值	
-	-

例如：

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

```
adc_tempsensor_vrefint_enable();
```

### 函数 adc\_tempsensor\_vrefint\_disable

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

表 3-11. 函数 adc\_tempsensor\_vrefint\_disable

函数名称	adc_tempsensor_vrefint_disable
函数原形	void adc_tempsensor_vrefint_disable(void);
功能描述	温度传感器和Vrefint通道禁能
先决条件	-
被调用函数	-
输入参数{in}	
-	-
输出参数{out}	
-	-

返回值	
-	-

例如：

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

```
adc_tempsensor_vrefint_disable();
```

### 函数 `adc_vbat_enable`

函数`adc_vbat_enable`描述见下表：

**表 3-12. 函数 `adc_vbat_enable`**

函数名称	adc_vbat_enable
函数原形	void adc_vbat_enable(void);
功能描述	Vbat通道使能
先决条件	-
被调用函数	-
输入参数{in}	
-	-
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* enable the vbat channel */
```

```
adc_vbat_enable();
```

### 函数 `adc_vbat_disable`

函数`adc_vbat_disable`描述见下表：

**表 3-13. 函数 `adc_vbat_disable`**

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

-	-
---	---

例如:

```
/* disable the vbat channel */
```

```
adc_vbat_disable();
```

### 函数 `adc_discontinuous_mode_config`

函数`adc_discontinuous_mode_config`描述见下表:

**表 3-14. 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 channel_group, uint8_t length);</code>
功能描述	配置ADC间断模式
先决条件	-
被调用函数	-
输入参数{in}	
<code>adc_channel_group</code>	通道组选择
<code>ADC_REGULAR_CHANNEL</code>	规则通道组
<code>ADC_INSERTED_CHANNEL</code>	注入通道组
<code>ADC_CHANNEL_DISCON_DISABLE</code>	规则通道组和注入通道组间断模式禁能
输入参数{in}	
<code>length</code>	间断模式下的转换数目, 规则通道组取值为1..8, 注入通道组取值无意义
输出参数{out}	
-	-
返回值	
-	-

例如:

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

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

### 函数 `adc_special_function_config`

函数`adc_special_function_config`描述见下表:

**表 3-15. 函数 `adc_special_function_config`**

函数名称	<code>adc_special_function_config</code>
函数原形	<code>void adc_special_function_config(uint32_t function, ControlStatus newvalue);</code>

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

例如：

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

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

### 函数 adc\_data\_alignment\_config

函数adc\_alignment\_config描述见下表：

表 3-16. 函数 adc\_data\_alignment\_config

函数名称	adc_data_alignment_config
函数原形	void adc_data_alignment_config(uint32_t data_alignment);
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}	
data_alignment	数据对齐方式选择
ADC_DATAALIGN_RIGHT	右对齐
ADC_DATAALIGN_LEFT	左对齐
输出参数{out}	
-	-
返回值	

-	-
---	---

例如:

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

```
adc_data_alignment_config(ADC_DATAALIGN_RIGHT);
```

### 函数 `adc_channel_length_config`

函数`adc_channel_length_config`描述见下表:

**表 3-17. 函数 `adc_channel_length_config`**

函数名称	<code>adc_channel_length_config</code>
函数原形	<code>void adc_channel_length_config(uint8_t channel_group, uint32_t length);</code>
功能描述	配置规则通道组或注入通道组的长度
先决条件	-
被调用函数	-
输入参数{in}	
<b>channel_group</b>	通道组选择
<code>ADC_REGULAR_CHANNEL</code>	规则通道组
<code>ADC_INSERTED_CHANNEL</code>	注入通道组
输入参数{in}	
<b>length</b>	通道长度, 规则通道组为1-16, 注入通道组为1-4
输出参数{out}	
-	-
返回值	
-	-

例如:

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

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

### 函数 `adc_regular_channel_config`

函数`adc_regular_channel_config`描述见下表:

**表 3-18. 函数 `adc_regular_channel_config`**

函数名称	<code>adc_regular_channel_config</code>
函数原形	<code>void adc_regular_channel_config(uint8_t rank, uint8_t channel, uint32_t sample_time);</code>
功能描述	配置ADC规则通道组
先决条件	-

被调用函数	-
输入参数{in}	
rank	规则组通道序列，取值范围为0~15
输入参数{in}	
channel	ADC通道选择
ADC_CHANNEL_x	ADC通道x(x=0..18)
输入参数{in}	
sample_time	采样时间
ADC_SAMPLETIME_1POINT5	1.5 周期
ADC_SAMPLETIME_7POINT5	7.5 周期
ADC_SAMPLETIME_13POINT5	13.5 周期
ADC_SAMPLETIME_28POINT5	28.5 周期
ADC_SAMPLETIME_41POINT5	41.5 周期
ADC_SAMPLETIME_55POINT5	55.5 周期
ADC_SAMPLETIME_71POINT5	71.5 周期
ADC_SAMPLETIME_239POINT5	239.5 周期
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* configure ADC regular channel */
```

```
adc_regular_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 3-19. 函数 adc\_inserted\_channel\_config

函数名称	adc_inserted_channel_config
函数原形	void adc_inserted_channel_config(uint8_t rank, uint8_t channel, uint32_t sample_time);
功能描述	配置ADC注入通道组
先决条件	-



被调用函数	-
输入参数{in}	
rank	注入组通道序列，取值范围为0~3
输入参数{in}	
channel	ADC通道选择
ADC_CHANNEL_x	ADC通道x (x=0..18)
输入参数{in}	
sample_time	采样时间
ADC_SAMPLETIME_1POINT5	1.5 周期
ADC_SAMPLETIME_7POINT5	7.5 周期
ADC_SAMPLETIME_13POINT5	13.5 周期
ADC_SAMPLETIME_28POINT5	28.5 周期
ADC_SAMPLETIME_41POINT5	41.5 周期
ADC_SAMPLETIME_55POINT5	55.5 周期
ADC_SAMPLETIME_71POINT5	71.5 周期
ADC_SAMPLETIME_239POINT5	239.5 周期
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* configure 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 3-20. 函数 adc\_inserted\_channel\_offset\_config

函数名称	adc_inserted_channel_offset_config
函数原形	void adc_inserted_channel_offset_config(uint8_t inserted_channel, uint16_t offset);
功能描述	配置ADC注入通道组数据偏移值
先决条件	-

被调用函数	-
输入参数{in}	
<b>inserted_channel</b>	注入通道选择
<i>ADC_INSERTED_CHANNEL_x</i>	注入通道, x=0,1,2,3
输入参数{in}	
<b>offset</b>	数据偏移值, 取值范围为0~4095
输出参数{out}	
-	-
返回值	
-	-

例如:

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

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

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

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

表 3-21. 函数 **adc\_external\_trigger\_config**

函数名称	adc_external_trigger_config
函数原形	void adc_external_trigger_config(uint8_t channel_group, ControlStatus newvalue);
功能描述	配置ADC外部触发
先决条件	-
被调用函数	-
输入参数{in}	
<b>channel_group</b>	通道组选择
<i>ADC_REGULAR_CHANNEL</i>	规则通道组
<i>ADC_INSERTED_CHANNEL</i>	注入通道组
输入参数{in}	
<b>newvalue</b>	通道使能禁能
<i>ENABLE</i>	使能
<i>DISABLE</i>	禁能
输出参数{out}	
-	-
返回值	
-	-

例如:

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

```
adc_external_trigger_config(ADC_INSERTED_CHANNEL_0, ENABLE);
```

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

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

**表 3-22. 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 channel_group, uint32_t external_trigger_source);
功能描述	配置ADC外部触发源
先决条件	-
被调用函数	-
输入参数{in}	
channel_group	通道组选择
ADC_REGULAR_CHANNEL	规则通道组
ADC_INSERTED_CHANNEL	注入通道组
输入参数{in}	
external_trigger_source	规则通道组或注入通道组触发源
ADC_EXTTRIG_REGULAR_T0_CH0	TIMER0 CH0事件（规则组）
ADC_EXTTRIG_REGULAR_T0_CH1	TIMER0 CH1事件（规则组）
ADC_EXTTRIG_REGULAR_T0_CH2	TIMER0 CH2事件（规则组）
ADC_EXTTRIG_REGULAR_T1_CH1	TIMER1 CH1事件（规则组）
ADC_EXTTRIG_REGULAR_T2_TRGO	TIMER2 TRGO事件（规则组）
ADC_EXTTRIG_REGULAR_T14_CH0	TIMER14 CH0事件（规则组）
ADC_EXTTRIG_REGULAR_EXTI_11	外部中断线11（规则组）
ADC_EXTTRIG_REGULAR_NONE	软件触发（规则组）
ADC_EXTTRIG_	TIMER0 TRGO事件（注入组）

<i>INSERTED_T0 _TRGO</i>	
<i>ADC_EXTTRIG_ INSERTED_T0_ CH3</i>	TIMER0 CH3事件（注入组）
<i>ADC_EXTTRIG_ INSERTED_T1_ TRGO</i>	TIMER1 TRGO事件（注入组）
<i>ADC_EXTTRIG_ INSERTED_T1_ CH0</i>	TIMER1 CH0事件（注入组）
<i>ADC_EXTTRIG_ INSERTED_T2_ CH3</i>	TIMER2 CH3事件（注入组）
<i>ADC_EXTTRIG_ INSERTED_T14_ TRGO</i>	TIMER14 TRGO事件（注入组）
<i>ADC_EXTTRIG_ INSERTED_EXTI _15</i>	外部中断线15（注入组）
<i>ADC_EXTTRIG_ INSERTED_NONE</i>	软件触发（注入组）
输出参数{out}	
-	-
返回值	
-	-

例如：

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

```
adc_external_trigger_source_config(ADC_REGULAR_CHANNEL,  
ADC_EXTTRIG_REGULAR_T0_CH0);
```

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

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

**表 3-23. 函数 **adc\_software\_trigger\_enable****

函数名称	adc_software_trigger_enable
函数原形	void adc_software_trigger_enable(uint8_t channel_group);
功能描述	ADC软件触发使能
先决条件	-
被调用函数	-
输入参数{in}	

<b>channel_group</b>	通道组选择
<i>ADC_REGULAR_CHANNEL</i>	规则通道组
<i>ADC_INSERTED_CHANNEL</i>	注入通道组
<b>输出参数{out}</b>	
-	-
<b>返回值</b>	
-	-

例如：

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

```
adc_software_trigger_enable(ADC_REGULAR_CHANNEL);
```

### 函数 **adc\_regular\_data\_read**

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

**表 3-24. 函数 **adc\_regular\_data\_read****

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

例如：

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

```
uint16_t adc_value = 0;
```

```
adc_value = adc_regular_data_read();
```

### 函数 **adc\_inserted\_data\_read**

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

**表 3-25. 函数 **adc\_inserted\_data\_read****

<b>函数名称</b>	adc_inserted_data_read
<b>函数原形</b>	uint16_t adc_inserted_data_read(uint8_t inserted_channel);
<b>功能描述</b>	读ADC注入组数据寄存器
<b>先决条件</b>	-

被调用函数	-
输入参数{in}	
inserted_channel	注入通道选择
ADC_INSERTED_CHANNEL_x	注入通道x, x=0,1,2,3
输出参数{out}	
-	-
返回值	
uint16_t	ADC转换值(0-0xFFFF)

例如:

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

uint16_t adc_value = 0;

adc_value = adc_inserted_data_read(ADC_INSERTED_CHANNEL_0);
```

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

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

表 3-26. 函数 adc\_watchdog\_single\_channel\_enable

函数名称	adc_watchdog_single_channel_enable
函数原形	void adc_watchdog_single_channel_enable(uint8_t channel);
功能描述	配置ADC模拟看门狗单通道有效
先决条件	-
被调用函数	-
输入参数{in}	
adc_channel	选择ADC通道
ADC_CHANNEL_x	ADC通道x(x=0..18)
输出参数{out}	
-	-
返回值	
-	-

例如:

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

adc_watchdog_single_channel_enable(ADC_CHANNEL_1);
```

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

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

表 3-27. 函数 adc\_watchdog\_group\_channel\_enable

函数名称	adc_watchdog_group_channel_enable
------	-----------------------------------

函数原形	void adc_watchdog_group_channel_enable(uint8_t channel_group);
功能描述	配置ADC模拟看门狗在通道组有效
先决条件	-
被调用函数	-
输入参数{in}	
channel_group	通道组使用模拟看门狗
ADC_REGULAR_CHANNEL	规则通道组
ADC_INSERTED_CHANNEL	注入通道组
ADC_REGULAR_INSERTED_CHANNEL	规则和注入通道组
输出参数{out}	
-	-
返回值	
-	-

例如：

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

### 函数 adc\_watchdog\_disable

函数adc\_watchdog\_disable描述见下表：

表 3-28. 函数 adc\_watchdog\_disable

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

例如：

```
/* disable ADC analog watchdog */
adc_watchdog_disable();
```

## 函数 `adc_watchdog_threshold_config`

函数`adc_watchdog_threshold_config`描述见下表：

表 3-29. 函数 `adc_watchdog_threshold_config`

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

例如：

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

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

## 函数 `adc_resolution_config`

函数`adc_resolution_config`描述见下表：

表 3-30. 函数 `adc_resolution_config`

函数名称	<code>adc_resolution_config</code>
函数原形	<code>void adc_resolution_config(uint32_t resolution);</code>
功能描述	配置ADC分辨率
先决条件	-
被调用函数	-
输入参数{in}	
<code>resolution</code>	ADC分辨率
<code>ADC_RESOLUTION_12B</code>	12位分辨率
<code>ADC_RESOLUTION_10B</code>	10位分辨率
<code>ADC_RESOLUTION_8B</code>	8位分辨率
<code>ADC_RESOLUTION_6B</code>	6位分辨率



_6B	
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 3-31. 函数 `adc_oversample_mode_config`

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

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

-	-
返回值	
-	-

Example:

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

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

### 函数 adc\_oversample\_mode\_enable

函数adc\_oversample\_mode\_enable描述见下表:

表 3-32. 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 3-33. 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 3-34. 函数 `adc_flag_get`

函数名称	<code>adc_flag_get</code>
函数原形	<code>FlagStatus adc_flag_get(uint32_t flag);</code>
功能描述	获取ADC标志位
先决条件	-
被调用函数	-
输入参数{in}	
<b>flag</b>	ADC标志位
<code>ADC_FLAG_WDE</code>	模拟看门狗事件标志位
<code>ADC_FLAG_EOC</code>	组转换结束标志位
<code>ADC_FLAG_EOIC</code>	注入通道组转换结束标志位
<code>ADC_FLAG_STIC</code>	注入通道组转换开始标志位
<code>ADC_FLAG_STRC</code>	规则通道组转换开始标志位
输出参数{out}	
-	-
返回值	
<b>FlagStatus</b>	SET 或 RESET

例如:

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

### 函数 `adc_flag_clear`

函数`adc_flag_clear`描述见下表:

表 3-35. 函数 `adc_flag_clear`

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

先决条件	-
被调用函数	-
输入参数{in}	
<b>adc_flag</b>	ADC标志位
<i>ADC_FLAG_WDE</i>	模拟看门狗事件标志位
<i>ADC_FLAG_EOC</i>	组转换结束标志位
<i>ADC_FLAG_EOIC</i>	注入通道组转换结束标志位
<i>ADC_FLAG_STIC</i>	注入通道组转换开始标志位
<i>ADC_FLAG_STRC</i>	规则通道组转换开始标志位
输出参数{out}	
-	-
返回值	
-	-

例如：

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

```
adc_flag_clear(ADC_FLAG_WDE);
```

### 函数 **adc\_interrupt\_enable**

函数adc\_interrupt\_enable描述见下表：

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

函数名称	adc_interrupt_enable
函数原形	void adc_interrupt_enable(uint32_t interrupt);
功能描述	ADC中断使能
先决条件	-
被调用函数	-
输入参数{in}	
<b>interrupt</b>	ADC中断标志位
<i>ADC_INT_WDE</i>	模拟看门狗中断标志位
<i>ADC_INT_EOC</i>	组转换结束中断标志位
<i>ADC_INT_EOIC</i>	注入通道组转换结束中断标志位
输出参数{out}	
-	-
返回值	
-	-

例如：

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

```
adc_interrupt_enable(ADC_INT_WDE);
```

## 函数 adc\_interrupt\_disable

函数adc\_interrupt\_disable描述见下表:

表 3-37. 函数 adc\_interrupt\_disable

函数名称	adc_interrupt_disable
函数原形	void adc_interrupt_disable(uint32_t interrupt);
功能描述	ADC中断禁能
先决条件	-
被调用函数	-
输入参数{in}	
interrupt	ADC中断标志位
ADC_INT_WDE	模拟看门狗中断标志位
ADC_INT_EOC	组转换结束中断标志位
ADC_INT_EOIC	注入通道组转换结束中断标志位
输出参数{out}	
-	-
返回值	
-	-

例如:

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

```
adc_interrupt_disable(ADC_INT_WDE);
```

## 函数 adc\_interrupt\_flag\_get

函数adc\_interrupt\_flag\_get描述见下表:

表 3-38. 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 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}	
flag	ADC中断标志位
ADC_INT_FLAG_WDE	模拟看门狗中断标志位
ADC_INT_FLAG_EOC	组转换结束中断标志位
ADC_INT_FLAG_EOIC	注入通道组转换结束中断标志位
输出参数{out}	

-	-
返回值	
FlagStatus	SET 或 RESET

例如：

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

```
FlagStatus flag_value;
```

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

### 函数 adc\_interrupt\_flag\_clear

函数adc\_interrupt\_flag\_clear描述见下表：

表 3-39. 函数 adc\_interrupt\_flag\_clear

函数名称	adc_interrupt_flag_clear
函数原形	void adc_interrupt_flag_clear(uint32_t flag);
功能描述	清除ADC中断标志位
先决条件	-
被调用函数	-
输入参数{in}	
flag	ADC中断标志位
ADC_INT_FLAG_WDE	模拟看门狗中断标志位
ADC_INT_FLAG_EOC	组转换结束中断标志位
ADC_INT_FLAG_EOIC	注入通道组转换结束中断标志位
输出参数{out}	
-	-
返回值	
-	-

例如：

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

```
adc_interrupt_flag_clear(ADC_INT_FLAG_WDE);
```

## 3.3. CEC

消费电子控制（CEC）是 HDMI（高清多媒体接口）标准的一部分。CEC 作为一种协议，提供了在用户环境中各种音像制品之间的高级控制功能。章节[3.3.1](#)描述了CEC的寄存器列表，章节[3.3.2](#)对CEC库函数进行说明。

### 3.3.1. 外设寄存器描述

CEC寄存器列表如下表所示：

**表 3-40. CEC 寄存器**

寄存器名称	寄存器描述
CEC_CTL	控制寄存器
CEC_CFG	配置寄存器
CEC_TDATA	数据发送寄存器
CEC_RDATA	数据接收寄存器
CEC_INTF	中断标志寄存器
CEC_INTEN	中断使能寄存器

### 3.3.2. 外设库函数说明

CEC库函数列表如下表所示：

**表 3-41. CEC 库函数**

库函数名称	库函数描述
cec_deinit	复位HDMI-CEC外设
cec_init	HDMI-CEC初始化各个参数
cec_error_config	检测到错误时进行相关配置
cec_enable	使能HDMI-CEC外设
cec_disable	关闭HDMI-CEC外设
cec_transmission_start	开始CEC传输
cec_transmission_end	关闭CEC传输
cec_listen_mode_enable	使能CEC监听模式
cec_listen_mode_disable	关闭CEC监听模式
cec_own_address_config	CEC自身地址配置
cec_sft_config	配置CEC信号空闲时间
cec_generate_errorbit_config	配置CEC是否产生错误位
cec_stop_receive_bre_config	监测到位上升错误时是否停止接收信息
cec_reception_tolerance_enable	CEC扩展接收位时间宽容度
cec_reception_tolerance_disable	CEC标准接收位时间宽容度
cec_data_send	CEC数据发送
cec_data_receive	CEC数据接收
cec_interrupt_enable	CEC中断使能
cec_interrupt_disable	CEC中断禁止
cec_flag_get	CEC标志位获取
cec_flag_clear	CEC标志位清除
cec_interrupt_flag_get	CEC中断标志位获取
cec_interrupt_flag_clear	CEC中断标志位清除



**函数 cec\_deinit**

函数cec\_deinit描述见下表:

**表 3-42. 函数 cec\_deinit**

函数名称	cec_deinit
函数原形	void cec_deinit(void);
功能描述	复位HDMI-CEC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 CEC*/
cec_deinit();
```

**函数 cec\_init**

函数cec\_init描述见下表:

**表 3-43. 函数 cec\_init**

函数名称	cec_init
函数原形	void cec_init(uint32_t sftmopt, uint32_t sft, uint32_t address);
功能描述	HDMI-CEC初始化各个参数
先决条件	-
被调用函数	-
输入参数{in}	
sftmopt	SFT开始选项位
CEC_SFT_START_STAOM	STAOM置1后SFT计数器开始计数
CEC_SFT_START_LAST	发收/接收结束后SFT计数器自动启动
输入参数{in}	
sft	信号空闲时间
CEC_SFT_PROTO_COL_PERIOD	SFT时间跟HDMI-CEC协议描述的一样
CEC_SFT_1POINT5_PERIOD	1.5个额定数据位周期

CEC_SFT_2POINT 5_PERIOD	2.5个额定数据位周期
CEC_SFT_3POINT 5_PERIOD	3.5个额定数据位周期
CEC_SFT_4POINT 5_PERIOD	4.5个额定数据位周期
CEC_SFT_5POINT 5_PERIOD	5.5个额定数据位周期
CEC_SFT_6POINT 5_PERIOD	6.5个额定数据位周期
CEC_SFT_7POINT 5_PERIOD	7.5个额定数据位周期
输入参数{in}	
address	地址设置
CEC_OWN_ADDRE SS_CLEAR	清除地址
CEC_OWN_ADDRE SSx	自身地址 (x=0-14)
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* init CEC*/
```

```
cec_init(CEC_SFT_START_STAOM, CEC_SFT_PROTOCOL_PERIOD,  
CEC_OWN_ADDRESS0);
```

### 函数 cec\_error\_config

函数cec\_error\_config描述见下表:

表 3-44. 函数 cec\_error\_config

函数名称	cec_error_config
函数原形	void cec_error_config(uint32_t broadcast, uint32_t singlecast_lbpe, uint32_t singlecast_bre, uint32_t rxbretp);
功能描述	检测到错误时进行相关配置
先决条件	-
被调用函数	-
输入参数{in}	
broadcast	广播信息模式下是否产生错误位
CEC_BROADCAST _ERROR_BIT_ON	广播信息模式下产生错误位

CEC_BROADCAST_ERROR_BIT_OFF	广播信息模式下不产生错误位
输入参数{in}	
singlecast_lbpe	在单播模式下监测到BPLE的时候是否产生错误位
CEC_LONG_PERIOD_ERROR_BIT_ON	在单播模式下监测到BPLE时产生错误位
CEC_LONG_PERIOD_ERROR_BIT_OFF	在单播模式下监测到BPLE时不产生错误位
输入参数{in}	
singlecast_bre	在单次传播模式下监测到BRE的时候是否产生错误位
CEC_RISING_PERIOD_ERROR_BIT_ON	在单播模式下监测到BRE的时候产生错误位
CEC_RISING_PERIOD_ERROR_BIT_OFF	在单播模式下监测到BRE的时候不产生错误位
输入参数{in}	
rxbrestp	监测到BRE时是否停止接收信息
CEC_STOP_RISING_ERROR_BIT_ON	停止接收BRE下的信息
CEC_STOP_RISING_ERROR_BIT_OFF	不停止接收BRE下的信息，数据位按额定时间采样（1.05ms）
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* configure generate Error-bit when detected some abnormal situation or not, whether stop
receive message when detected bit rising error */
```

```
cec_error_config(CEC_BROADCAST_ERROR_BIT_ON,
CEC_LONG_PERIOD_ERROR_BIT_ON, CEC_RISING_PERIOD_ERROR_BIT_ON,
CEC_STOP_RISING_ERROR_BIT_ON);
```

### 函数 cec\_enable

函数cec\_enable描述见下表：

表 3-45. 函数 cec\_enable

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

功能描述	使能HDMI-CEC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 HDMI-CEC controller */
```

```
cec_enable();
```

### 函数 cec\_disable

函数cec\_disable描述见下表：

**表 3-46. 函数 cec\_disable**

函数名称	cec_disable
函数原形	void cec_disable (void);
功能描述	禁用HDMI-CEC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 HDMI-CEC controller */
```

```
cec_disable ();
```

### 函数 cec\_transmission\_start

函数cec\_transmission\_start描述见下表：

**表 3-47. 函数 cec\_transmission\_start**

函数名称	cec_transmission_start
函数原形	void cec_transmission_start (void);
功能描述	开始CEC传输

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

例如：

```
/* start CEC message transmission */
cec_transmission_start ();
```

### 函数 cec\_transmission\_end

函数cec\_transmission\_end描述见下表：

表 3-48. 函数 cec\_transmission\_end

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

例如：

```
/* end CEC message transmission */
cec_transmission_end ();
```

### 函数 cec\_listen\_mode\_enable

函数cec\_listen\_mode\_enable描述见下表：

表 3-49. 函数 cec\_listen\_mode\_enable

函数名称	cec_listen_mode_enable
函数原形	void cec_listen_mode_enable (void);
功能描述	使能CEC监听模式
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 CEC listen mode */
```

```
cec_listen_mode_enable ();
```

### 函数 cec\_listen\_mode\_disable

函数cec\_listen\_mode\_disable描述见下表：

**表 3-50. 函数 cec\_listen\_mode\_disable**

函数名称	cec_listen_mode_disable
函数原形	void cec_listen_mode_disable (void);
功能描述	关闭CEC监听模式
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 CEC listen mode */
```

```
cec_listen_mode_disable ();
```

### 函数 cec\_own\_address\_config

函数cec\_own\_address\_config描述见下表：

**表 3-51. 函数 cec\_own\_address\_config**

函数名称	cec_own_address_config
函数原形	void cec_own_address_config(uint32_t address);
功能描述	CEC自身地址配置
先决条件	-
被调用函数	-

输入参数{in}	
address	地址选择
CEC_OWN_ADDRE SS_CLEAR	清除地址
CEC_OWN_ADDRE SSx	自身地址 (x=0-14)
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* configure and clear own address.the controller can be configured to multiple own address
*/
```

```
cec_own_address_config (CEC_OWN_ADDRESS_CLEAR);
```

### 函数 cec\_sft\_config

函数cec\_sft\_config描述见下表:

表 3-52. 函数 cec\_sft\_config

函数名称	cec_sft_config
函数原形	void cec_sft_config(uint32_t sftmopt, uint32_t sft);
功能描述	配置CEC信号空闲时间
先决条件	-
被调用函数	-
输入参数{in}	
sftmopt	SFT开始选项位
CEC_SFT_START_ STAOM	STAOM置1后SFT计数器开始计数
CEC_SFT_START_ LAST	发收/接收结束后SFT计数器自动启动
输入参数{in}	
sft	信号空闲时间
CEC_SFT_PROTO COL_PERIOD	SFT时间跟HDMI-CEC协议描述的一样
CEC_SFT_1POINT 5_PERIOD	1.5个额定数据位周期
CEC_SFT_2POINT 5_PERIOD	2.5个额定数据位周期
CEC_SFT_3POINT 5_PERIOD	3.5个额定数据位周期
CEC_SFT_4POINT	4.5个额定数据位周期

5_PERIOD	
CEC_SFT_5POINT 5_PERIOD	5.5个额定数据位周期
CEC_SFT_6POINT 5_PERIOD	6.5个额定数据位周期
CEC_SFT_7POINT 5_PERIOD	7.5个额定数据位周期
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* configure signal free time and the signal free time counter start option */
```

```
cec_sft_config (CEC_SFT_START_STAOM, CEC_SFT_PROTOCOL_PERIOD);
```

### 函数 cec\_generate\_errorbit\_config

函数cec\_generate\_errorbit\_config描述见下表：

表 3-53. 函数 cec\_generate\_errorbit\_config

函数名称	cec_generate_errorbit_config
函数原形	void cec_generate_errorbit_config(uint32_t broadcast, uint32_t singlecast_lbpe, uint32_t singlecast_bre);
功能描述	配置CEC是否产生错误位
先决条件	-
被调用函数	-
输入参数{in}	
broadcast	广播信息模式下是否产生错误位
CEC_BROADCAST_ERROR_BIT_ON	广播信息模式下产生错误位
CEC_BROADCAST_ERROR_BIT_OFF	广播信息模式下不产生错误位
输入参数{in}	
singlecast_lbpe	在单播模式下监测到BPLE的时候是否产生错误位
CEC_LONG_PERIOD_ERROR_BIT_ON	在单播模式下监测到BPLE时产生错误位
CEC_LONG_PERIOD_ERROR_BIT_OFF	在单播模式下监测到BPLE时不产生错误位
输入参数{in}	
singlecast_bre	在单次传播模式下监测到BRE的时候是否产生错误位
CEC_RISING_PERIOD	在单播模式下监测到BRE的时候产生错误位



OD_ERROR_BIT_ON	
CEC_RISING_PERIOD_ERROR_BIT_ON	在单播模式下监测到BRE的时候不产生错误位
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* configure generate Error-bit when detected some abnormal situation or not */
cec_generate_errorbit_config (CEC_BROADCAST_ERROR_BIT_ON,
CEC_LONG_PERIOD_ERROR_BIT_ON, CEC_RISING_PERIOD_ERROR_BIT_ON);
```

### 函数 cec\_stop\_receive\_bre\_config

函数cec\_stop\_receive\_bre\_config描述见下表：

表 3-54. 函数 cec\_stop\_receive\_bre\_config

函数名称	cec_stop_receive_bre_config
函数原形	void cec_stop_receive_bre_config(uint32_t rxbrestp);
功能描述	监测到位上升错误时是否停止接收信息
先决条件	-
被调用函数	-
输入参数{in}	
rxbrestp	监测到BRE时是否停止接收信息
CEC_STOP_RISING_ERROR_BIT_ON	停止接收BRE下的信息
CEC_STOP_RISING_ERROR_BIT_OFF	不停止接收BRE下的信息，数据位按额定时间采样（1.05ms）
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* whether stop receive message when detected bit rising error */
cec_stop_receive_bre_config (CEC_STOP_RISING_ERROR_BIT_ON);
```

### 函数 cec\_reception\_tolerance\_enable

函数 cec\_reception\_tolerance\_enable 描述见下表：

**表 3-55. 函数 cec\_reception\_tolerance\_enable**

函数名称	cec_reception_tolerance_enable
函数原形	void cec_reception_tolerance_enable (void);
功能描述	CEC扩展接收位时间宽容度
先决条件	-
被调用函数	-
输入参数{in}	
-	-
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* enable reception bit timing tolerance */
```

```
cec_reception_tolerance_enable ();
```

### 函数 cec\_reception\_tolerance\_disable

函数 cec\_reception\_tolerance\_disable 描述见下表：

**表 3-56. 函数 cec\_reception\_tolerance\_disable**

函数名称	cec_reception_tolerance_disable
函数原形	void cec_reception_tolerance_disable (void);
功能描述	CEC标准接收位时间宽容度
先决条件	-
被调用函数	-
输入参数{in}	
-	-
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* disable reception bit timing tolerance */
```

```
cec_reception_tolerance_disable ();
```

### 函数 cec\_data\_send

函数cec\_data\_send描述见下表:

表 3-57. 函数 cec\_data\_send

函数名称	cec_data_send
函数原形	void cec_data_send(uint8_t data);
功能描述	CEC数据发送
先决条件	-
被调用函数	-
输入参数{in}	
data	要发送的数据
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* send a data by the CEC peripheral */
```

```
cec_data_send (0x55);
```

### 函数 cec\_data\_receive

函数cec\_data\_receive描述见下表:

表 3-58. 函数 cec\_data\_receive

函数名称	cec_data_receive
函数原形	uint8_t cec_data_receive(void);
功能描述	CEC数据接收
先决条件	-
被调用函数	-
输入参数{in}	
-	-
输出参数{out}	
-	-
返回值	
Uin8_t	CEC接收到的数据值

例如:

```
/* receive a data by the CEC peripheral */
```

```
uint8_t data = 0;
```

```
data = cec_data_receive ();
```

**函数 cec\_interrupt\_enable**

函数cec\_interrupt\_enable描述见下表:

**表 3-59. 函数 cec\_interrupt\_enable**

函数名称	cec_interrupt_enable
函数原形	void cec_interrupt_enable(uint32_t flag);
功能描述	CEC中断使能
先决条件	-
被调用函数	-
<b>输入参数{in}</b>	
<b>flag</b>	中断使能位
CEC_INT_BR	字节接收中断使能
CEC_INT_REND	接收结束中断使能
CEC_INT_RO	接收过载中断使能
CEC_INT_BRE	位上升错误中断使能
CEC_INT_BPSE	短位周期错误中断使能
CEC_INT_BPLE	长位周期错误中断使能
CEC_INT_RAE	接收ACK错误中断使能
CEC_INT_ARBF	仲裁失败中断使能
CEC_INT_TBR	发送字节数据请求中断使能
CEC_INT_TEND	发送成功结束中断使能
CEC_INT_TU	发送数据缓冲区欠载中断使能
CEC_INT_TERR	中断使能中断使能
CEC_INT_TAERR	发送ACK错误中断使能
<b>输出参数{out}</b>	
-	-
<b>返回值</b>	
-	-

例如:

```
/* enable CEC_INT_BR interrupt */
cec_interrupt_enable (CEC_INT_BR);
```

**函数 cec\_interrupt\_disable**

函数cec\_interrupt\_disable描述见下表:

**表 3-60. 函数 cec\_interrupt\_disable**

函数名称	cec_interrupt_disable
函数原形	void cec_interrupt_disable (uint32_t flag);
功能描述	CEC中断禁止
先决条件	-

被调用函数	-
输入参数{in}	
flag	中断禁止位
CEC_INT_BR	字节接收中断禁止
CEC_INT_REND	接收结束中断禁止
CEC_INT_RO	接收过载中断禁止
CEC_INT_BRE	位上升错误中断禁止
CEC_INT_BPSE	短位周期错误中断禁止
CEC_INT_BPLE	长位周期错误中断禁止
CEC_INT_RAE	接收ACK错误中断禁止
CEC_INT_ARBF	仲裁失败中断禁止
CEC_INT_TBR	发送字节数据请求中断禁止
CEC_INT_TEND	发送成功结束中断禁止
CEC_INT_TU	发送数据缓冲区欠载中断禁止
CEC_INT_TERR	中断使能中断禁止
CEC_INT_TAERR	发送ACK错误中断禁止
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* disable CEC_INT_BR interrupt */
cec_interrupt_disable (CEC_INT_BR);
```

## 函数 cec\_flag\_get

函数cec\_flag\_get描述见下表：

表 3-61. 函数 cec\_flag\_get

函数名称	cec_flag_get
函数原形	FlagStatus cec_flag_get(uint32_t flag);
功能描述	CEC标志位获取
先决条件	-
被调用函数	-
输入参数{in}	
flag	标志位
CEC_FLAG_BR	字节接收标志
CEC_FLAG_REND	中断使能标志
CEC_FLAG_RO	接收过载标志
CEC_FLAG_BRE	位上升错误标志
CEC_FLAG_BPSE	短位周期错误标志

CEC_FLAG_BPLE	长位周期错误标志
CEC_FLAG_RAE	接收ACK错误标志
CEC_FLAG_ARBF	仲裁失败标志
CEC_FLAG_TBR	发送字节数据请求标志
CEC_FLAG_TEND	发送成功结束标志
CEC_FLAG_TU	发送数据缓冲区欠载标志
CEC_FLAG_TERR	发送错误标志
CEC_FLAG_TAER R	发送ACK错误标志
输出参数{out}	
-	-
返回值	
FlagStatus	SET或者RESET

例如:

```
/* get CEC_FLAG_BR status */
```

```
FlagStatus flag = reset;
```

```
flag = cec_flag_get (CEC_INT_BR);
```

### 函数 cec\_flag\_clear

函数cec\_flag\_clear描述见下表:

表 3-62. 函数 cec\_flag\_clear

函数名称	cec_flag_clear
函数原形	void cec_flag_clear(uint32_t flag);
功能描述	CEC标志位清除
先决条件	-
被调用函数	-
输入参数{in}	
flag	标志位
CEC_FLAG_BR	字节接收标志
CEC_FLAG_REND	接收结束标志
CEC_FLAG_RO	接收过载标志
CEC_FLAG_BRE	位上升错误标志
CEC_FLAG_BPSE	短位周期错误标志
CEC_FLAG_BPLE	长位周期错误标志
CEC_FLAG_RAE	接收ACK错误标志
CEC_FLAG_ARBF	仲裁失败标志
CEC_FLAG_TBR	发送字节数据请求标志
CEC_FLAG_TEND	发送成功结束标志
CEC_FLAG_TU	发送数据缓冲区欠载标志

CEC_FLAG_TERR	中断使能标志
CEC_FLAG_TAER R	发送ACK错误标志
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* clear CEC_FLAG_BR status */
```

```
cec_flag_get (CEC_INT_BR);
```

### 函数 cec\_interrupt\_flag\_get

函数cec\_interrupt\_flag\_get描述见下表:

表 3-63. 函数 cec\_interrupt\_flag\_get

函数名称	cec_interrupt_flag_get
函数原形	FlagStatus cec_interrupt_flag_get(uint32_t flag);
功能描述	CEC中断标志位获取
先决条件	-
被调用函数	-
输入参数{in}	
flag	中断标志位
CEC_INT_FLAG_BR R	字节接收中断标志
CEC_INT_FLAG_R END	中断使能中断标志
CEC_INT_FLAG_R O	接收过载中断标志
CEC_INT_FLAG_B RE	位上升错误中断标志
CEC_INT_FLAG_B PSE	短位周期错误中断标志
CEC_INT_FLAG_B PLE	长位周期错误中断标志
CEC_INT_FLAG_R AE	接收ACK错误中断标志
CEC_INT_FLAG_A RBF	仲裁失败中断标志
CEC_INT_FLAG_T BR	发送字节数据请求中断标志
CEC_INT_FLAG_T	发送成功结束中断标志

END	
CEC_INT_FLAG_T U	发送数据缓冲区欠载中断标志
CEC_INT_FLAG_T ERR	中断使能中断标志
CEC_INT_FLAG_T AERR	发送ACK错误中断标志
输出参数{out}	
-	-
返回值	
FlagStatus	SET或者RESET

例如:

```
/* get CEC_INT_FLAG_BR status */
```

```
FlagStatus flag = reset;
```

```
flag = cec_interrupt_flag_get (CEC_INT_FLAG_BR);
```

### 函数 cec\_interrupt\_flag\_clear

函数cec\_interrupt\_flag\_clear描述见下表:

表 3-64. 函数 cec\_interrupt\_flag\_clear

函数名称	cec_interrupt_flag_clear
函数原形	void cec_interrupt_flag_clear(uint32_t flag);
功能描述	CEC中断标志位清除
先决条件	-
被调用函数	-
输入参数{in}	
flag	中断标志位
CEC_INT_FLAG_B R	字节接收中断标志
CEC_INT_FLAG_R END	中断使能中断标志
CEC_INT_FLAG_R O	接收过载中断标志
CEC_INT_FLAG_B RE	位上升错误中断标志
CEC_INT_FLAG_B PSE	短位周期错误中断标志
CEC_INT_FLAG_B PLE	长位周期错误中断标志
CEC_INT_FLAG_R	接收ACK错误中断标志



AE	
CEC_INT_FLAG_A RBF	仲裁失败中断标志
CEC_INT_FLAG_T BR	发送字节数据请求中断标志
CEC_INT_FLAG_T END	发送成功结束中断标志
CEC_INT_FLAG_T U	发送数据缓冲区欠载中断标志
CEC_INT_FLAG_T ERR	中断使能中断标志
CEC_INT_FLAG_T AERR	发送ACK错误中断标志
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* clear CEC_INT_FLAG_BR status */
cec_interrupt_flag_clear(CEC_INT_FLAG_BR);
```

## 3.4. 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，结合DAC和TIMER的PWM输出，可以实现电流控制。章节[3.4.1](#)描述了CMP的寄存器列表，章节[3.4.2](#)对CMP库函数进行说明。

### 3.4.1. 外设寄存器说明

CMP寄存器列表如下表所示：

表 3-65. CMP 寄存器

寄存器名称	寄存器描述
CMP_CS	CMP控制状态寄存器

### 3.4.2. 外设库函数说明

CMP库函数列表如下表所示：

表 3-66. 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_enable	使能CMP
cmp_disable	禁能CMP
cmp_switch_enable	CMP开关模式使能
cmp_switch_disable	CMP开关模式禁能
cmp_window_enable	CMP窗口模式使能
cmp_window_disable	CMP窗口模式禁能
cmp_lock_enable	锁定CMP
cmp_output_level_get	获取CMP输出状态

### 枚举类型 cmp\_enum

表 3-67. 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 3-68. 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3-67. 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 3-69. 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3-67. 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_HIGH SPEED	高速/全功耗
CMP_MODE_MIDD LESPEED	中速/中功耗
CMP_MODE_LOW SPEED	低速/低功耗
CMP_MODE_VERY LOWSPEED	超低速/超低功耗
输入参数{in}	
inverting_input	反相输入源选择
CMP_INVERTING_I NPUT_1_4VREFIN T	VREFINT *1/4作为输入源
CMP_INVERTING_I NPUT_1_2VREFIN T	VREFINT *1/2作为输入源
CMP_INVERTING_I NPUT_3_4VREFIN T	VREFINT *3/4作为输入源
CMP_INVERTING_I NPUT_VREFINT	VREFINT作为输入源
CMP_INVERTING_I NPUT_DAC0_OUT0	PA4 (DAC0_OUT0) 作为输入源
CMP_INVERTING_I NPUT_PA5	PA5 作为输入源
CMP_INVERTING_I NPUT_PA0_PA2	PA0 作为CMP0输入源, PA2 作为CMP1输入源
输入参数{in}	

<b>output_hysteresis</b>	迟滞水平
<i>CMP_HYSTERESIS_NO</i>	无迟滞
<i>CMP_HYSTERESIS_LOW</i>	低迟滞
<i>CMP_HYSTERESIS_MIDDLE</i>	中迟滞
<i>CMP_HYSTERESIS_HIGH</i>	高迟滞
<b>输出参数{out}</b>	
-	-
<b>返回值</b>	
-	-

例如:

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

```
cmp_mode_init(CMP0, CMP_MODE_HIGHSPEED, CMP_INVERTING_INPUT_1_4VREF1NT, CMP_HYSTERESIS_NO);
```

## 函数 cmp\_output\_init

函数cmp\_output\_init描述见下表:

**表 3-70. 函数 cmp\_output\_init**

<b>函数名称</b>	cmp_output_init
<b>函数原型</b>	void cmp_output_init(cmp_enum cmp_periph, uint32_t output_selection, uint32_t output_polarity);
<b>功能描述</b>	CMP输出初始化
<b>先决条件</b>	-
<b>被调用函数</b>	-
<b>输入参数{in}</b>	
<b>cmp_periph</b>	参考枚举 <a href="#">表3-67. 枚举类型cmp_enum</a>
<b>输入参数{in}</b>	
<b>output_selection</b>	CMP输出选择
<i>CMP_OUTPUT_NONE</i>	输出无选择
<i>CMP_OUTPUT_TIMER0_BRKIN</i>	输出选择TIMER0_BRKIN
<i>CMP_OUTPUT_TIMER0_IC0</i>	输出选择TIMER0_IC0
<i>CMP_OUTPUT_TIMER0_OCPRECLR</i>	输出选择TIMER0_OCPRECLR

<code>CMP_OUTPUT_TIMER1_IC3</code>	输出选择TIMER1_IC3
<code>CMP_OUTPUT_TIMER1_OCPRECLR</code>	输出选择TIMER1_OCPRECLR
<code>CMP_OUTPUT_TIMER2_IC0</code>	输出选择TIMER2_IC0
<code>CMP_OUTPUT_TIMER2_OCPRECLR</code>	输出选择TIMER2_OCPRECLR
输入参数{in}	
<b>output_polarity</b>	CMP输出极性
<code>CMP_OUTPUT_POLARITY_INVERTED</code>	输出反相
<code>CMP_OUTPUT_POLARITY_NONINVERTED</code>	输出正相
输出参数{out}	
-	-
返回值	
-	-

例如：

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

```
cmp_output_init(CMP0, CMP_OUTPUT_TIMER0_IC0, CMP_OUTPUT_POLARITY_NONINVERTED);
```

## 函数 cmp\_enable

函数cmp\_enable描述见下表：

表 3-71. 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}	
<b>cmp_periph</b>	参考枚举 <a href="#">表3-67. 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 3-72. 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3-67. 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 3-73. 函数 cmp\_switch\_enable**

函数名称	cmp_switch_enable
函数原型	void 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 switch mode */
```

```
cmp_switch_enable();
```

### 函数 cmp\_switch\_disable

函数cmp\_switch\_disable描述见下表：

**表 3-74. 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 switch mode */
```

```
cmp_switch_enable();
```

### 函数 cmp\_window\_enable

函数cmp\_window\_enable描述见下表：

**表 3-75. 函数 cmp\_window\_enable**

函数名称	cmp_window_enable
函数原型	void cmp_window_enable(void);
功能描述	使能CMP窗口模式
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 window mode */
```

```
cmp_window_enable();
```

### 函数 cmp\_window\_disable

函数cmp\_window\_disable描述见下表:

表 3-76. 函数 cmp\_window\_disable

函数名称	cmp_window_disable
函数原型	void cmp_window_disable(void);
功能描述	禁能CMP窗口模式
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 window mode */
```

```
cmp_window_disable();
```

### 函数 cmp\_lock\_enable

函数cmp\_lock\_enable描述见下表:

表 3-77. 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3-67. 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\_output\_level\_get**

函数cmp\_output\_level\_get描述见下表:

**表 3-78. 函数 cmp\_output\_level\_get**

函数名称	cmp_output_level_get
函数原型	uint32_t cmp_output_level_get(uint32_t 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3-67. 枚举类型cmp_enum</a>
输出参数{out}	
-	-
返回值	
uint32_t	输出电平
CMP_OUTPUTLEV EL_HIGH	比较器输出高电平
CMP_OUTPUTLEV EL_LOW	比较器输出低电平

例如:

```
uint32_t level;

/* get CMP0 output level */

level = cmp_output_level_get(CMP0);
```

## 3.5. CRC

循环冗余校验码是一种用在数字网络和存储设备上的差错校验码，可以校验原始数据的偶然误差。章节[3.5.1](#)描述了CRC的寄存器列表，章节[3.5.2](#)对CRC库函数进行说明。

### 3.5.1. 外设寄存器说明

CRC寄存器列表如下表所示:

**表 3-79. CRC 寄存器**

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

### 3.5.2. 外设库函数说明

CRC库函数列表如下表所示：

**表 3-80. CRC 库函数**

库函数名称	库函数描述
crc_deinit	复位CRC计算单元
crc_reverse_output_data_enable	使能输出数据翻转功能
crc_reverse_output_data_disable	失能输出数据翻转功能
crc_data_register_reset	根据数据寄存器的复位值（0xFFFFFFFF）复位数据寄存器
crc_data_register_read	读数据寄存器
crc_free_data_register_read	读独立数据寄存器
crc_free_data_register_write	写独立数据寄存器
crc_init_data_register_write	写初值寄存器
crc_input_data_reverse_config	配置输入数据翻转功能
crc_polynomial_size_set	配置多项式长度
crc_polynomial_set	设置多项式寄存器数据
crc_single_data_calculate	CRC计算一个32位数据
crc_block_data_calculate	CRC计算一个32位数组

#### 函数 crc\_deinit

函数crc\_deinit描述见下表：

**表 3-81. 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_reverse_output_data_enable`

函数 `crc_reverse_output_data_enable` 描述见下表：

**表 3-82. 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_reverse_output_data_disable`

函数 `crc_reverse_output_data_disable` 描述见下表：

**表 3-83. 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_data_register_reset`

函数 `crc_data_register_reset` 描述见下表：

**表 3-84. 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_data_register_read`

函数 `crc_data_register_read` 描述见下表：

**表 3-85. 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 3-86. 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 3-87. 函数 `crc_free_data_register_write`**

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

例如：

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

crc_free_data_register_write(0x11);
```

### 函数 `crc_init_data_register_write`

函数 `crc_init_data_register_write` 描述见下表：

**表 3-88. 函数 `crc_init_data_register_write`**

函数名称	<code>crc_init_data_register_write</code>
函数原形	<code>void crc_init_data_register_write(uint32_t init_data)</code>
功能描述	写初值寄存器
先决条件	-
被调用函数	-
输入参数{in}	
<code>init_data</code>	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_input_data_reverse_config`

函数 `crc_input_data_reverse_config` 描述见下表：

**表 3-89. 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}	
<code>data_reverse</code>	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_polynomial_size_set`

函数 `crc_polynomial_size_set` 描述见下表：

**表 3-90. 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}	
<code>poly_size</code>	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
<code>CRC_CTL_PS_8</code>	8位多项式值用于CRC计算
<code>CRC_CTL_PS_7</code>	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 3-91. 函数 `crc_polynomial_set`**

函数名称	<code>crc_polynomial_set</code>
函数原形	<code>void crc_polynomial_set(uint32_t poly)</code>

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 3-92. 函数 crc\_single\_data\_calculate

函数名称	crc_single_data_calculate
函数原形	uint32_t crc_single_data_calculate(uint32_t sdata);
功能描述	CRC计算一个32位数据
先决条件	-
被调用函数	-
输入参数{in}	
sdata	设定的32位数据
输出参数{out}	
-	-
返回值	
uint32_t	32位CRC计算结果 (0-0xFFFFFFFF)

例如：

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

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

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

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

### 函数 crc\_block\_data\_calculate

函数crc\_block\_data\_calculate描述见下表：



表 3-93. 函数 `crc_block_data_calculate`

函数名称	<code>crc_block_data_calculate</code>
函数原形	<code>uint32_t crc_block_data_calculate(uint32_t array[], uint32_t size);</code>
功能描述	CRC计算一个32位数组
先决条件	-
被调用函数	-
输入参数{in}	
<b>array</b>	32位数据数组的指针
输入参数{in}	
<b>size</b>	数据长度
输出参数{out}	
-	-
返回值	
<b>uint32_t</b>	32位CRC计算结果 (0-0xFFFFFFFF)

例如:

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

#define BUFFER_SIZE    6

uint32_t valcrc = 0;

static const uint32_t data_buffer[BUFFER_SIZE] = {

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

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

## 3.6. CTC

CTC模块基于外部高精度的参考信号源来校准IRC48M的时钟频率，通过自动的或手动的调整校准值，以得到一个精准的IRC48M时钟。章节[3.6.1](#)描述了CTC的寄存器列表，章节[3.6.2](#)对CTC库函数进行说明。

### 3.6.1. 外设寄存器描述

CTC寄存器列表如下表所示:

表 3-94. CTC 寄存器

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

### 3.6.2. 外设库函数说明

CTC库函数列表如下表所示：

**表 3-95. CTC 库函数**

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

#### 函数 ctc\_deinit

函数ctc\_deinit描述见下表：

**表 3-96. 函数 ctc\_deinit**

函数名称	ctc_deinit
函数原形	void ctc_deinit (void)
功能描述	复位CTC单元
先决条件	-
被调用函数	rcu_periph_reset_enable / rcu_periph_reset_disable
输入参数{in}	
-	-
输出参数{out}	
-	-

返回值	
-	-

例如：

```
/* reset CTC */
```

```
ctc_deinit();
```

### 函数 ctc\_refsource\_polarity\_config

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

表 3-97. 函数 ctc\_refsource\_polarity\_config

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

例如：

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

```
ctc_refsource_polarity_config (CTC_REFSOURCE_POLARITY_RISING);
```

### 函数 ctc\_refsource\_signal\_select

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

表 3-98. 函数 ctc\_refsource\_signal\_select

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

<b>refs</b>	参考信号源
<i>CTC_REFSOURCE_G PIO</i>	选择GPIO输入信号
<i>CTC_REFSOURCE_LX TAL</i>	选择LXTAL时钟
<b>输出参数{out}</b>	
-	-
<b>返回值</b>	
-	-

例如:

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

```
ctc_refsource_signal_select (CTC_REFSOURCE_LXTAL);
```

### 函数 **ctc\_refsource\_prescaler\_config**

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

**表 3-99. 函数 ctc\_refsource\_prescaler\_config**

<b>函数名称</b>	ctc_refsource_prescaler_config
<b>函数原形</b>	void ctc_refsource_prescaler_config(uint32_t prescaler);
<b>功能描述</b>	参考信号源的分频设置
<b>先决条件</b>	-
<b>被调用函数</b>	-
<b>输入参数{in}</b>	
<b>prescaler</b>	分频系数
<i>CTC_REFSOURCE_P SC_OFF</i>	参考信号不分频
<i>CTC_REFSOURCE_P SC_DIV2</i>	参考信号2分频
<i>CTC_REFSOURCE_P SC_DIV4</i>	参考信号4分频
<i>CTC_REFSOURCE_P SC_DIV8</i>	参考信号8分频
<i>CTC_REFSOURCE_P SC_DIV16</i>	参考信号16分频
<i>CTC_REFSOURCE_P SC_DIV32</i>	参考信号32分频
<i>CTC_REFSOURCE_P SC_DIV64</i>	参考信号64分频
<i>CTC_REFSOURCE_P SC_DIV128</i>	参考信号128分频
<b>输出参数{out}</b>	

-	-
返回值	
-	-

例如：

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

```
ctc_refsource_prescaler_config(CTC_REFSOURCE_PSC_DIV2);
```

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

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

表 3-100. 函数 ctc\_clock\_limit\_value\_config

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

例如：

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

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

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

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

表 3-101. 函数 ctc\_counter\_reload\_value\_config

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

返回值	
-	-

例如：

```
/* configure CTC counter reload value */
```

```
ctc_counter_reload_value_config (0x00FF);
```

### 函数 ctc\_counter\_enable

函数ctc\_counter\_enable描述见下表：

**表 3-102. 函数 ctc\_counter\_enable**

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

例如：

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

