

**GigaDevice Semiconductor Inc.**

**GD32E23x**  
**Arm<sup>®</sup> Cortex<sup>®</sup>-M23 32-bit MCU**

**固件库  
使用指南**

**1.0 版本**

**(2020 年 12 月)**

# 目录

目录 .....	1
图索引 .....	4
表索引 .....	5
<b>1. 介绍 .....</b>	<b>18</b>
<b>1.1. 文档和固件库规则 .....</b>	<b>18</b>
1.1.1. 外设缩写 .....	18
1.1.2. 命名规则 .....	19
<b>2. 固件库概述 .....</b>	<b>20</b>
<b>2.1. 文件组织结构 .....</b>	<b>20</b>
2.1.1. Examples 文件夹 .....	21
2.1.2. Firmware 文件夹 .....	21
2.1.3. Template 文件夹 .....	21
2.1.4. Utilities 文件夹 .....	24
<b>2.2. 固件库文件描述 .....</b>	<b>24</b>
<b>3. 外设固件库 .....</b>	<b>25</b>
<b>3.1. 外设固件库概述 .....</b>	<b>25</b>
<b>3.2. ADC .....</b>	<b>25</b>
3.2.1. 外设寄存器描述 .....	25
3.2.2. 外设库函数说明 .....	26
<b>3.3. CMP .....</b>	<b>49</b>
3.3.1. 外设寄存器说明 .....	49
3.3.2. 外设库函数说明 .....	49
<b>3.4. CRC .....</b>	<b>55</b>
3.4.1. 外设寄存器说明 .....	55
3.4.2. 外设库函数说明 .....	56
<b>3.5. DBG .....</b>	<b>63</b>
3.5.1. 外设寄存器说明 .....	63
3.5.2. 外设库函数说明 .....	63
<b>3.6. DMA .....</b>	<b>68</b>
3.6.1. 外设寄存器说明 .....	68
3.6.2. 外设库函数说明 .....	68
<b>3.7. EXTI .....</b>	<b>85</b>
3.7.1. 外设寄存器说明 .....	85

---

3.7.2. 外设库函数说明 .....	86
<b>3.8. FMC .....</b>	<b>94</b>
3.8.1. 外设寄存器说明 .....	94
3.8.2. 外设库函数说明 .....	94
<b>3.9. FWDGT .....</b>	<b>111</b>
3.9.1. 外设寄存器说明 .....	111
3.9.2. 外设库函数说明 .....	112
<b>3.10. GPIO .....</b>	<b>117</b>
3.10.1. 外设寄存器说明 .....	117
3.10.2. 外设库函数说明 .....	118
<b>3.11. I2C .....</b>	<b>128</b>
3.11.1. 外设寄存器说明 .....	128
3.11.2. 外设库函数说明 .....	129
<b>3.12. MISC .....</b>	<b>153</b>
3.12.1. 外设寄存器说明 .....	153
3.12.2. 外设库函数说明 .....	154
<b>3.13. PMU .....</b>	<b>159</b>
3.13.1. 外设寄存器说明 .....	159
3.13.2. 外设库函数说明 .....	159
<b>3.14. RCU .....</b>	<b>167</b>
3.14.1. 外设寄存器说明 .....	167
3.14.2. 外设库函数说明 .....	168
<b>3.15. RTC .....</b>	<b>194</b>
3.15.1. 外设寄存器描述 .....	194
3.15.2. 外设库函数描述 .....	194
<b>3.16. SPI .....</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 .....	215
<b>3.17. SYSCFG .....</b>	<b>244</b>
3.17.1. 外设寄存器说明 .....	244
3.17.2. 外设库函数说明 .....	244
<b>3.18. TIMER .....</b>	<b>251</b>
3.18.1. 外设寄存器说明 .....	251
3.18.2. 外设库函数说明 .....	252
<b>3.19. USART .....</b>	<b>310</b>
3.19.1. 外设寄存器说明 .....	310
3.19.2. 外设库函数说明 .....	311
<b>3.20. WWDGT .....</b>	<b>361</b>
3.20.1. 外设寄存器说明 .....	361

---

3.20.2. 外设库函数说明 .....	362
<b>4. 版本历史 .....</b>	<b>367</b>

## 图索引

图 2-1. GD32E23x 固件库文件组织结构.....	20
图 2-2. 选择外设例程文件 .....	22
图 2-3. 拷贝外设例程文件 .....	22
图 2-4. 打开工程文件 .....	23
图 2-5. 配置工程文件 .....	23
图 2-6. 编译调试下载 .....	24

# 表索引

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

---

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

---

表 3-83. 函数 <code>dma_channel_enable</code> .....	73
表 3-84. 函数 <code>dma_channel_disable</code> .....	74
表 3-85. 函数 <code>dma_periph_address_config</code> .....	74
表 3-86. 函数 <code>dma_memory_address_config</code> .....	75
表 3-87. 函数 <code>dma_transfer_number_config</code> .....	75
表 3-88. 函数 <code>dma_transfer_number_get</code> .....	76
表 3-89. 函数 <code>dma_priority_config</code> .....	77
表 3-90. 函数 <code>dma_memory_width_config</code> .....	77
表 3-91. 函数 <code>dma_periph_width_config</code> .....	78
表 3-92. 函数 <code>dma_memory_increase_enable</code> .....	79
表 3-93. 函数 <code>dma_memory_increase_disable</code> .....	79
表 3-94. 函数 <code>dma_periph_increase_enable</code> .....	80
表 3-95. 函数 <code>dma_periph_increase_disable</code> .....	80
表 3-96. 函数 <code>dma_transfer_direction_config</code> .....	81
表 3-97. 函数 <code>dma_flag_get</code> .....	81
表 3-98. 函数 <code>dma_flag_clear</code> .....	82
表 3-99. 函数 <code>dma_interrupt_flag_get</code> .....	83
表 3-100. 函数 <code>dma_interrupt_flag_clear</code> .....	83
表 3-101. 函数 <code>dma_interrupt_enable</code> .....	84
表 3-102. 函数 <code>dma_interrupt_disable</code> .....	85
表 3-103. EXTI 寄存器 .....	86
表 3-104. EXTI 库函数 .....	86
表 3-105. 枚举类型 <code>exti_line_enum</code> .....	86
表 3-106. 枚举类型 <code>exti_mode_enum</code> .....	87
表 3-107. 枚举类型 <code>exti_trig_type_enum</code> .....	87
表 3-108. 函数 <code>exti_deinit</code> .....	87
表 3-109. 函数 <code>exti_init</code> .....	88
表 3-110. 函数 <code>exti_interrupt_enable</code> .....	89
表 3-111. 函数 <code>exti_event_enable</code> .....	89
表 3-112. 函数 <code>exti_interrupt_disable</code> .....	90
表 3-113. 函数 <code>exti_event_disable</code> .....	90
表 3-114. 函数 <code>exti_flag_get</code> .....	91
表 3-115. 函数 <code>exti_flag_clear</code> .....	91
表 3-116. 函数 <code>exti_interrupt_flag_get</code> .....	92
表 3-117. 函数 <code>exti_interrupt_flag_clear</code> .....	92
表 3-118. 函数 <code>exti_software_interrupt_enable</code> .....	93
表 3-119. 函数 <code>exti_software_interrupt_disable</code> .....	93
表 3-120. FMC 寄存器 .....	94
表 3-121. FMC 固件库函数 .....	94
表 3-122. 枚举类型 <code>fmc_state_enum</code> .....	95
表 3-123. 函数 <code>fmc_unlock</code> .....	95
表 3-124. 函数 <code>fmc_lock</code> .....	96
表 3-125. 函数 <code>fmc_wscnt_set</code> .....	96
表 3-126. 函数 <code>fmc_prefetch_enable</code> .....	97

---

表 3-127. 函数 fmc_prefetch_disable .....	97
表 3-128. 函数 fmc_page_erase .....	98
表 3-129. 函数 fmc_mass_erase .....	98
表 3-130. 函数 fmc_doubleword_program .....	99
表 3-131. 函数 fmc_word_program .....	99
表 3-132. 函数 ob_unlock .....	100
表 3-133. 函数 ob_lock .....	100
表 3-134. 函数 ob_reset .....	101
表 3-135. 函数 option_byte_value_get .....	101
表 3-136. 函数 ob_erase .....	102
表 3-137. 函数 ob_write_protection_enable .....	102
表 3-138. 函数 ob_security_protection_config .....	103
表 3-139. 函数 ob_user_write .....	103
表 3-140. 函数 ob_data_program .....	104
表 3-141. 函数 ob_user_get .....	105
表 3-142. 函数 ob_data_get .....	105
表 3-143. 函数 ob_write_protection_get .....	106
表 3-144. 函数 ob_obstat_plevel_get .....	106
表 3-145. 函数 fmc_interrupt_enable .....	107
表 3-146. 函数 fmc_interrupt_disable .....	107
表 3-147. 函数 fmc_flag_get .....	108
表 3-148. 函数 fmc_flag_clear .....	108
表 3-149. 函数 fmc_interrupt_flag_get .....	109
表 3-150. 函数 fmc_interrupt_flag_clear .....	110
表 3-151. 函数 fmc_state_get .....	110
表 3-152. 函数 fmc_ready_wait .....	111
表 3-153. FWDGT 寄存器 .....	111
表 3-154. FWDGT 库函数 .....	112
表 3-155. 函数 fwdgt_write_enable .....	112
表 3-156. 函数 fwdgt_write_disable .....	113
表 3-157. 函数 fwdgt_enable .....	113
表 3-158. 函数 fwdgt_prescaler_value_config .....	114
表 3-159. 函数 fwdgt_reload_value_config .....	114
表 3-160. 函数 fwdgt_window_value_config .....	115
表 3-161. 函数 fwdgt_counter_reload .....	115
表 3-162. 函数 fwdgt_config .....	116
表 3-163. 函数 fwdgt_flag_get .....	116
表 3-164. GPIO 寄存器 .....	117
表 3-165. GPIO 库函数 .....	118
表 3-166. 函数 gpio_deinit .....	118
表 3-167. 函数 gpio_mode_set .....	119
表 3-168. 函数 gpio_output_options_set .....	120
表 3-169. 函数 gpio_bit_set .....	121
表 3-170. 函数 gpio_bit_reset .....	121

---

表 3-171. 函数 gpio_bit_write.....	122
表 3-172. 函数 gpio_port_write.....	123
表 3-173. 函数 gpio_input_bit_get.....	123
表 3-174. 函数 gpio_input_port_get .....	124
表 3-175. 函数 gpio_output_bit_get .....	124
表 3-176. 函数 gpio_output_port_get.....	125
表 3-177. 函数 gpio_af_set.....	125
表 3-178. 函数 gpio_pin_lock.....	126
表 3-179. 函数 gpio_bit_toggle .....	127
表 3-180. 函数 gpio_port_toggle .....	128
表 3-181. I2C 寄存器.....	128
表 3-182. I2C 库函数.....	129
表 3-183. 函数 i2c_deinit .....	130
表 3-184. 函数 i2c_clock_config .....	130
表 3-185. 函数 i2c_mode_addr_config.....	131
表 3-186. 函数 i2c_smbus_type_config .....	132
表 3-187. 函数 i2c_ack_config .....	132
表 3-188. 函数 i2c_ackpos_config.....	133
表 3-189. 函数 i2c_master_addressing .....	134
表 3-190. 函数 i2c_dualaddr_enable .....	134
表 3-191. 函数 i2c_dualaddr_disable .....	135
表 3-192. 函数 i2c_enable.....	135
表 3-193. 函数 i2c_disable.....	136
表 3-194. 函数 i2c_start_on_bus .....	136
表 3-195. 函数 i2c_stop_on_bus.....	137
表 3-196. 函数 i2c_data_transmit .....	137
表 3-197. 函数 i2c_data_receive .....	138
表 3-198. 函数 i2c_dma_enable .....	138
表 3-199. 函数 i2c_dma_last_transfer_config .....	139
表 3-200. 函数 i2c_stretch_scl_low_config .....	140
表 3-201. 函数 i2c_slave_response_to_gcall_config .....	140
表 3-202. 函数 i2c_software_reset_config.....	141
表 3-203. 函数 i2c_pec_enable .....	142
表 3-204. 函数 i2c_pec_transfer_enable .....	142
表 3-205. 函数 i2c_pec_value_get .....	143
表 3-206. 函数 i2c_smbus_issue_alert .....	143
表 3-207. 函数 i2c_smbus_arp_enable .....	144
表 3-208. 函数 i2c_sam_enable .....	145
表 3-209. 函数 i2c_sam_disable .....	145
表 3-210. 函数 i2c_sam_timeout_enable .....	146
表 3-211. 函数 i2c_sam_timeout_disable .....	146
表 3-212. 函数 i2c_flag_get .....	147
表 3-213. 函数 i2c_flag_clear .....	148
表 3-214. 函数 i2c_interrupt_enable.....	149

---

表 3-215. 函数 i2c_interrupt_disable.....	150
表 3-216. 函数 i2c_interrupt_flag_get .....	150
表 3-217. 函数 i2c_interrupt_flag_clear .....	152
表 3-218. NVIC 寄存器.....	153
表 3-219. Systick 寄存器.....	154
表 3-220. 枚举类型 IRQn_Type.....	154
表 3-221. MISC 库函数 .....	155
表 3-222. 函数 nvic_irq_enable.....	155
表 3-223. 函数 nvic_irq_disable.....	156
表 3-224. 函数 nvic_system_reset.....	156
表 3-225. 函数 nvic_vector_table_set .....	156
表 3-226. 函数 system_lowpower_set.....	157
表 3-227. 函数 system_lowpower_reset .....	158
表 3-228. 函数 systick_clksource_set.....	158
表 3-229. PMU 寄存器 .....	159
表 3-230. PMU 库函数 .....	159
表 3-231. 函数 pmu_deinit.....	160
表 3-232. 函数 pmu_lvd_select.....	160
表 3-233. 函数 pmu_ldo_output_select .....	161
表 3-234. 函数 pmu_lvd_disable.....	161
表 3-235. 函数 pmu_to_sleepmode .....	162
表 3-236. 函数 pmu_to_deepsleepmode.....	162
表 3-237. 函数 pmu_to_standbymode .....	163
表 3-238. 函数 pmu_wakeup_pin_enable .....	164
表 3-239. 函数 pmu_wakeup_pin_disable .....	164
表 3-240. 函数 pmu_backup_write_enable .....	165
表 3-241. 函数 pmu_backup_write_disable .....	166
表 3-242. 函数 pmu_flag_clear .....	166
表 3-243. 函数 pmu_flag_get.....	167
表 3-244. RCU 寄存器 .....	167
表 3-245. RCU 库函数 .....	168
表 3-246. 函数 rcu_deinit.....	169
表 3-247. 函数 rcu_periph_clock_enable.....	170
表 3-248. 函数 rcu_periph_clock_disable.....	170
表 3-249. 函数 rcu_periph_clock_sleep_enable .....	171
表 3-250. 函数 rcu_periph_clock_sleep_disable .....	172
表 3-251. 函数 rcu_periph_reset_enable .....	172
表 3-252. 函数 rcu_periph_reset_disable .....	173
表 3-253. 函数 rcu_bkp_reset_enable.....	174
表 3-254. 函数 rcu_bkp_reset_disable .....	174
表 3-255. 函数 rcu_system_clock_source_config .....	175
表 3-256. 函数 rcu_system_clock_source_get .....	175
表 3-257. 函数 rcu_ahb_clock_config .....	176
表 3-258. 函数 rcu_apb1_clock_config .....	176

---

表 3-259. 函数 rcu_apb2_clock_config .....	177
表 3-260. 函数 rcu_adc_clock_config .....	177
表 3-261. 函数 rcu_ckout_config .....	178
表 3-262. 函数 rcu_pll_config .....	179
表 3-263. 函数 rcu_usart_clock_config .....	180
表 3-264. 函数 rcu_rtc_clock_config .....	180
表 3-265. 函数 rcu_hxtal_prediv_config .....	181
表 3-266. 函数 rcu_lxtal_drive_capability_config .....	182
表 3-267. 函数 rcu_flag_get .....	182
表 3-268. 函数 rcu_all_reset_flag_clear .....	183
表 3-269. 函数 rcu_interrupt_flag_get .....	184
表 3-270. 函数 rcu_interrupt_flag_clear .....	185
表 3-271. 函数 rcu_interrupt_enable .....	185
表 3-272. 函数 rcu_interrupt_disable .....	186
表 3-273. 函数 rcu_osc1_stab_wait .....	187
表 3-274. 函数 rcu_osc1_on .....	188
表 3-275. 函数 rcu_osc1_off .....	188
表 3-276. 函数 rcu_osc1_bypass_mode_enable .....	189
表 3-277. 函数 rcu_osc1_bypass_mode_disable .....	189
表 3-278. 函数 rcu_hxtal_clock_monitor_enable .....	190
表 3-279. 函数 rcu_hxtal_clock_monitor_disable .....	190
表 3-280. 函数 rcu_irc8m_adjust_value_set .....	191
表 3-281. 函数 rcu_irc28m_adjust_value_set .....	191
表 3-282. 函数 rcu_voltage_key_unlock .....	192
表 3-283. 函数 rcu_deepsleep_voltage_set .....	192
表 3-284. 函数 rcu_clock_freq_get .....	193
表 3-285. RTC 寄存器 .....	194
表 3-286. RTC 库函数 .....	195
表 3-287. 结构体 rtc_parameter_struct .....	195
表 3-288. 结构体 rtc_alarm_struct .....	196
表 3-289. 结构体 rtc_timestamp_struct .....	196
表 3-290. 结构体 rtc_tamper_struct .....	196
表 3-291. 函数 rtc_deinit .....	197
表 3-292. 函数 rtc_init .....	197
表 3-293. 函数 rtc_init_mode_enter .....	198
表 3-294. 函数 rtc_init_mode_exit .....	198
表 3-295. 函数 rtc_register_sync_wait .....	199
表 3-296. 函数 rtc_current_time_get .....	199
表 3-297. 函数 rtc_subsecond_get .....	200
表 3-298. 函数 rtc_alarm_config .....	200
表 3-299. 函数 rtc_alarm_subsecond_config .....	201
表 3-300. 函数 rtc_alarm_enable .....	202
表 3-301. 函数 rtc_alarm_disable .....	203
表 3-302. 函数 rtc_alarm_get .....	203

---

表 3-303. 函数 <code>rtc_alarm_subsecond_get</code> .....	204
表 3-304. 函数 <code>rtc_timestamp_enable</code> .....	204
表 3-305. 函数 <code>rtc_timestamp_disable</code> .....	205
表 3-306. 函数 <code>rtc_timestamp_get</code> .....	205
表 3-307. 函数 <code>rtc_timestamp_subsecond_get</code> .....	206
表 3-308. 函数 <code>rtc_timestamp_enable</code> .....	206
表 3-309. 函数 <code>rtc_tamper_disable</code> .....	207
表 3-310. 函数 <code>rtc_interrupt_enable</code> .....	207
表 3-311. 函数 <code>rtc_interrupt_disable</code> .....	208
表 3-312. 函数 <code>rtc_flag_get</code> .....	208
表 3-313. 函数 <code>rtc_flag_clear</code> .....	209
表 3-314. 函数 <code>rtc_alter_output_config</code> .....	210
表 3-315. 函数 <code>rtc_calibration_config</code> .....	211
表 3-316. 函数 <code>rtc_hour_adjust</code> .....	212
表 3-317. 函数 <code>rtc_second_adjust</code> .....	212
表 3-318. 函数 <code>rtc_bypass_shadow_enable</code> .....	213
表 3-319. 函数 <code>rtc_bypass_shadow_disable</code> .....	213
表 3-320. 函数 <code>rtc_refclock_detection_enable</code> .....	214
表 3-321. 函数 <code>rtc_refclock_detection_disable</code> .....	214
表 3-322. SPI/I2S 寄存器 .....	215
表 3-323. SPI/I2S 库函数 .....	215
表 3-324. 结构体 <code>spi_parameter_struct</code> .....	216
表 3-325. 函数 <code>spi_i2s_deinit</code> .....	217
表 3-326. 函数 <code>spi_struct_para_init</code> .....	218
表 3-327. 函数 <code>spi_init</code> .....	218
表 3-328. 函数 <code>spi_enable</code> .....	219
表 3-329. 函数 <code>spi_disable</code> .....	219
表 3-330. 函数 <code>i2s_init</code> .....	220
表 3-331. 函数 <code>i2s_psc_config</code> .....	221
表 3-332. 函数 <code>i2s_enable</code> .....	222
表 3-333. 函数 <code>i2s_disable</code> .....	223
表 3-334. 函数 <code>spi_nss_output_enable</code> .....	223
表 3-335. 函数 <code>spi_nss_output_disable</code> .....	224
表 3-336. 函数 <code>spi_nss_internal_high</code> .....	224
表 3-337. 函数 <code>spi_nss_internal_low</code> .....	225
表 3-338. 函数 <code>spi_dma_enable</code> .....	225
表 3-339. 函数 <code>spi_dma_disable</code> .....	226
表 3-340. 函数 <code>spi_i2s_data_frame_format_config</code> .....	227
表 3-341. 函数 <code>spi_i2s_data_transmit</code> .....	227
表 3-342. 函数 <code>spi_i2s_data_receive</code> .....	228
表 3-343. 函数 <code>spi_bidirectional_transfer_config</code> .....	228
表 3-344. 函数 <code>spi_crc_polynomial_set</code> .....	229
表 3-345. 函数 <code>spi_crc_polynomial_get</code> .....	230
表 3-346. 函数 <code>spi_crc_on</code> .....	230

表 3-347. 函数 spi_crc_off .....	231
表 3-348. 函数 spi_crc_next .....	231
表 3-349. 函数 spi_crc_get .....	232
表 3-350. 函数 spi_ti_mode_enable .....	232
表 3-351. 函数 spi_ti_mode_disable .....	233
表 3-352. 函数 spi_nssp_mode_enable .....	233
表 3-353. 函数 spi_nssp_mode_disable .....	234
表 3-354. 函数 qspi_enable .....	234
表 3-355. 函数 qspi_disable .....	235
表 3-356. 函数 qspi_write_enable .....	235
表 3-357. 函数 qspi_read_enable .....	236
表 3-358. 函数 qspi_io23_output_enable .....	236
表 3-359. 函数 qspi_io23_output_disable .....	237
表 3-360. 函数 spi_i2s_interrupt_enable .....	237
表 3-361. 函数 spi_i2s_interrupt_disable .....	238
表 3-362. 函数 spi_i2s_interrupt_flag_get .....	239
表 3-363. 函数 spi_i2s_flag_get .....	240
表 3-364. 函数 spi_crc_error_clear .....	241
表 3-365. 函数 spi_fifo_access_size_config .....	242
表 3-366. 函数 spi_transmit_odd_config .....	242
表 3-367. 函数 spi_receive_odd_config .....	243
表 3-368. 函数 spi_crc_length_set .....	243
表 3-369. SYSCFG 寄存器 .....	244
表 3-370. SYSCFG 库函数 .....	245
表 3-371. 函数 syscfg_deinit .....	245
表 3-372. 函数 syscfg_dma_remap_enable .....	245
表 3-373. 函数 syscfg_dma_remap_disable .....	246
表 3-374. 函数 syscfg_high_current_enable .....	247
表 3-375. 函数 syscfg_high_current_disable .....	247
表 3-376. 函数 syscfg_exti_line_config .....	248
表 3-377. 函数 syscfg_lock_config .....	249
表 3-378. 函数 irq_latency_set .....	249
表 3-379. 函数 syscfg_flag_get .....	250
表 3-380. 函数 syscfg_flag_clear .....	250
表 3-381. TIMER 寄存器 .....	251
表 3-382. TIMER 库函数 .....	252
表 3-383. 结构体 timer_parameter_struct .....	254
表 3-384. 结构体 timer_break_parameter_struct .....	254
表 3-385. 结构体 timer_oc_parameter_struct .....	255
表 3-386. 结构体 timer_ic_parameter_struct .....	255
表 3-387. 函数 timer_deinit .....	256
表 3-388. 函数 timer_struct_para_init .....	256
表 3-389. 函数 timer_init .....	257
表 3-390. 函数 timer_enable .....	257

---

表 3-391. 函数 timer_disable.....	258
表 3-392. 函数 timer_auto_reload_shadow_enable.....	259
表 3-393. 函数 timer_auto_reload_shadow_disable.....	259
表 3-394. 函数 timer_update_event_enable .....	260
表 3-395. 函数 timer_update_event_disable .....	260
表 3-396. 函数 timer_counter_alignment.....	261
表 3-397. 函数 timer_counter_up_direction .....	261
表 3-398. 函数 timer_counter_down_direction .....	262
表 3-399. 函数 timer_prescaler_config .....	262
表 3-400. 函数 timer_repetition_value_config.....	263
表 3-401. 函数 timer_autoreload_value_config.....	264
表 3-402. 函数 timer_counter_value_config .....	264
表 3-403. 函数 timer_counter_read .....	265
表 3-404. 函数 timer_prescaler_read .....	266
表 3-405. 函数 timer_single_pulse_mode_config.....	266
表 3-406. 函数 timer_update_source_config.....	267
表 3-407. 函数 timer_ocpre_clear_source_config.....	267
表 3-408. 函数 timer_interrupt_enable .....	268
表 3-409. 函数 timer_interrupt_disable .....	269
表 3-410. 函数 timer_interrupt_flag_get .....	270
表 3-411. 函数 timer_interrupt_flag_clear.....	271
表 3-412. 函数 timer_flag_get .....	272
表 3-413. 函数 timer_flag_clear .....	273
表 3-414. 函数 timer_dma_enable .....	274
表 3-415. 函数 timer_dma_disable .....	274
表 3-416. 函数 timer_channel_dma_request_source_select.....	275
表 3-417. 函数 timer_dma_transfer_config .....	276
表 3-418. 函数 timer_event_software_generate.....	278
表 3-419. 函数 timer_break_struct_para_init .....	279
表 3-420. 函数 timer_break_config.....	279
表 3-421. 函数 timer_break_enable .....	280
表 3-422. 函数 timer_break_disable .....	281
表 3-423. 函数 timer_automatic_output_enable .....	281
表 3-424. 函数 timer_automatic_output_disable .....	282
表 3-425. 函数 timer_primary_output_config .....	282
表 3-426. 函数 timer_channel_control_shadow_config .....	283
表 3-427. 函数 timer_channel_control_shadow_update_config .....	283
表 3-428. 函数 timer_channel_output_struct_para_init .....	284
表 3-429. 函数 timer_channel_output_config .....	285
表 3-430. 函数 timer_channel_output_mode_config .....	286
表 3-431. 函数 timer_channel_output_pulse_value_config .....	287
表 3-432. 函数 timer_channel_output_shadow_config .....	287
表 3-433. 函数 timer_channel_output_fast_config .....	288
表 3-434. 函数 timer_channel_output_clear_config .....	289

---

表 3-435. 函数 timer_channel_output_polarity_config .....	290
表 3-436. 函数 timer_channel_complementary_output_polarity_config .....	291
表 3-437. 函数 timer_channel_output_state_config .....	292
表 3-438. 函数 timer_channel_complementary_output_state_config .....	292
表 3-439. 函数 timer_channel_input_struct_para_init .....	293
表 3-440. 函数 timer_input_capture_config .....	294
表 3-441. 函数 timer_channel_input_capture_prescaler_config .....	295
表 3-442. 函数 timer_channel_capture_value_register_read .....	296
表 3-443. 函数 timer_input_pwm_capture_config .....	296
表 3-444. 函数 timer_hall_mode_config .....	297
表 3-445. 函数 timer_input_trigger_source_select .....	298
表 3-446. 函数 timer_master_output_trigger_source_select .....	299
表 3-447. 函数 timer_slave_mode_select .....	300
表 3-448. 函数 timer_master_slave_mode_config .....	301
表 3-449. 函数 timer_external_trigger_config .....	301
表 3-450. 函数 timer_quadrature_decoder_mode_config .....	302
表 3-451. 函数 timer_internal_clock_config .....	303
表 3-452. 函数 timer_internal_trigger_as_external_clock_config .....	304
表 3-453. 函数 timer_external_trigger_as_external_clock_config .....	305
表 3-454. 函数 timer_external_clock_mode0_config .....	306
表 3-455. 函数 timer_external_clock_mode1_config .....	307
表 3-456. 函数 timer_external_clock_mode1_disable .....	308
表 3-457. 函数 timer_channel_remap_config .....	308
表 3-458. 函数 timer_write_chxval_register_config .....	309
表 3-459. 函数 timer_output_value_selection_config .....	310
表 3-460. USART 寄存器 .....	310
表 3-461. USART 库函数 .....	311
表 3-462. 枚举类型 usart_flag_enum .....	313
表 3-463. 枚举类型 usart_interrupt_flag_enum .....	314
表 3-464. 枚举类型 usart_interrupt_enum .....	314
表 3-465. 枚举类型 usart_invert_enum .....	315
表 3-466. 函数 usart_deinit .....	315
表 3-467. 函数 usart_baudrate_set .....	315
表 3-468. 函数 usart_parity_config .....	316
表 3-469. 函数 usart_word_length_set .....	317
表 3-470. 函数 usart_stop_bit_set .....	317
表 3-471. 函数 usart_enable .....	318
表 3-472. 函数 usart_disable .....	318
表 3-473. 函数 usart_transmit_config .....	319
表 3-474. 函数 usart_receive_config .....	320
表 3-475. 函数 usart_data_first_config .....	320
表 3-476. 函数 usart_invert_config .....	321
表 3-477. 函数 usart_overrun_enable .....	322
表 3-478. 函数 usart_overrun_disable .....	322

---

表 3-479. 函数 usart_oversample_config .....	323
表 3-480. 函数 usart_sample_bit_config .....	324
表 3-481. 函数 usart_receiver_timeout_enable .....	324
表 3-482. 函数 usart_receiver_timeout_disable .....	325
表 3-483. 函数 usart_receiver_timeout_threshold_config .....	325
表 3-484. 函数 usart_data_transmit .....	326
表 3-485. 函数 usart_data_receive .....	326
表 3-486. 函数 usart_autobaud_detection_enable .....	327
表 3-487. 函数 usart_autobaud_detection_disable .....	327
表 3-488. 函数 usart_autobaud_detection_mode_config .....	328
表 3-489. 函数 usart_address_config .....	329
表 3-490. 函数 usart_address_detection_mode_config .....	329
表 3-491. 函数 usart_mute_mode_enable .....	330
表 3-492. 函数 usart_mute_mode_disable .....	330
表 3-493. 函数 usart_mute_mode_wakeup_config .....	331
表 3-494. 函数 usart_lin_mode_enable .....	331
表 3-495. 函数 usart_lin_mode_disable .....	332
表 3-496. 函数 usart_lin_break_dection_length_config .....	332
表 3-497. 函数 usart_halfduplex_enable .....	333
表 3-498. 函数 usart_halfduplex_disable .....	334
表 3-499. 函数 usart_clock_enable .....	334
表 3-500. 函数 usart_clock_disable .....	335
表 3-501. 函数 usart_synchronous_clock_config .....	335
表 3-502. 函数 usart_guard_time_config .....	336
表 3-503. 函数 usart_smartcard_mode_enable .....	337
表 3-504. 函数 usart_smartcard_mode_disable .....	337
表 3-505. 函数 usart_smartcard_mode_nack_enable .....	338
表 3-506. 函数 usart_smartcard_mode_nack_disable .....	338
表 3-507. 函数 usart_smartcard_mode_early_nack_enable .....	339
表 3-508. 函数 usart_smartcard_mode_early_nack_disable .....	339
表 3-509. 函数 usart_smartcard_autoretry_config .....	340
表 3-510. 函数 usart_block_length_config .....	340
表 3-511. 函数 usart_irda_mode_enable .....	341
表 3-512. 函数 usart_irda_mode_disable .....	341
表 3-513. 函数 usart_prescaler_config .....	342
表 3-514. 函数 usart_irda_lowpower_config .....	342
表 3-515. 函数 usart_hardware_flow_rts_config .....	343
表 3-516. 函数 usart_hardware_flow_cts_config .....	344
表 3-517. 函数 usart_hardware_flow_coherence_config .....	344
表 3-518. 函数 usart_rs485_driver_enable .....	345
表 3-519. 函数 usart_rs485_driver_disable .....	346
表 3-520. 函数 usart_driver_assertime_config .....	346
表 3-521. 函数 usart_driver_deassertime_config .....	347
表 3-522. 函数 usart_depolarity_config .....	347

---

表 3-523. 函数 usart_dma_receive_config .....	348
表 3-524. 函数 usart_dma_transmit_config.....	348
表 3-525. 函数 usart_reception_error_dma_disable .....	349
表 3-526. 函数 usart_reception_error_dma_enable .....	350
表 3-527. 函数 usart_wakeup_enable .....	350
表 3-528. 函数 usart_wakeup_disable .....	351
表 3-529. 函数 usart_wakeup_mode_config .....	351
表 3-530. 函数 usart_receive_fifo_enable.....	352
表 3-531. 函数 usart_receive_fifo_disable.....	352
表 3-532. 函数 usart_receive_fifo_counter_number .....	353
表 3-533. 函数 usart_flag_get .....	353
表 3-534. 函数 usart_flag_clear .....	355
表 3-535. 函数 usart_interrupt_enable .....	356
表 3-536. 函数 usart_interrupt_disable.....	357
表 3-537. 函数 usart_command_enable.....	358
表 3-538. 函数 usart_interrupt_flag_get .....	359
表 3-539. 函数 usart_interrupt_flag_clear .....	360
表 3-540. WWDGT 寄存器.....	361
表 3-541. WWDGT 库函数.....	362
表 3-542. 函数 wwdgt_deinit .....	362
表 3-543. 函数 wwdgt_enable .....	362
表 3-544. 函数 wwdgt_counter_update.....	363
表 3-545. 函数 wwdgt_config .....	363
表 3-546. 函数 wwdgt_interrupt_enable .....	364
表 3-547. 函数 wwdgt_flag_get.....	365
表 3-548. 函数 wwdgt_flag_clear .....	365
表 4-1. 版本历史.....	367

## 1. 介绍

本手册介绍了32位基于ARM微控制器GD32E23x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GD32E23x所有外设的性能特征。该固件库还包括每一个外设的驱动描述和基于评估板的固件库使用例程。通过使用本固件库，用户无需深入掌握细节，也可以轻松应用每一个外设。使用本固件库可以大大减少用户的编程时间，从而降低开发成本。

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

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

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

此份固件库使用手册的整体架构如下：

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

### 1.1. 文档和固件库规则

#### 1.1.1. 外设缩写

表 1-1. 外设缩写

外设缩写	说明
ADC	模数转换器
CMP	比较器
CRC	循环冗余校验计算单元
DBG	调试模块
DMA	直接存储器访问控制器
EXTI	外部中断事件控制器
FMC	闪存控制器
FWDGT	独立看门狗
GPIO/AFIO	通用和备用输入/输出接口

外设缩写	说明
I2C	内部集成电路总线接口
MISC	嵌套中断向量列表控制器
PMU	电源管理单元
RCU	复位和时钟单元
RTC	实时时钟
SPI/I2S	串行外设接口/片上音频接口
SYSCFG	系统配置
TIMER	定时器
USART	通用同步异步收发器
WWDGT	窗口看门狗

### 1.1.2. 命名规则

固件库遵从以下命名规则：

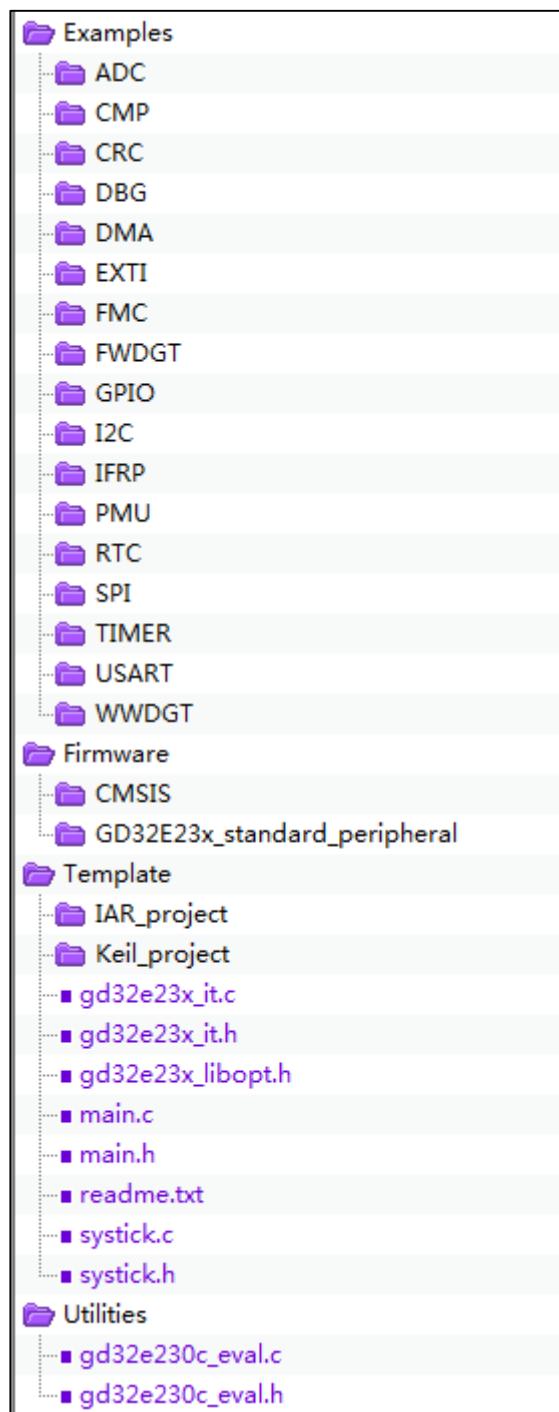
- XXX表示任一外设缩写，例如：ADC。更多缩写相关信息参阅[外设缩写](#)；[外设缩写](#)
- 源文件和头文件命名都以“gd32e23x\_”作为开头，例如：gd32e23x\_adc.h；
- 常量仅被应用于一个文件的，定义于该文件中；被应用于多个文件的，在对应头文件中定义。所有常量都由英文字母大写书写；
- 寄存器作为常量处理。他们的命名都由英文字母大写书写。在大多数情况下，寄存器缩写规范与本用户手册一致；
- 变量名采用全部小写，有多个单词组成的，在单词之间以下划线分隔；
- 外设函数的命名以该外设的缩写加下划线为开头，有多个单词组成的，在单词之间以下划线分隔，所有外设函数都由英文字母小写书写。

## 2. 固件库概述

### 2.1. 文件组织结构

GD32E23x\_Firmware\_Library, 文件组织结构见下图:

图 2-1. GD32E23x 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;
- **gd32e23x\_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);
- **gd32e23x\_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);
- **gd32e23x.it.h**: 该头文件包含了所有的中断处理程序的原形;
- **systick.c**: 该源文件包含了使用**systick**的精准延时程序;
- **systick.h**: 该头文件包含了使用**systick**的精准延时程序的原形;
- **main.c**: 例程代码注: 所有的例程的使用, 都不受不同软件开发环境的影响。

### 2.1.2. Firmware 文件夹

**Firmware**文件夹包含组成固件库核心的所有子文件夹和文件:

- CMSIS子文件夹包含有**Cortex M23**内核的支持文件、基于**Cortex M23**内核处理器的启动代码和库引导文件以及基于**GD32E23x**的全局头文件和系统配置文件;
- **GD32E23x\_standard\_peripheral**子文件夹;
  - **Include**子文件夹包含了固件函数库所需的头文件, 用户无需修改该文件夹;
  - **Source**子文件夹包含了固件函数库所需的源文件, 用户无需修改该文件夹;

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

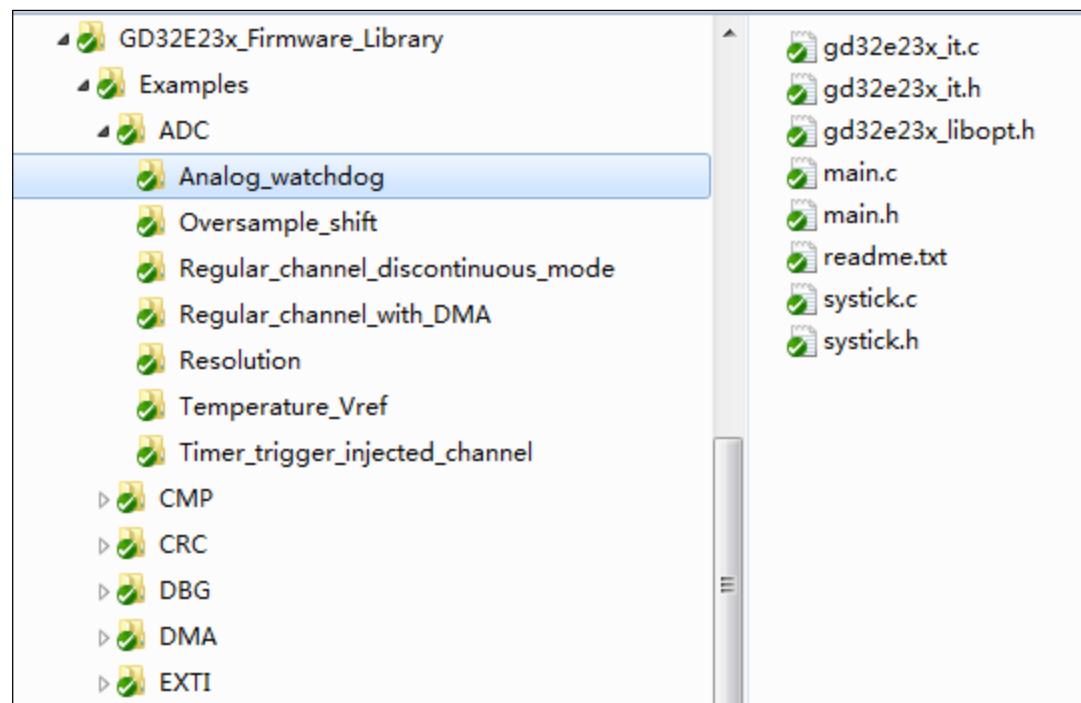
### 2.1.3. Template 文件夹

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

#### 选择文件

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

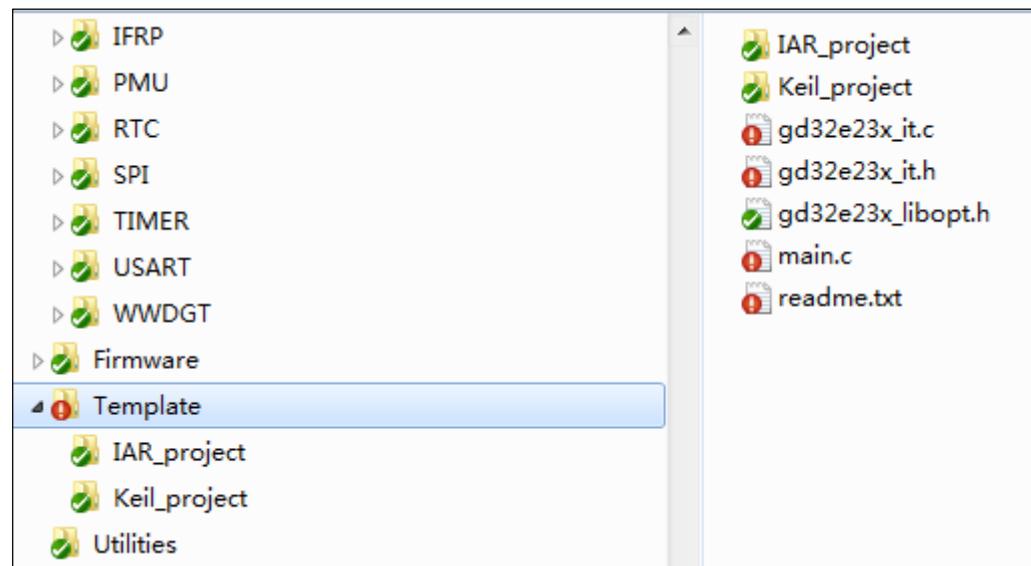
图 2-2. 选择外设例程文件



### 拷贝文件

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

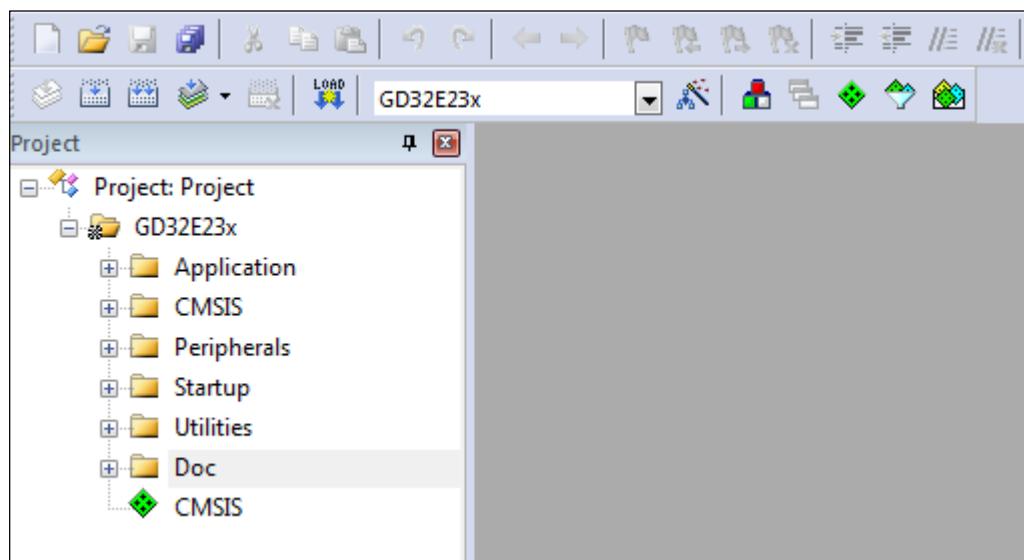
图 2-3. 拷贝外设例程文件



### 打开工程

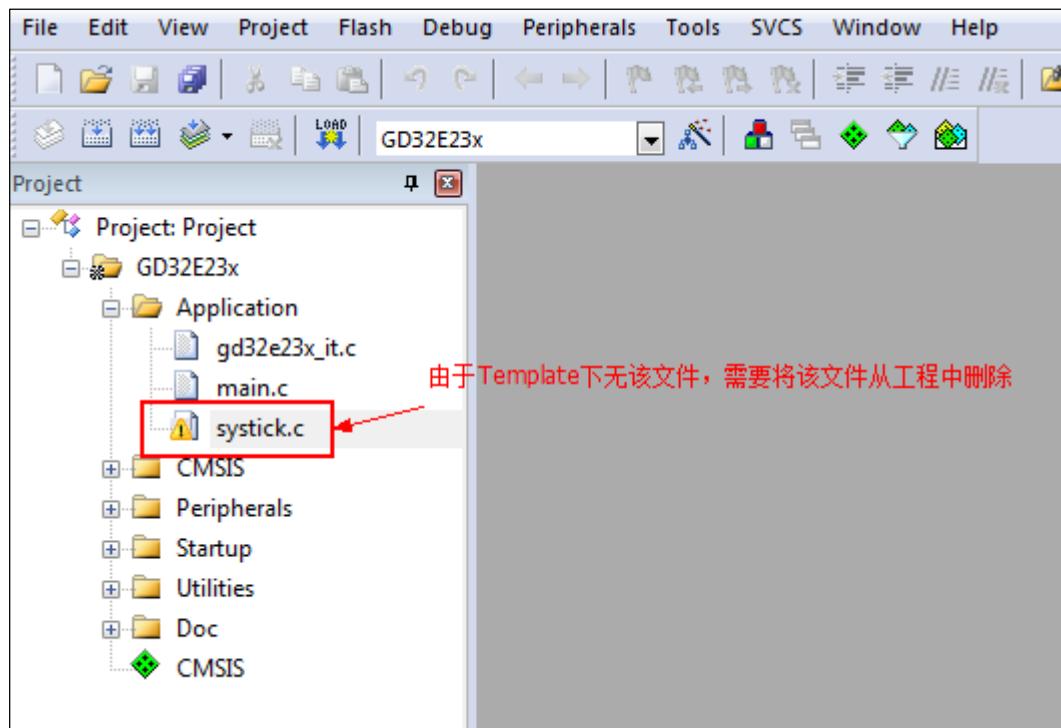
GD 提供 Keil 和 IAR 两种版本的工程，根据客户所安装的软件，打开不同的 project，如“Keil\_project”，打开\Template\Keil\_project\Project.uvprojx，如下图所示：