```
ctc_counter_enable ();
```

### 函数 ctc\_counter\_disable

函数ctc\_counter\_disable描述见下表：

**表 3-103. 函数 ctc\_counter\_disable**

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

-	-
---	---

例如:

```
/* disable CTC trim counter */
```

```
ctc_counter_disable ();
```

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

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

**表 3-104. 函数 ctc\_irc48m\_trim\_value\_config**

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

例如:

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

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

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

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

**表 3-105. 函数 ctc\_software\_refsource\_pulse\_generate**

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

例如：

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

```
ctc_software_refsource_pulse_generate ();
```

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

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

表 3-106. 函数 ctc\_hardware\_trim\_mode\_config

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

例如：

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

```
ctc_hardware_trim_mode_config (CTC_HARDWARE_TRIM_MODE_ENABLE);
```

### 函数 ctc\_counter\_capture\_value\_read

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

表 3-107. 函数 ctc\_counter\_capture\_value\_read

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



返回值	
uint16_t	读取计数器捕获值(0x0000 - 0xFFFF)

例如:

```
/* read CTC counter capture value */

uint16_t ctc_value = 0;

ctc_value = ctc_counter_capture_value_read ();
```

### 函数 ctc\_counter\_direction\_read

函数ctc\_counter\_direction\_read描述见下表:

表 3-108. 函数 ctc\_counter\_direction\_read

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

例如:

```
/* read ctc counter direction */

FlagStatus ctc_direction = SET;

ctc_direction = ctc_counter_direction_read ();
```

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

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

表 3-109. 函数 ctc\_counter\_reload\_value\_read

函数名称	ctc_counter_reload_value_read
函数原形	uint16_t ctc_counter_reload_value_read(void);
功能描述	读取CTC计数器重载值
先决条件	-
被调用函数	-
输入参数{in}	
-	-

输出参数{out}	
-	-
返回值	
uint16_t	读取计数器重载值的16位数据 (0x0000 - 0xFFFF)

例如:

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

uint16_t ctc_reload_value = 0;

ctc_reload_value = ctc_counter_reload_value_read ();
```

### 函数 ctc\_irc48m\_trim\_value\_read

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

表 3-110. 函数 ctc\_irc48m\_trim\_value\_read

函数名称	ctc_irc48m_trim_value_read
函数原形	uint8_t ctc_irc48m_trim_value_read(void);
功能描述	读IRC48M校准值
先决条件	-
被调用函数	-
输入参数{in}	
-	-
输出参数{out}	
-	-
返回值	
uint8_t	6位IRC48M校准值 (0-63)

例如:

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

uint8_t ctc_trim_value = 0;

ctc_trim_value = ctc_irc48m_trim_value_read ();
```

### 函数 ctc\_interrupt\_enable

函数ctc\_interrupt\_enable描述见下表:

表 3-111. 函数 ctc\_interrupt\_enable

函数名称	ctc_interrupt_enable
函数原形	void ctc_interrupt_enable(uint32_t interrupt);
功能描述	使能外设CTC中断
先决条件	-
被调用函数	-

输入参数{in}	
<b>interrupt</b>	CTC中断
<i>CTC_INT_CKOK</i>	时钟校准完成中断
<i>CTC_INT_CKWARN</i>	时钟校准警告中断
<i>CTC_INT_ERR</i>	错误中断
<i>CTC_INT_EREFP</i>	期望参考信号中断
输出参数{out}	
-	-
返回值	
-	-

例如：

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

### 函数 **ctc\_interrupt\_disable**

函数ctc\_interrupt\_disable描述见下表：

**表 3-112. 函数 ctc\_interrupt\_disable**

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

例如：

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

### 函数 **ctc\_flag\_get**

函数ctc\_flag\_get描述见下表：

表 3-113. 函数 `ctc_flag_get`

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

例如：

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

```
FlagStatus state = ctc_flag_get (CTC_FLAG_CKOK);
```

### 函数 `ctc_flag_clear`

函数`ctc_flag_clear`描述见下表：

表 3-114. 函数 `ctc_flag_clear`

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

<i>CTC_FLAG_TRIMERR</i>	校准值错误位
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* clear CTC flag status */
ctc_flag_clear (CTC_FLAG_CKOK);
```

### 函数 **ctc\_interrupt\_flag\_get**

函数ctc\_interrupt\_flag\_get描述见下表:

**表 3-115. 函数 ctc\_interrupt\_flag\_get**

函数名称	ctc_interrupt_flag_get
函数原形	FlagStatus ctc_interrupt_flag_get(uint32_t int_flag);
功能描述	获取CTC中断标志位
先决条件	-
被调用函数	-
输入参数{in}	
<b>int_flag</b>	CTC中断标志
<i>CTC_INT_FLAG_CKOK</i>	时钟校准完成中断标志位
<i>CTC_INT_FLAG_CKWARN</i>	时钟校准警告中断标志位
<i>CTC_INT_FLAG_ERR</i>	错误中断标志位
<i>CTC_INT_FLAG_EREFP</i>	期望参考信号中断标志位
<i>CTC_INT_FLAG_CKEERR</i>	时钟校准错误位
<i>CTC_INT_FLAG_REFMISS</i>	参考同步脉冲信号丢失
<i>CTC_INT_FLAG_TRIMERR</i>	校准值错误位
输出参数{out}	
-	-
返回值	
<b>FlagStatus</b>	SET或RESET

例如:

```
/* get CTC interrupt flag status */
FlagStatus state = ctc_interrupt_flag_get (CTC_INT_FLAG_CKOK);
```

**函数 ctc\_interrupt\_flag\_clear**

函数ctc\_interrupt\_flag\_clear描述见下表：

**表 3-116. 函数 ctc\_interrupt\_flag\_clear**

函数名称	ctc_interrupt_flag_clear
函数原形	void ctc_interrupt_flag_clear(uint32_t int_flag);
功能描述	清除CTC中断标志位
先决条件	-
被调用函数	-
输入参数{in}	
int_flag	CTC中断标志
CTC_INT_FLAG_CKOK	时钟校准完成中断标志位
CTC_INT_FLAG_CKWARN	时钟校准警告中断标志位
CTC_INT_FLAG_ERR	错误中断标志位
CTC_INT_FLAG_EREFP	期望参考信号中断标志位
CTC_INT_FLAG_CKEERR	时钟校准错误位
CTC_INT_FLAG_REFMISS	参考同步脉冲信号丢失
CTC_INT_FLAG_TRIMERR	校准值错误位
输出参数{out}	
-	-
返回值	
-	-

例如：

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

```
ctc_interrupt_flag_clear (CTC_INT_FLAG_CKOK);
```

## 3.7. DAC

数字/模拟转换器可以将12位的数字数据转换为外部引脚上的电压输出，章节[3.7.1](#)描述了DAC的寄存器列表，章节[3.7.2](#)对DAC库函数进行说明。

### 3.7.1. 外设寄存器说明

DAC寄存器列表如下表所示：

表 3-117. DAC 寄存器

寄存器名称	寄存器描述
DAC_CTL0	DACx控制寄存器
DAC_SWT	DACx软件触发寄存器
DAC_OUT0_R12DH	DAC_OUT0 12位右对齐数据保持寄存器
DAC_OUT0_L12DH	DAC_OUT0 12位左对齐数据保持寄存器
DAC_OUT0_R8DH	DAC_OUT0 8位右对齐数据保持寄存器
DAC_OUT0_DO	DAC_OUT0数据输出寄存器
DAC_STAT0	DAC状态寄存器0

### 3.7.2. 外设库函数说明

DAC库函数列表如下表所示：

表 3-118. DAC 库函数

库函数名称	库函数描述
dac_deinit	DAC外设复位
dac_enable	DAC使能
dac_disable	DAC禁能
dac_dma_enable	DAC的DMA功能使能
dac_dma_disable	DAC的DMA功能禁能
dac_output_buffer_enable	DAC输出缓冲区使能
dac_output_buffer_disable	DAC输出缓冲区禁能
dac_output_value_get	DAC输出数据获取
dac_data_set	DAC输出数据设置
dac_trigger_enable	DAC触发使能
dac_trigger_disable	DAC触发禁能
dac_trigger_source_config	DAC触发源选择
dac_software_trigger_enable	DAC软件触发使能
dac_wave_mode_config	DAC噪声波模式选择
dac_lfsr_noise_config	DAC LFSR模式设置
dac_triangle_noise_config	DAC三角波模式设置
dac_flag_get	DAC标志位获取
dac_flag_clear	DAC标志位清除
dac_interrupt_enable	DAC中断使能
dac_interrupt_disable	DAC中断禁能
dac_interrupt_flag_get	DAC中断标志位获取
dac_interrupt_flag_clear	DAC中断标志位清除

#### 函数 dac\_deinit

函数dac\_deinit描述见下表：

表 3-119. 函数 **dac\_deinit**

函数名称	dac_deinit
函数原型	void dac_deinit(uint32_t dac_periph);
功能描述	DAC外设复位
先决条件	-
被调用函数	rcu_periph_reset_enable / rcu_periph_reset_disable
输入参数{in}	
dac_periph	DAC外设
DACx	DAC外设选择 (x = 0)
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* deinitialize DAC0 */
```

```
dac_deinit(DAC0);
```

### 函数 **dac\_enable**

函数dac\_enable描述见下表:

表 3-120. 函数 **dac\_enable**

函数名称	dac_enable
函数原型	void dac_enable(uint32_t dac_periph, uint8_t dac_out);
功能描述	DAC使能
先决条件	-
被调用函数	-
输入参数{in}	
dac_periph	DAC外设
DACx	DAC外设选择 (x = 0)
输入参数{in}	
dac_out	DAC输出
DAC_OUTx	DAC输出通道选择 (x = 0)
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* enable DAC0_OUT0 */
```

```
dac_enable(DAC0, DAC_OUT0);
```



## 函数 `dac_disable`

函数 `dac_disable` 描述见下表：

**表 3-121. 函数 `dac_disable`**

函数名称	<code>dac_disable</code>
函数原型	<code>void dac_disable(uint32_t dac_periph, uint8_t dac_out);</code>
功能描述	DAC禁能
先决条件	-
被调用函数	-
输入参数{in}	
<code>dac_periph</code>	DAC外设
<code>DACx</code>	DAC外设选择（ <code>x = 0</code> ）
输入参数{in}	
<code>dac_out</code>	DAC输出
<code>DAC_OUTx</code>	DAC输出通道选择（ <code>x = 0</code> ）
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* disable DAC0_OUT0 */
dac_disable(DAC0, DAC_OUT0);
```

## 函数 `dac_dma_enable`

函数 `dac_dma_enable` 描述见下表：

**表 3-122. 函数 `dac_dma_enable`**

函数名称	<code>dac_dma_enable</code>
函数原型	<code>void dac_dma_enable(uint32_t dac_periph, uint8_t dac_out);</code>
功能描述	DAC的DMA功能使能
先决条件	-
被调用函数	-
输入参数{in}	
<code>dac_periph</code>	DAC外设
<code>DACx</code>	DAC外设选择（ <code>x = 0</code> ）
输入参数{in}	
<code>dac_out</code>	DAC输出
<code>DAC_OUTx</code>	DAC输出通道选择（ <code>x = 0</code> ）
输出参数{out}	
-	-

返回值	
-	-

例如:

```
/* enable DAC0_OUT0 DMA function */
dac_dma_enable(DAC0, DAC_OUT0);
```

### 函数 **dac\_dma\_disable**

函数dac\_dma\_disable描述见下表:

**表 3-123. 函数 dac\_dma\_disable**

函数名称	dac_dma_disable
函数原型	void dac_dma_disable(uint32_t dac_periph, uint8_t dac_out);
功能描述	DAC的DMA功能禁能
先决条件	-
被调用函数	-
输入参数{in}	
<b>dac_periph</b>	DAC外设
<i>DACx</i>	DAC外设选择 (x = 0)
输入参数{in}	
<b>dac_out</b>	DAC输出
<i>DAC_OUTx</i>	DAC输出通道选择 (x = 0)
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* disable DAC0_OUT0 DMA function */
dac_dma_disable(DAC0, DAC_OUT0);
```

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

函数dac\_output\_buffer\_enable描述见下表:

**表 3-124. 函数 dac\_output\_buffer\_enable**

函数名称	dac_output_buffer_enable
函数原型	void dac_output_buffer_enable(uint32_t dac_periph, uint8_t dac_out);
功能描述	DAC输出缓冲区使能
先决条件	-
被调用函数	-
输入参数{in}	

<b>dac_periph</b>	DAC外设
<i>DACx</i>	DAC外设选择 (x = 0)
<b>输入参数{in}</b>	
<b>dac_out</b>	DAC输出
<i>DAC_OUTx</i>	DAC输出通道选择 (x = 0)
<b>输出参数{out}</b>	
-	-
<b>返回值</b>	
-	-

例如:

```
/* enable DAC0_OUT0 output buffer */
dac_output_buffer_enable(DAC0, DAC_OUT0);
```

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

函数dac\_output\_buffer\_disable描述见下表:

**表 3-125. 函数 **dac\_output\_buffer\_disable****

<b>函数名称</b>	dac_output_buffer_disable
<b>函数原型</b>	void dac_output_buffer_disable(uint32_t dac_periph, uint8_t dac_out);
<b>功能描述</b>	DAC输出缓冲区禁能
<b>先决条件</b>	-
<b>被调用函数</b>	-
<b>输入参数{in}</b>	
<b>dac_periph</b>	DAC外设
<i>DACx</i>	DAC外设选择 (x = 0)
<b>输入参数{in}</b>	
<b>dac_out</b>	DAC输出
<i>DAC_OUTx</i>	DAC输出通道选择 (x = 0)
<b>输出参数{out}</b>	
-	-
<b>返回值</b>	
-	-

例如:

```
/* disable DAC0_OUT0 output buffer */
dac_output_buffer_disable(DAC0, DAC_OUT0);
```

### 函数 **dac\_output\_value\_get**

函数dac\_output\_value\_get描述见下表:

表 3-126. 函数 `dac_output_value_get`

函数名称	<code>dac_output_value_get</code>
函数原型	<code>uint16_t dac_output_value_get(uint32_t dac_periph, uint8_t dac_out);</code>
功能描述	DAC输出数据获取
先决条件	-
被调用函数	-
输入参数{in}	
<code>dac_periph</code>	DAC外设
<code>DACx</code>	DAC外设选择（ <code>x = 0</code> ）
输入参数{in}	
<code>dac_out</code>	DAC输出
<code>DAC_OUTx</code>	DAC输出通道选择（ <code>x = 0</code> ）
输出参数{out}	
-	-
返回值	
<code>uint16_t</code>	外设DACx数据保持寄存器值（0~4095）

例如:

```
/* get the DAC0_OUT0 last data output value */
```

```
data = dac_output_value_get(DAC0, DAC_OUT0);
```

### 函数 `dac_data_set`

函数`dac_data_set`描述见下表:

表 3-127. 函数 `dac_data_set`

函数名称	<code>dac_data_set</code>
函数原型	<code>void dac_data_set(uint32_t dac_periph, uint8_t dac_out, uint32_t dac_align, uint16_t data);</code>
功能描述	DAC输出数据设置
先决条件	-
被调用函数	-
输入参数{in}	
<code>dac_periph</code>	DAC外设
<code>DACx</code>	DAC外设选择（ <code>x = 0</code> ）
输入参数{in}	
<code>dac_out</code>	DAC输出
<code>DAC_OUTx</code>	DAC输出通道选择（ <code>x = 0</code> ）
输入参数{in}	
<code>dac_align</code>	DAC对齐模式
<code>DAC_ALIGN_12B_R</code>	12位数据右对齐

<i>DAC_ALIGN_12B_L</i>	12位数据左对齐
<i>DAC_ALIGN_8B_R</i>	8位数据右对齐
输入参数{in}	
<b>data</b>	写入DACx的数据（0~4095）
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* set DAC0_OUT0 data holding register value */
dac_data_set(DAC0, DAC_OUT0, DAC_ALIGN_8B_R, 0xFF);
```

### 函数 **dac\_trigger\_enable**

函数dac\_trigger\_enable描述见下表:

表 3-128. 函数 **dac\_trigger\_enable**

函数名称	dac_trigger_enable
函数原型	void dac_trigger_enable(uint32_t dac_periph, uint8_t dac_out);
功能描述	DAC触发使能
先决条件	-
被调用函数	-
输入参数{in}	
<b>dac_periph</b>	DAC外设
<i>DACx</i>	DAC外设选择（x = 0）
输入参数{in}	
<b>dac_out</b>	DAC输出
<i>DAC_OUTx</i>	DAC输出通道选择（x = 0）
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* enable DAC0_OUT0 trigger */
dac_trigger_enable(DAC0, DAC_OUT0);
```

### 函数 **dac\_trigger\_disable**

函数dac\_trigger\_disable描述见下表:

表 3-129. 函数 `dac_trigger_disable`

函数名称	<code>dac_trigger_disable</code>
函数原型	<code>void dac_trigger_disable(uint32_t dac_periph, uint8_t dac_out);</code>
功能描述	DAC触发禁能
先决条件	-
被调用函数	-
输入参数{in}	
<code>dac_periph</code>	DAC外设
<code>DACx</code>	DAC外设选择 ( $x = 0$ )
输入参数{in}	
<code>dac_out</code>	DAC输出
<code>DAC_OUTx</code>	DAC输出通道选择 ( $x = 0$ )
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* disable DAC0_OUT0 trigger */
dac_trigger_disable(DAC0, DAC_OUT0);
```

### 函数 `dac_trigger_source_config`

函数`dac_trigger_source_config`描述见下表:

表 3-130. 函数 `dac_trigger_source_config`

函数名称	<code>dac_trigger_source_config</code>
函数原型	<code>void dac_trigger_source_config(uint32_t dac_periph, uint8_t dac_out, uint32_t triggersource);</code>
功能描述	DAC触发源选择
先决条件	-
被调用函数	-
输入参数{in}	
<code>dac_periph</code>	DAC外设
<code>DACx</code>	DAC外设选择 ( $x = 0$ )
输入参数{in}	
<code>dac_out</code>	DAC输出
<code>DAC_OUTx</code>	DAC输出通道选择 ( $x = 0$ )
输入参数{in}	
<code>triggersource</code>	DAC触发源
<code>DAC_TRIGGER_T5_TRGO</code>	TIMER5 TRGO

<code>DAC_TRIGGER_T14_TRGO</code>	TIMER14 TRGO
<code>DAC_TRIGGER_T2_TRGO</code>	TIMER2 TRGO
<code>DAC_TRIGGER_T1_TRGO</code>	TIMER1 TRGO
<code>DAC_TRIGGER_EXTI_9</code>	EXTI线9中断
<code>DAC_TRIGGER_SOFTWARE</code>	软件触发
输出参数{out}	
-	-
返回值	
-	-

例如:

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

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

### 函数 `dac_software_trigger_enable`

函数 `dac_software_trigger_enable` 描述见下表:

表 3-131. 函数 `dac_software_trigger_enable`

函数名称	<code>dac_software_trigger_enable</code>
函数原型	<code>void dac_software_trigger_enable(uint32_t dac_periph, uint8_t dac_out);</code>
功能描述	DAC软件触发使能
先决条件	-
被调用函数	-
输入参数{in}	
<code>dac_periph</code>	DAC外设
<code>DACx</code>	DAC外设选择 (x = 0)
输入参数{in}	
<code>dac_out</code>	DAC输出
<code>DAC_OUTx</code>	DAC输出通道选择 (x = 0)
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* enable DAC0_OUT0 software trigger */
```

```
dac_software_trigger_enable(DAC0, DAC_OUT0);
```

### 函数 `dac_wave_mode_config`

函数 `dac_wave_mode_config` 描述见下表：

**表 3-132. 函数 `dac_wave_mode_config`**

函数名称	<code>dac_wave_mode_config</code>
函数原型	<code>void dac_wave_mode_config(uint32_t dac_periph, uint8_t dac_out, uint32_t wave_mode);</code>
功能描述	DAC噪声波模式选择
先决条件	-
被调用函数	-
输入参数{in}	
<code>dac_periph</code>	DAC外设
<code>DACx</code>	DAC外设选择（ <code>x = 0</code> ）
输入参数{in}	
<code>dac_out</code>	DAC输出
<code>DAC_OUTx</code>	DAC输出通道选择（ <code>x = 0</code> ）
输入参数{in}	
<code>wave_mode</code>	噪声波模式选择
<code>DAC_WAVE_DISABLE</code>	噪声波模式禁能
<code>DAC_WAVE_MODE_LFSR</code>	LFSR噪声波模式
<code>DAC_WAVE_MODE_TRIANGLE</code>	三角波噪声波模式
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* configure DAC0_OUT0 wave mode */
```

```
dac_wave_mode_config(DAC0, DAC_OUT0, DAC_WAVE_DISABLE);
```

### 函数 `dac_lfsr_noise_config`

函数 `dac_lfsr_noise_config` 描述见下表：

**表 3-133. 函数 `dac_lfsr_noise_config`**

函数名称	<code>dac_lfsr_noise_config</code>
函数原型	<code>void dac_lfsr_noise_config(uint32_t dac_periph, uint8_t dac_out, uint32_t unmask_bits);</code>



功能描述	DAC LFSR模式设置
先决条件	-
被调用函数	-
输入参数{in}	
<b>dac_periph</b>	DAC外设
<i>DACx</i>	DAC外设选择 ( $x = 0$ )
输入参数{in}	
<b>dac_out</b>	DAC输出
<i>DAC_OUTx</i>	DAC输出通道选择 ( $x = 0$ )
输入参数{in}	
<b>unmask_bits</b>	噪声波的非屏蔽位宽
<i>DAC_LFSR_BIT0</i>	LFSR模式位0非屏蔽
<i>DAC_LFSR_BITSx_0</i>	LFSR模式位[x:0]非屏蔽 ( $x = 1, 2, 3..11$ )
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* configure DAC0_OUT0 LFSR noise mode */
```

```
dac_lfsr_noise_config(DAC0, DAC_OUT0, DAC_LFSR_BIT0);
```

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

函数 **dac\_triangle\_noise\_config** 描述见下表:

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

函数名称	dac_triangle_noise_config
函数原型	void dac_triangle_noise_config(uint32_t dac_periph, uint8_t dac_out, uint32_t amplitude);
功能描述	DAC三角波模式设置
先决条件	-
被调用函数	-
输入参数{in}	
<b>dac_periph</b>	DAC外设
<i>DACx</i>	DAC外设选择 ( $x = 0$ )
输入参数{in}	
<b>dac_out</b>	DAC输出
<i>DAC_OUTx</i>	DAC输出通道选择 ( $x = 0$ )
输入参数{in}	
<b>amplitude</b>	三角波幅值

<i>DAC_TRIANGLE_AMPLITUDE_x</i>	$x = 2^n - 1 \quad (n = 1..12)$
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* configure DAC0_OUT0 triangle noise mode */
```

```
dac_triangle_noise_config(DAC0, DAC_OUT0, DAC_TRIANGLE_AMPLITUDE_1);
```

### 函数 **dac\_flag\_get**

函数 **dac\_flag\_get** 描述见下表:

**表 3-135. 函数 **dac\_flag\_get****

函数名称	dac_flag_get
函数原型	FlagStatus dac_flag_get(uint32_t dac_periph, uint32_t flag);
功能描述	DAC标志位获取
先决条件	-
被调用函数	-
输入参数{in}	
<b>dac_periph</b>	DAC外设
<i>DACx</i>	DAC外设选择 ( $x = 0$ )
输入参数{in}	
<b>flag</b>	DAC状态标志位
<i>DAC_FLAG_DDUDR0</i>	DACx_OUT0 DMA欠载标志位
输出参数{out}	
-	-
返回值	
<b>FlagStatus</b>	DAC位状态 (SET或RESET)

例如:

```
/* get DAC0 flag */
```

```
FlagStatus flag;
```

```
flag = dac_flag_get(DAC0, DAC_FLAG_DDUDR0);
```

### 函数 **dac\_flag\_clear**

函数 **dac\_flag\_clear** 描述见下表:

表 3-136. 函数 `dac_flag_clear`

函数名称	<code>dac_flag_clear</code>
函数原型	<code>void dac_flag_clear(uint32_t dac_periph, uint32_t flag);</code>
功能描述	DAC标志位清除
先决条件	-
被调用函数	-
输入参数{in}	
<code>dac_periph</code>	DAC外设
<code>DACx</code>	DAC外设选择 ( $x = 0$ )
输入参数{in}	
<code>flag</code>	DAC状态标志位
<code>DAC_FLAG_DDUDR0</code>	DACx_OUT0 DMA欠载标志位
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* clear DAC0 flag */
```

```
dac_flag_clear(DAC0, DAC_FLAG_DDUDR0);
```

### 函数 `dac_interrupt_enable`

函数`dac_interrupt_enable`描述见下表:

表 3-137. 函数 `dac_interrupt_enable`

函数名称	<code>dac_interrupt_enable</code>
函数原型	<code>void dac_interrupt_enable(uint32_t dac_periph, uint32_t interrupt);</code>
功能描述	DAC中断使能
先决条件	-
被调用函数	-
输入参数{in}	
<code>dac_periph</code>	DAC外设
<code>DACx</code>	DAC外设选择 ( $x = 0$ )
输入参数{in}	
<code>interrupt</code>	DAC中断
<code>DAC_INT_DDUDR0</code>	DACx_OUT0 DMA欠载中断
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* enable DAC0 interrupt */
```

```
dac_interrupt_enable (DAC0, DAC_INT_DDUDRIE0);
```

### 函数 `dac_interrupt_disable`

函数 `dac_interrupt_disable` 描述见下表:

**表 3-138. 函数 `dac_interrupt_disable`**

函数名称	<code>dac_interrupt_disable</code>
函数原型	<code>void dac_interrupt_disable(uint32_t dac_periph, uint32_t interrupt);</code>
功能描述	DAC中断禁能
先决条件	-
被调用函数	-
输入参数{in}	
<code>dac_periph</code>	DAC外设
<code>DACx</code>	DAC外设选择 ( $x = 0$ )
输入参数{in}	
<code>interrupt</code>	DAC中断
<code>DAC_INT_DDUDRIE0</code>	<code>DACx_OUT0</code> DMA欠载中断
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* disable DAC0 interrupt */
```

```
dac_interrupt_disable (DAC0, DAC_INT_DDUDRIE0);
```

### 函数 `dac_interrupt_flag_get`

函数 `dac_interrupt_flag_get` 描述见下表:

**表 3-139. 函数 `dac_interrupt_flag_get`**

函数名称	<code>dac_interrupt_flag_get</code>
函数原型	<code>FlagStatus dac_interrupt_flag_get(uint32_t dac_periph, uint32_t int_flag);</code>
功能描述	DAC中断标志位获取
先决条件	-
被调用函数	-
输入参数{in}	
<code>dac_periph</code>	DAC外设
<code>DACx</code>	DAC外设选择 ( $x = 0$ )

输入参数{in}	
<b>int_flag</b>	DAC中断标志位
<i>DAC_INT_FLAG_D DUDR0</i>	DACx_OUT0 DMA欠载中断标志位
输出参数{out}	
-	-
返回值	
<b>FlagStatus</b>	DAC中断状态（SET或RESET）

例如:

```
/* get DAC0 interrupt flag */
```

```
flag = dac_interrupt_flag_get(DAC0, DAC_INT_FLAG_DDUDR0);
```

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

函数dac\_interrupt\_flag\_clear描述见下表:

**表 3-140. 函数 **dac\_interrupt\_flag\_clear****

<b>函数名称</b>	dac_interrupt_flag_clear
<b>函数原型</b>	FlagStatus dac_interrupt_flag_clear(uint32_t dac_periph, uint32_t int_flag);
<b>功能描述</b>	DAC中断标志位清除
<b>先决条件</b>	-
<b>被调用函数</b>	-
输入参数{in}	
<b>dac_periph</b>	DAC外设
<i>DACx</i>	DAC外设选择（x = 0）
输入参数{in}	
<b>int_flag</b>	DAC中断标志位
<i>DAC_INT_FLAG_D DUDR0</i>	DACx_OUT0 DMA欠载中断标志位
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* clear DAC0 interrupt flag */
```

```
dac_interrupt_flag_clear(DAC0, DAC_INT_FLAG_DDUDR0);
```

## 3.8. 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[3.8.1](#)描述了DBG的寄

寄存器列表，章节 [3.8.2](#) 对DBG库函数进行说明。

### 3.8.1. 外设寄存器说明

DBG寄存器列表如下表所示：

**表 3-141. 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

### 3.8.2. 外设库函数说明

DBG库函数列表如下表所示：

**表 3-142. 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 3-143. 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_TIMER1_HOLD	当内核停止时，保持TIMER1计数器计数值不变
DBG_TIMER2_HOLD	当内核停止时，保持TIMER2计数器计数值不变
DBG_TIMER5_HOLD	当内核停止时，保持TIMER5计数器计数值不变
DBG_TIMER13_HOLD	当内核停止时，保持TIMER13计数器计数值不变
DBG_TIMER14_HOLD	当内核停止时，保持TIMER14计数器计数值不变
DBG_TIMER15_HOLD	当内核停止时，保持TIMER15计数器计数值不变
DBG_TIMER16_HOLD	当内核停止时，保持TIMER16计数器计数值不变
DBG_I2C0_HOLD	当内核停止时，保持I2C0的SMBUS状态不变，用于调试
DBG_I2C1_HOLD	当内核停止时，保持I2C1的SMBUS状态不变，用于调试
DBG_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 3-144. 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 3-145. 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 3-146. 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 3-147. 函数 dbg\_low\_power\_disable

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



DBG_LOW_POWER_STANDBY	在待机模式下，保持调试器连接，可进行调试
输出参数{out}	
-	-
返回值	
-	-

例如：

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

```
dbg_low_power_disable(DBG_LOW_POWER_SLEEP);
```

### 函数 dbg\_periph\_enable

函数dbg\_periph\_enable描述见下表：

表 3-148. 函数 dbg\_periph\_enable

函数名称	dbg_periph_enable
函数原形	void dbg_periph_enable(dbg_periph_enum dbg_periph);
功能描述	使能外设的MCU调试保持功能
先决条件	-
被调用函数	-
输入参数{in}	
dbg_periph	请参考 <a href="#">表3-143. 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, 1, 2, 5, 13, 14, 15, 16）
DBG_I2Cx_HOLD	当内核停止时，保持I2Cx（x=0, 1）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_TIMER0_HOLD);
```

### 函数 dbg\_periph\_disable

函数dbg\_periph\_disable描述见下表：

表 3-149. 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	请参考 <a href="#">表3-143. 枚举类型dbg_periph_enum</a>
DBG_FWDGT_HOLD D	当内核停止时，保持FWDGT计数器时钟
DBG_WWDGT_HOLD LD	当内核停止时，保持WWDGT计数器时钟
DBG_TIMERx_HOLD D	当内核停止时，保持TIMERx计数器计数值不变（x=0, 1, 2, 5, 13, 14, 15, 16）
DBG_I2Cx_HOLD	当内核停止时，保持I2Cx（x=0,1）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_TIMER0_HOLD);
```

## 3.9. DMA

DMA控制器提供了一种硬件的方式在外设和存储器之间或者存储器和存储器之间传输数据，而无需CPU的介入，从而使CPU可以专注在处理其他系统功能上。章节[3.9.1](#)描述了DMA的寄存器列表，章节[3.9.2](#)对DMA库函数进行说明。

### 3.9.1. 外设寄存器说明

DMA寄存器列表如下表所示：

表 3-150. 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..6)	通道x控制寄存器
DMA_CHxCNT	通道x计数寄存器

寄存器名称	寄存器描述
(x=0..6)	
DMA_CHxPADDR (x=0..6)	通道x外设基地址寄存器
DMA_CHxMADDR (x=0..6)	通道x存储器基地址寄存器

### 3.9.2. 外设库函数说明

DMA库函数列表如下表所示：

**表 3-151. 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	DMA循环模式使能
dma_circulation_disable	DMA循环模式禁能
dma_memory_to_memory_enable	存储器到存储器DMA传输使能
dma_memory_to_memory_disable	存储器到存储器DMA传输禁能
dma_channel_enable	DMA通道x传输使能
dma_channel_disable	DMA通道x传输禁能
dma_periph_address_config	DMA通道x传输的外设基地址配置
dma_memory_address_config	DMA通道x传输的存储器基地址配置
dma_transfer_number_config	配置DMA通道x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	DMA通道x的传输软件优先级配置
dma_memory_width_config	DMA通道x传输的存储器数据宽度配置
dma_periph_width_config	DMA通道x传输的外设数据宽度配置
dma_memory_increase_enable	DMA通道x传输的存储器地址生成算法增量模式使能
dma_memory_increase_disable	DMA通道x传输的存储器地址生成算法增量模式禁能
dma_periph_increase_enable	DMA通道x传输的外设地址生成算法增量模式使能
dma_periph_increase_disable	DMA通道x传输的外设地址生成算法增量模式禁能
dma_transfer_direction_config	DMA通道x的传输方向配置
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	DMA通道x中断使能
dma_interrupt_disable	DMA通道x中断禁能
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

## 枚举 dma\_channel\_enum

表 3-152. 枚举类型 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
DMA_CH3	DMA通道3
DMA_CH4	DMA通道4
DMA_CH5	DMA通道5
DMA_CH6	DMA通道6

## 结构体 dma\_parameter\_struct

表 3-153. 结构体类型 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
periph_inc	外设地址生成算法模式
memory_addr	存储器基地址
memory_width	存储器数据传输宽度
memory_inc	存储器地址生成算法模式
direction	DMA通道数据传输方向
number	DMA通道数据传输数量
priority	DMA通道传输软件优先级

## 函数 dma\_deinit

函数dma\_deinit描述见下表:

表 3-154. 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, 具体参考 <a href="#">表3-152. 枚举类型dma_channel_enum</a> 。
DMA_CHx(x=0..6)	DMA通道选择
输出参数{out}	
-	-
返回值	
-	-

例如：

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

### 函数 dma\_struct\_para\_init

函数dma\_struct\_para\_init描述见下表：

表 3-155. 函数 dma\_struct\_para\_init

函数名称	dma_struct_para_init
函数原型	void dma_struct_para_init(dma_parameter_struct* init_struct);
功能描述	将DMA结构体中所有参数初始化为默认值
先决条件	无
被调用函数	无
输入参数{in}	
init_struct	初始化结构体，结构体成员参考 <a href="#">表3-153. 结构体类型dma_parameter_struct</a>
输出参数{out}	
-	-
返回值	
-	-

例如：

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

### 函数 dma\_init

函数dma\_init描述见下表：

表 3-156. 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，具体参考 <a href="#">表3-152. 枚举类型dma_channel_enum</a> 。
DMA_CHx(x=0..6)	DMA通道选择
输入参数{in}	
init_struct	初始化结构体，结构体成员参考 <a href="#">表3-153. 结构体类型dma_parameter_struct</a>
输出参数{out}	
-	-

返回值	
-	-

例如：

```
/* DMA channel0 initialize */
dma_parameter_struct 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 3-157. 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	无
被调用函数	无
输入参数{in}	
channelx	DMA通道，具体参考 <a href="#">表3-152. 枚举类型dma_channel_enum</a> 。
DMA_CHx(x=0..6)	DMA通道选择
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 3-158. 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	无
被调用函数	无
输入参数{in}	
channelx	DMA通道，具体参考 <a href="#">表3-152. 枚举类型dma_channel_enum</a> 。
DMA_CHx(x=0..6)	DMA通道选择
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 3-159. 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	无
被调用函数	无
输入参数{in}	
channelx	DMA通道，具体参考 <a href="#">表3-152. 枚举类型dma_channel_enum</a> 。
DMA_CHx(x=0..6)	DMA通道选择
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 3-160. 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	无
被调用函数	无
输入参数{in}	
channelx	DMA通道，具体参考 <a href="#">表3-152. 枚举类型dma_channel_enum</a> 。
DMA_CHx(x=0..6)	DMA通道选择
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 3-161. 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，具体参考 <a href="#">表3-152. 枚举类型dma_channel_enum</a> 。
DMA_CHx(x=0..6)	DMA通道选择
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 3-162. 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，具体参考 <a href="#">表3-152. 枚举类型dma_channel_enum</a> 。
DMA_CHx(x=0..6)	DMA通道选择
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 3-163. 函数 dma\_periph\_address\_config

函数名称	dma_periph_address_config
函数原型	void dma_periph_address_config(dma_channel_enum channelx, uint32_t address);
功能描述	DMA通道x传输的外设基地址配置
先决条件	无
被调用函数	无
输入参数{in}	
channelx	DMA通道，具体参考 <a href="#">表3-152. 枚举类型dma_channel_enum</a> 。
DMA_CHx(x=0..6)	DMA通道选择
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 3-164. 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	无
被调用函数	无
输入参数{in}	
channelx	DMA通道，具体参考 <a href="#">表3-152. 枚举类型dma_channel_enum</a> 。
DMA_CHx(x=0..6)	DMA通道选择
输入参数{in}	
address	存储器基地址
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 3-165. 函数 dma\_transfer\_number\_config

函数名称	dma_transfer_number_config
函数原型	void dma_transfer_number_config(dma_channel_enum channelx, uint32_t number);
功能描述	配置DMA通道x还有多少数据要传输
先决条件	无
被调用函数	无
输入参数{in}	
channelx	DMA通道，具体参考 <a href="#">表3-152. 枚举类型dma_channel_enum</a> 。
DMA_CHx(x=0..6)	DMA通道选择
输入参数{in}	

<b>number</b>	数据传输数量（0x00000000 – 0x0000FFFF）
<b>输出参数{out}</b>	
-	-
<b>返回值</b>	
-	-

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 3-166. 函数 dma\_transfer\_number\_get

<b>函数名称</b>	dma_transfer_number_get
<b>函数原型</b>	uint32_t dma_transfer_number_get(dma_channel_enum channelx);
<b>功能描述</b>	获取DMA通道x还有多少数据要传输
<b>先决条件</b>	无
<b>被调用函数</b>	无
<b>输入参数{in}</b>	
<b>channelx</b>	DMA通道，具体参考 <a href="#">表3-152. 枚举类型dma_channel_enum</a> 。
<b>DMA_CHx(x=0..6)</b>	DMA通道选择
<b>输出参数{out}</b>	
-	-
<b>返回值</b>	
<b>uint32_t</b>	DMA数据传输剩余数量（0x00000000 – 0x0000FFFF）

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 3-167. 函数 dma\_priority\_config

<b>函数名称</b>	dma_priority_config
<b>函数原型</b>	void dma_priority_config(dma_channel_enum channelx, uint32_t priority);
<b>功能描述</b>	DMA通道x的传输软件优先级配置

先决条件	无
被调用函数	无
输入参数{in}	
channelx	DMA通道，具体参考 <a href="#">表3-152. 枚举类型dma_channel_enum。</a>
DMA_CHx(x=0..6)	DMA通道选择
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 3-168. 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	无
被调用函数	无
输入参数{in}	
channelx	DMA通道，具体参考 <a href="#">表3-152. 枚举类型dma_channel_enum。</a>
DMA_CHx(x=0..6)	DMA通道选择
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

<code>DMA_MEMORY_WIDTH_16BIT</code>	16位数据传输宽度
<code>DMA_MEMORY_WIDTH_32BIT</code>	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 3-169. 函数 `dma_periph_width_config`

函数名称	<code>dma_periph_width_config</code>
函数原型	<code>void dma_periph_width_config(dma_channel_enum channelx, uint32_t pwidth);</code>
功能描述	DMA通道x传输的外设数据宽度配置
先决条件	无
被调用函数	无
输入参数{in}	
<b>channelx</b>	DMA通道，具体参考 <a href="#">表3-152. 枚举类型 <code>dma_channel_enum</code></a> 。
<code>DMA_CHx(x=0..6)</code>	DMA通道选择
输入参数{in}	
<b>pwidth</b>	外设数据传输宽度
<code>DMA_PERIPHERAL_WIDTH_8BIT</code>	8位数据传输宽度
<code>DMA_PERIPHERAL_WIDTH_16BIT</code>	16位数据传输宽度
<code>DMA_PERIPHERAL_WIDTH_32BIT</code>	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 3-170. 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	无
被调用函数	无
输入参数{in}	
channelx	DMA通道，具体参考 <a href="#">表3-152. 枚举类型dma_channel_enum</a> 。
DMA_CHx(x=0..6)	DMA通道选择
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 3-171. 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	无
被调用函数	无
输入参数{in}	
channelx	DMA通道，具体参考 <a href="#">表3-152. 枚举类型dma_channel_enum</a> 。
DMA_CHx(x=0..6)	DMA通道选择
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 3-172. 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	无
被调用函数	无
输入参数{in}	
channelx	DMA通道，具体参考 <a href="#">表3-152. 枚举类型dma_channel_enum</a> 。
DMA_CHx(x=0..6)	DMA通道选择
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 3-173. 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	无
被调用函数	无
输入参数{in}	
channelx	DMA通道，具体参考 <a href="#">表3-152. 枚举类型dma_channel_enum</a> 。
DMA_CHx(x=0..6)	DMA通道选择
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 3-174. 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	无
被调用函数	无
输入参数{in}	
channelx	DMA通道，具体参考 <a href="#">表3-152. 枚举类型dma_channel_enum</a> 。
DMA_CHx(x=0..6)	DMA通道选择
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 3-175. 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，具体参考 <a href="#">表3-152. 枚举类型dma_channel_enum</a> 。
DMA_CHx(x=0..6)	DMA通道选择
输入参数{in}	



flag	DMA标志
<i>DMA_FLAG_G</i>	DMA通道全局中断标志
<i>DMA_FLAG_FTF</i>	DMA通道传输完成标志
<i>DMA_FLAG_HTF</i>	DMA通道半传输完成标志
<i>DMA_FLAG_ERR</i>	DMA通道错误标志
输出参数{out}	
-	-
返回值	
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 3-176. 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，具体参考 <a href="#">表3-152. 枚举类型dma_channel_enum</a> 。
<i>DMA_CHx(x=0..6)</i>	DMA通道选择
输入参数{in}	
flag	DMA标志
<i>DMA_FLAG_G</i>	DMA通道全局中断标志
<i>DMA_FLAG_FTF</i>	DMA通道传输完成标志
<i>DMA_FLAG_HTF</i>	DMA通道半传输完成标志
<i>DMA_FLAG_ERR</i>	DMA通道错误标志
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* 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 3-177. 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, 具体参考 <a href="#">表3-152. 枚举类型dma_channel_enum</a> 。
DMA_CHx(x=0..6)	DMA通道选择
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 3-178. 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, 具体参考 <a href="#">表3-152. 枚举类型dma_channel_enum</a> 。
DMA_CHx(x=0..6)	DMA通道选择
输入参数{in}	
source	DMA中断源
DMA_INT_FTF	DMA通道传输完成中断

<i>DMA_INT_HTF</i>	DMA通道半传输完成中断
<i>DMA_INT_ERR</i>	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 3-179. 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，具体参考 <a href="#">表3-152. 枚举类型dma_channel_enum</a> 。
<i>DMA_CHx(x=0..6)</i>	DMA通道选择
输入参数{in}	
flag	DMA标志
<i>DMA_INT_FLAG_FTF</i>	DMA通道传输完成中断标志
<i>DMA_INT_FLAG_HTF</i>	DMA通道半传输完成中断标志
<i>DMA_INT_FLAG_ERR</i>	DMA通道错误中断标志
输出参数{out}	
-	-
返回值	
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 3-180. 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，具体参考 <a href="#">表3-152. 枚举类型dma_channel_enum</a> 。
DMA_CHx( x=0..6)	DMA通道选择
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	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);
}

```

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

### 3.10.1. 外设寄存器说明

EXTI寄存器列表如下表所示:

表 3-181. EXTI 寄存器

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

### 3.10.2. 外设库函数说明

EXTI库函数列表如下表所示:

表 3-182. EXTI 库函数

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

枚举类型 `exti_line_enum`

表 3-183. 枚举类型 `exti_line_enum`

成员名称	功能描述
EXTI_0	EXTI中断线0
EXTI_1	EXTI中断线1
EXTI_2	EXTI中断线2
EXTI_3	EXTI中断线3
EXTI_4	EXTI中断线4
EXTI_5	EXTI中断线5
EXTI_6	EXTI中断线6

EXTI_7	EXTI中断线7
EXTI_8	EXTI中断线8
EXTI_9	EXTI中断线9
EXTI_10	EXTI中断线10
EXTI_11	EXTI中断线11
EXTI_12	EXTI中断线12
EXTI_13	EXTI中断线13
EXTI_14	EXTI中断线14
EXTI_15	EXTI中断线15
EXTI_16	EXTI中断线16
EXTI_17	EXTI中断线17
EXTI_18	EXTI中断线18
EXTI_19	EXTI中断线19
EXTI_20	EXTI中断线20
EXTI_21	EXTI中断线21
EXTI_22	EXTI中断线22
EXTI_23	EXTI中断线23
EXTI_24	EXTI中断线24
EXTI_25	EXTI中断线25
EXTI_26	EXTI中断线26
EXTI_27	EXTI中断线27

#### 枚举类型 `exti_mode_enum`

表 3-184. 枚举类型 `exti_mode_enum`

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

#### 枚举类型 `exti_trig_type_enum`

表 3-185. 枚举类型 `exti_trig_type_enum`

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

#### 函数 `exti_deinit`

函数`exti_deinit`描述见下表：

表 3-186. 函数 `exti_deinit`

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

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

例如：

```
/* deinitialize the EXTI */
```

```
exti_deinit();
```

### 函数 exti\_init

函数exti\_init描述见下表：

表 3-187. 函数 exti\_init

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

例如：

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

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

## 函数 exti\_interrupt\_enable

函数exti\_interrupt\_enable描述见下表：

表 3-188. 函数 exti\_interrupt\_enable

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

例如：

```
/* enable the interrupts from EXTI line 0 */
```

```
exti_interrupt_enable(EXTI_0);
```

## 函数 exti\_interrupt\_disable

函数exti\_interrupt\_disable描述见下表：

表 3-189. 函数 exti\_interrupt\_disable

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

例如：

```
/* disable the interrupts from EXTI line 0 */
```

```
exti_interrupt_disable(EXTI_0);
```



### 函数 exti\_event\_enable

函数exti\_event\_enable描述见下表:

表 3-190. 函数 exti\_event\_enable

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

例如:

```
/* enable the events from EXTI line 0 */
```

```
exti_event_enable(EXTI_0);
```

### 函数 exti\_event\_disable

函数exti\_event\_disable描述见下表:

表 3-191. 函数 exti\_event\_disable

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

例如:

```
/* disable the events from EXTI line 0 */
```

```
exti_event_disable(EXTI_0);
```

**函数 exti\_software\_interrupt\_enable**

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

**表 3-192. 函数 exti\_software\_interrupt\_enable**

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

例如：

```
/* enable EXTI line 0 software interrupt */
```

```
exti_software_interrupt_enable(EXTI_0);
```

**函数 exti\_software\_interrupt\_disable**

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

**表 3-193. 函数 exti\_software\_interrupt\_disable**

函数名称	exti_software_interrupt_disable
函数原形	void exti_software_interrupt_disable(exti_line_enum linex);
功能描述	EXTI线x软件中断事件禁能
先决条件	-
被调用函数	-
输入参数{in}	
linex	EXTI线x，具体参考 <a href="#">表3-183. 枚举类型exti_line_enum</a>
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* disable EXTI line 0 software interrupt */
```

```
exti_software_interrupt_disable(EXTI_0);
```

### 函数 exti\_flag\_get

函数exti\_flag\_get描述见下表:

表 3-194. 函数 exti\_flag\_get

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

例如:

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

### 函数 exti\_flag\_clear

函数exti\_flag\_clear描述见下表:

表 3-195. 函数 exti\_flag\_clear

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

例如:

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

**函数 exti\_interrupt\_flag\_get**

函数exti\_interrupt\_flag\_get描述见下表：

**表 3-196. 函数 exti\_interrupt\_flag\_get**

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

例如：

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

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

**函数 exti\_interrupt\_flag\_clear**

函数exti\_interrupt\_flag\_clear描述见下表：

**表 3-197. 函数 exti\_interrupt\_flag\_clear**

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

例如：

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

```
exti_interrupt_flag_clear(EXTI_0);
```

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

#### 3.11.1. 外设寄存器说明

FMC寄存器列表如下：

**表 3-198. FMC 寄存器**

寄存器	描述
FMC_WS	等待状态寄存器
FMC_KEY	解锁寄存器
FMC_OBKEY	选项字节解锁寄存器
FMC_STAT	状态寄存器
FMC_CTL	控制寄存器
FMC_ADDR	地址寄存器
FMC_OBSTAT	选项字节状态寄存器
FMC_WP	写保护寄存器
FMC_PID	产品ID寄存器

#### 3.11.2. 外设库函数说明

FMC固件库函数列举如下表：

**表 3-199. FMC 固件库函数**

函数名称	函数描述
fmc_unlock	解锁FMC主编程块操作
fmc_lock	锁定FMC主编程块操作
fmc_wscnt_set	设置FMC等待状态计数值
fmc_wait_state_enable	使能等待状态功能
fmc_wait_state_disable	失能等待状态功能
fmc_page_erase	FMC 页擦除
fmc_mass_erase	FMC 全片擦除
fmc_word_program	在相应地址全字编程
fmc_halfword_program	在相应地址半字编程
fmc_word_reprogram	在不擦除编程地址的前提下进行字编程
ob_unlock	解锁选项字节操作
ob_lock	锁定选项字节操作
ob_reset	重装载选项字节，并产生一次系统复位
ob_erase	擦除选项字节
ob_write_protection_enable	使能写保护

函数名称	函数描述
ob_security_protection_config	配置安全保护
ob_user_write	写用户选项字节
ob_data_program	写数据选项字节
ob_user_get	获取用户选项字节
ob_data_get	获取数据选项字节
ob_write_protection_get	获取写保护选项字节
ob_obstat_plevel_get	在FMC_OBSTAT寄存器中获取FMC可选字节块的安全保护级别值
fmc_interrupt_enable	使能FMC中断
fmc_interrupt_disable	除能FMC中断
fmc_flag_get	检查标志位是否置位
fmc_flag_clear	清除FMC标志
fmc_interrupt_flag_get	获取FMC中断标志状态
fmc_interrupt_flag_clear	清除FMC中断标志状态
fmc_state_get	获取FMC状态
fmc_ready_wait	检查FMC是否准备好
ob_parm_get	获取当前选项字节数值
ob_value_modify	在当前选项字节值的基础上变更选项字节

## 枚举 fmc\_state\_enum

表 3-200. 枚举类型 fmc\_state\_enum

成员名称	功能描述
FMC_READY	操作完成
FMC_BUSY	操作进行中
FMC_PGERR	编程错误
FMC_WPERR	写保护错误
FMC_TOERR	超时错误
FMC_OB_HSPC	可选字节块高安全保护级别

## 结构体 ob\_parm\_struct

表 3-201. 结构体类型 ob\_parm\_struct

成员名称	功能描述
spc	选项字节安全保护值
user	选项字节用户值
data0	选项字节数据0值
data1	选项字节数据1值
wp0	选项字节写保护0值
wp1	选项字节写保护1值

## 函数 fmc\_unlock

函数fmc\_unlock描述见下表：

**表 3-202. 函数 fmc\_unlock**

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

例如：

```
/* unlock the main FMC operation */
```

```
fmc_unlock ( );
```

## 函数 fmc\_lock

函数fmc\_lock描述见下表：

**表 3-203. 函数 fmc\_lock**

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

例如：

```
/* lock the main FMC operation */
```

```
fmc_lock( );
```

### 函数 `fmc_wscnt_set`

函数 `fmc_wscnt_set` 描述见下表：

**表 3-204. 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	设置等待状态计数值
先决条件	-
被调用函数	-
输入参数{in}	
<b>wscnt</b>	等待状态计数值
<code>WS_WSCNT_0</code>	FMC 0个等待状态
<code>WS_WSCNT_1</code>	FMC 1个等待状态
<code>WS_WSCNT_2</code>	FMC 2个等待状态
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* set the wait state counter value */
```

```
fmc_wscnt_set (WS_WSCNT_1);
```

### 函数 `fmc_wait_state_enable`

函数 `fmc_wait_state_enable` 描述见下表：

**表 3-205. 函数 `fmc_wait_state_enable`**

函数名称	<code>fmc_wait_state_enable</code>
函数原型	<code>void fmc_wait_state_enable(void);</code>
功能描述	使能等待状态功能
先决条件	-
被调用函数	-
输入参数{in}	
-	-
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* fmc wait state enable */
```



```
fmc_wait_state_enable ( );
```

### 函数 fmc\_wait\_state\_disable

函数fmc\_wait\_state\_disable描述见下表：

**表 3-206. 函数 fmc\_wait\_state\_disable**

函数名称	fmc_wait_state_disable
函数原型	void fmc_wait_state_disable (void);
功能描述	失能等待状态功能
先决条件	-
被调用函数	-
输入参数{in}	
-	-
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* fmc wait state disable */
```

```
fmc_wait_state_disable ( );
```

### 函数 fmc\_page\_erase

函数fmc\_page\_erase描述见下表：

**表 3-207. 函数 fmc\_page\_erase**

函数名称	fmc_page_erase
函数原型	fmc_state_enum fmc_page_erase(uint32_t page_address);
功能描述	页擦除
先决条件	fmc_unlock
被调用函数	fmc_ready_wait
输入参数{in}	
page_address	页擦除首地址
输出参数{out}	
-	-
返回值	
fmc_state_enum	FMC状态值，详情参考枚举变量 <a href="#">表3-200. 枚举类型fmc_state_enum</a>

例如：

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