图 2-4. 打开工程文件



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

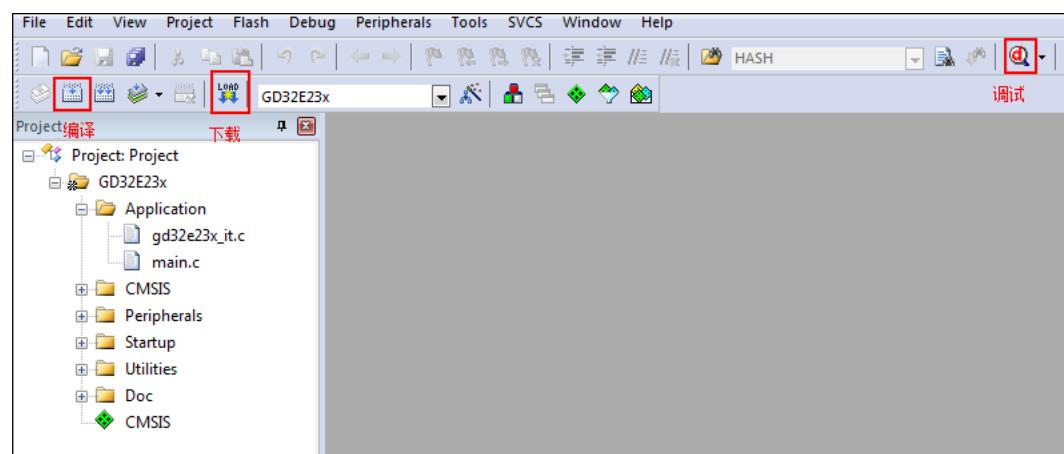
图 2-5. 配置工程文件



### 编译调试下载

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

图 2-6. 编译调试下载



## 2.1.4. Utilities 文件夹

Utilities文件夹包含运行固件库例程评估板的文件：

- gd32e230c\_eval.h文件是运行固件库例程所需关于评估板的头文件；
- gd32e230c\_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
gd32e23x_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。
gd32e23x_it.h	头文件，包含所有中断处理函数原形。
gd32e23x_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。
gd32e23x_xxx.h	外设PPP的头文件。包含外设PPP函数的定义，以及这些函数使用的变量。
gd32e23x_xxx.c	由C语言编写的外设PPP的驱动源程序文件。
systick.h	systick.c的头文件。包含systick配置函数的定义，以及外部用延时函数的定义。
systick.c	systick配置与延时函数源文件。
readme.txt	固件库例程使用及配置说明文档。

### 3. 外设固件库

#### 3.1. 外设固件库概述

外设固件库函数的描述格式如下表：

**表 3-1. 外设固件库函数描述格式**

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

#### 3.2. ADC

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

##### 3.2.1. 外设寄存器描述

ADC寄存器列表如下表所示：

**表 3-2. ADC 寄存器**

寄存器名称	寄存器描述
ADC_STAT	状态寄存器
ADC_CTL0	控制寄存器0
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_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_flag_get	获取ADC标志位
adc_flag_clear	清除ADC标志位
adc_interrupt_flag_get	获取ADC中断标志位
adc_interrupt_flag_clear	清除ADC中断标志位
adc_interrupt_enable	ADC中断使能
adc_interrupt_disable	ADC中断禁能
adc_watchdog_single_channel_enable	配置ADC模拟看门狗单通道有效
adc_watchdog_group_channel_enable	配置ADC模拟看门狗在通道组有效

库函数名称	库函数描述
adc_watchdog_disable	ADC模拟看门狗禁能
adc_watchdog_threshold_config	配置ADC模拟看门狗阈值
adc_resolution_config	配置ADC分辨率
adc_oversample_mode_config	配置ADC过采样模式
adc_oversample_mode_enable	使能ADC过采样
adc_oversample_mode_disable	禁能ADC过采样

### 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	-	
<b>输入参数{in}</b>		
-	-	
<b>输出参数{out}</b>		
-	-	
<b>返回值</b>		
-	-	

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

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

例如：

```
/* enable ADC DMA request */

adc_dma_mode_enable();
```

### 函数 **adc\_dma\_mode\_disable**

函数 **adc\_dma\_mode\_disable** 描述见下表：

**表 3-9. 函数 adc\_dma\_mode\_disable**

函数名称	adc_dma_mode_disable
函数原形	void adc_dma_mode_disable(void);
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\_discontinuous\_mode\_config**

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

**表 3-12. 函数 **adc\_discontinuous\_mode\_config****

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

例如:

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

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

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

**表 3-13. 函数 **adc\_special\_function\_config****

函数名称	adc_special_function_config
函数原形	void adc_special_function_config(uint32_t function, ControlStatus newvalue);
功能描述	使能或禁能ADC特殊功能
先决条件	-
被调用函数	-
<b>输入参数{in}</b>	
function	功能配置

<i>ADC_SCAN_MODE</i>	扫描模式选择
<i>ADC_INSERTED_C HANNEL_AUTO</i>	注入组自动转换
<i>ADC_CONTINUOU S_MODE</i>	连续模式选择
<b>输入参数{in}</b>	
<i>newvalue</i>	功能使能禁能
<i>ENABLE</i>	使能
<i>DISABLE</i>	禁能
<b>输出参数{out}</b>	
-	-
<b>返回值</b>	
-	-

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 3-14. 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	-
<b>输入参数{in}</b>	
<i>data_alignment</i>	数据对齐方式选择
<i>ADC_DATAALIGN_</i> <i>RIGHT</i>	LSB 对齐
<i>ADC_DATAALIGN_</i> <i>LEFT</i>	MSB 对齐
<b>输出参数{out}</b>	
-	-
<b>返回值</b>	
-	-

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 3-15. 函数 **adc\_channel\_length\_config****

函数名称	adc_channel_length_config	
函数原形	void adc_channel_length_config(uint8_t channel_group, uint32_t length);	
功能描述	配置规则通道组或注入通道组的长度	
先决条件	-	
被调用函数	-	
输入参数{in}		
<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>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 3-16. 函数 **adc\_regular\_channel\_config****

函数名称	adc_regular_channel_config	
函数原形	void adc_regular_channel_config(uint8_t rank, uint8_t channel, uint32_t sample_time);	
功能描述	配置ADC规则通道组	
先决条件	-	
被调用函数	-	
输入参数{in}		
<b>rank</b>	规则组通道序列, 取值范围为0~15	
输入参数{in}		
<b>channel</b>	ADC通道选择	
<i>ADC_CHANNEL_x</i>	ADC通道x (x=0..9,16,17)	
输入参数{in}		

<b>sample_time</b>	采样时间
<i>ADC_SAMPLETIME_1POINT5</i>	1.5 周期
<i>ADC_SAMPLETIME_7POINT5</i>	7.5 周期
<i>ADC_SAMPLETIME_13POINT5</i>	13.5 周期
<i>ADC_SAMPLETIME_28POINT5</i>	28.5 周期
<i>ADC_SAMPLETIME_41POINT5</i>	41.5 周期
<i>ADC_SAMPLETIME_55POINT5</i>	55.5 周期
<i>ADC_SAMPLETIME_71POINT5</i>	71.5 周期
<i>ADC_SAMPLETIME_239POINT5</i>	239.5 周期
<b>输出参数{out}</b>	
-	-
<b>返回值</b>	
-	-

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 3-17. 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	-
<b>输入参数{in}</b>	
<b>rank</b>	注入组通道序列，取值范围为0~3
<b>输入参数{in}</b>	
<b>channel</b>	ADC通道选择
<i>ADC_CHANNEL_x</i>	ADC 通道x (x=0..9,16,17)
<b>输入参数{in}</b>	

<b>sample_time</b>	采样时间
<i>ADC_SAMPLETIME_1POINT5</i>	1.5周期
<i>ADC_SAMPLETIME_7POINT5</i>	7.5周期
<i>ADC_SAMPLETIME_13POINT5</i>	13.5周期
<i>ADC_SAMPLETIME_28POINT5</i>	28.5周期
<i>ADC_SAMPLETIME_41POINT5</i>	41.5周期
<i>ADC_SAMPLETIME_55POINT5</i>	55.5周期
<i>ADC_SAMPLETIME_71POINT5</i>	71.5周期
<i>ADC_SAMPLETIME_239POINT5</i>	239.5周期
<b>输出参数{out}</b>	
-	-
<b>返回值</b>	
-	-

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 3-18. 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	-
<b>输入参数{in}</b>	
<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
<b>输入参数{in}</b>	
<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 3-19. 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 3-20. 函数 adc\_external\_trigger\_source\_config**

<b>函数名称</b>	adc_external_trigger_source_config
<b>函数原形</b>	void adc_external_trigger_source_config(uint8_t channel_group, uint32_t external_trigger_source);
<b>功能描述</b>	配置ADC外部触发源
<b>先决条件</b>	-
<b>被调用函数</b>	-
<b>输入参数{in}</b>	
<b>channel_group</b>	通道组选择
<b>ADC_REGULAR_CHANNEL</b>	规则通道组
<b>ADC_INSERTED_CHANNEL</b>	注入通道组
<b>输入参数{in}</b>	
<b>external_trigger_source</b>	规则通道组或注入通道组触发源
<b>ADC_EXTTRIG_REGULAR_T0_CH0</b>	TIMER0 CH0事件（规则组）
<b>ADC_EXTTRIG_REGULAR_T0_CH1</b>	TIMER0 CH1事件（规则组）
<b>ADC_EXTTRIG_REGULAR_T0_CH2</b>	TIMER0 CH2事件（规则组）
<b>ADC_EXTTRIG_REGULAR_T2_TRGO</b>	TIMER2 TRGO事件（规则组）
<b>ADC_EXTTRIG_REGULAR_T14_CH0</b>	TIMER14 CH0事件（规则组）
<b>ADC_EXTTRIG_REGULAR_EXTI_11</b>	外部中断线11（规则组）
<b>ADC_EXTTRIG_REGULAR_NONE</b>	软件触发（规则组）
<b>ADC_EXTTRIG_INSERTED_T0_TRGO</b>	TIMER0 TRGO事件（注入组）
<b>ADC_EXTTRIG_INSERTED_T0_CH3</b>	TIMER0 CH3事件（注入组）
<b>ADC_EXTTRIG_INSERTED_T2_CH3</b>	TIMER2 CH3事件（注入组）
<b>ADC_EXTTRIG_INSERTED_T14_TRGO</b>	TIMER14 TRGO事件（注入组）
<b>ADC_EXTTRIG_INSERTED_EXTI_15</b>	外部中断线15（注入组）
<b>ADC_EXTTRIG_INSERTED_NONE</b>	软件触发（注入组）

SERTED_NONE	
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 3-21. 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}	
channel_group	通道组选择
ADC_REGULAR_C HANNEL	规则通道组
ADC_INSERTED_C HANNEL	注入通道组
输出参数{out}	
-	-
返回值	
-	-

例如：

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

adc_software_trigger_enable(ADC_REGULAR_CHANNEL);
```

#### 函数 **adc\_regular\_data\_read**

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

**表 3-22. 函数 **adc\_regular\_data\_read****

函数名称	adc_regular_data_read
函数原形	uint16_t adc_regular_data_read(void);

功能描述	读ADC规则组数据寄存器
先决条件	-
被调用函数	-
<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\_data\_read** 描述见下表：

**表 3-23. 函数 **adc\_inserted\_data\_read****

函数名称	adc_inserted_data_read
函数原形	uint16_t adc_inserted_data_read(uint8_t inserted_channel);
功能描述	读ADC注入组数据寄存器
先决条件	-
被调用函数	-
<b>输入参数{in}</b>	
<b>inserted_channel</b>	注入通道选择
<b>ADC_INSERTED_C HANNEL_x</b>	注入通道x, x=0,1,2,3
<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 inserted group data register */

uint16_t adc_value = 0;

adc_value = adc_inserted_data_read ( ADC_INSERTED_CHANNEL_0);
```

### 函数 **adc\_flag\_get**

函数 **adc\_flag\_get** 描述见下表：

**表 3-24. 函数 adc\_flag\_get**

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

例如：

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

FlagStatus flag_value;

flag_value = adc_flag_get(ADC_FLAG_WDE);
```

### 函数 adc\_flag\_clear

函数 adc\_flag\_clear 描述见下表：

**表 3-25. 函数 adc\_flag\_clear**

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

返回值	
-	-

例如：

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

### 函数 **adc\_interrupt\_flag\_get**

函数 **adc\_interrupt\_flag\_get** 描述见下表：

**表 3-26. 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 3-27. 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	-
<b>输入参数{in}</b>	
<b>flag</b>	ADC中断标志位
<i>ADC_INT_FLAG_WDE</i>	模拟看门狗中断标志位
<i>ADC_INT_FLAG_EOC</i>	组转换结束中断标志位
<i>ADC_INT_FLAG_EOIC</i>	注入通道组转换结束中断标志位
<b>输出参数{out}</b>	
-	-
<b>返回值</b>	
-	-

例如：

```
/* clear the ADC analog watchdog interrupt bits*/
adc_interrupt_flag_clear(ADC_INT_FLAG_WDE);
```

#### 函数 **adc\_interrupt\_enable**

函数 **adc\_interrupt\_enable** 描述见下表：

**表 3-28. 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	-
<b>输入参数{in}</b>	
<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
<b>输出参数{out}</b>	
-	-
<b>返回值</b>	
-	-

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 3-29. 函数 adc\_interrupt\_disable**

函数名称	adc_interrupt_disable
函数原形	void adc_interrupt_enable(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\_watchdog\_single\_channel\_enable

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

**表 3-30. 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 Channelx(x=0..9,16,17)
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 3-31. 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	-
<b>输入参数{in}</b>	
<b>channel_group</b>	通道组使用模拟看门狗
<b>ADC_REGULAR_C HANNEL</b>	规则通道组
<b>ADC_INSERTED_C HANNEL</b>	注入通道组
<b>ADC_REGULAR_IN SERTED_CHANN E_L</b>	规则和注入通道组
<b>输出参数{out}</b>	
-	-
<b>返回值</b>	
-	-

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 3-32. 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	-
<b>输入参数{in}</b>	
-	-
<b>输出参数{out}</b>	
-	-

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 3-33. 函数 adc\_watchdog\_threshold\_config**

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

例如：

```
/* configure ADC analog watchdog threshold */
adc_watchdog_threshold_config(0x0400, 0xA000);
```

#### 函数 **adc\_resolution\_config**

函数 **adc\_resolution\_config** 描述见下表：

**表 3-34. 函数 adc\_resolution\_config**

函数名称	adc_resolution_config
函数原形	void adc_resolution_config(uint32_t resolution);
功能描述	配置ADC分辨率
先决条件	-
被调用函数	-
输入参数{in}	
resolution	ADC分辨率

<i>ADC_RESOLUTION_12B</i>	12位分辨率
<i>ADC_RESOLUTION_10B</i>	10位分辨率
<i>ADC_RESOLUTION_8B</i>	8位分辨率
<i>ADC_RESOLUTION_6B</i>	6位分辨率
输出参数{out}	
-	-
返回值	
-	-

Example:

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

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

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

**表 3-35. 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}	
<b>mode</b>	ADC过采样触发模式
<i>ADC_OVERSAMPLE_ALL_CONVERT</i>	在一个触发之后，对一个通道连续进行过采样转换
<i>ADC_OVERSAMPLE_ONE_CONVERT</i>	在一个触发之后，对一个通道只进行一次过采样转换
输入参数{in}	
<b>shift</b>	ADC过滤采样移位
<i>ADC_OVERSAMPLE_SHIFT_NONE</i>	不移位
<i>ADC_OVERSAMPLE_SHIFT_1B</i>	移1位

<i>ADC_OVERSA_MPLING_SHIFT_2B</i>	移2位
<i>ADC_OVERSA_MPLING_SHIFT_3B</i>	移3位
<i>ADC_OVERSA_MPLING_SHIFT_4B</i>	移4位
<i>ADC_OVERSA_MPLING_SHIFT_5B</i>	移5位
<i>ADC_OVERSA_MPLING_SHIFT_6B</i>	移6位
<i>ADC_OVERSA_MPLING_SHIFT_7B</i>	移7位
<i>ADC_OVERSA_MPLING_SHIFT_8B</i>	移8位
<b>输入参数{in}</b>	
<b>ratio</b>	ADC过采样率
<i>ADC_OVERSA_MPLING_RATIO_MUL2</i>	2x
<i>ADC_OVERSA_MPLING_RATIO_MUL4</i>	4x
<i>ADC_OVERSA_MPLING_RATIO_MUL8</i>	8x
<i>ADC_OVERSA_MPLING_RATIO_MUL16</i>	16x
<i>ADC_OVERSA_MPLING_RATIO_MUL32</i>	32x
<i>ADC_OVERSA_MPLING_RATIO_MUL64</i>	64x
<i>ADC_OVERSA_MPLING_RATIO</i>	128x

<i>_MUL128</i>	
<i>ADC_OVERSAMPLING_RATIO</i>	256x
<i>_MUL256</i>	
<b>输出参数{out}</b>	
-	-
<b>返回值</b>	
-	-

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 3-36. 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	-
<b>输入参数{in}</b>	
-	-
<b>输出参数{out}</b>	
-	-
<b>返回值</b>	
-	-

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 3-37. 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();
```

### 3.3. **CMP**

CMP通用比较器可独立工作，其输出端口可用于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，可将模拟信号作为触发源，结合定时器的PWM输出，可以实现电流控制。章节[3.3.1](#)描述了CMP的寄存器列表，章节[3.3.2](#)对CMP库函数进行说明

#### 3.3.1. 外设寄存器说明

CMP寄存器列表如下表所示：

**表 3-38. CMP 寄存器**

寄存器名称	寄存器描述
CMP_CS	控制状态寄存器

#### 3.3.2. 外设库函数说明

CMP库函数列表如下表所示：

**表 3-39. 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	除能CMP
cmp_switch_enable	使能CMP开关
cmp_switch_disable	除能CMP开关
cmp_output_level_get	获取CMP输出状态
cmp_lock_enable	锁定CMP

### 函数 cmp\_deinit

函数cmp\_deinit描述见下表：

**表 3-40. 函数 cmp\_deinit**

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

例如：

```
/* CMP deinitialize*/
cmp_deinit();
```

### 函数 cmp\_mode\_init

函数cmp\_mode\_init描述见下表：

**表 3-41. 函数 cmp\_mode\_init**

函数名称	cmp_mode_init
函数原型	void cmp_mode_init(operating_mode_enum operating_mode, inverting_input_enum inverting_input, cmp_hysteresis_enum 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}	
<b>operating_mode</b>	CMP模式，具体参考operating_mode_enum
<b>CMP_HIGHSPEED</b>	高速/全功耗
<b>CMP_Middlespeed</b>	中速/中功耗
<b>CMP_Lowspeed</b>	低速/低功耗
<b>CMP_Verylowspeed</b>	超低速/超低功耗
输入参数{in}	
<b>inverting_input</b>	CMP_IM输入选择，具体参考inverting_input_enum
<b>CMP_1_4VREFINT</b>	选择1/4VREFINT作为输入源

<b>CMP_1_2VREFINT</b>	选择1/2V <sub>REFINT</sub> 作为输入源
<b>CMP_3_4VREFINT</b>	选择3/4V <sub>REFINT</sub> 作为输入源
<b>CMP_VREFINT</b>	选择V <sub>REFINT</sub> 作为输入源
<b>CMP_PA4</b>	选择PA4作为输入源
<b>CMP_PA5</b>	选择PA5作为输入源
<b>CMP_PA0</b>	选择PA0作为输入源
<b>CMP_PA2</b>	选择PA2作为输入源
<b>输入参数{in}</b>	
<b>output_hysteresis</b>	CMP输出迟滞，具体参考cmp_hysteresis_enum
<b>CMP_HYSTERESIS_NO</b>	无迟滞
<b>CMP_HYSTERESIS_LOW</b>	低迟滞
<b>CMP_HYSTERESIS_MIDDLE</b>	中迟滞
<b>CMP_HYSTERESIS_HIGH</b>	高迟滞
<b>输出参数{out}</b>	
-	-
<b>返回值</b>	
-	-

例如：

```
/* CMP mode initialize*/
cmp_mode_init(CMP_HIGHSPEED,CMP_1_4VREFINT, CMP_HYSTERESIS_NO);
```

### 函数 cmp\_output\_init

函数cmp\_output\_init描述见下表：

**表 3-42. 函数 cmp\_output\_init**

函数名称	cmp_output_init
函数原型	void cmp_output_init(cmp_output_enum output_selection, uint32_t output_polarity);
功能描述	CMP输出初始化
先决条件	-
被调用函数	-
<b>输入参数{in}</b>	
<b>output_selection</b>	CMP输出选择，具体参考cmp_output_enum
<b>CMP_OUTPUT_NO_NE</b>	无选择
<b>CMP_OUTPUT_TIMER0_IN</b>	TIMER0中止输入

<i>CMP_OUTPUT_TI</i> <i>MEROIC0</i>	TIMER0 channel0输入捕获
<i>CMP_OUTPUT_TI</i> <i>MER0OCPRECLR</i>	TIMER0 OCPRE_CLR输入
<i>CMP_OUTPUT_TI</i> <i>MER2IC0</i>	TIMER2 channel0输入捕获
<i>CMP_OUTPUT_TI</i> <i>MER2OCPRECLR</i>	TIMER2 OCPRE_CLR输入
<b>输入参数{in}</b>	
<b>output_polarity</b>	输出极性
<i>CMP_OUTPUT_PO</i> <i>LARITY_INVERTED</i>	反相输出
<i>CMP_OUTPUT_PO</i> <i>LARITY_NOINVER</i> <i>TED</i>	正相输出
<b>输出参数{out}</b>	
-	-
<b>返回值</b>	
-	-

例如：

```
/* CMP output initialize*/
cmp_output_init(CMP_OUTPUT_TIMER0BKIN,
CMP_OUTPUT_POLARITY_NOINVERTED);
```

### 函数 cmp\_enable

函数cmp\_enable描述见下表：

**表 3-43. 函数 cmp\_enable**

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

例如：

```
/* enable CMP*/
```

```
cmp_enable();
```

### 函数 cmp\_disable

函数cmp\_disable描述见下表:

**表 3-44. 函数 cmp\_disable**

函数名称	cmp_disable
函数原型	void cmp_disable(void);
功能描述	除能CMP
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 CMP */
```

```
cmp_disable();
```

### 函数 cmp\_switch\_enable

函数cmp\_switch\_enable描述见下表:

**表 3-45. 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
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 CMP switch */
```

```
cmp_switch_enable();
```

### 函数 **cmp\_switch\_disable**

函数cmp\_switch\_disable描述见下表:

**表 3-46. 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	除能CMP开关
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 CMP switch */

cmp_switch_disable();
```

### 函数 **cmp\_output\_level\_get**

函数cmp\_output\_level\_get描述见下表:

**表 3-47. 函数 cmp\_output\_level\_get**

函数名称	cmp_output_level_get
函数原型	uint32_t cmp_output_level_get(void);
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}	
-	-
输出参数{out}	
-	-
返回值	
uint32_t	CMP_OUTPUTLEVEL_HIGH / CMP_OUTPUTLEVEL_LOW

例如:

```
/* get CMP output level */

cmp_output_level_get();
```

### 函数 **cmp\_lock\_enable**

函数**cmp\_lock\_enable**描述见下表:

**表 3-48. 函数 cmp\_lock\_enable**

函数名称	cmp_lock_enable
函数原型	void cmp_lock_enable(void);
功能描述	锁定CMP
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 CMP register */
cmp_lock_enable();
```

## 3.4. CRC

循环冗余校验码是一种用在数字网络和存储设备上的差错校验码，可以校验原始数据的偶然误差。章节[3.4.1](#)描述了CRC的寄存器列表，章节[3.4.2](#)对CRC库函数进行说明。

### 3.4.1. 外设寄存器说明

CRC寄存器列表如下表所示:

**表 3-49. CRC 寄存器**

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

### 3.4.2. 外设库函数说明

CRC库函数列表如下表所示：

**表 3-50. 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 3-51. 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 3-52. 函数 crc\_reverse\_output\_data\_enable**

函数名称	crc_reverse_output_data_enable
函数原形	void crc_reverse_output_data_enable (void);
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 3-53. 函数 crc\_reverse\_output\_data\_disable**

函数名称	crc_reverse_output_data_disable
函数原形	void crc_reverse_output_data_disable (void);
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 3-54. 函数 crc\_data\_register\_reset**

函数名称	crc_data_register_reset
函数原形	void crc_data_register_reset(void);
功能描述	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 3-55. 函数 crc\_data\_register\_read**

函数名称	crc_data_register_read
函数原形	uint32_t crc_data_register_read(void);
功能描述	读数据寄存器
先决条件	-
被调用函数	-
输入参数{in}	
-	-
输出参数{out}	
-	-
返回值	
uint32_t	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 3-56. 函数 crc\_free\_data\_register\_read**

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

例如：

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

uint8_t crc_value = 0;

crc_value = crc_free_data_register_read();
```

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

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

**表 3-57. 函数 crc\_free\_data\_register\_write**

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

例如：

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

crc_free_data_register_write(0x11);
```

### 函数 crc\_init\_data\_register\_write

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

**表 3-58. 函数 crc\_init\_data\_register\_write**

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

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 3-59. 函数 crc\_input\_data\_reverse\_config**

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

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 3-60. 函数 crc\_polynomial\_size\_set**

函数名称	crc_polynomial_size_set	
函数原形	void crc_polynomial_size_set(uint32_t poly_size)	
功能描述	配置多项式长度	
先决条件	-	
被调用函数	-	
输入参数{in}		
<b>poly_size</b>	多项式的长度	
CRC_CTL_PS_32	32位多项式值用于CRC计算	
CRC_CTL_PS_16	16位多项式值用于CRC计算	
CRC_CTL_PS_8	8位多项式值用于CRC计算	
CRC_CTL_PS_7	7位多项式值用于CRC计算	
输出参数{out}		
-	-	
返回值		
-	-	

例如:

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

### 函数 **crc\_polynomial\_set**

函数crc\_polynomial\_set描述见下表:

**表 3-61. 函数 crc\_polynomial\_set**

函数名称	crc_polynomial_set	
函数原形	void crc_polynomial_set(uint32_t poly)	
功能描述	设置多项式寄存器值	
先决条件	-	
被调用函数	-	
输入参数{in}		
<b>poly</b>	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 3-62. 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}	
<b>sdata</b>	设定的32位数据
输出参数{out}	
-	-
返回值	
<b>uint32_t</b>	32位CRC计算结果 (0-0xFFFFFFFF)

例如：

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

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

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

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

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

函数crc\_block\_data\_calculate描述见下表：

**表 3-63. 函数 crc\_block\_data\_calculate**

函数名称	crc_block_data_calculate
函数原形	uint32_t crc_block_data_calculate(uint32_t array[], uint32_t size);
功能描述	CRC计算一个32位数组
先决条件	-
被调用函数	-
输入参数{in}	
<b>array</b>	32位数据数组的指针
输入参数{in}	

<b>size</b>	数据长度
<b>输出参数{out}</b>	
-	-
<b>返回值</b>	
<b>uint32_t</b>	32位CRC计算结果 (0-0xFFFFFFFF)

例如：

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

#define BUFFER_SIZE      6

uint32_t valcrc = 0;

static const uint32_t data_buffer[BUFFER_SIZE] = {

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

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

## 3.5. **DBG**

调试系统帮助调试者在低功耗模式下调试或者进行一些外设调试。章节[3.5.1](#)描述了DBG的寄存器列表，章节[3.5.2](#)对DBG库函数进行说明。

### 3.5.1. 外设寄存器说明

DBG寄存器列表如下表所示：

**表 3-64. DBG 寄存器**

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

### 3.5.2. 外设库函数说明

DBG库函数列表如下表所示：

**表 3-65. 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 3-66. 枚举类型 `dbg_periph_enum`**

成员名称	功能描述
DBG_FWDGT_HOLD	当内核停止时，保持FWDGT计数器时钟
DBG_WWDGT_HOLD	当内核停止时，保持WWDGT计数器时钟
DBG_TIMER0_HOLD	当内核停止时，保持TIMER0计数器计数值不变
DBG_TIMER2_HOLD	当内核停止时，保持TIMER2计数器计数值不变
DBG_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 3-67. 函数 `dbg_deinit`**

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

例如：

```
/* reset DBG register */
dbg_deinit();
```

### 函数 `dbg_id_get`

函数`dbg_id_get`描述见下表：

**表 3-68. 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 3-69. 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_R_SLEEP	在睡眠模式下，保持调试器连接，可进行调试
DBG_LOW_POWER_R_DEEPSLEEP	在深度睡眠模式下，保持调试器连接，可进行调试
DBG_LOW_POWER_R_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 3-70. 函数 **dbg\_low\_power\_disable****

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

例如:

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

```
dbg_low_power_disable(DBG_LOW_POWER_SLEEP);
```

### 函数 **dbg\_periph\_enable**

函数**dbg\_periph\_enable**描述见下表:

**表 3-71. 函数 **dbg\_periph\_enable****

函数名称	dbg_periph_enable
函数原形	void dbg_periph_enable(dbg_periph_enum dbg_periph);
功能描述	使能外设的MCU调试保持功能
先决条件	-
被调用函数	-
<b>输入参数{in}</b>	
<b>dbg_periph</b>	参考枚举变量 <a href="#">表3-66. 枚举类型<b>dbg_periph_enum</b></a>

<i>DBG_FWDGT_HOLD</i>	当内核停止时，保持FWDGT计数器时钟
<i>DBG_WWDGT_HOLD</i>	当内核停止时，保持WWDGT计数器时钟
<i>DBG_TIMERx_HOLD</i>	当内核停止时，保持TIMERx计数器计数值不变 (x=0,2,5,13,14,15,16)
<i>DBG_I2Cx_HOLD</i>	当内核停止时，保持I2Cx (x=0,1) 的SMBUS状态不变，用于调试
<i>DBG_RTC_HOLD</i>	当内核停止时，保持RTC计数器，用于调试
<b>输出参数{out}</b>	
-	-
<b>返回值</b>	
-	-

例如：

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

dbg_periph_enable(DBG_TIMER0_HOLD);
```

### 函数 **dbg\_periph\_disable**

函数**dbg\_periph\_disable**描述见下表：

**表 3-72. 函数 **dbg\_periph\_disable****

函数名称	<b>dbg_periph_disable</b>
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	-
<b>输入参数{in}</b>	
<b>dbg_periph</b>	参考枚举变量 <a href="#">表3-66. 枚举类型<b>dbg_periph_enum</b></a>
<i>DBG_FWDGT_HOLD</i>	当内核停止时，保持FWDGT计数器时钟
<i>DBG_WWDGT_HOLD</i>	当内核停止时，保持WWDGT计数器时钟
<i>DBG_TIMERx_HOLD</i>	当内核停止时，保持TIMERx计数器计数值不变 (x=0,2,5,13,14,15,16)
<i>DBG_I2Cx_HOLD</i>	当内核停止时，保持I2Cx (x=0,1) 的SMBUS状态不变，用于调试
<i>DBG_RTC_HOLD</i>	当内核停止时，保持RTC计数器，用于调试
<b>输出参数{out}</b>	
-	-
<b>返回值</b>	
-	-

例如：

---

```

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

dbg_periph_disable(DBG_TIMER0_HOLD);

```

## 3.6. DMA

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

### 3.6.1. 外设寄存器说明

DMA寄存器列表如下表所示：

**表 3-73. 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..4)	通道x控制寄存器
DMA_CHxCNT (x=0..4)	通道x计数寄存器
DMA_CHxPADDR (x=0..4)	通道x外设地址寄存器
DMA_CHxMADDR (x=0..4)	通道x存储器地址寄存器

### 3.6.2. 外设库函数说明

DMA库函数列表如下表所示：

**表 3-74. 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传输的外设地址配置
dma_memory_address_config	DMA通道x传输的存储器地址配置

库函数名称	库函数描述
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_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
dma_interrupt_enable	DMA通道x中断使能
dma_interrupt_disable	DMA通道x中断禁能

### 结构体 **dma\_parameter\_struct**

**表 3-75. 结构体 `dma_parameter_struct`**

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

### 函数 **dma\_deinit**

函数 `dma_deinit` 描述见下表:

**表 3-76. 函数 `dma_deinit`**

函数名称	<code>dma_deinit</code>
函数原型	<code>void dma_deinit(dma_channel_enum channelx);</code>
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
<b>输入参数{in}</b>	
<code>channelx</code>	DMA通道

DMA_CHx( x=0..4)	DMA通道选择
输出参数{out}	
-	-
返回值	
-	-

例如：

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

### 函数 **dma\_struct\_para\_init**

函数 **dma\_struct\_para\_init** 描述见下表：

**表 3-77. 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	一个已经定义的dma_parameter_struct结构体变量地址
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 3-78. 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}	

<b>channelx</b>	DMA通道
<b>DMA_CHx( x=0..4)</b>	DMA通道选择
<b>输入参数{in}</b>	
<b>init_struct</b>	初始化结构体, 结构体成员参考 <a href="#">表3-75. 结构体dma_parameter_struct</a>
<b>输出参数{out}</b>	
-	-
<b>返回值</b>	
-	-

例如:

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

dma_struct_para_init(&dma_init_struct);
dma_init_struct.direction = DMA_PERIPHERAL_TO_MEMORY;
dma_init_struct.memory_addr = (uint32_t)g_destbuf;
dma_init_struct.memory_inc = DMA_MEMORY_INCREASE_ENABLE;
dma_init_struct.memory_width = DMA_MEMORY_WIDTH_8BIT;
dma_init_struct.number = TRANSFER_NUM;
dma_init_struct.periph_addr = (uint32_t)BANK0_WRITE_START_ADDR;
dma_init_struct.periph_inc = DMA_PERIPH_INCREASE_ENABLE;
dma_init_struct.periph_width = DMA_PERIPHERAL_WIDTH_8BIT;
dma_init_struct.priority = DMA_PRIORITY_ULTRA_HIGH;
dma_init(DMA_CH0, &dma_init_struct);
```

### 函数 **dma\_circulation\_enable**

函数 **dma\_circulation\_enable** 描述见下表:

**表 3-79. 函数 **dma\_circulation\_enable****

函数名称	<b>dma_circulation_enable</b>
函数原型	void dma_circulation_enable(dma_channel_enum channelx);
功能描述	DMA循环模式使能
先决条件	相应通道使能位CHEN需为0
被调用函数	无
<b>输入参数{in}</b>	
<b>channelx</b>	DMA通道
<b>DMA_CHx(x=0..4)</b>	DMA通道选择
<b>输出参数{out}</b>	
-	-
<b>返回值</b>	
-	-

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 3-80. 函数 **dma\_circulation\_disable****

函数名称	dma_circulation_disable
函数原型	void dma_circulation_disable(dma_channel_enum channelx);
功能描述	DMA循环模式禁能
先决条件	相应通道使能位CHEN需为0
被调用函数	无
输入参数{in}	
channelx	DMA通道
DMA_CHx(x=0..4)	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 3-81. 函数 **dma\_memory\_to\_memory\_enable****

函数名称	dma_memory_to_memory_enable
函数原型	void dma_memory_to_memory_enable(dma_channel_enum channelx);
功能描述	存储器到存储器DMA传输使能
先决条件	相应通道使能位CHEN需为0
被调用函数	无
输入参数{in}	
channelx	DMA通道
DMA_CHx(x=0..4)	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 3-82. 函数 **dma\_memory\_to\_memory\_disable****

函数名称	dma_memory_to_memory_disable
函数原形	void dma_memory_to_memory_disable(dma_channel_enum channelx);
功能描述	存储器到存储器DMA传输禁能
先决条件	相应通道使能位CHEN需为0
被调用函数	无
输入参数{in}	
channelx	DMA通道
DMA_CHx(x=0..4)	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 3-83. 函数 **dma\_channel\_enable****

函数名称	dma_channel_enable
函数原型	void dma_channel_enable(dma_channel_enum channelx);
功能描述	DMA通道x传输使能
先决条件	无
被调用函数	无
输入参数{in}	
channelx	DMA通道
DMA_CHx(x=0..4)	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 3-84. 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
<b>输入参数{in}</b>	
channelx	DMA通道
DMA_CHx(x=0..4)	DMA通道选择
<b>输出参数{out}</b>	
-	-
<b>返回值</b>	
-	-

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 3-85. 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传输的外设基址配置
先决条件	相应通道使能位CHEN需为0
被调用函数	无
<b>输入参数{in}</b>	
channelx	DMA通道
DMA_CHx(x=0..4)	DMA通道选择
<b>输入参数{in}</b>	
address	外设基地址
<b>输出参数{out}</b>	
-	-
<b>返回值</b>	
-	-

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 3-86. 函数 **dma\_memory\_address\_config****

函数名称	dma_memory_address_config
函数原型	void dma_memory_address_config(dma_channel_enum channelx, uint32_t address);
功能描述	DMA通道x传输的存储器基地址配置
先决条件	相应通道使能位CHEN需为0
被调用函数	无
输入参数{in}	
channelx	DMA通道
DMA_CHx(x=0..4)	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 3-87. 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	相应通道使能位CHEN需为0
被调用函数	无
输入参数{in}	

<b>channelx</b>	DMA通道
<b>DMA_CHx(x=0..4)</b>	DMA通道选择
<b>输入参数{in}</b>	
<b>number</b>	数据传输数量 (0x0 – 0xFFFF)
<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 3-88. 函数 **dma\_transfer\_number\_get****

函数名称	dma_transfer_number_get
函数原型	uint32_t dma_transfer_number_get(dma_channel_enum channelx);
功能描述	获取DMA通道x还有多少数据要传输
先决条件	相应通道使能位CHEN需为0
被调用函数	无
<b>输入参数{in}</b>	
<b>channelx</b>	DMA通道
<b>DMA_CHx(x=0..4)</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 (0x0 – 0xFFFF)

例如：

```
/* get DMA channel0 transfer number */

uint32_t number = 0;

number = dma_transfer_number_get(DMA_CH0);
```

### 函数 **dma\_priority\_config**

函数 **dma\_priority\_config** 描述见下表：

**表 3-89. 函数 `dma_priority_config`**

函数名称	<code>dma_priority_config</code>
函数原型	<code>void dma_priority_config(dma_channel_enum channelx, uint32_t priority);</code>
功能描述	DMA通道x的传输软件优先级配置
先决条件	相应通道使能位CHEN需为0
被调用函数	无
<b>输入参数{in}</b>	
<code>channelx</code>	DMA通道
<code>DMA_CHx(x=0..4)</code>	DMA通道选择
<b>输入参数{in}</b>	
<code>priority</code>	DMA通道软件优先级
<code>DMA_PRIORITY_LOW</code>	低优先级
<code>DMA_PRIORITY_MEDIUM</code>	中优先级
<code>DMA_PRIORITY_HIGH</code>	高优先级
<code>DMA_PRIORITY_ULTRA_HIGH</code>	极高优先级
<b>输出参数{out}</b>	
-	-
<b>返回值</b>	
-	-

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 3-90. 函数 `dma_memory_width_config`**

函数名称	<code>dma_memory_width_config</code>
函数原型	<code>void dma_memory_width_config(dma_channel_enum channelx, uint32_t mwidth);</code>
功能描述	DMA通道x传输的存储器数据宽度配置
先决条件	相应通道使能位CHEN需为0
被调用函数	无
<b>输入参数{in}</b>	
<code>channelx</code>	DMA通道
<code>DMA_CHx(x=0..4)</code>	DMA通道选择
<b>输入参数{in}</b>	

<b>mwidth</b>	存储器数据传输宽度
<i>DMA_MEMORY_WIDTH_8BIT</i>	8位数据传输宽度
<i>DMA_MEMORY_WIDTH_16BIT</i>	16位数据传输宽度
<i>DMA_MEMORY_WIDTH_32BIT</i>	32位数据传输宽度
<b>输出参数{out}</b>	
-	-
<b>返回值</b>	
-	-

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 3-91. 函数 **dma\_periph\_width\_config****

<b>函数名称</b>	dma_periph_width_config
<b>函数原型</b>	void dma_periph_width_config(dma_channel_enum channelx, uint32_t pwidth);
<b>功能描述</b>	DMA通道x传输的外设数据宽度配置
<b>先决条件</b>	相应通道使能位CHEN需为0
<b>被调用函数</b>	无
<b>输入参数{in}</b>	
<b>pwidth</b>	外设数据传输宽度
<i>DMA_PERIPHERAL_WIDTH_8BIT</i>	8位数据传输宽度
<i>DMA_PERIPHERAL_WIDTH_16BIT</i>	16位数据传输宽度
<i>DMA_PERIPHERAL_WIDTH_32BIT</i>	32位数据传输宽度
<b>输出参数{out}</b>	
-	-
<b>返回值</b>	
-	-

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 3-92. 函数 **dma\_memory\_increase\_enable****

函数名称	dma_memory_increase_enable
函数原型	void dma_memory_increase_enable(dma_channel_enum channelx);
功能描述	DMA通道x传输的存储器地址生成算法增量模式使能
先决条件	相应通道使能位CHEN需为0
被调用函数	无
<b>输入参数{in}</b>	
channelx	DMA通道
DMA_CHx(x=0..4)	DMA通道选择
<b>输出参数{out}</b>	
-	-
<b>返回值</b>	
-	-

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 3-93. 函数 **dma\_memory\_increase\_disable****

函数名称	dma_memory_increase_disable
函数原型	void dma_memory_increase_disable(dma_channel_enum channelx);
功能描述	DMA通道x传输的存储器地址生成算法增量模式禁能
先决条件	相应通道使能位CHEN需为0
被调用函数	无
<b>输入参数{in}</b>	
channelx	DMA通道
DMA_CHx(x=0..4)	DMA通道选择
<b>输出参数{out}</b>	
-	-
<b>返回值</b>	
-	-

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 3-94. 函数 **dma\_periph\_increase\_enable****

函数名称	dma_periph_increase_enable
函数原型	void dma_periph_increase_enable(dma_channel_enum channelx);
功能描述	DMA通道x传输的外设地址生成算法增量模式使能
先决条件	相应通道使能位CHEN需为0
被调用函数	无
<b>输入参数{in}</b>	
channelx	DMA通道
DMA_CHx(x=0..4)	DMA通道选择
<b>输出参数{out}</b>	
-	-
<b>返回值</b>	
-	-

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 3-95. 函数 **dma\_periph\_increase\_disable****

函数名称	dma_periph_increase_disable
函数原型	void dma_periph_increase_disable(dma_channel_enum channelx);
功能描述	DMA通道x传输的外设地址生成算法增量模式禁能
先决条件	相应通道使能位CHEN需为0
被调用函数	无
<b>输入参数{in}</b>	
channelx	DMA通道
DMA_CHx(x=0..4)	DMA通道选择
<b>输出参数{out}</b>	
-	-
<b>返回值</b>	
-	-

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 3-96. 函数 **dma\_transfer\_direction\_config****