```
fmc_state_enum state = fmc_page_erase ( 0x08004000);
```

**函数 fmc\_mass\_erase**

函数fmc\_mass\_erase描述见下表:

**表 3-208. 函数 fmc\_mass\_erase**

函数名称	fmc_mass_erase
函数原型	fmc_state_enum fmc_mass_erase(void);
功能描述	全片擦除
先决条件	fmc_unlock
被调用函数	fmc_ready_wait
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3-200. 枚举类型fmc_state_enum</a>

例如:

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

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

**函数 fmc\_word\_program**

函数fmc\_word\_program描述见下表:

**表 3-209. 函数 fmc\_word\_program**

函数名称	fmc_word_program
函数原型	fmc_state_enum fmc_word_program(uint32_t address, uint32_t data);
功能描述	对相应地址全字编程
先决条件	fmc_unlock
被调用函数	fmc_ready_wait
输入参数{in}	
address	编程地址
输入参数{in}	
data	编程数据
输出参数{out}	
-	-
返回值	
fmc_state_enum	FMC状态值, 详情参考枚举变量 <a href="#">表3-200. 枚举类型fmc_state_enum</a>

例如:

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

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

## 函数 `fmc_halfword_program`

函数 `fmc_halfword_program` 描述见下表：

**表 3-210. 函数 `fmc_halfword_program`**

函数名称	<code>fmc_halfword_program</code>
函数原型	<code>fmc_state_enum fmc_halfword_program(uint32_t address, uint16_t data);</code>
功能描述	对相应地址半字编程
先决条件	<code>fmc_unlock</code>
被调用函数	<code>fmc_ready_wait</code>
输入参数{in}	
<b>address</b>	编程地址
输入参数{in}	
<b>data</b>	编程数据
输出参数{out}	
-	-
返回值	
<b>fmc_state_enum</b>	FMC状态值，详情参考枚举变量 <a href="#">表3-200. 枚举类型 <code>fmc_state_enum</code></a>

例如：

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

```
fmc_state_enum state = fmc_halfword_program (0x08004000, 0xaadd);
```

## 函数 `fmc_word_reprogram`

函数 `fmc_word_reprogram` 描述见下表：

**表 3-211. 函数 `fmc_word_reprogram`**

函数名称	<code>fmc_word_reprogram</code>
函数原型	<code>fmc_state_enum fmc_word_reprogram(uint32_t address, uint32_t data);</code>
功能描述	对相应地址在不擦除原数据的前提下全字编程
先决条件	<code>fmc_unlock</code>
被调用函数	<code>fmc_ready_wait</code>
输入参数{in}	
<b>address</b>	编程地址
输入参数{in}	
<b>data</b>	编程数据
输出参数{out}	
-	-
返回值	
<b>fmc_state_enum</b>	FMC状态值，详情参考枚举变量 <a href="#">表3-200. 枚举类型 <code>fmc_state_enum</code></a>

例如：

```
/* FMC program a word at the corresponding address without erasing */
```

```
fmc_state_enum state;
```

```
state = fmc_word_program (0x08004000, 0x01234567);
```

```
state = fmc_word_program (0x08004000, 0xd583179b);
```

### 函数 ob\_unlock

函数ob\_unlock描述见下表：

**表 3-212. 函数 ob\_unlock**

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

例如：

```
/* unlock the option byte operation */
```

```
ob_unlock ( );
```

### 函数 ob\_lock

函数ob\_lock描述见下表：

**表 3-213. 函数 ob\_lock**

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

例如：

```
/* lock the option byte operation */
```

```
ob_lock ( );
```

### 函数 ob\_reset

函数ob\_reset描述见下表：

**表 3-214. 函数 ob\_reset**

函数名称	ob_reset
函数原型	void ob_reset (void);
功能描述	重装载选项字节，并产生一次系统复位
先决条件	-
被调用函数	-
输入参数{in}	
-	-
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* reload the option byte and generate a system reset */
```

```
ob_reset ( );
```

### 函数 ob\_erase

函数ob\_erase描述见下表：

**表 3-215. 函数 ob\_erase**

函数名称	ob_erase
函数原型	void ob_erase(void);
功能描述	擦除选项字节
先决条件	ob_unlock
被调用函数	fmc_ready_wait
输入参数{in}	
-	-
输出参数{out}	
-	-
返回值	
fmc_state_enum	FMC状态值，详情参考枚举变量 <a href="#">表3-200. 枚举类型fmc_state_enum</a>

例如：

```
/* erase the FMC option byte */
```

```
fmc_state_enum fmc_state = ob_erase ( );
```

### 函数 ob\_write\_protection\_enable

函数ob\_write\_protection\_enable描述见下表：

**表 3-216. 函数 ob\_write\_protection\_enable**

函数名称	ob_write_protection_enable
函数原型	fmc_state_enum ob_write_protection_enable(uint16_t ob_wp);
功能描述	使能写保护
先决条件	ob_unlock
被调用函数	fmc_ready_wait
输入参数{in}	
ob_wp	写保护单元
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3-200. 枚举类型fmc_state_enum</a>

例如：

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

```
fmc_state_enum state = ob_write_protection_enable (0x7C);
```

### 函数 ob\_security\_protection\_config

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

**表 3-217. 函数 ob\_security\_protection\_config**

函数名称	ob_security_protection_config
函数原型	fmc_state_enum ob_security_protection_config (uint16_t ob_spc);
功能描述	配置安全保护
先决条件	ob_unlock
被调用函数	fmc_ready_wait
输入参数{in}	
ob_spc	安全保护
FMC_NSPC	无安全保护
FMC_LSPC	低保护级别
FMC_HSPC	高保护级别
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3-200. 枚举类型fmc_state_enum</a>

例如：

```
/* enable security protection */
```

```
fmc_state_enum state = ob_security_protection_config (FMC_USPC);
```

### 函数 ob\_user\_write

函数ob\_user\_write描述见下表：

**表 3-218. 函数 ob\_user\_write**

函数名称	ob_user_write
函数原型	fmc_state_enum ob_user_write(uint8_t ob_user);
功能描述	编辑用户选项字节
先决条件	ob_unlock
被调用函数	fmc_ready_wait
输入参数{in}	
ob_user	用户定义的选项字节
OB_FWDGT_HW	硬件看门狗
OB_DEEPSLEEP_RST	进入深度睡眠时不复位
OB_STDBY_RST	进入深度睡眠时产生复位
OB_BOOT1_SET_1	BOOT1位是1
OB_VDDA_DISABLER	失能V <sub>DDA</sub> 监视器
OB_SRAM_PARITY_ENABLE	使能SRAM奇偶校验
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3-200. 枚举类型fmc_state_enum</a>

例如：

```
/* configure user option byte */
```

```
fmc_state_enum state = ob_user_write(OB_FWDGT_HW & OB_DEEPSLEEP_RST);
```

### 函数 ob\_data\_program

函数ob\_data\_program描述见下表：

**表 3-219. 函数 ob\_data\_program**

函数名称	ob_data_program
函数原型	fmc_state_enum ob_data_program(uint32_t address, uint8_t data);
功能描述	编程数字选项字节
先决条件	ob_unlock

被调用函数	fmc_ready_wait
输入参数{in}	
address	编程数字选项字节地址
OB_DATA_ADDR0	编程数字选项字节地址0
OB_DATA_ADDR1	编程数字选项字节地址1
输入参数{in}	
address	OB_DATA_ADDR0或OB_DATA_ADDR1
输入参数{in}	
data	所编程数值
输出参数{out}	
-	-
返回值	
fmc_state_enum	FMC状态值，详情参考枚举变量 <a href="#">表3-200. 枚举类型fmc_state_enum</a>

例如：

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

```
fmc_state_enum state = ob_data_program (OB_DATA_ADDR0, 0x56);
```

### 函数 ob\_user\_get

函数ob\_user\_get描述见下表：

表 3-220. 函数 ob\_user\_get

函数名称	ob_user_get
函数原型	uint8_t ob_user_get(void);
功能描述	获取FMC_OBSTAT寄存器中的用户选项字节
先决条件	-
被调用函数	-
输入参数{in}	
-	-
输出参数{out}	
-	-
返回值	
uint8_t	选项字节用户数值（0x00 – 0xFF）

例如：

```
/* get the FMC user option byte */
```

```
uint8_t user = ob_user_get ( );
```

### 函数 ob\_data\_get

函数ob\_data\_get描述见下表：



表 3-221. 函数 ob\_data\_get

函数名称	ob_data_get
函数原型	uint16_t ob_data_get(void);
功能描述	获取FMC_OBSTAT寄存器中的数据选项字节
先决条件	-
被调用函数	-
输入参数{in}	
-	-
输出参数{out}	
-	-
返回值	
uint16_t	选项字节数据值（0x0 – 0xFFFF）

例如：

```
/* get the FMC data option byte */
```

```
Uint16_t data = ob_data_get ( );
```

### 函数 ob\_write\_protection\_get

函数ob\_write\_protection\_get描述见下表：

表 3-222. 函数 ob\_write\_protection\_get

函数名称	ob_write_protection_get
函数原型	uint16_t ob_write_protection_get(void);
功能描述	在FMC_WP寄存器中获取FMC可选字节块的擦/写保护位的值
先决条件	-
被调用函数	-
输入参数{in}	
-	-
输出参数{out}	
-	-
返回值	
Uint16_t	选项字节写保护数值（0x0 – 0xFFFF）

例如：

```
/* get the FMC option byte write protection */
```

```
uint32_t wp = ob_write_protection_get ( );
```

### 函数 ob\_obstat\_plevel\_get

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

表 3-223. 函数 ob\_obstat\_plevel\_get

函数名称	ob_obstat_plevel_get
函数原型	uint32_t ob_obstat_plevel_get(void);
功能描述	在FMC_OBSTAT寄存器中获取FMC可选字节块的安全保护级别值
先决条件	-
被调用函数	-
输入参数{in}	
-	-
输出参数{out}	
-	-
返回值	
uint8_t	the value of PLEVEL(0x0,0x01,0x03)

例如：

```
/* get the FMC option byte security protection */
uint32_t obstat_plevel = ob_obstat_plevel_get ( );
```

### 函数 fmc\_interrupt\_enable

函数fmc\_interrupt\_enable描述见下表：

表 3-224. 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
FMC_INT_END	FMC编程完成中断
FMC_INT_ERR	FMC错误中断
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* enable FMC interrupt */
fmc_interrupt_enable(FMC_INT_END);
```

### 函数 fmc\_interrupt\_disable

函数fmc\_interrupt\_disable描述见下表：

表 3-225. 函数 `fmc_interrupt_disable`

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

例如：

```
/* disable FMC interrupt */
fmc_interrupt_disable(FMC_INT_END);
```

### 函数 `fmc_flag_get`

函数`fmc_flag_get`描述见下表：

表 3-226. 函数 `fmc_flag_get`

函数名称	<code>fmc_flag_get</code>
函数原型	<code>FlagStatus fmc_flag_get(uint32_t flag);</code>
功能描述	检查标志是否置位
先决条件	-
被调用函数	-
输入参数{in}	
<b>flag</b>	检查FMC标志
<code>FMC_FLAG_BUSY</code>	FMC忙碌标志
<code>FMC_FLAG_PGER</code> <code>R</code>	FMC操作错误标志
<code>FMC_FLAG_WPER</code> <code>R</code>	FMC写保护错误标志
<code>FMC_FLAG_END</code>	FMC操作完成标志
输出参数{out}	
-	-
返回值	
<b>FlagStatus</b>	SET 或 RESET

例如：

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

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

### 函数 fmc\_flag\_clear

函数fmc\_flag\_clear描述见下表：

表 3-227. 函数 fmc\_flag\_clear

函数名称	fmc_flag_clear
函数原型	void fmc_flag_clear(uint32_t flag);
功能描述	写1清除FMC标志
先决条件	-
被调用函数	-
输入参数{in}	
flag	清除FMC标志
FMC_FLAG_PGER R	FMC操作错误标志
FMC_FLAG_WPER R	FMC写保护错误标志
FMC_FLAG_END	FMC操作完成标志
输出参数{out}	
-	-
返回值	
-	-

例如：

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

```
FlagStatus flag = fmc_flag_clear(FMC_FLAG_END);
```

### 函数 fmc\_interrupt\_flag\_get

函数fmc\_interrupt\_flag\_get描述见下表：

表 3-228. 函数 fmc\_interrupt\_flag\_get

函数名称	fmc_interrupt_flag_get
函数原型	FlagStatus fmc_interrupt_flag_get(uint32_t flag);
功能描述	获取FMC中断标志状态
先决条件	-
被调用函数	-
输入参数{in}	
flag	中断标志
FMC_FLAG_PGER R	FMC操作错误标志

<i>FMC_FLAG_WPER</i> R	FMC写保护错误标志
<i>FMC_FLAG_END</i>	FMC操作完成标志
输出参数{out}	
-	-
返回值	
FlagStatus	SET 或 RESET

例如:

```
/* get FMC interrupt flag */
```

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

### 函数 fmc\_interrupt\_flag\_clear

函数fmc\_interrupt\_flag\_clear描述见下表:

表 3-229. 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	通过写1清除FMC中断标志
先决条件	-
被调用函数	-
输入参数{in}	
flag	清除FMC中断标志
<i>FMC_FLAG_PGER</i> R	FMC操作错误标志
<i>FMC_FLAG_WPER</i> R	FMC写保护错误标志
<i>FMC_FLAG_END</i>	FMC操作完成标志
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* clear FMC interrupt flag */
```

```
FlagStatus flag = fmc_interrupt_flag_clear (FMC_FLAG_PGERR);
```

### 函数 fmc\_state\_get

函数fmc\_state\_get描述见下表:

表 3-230. 函数 `fmc_state_get`

函数名称	<code>fmc_state_get</code>
函数原型	<code>fmc_state_enum fmc_state_get(void);</code>
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	
<code>fmc_state_enum</code>	FMC状态值，详情参考枚举变量 <a href="#">表3-200. 枚举类型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 3-231. 函数 `fmc_ready_wait`

函数名称	<code>fmc_ready_wait</code>
函数原型	<code>fmc_state_enum fmc_ready_wait(uint32_t timeout);</code>
功能描述	检查FMC是否准备好
先决条件	-
被调用函数	<code>fmc_state_get();</code>
输入参数{in}	
<code>timeout</code>	循环计数次数
输出参数{out}	
-	-
返回值	
<code>fmc_state_enum</code>	FMC状态值，详情参考枚举变量 <a href="#">表3-200. 枚举类型fmc_state_enum</a>

例如：

```
/* check whether FMC is ready or not */
```

```
fmc_state_enum state = fmc_ready_wait (0x00001000 );
```

### 函数 `ob_parm_get`

函数`ob_parm_get`描述见下表：

表 3-232. 函数 ob\_parm\_get

函数名称	ob_parm_get
函数原型	ob_parm_get(ob_parm_struct *ob_parm);
功能描述	获取当前选项字节值
先决条件	-
被调用函数	-
输入参数{in}	
ob_parm	指向选项字节参数结构体 <a href="#">表3-201. 结构体类型ob_parm_struct</a>
输出参数{out}	
ob_parm	指向选项字节参数结构体 <a href="#">表3-201. 结构体类型ob_parm_struct</a>
返回值	
-	-

例如：

```
/* get current option byte value */
ob_parm_struct ob_parm;
ob_parm_get(&ob_parm);
```

### 函数 ob\_value\_modify

函数ob\_value\_modify描述见下表：

表 3-233. 函数 ob\_value\_modify

函数名称	ob_value_modify
函数原型	Void ob_value_modify(uint32_t address, uint16_t value, ob_parm_struct *ob_parm)
功能描述	基于当前选项字节数值按照指定地址和数值修改选项字节值
先决条件	-
被调用函数	-
输入参数{in}	
address	目标地址
输入参数{in}	
value	目标数值
输入参数{in}	
ob_parm	指向选项字节参数结构体 <a href="#">表3-201. 结构体类型ob_parm_struct</a>
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* modify the target option byte depending on the original value */
```

```
ob_parm_struct ob_parm;

ob_value_modify(OB_WP_ADDR0, ob_wp ,&ob_parm);
```

## 3.12. 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[3.12.1](#)描述了FWDGT的寄存器列表，章节[3.12.2](#)对FWDGT库函数进行说明。

### 3.12.1. 外设寄存器说明

FWDGT寄存器列表如下表所示：

**表 3-234. FWDGT 寄存器**

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

### 3.12.2. 外设库函数说明

FWDGT库函数列表如下表所示：

**表 3-235. FWDGT 库函数**

库函数名称	库函数描述
fwdgt_write_enable	使能对寄存器FWDGT_PSC，FWDGT_RLD和FWDGT_WND的写操作
fwdgt_write_disable	禁能对寄存器FWDGT_PSC，FWDGT_RLD和FWDGT_WND的写操作
fwdgt_enable	使能FWDGT
fwdgt_prescaler_value_config	配置独立看门狗定时器预分频值
fwdgt_reload_value_config	配置独立看门狗定时器重装载值
fwdgt_window_value_config	配置独立看门狗定时器计数窗口值
fwdgt_counter_reload	按照FWDGT_RLD寄存器的值重装载FWDGT计数器
fwdgt_config	设置FWDGT重装载值、预分频值
fwdgt_flag_get	获取FWDGT标志位状态

#### 函数 fwdgt\_write\_enable

函数fwdgt\_write\_enable描述见下表：



表 3-236. 函数 fwdgt\_write\_enable

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

例如：

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

```
fwdgt_write_enable ( );
```

### 函数 fwdgt\_write\_disable

函数fwdgt\_write\_disable描述见下表：

表 3-237. 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	禁能对寄存器FWDGT_PSC，FWDGT_RLD和FWDGT_WND的写操作
先决条件	-
输入参数{in}	
-	-
输出参数{out}	
-	-
返回值	
-	-

例如：

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

```
fwdgt_write_disable ( );
```

### 函数 fwdgt\_enable

函数fwdgt\_enable描述见下表：

表 3-238. 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	-
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 3-239. 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计数器窗口值
先决条件	-
输入参数{in}	
<b>prescaler_value</b>	预分频值
<i>FWDGT_PSC_DIV4</i>	FWDGT预分频值设为4
<i>FWDGT_PSC_DIV8</i>	FWDGT预分频值设为8
<i>FWDGT_PSC_DIV16</i>	FWDGT预分频值设为16
<i>FWDGT_PSC_DIV32</i>	FWDGT预分频值设为32
<i>FWDGT_PSC_DIV64</i>	FWDGT预分频值设为64
<i>FWDGT_PSC_DIV128</i>	FWDGT预分频值设为128
<i>FWDGT_PSC_DIV256</i>	FWDGT预分频值设为256
输出参数{out}	
-	-
返回值	
<b>ErrStatus</b>	ERROR / SUCCESS

例如：

```
/* set FWDGT prescalervalue to 256 */
```

ErrStatus flag;

flag = fwdgt\_prescaler\_value\_config (FWDGT\_PSC\_DIV256);

### 函数 fwdgt\_reload\_value\_config

函数fwdgt\_reload\_value\_config描述见下表:

表 3-240. 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重装载值
先决条件	-
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 0x0FFF */
```

ErrStatus flag;

flag = fwdgt\_reload\_value\_config (0x0FFF);

### 函数 fwdgt\_window\_value\_config

函数fwdgt\_window\_value\_config描述见下表:

表 3-241. 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	-
输入参数{in}	
window_value	窗口值,数值范围为 0x0000 – 0x0FFF
输出参数{out}	
-	-
返回值	
ErrStatus	ERROR / SUCCESS

例如:

```
/* set FWDGT window value to 0x0FFF */
```

ErrStatus flag;

flag = fwdgt\_window\_value\_config (0x0FFF);

### 函数 fwdgt\_counter\_reload

函数fwdgt\_counter\_reload描述见下表:

表 3-242. 函数 fwdgt\_counter\_reload

函数名称	fwdgt_counter_reload
函数原型	void fwdgt_counter_reload(void);
功能描述	按照FWDGT_RLD寄存器的值重装载IWDG计数器
先决条件	-
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 3-243. 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	-
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

4	
FWDGT_PSC_DIV1 28	FWDGT预分频值设为128
FWDGT_PSC_DIV2 56	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 3-244. 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	-
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);
```

### 3.13. GPIO

GPIO用来实现各片上设备的逻辑输入/输出功能。章节[3.13.1](#)描述了GPIO的寄存器列表，章节[3.13.2](#)对GPIO库函数进行说明。

#### 3.13.1. 外设寄存器说明

GPIO寄存器列表如下表所示：

表 3-245. 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_OSPD0	端口输出速度寄存器0
GPIOx_PUD	端口上拉/下拉寄存器
GPIOx_ISTAT	端口输入状态寄存器
GPIOx_OCTL	端口输出控制寄存器
GPIOx_BOP	端口位操作寄存器
GPIOx_LOCK	端口配置锁定寄存器
GPIO_AFSEL0	备用功能选择寄存器0
GPIO_AFSEL1	备用功能选择寄存器1
GPIO_BC	位清除寄存器
GPIO_TG	端口位翻转寄存器
GPIO_OSPD1	端口输出速度寄存器1

#### 3.13.2. 外设库函数说明

GPIO库函数列表如下表所示：

表 3-246. 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

库函数名称	库函数描述
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 3-247. 函数 gpio\_deinit**

函数名称	gpio_deinit
函数原型	void gpio_deinit(uint32_t gpio_periph);
功能描述	复位外设GPIOx
先决条件	-
被调用函数	rcu_periph_reset_enable / rcu_periph_reset_disable
输入参数{in}	
gpio_periph	GPIO端口
GPIOx	GPIOx(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 3-248. 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	rcu_periph_reset_enable / rcu_periph_reset_disable
输入参数{in}	
gpio_periph	GPIO端口
GPIOx	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> <i>T</i>	输出模式
<i>GPIO_MODE_AF</i>	备用功能模式
<i>GPIO_MODE_ANALOG</i> <i>G</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> <i>WN</i>	带下拉电阻
输入参数{in}	
<b>pin</b>	GPIO pin
<i>GPIO_PIN_x</i>	GPIO_PIN_x(x=0..15)
<i>GPIO_PIN_ALL</i>	All pins
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* 配置PA0为上拉输入模式*/
```

```
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 3-249. 函数 gpio\_output\_options\_set

<b>函数名称</b>	gpio_output_options_set
<b>函数原型</b>	void gpio_output_options_set(uint32_t gpio_periph, uint8_t otype, uint32_t speed, uint32_t pin);
<b>功能描述</b>	设置GPIO输出模式和速度
<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>	GPIOx(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_2MHZ</i>	最大输出速度为2MHz
<i>GPIO_OSPEED_10MHZ</i>	最大输出速度为10MHz
<i>GPIO_OSPEED_50MHZ</i>	最大输出速度为50MHz
<i>GPIO_OSPEED_MAX</i>	GPIO高输出速度，最大速度超过50MHz
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 3-250. 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}	
<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	
-	-

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 3-251. 函数 gpio\_bit\_reset**

函数名称	gpio_bit_reset
函数原型	void gpio_bit_reset(uint32_t gpio_periph,uint32_t pin);
功能描述	复位引脚值
先决条件	-
被调用函数	-
输入参数{in}	
gpio_periph	GPIO端口
GPIOx	端口选择(x = A,B,C,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:

```
/* 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 3-252. 函数 gpio\_bit\_write**

函数名称	gpio_bit_write
函数原型	void gpio_bit_write(uint32_t gpio_periph,uint32_t pin,bit_status bit_value);
功能描述	将特定的值写入引脚

先决条件	-
被调用函数	-
输入参数{in}	
gpio_periph	GPIO端口
GPIOx	端口选择(x = A,B,C,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
输入参数{in}	
bit_value	设置或清除
RESET	清除引脚值
SET	设置引脚值
输出参数{out}	
-	-
返回值	
-	-

例如:

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

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

## 函数 gpio\_port\_write

函数gpio\_port\_write描述见下表:

表 3-253. 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 3-254. 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 3-255. 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

<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 input value of Port A */

uint16_t port_state;

port_state = gpio_input_bit_get (GPIOA);
```

### 函数 gpio\_output\_bit\_get

函数gpio\_output\_bit\_get描述见下表:

表 3-256. 函数 gpio\_output\_bit\_get

函数名称	gpio_output_bit_get
函数原型	FlagStatus gpio_output_bit_get(uint32_t gpio_periph,uint32_t pin);
功能描述	获取引脚的输出值
先决条件	-
被调用函数	-
输入参数{in}	
<b>gpio_periph</b>	GPIO端口
<i>GPIOx</i>	端口选择(x = A,B,C,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 3-257. 函数 `gpio_output_port_get`

函数名称	<code>gpio_output_port_get</code>
函数原型	<code>uint16_t gpio_output_port_get(uint32_t gpio_periph);</code>
功能描述	获取引脚的输出值
先决条件	-
被调用函数	-
输入参数{in}	
<code>gpio_periph</code>	GPIO端口
<code>GPIOx</code>	端口选择(x = A,B,C,D,F)
输出参数{out}	
-	-
返回值	
<code>uint16_t</code>	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 3-258. 函数 `gpio_af_set`

函数名称	<code>gpio_af_set</code>
函数原型	<code>void gpio_af_set(uint32_t gpio_periph, uint32_t alt_func_num, uint32_t pin);</code>
功能描述	设置GPIO的备用功能
先决条件	-
被调用函数	-
输入参数{in}	
<code>gpio_periph</code>	GPIO 端口
<code>GPIOx</code>	GPIOx(x = A,B,C)
输入参数{in}	
<code>alt_func_num</code>	GPIO 引脚备用功能, 请参见特定设备的数据手册
<code>GPIO_AF_0</code>	TIMER2, TIMER13, TIMER14, TIMER16, SPI0, SPI1, I2S0, CK_OUT, USART0, CEC, IFRP, TSI, CTC, I2C0, I2C1, SWDIO, SWCLK
<code>GPIO_AF_1</code>	USART0, USART1, TIMER2, TIMER14, I2C0, I2C1, IFRP, CEC
<code>GPIO_AF_2</code>	TIMER0, TIMER1, TIMER15, TIMER16, I2S0
<code>GPIO_AF_3</code>	TSI, I2C0, TIMER14
<code>GPIO_AF_4 (port A,B only)</code>	USART1, I2C0, I2C1, TIMER13
<code>GPIO_AF_5 (port A,B only)</code>	TIMER15, TIMER16, USBFS, I2S0

<i>only</i> )	
<i>GPIO_AF_6 (port A,B only)</i>	CTC, SPI1
<i>GPIO_AF_7 (port A,B only)</i>	CMP0, CMP1
输入参数{in}	
<b>pin</b>	GPIO引脚
<i>GPIO_PIN_x</i>	引脚选择 (x=0..15)
<i>GPIO_PIN_ALL</i>	所有引脚
输出参数{out}	
-	-
返回值	
-	-

例如:

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

```
gpio_af_set(GPIOA, GPIO_AF_0, GPIO_PIN_0);
```

### 函数 gpio\_pin\_lock

函数gpio\_pin\_lock描述见下表:

表 3-259. 函数 gpio\_pin\_lock

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

例如:

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

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

### 函数 gpio\_bit\_toggle

函数gpio\_bit\_toggle描述见下表：

**表 3-260. 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	GPIOx(x = A,B,C,D,F)
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 3-261. 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);
```

## 3.14. I2C

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

### 3.14.1. 外设寄存器说明

I2C寄存器列表如下表所示：

**表 3-262. I2C 寄存器**

寄存器名称	寄存器描述
I2C_CTL0	控制寄存器0
I2C_CTL1	控制寄存器1
I2C_SADDR0	从机地址寄存器0
I2C_SADDR1	从机地址寄存器1
I2C_DATA	传输缓冲区寄存器
I2C_STAT0	传输状态寄存器0
I2C_STAT1	传输状态寄存器1
I2C_CKCFG	时钟配置寄存器
I2C_RT	上升时间寄存器
I2C_FMPCFG	快速+ 模式配置寄存器

### 3.14.2. 外设库函数说明

I2C库函数列表如下表所示：

**表 3-263. I2C 库函数**

库函数名称	库函数描述
i2c_deinit	复位外设I2C
i2c_clock_config	配置I2C时钟
i2c_mode_addr_config	配置I2C地址
i2c_smbus_type_config	SMBus类型选择
i2c_ack_config	是否发送ACK
i2c_ackpos_config	配置在接收模式下ACK和PEC的位置
i2c_master_addressing	主机发送从机地址

库函数名称	库函数描述
i2c_dualaddr_enable	双地址模式使能
i2c_dualaddr_disable	双地址模式禁能
i2c_enable	使能I2C模块
i2c_disable	关闭I2C模块
i2c_start_on_bus	在I2C总线上生成起始位
i2c_stop_on_bus	在I2C总线上生成停止位
i2c_data_transmit	发送数据
i2c_data_receive	接收数据
i2c_dma_config	I2C DMA模式配置
i2c_dma_last_transfer_config	配置下一个DMA EOT是否最后一次传输
i2c_stretch_scl_low_config	当从机数据没有准备好时是否拉低SCL
i2c_slave_response_to_gcall_config	从机是否响应广播呼叫
i2c_software_reset_config	配置I2C软件复位
i2c_pec_config	报文错误校验使能
i2c_pec_transfer_config	传输PEC配置
i2c_pec_value_get	获取报文错误校验值
i2c_smbus_alert_config	通过SMBA引脚配置I2C值
i2c_smbus_arp_config	在SMBus下配置ARP协议
i2c_flag_get	获取I2C标志位
i2c_flag_clear	清除I2C标志位
i2c_interrupt_enable	中断使能
i2c_interrupt_disable	中断除能
i2c_interrupt_flag_get	中断标志位获取
i2c_interrupt_flag_clear	中断标志位清除

## 枚举 i2c\_flag\_enum

表 3-264. 枚举类型 i2c\_flag\_enum

成员名称	功能描述
I2C_FLAG_SBSEND	主机模式下发送START起始位
I2C_FLAG_ADDSEND	主机模式下成功发送了地址，从机模式下接收到了地址并且和自身的地址匹配
I2C_FLAG_BTC	字节发送结束中断使能
I2C_FLAG_ADD10SEND	主机模式下10位地址的地址头被发送
I2C_FLAG_STPDET	从机模式下监测到STOP结束位
I2C_FLAG_RBNE	接收期间I2C_DATA非空
I2C_FLAG_TBE	发送期间I2C_DATA为空
I2C_FLAG_BERR	总线错误，表示I2C总线上发生了预料之外的START起始位或STOP结束位
I2C_FLAG_LOSTARB	主机模式下仲裁丢失
I2C_FLAG_AERR	应答错误

成员名称	功能描述
I2C_FLAG_OUERR	从机接收模式下，发生了过载或欠载事件
I2C_FLAG_PECERR	接收数据时PEC错误
I2C_FLAG_SMBTO	SMBus模式下超时信号
I2C_FLAG_SMBALT	SMBus警报状态
I2C_FLAG_MASTER	表明I2C时钟在主机模式还是从机模式的标志位
I2C_FLAG_I2CBSY	忙标志
I2C_FLAG_TR	I2C作发送端还是接收端
I2C_FLAG_RXGC	I2C作发送端还是接收端
I2C_FLAG_DEFSMB	从机模式下SMBus主机地址头
I2C_FLAG_HSTSMB	从机模式下监测到SMBus主机地址头
I2C_FLAG_DUMOD	从机模式下双标志位表明哪个地址和双地址模式匹配

### 枚举 i2c\_interrupt\_flag\_enum

表 3-265. 枚举类型 i2c\_interrupt\_flag\_enum

成员名称	功能描述
I2C_INT_FLAG_SBSEND	主机模式下发送START起始位中断标志
I2C_INT_FLAG_ADDSEND	主机模式下成功发送了地址，从机模式下接收到了地址并且和自身的地址匹配中断标志
I2C_INT_FLAG_BTC	字节发送结束
I2C_INT_FLAG_ADD10SEND	主机模式下10位地址的地址头被发送中断标志
I2C_INT_FLAG_STPDET	从机模式下监测到STOP结束位中断标志
I2C_INT_FLAG_RBNE	接收期间I2C_DATA非空中断标志
I2C_INT_FLAG_TBE	发送期间I2C_DATA为空中断标志
I2C_INT_FLAG_BERR	总线错误，表示I2C总线上发生了预料之外的START起始位或STOP结束位中断标志
I2C_INT_FLAG_LOSTARB	主机模式下仲裁丢失中断标志
I2C_INT_FLAG_AERR	应答错误中断标志
I2C_INT_FLAG_OUERR	从机接收模式下，发生了过载或欠载事件中断标志
I2C_INT_FLAG_PECERR	接收数据时PEC错误中断标志
I2C_INT_FLAG_SMBTO	SMBus模式下超时信号中断标志
I2C_INT_FLAG_SMBALT	SMBus警报状态中断标志

### 枚举 i2c\_interrupt\_enum

表 3-266. 枚举类型 i2c\_interrupt\_enum

成员名称	功能描述
I2C_INT_ERR	错误中断使能
I2C_INT_EV	事件中断使能
I2C_INT_BUF	缓冲区中断使能

**函数 i2c\_deinit**

函数i2c\_deinit描述见下表:

**表 3-267. 函数 i2c\_deinit**

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

例如:

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

```
i2c_deinit(I2C0);
```

**函数 i2c\_clock\_config**

函数i2c\_clock\_config描述见下表:

**表 3-268. 函数 i2c\_clock\_config**

函数名称	i2c_clock_config
函数原型	void i2c_clock_config(uint32_t i2c_periph, uint32_t clkspeed, uint32_t dutycyc);
功能描述	配置I2C时钟
先决条件	-
被调用函数	rcu_clock_freq_get
输入参数{in}	
i2c_periph	I2C外设
I2Cx	(x=0,1)
输入参数{in}	
clkspeed	i2c时钟速率
输入参数{in}	
dutycyc	快速模式下占空比
I2C_DTCY_2	T_low/T_high=2
I2C_DTCY_16_9	T_low/T_high=16/9
输出参数{out}	
-	-

返回值	
-	-

例如：

```
/* configure I2C0 clock speed as 100KHz */
```

```
i2c_clock_config(I2C0, 100000, I2C_DTCY_2);
```

### 函数 i2c\_mode\_addr\_config

函数i2c\_mode\_addr\_config描述见下表：

表 3-269. 函数 i2c\_mode\_addr\_config

函数名称	i2c_mode_addr_config
函数原型	void i2c_mode_addr_config(uint32_t i2c_periph, uint32_t mode, uint32_t addformat, uint32_t addr);
功能描述	配置I2C地址
先决条件	-
被调用函数	-
输入参数{in}	
i2c_periph	I2C外设
I2Cx	(x=0,1)
输入参数{in}	
i2cmod	模式选择
I2C_I2CMODE_ENABLE	I2C 模式
I2C_SMBUSMODE_ENABLE	SMBus 模式
输入参数{in}	
addformat	7bits 或 10bits
I2C_ADDFORMAT_7BITS	地址格式为7bits
I2C_ADDFORMAT_10BITS	地址格式为10bits
输入参数{in}	
addr	I2C地址
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* configure I2C0 address as 0x82, using 7 bits */
```

i2c\_mode\_addr\_config(I2C0, I2C\_I2CMODE\_ENABLE, I2C\_ADDFORMAT\_7BITS, 0x82);

### 函数 i2c\_smbus\_type\_config

函数i2c\_smbus\_type\_config描述见下表:

表 3-270. 函数 i2c\_smbus\_type\_config

函数名称	i2c_smbus_type_config
函数原型	void i2c_smbus_type_config(uint32_t i2c_periph, uint32_t type);
功能描述	SMBus类型选择
先决条件	-
被调用函数	-
输入参数{in}	
i2c_periph	I2C外设
I2Cx	(x=0,1)
输入参数{in}	
type	主机或从机
I2C_SMBUS_DEVICE	从机
I2C_SMBUS_HOST	主机
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* config I2C0 as SMBUS host type */
```

```
i2c_smbus_type_config (I2C0, I2C_SMBUS_HOST);
```

### 函数 i2c\_ack\_config

函数i2c\_ack\_config描述见下表:

表 3-271. 函数 i2c\_ack\_config

函数名称	i2c_ack_config
函数原型	void i2c_ack_config(uint32_t i2c_periph, uint32_t ack);
功能描述	是否发送ACK
先决条件	-
被调用函数	-
输入参数{in}	
i2c_periph	I2C外设
I2Cx	(x=0,1)
输入参数{in}	

<b>ack</b>	是否发送ACK
<i>I2C_ACK_ENABLE</i>	ACK会被发送
<i>I2C_ACK_DISABLE</i>	ACK不会发送
<b>输出参数{out}</b>	
-	-
<b>返回值</b>	
-	-

例如：

```
/* I2C0 will sent ACK */
```

```
i2c_ack_config (I2C0, I2C_ACK_ENABLE);
```

### 函数 i2c\_ackpos\_config

函数i2c\_ackpos\_config描述见下表：

**表 3-272. 函数 i2c\_ackpos\_config**

<b>函数名称</b>	i2c_ackpos_config
<b>函数原型</b>	void i2c_ackpos_config(uint32_t i2c_periph, uint32_t pos);
<b>功能描述</b>	配置在接收模式下ACK和PEC的位置
<b>先决条件</b>	-
<b>被调用函数</b>	-
<b>输入参数{in}</b>	
<b>i2c_periph</b>	I2C外设
<i>I2Cx</i>	(x=0,1)
<b>输入参数{in}</b>	
<b>pos</b>	ACK位置
<i>I2C_ACKPOS_CURRENT</i>	当前正在接收的字节是否发送ACK
<i>I2C_ACKPOS_NEXT</i>	下一个接收的字节是否发送ACK
<b>输出参数{out}</b>	
-	-
<b>返回值</b>	
-	-

例如：

```
/* The ACK of I2C0 is send for the current frame */
```

```
i2c_ackpos_config (I2C0, I2C_ACKPOS_CURRENT);
```

### 函数 i2c\_master\_addressing

函数i2c\_master\_addressing描述见下表：

表 3-273. 函数 i2c\_master\_addressing

函数名称	i2c_master_addressing
函数原型	void i2c_master_addressing(uint32_t i2c_periph, uint32_t addr, uint32_t trandirection);
功能描述	主机发送从机地址
先决条件	-
被调用函数	-
输入参数{in}	
i2c_periph	I2C外设
I2Cx	(x=0,1)
输入参数{in}	
addr	从机地址
输入参数{in}	
trandirection	发送或接收
I2C_TRANSMITTER	发送
I2C_RECEIVER	接收
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* send slave address to I2C bus and I2C0 act as receiver */
```

```
i2c_master_addressing(I2C0, 0x82, I2C_RECEIVER);
```

### 函数 i2c\_dualaddr\_enable

函数i2c\_dualaddr\_enable描述见下表：

表 3-274. 函数 i2c\_dualaddr\_enable

函数名称	i2c_dualaddr_enable
函数原型	void i2c_dualaddr_enable(uint32_t i2c_periph, uint32_t addr);
功能描述	双地址模式使能
先决条件	-
被调用函数	-
输入参数{in}	
i2c_periph	I2C外设
I2Cx	(x=0,1)
输入参数{in}	
addr	双地址模式下第二个地址
输出参数{out}	



-	-
返回值	
-	-

例如：

```
/* enable I2C0 dual-address */
```

```
i2c_dualaddr_enable (I2C0, 0x80);
```

### 函数 i2c\_dualaddr\_disable

函数i2c\_dualaddr\_disable描述见下表：

表 3-275. 函数 i2c\_dualaddr\_disable

函数名称	i2c_dualaddr_disable
函数原型	void i2c_dualaddr_disable(uint32_t i2c_periph);
功能描述	双地址模式禁能
先决条件	-
被调用函数	-
输入参数{in}	
i2c_periph	I2C外设
I2Cx	(x=0,1)
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* disable dual-address mode */
```

```
i2c_dualaddr_disable (I2C0);
```

### 函数 i2c\_enable

函数i2c\_enable描述见下表：

表 3-276. 函数 i2c\_enable

函数名称	i2c_enable
函数原型	void i2c_enable(uint32_t i2c_periph);
功能描述	使能I2C模块
先决条件	-
被调用函数	-
输入参数{in}	
i2c_periph	I2C外设
I2Cx	(x=0,1)

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 3-277. 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模块
先决条件	-
被调用函数	-
输入参数{in}	
i2c_periph	I2C外设
I2Cx	(x=0,1)
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 3-278. 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

<i>I2Cx</i>	(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 3-279. 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
<i>I2Cx</i>	(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 3-280. 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}	

<b>i2c_periph</b>	I2C外设
<i>I2Cx</i>	(x=0,1)
<b>输入参数{in}</b>	
<b>data</b>	传输的数据
<b>输出参数{out}</b>	
-	-
<b>返回值</b>	
-	-

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 3-281. 函数 i2c\_data\_receive

<b>函数名称</b>	i2c_data_receive
<b>函数原型</b>	uint8_t i2c_data_receive(uint32_t i2c_periph);
<b>功能描述</b>	接收数据
<b>先决条件</b>	-
<b>被调用函数</b>	-
<b>输入参数{in}</b>	
<b>i2c_periph</b>	I2C外设
<i>I2Cx</i>	(x=0,1)
<b>输出参数{out}</b>	
-	-
<b>返回值</b>	
<b>uint8_t</b>	0x00..0xFF

例如：

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

```
uint8_t i2c_receiver;
```

```
i2c_receiver = i2c_data_receive (I2C0);
```

### 函数 i2c\_dma\_config

函数i2c\_dma\_config描述见下表：

表 3-282. 函数 i2c\_dma\_config

<b>函数名称</b>	i2c_dma_config
<b>函数原型</b>	void i2c_dma_config (uint32_t i2c_periph, uint32_t dmastate);

功能描述	I2C DMA模式使能
先决条件	-
被调用函数	-
输入参数{in}	
i2c_periph	I2C外设
I2Cx	(x=0,1)
输入参数{in}	
dmastate	开启或关闭
I2C_DMA_ON	DMA模式开启
I2C_DMA_OFF	DMA模式关闭
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* I2C0 DMA mode config */
```

```
i2c_dma_config (I2C0, I2C_DMA_ON);
```

### 函数 i2c\_dma\_last\_transfer\_config

函数i2c\_dma\_last\_transfer\_config描述见下表：

表 3-283. 函数 i2c\_dma\_last\_transfer\_config

函数名称	i2c_dma_last_transfer_config
函数原型	void i2c_dma_last_transfer_config(uint32_t i2c_periph, uint32_t dmalast);
功能描述	配置下一个DMA EOT是否是DMA最后一次传输
先决条件	-
被调用函数	-
输入参数{in}	
i2c_periph	I2C外设
I2Cx	(x=0,1)
输入参数{in}	
dmalast	下一个DMA EOT是否是DMA最后一次传输
I2C_DMALST_ON	下一个DMA EOT是DMA最后一次传输
I2C_DMALST_OFF	下一个DMA EOT不是DMA最后一次传输
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* next DMA EOT is the last transfer */
```

i2c\_dma\_last\_transfer\_config (I2C0, I2C\_DMALST\_ON);

### 函数 i2c\_stretch\_scl\_low\_config

函数i2c\_stretch\_scl\_low\_config描述见下表:

表 3-284. 函数 i2c\_stretch\_scl\_low\_config

函数名称	i2c_stretch_scl_low_config
函数原型	void i2c_stretch_scl_low_config(uint32_t i2c_periph, uint32_t stretchpara);
功能描述	在从机模式下数据没有准备好时是否拉低SCL
先决条件	-
被调用函数	-
输入参数{in}	
i2c_periph	I2C外设
I2Cx	(x=0,1)
输入参数{in}	
stretchpara	是否拉低SCL
I2C_SCLSTRETCH_ENABLE	拉低SCL
I2C_SCLSTRETCH_DISABLE	不拉低SCL
输出参数{out}	
-	-
返回值	
-	-

例如:

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

```
i2c_stretch_scl_low_config (I2C0, I2C_SCLSTRETCH_ENABLE);
```

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

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

表 3-285. 函数 i2c\_slave\_response\_to\_gcall\_config

函数名称	i2c_slave_response_to_gcall_config
函数原型	void i2c_slave_response_to_gcall_config(uint32_t i2c_periph, uint32_t gcallpara);
功能描述	从机是否响应广播呼叫
先决条件	-
被调用函数	-
输入参数{in}	
i2c_periph	I2C外设

<i>I2Cx</i>	(x=0,1)
输入参数{in}	
<b>gcallpara</b>	是否响应广播呼叫
<i>I2C_GCEN_ENABL</i> <i>E</i>	从机响应广播呼叫
<i>I2C_GCEN_DISABL</i> <i>E</i>	从机不响应广播呼叫
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* I2C0 will response to a general call */
```

```
i2c_slave_response_to_gcall_config (I2C0, I2C_GCEN_ENABLE);
```

### 函数 i2c\_software\_reset\_config

函数i2c\_software\_reset\_config描述见下表:

表 3-286. 函数 i2c\_software\_reset\_config

函数名称	i2c_software_reset_config
函数原型	void i2c_software_reset_config(uint32_t i2c_periph, uint32_t sreset);
功能描述	配置I2C软件复位
先决条件	-
被调用函数	-
输入参数{in}	
<b>i2c_periph</b>	I2C外设
<i>I2Cx</i>	(x=0,1)
输入参数{in}	
<b>sreset</b>	是否复位
<i>I2C_SRESET_SET</i>	复位
<i>I2C_SRESET_RES</i> <i>ET</i>	没有复位
输出参数{out}	
-	-
返回值	
-	-

例如:

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

```
i2c_software_reset_config (I2C0, I2C_SRESET_SET);
```

**函数 i2c\_pec\_config**

函数i2c\_pec\_config描述见下表:

**表 3-287. 函数 i2c\_pec\_config**

函数名称	i2c_pec_config
函数原型	void i2c_pec_config(uint32_t i2c_periph, uint32_t pecstate);
功能描述	是否使能报文错误校验
先决条件	-
被调用函数	-
输入参数{in}	
i2c_periph	I2C外设
I2Cx	(x=0,1)
输入参数{in}	
pecpara	开启或关闭
I2C_PEC_ENABLE	报文错误校验使能
I2C_PEC_DISABLE	报文错误校验关闭
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* config I2C PEC calculation */
```

```
i2c_pec_config (I2C0, I2C_PEC_ENABLE);
```

**函数 i2c\_pec\_transfer\_config**

函数i2c\_pec\_transfer\_config描述见下表:

**表 3-288. 函数 i2c\_pec\_transfer\_config**

函数名称	i2c_pec_transfer_config
函数原型	void i2c_pec_transfer_config(uint32_t i2c_periph, uint32_t pecpara);
功能描述	配置I2C传输PEC值
先决条件	-
被调用函数	-
输入参数{in}	
i2c_periph	I2C外设
I2Cx	(x=0,1)
输入参数{in}	
pecpara	是否传输PEC
I2C_PECTRANS_ENABLE	传输PEC



<i>I2C_PECTRANS_DISABLE</i>	不传输PEC
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* I2C0 transfer PEC */
```

```
i2c_pec_transfer_config (I2C0, I2C_PECTRANS_ENABLE);
```

### 函数 i2c\_pec\_value\_get

函数i2c\_pec\_value\_get描述见下表:

表 3-289. 函数 i2c\_pec\_value\_get

函数名称	i2c_pec_value_get
函数原型	uint8_t i2c_pec_value_get(uint32_t i2c_periph);
功能描述	获取报文错误校验值
先决条件	-
被调用函数	-
输入参数{in}	
i2c_periph	I2C外设
I2Cx	(x=0,1)
输出参数{out}	
-	-
返回值	
uint8_t	PEC值

例如:

```
/* I2C0 get packet error checking value */
```

```
uint8_t pec_value;
```

```
pec_value = i2c_pec_value_get (I2C0);
```

### 函数 i2c\_smbus\_alert\_config

函数i2c\_smbus\_alert\_config描述见下表:

表 3-290. 函数 i2c\_smbus\_alert\_config

函数名称	i2c_smbus_alert_config
函数原型	void i2c_smbus_alert_config (uint32_t i2c_periph, uint32_t smbuspara);
功能描述	通过SMBA引脚配置I2C值
先决条件	-

被调用函数	-
输入参数{in}	
i2c_periph	I2C外设
I2Cx	(x=0,1)
输入参数{in}	
smbuspara	是否通过SMBA引脚发送警告
I2C_SALTSEND_ENABLE	通过SMBA引脚发送警告
I2C_SALTSEND_DISABLE	不通过SMBA引脚发送警告
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* I2C0 issue alert through SMBA pin enable */
```

```
i2c_smbus_alert_config (I2C0, I2C_SALTSEND_ENABLE);
```

### 函数 i2c\_smbus\_arp\_config

函数i2c\_smbus\_arp\_config描述见下表：

表 3-291. 函数 i2c\_smbus\_arp\_config

函数名称	i2c_smbus_arp_config
函数原型	void i2c_smbus_arp_config(uint32_t i2c_periph, uint32_t arpstate);
功能描述	在SMBus下配置ARP协议
先决条件	-
被调用函数	-
输入参数{in}	
i2c_periph	I2C外设
I2Cx	(x=0,1)
输入参数{in}	
arpstate	SMBus下ARP协议是否开启
I2C_ARP_ENABLE	使能ARP
I2C_ARP_DISABLE	关闭ARP
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* config I2C0 ARP protocol in SMBus switch */
```

i2c\_smbus\_arp\_config (I2C0, I2C\_ARP\_ENABLE);

## 函数 i2c\_flag\_get

函数i2c\_flag\_get描述见下表:

表 3-292. 函数 i2c\_flag\_get

函数名称	i2c_flag_get
函数原型	FlagStatus i2c_flag_get(uint32_t i2c_periph, i2c_flag_enum flag);
功能描述	标志位获取
先决条件	-
被调用函数	-
输入参数{in}	
i2c_periph	I2C外设
I2Cx	(x=0,1)
输入参数{in}	
flag	需要获取的标志位
I2C_FLAG_SBSEN D	起始位是否发送
I2C_FLAG_ADDSE ND	主机模式下地址是否发送/从机模式下地址是否匹配
I2C_FLAG_BTC	字节传输完成
I2C_FLAG_ADD10 SEND	主机模式下10位地址地址头发送完成
I2C_FLAG_STPDE T	从机模式下监测到STOP结束位
I2C_FLAG_RBNE	接收期间I2C_DATA非空
I2C_FLAG_TBE	发送期间I2C_DATA为空
I2C_FLAG_BERR	总线错误, 表示I2C总线上发生了预料之外的START起始位或STOP结束位
I2C_FLAG_LOSTA RB	主机模式下仲裁丢失
I2C_FLAG_AERR	应答错误
I2C_FLAG_OUERR	当禁用SCL拉低功能后, 在从机模式下发生了过载或欠载事件
I2C_FLAG_PECER R	接收数据时发生PEC错误
I2C_FLAG_SMBTO	SMBus模式下超时信号
I2C_FLAG_SMBAL T	SMBus警报状态
I2C_FLAG_MASTE R	表明I2C时钟在主机模式还是从机模式的标志位
I2C_FLAG_I2CBSY	忙标志
I2C_FLAG_TR	I2C作发送端还是接收端
I2C_FLAG_RXGC	是否接收到广播地址(00h)

<i>I2C_FLAG_DEFSM</i> <i>B</i>	从机模式下SMBus主机地址头
<i>I2C_FLAG_HSTSM</i> <i>B</i>	从机模式下监测到SMBus主机地址头
<i>I2C_FLAG_DUMOD</i>	从机模式下双标志位表明哪个地址和双地址模式匹配
输出参数{out}	
-	-
返回值	
FlagStatus	SET / RESET

例如:

```
/* check whether start condition send out */
```

```
FlagStatus flag_state = RESET;
```

```
flag_state = i2c_flag_get (I2C0, I2C_FLAG_SBSEND);
```

### 函数 i2c\_flag\_clear

函数i2c\_flag\_clear描述见下表:

表 3-293. 函数 i2c\_flag\_clear

函数名称	i2c_flag_clear
函数原型	void i2c_flag_clear(uint32_t i2c_periph, i2c_flag_enum flag)
功能描述	清除标志位
先决条件	-
被调用函数	-
输入参数{in}	
i2c_periph	I2C外设
<i>I2Cx</i>	(x=0,1)
输入参数{in}	
flag	标志位类型
<i>I2C_FLAG_SMBAL</i> <i>T</i>	SMBus警报状态
<i>I2C_FLAG_SMBTO</i>	SMBus模式下超时信号
<i>I2C_FLAG_PECER</i> <i>R</i>	接收数据时PEC错误
<i>I2C_FLAG_OUERR</i>	当禁用SCL拉低功能后, 在从机模式下发生了过载或欠载事件
<i>I2C_FLAG_AERR</i>	应答错误
<i>I2C_FLAG_LOSTA</i> <i>RB</i>	主机模式下仲裁丢失
<i>I2C_FLAG_BERR</i>	总线错误
<i>I2C_FLAG_ADDSE</i> <i>ND</i>	主机模式下地址是否发送/从机模式下地址是否匹配, 通过读I2C_STAT0和I2C_STAT1来清除

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 3-294. 函数 i2c\_interrupt\_enable**

函数名称	i2c_interrupt_enable
函数原型	void i2c_interrupt_enable(uint32_t i2c_periph, i2c_interrupt_enum interrupt);
功能描述	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>	中断类型
<i>I2C_INT_ERR</i>	错误中断使能
<i>I2C_INT_EV</i>	事件中断使能
<i>I2C_INT_BUF</i>	缓冲区中断使能
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* enable I2C0 event interrupt */
```

```
i2c_interrupt_enable (I2C0, I2C_INT_EV);
```

### 函数 i2c\_interrupt\_disable

函数i2c\_interrupt\_disable描述见下表：

**表 3-295. 函数 i2c\_interrupt\_disable**

函数名称	i2c_interrupt_disable
函数原型	void i2c_interrupt_disable(uint32_t i2c_periph, i2c_interrupt_enum interrupt);

功能描述	中断禁能
先决条件	-
被调用函数	-
输入参数{in}	
i2c_periph	I2C外设
I2Cx	(x=0,1)
输入参数{in}	
interrupt	中断类型
I2C_INT_ERR	错误中断使能
I2C_INT_EV	事件中断使能
I2C_INT_BUF	缓冲区中断使能
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* disable I2C0 event interrupt */
```

```
i2c_interrupt_disable (I2C0, I2C_INT_EV);
```

### 函数 i2c\_interrupt\_flag\_get

函数i2c\_interrupt\_flag\_get描述见下表：

表 3-296. 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	中断标志位获取
先决条件	-
被调用函数	-
输入参数{in}	
i2c_periph	I2C外设
I2Cx	(x=0,1)
输入参数{in}	
int_flag	中断标志
I2C_INT_FLAG_SB SEND	主机模式下发送START起始位
I2C_INT_FLAG_AD DSEND	主机模式下成功发送了地址 / 从机模式下接收到了地址并且和自身的地址匹配
I2C_INT_FLAG_BT C	字节发送结束

<i>I2C_INT_FLAG_AD D10SEND</i>	主机模式下10位地址地址头被发送
<i>I2C_INT_FLAG_ST PDET</i>	从机模式下监测到STOP结束位
<i>I2C_INT_FLAG_RB NE</i>	接收期间I2C_DATA非空
<i>I2C_INT_FLAG_TB E</i>	发送期间I2C_DATA为空
<i>I2C_INT_FLAG_BE RR</i>	总线错误
<i>I2C_INT_FLAG_LO STARB</i>	主机模式下仲裁丢失
<i>I2C_INT_FLAG_AE RR</i>	应答错误
<i>I2C_INT_FLAG_OU ERR</i>	当禁用SCL拉低功能后，在从机模式下发生了过载或欠载事件
<i>I2C_INT_FLAG_PE CERR</i>	接收数据时PEC错误
<i>I2C_INT_FLAG_SM BTO</i>	SMBus模式下超时信号
<i>I2C_INT_FLAG_SM BALT</i>	SMBus警报状态
输出参数{out}	
-	-
返回值	
FlagStatus	SET / RESET

例如：

```
/* check the byte transmission finishes interrupt flag is set or not */
```

```
FlagStatus flag_state = RESET;
```

```
flag_state = i2c_interrupt_flag_get (I2C0, I2C_INT_FLAG_BTC);
```

### 函数 i2c\_interrupt\_flag\_clear

函数i2c\_interrupt\_flag\_clear描述见下表：

**表 3-297. 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	中断标志位清除
先决条件	-

被调用函数	-
输入参数{in}	
i2c_periph	I2C外设
I2Cx	(x=0,1)
输入参数{in}	
int_flag	中断标志
I2C_INT_FLAG_AD DSEND	主机模式下成功发送了地址 / 从机模式下接收到了地址并且和自身的地址匹配
I2C_INT_FLAG_BE RR	总线错误
I2C_INT_FLAG_LO STARB	主机模式下仲裁丢失
I2C_INT_FLAG_AE RR	应答错误
I2C_INT_FLAG_OU ERR	当禁用SCL 拉低功能后，在从机模式下发生了过载或欠载事件
I2C_INT_FLAG_PE CERR	接收数据时PEC错误
I2C_INT_FLAG_SM BTO	SMBus模式下超时信号
I2C_INT_FLAG_SM BALT	SMBus警报状态
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* clear the acknowledge error interrupt flag */
```

```
i2c_interrupt_flag_clear (I2C0, I2C_INT_FLAG_AERR);
```

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



## 3.15.1. 外设寄存器说明

表 3-298. NVIC 寄存器

寄存器名称	寄存器描述
ISER <sup>(1)</sup>	中断使能寄存器
ICER <sup>(1)</sup>	中断禁能寄存器
ISPR <sup>(1)</sup>	中断挂起寄存器
ICPR <sup>(1)</sup>	中断清除寄存器
IABR <sup>(1)</sup>	中断活动状态寄存器
IP <sup>(1)</sup>	中断优先级寄存器
STIR <sup>(1)</sup>	软触发中断寄存器
CPUID <sup>(2)</sup>	CPUID寄存器
ICSR <sup>(2)</sup>	中断控制及状态寄存器
VTOR <sup>(2)</sup>	向量表偏移量寄存器
AIRCR <sup>(2)</sup>	应用程序中断及复位控制寄存器
SCR <sup>(2)</sup>	系统控制寄存器
CCR <sup>(2)</sup>	配置与控制寄存器
SHP <sup>(2)</sup>	系统异常优先级寄存器
SHCSR <sup>(2)</sup>	系统异常控制及状态寄存器
CFSR <sup>(2)</sup>	配置错误状态寄存器
HFSR <sup>(2)</sup>	硬错误状态寄存器
DFSR <sup>(2)</sup>	调试错误状态寄存器
MMFAR <sup>(2)</sup>	存储管理错误地址寄存器
BFAR <sup>(2)</sup>	总线错误地址寄存器
AFSR <sup>(2)</sup>	辅助错误地址寄存器
PFR <sup>(2)</sup>	处理器特性寄存器
DFR <sup>(2)</sup>	调试特性寄存器
ADR <sup>(2)</sup>	辅助特性寄存器
MMFR <sup>(2)</sup>	存储模型特性寄存器
ISAR <sup>(2)</sup>	指令设置属性寄存器
CPACR <sup>(2)</sup>	协处理器访问控制寄存器

1. 参考 core\_cm4.h 文件中定义的结构体类型 NVIC\_Type
2. 参考 core\_cm4.h 文件中定义的结构体类型 SCB\_Type

表 3-299. 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\_cm4.h 文件中定义的结构体类型 SysTick\_Type

### 3.15.2. 外设库函数说明

MISC 库函数列表如下表所示：

**表 3-300. 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
nvic_system_reset	复位
nvic_vector_table_set	设置向量表地址
system_lowpower_set	设置系统低功耗模式状态
system_lowpower_reset	复位系统低功耗模式状态
systick_clksource_set	设置系统定时器时钟源

### 枚举类型 IRQn\_Type

**表 3-301. 枚举类型 IRQn\_Type**

成员名称	功能描述
WWDGT_IRQn	窗口看门狗中断
LVD_IRQn	连接到 EXTI 线的 LVD 中断
RTC_IRQn	侵入检测中断
FMC_IRQn	FMC 全局中断
RCU_CTC_IRQn	RCU 与 CTC 全局中断
EXTI0_1_IRQn	EXTI 线 0 和线 1 中断
EXTI2_3_IRQn	EXTI 线 2 和线 3 中断
EXTI4_15_IRQn	EXTI 线 4 到线 15 中断
TSI_IRQn	TSI 中断
DMA_Channel0_IRQn	DMA0 通道 0 全局中断
DMA_Channel1_2_IRQn	DMA0 通道 1 与通道 2 全局中断
DMA_Channel3_4_IRQn	DMA0 通道 3 与通道 4 全局中断
ADC_CMP_IRQn	ADC 与 CMP 全局中断
TIMER0_BRK_UP_TRG_COM_IRQn	TIMER0 的中止，更新，触发和换相中断
TIMER0_Channel_1_IRQn	TIMER0 的捕获中断
TIMER1_IRQn	TIMER1 全局中断
TIMER2_IRQn	TIMER2 全局中断
TIMER5_DAC_IRQn	TIMER5 与 DAC 全局中断
TIMER13_IRQn	TIMER13 全局中断

成员名称	功能描述
TIMER14_IRQn	TIMER14 全局中断
TIMER15_IRQn	TIMER15 全局中断
TIMER16_IRQn	TIMER16 全局中断
I2C0_EV_IRQn	I2C0 事件中断
I2C1_EV_IRQn	I2C1 事件中断
SPI0_IRQn	SPI0 全局中断
SPI1_IRQn	SPI1 全局中断
USART0_IRQn	USART0 全局中断
USART1_IRQn	USART1 全局中断
CEC_IRQn	CEC 全局中断
I2C0_ER_IRQn	I2C0 错误中断
I2C1_ER_IRQn	I2C1 错误中断
DMA_Channel5_6_IRQn	DMA0 通道 5 与通道 6 全局中断
USBFS_WKUP_IRQn	USBFS 唤醒中断
USBFS_IRQn	USBFS 全局中断

## 函数 nvic\_priority\_group\_set

函数nvic\_priority\_group\_set描述见下表：

表 3-302. 函数 nvic\_priority\_group\_set

函数名称	nvic_priority_group_set
函数原形	void nvic_priority_group_set(uint32_t nvic_prigroup);
功能描述	配置优先级组的位长度
先决条件	-
被调用函数	-
输入参数{in}	
nvic_prigroup	优先级组
NVIC_PRIGROUP_PRE0_SUB4	0位用于抢占优先级，4位用于响应优先级
NVIC_PRIGROUP_PRE1_SUB3	1位用于抢占优先级，3位用于响应优先级
NVIC_PRIGROUP_PRE2_SUB2	2位用于抢占优先级，2位用于响应优先级
NVIC_PRIGROUP_PRE3_SUB1	3位用于抢占优先级，1位用于响应优先级
NVIC_PRIGROUP_PRE4_SUB0	4位用于抢占优先级，0位用于响应优先级
输出参数{out}	
-	-

返回值	
-	-

例如：

```
/* priority group configuration , 0 bits for pre-emption priority 4 bits for subpriority */
nvic_priority_group_set(NVIC_PRIGROUP_PRE0_SUB4);
```

### 函数 nvic\_irq\_enable

函数nvic\_irq\_enable描述见下表：

表 3-303. 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	使能中断，配置中断的优先级
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3-301. 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 3-304. 函数 nvic\_irq\_disable

函数名称	nvic_irq_disable
函数原形	void nvic_irq_disable (uint8_t nvic_irq);
功能描述	禁能中断
先决条件	-
被调用函数	NVIC_DisableIRQ
输入参数{in}	
nvic_irq	NVIC中断，参考枚举类型 <a href="#">表3-301. 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);
```

### 数 nvic\_vector\_table\_set

函数nvic\_vector\_table\_set描述见下表：

**表 3-305. 函数 nvic\_vector\_table\_set**

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

例如：

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

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

### 函数 system\_lowpower\_set

函数system\_lowpower\_set描述见下表：

**表 3-306. 函数 system\_lowpower\_set**

函数名称	system_lowpower_set
函数原形	void system_lowpower_set(uint8_t lowpower_mode);
功能描述	系统低功耗模式状态的管理

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

例如：

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

### 函数 system\_lowpower\_reset

函数system\_lowpower\_reset描述见下表：

表 3-307. 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	系统将通过退出ISR退出低功耗模式
SCB_LPM_DEEPSLEEP	系统进入sleep模式
SCB_LPM_WAKE_BY_ALL_INT	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 3-308. 函数 `systick_clksource_set`**

函数名称	<code>systick_clksource_set</code>
函数原形	<code>void systick_clksource_set(uint32_t systick_clksource);</code>
功能描述	设置SysTick时钟源
先决条件	-
被调用函数	-
输入参数{in}	
<code>systick_clksource</code>	SysTick时钟源
<code>SYSTICK_CLKSOURCE_HCLK</code>	SysTick时钟源为AHB时钟
<code>SYSTICK_CLKSOURCE_HCLK_DIV8</code>	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);
```

## 3.16. PMU

电源管理单元提供了三种省电模式，包括睡眠模式，深度睡眠模式和待机模式。章节 [3.16.1](#) 描述了 PMU 的寄存器列表，章节 [3.16.2](#) 对 PMU 库函数进行说明。

### 3.16.1. 外设寄存器说明

PMU 寄存器列表如下表所示：

**表 3-309. PMU 寄存器**

寄存器名称	寄存器描述
PMU_CTL	PMU控制寄存器
PMU_CS	PMU控制和状态寄存器

### 3.16.2. 外设库函数说明

PMU 库函数列表如下表所示：

**表 3-310. PMU 库函数**

库函数名称	库函数描述
pmu_deinit	复位外设PMU
pmu_lvd_select	选择低压检测阈值
pmu_ldo_output_select	LDO输出电压选择
pmu_lvd_disable	关闭低压检测器
pmu_lowdriver_mode_enable	使能深度睡眠下低驱动模式
pmu_lowdriver_mode_disable	禁能深度睡眠下低驱动模式
pmu_highdriver_mode_enable	使能高驱动模式
pmu_highdriver_mode_disable	禁能高驱动模式
pmu_highdriver_switch_select	高驱动模式切换器选择
pmu_lowpower_driver_config	使用低功耗LDO时低驱动模式配置
pmu_normalpower_driver_config	使用正常功耗LDO时低驱动模式配置
pmu_to_sleepmode	进入睡眠模式
pmu_to_deepsleepmode	进入深度睡眠模式
pmu_to_standbymode	进入待机模式
pmu_wakeup_pin_enable	WKUP引脚唤醒使能
pmu_wakeup_pin_disable	WKUP引脚唤醒失能
pmu_backup_write_enable	备份域写使能
pmu_backup_write_disable	备份域写失能
pmu_flag_get	获取标志位
pmu_flag_clear	清除标志位

#### 函数 pmu\_deinit

函数pmu\_deinit描述见下表：

**表 3-311. 函数 pmu\_deinit**

函数名称	pmu_deinit
函数原型	void pmu_deinit(void);
功能描述	复位外设PMU
先决条件	-
被调用函数	rcu_periph_reset_enable / rcu_periph_reset_disable
输入参数{in}	
-	-
输出参数{out}	
-	-
返回值	
-	-



例如:

```
/* reset PMU */
```

```
pmu_deinit();
```

### 函数 pmu\_lvd\_select

函数pmu\_lvd\_select描述见下表:

表 3-312. 函数 pmu\_lvd\_select

函数名称	pmu_lvd_select
函数原型	void pmu_lvd_select(uint32_t lvdn);
功能描述	选择低压检测阈值
先决条件	-
被调用函数	-
输入参数{in}	
lvdn	电压阈值
PMU_LVDT_0	电压阈值为2.1V
PMU_LVDT_1	电压阈值为2.3V
PMU_LVDT_2	电压阈值为2.4V
PMU_LVDT_3	电压阈值为2.6V
PMU_LVDT_4	电压阈值为2.7V
PMU_LVDT_5	电压阈值为2.9V
PMU_LVDT_6	电压阈值为3.0V
PMU_LVDT_7	电压阈值为3.1V
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* select low voltage detector threshold as 3.1V */
```

```
pmu_lvd_select (PMU_LVDT_7);
```

### 函数 pmu\_ldo\_output\_select

函数pmu\_ldo\_output\_select描述见下表:

表 3-313. 函数 pmu\_ldo\_output\_select

函数名称	pmu_ldo_output_select
函数原型	void pmu_ldo_output_select(uint32_t ldo_output);
功能描述	内部电压调节器 (LDO) 输出电压选择
先决条件	-

被调用函数	-
输入参数{in}	
ldo_output	输出电压模式
PMU_LDOVS_LOW	输出低电压模式
PMU_LDOVS_MID	输出中等电压模式
PMU_LDOVS_HIG H	输出高电压模式
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* select output low voltage mode */
pmu_ldo_output_select (PMU_LDOVS_LOW);
```

### 函数 pmu\_lvd\_disable

函数pmu\_lvd\_disable描述见下表:

表 3-314. 函数 pmu\_lvd\_disable

函数名称	pmu_lvd_disable
函数原型	void pmu_lvd_disable (void);
功能描述	关闭低压检测器
先决条件	-
被调用函数	-
输入参数{in}	
-	-
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* disable PMU lvd */
pmu_lvd_disable ();
```

### 函数 pmu\_lowdriver\_mode\_enable

函数pmu\_lowdriver\_mode\_enable描述见下表:

表 3-315. 函数 pmu\_lowdriver\_mode\_enable

函数名称	pmu_lowdriver_mode_enable
------	---------------------------

函数原型	void pmu_lowdriver_mode_enable (void);
功能描述	使能深度睡眠下低驱动模式
先决条件	-
被调用函数	-
输入参数{in}	
-	-
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* enable low-driver mode */
pmu_lowdriver_mode_enable ();
```

### 函数 pmu\_lowdriver\_mode\_disable

函数pmu\_lowdriver\_mode\_disable描述见下表：

**表 3-316. 函数 pmu\_lowdriver\_mode\_disable**

函数名称	pmu_lowdriver_mode_disable
函数原型	void pmu_lowdriver_mode_disable (void);
功能描述	禁能深度睡眠下低驱动模式
先决条件	-
被调用函数	-
输入参数{in}	
-	-
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* disable low-driver mode */
pmu_lowdriver_mode_disable ();
```

### 函数 pmu\_highdriver\_mode\_enable

函数pmu\_highdriver\_mode\_enable描述见下表：

**表 3-317. 函数 pmu\_highdriver\_mode\_enable**

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

功能描述	使能高驱动模式
先决条件	-
被调用函数	-
输入参数{in}	
-	-
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* enable high-driver mode */
```

```
pmu_highdriver_mode_enable ();
```

### 函数 pmu\_highdriver\_mode\_disable

函数pmu\_highdriver\_mode\_disable描述见下表：

**表 3-318. 函数 pmu\_highdriver\_mode\_disable**

函数名称	pmu_highdriver_mode_disable
函数原型	void pmu_highdriver_mode_disable(void);
功能描述	失能高驱动模式
先决条件	-
被调用函数	-
输入参数{in}	
-	-
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* disable high-driver mode */
```

```
pmu_highdriver_mode_disable ();
```

### 函数 pmu\_highdriver\_switch\_select

函数pmu\_highdriver\_switch\_select描述见下表：

**表 3-319. 函数 pmu\_highdriver\_switch\_select**

函数名称	pmu_highdriver_switch_select
函数原型	void pmu_highdriver_switch_select(uint32_t highdr_switch);
功能描述	高驱动模式切换器选择

先决条件	-
被调用函数	-
输入参数{in}	
highdr_switch	高驱动模式切换器
PMU_HIGHDR_SW ITCH_NONE	失能高驱动模式切换器
PMU_HIGHDR_SW ITCH_EN	使能高驱动模式切换器
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* enable high-driver mode switch */
```

```
pmu_highdriver_switch_select (PMU_HIGHDR_SWITCH_EN);
```

### 函数 pmu\_lowpower\_driver\_config

函数pmu\_lowpower\_driver\_config描述见下表：

表 3-320. 函数 pmu\_lowpower\_driver\_config

函数名称	pmu_lowpower_driver_config
函数原型	void pmu_lowpower_driver_config(uint32_t mode);
功能描述	使用低驱动LDO时低驱动模式配置
先决条件	-
被调用函数	-
输入参数{in}	
mode	驱动模式
PMU_NORMALDR_ LOWPWR	使用低驱动LDO时工作在正常驱动模式
PMU_LOWDR_LO WPWR	使用低驱动LDO且LDEN为1时低驱动模式使能
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* low-driver mode when use low power LDO */
```

```
pmu_lowpower_driver_config (PMU_NORMALDR_LOWPWR);
```

**函数 pmu\_normalpower\_driver\_config**

函数pmu\_normalpower\_driver\_config描述见下表:

**表 3-321. 函数 pmu\_normalpower\_driver\_config**

函数名称	pmu_normalpower_driver_config
函数原型	void pmu_normalpower_driver_config(uint32_t mode);
功能描述	使用正常驱动LDO时低驱动模式配置
先决条件	-
被调用函数	-
输入参数{in}	
mode	驱动模式
PMU_NORMALDR_ NORMALPWR	使用正常驱动LDO时工作在正常驱动模式
PMU_NORMALDR_ NORMALPWR	使用正常驱动LDO且LDEN为1时时低驱动驱动模式使能
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* low-driver mode when use low power LDO */
```

```
pmu_normalpower_driver_config(PMU_NORMALDR_LOWPWR);
```

**函数 pmu\_to\_sleepmode**

函数pmu\_to\_sleepmode描述见下表:

**表 3-322. 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 3-323. 函数 pmu\_to\_deepsleepmode

函数名称	pmu_to_deepsleepmode
函数原型	void pmu_to_deepsleepmode(uint32_t ldo, uint32_t lowdrive, uint8_t deepsleepmodecmd);
功能描述	进入深度睡眠模式
先决条件	-
被调用函数	-
输入参数{in}	
ldo	LDO工作模式
PMU_LDO_NORMAL	当系统进入深度睡眠模式时，LDO仍正常工作
PMU_LDO_LOWPOWER	当系统进入深度睡眠模式时，LDO进入低功耗模式
输入参数{in}	
lowdrive	使能或失能低驱动模式
PMU_LOWDRIVER_ENABLE	在深度睡眠模式时，使能低驱动模式
PMU_LOWDRIVER_DISABLE	在深度睡眠模式时，失能低驱动模式
输入参数{in}	
deepsleepmodecmd	进入深度睡眠模式命令
WFI_CMD	WFI命令
WFE_CMD	WFE命令
输出参数{out}	
-	-
返回值	
-	-

例如：

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

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

## 函数 pmu\_to\_standbymode

函数pmu\_to\_standbymode描述见下表:

表 3-324. 函数 pmu\_to\_standbymode

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

例如:

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

```
pmu_to_standbymode();
```

## 函数 pmu\_wakeup\_pin\_enable

函数pmu\_wakeup\_pin\_enable描述见下表:

表 3-325. 函数 pmu\_wakeup\_pin\_enable

函数名称	pmu_wakeup_pin_enable
函数原型	void pmu_wakeup_pin_enable(uint32_t wakeup_pin);
功能描述	WKUP引脚唤醒使能
先决条件	-
被调用函数	-
输入参数{in}	
wakeup_pin	Wakeup pin
PMU_WAKEUP_PIN0	WKUP Pin 0 (PA0)
PMU_WAKEUP_PIN1	WKUP Pin 1 (PC13)
PMU_WAKEUP_PIN4	WKUP Pin 4 (PC5)
PMU_WAKEUP_PIN5	WKUP Pin 5 (PB5)
PMU_WAKEUP_PIN6	WKUP Pin 6 (PB15)



输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* enable wakeup pin6 */
```

```
pmu_wakeup_pin_enable (PMU_WAKEUP_PIN6);
```

### 函数 pmu\_wakeup\_pin\_disable

函数pmu\_wakeup\_pin\_disable描述见下表:

**表 3-326. 函数 pmu\_wakeup\_pin\_disable**

函数名称	pmu_wakeup_pin_disable
函数原型	void pmu_wakeup_pin_disable(uint32_t wakeup_pin);
功能描述	WKUP引脚唤醒失能
先决条件	-
被调用函数	-
输入参数{in}	
wakeup_pin	Wakeup pin
PMU_WAKEUP_PIN0	WKUP Pin 0 (PA0)
PMU_WAKEUP_PIN1	WKUP Pin 1 (PC13)
PMU_WAKEUP_PIN4	WKUP Pin 4 (PC5)
PMU_WAKEUP_PIN5	WKUP Pin 5 (PB5)
PMU_WAKEUP_PIN6	WKUP Pin 6 (PB15)
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* disable wakeup pin6 */
```

```
pmu_wakeup_pin_disable (PMU_WAKEUP_PIN6);
```

### 函数 pmu\_backup\_write\_enable

函数pmu\_backup\_write\_enable描述见下表:

表 3-327. 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 3-328. 函数 pmu\_backup\_write\_disable

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

例如:

```
/* disable backup domain write */
pmu_backup_write_disable ();
```

### 函数 pmu\_flag\_get

函数pmu\_flag\_get描述见下表:

表 3-329. 函数 `pmu_flag_get`

函数名称	<code>pmu_flag_get</code>
函数原型	<code>FlagStatus pmu_flag_get(uint32_t flag);</code>
功能描述	获取标志位
先决条件	-
被调用函数	-
输入参数{in}	
<code>flag_clear</code>	标志位
<code>PMU_FLAG_WAKEUP</code>	唤醒标志
<code>PMU_FLAG_STANDBY</code>	待机标志
<code>PMU_FLAG_LVD</code>	低电压状态标志
<code>PMU_FLAG_LDOVSR</code>	LDO电压选择就绪标志
<code>PMU_FLAG_HDR</code>	高驱动就绪标志
<code>PMU_FLAG_HDSR</code>	高驱动切换器就绪标志
<code>PMU_FLAG_LDR</code>	低驱动模式就绪标志
输出参数{out}	
-	-
返回值	
<code>FlagStatus</code>	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	<code>pmu_flag_clear</code>
函数原型	<code>void pmu_flag_clear(uint32_t flag);</code>
功能描述	清除标志位
先决条件	-
被调用函数	-
输入参数{in}	
<code>flag_clear</code>	标志位
<code>PMU_FLAG_RESET_WAKEUP</code>	清除唤醒标志

<i>PMU_FLAG_RESE</i> <i>T_STANDBY</i>	清除待机标志
输出参数{out}	
-	
返回值	
-	

例如:

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

```
pmu_flag_clear (PMU_FLAG_RESET_WAKEUP);
```

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

### 3.17.1. 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	控制寄存器0
RCU_CFG0	配置寄存器0
RCU_INT	中断寄存器
RCU_APB2RST	APB2复位寄存器
RCU_APB1RST	APB1复位寄存器
RCU_AHBEN	AHB使能寄存器
RCU_APB2EN	APB2使能寄存器
RCU_APB1EN	APB1使能寄存器
RCU_BDCTL	备份域控制寄存器
RCU_RSTSCK	复位源/时钟寄存器
RCU_AHBRST	AHB复位寄存器
RCU_CFG1	配置寄存器1
RCU_CFG2	配置寄存器2
RCU_CTL1	控制寄存器1
RCU_ADDCTL	附加时钟控制寄存器
RCU_ADDINT	附加时钟中断寄存器
RCU_ADDAPB1EN	APB1附加使能寄存器
RCU_ADDAPB1RST	APB1附加复位寄存器

寄存器名称	寄存器描述
RCU_VKEY	电源解锁寄存器
RCU_DSV	深度睡眠模式电压寄存器

### 3.17.2. 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	复位RCU
rcu_periph_clock_enable	使能外设时钟
rcu_periph_clock_disable	禁能外设时钟
rcu_periph_clock_sleep_enable	在睡眠模式下，使能外设时钟
rcu_periph_clock_sleep_disable	在睡眠模式下，禁能外设时钟
rcu_periph_reset_enable	外设时钟复位使能
rcu_periph_reset_disable	外设时钟复位除能
rcu_bkp_reset_enable	备份域时钟复位使能
rcu_bkp_reset_disable	备份域时钟复位除能
rcu_system_clock_source_config	配置选择系统时钟源
rcu_system_clock_source_get	获取系统时钟源选择状态
rcu_ahb_clock_config	配置AHB时钟预分频选择
rcu_apb1_clock_config	配置APB1时钟预分频选择
rcu_apb2_clock_config	配置APB2时钟预分频选择
rcu_adc_clock_config	配置ADC时钟预分频选择
rcu_usbfs_clock_config	配置USBFS预分频系数
rcu_ckout_config	配置CKOUT时钟源选择及分频系数
rcu_pll_preselection_config	配置主PLL时钟源预选项
rcu_pll_config	配置PLL时钟源及倍频因子
rcu_usart_clock_config	配置串口时钟
rcu_cec_clock_config	配置CEC时钟
rcu_rtc_clock_config	配置RTC时钟
rcu_ck48m_clock_config	配置IRC48M时钟
rcu_hxtal_prediv_config	配置HXTAL作为PLL输入源分频因子
rcu_lxtal_drive_capability_config	配置LXTAL的驱动力
rcu_flag_get	获取时钟稳定状态和外设复位标志
rcu_all_reset_flag_clear	清除复位标志
rcu_interrupt_flag_get	获取时钟中断和CKM中断标志
rcu_interrupt_flag_clear	清除中断标志
rcu_interrupt_enable	时钟稳定中断使能
rcu_interrupt_disable	时钟稳定中断除能

库函数名称	库函数描述
rcu_osc_i_stab_wait	等待振荡器稳定标志位置位
rcu_osc_i_on	打开振荡器
rcu_osc_i_off	关闭振荡器
rcu_osc_i_bypass_mode_enable	使能时钟旁路模式
rcu_osc_i_bypass_mode_disable	除能时钟旁路模式
rcu_hxtal_clock_monitor_enable	使能HXTAL时钟监视器
rcu_hxtal_clock_monitor_disable	禁能HXTAL时钟监视器
rcu_irc8m_adjust_value_set	设置内部8MHz RC振荡器时钟调整值
rcu_irc28m_adjust_value_set	设置内部28MHz RC振荡器时钟调整值
rcu_voltage_key_unlock	解锁电压锁定
rcu_deepsleep_voltage_set	设置深度睡眠模式内核电压值
rcu_clock_freq_get	获取系统、总线或外设时钟频率

### 枚举 rcu\_periph\_enum

表 3-333. 枚举类型 rcu\_periph\_enum

成员名称	功能描述
RCU_DMA	DMA时钟
RCU_CRC	CRC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_TSI	TSI时钟
RCU_CFGCMP	CGFCMP时钟
RCU_ADC	ADC时钟
RCU_TIMER0	TIMER0时钟
RCU_SPI0	SPI0时钟
RCU_USART0	USART0时钟
RCU_TIMER14	TIMER14时钟
RCU_TIMER15	TIMER15时钟
RCU_TIMER16	TIMER16时钟
RCU_TIMER1	TIMER1时钟
RCU_TIMER2	TIMER2时钟
RCU_TIMER13	TIMER13时钟
RCU_WWDGT	WWDGT时钟
RCU_SPI1	SPI1时钟
RCU_USART1	USART1时钟
RCU_I2C0	I2C0时钟
RCU_I2C1	I2C1时钟

RCU_PMU	PMU时钟
RCU_DAC	DAC时钟（仅对GD32F350 GD32F355 GD32F370适用）
RCU_CEC	CEC时钟（仅对GD32F350 GD32F355 GD32F370适用）
RCU_TIMER5	TIMER5时钟（仅对GD32F350 GD32F355 GD32F370适用）
RCU_USBFS	USBFS时钟（仅对GD32F350 GD32F355 GD32F370适用）
RCU_RTC	RTC时钟
RCU_CTC	CTC时钟

### 枚举 rcu\_periph\_sleep\_enum

表 3-334. 枚举类型 rcu\_periph\_sleep\_enum

成员名称	功能描述
RCU_SRAM_SLP	SRAM接口时钟
RCU_FMC_SLP	FMC时钟

### 枚举 rcu\_periph\_reset\_enum

表 3-335. 枚举类型 rcu\_periph\_reset\_enum

成员名称	功能描述
RCU_GPIOARST	复位GPIOA时钟
RCU_GPIOBRST	复位GPIOB时钟
RCU_GPIOCRST	复位GPIOC时钟
RCU_GPIODRST	复位GPIOD时钟
RCU_GPIOFRST	复位GPIOF时钟
RCU_TSIRST	复位TSI时钟
RCU_CFGCMRST	复位CFGCMP时钟
RCU_ADCRST	复位ADC时钟
RCU_TIMER0RST	复位TIMER0时钟
RCU_SPI0RST	复位SPI0时钟
RCU_USART0RST	复位USART0时钟
RCU_TIMER14RST	复位TIMER14时钟
RCU_TIMER15RST	复位TIMER15时钟
RCU_TIMER16RST	复位TIMER16时钟
RCU_TIMER1RST	复位TIMER1时钟
RCU_TIMER2RST	复位TIMER2时钟
RCU_TIMER13RST	复位TIMER13时钟
RCU_WWDGTRST	复位WWDGT时钟
RCU_SPI1RST	复位SPI1时钟
RCU_USART1RST	复位USART1时钟
RCU_I2C0RST	复位I2C0时钟
RCU_I2C1RST	复位I2C1时钟
RCU_PMURST	复位PMU时钟
RCU_DACRST	复位DAC时钟（仅对GD32F350 GD32F355 GD32F370适用）

RCU_CECRST	复位CEC时钟（仅对GD32F350 GD32F355 GD32F370适用）
RCU_TIMER5RST	复位TIMER5时钟（仅对GD32F350 GD32F355 GD32F370适用）
RCU_USBF5RST	复位USBFS时钟（仅对GD32F350 GD32F355 GD32F370适用）
RCU_CTCRST	复位CTC时钟

### 枚举 rcu\_flag\_enum

表 3-336. 枚举类型 rcu\_flag\_enum

成员名称	功能描述
RCU_FLAG_IRC40 KSTB	IRC40K振荡器稳定标志
RCU_FLAG_LXTAL STB	外部低速晶振稳定标志
RCU_FLAG_IRC8M STB	IRC8M振荡器稳定标志
RCU_FLAG_HXTAL STB	外部高速晶振稳定标志
RCU_FLAG_PLLST B	PLL稳定标志
RCU_FLAG_IRC28 MSTB	IRC28M振荡器稳定标志
RCU_FLAG_IRC48 MSTB	IRC48M振荡器稳定标志
RCU_FLAG_V12RS T	1.2V电源域复位标志
RCU_FLAG_OBLR ST	可选字节装载器复位标志
RCU_FLAG_EPRS T	外部引脚复位标志
RCU_FLAG_PORR ST	电源复位标志
RCU_FLAG_SWRS T	软件复位标志
RCU_FLAG_FWDG TRST	独立看门狗复位标志
RCU_FLAG_WWD GTRST	窗口看门狗复位标志
RCU_FLAG_LPRST	低功耗复位标志



## 枚举 rcu\_int\_flag\_enum

表 3-337. 枚举类型 rcu\_int\_flag\_enum

成员名称	功能描述
RCU_INT_FLAG_IR C40KSTB	IRC40K时钟稳定中断标志
RCU_INT_FLAG_L XTALSTB	外部低速晶振时钟稳定中断标志
RCU_INT_FLAG_IR C8MSTB	IRC8M时钟稳定中断标志
RCU_INT_FLAG_H XTALSTB	外部高速晶振时钟稳定中断标志
RCU_INT_FLAG_P LLSTB	PLL时钟稳定中断标志
RCU_INT_FLAG_IR C28MSTB	IRC28M时钟稳定中断标志
RCU_INT_FLAG_C KM	外部高速晶振时钟阻塞中断标志
RCU_INT_FLAG_IR C48MSTB	IRC48M时钟稳定中断标志

## 枚举 rcu\_int\_flag\_clear\_enum

表 3-338. 枚举类型 rcu\_int\_flag\_clear\_enum

成员名称	功能描述
RCU_INT_FLAG_IR C40KSTB_CLR	IRC40K时钟稳定中断清除标志
RCU_INT_FLAG_L XTALSTB_CLR	外部低速晶振时钟稳定中断清除标志
RCU_INT_FLAG_IR C8MSTB_CLR	IRC8M时钟稳定中断清除标志
RCU_INT_FLAG_H XTALSTB_CLR	外部高速晶振时钟稳定中断清除标志
RCU_INT_FLAG_P LLSTB_CLR	PLL时钟稳定中断清除标志
RCU_INT_FLAG_IR C28MSTB_CLR	IRC28M时钟稳定中断清除标志
RCU_INT_FLAG_C KM_CLR	外部高速晶振时钟阻塞中断清除标志
RCU_INT_FLAG_IR C48MSTB_CLR	IRC48M时钟稳定中断清除标志

## 枚举 rcu\_int\_enum

表 3-339. 枚举类型 rcu\_int\_enum

成员名称	功能描述
RCU_INT_IRC40KS TB	IRC40K时钟稳定中断
RCU_INT_LXTALS TB	外部低速晶振时钟稳定中断
RCU_INT_IRC8MS TB	IRC8M时钟稳定中断
RCU_INT_HXTALS TB	外部高速晶振时钟稳定中断
RCU_INT_PLLSTB	PLL时钟稳定中断
RCU_INT_IRC28M STB	IRC28M时钟稳定中断
RCU_INT_IRC48M STB	IRC48M时钟稳定中断

## 枚举 rcu\_adc\_clock\_enum

表 3-340. 枚举类型 rcu\_adc\_clock\_enum

成员名称	功能描述
RCU_ADCCCK_IRC2 8M_DIV2	选择IRC28M的2分频作为ADC的时钟源
RCU_ADCCCK_IRC2 8M	选择IRC28M作为ADC的时钟源
RCU_ADCCCK_APB 2_DIV2	选择APB的2分频作为ADC的时钟源
RCU_ADCCCK_AHB _DIV3	选择AHB的3分频作为ADC的时钟源
RCU_ADCCCK_APB 2_DIV4	选择APB的4分频作为ADC的时钟源
RCU_ADCCCK_AHB _DIV5	选择AHB的5分频作为ADC的时钟源
RCU_ADCCCK_APB 2_DIV6	选择APB的6分频作为ADC的时钟源
RCU_ADCCCK_AHB _DIV7	选择AHB的7分频作为ADC的时钟源
RCU_ADCCCK_APB 2_DIV8	选择APB的8分频作为ADC的时钟源
RCU_ADCCCK_AHB _DIV9	选择AHB的9分频作为ADC的时钟源

### 枚举 rcu\_osci\_type\_enum

表 3-341. 枚举类型 rcu\_osci\_type\_enum

成员名称	功能描述
RCU_HXTAL	外部高速振荡器
RCU_LXTAL	外部低速振荡器
RCU_IRC8M	IRC8M振荡器
RCU_IRC28M	IRC28M振荡器
RCU_IRC48M	IRC48M振荡器
RCU_IRC40K	IRC40K振荡器
RCU_PLL_CK	锁相环时钟

### 枚举 rcu\_clock\_freq\_enum

表 3-342. 枚举类型 rcu\_clock\_freq\_enum

成员名称	功能描述
CK_SYS	系统时钟
CK_AHB	AHB时钟
CK_APB1	APB1时钟
CK_APB2	APB2时钟
CK_ADC	ADC时钟
CK_CEC	CEC时钟
CK_USART	USART时钟

### 函数 rcu\_deinit

函数rcu\_deinit描述见下表：

表 3-343. 函数 rcu\_deinit

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

例如：

```
/* deinitialize the RCU */
```

```
rcu_deinit();
```

### 函数 rcu\_periph\_clock\_enable

函数rcu\_periph\_clock\_enable描述见下表:

表 3-344. 函数 rcu\_periph\_clock\_enable

函数名称	rcu_periph_clock_enable
函数原形	void rcu_periph_clock_enable(rcu_periph_enum periph);
功能描述	使能外设时钟
先决条件	-
被调用函数	-
输入参数{in}	
periph	RCU外设, 具体参考 <a href="#">表3-333. 枚举类型rcu_periph_enum</a> 。
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* enable the USART0 clock */  
rcu_periph_clock_enable(RCU_USART0);
```

### 函数 rcu\_periph\_clock\_disable

函数rcu\_periph\_clock\_disable描述见下表:

表 3-345. 函数 rcu\_periph\_clock\_disable

函数名称	rcu_periph_clock_disable
函数原形	void rcu_periph_clock_disable(rcu_periph_enum periph);
功能描述	禁能外设时钟
先决条件	-
被调用函数	-
输入参数{in}	
periph	RCU外设, 具体参考 <a href="#">表3-333. 枚举类型rcu_periph_enum</a> 。
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* disable the USART0 clock */  
rcu_periph_clock_disable(RCU_USART0);
```

### 函数 rcu\_periph\_clock\_sleep\_enable

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

**表 3-346. 函数 rcu\_periph\_clock\_sleep\_enable**

函数名称	rcu_periph_clock_sleep_enable
函数原形	void rcu_periph_clock_sleep_enable(rcu_periph_sleep_enum periph);
功能描述	在睡眠模式下，使能外设时钟
先决条件	-
被调用函数	-
输入参数{in}	
periph	RCU外设，具体参考 <a href="#">表3-334. 枚举类型rcu_periph_sleep_enum</a> 。
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* enable the FMC clock when in sleep mode */
rcu_periph_clock_sleep_enable(RCU_FMC_SLP);
```

### 函数 rcu\_periph\_clock\_sleep\_disable

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

**表 3-347. 函数 rcu\_periph\_clock\_sleep\_disable**

函数名称	rcu_periph_clock_sleep_disable
函数原形	void rcu_periph_clock_sleep_disable(rcu_periph_sleep_enum periph);
功能描述	在睡眠模式下，禁能外设时钟
先决条件	-
被调用函数	-
输入参数{in}	
periph	RCU外设，具体参考 <a href="#">表3-334. 枚举类型rcu_periph_sleep_enum</a> 。
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* disable the FMC clock when in sleep mode */
rcu_periph_clock_sleep_disable(RCU_FMC_SLP);
```

### 函数 rcu\_periph\_reset\_enable

函数rcu\_periph\_reset\_enable描述见下表:

表 3-348. 函数 rcu\_periph\_reset\_enable

函数名称	rcu_periph_reset_enable
函数原形	void rcu_periph_reset_enable(rcu_periph_reset_enum periph_reset);
功能描述	使能外设复位
先决条件	-
被调用函数	-
输入参数{in}	
periph_reset	RCU外设复位, 具体参考 <a href="#">表3-335. 枚举类型rcu_periph_reset_enum</a> 。
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* enable SPI0 reset */  
  
rcu_periph_reset_enable(RCU_SPI0RST);
```

### 函数 rcu\_periph\_reset\_disable

函数rcu\_periph\_reset\_disable描述见下表:

表 3-349. 函数 rcu\_periph\_reset\_disable

函数名称	rcu_periph_reset_disable
函数原形	void rcu_periph_reset_disable(rcu_periph_reset_enum periph_reset);
功能描述	禁能外设复位
先决条件	-
被调用函数	-
输入参数{in}	
periph_reset	RCU外设复位, 具体参考 <a href="#">表3-335. 枚举类型rcu_periph_reset_enum</a> 。
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* disable SPI0 reset */  
  
rcu_periph_reset_disable(RCU_SPI0RST);
```

### 函数 rcu\_bkp\_reset\_enable

函数rcu\_bkp\_reset\_enable描述见下表：

**表 3-350. 函数 rcu\_bkp\_reset\_enable**

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

例如：

```
/* reset the BKP domain */
rcu_bkp_reset_enable();
```

### 函数 rcu\_bkp\_reset\_disable

函数rcu\_bkp\_reset\_disable描述见下表：

**表 3-351. 函数 rcu\_bkp\_reset\_disable**

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

例如：

```
/* disable the BKP domain reset */
rcu_bkp_reset_disable();
```

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

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

**表 3-352. 函数 rcu\_system\_clock\_source\_config**

函数名称	rcu_system_clock_source_config
函数原形	void rcu_system_clock_source_config(uint32_t ck_sys);
功能描述	配置选择系统时钟源
先决条件	-
被调用函数	-
输入参数{in}	
ck_sys	系统时钟源选择
RCU_CKSYSSRC_I RC8M	选择CK_IRC8M时钟作为CK_SYS时钟源
RCU_CKSYSSRC_ HXTAL	选择CK_HXTAL时钟作为CK_SYS时钟源
RCU_CKSYSSRC_ PLL	选择CK_PLL时钟作为CK_SYS时钟源
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* configure the CK_HXTAL as the CK_SYS source */
```

```
rcu_system_clock_source_config(RCU_CKSYSSRC_HXTAL);
```

**函数 rcu\_system\_clock\_source\_get**

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

**表 3-353. 函数 rcu\_system\_clock\_source\_get**

函数名称	rcu_system_clock_source_get
函数原形	uint32_t rcu_system_clock_source_get(void);
功能描述	获取系统时钟源选择状态
先决条件	-
被调用函数	-
输入参数{in}	
-	-
输出参数{out}	
-	-
返回值	
uint32_t	RCU_SCSS_IRC8M/RCU_SCSS_HXTAL/RCU_SCSS_PLL



例如：

```
uint32_t temp_cksys_status;

/* get the CK_SYS source */

temp_cksys_status = rcu_system_clock_source_get();
```

### 函数 rcu\_ahb\_clock\_config

函数rcu\_ahb\_clock\_config描述见下表：

表 3-354. 函数 rcu\_ahb\_clock\_config

函数名称	rcu_ahb_clock_config
函数原形	void rcu_ahb_clock_config(uint32_t ck_ahb);
功能描述	配置AHB时钟预分频选择
先决条件	-
被调用函数	-
输入参数{in}	
ck_ahb	AHB预分频选择
RCU_AHB_CKSYS_DIVx	选择CK_SYS时钟x分频（x=1, 2, 4, 8, 16, 64, 128, 256, 512）
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* configure CK_SYS/128 */

rcu_ahb_clock_config(RCU_AHB_CKSYS_DIV128);
```

### 函数 rcu\_apb1\_clock\_config

函数rcu\_apb1\_clock\_config描述见下表：

表 3-355. 函数 rcu\_apb1\_clock\_config

函数名称	rcu_apb1_clock_config
函数原形	void rcu_apb1_clock_config(uint32_t ck_apb1);
功能描述	配置APB1时钟预分频选择
先决条件	-
被调用函数	-
输入参数{in}	
ck_apb1	APB1预分频选择
RCU_APB1_CKAHB_DIVx	选择CK_AHB时钟x分频作为CK_APB1时钟（x=1,2,4,8,16）

输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* configure CK_AHB/16 as CK_APB1 */
rcu_apb1_clock_config(RCU_APB1_CKAHB_DIV16);
```

### 函数 rcu\_apb2\_clock\_config

函数rcu\_apb2\_clock\_config描述见下表：

**表 3-356. 函数 rcu\_apb2\_clock\_config**

函数名称	rcu_apb2_clock_config
函数原形	void rcu_apb2_clock_config(uint32_t ck_apb2);
功能描述	配置APB2时钟预分频选择
先决条件	-
被调用函数	-
输入参数{in}	
<b>ck_apb2</b>	APB2预分频选择
<i>RCU_APB2_CKAHB_DIVx</i>	选择CK_AHB时钟x分频作为CK_APB2时钟（x=1,2,4,8,16）
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* configure CK_AHB/8 as CK_APB2 */
rcu_apb2_clock_config(RCU_APB2_CKAHB_DIV8);
```

### 函数 rcu\_adc\_clock\_config

函数rcu\_adc\_clock\_config描述见下表：

**表 3-357. 函数 rcu\_adc\_clock\_config**

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

<b>ck_adc</b>	ADC预分频选择，具体参考 <a href="#">表3-340. 枚举类型rcu_adc_clock_enum</a> 。
<b>输出参数{out}</b>	
-	-
<b>返回值</b>	
-	-

例如：

```
/* select CK_IRC28M as CK_ADC */
rcu_adc_clock_config(RCU_ADCCCK_IRC28M);
```

### 函数 rcu\_usbfs\_clock\_config

函数rcu\_usbfs\_clock\_config描述见下表：

**表 3-358. 函数 rcu\_usbfs\_clock\_config**

<b>函数名称</b>	rcu_usbfs_clock_config
<b>函数原形</b>	void rcu_usbfs_clock_config(uint32_t ck_usbfs);
<b>功能描述</b>	配置USBFS预分频系数
<b>先决条件</b>	-
<b>被调用函数</b>	-
<b>输入参数{in}</b>	
<b>ck_usbfs</b>	USBFS时钟预分频系数
<i>RCU_USBFS_CKP_LL_DIV1</i>	选择（CK_PLL）作为USBFS时钟
<i>RCU_USBFS_CKP_LL_DIV1_5</i>	选择（CK_PLL / 1.5）作为USBFS时钟
<i>RCU_USBFS_CKP_LL_DIV2</i>	选择（CK_PLL / 2）作为USBFS时钟
<i>RCU_USBFS_CKP_LL_DIV2_5</i>	选择（CK_PLL / 2.5）作为USBFS时钟
<i>RCU_USBFS_CKP_LL_DIV3</i>	选择（CK_PLL / 3）作为USBFS时钟
<i>RCU_USBFS_CKP_LL_DIV3_5</i>	选择（CK_PLL / 3.5）作为USBFS时钟
<b>输出参数{out}</b>	
-	-
<b>返回值</b>	
-	-

例如：

```
/* configure the CK_PLL as CK_USBFS */
rcu_usbfs_clock_config(RCU_USBFS_CKPLL_DIV1);
```

## 函数 rcu\_ckout\_config

函数rcu\_ckout\_config描述见下表：

表 3-359. 函数 rcu\_ckout\_config

函数名称	rcu_ckout_config
函数原形	void rcu_ckout_config(uint32_t ckout_src, uint32_t ckout_div);
功能描述	配置CKOUT时钟源选择及分频系数
先决条件	-
被调用函数	-
输入参数{in}	
ckout_src	CKOUT时钟源选择
RCU_CKOUTSRC_NONE	无时钟输出
RCU_CKOUTSRC_IRC28M	选择内部28M RC振荡器时钟
RCU_CKOUTSRC_IRC40K	选择内部40K RC振荡器时钟
RCU_CKOUTSRC_LXTAL	选择外部低速晶体振荡器时钟（LXTAL）
RCU_CKOUTSRC_CKSYS	选择系统时钟CK_SYS
RCU_CKOUTSRC_IRC8M	选择内部8M RC振荡器时钟
RCU_CKOUTSRC_HXTAL	选择外部高速晶体振荡器时钟（HXTAL）
RCU_CKOUTSRC_CKPLL_DIV1	选择CK_PLL时钟
RCU_CKOUTSRC_CKPLL_DIV2	选择（CK_PLL / 2）时钟
输入参数{in}	
ckout_div	CKOUT分频系数
RCU_CKOUT_DIVx	将CKOUT所选时钟x分频（x=1,2,4,8,16,32,64,128）
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* configure the HXTAL as CK_OUT clock source */
```

```
rcu_ckout_config(RCU_CKOUTSRC_HXTAL);
```

**函数 rcu\_pll\_preselection\_config**

函数rcu\_pll\_preselection\_config描述见下表:

**表 3-360. 函数 rcu\_pll\_preselection\_config**

函数名称	rcu_pll_preselection_config
函数原形	void rcu_pll_preselection_config(uint32_t pll_presel);
功能描述	配置主PLL时钟源预选项
先决条件	-
被调用函数	-
输入参数{in}	
pll_presel	PLL时钟源预选项选择
RCU_PLLPRESEL_IRC48M	预选IRC48M作为PLL时钟的时钟源
RCU_PLLPRESEL_HXTAL	预选HXTAL作为PLL时钟的时钟源
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* select HXTAL as PLL preselection clock */
```

```
rcu_pll_preselection_config(RCU_PLLPRESEL_HXTAL);
```

**函数 rcu\_pll\_config**

函数rcu\_pll\_config描述见下表:

**表 3-361. 函数 rcu\_pll\_config**

函数名称	rcu_pll_config
函数原形	void rcu_pll_config(uint32_t pll_src, uint32_t pll_mul);
功能描述	配置主PLL时钟及倍频因子
先决条件	-
被调用函数	-
输入参数{in}	
pll_src	PLL时钟源选择
RCU_PLLSRC_IRC8M_DIV2	(IRC8M / 2)被选择为PLL时钟的时钟源
RCU_PLLSRC_HXTAL_IRC48M	选择HXTAL时钟或IRC48M时钟作为PLL时钟的时钟源
输入参数{in}	
pll_mul	PLL时钟倍频因子

<i>RCU_PLL_MULx</i>	PLL源时钟 * x (x = 2..64)
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* configure the PLL */
```

```
rcu_pll_config(RCU_PLLSRC_HXTAL_IRC48M, RCU_PLL_MUL10);
```

### 函数 **rcu\_usart\_clock\_config**

函数rcu\_usart\_clock\_config描述见下表:

**表 3-362. 函数 rcu\_usart\_clock\_config**

函数名称	rcu_usart_clock_config
函数原形	void rcu_usart_clock_config(uint32_t ck_usart);
功能描述	配置串口时钟
先决条件	-
被调用函数	-
输入参数{in}	
<b>ck_usart</b>	USART0输入时钟源
<i>RCU_USART0SRC</i> <i>_CKAPB2</i>	选择CK_APB2时钟作为CK_USART0时钟
<i>RCU_USART0SRC</i> <i>_CKSYS</i>	选择CK_SYS时钟作为CK_USART0时钟
<i>RCU_USART0SRC</i> <i>_LXTAL</i>	选择CK_LXTAL时钟作为CK_USART0时钟
<i>RCU_USART0SRC</i> <i>_IRC8M</i>	选择CK_IRC8M时钟作为CK_USART0时钟
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* configure the USART */
```

```
rcu_usart_clock_config(RCU_USART0SRC_CKAPB2);
```

### 函数 **rcu\_cec\_clock\_config**

函数rcu\_cec\_clock\_config描述见下表:

表 3-363. 函数 rcu\_cec\_clock\_config

函数名称	rcu_cec_clock_config
函数原形	void rcu_cec_clock_config(uint32_t ck_cec);
功能描述	配置CEC时钟
先决条件	-
被调用函数	-
输入参数{in}	
ck_cec	CEC时钟源选择
RCU_CECSRC_IRC8M_DIV244	选择CK_IRC8M / 244作为CEC时钟源
RCU_CECSRC_LXTAL	选择CK_LXTAL作为CEC时钟源
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* configure the CEC clock source selection */
rcu_cec_clock_config(RCU_CECSRC_LXTAL);
```

### 函数 rcu\_rtc\_clock\_config

函数rcu\_rtc\_clock\_config描述见下表:

表 3-364. 函数 rcu\_rtc\_clock\_config

函数名称	rcu_rtc_clock_config
函数原形	void rcu_rtc_clock_config(uint32_t rtc_clock_source);
功能描述	配置RTC时钟
先决条件	-
被调用函数	-
输入参数{in}	
rtc_clock_source	RTC时钟源选择
RCU_RTCSRC_NONE	未选择时钟
RCU_RTCSRC_LXTAL	选择CK_LXTAL作为RTC时钟源
RCU_RTCSRC_IRC40K	选择内部40K RC振荡器时钟作为RTC时钟源
RCU_RTCSRC_HXTAL_DIV32	选择外部高速晶振32分频作为RTC时钟源
输出参数{out}	
-	-

返回值	
-	-

例如：

```
/* configure the RTC clock source selection */
```

```
rcu_rtc_clock_config(RCU_RTCSRC_IRC40K);
```

### 函数 rcu\_ck48m\_clock\_config

函数rcu\_ck48m\_clock\_config描述见下表：

表 3-365. 函数 rcu\_ck48m\_clock\_config

函数名称	rcu_ck48m_clock_config
函数原形	void rcu_ck48m_clock_config(uint32_t ck48m_clock_source);
功能描述	配置CK48M时钟源
先决条件	-
被调用函数	-
输入参数{in}	
ck48m_clock_source	CK48M时钟源选择
RCU_CK48MSRC_PLL48M	选择CK_PLL48M作为CK48M时钟源
RCU_CK48MSRC_IRC48M	选择CK_IRC48M作为CK48M时钟源
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* configure the CK_48M clock source selection */
```

```
rcu_ck48m_clock_config (RCU_CK48MSRC_IRC48M);
```