函数名称	<b>dma_transfer_direction_config</b>
函数原型	<code>void dma_transfer_direction_config(dma_channel_enum channelx, uint32_t direction);</code>
功能描述	DMA通道x的传输方向配置
先决条件	相应通道使能位CHEN需为0
被调用函数	无
<b>输入参数{in}</b>	
<b>channelx</b>	DMA通道
<b>DMA_CHx(x=0..4)</b>	DMA通道选择
<b>输入参数{in}</b>	
<b>direction</b>	数据传输方向
<b>DMA_PERIPHERAL_TO_MEMORY</b>	读取外设中数据，写入存储器
<b>DMA_MEMORY_TO_PERIPHERAL</b>	读取存储器中数据，写入外设
<b>输出参数{out}</b>	
-	-
<b>返回值</b>	
-	-

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 3-97. 函数 **dma\_flag\_get****

函数名称	<b>dma_flag_get</b>
函数原型	<code>FlagStatus dma_flag_get(dma_channel_enum channelx, uint32_t flag);</code>
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
<b>输入参数{in}</b>	
<b>channelx</b>	DMA通道
<b>DMA_CHx(x=0..4)</b>	DMA通道选择
<b>输入参数{in}</b>	
<b>flag</b>	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
<b>输出参数{out}</b>	
-	-
<b>返回值</b>	
<b>FlagStatus</b>	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 3-98. 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
<b>输入参数{in}</b>	
<b>channelx</b>	DMA通道
<b>DMA_CHx(x=0..4)</b>	DMA通道选择
<b>输入参数{in}</b>	
<b>flag</b>	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
<b>输出参数{out}</b>	
-	-
<b>返回值</b>	
-	-

例如：

```
/* clear DMA channel0 flag

dma_flag_clear(DMA_CH0, DMA_FLAG_FTF);
```

### 函数 **dma\_interrupt\_flag\_get**

函数 **dma\_interrupt\_flag\_get** 描述见下表:

**表 3-99. 函数 **dma\_interrupt\_flag\_get****

函数名称	dma_interrupt_flag_get
函数原型	FlagStatus dma_interrupt_flag_get(dma_channel_enum channelx, uint32_t flag);
功能描述	获取DMA通道x中断标志位状态
先决条件	无
被调用函数	无
输入参数{in}	
channelx	DMA通道
DMA_CHx(x=0..4)	DMA通道选择
输入参数{in}	
flag	DMA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_ERR	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 3-100. 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}	

<b>channelx</b>	DMA通道
<i>DMA_CHx( x=0..4)</i>	DMA通道选择
<b>输入参数{in}</b>	
<b>flag</b>	DMA标志
<i>DMA_INT_FLAG_G</i>	DMA通道全局中断标志
<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
<b>输出参数{out}</b>	
-	-
<b>返回值</b>	
-	-

例如：

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

### 函数 **dma\_interrupt\_enable**

函数 **dma\_interrupt\_enable** 描述见下表：

**表 3-101. 函数 **dma\_interrupt\_enable****

<b>函数名称</b>	<b>dma_interrupt_enable</b>
<b>函数原型</b>	void dma_interrupt_enable(dma_channel_enum channelx, uint32_t source);
<b>功能描述</b>	DMA通道x中断使能
<b>先决条件</b>	无
<b>被调用函数</b>	无
<b>输入参数{in}</b>	
<b>channelx</b>	DMA通道
<i>DMA_CHx( x=0..4)</i>	DMA通道选择
<b>输入参数{in}</b>	
<b>source</b>	DMA中断源
<i>DMA_INT_FTF</i>	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
<b>输出参数{out}</b>	
-	-
<b>返回值</b>	
-	-

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 3-102. 函数 **dma\_interrupt\_disable****

函数名称	dma_interrupt_disable
函数原型	void dma_interrupt_disable(dma_channel_enum channelx, uint32_t source);
功能描述	DMA通道x中断禁能
先决条件	无
被调用函数	无
输入参数{in}	
channelx	DMA通道
DMA_CHx(x=0..4)	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：

```
/* disable DMA channel0 interrupt */

dma_interrupt_disable(DMA_CH0, DMA_INT_FTF);
```

## 3.7. **EXTI**

EXTI是MCU中的中断/事件控制器，包括21个相互独立的边沿检测电路并且能够向处理器内核产生中断请求或唤醒事件。章节[3.7.1](#)描述了EXTI的寄存器列表，章节[3.7.2](#)对EXTI库函数进行说明。

### 3.7.1. 外设寄存器说明

EXTI寄存器列表如下表所示：

**表 3-103. EXTI 寄存器**

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

### 3.7.2. 外设库函数说明

EXTI库函数列表如下表所示:

**表 3-104. EXTI 库函数**

库函数名称	库函数描述
exti_deinit	复位EXTI，将EXTI的所有寄存器恢复成初始值
exti_init	初始化EXTI线x
exti_interrupt_enable	EXTI线x中断使能
exti_event_enable	EXTI线x事件使能
exti_interrupt_disable	EXTI线x中断禁能
exti_event_disable	EXTI线x事件禁能
exti_flag_get	获取EXTI线x标志位
exti_flag_clear	清除EXTI线x标志位
exti_interrupt_flag_get	获取EXTI线x中断标志位
exti_interrupt_flag_clear	清除EXTI线x中断标志位
exti_software_interrupt_enable	使能EXTI线x软件中断
exti_software_interrupt_disable	禁能EXTI线x软件中断

枚举类型 `exti_line_enum`

**表 3-105. 枚举类型 exti\_line\_enum**

枚举名称	枚举描述
EXTI_0	EXTI线0
EXTI_1	EXTI线1
EXTI_2	EXTI线2
EXTI_3	EXTI线3
EXTI_4	EXTI线4
EXTI_5	EXTI线5
EXTI_6	EXTI线6
EXTI_7	EXTI线7
EXTI_8	EXTI线8
EXTI_9	EXTI线9

枚举名称	枚举描述
EXTI_10	EXTI线10
EXTI_11	EXTI线11
EXTI_12	EXTI线12
EXTI_13	EXTI线13
EXTI_14	EXTI线14
EXTI_15	EXTI线15
EXTI_16	EXTI线16
EXTI_17	EXTI线17
EXTI_19	EXTI线19
EXTI_25	EXTI线25
EXTI_26	EXTI线26
EXTI_27	EXTI线27

### 枚举类型 exti\_mode\_enum

表 3-106. 枚举类型 exti\_mode\_enum

枚举名称	枚举描述
EXTI_INTERRUPT	EXTI中断模式
EXTI_EVENT	EXTI事件模式

### 枚举类型 exti\_trig\_type\_enum

表 3-107. 枚举类型 exti\_trig\_type\_enum

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

### 函数 exti\_deinit

函数exti\_deinit描述见下表：

表 3-108. 函数 exti\_deinit

函数名称	exti_deinit
函数原形	void exti_deinit(void);
功能描述	复位EXTI，将EXTI的所有寄存器恢复成初始值
先决条件	-
被调用函数	-
输入参数{in}	
-	-
输出参数{out}	
-	-
返回值	

-	-
---	---

例如：

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

```
exti_deinit();
```

### 函数 exti\_init

函数exti\_init描述见下表：

**表 3-109. 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	-
<b>输入参数{in}</b>	
linex	EXTI线x
EXTI_x	x=0..17,19,21
<b>输入参数{in}</b>	
mode	EXTI模式
EXTI_INTERRUPT	中断模式
EXTI_EVENT	事件模式
<b>输入参数{in}</b>	
trig_type	触发类型
EXTI_TRIG_RISING	上升沿触发
EXTI_TRIG_FALLING	下降沿触发
EXTI_TRIG_BOTH	上升沿和下降沿均触发
<b>输出参数{out}</b>	
-	-
<b>返回值</b>	
-	-

例如：

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

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

### 函数 exti\_interrupt\_enable

函数exti\_interrupt\_enable描述见下表：

**表 3-110. 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
EXTI_x	x=0,1,2..27
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\_event\_enable

函数exti\_event\_enable描述见下表：

**表 3-111. 函数 exti\_event\_enable**

函数名称	exti_event_enable
函数原形	void exti_event_enable(exti_line_enum linex);
功能描述	EXTI线x事件使能
先决条件	-
被调用函数	-
输入参数{in}	
linex	EXTI线x
EXTI_x	x=0,1,2..27
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\_interrupt\_disable

函数exti\_interrupt\_disable描述见下表：

**表 3-112. 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
EXTI_x	x=0,1,2..27
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\_disable

函数exti\_event\_disable描述见下表：

**表 3-113. 函数 exti\_event\_disable**

函数名称	exti_event_disable
函数原形	void exti_event_disable(exti_line_enum linex);
功能描述	EXTI线x事件禁能
先决条件	-
被调用函数	-
输入参数{in}	
linex	EXTI线x
EXTI_x	x=0,1,2..27
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\_flag\_get

函数exti\_flag\_get描述见下表：

表 3-114. 函数 exti\_flag\_get

函数名称	exti_flag_get
函数原形	FlagStatus exti_flag_get(exti_line_enum linex);
功能描述	获取EXTI线x标志位
先决条件	-
被调用函数	-
输入参数{in}	
linex	EXTI线x
EXTI_x	x=0,1,2..17, 19, 21
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 3-115. 函数 exti\_flag\_clear

函数名称	exti_flag_clear
函数原形	void exti_flag_clear(exti_line_enum linex);
功能描述	清除EXTI线x标志位
先决条件	-
被调用函数	-
输入参数{in}	
linex	EXTI线x
EXTI_x	x=0,1,2..17, 19, 21
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 3-116. 函数 exti\_interrupt\_flag\_get

函数名称	exti_interrupt_flag_get
函数原形	FlagStatus exti_interrupt_flag_get(exti_line_enum linex);
功能描述	获取EXTI线x中断标志位
先决条件	-
被调用函数	-
输入参数{in}	
linex	EXTI线x
EXTI_x	x=0,1,2..17, 19, 21
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 3-117. 函数 exti\_interrupt\_flag\_clear

函数名称	exti_interrupt_flag_clear
函数原形	void exti_interrupt_flag_clear(exti_line_enum linex);
功能描述	清除EXTI线x中断标志位
先决条件	-
被调用函数	-
输入参数{in}	
linex	EXTI线x
EXTI_x	x=0,1,2..17, 19, 21
输出参数{out}	
-	-
返回值	
-	-

例如：

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

exti_interrupt_flag_clear(EXTI_0);
```

### 函数 exti\_software\_interrupt\_enable

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

表 3-118. 函数 exti\_software\_interrupt\_enable

函数名称	exti_software_interrupt_enable
函数原形	void exti_software_interrupt_enable(exti_line_enum linex);
功能描述	使能EXTI线x软件中断
先决条件	-
被调用函数	-
输入参数{in}	
linex	EXTI线x
EXTI_x	x=0,1,2..17, 19, 21
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 3-119. 函数 exti\_software\_interrupt\_disable

函数名称	exti_software_interrupt_disable
函数原形	void exti_software_interrupt_disable(exti_line_enum linex);
功能描述	禁能EXTI线x软件中断
先决条件	-
被调用函数	-
输入参数{in}	
linex	EXTI线x
EXTI_x	x=0,1,2..17, 19, 21
输出参数{out}	
-	-
返回值	
-	-

例如：

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

## 3.8. FMC

FMC是MCU中的Flash控制器，其中包括存储数据的主编程块和选项字节。章节[3.8.1](#)描述了FMC的寄存器列表，章节[3.8.2](#)对FMC库函数进行说明。

### 3.8.1. 外设寄存器说明

FMC寄存器列表如下：

**表 3-120. 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

### 3.8.2. 外设库函数说明

FMC固件库函数列举如下表：

**表 3-121. 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_prefetch_enable	使能pre-fetch
fmc_prefetch_disable	失能pre-fetch
fmc_page_erase	FMC 页擦除
fmc_mass_erase	FMC 全片擦除
fmc_doubleword_program	在相应地址双字编程
fmc_word_program	在相应地址全字编程
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
option_byte_value_get	获取选项字节值
ob_erase	擦除选项字节
ob_write_protection_enable	使能写保护
ob_security_protection_config	配置安全保护

函数名称	函数描述
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

### 枚举类型 fmc\_state\_enum

表 3-122. 枚举类型 fmc\_state\_enum

枚举名称	枚举描述
FMC_READY	操作完成
FMC_BUSY	操作进行中
FMC_PGERR	编程错误
FMC_PGAERR	编程对齐错误
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

### 函数 fmc\_unlock

函数fmc\_unlock描述见下表:

表 3-123. 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 3-124. 函数 Function 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 3-125. 函数 fmc\_wscnt\_set**

函数名称	fmc_wscnt_set
函数原型	void fmc_wscnt_set(uint32_t wscnt);
功能描述	设置等待状态计数值
先决条件	-
被调用函数	-
输入参数{in}	
wscnt	等待状态计数值
WS_WSCNT_0	FMC 0个等待状态
WS_WSCNT_1	FMC 1个等待状态
WS_WSCNT_2	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\_prefetch\_enable

函数fmc\_prefetch\_enable描述见下表：

**表 3-126. 函数 fmc\_prefetch\_enable**

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

例如：

```
/* enable pre-fetch */

fmc_prefetch_enable( );
```

### 函数 fmc\_prefetch\_disable

函数fmc\_prefetch\_disable描述见下表：

**表 3-127. 函数 fmc\_prefetch\_disable**

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

-	-
返回值	
-	-

例如：

```
/* disable pre-fetch */

fmc_prefetch_disable();
```

### 函数 fmc\_page\_erase

函数fmc\_page\_erase描述见下表：

**表 3-128. 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3-122. 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 3-129. 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	
<b>fmc_state_enum</b>	FMC状态值，详情参考枚举变量 <a href="#">表3-122. 枚举类型fmc_state_enum</a>

例如：

```
/* erase whole chip */

fmc_state_enum state = fmc_mass_erase();
```

### 函数 fmc\_doubleword\_program

函数fmc\_doubleword\_program描述见下表：

**表 3-130. 函数 fmc\_doubleword\_program**

函数名称	fmc_doubleword_program
函数原型	fmc_state_enum fmc_doubleword_program(uint32_t address, uint64_t data);
功能描述	对相应地址双字编程
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	
<b>fmc_state_enum</b>	FMC状态值，详情参考枚举变量 <a href="#">表3-122. 枚举类型fmc_state_enum</a>

例如：

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

fmc_state_enum state = fmc_word_program(0x08004000, 0xaabbccddeeff0055);
```

### 函数 fmc\_word\_program

函数fmc\_word\_program描述见下表：

**表 3-131. 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}	

<b>data</b>	编程数据
<b>输出参数{out}</b>	
-	-
<b>返回值</b>	
<b>fmc_state_enum</b>	FMC状态值, 详情参考枚举变量 <a href="#">表3-122. 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 **ob\_unlock**

函数ob\_unlock描述见下表:

**表 3-132. 函数 ob\_unlock**

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

例如:

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

ob_unlock ( );
```

### 函数 **ob\_lock**

函数ob\_lock描述见下表:

**表 3-133. 函数 ob\_lock**

<b>函数名称</b>	ob_lock
<b>函数原型</b>	void ob_lock(void);
<b>功能描述</b>	锁定选项字节操作
<b>先决条件</b>	fmc_lock
<b>被调用函数</b>	-
<b>输入参数{in}</b>	
-	-

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 3-134. 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 **option\_byte\_value\_get**

函数option\_byte\_value\_get描述见下表：

**表 3-135. 函数 option\_byte\_value\_get**

函数名称	option_byte_value_get
函数原型	uint32_t option_byte_value_get(uint32_t addr);
功能描述	获取选项字节值
先决条件	-
被调用函数	-
输入参数{in}	
-	-
输出参数{out}	

-	-
返回值	
<b>uint32_t</b>	目标选项字节的值

例如：

```
/* get option byte value */

uint32_t temp;

temp = option_byte_value_get(0x1fff f800);
```

### 函数 **ob\_erase**

函数**ob\_erase**描述见下表：

**表 3-136. 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	
<b>fmc_state_enum</b>	FMC状态值，详情参考枚举变量 <a href="#">表3-122. 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 3-137. 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}	
<b>ob_wp</b>	写保护单元
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3-122. 枚举类型fmc_state_enum</a>

例如:

```
/* enable write protection */

fmc_state_enum state = ob_write_protection_enable (0x01);
```

### 函数 **ob\_security\_protection\_config**

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

**表 3-138. 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}	
<b>ob_spc</b>	安全保护
<i>FMC_NSPC</i>	无安全保护
<i>FMC_LSPC</i>	低保护级别
<i>FMC_HSPC</i>	高保护级别
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3-122. 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 3-139. 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}	

<b>ob_user</b>	用户定义的选项字节
<b>OB_FWDGT_HW</b>	硬件看门狗
<b>OB_DEEPSLEEP_RST</b>	进入深度睡眠时不复位
<b>OB_STDBY_RST</b>	进入深度睡眠时产生复位
<b>OB_BOOT1_SET_1</b>	BOOT1位是1
<b>OB_VDDA_DISABLE</b>	去使能 $V_{DDA}$ 监视器
<b>OB_SRAM_PARITY_ENABLE</b>	使能SRAM奇偶校验
<b>输出参数{out}</b>	
-	-
<b>返回值</b>	
<b>fmc_state_enum</b>	FMC状态值, 详情参考枚举变量 <a href="#">表3-122. 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,
                                      OB_STDBY_RST);
```

### 函数 **ob\_data\_program**

函数**ob\_data\_program**描述见下表:

**表 3-140. 函数 ob\_data\_program**

<b>函数名称</b>	ob_data_program
<b>函数原型</b>	fmc_state_enum ob_data_program(uint16_t data);
<b>功能描述</b>	编程数字选项字节
<b>先决条件</b>	ob_unlock
<b>被调用函数</b>	fmc_ready_wait
<b>输入参数{in}</b>	
<b>address</b>	编程数字选项字节地址
<b>OB_DATA_ADDR0</b>	编程数字选项字节地址0
<b>OB_DATA_ADDR1</b>	编程数字选项字节地址1
<b>输入参数{in}</b>	
<b>data</b>	所编程数值
<b>输出参数{out}</b>	
-	-
<b>返回值</b>	
<b>fmc_state_enum</b>	FMC状态值, 详情参考枚举变量 <a href="#">表3-122. 枚举类型fmc_state_enum</a>

例如:

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

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

### 函数 **ob\_user\_get**

函数ob\_user\_get描述见下表:

**表 3-141. 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	
<b>uint8_t</b>	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 3-142. 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	
<b>uint16_t</b>	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 3-143. 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		
<b>Uint16_t</b>	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 3-144. 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		
<b>uint8_t</b>	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 3-145. 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 3-146. 函数 fmc\_interrupt\_disable**

函数名称	fmc_interrupt_disable
函数原型	void fmc_interrupt_disable(uint32_t interrupt);
功能描述	除能FMC中断
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:

```
/* 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 3-147. 函数 fmc\_flag\_get**

函数名称	fmc_flag_get	
函数原型	FlagStatus fmc_flag_get(uint32_t flag);	
功能描述	检查标志是否置位	
先决条件	-	
被调用函数	-	
输入参数{in}		
flag	检查FMC标志	
FMC_FLAG_BUSY	FMC忙碌标志	
FMC_FLAG_PGER R	FMC操作错误标志	
FMC_FLAG_PGAE RR	FMC编程对齐错误标志	
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		
FlagStatus	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 3-148. 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	FMC操作错误标志	

<i>R</i>	
<i>FMC_FLAG_PGAE</i> <i>RR</i>	FMC编程对齐错误标志
<i>FMC_FLAG_WPER</i> <i>R</i>	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 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 3-149. 函数 fmc\_interrupt\_flag\_get**

函数名称	fmc_interrupt_flag_get
函数原型	FlagStatus fmc_interrupt_flag_get(fmc_interrupt_flag_enum flag);
功能描述	获取FMC中断标志状态
先决条件	-
被调用函数	-
输入参数{in}	
<i>flag</i>	中断标志
<i>FMC_INT_FLAG_P</i> <i>GERR</i>	FMC操作错误标志
<i>FMC_INT_FLAG_P</i> <i>GAERR</i>	FMC编程对齐错误标志
<i>FMC_INT_FLAG_W</i> <i>PERR</i>	FMC写保护错误标志
<i>FMC_INT_FLAG_E</i> <i>ND</i>	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 interrupt flag */

FlagStatus flag = fmc_interrupt_flag_get (FMC_INT_FLAG_PGERR);
```

### 函数 fmc\_interrupt\_flag\_clear

函数fmc\_interrupt\_flag\_clear描述见下表:

**表 3-150. 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 int_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
FMC_INT_FLAG_P GERR	FMC操作错误标志
FMC_INT_FLAG_P GAERR	FMC编程对齐错误标志
FMC_INT_FLAG_W PERR	FMC写保护错误标志
FMC_INT_FLAG_E ND	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_INT_FLAG_BANK0_PGERR);
```

### 函数 fmc\_state\_get

函数fmc\_state\_get描述见下表:

**表 3-151. 函数 fmc\_state\_get**

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

<b>fmc_state_enum</b>	FMC状态值，详情参考枚举变量 <a href="#">表3-122. 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 3-152. 函数 fmc\_ready\_wait**

函数名称	fmc_ready_wait
函数原型	fmc_state_enum fmc_ready_wait(uint32_t timeout);
功能描述	检查FMC是否准备好
先决条件	-
被调用函数	fmc_state_get();
输入参数{in}	
timeout	循环计数次数
输出参数{out}	
-	-
返回值	
<b>fmc_state_enum</b>	FMC状态值，详情参考枚举变量 <a href="#">表3-122. 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 );
```

## 3.9. FWDGT

独立看门狗定时器（FWDGT）是一个硬件计时电路，用来监测由软件故障导致的系统故障。适合于需要独立环境且对计时精度要求不高的场合。章节[3.9.1](#)描述了FWDGT的寄存器列表，章节[3.9.2](#)对FWDGT库函数进行说明。

### 3.9.1. 外设寄存器说明

FWDGT寄存器列表如下表所示：

**表 3-153. FWDGT 寄存器**

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

寄存器名称	寄存器描述
FWDGT_WND	窗口寄存器

### 3.9.2. 外设库函数说明

FWDGT库函数列表如下表所示:

**表 3-154. 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	失能对寄存器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时钟预分频数
fwdgt_reload_value_config	配置独立看门狗定时器计数器重装载值
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FWDG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 3-155. 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	-
被调用函数	-
<b>输入参数{in}</b>	
-	-
<b>输出参数{out}</b>	
-	-
<b>返回值</b>	
-	-

例如:

```
/* enable write access to FWDGT_PSC and FWDGT_RLD and FWDGT_WND */
fwdgt_write_enable();
```

### 函数 **fwdgt\_write\_disable**

函数fwdgt\_write\_disable描述见下表:

**表 3-156. 函数 fwdgt\_write\_disable**

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

例如:

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

fwdgt_write_disable();
```

### 函数 **fwdgt\_enable**

函数fwdgt\_enable描述见下表:

**表 3-157. 函数 fwdgt\_enable**

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

例如:

```
/* start the FWDGT counter */

fwdgt_enable();
```

### 函数 `fwdgt_prescaler_value_config`

函数`fwdgt_prescaler_value_config`描述见下表：

**表 3-158. 函数 `fwdgt_prescaler_value_config`**

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

例如：

```
/* set FWDGT prescaler to 4 */
ErrStatus flag;
flag = fwdgt_prescaler_value_config (FWDGT_PSC_DIV4);
```

### 函数 `fwdgt_reload_value_config`

函数`fwdgt_reload_value_config`描述见下表：

**表 3-159. 函数 `fwdgt_reload_value_config`**

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

例如：

```
/* set FWDGT reload value to 0xFFFF */
ErrStatus flag;
```

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

### 函数 **fwdgt\_window\_value\_reload**

函数fwdgt\_window\_value\_config描述见下表:

**表 3-160. 函数 fwdgt\_window\_value\_config**

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

例如:

```
/* set FWDGT window value to 0xFFFF */
ErrStatus flag;
flag = fwdgt_window_value_config (0xFFFF);
```

### 函数 **fwdgt\_counter\_reload**

函数fwdgt\_counter\_reload描述见下表:

**表 3-161. 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FWDG计数器	
先决条件	-	
被调用函数	-	
输入参数{in}		
-	-	
输出参数{out}		
-	-	
返回值		
-	-	

例如:

```
/* reload FWDGT counter */
```

```
fwdgt_counter_reload ( );
```

### 函数 fwdgt\_config

函数fwdgt\_config描述见下表:

表 3-162. 函数 fwdgt\_config

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

例如:

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

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

### 函数 fwdgt\_flag\_get

函数fwdgt\_flag\_get描述见下表:

表 3-163. 函数 fwdgt\_flag\_get

函数名称	fwdgt_flag_get
函数原型	FlagStatus fwdgt_flag_get(uint16_t flag);

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

例如：

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

FlagStatus status;

status = fwdgt_flag_get (FWDGT_FLAG_PUD);
```

## 3.10. GPIO

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

### 3.10.1. 外设寄存器说明

GPIO寄存器列表如下表所示：

**表 3-164. 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_OSPEED0	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
GPIOx_AFSEL0	备用功能选择寄存器0
GPIOx_AFSEL1	备用功能选择寄存器1

寄存器名称	寄存器描述
GPIOx_BC	位清除寄存器
GPIOx_TG	端口位翻转寄存器

### 3.10.2. 外设库函数说明

GPIO库函数列表如下表所示:

**表 3-165. GPIO 库函数**

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

#### 函数 gpio\_deinit

函数gpio\_deinit描述见下表:

**表 3-166. 函数 gpio\_deinit**

函数名称	gpio_deinit
函数原型	void gpio_deinit(uint32_t gpio_periph);
功能描述	复位外设GPIOx
先决条件	-
被调用函数	rcu_periph_reset_enable / rcu_periph_reset_disable
输入参数{in}	
gpio_periph	GPIO端口
GPIOx	端口选择(x = A,B,C,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 3-167. 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,F)
输入参数{in}	
mode	GPIO引脚模式
GPIO_MODE_INPUT	输入模式
GPIO_MODE_OUTPUT_T	输出模式
GPIO_MODE_AF	备用功能模式
GPIO_MODE_ANALOG	模拟模式
输入参数{in}	
pull_up_down	GPIO引脚上拉下拉电阻设置
GPIO_PUPD_NONE	悬空模式，无上拉和下拉
GPIO_PUPD_PULLUP	带上拉电阻
GPIO_PUPD_PULLDOWN	带下拉电阻
输入参数{in}	
pin	GPIO pin
GPIO_PIN_x	引脚选择(x=0..15) (GD32E231上不存在PB9/PC13)
GPIO_PIN_ALL	所有引脚 (GD32E231上不存在PB9/PC13)
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 3-168. 函数 gpio\_output\_options\_set**

函数名称	gpio_output_options_set
函数原型	void gpio_output_options_set(uint32_t gpio_periph, uint8_t otype, uint32_t speed, uint32_t pin);
功能描述	设置GPIO输出模式和速度
先决条件	-
被调用函数	-
<b>输入参数{in}</b>	
<b>gpio_periph</b>	GPIO端口
<b>GPIOx</b>	端口选择(x = A,B,C,F)
<b>输入参数{in}</b>	
<b>otype</b>	GPIO引脚输出模式
<b>GPIO_OTYPE_PP</b>	推挽输出模式
<b>GPIO_OTYPE_OD</b>	开漏输出模式
<b>输入参数{in}</b>	
<b>speed</b>	GPIO引脚输出最大速度
<b>GPIO_OSPEED_2MHZ</b>	最大输出速度为2MHz
<b>GPIO_OSPEED_10MHZ</b>	最大输出速度为10MHz
<b>GPIO_OSPEED_50MHZ</b>	最大输出速度为50MHz
<b>输入参数{in}</b>	
<b>pin</b>	GPIO引脚
<b>GPIO_PIN_x</b>	引脚选择 (x=0..15) (GD32E231上不存在PB9/PC13)
<b>GPIO_PIN_ALL</b>	所有引脚 (GD32E231上不存在PB9/PC13)
<b>输出参数{out}</b>	
-	-
<b>返回值</b>	
-	-

例如：

```
/* 配置PA0工作于推挽输出模式 */
```

```
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 3-169. 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	-
<b>输入参数{in}</b>	
gpio_periph	GPIO端口
GPIOx	端口选择(x = A,B,C,F)
<b>输入参数{in}</b>	
pin	GPIO引脚
GPIO_PIN_x	引脚选择 (x=0..15) (GD32E231上不存在PB9/PC13)
GPIO_PIN_ALL	所有引脚 (GD32E231上不存在PB9/PC13)
<b>输出参数{out}</b>	
-	-
<b>返回值</b>	
-	-

例如:

```
/* set PA0*/
gpio_bit_set (GPIOA, GPIO_PIN_0);
```

### 函数 **gpio\_bit\_reset**

函数**gpio\_bit\_reset**描述见下表:

**表 3-170. 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	-
<b>输入参数{in}</b>	
gpio_periph	GPIO端口
GPIOx	端口选择(x = A,B,C,F)
<b>输入参数{in}</b>	
pin	GPIO引脚
GPIO_PIN_x	引脚选择 (x=0..15) (GD32E231上不存在PB9/PC13)
GPIO_PIN_ALL	所有引脚 (GD32E231上不存在PB9/PC13)
<b>输出参数{out}</b>	

-	-
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 3-171. 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,F)
输入参数{in}	
pin	GPIO引脚
GPIO_PIN_x	引脚选择 (x=0..15) (GD32E231上不存在PB9/PC13)
GPIO_PIN_ALL	所有引脚 (GD32E231上不存在PB9/PC13)
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 3-172. 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,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 3-173. 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,F)
输入参数{in}	
pin	GPIO引脚
GPIO_PIN_x	引脚选择 (x=0..15) (GD32E231上不存在PB9/PC13)
GPIO_PIN_ALL	所有引脚 (GD32E231上不存在PB9/PC13)
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 3-174. 函数 gpio\_input\_port\_get**

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

例如：

```

/* get input value of Port A */
uint16_t port_state;
port_state = gpio_input_bit_get (GPIOA);

```

### 函数 **gpio\_output\_bit\_get**

函数gpio\_output\_bit\_get描述见下表：

**表 3-175. 函数 gpio\_output\_bit\_get**

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

<code>GPIO_PIN_X</code>	引脚选择 (x=0..15) (GD32E231上不存在PB9/PC13)
<code>GPIO_PIN_ALL</code>	所有引脚 (GD32E231上不存在PB9/PC13)
<b>输出参数{out}</b>	
-	-
<b>返回值</b>	
<code>FlagStatus</code>	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 3-176. 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	-
<b>输入参数{in}</b>	
<code>gpio_periph</code>	GPIO端口
<code>GPIOx</code>	端口选择(x = A,B,C,F)
<b>输出参数{out}</b>	
-	-
<b>返回值</b>	
<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 3-177. 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	-
<b>输入参数{in}</b>	
<b>gpio_periph</b>	GPIO 端口
<b>GPIOx</b>	GPIOx(x = A,B,C)
<b>输入参数{in}</b>	
<b>alt_func_num</b>	GPIO 引脚备用功能, 请参见特定设备的数据手册
<b>GPIO_AF_0</b>	<i>TIMER13, TIMER14, TIMER16, SPI0, SPI1, I2S0, CK_OUT, USART0, I2C0, I2C1, SWDIO, SWCLK</i>
<b>GPIO_AF_1</b>	<i>USART0, USART1, TIMER2, TIMER14, I2C0, I2C1</i>
<b>GPIO_AF_2</b>	<i>TIMER0, TIMER1, TIMER15, TIMER16, I2S0</i>
<b>GPIO_AF_3</b>	<i>I2C0, TIMER14</i>
<b>GPIO_AF_4 (port A,B only)</b>	<i>USART1, I2C0, I2C1, TIMER13</i>
<b>GPIO_AF_5 (port A,B only)</b>	<i>TIMER15, TIMER16, I2S0</i>
<b>GPIO_AF_6 (port A,B only)</b>	<i>SPI1</i>
<b>GPIO_AF_7 (port A,B only)</b>	<i>CMP</i>
<b>输入参数{in}</b>	
<b>pin</b>	GPIO引脚
<b>GPIO_PIN_x</b>	引脚选择 (x=0..15) (GD32E231上不存在PB9/PC13)
<b>GPIO_PIN_ALL</b>	所有引脚 (GD32E231上不存在PB9/PC13)
<b>输出参数{out}</b>	
-	-
<b>返回值</b>	
-	-

例如：

```
/*set PA0 alternate function 0*/
gpio_af_set(GPIOA, GPIO_AF_0, GPIO_PIN_0);
```

### 函数 **gpio\_pin\_lock**

函数**gpio\_pin\_lock**描述见下表：

表 3-178. 函数 **gpio\_pin\_lock**

函数名称	<b>gpio_pin_lock</b>
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	-

<b>被调用函数</b>	-
<b>输入参数{in}</b>	
<b>gpio_periph</b>	GPIO端口
<b>GPIOx</b>	端口选择(x = A,B)
<b>输入参数{in}</b>	
<b>pin</b>	GPIO引脚
<b>GPIO_PIN_x</b>	引脚选择 (x=0..15) (GD32E231上不存在PB9)
<b>GPIO_PIN_ALL</b>	所有引脚 (GD32E231上不存在PB9)
<b>输出参数{out}</b>	
-	-
<b>返回值</b>	
-	-

例如：

```
/* lock PA0 */
gpio_pin_lock (GPIOA, GPIO_PIN_0);
```

### 函数 **gpio\_bit\_toggle**

函数gpio\_bit\_toggle描述见下表：

**表 3-179. 函数 gpio\_bit\_toggle**

<b>函数名称</b>	gpio_bit_toggle
<b>函数原型</b>	void gpio_bit_toggle(uint32_t gpio_periph, uint32_t pin);
<b>功能描述</b>	翻转GPIO引脚状态
<b>先决条件</b>	-
<b>被调用函数</b>	-
<b>输入参数{in}</b>	
<b>gpio_periph</b>	GPIOx(x = A,B,C,F)
<b>GPIOx</b>	GPIOx(x = A,B,C,F)
<b>输入参数{in}</b>	
<b>pin</b>	GPIO引脚
<b>GPIO_PIN_x</b>	引脚选择 (x=0..15) (GD32E231上不存在PB9/PC13)
<b>GPIO_PIN_ALL</b>	所有引脚 (GD32E231上不存在PB9/PC13)
<b>输出参数{out}</b>	
-	-
<b>返回值</b>	
-	-

例如：

```
/* 翻转PA0 */
gpio_bit_toggle (GPIOA, GPIO_PIN_0);
```

### 函数 `gpio_port_toggle`

函数`gpio_port_toggle`描述见下表:

**表 3-180. 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}		
<code>gpio_periph</code>	GPIO端口	
<code>GPIOx</code>	GPIOx(x = A,B,C,F)	
输出参数{out}		
-	-	
返回值		
-	-	

例如:

```
/* toggle GPIOA*/
gpio_port_toggle (GPIOA);
```

## 3.11. I2C

I2C（内部集成电路总线）模块提供了符合工业标准的两线串行制接口，可用于MCU和外部I2C设备的通讯。章节[3.11.1](#)描述了I2C的寄存器列表，章节[3.11.2](#)对I2C库函数进行说明。

### 3.11.1. 外设寄存器说明

I2C寄存器列表如下表所示:

**表 3-181. 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_SAMCS	SAM控制状态寄存器

寄存器名称	寄存器描述
I2C_FMPCFG	快速+ 模式配置寄存器

### 3.11.2. 外设库函数说明

I2C库函数列表如下表所示：

**表 3-182. I2C 库函数**

库函数名称	库函数描述
i2c_deinit	复位外设I2C
i2c_clock_config	配置I2C时钟
i2c_mode_addr_config	配置I2C地址
i2c_smbus_type_config	SMBus类型选择
i2c_ack_config	是否发送ACK
i2c_ackpos_config	ACK位置配置
i2c_master_addressing	主机发送从机地址
i2c_dualaddr_enable	双地址模式使能
i2c_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_enable	I2C DMA模式使能
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_enable	报文错误校验使能
i2c_pec_transfer_enable	传输PEC值使能
i2c_pec_value_get	获取报文错误校验值
i2c_smbus_issue_alert	通过SMBA引脚发送警告
i2c_smbus_arp_enable	SMBus下ARP协议是否开启
i2c_sam_enable	使能SAM_V接口
i2c_sam_disable	关闭SAM_V接口
i2c_sam_timeout_enable	使能SAM_V接口超时检测
i2c_sam_timeout_disable	关闭SAM_V接口超时检测
i2c_flag_get	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

库函数名称	库函数描述
i2c_interrupt_flag_clear	中断标志位清除

### 函数 i2c\_deinit

函数i2c\_deinit描述见下表：

**表 3-183. 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 3-184. 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 3-185. 函数 i2c\_mode\_addr\_config**

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}	
i2cmode	模式选择
I2C_I2CMODE_ENABLE	I2C 模式
I2C_SMBUSMODE_ENABLE	SMBus 模式
输入参数{in}	
addformat	7bits 或 10bits
I2C_ADDFORMAT_7BITS	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 3-186. 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_DEVI_CE	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 3-187. 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
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* I2C0 will sent ACK */

i2c_ack_config (I2C0, I2C_ACK_ENABLE);
```

### 函数 i2c\_ackpos\_config

函数i2c\_ackpos\_config描述见下表：

**表 3-188. 函数 i2c\_ackpos\_config**

函数名称	i2c_ackpos_config
函数原型	void i2c_ackpos_config(uint32_t i2c_periph, uint32_t pos);
功能描述	ACK位置配置
先决条件	-
被调用函数	-
输入参数{in}	
<b>i2c_periph</b>	I2C外设
<i>I2Cx</i>	(x=0,1)
输入参数{in}	
<b>pos</b>	ACK位置
<i>I2C_ACKPOS_CUR</i> <i>RENT</i>	当前正在接收的字节是否发送ACK
<i>I2C_ACKPOS_NEX</i> <i>T</i>	下一个接收的字节是否发送ACK
输出参数{out}	
-	-
返回值	
-	-

例如：

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

i2c_ackpos_config (I2C0, I2C_ACKPOS_CURRENT);
```

### 函数 i2c\_master\_addressing

函数i2c\_master\_addressing描述见下表:

**表 3-189. 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)
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 3-190. 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 3-191. 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 3-192. 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

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

例如：

```
/* enable I2C0 */

i2c_enable (I2C0);
```

### 函数 i2c\_disable

函数i2c\_disable描述见下表：

**表 3-193. 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	-
<b>输入参数{in}</b>	
i2c_periph	I2C外设
<i>I2Cx</i>	(x=0,1)
<b>输出参数{out}</b>	
-	-
<b>返回值</b>	
-	-

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 3-194. 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	-
<b>输入参数{in}</b>	

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

例如：

```
/* 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 3-195. 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	-
<b>输入参数{in}</b>	
<b>i2c_periph</b>	I2C外设
<i>I2Cx</i>	(x=0,1)
<b>输出参数{out}</b>	
-	-
<b>返回值</b>	
-	-

例如：

```
/* 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 3-196. 函数 i2c\_data\_transmit**

函数名称	i2c_data_transmit
函数原型	void i2c_data_transmit(uint32_t i2c_periph, uint8_t data);
功能描述	发送数据
先决条件	-
被调用函数	-

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

例如：

```
/* I2C0 transmit data */

i2c_data_transmit (I2C0, 0x80);
```

### 函数 i2c\_data\_receive

函数i2c\_data\_receive描述见下表：

表 3-197. 函数 i2c\_data\_receive

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

例如：

```
/* I2C0 receive data */

uint8_t i2c_receiver;

i2c_receiver = i2c_data_receive (I2C0);
```

### 函数 i2c\_dma\_enable

函数i2c\_dma\_enable描述见下表：

表 3-198. 函数 i2c\_dma\_enable

函数名称	i2c_dma_enable

函数原型	void i2c_dma_enable(uint32_t i2c_periph, uint32_t dmastate);
功能描述	I2C DMA模式使能
先决条件	-
被调用函数	-
<b>输入参数{in}</b>	
i2c_periph	I2C外设
I2Cx	(x=0,1)
<b>输入参数{in}</b>	
dmastate	开启或关闭
I2C_DMA_ON	DMA模式开启
I2C_DMA_OFF	DMA模式关闭
<b>输出参数{out}</b>	
-	-
<b>返回值</b>	
-	-

例如：

```
/* I2C0 DMA mode enable */
i2c_dma_enable (I2C0, I2C_DMA_ON);
```

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

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

**表 3-199. 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	-
<b>输入参数{in}</b>	
i2c_periph	I2C外设
I2Cx	(x=0,1)
<b>输入参数{in}</b>	
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
<b>输出参数{out}</b>	
-	-
<b>返回值</b>	
-	-

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 3-200. 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 3-201. 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}	

<b>i2c_periph</b>	I2C外设
<i>I2Cx</i>	(x=0,1)
<b>输入参数{in}</b>	
<b>gcallpara</b>	是否响应广播呼叫
<i>I2C_GCEN_ENABLE</i> <i>E</i>	从机响应广播呼叫
<i>I2C_GCEN_DISABLE</i> <i>E</i>	从机不响应广播呼叫
<b>输出参数{out}</b>	
-	-
<b>返回值</b>	
-	-

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 3-202. 函数 i2c\_software\_reset\_config**

<b>函数名称</b>	i2c_software_reset_config
<b>函数原型</b>	void i2c_software_reset_config(uint32_t i2c_periph, uint32_t sreset);
<b>功能描述</b>	配置I2C软件复位
<b>先决条件</b>	-
<b>被调用函数</b>	-
<b>输入参数{in}</b>	
<b>i2c_periph</b>	I2C外设
<i>I2Cx</i>	(x=0,1)
<b>输入参数{in}</b>	
<b>sreset</b>	是否复位
<i>I2C_SRESET_SET</i>	复位
<i>I2C_SRESET_RESET</i> <i>ET</i>	没有复位
<b>输出参数{out}</b>	
-	-
<b>返回值</b>	
-	-

例如：

```
/* software reset I2C0*/
i2c_software_reset_config (I2C0, I2C_SRESET_SET);
```

### 函数 i2c\_pec\_enable

函数i2c\_pec\_enable描述见下表:

表 3-203. 函数 i2c\_pec\_enable

函数名称	i2c_pec_enable
函数原型	void i2c_pec_enable(uint32_t i2c_periph, uint32_t pecstate);
功能描述	报文错误校验使能
先决条件	-
被调用函数	-
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:

```
/* enable I2C PEC calculation */
i2c_pec_enable (I2C0, I2C_PEC_ENABLE);
```

### 函数 i2c\_pec\_transfer\_enable

函数i2c\_pec\_transfer\_enable描述见下表:

表 3-204. 函数 i2c\_pec\_transfer\_enable

函数名称	i2c_pec_transfer_enable
函数原型	void i2c_pec_transfer_enable(uint32_t i2c_periph, uint32_t pecpara);
功能描述	I2C是否传输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
I2C_PECTRANS_DISABLE	不传输PEC

SABLE	
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* I2C0 transfer PEC */

i2c_pec_transfer_enable (I2C0, I2C_PECTRANS_ENABLE);
```

#### 函数 i2c\_pec\_value\_get

函数i2c\_pec\_value\_get描述见下表：

**表 3-205. 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\_issue\_alert

函数i2c\_smbus\_issue\_alert描述见下表：

**表 3-206. 函数 i2c\_smbus\_issue\_alert**

函数名称	i2c_smbus_issue_alert
函数原型	void i2c_smbus_issue_alert(uint32_t i2c_periph, uint32_t smbuspara);
功能描述	通过SMBA引脚发送警告
先决条件	-
被调用函数	-

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

例如：

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

i2c_smbus_issue_alert (I2C0, I2C_SALTSEND_ENABLE);
```

#### 函数 i2c\_smbus\_arp\_enable

函数i2c\_smbus\_arp\_enable描述见下表：

表 3-207. 函数 i2c\_smbus\_arp\_enable

输入参数{in}	
<b>i2c_periph</b>	I2C外设
<i>I2Cx</i>	(x=0,1)
输入参数{in}	
<b>arpstate</b>	SMBus下ARP协议是否开启
<i>I2C_ARP_ENABLE</i>	使能ARP
<i>I2C_ARP_DISABLE</i>	关闭ARP
输出参数{out}	
-	-
返回值	
-	-

例如：

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

i2c_smbus_arp_enable (I2C0, I2C_ARP_ENABLE);
```

### 函数 i2c\_sam\_enable

函数i2c\_sam\_enable描述见下表:

**表 3-208. 函数 i2c\_sam\_enable**

函数名称	i2c_sam_enable
函数原型	void i2c_sam_enable(uint32_t i2c_periph);
功能描述	使能SAM_V接口
先决条件	-
被调用函数	-
输入参数{in}	
i2c_periph	I2C外设
I2Cx	(x=0,1)
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* enable I2C0 SAM_V interface*/
i2c_sam_enable (I2C0);
```

### 函数 i2c\_sam\_disable

函数i2c\_sam\_disable描述见下表:

**表 3-209. 函数 i2c\_sam\_disable**

函数名称	i2c_sam_disable
函数原型	void i2c_sam_disable (uint32_t i2c_periph);
功能描述	关闭SAM_V接口
先决条件	-
被调用函数	-
输入参数{in}	
i2c_periph	I2C外设
I2Cx	(x=0,1)
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* disable I2C0 SAM_V interface*/
i2c_sam_disable (I2C0);
```

### 函数 i2c\_sam\_timeout\_enable

函数i2c\_sam\_timeout\_enable描述见下表:

**表 3-210. 函数 i2c\_sam\_timeout\_enable**

函数名称	i2c_sam_timeout_enable
函数原型	void i2c_sam_timeout_enable (uint32_t i2c_periph);
功能描述	使能SAM_V接口超时检测
先决条件	-
被调用函数	-
输入参数{in}	
i2c_periph	I2C外设
I2Cx	(x=0,1)
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* enable I2C0 SAM_V interface timeout detect */
i2c_sam_timeout_enable (I2C0);
```

### 函数 i2c\_sam\_timeout\_disable

函数i2c\_sam\_timeout\_disable描述见下表:

**表 3-211. 函数 i2c\_sam\_timeout\_disable**

函数名称	i2c_sam_timeout_disable
函数原型	void i2c_sam_timeout_disable (uint32_t i2c_periph);
功能描述	关闭SAM_V接口超时检测
先决条件	-
被调用函数	-
输入参数{in}	
i2c_periph	I2C外设
I2Cx	(x=0,1)
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* disable I2C0 SAM_V interface timeout detect */
i2c_sam_timeout_disable (I2C0);
```

### 函数 i2c\_flag\_get

函数i2c\_flag\_get描述见下表:

**表 3-212. 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_SBSN_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)
I2C_FLAG_DEFSM_B	从机模式下SMBus主机地址头

<i>I2C_FLAG_HSTSM_B</i>	从机模式下监测到SMBus主机地址头
<i>I2C_FLAG_DUMOD</i>	从机模式下双标志位表明哪个地址和双地址模式匹配
<i>I2C_FLAG_TFF</i>	发送帧下降沿标志
<i>I2C_FLAG_TFR</i>	发送帧上升沿标志
<i>I2C_FLAG_RFF</i>	接收帧下降沿标志
<i>I2C_FLAG_RFR</i>	接收帧上升沿标志
<b>输出参数{out}</b>	
-	-
<b>返回值</b>	
<b>FlagStatus</b>	SET / RESET

例如：

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

FlagStatus flag_state = RESET;

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

### 函数 i2c\_flag\_clear

函数i2c\_flag\_clear描述见下表：

**表 3-213. 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	-
<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>flag</b>	标志位类型
<i>I2C_FLAG_SMBAL_T</i>	SMBus警报状态
<i>I2C_FLAG_SMBTO</i>	SMBus模式下超时信号
<i>I2C_FLAG_PECER_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_RB</i>	主机模式下仲裁丢失
<i>I2C_FLAG_BERR</i>	总线错误

<i>I2C_FLAG_ADDSE_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
<i>I2C_FLAG_TFF</i>	发送帧下降沿标志
<i>I2C_FLAG_TFR</i>	发送帧上升沿标志
<i>I2C_FLAG_RFF</i>	接收帧下降沿标志
<i>I2C_FLAG_RFR</i>	接收帧上升沿标志
<b>输出参数{out}</b>	
-	-
<b>返回值</b>	
-	-

例如：

```
/* clear a bus error flag*/
i2c_flag_clear (I2C0, I2C_FLAG_BERR);
```

#### 函数 i2c\_interrupt\_enable

函数i2c\_interrupt\_enable描述见下表：

**表 3-214. 函数 i2c\_interrupt\_enable**

<b>i2c_interrupt_enable</b>	
<b>函数原型</b>	void i2c_interrupt_enable(uint32_t i2c_periph, i2c_interrupt_enum interrupt);
<b>功能描述</b>	中断使能
<b>先决条件</b>	-
<b>被调用函数</b>	-
<b>输入参数{in}</b>	
<i>i2c_periph</i>	I2C外设
<i>I2Cx</i>	(x=0,1)
<b>输入参数{in}</b>	
<i>interrupt</i>	中断类型
<i>I2C_INT_ERR</i>	错误中断使能
<i>I2C_INT_EV</i>	事件中断使能
<i>I2C_INT_BUF</i>	缓冲区中断使能
<i>I2C_INT_TFF</i>	发送帧下降沿中断使能
<i>I2C_INT_TFR</i>	发送帧上升沿中断使能
<i>I2C_INT_RFF</i>	接收帧下降沿中断使能
<i>I2C_INT_RFR</i>	接收帧上升沿中断使能
<b>输出参数{out}</b>	
-	-
<b>返回值</b>	
-	-

例如：

```
/* enable I2C0 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 3-215. 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	中断除能
先决条件	-
被调用函数	-
输入参数{in}	
i2c_periph	I2C外设
I2Cx	(x=0,1)
输入参数{in}	
interrupt	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
I2C_INT_TFF	发送帧下降沿中断使能
I2C_INT_TFR	发送帧上升沿中断使能
I2C_INT_RFF	接收帧下降沿中断使能
I2C_INT_RFR	接收帧上升沿中断使能
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 3-216. 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

<b>先决条件</b>	-
<b>被调用函数</b>	-
<b>输入参数{in}</b>	
<i>i2c_periph</i>	I2C外设
<i>I2Cx</i>	(x=0,1)
<b>输入参数{in}</b>	
<b>int_flag</b>	中断标志
<i>I2C_INT_FLAG_SBSEND</i>	主机模式下发送START起始位
<i>I2C_INT_FLAG_ADSENDC</i>	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
<i>I2C_INT_FLAG_BT</i>	字节发送结束
<i>I2C_INT_FLAG_ADD10SEND</i>	主机模式下10位地址地址头被发送
<i>I2C_INT_FLAG_STPDET</i>	从机模式下监测到STOP结束位
<i>I2C_INT_FLAG_RBNE</i>	接收期间I2C_DATA非空
<i>I2C_INT_FLAG_TB</i>	发送期间I2C_DATA为空
<i>I2C_INT_FLAG_BERRR</i>	总线错误
<i>I2C_INT_FLAG_LOSTARB</i>	主机模式下仲裁丢失
<i>I2C_INT_FLAG_AER</i>	应答错误
<i>I2C_INT_FLAG_OVERRR</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_PECERR</i>	接收数据时PEC错误
<i>I2C_INT_FLAG_SMBTO</i>	SMBus模式下超时信号
<i>I2C_INT_FLAG_SMBAL</i>	SMBus警报状态
<i>I2C_INT_FLAG_TFF</i>	发送帧下降沿中断标志位
<i>I2C_INT_FLAG_TFR</i>	发送帧上升沿中断标志位
<i>I2C_INT_FLAG_RFF</i>	接收帧下降沿中断标志位
<i>I2C_INT_FLAG_RFRR</i>	接收帧上升沿中断标志位

输出参数{out}	
-	-
返回值	
<b>FlagStatus</b>	SET / RESET

例如：

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

FlagStatus flag_state = RESET;

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

### 函数 **i2c\_interrupt\_flag\_clear**

函数*i2c\_interrupt\_flag\_clear*描述见下表：

**表 3-217. 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}	
<b>i2c_periph</b>	I2C外设
<b>I2Cx</b>	(x=0,1)
输入参数{in}	
<b>int_flag</b>	中断标志
<b>I2C_INT_FLAG_AD_DSEND</b>	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
<b>I2C_INT_FLAG_BE_RR</b>	总线错误
<b>I2C_INT_FLAG_LO_STARB</b>	主机模式下仲裁丢失
<b>I2C_INT_FLAG_AE_RR</b>	应答错误
<b>I2C_INT_FLAG_OU_ERR</b>	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
<b>I2C_INT_FLAG_PE_CERR</b>	接收数据时PEC错误
<b>I2C_INT_FLAG_SM_BTO</b>	SMBus模式下超时信号
<b>I2C_INT_FLAG_SM_BALT</b>	SMBus警报状态
<b>I2C_INT_FLAG_TF</b>	发送帧下降沿中断标志位

<i>F</i>	
<i>I2C_INT_FLAG_TF</i> <i>R</i>	发送帧上升沿中断标志位
<i>I2C_INT_FLAG_RF</i> <i>F</i>	接收帧下降沿中断标志位
<i>I2C_INT_FLAG_RF</i> <i>R</i>	接收帧上升沿中断标志位
<b>输出参数{out}</b>	
-	-
<b>返回值</b>	
-	-

例如：

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

i2c_interrupt_flag_clear (I2C0, I2C_INT_FLAG_AERR);
```

## 3.12. MISC

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

### 3.12.1. 外设寄存器说明

**表 3-218. NVIC 寄存器**

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

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

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

**表 3-219. Systick 寄存器**

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

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

### 3.12.2. 外设库函数说明

**枚举类型 IRQn\_Type**

**表 3-220. 枚举类型 IRQn\_Type**

成员名称	功能描述
WWDGT_IRQHandler	窗口看门狗中断
LVD_IRQHandler	连接到 EXTI 线的 LVD 中断
RTC_IRQHandler	RTC 全局中断
FMC_IRQHandler	FMC 全局中断
RCU_IRQHandler	RCU 全局中断
EXTI0_1_IRQHandler	EXTI 线 0 中断
EXTI2_3_IRQHandler	EXTI 线 1 中断
EXTI4_15_IRQHandler	EXTI 线 2 中断
DMA_Channel0_IRQHandler	DMA0 通道 0 全局中断
DMA_Channel1_2_IRQHandler	DMA0 通道 1 全局中断
DMA_Channel3_4_IRQHandler	DMA0 通道 2 全局中断
ADC_CMP_IRQHandler	ADC0 和 ADC1 全局中断
TIMER0_BRK_UP_TRG_COM_IRQHandler	TIMER0 中止, 更新, 触发与通道换相中断
TIMER0_Channel_IRQHandler	TIMER0 捕获比较中断
TIMER2_IRQHandler	TIMER2 全局中断
TIMER5_IRQHandler	TIMER5 全局中断
TIMER13_IRQHandler	TIMER13 全局中断
TIMER14_IRQHandler	TIMER14 全局中断
TIMER15_IRQHandler	TIMER15 全局中断
TIMER16_IRQHandler	TIMER16 全局中断
I2C0_EV_IRQHandler	I2C0 事件中断
I2C1_EV_IRQHandler	I2C1 事件中断

SPI0_IRQHandler	SPI0 全局中断
SPI1_IRQHandler	SPI1 全局中断
USART0_IRQHandler	USART0 全局中断
USART1_IRQHandler	USART1 全局中断
I2C0_ER_IRQHandler	I2C0 错误中断
I2C1_ER_IRQHandler	I2C1 错误中断

MISC 库函数列表如下表所示：

**表 3-221. 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

### 函数 nvic\_irq\_enable

函数nvic\_irq\_enable描述见下表：

**表 3-222. 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3-220. 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_IRQHandler,1);
```

### 函数 nvic\_irq\_disable

函数nvic\_irq\_disable描述见下表:

**表 3-223. 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3-220. 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);
```

**表 3-224. 函数 nvic\_system\_reset**

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

例如:

```
/* reset the MCU*/

nvic_system_reset();
```

### 函数 nvic\_vector\_table\_set

函数nvic\_vector\_table\_set描述见下表:

**表 3-225. 函数 nvic\_vector\_table\_set**

函数名称	nvic_vector_table_set	
------	-----------------------	--

函数原形	void nvic_vector_table_set(uint32_t nvic_vict_tab, uint32_t offset);
功能描述	设置向量表地址
先决条件	-
被调用函数	-
<b>输入参数{in}</b>	
<b>nvic_vict_tab</b>	RAM 或者 FLASH基地址
<i>NVIC_VECTTAB_R AM</i>	RAM 基地址
<i>NVIC_VECTTAB_F LASH</i>	FLASH基地址
<b>输入参数{in}</b>	
<b>offset</b>	向量表偏移量（向量表地址=基地址+偏移量）
<b>输出参数{out}</b>	
-	-
<b>返回值</b>	
-	-

例如：

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

nvic_vector_table_set (NVIC_VECTTAB_FLASH,0x200);
```

### 函数 **system\_lowpower\_set**

函数**system\_lowpower\_set**描述见下表：

**表 3-226. 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	-
<b>输入参数{in}</b>	
<b>lowpower_mode</b>	系统低功耗模式的状态
<i>SCB_LPM_SLEEP_ EXIT_ISR</i>	该位为1时，退出ISR时一直处于低功耗模式
<i>SCB_LPM_DEEPSL EEP</i>	该位为1时，系统处于deep sleep模式
<i>SCB_LPM_WAKE_ BY_ALL_INT</i>	该位为1时，低功耗模式可以被所有中断唤醒（无论中断是否被使能）
<b>输出参数{out}</b>	
-	-
<b>返回值</b>	
-	-

例如：

```
/* the system always enter low power mode by exiting from ISR */

system_lowpower_set(SCB_LPM_SLEEP_EXIT_ISR);
```

### 函数 **system\_lowpower\_reset**

函数system\_lowpower\_reset描述见下表：

**表 3-227. 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 3-228. 函数 systick\_clksource\_set**

函数名称	systick_clksource_set
函数原形	void systick_clksource_set(uint32_t systick_clksource);
功能描述	设置SysTick时钟源
先决条件	-
被调用函数	-
输入参数{in}	
systick_clksource	SysTick时钟源

<code>SYSTICK_CLKSOU</code>	SysTick时钟源为AHB时钟
<code>RCE_HCLK</code>	
<code>SYSTICK_CLKSOU</code>	SysTick时钟源为AHB时钟的8分频
<code>RCE_HCLK_DIV8</code>	
<b>输出参数{out}</b>	
-	-
<b>返回值</b>	
-	-

例如：

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

## 3.13. PMU

电源管理单元提供了三种省电模式，包括睡眠模式，深度睡眠模式和待机模式。章节 [3.13.1](#) 描述了 PMU 的寄存器列表，章节 [3.13.2](#) 对 PMU 库函数进行说明。

### 3.13.1. 外设寄存器说明

PMU 寄存器列表如下表所示：

**表 3-229. PMU 寄存器**

寄存器名称	寄存器描述
<code>PMU_CTL</code>	PMU控制寄存器
<code>PMU_CS</code>	PMU控制和状态寄存器

### 3.13.2. 外设库函数说明

PMU 库函数列表如下表所示：

**表 3-230. PMU 库函数**

库函数名称	库函数描述
<code>pmu_deinit</code>	复位外设PMU
<code>pmu_lvd_select</code>	选择低压检测阈值
<code>pmu_ldo_output_select</code>	LDO输出电压选择
<code>pmu_lvd_disable</code>	关闭低压检测器
<code>pmu_to_sleepmode</code>	进入睡眠模式
<code>pmu_to_deepsleepmode</code>	进入深度睡眠模式
<code>pmu_to_standbymode</code>	进入待机模式
<code>pmu_wakeup_pin_enable</code>	WKUP引脚唤醒使能
<code>pmu_wakeup_pin_disable</code>	WKUP引脚唤醒失能
<code>pmu_backup_write_enable</code>	备份域写使能

库函数名称	库函数描述
pmu_backup_write_disable	备份域写失能
pmu_flag_clear	清除标志位
pmu_flag_get	获取标志位

### 函数 pmu\_deinit

函数 pmu\_deinit 描述见下表:

**表 3-231. 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 3-232. 函数 pmu\_lvd\_select**

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

<i>PMU_LVDT_6</i>	电压阈值为3.0V
<i>PMU_LVDT_7</i>	电压阈值为3.1V
<b>输出参数{out}</b>	
-	-
<b>返回值</b>	
-	-

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 3-233. 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	-
<b>输入参数{in}</b>	
<b>ldo_output</b>	输出电压模式
<i>PMU_LDOVS_LOW</i>	输出低电压模式
<i>PMU_LDOVS_HIGH</i>	输出高电压模式
<b>输出参数{out}</b>	
-	-
<b>返回值</b>	
-	-

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 3-234. 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\_to\_sleepmode

函数 pmu\_to\_sleepmode 描述见下表：

**表 3-235. 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 3-236. 函数 pmu\_to\_deepsleepmode**

函数名称	pmu_to_deepsleepmode
函数原型	void pmu_to_deepsleepmode(uint32_t ldo,uint8_t deepsleepmodecmd);

功能描述	进入深度睡眠模式
先决条件	-
被调用函数	-
<b>输入参数{in}</b>	
<b>ldo</b>	LDO工作模式
<b>PMU_LDO_NORMAL</b> <i>L</i>	当系统进入深度睡眠模式时，LDO仍正常工作
<b>PMU_LDO_LOWPO</b> <i>WER</i>	当系统进入深度睡眠模式时，LDO进入低功耗模式
<b>输入参数{in}</b>	
<b>deepsleepmodecm</b> <i>d</i>	进入深度睡眠模式命令
<b>WFI_CMD</b>	WFI命令
<b>WFE_CMD</b>	WFE命令
<b>输出参数{out}</b>	
-	-
<b>返回值</b>	
-	-

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 3-237. 函数 pmu\_to\_standbymode

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

例如：

---

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

```
pmu_to_standby (WFI_CMD);
```

### 函数 pmu\_wakeup\_pin\_enable

函数 pmu\_wakeup\_pin\_enable 描述见下表:

**表 3-238. 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_PI_N0	WKUP Pin 0 (PA0)
PMU_WAKEUP_PI_N1	WKUP Pin 1 (PC13)
PMU_WAKEUP_PI_N5	WKUP Pin 5 (PB5)
PMU_WAKEUP_PI_N6	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 3-239. 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}	

<b>wakeup_pin</b>	Wakeup pin
<i>PMU_WAKEUP_PIN0</i>	WKUP Pin 0 (PA0)
<i>PMU_WAKEUP_PIN1</i>	WKUP Pin 1 (PC13)
<i>PMU_WAKEUP_PIN5</i>	WKUP Pin 5 (PB5)
<i>PMU_WAKEUP_PIN6</i>	WKUP Pin 6 (PB15)
<b>输出参数{out}</b>	
-	-
<b>返回值</b>	
-	-

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 3-240. 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	-
<b>输入参数{in}</b>	
-	-
<b>输出参数{out}</b>	
-	-
<b>返回值</b>	
-	-

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 3-241. 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\_clear

函数 pmu\_flag\_clear 描述见下表：

**表 3-242. 函数 pmu\_flag\_clear**

函数名称	pmu_flag_clear
函数原型	void pmu_flag_clear(uint32_t flag_clear);
功能描述	清除标志位
先决条件	-
被调用函数	-
输入参数{in}	
flag_clear	标志位
PMU_FLAG_RESET_WAKEUP	清除唤醒标志
PMU_FLAG_RESET_STANDBY	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);
```

### 函数 pmu\_flag\_get

函数 pmu\_flag\_get 描述见下表:

**表 3-243. 函数 pmu\_flag\_get**

函数名称	pmu_flag_get
函数原型	FlagStatus pmu_flag_get(uint32_t flag);
功能描述	获取标志位
先决条件	-
被调用函数	-
输入参数{in}	
flag_clear	标志位
PMU_FLAG_WAKEUP	唤醒标志
PMU_FLAG_STANDBY	待机标志
PMU_FLAG_LVD	低电压状态标志
输出参数{out}	
-	-
返回值	
FlagStatus	SET或RESET

例如:

```
/* get flag state */

FlagStatus status;

status = pmu_flag_get (PMU_FLAG_WAKEUP);
```

## 3.14. RCU

RCU 是复位和时钟单元，复位控制包括三种控制方式：电源复位、系统复位和备份域复位。时钟控制单元提供了一系列频率的时钟功能。章节 [3.14.1](#) 描述了 RCU 的寄存器列表，章节 [3.14.2](#) 对 RCU 库函数进行说明。

### 3.14.1. 外设寄存器说明

RCU 寄存器列表如下表所示：

**表 3-244. 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

寄存器名称	寄存器描述
RCU_APB2RST	APB2复位寄存器
RCU_APB1RST	APB1复位寄存器
RCU_AHBBEN	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_VKEY	电源解锁寄存器
RCU_DSV	深度睡眠模式电压寄存器

### 3.14.2. 外设库函数说明

RCU库函数列表如下表所示：

**表 3-245. 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_ckout_config	配置CKOUT时钟源选择及分频系数
rcu_pll_config	配置主PLL时钟
rcu_usart_clock_config	配置串口时钟
rcu_rtc_clock_config	配置RTC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

库函数名称	库函数描述
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
rcu_osc_stab_wait	等待振荡器稳定标志位置位或振荡器起振超时
rcu_osc_on	打开振荡器
rcu_osc_off	关闭振荡器
rcu_osc_bypass_mode_enable	使能时钟旁路模式
rcu_osc_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

### 函数 **rcu\_deinit**

函数rcu\_deinit描述见下表：

**表 3-246. 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	-
<b>输入参数{in}</b>	
-	-
<b>输出参数{out}</b>	
-	-
<b>返回值</b>	
-	-

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 3-247. 函数 `rcu_periph_clock_enable`**

函数名称	<code>rcu_periph_clock_enable</code>
函数原形	<code>void rcu_periph_clock_enable(rcu_periph_enum periph);</code>
功能描述	使能外设时钟
先决条件	-
被调用函数	-
<b>输入参数{in}</b>	
<code>periph</code>	RCU外设, 具体参考 <code>rcu_periph_enum</code>
<code>RCU_GPIOx</code>	GPIOx时钟(x=A,B,C,F)
<code>RCU_DMA</code>	DMA时钟
<code>RCU_CRC</code>	CRC时钟
<code>RCU_CFGCMP</code>	CFGCMP时钟
<code>RCU_ADC</code>	ADC时钟
<code>RCU_TIMERx</code>	TIMERx时钟(x=0,2,5,13,14,15,16)
<code>RCU_SPIx</code>	SPIx时钟(x=0,1)
<code>RCU_USARTx</code>	USARTx时钟(x=0,1)
<code>RCU_WWDGT</code>	WWDGT时钟
<code>RCU_I2Cx</code>	I2Cx时钟(x=0,1)
<code>RCU_PMU</code>	PMU时钟
<code>RCU_RTC</code>	RTC时钟
<code>RCU_DBGMCU</code>	DBGMCU时钟
<b>输出参数{out}</b>	
-	-
<b>返回值</b>	
-	-

例如:

```
/* enable the USART0 clock */
rcu_periph_clock_enable(RCU_USART0);
```

### 函数 `rcu_periph_clock_disable`

函数`rcu_periph_clock_disable`描述见下表:

**表 3-248. 函数 `rcu_periph_clock_disable`**

函数名称	<code>rcu_periph_clock_disable</code>
函数原形	<code>void rcu_periph_clock_disable(rcu_periph_enum periph);</code>
功能描述	禁能外设时钟
先决条件	-
被调用函数	-

输入参数{in}	
<b>periph</b>	RCU外设, 具体参考rcu_periph_enum
<i>RCU_GPIOx</i>	GPIOx时钟(x=A,B,C,F)
<i>RCU_DMA</i>	DMA时钟
<i>RCU_CRC</i>	CRC时钟
<i>RCU_CFGCMP</i>	CFGCMP时钟
<i>RCU_ADC</i>	ADC时钟
<i>RCU_TIMERx</i>	TIMERx时钟(x=0,2,5,13,14,15,16)
<i>RCU_SPIx</i>	SPIx时钟(x=0,1)
<i>RCU_USARTx</i>	USARTx时钟(x=0,1)
<i>RCU_WWDGT</i>	WWDGT时钟
<i>RCU_I2Cx</i>	I2Cx时钟(x=0,1)
<i>RCU_PMU</i>	PMU时钟
<i>RCU_RTC</i>	RTC时钟
<i>RCU_DBGMCU</i>	DBGMCU时钟
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* disable the USART0 clock */
rcu_periph_clock_disable(RCU_USART0);
```

#### 函数 **rcu\_periph\_clock\_sleep\_enable**

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

**表 3-249. 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}	
<b>periph</b>	RCU外设, 参考rcu_periph_sleep_enum
<i>RCU_FMC_SLP</i>	FMC时钟
<i>RCU_SRAM_SLP</i>	SRAM时钟
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 3-250. 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	-
<b>输入参数{in}</b>	
<i>periph</i>	RCU外设，参考rcu_periph_sleep_enum
<i>RCU_FMC_SLP</i>	FMC时钟
<i>RCU_SRAM_SLP</i>	SRAM时钟
<b>输出参数{out}</b>	
-	-
<b>返回值</b>	
-	-

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 3-251. 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	-
<b>输入参数{in}</b>	
<i>periph_reset</i>	RCU外设复位，参考rcu_periph_reset_enum
<i>RCU_GPIOxRST</i>	复位GPIOx时钟(x=A,B,C,F)
<i>RCU_CFGCMPRST</i>	复位CFGCMP时钟
<i>RCU_ADCRST</i>	复位ADC时钟
<i>RCU_TIMERxRST</i>	复位TIMERx时钟(x=0,2,5,13,14,15,16)

<i>RCU_SPIxRST</i>	复位SPIx时钟(x=0,1)
<i>RCU_USARTTxRST</i>	复位USARTTx时钟(x=0,1)
<i>RCU_WWDGTRST</i>	复位WWDGTRST时钟
<i>RCU_I2CxRST</i>	复位I2Cx时钟(x=0,1)
<i>RCU_PMURST</i>	复位PMU时钟
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 3-252. 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}	
<b>periph_reset</b>	RCU外设复位，参考rcu_periph_reset_enum
<i>RCU_GPIOxRST</i>	除能复位GPIOx时钟(x=A,B,C,F)
<i>RCU_CFGCMPRST</i>	除能复位CFGCMP时钟
<i>RCU_ADCRST</i>	除能复位ADC时钟
<i>RCU_TIMERxRST</i>	除能复位TIMERx时钟(x=0,2,5,13,14,15,16)
<i>RCU_SPIxRST</i>	除能复位SPIx时钟(x=0,1)
<i>RCU_USARTTxRST</i>	除能复位USARTTx时钟(x=0,1)
<i>RCU_WWDGTRST</i>	除能复位WWDGTRST时钟
<i>RCU_I2CxRST</i>	除能复位I2Cx时钟(x=0,1)
<i>RCU_PMURST</i>	除能复位PMU时钟
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 3-253. 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 3-254. 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 3-255. 函数 `rcu_system_clock_source_config`**

函数名称	<code>rcu_system_clock_source_config</code>
函数原形	<code>void rcu_system_clock_source_config(uint32_t ck_sys);</code>
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}	
<code>ck_sys</code>	系统时钟源选择
<code>RCU_CKSYSRC_RC8M</code>	选择CK_IRC8M时钟作为CK_SYS时钟源
<code>RCU_CKSYSRC_HXTAL</code>	选择CK_HXTAL时钟作为CK_SYS时钟源
<code>RCU_CKSYSRC_PLL</code>	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_CKSYSRC_HXTAL);
```

### 函数 `rcu_system_clock_source_get`

函数`rcu_system_clock_source_get`描述见下表:

**表 3-256. 函数 `rcu_system_clock_source_get`**

函数名称	<code>rcu_system_clock_source_get</code>
函数原形	<code>uint32_t rcu_system_clock_source_get(void);</code>
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	
<code>uint32_t</code>	<code>RCU_SCSS_IRC8M/RCU_SCSS_HXTAL/RCU_SCSS_PLL</code>

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 3-257. 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 3-258. 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_CKAH_B_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 3-259. 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
<b>RCU_APB2_CKAH_B_DIVx</b>	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 3-260. 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rcu_adc_clock_enum
<i>RCU_ADCCK_IRC2_8M_DIV2</i>	选择 (IRC28M / 2) 作为CK_ADC时钟
<i>RCU_ADCCK_IRC2_8M</i>	选择内部28M RC振荡器时钟作为CK_ADC时钟
<i>RCU_ADCCK_AHB_DIVx</i>	选择AHB时钟的x分频作为CK_ADC时钟 (x=3,5,7,9)
<i>RCU_ADCCK_APB2_DIVx</i>	选择APB2时钟的x分频作为CK_ADC时钟 (x=2,4,6,8)
<b>输出参数{out}</b>	
-	-
<b>返回值</b>	
-	-

例如：

```
/* configure the ADC prescaler factor */

rcu_adc_clock_config(RCU_ADCCK_IRC28M);
```

### 函数 **rcu\_ckout\_config**

函数rcu\_ckout\_config描述见下表：

**表 3-261. 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	-
<b>输入参数{in}</b>	
<b>ckout_src</b>	CKOUT时钟源选择
<i>RCU_CKOUTSRC_NONE</i>	无时钟输出
<i>RCU_CKOUTSRC_IRC28M</i>	选择内部28M RC振荡器时钟
<i>RCU_CKOUTSRC_IRC40K</i>	选择内部40K RC振荡器时钟
<i>RCU_CKOUTSRC_LXTAL</i>	选择外部低速晶体振荡器时钟 (LXTAL)
<i>RCU_CKOUTSRC_CKSYS</i>	选择系统时钟CK_SYS
<i>RCU_CKOUTSRC_IRC8M</i>	选择内部8M RC振荡器时钟
<i>RCU_CKOUTSRC_HXTAL</i>	选择外部高速晶体振荡器时钟 (HXTAL)

<i>HXTAL</i>	
<i>RCU_CKOUTSRC_CKPLL_DIV1</i>	选择CK_PLL时钟
<i>RCU_CKOUTSRC_CKPLL_DIV2</i>	选择(CK_PLL / 2)时钟
<b>输入参数{in}</b>	
<i>ckout_div</i>	CKOUT分频系数
<i>RCU_CKOUT_DIVx</i>	将CKOUT所选时钟x分频(x=1,2,4,8,16,32,64,128)
<b>输出参数{out}</b>	
-	-
<b>返回值</b>	
-	-

例如：

```
/* configure the HXTAL as CK_OUT clock source */
rcu_ckout_config(RCU_CKOUTSRC_HXTAL, RCU_CKOUT_DIV1);
```

### 函数 **rcu\_pll\_config**

函数rcu\_pll\_config描述见下表：

**表 3-262. 函数 **rcu\_pll\_config****

函数名称	<b>rcu_pll_config</b>
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
先决条件	-
被调用函数	-
<b>输入参数{in}</b>	
<i>pll_src</i>	PLL时钟源选择
<i>RCU_PLLSRC_IRC8M_DIV2</i>	(IRC8M / 2)被选择为PLL时钟的时钟源
<i>RCU_PLLSRC_HXTAL</i>	HXTAL时钟被选择为PLL时钟的时钟源
<b>输入参数{in}</b>	
<i>pll_mul</i>	PLL时钟倍频因子
<i>RCU_PLL_MULx</i>	PLL源时钟 * x (x = 2..32)
<b>输出参数{out}</b>	
-	-
<b>返回值</b>	
-	-

例如：

```
/* configure the PLL */
```

---

```
rcu_pll_config(RCU_PLLSRC_HXTAL, RCU_PLL_MUL10);
```

### 函数 `rcu_usart_clock_config`

函数`rcu_usart_clock_config`描述见下表：

**表 3-263. 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	-
<b>输入参数{in}</b>	
<code>ck_usart</code>	USART0输入时钟源
<code>RCU_USART0SRC_CKAPB2</code>	选择CK_APB2时钟作为CK_USART0时钟
<code>RCU_USART0SRC_CKSYS</code>	选择CK_SYS时钟作为CK_USART0时钟
<code>RCU_USART0SRC_LXTAL</code>	选择CK_LXTAL时钟作为CK_USART0时钟
<code>RCU_USART0SRC_IRC8M</code>	选择CK_IRC8M时钟作为CK_USART0时钟
<b>输出参数{out}</b>	
-	-
<b>返回值</b>	
-	-

例如：

```
/* configure the USART */
rcu_usart_clock_config(RCU_USART0SRC_CKAPB2);
```

### 函数 `rcu_rtc_clock_config`

函数`rcu_rtc_clock_config`描述见下表：

**表 3-264. 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	-
<b>输入参数{in}</b>	
<code>rtc_clock_source</code>	RTC时钟源选择

<i>RCU_RTC_SRC_NO_NE</i>	未选择时钟
<i>RCU_RTC_SRC_LX_TAL</i>	选择CK_LXTAL作为RTC时钟源
<i>RCU_RTC_SRC_IRC_40K</i>	选择内部40K RC振荡器时钟作为RTC时钟源
<i>RCU_RTC_SRC_HXTAL_DIV32</i>	选择外部高速晶振32分频作为RTC时钟源
<b>输出参数{out}</b>	
-	-
<b>返回值</b>	
-	-

例如：

```
/* configure the RTC clock source selection */
rcu_rtc_clock_config(RCU_RTC_SRC_IRC40K);
```

#### 函数 **rcu\_hxtal\_prediv\_config**

函数rcu\_hxtal\_prediv\_config描述见下表：

**表 3-265. 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	-
<b>输入参数{in}</b>	
<b>hxtal_prediv</b>	PLL时钟源分频因子选择
<i>RCU_PLL_PREDIVx</i>	HXTAL的x分频作为PLL时钟 ( $x=1..16$ )
<b>输出参数{out}</b>	
-	-
<b>返回值</b>	
-	-

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 3-266. 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}	
<i>lxtal_dricap</i>	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 3-267. 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}	
<i>flag</i>	时钟稳定和外设复位标志，参考rcu_flag_enum
<i>RCU_FLAG_IRC40_KSTB</i>	IRC40K稳定标志
<i>RCU_FLAG_LXTAL_STB</i>	LXTAL稳定标志
<i>RCU_FLAG_IRC8M_STB</i>	IRC8M稳定标志

<i>RCU_FLAG_HXTAL_STB</i>	HXTAL稳定标志
<i>RCU_FLAG_PLLST_B</i>	PLL时钟稳定标志
<i>RCU_FLAG_IRC28_MSTB</i>	IRC28M稳定标志
<i>RCU_FLAG_V12RS_T</i>	1.2V电压域复位标志
<i>RCU_FLAG_OBLR_ST</i>	选项字节复位标志
<i>RCU_FLAG_EPRS_T</i>	外部引脚复位标志
<i>RCU_FLAG_PORR_ST</i>	电源复位标志
<i>RCU_FLAG_SWRS_T</i>	软件复位标志
<i>RCU_FLAG_FWDG_TRST</i>	独立看门狗复位标志
<i>RCU_FLAG_WWD_GTRST</i>	窗口看门狗复位标志
<i>RCU_FLAG_LPRST</i>	低电压复位标志
<b>输出参数{out}</b>	
-	-
<b>返回值</b>	
-	SET或RESET

例如：

```
/* get the clock stabilization flag */

if(RESET != rcu_flag_get(RCU_FLAG_LXTALSTB)){
}
```

#### 函数 **rcu\_all\_reset\_flag\_clear**

函数**rcu\_all\_reset\_flag\_clear**描述见下表：

**表 3-268. 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	-
<b>输入参数{in}</b>	
-	-

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 3-269. 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}	
<b>Int_flag</b>	中断以及CKM标志，参考rcu_int_flag_enum
<b>RCU_INT_FLAG_IR_C40KSTB</b>	IRC40K稳定中断标志
<b>RCU_INT_FLAG_L_XTALSTB</b>	LXTAL稳定中断标志
<b>RCU_INT_FLAG_IR_C8MSTB</b>	IRC8M稳定中断标志
<b>RCU_INT_FLAG_H_XTALSTB</b>	HXTAL稳定中断标志
<b>RCU_INT_FLAG_P_LLSTB</b>	PLL稳定中断标志
<b>RCU_INT_FLAG_IR_C28MSTB</b>	IRC28M稳定中断标志
<b>RCU_INT_FLAG_C_KM</b>	HXTAL时钟阻塞中断标志
输出参数{out}	
-	-
返回值	
<b>FlagStatus</b>	SET 或 RESET

例如：

```
/* get the clock stabilization interrupt flag */
if(SET == rcu_interrupt_flag_get(RCU_INT_FLAG_HXTALSTB)){
```

}

### 函数 **rcu\_interrupt\_flag\_clear**

函数rcu\_interrupt\_flag\_clear描述见下表：

**表 3-270. 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	-
<b>输入参数{in}</b>	
int_flag_clear	时钟稳定和阻塞中断标志清除，参考rcu_int_flag_clear_enum
RCU_INT_FLAG_IR_C40KSTB_CLR	清除IRC40K稳定中断标志
RCU_INT_FLAG_L_XTALSTB_CLR	清除LXTAL稳定中断标志
RCU_INT_FLAG_IR_C8MSTB_CLR	清除IRC8M稳定中断标志
RCU_INT_FLAG_H_XTALSTB_CLR	清除HXTAL稳定中断标志
RCU_INT_FLAG_P_LLSTB_CLR	清除PLL稳定中断标志
RCU_INT_FLAG_IR_C28MSTB_CLR	清除IRC28M稳定中断标志
RCU_INT_FLAG_C_KM_CLR	清除HXTAL时钟阻塞中断标志
<b>输出参数{out}</b>	
-	-
<b>返回值</b>	
-	-

例如：

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

### 函数 **rcu\_interrupt\_enable**

函数rcu\_interrupt\_enable描述见下表：

**表 3-271. 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);

<b>功能描述</b>	使能时钟稳定中断
<b>先决条件</b>	-
<b>被调用函数</b>	-
<b>输入参数{in}</b>	
<b>stb_int</b>	时钟稳定中断, 具体参考rcu_int_enum
<i>RCU_INT_IRC40KS</i> <i>TB</i>	使能IRC40K稳定中断
<i>RCU_INT_LXTALS</i> <i>TB</i>	使能LXTAL稳定中断
<i>RCU_INT_IRC8MS</i> <i>TB</i>	使能IRC8M稳定中断
<i>RCU_INT_HXTALS</i> <i>TB</i>	使能HXTAL稳定中断
<i>RCU_INT_PLLSTB</i>	使能PLL稳定中断
<i>RCU_INT_IRC28M</i> <i>STB</i>	使能IRC28M稳定中断
<b>输出参数{out}</b>	
-	-
<b>返回值</b>	
-	-

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 3-272. 函数 **rcu\_interrupt\_disable****

<b>函数名称</b>	rcu_interrupt_disable
<b>函数原形</b>	void rcu_interrupt_disable (rcu_int_enum stab_int);
<b>功能描述</b>	除能时钟稳定中断
<b>先决条件</b>	-
<b>被调用函数</b>	-
<b>输入参数{in}</b>	
<b>stb_int</b>	时钟稳定中断, 具体参考rcu_int_enum
<i>RCU_INT_IRC40KS</i> <i>TB</i>	除能IRC40K稳定中断
<i>RCU_INT_LXTALS</i> <i>TB</i>	除能LXTAL稳定中断
<i>RCU_INT_IRC8MS</i> <i>TB</i>	除能IRC8M稳定中断

<i>RCU_INT_HXTALS</i>	除能HXTAL稳定中断
<i>RCU_INT_PLLSTB</i>	除能PLL稳定中断
<i>RCU_INT_IRC28M</i>	除能IRC28M稳定中断
<b>输出参数{out}</b>	
-	-
<b>返回值</b>	
-	-

例如：

```
/* disable the HXTAL stabilization interrupt */

rcu_interrupt_disable(RCU_INT_HXTALSTB);
```

### 函数 **rcu\_osc\_stab\_wait**

函数rcu\_osc\_stab\_wait描述见下表：

**表 3-273. 函数 **rcu\_osc\_stab\_wait****

函数名称	rcu_osc_stab_wait
函数原形	ErrStatus rcu_osc_stab_wait(rcu_osc_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
<b>输入参数{in}</b>	
<b>osci</b>	振荡器类型，参考rcu_osc_type_enum
<i>RCU_HXTAL</i>	高速晶体振荡器
<i>RCU_LXTAL</i>	低速晶体振荡器
<i>RCU_IRC8M</i>	内部8M RC振荡器
<i>RCU_IRC28M</i>	内部28M RC振荡器
<i>RCU_IRC40K</i>	内部40K RC振荡器
<i>RCU_PLL_CK</i>	锁相环
<b>输出参数{out}</b>	
-	-
<b>返回值</b>	
<b>ErrStatus</b>	SUCCESS 或 ERROR

例如：

```
/* wait for oscillator stabilization flag */

if(SUCCESS == rcu_osc_stab_wait(RCU_HXTAL)){
}
```

### 函数 **rcu\_oscி\_on**

函数rcu\_oscி\_on描述见下表:

**表 3-274. 函数 **rcu\_oscி\_on****

函数名称	rcu_oscி_on
函数原形	void rcu_oscி_on(rcu_oscி_type_enum oscி);
功能描述	打开振荡器
先决条件	-
被调用函数	-
<b>输入参数{in}</b>	
<b>oscி</b>	振荡器类型, 参考rcu_oscி_type_enum
<i>RCU_HXTAL</i>	高速晶体振荡器
<i>RCU_LXTAL</i>	低速晶体振荡器
<i>RCU_IRC8M</i>	内部8M RC振荡器
<i>RCU_IRC28M</i>	内部28M RC振荡器
<i>RCU_IRC40K</i>	内部40K RC振荡器
<i>RCU_PLL_CK</i>	锁相环
<b>输出参数{out}</b>	
-	-
<b>返回值</b>	
-	-

例如:

```
/* turn on the high speed crystal oscillator */
rcu_oscி_on(RCU_HXTAL);
```

### 函数 **rcu\_oscி\_off**

函数rcu\_oscி\_off描述见下表:

**表 3-275. 函数 **rcu\_oscி\_off****

函数名称	rcu_oscி_off
函数原形	void rcu_oscி_off(rcu_oscி_type_enum oscி);
功能描述	关闭振荡器
先决条件	-
被调用函数	-
<b>输入参数{in}</b>	
<b>oscி</b>	振荡器类型, 参考rcu_oscி_type_enum
<i>RCU_HXTAL</i>	高速晶体振荡器
<i>RCU_LXTAL</i>	低速晶体振荡器
<i>RCU_IRC8M</i>	内部8M RC振荡器
<i>RCU_IRC28M</i>	内部28M RC振荡器
<i>RCU_IRC40K</i>	内部40K RC振荡器

<i>RCU_PLL_CK</i>	锁相环
<b>输出参数{out}</b>	
-	-
<b>返回值</b>	
-	-

例如：

```
/* turn off the high speed crystal oscillator */

rcu_osc_i_off(RCU_HXTAL);
```

#### 函数 **rcu\_osc\_i\_bypass\_mode\_enable**

函数rcu\_osc\_i\_bypass\_mode\_enable描述见下表：

**表 3-276. 函数 rcu\_osc\_i\_bypass\_mode\_enable**

函数名称	rcu_osc_i_bypass_mode_enable
函数原形	void rcu_osc_i_bypass_mode_enable(rcu_osc_i_type_enum osci);
功能描述	使能振荡器时钟旁路模式
先决条件	HXTALEN或LXTALEN应在使能振荡器时钟旁路模式前先复位
被调用函数	-
<b>输入参数{in}</b>	
<b>osci</b>	振荡器类型，参考rcu_osc_i_type_enum
<i>RCU_HXTAL</i>	高速晶体振荡器
<i>RCU_LXTAL</i>	低速晶体振荡器
<b>输出参数{out}</b>	
-	-
<b>返回值</b>	
-	-

例如：

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

rcu_osc_i_bypass_mode_enable(RCU_HXTAL);
```

#### 函数 **rcu\_osc\_i\_bypass\_mode\_disable**

函数rcu\_osc\_i\_bypass\_mode\_disable描述见下表：

**表 3-277. 函数 rcu\_osc\_i\_bypass\_mode\_disable**

函数名称	rcu_osc_i_bypass_mode_disable
函数原形	void rcu_osc_i_bypass_mode_disable(rcu_osc_i_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}	
<b>osci</b>	振荡器类型, 参考rcu_osc_type_enum
<i>RCU_HXTAL</i>	高速晶体振荡器
<i>RCU_LXTAL</i>	低速晶体振荡器
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 3-278. 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 3-279. 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	-

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

例如：

```
/* 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 3-280. 函数 rcu\_irc8m\_adjust\_value\_set**

<b>函数名称</b>	rcu_irc8m_adjust_value_set
<b>函数原形</b>	void rcu_irc8m_adjust_value_set(uint32_t irc8m_adjval);
<b>功能描述</b>	设置内部8MHz RC振荡器时钟调整值
<b>先决条件</b>	-
<b>被调用函数</b>	-
<b>输入参数{in}</b>	
<b>irc8m_adjval</b>	IRC8M调整值（0到0x1F之间）
<b>输出参数{out}</b>	
-	-
<b>返回值</b>	
-	-

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 3-281. 函数 rcu\_irc28m\_adjust\_value\_set**

<b>函数名称</b>	rcu_irc28m_adjust_value_set
<b>函数原形</b>	void rcu_irc28m_adjust_value_set(uint32_t irc28m_adjval);
<b>功能描述</b>	设置内部28MHz RC振荡器时钟调整值
<b>先决条件</b>	-
<b>被调用函数</b>	-

输入参数{in}	
irc28m_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 3-282. 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 3-283. 函数 rcu\_deepsleep\_voltage\_set**

函数名称	rcu_deepsleep_voltage_set
函数原形	void rcu_deepsleep_voltage_set(uint32_t dsvol);
功能描述	设置深度睡眠模式电压值
先决条件	-
被调用函数	-
输入参数{in}	

<b>dsvol</b>	深度睡眠模式电压值
<i>RCU_DEEPSLEEP_V_1_0</i>	在深度睡眠模式下内核电压为1.0V
<i>RCU_DEEPSLEEP_V_0_9</i>	在深度睡眠模式下内核电压为0.9V
<i>RCU_DEEPSLEEP_V_0_8</i>	在深度睡眠模式下内核电压为0.8V
<i>RCU_DEEPSLEEP_V_1_2</i>	在深度睡眠模式下内核电压为1.2V
<b>输出参数{out}</b>	
-	-
<b>返回值</b>	
-	-

例如：

```
/* set the deep-sleep mode voltage */
rcu_deepsleep_voltage_set(RCU_DEEPSLEEP_V_1_0);
```