### 函数 rcu\_hxtal\_prediv\_config

函数rcu\_hxtal\_prediv\_config描述见下表：

表 3-366. 函数 rcu\_hxtal\_prediv\_config

函数名称	rcu_hxtal_prediv_config
函数原形	void rcu_hxtal_prediv_config(uint32_t hxtal_prediv);
功能描述	配置HXTAL作为PLL输入源分频因子
先决条件	-
被调用函数	-



输入参数{in}	
<b>hxtal_prediv</b>	PLL时钟源分频因子选择
<i>RCU_PLL_PREDVx</i>	HXTAL或TRC48M的x分频作为PLL时钟（x=1..16）
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* configure the PLL clock source selection */
```

```
rcu_hxtal_prediv_config(RCU_PLL_PREDIV2);
```

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

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

**表 3-367. 函数 rcu\_lxtal\_drive\_capability\_config**

函数名称	rcu_lxtal_drive_capability_config
函数原形	void rcu_lxtal_drive_capability_config(uint32_t lxtal_dricap);
功能描述	配置LXTAL驱动能力
先决条件	-
被调用函数	-
输入参数{in}	
<b>lxtal_dricap</b>	LXTAL驱动能力
<i>RCU_LXTAL_LOW_DRI</i>	低驱动力
<i>RCU_LXTAL_MED_LOWDRI</i>	中低驱动力
<i>RCU_LXTAL_MED_HIGHDRI</i>	中高驱动力
<i>RCU_LXTAL_HIGH_DRI</i>	高驱动力
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* configure the LXTAL drive capability */
```

```
rcu_lxtal_drive_capability_config(RCU_LAXTAL_LOWDRI);
```

## 函数 rcu\_flag\_get

函数rcu\_flag\_get描述见下表:

表 3-368. 函数 rcu\_flag\_get

函数名称	rcu_flag_get
函数原形	FlagStatus rcu_flag_get (rcu_flag_enum flag);
功能描述	获取时钟稳定和外设复位标志
先决条件	-
被调用函数	-
输入参数{in}	
flag	时钟稳定和外设复位标志，参考 <a href="#">表3-336. 枚举类型rcu_flag_enum</a> 。
输出参数{out}	
-	-
返回值	
-	SET或RESET

例如:

```
/* get the clock stabilization flag */
if(RESET != rcu_flag_get(RCU_FLAG_LXTALSTB)){
}
```

## 函数 rcu\_all\_reset\_flag\_clear

函数rcu\_all\_reset\_flag\_clear描述见下表:

表 3-369. 函数 rcu\_all\_reset\_flag\_clear

函数名称	rcu_all_reset_flag_clear
函数原形	void rcu_all_reset_flag_clear(void);
功能描述	清除所有复位标志位
先决条件	-
被调用函数	-
输入参数{in}	
-	-
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* clear all the reset flag */
rcu_all_reset_flag_clear();
```

## 函数 rcu\_interrupt\_flag\_get

函数rcu\_interrupt\_flag\_get描述见下表：

表 3-370. 函数 rcu\_interrupt\_flag\_get

函数名称	rcu_interrupt_flag_get
函数原形	FlagStatus rcu_interrupt_flag_get(rcu_int_flag_enum int_flag);
功能描述	获取时钟稳定中断和时钟阻塞中断标志
先决条件	-
被调用函数	-
输入参数{in}	
Int_flag	中断以及CKM标志，参考 <a href="#">表3-337. 枚举类型rcu_int_flag_enum</a> 。
输出参数{out}	
-	-
返回值	
FlagStatus	SET 或 RESET

例如：

```
/* get the clock stabilization interrupt flag */
if(SET == rcu_interrupt_flag_get(RCU_INT_FLAG_HXTALSTB)){
}
```

## 函数 rcu\_interrupt\_flag\_clear

函数rcu\_interrupt\_flag\_clear描述见下表：

表 3-371. 函数 rcu\_interrupt\_flag\_clear

函数名称	rcu_interrupt_flag_clear
函数原形	void rcu_interrupt_flag_clear (rcu_int_flag_clear_enum int_flag_clear);
功能描述	清除中断标志和时钟阻塞中断标志
先决条件	-
被调用函数	-
输入参数{in}	
int_flag_clear	时钟稳定和阻塞中断标志清除，参考 <a href="#">表3-338. 枚举类型rcu_int_flag_clear_enum</a> 。
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* clear the interrupt HXTAL stabilization interrupt flag */
```

```
rcu_interrupt_flag_clear(RCU_INT_FLAG_HXTALSTB_CLR);
```

### 函数 rcu\_interrupt\_enable

函数rcu\_interrupt\_enable描述见下表：

**表 3-372. 函数 rcu\_interrupt\_enable**

函数名称	rcu_interrupt_enable
函数原形	void rcu_interrupt_enable (rcu_int_enum stab_int);
功能描述	使能时钟稳定中断
先决条件	-
被调用函数	-
输入参数{in}	
stab_int	时钟稳定中断，具体参考 <a href="#">表3-339. 枚举类型rcu_int_enum</a> 。
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* enable the HXTAL stabilization interrupt */
rcu_interrupt_enable(RCU_INT_HXTALSTB);
```

### 函数 rcu\_interrupt\_disable

函数rcu\_interrupt\_disable描述见下表：

**表 3-373. 函数 rcu\_interrupt\_disable**

函数名称	rcu_interrupt_disable
函数原形	void rcu_interrupt_disable (rcu_int_enum stab_int);
功能描述	除能时钟稳定中断
先决条件	-
被调用函数	-
输入参数{in}	
stab_int	时钟稳定中断，具体参考 <a href="#">表3-339. 枚举类型rcu_int_enum</a> 。
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* disable the HXTAL stabilization interrupt */
rcu_interrupt_disable(RCU_INT_HXTALSTB);
```

**函数 rcu\_osc\_i\_stab\_wait**

函数rcu\_osc\_i\_stab\_wait描述见下表：

**表 3-374. 函数 rcu\_osc\_i\_stab\_wait**

函数名称	rcu_osc_i_stab_wait
函数原形	ErrStatus rcu_osc_i_stab_wait(rcu_osc_i_type_enum osci);
功能描述	等待振荡器稳定标志位置位或振荡器起振超时
先决条件	-
被调用函数	rcu_flag_get
输入参数{in}	
osci	振荡器类型，具体参考 <a href="#">表3-341. 枚举类型rcu_osc_i_type_enum</a> 。
输出参数{out}	
-	-
返回值	
ErrStatus	SUCCESS 或 ERROR

例如：

```
/* wait for oscillator stabilization flag */
if(SUCCESS == rcu_osc_i_stab_wait(RCU_HXTAL)){
}
```

**函数 rcu\_osc\_i\_on**

函数rcu\_osc\_i\_on描述见下表：

**表 3-375. 函数 rcu\_osc\_i\_on**

函数名称	rcu_osc_i_on
函数原形	void rcu_osc_i_on(rcu_osc_i_type_enum osci);
功能描述	打开振荡器
先决条件	-
被调用函数	-
输入参数{in}	
osci	振荡器类型，具体参考 <a href="#">表3-341. 枚举类型rcu_osc_i_type_enum</a> 。
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* turn on the high speed crystal oscillator */
rcu_osc_i_on(RCU_HXTAL);
```

## 函数 rcu\_osci\_off

函数rcu\_osci\_off描述见下表：

表 3-376. 函数 rcu\_osci\_off

函数名称	rcu_osci_off
函数原形	void rcu_osci_off(rcu_osci_type_enum osci);
功能描述	关闭振荡器
先决条件	-
被调用函数	-
输入参数{in}	
osci	振荡器类型，具体参考 <a href="#">表3-341. 枚举类型rcu_osci_type_enum。</a>
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* turn off the high speed crystal oscillator */
```

```
rcu_osci_off(RCU_HXTAL);
```

## 函数 rcu\_osci\_bypass\_mode\_enable

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

表 3-377. 函数 rcu\_osci\_bypass\_mode\_enable

函数名称	rcu_osci_bypass_mode_enable
函数原形	void rcu_osci_bypass_mode_enable(rcu_osci_type_enum osci);
功能描述	使能振荡器时钟旁路模式
先决条件	HXTALEN或LXTALEN应在使能振荡器时钟旁路模式前先复位
被调用函数	-
输入参数{in}	
osci	振荡器类型，具体参考 <a href="#">表3-341. 枚举类型rcu_osci_type_enum。</a>
输出参数{out}	
-	-
返回值	
-	-

例如：

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

```
rcu_osci_bypass_mode_enable(RCU_HXTAL);
```

### 函数 rcu\_osc\_bypass\_mode\_disable

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

**表 3-378. 函数 rcu\_osc\_bypass\_mode\_disable**

函数名称	rcu_osc_bypass_mode_disable
函数原形	void rcu_osc_bypass_mode_disable(rcu_osc_type_enum osci);
功能描述	除能振荡器时钟旁路模式
先决条件	HXTALEN或LXTALEN应在使能振荡器时钟旁路模式前先复位
被调用函数	-
输入参数{in}	
osci	振荡器类型，具体参考 <a href="#">表3-341. 枚举类型rcu_osc_type_enum</a> 。
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* disable the HXTAL clock monitor */
```

```
rcu_hxtal_clock_monitor_disable();
```

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

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

**表 3-379. 函数 rcu\_hxtal\_clock\_monitor\_enable**

函数名称	rcu_hxtal_clock_monitor_enable
函数原形	void rcu_hxtal_clock_monitor_enable(void);
功能描述	使能HXTAL时钟监视器
先决条件	-
被调用函数	-
输入参数{in}	
-	-
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* enable the HXTAL clock monitor */
```

```
rcu_hxtal_clock_monitor_enable();
```

**函数 rcu\_hxtal\_clock\_monitor\_disable**

函数rcu\_hxtal\_clock\_monitor\_disable描述见下表：

**表 3-380. 函数 rcu\_hxtal\_clock\_monitor\_disable**

函数名称	rcu_hxtal_clock_monitor_disable
函数原形	void rcu_hxtal_clock_monitor_disable(void);
功能描述	除能HXTAL时钟监视器
先决条件	-
被调用函数	-
输入参数{in}	
-	-
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* disable the HXTAL clock monitor */
```

```
rcu_hxtal_clock_monitor_disable();
```

**函数 rcu\_irc8m\_adjust\_value\_set**

函数rcu\_irc8m\_adjust\_value\_set描述见下表：

**表 3-381. 函数 rcu\_irc8m\_adjust\_value\_set**

函数名称	rcu_irc8m_adjust_value_set
函数原形	void rcu_irc8m_adjust_value_set(uint32_t irc8m_adjval);
功能描述	设置内部8MHz RC振荡器时钟调整值
先决条件	-
被调用函数	-
输入参数{in}	
irc8m_adjval	IRC8M调整值（0到0x1F之间）
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* set the IRC8M adjust value */
```

```
rcu_irc8m_adjust_value_set(0x10);
```



**函数 rcu\_irc28m\_adjust\_value\_set**

函数rcu\_irc28m\_adjust\_value\_set描述见下表:

**表 3-382. 函数 rcu\_irc28m\_adjust\_value\_set**

函数名称	rcu_irc28m_adjust_value_set
函数原形	void rcu_irc28m_adjust_value_set(uint32_t irc28m_adjval);
功能描述	设置内部28MHz RC振荡器时钟调整值
先决条件	-
被调用函数	-
输入参数{in}	
irc8m_adjval	IRC28M调整值（0到0x1F之间）
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* set the IRC28M adjust value */
rcu_irc28m_adjust_value_set(0x10);
```

**函数 rcu\_voltage\_key\_unlock**

函数rcu\_voltage\_key\_unlock描述见下表:

**表 3-383. 函数 rcu\_voltage\_key\_unlock**

函数名称	rcu_voltage_key_unlock
函数原形	void rcu_voltage_key_unlock(void);
功能描述	解锁电压寄存器
先决条件	-
被调用函数	-
输入参数{in}	
-	-
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* unlock the voltage key*/
rcu_voltage_key_unlock();
```

**函数 rcu\_deepsleep\_voltage\_set**

函数rcu\_deepsleep\_voltage\_set描述见下表:

**表 3-384. 函数 rcu\_deepsleep\_voltage\_set**

函数名称	rcu_deepsleep_voltage_set
函数原形	void rcu_deepsleep_voltage_set(uint32_t dsvol);
功能描述	设置深度睡眠模式电压值
先决条件	-
被调用函数	-
输入参数{in}	
dsvol	深度睡眠模式电压值
RCU_DEEPSLEEP_V_0	在深度睡眠模式下内核电压为缺省值
RCU_DEEPSLEEP_V_1	在深度睡眠模式下内核电压为（缺省值-0.1）V（不推荐客户使用）
RCU_DEEPSLEEP_V_2	在深度睡眠模式下内核电压为（缺省值-0.2）V（不推荐客户使用）
RCU_DEEPSLEEP_V_3	在深度睡眠模式下内核电压为（缺省值-0.3）V（不推荐客户使用）
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* set the deep-sleep mode voltage */
rcu_deepsleep_voltage_set(RCU_DEEPSLEEP_V_0);
```

**函数 rcu\_clock\_freq\_get**

函数rcu\_clock\_freq\_get描述见下表:

**表 3-385. 函数 rcu\_clock\_freq\_get**

函数名称	rcu_clock_freq_get
函数原形	uint32_t rcu_clock_freq_get(rcu_clock_freq_enum clock);
功能描述	获取系统、总线以及外设时钟频率
先决条件	-
被调用函数	-
输入参数{in}	
clock	要获取的时钟频率，具体参考 <a href="#">表3-342. 枚举类型rcu_clock_freq_enum</a>
输出参数{out}	
-	-

返回值	
uint32_t	系统时钟/AHB时钟/APB1时钟/APB2时钟/ADC时钟/USART时钟频率

例如:

```
uint32_t temp_freq;

/* get the system clock frequency */

temp_freq = rcu_clock_freq_get(CK_SYS);
```

## 3.18. 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[3.18.1](#)描述了RTC的寄存器列表，章节[3.18.2](#)对RTC库函数进行说明。

### 3.18.1. 外设寄存器描述

RTC寄存器列表如下表所示:

表 3-386. RTC 寄存器

寄存器名称	寄存器描述
RTC_TIME	RT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
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_TAMP	RTC侵入寄存器
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
RTC_BKP4	RTC备份域寄存器4

### 3.18.2. 外设库函数描述

RTC库函数列表如下表所示：

**表 3-387. RTC 库函数**

库函数名称	库函数描述
rtc_deinit	复位大多数RTC寄存器
rtc_init	初始化RTC寄存器
rtc_init_mode_enter	进入RTC初始化模式
rtc_init_mode_exit	退出RTC初始化模式
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_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_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_alarm_enable	使能RTC闹钟
rtc_alarm_disable	失能RTC闹钟
rtc_timestamp_enable	使能RTC时间戳
rtc_timestamp_disable	失能RTC时间戳
rtc_timestamp_get	获取RTC时间戳时间和日期
rtc_timestamp_subsecond_get	获取RTC时间戳亚秒值
rtc_tamper_enable	使能RTC侵入检测
rtc_tamper_disable	失能RTC侵入检测
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
rtc_alter_output_config	配置RTC备用输出源
rtc_calibration_config	配置RTC校准寄存器
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_bypass_shadow_enable	使能RTC影子寄存器
rtc_bypass_shadow_disable	失能RTC影子寄存器
rtc_refclock_detection_enable	使能RTC参考时钟检测功能
rtc_refclock_detection_disable	失能RTC参考时钟检测功能

## 结构体 `rtc_parameter_struct`

表 3-388. 结构体类型 `rtc_parameter_struct`

Member name	Function description
<code>rtc_year</code>	RTC年份值: 0x0 - 0x99 (BCD格式)
<code>rtc_month</code>	RTC月份值
<code>rtc_date</code>	RTC日期值: 0x1 - 0x31 (BCD格式)
<code>rtc_day_of_week</code>	RTC星期值
<code>rtc_hour</code>	RTC小时值
<code>rtc_minute</code>	RTC分钟值: 0x0 - 0x59 (BCD格式)
<code>rtc_second</code>	RTC秒值: 0x0 - 0x59 (BCD格式)
<code>rtc_factor_asyn</code>	RTC一步分频值: 0x0 - 0x7F
<code>rtc_factor_syn</code>	RTC同步分频值: 0x0 - 0x7FFF
<code>rtc_am_pm</code>	RTC AM / PM值
<code>rtc_display_format</code>	RTC时间格式

## 结构体 `rtc_alarm_struct`

表 3-389. 结构体类型 `rtc_alarm_struct`

Member name	Function description
<code>rtc_alarm_mask</code>	RTC闹钟屏蔽
<code>rtc_weekday_or_date</code>	指定RTC闹钟是日期还是星期几
<code>rtc_alarm_day</code>	RTC闹钟日期或者星期几的值
<code>rtc_alarm_hour</code>	RTC闹钟小时值
<code>rtc_alarm_minute</code>	RTC闹钟分钟值: 0x0 - 0x59 (BCD格式)
<code>rtc_alarm_second</code>	RTC闹钟秒数值: 0x0 - 0x59 (BCD格式)
<code>rtc_am_pm</code>	RTC闹钟AM / PM数值

## 结构体 `rtc_timestamp_struct`

表 3-390. 结构体 `rtc_timestamp_struct`

Member name	Function description
<code>rtc_timestamp_month</code>	RTC时间戳月份值
<code>rtc_timestamp_date</code>	RTC 时间戳日期值: 0x1 - 0x31 (BCD格式)
<code>rtc_timestamp_day</code>	RTC时间戳星期值
<code>rtc_timestamp_hour</code>	RTC 时间戳小时值
<code>rtc_timestamp_minute</code>	RTC时间戳分钟值: 0x0 - 0x59 (BCD格式)
<code>rtc_timestamp_second</code>	RTC时间戳秒数值: 0x0 - 0x59 (BCD格式)
<code>rtc_am_pm</code>	RTC时间戳AM/PM数值

## 结构体 rtc\_tamper\_struct

表 3-391. 结构体 rtc\_tamper\_struct

Member name	Function description
rtc_tamper_source	RTC侵入检测源
rtc_tamper_trigger	RTC侵入事件检测触发沿
rtc_tamper_filter	RTC侵入事件检测在电平检测期间需要的连续采样次数
rtc_tamper_sample_frequency	RTC侵入事件电平模式检测的采样频率
rtc_tamper_precharge_enable	RTC在电压电平检测期间的预充电功能
rtc_tamper_precharge_time	RTC侵入事件电平检测采样预充电时间，如果预充电功能使能
rtc_tamper_with_timestamp	RTC侵入事件触发时间戳

## 函数 rtc\_deinit

函数rtc\_deinit描述见下表:

表 3-392. 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	-
输入参数{in}	
-	-
输出参数{out}	
-	-
返回值	
ErrStatus	ERROR或SUCCESS

例如:

```
ErrStatus error_status;

/* reset most of the RTC registers */

ErrStatus error_status = rtc_deinit ( );
```

## 函数 rtc\_init

函数rtc\_init描述见下表:

表 3-393. 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	-
输入参数{in}	
rtc_initpara_struct	初始化结构体，结构体成员参考 <a href="#">表3-388. 结构体类型rtc_parameter_struct</a>
输出参数{out}	
-	-
返回值	
ErrStatus	ERROR或SUCCESS

例如:

```
rtc_parameter_struct rtc_initpara_struct;

ErrStatus error_status;

/* initialize RTC registers */

ErrStatus error_status = rtc_init (& rtc_initpara_struct);
```

### 函数 rtc\_init\_mode\_enter

函数rtc\_init\_mode\_enter描述见下表:

**表 3-394. 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	-
输入参数{in}	
-	-
输出参数{out}	
-	-
返回值	
ErrStatus	ERROR或SUCCESS

例如:

```
ErrStatus error_status

/* enter RTC init mode */

ErrStatus error_status = rtc_init_mode_enter ( );
```

### 函数 rtc\_init\_mode\_exit

函数rtc\_init\_mode\_exit描述见下表:

表 3-395. 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	-
输入参数{in}	
-	-
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* exit RTC init mode */
```

```
rtc_init_mode_exit ( );
```

### 函数 rtc\_register\_sync\_wait

函数rtc\_register\_sync\_wait描述见下表:

表 3-396. 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, 并且影子寄存器被更新
先决条件	-
输入参数{in}	
-	-
输入参数{in}	
-	-
返回值	
ErrStatus	ERROR或SUCCESS

例如:

```
ErrStatus error_status;
```

```
/* wait until RTC_TIME and RTC_DATE registers are synchronized with APB clock, and the shadow registers are updated */
```

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

### 函数 rtc\_current\_time\_get

函数rtc\_current\_time\_get描述见下表:



表 3-397. 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	-
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3-388. 结构体类型<code>rtc_parameter_struct</code></a>
返回值	
-	-

例如：

```
rtc_parameter_struct rtc_initpara_struct;

/* get current time and date */
rtc_current_time_get (& rtc_initpara_struct);
```

### 函数 `rtc_subsecond_get`

函数`rtc_subsecond_get`描述见下表：

表 3-398. 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	-
输入参数{in}	
-	-
输出参数{out}	
-	-
返回值	
<code>uint32_t</code>	当前的亚秒值(0x00-0xFFFF)

例如：

```
uint32_t sub_second;

/* get current subsecond value */
uint32_t sub_second = rtc_subsecond_get ( );
```

### 函数 `rtc_alarm_config`

函数`rtc_alarm_config`描述见下表：

表 3-399. 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	-
输入参数{in}	
rtc_alarm_time	闹钟结构体，结构体成员参考 <a href="#">表3-389. 结构体类型rtc_alarm_struct</a>
输出参数{out}	
-	-
返回值	
-	-

例如：

```
rtc_alarm_struct rtc_alarm_time;

/* rtc_alarm_config */

rtc_alarm_config (&rtc_alarm_time);
```

### 函数 rtc\_alarm\_subsecond\_config

函数rtc\_alarm\_subsecond\_config描述见下表：

表 3-400. 函数 rtc\_alarm\_subsecond\_config

函数名称	rtc_alarm_subsecond_config
函数原型	void rtc_alarm_subsecond_config(uint32_t mask_subsecond, uint32_t subsecond);
功能描述	配置RTC闹钟的亚秒值
先决条件	-
输入参数{in}	
mask_subsecond	闹钟亚秒屏蔽位
RTC_MASKSSC_0_14	屏蔽闹钟亚秒设置
RTC_MASKSSC_1_14	屏蔽RTC_ALRM0SS_SSC[14:1]，SSC[0]位用于时间匹配
RTC_MASKSSC_2_14	屏蔽RTC_ALRM0SS_SSC[14:2]，SSC[1:0]位用于时间匹配
RTC_MASKSSC_3_14	屏蔽RTC_ALRM0SS_SSC[14:3]，SSC[2:0]位用于时间匹配
RTC_MASKSSC_4_14	屏蔽RTC_ALRM0SS_SSC[14:4]，SSC[3:0]位用于时间匹配
RTC_MASKSSC_5_14	屏蔽RTC_ALRM0SS_SSC[14:5]，SSC[4:0]位用于时间匹配
RTC_MASKSSC_6_14	屏蔽RTC_ALRM0SS_SSC[14:6]，SSC[5:0]位用于时间匹配
RTC_MASKSSC_7_14	屏蔽RTC_ALRM0SS_SSC[14:7]，SSC[6:0]位用于时间匹配
RTC_MASKSSC_8_14	屏蔽RTC_ALRM0SS_SSC[14:8]，SSC[7:0]位用于时间匹配
RTC_MASKSSC_9_14	屏蔽RTC_ALRM0SS_SSC[14:9]，SSC[8:0]位用于时间匹配
RTC_MASKSSC_10_14	屏蔽RTC_ALRM0SS_SSC[14:10]，SSC[9:0]位用于时间匹配
4	
RTC_MASKSSC_11_14	屏蔽RTC_ALRM0SS_SSC[14:11]，SSC[10:0]位用于时间匹配

4	
RTC_MASKSSC_12_1	屏蔽RTC_ALARM0SS_SSC[14:12], SSC[11:0]位用于时间匹配
4	
RTC_MASKSSC_13_1	屏蔽RTC_ALARM0SS_SSC[14:13], SSC[12:0]位用于时间匹配
4	
RTC_MASKSSC_14	屏蔽RTC_ALARM0SS_SSC[14], SSC[13:0]位用于时间匹配
RTC_MASKSSC_NON	无屏蔽, SSC[14:0]位用于时间匹配
E	
输入参数{in}	
subsecond	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\_get

函数rtc\_alarm\_get描述见下表:

表 3-401. 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	-
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3-389. 结构体类型rtc_alarm_struct</a>
返回值	
-	-

例如:

```
rtc_alarm_struct rtc_alarm_time;

/* get RTC alarm */
rtc_alarm_get (&rtc_alarm_time);
```

**函数 rtc\_alarm\_subsecond\_get**

函数rtc\_alarm\_subsecond\_get描述见下表:

**表 3-402. 函数 rtc\_alarm\_subsecond\_get**

函数名称	rtc_alarm_subsecond_get
函数原型	uint32_t rtc_alarm_subsecond_get(void);
功能描述	获取RTC闹钟亚秒值
先决条件	-
输入参数{in}	
-	-
输出参数{out}	
-	-
返回值	
uint32_t	RTC 闹钟亚秒值(0x0 - 0x7FFF)

例如:

```
uint32_t subsecond;
```

```
/* get RTC alarm subsecond */
```

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

**函数 rtc\_alarm\_enable**

函数rtc\_alarm\_enable描述见下表:

**表 3-403. 函数 rtc\_alarm\_enable**

函数名称	rtc_alarm_enable
函数原型	void rtc_alarm_enable(void);
功能描述	使能RTC闹钟
先决条件	-
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 3-404. 函数 rtc\_alarm\_disable**

函数名称	rtc_alarm_disable
函数原型	ErrStatus rtc_alarm_disable(void);
功能描述	失能RTC闹钟
先决条件	-
输入参数{in}	
-	-
输出参数{out}	
-	-
返回值	
ErrStatus	ERROR或SUCCESS

例如:

```
ErrStatus error_status;
```

```
/* disable RTC alarm */
```

```
ErrStatus error_status = rtc_alarm_disable ( );
```

**函数 rtc\_timestamp\_enable**

函数can\_init描述见下表:

**表 3-405. 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	-
输入参数{in}	
edge	选定哪种边沿触发时间戳检测
RTC_TIMESTAMP_RISING_EDGE	上升沿是时间戳事件有效检测沿
RTC_TIMESTAMP_FALLING_EDGE	下降沿是时间戳事件有效检测沿
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* enable RTC time-stamp */
```

rtc\_timestamp\_enable (RTC\_TIMESTAMP\_RISING\_EDGE);

### 函数 rtc\_timestamp\_disable

函数rtc\_timestamp\_disable描述见下表:

表 3-406. 函数 rtc\_timestamp\_disable

函数名称	rtc_timestamp_disable
函数原型	void rtc_timestamp_disable(void);
功能描述	失能RTC时间戳
先决条件	-
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 3-407. 函数 rtc\_timestamp\_get

函数名称	rtc_timestamp_get
函数原型	void rtc_timestamp_get(rtc_timestamp_struct* rtc_timestamp);
功能描述	获取RTC时间戳时间和日期
先决条件	-
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3-390. 结构体rtc_timestamp_struct</a>
返回值	
-	-

例如:

```
rtc_timestamp_struct rtc_timestamp;
```

```
/* get RTC timestamp time and date */
```

```
rtc_timestamp_get (& rtc_timestamp);
```

**函数 rtc\_timestamp\_subsecond\_get**

函数rtc\_timestamp\_subsecond\_get描述见下表:

**表 3-408. 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	-
输入参数{in}	
-	-
输出参数{out}	
-	-
返回值	
uint32_t	RTC时间戳亚秒值

例如:

```
uint32_t subsecond;
```

```
/* get RTC time-stamp subsecond */
```

```
uint32_t subsecond = rtc_timestamp_subsecond_get ( );
```

**函数 rtc\_tamper\_enable**

函数rtc\_tamper\_enable描述见下表:

**表 3-409. 函数 rtc\_tamper\_enable**

函数名称	rtc_tamper_enable
函数原型	void rtc_tamper_enable(rtc_tamper_struct* rtc_tamper);
功能描述	使能RTC侵入检测
先决条件	-
输入参数{in}	
rtc_tamper	tamper化结构体, 结构体成员参考 <a href="#">表3-391. 结构体rtc_tamper_struct</a>
输出参数{out}	
-	-
返回值	
-	-

例如:

```
rtc_tamper_struct rtc_tamper
```

```
/* enable RTC tamper */
```

```
rtc_tamper_enable (& rtc_tamper);
```

### 函数 rtc\_tamper\_disable

函数rtc\_tamper\_disable描述见下表：

表 3-410. 函数 rtc\_tamper\_disable

函数名称	rtc_tamper_disable
函数原型	void rtc_tamper_disable(uint32_t source);
功能描述	失能RTC侵入检测
先决条件	-
输入参数{in}	
source	选定被失能的侵入检测来源
RTC_TAMPER0	RTC tamper0
RTC_TAMPER1	RTC tamper1
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* disable RTC tamper */
rtc_tamper_disable (RTC_TAMPER0);
```

### 函数 rtc\_interrupt\_enable

函数rtc\_interrupt\_enable描述见下表：

表 3-411. 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	-
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_ALARM	闹钟中断
RTC_INT_TAMP	侵入检测中断
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* enable specified RTC interrupt */
```



```
rtc_interrupt_enable (RTC_INT_TAMP);
```

### 函数 `rtc_interrupt_disable`

函数`rtc_interrupt_disable`描述见下表：

**表 3-412. 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	-
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_ALARM</code>	闹钟中断
<code>RTC_INT_TAMP</code>	侵入检测中断
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* disble RTC ALARM interrupt */
```

```
rtc_interrupt_disable (RTC_INT_TAMP);
```

### 函数 `rtc_flag_get`

函数`rtc_flag_get`描述见下表：

**表 3-413. 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	-
输入参数{in}	
<b>flag</b>	选定被获取的中断标志
<code>RTC_FLAG_RECALIBRATION</code>	平滑校准挂起标志
<code>RTC_FLAG_TAMP1</code>	tamper 1事件标志
<code>RTC_FLAG_TAMP0</code>	tamper 0事件标志
<code>RTC_FLAG_TIMESTAMP_OVERFLOW</code>	时间戳事件溢出标志
<code>RTC_FLAG_TIMESTAMP</code>	时间戳事件标志

<i>MP</i>	
<i>RTC_FLAG_ALARM0</i>	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_SHIFT</i>	移位功能操作挂起标志
<i>RTC_FLAG_ALARM0_WRITTEN</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_TIMESTAMP);
```

### 函数 **rtc\_flag\_clear**

函数rtc\_flag\_clear描述见下表：

**表 3-414. 函数 **rtc\_flag\_clear****

函数名称	rtc_flag_clear
函数原型	void rtc_flag_clear(uint32_t flag);
功能描述	清除指定中断标志位
先决条件	-
输入参数{in}	
<b>flag</b>	要清除的中断标志位
<i>RTC_FLAG_TAMP1</i>	tamper 1 事件标志
<i>RTC_FLAG_TAMP0</i>	tamper 0 事件标志
<i>RTC_FLAG_TIMESTAMP_OVERFLOW</i>	时间戳事件溢出标志
<i>RTC_FLAG_TIMESTAMP_MP</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：

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

```
rtc_flag_clear (RTC_FLAG_TIMESTAMP);
```

### 函数 rtc\_alter\_output\_config

函数rtc\_alter\_output\_config描述见下表：

表 3-415. 函数 rtc\_alter\_output\_config

函数名称	rtc_alter_output_config
函数原型	void rtc_alter_output_config(uint32_t source, uint32_t mode);
功能描述	配置RTC备用输出源
先决条件	-
输入参数{in}	
source	指定输出信号
RTC_CALIBRATION_5 12HZ	当LSE时钟频率为32768Hz并且RTC_PSC 为默认值，输出512Hz信号
RTC_CALIBRATION_1 HZ	当LSE时钟频率为32768Hz并且RTC_PSC 为默认值，输出1Hz信号
RTC_ALARM_HIGH	当设置了闹钟标志置位，输出引脚为高电平
RTC_ALARM_LOW	当设置了闹钟标志置位，输出引脚为低电平
输入参数{in}	
mode	当输出闹钟信号时指定输出引脚(PC13)的模式
RTC_ALARM_OUTPU T_OD	开漏输出
RTC_ALARM_OUTPU T_PP	推挽输出
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* configure rtc alternate output source */
```

```
rtc_alter_output_config (RTC_ALARM_LOW, RTC_ALARM_OUTPUT_PP);
```

### 函数 rtc\_calibration\_config

函数rtc\_calibration\_config描述见下表：

表 3-416. 函数 rtc\_calibration\_config

Function name	rtc_calibration_config
Function prototype	ErrStatus rtc_calibration_config(uint32_t window, uint32_t plus, uint32_t minus);
Function descriptions	配置RTC校准寄存器
Precondition	-

输入参数{in}	
<b>window</b>	选择校准窗口
<i>RTC_CALIBRATION_WINDOW_32S</i>	如果RTCCLK = 32768Hz在32秒校准窗增加2exp20 RTCCLK周期
<i>RTC_CALIBRATION_WINDOW_16S</i>	如果RTCCLK = 32768 Hz在16秒校准窗增加2exp19 RTCCLK周期
<i>RTC_CALIBRATION_WINDOW_8S</i>	如果RTCCLK = 32768 Hz在8秒校准窗增加2exp18 RTCCLK周期
输入参数{in}	
<b>plus</b>	增加或者不增加RTC脉冲
<i>RTC_CALIBRATION_PLUS_SET</i>	每2048个RTC脉冲增加一个RTC脉冲
<i>RTC_CALIBRATION_PLUS_RESET</i>	无影响
输入参数{in}	
<b>minus</b>	在校准窗口期间RTC减少的时钟(0x0 - 0x1FF)
Output parameter{out}	
-	-
Return value	
<b>ErrStatus</b>	ERROR或SUCCESS

例如:

```
ErrStatus error_status;
```

```
/* configure RTC calibration register */
```

```
ErrStatus error_status = rtc_calibration_config(RTC_CALIBRATION_WINDOW_32S,
RTC_CALIBRATION_PLUS_SET, 0x1FF);
```

## 函数 rtc\_hour\_adjust

函数rtc\_hour\_adjust描述见下表:

表 3-417. 函数 rtc\_hour\_adjust

<b>函数名称</b>	rtc_hour_adjust
<b>函数原型</b>	void rtc_hour_adjust(uint32_t operation);
<b>功能描述</b>	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
<b>先决条件</b>	-
输入参数{in}	
<b>operation</b>	小时调整操作
<i>RTC_CTL_A1H</i>	增加一个小时
<i>RTC_CTL_S1H</i>	减少一个小时
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 3-418. 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	-
输入参数{in}	
<b>add</b>	在当前时间上增加1S或者不增加
<i>RTC_SHIFT_ADD1S_R ESET</i>	无影响
<i>RTC_SHIFT_ADD1S_S ET</i>	在当前时间增加1秒
输入参数{in}	
<b>minus</b>	在当前是时间上减少的亚秒值(0x0 - 0x7FFF)
输出参数{out}	
-	-
返回值	
-	-

例如：

```
ErrStatus error_status;
```

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

```
ErrStatus error_status = rtc_second_adjust (RTC_SHIFT_ADD1S_SET, 0);
```

### 函数 rtc\_bypass\_shadow\_enable

函数rtc\_bypass\_shadow\_enable描述见下表：

**表 3-419. 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	-

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 3-420. 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	-
输入参数{in}	
-	-
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* disable RTC bypass shadow registers function */
```

```
rtc_bypass_shadow_disable ( );
```

### 函数 rtc\_refclock\_detection\_enable

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

表 3-421. 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	-
输入参数{in}	
-	-
输出参数{out}	

-	-
返回值	
ErrStatus	ERROR或SUCCESS

例如：

```
ErrStatus error_status;
```

```
/* 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 3-422. 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	-
输入参数{in}	
-	-
输出参数{out}	
-	-
返回值	
ErrStatus	ERROR或SUCCESS

例如：

```
ErrStatus error_status;
```

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

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

## 3.19. SPI/I2S

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[3.19.1](#)描述了SPI/I2S的寄存器列表，章节[3.19.2](#)对SPI/I2S库函数进行说明。

### 3.19.1. 外设寄存器说明

SPI/I2S寄存器列表如下表所示：

表 3-423. SPI/I2S 寄存器

寄存器名称	寄存器描述
SPI_CTL0	控制寄存器0
SPI_CTL1	控制寄存器1
SPI_STAT	状态寄存器
SPI_DATA	数据寄存器
SPI_CRCPOLY	CRC多项式寄存器
SPI_RCRC	接收CRC寄存器
SPI_TCRC	发送CRC寄存器
SPI_I2SCTL	I2S控制寄存器
SPI_I2SPSC	I2S时钟分频寄存器
SPI_QCTL	四路SPI控制寄存器

### 3.19.2. 外设库函数说明

SPI/I2S库函数列表如下表所示：

表 3-424. SPI/I2S 库函数

库函数名称	库函数描述
spi_i2s_deinit	复位外设SPIx/I2Sx
spi_struct_para_init	将SPI结构体中所有参数初始化为默认值
spi_init	初始化外设SPIx
spi_enable	使能外设SPIx
spi_disable	禁能外设SPIx
i2s_init	初始化外设I2Sx
i2s_psc_config	配置I2Sx预分频器
i2s_enable	使能外设I2Sx
i2s_disable	禁能外设I2Sx
spi_nss_output_enable	使能外设SPIx NSS输出
spi_nss_output_disable	禁能外设SPIx NSS输出
spi_nss_internal_high	NSS软件模式下NSS引脚拉高
spi_nss_internal_low	NSS软件模式下NSS引脚拉低
spi_dma_enable	使能外设SPIx的DMA功能
spi_dma_disable	禁能外设SPIx的DMA功能
spi_i2s_data_frame_format_config	配置外设SPIx/I2Sx数据帧格式
spi_bidirectional_transfer_config	配置外设SPIx的数据传输方向
spi_i2s_data_transmit	发送数据
spi_i2s_data_receive	接收数据
spi_i2s_format_error_clear	清除SPIx/I2Sx帧错误标志
spi_crc_polynomial_set	设置外设SPIx的CRC多项式值
spi_crc_polynomial_get	获取外设SPIx的CRC多项式值
spi_crc_on	打开外设SPIx的CRC功能



库函数名称	库函数描述
spi_crc_off	关闭外设SPIx的CRC功能
spi_crc_next	设置外设SPIx下一次传输数据为CRC值
spi_crc_get	外设SPIx获取CRC值
spi_crc_error_clear	清除SPIx CRC错误标志
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SPI四线模式
spi_quad_disable	禁能SPI四线模式
spi_quad_write_enable	使能SPI四线写
spi_quad_read_enable	使能SPI四线读
spi_i2s_flag_get	获取外设SPIx/I2Sx标志状态
spi_i2s_interrupt_enable	使能外设SPIx/I2Sx中断
spi_i2s_interrupt_disable	禁能外设SPIx/I2Sx中断
spi_i2s_interrupt_flag_get	获取外设SPIx/I2Sx中断状态

### 结构体 spi\_parameter\_struct

表 3-425. 结构体类型 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_BDRECEIVE, SPI_TRANSMODE_BDTRANSMIT)
frame_size	数据帧格式配置 (SPI_FRAME_SIZE_8BIT, SPI_FRAME_SIZE_16BIT)
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 3-426. 函数 spi\_i2s\_deinit

函数名称	spi_i2s_deinit
函数原形	void spi_i2s_deinit(uint32_t spi_periph);
功能描述	复位外设SPIx/I2Sx
先决条件	-
被调用函数	rcu_periph_reset_enable / rcu_periph_reset_disable
输入参数{in}	
spi_periph	外设SPIx
SPIx	x=0,1
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* reset SPI0 */
```

```
spi_i2s_deinit(SPI0);
```

### 函数 spi\_struct\_para\_init

函数spi\_struct\_para\_init描述见下表：

表 3-427. 函数 spi\_struct\_para\_init

函数名称	spi_struct_para_init
函数原形	void spi_struct_para_init(spi_parameter_struct* spi_struct);
功能描述	将SPI结构体参数初始化为默认值
先决条件	-
被调用函数	-
输入参数{in}	
spi_struct	SPI初始化结构体，结构体成员参考 <a href="#">表3-425. 结构体类型 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 3-428. 函数 spi\_init**

函数名称	spi_init
函数原形	void spi_init(uint32_t spi_periph, spi_parameter_struct* spi_struct);
功能描述	初始化外设SPIx
先决条件	-
被调用函数	-
输入参数{in}	
spi_periph	外设SPIx
SPIx	x=0,1
输入参数{in}	
spi_struct	初始化结构体，结构体成员参考 <a href="#">表3-425. 结构体类型 spi_parameter_struct</a> 。
输出参数{out}	
-	-
返回值	
-	-

例如:

```

/* initialize SPI0 */

spi_parameter_struct spi_init_struct;

spi_init_struct.trans_mode      = SPI_TRANSMODE_BDTRANSMIT;
spi_init_struct.device_mode     = SPI_MASTER;
spi_init_struct.frame_size      = SPI_FRAME_SIZE_8BIT;
spi_init_struct.clock_polarity_phase = SPI_CK_PL_HIGH_PH_2EDGE;
spi_init_struct.nss             = SPI_NSS_SOFT;
spi_init_struct.prescale        = SPI_PSC_8;
spi_init_struct.endian          = SPI_ENDIAN_MSB;

spi_init(SPI0, &spi_init_struct);

```

**函数 spi\_enable**

函数spi\_enable描述见下表:

**表 3-429. 函数 spi\_enable**

函数名称	spi_enable
函数原形	void spi_enable(uint32_t spi_periph);

功能描述	使能外设SPIx
先决条件	-
被调用函数	-
输入参数{in}	
spi_periph	外设SPIx
SPIx	x=0,1
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* enable SPI0 */
spi_enable(SPI0);
```

### 函数 spi\_disable

函数spi\_disable描述见下表：

表 3-430. 函数 spi\_disable

函数名称	spi_disable
函数原形	void spi_disable(uint32_t spi_periph);
功能描述	禁能外设SPIx
先决条件	-
被调用函数	-
输入参数{in}	
spi_periph	外设SPIx
SPIx	x=0,1
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* disable SPI0 */
spi_disable(SPI0);
```

### 函数 i2s\_init

函数i2s\_init描述见下表：

表 3-431. 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I2S0
先决条件	-
被调用函数	-
输入参数{in}	
spi_periph	外设I2S0
SPIx	x=0
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_MASTERTX	I2S主机发送模式
I2S_MODE_MASTERRX	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 3-432. 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I2S0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I2S0
SPIx	x=0
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}	
frameformat	I2S数据长度和通道长度
I2S_FRAMEFORMA T_DT16B_CH16B	I2S数据长度为16位，通道长度为16位
I2S_FRAMEFORMA T_DT16B_CH32B	I2S数据长度为16位，通道长度为32位
I2S_FRAMEFORMA T_DT24B_CH32B	I2S数据长度为24位，通道长度为32位
I2S_FRAMEFORMA	I2S数据长度为32位，通道长度为32位

<i>T_DT32B_CH32B</i>	
输入参数{in}	
<b>mckout</b>	I2S_MCK输出使能
<i>I2S_MCKOUT_ENABLE</i>	I2S_MCK输出使能
<i>I2S_MCKOUT_DISABLE</i>	I2S_MCK输出禁止
输出参数{out}	
-	-
返回值	
-	-

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 3-433. 函数 i2s\_enable

函数名称	i2s_enable
函数原形	void i2s_enable(uint32_t spi_periph);
功能描述	使能外设I2S0
先决条件	-
被调用函数	-
输入参数{in}	
<b>spi_periph</b>	外设I2S0
<i>SPIx</i>	x=0
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* enable I2S0 */
```

```
i2s_enable(SPI0);
```

### 函数 i2s\_disable

函数i2s\_disable描述见下表：

表 3-434. 函数 i2s\_disable

函数名称	i2s_disable
函数原形	void i2s_disable(uint32_t spi_periph);
功能描述	禁能外设I2S0
先决条件	-
被调用函数	-
输入参数{in}	
spi_periph	外设I2S0
SPIx	x=0
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 3-435. 函数 spi\_nss\_output\_enable

函数名称	spi_nss_output_enable
函数原形	void spi_nss_output_enable(uint32_t spi_periph);
功能描述	使能外设SPIx NSS输出
先决条件	-
被调用函数	-
输入参数{in}	
spi_periph	外设SPIx
SPIx	x=0,1
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* enable SPI0 NSS output */
```

```
spi_nss_output_enable(SPI0);
```

### 函数 spi\_nss\_output\_disable

函数spi\_nss\_output\_disable描述见下表：



表 3-436. 函数 `spi_nss_output_disable`

函数名称	<code>spi_nss_output_disable</code>
函数原形	<code>void spi_nss_output_disable(uint32_t spi_periph);</code>
功能描述	禁能外设SPIx NSS输出
先决条件	-
被调用函数	-
输入参数{in}	
<code>spi_periph</code>	外设SPIx
<code>SPIx</code>	x=0,1
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* disable SPI0 NSS output */
spi_nss_output_disable(SPI0);
```

### 函数 `spi_nss_internal_high`

函数`spi_nss_internal_high`描述见下表：

表 3-437. 函数 `spi_nss_internal_high`

函数名称	<code>spi_nss_internal_high</code>
函数原形	<code>void spi_nss_internal_high(uint32_t spi_periph);</code>
功能描述	NSS软件模式下NSS引脚拉高
先决条件	-
被调用函数	-
输入参数{in}	
<code>spi_periph</code>	外设SPIx
<code>SPIx</code>	x=0,1
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* SPI0 NSS pin is pulled high level in software mode */
spi_nss_internal_high(SPI0);
```

### 函数 `spi_nss_internal_low`

函数`spi_nss_internal_low`描述见下表：

表 3-438. 函数 `spi_nss_internal_low`

函数名称	<code>spi_nss_internal_low</code>
函数原形	<code>void spi_nss_internal_low(uint32_t spi_periph);</code>
功能描述	NSS软件模式下NSS引脚拉低
先决条件	-
被调用函数	-
输入参数{in}	
<code>spi_periph</code>	外设SPIx
<code>SPIx</code>	x=0,1
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* SPI0 NSS pin is pulled low level in software mode */
```

```
spi_nss_internal_low(SPI0);
```

### 函数 `spi_dma_enable`

函数`spi_dma_enable`描述见下表：

表 3-439. 函数 `spi_dma_enable`

函数名称	<code>spi_dma_enable</code>
函数原形	<code>void spi_dma_enable(uint32_t spi_periph, uint8_t dma);</code>
功能描述	使能外设SPIx的DMA功能
先决条件	-
被调用函数	-
输入参数{in}	
<code>spi_periph</code>	外设SPIx
<code>SPIx</code>	x=0,1
输入参数{in}	
<code>dma</code>	SPI DMA模式
<code>SPI_DMA_TRANSMIT</code>	SPI发送缓冲区DMA使能
<code>SPI_DMA_RECEIVE</code>	SPI接收缓冲区DMA使能
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* 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 3-440. 函数 spi\_dma\_disable**

函数名称	spi_dma_disable
函数原形	void spi_dma_disable(uint32_t spi_periph, uint8_t dma);
功能描述	禁能外设SPIx的DMA功能
先决条件	-
被调用函数	-
输入参数{in}	
spi_periph	外设SPIx
SPIx	x=0,1
输入参数{in}	
dma	SPI DMA模式
SPI_DMA_TRANSMIT	SPI发送缓冲区DMA使能
SPI_DMA_RECEIVE	SPI接收缓冲区DMA使能
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* disable SPI0 transmit data DMA function */
```

```
spi_dma_disable(SPI0, SPI_DMA_TRANSMIT);
```

### 函数 spi\_i2s\_data\_frame\_format\_config

函数spi\_i2s\_data\_frame\_format\_config描述见下表:

**表 3-441. 函数 spi\_i2s\_data\_frame\_format\_config**

函数名称	spi_i2s_data_frame_format_config
函数原形	void spi_i2s_data_frame_format_config(uint32_t spi_periph, uint16_t frame_format);
功能描述	配置外设SPIx/I2S0数据帧格式
先决条件	-
被调用函数	-
输入参数{in}	

<b>spi_periph</b>	外设SPIx
<i>SPIx</i>	x=0,1
输入参数{in}	
<b>frame_format</b>	SPI帧大小
<i>SPI_FRAME_SIZE_8BIT</i>	SPI 8位数据帧格式
<i>SPI_FRAME_SIZE_16BIT</i>	SPI 16位数据帧格式
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* configure SPI0/I2S0 data frame format size is 16 bits */
```

```
spi_i2s_data_frame_format_config(SPI1, SPI_FRAME_SIZE_16BIT);
```

### 函数 spi\_bidirectional\_transfer\_config

函数spi\_bidirectional\_transfer\_config描述见下表：

表 3-442. 函数 spi\_bidirectional\_transfer\_config

函数名称	spi_bidirectional_transfer_config
函数原形	void spi_bidirectional_transfer_config(uint32_t spi_periph, uint32_t transfer_direction);
功能描述	配置外设SPIx的数据传输方向
先决条件	-
被调用函数	-
输入参数{in}	
<b>spi_periph</b>	外设SPIx
<i>SPIx</i>	x=0,1
输入参数{in}	
<b>transfer_direction</b>	SPI双向传输输出使能
<i>SPI_BIDIRECTIONAL_TRANSMIT</i>	SPI工作在只发送模式
<i>SPI_BIDIRECTIONAL_RECEIVE</i>	SPI工作在只接收模式
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* SPI0 works in transmit-only mode */
```

```
spi_bidirectional_transfer_config(SPI0, SPI_BIDIRECTIONAL_TRANSMIT);
```

### 函数 spi\_i2s\_data\_transmit

函数spi\_i2s\_data\_transmit描述见下表：

表 3-443. 函数 spi\_i2s\_data\_transmit

函数名称	spi_i2s_data_transmit
函数原形	void spi_i2s_data_transmit(uint32_t spi_periph, uint16_t data);
功能描述	SPI发送数据
先决条件	-
被调用函数	-
输入参数{in}	
spi_periph	外设SPIx
SPIx	x=0,1
输入参数{in}	
data	16位数据
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* SPI0 transmit data */
```

```
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 3-444. 函数 spi\_i2s\_data\_receive

函数名称	spi_i2s_data_receive
函数原形	uint16_t spi_i2s_data_receive(uint32_t spi_periph);
功能描述	SPI接收数据
先决条件	-
被调用函数	-
输入参数{in}	
spi_periph	外设SPIx
SPIx	x=0,1
输出参数{out}	
-	-
返回值	

uint16_t	16位数据
----------	-------

例如:

```
/* SPI0 receive data */
```

```
spi0_receive_array[receive_n] = spi_i2s_data_receive(SPI0);
```

### 函数 spi\_i2s\_format\_error\_clear

函数spi\_i2s\_format\_error\_clear描述见下表:

表 3-445. 函数 spi\_i2s\_format\_error\_clear

函数名称	spi_i2s_format_error_clear
函数原形	void spi_i2s_format_error_clear(uint32_t spi_periph, uint32_t flag);
功能描述	清除SPIx/I2Sx帧错误标志
先决条件	-
被调用函数	-
输入参数{in}	
spi_periph	外设SPIx
SPIx	x=0,1
输入参数{in}	
flag	SPI/I2S帧错误标志
SPI_FLAG_FERR	SPI帧错误标志, 仅适用于TI模式
I2S_FLAG_FERR	I2S帧错误标志
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* clear SPI0 format error flag status */
```

```
spi_i2s_format_error_clear(SPI0, SPI_FLAG_FERR);
```

### 函数 spi\_crc\_polynomial\_set

函数spi\_crc\_polynomial\_set描述见下表:

表 3-446. 函数 spi\_crc\_polynomial\_set

函数名称	spi_crc_polynomial_set
函数原形	void spi_crc_polynomial_set(uint32_t spi_periph, uint16_t crc_poly);
功能描述	设置外设SPIx的CRC多项式值
先决条件	-
被调用函数	-
输入参数{in}	

<b>spi_periph</b>	外设SPIx
<i>SPIx</i>	x=0,1
输入参数{in}	
<b>crc_poly</b>	CRC多项式值
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* set SPI0 CRC polynomial */
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 3-447. 函数 spi\_crc\_polynomial\_get

函数名称	spi_crc_polynomial_get
函数原形	uint16_t spi_crc_polynomial_get(uint32_t spi_periph);
功能描述	获取外设SPIx的CRC多项式值
先决条件	-
被调用函数	-
输入参数{in}	
<b>spi_periph</b>	外设SPIx
<i>SPIx</i>	x=0,1
输出参数{out}	
-	-
返回值	
<b>uint16_t</b>	16位CRC多项式值（0-0xFFFF）

例如：

```
/* get SPI0 CRC polynomial */
uint16_t crc_val;
crc_val = spi_crc_polynomial_get(SPI0);
```

### 函数 spi\_crc\_on

函数spi\_crc\_on描述见下表：

表 3-448. 函数 spi\_crc\_on

函数名称	spi_crc_on
函数原形	void spi_crc_on(uint32_t spi_periph);

功能描述	打开外设SPIx的CRC功能
先决条件	-
被调用函数	-
输入参数{in}	
spi_periph	外设SPIx
SPIx	x=0,1
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* turn on SPI0 CRC function */
```

```
spi_crc_on(SPI0);
```

### 函数 spi\_crc\_off

函数spi\_crc\_off描述见下表：

表 3-449. 函数 spi\_crc\_off

函数名称	spi_crc_off
函数原形	void spi_crc_off(uint32_t spi_periph);
功能描述	关闭外设SPIx的CRC功能
先决条件	-
被调用函数	-
输入参数{in}	
spi_periph	外设SPIx
SPIx	x=0,1
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* turn off SPI0 CRC function */
```

```
spi_crc_off(SPI0);
```

### 函数 spi\_crc\_next

函数spi\_crc\_next描述见下表：

表 3-450. 函数 spi\_crc\_next

函数名称	spi_crc_next
------	--------------



函数原形	void spi_crc_next(uint32_t spi_periph);
功能描述	设置外设SPIx下一次传输数据为CRC值
先决条件	-
被调用函数	-
输入参数{in}	
spi_periph	外设SPIx
SPIx	x=0,1
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* SPI0 next data is CRC value */
```

```
spi_crc_next(SPI0);
```

### 函数 spi\_crc\_get

函数spi\_crc\_get描述见下表：

表 3-451. 函数 spi\_crc\_get

函数名称	spi_crc_get
函数原形	uint16_t spi_crc_get(uint32_t spi_periph, uint8_t crc);
功能描述	外设SPIx获取CRC值
先决条件	-
被调用函数	-
输入参数{in}	
spi_periph	外设SPIx
SPIx	x=0,1
输入参数{in}	
crc	SPI CRC值
SPI_CRC_TX	获取发送CRC寄存器值
SPI_CRC_RX	获取接收CRC寄存器值
输出参数{out}	
-	-
返回值	
uint16_t	16位CRC值（0-0xFFFF）

例如：

```
/* get SPI0 CRC send value */
```

```
uint16_t crc_val;
```

```
crc_val = spi_crc_get(SPI0, SPI_CRC_TX);
```

**函数 spi\_crc\_error\_clear**

函数spi\_crc\_error\_clear描述见下表：

**表 3-452. 函数 spi\_crc\_error\_clear**

函数名称	spi_crc_error_clear
函数原形	void spi_crc_error_clear(uint32_t spi_periph);
功能描述	清除SPIx CRC错误标志
先决条件	-
被调用函数	-
输入参数{in}	
spi_periph	外设SPIx
SPIx	x=0,1
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* clear SPI0 CRC error flag status */
```

```
spi_crc_error_clear(SPI0);
```

**函数 spi\_ti\_mode\_enable**

函数spi\_ti\_mode\_enable描述见下表：

**表 3-453. 函数 spi\_ti\_mode\_enable**

函数名称	spi_ti_mode_enable
函数原形	void spi_ti_mode_enable(uint32_t spi_periph);
功能描述	使能SPI TI模式
先决条件	-
被调用函数	-
输入参数{in}	
spi_periph	外设SPIx
SPIx	x=0,1
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* enable SPI0 TI mode */
```

```
spi_ti_mode_enable(SPI0);
```

**函数 spi\_ti\_mode\_disable**

函数spi\_ti\_mode\_disable描述见下表:

**表 3-454. 函数 spi\_ti\_mode\_disable**

函数名称	spi_ti_mode_disable
函数原形	void spi_ti_mode_disable(uint32_t spi_periph);
功能描述	禁能SPI TI模式
先决条件	-
被调用函数	-
输入参数{in}	
spi_periph	外设SPIx
SPIx	x=0,1
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* disable SPI0 TI mode */
spi_ti_mode_disable(SPI0);
```

**函数 spi\_nssp\_mode\_enable**

函数spi\_nssp\_mode\_enable描述见下表:

**表 3-455. 函数 spi\_nssp\_mode\_enable**

函数名称	spi_nssp_mode_enable
函数原形	void spi_nssp_mode_enable(uint32_t spi_periph);
功能描述	使能SPI NSS脉冲模式
先决条件	-
被调用函数	-
输入参数{in}	
spi_periph	外设SPIx
SPIx	x=0,1
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 3-456. 函数 spi\_nssp\_mode\_disable**

函数名称	spi_nssp_mode_disable
函数原形	void spi_nssp_mode_disable(uint32_t spi_periph);
功能描述	禁能SPI NSS脉冲模式
先决条件	-
被调用函数	-
输入参数{in}	
spi_periph	外设SPIx
SPIx	x=0,1
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 3-457. 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SPI四线模式
先决条件	-
被调用函数	-
输入参数{in}	
spi_periph	外设SPIx
SPIx	x=1
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 3-458. 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	禁能SPI四线模式
先决条件	-
被调用函数	-
输入参数{in}	
spi_periph	外设SPIx
SPIx	x=1
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 3-459. 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SPI四线写
先决条件	-
被调用函数	-
输入参数{in}	
spi_periph	外设SPIx
SPIx	x=1
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 3-460. 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SPI四线读
先决条件	-
被调用函数	-
输入参数{in}	
spi_periph	外设SPIx
SPIx	x=1
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* enable SPI1 quad wire read */
```

```
spi_quad_read_enable(SPI1);
```

**函数 spi\_i2s\_flag\_get**

函数spi\_i2s\_flag\_get描述见下表:

**表 3-461. 函数 spi\_i2s\_flag\_get**

函数名称	spi_i2s_flag_get
函数原形	FlagStatus spi_i2s_flag_get(uint32_t spi_periph, uint32_t flag);
功能描述	获取外设SPIx/I2S0标志状态
先决条件	-
被调用函数	-
输入参数{in}	
spi_periph	外设SPIx
SPIx	x=0,1
输入参数{in}	
flag	SPI/I2S标志状态
SPI_FLAG_TBE	SPI发送缓冲区空标志
SPI_FLAG_RBNE	SPI接收缓冲区非空标志
SPI_FLAG_TRANS	SPI通信进行中标志
SPI_FLAG_RXORERR	SPI接收过载错误标志
SPI_FLAG_CONFE	SPI配置错误标志

<i>RR</i>	
<i>SPI_FLAG_CRCER</i> <i>R</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_RXORE</i> <i>RR</i>	I2S接收过载错误标志
<i>I2S_FLAG_TXURE</i> <i>RR</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
输出参数{out}	
-	-
返回值	
<b>FlagStatus</b>	SET或RESET

例如:

```
/* get SPI0 transmit buffer empty flag status */
while(RESET == spi_i2s_flag_get(SPI0, SPI_FLAG_TBE));
spi_i2s_data_transmit(SPI0, spi0_send_array[send_n++]);
```

### 函数 spi\_i2s\_interrupt\_enable

函数spi\_i2s\_interrupt\_enable描述见下表:

表 3-462. 函数 spi\_i2s\_interrupt\_enable

函数名称	spi_i2s_interrupt_enable
函数原形	void spi_i2s_interrupt_enable(uint32_t spi_periph, uint8_t interrupt);
功能描述	使能外设SPIx/I2S0中断
先决条件	-
被调用函数	-
输入参数{in}	
<b>spi_periph</b>	外设SPIx
<i>SPIx</i>	x=0,1
输入参数{in}	
<b>interrupt</b>	SPI/I2S中断
<i>SPI_I2S_INT_TBE</i>	发送缓冲区空中断使能
<i>SPI_I2S_INT_RBNE</i>	接收缓冲区非空中断使能
<i>SPI_I2S_INT_ERR</i>	错误中断使能
输出参数{out}	

-	-
返回值	
-	-

例如：

```
/* enable SPI0 transmit buffer empty interrupt */
```

```
spi_i2s_interrupt_enable(SPI0, SPI_I2S_INT_TBE);
```

### 函数 spi\_i2s\_interrupt\_disable

函数spi\_i2s\_interrupt\_disable描述见下表：

**表 3-463. 函数 spi\_i2s\_interrupt\_disable**

函数名称	spi_i2s_interrupt_disable
函数原形	void spi_i2s_interrupt_disable(uint32_t spi_periph, uint8_t interrupt);
功能描述	禁能外设SPIx/I2S0中断
先决条件	-
被调用函数	-
输入参数{in}	
spi_periph	外设SPIx
SPIx	x=0,1
输入参数{in}	
interrupt	SPI/I2S中断
SPI_I2S_INT_TBE	发送缓冲区空中断使能
SPI_I2S_INT_RBNE	接收缓冲区非空中断使能
SPI_I2S_INT_ERR	错误中断使能
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* disable SPI0 transmit buffer empty interrupt */
```

```
spi_i2s_interrupt_disable(SPI0, SPI_I2S_INT_TBE);
```

### 函数 spi\_i2s\_interrupt\_flag\_get

函数spi\_i2s\_interrupt\_flag\_get描述见下表：

**表 3-464. 函数 spi\_i2s\_interrupt\_flag\_get**

函数名称	spi_i2s_interrupt_flag_get
函数原形	FlagStatus spi_i2s_interrupt_flag_get(uint32_t spi_periph, uint8_t interrupt);
功能描述	获取外设SPIx/I2S0中断状态



先决条件	-
被调用函数	-
输入参数{in}	
<b>spi_periph</b>	外设SPIx
<i>SPIx</i>	x=0,1
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 */
if(RESET != spi_i2s_interrupt_flag_get(SPI0, SPI_I2S_INT_FLAG_TBE)){
    while(RESET == spi_i2s_flag_get(SPI0, SPI_FLAG_TBE));
    spi_i2s_data_transmit(SPI0, spi0_send_array[send_n++]);
}

```

## 3.20. SYSCFG

章节 [3.20.1](#) 描述了SYSCFG的寄存器列表，章节 [3.20.2](#) 对SYSCFG库函数进行说明。

### 3.20.1. 外设寄存器说明

SYSCFG寄存器列表如下表所示：

表 3-465. 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_CFG2	系统配置寄存器2
SYSCFG_CPU_IRQ_LAT	IRQ延迟寄存器

### 3.20.2. 外设库函数说明

SYSCFG库函数列表如下表所示:

表 3-466. SYSCFG 库函数

库函数名称	库函数描述
syscfg_deinit	复位SYSCFG寄存器
syscfg_dma_remap_enable	使能DMA通道重映射
syscfg_dma_remap_disable	失能DMA通道重映射
syscfg_high_current_enable	使能PB9引脚大电流能力
syscfg_high_current_disable	失能PB9引脚大电流能力
syscfg_exti_line_config	配置GPIO引脚作为EXTI
syscfg_lock_config	将TIMER 0/14/15/16中断输入连接到所选参数
irq_latency_set	设置延迟值
syscfg_flag_get	得到SYSCFG_CFG2的标志位
syscfg_flag_clear	清除SYSCFG_CFG2的标志位

#### 函数 syscfg\_deinit

函数syscfg\_deinit描述见下表:

表 3-467. 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\_dma\_remap\_enable

函数syscfg\_dma\_remap\_enable描述见下表:

**表 3-468. 函数 syscfg\_dma\_remap\_enable**

函数名称	syscfg_dma_remap_enable
函数原形	void syscfg_dma_remap_enable(uint32_t syscfg_dma_remap);
功能描述	使能DMA通道重映射
先决条件	-
被调用函数	-
输入参数{in}	
syscfg_dma_remap	指定要重新映射的DMA通道
SYSCFG_DMA_REMAP_TIMER16	重新映射TIMER 16通道0和向通道1发送DMA请求
SYSCFG_DMA_REMAP_TIMER15	重新映射TIMER 15通道5和向通道3发送DMA请求
SYSCFG_DMA_REMAP_USART0RX	将AUARTAR0 RX DMA请求重新映射到通道4
SYSCFG_DMA_REMAP_USART0TX	将AUARTAR0 TX DMA请求重新映射到通道3
SYSCFG_DMA_REMAP_ADC	从通道0重新映射ADC DMA请求到通道1
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* enable DMA channel remap*/
```

```
syscfg_dma_remap_enable(SYSCFG_DMA_REMAP_TIMER16);
```

### 函数 syscfg\_dma\_remap\_disable

函数syscfg\_dma\_remap\_disable描述见下表:

**表 3-469. 函数 syscfg\_dma\_remap\_disable**

函数名称	syscfg_dma_remap_disable
函数原形	void syscfg_dma_remap_disable(uint32_t syscfg_dma_remap);

功能描述	失能DMA通道重映射
先决条件	-
被调用函数	-
输入参数{in}	
syscfg_dma_remap	指定要重新映射的DMA通道
SYSCFG_DMA_REMAP_TIMER16	重新映射TIMER 16通道0和向通道1发送DMA请求
SYSCFG_DMA_REMAP_TIMER15	重新映射TIMER 15通道5和向通道3发送DMA请求
SYSCFG_DMA_REMAP_USART0RX	将AUARTAR0 RX DMA请求重新映射到通道4
SYSCFG_DMA_REMAP_USART0TX	将AUARTAR0 TX DMA请求重新映射到通道3
SYSCFG_DMA_REMAP_ADC	从通道0重新映射ADC DMA请求到通道1
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* disable DMA channel remap*/
```

```
syscfg_dma_remap_disable(SYSCFG_DMA_REMAP_TIMER16);
```

## 函数 syscfg\_high\_current\_enable

函数syscfg\_high\_current\_enable描述见下表：

表 3-470. 函数 syscfg\_high\_current\_enable

函数名称	syscfg_high_current_enable
函数原形	void syscfg_high_current_enable(void);
功能描述	使能PB9引脚大电流能力
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 PB9 high current capability */
```

```
syscfg_high_current_enable();
```

### 函数 syscfg\_high\_current\_disable

函数syscfg\_high\_current\_disable描述见下表：

**表 3-471. 函数 syscfg\_high\_current\_disable**

函数名称	syscfg_high_current_disable
函数原形	void syscfg_high_current_disable(void);
功能描述	失能PB9引脚大电流能力
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 PB9 high current capability */
```

```
syscfg_high_current_disable();
```

### 函数 syscfg\_exti\_line\_config

函数syscfg\_exti\_line\_config描述见下表：

**表 3-472. 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	x=0..15(GPIOA, GPIOB, GPIOC, GPIOD, GPIOF)
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\_lock\_config

函数syscfg\_lock\_config描述见下表：

**表 3-473. 函数 syscfg\_lock\_config**

函数名称	syscfg_lock_config
函数原形	void syscfg_lock_config(uint32_t syscfg_lock);
功能描述	将TIMER 0/14/15/16中断输入连接到所选参数
先决条件	-
被调用函数	-
输入参数{in}	
exti_port	指定EXTI使用的GPIO端口
SYSCFG_LOCK_LOCKUP	Cortex-M4锁定输出连接到断开输入
SYSCFG_LOCK_SRAM_PARITY_ERROR	SRAM_PARITY校验错误连接到断开输入
SYSCFG_LOCK_LVD	LVD中断连接到断开输入
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* configure syscfg lock*/
```

```
syscfg_lock_config(SYSCFG_LOCK_LOCKUP);
```

### 函数 syscfg\_flag\_get

函数syscfg\_flag\_get描述见下表：

**表 3-474. 函数 syscfg\_flag\_get**

函数名称	syscfg_flag_get
函数原形	FlagStatus syscfg_flag_get(uint32_t syscfg_flag);

功能描述	校验SYSCFG_CFG2寄存器中指定的标志位是否置位
先决条件	-
被调用函数	-
输入参数{in}	
syscfg_flag	中断标志
SYSCFG_SRAM_P CEF	SRAM奇偶校验错误标志
输出参数{out}	
-	-
返回值	
FlagStatus	SET或者RESET

例如：

```
/* get syscfg flag */
```

```
FlagStatus status;
```

```
status = syscfg_flag_get(SYSCFG_SRAM_PCEF);
```

### 函数 syscfg\_flag\_clear

函数syscfg\_flag\_clear描述见下表：

表 3-475. 函数 syscfg\_flag\_clear

函数名称	syscfg_flag_gclear
函数原形	void syscfg_flag_clear(uint32_t syscfg_flag);
功能描述	清除SYSCFG_CFG2的标志位
先决条件	-
被调用函数	-
输入参数{in}	
syscfg_flag	中断标志
SYSCFG_SRAM_P CEF	SRAM奇偶校验错误标志
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* clear syscfg flag */
```

```
syscfg_flag_clear(SYSCFG_SRAM_PCEF);
```

**函数 syscfg\_compensation\_config**

函数syscfg\_compensation\_config描述见下表:

**表 3-476. 函数 syscfg\_compensation\_config**

函数名称	syscfg_compensation_config
函数原形	void syscfg_compensation_config(uint32_t syscfg_compensation);
功能描述	IO补偿单元设置
先决条件	-
被调用函数	-
输入参数{in}	
syscfg_compensation	IO补偿单元模式
SYSCFG_COMPENSATION_ENABLE	IO补偿单元打开
SYSCFG_COMPENSATION_DISABLE	IO补偿单元关闭
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* enable the I/O compensation cell */
```

```
syscfg_compensation_config (SYSCFG_COMPENSATION_ENABLE);
```

**函数 syscfg\_cps\_rdy\_flag\_get**

函数syscfg\_cps\_rdy\_flag\_get描述见下表:

**表 3-477. 函数 syscfg\_cps\_rdy\_flag\_get**

函数名称	syscfg_cps_rdy_flag_get
函数原形	FlagStatus syscfg_cps_rdy_flag_get(void);
功能描述	IO补偿单元是否准备就绪
先决条件	-
被调用函数	-
输入参数{in}	
-	-
输出参数{out}	
-	-
返回值	
FlagStatus	SET或者RESET

例如:



```
/* check if the I/O compensation cell ready flag is set or not */
```

```
FlagStatus ready_flag;
```

```
ready_flag = syscfg_cps_rdy_flag_get ();
```

## 3.21. 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(TIMERx, x=1, 2)，通用定时器L2(TIMER13)，通用定时器L3(TIMER14)，通用定时器L4(TIMERx, x=15, 16)，基本定时器(TIMER5)，不同类型的定时器具体功能有所差别。章节[3.21.1](#)描述了TIMER的寄存器列表，章节[3.21.2](#)对TIMER库函数进行说明。

### 3.21.1. 外设寄存器说明

TIMER寄存器列表如下表所示：

**表 3-478. TIMER 寄存器**

寄存器名称	寄存器描述
TIMERx_CTL0(TIMERx, x=0, 1, 2, 5, 13, 14, 15, 16)	控制寄存器 0
TIMERx_CTL1(TIMERx, x=0, 1, 2, 5, 13, 14, 15, 16)	控制寄存器 1
TIMERx_SMCFG(TIMERx, x=0, 1, 2, 14)	从模式配置寄存器
TIMERx_DMAINTEN(TIMERx, x=0, 1, 2, 5, 13, 14, 15, 16)	DMA 和中断使能寄存器
TIMERx_INTF(TIMERx, x=0, 1, 2, 5, 13, 14, 15, 16)	中断标志寄存器
TIMERx_SWEVG(TIMERx, x=0, 1, 2, 5, 13, 14, 15, 16)	软件事件产生寄存器
TIMERx_CHCTL0(TIMERx, x=0, 1, 2, 13, 14, 15, 16)	通道控制寄存器 0
TIMERx_CHCTL1(TIMERx, x=0, 1, 2)	通道控制寄存器 1
TIMERx_CHCTL2(TIMERx, x=0, 1, 2, 13, 14, 15, 16)	通道控制寄存器 2
TIMERx_CNT(TIMERx, x=0, 1, 2, 5, 13, 14, 15, 16)	计数器寄存器
TIMERx_PSC(TIMERx, x=0, 1, 2, 5, 13, 14, 15, 16)	预分频寄存器
TIMERx_CAR(TIMERx, x=0, 1, 2, 5, 13, 14, 15, 16)	计数器自动重载寄存器
TIMERx_CREP(TIMERx, x=0, 5, 14, 15, 16)	重复计数寄存器
TIMERx_CH0CV(TIMERx, x=0, 1, 2, 13, 14, 15, 16)	通道 0 捕获/比较寄存器
TIMERx_CH1CV(TIMERx, x=0, 1, 2, 14)	通道 1 捕获/比较寄存器
TIMERx_CH2CV(TIMERx, x=0, 1, 2)	通道 2 捕获/比较寄存器
TIMERx_CH3CV(TIMERx, x=0, 1, 2)	通道 3 捕获/比较寄存器
TIMERx_IRMP(TIMERx, x=13)	通道输入重映射寄存器
TIMERx_CCHP(TIMERx, x=0, 1, 2, 14, 15, 16)	互补通道保护寄存器
TIMERx_DMACFG(TIMERx, x=0, 1, 2, 14, 15, 16)	DMA 配置寄存器
TIMERx_DMATB(TIMERx, x=0, 1, 2, 14, 15, 16)	DMA 发送缓冲区寄存器
TIMERx_CFG(TIMERx, x=0, 1, 2, 13, 14, 15, 16)	配置寄存器

### 3.21.2. 外设库函数说明

TIMER库函数列表如下表所示：

**表 3-479. 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_ocpre_clear_source_config	配置 TIMERx 的 OCPRE 清除源选择
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
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
timer_break_config	配置中止功能
timer_break_enable	使能 TIMERx 的中止功能

库函数名称	库函数描述
timer_break_disable	禁能 TIMERx 的中止功能
timer_automatic_output_enable	自动输出使能
timer_automatic_output_disable	自动输出禁能
timer_primary_output_config	所有的通道输出使能
timer_channel_control_shadow_config	通道换相控制影子寄存器配置
timer_channel_control_shadow_update_config	通道换相控制影子寄存器更新控制
timer_channel_output_struct_para_init	将 TIMER 通道输出参数结构体中所有参数初始化为默认值
timer_channel_output_config	外设 TIMERx 的通道输出配置
timer_channel_output_mode_config	配置外设 TIMERx 通道输出比较模式
timer_channel_output_pulse_value_config	配置外设 TIMERx 的通道输出比较值
timer_channel_output_shadow_config	配置 TIMERx 通道输出比较影子寄存器功能
timer_channel_output_fast_config	配置 TIMERx 通道输出比较快速功能
timer_channel_output_clear_config	配置 TIMERx 的通道输出比较清 0 功能
timer_channel_output_polarity_config	通道输出极性配置
timer_channel_complementary_output_polarity_config	互补通道输出极性配置
timer_channel_output_state_config	配置通道状态
timer_channel_complementary_output_state_config	配置互补通道输出状态
timer_channel_input_struct_para_init	将 TIMER 通道输入参数结构体中所有参数初始化为默认值
timer_input_capture_config	配置 TIMERx 输入捕获参数
timer_channel_input_capture_prescaler_config	配置 TIMERx 通道输入捕获预分频值
timer_channel_capture_value_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
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

库函数名称	库函数描述
timer_internal_clock_config	TIMERx 配置为内部时钟模式
timer_internal_trigger_as_external_clock_config	配置 TIMERx 的内部触发为时钟源
timer_external_trigger_as_external_clock_config	配置 TIMERx 的外部触发作为时钟源
timer_external_clock_mode0_config	配置 TIMERx 外部时钟模式 0, ETI 作为时钟源
timer_external_clock_mode1_config	配置 TIMERx 外部时钟模式 1
timer_external_clock_mode1_disable	禁能 TIMERx 外部时钟模式 1
timer_channel_remap_config	配置 TIMERx 通道重映射功能
timer_write_chxval_register_config	配置 TIMERx 写 CHxVAL 选择位
timer_output_value_selection_config	配置定时器输出值选择

### 结构体 timer\_parameter\_struct

表 3-480. 结构体类型 timer\_parameter\_struct

成员名称	功能描述
prescaler	预分频值 (0~65535)
alignedmode	对齐模式 (TIMER_COUNTER_EDGE, TIMER_COUNTER_CENTER_DOWN, TIMER_COUNTER_CENTER_UP, TIMER_COUNTER_CENTER_BOTH)
counterdirection	计数方向 (TIMER_COUNTER_UP, TIMER_COUNTER_DOWN)
period	周期 (0~65535)
clockdivision	时钟分频因子 (TIMER_CKDIV_DIV1, TIMER_CKDIV_DIV2, TIMER_CKDIV_DIV4)
repetitioncounter	重复计数器值 (0~255)

### 结构体 timer\_break\_parameter\_struct

表 3-481. 结构体类型 timer\_break\_parameter\_struct

成员名称	功能描述
runoffstate	运行模式下“关闭状态”配置 (TIMER_ROS_STATE_ENABLE, TIMER_ROS_STATE_DISABLE)
ideloffstate	空闲模式下“关闭状态”配置 (TIMER_IOS_STATE_ENABLE, TIMER_IOS_STATE_DISABLE)
deadtime	死区时间 (0~255)
breakpolarity	中止信号极性 (TIMER_BREAK_POLARITY_LOW, TIMER_BREAK_POLARITY_HIGH)
outputautostate	自动输出使能 (TIMER_OUTAUTO_ENABLE, TIMER_OUTAUTO_DISABLE)
protectmode	互补寄存器保护控制 (TIMER_CCHP_PROT_OFF, TIMER_CCHP_PROT_0, TIMER_CCHP_PROT_1, TIMER_CCHP_PROT_2)
breakstate	中止使能 (TIMER_BREAK_ENABLE, TIMER_BREAK_DISABLE)

### 结构体 timer\_oc\_parameter\_struct

表 3-482. 结构体类型 timer\_oc\_parameter\_struct

成员名称	功能描述
outputstate	通道输出状态 (TIMER_CCX_ENABLE, TIMER_CCX_DISABLE)
outputnstate	互补通道输出状态 (TIMER_CCXN_ENABLE, TIMER_CCXN_DISABLE)
ocpolarity	通道输出极性 (TIMER_OC_POLARITY_HIGH, TIMER_OC_POLARITY_LOW)
ocnpolarity	互补通道输出极性 (TIMER_OCN_POLARITY_HIGH, TIMER_OCN_POLARITY_LOW)
ocidlestate	空闲状态下通道输出 (TIMER_OC_IDLE_STATE_LOW, TIMER_OC_IDLE_STATE_HIGH)
ocnidlestate	空闲状态下互补通道输出 (TIMER_OCN_IDLE_STATE_LOW, TIMER_OCN_IDLE_STATE_HIGH)

### 结构体 timer\_ic\_parameter\_struct

表 3-483. 结构体类型 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 3-484. 函数 timer\_deinit

函数名称	timer_deinit
函数原型	void timer_deinit(uint32_t timer_periph);
功能描述	复位外设 TIMEx
先决条件	-
被调用函数	rcu_periph_reset_enable / rcu_periph_reset_disable
输入参数{in}	
timer_periph	TIMER 外设
TIMEx(x=0..2, 5, 13..16)	TIMER 外设选择
输出参数{out}	
-	-
返回值	

-	-
---	---

例如:

```
/* reset TIMER0 */
```

```
timer_deinit(TIMER0);
```

### 函数 timer\_struct\_para\_init

函数timer\_struct\_para\_init描述见下表:

**表 3-485. 函数 timer\_struct\_para\_init**

函数名称	timer_struct_para_init
函数原型	void timer_struct_para_init(timer_parameter_struct* initpara);
功能描述	将 TIMER 初始化参数结构体中所有参数初始化为默认值
先决条件	-
被调用函数	-
输入参数{in}	
initpara	TIMER 初始化结构体, 结构体成员参考 <a href="#">表 3-480. 结构体类型 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 3-486. 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 TIMEx
先决条件	-
被调用函数	-
输入参数{in}	
timer_periph	TIMER 外设
TIMEx(x=0..2, 5, 13..16)	TIMER 外设选择

输入参数{in}	
initpara	TIMER 初始化结构体，结构体成员参考 <a href="#">表 3-480. 结构体类型 timer_parameter_struct</a>
输出参数{out}	
-	-
返回值	
-	-

例如：

```

/* initialize TIMER0 */

timer_parameter_struct timer_initpara;

timer_initpara.prescaler      = 107;

timer_initpara.alignedmode    = TIMER_COUNTER_EDGE;

timer_initpara.counterdirection = TIMER_COUNTER_UP;

timer_initpara.period         = 999;

timer_initpara.clockdivision   = TIMER_CKDIV_DIV1;

timer_initpara.repetitioncounter = 1;

timer_init(TIMER0, &timer_initpara);

```

### 函数 timer\_enable

函数timer\_enable描述见下表：

**表 3-487. 函数 timer\_enable**

函数名称	timer_enable
函数原型	void timer_enable(uint32_t timer_periph);
功能描述	使能外设 TIMERx
先决条件	-
被调用函数	-
输入参数{in}	
timer_periph	TIMER 外设
TIMERx(x=0..2, 5, 13..16)	TIMER 外设选择
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* enable TIMER0 */
```

```
timer_enable (TIMER0);
```

### 函数 timer\_disable

函数timer\_disable描述见下表：

**表 3-488. 函数 timer\_disable**

函数名称	timer_disable
函数原型	void timer_disable(uint32_t timer_periph);
功能描述	禁能外设 TIMERx
先决条件	-
被调用函数	-
输入参数{in}	
timer_periph	TIMER 外设
TIMERx(x=0..2, 5, 13..16)	TIMER 外设选择
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* disable TIMER0 */
```

```
timer_disable (TIMER0);
```

### 函数 timer\_auto\_reload\_shadow\_enable

函数timer\_auto\_reload\_shadow\_enable描述见下表：

**表 3-489. 函数 timer\_auto\_reload\_shadow\_enable**

函数名称	timer_auto_reload_shadow_enable
函数原型	void timer_auto_reload_shadow_enable(uint32_t timer_periph);
功能描述	TIMERx 自动重载影子使能
先决条件	-
被调用函数	-
输入参数{in}	
timer_periph	TIMER 外设
TIMERx(x=0..2, 5, 13..16)	TIMER 外设选择
输出参数{out}	
-	-
返回值	



-	-
---	---

例如:

```
/* enable the TIMER0 auto reload shadow function */
```

```
timer_auto_reload_shadow_enable (TIMER0);
```

### 函数 timer\_auto\_reload\_shadow\_disable

函数timer\_auto\_reload\_shadow\_disable描述见下表:

**表 3-490. 函数 timer\_auto\_reload\_shadow\_disable**

函数名称	timer_auto_reload_shadow_disable
函数原型	void timer_auto_reload_shadow_disable (uint32_t timer_periph);
功能描述	TIMERx 自动重载影子禁能
先决条件	-
被调用函数	-
输入参数{in}	
timer_periph	TIMER 外设
TIMERx(x=0..2, 5, 13..16)	TIMER 外设选择
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* disable the TIMER0 auto reload shadow function */
```

```
timer_auto_reload_shadow_disable (TIMER0);
```

### 函数 timer\_update\_event\_enable

函数timer\_update\_event\_enable描述见下表:

**表 3-491. 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}	
timer_periph	TIMER 外设
TIMERx(x=0..2, 5, 13..16)	TIMER 外设选择

输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* enable TIMER0 the update event */
```

```
timer_update_event_enable (TIMER0);
```

### 函数 timer\_update\_event\_disable

函数timer\_update\_event\_disable描述见下表：

**表 3-492. 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(x=0..2, 5, 13..16)	TIMER 外设选择
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* disable TIMER0 the update event */
```

```
timer_update_event_disable (TIMER0);
```

### 函数 timer\_counter\_alignment

函数timer\_counter\_alignment描述见下表：

**表 3-493. 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}	

<b>timer_periph</b>	TIMER 外设
<i>TIMERx(x=0..2)</i>	TIMER 外设选择
<b>输入参数{in}</b>	
<b>aligned</b>	对齐模式
<i>TIMER_COUNTER_EDGE</i>	无中央对齐计数模式(边沿对齐模式)，DIR 位指定了计数方向
<i>TIMER_COUNTER_CENTER_DOWN</i>	中央对齐向下计数置 1 模式。计数器在中央计数模式计数，通道被配置在输出模式（TIMERx_CHCTL0 寄存器中 CHxMS=00），只有在向下计数时，通道的比较中断标志置 1
<i>TIMER_COUNTER_CENTER_UP</i>	中央对齐向上计数置 1 模式。计数器在中央计数模式计数，通道被配置在输出模式（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
<i>TIMER_COUNTER_CENTER_BOTH</i>	中央对齐上下计数置 1 模式。计数器在中央计数模式计数，通道被配置在输出模式（TIMERx_CHCTL0 寄存器中 CHxMS=00），在向上和向下计数时，通道的比较中断标志都会置 1
<b>输出参数{out}</b>	
-	-
<b>返回值</b>	
-	-

例如：

```
/* set TIMER0 counter center-aligned and counting up assert mode */
```

```
timer_counter_alignment (TIMER0, TIMER_COUNTER_CENTER_UP);
```

### 函数 timer\_counter\_up\_direction

函数timer\_counter\_up\_direction描述见下表：

**表 3-494. 函数 timer\_counter\_up\_direction**

<b>函数名称</b>	timer_counter_up_direction
<b>函数原型</b>	void timer_counter_up_direction(uint32_t timer_periph);
<b>功能描述</b>	设置外设 TIMERx 向上计数
<b>先决条件</b>	计数器设置为无中央对齐计数模式（边沿对齐模式）
<b>被调用函数</b>	-
<b>输入参数{in}</b>	
<b>timer_periph</b>	TIMER 外设
<i>TIMERx(x=0..2)</i>	TIMER 外设选择
<b>输出参数{out}</b>	
-	-
<b>返回值</b>	
-	-

例如：

```
/* set TIMER0 counter up direction */
```

```
timer_counter_up_direction (TIMER0);
```

### 函数 timer\_counter\_down\_direction

函数timer\_counter\_down\_direction描述见下表：

**表 3-495. 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
TIMERx(x=0..2)	TIMER 外设选择
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* set TIMER0 counter down direction */
```

```
timer_counter_down_direction (TIMER0);
```

### 函数 timer\_prescaler\_config

函数timer\_prescaler\_config描述见下表：

**表 3-496. 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
TIMERx(x=0..2, 5, 13..16)	TIMER 外设选择
输入参数{in}	
prescaler	预分频值，0~65535
输入参数{in}	

<b>pscreload</b>	预分频值加载模式
<i>TIMER_PSC_RELO AD_NOW</i>	预分频值立即加载
<i>TIMER_PSC_RELO AD_UPDATE</i>	预分频值在下次更新事件发生时加载
<b>输出参数{out}</b>	
-	-
<b>返回值</b>	
-	-

例如：

```
/* configure TIMER0 prescaler */
```

```
timer_prescaler_config (TIMER0, 3000, TIMER_PSC_RELOAD_NOW);
```

### 函数 timer\_repetition\_value\_config

函数timer\_repetition\_value\_config描述见下表：

**表 3-497. 函数 timer\_repetition\_value\_config**

<b>函数名称</b>	timer_repetition_value_config
<b>函数原型</b>	void timer_repetition_value_config(uint32_t timer_periph, uint16_t repetition);
<b>功能描述</b>	配置外设 TIMERx 的重复计数器
<b>先决条件</b>	-
<b>被调用函数</b>	-
<b>输入参数{in}</b>	
<b>timer_periph</b>	TIMER 外设
<i>TIMERx(x=0, 15, 16)</i>	TIMER 外设选择
<b>输入参数{in}</b>	
<b>repetition</b>	重复计数器值，取值范围 0~255
<b>输出参数{out}</b>	
-	-
<b>返回值</b>	
-	-

例如：

```
/* configure TIMER0 repetition register value */
```

```
timer_repetition_value_config (TIMER0, 98);
```

### 函数 timer\_autoreload\_value\_config

函数timer\_autoreload\_value\_config描述见下表：

表 3-498. 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(x=0..2, 5, 13..16)	TIMER 外设选择
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 TIMER autoreload register value */
```

```
timer_autoreload_value_config (TIMER0, 3000);
```

### 函数 timer\_counter\_value\_config

函数timer\_counter\_value\_config描述见下表:

表 3-499. 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(x=0..2, 5, 13..16)	TIMER 外设选择
输入参数{in}	
counter	计数器值 (0-65535)
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* configure TIMER0 counter register value */
```

```
timer_counter_value_config (TIMER0, 3000);
```

### 函数 timer\_counter\_read

函数timer\_counter\_read描述见下表：

表 3-500. 函数 timer\_counter\_read

函数名称	timer_counter_read
函数原型	uint32_t timer_counter_read(uint32_t timer_periph);
功能描述	读取外设 TIMERx 的计数器值
先决条件	-
被调用函数	-
输入参数{in}	
timer_periph	TIMER 外设
TIMERx(x=0..2, 5, 13..16)	TIMER 外设选择
输出参数{out}	
-	-
返回值	
uint32_t	外设 TIMERx 的计数器值 (0~65535)

例如：

```
/* read TIMER0 counter value */
```

```
uint32_t i = 0;
```

```
i = timer_counter_read (TIMER0);
```

### 函数 timer\_prescaler\_read

函数timer\_prescaler\_read描述见下表：

表 3-501. 函数 timer\_prescaler\_read

函数名称	timer_prescaler_read
函数原型	uint16_t timer_prescaler_read(uint32_t timer_periph);
功能描述	读取外设 TIMERx 的预分频器值
先决条件	-
被调用函数	-
输入参数{in}	
timer_periph	TIMER 外设
TIMERx(x=0..2, 5, 13..16)	TIMER 外设选择
输出参数{out}	
-	-

返回值	
<b>uint16_t</b>	外设 TIMERx 的预分频器值 (0~65535)

例如:

```
/* read TIMER0 prescaler value */

uint16_t i = 0;

i = timer_prescaler_read (TIMER0);
```

### 函数 timer\_single\_pulse\_mode\_config

函数 timer\_single\_pulse\_mode\_config 描述见下表:

表 3-502. 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}	
<b>timer_periph</b>	TIMER 外设
TIMERx(x=0..2, 5, 14..16)	TIMER 外设选择
输入参数{in}	
<b>spmode</b>	脉冲模式
TIMER_SP_MODE_SINGLE	单脉冲模式计数
TIMER_SP_MODE_REPETITIVE	重复模式计数
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* configure TIMER0 single pulse mode */

timer_single_pulse_mode_config (TIMER0, TIMER_SP_MODE_SINGLE);
```

### 函数 timer\_update\_source\_config

函数 timer\_update\_source\_config 描述见下表:

表 3-503. 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 TIMEx 的更新源
先决条件	-
被调用函数	-
输入参数{in}	
timer_periph	TIMER 外设
TIMEx(x=0..2, 5, 13..16)	TIMER 外设选择
输入参数{in}	
update	更新源
TIMER_UPDATE_SRC_GLOBAL	下述任一事件产生更新中断或 DMA 请求： – UPG 位被置 1 – 计数器溢出/下溢 – 从模式控制器产生的更新
TIMER_UPDATE_SRC_REGULAR	只有计数器溢出/下溢才产生更新中断或 DMA 请求
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* configure TIMER update only by counter overflow/underflow */
```

```
timer_update_source_config (TIMER0, TIMER_UPDATE_SRC_REGULAR);
```

### 函数 timer\_ocpre\_clear\_source\_config

函数timer\_ocpre\_clear\_source\_config描述见下表：

表 3-504. 函数 timer\_ocpre\_clear\_source\_config

函数名称	timer_ocpre_clear_source_config
函数原型	void timer_ocpre_clear_source_config (uint32_t timer_periph, uint8_t ocpreclear);
功能描述	配置外设 TIMEx 的 OCPRE 清除源选择
先决条件	-
被调用函数	-
输入参数{in}	
timer_periph	TIMER 外设
TIMEx(x=0..2)	TIMER 外设选择
输入参数{in}	
ocpreclear	清除源
TIMER_OCPRE_CLEAR_SOURCE_CLEAR	OCPRE_CLR_INT 连接到 OCPRE_CLR 输入

<i>R</i>	
<i>TIMER_OCPRE_CLEAR_SOURCE_ETIF</i>	OCPRE_CLR_INT 连接到 ETIF
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* configure TIMER0 OCPRE_CLR_INT is connected to the OCPRE_CLR input */
timer_ocpre_clear_source_config(TIMER0, TIMER_OCPRE_CLEAR_SOURCE_CLR);
```

### 函数 timer\_interrupt\_enable

函数timer\_interrupt\_enable描述见下表：

表 3-505. 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，TIMERx (x=0..2, 5, 13..16)
TIMER_INT_CH0	通道0比较/捕获中断，TIMERx(x=0..2, 13..16)
TIMER_INT_CH1	通道 1 比较/捕获中断，TIMERx(x=0..2, 14)
TIMER_INT_CH2	通道 2 比较/捕获中断，TIMERx(x=0..2)
TIMER_INT_CH3	通道 3 比较/捕获中断，TIMERx(x=0..2)
TIMER_INT_CMT	换相更新中断，TIMERx (x=0, 14..16)
TIMER_INT_TRG	触发中断，TIMERx(x=0..2, 14)
TIMER_INT_BRK	中止中断，TIMERx(x=0, 14..16)
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* enable the TIMER0 update interrupt */
```

timer\_interrupt\_enable (TIMER0, TIMER\_INT\_UP);

### 函数 timer\_interrupt\_disable

函数timer\_interrupt\_disable描述见下表:

表 3-506. 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	更新中断, TIMERx (x=0..2, 5, 13..16)
TIMER_INT_CH0	通道0比较/捕获中断, TIMERx(x=0..2, 13..16)
TIMER_INT_CH1	通道 1 比较/捕获中断, TIMERx(x=0..2, 14)
TIMER_INT_CH2	通道 2 比较/捕获中断, TIMERx(x=0..2)
TIMER_INT_CH3	通道 3 比较/捕获中断, TIMERx(x=0..2)
TIMER_INT_CMT	换相更新中断, TIMERx (x=0, 14..16)
TIMER_INT_TRG	触发中断, TIMERx(x=0..2, 14)
TIMER_INT_BRK	中止中断, TIMERx(x=0, 14..16)
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* disable the TIMER0 update interrupt */
```

```
timer_interrupt_disable (TIMER0, TIMER_INT_UP);
```

### 函数 timer\_interrupt\_flag\_get

函数timer\_interrupt\_flag\_get描述见下表:

表 3-507. 函数 timer\_interrupt\_flag\_get

函数名称	timer_interrupt_flag_get
函数原型	FlagStatus timer_interrupt_flag_get(uint32_t timer_periph, uint32_t interrupt);
功能描述	获取外设 TIMERx 中断标志
先决条件	-

被调用函数	-
输入参数{in}	
timer_periph	TIMER 外设
TIMERx	参考具体参数
输入参数{in}	
interrupt	中断源
TIMER_INT_FLAG_UP	更新中断, TIMERx (x=0..2, 5, 13..16)
TIMER_INT_FLAG_CH0	通道0比较/捕获中断, TIMERx(x=0..2, 13..16)
TIMER_INT_FLAG_CH1	通道 1 比较/捕获中断, TIMERx(x=0..2, 14)
TIMER_INT_FLAG_CH2	通道 2 比较/捕获中断, TIMERx(x=0..2)
TIMER_INT_FLAG_CH3	通道 3 比较/捕获中断, TIMERx(x=0..2)
TIMER_INT_FLAG_CMT	换相更新中断, TIMERx (x=0, 14..16)
TIMER_INT_FLAG_TRG	触发中断, TIMERx(x=0..2, 14)
TIMER_INT_FLAG_BRK	中止中断, TIMERx(x=0, 14..16)
输出参数{out}	
-	-
返回值	
FlagStatus	SET 或者 RESET

例如:

```
/* get TIMERO update interrupt flag */
```

```
FlagStatus Flag_interrupt = RESET;
```

```
Flag_interrupt = timer_interrupt_flag_get (TIMERO, TIMER_INT_FLAG_UP);
```

### 函数 timer\_interrupt\_flag\_clear

函数timer\_interrupt\_flag\_clear描述见下表:

表 3-508. 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}	
<b>timer_periph</b>	TIMER 外设
<i>TIMERx</i>	参考具体参数
输入参数{in}	
<b>interrupt</b>	中断源
<i>TIMER_INT_FLAG_UP</i>	更新中断, TIMERx (x=0..2, 5, 13..16)
<i>TIMER_INT_FLAG_CH0</i>	通道0比较/捕获中断, TIMERx(x=0..2, 13..16)
<i>TIMER_INT_FLAG_CH1</i>	通道 1 比较/捕获中断, TIMERx(x=0..2, 14)
<i>TIMER_INT_FLAG_CH2</i>	通道 2 比较/捕获中断, TIMERx(x=0..2)
<i>TIMER_INT_FLAG_CH3</i>	通道 3 比较/捕获中断, TIMERx(x=0..2)
<i>TIMER_INT_FLAG_CMT</i>	换相更新中断, TIMERx (x=0, 14..16)
<i>TIMER_INT_FLAG_TRG</i>	触发中断, TIMERx(x=0..2, 14)
<i>TIMER_INT_FLAG_BRK</i>	中止中断, TIMERx(x=0, 14..16)
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* clear TIMER0 update interrupt flag */
```

```
timer_interrupt_flag_clear (TIMER0, TIMER_INT_FLAG_UP);
```

### 函数 timer\_flag\_get

函数timer\_flag\_get描述见下表:

表 3-509. 函数 timer\_flag\_get

<b>函数名称</b>	timer_flag_get
<b>函数原型</b>	FlagStatus timer_flag_get(uint32_t timer_periph, uint32_t flag);
<b>功能描述</b>	获取外设 TIMERx 的状态标志
<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>flag</b>	状态标志
<i>TIMER_FLAG_UP</i>	更新标志, <i>TIMERx</i> ( <i>x</i> =0..2, 5, 13..16)
<i>TIMER_FLAG_CH0</i>	通道 0 比较/捕获标志, <i>TIMERx</i> ( <i>x</i> =0..2, 13..16)
<i>TIMER_FLAG_CH1</i>	通道 1 比较/捕获标志, <i>TIMERx</i> ( <i>x</i> =0..2, 14)
<i>TIMER_FLAG_CH2</i>	通道 2 比较/捕获标志, <i>TIMERx</i> ( <i>x</i> =0..2)
<i>TIMER_FLAG_CH3</i>	通道 3 比较/捕获标志, <i>TIMERx</i> ( <i>x</i> =0..2)
<i>TIMER_FLAG_CMT</i>	通道换相更新标志, <i>TIMERx</i> ( <i>x</i> =0, 14..16)
<i>TIMER_FLAG_TRG</i>	触发标志, <i>TIMERx</i> ( <i>x</i> =0..2, 14)
<i>TIMER_FLAG_BRK</i>	中止标志位, <i>TIMERx</i> ( <i>x</i> =0, 14..16)
<i>TIMER_FLAG_CH0</i> O	通道 0 捕获溢出标志, <i>TIMERx</i> ( <i>x</i> =0..2, 3..16)
<i>TIMER_FLAG_CH1</i> O	通道 1 捕获溢出标志, <i>TIMERx</i> ( <i>x</i> =0..2, 14)
<i>TIMER_FLAG_CH2</i> O	通道 2 捕获溢出标志, <i>TIMERx</i> ( <i>x</i> =0..2)
<i>TIMER_FLAG_CH3</i> O	通道 3 捕获溢出标志, <i>TIMERx</i> ( <i>x</i> =0..2)
输出参数{out}	
-	-
返回值	
<b>FlagStatus</b>	SET 或者 RESET

例如:

```
/* get TIMER0 update flags */
```

```
FlagStatus Flag_status = RESET;
```

```
Flag_status = timer_flag_get (TIMER0, TIMER_FLAG_UP);
```

### 函数 timer\_flag\_clear

函数timer\_flag\_clear描述见下表:

表 3-510. 函数 timer\_flag\_clear

<b>函数名称</b>	timer_flag_clear
<b>函数原型</b>	void timer_flag_clear(uint32_t timer_periph, uint32_t flag);
<b>功能描述</b>	清除外设 <i>TIMERx</i> 状态标志
<b>先决条件</b>	-
<b>被调用函数</b>	-
输入参数{in}	
<b>timer_periph</b>	<i>TIMER</i> 外设
<i>TIMERx</i>	参考具体参数
输入参数{in}	

flag	状态标志
<i>TIMER_FLAG_UP</i>	更新标志, <i>TIMERx</i> ( <i>x</i> =0..2, 5, 13..16)
<i>TIMER_FLAG_CH0</i>	通道 0 比较/捕获标志, <i>TIMERx</i> ( <i>x</i> =0..2, 13..16)
<i>TIMER_FLAG_CH1</i>	通道 1 比较/捕获标志, <i>TIMERx</i> ( <i>x</i> =0..2, 14)
<i>TIMER_FLAG_CH2</i>	通道 2 比较/捕获标志, <i>TIMERx</i> ( <i>x</i> =0..2)
<i>TIMER_FLAG_CH3</i>	通道 3 比较/捕获标志, <i>TIMERx</i> ( <i>x</i> =0..2)
<i>TIMER_FLAG_CMT</i>	通道换相更新标志, <i>TIMERx</i> ( <i>x</i> =0, 14..16)
<i>TIMER_FLAG_TRG</i>	触发标志, <i>TIMERx</i> ( <i>x</i> =0..2, 14)
<i>TIMER_FLAG_BRK</i>	中止标志位, <i>TIMERx</i> ( <i>x</i> =0, 14..16)
<i>TIMER_FLAG_CH0</i> O	通道 0 捕获溢出标志, <i>TIMERx</i> ( <i>x</i> =0..2, 13..16)
<i>TIMER_FLAG_CH1</i> O	通道 1 捕获溢出标志, <i>TIMERx</i> ( <i>x</i> =0..2, 14)
<i>TIMER_FLAG_CH2</i> O	通道 2 捕获溢出标志, <i>TIMERx</i> ( <i>x</i> =0..2)
<i>TIMER_FLAG_CH3</i> O	通道 3 捕获溢出标志, <i>TIMERx</i> ( <i>x</i> =0..2)
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* clear TIMER0 update flags */
```

```
timer_flag_clear (TIMER0, TIMER_FLAG_UP);
```

## 函数 timer\_dma\_enable

函数timer\_dma\_enable描述见下表:

表 3-511. 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 <i>TIMERx</i> 的 DMA 使能
先决条件	-
被调用函数	-
输入参数{in}	
timer_periph	<i>TIMER</i> 外设
<i>TIMERx</i>	参考具体参数
输入参数{in}	
dma	DMA 源
<i>TIMER_DMA_UPD</i>	更新 DMA 请求, <i>TIMERx</i> ( <i>x</i> =0..2, 5, 14..16)

<i>TIMER_DMA_CH0</i> <i>D</i>	通道 0 比较/捕获 DMA 请求, TIMERx(x=0..2, 14..16)
<i>TIMER_DMA_CH1</i> <i>D</i>	通道 1 比较/捕获 DMA 请求, TIMERx(x=0..2, 4)
<i>TIMER_DMA_CH2</i> <i>D</i>	通道 2 比较/捕获 DMA 请求, TIMERx(x=0..2)
<i>TIMER_DMA_CH3</i> <i>D</i>	通道 3 比较/捕获 DMA 请求, TIMERx(x=0..2)
<i>TIMER_DMA_CMT</i> <i>D</i>	换相 DMA 更新请求, TIMERx(x=0, 14)
<i>TIMER_DMA_TRG</i> <i>D</i>	触发 DMA 请求使能, TIMERx(x=0..2, 14)
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* enable the TIMER0 update DMA */
```

```
timer_dma_enable (TIMER0, TIMER_DMA_UPD);
```

## 函数 timer\_dma\_disable

函数timer\_dma\_disable描述见下表:

表 3-512. 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, 5, 14..16)
<i>TIMER_DMA_CH0</i> <i>D</i>	通道 0 比较/捕获 DMA 请求, TIMERx(x=0..2, 14..16)
<i>TIMER_DMA_CH1</i> <i>D</i>	通道 1 比较/捕获 DMA 请求, TIMERx(x=0..2, 14)
<i>TIMER_DMA_CH2</i> <i>D</i>	通道 2 比较/捕获 DMA 请求, TIMERx(x=0..2)



<i>TIMER_DMA_CH3</i> <i>D</i>	通道 3 比较/捕获 DMA 请求, <i>TIMERx</i> ( <i>x</i> =0..2)
<i>TIMER_DMA_CMT</i> <i>D</i>	换相 DMA 更新请求, <i>TIMERx</i> ( <i>x</i> =0, 14)
<i>TIMER_DMA_TRG</i> <i>D</i>	触发 DMA 请求使能, <i>TIMERx</i> ( <i>x</i> =0..2, 14)
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* disable the TIMER0 update DMA */
```

```
timer_dma_disable (TIMER0, TIMER_DMA_UPD);
```

### 函数 timer\_channel\_dma\_request\_source\_select

函数 timer\_channel\_dma\_request\_source\_select 描述见下表:

表 3-513. 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 <i>TIMERx</i> 的通道 DMA 请求源选择
先决条件	-
被调用函数	-
输入参数{in}	
<b>timer_periph</b>	<i>TIMER</i> 外设
<i>TIMERx</i> ( <i>x</i> =0..2, 14,..16)	<i>TIMER</i> 外设选择
输入参数{in}	
<b>dma_request</b>	通道的 DMA 请求源选择
<i>TIMER_DMAREQUEST_CHANNELEVENT</i>	当通道捕获/比较事件发生时, 发送通道 <i>n</i> 的 DMA 请求
<i>TIMER_DMAREQUEST_UPDATEEVENT</i>	当更新事件发生, 发送通道 <i>n</i> 的 DMA 请求
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* TIMER0 channel DMA request of channel n is sent when channel y event occurs */
```

```
timer_channel_dma_request_source_select (TIMER0,  
TIMER_DMAREQUEST_CHANNELEVENT);
```

### 函数 timer\_dma\_transfer\_config

函数timer\_dma\_transfer\_config描述见下表：

表 3-514. 函数 timer\_dma\_transfer\_config

函数名称	timer_dma_transfer_config
函数原型	void timer_dma_transfer_config(uint32_t timer_periph, uint32_t dma_baseaddr, uint32_t dma_lenth);
功能描述	配置外设 TIMERx 的 DMA 模式
先决条件	-
被调用函数	-
输入参数{in}	
timer_periph	TIMER 外设
TIMERx(x=0..2, 14..16)	定时器外设选择
输入参数{in}	
dma_baseaddr	DMA 传输起始地址
TIMER_DMACFG_DMATA_CTL0	DMA 传输起始地址：TIMER_DMACFG_DMATA_CTL0, TIMERx(x=0..2, 14..16)
TIMER_DMACFG_DMATA_CTL1	DMA 传输起始地址：TIMER_DMACFG_DMATA_CTL1, TIMERx(x=0..2, 14..16)
TIMER_DMACFG_DMATA_SMCFG	DMA 传输起始地址：TIMER_DMACFG_DMATA_SMCFG, TIMERx(x=0..2, 14)
TIMER_DMACFG_DMATA_DMAINTEN	DMA 传输起始地址：TIMER_DMACFG_DMATA_DMAINTEN, TIMERx(x=0..2, 14..16)
TIMER_DMACFG_DMATA_INTF	DMA 传输起始地址：TIMER_DMACFG_DMATA_INTF, TIMERx(x=0..2, 14..16)
TIMER_DMACFG_DMATA_SWEVG	DMA 传输起始地址：TIMER_DMACFG_DMATA_SWEVG, TIMERx(x=0..2, 14..16)
TIMER_DMACFG_DMATA_CHCTL0	DMA 传输起始地址：TIMER_DMACFG_DMATA_CHCTL0, TIMERx(x=0..2, 14..16)
TIMER_DMACFG_DMATA_CHCTL1	DMA 传输起始地址：TIMER_DMACFG_DMATA_CHCTL1, TIMERx(x=0..2)
TIMER_DMACFG_DMATA_CHCTL2	DMA 传输起始地址：TIMER_DMACFG_DMATA_CHCTL2, TIMERx (x=0..2, 14..16)
TIMER_DMACFG_	DMA 传输起始地址：TIMER_DMACFG_DMATA_CNT, TIMERx (x=0..2,