#### 函数 **rcu\_clock\_freq\_get**

函数rcu\_clock\_freq\_get描述见下表：

**表 3-284. 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	-
<b>输入参数{in}</b>	
<b>clock</b>	要获取的时钟频率，具体参考rcu_clock_freq_enum
<b>CK_SYS</b>	系统时钟频率
<b>CK_AHB</b>	AHB时钟频率
<b>CK_APB1</b>	APB1时钟频率
<b>CK_APB2</b>	APB2时钟频率
<b>CK_ADC</b>	ADC时钟频率
<b>CK_USART</b>	USART时钟频率
<b>输出参数{out}</b>	
-	-
<b>返回值</b>	
<b>uint32_t</b>	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);

```

## 3.15. RTC

实时时钟RTC通常被用作时钟日历。位于备份域中的RTC电路，包含一个32位的累加计数器、一个闹钟、一个预分频器、一个分频器以及RTC时钟配置寄存器。章节[3.15.1](#)描述了RTC的寄存器列表，章节[3.15.2](#)对RTC库函数进行说明。

### 3.15.1. 外设寄存器描述

RTC寄存器列表如下表所示：

**表 3-285. RTC 寄存器**

寄存器名称	寄存器描述
RTC_TIME	RTC时间寄存器
RTC_DATE	RTC日期寄存器
RTC_CTL	RTC控制寄存器
RTC_STAT	RTC状态寄存器
RTC_PSC	RTC预分频寄存器
RTC_ALRM0TD	RTC闹钟0时间日期寄存器
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

### 3.15.2. 外设库函数描述

RTC库函数列表如下表所示：

**表 3-286. 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 3-287. 结构体 rtc\_parameter\_struct**

Member name	Function description
rtc_year	RTC年份值: 0x0 - 0x99(BCD 格式)
rtc_month	RTC月份值(BCD 格式)

rtc_date	RTC日期值: 0x1 - 0x31(BCD 格式)
rtc_day_of_week	RTC星期值(BCD 格式)
rtc_hour	RTC 小时值: 0x1 - 0x12(BCD 格式) or 0x0 - 0x23(BCD 格式)
rtc_minute	RTC分钟值: 0x0 - 0x59(BCD 格式)
rtc_second	RTC秒值: 0x0 - 0x59(BCD 格式)
rtc_factor_asyn	RTC一步分频值: 0x0 - 0x7F
rtc_factor_syn	RTC同步分频值: 0x0 - 0x7FFF
rtc_am_pm	RTC AM/PM 值
rtc_display_format	RTC时间格式

### 结构体 `rtc_alarm_struct`

表 3-288. 结构体 `rtc_alarm_struct`

Member name	Function description
rtc_alarm_mask	RTC闹钟屏蔽
rtc_weekday_or_dat e	指定RTC闹钟是日期还是星期几
rtc_alarm_day	RTC闹钟日期或者星期几的值(BCD 格式)
rtc_alarm_hour	RTC闹钟小时值: 0x1 - 0x12(BCD 格式) or 0x0 - 0x23(BCD 格式)
rtc_alarm_minute	RTC闹钟分钟值: 0x0 - 0x59(BCD 格式)
rtc_alarm_second	RTC闹钟秒数值: 0x0 - 0x59(BCD 格式)
rtc_am_pm	RTC闹钟AM/PM数值

### 结构体 `rtc_timestamp_struct`

表 3-289. 结构体 `rtc_timestamp_struct`

Member name	Function description
rtc_timestamp_mont h	RTC时间戳月份值
rtc_timestamp_date	RTC 时间戳日期值: 0x1 - 0x31(BCD 格式)
rtc_timestamp_day	RTC时间戳星期值(BCD 格式)
rtc_timestamp_hour	RTC 时间戳小时值: 0x1 - 0x12(BCD 格式) or 0x0 - 0x23(BCD 格式)
rtc_timestamp_minu te	RTC时间戳分钟值: 0x0 - 0x59(BCD 格式)
rtc_timestamp_seco nd	RTC时间戳秒数值: 0x0 - 0x59(BCD 格式)
rtc_am_pm	RTC时间戳AM/PM数值

### 结构体 `rtc_tamper_struct`

表 3-290. 结构体 `rtc_tamper_struct`

Member name	Function description

<code>rtc_tamper_source</code>	RTC侵入检测源
<code>rtc_tamper_trigger</code>	RTC侵入事件检测触发沿
<code>rtc_tamper_filter</code>	RTC 侵入事件检测在电平检测期间需要的连续采样次数
<code>rtc_tamper_sample_frequency</code>	RTC侵入事件电平模式检测的采样频率
<code>rtc_tamper_preamble_enable</code>	RTC在电压电平检测期间的预充电功能
<code>rtc_tamper_preamble_time</code>	RTC侵入事件电平检测采样预充电时间，如果预充电功能使能
<code>rtc_tamper_with_timestamp</code>	RTC侵入事件触发时间戳

### 函数 `rtc_deinit`

函数`rtc_deinit`描述见下表：

表 3-291. 函数 `rtc_deinit`

函数名称	<code>rtc_deinit</code>
函数原型	<code>ErrStatus rtc_deinit(void);</code>
功能描述	复位大多数RTC寄存器
先决条件	-
被调用函数	<code>rcu_periph_reset_enable/ rcu_periph_reset_disable</code>
输入参数{in}	
-	-
输出参数{out}	
-	-
返回值	
<code>ErrStatus</code>	ERROR或SUCCESS

例如：

```
/* reset most of the RTC registers*/
```

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

### 函数 `rtc_init`

函数`rtc_init`描述见下表：

表 3-292. 函数 `rtc_init`

函数名称	<code>rtc_init</code>
函数原型	<code>ErrStatus rtc_init(rtc_parameter_struct* rtc_initpara_struct);</code>
功能描述	初始化RTC寄存器
先决条件	-
被调用函数	-

输入参数{in}	
<b>rtc_initpara_struct</b>	初始化结构体，结构体成员参考 <a href="#">表3-287. 结构体 rtc_parameter_struct</a>
输出参数{out}	
-	-
返回值	
<b>ErrStatus</b>	ERROR或SUCCESS

例如：

```
/* reset most of the RTC registers*/
ErrStatus error_status = rtc_init();
```

### 函数 **rtc\_init\_mode\_enter**

函数**rtc\_init\_mode\_enter**描述见下表：

**表 3-293. 函数 rtc\_init\_mode\_enter**

函数名称	rtc_init_mode_enter
函数原型	ErrStatus rtc_init_mode_enter(void);
功能描述	进入RTC初始化模式
先决条件	-
被调用函数	-
输入参数{in}	
-	-
输出参数{out}	
-	-
返回值	
<b>ErrStatus</b>	ERROR或 SUCCESS

例如：

```
/*enter RTC init mode*/
ErrStatus error_status = rtc_init_mode_enter();
```

### 函数 **rtc\_init\_mode\_exit**

函数**rtc\_init\_mode\_exit**描述见下表：

**表 3-294. 函数 rtc\_init\_mode\_exit**

函数名称	rtc_init_mode_exit
函数原型	void rtc_init_mode_exit(void);
功能描述	退出RTC初始化模式
先决条件	-
被调用函数	-

输入参数{in}	
-	-
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/*exit RTC init mode*/
rtc_init_mode_exit();
```

### 函数 **rtc\_register\_sync\_wait**

函数**rtc\_register\_sync\_wait**描述见下表：

**表 3-295. 函数 **rtc\_register\_sync\_wait****

函数名称	rtc_register_sync_wait
函数原型	ErrStatus rtc_register_sync_wait(void);
功能描述	等待直到RTC_TIME和RTC_DATE寄存器与APB时钟同步，并且阴影寄存器被更新
先决条件	-
被调用函数	-
输入参数{in}	
-	-
输入参数{in}	
-	-
返回值	
ErrStatus	ERROR或SUCCESS

例如：

```
/*wait until RTC_TIME and RTC_DATE registers are synchronized with APB clock, and the
shadow registers are updated*/
ErrStatus error_status = rtc_register_sync_wait();
```

### 函数 **rtc\_current\_time\_get**

函数**rtc\_current\_time\_get**描述见下表：

**表 3-296. 函数 **rtc\_current\_time\_get****

函数名称	rtc_current_time_get
函数原型	void rtc_current_time_get(rtc_parameter_struct* rtc_initpara_struct);
功能描述	获取当前的时间和日期

先决条件	-
被调用函数	-
输入参数{in}	
-	-
输出参数{out}	
<b>rtc_initpara_struct</b>	初始化结构体, 结构体成员参考 <a href="#">表3-287. 结构体 rtc_parameter_struct</a>
返回值	
-	-

例如:

```
/*get current time and date*/
rtc_parameter_struct rtc_initpara_struct;
rtc_current_time_get (&rtc_initpara_struct);
```

### 函数 **rtc\_subsecond\_get**

函数**rtc\_subsecond\_get**描述见下表:

**表 3-297. 函数 rtc\_subsecond\_get**

函数名称	rtc_subsecond_get
函数原型	uint32_t rtc_subsecond_get(void);
功能描述	获取当前的亚秒值
先决条件	-
被调用函数	-
输入参数{in}	
-	-
输出参数{out}	
-	-
返回值	
<b>uint32_t</b>	当前的亚秒值(0x00-0xFFFF)

例如:

```
/*get current subsecond value*/
uint32_t sub_second = rtc_subsecond_get();
```

### 函数 **rtc\_alarm\_config**

函数**rtc\_alarm\_config**描述见下表:

**表 3-298. 函数 rtc\_alarm\_config**

函数名称	rtc_alarm_config
函数原型	void rtc_alarm_config(rtc_alarm_struct* rtc_alarm_time);

功能描述	配置RTC闹钟
先决条件	-
被调用函数	-
<b>输入参数{in}</b>	
rtc_alarm_time	闹钟结构体, 结构体成员参考 <a href="#">表3-288. 结构体rtc_alarm_struct</a>
<b>输出参数{out}</b>	
-	-
<b>返回值</b>	
-	-

例如:

```
/*rtc_alarm_config*/
rtc_alarm_struct rtc_alarm_time;
rtc_alarm_config (&rtc_alarm_time);
```

### 函数 **rtc\_alarm\_subsecond\_config**

函数`rtc_alarm_subsecond_config`描述见下表:

**表 3-299. 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	-
被调用函数	-
<b>输入参数{in}</b>	
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
RTC_MASKSSC_11_14	屏蔽RTC_ALRM0SS_SSC[14:11], SSC[10:0]位用于时间匹配

<i>RTC_MASKSSC_12_4</i>	屏蔽RTC_ALRM0SS_SSC[14:12], SSC[11:0]位用于时间匹配
<i>RTC_MASKSSC_13_4</i>	屏蔽RTC_ALRM0SS_SSC[14:13], SSC[12:0]位用于时间匹配
<i>RTC_MASKSSC_14</i>	屏蔽RTC_ALRM0SS_SSC[14], SSC[13:0]位用于时间匹配
<i>RTC_MASKSSC_NON_E</i>	无屏蔽, SSC[14:0]位用于时间匹配
<b>输入参数{in}</b>	
<b>subsecond</b>	闹钟亚秒值(0x000 - 0x7FFF)
<b>输出参数{out}</b>	
-	-
<b>返回值</b>	
-	-

例如:

```
/*configure subsecond of RTC alarm*/
rtc_subsecond_config(RTC_MASKSSC_9_14, 0x7FFF);
```

### 函数 **rtc\_alarm\_enable**

函数**rtc\_alarm\_enable**描述见下表:

**表 3-300. 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	-
被调用函数	-
<b>输入参数{in}</b>	
-	-
<b>输出参数{out}</b>	
-	-
<b>返回值</b>	
-	-

例如:

```
/*enable RTC alarm*/
rtc_alarm_enable();
```

### 函数 **rtc\_alarm\_disable**

函数**rtc\_alarm\_disable**描述见下表:

表 3-301. 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	-
被调用函数	-
输入参数{in}	
-	-
输出参数{out}	
-	-
返回值	
ErrStatus	ERROR或SUCCESS

例如：

```
/*disable RTC alarm*/
ErrStatus error_status = rtc_alarm_disable();
```

### 函数 `rtc_alarm_get`

函数`rtc_alarm_get`描述见下表：

表 3-302. 函数 `rtc_alarm_get`

函数名称	rtc_alarm_get
函数原型	void rtc_alarm_get(rtc_alarm_struct* rtc_alarm_time);
功能描述	获取RTC闹钟
先决条件	-
被调用函数	-
输入参数{in}	
-	-
输出参数{out}	
<code>rtc_alarm_time</code>	闹钟结构体，结构体成员参考 <a href="#">表3-288. 结构体<code>rtc_alarm_struct</code></a>
返回值	
-	-

例如：

```
/*disable RTC alarm*/
rtc_alarm_struct rtc_alarm_time;
rtc_alarm_get (&rtc_alarm_time);
```

### 函数 rtc\_alarm\_subsecond\_get

函数rtc\_alarm\_subsecond\_get描述见下表:

表 3-303. 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	-
被调用函数	-
输入参数{in}	
-	-
输出参数{out}	
-	-
返回值	
uint32_t	RTC 闹钟亚秒值(0x0-0x3FFF)

例如:

```
/*get RTC alarm subsecond*/
uint32_t subsecond =  rtc_alarm_subsecond_get();
```

### 函数 rtc\_timestamp\_enable

函数can\_init描述见下表:

表 3-304. 函数 rtc\_timestamp\_enable

函数名称	rtc_timestamp_enable
函数原型	void rtc_timestamp_enable(uint32_t edge);
功能描述	使能RTC时间戳
先决条件	-
被调用函数	-
输入参数{in}	
edge	选定哪种边沿触发时间戳检测
RTC_TIMESTAMP_RISING_EDGE	上升沿是时间戳事件有效检测沿
RTC_TIMESTAMP_FALLING_EDGE	下降沿是时间戳事件有效检测沿
输出参数{out}	
-	-
返回值	
-	-

例如:

---

```
/*enable RTC time-stamp*/
rtc_timestamp_enable (RTC_TIMESTAMP_RISING_EDGE);
```

### 函数 **rtc\_timestamp\_disable**

函数`rtc_timestamp_disable`描述见下表：

**表 3-305. 函数 `rtc_timestamp_disable`**

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

例如：

```
/*disable RTC time-stamp*/
rtc_timestamp_disable();
```

### 函数 **rtc\_timestamp\_get**

函数`rtc_timestamp_get`描述见下表：

**表 3-306. 函数 `rtc_timestamp_get`**

函数名称	rtc_timestamp_get
函数原型	void rtc_timestamp_get(rtc_timestamp_struct* rtc_timestamp);
功能描述	获取RTC时间戳时间和日期
先决条件	-
被调用函数	-
输入参数{in}	
-	-
输出参数{out}	
rtc_timestamp	时间戳结构体，结构体成员参考 <a href="#">表3-289. 结构体 <code>rtc_timestamp_struct</code></a>
返回值	
-	-

例如：

```
/* get RTC timestamp time and date */

rtc_timestamp_struct rtc_timestamp;

rtc_timestamp_get(& rtc_timestamp);
```

### **函数 `rtc_timestamp_subsecond_get`**

函数`rtc_timestamp_subsecond_get`描述见下表：

**表 3-307. 函数 `rtc_timestamp_subsecond_get`**

函数名称	rtc_timestamp_subsecond_get
函数原型	uint32_t rtc_timestamp_subsecond_get(void);
功能描述	获取RTC时间戳亚秒值
先决条件	-
被调用函数	-
输入参数{in}	
-	-
输出参数{out}	
-	-
返回值	
uint32_t	RTC时间戳亚秒值

例如：

```
/* get RTC time-stamp subsecond */

uint32_t subsecond = rtc_timestamp_subsecond_get();
```

### **函数 `rtc_tamper_enable`**

函数`rtc_tamper_enable`描述见下表：

**表 3-308. 函数 `rtc_timestamp_enable`**

函数名称	rtc_tamper_enable
函数原型	void rtc_tamper_enable(rtc_tamper_struct* rtc_tamper);
功能描述	使能RTC侵入检测
先决条件	-
被调用函数	-
输入参数{in}	
rtc_tamper	tamper化结构体，结构体成员参考 <a href="#">表3-290. 结构体<code>rtc_tamper_struct</code></a>
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* enable RTC tamper */

rtc_tamper_struct rtc_tamper

rtc_tamper_enable(& rtc_tamper);
```

### **函数 `rtc_tamper_disable`**

函数`rtc_tamper_disable`描述见下表：

**表 3-309. 函数 `rtc_tamper_disable`**

函数名称	rtc_tamper_disable
函数原型	void rtc_tamper_disable(uint32_t source);
功能描述	失能RTC侵入检测
先决条件	-
被调用函数	-
<b>输入参数{in}</b>	
<b>source</b>	选定被失能的侵入检测来源
<i>RTC_TAMPER0</i>	RTC tamper0
<i>RTC_TAMPER1</i>	RTC tamper1
<b>输出参数{out}</b>	
-	-
<b>返回值</b>	
-	-

例如：

```
/* disable RTC tamper */

rtc_tamper_disable(RTC_TAMPER0);
```

### **函数 `rtc_interrupt_enable`**

函数`rtc_interrupt_enable`描述见下表：

**表 3-310. 函数 `rtc_interrupt_enable`**

函数名称	rtc_interrupt_enable
函数原型	void rtc_interrupt_enable(uint32_t interrupt);
功能描述	使能RTC指定的中断
先决条件	-
被调用函数	-
<b>输入参数{in}</b>	
<b>interrupt</b>	选定被使能的中断源
<i>RTC_INT_TIMESTAMP</i>	时间戳中断

<i>RTC_INT_ALARM</i>	闹钟中断
<i>RTC_INT_TAMP</i>	侵入检测中断
<b>输出参数{out}</b>	
-	-
<b>返回值</b>	
-	-

例如：

```
/* enable specified RTC interrupt*/
rtc_interrupt_enable(RTC_INT_TAMP);
```

### 函数 **rtc\_interrupt\_disable**

函数`rtc_interrupt_disable`描述见下表：

**表 3-311. 函数 `rtc_interrupt_disable`**

函数名称	<code>rtc_interrupt_disable</code>
函数原型	<code>void rtc_interrupt_disable(uint32_t interrupt);</code>
功能描述	失能RTC指定中断
先决条件	-
被调用函数	-
<b>输入参数{in}</b>	
<code>interrupt</code>	选定被失能的RTC中断
<i>RTC_INT_TIMESTAMP</i>	时间戳中断
<i>RTC_INT_ALARM</i>	闹钟中断
<i>RTC_INT_TAMP</i>	侵入检测中断
<b>输出参数{out}</b>	
-	-
<b>返回值</b>	
-	-

例如：

```
/* disable RTC ALARM interrupt */

rtc_interrupt_disable(RTC_INT_TAMP);
```

### 函数 **rtc\_flag\_get**

函数`rtc_flag_get`描述见下表：

**表 3-312. 函数 `rtc_flag_get`**

函数名称	<code>rtc_flag_get</code>
函数原型	<code>FlagStatus rtc_flag_get(uint32_t flag);</code>

功能描述	获取指定中断标志位
先决条件	-
被调用函数	-
<b>输入参数{in}</b>	
<b>flag</b>	选定被获取的中断标志
<i>RTC_FLAG_RECALIBRATION</i>	平滑校准挂起标志
<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_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
<b>输出参数{out}</b>	
-	-
<b>返回值</b>	
FlagStatus	SET 或 RESET

例如：

```
/* check time-stamp event flag */
FlagStatus = rtc_flag_get(RTC_FLAG_TIMESTAMP)
```

### 函数 `rtc_flag_clear`

函数`rtc_flag_clear`描述见下表：

表 3-313. 函数 `rtc_flag_clear`

函数名称	rtc_flag_clear
函数原型	void rtc_flag_clear(uint32_t flag);
功能描述	清除指定中断标志位
先决条件	-
被调用函数	-
<b>输入参数{in}</b>	
<b>flag</b>	要清除的中断标志位
<i>RTC_FLAG_TAMP1</i>	tamper 1事件标志

<i>RTC_FLAG_TAMPO</i>	tamper 0事件标志
<i>RTC_FLAG_TIMESTAMP_MP_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
<b>输出参数{out}</b>	
-	-
<b>返回值</b>	
-	-

例如：

```
/* cleartime-stamp event flag */

rtc_flag_clear (RTC_FLAG_TIMESTAMP);
```

### 函数 `rtc_alter_output_config`

函数`rtc_alter_output_config`描述见下表：

**表 3-314. 函数 `rtc_alter_output_config`**

函数名称	<code>rtc_alter_output_config</code>
函数原型	<code>void rtc_alter_output_config(uint32_t source, uint32_t mode);</code>
功能描述	配置RTC备用输出源
先决条件	-
被调用函数	-
<b>输入参数{in}</b>	
<b>source</b>	指定输出信号
<i>RTC_CALIBRATION_5HZ</i>	当LSE时钟频率为32768Hz并且RTC_PSC 为默认值，输出512Hz 信号
<i>RTC_CALIBRATION_1HZ</i>	当LSE时钟频率为32768Hz并且RTC_PSC 为默认值，输出1Hz 信号
<i>RTC_ALARM_HIGH</i>	当设置了闹钟标志置位，输出引脚为高电平
<i>RTC_ALARM_LOW</i>	当设置了闹钟标志置位，输出引脚为低电平
<b>输入参数{in}</b>	
<b>mode</b>	当输出闹钟信号时指定输出引脚(PC13)的模式
<i>RTC_ALARM_OUTPUT_OD</i>	开漏输出
<i>RTC_ALARM_OUTPUT_PP</i>	推挽输出
<b>输出参数{out}</b>	
-	-

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 3-315. 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	-
The called functions	-
输入参数{in}	
window	选择校准窗口
RTC_CALIBRATION_WINDOW_32S	如果RTCCLK = 32768 Hz在32秒校准窗增加2exp20 RTCCLK 周期
RTC_CALIBRATION_WINDOW_16S	如果RTCCLK = 32768 Hz在16秒校准窗增加2exp19 RTCCLK周期
RTC_CALIBRATION_WINDOW_8S	如果RTCCLK = 32768 Hz在8秒校准窗增加2exp18 RTCCLK周期
输入参数{in}	
plus	增加或者不增加RTC脉冲
RTC_CALIBRATION_P_LUS_SET	每2048个RTC脉冲增加一个RTC脉冲
RTC_CALIBRATION_P_LUS_RESET	无影响
输入参数{in}	
minus	在校准窗口期间RTC减少的时钟(0x0 - 0x1FF)
Output parameter{out}	
-	-
Return value	
ErrStatus	ERROR或SUCCESS

例如：

```
/* configure RTC calibration register*/

ErrStatus error_status = rtc_calibration_config(RTC_CALIBRATION_WINDOW_32S,
```

```
RTC_CALIBRATION_PLUS_SET, 0xFF);
```

### 函数 **rtc\_hour\_adjust**

函数`rtc_hour_adjust`描述见下表：

**表 3-316. 函数 `rtc_hour_adjust`**

函数名称	rtc_hour_adjust
函数原型	<code>void rtc_hour_adjust(uint32_t operation);</code>
功能描述	通过在当前时间上增加或者减少一个小时来适应夏令时和冬令时
先决条件	-
被调用函数	-
<b>输入参数{in}</b>	
<b>operation</b>	小时调整操作
<code>RTC_CTL_A1H</code>	增加一个小时
<code>RTC_CTL_S1H</code>	减少一个小时
<b>输出参数{out}</b>	
-	-
<b>返回值</b>	
-	-

例如：

```
/* adjust the daylight saving time by adding one hour from the current time */
rtc_hour_adjust(RTC_CTL_A1H);
```

### 函数 **rtc\_second\_adjust**

函数`rtc_second_adjust`描述见下表：

**表 3-317. 函数 `rtc_second_adjust`**

函数名称	rtc_second_adjust
函数原型	<code>ErrStatus rtc_second_adjust(uint32_t add, uint32_t minus);</code>
功能描述	调整RTC当前时间的秒或亚秒值
先决条件	-
被调用函数	-
<b>输入参数{in}</b>	
<b>add</b>	在当前时间上增加1S或者不增加
<code>RTC_SHIFT_ADD1S_R</code> <code>ESET</code>	无影响
<code>RTC_SHIFT_ADD1S_S</code> <code>ET</code>	在当前时间增加1秒
<b>输入参数{in}</b>	
<b>minus</b>	在当前是时间上减少的亚秒值(0x0 - 0x7FFF)

输出参数{out}	
-	-
返回值	
-	-

例如：

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

ErrStatus error_status = rtc_second_adjust(RTC_SHIFT_ADD1S_SET, 0);
```

#### 函数 **rtc\_bypass\_shadow\_enable**

函数`rtc_bypass_shadow_enable`描述见下表：

**表 3-318. 函数 `rtc_bypass_shadow_enable`**

函数名称	rtc_bypass_shadow_enable
函数原型	void rtc_bypass_shadow_enable(void);
功能描述	使能RTC影子寄存器
先决条件	-
被调用函数	-
输入参数{in}	
-	-
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* enable RTC bypass shadow registers function*/

rtc_bypass_shadow_enable();
```

#### 函数 **rtc\_bypass\_shadow\_disable**

函数`rtc_bypass_shadow_disable`描述见下表：

**表 3-319. 函数 `rtc_bypass_shadow_disable`**

函数名称	rtc_bypass_shadow_disable
函数原型	void rtc_bypass_shadow_disable (void);
功能描述	失能RTC影子寄存器
先决条件	-
被调用函数	-
输入参数{in}	
-	-
输出参数{out}	

-	-
返回值	
-	-

例如：

```
/* disable RTC bypass shadow registers function*/
rtc_bypass_shadow_disable();
```

#### 函数 **rtc\_refclock\_detection\_enable**

函数`rtc_refclock_detection_enable`描述见下表：

**表 3-320. 函数 `rtc_refclock_detection_enable`**

函数名称	rtc_refclock_detection_enable
函数原型	ErrStatus rtc_refclock_detection_enable(void);
功能描述	使能RTC参考时钟检测功能
先决条件	-
被调用函数	rtc_init_mode_enter/rtc_init_mode_exit
输入参数{in}	
-	-
输出参数{out}	
-	-
返回值	
ErrStatus	ERROR 或 SUCCESS

例如：

```
/* enable RTC reference clock detection function*/
ErrStatus error_status = rtc_refclock_detection_enable();
```

#### 函数 **rtc\_refclock\_detection\_disable**

函数`rtc_refclock_detection_disable`描述见下表：

**表 3-321. 函数 `rtc_refclock_detection_disable`**

函数名称	rtc_refclock_detection_disable
函数原型	ErrStatus rtc_refclock_detection_disable(void);
功能描述	失能RTC参考时钟检测功能
先决条件	-
被调用函数	rtc_init_mode_enter/rtc_init_mode_exit
输入参数{in}	
-	-
输出参数{out}	
-	-

返回值	
ErrStatus	ERROR or SUCCESS

例如：

```
/* disableRTC reference clock detection function*/

ErrStatus error_status = rtc_refclock_detection_disable();
```

## 3.16. SPI

SPI/I2S模块可以通过SPI协议或I2S音频协议与外部设备进行通信。章节[3.16.1](#)描述了SPI/I2S的寄存器列表，章节[3.16.2](#)对SPI/I2S库函数进行说明。

### 3.16.1. 外设寄存器说明

SPI/I2S寄存器列表如下表所示：

**表 3-322. 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

### 3.16.2. 外设库函数说明

SPI/I2S库函数列表如下表所示：

**表 3-323. SPI/I2S 库函数**

库函数名称	库函数描述
spi_i2s_deinit	复位外设SPIx/I2Sx
spi_struct_para_init	将SPI结构体中所有参数初始化为默认值
spi_init	初始化外设SPIx
spi_enable	使能外设SPIx
spi_disable	失能外设SPIx
i2s_init	初始化外设I2Sx
i2s_psc_config	配置I2Sx预分频器

库函数名称	库函数描述
i2s_enable	使能外设I2Sx
i2s_disable	失能外设I2Sx
spi_nss_output_enable	使能外设SPIxNSS输出
spi_nss_output_disable	失能外设SPIxNSS输出
spi_nss_internal_high	NSS软件模式下NSS引脚拉高
spi_nss_internal_low	NSS软件模式下NSS引脚拉低
spi_dma_enable	使能外设SPIx的DMA功能
spi_dma_disable	失能外设SPIx的DMA功能
spi_i2s_data_frame_format_config	配置外设SPIx/I2Sx数据帧格式
spi_i2s_data_transmit	发送数据
spi_i2s_data_receive	接收数据
spi_bidirectional_transfer_config	配置外设SPIx的数据传输方向
spi_crc_polynomial_set	设置外设SPIx的CRC多项式值
spi_crc_polynomial_get	获取外设SPIx的CRC多项式值
spi_crc_on	打开外设SPIx的CRC功能
spi_crc_off	关闭外设SPIx的CRC功能
spi_crc_next	设置外设SPIx下一次传输数据为CRC值
spi_crc_get	外设SPIx获取CRC值
spi_ti_mode_enable	使能SPI TI模式
spi_ti_mode_disable	禁能SPI TI模式
spi_nssp_mode_enable	使能SPI NSS脉冲模式
spi_nssp_mode_disable	禁能SPI NSS脉冲模式
qspi_enable	使能四线SPI模式
qspi_disable	禁能四线SPI模式
qspi_write_enable	使能四线SPI写
qspi_read_enable	使能四线SPI读
qspi_io23_output_enable	使能SPI_IO2和SPI_IO3输出
qspi_io23_output_disable	禁能SPI_IO2和SPI_IO3输出
spi_i2s_interrupt_enable	使能外设SPIx/I2Sx中断
spi_i2s_interrupt_disable	失能外设SPIx/I2Sx中断
spi_i2s_interrupt_flag_get	获取外设SPIx/I2Sx中断状态
spi_i2s_flag_get	获取外设SPIx/I2Sx标志状态
spi_crc_error_clear	清除SPIx CRC错误标志状态
spi_fifo_access_size_config	配置SPI FIFO访问大小
spi_transmit_odd_config	配置SPI1通过DMA发送的数据总数是否是奇数
spi_receive_odd_config	配置SPI1通过DMA接收到的数据总数是否是奇数
spi_crc_length_set	设置SPI1 CRC长度

### 结构体 spi\_parameter\_struct

表 3-324. 结构体 spi\_parameter\_struct

成员名称	功能描述
------	------

device_mode	主机或设备模式配置 (SPI_MASTER, SPI_SLAVE)
trans_mode	传输模式 (SPI_TRANSMODE_FULLDUPLEX, SPI_TRANSMODE_RECEIVEONLY, SPI_TRANSMODE_BDRECEIVE, SPI_TRANSMODE_BDTRANSMIT)
frame_size	数据帧格式配置 (SPI_FRAMESIZE_xBIT, x=4,5,..16)
nss	NSS由软件或硬件控制配置 (SPI_NSS_SOFT, SPI_NSS_HARD)
Endian	大端或小端模式配置 (SPI_ENDIAN_MSB, SPI_ENDIAN_LSB)
clock_polarity_phase	相位和极性配置 (SPI_CK_PL_LOW_PH_1EDGE, SPI_CK_PL_HIGH_PH_1EDGE, SPI_CK_PL_LOW_PH_2EDGE, SPI_CK_PL_HIGH_PH_2EDGE)
prescale	预分频器配置 (SPI_PSC_n (n=2,4,8,16,32,64,128,256))

### 函数 **spi\_i2s\_deinit**

函数spi\_i2s\_deinit描述见下表:

**表 3-325. 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 3-326. 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3-324. 结构体spi_parameter_struct</a>
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* initialize the parameters of SPI */
spi_parameter_struct spi_init_struct;
spi_struct_para_init(&spi_init_struct);
```

### 函数 spi\_init

函数spi\_init描述见下表:

表 3-327. 函数 spi\_init

函数名称	spi_init
函数原形	ErrStatus spi_init(uint32_t spi_periph, spi_parameter_struct* spi_struct);
功能描述	初始化外设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3-324. 结构体spi_parameter_struct</a>
输出参数{out}	
-	-
返回值	
ErrStatus	ERROR或者SUCCESS

例如:

```
/* initialize SPI0 */
spi_parameter_struct spi_init_struct;
ErrStatus errstatus = ERROR;
```

---

```

spi_init_struct.trans_mode          = SPI_TRANSMODE_BDTRANSMIT;
spi_init_struct.device_mode        = SPI_MASTER;
spi_init_struct.frame_size         = SPI_FRAMESIZE_8BIT;
spi_init_struct.clock_polarity_phase = SPI_CK_PL_HIGH_PH_2EDGE;
spi_init_struct.nss                = SPI_NSS_SOFT;
spi_init_struct.prescale           = SPI_PSC_8;
spi_init_struct.endian              = SPI_ENDIAN_MSB;

errorstatus = spi_init(SPI0, &spi_init_struct);

```

### 函数 **spi\_enable**

函数spi\_enable描述见下表：

**表 3-328. 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 3-329. 函数 spi\_disable**

函数名称	spi_disable
函数原形	void spi_disable(uint32_t spi_periph);
功能描述	失能外设SPIx
先决条件	-
被调用函数	-

输入参数{in}	
<b>spi_periph</b>	外设SPIx
<b>SPIx</b>	x=0,1
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 3-330. 函数 i2s\_init**

函数名称	i2s_init
<b>函数原形</b>	void i2s_init(uint32_t spi_periph,uint32_t mode, uint32_t standard, uint32_t ckpl);
<b>功能描述</b>	初始化外设I2S0
<b>先决条件</b>	-
<b>被调用函数</b>	-
输入参数{in}	
<b>spi_periph</b>	外设I2S0
<b>SPIx</b>	x=0
输入参数{in}	
<b>mode</b>	I2S运行模式
<i>I2S_MODE_SLAVE</i> <i>TX</i>	I2S从机发送模式
<i>I2S_MODE_SLAVE</i> <i>RX</i>	I2S从机接收模式
<i>I2S_MODE_MASTER</i> <i>RTX</i>	I2S主机发送模式
<i>I2S_MODE_MASTER</i> <i>RRX</i>	I2S主机接收模式
输入参数{in}	
<b>standard</b>	I2S标准选择
<i>I2S_STD_PHILLIPS</i>	I2S飞利浦标准
<i>I2S_STD_MSB</i>	I2S MSB对齐标准
<i>I2S_STD_LSB</i>	I2S LSB对齐标准
<i>I2S_STD_PCM_SHOT</i> <i>RT</i>	I2S PCM短帧标准

<i>I2S_STD_PCMLONG</i>	I2S PCM长帧标准
<b>输入参数{in}</b>	
<i>ckpl</i>	I2S空闲状态时钟极性
<i>I2S_CKPL_LOW</i>	I2S_CK空闲状态为低电平
<i>I2S_CKPL_HIGH</i>	I2S_CK空闲状态为高电平
<b>输出参数{out}</b>	
-	-
<b>返回值</b>	
-	-

例如：

```
/* initialize I2S0 */

i2s_init(SPI0, I2S_MODE_MASTERTX, I2S_STD_PHILLIPS, I2S_CKPL_LOW);
```

### 函数 i2s\_psc\_config

函数i2s\_psc\_config描述见下表：

**表 3-331. 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
<b>输入参数{in}</b>	
<i>spi_periph</i>	外设I2S0
<i>SPIx</i>	x=0
<b>输入参数{in}</b>	
<i>audiosample</i>	I2S音频采样频率
<i>I2S_AUDIOSAMPL_E_8K</i>	音频采样频率为8KHz
<i>I2S_AUDIOSAMPL_E_11K</i>	音频采样频率为11KHz
<i>I2S_AUDIOSAMPL_E_16K</i>	音频采样频率为16KHz
<i>I2S_AUDIOSAMPL_E_22K</i>	音频采样频率为22KHz
<i>I2S_AUDIOSAMPL_E_32K</i>	音频采样频率为32KHz
<i>I2S_AUDIOSAMPL_E_44K</i>	音频采样频率为44KHz

<i>I2S_AUDIOSAMPL_E_48K</i>	音频采样频率为48KHz
<i>I2S_AUDIOSAMPL_E_96K</i>	音频采样频率为96KHz
<i>I2S_AUDIOSAMPL_E_192K</i>	音频采样频率为192KHz
<b>输入参数{in}</b>	
<b>frameformat</b>	I2S数据长度和通道长度
<i>I2S_FRAMEFORMAT_DT16B_CH16B</i>	I2S数据长度为16位，通道长度为16位
<i>I2S_FRAMEFORMAT_DT16B_CH32B</i>	I2S数据长度为16位，通道长度为32位
<i>I2S_FRAMEFORMAT_DT24B_CH32B</i>	I2S数据长度为24位，通道长度为32位
<i>I2S_FRAMEFORMAT_DT32B_CH32B</i>	I2S数据长度为32位，通道长度为32位
<b>输入参数{in}</b>	
<b>mckout</b>	I2S_MCK输出使能
<i>I2S_MCKOUT_ENA_BLE</i>	I2S_MCK输出使能
<i>I2S_MCKOUT_DISABLE</i>	I2S_MCK输出禁止
<b>输出参数{out}</b>	
-	-
<b>返回值</b>	
-	-

例如：

```
/* configure I2S0 prescaler */

i2s_psc_config(SPI0, I2S_AUDIOSAMPLE_44K, I2S_FRAMEFORMAT_DT16B_CH16B,
I2S_MCKOUT_DISABLE);
```

### 函数 i2s\_enable

函数i2s\_enable描述见下表：

**表 3-332. 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	-
<b>输入参数{in}</b>	

<b>spi_periph</b>	外设I2S0
<b>SPIx</b>	x=0
<b>输出参数{out}</b>	
-	-
<b>返回值</b>	
-	-

例如：

```
/* enable I2S0 */
```

```
i2s_enable(SPI0);
```

### 函数 i2s\_disable

函数i2s\_disable描述见下表：

**表 3-333. 函数 i2s\_disable**

函数名称	i2s_disable
函数原形	void i2s_disable(uint32_t spi_periph);
功能描述	失能外设I2S0
先决条件	-
被调用函数	-
<b>输入参数{in}</b>	
<b>spi_periph</b>	外设I2S0
<b>SPIx</b>	x=0
<b>输出参数{out}</b>	
-	-
<b>返回值</b>	
-	-

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 3-334. 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}	
<b>spi_periph</b>	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 3-335. 函数 spi\_nss\_output\_disable**

函数名称	spi_nss_output_disable
函数原形	void spi_nss_output_disable(uint32_t spi_periph);
功能描述	失能外设SPIx NSS输出
先决条件	-
被调用函数	-
输入参数{in}	
<b>spi_periph</b>	外设SPIx
SPIx	x=0,1
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 3-336. 函数 spi\_nss\_internal\_high**

函数名称	spi_nss_internal_high
函数原形	void spi_nss_internal_high(uint32_t spi_periph);
功能描述	NSS软件模式下NSS引脚拉高
先决条件	-

被调用函数	-
<b>输入参数{in}</b>	
<b>spi_periph</b>	外设SPIx
<b>SPIx</b>	x=0,1
<b>输出参数{out}</b>	
-	-
<b>返回值</b>	
-	-

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 3-337. 函数 spi\_nss\_internal\_low**

函数名称	spi_nss_internal_low
函数原形	void spi_nss_internal_low(uint32_t spi_periph);
功能描述	NSS软件模式下NSS引脚拉低
先决条件	-
被调用函数	-
<b>输入参数{in}</b>	
<b>spi_periph</b>	外设SPIx
<b>SPIx</b>	x=0,1
<b>输出参数{out}</b>	
-	-
<b>返回值</b>	
-	-

例如：

```
/* SPI0 NSS pin is pulled low level in software mode */
spi_nss_internal_low(SPI0);
```

### 函数 **spi\_dma\_enable**

函数spi\_dma\_enable描述见下表：

**表 3-338. 函数 spi\_dma\_enable**

函数名称	spi_dma_enable
函数原形	void spi_dma_enable(uint32_t spi_periph, uint8_t dma);
功能描述	使能外设SPIx的DMA功能

先决条件	-
被调用函数	-
<b>输入参数{in}</b>	
<b>spi_periph</b>	外设SPIx
<i>SPIx</i>	x=0,1
<b>输入参数{in}</b>	
<b>dma</b>	SPI DMA模式
<i>SPI_DMA_TRANSMIT</i>	SPI发送缓冲区DMA使能
<i>SPI_DMA_RECEIVE</i>	SPI接收缓冲区DMA使能
<b>输出参数{out}</b>	
-	-
<b>返回值</b>	
-	-

例如：

```
/* enable SPI0 transmit data DMA function */
spi_dma_enable(SPI0, SPI_DMA_TRANSMIT);
```

### 函数 **spi\_dma\_disable**

函数spi\_dma\_disable描述见下表：

**表 3-339. 函数 spi\_dma\_disable**

函数名称	spi_dma_disable
函数原形	void spi_dma_disable(uint32_t spi_periph, uint8_t dma);
功能描述	失能外设SPIx的DMA功能
先决条件	-
被调用函数	-
<b>输入参数{in}</b>	
<b>spi_periph</b>	外设SPIx
<i>SPIx</i>	x=0,1
<b>输入参数{in}</b>	
<b>dma</b>	SPI DMA模式
<i>SPI_DMA_TRANSMIT</i>	SPI发送缓冲区DMA使能
<i>SPI_DMA_RECEIVE</i>	SPI接收缓冲区DMA使能
<b>输出参数{out}</b>	
-	-
<b>返回值</b>	
-	-

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 3-340. 函数 spi\_i2s\_data\_frame\_format\_config**

函数名称	spi_i2s_data_frame_format_config
函数原形	ErrStatus spi_i2s_data_frame_format_config(uint32_t spi_periph, uint16_t frame_format);
功能描述	配置外设SPIx/I2S0数据帧格式
先决条件	-
被调用函数	-
输入参数{in}	
spi_periph	外设SPIx
SPIx	x=0,1
输入参数{in}	
frame_format	SPI帧大小
SPI_FRAMESIZE_x BIT	SPI x位数据帧格式, x=4,5,..16
输出参数{out}	
-	-
返回值	
ErrStatus	ERROR或者SUCCESS-

例如：

```
/* configure SPI0/I2S0 data frame format size is 16 bits */

spi_i2s_data_frame_format_config(SPI1, SPI_FRAMESIZE_16BIT);
```

### 函数 **spi\_i2s\_data\_transmit**

函数spi\_i2s\_data\_transmit描述见下表：

**表 3-341. 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

<i>SPIx</i>	x=0,1
<b>输入参数{in}</b>	
<b>data</b>	16位数据
<b>输出参数{out}</b>	
-	-
<b>返回值</b>	
-	-

例如：

```
/* SPI0 transmit data */

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 3-342. 函数 spi\_i2s\_data\_receive**

函数名称	spi_i2s_data_receive
函数原形	uint16_t spi_i2s_data_receive(uint32_t spi_periph);
功能描述	SPI接收数据
先决条件	-
被调用函数	-
<b>输入参数{in}</b>	
<b>spi_periph</b>	外设SPIx
<i>SPIx</i>	x=0,1
<b>输出参数{out}</b>	
-	-
<b>返回值</b>	
<b>uint16_t</b>	16位数据

例如：

```
/* SPI0 receive data */

spi0_receive_array[receive_n] = spi_i2s_data_receive(SPI0);
```

#### 函数 **spi\_bidirectional\_transfer\_config**

函数spi\_bidirectional\_transfer\_config描述见下表：

**表 3-343. 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

<b>先决条件</b>	-
<b>被调用函数</b>	-
<b>输入参数{in}</b>	
<b>spi_periph</b>	外设SPIx
<b>SPIx</b>	x=0,1
<b>输入参数{in}</b>	
<b>transfer_direction</b>	SPI双向传输输出使能
<b>SPI_BIDIRECTION</b>	SPI工作在只发送模式
<b>AL_TRANSMIT</b>	
<b>SPI_BIDIRECTION</b>	SPI工作在只接收模式
<b>AL_RECEIVE</b>	
<b>输出参数{out}</b>	
-	-
<b>返回值</b>	
-	-

例如：

```
/* SPI0 works in transmit-only mode */
spi_bidirectional_transfer_config(SPI0, SPI_BIDIRECTIONAL_TRANSMIT);
```

#### 函数 **spi\_crc\_polynomial\_set**

函数spi\_crc\_polynomial\_set描述见下表：

**表 3-344. 函数 spi\_crc\_polynomial\_set**

<b>函数名称</b>	spi_crc_polynomial_set
<b>函数原形</b>	void spi_crc_polynomial_set(uint32_t spi_periph, uint16_t crc_poly);
<b>功能描述</b>	设置外设SPIx的CRC多项式值
<b>先决条件</b>	-
<b>被调用函数</b>	-
<b>输入参数{in}</b>	
<b>spi_periph</b>	外设SPIx
<b>SPIx</b>	x=0,1
<b>输入参数{in}</b>	
<b>crc_poly</b>	CRC多项式值
<b>输出参数{out}</b>	
-	-
<b>返回值</b>	
-	-

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 3-345. 函数 spi\_crc\_polynomial\_get**

函数名称	spi_crc_polynomial_get
函数原形	uint16_t spi_crc_polynomial_get(uint32_t spi_periph);
功能描述	获取外设SPIx的CRC多项式值
先决条件	-
被调用函数	-
输入参数{in}	
spi_periph	外设SPIx
SPIx	x=0,1
输出参数{out}	
-	-
返回值	
uint16_t	16位CRC多项式值 (0-0xFFFF)

例如:

```
/* get SPI0 CRC polynomial */

uint16_t crc_val;

crc_val = spi_crc_polynomial_get(SPI0);
```

### 函数 **spi\_crc\_on**

函数spi\_crc\_on描述见下表:

**表 3-346. 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 3-347. 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 3-348. 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 3-349. 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\_ti\_mode\_enable**

函数**spi\_ti\_mode\_enable**描述见下表:

**表 3-350. 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 3-351. 函数 spi\_ti\_mode\_disable**

函数名称	spi_ti_mode_disable
函数原形	void spi_ti_mode_disable(uint32_t spi_periph);
功能描述	失能SPI TI模式
先决条件	-
被调用函数	-
输入参数{in}	
spi_periph	外设SPIx
SPIx	x=0,1
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 3-352. 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 3-353. 函数 **spi\_nssp\_mode\_disable**

函数名称	spi_nssp_mode_disable
函数原形	void spi_nssp_mode_disable(uint32_t spi_periph);
功能描述	失能SPI NSS脉冲模式
先决条件	-
被调用函数	-
输入参数{in}	
spi_periph	外设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 **qspi\_enable**

函数qspi\_enable描述见下表：

表 3-354. 函数 **qspi\_enable**

函数名称	qspi_enable
函数原形	void qspi_enable(uint32_t spi_periph);
功能描述	使能四线SPI模式
先决条件	-
被调用函数	-
输入参数{in}	
spi_periph	外设SPIx

<i>SPIx</i>	x=1
<b>输出参数{out}</b>	
-	-
<b>返回值</b>	
-	-

例如：

```
/* enable SPI1 quad wire mode */

qspi_enable(SPI1);
```

### 函数 **qspi\_disable**

函数qspi\_disable描述见下表：

**表 3-355. 函数 qspi\_disable**

函数名称	qspi_disable
函数原形	void qspi_disable(uint32_t spi_periph);
功能描述	失能四线SPI模式
先决条件	-
被调用函数	-
<b>输入参数{in}</b>	
<i>spi_periph</i>	外设SPIx
<i>SPIx</i>	x=1
<b>输出参数{out}</b>	
-	-
<b>返回值</b>	
-	-

例如：

```
/* disable SPI1 quad wire mode */

qspi_disable(SPI1);
```

### 函数 **qspi\_write\_enable**

函数qspi\_write\_enable描述见下表：

**表 3-356. 函数 qspi\_write\_enable**

函数名称	qspi_write_enable
函数原形	void qspi_write_enable(uint32_t spi_periph);
功能描述	使能四线SPI写
先决条件	-
被调用函数	-
<b>输入参数{in}</b>	

<b>spi_periph</b>	外设SPIx
<i>SPIx</i>	x=1
<b>输出参数{out}</b>	
-	-
<b>返回值</b>	
-	-

例如：

```
/* enable SPI1 quad wire write */
qspi_write_enable(SPI1);
```

### 函数 qspi\_read\_enable

函数qspi\_read\_enable描述见下表：

**表 3-357. 函数 qspi\_read\_enable**

函数名称	qspi_read_enable
函数原形	void qspi_read_enable(uint32_t spi_periph);
功能描述	使能四线SPI读
先决条件	-
被调用函数	-
<b>输入参数{in}</b>	
<b>spi_periph</b>	外设SPIx
<i>SPIx</i>	x=1
<b>输出参数{out}</b>	
-	-
<b>返回值</b>	
-	-

例如：

```
/* enable SPI1 quad wire read */
qspi_read_enable(SPI1);
```

### 函数 qspi\_io23\_output\_enable

函数qspi\_io23\_output\_enable描述见下表：

**表 3-358. 函数 qspi\_io23\_output\_enable**

函数名称	qspi_io23_output_enable
函数原形	void qspi_io23_output_enable(uint32_t spi_periph);
功能描述	使能SPI_IO2和SPI_IO3输出
先决条件	-
被调用函数	-

输入参数{in}	
<b>spi_periph</b>	外设SPIx
<b>SPIx</b>	x=1
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* enable SPI1 SPI_IO2 and SPI_IO3 pin output */

qspi_io23_output_enable(SPI1);
```

### 函数 **qspi\_io23\_output\_disable**

函数qspi\_io23\_output\_disable描述见下表：

**表 3-359. 函数 qspi\_io23\_output\_disable**

函数名称	qspi_io23_output_disable
函数原形	void qspi_io23_output_disable(uint32_t spi_periph);
功能描述	失能SPI_IO2和SPI_IO3输出
先决条件	-
被调用函数	-
输入参数{in}	
<b>spi_periph</b>	外设SPIx
<b>SPIx</b>	x=1
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* disable SPI1 SPI_IO2 and SPI_IO3 pin output */

qspi_io23_output_disable(SPI1);
```

### 函数 **spi\_i2s\_interrupt\_enable**

函数spi\_i2s\_interrupt\_enable描述见下表：

**表 3-360. 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	-

<b>被调用函数</b>	-
<b>输入参数{in}</b>	
<b>spi_periph</b>	外设SPIx
<b>SPIx</b>	x=0,1
<b>输入参数{in}</b>	
<b>interrupt</b>	SPI/I2S中断
<b>SPI_I2S_INT_TBE</b>	发送缓冲区空中断使能
<b>SPI_I2S_INT_RBNE</b>	接收缓冲区非空中断使能
<b>SPI_I2S_INT_ERR</b>	错误中断使能
<b>输出参数{out}</b>	
-	-
<b>返回值</b>	
-	-

例如：

```
/* enable SPI0 transmit buffer empty interrupt */
spi_i2s_interrupt_enable(SPI0, SPI_I2S_INT_TBE);
```

### 函数 **spi\_i2s\_interrupt\_disable**

函数spi\_i2s\_interrupt\_disable描述见下表：

**表 3-361. 函数 spi\_i2s\_interrupt\_disable**

<b>函数名称</b>	spi_i2s_interrupt_disable
<b>函数原形</b>	void spi_i2s_interrupt_disable(uint32_t spi_periph, uint8_t interrupt);
<b>功能描述</b>	失能外设SPIx/I2S0中断
<b>先决条件</b>	-
<b>被调用函数</b>	-
<b>输入参数{in}</b>	
<b>spi_periph</b>	外设SPIx
<b>SPIx</b>	x=0,1
<b>输入参数{in}</b>	
<b>interrupt</b>	SPI/I2S中断
<b>SPI_I2S_INT_TBE</b>	发送缓冲区空中断使能
<b>SPI_I2S_INT_RBNE</b>	接收缓冲区非空中断使能
<b>SPI_I2S_INT_ERR</b>	错误中断使能
<b>输出参数{out}</b>	
-	-
<b>返回值</b>	
-	-

例如：

```
/* disable SPI0 transmit buffer empty interrupt */
```

```
spi_i2s_interrupt_disable(SPI0, SPI_I2S_INT_TBE);
```

### 函数 **spi\_i2s\_interrupt\_flag\_get**

函数**spi\_i2s\_interrupt\_flag\_get**描述见下表：

**表 3-362. 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}	
spi_periph	外设SPIx
SPIx	x=0,1
输入参数{in}	
interrupt	SPI/I2S中断状态
SPI_I2S_INT_FLAG_TBE	发送缓冲区空中断
SPI_I2S_INT_FLAG_RBNE	接收缓冲区非空中断
SPI_I2S_INT_FLAG_RXORERR	接收过载错误中断
SPI_INT_FLAG_CO_NFERR	配置错误中断
SPI_INT_FLAG_CR_CERR	CRC错误中断
I2S_INT_FLAG_TX_URERR	发送欠载错误中断
SPI_I2S_INT_FLAG_FERR	格式错误中断
输出参数{out}	
-	-
返回值	
FlagStatus	SET 或 RESET

例如：

```
/* get SPI0 transmit buffer empty interrupt status */

if(RESET != spi_i2s_interrupt_flag_get(SPI0, SPI_I2S_INT_FLAG_TBE)){
    while(RESET == spi_i2s_flag_get(SPI0, SPI_FLAG_TBE));
    spi_i2s_data_transmit(SPI0, spi0_send_array[send_n++]);
}
```

### 函数 **spi\_i2s\_flag\_get**

函数spi\_i2s\_flag\_get描述见下表：

**表 3-363. 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	-
<b>输入参数{in}</b>	
<b>spi_periph</b>	外设SPIx
<b>SPIx</b>	x=0,1
<b>输入参数{in}</b>	
<b>flag</b>	SPI/I2S标志状态
<b>SPI_FLAG_TBE</b>	SPI发送缓冲区空标志
<b>SPI_FLAG_RBNE</b>	SPI接收缓冲区非空标志
<b>SPI_FLAG_TRANS</b>	SPI通信进行中标志
<b>SPI_FLAG_RXORE_RR</b>	SPI接收过载错误标志
<b>SPI_FLAG_CONFE_RR</b>	SPI配置错误标志
<b>SPI_FLAG_CRCER_R</b>	SPI CRC错误标志
<b>SPI_FLAG_FERR</b>	SPI格式错误标志
<b>I2S_FLAG_TBE</b>	I2S发送缓冲区空标志
<b>I2S_FLAG_RBNE</b>	I2S接收缓冲区非空标志
<b>I2S_FLAG_TRANS</b>	I2S通信进行中标志
<b>I2S_FLAG_RXORE_RR</b>	I2S接收过载错误标志
<b>I2S_FLAG_TXURE_RR</b>	I2S发送欠载错误标志
<b>I2S_FLAG_CH</b>	I2S通道标志
<b>I2S_FLAG_FERR</b>	I2S格式错误标志
以下参数只适用于SPI	
<b>SPI_TXLVL_EMPT_Y</b>	SPI TXFIFO空
<b>SPI_TXLVL_QUARTER_FULL</b>	SPI TXFIFO四分之一满
<b>SPI_TXLVL_HALF_FULL</b>	SPI TXFIFO半满
<b>SPI_TXLVL_FULL</b>	TXFIFO全满
<b>SPI_RXLVL_EMPT</b>	SPI RXFIFO空

Y	
SPI_RXLVL_QUARTER_FULL	SPI RXFIFO四分之一满
SPI_RXLVL_HALF_FULL	SPI RXFIFO半满
SPI_RXLVL_FULL	RXFIFO全满
输出参数{out}	
-	-
返回值	
FlagStatus	SET 或 RESET

例如：

```
/* get SPI0 transmit buffer empty flag status */
while(RESET == spi_i2s_flag_get(SPI0, SPI_FLAG_TBE));
spi_i2s_data_transmit(SPI0, spi0_send_array[send_n++]);
```

#### 函数 **spi\_crc\_error\_clear**

函数spi\_crc\_error\_clear描述见下表：

表 3-364. 函数 **spi\_crc\_error\_clear**

函数名称	spi_crc_error_clear
函数原形	void spi_crc_error_clear(uint32_t spi_periph);
功能描述	清除SPIx CRC错误标志状态
先决条件	-
被调用函数	-
输入参数{in}	
spi_periph	外设SPIx
SPIx	x=0,1
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* clear SPI0 CRC error flag status */
spi_crc_error_clear(SPI0);
```

#### 函数 **spi\_fifo\_access\_size\_config**

函数spi\_fifo\_access\_size\_config描述见下表：

**表 3-365. 函数 spi\_fifo\_access\_size\_config**

函数名称	spi_fifo_access_size_config	
函数原形	void spi_fifo_access_size_config(uint32_t spi_periph, uint16_t fifo_access_size);	
功能描述	配置SPI1的FIFO访问大小	
先决条件	-	
被调用函数	-	
输入参数{in}		
spi_periph	外设SPIx	
SPIx	x=1	
输入参数{in}		
fifo_access_size	fifo访问大小	
SPI_HALFWORD_ACCESS	半字访问	
SPI_BYTE_ACCESS	字节访问	
输出参数{out}		
-	-	
返回值		
-	-	

例如：

```
/* configure SPI1 access size half word */
spi_fifo_access_size_config(SPI1, SPI_HALFWORD_ACCESS);
```

#### 函数 spi\_transmit\_odd\_config

函数spi\_transmit\_odd\_config描述见下表：

**表 3-366. 函数 spi\_transmit\_odd\_config**

函数名称	spi_transmit_odd_config	
函数原形	void spi_transmit_odd_config(uint32_t spi_periph, uint16_t odd);	
功能描述	配置SPI1通过DMA发送的数据总数是否为奇数	
先决条件	-	
被调用函数	-	
输入参数{in}		
spi_periph	外设SPIx	
SPIx	x=1	
输入参数{in}		
odd	DMA通道发送的字节数是奇数还是偶数	
SPI_TXDMA EVEN	DMA发送的字节数是偶数	
SPI_TXDMA ODD	DMA发送的字节数是奇数	
输出参数{out}		

-	-
返回值	
-	-

例如：

```
/* configure SPI1 total number of data to transmit by DMA is odd */

spi_transmit_odd_config(SPI1, SPI_TXDMA_ODD);
```

### 函数 **spi\_receive\_odd\_config**

函数spi\_receive\_odd\_config描述见下表：

**表 3-367. 函数 spi\_receive\_odd\_config**

函数名称	spi_receive_odd_config
函数原形	void spi_receive_odd_config(uint32_t spi_periph, uint16_t odd);
功能描述	配置SPI1通过DMA接收到的数据总数是否为奇数
先决条件	-
被调用函数	-
输入参数{in}	
spi_periph	外设SPIx
SPIx	x=1
输入参数{in}	
odd	DMA通道接收的字节数时奇数还是偶数
SPI_RXDMA EVEN	DMA接收的字节数是偶数
SPI_RXDMA ODD	DMA接收的字节数是奇数
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* configure SPI1 total number of data to receive by DMA is odd */

spi_receive_odd_config(SPI1, SPI_RXDMA_ODD);
```

### 函数 **spi\_crc\_length\_set**

函数spi\_crc\_length\_set描述见下表：

**表 3-368. 函数 spi\_crc\_length\_set**

函数名称	spi_crc_length_set
函数原形	void spi_crc_length_set(uint32_t spi_periph, uint16_t crc_length);
功能描述	设置CRC长度
先决条件	-

<b>被调用函数</b>	-
<b>输入参数{in}</b>	
<b>spi_periph</b>	外设SPIx
<b>SPIx</b>	x=1
<b>输入参数{in}</b>	
<b>crc_length</b>	crc长度
<b>SPI_CRC_8BIT</b>	CRC长度为8位数据
<b>SPI_CRC_16BIT</b>	CRC长度为16位数据
<b>输出参数{out}</b>	
-	-
<b>返回值</b>	
-	-

例如：

```
/*set SPI1 CRC length 16 bits */
spi_crc_length_set(SPI1,SPI_CRC_16BIT);
```

## 3.17. SYSCFG

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

### 3.17.1. 外设寄存器说明

SYSCFG寄存器列表如下表所示：

**表 3-369. SYSCFG 寄存器**

寄存器名称	寄存器描述
<b>SYSCFG_CFG0</b>	配置寄存器0
<b>SYSCFG_EXTISSL0</b>	EXTI源选择寄存器0
<b>SYSCFG_EXTISSL1</b>	EXTI源选择寄存器1
<b>SYSCFG_EXTISSL2</b>	EXTI源选择寄存器2
<b>SYSCFG_EXTISSL3</b>	EXTI源选择寄存器3
<b>SYSCFG_CFG2</b>	系统配置寄存器2
<b>SYSCFG_CPU_IRQ_LAT</b>	IRQ延迟寄存器

### 3.17.2. 外设库函数说明

SYSCFG库函数列表如下表所示：

**表 3-370. 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timer0/14/15/16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 3-371. 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 3-372. 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}	
<b>syscfg_dma_remap_p</b>	指定要重新映射的DMA通道
<b>SYSCFG_DMA_REMAP_TIMER16</b>	重新映射Time16通道0和向通道1发送DMA请求
<b>SYSCFG_DMA_REMAP_TIMER15</b>	重新映射Time15通道5和向通道3发送DMA请求
<b>SYSCFG_DMA_REMAP_USART0RX</b>	将UART0 RX DMA请求重新映射到通道4
<b>SYSCFG_DMA_REMAP_USART0TX</b>	将UART0 TX DMA请求重新映射到通道3
<b>SYSCFG_DMA_REMAP_ADC</b>	从通道0重新映射ADC DMA请求到通道1
<b>SYSCFG_PA11_REMAP_PA12</b>	PA11和PA12重新映射位
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 3-373. 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}	
<b>syscfg_dma_remap_p</b>	指定要重新映射的DMA通道
<b>SYSCFG_DMA_REMAP_TIMER16</b>	重新映射Time16通道0和向通道1发送DMA请求
<b>SYSCFG_DMA_REMAP_TIMER15</b>	重新映射Time15通道5和向通道3发送DMA请求
<b>SYSCFG_DMA_REMAP_USART0RX</b>	将UART0 RX DMA请求重新映射到通道4

<code>SYSCFG_DMA_REMAP_USART0TX</code>	将UART0 TX DMA请求重新映射到通道3
<code>SYSCFG_DMA_REMAP_ADC</code>	从通道0重新映射ADC DMA请求到通道1
<code>SYSCFG_PA11_REMAP_PA12</code>	PA11和PA12重新映射位
<b>输出参数{out}</b>	
-	-
<b>返回值</b>	
-	-

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 3-374. 函数 `syscfg_high_current_enable`**

函数名称	<code>syscfg_high_current_enable</code>
函数原形	<code>void syscfg_high_current_enable(void);</code>
功能描述	使能PB9引脚大电流能力
先决条件	-
被调用函数	-
<b>输入参数{in}</b>	
-	-
<b>输出参数{out}</b>	
-	-
<b>返回值</b>	
-	-

例如：

```
/* enable 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 3-375. 函数 `syscfg_high_current_disable`**

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

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 3-376. 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}	
<b>exti_port</b>	指定EXTI使用的GPIO端口
<b>EXTI_SOURCE_GP /Ox</b>	x=A,B,C,F
输入参数{in}	
<b>exti_pin</b>	EXTI引脚
<b>EXTI_SOURCE_PINx</b>	x=0..15(GPIOA, GPIOB), x=13..15(GPIOC), x = 0..1..6..7 (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 3-377. 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timer0/14/15/16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_L OCKUP	Cortex-M23锁定输出连接到断开输入
SYSCFG_LOCK_S RAM_PARITY_ERR OR	SRAM_PARITY校验错误连接到断开输入
SYSCFG_LOCK_LV D	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 **irq\_latency\_set**

函数**irq\_latency\_set**描述见下表:

**表 3-378. 函数 irq\_latency\_set**

函数名称	irq_latency_set
函数原形	void irq_latency_set(uint8_t irq_latency);
功能描述	设置延迟时间值
先决条件	-
被调用函数	-
输入参数{in}	
irq_latency	延迟时间值
0x00-0xFF	延迟时间值
输出参数{out}	
-	-
返回值	

-	-
---	---

例如：

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

```
irq_latency_set (0xFF);
```

### 函数 **syscfg\_flag\_get**

函数**syscfg\_flag\_get**描述见下表：

**表 3-379. 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}	
<b>syscfg_flag</b>	中断标志
<b>SYSCFG_SRAM_P CEF</b>	SRAM奇偶校验错误标志
输出参数{out}	
-	-
返回值	
<b>FlagStatus</b>	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 3-380. 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}	
<b>syscfg_flag</b>	中断标志
<b>SYSCFG_SRAM_P</b>	SRAM奇偶校验错误标志

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

例如：

```
/* clear syscfg flag */

syscfg_flag_clear(SYSCFG_SRAM_PCEF);
```

## 3.18. TIMER

定时器含有可编程的一个无符号计数器，支持输入捕获和输出比较，分为五种类型：高级定时器(TIMER0)，通用定时器L0(TIMER2)，通用定时器L2(TIMER13)，通用定时器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[3.18.1](#)描述了TIMER的寄存器列表，章节[3.18.2](#)对TIMER库函数进行说明。

### 3.18.1. 外设寄存器说明

TIMER寄存器列表如下表所示：

**表 3-381. TIMER 寄存器**

寄存器名称	寄存器描述
TIMER_CTL0(timerx, x=0, 2, 5, 13, 14, 15, 16)	控制寄存器0
TIMERx_CTL1(timerx, x=0, 2, 5, 13, 14, 15, 16)	控制寄存器1
TIMERx_SMCFG(timerx, x=0, 2, 14)	从模式配置寄存器
TIMERx_DMAINTEN(timerx, x=0, 2, 5, 13, 14, 15, 16)	DMA和中断使能寄存器
TIMERx_INTF(timerx, x=0, 2, 5, 13, 14, 15, 16)	中断标志寄存器
TIMERx_SWEVG(timerx, x=0, 2, 5, 13, 14, 15, 16)	软件事件产生寄存器
TIMERx_CHCTL0(timerx, x=0, 2, 13, 14, 15, 16)	通道控制寄存器0
TIMERx_CHCTL1(timerx, x=0, 2)	通道控制寄存器1
TIMERx_CHCTL2(timerx, x=0, 2, 13, 14, 15, 16)	通道控制寄存器2
TIMERx_CNT(timerx, x=0, 2, 5, 13, 14, 15, 16)	计数器寄存器
TIMERx_PSC(timerx, x=0, 2, 5, 13, 14, 15, 16)	预分频寄存器
TIMERx_CAR(timerx, x=0, 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, 2, 13, 14, 15, 16)	通道0捕获/比较寄存器

寄存器名称	寄存器描述
TIMERx_CH1CV(timerx, x=0, 2, 14)	通道1捕获/比较寄存器
TIMERx_CH2CV(timerx, x=0, 2)	通道2捕获/比较寄存器
TIMERx_CH3CV(timerx, x=0, 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, 2, 14, 15, 16)	互补通道保护寄存器
TIMERx_DMACFG(timerx, x=0, 2, 14, 15, 16)	DMA配置寄存器
TIMERx_DMATB(timerx, x=0, 2, 14, 15, 16)	DMA发送缓冲区寄存器
TIMERx_CFG(timerx, x=0, 2, 13, 14, 15, 16)	配置寄存器

### 3.18.2. 外设库函数说明

TIMER库函数列表如下表所示：

**表 3-382. 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_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_flag_get	外设TIMERx标志位获取
timer_flag_clear	外设TIMERx标志位清除

库函数名称	库函数描述
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
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

库函数名称	库函数描述
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
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 3-383. 结构体 timer\_parameter\_struct**

成员名称	功能描述
prescaler	预分频值 (0~65535)
alignedmode	对齐模式 (TIMER_COUNTER_EDGE, TIMER_COUNTER_CENTER_DOWN, TIMER_COUNTER_CENTER_UP, TIMER_COUNTER_CENTER_BOTH)
counterdirection	计数方向 (TIMER_COUNTER_UP, TIMER_COUNTER_DOWN)
period	周期 (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 3-384. 结构体 timer\_break\_parameter\_struct**

成员名称	功能描述
runoffstate	运行模式下“关闭状态”配置 (TIMER_ROS_STATE_ENABLE,

成员名称	功能描述
	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 3-385. 结构体 timer\_oc\_parameter\_struct

成员名称	功能描述
outputstate	通道输出状态 (TIMER_CCX_ENABLE, TIMER_CCX_DISABLE)
outputnstate	互补通道输出状态 (TIMER_CCXN_ENABLE, TIMER_CCXN_DISABLE)
ocpolarity	通道输出极性 (TIMER_OC_POLARITY_HIGH, TIMER_OC_POLARITY_LOW)
ocnpolarity	互补通道输出极性 (TIMER_OCN_POLARITY_HIGH, TIMER_OCN_POLARITY_LOW)
ocidlestate	空闲状态下通道输出 (TIMER_OC_IDLE_STATE_LOW, TIMER_OC_IDLE_STATE_HIGH)
ocnidlestate	空闲状态下互补通道输出 (TIMER_OCN_IDLE_STATE_LOW, TIMER_OCN_IDLE_STATE_HIGH)

### 结构体 timer\_ic\_parameter\_struct

表 3-386. 结构体 timer\_ic\_parameter\_struct

成员名称	功能描述
icpolarity	通道输入极性 (TIMER_IC_POLARITY_RISING, TIMER_IC_POLARITY_FALLING, TIMER_IC_POLARITY_BOTH_EDGE)
icselection	通道输入模式选择 (TIMER_IC_SELECTION_DIRECTTI, TIMER_IC_SELECTION_INDIRECTTI, TIMER_IC_SELECTION_ITS)
icprescaler	通道输入捕获预分频 (TIMER_IC_PSC_DIV1, TIMER_IC_PSC_DIV2, TIMER_IC_PSC_DIV4, TIMER_IC_PSC_DIV8)
icfilter	通道输入捕获滤波 (0~15)

### 函数 timer\_deinit

函数timer\_deinit描述见下表:

**表 3-387. 函数 timer\_deinit**

函数名称	timer_deinit
函数原型	void timer_deinit(uint32_t timer_periph);
功能描述	复位外设TIMERx
先决条件	-
被调用函数	rcu_periph_reset_enable / rcu_periph_reset_disable
输入参数{in}	
timer_periph	TIMER外设
TIMERx(x=0, 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 3-388. 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3-383. 结构体 timer_parameter_struct</a>
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* initialize TIMER init parameter struct with a default value */
timer_parameter_struct timer_initpara;
timer_struct_para_init(timer_initpara);
```

### 函数 timer\_init

函数timer\_init描述见下表：

**表 3-389. 函数 timer\_init**

函数名称	timer_init
函数原型	void timer_init(uint32_t timer_periph, timer_parameter_struct* initpara);
功能描述	初始化外设TIMERx
先决条件	-
被调用函数	-
输入参数{in}	
timer_periph	TIMER外设
TIMERx(x=0, 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3-383. 结构体 timer_parameter_struct</a>
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* initialize TIMER0 */
timer_parameter_struct timer_initpara;
timer_initpara.prescaler      = 107;
timer_initpara.alignedmode    = TIMER_COUNTER_EDGE;
timer_initpara.counterdirection = TIMER_COUNTER_UP;
timer_initpara.period         = 999;
timer_initpara.clockdivision  = TIMER_CKDIV_DIV1;
timer_initpara.repetitioncounter = 1;
timer_init(TIMER0, &timer_initpara);
```

### 函数 timer\_enable

函数timer\_enable描述见下表：

**表 3-390. 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	-
<b>输入参数{in}</b>	
timer_periph	TIMER外设
TIMERx(x=0, 2, 5, 13..16)	TIMER外设选择
<b>输出参数{out}</b>	
-	-
<b>返回值</b>	
-	-

例如：

```
/* enable TIMER0 */
timer_enable (TIMER0);
```

### 函数 timer\_disable

函数timer\_disable描述见下表：

**表 3-391. 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	-
<b>输入参数{in}</b>	
timer_periph	TIMER外设
TIMERx(x=0, 2, 5, 13..16)	TIMER外设选择
<b>输出参数{out}</b>	
-	-
<b>返回值</b>	
-	-

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 3-392. 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 3-393. 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 3-394. 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 3-395. 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 3-396. 函数 timer\_counter\_alignment**

函数名称	timer_counter_alignment
函数原型	void timer_counter_alignment(uint32_t timer_periph, uint16_t aligned);
功能描述	设置外设TIMERx的对齐模式
先决条件	-
被调用函数	-
输入参数{in}	
timer_periph	TIMER外设
TIMERx(x=0, 2)	TIMER外设选择
输入参数{in}	
aligned	对齐模式
TIMER_COUNTER_EDGE	无中央对齐计数模式(边沿对齐模式), DIR位指定了计数方向
TIMER_COUNTER_CENTER_DOWN	中央对齐向下计数置1模式。计数器在中央计数模式计数, 通道被配置在输出模式 (TIMERx_CHCTL0寄存器中CHxMS=00), 只有在向下计数时, 通道的比较中断标志置1
TIMER_COUNTER_CENTER_UP	中央对齐向上计数置1模式。计数器在中央计数模式计数, 通道被配置在输出模式 (TIMERx_CHCTL0寄存器中CHxMS=00), 只有在向上计数时, 通道的比较中断标志置1
TIMER_COUNTER_CENTER_BOTH	中央对齐上下计数置1模式。计数器在中央计数模式计数, 通道被配置在输出模式 (TIMERx_CHCTL0寄存器中CHxMS=00), 在向上和向下计数时, 通道的比较中断标志都会置1
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* set TIMER0 counter center-aligned and counting up assert mode */

timer_counter_alignment (TIMER0, TIMER_COUNTER_CENTER_UP);
```

### 函数 timer\_counter\_up\_direction

函数timer\_counter\_up\_direction描述见下表:

**表 3-397. 函数 timer\_counter\_up\_direction**

函数名称	timer_counter_up_direction
函数原型	void timer_counter_up_direction(uint32_t timer_periph);

功能描述	设置外设TIMERx向上计数
先决条件	计数器设置为无中央对齐计数模式（边沿对齐模式）
被调用函数	-
<b>输入参数{in}</b>	
timer_periph	TIMER外设
TIMERx(x=0, 2)	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 3-398. 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	-
<b>输入参数{in}</b>	
timer_periph	TIMER外设
TIMERx(x=0, 2)	TIMER外设选择
<b>输出参数{out}</b>	
-	-
<b>返回值</b>	
-	-

例如：

```
/* set TIMER0 counter down direction */

timer_counter_down_direction (TIMER0);
```

#### 函数 **timer\_prescaler\_config**

函数timer\_prescaler\_config描述见下表：

**表 3-399. 函数 timer\_prescaler\_config**

函数名称	timer_prescaler_config
------	------------------------

<b>函数原型</b>	void timer_prescaler_config(uint32_t timer_periph, uint16_t prescaler, uint8_t pscreload);
<b>功能描述</b>	配置外设TIMERx预分频器
<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, 2, 5, 13..16)</i>	TIMER外设选择
<b>输入参数{in}</b>	
<b>prescaler</b>	预分频值, 0~65535
<b>输入参数{in}</b>	
<b>pscreload</b>	预分频值加载模式
<i>TIMER_PSC_RELOAD_NOW</i>	预分频值立即加载
<i>TIMER_PSC_RELOAD_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 3-400. 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>	
-	-

返回值	
-	-

例如：

```
/* configure TIMER0 repetition register value */

timer_repetition_value_config (TIMER0, 98);
```

#### 函数 timer\_autoreload\_value\_config

函数timer\_autoreload\_value\_config描述见下表：

表 3-401. 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 3-402. 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}	

<b>timer_periph</b>	TIMER外设
<i>TIMERx(x=0, 2, 5, 13..16)</i>	TIMER外设选择
<b>输入参数{in}</b>	
<b>counter</b>	计数器值 (0-65535)
<b>输出参数{out}</b>	
-	-
<b>返回值</b>	
-	-

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 3-403. 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	-
<b>输入参数{in}</b>	
<b>timer_periph</b>	TIMER外设
<i>TIMERx(x=0, 2, 5, 13..16)</i>	TIMER外设选择
<b>输出参数{out}</b>	
-	-
<b>返回值</b>	
<b>uint32_t</b>	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 3-404. 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	
uint16_t	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 3-405. 函数 timer\_single\_pulse\_mode\_config**

函数名称	timer_single_pulse_mode_config
函数原型	void timer_single_pulse_mode_config(uint32_t timer_periph, uint8_t spmode);
功能描述	配置外设TIMERx的单脉冲模式
先决条件	-
被调用函数	-
输入参数{in}	
timer_periph	TIMER外设
TIMERx(x=0, 2, 5, 14..16)	TIMER外设选择
输入参数{in}	
spmode	脉冲模式
TIMER_SP_MODE_SINGLE	单脉冲模式计数
TIMER_SP_MODE_REPETITIVE	重复模式计数
输出参数{out}	
-	-

返回值	
-	-

例如：

```
/* configure TIMER0 single pulse mode */

timer_single_pulse_mode_config (TIMER0, TIMER_SP_MODE_SINGLE);
```

### 函数 timer\_update\_source\_config

函数timer\_update\_source\_config描述见下表：

**表 3-406. 函数 timer\_update\_source\_config**

函数名称	timer_update_source_config
函数原型	void timer_update_source_config(uint32_t timer_periph, uint32_t update);
功能描述	配置外设TIMERx的更新源
先决条件	-
被调用函数	-
输入参数{in}	
timer_periph	TIMER外设
TIMERx(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 3-407. 函数 timer\_ocpre\_clear\_source\_config**

函数名称	timer_ocpre_clear_source_config
------	---------------------------------

<b>函数原型</b>	void timer_ocpre_clear_source_config (uint32_t timer_periph, uint8_t ocpclear);
<b>功能描述</b>	配置外设TIMERx的OCPRE清除源选择
<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, 2)</i>	TIMER外设选择
<b>输入参数{in}</b>	
<b>ocpreclear</b>	清除源
<i>TIMER_OCPRE_CLR_R</i>	OCPRE_CLR_INT连接到OCPRE_CLR输入
<i>TIMER_OCPRE_CLRF</i>	OCPRE_CLR_INT连接到ETIF
<b>输出参数{out}</b>	
-	-
<b>返回值</b>	
-	-

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 3-408. 函数 timer\_interrupt\_enable**

<b>函数名称</b>	timer_interrupt_enable
<b>函数原型</b>	void timer_interrupt_enable(uint32_t timer_periph, uint32_t interrupt);
<b>功能描述</b>	外设TIMERx中断使能
<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>interrupt</b>	中断源
<i>TIMER_INT_UP</i>	更新中断, TIMERx (x=0, 2, 5, 13..16)
<i>TIMER_INT_CH0</i>	通道0比较/捕获中断, TIMERx(x=0, 2, 13..16)
<i>TIMER_INT_CH1</i>	通道1比较/捕获中断, TIMERx(x=0, 2, 14)

<i>TIMER_INT_CH2</i>	通道2比较/捕获中断, TIMERx(x=0, 2)
<i>TIMER_INT_CH3</i>	通道3比较/捕获中断, TIMERx(x=0, 2)
<i>TIMER_INT_CMT</i>	换相更新中断, TIMERx (x=0, 14..16)
<i>TIMER_INT_TRG</i>	触发中断, TIMERx(x=0, 2, 14)
<i>TIMER_INT_BRK</i>	中止中断, TIMERx(x=0, 14..16)
<b>输出参数{out}</b>	
-	-
<b>返回值</b>	
-	-

例如:

```
/* enable the TIMER0 update interrupt */

timer_interrupt_enable (TIMER0, TIMER_INT_UP);
```

### 函数 **timer\_interrupt\_disable**

函数timer\_interrupt\_disable描述见下表:

**表 3-409. 函数 timer\_interrupt\_disable**

函数名称	timer_interrupt_disable
函数原型	void timer_interrupt_disable (uint32_t timer_periph, uint32_t interrupt);
功能描述	外设TIMERx中断禁能
先决条件	-
被调用函数	-
<b>输入参数{in}</b>	
<b>timer_periph</b>	TIMER外设
<i>TIMERx</i>	参考具体参数
<b>输入参数{in}</b>	
<b>interrupt</b>	中断源
<i>TIMER_INT_UP</i>	更新中断, TIMERx (x=0, 2, 5, 13..16)
<i>TIMER_INT_CH0</i>	通道0比较/捕获中断, TIMERx(x=0, 2, 13..16)
<i>TIMER_INT_CH1</i>	通道1比较/捕获中断, TIMERx(x=0, 2, 14)
<i>TIMER_INT_CH2</i>	通道2比较/捕获中断, TIMERx(x=0, 2)
<i>TIMER_INT_CH3</i>	通道3比较/捕获中断, TIMERx(x=0, 2)
<i>TIMER_INT_CMT</i>	换相更新中断, TIMERx (x=0, 14..16)
<i>TIMER_INT_TRG</i>	触发中断, TIMERx(x=0, 2, 14)
<i>TIMER_INT_BRK</i>	中止中断, TIMERx(x=0, 14..16)
<b>输出参数{out}</b>	
-	-
<b>返回值</b>	
-	-

例如:

```
/* disable the TIMER0 update interrupt */

timer_interrupt_disable (TIMER0, TIMER_INT_UP);
```

### 函数 timer\_interrupt\_flag\_get

函数timer\_interrupt\_flag\_get描述见下表:

表 3-410. 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 TIMER0 update interrupt flag */

FlagStatus Flag_interrupt = RESET;
```

```
Flag_interrupt = timer_interrupt_flag_get (TIMER0, TIMER_INT_FLAG_UP);
```

### 函数 `timer_interrupt_flag_clear`

函数`timer_interrupt_flag_clear`描述见下表：

**表 3-411. 函数 `timer_interrupt_flag_clear`**

函数名称	timer_interrupt_flag_clear
函数原型	<code>void timer_interrupt_flag_clear(uint32_t timer_periph, uint32_t interrupt);</code>
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}	
<code>timer_periph</code>	TIMER外设
<code>TIMERx</code>	参考具体参数
输入参数{in}	
<code>interrupt</code>	中断源
<code>TIMER_INT_FLAG_UP</code>	更新中断, TIMERx (x=0, 2, 5, 13..16)
<code>TIMER_INT_FLAG_CH0</code>	通道0比较/捕获中断, TIMERx(x=0, 2, 13..16)
<code>TIMER_INT_FLAG_CH1</code>	通道1比较/捕获中断, TIMERx(x=0, 2, 14)
<code>TIMER_INT_FLAG_CH2</code>	通道2比较/捕获中断, TIMERx(x=0, 2)
<code>TIMER_INT_FLAG_CH3</code>	通道3比较/捕获中断, TIMERx(x=0, 2)
<code>TIMER_INT_FLAG_CMT</code>	换相更新中断, TIMERx (x=0, 14..16)
<code>TIMER_INT_FLAG_TRG</code>	触发中断, TIMERx(x=0, 2, 14)
<code>TIMER_INT_FLAG_BRK</code>	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 3-412. 函数 timer\_flag\_get**

函数名称	timer_flag_get
函数原型	FlagStatus timer_flag_get(uint32_t timer_periph, uint32_t flag);
功能描述	获取外设TIMERx的状态标志
先决条件	-
被调用函数	-
输入参数{in}	
timer_periph	TIMER外设
TIMERx	参考具体参数
输入参数{in}	
flag	状态标志
TIMER_FLAG_UP	更新标志, TIMERx(x=0, 2, 5, 13..16)
TIMER_FLAG_CH0	通道0比较/捕获标志, TIMERx(x=0, 2, 13..16)
TIMER_FLAG_CH1	通道1比较/捕获标志, TIMERx(x=0, 2, 14)
TIMER_FLAG_CH2	通道2比较/捕获标志, TIMERx(x=0, 2)
TIMER_FLAG_CH3	通道3比较/捕获标志, TIMERx(x=0, 2)
TIMER_FLAG_CMT	通道换相更新标志, TIMERx(x=0, 14..16)
TIMER_FLAG_TRG	触发标志, TIMERx(x=0, 2, 14)
TIMER_FLAG_BRK	中止标志位, TIMERx(x=0, 14..16)
TIMER_FLAG_CH0_O	通道0捕获溢出标志, TIMERx(x=0, 2, 3..16)
TIMER_FLAG_CH1_O	通道1捕获溢出标志, TIMERx(x=0, 2, 14)
TIMER_FLAG_CH2_O	通道2捕获溢出标志, TIMERx(x=0, 2)
TIMER_FLAG_CH3_O	通道3捕获溢出标志, TIMERx(x=0, 2)
输出参数{out}	
-	-
返回值	
FlagStatus	SET或者RESET

例如:

```
/* get TIMER0 update flags */
FlagStatus Flag_status = RESET;
Flag_status = timer_flag_get (TIMER0, TIMER_FLAG_UP);
```

### 函数 **timer\_flag\_clear**

函数**timer\_flag\_clear**描述见下表：

**表 3-413. 函数 timer\_flag\_clear**

函数名称	timer_flag_clear
函数原型	void timer_flag_clear(uint32_t timer_periph, uint32_t flag);
功能描述	清除外设TIMERx状态标志
先决条件	-
被调用函数	-
<b>输入参数{in}</b>	
timer_periph	TIMER外设
TIMERx	参考具体参数
<b>输入参数{in}</b>	
flag	状态标志
TIMER_FLAG_UP	更新标志, TIMERx(x=0, 2, 5, 13..16)
TIMER_FLAG_CH0	通道0比较/捕获标志, TIMERx(x=0, 2, 13..16)
TIMER_FLAG_CH1	通道1比较/捕获标志, TIMERx(x=0, 2, 14)
TIMER_FLAG_CH2	通道2比较/捕获标志, TIMERx(x=0, 2)
TIMER_FLAG_CH3	通道3比较/捕获标志, TIMERx(x=0, 2)
TIMER_FLAG_CMT	通道换相更新标志, TIMERx(x=0, 14..16)
TIMER_FLAG_TRG	触发标志, TIMERx(x=0, 2, 14)
TIMER_FLAG_BRK	中止标志位, TIMERx(x=0, 14..16)
TIMER_FLAG_CH0_O	通道0捕获溢出标志, TIMERx(x=0, 2, 13..16)
TIMER_FLAG_CH1_O	通道1捕获溢出标志, TIMERx(x=0, 2, 14)
TIMER_FLAG_CH2_O	通道2捕获溢出标志, TIMERx(x=0, 2)
TIMER_FLAG_CH3_O	通道3捕获溢出标志, TIMERx(x=0, 2)
<b>输出参数{out}</b>	
-	-
<b>返回值</b>	
-	-

例如：

```
/* clear TIMER0 update flags */

timer_flag_clear (TIMER0, TIMER_FLAG_UP);
```

### 函数 **timer\_dma\_enable**

函数**timer\_dma\_enable**描述见下表：

表 3-414. 函数 timer\_dma\_enable

函数名称	timer_dma_enable
函数原型	void timer_dma_enable(uint32_t timer_periph, uint16_t dma);
功能描述	外设TIMERx的DMA使能
先决条件	-
被调用函数	-
输入参数{in}	
timer_periph	TIMER外设
TIMERx	参考具体参数
输入参数{in}	
dma	DMA源
TIMER_DMA_UPD	更新DMA请求, TIMERx(x=0, 2, 5, 14..16)
TIMER_DMA_CH0_D	通道0比较/捕获 DMA请求, TIMERx(x=0, 2, 14..16)
TIMER_DMA_CH1_D	通道1比较/捕获 DMA请求, TIMERx(x=0..2, 4)
TIMER_DMA_CH2_D	通道2比较/捕获 DMA请求, TIMERx(x=0, 2)
TIMER_DMA_CH3_D	通道3比较/捕获 DMA请求, TIMERx(x=0, 2)
TIMER_DMA_CMT_D	换相DMA更新请求, TIMERx(x=0, 14)
TIMER_DMA_TRG_D	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 3-415. 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}	
<b>timer_periph</b>	TIMER外设
<i>TIMERx</i>	参考具体参数
输入参数{in}	
<b>dma</b>	DMA源
<i>TIMER_DMA_UPD</i>	更新DMA请求, TIMERx(x=0, 2, 5, 14..16)
<i>TIMER_DMA_CH0_D</i>	通道0比较/捕获 DMA请求, TIMERx(x=0, 2, 14..16)
<i>TIMER_DMA_CH1_D</i>	通道1比较/捕获 DMA请求, TIMERx(x=0..2, 14)
<i>TIMER_DMA_CH2_D</i>	通道2比较/捕获 DMA请求, TIMERx(x=0, 2)
<i>TIMER_DMA_CH3_D</i>	通道3比较/捕获 DMA请求, TIMERx(x=0, 2)
<i>TIMER_DMA_CMT_D</i>	换相DMA更新请求, TIMERx(x=0, 14)
<i>TIMER_DMA_TRG_D</i>	触发DMA请求使能, TIMERx(x=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 3-416. 函数 timer\_channel\_dma\_request\_source\_select**

函数名称	timer_channel_dma_request_source_select
函数原型	void timer_channel_dma_request_source_select(uint32_t timer_periph, uint32_t dma_request);
功能描述	外设TIMERx的通道DMA请求源选择
先决条件	-
被调用函数	-
输入参数{in}	
<b>timer_periph</b>	TIMER外设
<i>TIMERx(x=0, 2, 14..16)</i>	TIMER外设选择
输入参数{in}	

<b>dma_request</b>	通道的DMA请求源选择
<i>TIMER_DMAREQUST_CHANNELEVENT</i>	当通道捕获/比较事件发生时，发送通道n的DMA请求
<i>TIMER_DMAREQUST_UPDATEEVENT</i>	当更新事件发生，发送通道n的DMA请求
<b>输出参数{out}</b>	
-	-
<b>返回值</b>	
-	-

例如：

```
/* TIMER0 channel DMA request of channel n is sent when channel y event occurs */

timer_channel_dma_request_source_select(TIMER0,
TMR_DMAREQUEST_CHANNELEVENT);
```

### 函数 **timer\_dma\_transfer\_config**

函数timer\_dma\_transfer\_config描述见下表：

**表 3-417. 函数 timer\_dma\_transfer\_config**

<b>函数名称</b>	timer_dma_transfer_config
<b>函数原型</b>	void timer_dma_transfer_config(uint32_t timer_periph, uint32_t dma_baseaddr, uint32_t dma_lenth);
<b>功能描述</b>	配置外设TIMERx的DMA模式
<b>先决条件</b>	-
<b>被调用函数</b>	-
<b>输入参数{in}</b>	
<b>timer_periph</b>	TIMER外设
<i>TIMERx(x=0, 2, 14..16)</i>	定时器外设选择
<b>输入参数{in}</b>	
<b>dma_baseaddr</b>	DMA传输起始地址
<i>TIMER_DMACFG_DMATA_CTL0</i>	DMA传输起始地址: TIMER_DMACFG_DMATA_CTL0, TIMERx(x=0, 2, 14..16)
<i>TIMER_DMACFG_DMATA_CTL1</i>	DMA传输起始地址: TIMER_DMACFG_DMATA_CTL1, TIMERx(x=0, 2, 14..16)
<i>TIMER_DMACFG_DMATA_SMCFG</i>	DMA传输起始地址: TIMER_DMACFG_DMATA_SMCFG, TIMERx(x=0, 2, 14)
<i>TIMER_DMACFG_DMATA_DMAINTEN</i>	DMA传输起始地址: TIMER_DMACFG_DMATA_DMAINTEN, TIMERx(x=0, 2, 14..16)