<i>DMATA_CNT</i>	14..16)
<i>TIMER_DMACFG_DMATA_PSC</i>	DMA 传输起始地址: <i>TIMER_DMACFG_DMATA_PSC</i> , <i>TIMERx</i> ( <i>x</i> =0..2, 14..16)
<i>TIMER_DMACFG_DMATA_CAR</i>	DMA 传输起始地址: <i>TIMER_DMACFG_DMATA_CAR</i> , <i>TIMERx</i> ( <i>x</i> =0..2, 14..16)
<i>TIMER_DMACFG_DMATA_CREP</i>	DMA 传输起始地址: <i>TIMER_DMACFG_DMATA_CREP</i> , <i>TIMERx</i> ( <i>x</i> =0, 14..16)
<i>TIMER_DMACFG_DMATA_CH0CV</i>	DMA 传输起始地址: <i>TIMER_DMACFG_DMATA_CH0CV</i> , <i>TIMERx</i> ( <i>x</i> =0..2, 14..16)
<i>TIMER_DMACFG_DMATA_CH1CV</i>	DMA 传输起始地址: <i>TIMER_DMACFG_DMATA_CH1CV</i> , <i>TIMERx</i> ( <i>x</i> =0..2, 14)
<i>TIMER_DMACFG_DMATA_CH2CV</i>	DMA 传输起始地址: <i>TIMER_DMACFG_DMATA_CH2CV</i> , <i>TIMERx</i> ( <i>x</i> =0..2)
<i>TIMER_DMACFG_DMATA_CH3CV</i>	DMA 传输起始地址: <i>TIMER_DMACFG_DMATA_CH3CV</i> , <i>TIMERx</i> ( <i>x</i> =0..2)
<i>TIMER_DMACFG_DMATA_CCHP</i>	DMA 传输起始地址: <i>TIMER_DMACFG_DMATA_CCHP</i> , <i>TIMERx</i> ( <i>x</i> =0, 14..16)
<i>TIMER_DMACFG_DMATA_DMACFG</i>	DMA 传输起始地址: <i>TIMER_DMACFG_DMATA_DMACFG</i> , <i>TIMERx</i> ( <i>x</i> =0..2, 14..16)
输入参数{in}	
<b>dma_lenth</b>	DMA传输长度
<i>TIMER_DMACFG_DMATC_xTRANSFER</i>	<i>x</i> =1..18, DMA传输 <i>x</i> 次
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* configure the TIMER0 DMA transfer */
```

```
timer_dma_transfer_config (TIMER0, TIMER_DMACFG_DMATA_CTL0,  
TIMER_DMACFG_DMATC_5TRANSFER);
```

### 函数 timer\_event\_software\_generate

函数timer\_event\_software\_generate描述见下表:

**表 3-515. 函数 timer\_event\_software\_generate**

函数名称	timer_event_software_generate
函数原型	void timer_event_software_generate(uint32_t timer_periph, uint16_t event);
功能描述	软件产生事件
先决条件	-

被调用函数	-
输入参数{in}	
timer_periph	TIMER 外设
TIMERx	参考具体参数
输入参数{in}	
event	事件源
TIMER_EVENT_SR C_UPG	更新事件产生, TIMERx(x=0..2, 5, 13..16)
TIMER_EVENT_SR C_CH0G	通道 0 捕获或比较事件发生, TIMERx(x=0..2, 13..16)
TIMER_EVENT_SR C_CH1G	通道 1 捕获或比较事件发生, TIMERx(x=0..2, 14)
TIMER_EVENT_SR C_CH2G	通道 2 捕获或比较事件发生, TIMERx(x=0..2)
TIMER_EVENT_SR C_CH3G	通道 3 捕获或比较事件发生, TIMERx(x=0..2)
TIMER_EVENT_SR C_CMTG	通道换相更新事件发生, TIMERx(x=0, 14..16)
TIMER_EVENT_SR C_TRGG	触发事件产生, TIMERx(x=0..2, 14)
TIMER_EVENT_SR C_BRKG	产生中止事件, TIMERx(x=0, 14..16)
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* software generate update event*/
```

```
timer_event_software_generate (TIMER0, TIMER_EVENT_SRC_UPG);
```

### 函数 timer\_break\_struct\_para\_init

函数timer\_break\_struct\_para\_init描述见下表:

表 3-516. 函数 timer\_break\_struct\_para\_init

函数名称	timer_break_struct_para_init
函数原型	void timer_break_struct_para_init(timer_break_parameter_struct* breakpara);
功能描述	将 TIMER 中止功能参数结构体中所有参数初始化为默认值
先决条件	-
被调用函数	-
输入参数{in}	
breakpara	中止功能配置结构体, 详见 <a href="#">表 3-481. 结构体类型</a>

	<a href="#"><u>timer_break_parameter_struct</u></a>
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* initialize TIMER break parameter struct with a default value */
```

```
timer_break_parameter_struct timer_breakpara;
```

```
timer_break_struct_para_init(timer_breakpara);
```

### 函数 timer\_break\_config

函数timer\_break\_config描述见下表：

表 3-517. 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, 14..16)	TIMER 外设选择
输入参数{in}	
breakpara	中止功能配置结构体，详见 <a href="#">表 3-481. 结构体类型</a> <a href="#"><u>timer_break_parameter_struct</u></a>
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* configure TIMER0 break function */
```

```
timer_break_parameter_struct timer_breakpara;
```

```
timer_breakpara.runoffstate = TIMER_ROS_STATE_DISABLE;
```

```
timer_breakpara.ideloffstate = TIMER_IOS_STATE_DISABLE ;
```

```
timer_breakpara.deadtime = 255;
```

```

timer_breakpara.breakpolarity    = TIMER_BREAK_POLARITY_LOW;

timer_breakpara.outputautostate  = TIMER_OUTAUTO_ENABLE;

timer_breakpara.protectmode      = TIMER_CCHP_PROT_0;

timer_breakpara.breakstate       = TIMER_BREAK_ENABLE;

timer_break_config(TIMER0,&timer_breakpara);

```

### 函数 timer\_break\_enable

函数timer\_break\_enable描述见下表:

**表 3-518. 函数 timer\_break\_enable**

函数名称	timer_break_enable
函数原型	void timer_break_enable(uint32_t timer_periph);
功能描述	使能 TIMERx 的中止功能
先决条件	只有在 TIMERx_CCHP 寄存器的 PROT [1:0] =00 时，才可修改
被调用函数	-
输入参数{in}	
timer_periph	TIMER 外设
TIMERx(x=0, 14..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 3-519. 函数 timer\_break\_disable**

函数名称	timer_break_disable
函数原型	void timer_break_disable (uint32_t timer_periph);
功能描述	禁能 TIMERx 的中止功能
先决条件	只有在 TIMERx_CCHP 寄存器的 PROT [1:0] =00 时，才可修改
被调用函数	-
输入参数{in}	
timer_periph	TIMER 外设
TIMERx(x=0, 14..16)	TIMER 外设选择

14..16)	
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 3-520. 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 TIMEx_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
TIMEx(x=0, 14..16)	TIMER 外设选择
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* enable TIMER0 output automatic function */
```

```
timer_automatic_output_enable (TIMER0);
```

### 函数 timer\_automatic\_output\_disable

函数timer\_automatic\_output\_disable描述见下表：

表 3-521. 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 TIMEx_CCHP 寄存器的 PROT [1:0] = 00 时，才可修改
被调用函数	-

输入参数{in}	
<b>timer_periph</b>	TIMER 外设
<i>TIMERx</i> ( <i>x</i> =0, 14..16)	TIMER 外设选择
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* disable TIMER0 output automatic function */
```

```
timer_automatic_output_disable (TIMER0);
```

### 函数 timer\_primary\_output\_config

函数timer\_primary\_output\_config描述见下表:

表 3-522. 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}	
<b>timer_periph</b>	TIMER 外设
<i>TIMERx</i> ( <i>x</i> =0, 14..16)	TIMER 外设选择
输入参数{in}	
<b>newvalue</b>	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 3-523. 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, 14..16)	TIMER 外设选择
输入参数{in}	
newvalue	控制状态
ENABLE	使能
DISABLE	禁能
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* channel capture/compare control shadow register enable */
```

```
timer_channel_control_shadow_config (TIMER0, ENABLE);
```

### 函数 timer\_channel\_control\_shadow\_update\_config

函数timer\_channel\_control\_shadow\_update\_config描述见下表:

表 3-524. 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, 14..16)	TIMER 外设选择
输入参数{in}	
ccuctl	通道换相控制影子寄存器更新控制
TIMER_UPDATECT	CMTG 位被置 1 时更新影子寄存器

<i>L_CCU</i>	
<i>TIMER_UPDATECT</i> <i>L_CCUTRI</i>	当 CMTG 位被置 1 或检测到 TRIGI 上升沿时，影子寄存器更新
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* configure TIMER0 channel control shadow register update when CMTG bit is set */
timer_channel_control_shadow_update_config (TIMER0, TIMER_UPDATECTL_CCU);
```

### 函数 timer\_channel\_output\_struct\_para\_init

函数timer\_channel\_output\_struct\_para\_init描述见下表：

表 3-525. 函数 timer\_channel\_output\_struct\_para\_init

函数名称	timer_channel_output_struct_para_init
函数原型	void timer_channel_output_struct_para_init(timer_oc_parameter_struct* ocpara);
功能描述	将 TIMER 通道输出参数结构体中所有参数初始化为默认值
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="#">表 3-482. 结构体类型 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 3-526. 函数 timer\_channel\_output\_config

函数名称	timer_channel_output_config
函数原型	void timer_channel_output_config(uint32_t timer_periph, uint16_t channel, timer_oc_parameter_struct* ocpara);

功能描述	外设 TIMERx 的通道输出配置
先决条件	-
被调用函数	-
输入参数{in}	
timer_periph	TIMER 外设
TIMERx	参考具体参数
输入参数{in}	
channel	待配置通道
TIMER_CH_0	通道 0, TIMERx(x=0..2, 13..16)
TIMER_CH_1	通道 1, TIMERx(x=0..2, 14)
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}	
ocpara	输出通道结构体, 详见 <a href="#">表 3-482. 结构体类型 timer_oc_parameter_struct</a>
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* configure TIMER0 channel 0 output function */

timer_oc_parameter_struct timer_ocintpara;

timer_ocintpara.outputstate = TIMER_CCX_ENABLE;

timer_ocintpara.outputnstate = TIMER_CCXN_ENABLE;

timer_ocintpara.ocpolarity = TIMER_OC_POLARITY_HIGH;

timer_ocintpara.ocnpolarity = TIMER_OCN_POLARITY_HIGH;

timer_ocintpara.ocidlestate = TIMER_OC_IDLE_STATE_HIGH;

timer_ocintpara.ocnidlestate = TIMER_OCN_IDLE_STATE_LOW;

timer_channel_output_config(TIMER0, TIMER_CH_0, &timer_ocintpara);
```

### 函数 timer\_channel\_output\_mode\_config

函数 timer\_channel\_output\_mode\_config 描述见下表:

表 3-527. 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, <i>TIMERx</i> ( <i>x</i> =0..2, 13..16)
<i>TIMER_CH_1</i>	通道 1, <i>TIMERx</i> ( <i>x</i> =0..2, 14)
<i>TIMER_CH_2</i>	通道 2, <i>TIMERx</i> ( <i>x</i> =0..2)
<i>TIMER_CH_3</i>	通道 3, <i>TIMERx</i> ( <i>x</i> =0..2)
输入参数{in}	
<b>ocmode</b>	通道输出比较模式
<i>TIMER_OC_MODE_TIMING</i>	冻结模式
<i>TIMER_OC_MODE_ACTIVE</i>	匹配时设置为高
<i>TIMER_OC_MODE_INACTIVE</i>	匹配时设置为低
<i>TIMER_OC_MODE_TOGGLE</i>	匹配时翻转
<i>TIMER_OC_MODE_LOW</i>	强制为低
<i>TIMER_OC_MODE_HIGH</i>	强制为高
<i>TIMER_OC_MODE_PWM0</i>	PWM 模式 0
<i>TIMER_OC_MODE_PWM1</i>	PWM 模式 1
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* configure TIMER0 channel PWM 0 mode */
```

```
timer_channel_output_mode_config(TIMER0, TIMER_CH_0, TIMER_OC_MODE_PWM0);
```

### 函数 timer\_channel\_output\_pulse\_value\_config

函数timer\_channel\_output\_pulse\_value\_config描述见下表:

表 3-528. 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..16)
TIMER_CH_1	通道 1, TIMERx (x=0..2, 14)
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 TIMER0 channel 0 output pulse value */
timer_channel_output_pulse_value_config(TIMER0, TIMER_CH_0, 399);
```

### 函数 timer\_channel\_output\_shadow\_config

函数timer\_channel\_output\_shadow\_config描述见下表:

表 3-529. 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}	

<b>channel</b>	待配置通道
<i>TIMER_CH_0</i>	通道 0, <i>TIMERx</i> ( <i>x</i> =0..2, 13..16)
<i>TIMER_CH_1</i>	通道 1, <i>TIMERx</i> ( <i>x</i> =0..2, 14)
<i>TIMER_CH_2</i>	通道 2, <i>TIMERx</i> ( <i>x</i> =0..2)
<i>TIMER_CH_3</i>	通道 3, <i>TIMERx</i> ( <i>x</i> =0..2)
<b>输入参数{in}</b>	
<b>ocshadow</b>	输出比较影子寄存器功能状态
<i>TIMER_OC_SHADOW_ENABLE</i>	使能输出比较影子寄存器
<i>TIMER_OC_SHADOW_DISABLE</i>	禁能输出比较影子寄存器
<b>输出参数{out}</b>	
-	-
<b>返回值</b>	
-	-

例如:

```
/*configure TIMER0 channel 0 output shadow function */
```

```
timer_channel_output_shadow_config (TIMER0, TIMER_CH_0,  
TIMER_OC_SHADOW_ENABLE);
```

### 函数 timer\_channel\_output\_fast\_config

函数timer\_channel\_output\_fast\_config描述见下表:

**表 3-530. 函数 timer\_channel\_output\_fast\_config**

<b>函数名称</b>	timer_channel_output_fast_config
<b>函数原型</b>	void timer_channel_output_fast_config(uint32_t timer_periph, uint16_t channel, uint16_t ocfast);
<b>功能描述</b>	配置 <i>TIMERx</i> 通道输出比较快速功能
<b>先决条件</b>	-
<b>被调用函数</b>	-
<b>输入参数{in}</b>	
<b>timer_periph</b>	<i>TIMER</i> 外设
<i>TIMERx</i>	参考具体参数
<b>输入参数{in}</b>	
<b>channel</b>	待配置通道
<i>TIMER_CH_0</i>	通道 0, <i>TIMERx</i> ( <i>x</i> =0..2, 13..16)
<i>TIMER_CH_1</i>	通道 1, <i>TIMERx</i> ( <i>x</i> =0..2, 14)
<i>TIMER_CH_2</i>	通道 2, <i>TIMERx</i> ( <i>x</i> =0..2)
<i>TIMER_CH_3</i>	通道 3, <i>TIMERx</i> ( <i>x</i> =0..2)
<b>输入参数{in}</b>	

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

例如：

```
/* configure TIMER0 channel 0 output fast function */
```

```
timer_channel_output_fast_config (TIMER0, TIMER_CH_0, TIMER_OC_FAST_ENABLE);
```

### 函数 timer\_channel\_output\_clear\_config

函数timer\_channel\_output\_clear\_config描述见下表：

**表 3-531. 函数 timer\_channel\_output\_clear\_config**

<b>函数名称</b>	timer_channel_output_clear_config
<b>函数原型</b>	void timer_channel_output_clear_config(uint32_t timer_periph, uint16_t channel, uint16_t occlear);
<b>功能描述</b>	配置 TIMEx 的通道输出比较清 0 功能
<b>先决条件</b>	-
<b>被调用函数</b>	-
<b>输入参数{in}</b>	
<b>timer_periph</b>	TIMER 外设
<i>TIMERx</i>	参考具体参数
<b>输入参数{in}</b>	
<b>channel</b>	待配置通道
<i>TIMER_CH_0</i>	通道 0, TIMEx (x=0..2)
<i>TIMER_CH_1</i>	通道 1, TIMEx (x=0..2)
<i>TIMER_CH_2</i>	通道 2, TIMEx (x=0..2)
<i>TIMER_CH_3</i>	通道 3, TIMEx (x=0..2)
<b>输入参数{in}</b>	
<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
<b>输出参数{out}</b>	
-	-

返回值	
-	-

例如：

```
/* configure TIMER0 channel 0 output clear function */

timer_channel_output_clear_config (TIMER0, TIMER_CH_0,
TIMER_OC_CLEAR_ENABLE);
```

### 函数 timer\_channel\_output\_polarity\_config

函数timer\_channel\_output\_polarity\_config描述见下表：

**表 3-532. 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..16)
<i>TIMER_CH_1</i>	通道 1，TIMERx (x=0..2, 14)
<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>ocpolarity</b>	通道输出极性
<i>TIMER_OC_POLARITY_HIGH</i>	通道输出极性高电平有效
<i>TIMER_OC_POLARITY_LOW</i>	通道输出极性低电平有效
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* configure TIMER0 channel 0 output polarity */

timer_channel_output_polarity_config (TIMER0, TIMER_CH_0,
```



TIMER\_OC\_POLARITY\_HIGH);

### 函数 timer\_channel\_complementary\_output\_polarity\_config

函数timer\_channel\_complementary\_output\_polarity\_config描述见下表:

表 3-533. 函数 timer\_channel\_complementary\_output\_polarity\_config

函数名称	timer_channel_complementary_output_polarity_config
函数原型	void timer_channel_complementary_output_polarity_config(uint32_t timer_periph, uint16_t channel, uint16_t ocpolarity);
功能描述	互补通道输出极性配置
先决条件	-
被调用函数	-
输入参数{in}	
timer_periph	TIMER 外设
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..16)
TIMER_CH_1	通道 1, TIMERx (x=0..2, 14)
TIMER_CH_2	通道 2, TIMERx (x=0..2)
TIMER_CH_3	通道 2, TIMERx (x=0..2)
输入参数{in}	
ocpolarity	互补通道输出极性
TIMER_OCN_POLARITY_HIGH	互补通道输出极性高电平有效
TIMER_OCN_POLARITY_LOW	互补通道输出极性低电平有效
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* configure TIMER0 channel 0 complementary output polarity */
timer_channel_complementary_output_polarity_config (TIMER0, TIMER_CH_0,
TIMER_OCN_POLARITY_HIGH);
```

### 函数 timer\_channel\_output\_state\_config

函数timer\_channel\_output\_state\_config描述见下表:

表 3-534. 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..16)
TIMER_CH_1	通道 1, TIMERx (x=0..2, 14)
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}	
state	通道状态
TIMER_CCX_ENABLE	通道使能
TIMER_CCX_DISABLE	通道禁能
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* configure TIMER0 channel 0 enable state */
```

```
timer_channel_output_state_config (TIMER0, TIMER_CH_0, TIMER_CCX_ENABLE);
```

### 函数 timer\_channel\_complementary\_output\_state\_config

函数timer\_channel\_complementary\_output\_state\_config描述见下表：

表 3-535. 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}	
<b>channel</b>	待配置通道
<i>TIMER_CH_0</i>	通道 0, <i>TIMERx</i> ( $x=0, 14..16$ )
<i>TIMER_CH_1</i>	通道 1, <i>TIMERx</i> ( $x=0$ )
<i>TIMER_CH_2</i>	通道 2, <i>TIMERx</i> ( $x=0$ )
输入参数{in}	
<b>state</b>	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 3-536. 函数 timer\_channel\_input\_struct\_para\_init

函数名称	timer_channel_input_struct_para_init
函数原型	void timer_channel_input_struct_para_init(timer_ic_parameter_struct* icpara);
功能描述	将 TIMER 通道输入参数结构体中所有参数初始化为默认值
先决条件	-
被调用函数	-
输入参数{in}	
<b>icpara</b>	通道输入结构体, 详见 <a href="#">表 3-483. 结构体类型 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 3-537. 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..16)
TIMER_CH_1	通道 1, TIMERx (x=0..2, 14)
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}	
icpara	输入捕获结构体, 详见 <a href="#">表 3-483. 结构体类型 timer_ic_parameter_struct</a> 。
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* configure TIMER0 input capture parameter */
timer_ic_parameter_struct timer_icinitpara;

timer_icinitpara.icpolarity = TIMER_IC_POLARITY_RISING;
timer_icinitpara.icselection = TIMER_IC_SELECTION_DIRECTTTI;
timer_icinitpara.icprescaler = TIMER_IC_PSC_DIV1;
timer_icinitpara.icfilter = 0x0;

timer_input_capture_config(TIMER0, TIMER_CH_0, &timer_icinitpara);
```

### 函数 timer\_channel\_input\_capture\_prescaler\_config

函数timer\_channel\_input\_capture\_prescaler\_config描述见下表:

表 3-538. 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 TIMEx 通道输入捕获预分频值
先决条件	-
被调用函数	-
输入参数{in}	
timer_periph	TIMER 外设
TIMEx	参考具体参数
输入参数{in}	
channel	待配置通道
TIMER_CH_0	通道 0, TIMEx (x=0..2, 13..16)
TIMER_CH_1	通道 1, TIMEx (x=0..2, 14)
TIMER_CH_2	通道 2, TIMEx (x=0..2)
TIMER_CH_3	通道 3, TIMEx (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 TIMER0 channel 0 input capture prescaler value */
timer_channel_input_capture_prescaler_config (TIMER0, TIMER_CH_0,
TIMER_IC_PSC_DIV2);
```

### 函数 timer\_channel\_capture\_value\_register\_read

函数timer\_channel\_capture\_value\_register\_read描述见下表:

表 3-539. 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..16)
TIMER_CH_1	通道 1, TIMERx (x=0..2, 14)
TIMER_CH_2	通道 2, TIMERx (x=0..2)
TIMER_CH_3	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 TIMER0 channel 0 capture compare register value */
```

```
uint32_t ch0_value = 0;
```

```
ch0_value = timer_channel_capture_value_register_read (TIMER0, TIMER_CH_0);
```

### 函数 timer\_input\_pwm\_capture\_config

函数timer\_input\_pwm\_capture\_config描述见下表:

表 3-540. 函数 timer\_input\_pwm\_capture\_config

函数名称	timer_input_pwm_capture_config
函数原型	void timer_input_pwm_capture_config(uint32_t timer_periph, uint16_t channel, timer_ic_parameter_struct* icpwm);
功能描述	配置 TIMERx 捕获 PWM 输入参数
先决条件	-
被调用函数	timer_channel_input_capture_prescaler_config
输入参数{in}	
timer_periph	TIMER 外设
TIMERx(x=0..2, 14)	TIMER 外设选择
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 3-483. 结构体类型 timer_ic_parameter_struct</a>
输出参数{out}	
-	-
返回值	
-	-

例如:

```

/* configure TIMER0 input pwm capture parameter */
timer_ic_parameter_struct timer_icinitpara;

timer_icinitpara.icpolarity = TIMER_IC_POLARITY_RISING;

timer_icinitpara.icselection = TIMER_IC_SELECTION_DIRECTTTI;

timer_icinitpara.icprescaler = TIMER_IC_PSC_DIV1;

timer_icinitpara.icfilter = 0x0;

timer_input_pwm_capture_config (TIMER0, TIMER_CH_0, &timer_icinitpara);

```

### 函数 timer\_hall\_mode\_config

函数timer\_hall\_mode\_config描述见下表:

表 3-541. 函数 timer\_hall\_mode\_config

函数名称	timer_hall_mode_config
函数原型	void timer_hall_mode_config(uint32_t timer_periph, uint8_t hallmode);
功能描述	配置 TIMERx 的 HALL 接口功能
先决条件	-
被调用函数	-
输入参数{in}	
timer_periph	TIMER 外设
TIMERx(x=0..2)	TIMER 外设选择
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 TIMER0 hall sensor mode */
```

```
timer_hall_mode_config (TIMER0, TIMER_HALLINTERFACE_ENABLE);
```

### 函数 timer\_input\_trigger\_source\_select

函数timer\_input\_trigger\_source\_select描述见下表：

**表 3-542. 函数 timer\_input\_trigger\_source\_select**

函数名称	timer_input_trigger_source_select
函数原型	void timer_input_trigger_source_select(uint32_t timer_periph, uint32_t intrigger);
功能描述	TIMERx 的输入触发源选择
先决条件	SMC[2:0] = 000
被调用函数	-
输入参数{in}	
timer_periph	TIMER 外设
TIMERx(x=0, 2, 14)	TIMER 外设选择
输入参数{in}	
intrigger	待选择的触发源
TIMER_SMCFG_T RGSEL_ITI0	内部触发输入 0(ITI0, TIMERx(x=0..2, 14))
TIMER_SMCFG_T RGSEL_ITI1	内部触发输入 1(ITI1, TIMERx(x=0..2, 14))
TIMER_SMCFG_T RGSEL_ITI2	内部触发输入 2(ITI2, TIMERx(x=0..2))
TIMER_SMCFG_T RGSEL_ITI3	内部触发输入 3(ITI3, TIMERx(x=0..2, 14))
TIMER_SMCFG_T RGSEL_CIOF_ED	CIO 的边沿标志位 (CIOF_ED, TIMERx(x=0..2, 14))
TIMER_SMCFG_T RGSEL_CIOFE0	滤波后的通道 0 输入 (CIOFE0, TIMERx(x=0..2, 14))
TIMER_SMCFG_T RGSEL_CI1FE1	滤波后的通道 1 输入(CI1FE1, TIMERx(x=0..2, 14))
TIMER_SMCFG_T RGSEL_ETIFP	滤波后的外部触发输入(ETIFP, TIMERx(x=0..2))
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* select TIMER0 input trigger source */
```



timer\_input\_trigger\_source\_select (TIMER0, TIMER\_SMCFG\_TRGSEL\_ITI0);

## 函数 timer\_master\_output\_trigger\_source\_select

函数timer\_master\_output\_trigger\_source\_select描述见下表:

表 3-543. 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(x=0..2, 5, 14)	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 TIMER0 master mode output trigger source */
```

```
timer_master_output_trigger_source_select (TIMER0, TIMER_TRI_OUT_SRC_RESET);
```

### 函数 timer\_slave\_mode\_select

函数timer\_slave\_mode\_select描述见下表：

**表 3-544. 函数 timer\_slave\_mode\_select**

函数名称	timer_slave_mode_select
函数原型	void timer_slave_mode_select(uint32_t timer_periph, uint32_t slavemode);
功能描述	TIMERx 从模式配置
先决条件	-
被调用函数	-
输入参数{in}	
timer_periph	TIMER 外设
TIMERx(x=0..2, 14)	TIMER 外设选择
输入参数{in}	
slavemode	从模式
TIMER_SLAVE_MODE_DISABLE	关闭从模式, TIMERx(x=0..2, 14)
TIMER_QUAD_ENCODER_MODE0	正交译码器模式 0, TIMERx(x=0..2)
TIMER_QUAD_ENCODER_MODE1	正交译码器模式 1, TIMERx(x=0..2)
TIMER_QUAD_ENCODER_MODE2	正交译码器模式 2, TIMERx(x=0..2)
TIMER_SLAVE_MODE_RESTART	复位模式, TIMERx(x=0..2, 14)
TIMER_SLAVE_MODE_PAUSE	暂停模式, TIMERx(x=0..2, 14)
TIMER_SLAVE_MODE_EVENT	事件模式, TIMERx(x=0..2, 14)
TIMER_SLAVE_MODE_EXTERNAL0	外部时钟模式 0, TIMERx(x=0..2, 14)
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* select TIMER0 slave mode */
```

```
timer_slave_mode_select (TIMER0, TIMER_QUAD_DECODER_MODE0);
```

### 函数 timer\_master\_slave\_mode\_config

函数timer\_master\_slave\_mode\_config描述见下表：

表 3-545. 函数 timer\_master\_slave\_mode\_config

函数名称	timer_master_slave_mode_config
函数原型	void timer_master_slave_mode_config(uint32_t timer_periph, uint8_t masterslave);
功能描述	TIMERx 主从模式配置
先决条件	-
被调用函数	-
输入参数{in}	
timer_periph	TIMER 外设
TIMERx(x=0..2, 14)	TIMER 外设选择
输入参数{in}	
masterslave	主从模式使能状态
TIMER_MASTER_SLAVE_MODE_ENABLE	主从模式使能
TIMER_MASTER_SLAVE_MODE_DISABLE	主从模式禁能
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* configure TIMER0 master slave mode */
```

```
timer_master_slave_mode_config (TIMER0, TIMER_MASTER_SLAVE_MODE_ENABLE);
```

### 函数 timer\_external\_trigger\_config

函数timer\_external\_trigger\_config描述见下表：

表 3-546. 函数 timer\_external\_trigger\_config

函数名称	timer_external_trigger_config
函数原型	void timer_external_trigger_config(uint32_t timer_periph, uint32_t extprescaler, uint32_t expolarity, uint32_t extfilter);
功能描述	配置 TIMERx 外部触发输入
先决条件	-
被调用函数	-

输入参数{in}	
<b>timer_periph</b>	TIMER 外设
<i>TIMERx</i> (x=0..2)	TIMER 外设选择
输入参数{in}	
<b>extprescaler</b>	外部触发预分频
<i>TIMER_EXT_TRI_P</i> <i>SC_OFF</i>	不分频
<i>TIMER_EXT_TRI_P</i> <i>SC_DIV2</i>	2 分频
<i>TIMER_EXT_TRI_P</i> <i>SC_DIV4</i>	4 分频
<i>TIMER_EXT_TRI_P</i> <i>SC_DIV8</i>	8 分频
输入参数{in}	
<b>expolarity</b>	外部触发输入极性
<i>TIMER_ETP_FALLI</i> <i>NG</i>	低电平或者下降沿有效
<i>TIMER_ETP_RISIN</i> <i>G</i>	高电平或者上升沿有效
输入参数{in}	
<b>extfilter</b>	外部触发滤波控制 (0~15)
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* configure TIMER0 external trigger input */
```

```
timer_external_trigger_config (TIMER0, TIMER_EXT_TRI_PSC_DIV2,  
TIMER_ETP_FALLING, 10);
```

### 函数 timer\_quadrature\_decoder\_mode\_config

函数timer\_quadrature\_decoder\_mode\_config描述见下表:

表 3-547. 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(x=0..2)</i>	TIMER 外设选择
<b>输入参数{in}</b>	
<b>decomode</b>	编码器模式
<i>TIMER_QUAD_DECODER_MODE0</i>	根据 CI0FE0 的电平，计数器在 CI1FE1 的边沿向上/下计数
<i>TIMER_QUAD_DECODER_MODE1</i>	根据 CI1FE1 的电平，计数器在 CI0FE0 的边沿向上/下计数
<i>TIMER_QUAD_DECODER_MODE2</i>	根据另一个信号的输入电平，计数器在 CI0FE0 和 CI1FE1 的边沿向上/下计数
<b>输入参数{in}</b>	
<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
<b>输入参数{in}</b>	
<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
<b>输出参数{out}</b>	
-	-
<b>返回值</b>	
-	-

例如：

```
/* configure TIMER0 quadrature decoder mode */
```

```
timer_quadrature_decoder_mode_config (TIMER0, TIMER_QUAD_DECODER_MODE0,  
TIMER_IC_POLARITY_RISING, TIMER_IC_POLARITY_RISING);
```

### 函数 timer\_internal\_clock\_config

函数timer\_internal\_clock\_config描述见下表：

表 3-548. 函数 timer\_internal\_clock\_config

<b>函数名称</b>	timer_internal_clock_config
<b>函数原型</b>	void timer_internal_clock_config(uint32_t timer_periph);
<b>功能描述</b>	TIMERx 配置为内部时钟模式
<b>先决条件</b>	-
<b>被调用函数</b>	-
<b>输入参数{in}</b>	

<b>timer_periph</b>	TIMER 外设
<i>TIMERx</i> ( <i>x</i> =0..2, 14)	TIMER 外设选择
<b>输出参数{out}</b>	
-	-
<b>返回值</b>	
-	-

例如:

```
/* configure TIMER0 internal clock mode */
```

```
timer_internal_clock_config (TIMER0);
```

### 函数 timer\_internal\_trigger\_as\_external\_clock\_config

函数timer\_internal\_trigger\_as\_external\_clock\_config描述见下表:

**表 3-549. 函数 timer\_internal\_trigger\_as\_external\_clock\_config**

<b>函数名称</b>	timer_internal_trigger_as_external_clock_config
<b>函数原型</b>	void timer_internal_trigger_as_external_clock_config(uint32_t timer_periph, uint32_t intrigger);
<b>功能描述</b>	配置 TIMERx 的内部触发为时钟源
<b>先决条件</b>	-
<b>被调用函数</b>	timer_input_trigger_source_select
<b>输入参数{in}</b>	
<b>timer_periph</b>	TIMER 外设
<i>TIMERx</i> ( <i>x</i> =0..2, 14)	TIMER 外设选择
<b>输入参数{in}</b>	
<b>intrigger</b>	被选择的内部触发源
<i>TIMER_SMCFG_TRGSEL_ITI0</i>	选择内部触发 0 (ITI0)为时钟源, TIMERx( <i>x</i> =0..2, 14)
<i>TIMER_SMCFG_TRGSEL_ITI1</i>	选择内部触发 1 (ITI1)为时钟源, TIMERx( <i>x</i> =0..2, 14)
<i>TIMER_SMCFG_TRGSEL_ITI2</i>	选择内部触发 2 (ITI2)为时钟源, TIMERx( <i>x</i> =0..2)
<b>输出参数{out}</b>	
-	-
<b>返回值</b>	
-	-

例如:

```
/* configure TIMER0 the internal trigger ITI0 as external clock input */
```

```
timer_internal_trigger_as_external_clock_config (TIMER0, TIMER_SMCFG_TRGSEL_ITI0);
```

**函数 timer\_external\_trigger\_as\_external\_clock\_config**

函数timer\_external\_trigger\_as\_external\_clock\_config描述见下表：

**表 3-550. 函数 timer\_external\_trigger\_as\_external\_clock\_config**

函数名称	timer_external_trigger_as_external_clock_config
函数原型	void timer_external_trigger_as_external_clock_config(uint32_t timer_periph, uint32_t extrigger, uint16_t expolarity, uint32_t extfilter);
功能描述	配置 TIMEx 的外部触发作为时钟源
先决条件	-
被调用函数	timer_input_trigger_source_select
输入参数{in}	
timer_periph	TIMER 外设
TIMERx(x=0..2, 14)	TIMER 外设选择
输入参数{in}	
extrigger	外部触发源
TIMER_SMCFG_TRGSEL_CIOF_ED	CI0 的边沿标志(CIOF_ED)
TIMER_SMCFG_TRGSEL_CIOFE0	滤波后的通道 0 输入(CIOFE0)
TIMER_SMCFG_TRGSEL_CIIFE1	滤波后的通道 1 输入(CI1FE1)
输入参数{in}	
expolarity	外部触发源极性
TIMER_IC_POLARITY_RISING	外部触发源高电平或者上升沿有效
TIMER_IC_POLARITY_FALLING	外部触发源低电平或者下降沿有效
TIMER_IC_POLARITY_BOTH_EDGE	下降沿或者上升沿有效
输入参数{in}	
extfilter	滤波参数（0~15）
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* configure TIMER0 the external trigger CIOFE0 as external clock input */
timer_external_trigger_as_external_clock_config (TIMER0,
TIMER_SMCFG_TRGSEL_CIOFE0, TIMER_IC_POLARITY_RISING, 0);
```

## 函数 timer\_external\_clock\_mode0\_config

函数timer\_external\_clock\_mode0\_config描述见下表:

表 3-551. 函数 timer\_external\_clock\_mode0\_config

函数名称	timer_external_clock_mode0_config
函数原型	void timer_external_clock_mode0_config(uint32_t timer_periph, uint32_t extprescaler, uint32_t expolarity, uint32_t extfilter);
功能描述	配置 TIMERx 外部时钟模式 0，ETI 作为时钟源
先决条件	-
被调用函数	timer_external_trigger_config
输入参数{in}	
timer_periph	TIMER 外设
TIMERx(x=0..2)	TIMER 外设选择
输入参数{in}	
extprescaler	ETI 触发源预分频值
TIMER_EXT_TRI_P SC_OFF	不分频
TIMER_EXT_TRI_P SC_DIV2	2 分频
TIMER_EXT_TRI_P SC_DIV4	4 分频
TIMER_EXT_TRI_P SC_DIV8	8 分频
输入参数{in}	
expolarity	ETI 触发源极性
TIMER_ETP_FALLI NG	下降沿或者低电平有效
TIMER_ETP_RISIN G	上升沿或者高电平有效
输入参数{in}	
extfilter	ETI 触发源滤波参数 (0~15)
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* configure TIMER0 the external clock mode0 */
```

```
timer_external_clock_mode0_config (TIMER0, TIMER_EXT_TRI_PSC_DIV2,  
TIMER_ETP_FALLING, 0);
```



## 函数 timer\_external\_clock\_mode1\_config

函数timer\_external\_clock\_mode1\_config描述见下表:

表 3-552. 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 TIMEx 外部时钟模式 1
先决条件	-
被调用函数	timer_external_trigger_config
输入参数{in}	
timer_periph	TIMER 外设
TIMERx(x=0..2)	TIMER 外设选择
输入参数{in}	
extprescaler	ETI 触发源预分频值
TIMER_EXT_TRI_P SC_OFF	不分频
TIMER_EXT_TRI_P SC_DIV2	2 分频
TIMER_EXT_TRI_P SC_DIV4	4 分频
TIMER_EXT_TRI_P SC_DIV8	8 分频
输入参数{in}	
expolarity	ETI 触发源极性
TIMER_ETP_FALLI NG	下降沿或者低电平有效
TIMER_ETP_RISIN G	上升沿或者高电平有效
输入参数{in}	
extfilter	ETI 触发源滤波参数 (0~15)
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* configure TIMER0 the external clock mode1 */
```

```
timer_external_clock_mode1_config (TIMER0, TIMER_EXT_TRI_PSC_DIV2,  
TIMER_ETP_FALLING, 0);
```

**函数 timer\_external\_clock\_mode1\_disable**

函数timer\_external\_clock\_mode1\_disable描述见下表:

**表 3-553. 函数 timer\_external\_clock\_mode1\_disable**

函数名称	timer_external_clock_mode1_disable
函数原型	void timer_external_clock_mode1_disable(uint32_t timer_periph);
功能描述	TIMERx 外部时钟模式 1 禁能
先决条件	-
被调用函数	-
输入参数{in}	
timer_periph	TIMER 外设
TIMERx(x=0..2)	TIMER 外设选择
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* disable TIMER0 the external clock mode1 */
timer_external_clock_mode1_disable (TIMER0);
```

**函数 timer\_channel\_remap\_config**

函数timer\_channel\_remap\_config描述见下表:

**表 3-554. 函数 timer\_channel\_remap\_config**

函数名称	timer_channel_remap_config
函数原型	void timer_channel_remap_config (uint32_t timer_periph, uint32_t remap);
功能描述	配置 TIMERx 通道重映射功能
先决条件	-
被调用函数	-
输入参数{in}	
timer_periph	TIMER 外设
TIMERx(x=13)	TIMER 外设选择
输入参数{in}	
remap	重映射功能选择
TIMER13_C10_RMP_GPIO	通道 0 连接到 GPIO
TIMER13_C10_RMP_RTCCLK	通道 0 连接到 RTCCLK
TIMER13_C10_RMP_HXTAL_DIV32	通道 0 连接到 HXTAL/32

<i>TIMER13_CIO_RMP</i> <i>_CKOUTSEL</i>	通道 0 连接到 CKOUTSEL
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* configure TIMER13 channel 0 input is connected to GPIO */
```

```
timer_channel_remap_config (TIMER13, TIMER13_CIO_RMP_GPIO);
```

### 函数 timer\_write\_chxval\_register\_config

函数timer\_write\_chxval\_register\_config描述见下表：

**表 3-555. 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}	
<b>timer_periph</b>	TIMER 外设
<i>TIMERx</i> ( <i>x</i> =0..2, 13..16)	TIMER 外设选择
输入参数{in}	
<b>ccsel</b>	写 CHxVAL 寄存器选择位
<i>TIMER_CHVSEL_D</i> <i>ISABLE</i>	无影响
<i>TIMER_CHVSEL_E</i> <i>NABLE</i>	当写入捕获比较寄存器的值与寄存器当前值相等时，写入操作无效。
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* configure TIMER0 write CHxVAL register selection */
```

```
timer_write_chxval_register_config(TIMER0, TIMER_CHVSEL_ENABLE);
```

### 函数 timer\_output\_value\_selection\_config

函数timer\_output\_value\_selection\_config描述见下表：

表 3-556. 函数 timer\_output\_value\_selection\_config

函数名称	timer_output_value_selection_config
函数原型	void timer_output_value_selection_config(uint32_t timer_periph, uint16_t outsel);
功能描述	配置 TIMER 输出值选择位
先决条件	-
被调用函数	-
输入参数{in}	
timer_periph	TIMER 外设
TIMERx(x=0, 14..16)	TIMER 外设选择
输入参数{in}	
ccsel	输出值选择位
TIMER_OUTSEL_DISABLE	无影响
TIMER_OUTSEL_ENABLE	如果 POEN 位与 IOS 位均为 0，则输出无效。
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* configure TIMER output value selection */
timer_output_value_selection_config(TIMER0, TIMER_OUTSEL_ENABLE);
```

## 3.22. TSI

触摸传感控制器（TSI）为触摸按键、滑块、电容近距感测等应用提供了简易的解决方案。章节 [3.22.1](#) 描述了 TSI 的寄存器列表，章节 [3.22.2](#) 对 TSI 库函数进行说明。

### 3.22.1. 外设寄存器描述

TSI 寄存器列表如下表所示：

表 3-557. TSI 寄存器

寄存器名称	寄存器描述
TSI_CTL0	控制寄存器 0
TSI_INTEN	中断使能寄存器
TSI_INTC	中断标志位清除寄存器
TSI_INTF	中断标志位寄存器
TSI_PHM	引脚迟滞模式寄存器

寄存器名称	寄存器描述
TSI_ASW	模拟开关寄存器
TSI_SAMPCFG	采样配置寄存器
TSI_CHCFG	通道配置寄存器
TSI_GCTL	组控制寄存器
TSI_GxCYCN (x=0..5)	组 x 周期数寄存器
TSI_CTL1	控制寄存器 1

### 3.22.2. 外设库函数说明

TSI库函数列表如下表所示：

**表 3-558. TSI 库函数**

库函数名称	库函数描述
tsi_deinit	复位TSI外设
tsi_init	初始化TSI
tsi_enable	使能TSI模块
tsi_disable	禁能TSI模块
tsi_sample_pin_enable	使能TSI采样引脚
tsi_sample_pin_disable	禁能TSI采样引脚
tsi_channel_pin_enable	使能TSI通道引脚
tsi_channel_pin_disable	禁能TSI通道引脚
tsi_software_mode_config	配置TSI为软件触发模式
tsi_software_start	软件启动一次电荷转移序列
tsi_software_stop	软件停止已启动的电荷转移序列
tsi_hardware_mode_config	配置TSI为硬件触发模式
tsi_pin_mode_config	状态机为空闲状态时TSI引脚的模式
tsi_extend_charge_config	配置TSI扩展充电状态最大持续时间
tsi_pulse_config	配置TSI的电荷转移状态和充电状态的持续时间
tsi_max_number_config	配置TSI电荷转移序列的最大充电、转移周期数
tsi_hysteresis_on	引脚施密特触发迟滞模式使能
tsi_hysteresis_off	引脚施密特触发迟滞模式禁能
tsi_analog_on	模拟开关闭合
tsi_analog_off	模拟开关断开
tsi_group_enable	TSI引脚组使能
tsi_group_disable	TSI引脚组禁能
tsi_group_status_get	电荷转移完成状态获取
tsi_group0_cycle_get	电荷转移序列完成时组0执行的周期数
tsi_group1_cycle_get	电荷转移序列完成时组1执行的周期数
tsi_group2_cycle_get	电荷转移序列完成时组2执行的周期数
tsi_group3_cycle_get	电荷转移序列完成时组3执行的周期数

库函数名称	库函数描述
tsi_group4_cycle_get	电荷转移序列完成时组4执行的周期数
tsi_group5_cycle_get	电荷转移序列完成时组5执行的周期数
tsi_flag_get	TSI标志位获取
tsi_flag_clear	TSI标志位清除
tsi_interrupt_enable	TSI中断使能
tsi_interrupt_disable	TSI中断禁能
tsi_interrupt_flag_get	TSI中断标志位获取
tsi_interrupt_flag_clear	TSI中断标志位清除

### 函数 tsi\_deinit

函数tsi\_deinit描述见下表：

**表 3-559. 函数 tsi\_deinit**

函数名称	tsi_deinit
函数原形	void tsi_deinit (void);
功能描述	复位TSI外设
先决条件	-
被调用函数	rcu_periph_reset_enable / rcu_periph_reset_disable
输入参数{in}	
-	-
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* reset TSI */
```

```
tsi_deinit ();
```

### 函数 tsi\_init

函数tsi\_init描述见下表：

**表 3-560. 函数 tsi\_init**

函数名称	tsi_init
函数原形	void tsi_init (uint32_t prescaler, uint32_t charge_duration, uint32_t transfer_duration, uint32_t max_number);
功能描述	TSI初始化各个参数
先决条件	-
被调用函数	-
输入参数{in}	

<b>prescaler</b>	CTCLK时钟分频系数
<i>TSI_CTCDIV_DIV1</i>	CTCLK等于HCLK
<i>TSI_CTCDIV_DIV2</i>	HCLK 2分频
<i>TSI_CTCDIV_DIV4</i>	HCLK 4分频
<i>TSI_CTCDIV_DIV8</i>	HCLK 8分频
<i>TSI_CTCDIV_DIV16</i>	HCLK 16分频
<i>TSI_CTCDIV_DIV32</i>	HCLK 32分频
<i>TSI_CTCDIV_DIV64</i>	HCLK 64分频
<i>TSI_CTCDIV_DIV128</i>	HCLK 128分频
<i>TSI_CTCDIV_DIV256</i>	HCLK 256分频
<i>TSI_CTCDIV_DIV512</i>	HCLK 512分频
<i>TSI_CTCDIV_DIV1024</i>	HCLK 1024分频
<i>TSI_CTCDIV_DIV2048</i>	HCLK 2048分频
<i>TSI_CTCDIV_DIV4096</i>	HCLK 4096分频
<i>TSI_CTCDIV_DIV8192</i>	HCLK 8192分频
<i>TSI_CTCDIV_DIV16384</i>	HCLK 16384分频
<i>TSI_CTCDIV_DIV32768</i>	HCLK 32768分频
<b>输入参数{in}</b>	
<b>charge_duration</b>	充电状态持续时间
<i>TSI_CHARGE_1CTCLK(x=1..16)</i>	时间位X*t <sub>CTCLK</sub> (x=1..16)
<b>输入参数{in}</b>	
<b>transfer_duration</b>	电荷转移状态持续时间
<i>TSI_TRANSFER_xCTCLK(x=1..16)SS_CLEAR</i>	时间位X*t <sub>CTCLK</sub> (x=1..16)
<b>输入参数{in}</b>	
<b>max_number</b>	电荷转移序列的最大充电、转移周期数
<i>TSI_MAXNUM255</i>	255个周期
<i>TSI_MAXNUM511</i>	511个周期
<i>TSI_MAXNUM1023</i>	1023个周期
<i>TSI_MAXNUM2047</i>	2047个周期
<i>TSI_MAXNUM4095</i>	4095个周期

TSI_MAXNUM8191	8191个周期
TSI_MAXNUM16383	16383个周期
3	
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* init TSI */
```

```
tsi_init (TSI_CTCDIV_DIV4096, TSI_CHARGE_10CTCLK, TSI_TRANSFER_8CTCLK,
TSI_MAXNUM511);
```

### 函数 tsi\_enable

函数tsi\_enable描述见下表：

表 3-561. 函数 cec\_enable

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

例如：

```
/* enable TSI module */
```

```
tsi_enable ();
```

### 函数 tsi\_disable

函数tsi\_disable描述见下表：

表 3-562. 函数 tsi\_disable

函数名称	tsi_disable
函数原形	void tsi_disable (void);
功能描述	禁用TSI外设
先决条件	-



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

例如：

```
/* disable TSI module */
```

```
tsi_disable();
```

### 函数 tsi\_sample\_pin\_enable

函数tsi\_sample\_pin\_enable描述见下表：

表 3-563. 函数 tsi\_sample\_pin\_enable

函数名称	tsi_sample_pin_enable
函数原形	void tsi_sample_pin_enable (uint32_t sample);
功能描述	使能TSI采样引脚
先决条件	-
被调用函数	-
输入参数{in}	
sample	采样引脚
TSI_SAMPCFG_GxPy (x=0..5,y=0..3)	GxPy引脚是采样引脚
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* enable G5P3 sample pin */
```

```
tsi_sample_pin_enable (TSI_SAMPCFG_G5P3);
```

### 函数 tsi\_sample\_pin\_disable

函数tsi\_sample\_pin\_disable描述见下表：

表 3-564. 函数 tsi\_sample\_pin\_disable

函数名称	tsi_sample_pin_disable
函数原形	void tsi_sample_pin_disable (uint32_t sample);
功能描述	禁用TSI采样引脚

先决条件	-
被调用函数	-
输入参数{in}	
<b>sample</b>	采样引脚
<i>TSI_SAMPCFG_GxPy</i> (x=0..5,y=0..3)	GxPy引脚不是采样引脚
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* disable G5P3 sample pin */
```

```
tsi_sample_pin_disable (TSI_SAMPCFG_G5P3);
```

### 函数 tsi\_channel\_pin\_enable

函数tsi\_channel\_pin\_enable描述见下表:

**表 3-565. 函数 tsi\_channel\_pin\_enable**

函数名称	tsi_channel_pin_enable
函数原形	void tsi_channel_pin_enable (uint32_t channel);
功能描述	使能TSI通道引脚
先决条件	-
被调用函数	-
输入参数{in}	
<b>channel</b>	通道引脚
<i>TSI_CHCFG_GxPy</i> (x=0..5,y=0..3)	GxPy引脚是通道引脚
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* enable G5P3 channel pin */
```

```
tsi_channel_pin_enable (TSI_CHCFG_G5P3);
```

### 函数 tsi\_channel\_pin\_disable

函数tsi\_channel\_pin\_disable描述见下表:

表 3-566. 函数 tsi\_channel\_pin\_disable

函数名称	tsi_channel_pin_disable
函数原形	void tsi_channel_pin_disable (uint32_t channel);
功能描述	禁用TSI通道引脚
先决条件	-
被调用函数	-
输入参数{in}	
channel	通道引脚
TSI_CHCFG_GxPy( x=0..5,y=0..3)	GxPy引脚不是通道引脚
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* disable G5P3 channel pin */
tsi_channel_pin_disable (TSI_CHCFG_G5P3);
```

### 函数 tsi\_software\_mode\_config

函数tsi\_software\_mode\_config描述见下表:

表 3-567. 函数 tsi\_software\_mode\_config

函数名称	tsi_software_mode_config
函数原形	void tsi_software_mode_config (void);
功能描述	配置TSI为软件触发模式
先决条件	-
被调用函数	-
输入参数{in}	
-	-
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* configure TSI triggering by software */
tsi_software_mode_config ();
```

### 函数 tsi\_software\_start

函数tsi\_software\_start描述见下表:

**表 3-568. 函数 tsi\_software\_start**

函数名称	tsi_software_start
函数原形	void tsi_software_start (void);
功能描述	软件启动一次电荷转移序列
先决条件	-
被调用函数	-
输入参数{in}	
-	-
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* start a charge-transfer sequence when TSI is in software trigger mode */
tsi_software_start ();
```

### 函数 tsi\_software\_stop

函数tsi\_software\_stop描述见下表：

**表 3-569. 函数 tsi\_software\_stop**

函数名称	tsi_software_stop
函数原形	void tsi_software_stop (void);
功能描述	软件停止已启动的电荷转移序列
先决条件	-
被调用函数	-
输入参数{in}	
-	-
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* stop a charge-transfer sequence when TSI is in software trigger mode */
tsi_software_stop ();
```

### 函数 tsi\_hardware\_mode\_config

函数tsi\_hardware\_mode\_config描述见下表：

表 3-570. 函数 tsi\_hardware\_mode\_config

函数名称	tsi_hardware_mode_config
函数原形	void tsi_hardware_mode_config (uint8_t trigger_edge);
功能描述	配置TSI为硬件触发模式
先决条件	-
被调用函数	-
输入参数{in}	
trigger_edge	触发边沿
TSI_FALLING_TRIGGER	下降沿触发
TSI_RISING_TRIGGER	上升沿触发
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* configure TSI triggering by hardware */
```

```
tsi_hardware_mode_config (TSI_FALLING_TRIGGER);
```

### 函数 tsi\_pin\_mode\_config

函数tsi\_pin\_mode\_config描述见下表：

表 3-571. 函数 tsi\_pin\_mode\_config

函数名称	tsi_pin_mode_config
函数原形	void tsi_pin_mode_config (uint8_t pin_mode);
功能描述	状态机为空闲状态时TSI引脚的模式
先决条件	-
被调用函数	-
输入参数{in}	
pin_mode	引脚模式
TSI_OUTPUT_LOW	TSI引脚输出低电平
TSI_INPUT_FLOATING	TSI引脚保持悬空输入模式
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* configure TSI pin mode when charge-transfer sequence is IDLE */
```