<i>TIMER_DMACFG_DMATA_INTF</i>	DMA传输起始地址: TIMER_DMACFG_DMATA_INTF, TIMERx(x=0, 2, 14..16)
<i>TIMER_DMACFG_DMATA_SWEVG</i>	DMA传输起始地址: TIMER_DMACFG_DMATA_SWEVG, TIMERx(x=0, 2, 14..16)
<i>TIMER_DMACFG_DMATA_CHCTL0</i>	DMA传输起始地址: TIMER_DMACFG_DMATA_CHCTL0, TIMERx(x=0, 2, 14..16)
<i>TIMER_DMACFG_DMATA_CHCTL1</i>	DMA传输起始地址: TIMER_DMACFG_DMATA_CHCTL1, TIMERx(x=0, 2)
<i>TIMER_DMACFG_DMATA_CHCTL2</i>	DMA传输起始地址: TIMER_DMACFG_DMATA_CHCTL2, TIMERx (x=0, 2, 14..16)
<i>TIMER_DMACFG_DMATA_CNT</i>	DMA传输起始地址: TIMER_DMACFG_DMATA_CNT, TIMERx (x=0, 2, 14..16)
<i>TIMER_DMACFG_DMATA_PSC</i>	DMA传输起始地址: TIMER_DMACFG_DMATA_PSC, TIMERx (x=0, 2, 14..16)
<i>TIMER_DMACFG_DMATA_CAR</i>	DMA传输起始地址: TIMER_DMACFG_DMATA_CAR, TIMERx (x=0, 2, 14..16)
<i>TIMER_DMACFG_DMATA_CREP</i>	DMA传输起始地址: TIMER_DMACFG_DMATA_CREP, TIMERx (x=0, 14..16)
<i>TIMER_DMACFG_DMATA_CH0CV</i>	DMA传输起始地址: TIMER_DMACFG_DMATA_CH0CV, TIMERx (x=0, 2, 14..16)
<i>TIMER_DMACFG_DMATA_CH1CV</i>	DMA传输起始地址: TIMER_DMACFG_DMATA_CH1CV, TIMERx(x=0, 2, 14)
<i>TIMER_DMACFG_DMATA_CH2CV</i>	DMA传输起始地址: TIMER_DMACFG_DMATA_CH2CV, TIMERx(x=0, 2)
<i>TIMER_DMACFG_DMATA_CH3CV</i>	DMA传输起始地址: TIMER_DMACFG_DMATA_CH3CV, TIMERx(x=0, 2)
<i>TIMER_DMACFG_DMATA_CCHP</i>	DMA传输起始地址: TIMER_DMACFG_DMATA_CCHP, TIMERx (x=0, 14..16)
<i>TIMER_DMACFG_DMATA_DMACFG</i>	DMA传输起始地址: TIMER_DMACFG_DMATA_DMACFG, TIMERx (x=0, 2, 14..16)
<b>输入参数{in}</b>	
<i>dma_lenth</i>	DMA传输长度
<i>TIMER_DMACFG_DMATC_xTRANSFER</i>	x=1..18, DMA传输 x 次
<b>输出参数{out}</b>	
-	-
<b>返回值</b>	
-	-

例如:

```
/* configure the TIMER0 DMA transfer */
```

```
timer_dma_transfer_config (TIMER0, TIMER_DMACFG_DMATA_CTL0,
    TIMER_DMACFG_DMATC_5TRANSFER);
```

### 函数 timer\_event\_software\_generate

函数timer\_event\_software\_generate描述见下表：

表 3-418. 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 3-419. 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3-384. 结构体 timer_break_parameter_struct</a>
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* initialize TIMER break parameter struct with a default value */

timer_break_parameter_struct timer_breakpara;

timer_break_struct_para_init(timer_breakpara);
```

### 函数 timer\_break\_config

函数timer\_break\_config描述见下表:

**表 3-420. 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3-384. 结构体 timer_break_parameter_struct</a>
输出参数{out}	
-	-

返回值	
-	-

例如：

```
/* configure TIMER0 break function */

timer_break_parameter_struct timer_breakpara;

timer_breakpara.runoffstate      = TIMER_ROS_STATE_DISABLE;
timer_breakpara.ideloffstate     = TIMER_IOS_STATE_DISABLE ;
timer_breakpara.deadtime         = 255;
timer_breakpara.breakpolarity    = TIMER_BREAK_POLARITY_LOW;
timer_breakpara.outputautostate = TIMER_OUTAUTO_ENABLE;
timer_breakpara.protectmode      = TIMER_CCHP_PROT_0;
timer_breakpara.breakstate       = TIMER_BREAK_ENABLE;
timer_break_config(TIMER0,&timer_breakpara);
```

### 函数 **timer\_break\_enable**

函数timer\_break\_enable描述见下表：

表 3-421. 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 3-422. 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
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 3-423. 函数 timer\_automatic\_output\_enable**

函数名称	timer_automatic_output_enable
函数原型	void timer_automatic_output_enable(uint32_t timer_periph);
功能描述	自动输出使能
先决条件	只有在TIMERx_CCHP寄存器的PROT [1:0] =00 时，才可修改
被调用函数	-
输入参数{in}	
timer_periph	TIMER外设
TIMERx(x=0, 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 3-424. 函数 timer\_automatic\_output\_disable**

函数名称	timer_automatic_output_disable
函数原型	void timer_automatic_output_disable (uint32_t timer_periph);
功能描述	自动输出禁能
先决条件	只有在TIMERx_CCHP寄存器的PROT [1:0] = 00时，才可修改
被调用函数	-
输入参数{in}	
timer_periph	TIMER外设
TIMERx(x=0, 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 3-425. 函数 timer\_primary\_output\_config**

函数名称	timer_primary_output_config
函数原型	void timer_primary_output_config(uint32_t timer_periph, ControlStatus newvalue);
功能描述	所有的通道输出使能
先决条件	-
被调用函数	-
输入参数{in}	
timer_periph	TIMER外设
TIMERx(x=0,14..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：

```
/* 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 3-426. 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 3-427. 函数 timer\_channel\_control\_shadow\_update\_config**

函数名称	timer_channel_control_shadow_update_config
函数原型	void timer_channel_control_shadow_update_config(uint32_t timer_periph,

	uint8_t ccuctl);
功能描述	通道换相控制影子寄存器更新控制
先决条件	-
被调用函数	-
<b>输入参数{in}</b>	
timer_periph	TIMER外设
TIMERx(x=0, 14..16)	TIMER外设选择
<b>输入参数{in}</b>	
ccuctl	通道换相控制影子寄存器更新控制
TIMER_UPDATECTL_CCU	CMTG位被置1时更新影子寄存器
TIMER_UPDATECTL_CCUTRI	当CMTG位被置1或检测到TRIGI上升沿时，影子寄存器更新
<b>输出参数{out}</b>	
-	-
<b>返回值</b>	
-	-

例如：

```
/* configure TIMER0 channel control shadow register update when CMTG bit is set */
timer_channel_control_shadow_update_config (TIMER0, TIMER_UPDATECTL_CCU);
```

#### 函数 timer\_channel\_output\_struct\_para\_init

函数timer\_channel\_output\_struct\_para\_init描述见下表：

表 3-428. 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	-
<b>输入参数{in}</b>	
ocpara	输出通道结构体，详见 <a href="#">表3-385. 结构体timer_oc_parameter_struct</a>
<b>输出参数{out}</b>	
-	-
<b>返回值</b>	
-	-

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 3-429. 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3-385. 结构体timer_oc_parameter_struct</a>
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* configure TIMER0 channel 0 output function */

timer_oc_parameter_struct timer_ocintpara;

timer_ocintpara.outputstate = TIMER_CCX_ENABLE;
timer_ocintpara.outputnstate = TIMER_CCXN_ENABLE;
timer_ocintpara.ocpolarity = TIMER_OC_POLARITY_HIGH;
timer_ocintpara.ocnpolarity = TIMER_OCN_POLARITY_HIGH;
timer_ocintpara.ocidlestate = TIMER_OC_IDLE_STATE_HIGH;
timer_ocintpara.ocnidlestate = TIMER_OCN_IDLE_STATE_LOW;
```

```
timer_channel_output_config(TIMER0, TIMER_CH_0, &timer_ocintpara);
```

### 函数 timer\_channel\_output\_mode\_config

函数timer\_channel\_output\_mode\_config描述见下表：

**表 3-430. 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}	
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}	
ocmode	通道输出比较模式
TIMER_OC_MODE_TIMING	冻结模式
TIMER_OC_MODE_ACTIVE	匹配时设置为高
TIMER_OC_MODE_INACTIVE	匹配时设置为低
TIMER_OC_MODE_TOGGLE	匹配时翻转
TIMER_OC_MODE_LOW	强制为低
TIMER_OC_MODE_HIGH	强制为高
TIMER_OC_MODE_PWM0	PWM模式0
TIMER_OC_MODE_PWM1	PWM模式1
输出参数{out}	
-	-
返回值	

-	-
---	---

例如：

```
/* configure TIMER0 channel PWM 0 mode */

timer_channel_output_mode_config(TIMER0, TIMER_CH_0, TIMER_OC_MODE_PWM0);
```

#### 函数 **timer\_channel\_output\_pulse\_value\_config**

函数timer\_channel\_output\_pulse\_value\_config描述见下表：

**表 3-431. 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	-
<b>输入参数{in}</b>	
timer_periph	TIMER外设
TIMERx	参考具体参数
<b>输入参数{in}</b>	
channel	待配置通道
TIMER_CH_0	通道0, TIMERx (x=0, 2, 13..16)
TIMER_CH_1	通道1, TIMERx 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)
<b>输入参数{in}</b>	
pulse	通道输出比较值 (0~65535)
<b>输出参数{out}</b>	
-	-
<b>返回值</b>	
-	-

例如：

```
/* configure TIMER0 channel 0 output pulse value */

timer_channel_output_pulse_value_config(TIMER0, TIMER_CH_0, 399);
```

#### 函数 **timer\_channel\_output\_shadow\_config**

函数timer\_channel\_output\_shadow\_config描述见下表：

**表 3-432. 函数 timer\_channel\_output\_shadow\_config**

函数名称	timer_channel_output_shadow_config
------	------------------------------------

<b>函数原型</b>	void timer_channel_output_shadow_config(uint32_t timer_periph, uint16_t channel, uint16_t ocshadow);
<b>功能描述</b>	配置TIMERx通道输出比较影子寄存器功能
<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
<b>TIMERx</b>	参考具体参数
<b>输入参数{in}</b>	
<b>channel</b>	待配置通道
<b>TIMER_CH_0</b>	通道0, TIMERx (x=0, 2, 13..16)
<b>TIMER_CH_1</b>	通道1, TIMERx (x=0, 2, 14)
<b>TIMER_CH_2</b>	通道2, TIMERx (x=0, 2)
<b>TIMER_CH_3</b>	通道3, TIMERx (x=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
<b>TIMER_OC_SHAD_OW_ENABLE</b>	使能输出比较影子寄存器
<b>TIMER_OC_SHAD_OW_DISABLE</b>	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 3-433. 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TIMERx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>	TIMER外设

<i>TIMERx</i>	参考具体参数
<b>输入参数{in}</b>	
<b>channel</b>	待配置通道
<i>TIMER_CH_0</i>	通道0, TIMERx (x=0, 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)
<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 3-434. 函数 timer\_channel\_output\_clear\_config**

函数名称	timer_channel_output_clear_config
函数原型	void timer_channel_output_clear_config(uint32_t timer_periph, uint16_t channel, uint16_t occlear);
功能描述	配置TIMERx的通道输出比较清0功能
先决条件	-
被调用函数	-
<b>输入参数{in}</b>	
<b>timer_periph</b>	TIMER外设
<i>TIMERx</i>	参考具体参数
<b>输入参数{in}</b>	
<b>channel</b>	待配置通道
<i>TIMER_CH_0</i>	通道0, TIMERx (x=0, 2)
<i>TIMER_CH_1</i>	通道1, TIMERx (x=0, 2)
<i>TIMER_CH_2</i>	通道2, TIMERx (x=0, 2)
<i>TIMER_CH_3</i>	通道3, TIMERx (x=0, 2)
<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>	
-	-
<b>返回值</b>	
-	-

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 3-435. 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	-
<b>输入参数{in}</b>	
<b>timer_periph</b>	TIMER外设
<i>TIMERx</i>	参考具体参数
<b>输入参数{in}</b>	
<b>channel</b>	待配置通道
<i>TIMER_CH_0</i>	通道0, TIMERx (x=0, 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)
<b>输入参数{in}</b>	
<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
<b>输出参数{out}</b>	
-	-

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 3-436. 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 3-437. 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_ENAB LE	通道使能
TIMER_CCX_DISA BLE	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 3-438. 函数 timer\_channel\_complementary\_output\_state\_config**

函数名称	timer_channel_complementary_output_state_config
函数原型	void timer_channel_complementary_output_state_config(uint32_t timer_periph, uint16_t channel, uint16_t ocnstate);
功能描述	配置互补通道输出状态

先决条件	-
被调用函数	-
<b>输入参数{in}</b>	
<b>timer_periph</b>	TIMER外设
<b>TIMERx</b>	参考具体参数
<b>输入参数{in}</b>	
<b>channel</b>	待配置通道
<b>TIMER_CH_0</b>	通道0, TIMERx (x=0, 14..16)
<b>TIMER_CH_1</b>	通道1, TIMERx (x=0)
<b>TIMER_CH_2</b>	通道2, TIMERx (x=0)
<b>输入参数{in}</b>	
<b>state</b>	互补通道状态
<b>TIMER_CCXN_ENA BLE</b>	互补通道使能
<b>TIMER_CCXN_DIS ABLE</b>	互补通道禁能
<b>输出参数{out}</b>	
-	-
<b>返回值</b>	
-	-

例如：

```
/* configure TIMER0 channel 0 complementary output enable state */

timer_channel_complementary_output_state_config (TIMER0, TIMER_CH_0,
TIMER_CCXN_ENABLE);
```

#### 函数 **timer\_channel\_input\_struct\_para\_init**

函数**timer\_channel\_input\_struct\_para\_init**描述见下表：

**表 3-439. 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	-
<b>输入参数{in}</b>	
<b>icpara</b>	通道输入结构体, 详见 <a href="#">表3-386. 结构体timer_ic_parameter_struct</a> 。
<b>输出参数{out}</b>	
-	-
<b>返回值</b>	
-	-

例如：

```
/* initialize TIMER channel input parameter struct with a default value */

timer_ic_parameter_struct timer_icinitpara;

timer_channel_input_struct_para_init(&timer_icinitpara);
```

### 函数 timer\_input\_capture\_config

函数timer\_input\_capture\_config描述见下表：

**表 3-440. 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3-386. 结构体timer_ic_parameter_struct</a> 。
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* configure TIMER0 input capture parameter */

timer_ic_parameter_struct timer_icinitpara;

timer_icinitpara.icpolarity = TIMER_IC_POLARITY_RISING;
timer_icinitpara.icselection = TIMER_IC_SELECTION_DIRECTTI;
timer_icinitpara.icprescaler = TIMER_IC_PSC_DIV1;
timer_icinitpara.icfilter = 0x0;
```

```
timer_input_capture_config(TIMER0, TIMER_CH_0, &timer_icinitpara);
```

### 函数 timer\_channel\_input\_capture\_prescaler\_config

函数timer\_channel\_input\_capture\_prescaler\_config描述见下表:

**表 3-441. 函数 timer\_channel\_input\_capture\_prescaler\_config**

函数名称	timer_channel_input_capture_prescaler_config
函数原型	void timer_channel_input_capture_prescaler_config(uint32_t timer_periph, uint16_t channel, uint16_t prescaler);
功能描述	配置TIMERx通道输入捕获预分频值
先决条件	-
被调用函数	-
<b>输入参数{in}</b>	
timer_periph	TIMER外设
TIMERx	参考具体参数
<b>输入参数{in}</b>	
channel	待配置通道
TIMER_CH_0	通道0, TIMERx (x=0, 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)
<b>输入参数{in}</b>	
prescaler	通道输入捕获预分频值
TIMER_IC_PSC_DI V1	不分频
TIMER_IC_PSC_DI V2	2分频
TIMER_IC_PSC_DI V4	4分频
TIMER_IC_PSC_DI V8	8分频
<b>输出参数{out}</b>	
-	-
<b>返回值</b>	
-	-

例如:

```
/* configure TIMER0 channel 0 input capture prescaler value */

timer_channel_input_capture_prescaler_config (TIMER0, TIMER_CH_0,
TICKER_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 3-442. 函数 `timer_channel_capture_value_register_read`**

函数名称	timer_channel_capture_value_register_read
函数原型	<code>uint32_t timer_channel_capture_value_register_read(uint32_t timer_periph,                   uint16_t channel);</code>
功能描述	读取通道捕获值
先决条件	-
被调用函数	-
<b>输入参数{in}</b>	
<code>timer_periph</code>	TIMER外设
<code>TIMERx</code>	参考具体参数
<b>输入参数{in}</b>	
<code>channel</code>	待配置通道
<code>TIMER_CH_0</code>	通道0, TIMERx (x=0, 2, 13..16)
<code>TIMER_CH_1</code>	通道1, TIMERx (x=0, 2, 14)
<code>TIMER_CH_2</code>	通道2, TIMERx (x=0, 2)
<code>TIMER_CH_3</code>	通道3, TIMERx (x=0, 2)
<b>输出参数{out}</b>	
-	-
<b>返回值</b>	
<code>uint32_t</code>	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 3-443. 函数 `timer_input_pwm_capture_config`**

函数名称	timer_input_pwm_capture_config
函数原型	<code>void timer_input_pwm_capture_config(uint32_t timer_periph, uint16_t channel,                   timer_ic_parameter_struct* icpwm);</code>
功能描述	配置TIMERx捕获PWM输入参数
先决条件	-
被调用函数	<code>timer_channel_input_capture_prescaler_config</code>
<b>输入参数{in}</b>	
<code>timer_periph</code>	TIMER外设

<i>TIMERx(x=0, 2, 14)</i>	TIMER外设选择
输入参数{in}	
<b>channel</b>	待配置通道
<i>TIMER_CH_0</i>	通道0
<i>TIMER_CH_1</i>	通道1
输入参数{in}	
<b>icpwm</b>	输入捕获结构体, 详见 <a href="#">表3-386. 结构体timer_ic_parameter_struct</a>
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* configure TIMER0 input pwm capture parameter */
timer_ic_parameter_struct timer_icinitpara;
timer_icinitpara.icpolarity = TIMER_IC_POLARITY_RISING;
timer_icinitpara.icselection = TIMER_IC_SELECTION_DIRECTTI;
timer_icinitpara.icprescaler = TIMER_IC_PSC_DIV1;
timer_icinitpara.icfilter = 0x0;
timer_input_pwm_capture_config (TIMER0, TIMER_CH_0, &timer_icinitpara);
```

### 函数 timer\_hall\_mode\_config

函数timer\_hall\_mode\_config描述见下表:

表 3-444. 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}	
<b>timer_periph</b>	TIMER外设
<i>TIMERx(x=0, 2)</i>	TIMER外设选择
输入参数{in}	
<b>hallmode</b>	HALL接口功能状态
<i>TIMER_HALLINTERRUPT_ENABLE</i>	HALL接口使能
<i>TIMER_HALLINTERRUPT_DISABLE</i>	HALL接口禁能

输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* configure TIMER0 hall sensor mode */

timer_hall_mode_config (TIMER0, TIMER_HALLINTERFACE_ENABLE);
```

### 函数 timer\_input\_trigger\_source\_select

函数timer\_input\_trigger\_source\_select描述见下表：

**表 3-445. 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, 14))
TIMER_SMCFG_T_RGSEL_ITI3	内部触发输入3(ITI3, TIMERx(x=0, 2, 14))
TIMER_SMCFG_T_RGSEL_CI0F_ED	CI0的边沿标志位 (CI0F_ED, TIMERx(x=0, 2, 14))
TIMER_SMCFG_T_RGSEL_CI0FE0	滤波后的通道0输入 (CI0FE0, 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_ITIO);
```

#### 函数 timer\_master\_output\_trigger\_source\_select

函数timer\_master\_output\_trigger\_source\_select描述见下表：

表 3-446. 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 3-447. 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_ENCODER_MODE0	编码器模式0, TIMERx(x=0, 2)
TIMER_ENCODER_MODE1	编码器模式1, TIMERx(x=0, 2)
TIMER_ENCODER_MODE2	编码器模式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_ENCODER_MODE0);
```

#### 函数 timer\_master\_slave\_mode\_config

函数timer\_master\_slave\_mode\_config描述见下表：

表 3-448. 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_ENA_BLE	主从模式使能
TIMER_MASTER_SLAVE_MODE_DISA_BLE	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 3-449. 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	-
<b>输入参数{in}</b>	
timer_periph	TIMER外设
TIMERx(x=0, 2)	TIMER外设选择
<b>输入参数{in}</b>	
extprescaler	外部触发预分频
TIMER_EXT_TRI_P SC_OFF	不分频
TIMER_EXT_TRI_P SC_DIV2	2分频
TIMER_EXT_TRI_P SC_DIV4	4分频
TIMER_EXT_TRI_P SC_DIV8	8分频
<b>输入参数{in}</b>	
expolarity	外部触发输入极性
TIMER_ETP_FALLING	低电平或者下降沿有效
TIMER_ETP_RISING	高电平或者上升沿有效
<b>输入参数{in}</b>	
extfilter	外部触发滤波控制 (0~15)
<b>输出参数{out}</b>	
-	-
<b>返回值</b>	
-	-

例如：

```
/* configure TIMER0 external trigger input */

timer_external_trigger_config (TIMER0, TIMER_EXT_TRI_PSC_DIV2,
    TIMER_ETP_FALLING, 10);
```

#### 函数 timer\_quadrature\_decoder\_mode\_config

函数timer\_quadrature\_decoder\_mode\_config描述见下表：

**表 3-450. 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);

<b>功能描述</b>	TIMERx配置为编码器模式
<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
<b>TIMERx(x=0, 2)</b>	TIMER外设选择
<b>输入参数{in}</b>	
<b>decomode</b>	编码器模式
<b>TIMER_ENCODER_MODE0</b>	根据CI0FE0的电平，计数器在CI1FE1的边沿向上/下计数
<b>TIMER_ENCODER_MODE1</b>	根据CI1FE1的电平，计数器在CI0FE0的边沿向上/下计数
<b>TIMER_ENCODER_MODE2</b>	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
<b>TIMER_IC_POLARITY_RISING</b>	捕获上升边沿
<b>TIMER_IC_POLARITY_FALLING</b>	捕获下降边沿
<b>输入参数{in}</b>	
<b>ic1polarity</b>	IC1极性
<b>TIMER_IC_POLARITY_RISING</b>	捕获上升边沿
<b>TIMER_IC_POLARITY_FALLING</b>	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_ENCODER_MODE0,
TIMER_IC_POLARITY_RISING, TIMER_IC_POLARITY_RISING);
```

### 函数 **timer\_internal\_clock\_config**

函数timer\_internal\_clock\_config描述见下表：

**表 3-451. 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);

功能描述	TIMERx配置为内部时钟模式
先决条件	-
被调用函数	-
输入参数{in}	
timer_periph	TIMER外设
TIMERx(x=0, 2, 14)	TIMER外设选择
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* configure TIMER0 internal clock mode */

timer_internal_clock_config(TIMER0);
```

#### 函数 **timer\_internal\_trigger\_as\_external\_clock\_config**

函数timer\_internal\_trigger\_as\_external\_clock\_config描述见下表：

表 3-452. 函数 **timer\_internal\_trigger\_as\_external\_clock\_config**

函数名称	timer_internal_trigger_as_external_clock_config
函数原型	void timer_internal_trigger_as_external_clock_config(uint32_t timer_periph, uint32_t intrigger);
功能描述	配置TIMERx的内部触发为时钟源
先决条件	-
被调用函数	timer_input_trigger_source_select
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	被选择的内部触发源
TIMER_SMCFG_T RGSEL_ITI0	选择内部触发0 (ITI0)为时钟源, TIMERx(x=0, 2, 14)
TIMER_SMCFG_T RGSEL_ITI1	选择内部触发1 (ITI1)为时钟源, TIMERx(x=0, 2, 14)
TIMER_SMCFG_T RGSEL_ITI2	选择内部触发2 (ITI2)为时钟源, TIMERx(x=0, 2)
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* configure TIMER0 the internal trigger ITI0 as external clock input */

timer_internal_trigger_as_external_clock_config (TIMER0, TIMER_SMCFG_TRGSEL_ITI0);
```

### 函数 timer\_external\_trigger\_as\_external\_clock\_config

函数timer\_external\_trigger\_as\_external\_clock\_config描述见下表：

表 3-453. 函数 timer\_external\_trigger\_as\_external\_clock\_config

函数名称	timer_external_trigger_as_external_clock_config
函数原型	void timer_external_trigger_as_external_clock_config(uint32_t timer_periph, uint32_t exttrigger, uint16_t expolarity, uint32_t extfilter);
功能描述	配置TIMERx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}	
exttrigger	外部触发源
TIMER_SMCFG_T RGSEL_CI0F_ED	CI0的边沿标志(CI0F_ED)
TIMER_SMCFG_T RGSEL_CI0FE0	滤波后的通道0输入(CI0FE0)
TIMER_SMCFG_T RGSEL_CI1FE1	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_POLARI TY_RISING	外部触发源高电平或者上升沿有效
TIMER_IC_POLARI TY_FALLING	外部触发源低电平或者下降沿有效
TIMER_IC_POLARI TY_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 CI0FE0 as external clock input */

timer_external_trigger_as_external_clock_config (TIMER0,
```

```
TIMER_SMCFG_TRGSEL_CI0FE0, TIMER_IC_POLARITY_RISING, 0);
```

### 函数 timer\_external\_clock\_mode0\_config

函数timer\_external\_clock\_mode0\_config描述见下表:

**表 3-454. 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_FALLING	下降沿或者低电平有效
TIMER_ETP_RISING	上升沿或者高电平有效
输入参数{in}	
extfilter	ETI触发源滤波参数(0~15)
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* configure TIMER0 the external clock mode0 */
timer_external_clock_mode0_config (TIMER0, TIMER_EXT_TRI_PSC_DIV2,
TMR_ETP_FALLING, 0);
```

### 函数 `timer_external_clock_mode1_config`

函数`timer_external_clock_mode1_config`描述见下表:

**表 3-455. 函数 `timer_external_clock_mode1_config`**

函数名称	<code>timer_external_clock_mode1_config</code>
函数原型	<code>void timer_external_clock_mode1_config(uint32_t timer_periph, uint32_t extprescaler, uint32_t expolarity, uint32_t extfilter);</code>
功能描述	配置TIMERx外部时钟模式1
先决条件	-
被调用函数	<code>timer_external_trigger_config</code>
<b>输入参数{in}</b>	
<code>timer_periph</code>	TIMER外设
<code>TIMERx(x=0, 2)</code>	TIMER外设选择
<b>输入参数{in}</b>	
<code>extprescaler</code>	ETI触发源预分频值
<code>TIMER_EXT_TRI_P SC_OFF</code>	不分频
<code>TIMER_EXT_TRI_P SC_DIV2</code>	2分频
<code>TIMER_EXT_TRI_P SC_DIV4</code>	4分频
<code>TIMER_EXT_TRI_P SC_DIV8</code>	8分频
<b>输入参数{in}</b>	
<code>expolarity</code>	ETI触发源极性
<code>TIMER_ETP_FALLING</code>	下降沿或者低电平有效
<code>TIMER_ETP_RISING</code>	上升沿或者高电平有效
<b>输入参数{in}</b>	
<code>extfilter</code>	ETI触发源滤波参数 (0~15)
<b>输出参数{out}</b>	
-	-
<b>返回值</b>	
-	-

例如:

```
/* configure TIMER0 the external clock mode1 */

timer_external_clock_mode1_config (TIMER0, TIMER_EXT_TRI_PSC_DIV2,
TICKER_ETP_FALLING, 0);
```

### 函数 `timer_external_clock_mode1_disable`

函数`timer_external_clock_mode1_disable`描述见下表：

**表 3-456. 函数 `timer_external_clock_mode1_disable`**

函数名称	timer_external_clock_mode1_disable
函数原型	<code>void timer_external_clock_mode1_disable(uint32_t timer_periph);</code>
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}	
<code>timer_periph</code>	TIMER外设
<code>TIMERx(x=0, 2)</code>	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 3-457. 函数 `timer_channel_remap_config`**

函数名称	timer_channel_remap_config
函数原型	<code>void timer_channel_remap_config (uint32_t timer_periph, uint32_t remap);</code>
功能描述	配置TIMERxt通道重映射功能
先决条件	-
被调用函数	-
输入参数{in}	
<code>timer_periph</code>	TIMER外设
<code>TIMERx(x=13)</code>	TIMER外设选择
输入参数{in}	
<code>remap</code>	重映射功能选择
<code>TIMER13_C10_RMP_GPIO</code>	通道0连接到GPIO
<code>TIMER13_C10_RMP_RTCCLK</code>	通道0连接到RTCCLK
<code>TIMER13_C10_RMP_HXTAL_DIV32</code>	通道0连接到HXTAL/32
<code>TIMER13_C10_RMP</code>	通道0连接到CKOUTSEL

<u>_CKOUTSEL</u>	
<b>输出参数{out}</b>	
-	-
<b>返回值</b>	
-	-

例如：

```
/* configure TIMER13 channel 0 input is connected to GPIO */

timer_channel_remap_config (TIMER13, TIMER13_CI0_RMP_GPIO);
```

#### 函数 **timer\_write\_chxval\_register\_config**

函数timer\_write\_chxval\_register\_config描述见下表：

**表 3-458. 函数 timer\_write\_chxval\_register\_config**

函数名称	timer_write_chxval_register_config
函数原型	void timer_write_chxval_register_config(uint32_t timer_periph, uint16_t ccsel);
功能描述	配置TIMERx写CHxVAL选择位
先决条件	-
被调用函数	-
<b>输入参数{in}</b>	
<b>timer_periph</b>	TIMER外设
<i>TIMERx(x=0, 2, 13..16)</i>	TIMER外设选择
<b>输入参数{in}</b>	
<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。
<b>输出参数{out}</b>	
-	-
<b>返回值</b>	
-	-

例如：

```
/* configure TIMER0 write CHxVAL register selection */

timer_write_chxval_register_config(TIMER0, TIMER_CHVSEL_ENABLE);
```

#### 函数 **timer\_output\_value\_selection\_config**

函数timer\_output\_value\_selection\_config描述见下表：

表 3-459. 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_D ISABLE	无影响
TIMER_OUTSEL_E NABLE	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);
```

## 3.19. USART

通用同步异步收发器(USART)提供了一个灵活方便的串行数据交换接口，章节[3.19.1](#)描述了USART的寄存器列表，章节[3.19.2](#)对USART库函数进行说明。

### 3.19.1. 外设寄存器说明

USART寄存器列表如下表所示：

表 3-460. 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

寄存器名称	寄存器描述
USART_RT	接收超时寄存器
USART_CMD	请求寄存器
USART_STAT	状态寄存器
USART_INTC	中断标志清除寄存器
USART_RDATA	数据接收寄存器
USART_TDATA	数据发送寄存器
USART_CHC	兼容性控制寄存器
USART_RFCS	接收FIFO控制和状态寄存器

### 3.19.2. 外设库函数说明

USART库函数列表如下表所示：

**表 3-461. USART 库函数**

库函数名称	库函数描述
uart_deinit	复位外设USART
uart_baudrate_set	配置USART波特率
uart_parity_config	配置USART奇偶校验
uart_word_length_set	配置USART字长
uart_stop_bit_set	配置USART停止位
uart_enable	使能USART
uart_disable	失能USART
uart_transmit_config	USART发送配置
uart_receive_config	USART接收配置
uart_data_first_config	配置数据传输时低位在前或高位在前
uart_invert_config	配置USART反转功能
uart_overrun_enable	使能USART溢出禁止功能
uart_overrun_disable	失能USART溢出禁止功能
uart_oversample_config	配置USART过采样模式
uart_sample_bit_config	配置USART单次采样方式
uart_receiver_timeout_enable	使能USART接收超时
uart_receiver_timeout_disable	失能USART接收超时
uart_receiver_timeout_threshold_config	设置USART接收超时阈值
uart_data_transmit	USART发送数据功能
uart_data_receive	USART接收数据功能
uart_autobaud_detection_enable	使能USART自动波特率检测
uart_autobaud_detection_disable	失能USART自动波特率检测
uart_autobaud_detection_mode_config	配置USART自动波特率检测模式
uart_address_config	在地址掩码唤醒模式下配置USART地址
uart_address_detection_mode_config	配置USART地址检测模式

库函数名称	库函数描述
g	
uart_mute_mode_enable	使能USART静默模式
uart_mute_mode_disable	失能USART静默模式
uart_mute_mode_wakeup_config	配置USART静默模式唤醒方式
uart_lin_mode_enable	使能USART LIN模式
uart_lin_mode_disable	失能USART LIN模式
uart_lin_break_detection_length_config	配置USART LIN模式中断帧长度
uart_halfduplex_enable	使能USART半双工模式
uart_halfduplex_disable	失能USART半双工模式
uart_clock_enable	使能USART CK引脚
uart_clock_disable	失能USART CK引脚
uart_synchronous_clock_config	配置USART同步通讯模式参数
uart_guard_time_config	在USART智能卡模式下配置保护时间值
uart_smartcard_mode_enable	使能USART智能卡模式
uart_smartcard_mode_disable	失能USART智能卡模式
uart_smartcard_mode_nack_enable	在USART智能卡模式下使能NACK
uart_smartcard_mode_nack_disable	在USART智能卡模式下失能NACK
uart_smartcard_mode_early_nack_enable	使能USART智能卡模式提前NACK
uart_smartcard_mode_early_nack_disable	失能USART智能卡模式提前NACK
uart_smartcard_autoretry_config	配置智能卡自动重试次数
uart_block_length_config	配置智能卡T=1的接收时块的长度
uart_irda_mode_enable	使能USART串行红外编解码功能模块
uart_irda_mode_disable	失能USART串行红外编解码功能模块
uart_prescaler_config	在USART IrDA低功耗模式下配置外设时钟分频系数
uart_irda_lowpower_config	配置USART IrDA低功耗模式
uart_hardware_flow_rts_config	配置USART RTS硬件控制流
uart_hardware_flow_cts_config	配置USART CTS硬件控制流
uart_hardware_flow_coherence_config	配置硬件流控兼容模式
uart_rs485_driver_enable	使能USART rs485驱动
uart_rs485_driver_disable	失能USART rs485驱动
uart_driver_assertime_config	配置USART驱动使能置位时间
uart_driver_deassertime_config	配置USART驱动使能置低时间
uart_depolarity_config	配置USART驱动使能极性模式
uart_dma_receive_config	配置USART DMA接收
uart_dma_transmit_config	配置USART DMA发送
uart_reception_error_dma_disable	USART接收错误时禁能DMA
uart_reception_error_dma_enable	USART接收错误时使能DMA

库函数名称	库函数描述
uart_wakeup_enable	使能USART唤醒
uart_wakeup_disable	失能USART唤醒
uart_wakeup_mode_config	配置USART唤醒模式
uart_receive_fifo_enable	使能接收FIFO
uart_receive_fifo_disable	失能接收FIFO
uart_receive_fifo_counter_number	读取接收FIFO计数器的值
uart_flag_get	得到STAT/RFCs寄存器中的标志
uart_flag_clear	清除USART状态
uart_interrupt_enable	使能USART中断
uart_interrupt_disable	失能USART中断
uart_command_enable	使能USART请求
uart_interrupt_flag_get	得到USART中断和标志状态
uart_interrupt_flag_clear	清除USART中断标志位

### 枚举类型 `uart_flag_enum`

表 3-462. 枚举类型 `uart_flag_enum`

成员名称	功能描述
USART_FLAG_REA	接收使能通知标志
USART_FLAG_TEA	发送使能通知标志
USART_FLAG_WU	从深度睡眠模式唤醒标志
USART_FLAG_RWU	接收器从静默模式唤醒
USART_FLAG_SB	断开信号发送标志
USART_FLAG_AM	地址匹配标志
USART_FLAG_BSY	忙标志
USART_FLAG_ABD	自动波特率检测标志
USART_FLAG_ABDE	自动波特率检测错误
USART_FLAG_EB	块结束标志
USART_FLAG_RT	接收超时标志
USART_FLAG_CTS	CTS电平
USART_FLAG_CTSF	CTS变化标志
USART_FLAG_LBD	LIN断开检测标志
USART_FLAG_TBE	发送数据寄存器空
USART_FLAG_TC	发送完成
USART_FLAG_RBNE	读数据缓冲区非空
USART_FLAG_IDLE	空闲线检测标志
USART_FLAG_ORERR	溢出错误
USART_FLAG_NERR	噪声错误标志
USART_FLAG_FERR	帧错误
USART_FLAG_PERR	校验错误
USART_FLAG_EPERR	校验错误超前检测标志
USART_FLAG_RFFINT	接收FIFO满中断标志

成员名称	功能描述
USART_FLAG_RFF	接收FIFO满标志
USART_FLAG_RFE	接收FIFO空标志

### 枚举类型 `usart_interrupt_flag_enum`

表 3-463. 枚举类型 `usart_interrupt_flag_enum`

成员名称	功能描述
USART_INT_FLAG_EB	块结束中断标志
USART_INT_FLAG_RT	接收超时中断标志
USART_INT_FLAG_AM	地址匹配中断标志
USART_INT_FLAG_PERR	奇偶校验错误中断标志
USART_INT_FLAG_TBE	发送寄存器空中断标志
USART_INT_FLAG_TC	发送完成中断标志
USART_INT_FLAG_RBNE	读缓冲区非空中断标志
USART_INT_FLAG_RBNE_ORE RR	读缓冲区非空和溢出中断标志
USART_INT_FLAG_IDLE	空闲线检测中断标志
USART_INT_FLAG_LBD	LIN断开检测中断标志
USART_INT_FLAG_WU	从深度睡眠模式唤醒中断标志
USART_INT_FLAG_CTS	CTS中断标志
USART_INT_FLAG_ERR_NERR	噪声错误中断标志
USART_INT_FLAG_ERR_ORER R	溢出错误中断标志
USART_INT_FLAG_ERR_FERR	帧错误中断标志
USART_INT_FLAG_RFF	接收FIFO满中断标志

### 枚举类型 `usart_interrupt_enum`

表 3-464. 枚举类型 `usart_interrupt_enum`

成员名称	功能描述
USART_INT_EB	块结束中断使能
USART_INT_RT	接收超时中断使能
USART_INT_AM	地址匹配中断使能
USART_INT_PERR	奇偶校验错误中断使能
USART_INT_TBE	发送寄存器空中断使能
USART_INT_TC	发送完成中断使能
USART_INT_RBNE	读缓冲区非空中断和溢出错误中断使能
USART_INT_IDLE	空闲线检测中断使能
USART_INT_LBD	LIN断开检测中断使能
USART_INT_WU	从深度睡眠模式唤醒中断使能
USART_INT_CTS	CTS中断使能
USART_INT_ERR	错误中断使能

成员名称	功能描述
USART_INT_RFF	接收FIFO满中断使能

### 枚举类型 **usart\_invert\_enum**

**表 3-465. 枚举类型 usart\_invert\_enum**

成员名称	功能描述
USART_DINV_ENABLE	数据位反转
USART_DINV_DISABLE	数据位不反转
USART_TXPIN_ENABLE	TX管脚电平反转
USART_TXPIN_DISABLE	TX管脚电平不反转
USART_RXPIN_ENABLE	RX管脚电平反转
USART_RXPIN_DISABLE	RX管脚电平不反转
USART_SWAP_ENABLE	交换TX/RX管脚
USART_SWAP_DISABLE	不交换TX/RX管脚

### 函数 **usart\_deinit**

函数usart\_deinit描述见下表：

**表 3-466. 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 3-467. 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
<b>输入参数{in}</b>	
usart_periph	外设USARTTx
USARTx	x=0,1
<b>输入参数{in}</b>	
baudval	波特率值
<b>输出参数{out}</b>	
-	-
<b>返回值</b>	
-	-

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 3-468. 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	-
<b>输入参数{in}</b>	
usart_periph	外设USARTTx
USARTx	x=0,1
<b>输入参数{in}</b>	
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
<b>输出参数{out}</b>	
-	-
<b>返回值</b>	
-	-

例如：

```
/* configure USART0 parity */
```

```
uart_parity_config(USART0, USART_PM EVEN);
```

### 函数 **uart\_word\_length\_set**

函数**uart\_word\_length\_set**描述见下表:

**表 3-469. 函数 **uart\_word\_length\_set****

函数名称	uart_word_length_set
函数原型	void usart_word_length_set(uint32_t usart_periph, uint32_t wlen);
功能描述	配置USART字长
先决条件	-
被调用函数	-
<b>输入参数{in}</b>	
<b>usart_periph</b>	外设USARTx
<b>USARTx</b>	x=0,1
<b>输入参数{in}</b>	
<b>wlen</b>	配置USART字长
<b>USART_WL_8BIT</b>	8 bits
<b>USART_WL_9BIT</b>	9 bits
<b>输出参数{out}</b>	
-	-
<b>返回值</b>	
-	-

例如:

```
/* configure USART0 word length */
uart_word_length_set(USART0, USART_WL_9BIT);
```

### 函数 **uart\_stop\_bit\_set**

函数**uart\_stop\_bit\_set**描述见下表:

**表 3-470. 函数 **uart\_stop\_bit\_set****

函数名称	uart_stop_bit_set
函数原型	void usart_stop_bit_set(uint32_t usart_periph, uint32_t strlen);
功能描述	配置USART停止位
先决条件	-
被调用函数	-
<b>输入参数{in}</b>	
<b>usart_periph</b>	外设USARTx
<b>USARTx</b>	x=0,1
<b>输入参数{in}</b>	
<b>strlen</b>	配置USART停止位
<b>USART_STB_1BIT</b>	1 bit

<i>USART_STB_0_5BIT</i>	0.5 bit
<i>USART_STB_2BIT</i>	2 bit
<i>USART_STB_1_5BIT</i>	1.5 bit
<b>输出参数{out}</b>	
-	-
<b>返回值</b>	
-	-

例如：

```
/* configure USART0 stop bit length */

uart_stop_bit_set(USART0, USART_STB_1_5BIT);
```

### 函数 **uart\_enable**

函数uart\_enable描述见下表：

**表 3-471. 函数 **uart\_enable****

函数名称	uart_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	-
<b>输入参数{in}</b>	
<b>usart_periph</b>	外设USARTx
<b>USARTx</b>	x=0,1
<b>输出参数{out}</b>	
-	-
<b>返回值</b>	
-	-

例如：

```
/* enable USART0 */

uart_enable(USART0);
```

### 函数 **uart\_disable**

函数uart\_disable描述见下表：

**表 3-472. 函数 **uart\_disable****

函数名称	uart_disable
函数原型	void usart_disable(uint32_t usart_periph);

功能描述	失能USART
先决条件	-
被调用函数	-
<b>输入参数{in}</b>	
<b>usart_periph</b>	外设USARTx
<b>USARTx</b>	x=0,1
<b>输出参数{out}</b>	
-	-
<b>返回值</b>	
-	-

例如：

```
/* disable USART0 */

uart_disable(USART0);
```

### 函数 **uart\_transmit\_config**

函数**uart\_transmit\_config**描述见下表：

**表 3-473. 函数 **uart\_transmit\_config****

函数名称	<b>uart_transmit_config</b>
函数原型	void <b>uart_transmit_config</b> (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	-
<b>输入参数{in}</b>	
<b>usart_periph</b>	外设USARTx
<b>USARTx</b>	x=0,1
<b>输入参数{in}</b>	
<b>txconfig</b>	使能/失能USART发送器
<b>USART_TRANSMIT_ENABLE</b>	使能USART发送
<b>USART_TRANSMIT_DISABLE</b>	失能USART发送
<b>输出参数{out}</b>	
-	-
<b>返回值</b>	
-	-

例如：

```
/* configure USART0 transmitter */

uart_transmit_config(USART0,USART_TRANSMIT_ENABLE);
```

### 函数 **uart\_receive\_config**

函数uart\_receive\_config描述见下表：

**表 3-474. 函数 **uart\_receive\_config****

函数名称	uart_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	-
<b>输入参数{in}</b>	
usart_periph	外设USARTx
USARTx	x=0,1
<b>输入参数{in}</b>	
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
<b>输出参数{out}</b>	
-	-
<b>返回值</b>	
-	-

例如：

```
/* configure USART0 receiver */
usart_receive_config(USART0, USART_RECEIVE_ENABLE);
```

### 函数 **uart\_data\_first\_config**

函数uart\_data\_first\_config描述见下表：

**表 3-475. 函数 **uart\_data\_first\_config****

函数名称	uart_data_first_config
函数原型	void usart_data_first_config(uint32_t usart_periph, uint32_t msbf);
功能描述	配置数据传输时低位在前或高位在前
先决条件	-
被调用函数	-
<b>输入参数{in}</b>	
usart_periph	外设USARTx
USARTx	x=0,1
<b>输入参数{in}</b>	
msbf	数据传输时低位在前/高位在前
USART_MSBF_LS	数据传输时低位在前

<i>B</i>	
<i>USART_MSBF_MS</i> <i>B</i>	数据传输时高位在前
<b>输出参数{out}</b>	
-	-
<b>返回值</b>	
-	-

例如：