```
tsi_pin_mode_config (TSI_OUTPUT_LOW);
```

### 函数 tsi\_extend\_charge\_config

函数tsi\_extend\_charge\_config描述见下表:

**表 3-572. 函数 tsi\_extend\_charge\_config**

函数名称	tsi_extend_charge_config
函数原形	void tsi_extend_charge_config(ControlStatus extend,uint8_t prescaler,uint32_t max_duration);
功能描述	配置TSI扩展充电状态最大持续时间
先决条件	-
被调用函数	-
输入参数{in}	
<b>extend</b>	扩展充电状态是否使能
ENABLE	使能
DISABLE	不使能
输入参数{in}	
<b>prescaler</b>	ECCLK时钟分频系数
TSI_EXTEND_DIV1	HCLK不分频
TSI_EXTEND_DIV2	HCLK2分频
TSI_EXTEND_DIV3	HCLK3分频
TSI_EXTEND_DIV4	HCLK4分频
TSI_EXTEND_DIV5	HCLK5分频
TSI_EXTEND_DIV6	HCLK6分频
TSI_EXTEND_DIV7	HCLK7分频
TSI_EXTEND_DIV8	HCLK8分频
输入参数{in}	
<b>max_duration</b>	扩展充电状态最大持续时间
value range 1...128	时间位X*t <sub>ECCLK</sub> (X=1..128)
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* configure extend charge state */
```

```
tsi_extend_charge_config (ENABLE, TSI_EXTEND_DIV2, 10);
```

### 函数 tsi\_pulse\_config

函数tsi\_pulse\_config描述见下表:

表 3-573. 函数 tsi\_pulse\_config

函数名称	tsi_pulse_config
函数原形	void tsi_pulse_config(uint32_t prescaler,uint32_t charge_duration,uint32_t transfer_duration);
功能描述	配置TSI的电荷转移状态和充电状态的持续时间
先决条件	-
被调用函数	-
输入参数{in}	
prescaler	ECCLK时钟分频系数
TSI_CTCDIV_DIV1	HCLK不分频
TSI_CTCDIV_DIV2	HCLK2分频
TSI_CTCDIV_DIV4	HCLK4分频
TSI_CTCDIV_DIV8	HCLK8分频
TSI_CTCDIV_DIV16	HCLK16分频
TSI_CTCDIV_DIV32	HCLK32分频
TSI_CTCDIV_DIV64	HCLK64分频
TSI_CTCDIV_DIV128	HCLK128分频
TSI_CTCDIV_DIV256	HCLK256分频
TSI_CTCDIV_DIV512	HCLK512分频
TSI_CTCDIV_DIV1024	HCLK1024分频
TSI_CTCDIV_DIV2048	HCLK2048分频
TSI_CTCDIV_DIV4096	HCLK4096分频
TSI_CTCDIV_DIV8192	HCLK8192分频
TSI_CTCDIV_DIV16384	HCLK16384分频
TSI_CTCDIV_DIV32768	HCLK32768分频
输入参数{in}	
charge_duration	充电状态持续时间
TSI_CHARGE_1CT_CLK (x=1..16)	时间位X*t <sub>CTCLK</sub> (x=1..16)
输入参数{in}	
transfer_duration	电荷转移状态持续时间
TSI_TRANSFER_x CTCLK (x=1..16)	时间位X*t <sub>CTCLK</sub> (x=1..16)

输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* configure charge pulse and transfer pulse */
```

```
tsi_pulse_config (TSI_CTCDIV_DIV1024, TSI_CHARGE_10CTCLK,  
TSI_TRANSFER_8CTCLK);
```

### 函数 tsi\_max\_number\_config

函数tsi\_max\_number\_config描述见下表：

表 3-574. 函数 tsi\_max\_number\_config

函数名称	tsi_max_number_config
函数原形	void tsi_max_number_config (uint32_t max_number);
功能描述	配置TSI电荷转移序列的最大充电、转移周期数
先决条件	-
被调用函数	-
输入参数{in}	
max_number	电荷转移序列的最大充电、转移周期数
TSI_MAXNUM255	255个周期
TSI_MAXNUM511	511个周期
TSI_MAXNUM1023	1023个周期
TSI_MAXNUM2047	2047个周期
TSI_MAXNUM4095	4095个周期
TSI_MAXNUM8191	8191个周期
TSI_MAXNUM16383	16383个周期
3	
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* configure the max cycle number of a charge-transfer sequence */
```

```
tsi_max_number_config (TSI_MAXNUM1023);
```

### 函数 tsi\_hysteresis\_on

函数tsi\_hysteresis\_on描述见下表：



表 3-575. 函数 tsi\_hysteresis\_on

函数名称	tsi_hysteresis_on
函数原形	void tsi_hysteresis_on(uint32_t group_pin);
功能描述	引脚施密特触发迟滞模式使能
先决条件	-
被调用函数	-
输入参数{in}	
group_pin	引脚迟滞模式使能
TSI_PHM_GxPy (x=0..5,y=0..3)	引脚GxPy施密特触发迟滞模式使能(x=0..5,y=0..3)
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* switch on hysteresis pin */
tsi_hysteresis_on (TSI_PHM_G5P3);
```

### 函数 tsi\_hysteresis\_off

函数tsi\_hysteresis\_off描述见下表:

表 3-576. 函数 tsi\_hysteresis\_off

函数名称	tsi_hysteresis_off
函数原形	void tsi_hysteresis_off (uint32_t group_pin);
功能描述	引脚施密特触发迟滞模式禁用
先决条件	-
被调用函数	-
输入参数{in}	
group_pin	引脚迟滞模式禁用
TSI_PHM_GxPy(x= 0..5,y=0..3)	引脚GxPy施密特触发迟滞模式禁用(x=0..5,y=0..3)
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* switch off hysteresis pin */
tsi_hysteresis_off (TSI_PHM_G5P3);
```

## 函数 tsi\_analog\_on

函数tsi\_analog\_on描述见下表:

表 3-577. 函数 tsi\_analog\_on

函数名称	tsi_analog_on
函数原形	void tsi_analog_on (uint32_t group_pin);
功能描述	模拟开关闭合
先决条件	-
被调用函数	-
输入参数{in}	
group_pin	模拟开关状态
TSI_ASW_GxPy (x=0..5,y=0..3)	引脚GxPy 的模拟开关闭合(x=0..5,y=0..3)
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* switch on analog pin */
tsi_analog_on (TSI_ASW_G5P3);
```

## 函数 tsi\_analog\_off

函数tsi\_analog\_off描述见下表:

表 3-578. 函数 tsi\_analog\_off

函数名称	tsi_analog_off
函数原形	void tsi_analog_off (uint32_t group_pin);
功能描述	模拟开关断开
先决条件	-
被调用函数	-
输入参数{in}	
group_pin	模拟开关状态
TSI_ASW_GxPy (x=0..5,y=0..3)	引脚GxPy 的模拟开关断开(x=0..5,y=0..3)
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* switch off analog pin */
```

```
tsi_analog_off (TSI_ASW_G5P3);
```

### 函数 tsi\_group\_enable

函数tsi\_group\_enable描述见下表：

**表 3-579. 函数 tsi\_group\_enable**

函数名称	tsi_group_enable
函数原形	void tsi_group_enable (uint32_t group);
功能描述	TSI引脚组使能
先决条件	-
被调用函数	-
输入参数{in}	
group	引脚组
TSI_GCTL_GEx(x=0..5)	引脚组x使能
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* enable group 5 */
```

```
tsi_group_enable (TSI_GCTL_GE5);
```

### 函数 tsi\_group\_disable

函数tsi\_group\_disable描述见下表：

**表 3-580. 函数 tsi\_group\_disable**

函数名称	tsi_group_disable
函数原形	void tsi_group_disable (uint32_t group);
功能描述	TSI引脚组禁用
先决条件	-
被调用函数	-
输入参数{in}	
group	引脚组
TSI_GCTL_GEx(x=0..5)	引脚组x禁用
输出参数{out}	
-	-
返回值	

-	-
---	---

例如:

```
/* disable group 5 */
```

```
tsi_group_disable (TSI_GCTL_GE5);
```

### 函数 tsi\_group\_status\_get

函数tsi\_group\_status\_get描述见下表:

表 3-581. 函数 tsi\_group\_status\_get

函数名称	tsi_group_status_get
函数原形	FlagStatus tsi_group_status_get (uint32_t group);
功能描述	电荷转移完成状态获取
先决条件	-
被调用函数	-
输入参数{in}	
group	引脚组
TSI_GCTL_GCx(x=0..5)	组x电荷转移完成
输出参数{out}	
-	-
返回值	
FlagStatus	SET或RESET

例如:

```
/* get group complete status */
```

```
FlagStatus flag = RESET;
```

```
flag = tsi_flag_get (TSI_GCTL_GC5);
```

### 函数 tsi\_group0\_cycle\_get

函数tsi\_group0\_cycle\_get描述见下表:

表 3-582. 函数 tsi\_group0\_cycle\_get

函数名称	tsi_group0_cycle_get
函数原形	uint16_t tsi_group0_cycle_get (void);
功能描述	电荷转移序列完成时组0执行的周期数
先决条件	-
被调用函数	-
输入参数{in}	
-	-
输出参数{out}	

-	-
返回值	
uint16_t	0 - 8192

例如：

```
/* get the cycle number for group0 as soon as a charge-transfer sequence completes */
```

```
uint16_t flag = 0;
```

```
flag = tsi_group0_cycle_get ();
```

### 函数 tsi\_group1\_cycle\_get

函数tsi\_group1\_cycle\_get描述见下表：

表 3-583. 函数 tsi\_group1\_cycle\_get

函数名称	tsi_group1_cycle_get
函数原形	uint16_t tsi_group1_cycle_get (void);
功能描述	电荷转移序列完成时组1执行的周期数
先决条件	-
被调用函数	-
输入参数{in}	
-	-
输出参数{out}	
-	-
返回值	
uint16_t	0-8192

例如：

```
/* get the cycle number for group1 as soon as a charge-transfer sequence completes */
```

```
uint16_t flag = 0;
```

```
flag = tsi_group1_cycle_get ();
```

### 函数 tsi\_group2\_cycle\_get

函数tsi\_group2\_cycle\_get描述见下表：

表 3-584. 函数 tsi\_group2\_cycle\_get

函数名称	tsi_group2_cycle_get
函数原形	uint16_t tsi_group2_cycle_get (void);
功能描述	电荷转移序列完成时组2执行的周期数
先决条件	-
被调用函数	-
输入参数{in}	

-	-
输出参数{out}	
-	-
返回值	
uint16_t	0 - 8192

例如:

```
/* get the cycle number for group2 as soon as a charge-transfer sequence completes */
```

```
uint16_t flag = 0;
```

```
flag = tsi_group2_cycle_get ();
```

### 函数 tsi\_group3\_cycle\_get

函数tsi\_group3\_cycle\_get描述见下表:

表 3-585. 函数 tsi\_group3\_cycle\_get

函数名称	tsi_group3_cycle_get
函数原形	uint16_t tsi_group3_cycle_get(void);
功能描述	电荷转移序列完成时组3执行的周期数
先决条件	-
被调用函数	-
输入参数{in}	
-	-
输出参数{out}	
-	-
返回值	
uint16_t	0-8192

例如:

```
/* get the cycle number for group3 as soon as a charge-transfer sequence completes */
```

```
uint16_t flag = 0;
```

```
flag = tsi_group3_cycle_get ();
```

### 函数 tsi\_group4\_cycle\_get

函数tsi\_group4\_cycle\_get描述见下表:

表 3-586. 函数 tsi\_group4\_cycle\_get

函数名称	tsi_group4_cycle_get
函数原形	uint16_t tsi_group4_cycle_get (void);
功能描述	电荷转移序列完成时组4执行的周期数
先决条件	-

被调用函数	-
输入参数{in}	
-	-
输出参数{out}	
-	-
返回值	
uint16_t	0 - 8192

例如：

```
/* get the cycle number for group4 as soon as a charge-transfer sequence completes */
```

```
uint16_t flag = 0;
```

```
flag = tsi_group4_cycle_get ();
```

### 函数 tsi\_group5\_cycle\_get

函数tsi\_group5\_cycle\_get描述见下表：

表 3-587. 函数 tsi\_group5\_cycle\_get

函数名称	tsi_group5_cycle_get
函数原形	uint16_t tsi_group5_cycle_get (void);
功能描述	电荷转移序列完成时组5执行的周期数
先决条件	-
被调用函数	-
输入参数{in}	
-	-
输出参数{out}	
-	-
返回值	
uint16_t	0 - 8192

例如：

```
/* get the cycle number for group5 as soon as a charge-transfer sequence completes */
```

```
uint16_t flag = 0;
```

```
flag = tsi_group5_cycle_get ();
```

### 函数 tsi\_flag\_get

函数tsi\_flag\_get描述见下表：

表 3-588. 函数 tsi\_flag\_get

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

功能描述	TSI标志位获取
先决条件	-
被调用函数	-
输入参数{in}	
flag	标志位
TSI_FLAG_CTCF	电荷转移完成标志
TSI_FLAG_MNERR	最大循环次数错误标志
输出参数{out}	
-	-
返回值	
FlagStatus	SET或RESET

例如：

```
/* get TSI_FLAG_CTCF flag */
```

```
FlagStatus flag = RESET;
```

```
flag = tsi_flag_get (TSI_FLAG_CTCF);
```

### 函数 tsi\_flag\_clear

函数tsi\_flag\_clear描述见下表：

表 3-589. 函数 tsi\_flag\_clear

函数名称	tsi_flag_clear
函数原形	void tsi_flag_clear (uint32_t flag);
功能描述	TSI标志位清除
先决条件	-
被调用函数	-
输入参数{in}	
flag	标志位
TSI_FLAG_CTCF	电荷转移完成标志
TSI_FLAG_MNERR	最大循环次数错误标志
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* clear TSI_FLAG_CTCF_CLR flag */
```

```
tsi_flag_clear (TSI_FLAG_CTCF_CLR);
```



**函数 tsi\_interrupt\_enable**

函数tsi\_interrupt\_enable描述见下表:

**表 3-590. 函数 tsi\_interrupt\_enable**

函数名称	tsi_interrupt_enable
函数原形	void tsi_interrupt_enable (uint32_t source);
功能描述	TSI中断使能
先决条件	-
被调用函数	-
输入参数{in}	
source	中断使能位
TSI_INT_CCTCF	电荷转移完成标志中断使能
TSI_INT_MNERR	最大循环次数错误中断使能
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* enable TSI_INT_CCTCF interrupt */
tsi_interrupt_enable (TSI_INT_CCTCF);
```

**函数 tsi\_interrupt\_disable**

函数tsi\_interrupt\_disable描述见下表:

**表 3-591. 函数 tsi\_interrupt\_disable**

函数名称	tsi_interrupt_disable
函数原形	void tsi_interrupt_disable(uint32_t source);
功能描述	TSI中断禁能
先决条件	-
被调用函数	-
输入参数{in}	
source	中断使能位
TSI_INT_CCTCF	电荷转移完成标志中断禁能
TSI_INT_MNERR	最大循环次数错误中断禁能
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* disable TSI_INT_CCTCF interrupt */
```

```
tsi_interrupt_disable (TSI_INT_CCTCF);
```

### 函数 tsi\_interrupt\_flag\_get

函数tsi\_interrupt\_flag\_get描述见下表：

表 3-592. 函数 tsi\_interrupt\_flag\_get

函数名称	tsi_interrupt_flag_get
函数原形	FlagStatus tsi_interrupt_flag_get (uint32_t flag);
功能描述	TSI中断标志位获取
先决条件	-
被调用函数	-
输入参数{in}	
flag	中断标志位
TSI_INT_FLAG_CT CF	电荷转移完成标志中断
TSI_INT_FLAG_MN ERR	最大循环次数错误标志中断
输出参数{out}	
-	-
返回值	
FlagStatus	SET或RESET

例如：

```
/* get TSI_INT_FLAG_CTCF interrupt flag */
```

```
FlagStatus flag = RESET;
```

```
flag = tsi_interrupt_flag_get (TSI_INT_FLAG_CTCF);
```

### 函数 tsi\_interrupt\_flag\_clear

函数tsi\_interrupt\_flag\_clear描述见下表：

表 3-593. 函数 tsi\_interrupt\_flag\_clear

函数名称	tsi_interrupt_flag_clear
函数原形	void tsi_interrupt_flag_clear (uint32_t flag);
功能描述	TSI中断标志位清除
先决条件	-
被调用函数	-
输入参数{in}	
flag	中断标志位
TSI_INT_FLAG_CT CF	电荷转移完成标志中断

<i>TSI_INT_FLAG_MN ERR</i>	最大循环次数错误标志中断
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* clear TSI_INT_FLAG_CTCF interrupt flag */
tsi_interrupt_flag_clear (TSI_INT_FLAG_CTCF);
```

### 3.23. 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[3.23.1](#)描述了USART的寄存器列表，章节[3.23.2](#)对USART库函数进行说明。

#### 3.23.1. 外设寄存器说明

USART寄存器列表如下表所示:

表 3-594. 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_RFCS	接收FIFO控制和状态寄存器

#### 3.23.2. 外设库函数说明

USART库函数列表如下表所示:

表 3-595. USART 库函数

库函数名称	库函数描述
usart_deinit	复位外设USART

库函数名称	库函数描述
usart_baudrate_set	配置USART波特率
usart_parity_config	配置USART奇偶校验
usart_word_length_set	配置USART字长
usart_stop_bit_set	配置USART停止位
usart_enable	使能USART
usart_disable	失能USART
usart_transmit_config	USART发送配置
usart_receive_config	USART接收配置
usart_data_first_config	配置数据传输时低位在前或高位在前
usart_invert_config	配置USART反转功能
usart_oversample_enable	使能USART溢出禁止功能
usart_oversample_disable	失能USART溢出禁止功能
usart_oversample_config	配置USART过采样模式
usart_sample_bit_config	配置USART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_address_config	在地址掩码唤醒模式下配置USART地址
usart_address_detection_mode_config	配置USART地址检测模式
usart_mute_mode_enable	使能USART静默模式
usart_mute_mode_disable	失能USART静默模式
usart_mute_mode_wakeup_config	配置USART静默模式唤醒方式
usart_lin_mode_enable	使能USART LIN模式
usart_lin_mode_disable	失能USART LIN模式
usart_lin_break_detection_length_config	配置USART LIN模式中断帧长度
usart_halfduplex_enable	使能USART半双工模式
usart_halfduplex_disable	失能USART半双工模式
usart_clock_enable	使能USART CK引脚
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

库函数名称	库函数描述
nable	
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低功耗模式下或者SmartCard模式配置外设时钟分频系数
usart_irda_lowpower_config	配置USART IrDA低功耗模式
usart_hardware_flow_rts_config	配置USART RTS硬件控制流
usart_hardware_flow_cts_config	配置USART CTS硬件控制流
usart_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_command_enable	使能USART请求
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/RFCs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

## Enum usart\_flag\_enum

表 3-596. 枚举类型 usart\_flag\_enum

Member name	Function description
USART_FLAG_REA	接收使能通知标志
USART_FLAG_TEA	发送使能通知标志

Member name	Function description
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	ADDR匹配标志
USART_FLAG_BSY	忙标志
USART_FLAG_EB	快结束标志
USART_FLAG_RT	接收超时标志
USART_FLAG_CTS	CTS电平
USART_FLAG_CTSF	CTS变化标志
USART_FLAG_LBD	LIN断开检测标志
USART_FLAG_TBE	发送数据寄存器空
USART_FLAG_TC	发送完成标志
USART_FLAG_RBNE	读数据缓冲区非空标志
USART_FLAG_IDLE	空闲线检测标志
USART_FLAG_ORERR	溢出错误标志
USART_FLAG_NERR	噪声错误标志
USART_FLAG_FERR	帧错误标志
USART_FLAG_PERR	校验错误标志
USART_FLAG_RFF	接收FIFO满标志
USART_FLAG_RFE	接收FIFO空标志

## Enum usart\_interrupt\_flag\_enum

表 3-597. 枚举类型 usart\_interrupt\_flag\_enum

Member name	Function description
USART_INT_FLAG_EB	快结束中断标志
USART_INT_FLAG_RT	接收超时中断标志
USART_INT_FLAG_AM	ADDR匹配中断标志
USART_INT_FLAG_PERR	校验错误中断标志
USART_INT_FLAG_TBE	发送数据寄存器空中断标志
USART_INT_FLAG_TC	发送完成中断标志
USART_INT_FLAG_RBNE	读数据缓冲区非空中断标志
USART_INT_FLAG_RBNE_ORE RR	溢出错误中断标志
USART_INT_FLAG_IDLE	空闲线检测中断标志
USART_INT_FLAG_LBD	LIN断开检测中断标志
USART_INT_FLAG_WU	从深度睡眠模式唤醒中断标志
USART_INT_FLAG_CTS	CTS中断中断标志
USART_INT_FLAG_ERR_NERR	噪声错误中断标志
USART_INT_FLAG_ERR_ORER R	溢出错误中断标志

Member name	Function description
USART_INT_FLAG_ERR_FERR	帧错误中断标志
USART_INT_FLAG_RFFINT	接收FIFO满中断标志

### Enum usart\_interrupt\_enum

表 3-598. 枚举类型 usart\_interrupt\_enum

Member name	Function description
USART_INT_EB	块尾中断使能
USART_INT_RT	接收超时中断使能
USART_INT_AM	ADDR字符匹配中断使能
USART_INT_PERR	校验错误中断使能
USART_INT_TBE	发送寄存器空中断使能
USART_INT_TC	发送完成中断使能
USART_INT_RBNE	读数据缓冲区非空中断和过载错误中断使能
USART_INT_IDLE	IDLE线检测中断使能
USART_INT_LBD	LIN断开信号检测中断使能
USART_INT_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

### Enum usart\_invert\_enum

表 3-599. 枚举类型 usart\_invert\_enum

Member name	Function description
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 3-600. 函数 usart\_deinit

函数名称	usart_deinit
函数原型	void usart_deinit(uint32_t usart_periph);
功能描述	复位外设USARTx
先决条件	-

被调用函数	rcu_periph_reset_enable / rcu_periph_reset_disable
输入参数{in}	
usart_periph	外设USARTx
USARTx	x=0,1
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 3-601. 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
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 3-602. 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
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 3-603. 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
输入参数{in}	
wlen	配置USART字长
USART_WL_8BIT	8位
USART_WL_9BIT	9位
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 3-604. 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
输入参数{in}	
stblen	配置USART停止位
USART_STB_1BIT	1位
USART_STB_0_5BIT	0.5位
USART_STB_2BIT	2位
USART_STB_1_5BIT	1.5位
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* configure USART0 stop bit length */
```

```
usart_stop_bit_set(USART0, USART_STB_1_5BIT);
```

### 函数 usart\_enable

函数usart\_enable描述见下表:

**表 3-605. 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
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 3-606. 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
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 3-607. 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
输入参数{in}	
txconfig	使能/失能USART发送器
USART_TRANSMIT_ENABLE	使能USART发送
USART_TRANSMIT_DISABLE	失能USART发送
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* configure USART0 transmitter */
```

```
usart_transmit_config(USART0, USART_TRANSMIT_ENABLE);
```

### 函数 usart\_receive\_config

函数usart\_receive\_config描述见下表：

表 3-608. 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
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 3-609. 函数 usart\_data\_first\_config**

函数名称	usart_data_first_config
函数原型	void usart_data_first_config(uint32_t usart_periph, uint32_t msbf);
功能描述	配置数据传输时低位在前或高位在前
先决条件	-
被调用函数	-
输入参数{in}	
usart_periph	外设USARTx
USARTx	x=0,1
输入参数{in}	
msbf	数据传输时低位在前/高位在前
USART_MSBF_LS B	数据传输时低位在前
USART_MSBF_MS B	数据传输时高位在前
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 3-610. 函数 usart\_invert\_config**

函数名称	usart_invert_config
函数原型	void usart_invert_config(uint32_t usart_periph, usart_invert_enum invertpara);
功能描述	配置USART反转功能
先决条件	-

被调用函数	-
输入参数{in}	
usart_periph	外设USARTx
USARTx	x=0,1
输入参数{in}	
invertpara	参考 <a href="#">表3-599. 枚举类型usart_invert_enum</a>
USART_DINV_ENABLE	数据位电平反转
USART_DINV_DISABLE	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 3-611. 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
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* enable USART0 overrun */
usart_overrun_enable (USART0);
```

### 函数 usart\_overrun\_disable

函数usart\_overrun\_disable描述见下表:

表 3-612. 函数 usart\_overrun\_disable

函数名称	usart_overrun_disable
函数原型	void usart_overrun_disable (uint32_t usart_periph);
功能描述	失能USART溢出禁止功能
先决条件	-
被调用函数	-
输入参数{in}	
usart_periph	外设USARTx
USARTx	x=0,1
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* disable USART0 overrun */
usart_overrun_disable (USART0);
```

### 函数 usart\_oversample\_config

函数usart\_oversample\_config描述见下表:

表 3-613. 函数 usart\_oversample\_config

函数名称	usart_oversample_config
函数原型	void usart_oversample_config(uint32_t usart_periph,uint32_t oversamp);
功能描述	配置USART过采样模式
先决条件	-
被调用函数	-
输入参数{in}	

<b>usart_periph</b>	外设USARTx
USARTx	x=0,1
<b>输入参数{in}</b>	
<b>oversamp</b>	过采样值
USART_OVSMOD_8	8倍过采样
USART_OVSMOD_16	16倍过采样
<b>输出参数{out}</b>	
-	-
<b>返回值</b>	
-	-

例如：

```
/* configure 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 3-614. 函数 usart\_sample\_bit\_config**

<b>函数名称</b>	usart_sample_bit_config
<b>函数原型</b>	void usart_sample_bit_config(uint32_t usart_periph, uint32_t osb);
<b>功能描述</b>	配置USART单次采样方式
<b>先决条件</b>	-
<b>被调用函数</b>	-
<b>输入参数{in}</b>	
<b>usart_periph</b>	外设USARTx
USARTx	x=0,1
<b>输入参数{in}</b>	
<b>osb</b>	单次采样方式
USART_OSB_1BIT	1次采样方法
USART_OSB_3BIT	3次采样方法
<b>输出参数{out}</b>	
-	-
<b>返回值</b>	
-	-

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 3-615. 函数 usart\_receiver\_timeout\_enable**

函数名称	usart_receiver_timeout_enable
函数原型	void usart_receiver_timeout_enable(uint32_t usart_periph);
功能描述	使能USART接收超时
先决条件	-
被调用函数	-
输入参数{in}	
usart_periph	外设USARTx
USARTx	x=0
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* enable USART0 receiver timeout */
usart_receiver_timeout_enable(USART0);
```

**函数 usart\_receiver\_timeout\_disable**

函数usart\_receiver\_timeout\_disable描述见下表：

**表 3-616. 函数 usart\_receiver\_timeout\_disable**

函数名称	usart_receiver_timeout_disable
函数原型	void usart_receiver_timeout_disable(uint32_t usart_periph);
功能描述	失能USART接收超时
先决条件	-
被调用函数	-
输入参数{in}	
usart_periph	外设USARTx
USARTx	x=0
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* disable USART0 receiver timeout */
usart_receiver_timeout_disable(USART0);
```

### 函数 `usart_receiver_timeout_threshold_config`

函数 `usart_receiver_timeout_threshold_config` 描述见下表：

**表 3-617. 函数 `usart_receiver_timeout_threshold_config`**

函数名称	<code>usart_receiver_timeout_threshold_config</code>
函数原型	<code>void usart_receiver_timeout_threshold_config(uint32_t usart_periph, uint32_t rtimeout);</code>
功能描述	设置USART接收超时阈值
先决条件	-
被调用函数	-
输入参数{in}	
<code>usart_periph</code>	外设USARTx
<code>USARTx</code>	x=0
输入参数{in}	
<code>rtimeout</code>	超时时间
0x00000000- 0x00FFFFFF	超时时间值
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* set the receiver timeout threshold of USART0 */
```

```
usart_receiver_timeout_threshold_config(USART0, 115200*3);
```

### 函数 `usart_data_transmit`

函数 `usart_data_transmit` 描述见下表：

**表 3-618. 函数 `usart_data_transmit`**

函数名称	<code>usart_data_transmit</code>
函数原型	<code>void usart_data_transmit(uint32_t usart_periph, uint16_t data);</code>
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}	
<code>usart_periph</code>	外设USARTx
<code>USARTx</code>	x=0,1
输入参数{in}	
<code>data</code>	发送的数据
0-0x1FF	发送的数据

输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* USART0 transmit data */
```

```
usart_data_transmit(USART0, 0xAA);
```

### 函数 usart\_data\_receive

函数usart\_data\_receive描述见下表：

**表 3-619. 函数 usart\_data\_receive**

函数名称	usart_data_receive
函数原型	void usart_data_receive(uint32_t usart_periph);
功能描述	USART接收数据功能
先决条件	-
被调用函数	-
输入参数{in}	
usart_periph	外设USARTx
USARTx	x=0,1
输出参数{out}	
-	-
返回值	
uint32_t	接收到的数据（0-0xFF）

例如：

```
/* USART0 receive data */
```

```
uint16_t temp ;
```

```
temp = usart_data_receive(USART0);
```

### 函数 usart\_address\_config

函数usart\_address\_config描述见下表：

**表 3-620. 函数 usart\_address\_config**

函数名称	usart_address_config
函数原型	void usart_address_config(uint32_t usart_periph, uint8_t addr);
功能描述	在地址掩码唤醒模式下配置USART地址
先决条件	-
被调用函数	-
输入参数{in}	

<b>usart_periph</b>	外设USARTx
<i>USARTx</i>	x=0,1
<b>输入参数{in}</b>	
<b>addr</b>	USART地址
<i>0-0xFF</i>	USART地址
<b>输出参数{out}</b>	
-	-
<b>返回值</b>	
-	-

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 3-621. 函数 usart\_address\_detection\_mode\_config**

<b>函数名称</b>	usart_address_detection_mode_config
<b>函数原型</b>	void usart_address_detection_mode_config(uint32_t usart_periph, uint32_t addmod);
<b>功能描述</b>	配置USART地址检测模式
<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=0,1
<b>输入参数{in}</b>	
<b>addmod</b>	地址检测模式
<i>USART_ADDM_4BIT</i>	4位地址检测
<i>USART_ADDM_FULLBIT</i>	全位地址检测
<b>输出参数{out}</b>	
-	-
<b>返回值</b>	
-	-

例如:

```
/*configure address detection mode */
```

```
usart_address_config(USART0, USART_ADDM_4BIT);
```

**函数 usart\_mute\_mode\_enable**

函数usart\_mute\_mode\_enable描述见下表:

**表 3-622. 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
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 3-623. 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
-	-
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 3-624. 函数 `usart_mute_mode_wakeup_config`**

函数名称	<code>usart_mute_mode_wakeup_config</code>
函数原型	<code>void usart_mute_mode_wakeup_config(uint32_t usart_periph, uint32_t wmethod);</code>
功能描述	配置USART静默模式唤醒方式
先决条件	-
被调用函数	-
输入参数{in}	
<code>usart_periph</code>	外设USARTx
<code>USARTx</code>	x=0,1
输入参数{in}	
<code>wmethod</code>	两种方法用于进入或退出静默模式
<code>USART_WM_IDLE</code>	空闲线唤醒
<code>USART_WM_ADDR</code>	地址掩码唤醒
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* configure USART0 wakeup method in mute mode */
```

```
usart_mute_mode_wakeup_config(USART0, USART_WM_IDLE);
```

## 函数 `usart_lin_mode_enable`

函数 `usart_lin_mode_enable` 描述见下表：

**表 3-625. 函数 `usart_lin_mode_enable`**

函数名称	<code>usart_lin_mode_enable</code>
函数原型	<code>void usart_lin_mode_enable(uint32_t usart_periph);</code>
功能描述	使能USART LIN模式
先决条件	-
被调用函数	-
输入参数{in}	
<code>usart_periph</code>	外设USARTx
<code>USARTx</code>	x=0
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 3-626. 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 3-627. 函数 usart\_lin\_break\_dection\_length\_config

函数名称	usart_lin_break_dection_length_config
函数原型	void usart_lin_break_dection_length_config(uint32_t usart_periph, uint32_t lflen);
功能描述	配置USART LIN模式中断帧长度
先决条件	-
被调用函数	-
输入参数{in}	
usart_periph	外设USARTx
USARTx	x=0
输入参数{in}	

<b>lblen</b>	LIN模式中断帧长度
<i>USART_LBLEN_10</i> <i>B</i>	断开帧长度为10 bits
<i>USART_LBLEN_11</i> <i>B</i>	断开帧长度为11 bits
<b>输出参数{out}</b>	
-	-
<b>返回值</b>	
-	-

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 3-628. 函数 usart\_halfduplex\_enable

<b>函数名称</b>	usart_halfduplex_enable
<b>函数原型</b>	void usart_halfduplex_enable(uint32_t usart_periph);
<b>功能描述</b>	使能USART半双工模式
<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=0,1
<b>输出参数{out}</b>	
-	-
<b>返回值</b>	
-	-

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 3-629. 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);



功能描述	失能USART半双工模式
先决条件	-
被调用函数	-
输入参数{in}	
usart_periph	外设USARTx
USARTx	x=0,1
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 3-630. 函数 usart\_clock\_enable

函数名称	usart_clock_enable
函数原型	void usart_clock_enable(uint32_t usart_periph);
功能描述	使能USART CK引脚
先决条件	-
被调用函数	-
输入参数{in}	
usart_periph	外设USARTx
USARTx	x=0
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* enable USART0 CK pin */
```

```
usart_synchronous_clock_enable(USART0);
```

### 函数 usart\_clock\_disable

函数usart\_clock\_disable描述见下表：

表 3-631. 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
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* disable USART0 CK pin */
```

```
usart_clock_disable(USART0);
```

### 函数 usart\_synchronous\_clock\_config

函数usart\_synchronous\_clock\_config描述见下表：

表 3-632. 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
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 3-633. 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}	
guat	保护时间值
0-0x000000FF	保护时间值
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* configure USART0 guard time value in smartcard mode */
usart_guard_time_config(USART0, 0x0000 0055);
```

### 函数 usart\_smartcard\_mode\_enable

函数usart\_smartcard\_mode\_enable描述见下表：

表 3-634. 函数 usart\_smartcard\_mode\_enable

函数名称	usart_smartcard_mode_enable
函数原型	void usart_smartcard_mode_enable(uint32_t usart_periph);

功能描述	使能USART智能卡模式
先决条件	-
被调用函数	-
输入参数{in}	
usart_periph	外设USARTx
USARTx	x=0
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* USART0 smartcard mode enable */
```

```
usart_smartcard_mode_enable(USART0);
```

### 函数 usart\_smartcard\_mode\_disable

函数usart\_smartcard\_mode\_disable描述见下表：

表 3-635. 函数 usart\_smartcard\_mode\_disable

函数名称	usart_smartcard_mode_disable
函数原型	void usart_smartcard_mode_disable(uint32_t usart_periph);
功能描述	失能USART智能卡模式
先决条件	-
被调用函数	-
输入参数{in}	
usart_periph	外设USARTx
USARTx	x=0
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* USART0 smartcard mode disable */
```

```
usart_smartcard_mode_disable(USART0);
```

### 函数 usart\_smartcard\_mode\_nack\_enable

函数usart\_smartcard\_mode\_nack\_enable描述见下表：

表 3-636. 函数 usart\_smartcard\_mode\_nack\_enable

函数名称	usart_smartcard_mode_nack_enable
------	----------------------------------

函数原型	void usart_smartcard_mode_nack_enable(uint32_t usart_periph);
功能描述	在USART智能卡模式下使能NACK
先决条件	-
被调用函数	-
输入参数{in}	
usart_periph	外设USARTx
USARTx	x=0
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* enable USART0 NACK in smartcard mode */
usart_smartcard_mode_nack_enable(USART0);
```

### 函数 usart\_smartcard\_mode\_nack\_disable

函数usart\_smartcard\_mode\_nack\_disable描述见下表：

表 3-637. 函数 usart\_smartcard\_mode\_nack\_disable

函数名称	usart_smartcard_mode_nack_disable
函数原型	void usart_smartcard_mode_nack_disable(uint32_t usart_periph);
功能描述	在USART智能卡模式下失能NACK
先决条件	-
被调用函数	-
输入参数{in}	
usart_periph	外设USARTx
USARTx	x=0
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 3-638. 函数 `usart_smartcard_mode_early_nack_enable`

函数名称	<code>usart_smartcard_mode_early_nack_enable</code>
函数原型	<code>void usart_smartcard_mode_early_nack_enable (uint32_t usart_periph);</code>
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}	
<code>usart_periph</code>	外设USARTx
<code>USARTx</code>	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 3-639. 函数 `usart_smartcard_mode_early_nack_disable`

函数名称	<code>usart_smartcard_mode_early_nack_disable</code>
函数原型	<code>void usart_smartcard_mode_early_nack_disable (uint32_t usart_periph);</code>
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}	
<code>usart_periph</code>	外设USARTx
<code>USARTx</code>	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 3-640. 函数 `usart_smartcard_autoretry_config`

函数名称	<code>usart_smartcard_autoretry_config</code>
函数原型	<code>void usart_smartcard_autoretry_config(uint32_t usart_periph, uint32_t scrtnum);</code>
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}	
<code>usart_periph</code>	外设USARTx
<code>USARTx</code>	x=0
输入参数{in}	
<code>scrtnum</code>	智能卡自动重试次数
<code>0-0x00000007</code>	自动重试次数
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 3-641. 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
<code>0-0x000000FF</code>	块长度
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 3-642. 函数 usart\_irda\_mode\_enable

函数名称	usart_irda_mode_enable
函数原型	void usart_irda_mode_enable(uint32_t usart_periph);
功能描述	使能USART串行红外编解码功能模块
先决条件	-
被调用函数	-
输入参数{in}	
usart_periph	外设USARTx
USARTx	x=0
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 3-643. 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 3-644. 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
0-0xFF	时钟分频系数
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* configure the USART0 peripheral clock prescaler in USART IrDA low-power mode */
```

```
usart_prescaler_config(USART0, 0x00);
```

### 函数 usart\_irda\_lowpower\_config

函数usart\_irda\_lowpower\_config描述见下表:

表 3-645. 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 3-646. 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
输入参数{in}	
rtsconfig	使能/失能RTS
USART_RTS_ENABLE	使能RTS
USART_RTS_DISABLE	失能RTS
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* configure USART0 hardware flow control RTS */
```

```
usart_hardware_flow_rts_config(USART0, USART_RTS_ENABLE);
```

**函数 usart\_hardware\_flow\_cts\_config**

函数usart\_hardware\_flow\_cts\_config描述见下表：

**表 3-647. 函数 usart\_hardware\_flow\_cts\_config**

函数名称	usart_hardware_flow_cts_config
函数原型	void usart_hardware_flow_cts_config(uint32_t usart_periph, uint32_t ctsconfig);
功能描述	配置USART CTS硬件控制流
先决条件	-
被调用函数	-
输入参数{in}	
usart_periph	外设USARTx
USARTx	x=0,1
输入参数{in}	
ctsconfig	使能/失能CTS
USART_CTS_ENA BLE	使能CTS
USART_CTS_DISA BLE	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\_rs485\_driver\_enable**

函数usart\_rs485\_driver\_enable描述见下表：

**表 3-648. 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
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 3-649. 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
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 3-650. 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
输入参数{in}	

<b>deatime</b>	驱动使能置位时间
0-0x0000001F	驱动使能置位时间
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 3-651. 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}	
<b>usart_periph</b>	外设USARTx
USARTx	x=0,1
输入参数{in}	
<b>dedtime</b>	驱动使能置低时间
0-0x0000001F	驱动使能置低时间
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 3-652. 函数 usart\_depolarity\_config

函数名称	usart_depolarity_config
函数原型	void usart_depolarity_config(uint32_t usart_periph, uint32_t dep);

功能描述	配置USART驱动使能极性模式
先决条件	-
被调用函数	-
输入参数{in}	
usart_periph	外设USARTx
USARTx	x=0,1
输入参数{in}	
dep	驱动使能的极性选择模式
USART_DEP_HIGH	DE信号高有效
USART_DEP_LOW	DE信号低有效
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* configure driver enable polarity mode */
usart_driver_depolarity_config(USART0, USART_DEP_HIGH);
```

## 函数 usart\_dma\_receive\_config

函数usart\_dma\_receive\_config描述见下表:

表 3-653. 函数 usart\_dma\_receive\_config

函数名称	usart_dma_receive_config
函数原型	void usart_dma_receive_config(uint32_t usart_periph, uint8_t dmacmd);
功能描述	配置USART DMA接收功能
先决条件	-
被调用函数	-
输入参数{in}	
usart_periph	外设USARTx
USARTx	x=0,1
输入参数{in}	
dmacmd	USART DMA模式
USART_RECEIVE_DMA_ENABLE	使能DMA接收功能
USART_RECEIVE_DMA_DISABLE	失能DMA接收功能
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* USART0 DMA enable for reception */
```

```
usart_dma_receive_config(USART0, USART_RECEIVE_DMA_ENABLE);
```

### 函数 usart\_dma\_transmit\_config

函数usart\_dma\_transmit\_config描述见下表:

表 3-654. 函数 usart\_dma\_transmit\_config

函数名称	usart_dma_transmit_config
函数原型	void usart_dma_transmit_config(uint32_t usart_periph, uint8_t dmacmd);
功能描述	配置 USART DMA发送功能
先决条件	-
被调用函数	-
输入参数{in}	
usart_periph	外设USARTx
USARTx	x=0,1
输入参数{in}	
dmacmd	USART DMA模式
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\_enable

函数usart\_reception\_error\_dma\_enable描述见下表:

表 3-655. 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}	

<b>usart_periph</b>	外设USARTx
USARTx	x=0,1
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* enable DMA on reception error */
```

```
usart_reception_error_dma_enable(USART0);
```

### 函数 usart\_reception\_error\_dma\_disable

函数usart\_reception\_error\_dma\_disable描述见下表：

**表 3-656. 函数 usart\_reception\_error\_dma\_disable**

<b>函数名称</b>	usart_reception_error_dma_disable
<b>函数原型</b>	void usart_reception_error_dma_disable (uint32_t usart_periph);
<b>功能描述</b>	USART接收错误时失能DMA
<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
USARTx	x=0,1
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* disable DMA on reception error */
```

```
usart_reception_error_dma_disable (USART0);
```

### 函数 usart\_wakeup\_enable

函数usart\_reception\_wakeup\_enable描述见下表：

**表 3-657. 函数 usart\_wakeup\_enable**

<b>函数名称</b>	usart_wakeup_enable
<b>函数原型</b>	void usart_wakeup_enable(uint32_t usart_periph);
<b>功能描述</b>	使能USART唤醒
<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
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\_reception\_wakeup\_disable描述见下表:

**表 3-658. 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}	
<b>usart_periph</b>	外设USARTx
<i>USARTx</i>	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\_reception\_mode\_config描述见下表:

**表 3-659. 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}	
wum	唤醒模式
USART_WUM_ADD R	WUF在地址匹配时置位
USART_WUM_STA RTB	WUF在检测到起始位时置位
USART_WUM_RBN E	WUF在检测到RBNE时置位
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* configure USART0 wake up mode */
```

```
usart_wakeup_mode_config(USART0, USART_WUM_ADDR);
```

### 函数 usart\_command\_enable

函数usart\_command\_enable描述见下表：

表 3-660. 函数 usart\_command\_enable

函数名称	usart_command_enable
函数原型	void usart_command_enable(uint32_t usart_periph, uint32_t cmdtype);
功能描述	使能USART请求
先决条件	-
被调用函数	-
输入参数{in}	
usart_periph	外设USARTx
USARTx	x=0,1
输入参数{in}	
cmdtype	请求类型
USART_CMD_SBK CMD	发送断开帧请求
USART_CMD_MM CMD	静模式请求
USART_CMD_RXF CMD	接收数据清空请求

USART_CMD_TXF CMD	发送数据清空请求
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* enable USART0 command */
```

```
usart_command_enable(USART0, USART_CMD_SBKCMD);
```

### 函数 usart\_receive\_fifo\_enable

函数usart\_receive\_fifo\_enable描述见下表：

表 3-661. 函数 usart\_receive\_fifo\_enable

函数名称	usart_receive_fifo_enable
函数原型	void usart_receive_fifo_enable(uint32_t usart_periph);
功能描述	使能接收FIFO
先决条件	-
被调用函数	-
输入参数{in}	
usart_periph	外设USARTx
USARTx	x=0,1
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 3-662. 函数 usart\_receive\_fifo\_disable

函数名称	usart_receive_fifo_disable
函数原型	void usart_receive_fifo_disable(uint32_t usart_periph);
功能描述	失能接收FIFO
先决条件	-
被调用函数	-

输入参数{in}	
<b>usart_periph</b>	外设USARTx
<i>USARTx</i>	x=0,1
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 3-663. 函数 usart\_receive\_fifo\_counter\_number**

函数名称	usart_receive_fifo_counter_number
函数原型	uint8_t usart_receive_fifo_counter_number(uint32_t usart_periph);
功能描述	读取接收FIFO计数器的值
先决条件	-
被调用函数	-
输入参数{in}	
<b>usart_periph</b>	外设USARTx
<i>USARTx</i>	x=0,1
输出参数{out}	
-	-
返回值	
<b>uint8_t</b>	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 3-664. 函数 usart\_flag\_get**

函数名称	usart_flag_get
函数原型	FlagStatus usart_flag_get(uint32_t usart_periph, usart_flag_enum flag);
功能描述	获取USART STAT/RFCs寄存器标志位

先决条件	-
被调用函数	-
输入参数{in}	
usart_periph	外设USARTx
USARTx	x=0,1
输入参数{in}	
flag	USART标志位，参考 <a href="#">表3-596. 枚举类型usart_flag_enum</a>
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 3-665. 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
输入参数{in}	
flag	USART标志位，参考 <a href="#">表3-596. 枚举类型usart_flag_enum</a>
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_OERR	溢出错误标志
USART_FLAG_IDLE	空闲线检测标志

USART_FLAG_TC	发送完成标志
USART_FLAG_LBD	LIN断开检测标志
USART_FLAG_CTS F	CTS变化标志
USART_FLAG_RT	接收超时标志
USART_FLAG_EB	块结束标志
USART_FLAG_AM	ADDR匹配标志
USART_FLAG_WU	从深度睡眠模式唤醒标志
USART_FLAG_EPE RR	校验错误超前检测标志
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* clear USART0 flag */
usart_flag_clear(USART0,USART_FLAG_TC);
```

## 函数 usart\_interrupt\_enable

函数usart\_interrupt\_enable描述见下表：

表 3-666. 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
输入参数{in}	
interrupt	USART中断，参考 <a href="#">表3-598. 枚举类型usart_interrupt_enum</a>
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 3-667. 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
输入参数{in}	
interrupt	USART中断，参考 <a href="#">表3-598. 枚举类型usart_interrupt_enum</a>
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 3-668. 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
输入参数{in}	
int_flag	USART中断标志，参考 <a href="#">表3-597. 枚举类型usart_interrupt_flag_enum</a>

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 3-669. 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
输入参数{in}	
int_flag	USART中断标志, 参考 <a href="#">表3-597. 枚举类型usart_interrupt_flag_enum</a>
USART_INT_FLAG_PERR	校验错误中断标志
USART_INT_FLAG_ERR_FERR	帧错误中断标志
USART_INT_FLAG_ERR_NERR	噪声错误中断标志
USART_INT_FLAG_RBNE_ORERR	读数据缓冲区非空中断和溢出错误中断标志
USART_INT_FLAG_ERR_ORERR	过载错误中断标志
USART_INT_FLAG_IDLE	IDLE线检测中断标志
USART_INT_FLAG_TC	发送完成中断标志
USART_INT_FLAG_LBD	LIN断开检测中断标志



USART_INT_FLAG _CTS	CTS变化中断标志
USART_INT_FLAG _RT	接收超时事件中断标志
USART_INT_FLAG _EB	块结束事件中断标志
USART_INT_FLAG _AM	ADDR匹配中断标志
USART_INT_FLAG _WU	从深度睡眠模式唤醒中断标志
USART_INT_FLAG _RFF	接收FIFO满中断标志
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* clear the USART0 interrupt flag */
usart_interrupt_flag_clear(USART0, USART_INT_FLAG_TC);
```

## 3.24. 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[3.24.1](#)描述了WWDGT的寄存器列表，章节[3.24.2](#)对WWDGT库函数进行说明。

### 3.24.1. 外设寄存器说明

WWDGT寄存器列表如下表所示:

**表 3-670. WWDGT 寄存器**

寄存器名称	寄存器描述
WWDGT_CTL	WWDGT控制寄存器
WWDGT_CFG	WWDGT配置寄存器
WWDGT_STAT	WWDGT状态寄存器

### 3.24.2. 外设库函数说明

WWDGT库函数列表如下表所示:

表 3-671. WWDGT 库函数

库函数名称	库函数说明
wwdgt_deinit	将WWDGT寄存器重设为缺省值
wwdgt_enable	使能WWDGT
wwdgt_counter_update	设置WWDGT计数器更新值
wwdgt_config	设置WWDGT计数器值、窗口值和预分频值
wwdgt_interrupt_enable	使能WWDGT提前唤醒中断
wwdgt_flag_get	检查WWDGT提前唤醒中断标志位是否置位
wwdgt_flag_clear	清除WWDGT提前唤醒中断标志位状态

### 函数 wwdgt\_deinit

函数wwdgt\_deinit描述见下表:

表 3-672. 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	-
输入参数{in}	
-	-
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* reset the window watchdog timer configuration */
```

```
wwdgt_deinit ( );
```

### 函数 wwdgt\_enable

函数wwdgt\_enable描述见下表:

表 3-673. 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	-
输入参数{in}	
-	-
输出参数{out}	
-	-

返回值	
-	-

例如：

```
/* start the WWDGT counter */
```

```
wwdgt_enable ( );
```

### 函数 wwdgt\_counter\_update

函数wwdgt\_counter\_update描述见下表：

**表 3-674. 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	-
输入参数{in}	
<b>counter_value</b>	计数器值，数值范围为0x00 - 0x7F
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* update WWDGT counter to 0x7F */
```

```
wwdgt_counter_update (127);
```

### 函数 wwdgt\_config

函数wwdgt\_config描述见下表：

**表 3-675. 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	-
输入参数{in}	
<b>counter</b>	定时器计数值，数值范围0x00 - 0x7F
输入参数{in}	
<b>window</b>	窗口值，数值范围0x00 - 0x7F
输入参数{in}	
<b>prescaler</b>	WWDGT预分频值
<b>WWDGT_CFG_PSC</b>	WWDGT计数器时钟为（PCLK/4096）/1

<code>_DIV1</code>	
<code>WWDGT_CFG_PSC</code> <code>_DIV2</code>	WWDGT计数器时钟为 (PCLK/4096) /2
<code>WWDGT_CFG_PSC</code> <code>_DIV4</code>	WWDGT计数器时钟为 (PCLK/4096) /4
<code>WWDGT_CFG_PSC</code> <code>_DIV8</code>	WWDGT计数器时钟为 (PCLK/4096) /8
输出参数{out}	
-	-
Return value	
-	-

例如:

```
/* configure WWDGT counter value to 0x7F, window value to 0x50, prescaler divider value to 8 */
```

```
wwdgt_config (127, 80, WWDGT_CFG_PSC_DIV8);
```

### 函数 `wwdgt_interrupt_enable`

函数 `wwdgt_interrupt_enable` 描述见下表:

**表 3-676. 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	-
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 3-677. 函数 `wwdgt_flag_get`**

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

函数原型	FlagStatus wwdgt_flag_get(void);
功能描述	检查WWDGT提前唤醒中断标志位是否置位
先决条件	-
输入参数{in}	
-	-
输出参数{out}	
-	-
返回值	
FlagStatus	SET or RESET

例如：

```
/* test if the counter value update has reached the 0x40 */
```

```
FlagStatus status;
```

```
status = wwdgt_flag_get ( );
```

### 函数 wwdgt\_flag\_clear

函数wwdgt\_flag\_clear描述见下表：

表 3-678. 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	-
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 ( );
```

### 3.25. USBFS

USBFS 的固件库函数介绍请参考文档 **GD32F3x0-Firmware-Library-USB User Manual\_V1.0**.

## 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	2019 年 6 月 1 日
1.1	1. 修改章节 <b>3.10.2</b> 与 <b>3.15.2</b> 的库函数表格 2. 章节 <b>3.17.2</b> 增加函数 <code>rcu_deepsleep_voltage_set</code> 的输入变量 <code>dsvol</code> 的描述	2021 年 6 月 1 日
1.2	1. 新增 GD32F310 系列产品	2022 年 1 月 6 日
1.3	1. 章节 <b>3.12. FWDGT</b> 内容修改。 2. 章节 <b>3.16. PMU</b> 内容修改。 3. 章节 <b>3.19 SPI/I2S</b> 内容修改。 4. 章节 <b>3.23. USART</b> 内容修改。	2022 年 6 月 30 日
1.4	1. DAC CMP 一致性修改。	2023 年 12 月 31 日
1.5	1. 添加 GD32F355 和 GD32F370。	2024 年 3 月 20 日
1.6	1. <b>表 3-70. 函数 <code>cmp_output_init</code></b> 修改 <code>CMP_OUTPUT_TIMER0_BKIN</code> 为 <code>CMP_OUTPUT_TIMER0_BRKIN</code> 。 2. <b>函数 <code>i2s_init</code></b> : I2S 参数宏从 <code>I2S_STD_PHILLIPS</code> 修改为 <code>I2S_STD_PHILIPS</code> 。 3. 章节 <b>3.19 SPI/I2S</b> : 删除函数 <code>spi_quad_io23_output_enable</code> 、函数 <code>spi_quad_io23_output_disable</code> 。 4. 章节 <b>2. 固件库概述</b> : 增加 <code>GD32EBuilder_project</code> 、 <code>GD32EBuilder</code> 环境描述。	2026 年 1 月 1 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.