```
/* configure LSB of data first */
uart_data_first_config(USART0, USART_MSBF_LSB);
```

### 函数 **uart\_invert\_config**

函数uart\_invert\_config描述见下表：

**表 3-476. 函数 **uart\_invert\_config****

函数名称	uart_invert_config
函数原型	void usart_invert_config(uint32_t usart_periph, usart_invert_enum invertpara);
功能描述	配置USART反转功能
先决条件	-
被调用函数	-
<b>输入参数{in}</b>	
<b>usart_periph</b>	外设USARTx
<b>USARTx</b>	x=0,1
<b>输入参数{in}</b>	
<b>invertpara</b>	参考 <a href="#">表3-465. 枚举类型usart_invert_enum</a>
<i>USART_DINV_ENA</i> <i>BLE</i>	数据位电平反转
<i>USART_DINV_DIS</i> <i>ABLE</i>	数据位电平不反转
<i>USART_TXPIN_EN</i> <i>ABLE</i>	TX引脚电平反转
<i>USART_TXPIN_DIS</i> <i>ABLE</i>	TX引脚电平不反转
<i>USART_RXPIN_EN</i> <i>ABLE</i>	RX引脚电平反转
<i>USART_RXPIN_DI</i> <i>SABLE</i>	RX引脚电平不反转
<i>USART_SWAP_EN</i> <i>ABLE</i>	TX和RX管脚功能被交换
<i>USART_SWAP_DIS</i> <i>ABLE</i>	TX和RX管脚功能不被交换

输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* configure USART0 inversion */
uart_invert_config(USART0, USART_DINV_ENABLE);
```

#### 函数 **uart\_overrun\_enable**

函数uart\_overrun\_enable描述见下表：

**表 3-477. 函数 **uart\_overrun\_enable****

函数名称	uart_overrun_enable
函数原型	void uart_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 */
uart_overrun_enable (USART0);
```

#### 函数 **uart\_overrun\_disable**

函数uart\_overrun\_disable描述见下表：

**表 3-478. 函数 **uart\_overrun\_disable****

函数名称	uart_overrun_disable
函数原型	void uart_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

<i>USARTx</i>	x=0,1
<b>输出参数{out}</b>	
-	-
<b>返回值</b>	
-	-

例如：

```
/* disable USART0 overrun */

uart_overrun_disable (USART0);
```

#### 函数 **uart\_oversample\_config**

函数uart\_oversample\_config描述见下表：

**表 3-479. 函数 **uart\_oversample\_config****

函数名称	uart_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	-
<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>oversamp</b>	过采样值
<i>USART_OVSMOD_8</i>	8倍过采样
<i>USART_OVSMOD_16</i>	16倍过采样
<b>输出参数{out}</b>	
-	-
<b>返回值</b>	
-	-

例如：

```
/* config USART0 oversampling by 8 */

uart_oversample_config(USART0,USART_OVSMOD_8);
```

#### 函数 **uart\_sample\_bit\_config**

函数uart\_sample\_bit\_config描述见下表：

**表 3-480. 函数 usart\_sample\_bit\_config**

函数名称	usart_sample_bit_config
函数原型	void usart_sample_bit_config(uint32_t usart_periph,uint32_t osb);
功能描述	配置USART单次采样方式
先决条件	-
被调用函数	-
<b>输入参数{in}</b>	
<b>usart_periph</b>	外设USARTx
<b>USARTx</b>	x=0,1
<b>输入参数{in}</b>	
<b>osb</b>	单次采样方式
<b>USART_OSB_1BIT</b>	1次采样方法
<b>USART_OSB_3BIT</b>	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 3-481. 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	-
<b>输入参数{in}</b>	
<b>usart_periph</b>	外设USARTx
<b>USARTx</b>	x=0
<b>输出参数{out}</b>	
-	-
<b>返回值</b>	
-	-

例如：

```
/* enable USART0 receiver timeout */
```

```
uart_receiver_timeout_enable(USART0);
```

### 函数 **uart\_receiver\_timeout\_disable**

函数uart\_receiver\_timeout\_disable描述见下表:

**表 3-482. 函数 **uart\_receiver\_timeout\_disable****

函数名称	uart_receiver_timeout_disable
函数原型	void uart_receiver_timeout_disable(uint32_t usart_periph);
功能描述	失能USART接收超时
先决条件	-
被调用函数	-
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 */
uart_receiver_timeout_disable(USART0);
```

### 函数 **uart\_receiver\_timeout\_threshold\_config**

函数uart\_receiver\_timeout\_threshold\_config描述见下表:

**表 3-483. 函数 **uart\_receiver\_timeout\_threshold\_config****

函数名称	uart_receiver_timeout_threshold_config
函数原型	void uart_receiver_timeout_threshold_config(uint32_t usart_periph, uint32_t rtimeout);
功能描述	设置USART接收超时阈值
先决条件	-
被调用函数	-
输入参数{in}	
usart_periph	外设USARTx
USARTx	x=0
输入参数{in}	
rtimeout	超时时间 (0x00000000-0x00FFFFFF)
输出参数{out}	
-	-
返回值	

-	-
---	---

例如：

```
/* set the receiver timeout threshold of USART0*/
uart_receiver_timeout_threshold_config(USART0,115200*3);
```

### **函数 `uart_data_transmit`**

函数`uart_data_transmit`描述见下表：

**表 3-484. 函数 `uart_data_transmit`**

函数名称	uart_data_transmit
函数原型	void usart_data_transmit(uint32_t usart_periph, uint32_t data);
功能描述	USART发送数据功能
先决条件	-
被调用函数	-
<b>输入参数{in}</b>	
<b>usart_periph</b>	外设USARTx
<b>USARTx</b>	x=0,1
<b>输入参数{in}</b>	
<b>data</b>	发送的数据 (0-0xFF)
<b>输出参数{out}</b>	
-	-
<b>返回值</b>	
-	-

例如：

```
/* USART0 transmit data */

uart_data_transmit(USART0, 0xAA);
```

### **函数 `uart_data_receive`**

函数`uart_data_receive`描述见下表：

**表 3-485. 函数 `uart_data_receive`**

函数名称	uart_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	-
<b>输入参数{in}</b>	
<b>usart_periph</b>	外设USARTx
<b>USARTx</b>	x=0,1

输出参数{out}	
-	-
返回值	
<b>uint32_t</b>	接收到的数据 (0-0xFF)

例如：

```
/* USART0 receive data */

uint16_t temp;

temp = usart_data_receive(USART0);
```

### 函数 **usart\_autobaud\_detection\_enable**

函数usart\_autobaud\_detection\_enable描述见下表：

**表 3-486. 函数 usart\_autobaud\_detection\_enable**

函数名称	usart_autobaud_detection_enable
函数原型	void usart_autobaud_detection_enable(uint32_t usart_periph);
功能描述	使能USART自动波特率检测
先决条件	-
被调用函数	-
输入参数{in}	
<b>usart_periph</b>	外设USARTx
<b>USARTx</b>	x=0
输出参数{out}	
-	-
返回值	

例如：

```
/* enable USART0 auto baud rate detection */

usart_autobaud_detection_enable(USART0);
```

### 函数 **usart\_autobaud\_detection\_disable**

函数usart\_autobaud\_detection\_disable描述见下表：

**表 3-487. 函数 usart\_autobaud\_detection\_disable**

函数名称	usart_autobaud_detection_disable
函数原型	void usart_autobaud_detection_disable(uint32_t usart_periph);
功能描述	失能USART自动波特率检测
先决条件	-
被调用函数	-
输入参数{in}	

<b>usart_periph</b>	外设USARTx
<i>USARTx</i>	x=0
<b>输出参数{out}</b>	
-	-
<b>返回值</b>	

例如：

```
/* disable USART0 auto baud rate detection */
uart_autobaud_detection_disable(USART0);
```

#### 函数 **uart\_autobaud\_detection\_mode\_config**

函数uart\_autobaud\_detection\_mode\_config描述见下表：

**表 3-488. 函数 **uart\_autobaud\_detection\_mode\_config****

函数名称	uart_autobaud_detection_mode_config
函数原型	void uart_autobaud_detection_mode_config(uint32_t usart_periph, uint32_t abdmmod);
功能描述	USART自动波特率检测模式配置
先决条件	-
被调用函数	-
<b>输入参数{in}</b>	
<b>usart_periph</b>	外设USARTx
<i>USARTx</i>	x=0
<b>输入参数{in}</b>	
<i>abdmmod</i>	自动波特率检测模式
<i>USART_ABDM_FT OR</i>	下降沿到上升沿的测量
<i>USART_ABDM_FT OF</i>	下降沿对下降沿的测量
<b>输出参数{out}</b>	
-	-
<b>返回值</b>	

例如：

```
/* configure USART0 auto baud rate detection mode */
uart_autobaud_detection_mode_config(USART0, USART_ABDM_FTOR);
```

#### 函数 **uart\_address\_config**

函数uart\_address\_config描述见下表：

**表 3-489. 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	-
<b>输入参数{in}</b>	
usart_periph	外设USARTx
USARTx	x=0,1
<b>输入参数{in}</b>	
addr	USART地址 (0-0xFF)
<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 3-490. 函数 usart\_address\_detection\_mode\_config**

函数名称	usart_address_detection_mode_config
函数原型	void usart_address_detection_mode_config(uint32_t usart_periph, uint32_t addmod);
功能描述	配置USART地址检测模式
先决条件	-
被调用函数	-
<b>输入参数{in}</b>	
usart_periph	外设USARTx
USARTx	x=0,1
<b>输入参数{in}</b>	
addmod	地址检测模式
USART_ADDM_4BIT	4位地址检测
USART_ADDM_FULLBIT	全位地址检测
<b>输出参数{out}</b>	
-	-
<b>返回值</b>	

-	-
---	---

例如：

```
/*configure address detection mode */

uart_address_config(USART0, USART_ADDM_4BIT);
```

#### 函数 **uart\_mute\_mode\_enable**

函数uart\_mute\_mode\_enable描述见下表：

**表 3-491. 函数 **uart\_mute\_mode\_enable****

函数名称	uart_mute_mode_enable
函数原型	void usart_mute_mode_enable(uint32_t usart_periph);
功能描述	使能USART静默模式
先决条件	-
被调用函数	-
<b>输入参数{in}</b>	
usart_periph	外设USARTx
USARTx	x=0,1
<b>输出参数{out}</b>	
-	-
<b>返回值</b>	
-	-

例如：

```
/* enable USART0 receiver in mute mode */

uart_mute_mode_enable(USART0);
```

#### 函数 **uart\_mute\_mode\_disable**

函数uart\_mute\_mode\_disable描述见下表：

**表 3-492. 函数 **uart\_mute\_mode\_disable****

函数名称	uart_mute_mode_disable
函数原型	void usart_mute_mode_disable(uint32_t usart_periph);
功能描述	失能USART静默模式
先决条件	-
被调用函数	-
<b>输入参数{in}</b>	
usart_periph	外设USARTx
USARTx	x=0,1
-	-
<b>返回值</b>	

-	-
---	---

例如：

```
/* disable USART0 receiver in mute mode */
```

```
uart_mute_mode_disable(USART0);
```

#### **函数 `uart_mute_mode_wakeup_config`**

函数`uart_mute_mode_wakeup_config`描述见下表：

**表 3-493. 函数 `uart_mute_mode_wakeup_config`**

函数名称	uart_mute_mode_wakeup_config
函数原型	void usart_mute_mode_wakeup_config(uint32_t usart_periph, uint32_t wmethod);
功能描述	配置USART静默模式唤醒方式
先决条件	-
被调用函数	-
<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>wmethod</b>	两种方法用于进入或退出静默模式
<i>USART_WM_IDLE</i>	空闲线唤醒
<i>USART_WM_ADDR</i>	地址掩码唤醒
<b>输出参数{out}</b>	
-	-
<b>返回值</b>	
-	-

例如：

```
/* configure USART0 wakeup method in mute mode */
```

```
uart_mute_mode_wakeup_config(USART0, USART_WM_IDLE);
```

#### **函数 `uart_lin_mode_enable`**

函数`uart_lin_mode_enable`描述见下表：

**表 3-494. 函数 `uart_lin_mode_enable`**

函数名称	uart_lin_mode_enable
函数原型	void usart_lin_mode_enable(uint32_t usart_periph);
功能描述	使能USART LIN模式
先决条件	-
被调用函数	-

输入参数{in}	
<b>usart_periph</b>	外设USARTx
<b>USARTx</b>	x=0
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* USART0 LIN mode enable */

uart_lin_mode_enable(USART0);
```

#### 函数 **uart\_lin\_mode\_disable**

函数uart\_lin\_mode\_disable描述见下表：

**表 3-495. 函数 **uart\_lin\_mode\_disable****

函数名称	uart_lin_mode_disable
函数原型	void uart_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}	
<b>usart_periph</b>	外设USARTx
<b>USARTx</b>	x=0
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* USART0 LIN mode disable */

uart_lin_mode_disable(USART0);
```

#### 函数 **uart\_lin\_break\_dection\_length\_config**

函数uart\_lin\_break\_dection\_length\_config描述见下表：

**表 3-496. 函数 **uart\_lin\_break\_dection\_length\_config****

函数名称	uart_lin_break_dection_length_config
函数原型	void uart_lin_break_dection_length_config(uint32_t usart_periph, uint32_t lrlen);
功能描述	配置USART LIN模式中断帧长度

先决条件	-
被调用函数	-
<b>输入参数{in}</b>	
<b>usart_periph</b>	外设USARTx
<b>USARTx</b>	x=0
<b>输入参数{in}</b>	
<b>Irlen</b>	LIN模式中断帧长度
<b>USART_LBLEN_10B</b>	断开帧长度为10 bits
<b>USART_LBLEN_11B</b>	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 */

uart_lin_break_detection_length_config(USART0, USART_LBLEN_10B);
```

#### 函数 **uart\_halfduplex\_enable**

函数uart\_halfduplex\_enable描述见下表：

**表 3-497. 函数 **uart\_halfduplex\_enable****

函数名称	uart_halfduplex_enable
函数原型	void usart_halfduplex_enable(uint32_t usart_periph);
功能描述	使能USART半双工模式
先决条件	-
被调用函数	-
<b>输入参数{in}</b>	
<b>usart_periph</b>	外设USARTx
<b>USARTx</b>	x=0,1
<b>输出参数{out}</b>	
-	-
<b>返回值</b>	
-	-

例如：

```
/* enable USART0 half duplex mode */

uart_halfduplex_enable(USART0);
```

### 函数 `usart_halfduplex_disable`

函数`usart_halfduplex_disable`描述见下表：

**表 3-498. 函数 `usart_halfduplex_disable`**

函数名称	usart_halfduplex_disable
函数原型	<code>void usart_halfduplex_disable(uint32_t usart_periph);</code>
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}	
<code>usart_periph</code>	外设USARTx
<code>USARTx</code>	<code>x=0,1</code>
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 3-499. 函数 `usart_clock_enable`**

函数名称	usart_clock_enable
函数原型	<code>void usart_clock_enable(uint32_t usart_periph);</code>
功能描述	使能USART CK引脚
先决条件	-
被调用函数	-
输入参数{in}	
<code>usart_periph</code>	外设USARTx
<code>USARTx</code>	<code>x=0, 1</code>
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 3-500. 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	-
<b>输入参数{in}</b>	
<b>usart_periph</b>	外设USARTx
<b>USARTx</b>	x=0, 1
<b>输出参数{out}</b>	
-	-
<b>返回值</b>	
-	-

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 3-501. 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	-
<b>输入参数{in}</b>	
<b>usart_periph</b>	外设USARTx
<b>USARTx</b>	x=0, 1
<b>输入参数{in}</b>	
<b>clen</b>	CK信号长度
<b>USART_CLEN_NO_NE</b>	8位数据帧中有7个CK脉冲，9位数据帧中有8个CK脉冲
<b>USART_CLEN_EN</b>	8位数据帧中有8个CK脉冲，9位数据帧中有9个CK脉冲
<b>输入参数{in}</b>	
<b>cph</b>	时钟相位
<b>USART_CPH_1CK</b>	在首个时钟边沿采样第一个数据

<code>USART_CPH_2CK</code>	在第二个时钟边沿采样第一个数据
<b>输入参数{in}</b>	
<code>cpl</code>	时钟极性
<code>USART_CPL_LOW</code>	CK引脚不对外发送时保持为低电平
<code>USART_CPL_HIGH</code>	CK引脚不对外发送时保持为高电平
<b>输出参数{out}</b>	
-	-
<b>返回值</b>	
-	-

例如：

```
/* configure USART0 synchronous mode parameters */

uart_synchronous_clock_config(USART0,USART_CLEN_EN,USART_CPH_2CK,
USART_CPL_HIGH);
```

### 函数 `uart_guard_time_config`

函数`uart_guard_time_config`描述见下表：

表 3-502. 函数 `uart_guard_time_config`

函数名称	<code>uart_guard_time_config</code>
函数原型	<code>void uart_guard_time_config(uint32_t usart_periph,uint32_t guat);</code>
功能描述	在USART智能卡模式下配置保护时间值
先决条件	-
被调用函数	-
<b>输入参数{in}</b>	
<code>usart_periph</code>	外设USARTx
<code>USARTx</code>	x=0
<b>输入参数{in}</b>	
<code>guat</code>	保护时间值 (0-0x000000FF)
<b>输出参数{out}</b>	
-	-
<b>返回值</b>	
-	-

例如：

```
/* configure USART0 guard time value in smartcard mode */

uart_guard_time_config(USART0, 0x0000 0055);
```

### 函数 `uart_smartcard_mode_enable`

函数`uart_smartcard_mode_enable`描述见下表：

**表 3-503. 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 3-504. 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 3-505. 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 3-506. 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 3-507. 函数 usart\_smartcard\_mode\_early\_nack\_enable**

函数名称	usart_smartcard_mode_early_nack_enable
函数原型	void usart_smartcard_mode_early_nack_enable (uint32_t usart_periph);
功能描述	使能USART智能卡模式提前NACK
先决条件	-
被调用函数	-
输入参数{in}	
usart_periph	外设USARTTx
USARTTx	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 3-508. 函数 usart\_smartcard\_mode\_early\_nack\_disable**

函数名称	usart_smartcard_mode_early_nack_disable
函数原型	void usart_smartcard_mode_early_nack_disable (uint32_t usart_periph);
功能描述	失能USART智能卡模式提前NACK
先决条件	-
被调用函数	-
输入参数{in}	
usart_periph	外设USARTTx
USARTTx	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 3-509. 函数 `uart_smartcard_autoretry_config`

函数名称	<code>uart_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>	<code>x=0</code>
输入参数{in}	
<code>scrtnum</code>	智能卡自动重试次数 ( <code>0-0x00000007</code> )
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* configure smartcard auto-retry number */

uart_smartcard_autoretry_config (USART0, 0x00000007);
```

### 函数 `uart_block_length_config`

函数`uart_block_length_config`描述见下表：

表 3-510. 函数 `uart_block_length_config`

函数名称	<code>uart_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>	<code>x=0</code>
输入参数{in}	
<code>bl</code>	块长度 ( <code>0-0x000000FF</code> )
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* configure block length in Smartcard T=1 reception */
```

```
uart_block_length_config(USART0, 0x000000FF);
```

### 函数 usart\_irda\_mode\_enable

函数usart\_irda\_mode\_enable描述见下表:

**表 3-511. 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 3-512. 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 */
```

```
uart_irda_mode_disable(USART0);
```

### 函数 **uart\_prescaler\_config**

函数**uart\_prescaler\_config**描述见下表:

**表 3-513. 函数 **uart\_prescaler\_config****

函数名称	uart_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	-
<b>输入参数{in}</b>	
usart_periph	外设USARTx
USARTx	x=0
<b>输入参数{in}</b>	
psc	时钟分频系数 (0x00-0xFF)
<b>输出参数{out}</b>	
-	-
<b>返回值</b>	
-	-

例如:

```
/* configure the USART0 peripheral clock prescaler in USART IrDA low-power mode */
uart_prescaler_config(USART0, 0x00);
```

### 函数 **uart\_irda\_lowpower\_config**

函数**uart\_irda\_lowpower\_config**描述见下表:

**表 3-514. 函数 **uart\_irda\_lowpower\_config****

函数名称	uart_irda_lowpower_config
函数原型	void usart_irda_lowpower_config(uint32_t usart_periph, uint32_t irlp);
功能描述	配置USART IrDA低功耗模式
先决条件	-
被调用函数	-
<b>输入参数{in}</b>	
usart_periph	外设USARTx
USARTx	x=0

输入参数{in}	
<b>irlp</b>	IrDA低功耗模式或正常模式
<b>USART_IRLP_LOW</b>	低功耗模式
<b>USART_IRLP_NORMAL</b>	正常模式
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* configure USART0 IrDA low-power */
uart_irda_lowpower_config(USART0, USART_IRLP_LOW);
```

#### 函数 **uart\_hardware\_flow\_rts\_config**

函数uart\_hardware\_flow\_rts\_config描述见下表：

表 3-515. 函数 **uart\_hardware\_flow\_rts\_config**

函数名称	uart_hardware_flow_rts_config
函数原型	void uart_hardware_flow_rts_config(uint32_t usart_periph, uint32_t rtsconfig);
功能描述	配置USART RTS硬件控制流
先决条件	-
被调用函数	-
输入参数{in}	
<b>usart_periph</b>	外设USARTx
<b>USARTx</b>	x=0,1
输入参数{in}	
<b>rtsconfig</b>	使能/失能RTS
<b>USART_RTS_ENA_BLE</b>	使能RTS
<b>USART_RTS_DISA_BLE</b>	失能RTS
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* configure USART0 hardware flow control RTS */
uart_hardware_flow_rts_config(USART0, USART_RTS_ENABLE);
```

### 函数 **uart\_hardware\_flow\_cts\_config**

函数uart\_hardware\_flow\_cts\_config描述见下表：

**表 3-516. 函数 **uart\_hardware\_flow\_cts\_config****

函数名称	uart_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 */
uart_hardware_flow_cts_config(USART0, USART_CTS_ENABLE);
```

### 函数 **uart\_hardware\_flow\_coherence\_config**

函数uart\_hardware\_flow\_coherence\_config描述见下表：

**表 3-517. 函数 **uart\_hardware\_flow\_coherence\_config****

函数名称	uart_hardware_flow_coherence_config
函数原型	void usart_hardware_flow_coherence_config(uint32_t usart_periph, uint32_t hcm);
功能描述	配置硬件流控兼容模式
先决条件	-
被调用函数	-
输入参数{in}	

<b>usart_periph</b>	外设USARTx
<b>USARTx</b>	x=0,1
<b>输入参数{in}</b>	
<b>hcm</b>	硬件流控制兼容模式
<b>USART_HCM_NONE</b>	nRTS信号与USART_STAT0寄存器中RBNE位相同
<b>USART_HCM_EN</b>	nRTS信号在最后一个数据位被采样后被置位
<b>输出参数{out}</b>	
-	-
<b>返回值</b>	
-	-

例如：

```
/* configure hardware flow control coherence mode */

uart_hardware_flow_coherence_config(USART0, USART_HCM_NONE);
```

### 函数 **uart\_rs485\_driver\_enable**

函数uart\_rs485\_driver\_enable描述见下表：

**表 3-518. 函数 **uart\_rs485\_driver\_enable****

函数名称	uart_rs485_driver_enable
函数原型	void uart_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	-
<b>输入参数{in}</b>	
<b>usart_periph</b>	外设USARTx
<b>USARTx</b>	x=0,1
<b>输出参数{out}</b>	
-	-
<b>返回值</b>	
-	-

例如：

```
/* enable USART0 RS485 driver */

uart_rs485_driver_enable(USART0);
```

### 函数 **uart\_rs485\_driver\_disable**

函数uart\_rs485\_driver\_disable描述见下表：

表 3-519. 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 3-520. 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}	
deatime	驱动使能置位时间 (0-0x0000001F)
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* set USART0 driver assertime */
usart_driver_assertime_config(USART0,0x0000001F);
```

### 函数 `uart_driver_deassertime_config`

函数`uart_driver_deassertime_config`描述见下表:

**表 3-521. 函数 `uart_driver_deassertime_config`**

函数名称	uart_driver_deassertime_config
函数原型	<code>void usart_driver_deassertime_config(uint32_t usart_periph, uint32_t dedtime);</code>
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}	
<code>usart_periph</code>	外设USARTx
<code>USARTx</code>	x=0,1
输入参数{in}	
<code>dedtime</code>	驱动使能置低时间 (0-0x0000001F)
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* set USART0 driver deassertime */
uart_driver_deassertime_config(USART0,0x0000001F);
```

### 函数 `uart_depolarity_config`

函数`uart_depolarity_config`描述见下表:

**表 3-522. 函数 `uart_depolarity_config`**

函数名称	uart_depolarity_config
函数原型	<code>void usart_depolarity_config(uint32_t usart_periph, uint32_t dep);</code>
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}	
<code>usart_periph</code>	外设USARTx
<code>USARTx</code>	x=0,1
输入参数{in}	
<code>dep</code>	驱动使能的极性选择模式
<code>USART_DEP_HIGH</code>	DE信号高有效
<code>USART_DEP_LOW</code>	DE信号低有效
输出参数{out}	
-	-
返回值	

-	-
---	---

例如：

```
/* configure driver enable polarity mode */

uart_driver_depolarity_config(USART0, USART_DEP_HIGH);
```

### 函数 **uart\_dma\_receive\_config**

函数uart\_dma\_receive\_config描述见下表：

**表 3-523. 函数 **uart\_dma\_receive\_config****

函数名称	uart_dma_receive_config
函数原型	void usart_dma_receive_config(uint32_t usart_periph, uint32_t dmacmd);
功能描述	配置USART DMA接收功能
先决条件	-
被调用函数	-
<b>输入参数{in}</b>	
<b>usart_periph</b>	外设USARTx
<b>USARTx</b>	x=0,1
<b>输入参数{in}</b>	
<b>dmacmd</b>	DMA使能/失能DMA接收功能
<b>USART_DENR_ENABLE</b>	使能DMA接收功能
<b>USART_DENR_DISABLE</b>	失能DMA接收功能
<b>输出参数{out}</b>	
-	-
<b>返回值</b>	
-	-

例如：

```
/* USART0 DMA enable for reception */

uart_dma_receive_config(USART0, USART_DENR_ENABLE);
```

### 函数 **uart\_dma\_transmit\_config**

函数uart\_dma\_transmit\_config描述见下表：

**表 3-524. 函数 **uart\_dma\_transmit\_config****

函数名称	uart_dma_transmit_config
函数原型	void usart_dma_transmit_config(uint32_t usart_periph, uint32_t dmacmd);
功能描述	配置 USART DMA发送功能
先决条件	-

被调用函数	-
<b>输入参数{in}</b>	
<b>usart_periph</b>	外设USARTx
<b>USARTx</b>	x=0,1
<b>输入参数{in}</b>	
<b>dmacmd</b>	使能/失能DMA发送功能
<b>USART_DENT_EN_ABLE</b>	使能DMA发送功能
<b>USART_DENT_DIS_ABLE</b>	失能DMA发送功能
<b>输出参数{out}</b>	
-	-
<b>返回值</b>	
-	-

例如：

```
/* USART0 DMA enable for transmission */

uart_dma_transmit_config(USART0, USART_DENT_ENABLE);
```

#### 函数 **usart\_reception\_error\_dma\_disable**

函数usart\_reception\_error\_dma\_disable描述见下表：

**表 3-525. 函数 usart\_reception\_error\_dma\_disable**

函数名称	usart_reception_error_dma_disable
函数原型	void usart_reception_error_dma_disable (uint32_t usart_periph);
功能描述	USART接收错误时失能DMA
先决条件	-
被调用函数	-
<b>输入参数{in}</b>	
<b>usart_periph</b>	外设USARTx
<b>USARTx</b>	x=0,1
<b>输出参数{out}</b>	
-	-
<b>返回值</b>	
-	-

例如：

```
/* disable DMA on reception error */

uart_reception_error_dma_disable (USART0);
```

### 函数 **usart\_reception\_error\_dma\_enable**

函数usart\_reception\_error\_dma\_enable描述见下表：

**表 3-526. 函数 usart\_reception\_error\_dma\_enable**

函数名称	usart_reception_error_dma_enable
函数原型	void usart_reception_error_dma_enable(uint32_t usart_periph);
功能描述	USART接收错误时使能DMA
先决条件	-
被调用函数	-
输入参数{in}	
usart_periph	外设USARTx
USARTx	x=0,1
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* enable DMA on reception error */
usart_reception_error_dma_enable(USART0);
```

### 函数 **usart\_wakeup\_enable**

函数usart\_reception\_wakeup\_enable描述见下表：

**表 3-527. 函数 usart\_wakeup\_enable**

函数名称	usart_wakeup_enable
函数原型	void usart_wakeup_enable(uint32_t usart_periph);
功能描述	使能USART唤醒
先决条件	-
被调用函数	-
输入参数{in}	
usart_periph	外设USARTx
USARTx	x=0
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* USART0 wake up enable */
usart_wakeup_enable(USART0);
```

### 函数 usart\_wakeup\_disable

函数usart\_reception\_wakeup\_disable描述见下表：

**表 3-528. 函数 usart\_wakeup\_disable**

函数名称	usart_wakeup_disable
函数原型	void usart_wakeup_disable(uint32_t usart_periph);
功能描述	失能USART唤醒
先决条件	-
被调用函数	-
输入参数{in}	
usart_periph	外设USARTx
USARTx	x=0
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* USART0 wake up disable */
usart_wakeup_disable(USART0);
```

### 函数 usart\_wakeup\_mode\_config

函数usart\_reception\_mode\_config描述见下表：

**表 3-529. 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 */

uart_wakeup_mode_config(USART0, USART_WUM_ADDR);
```

### 函数 **uart\_receive\_fifo\_enable**

函数**uart\_receive\_fifo\_enable**描述见下表：

**表 3-530. 函数 **uart\_receive\_fifo\_enable****

函数名称	uart_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 */

uart_receive_fifo_enable (USART0);
```

### 函数 **uart\_receive\_fifo\_disable**

函数**uart\_receive\_fifo\_disable**描述见下表：

**表 3-531. 函数 **uart\_receive\_fifo\_disable****

函数名称	uart_receive_fifo_disable
函数原型	void usart_receive_fifo_disable(uint32_t usart_periph);
功能描述	失能接收FIFO
先决条件	-
被调用函数	-
输入参数{in}	
usart_periph	外设USARTx
USARTx	x=0,1

输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* disable receive FIFO */

uart_receive_fifo_disable(USART0);
```

#### 函数 **uart\_receive\_fifo\_counter\_number**

函数uart\_receive\_fifo\_counter\_number描述见下表：

**表 3-532. 函数 **uart\_receive\_fifo\_counter\_number****

函数名称	uart_receive_fifo_counter_number
函数原型	uint8_t usart_receive_fifo_counter_number(uint32_t usart_periph);
功能描述	读取接收FIFO计数器的值
先决条件	-
被调用函数	-
输入参数{in}	
usart_periph	外设USARTx
USARTx	x=0,1
输出参数{out}	
-	-
返回值	
uint8_t	接收FIFO计数器的值

例如：

```
/* read receive FIFO counter number */

uint8_t temp;

temp = usart_receive_fifo_counter_number(USART0);
```

#### 函数 **uart\_flag\_get**

函数uart\_flag\_get描述见下表：

**表 3-533. 函数 **uart\_flag\_get****

函数名称	uart_flag_get
函数原型	FlagStatus usart_flag_get(uint32_t usart_periph, usart_flag_enum flag);
功能描述	获取USART STAT/CHC/RFCs寄存器标志位
先决条件	-
被调用函数	-
输入参数{in}	

<b>usart_periph</b>	外设USARTx
<b>USARTx</b>	x=0,1
<b>输入参数{in}</b>	
<b>flag</b>	USART标志位, 参考 <a href="#">表3-462. 枚举类型usart_flag_enum</a> 只能选择一个参数
<b>USART_FLAG_PE RR</b>	校验错误标志
<b>USART_FLAG_FER R</b>	帧错误标志
<b>USART_FLAG_NE RR</b>	噪声错误标志
<b>USART_FLAG_OR ERR</b>	溢出错误标志
<b>USART_FLAG_IDL E</b>	空闲线检测标志
<b>USART_FLAG_RB NE</b>	读数据缓冲区非空标志
<b>USART_FLAG_TC</b>	发送完成标志
<b>USART_FLAG_TBE</b>	发送数据缓冲区空标志
<b>USART_FLAG_LBD</b>	LIN断开检测标志
<b>USART_FLAG_CTS F</b>	CTS变化标志
<b>USART_FLAG_CTS</b>	CTS电平
<b>USART_FLAG_RT</b>	接收超时标志
<b>USART_FLAG_EB</b>	块结束标志
<b>USART_FLAG_AB DE</b>	自动波特率检测错误
<b>USART_FLAG_AB D</b>	自动波特率检测标志
<b>USART_FLAG_BSY</b>	忙状态标志
<b>USART_FLAG_AM</b>	ADDR匹配标志
<b>USART_FLAG_SB</b>	断开信号发送标识
<b>USART_FLAG_RW U</b>	接收器从静默模式唤醒
<b>USART_FLAG_WU</b>	从深度睡眠模式唤醒标志
<b>USART_FLAG_TEA</b>	发送使能通知标志
<b>USART_FLAG_RE A</b>	接收使能通知标志
<b>USART_FLAG_EPE RR</b>	校验错误超前检测标志
<b>USART_FLAG_RFE</b>	接收FIFO空标志
<b>USART_FLAG_RFF</b>	接收FIFO满标志

<i>USART_FLAG_RFF</i>	接收FIFO满中断标志
<b>输出参数{out}</b>	
-	-
<b>返回值</b>	
<b>FlagStatus</b>	SET或RESET

例如：

```
/* get 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 3-534. 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	-
<b>输入参数{in}</b>	
<b>usart_periph</b>	外设USARTx
<b>USARTx</b>	x=0,1
<b>输入参数{in}</b>	
<b>flag</b>	USART标志位，参考 <a href="#">表3-462. 枚举类型usart_flag_enum</a> 只能选择一个参数
<i>USART_FLAG_PE</i>	校验错误标志
<i>RR</i>	
<i>USART_FLAG_FER</i>	帧错误标志
<i>R</i>	
<i>USART_FLAG_NE</i>	噪声错误标志
<i>RR</i>	
<i>USART_FLAG_OR</i>	溢出错误标志
<i>ERR</i>	
<i>USART_FLAG_IDL</i>	空闲线检测标志
<i>E</i>	
<i>USART_FLAG_TC</i>	发送完成标志
<i>USART_FLAG_LBD</i>	LIN断开检测标志
<i>USART_FLAG_CTS</i>	CTS变化标志
<i>F</i>	

<b>USART_FLAG_RT</b>	接收超时标志
<b>USART_FLAG_EB</b>	块结束标志
<b>USART_FLAG_AM</b>	ADDR匹配标志
<b>USART_FLAG_WU</b>	从深度睡眠模式唤醒标志
<b>USART_FLAG_EPE RR</b>	校验错误超前检测标志
<b>输出参数{out}</b>	
-	-
<b>返回值</b>	
-	-

例如：

```
/* clear USART0 flag */

uart_flag_clear(USART0,USART_FLAG_TC);
```

### 函数 **uart\_interrupt\_enable**

函数**uart\_interrupt\_enable**描述见下表：

**表 3-535. 函数 **uart\_interrupt\_enable****

<b>函数名称</b>	<b>uart_interrupt_enable</b>
<b>函数原型</b>	void uart_interrupt_enable(uint32_t usart_periph, usart_interrupt_enum interrupt);
<b>功能描述</b>	使能USART中断
<b>先决条件</b>	-
<b>被调用函数</b>	-
<b>输入参数{in}</b>	
<b>usart_periph</b>	外设USARTx
<b>USARTx</b>	x=0,1
<b>输入参数{in}</b>	
<b>interrupt</b>	USART中断USART标志位，参考 <a href="#">表3-464. 枚举类型usart_interrupt_enum</a> 只能选择一个参数
<b>USART_INT_IDLE</b>	IDLE线检测中断
<b>USART_INT_RBNE</b>	读数据缓冲区非空中断和过载错误中断
<b>USART_INT_TC</b>	发送完成中断
<b>USART_INT_TBE</b>	发送缓冲区空中断
<b>USART_INT_PERR</b>	校验错误中断
<b>USART_INT_AM</b>	ADDR匹配中断
<b>USART_INT_RT</b>	接收超时事件中断
<b>USART_INT_EB</b>	块结束事件中断
<b>USART_INT_LBD</b>	LIN断开信号检测中断
<b>USART_INT_ERR</b>	错误中断
<b>USART_INT_CTS</b>	CTS中断

<i>USART_INT_WU</i>	从深度睡眠模式唤醒中断
<i>USART_INT_RFF</i>	接收FIFO满中断
<b>输出参数{out}</b>	
-	-
<b>返回值</b>	
-	-

例如：

```
/* enable USART0 TBE interrupt */

uart_interrupt_enable(USART0, USART_INT_TBE);
```

#### 函数 **uart\_interrupt\_disable**

函数 **uart\_interrupt\_disable** 描述见下表：

**表 3-536. 函数 **uart\_interrupt\_disable****

函数名称	<b>uart_interrupt_disable</b>
函数原型	void <b>uart_interrupt_disable</b> (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	-
<b>输入参数{in}</b>	
<b>usart_periph</b>	外设USARTx
<b>USARTx</b>	x=0,1
<b>输入参数{in}</b>	
<b>interrupt</b>	USART中断USART标志位，参考 <a href="#">表3-464. 枚举类型usart_interrupt_enum</a> 只能选择一个参数
<i>USART_INT_IDLE</i>	IDLE线检测中断
<i>USART_INT_RBNE</i>	读数据缓冲区非空中断和过载错误中断
<i>USART_INT_TC</i>	发送完成中断
<i>USART_INT_TBE</i>	发送缓冲区空中断
<i>USART_INT_PERR</i>	校验错误中断
<i>USART_INT_AM</i>	ADDR匹配中断
<i>USART_INT_RT</i>	接收超时事件中断
<i>USART_INT_EB</i>	块结束事件中断
<i>USART_INT_LBD</i>	LIN断开信号检测中断
<i>USART_INT_ERR</i>	错误中断
<i>USART_INT_CTS</i>	CTS中断
<i>USART_INT_WU</i>	从深度睡眠模式唤醒中断
<i>USART_INT_RFF</i>	接收FIFO满中断
<b>输出参数{out}</b>	
-	-

返回值	
-	-

例如：

```
/* disable USART0 TBE interrupt */

uart_interrupt_disable(USART0, USART_INT_TBE);
```

### 函数 `uart_command_enable`

函数`uart_command_enable`描述见下表：

表 3-537. 函数 `uart_command_enable`

函数名称	uart_command_enable
函数原型	<code>void uart_command_enable(uint32_t usart_periph, uint32_t cmdtype);</code>
功能描述	使能USART请求
先决条件	-
被调用函数	-
输入参数{in}	
<code>usart_periph</code>	外设USARTx
<code>USARTx</code>	x=0,1
输入参数{in}	
<code>cmdtype</code>	请求类型
<code>USART_CMD_ABD</code> <code>CMD</code>	自动波特率检测请求
<code>USART_CMD_SBK</code> <code>CMD</code>	发送断开帧请求
<code>USART_CMD_MM</code> <code>CMD</code>	静模式请求
<code>USART_CMD_RXF</code> <code>CMD</code>	接收数据清空请求
<code>USART_CMD_TXF</code> <code>CMD</code>	发送数据清空请求
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* enable USART0 command */

uart_command_enable(USART0, USART_CMD_ABDCMD);
```

### 函数 usart\_interrupt\_flag\_get

函数usart\_interrupt\_flag\_get描述见下表:

**表 3-538. 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3-463. 枚举类型usart_interrupt_flag_enum</a> 只能选择一个参数
USART_INT_FLAG_EB	块结束事件中断标志
USART_INT_FLAG_RT	超时事件中断标志
USART_INT_FLAG_AM	ADDR匹配中断标志
USART_INT_FLAG_PERR	校验错误中断标志
USART_INT_FLAG_TBE	发送缓冲区空中断标志
USART_INT_FLAG_TC	发送完成中断标志
USART_INT_FLAG_RBNE	读数据缓冲区非空中断标志
USART_INT_FLAG_RBNE_ORERR	读数据缓冲区非空中断和溢出错误中断标志
USART_INT_FLAG_IDLE	IDLE线检测中断标志
USART_INT_FLAG_LBD	LIN断开检测中断标志
USART_INT_FLAG_WU	从深度睡眠模式唤醒中断标志
USART_INT_FLAG_CTS	CTS中断标志
USART_INT_FLAG_ERR_NERR	噪声错误中断标志

<i>USART_INT_FLAG</i>	过载错误中断标志
<i>USART_INT_FLAG</i> <i>_ERR_FERR</i>	帧错误中断标志
<i>USART_INT_FLAG</i> <i>_RFF</i>	接收FIFO满中断标志
<b>输出参数{out}</b>	
-	-
<b>返回值</b>	
<b>FlagStatus</b>	SET或RESET

例如：

```
/* get the USART0 interrupt flag status */

FlagStatus status;

status = usart_interrupt_flag_get(USART0, USART_INT_FLAG_RBNE);
```

### 函数 **usart\_interrupt\_flag\_clear**

函数**usart\_interrupt\_flag\_clear**描述见下表：

**表 3-539. 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 flag);
功能描述	清除USART中断标志位状态
先决条件	-
被调用函数	-
<b>输入参数{in}</b>	
<b>usart_periph</b>	外设USARTx
<b>USARTx</b>	x=0,1
<b>输入参数{in}</b>	
<b>flag</b>	USART中断标志, 参考 <a href="#">表3-463. 枚举类型usart_interrupt_flag_enum</a> 只能选择一个参数
<i>USART_INT_FLAG</i> <i>_PERR</i>	校验错误中断标志
<i>USART_INT_FLAG</i> <i>_ERR_FERR</i>	帧错误中断标志
<i>USART_INT_FLAG</i> <i>_ERR_NERR</i>	噪声错误中断标志
<i>USART_INT_FLAG</i> <i>_RBNE_ORERR</i>	读数据缓冲区非空中断和溢出错误中断标志
<i>USART_INT_FLAG</i> <i>_ERR_ORERR</i>	过载错误中断标志

<i>USART_INT_FLAG_IDLE</i>	IDLE线检测中断标志
<i>USART_INT_FLAG_TC</i>	发送完成中断标志
<i>USART_INT_FLAG_LBD</i>	LIN断开检测中断标志
<i>USART_INT_FLAG_CTS</i>	CTS变化中断标志
<i>USART_INT_FLAG_RT</i>	接收超时事件中断标志
<i>USART_INT_FLAG_EB</i>	块结束事件中断标志
<i>USART_INT_FLAG_AM</i>	ADDR匹配中断标志
<i>USART_INT_FLAG_WU</i>	从深度睡眠模式唤醒中断标志
<i>USART_INT_FLAG_RFF</i>	接收FIFO满中断标志
<b>输出参数{out}</b>	
-	-
<b>返回值</b>	
-	-

例如：

```
/* clear the USART0 interrupt flag */
uart_interrupt_flag_clear(USART0, USART_INT_FLAG_TC);
```

## 3.20. WWDGT

窗口看门狗定时器(WWDGT)用来监测由软件故障导致的系统故障。章节[3.20.1](#)描述了WWDGT的寄存器列表，章节[3.20.2](#)对WWDGT库函数进行说明。

### 3.20.1. 外设寄存器说明

WWDGT寄存器列表如下表所示：

**表 3-540. WWDGT 寄存器**

寄存器名称	寄存器描述
WWDGT_CTL	控制寄存器
WWDGT_CFG	配置寄存器
WWDGT_STAT	状态寄存器

### 3.20.2. 外设库函数说明

WWDGT库函数列表如下表所示：

**表 3-541. 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 3-542. 函数 wwdgt\_deinit**

函数名称	wwdgt_deinit
函数原型	void wwdgt_deinit(void);
功能描述	将WWDGT寄存器重设为缺省值
先决条件	-
被调用函数	-
输入参数{in}	
-	-
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* reset the WWDGT configuration */
wwdgt_deinit();
```

#### 函数 wwdgt\_enable

函数wwdgt\_enable描述见下表：

**表 3-543. 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	-

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

例如：

```
/* start the WWDGT counter */

wwdgt_enable();
```

### 函数 **wwdgt\_counter\_update**

函数 **wwdgt\_counter\_update** 描述见下表：

**表 3-544. 函数 **wwdgt\_counter\_update****

<b>函数名称</b>	wwdgt_counter_update
<b>函数原型</b>	void wwdgt_counter_update(uint16_t counter_value);
<b>功能描述</b>	设置WWDGT计数器更新值
<b>先决条件</b>	-
<b>被调用函数</b>	-
<b>输入参数{in}</b>	
<b>counter_value</b>	计数器值，数值范围为0x00000000 - 0x0000007F
<b>输出参数{out}</b>	
-	-
<b>返回值</b>	
-	-

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 3-545. 函数 **wwdgt\_config****

<b>函数名称</b>	wwdgt_config
<b>函数原型</b>	void wwdgt_config(uint16_t counter, uint16_t window, uint32_t prescaler);
<b>功能描述</b>	设置WWDGT计数器值、窗口值和预分频值
<b>先决条件</b>	-
<b>被调用函数</b>	-

输入参数{in}	
<b>counter</b>	定时器计数值, 数值范围0x00000000 - 0x0000007F
输入参数{in}	
<b>window</b>	窗口值, 数值范围0x00000000 - 0x0000007F
输入参数{in}	
<b>prescaler</b>	WWDGT预分频值
<i>WWDGT_CFG_PSC_DIV1</i>	WWDGT计数器时钟为 (PCLK/4096) /1
<i>WWDGT_CFG_PSC_DIV2</i>	WWDGT计数器时钟为 (PCLK/4096) /2
<i>WWDGT_CFG_PSC_DIV4</i>	WWDGT计数器时钟为 (PCLK/4096) /4
<i>WWDGT_CFG_PSC_DIV8</i>	WWDGT计数器时钟为 (PCLK/4096) /8
输出参数{out}	
-	-
Return value	
-	-

例如:

```
/* confiure WWDGT counter value to 0x7F, window value to 0x50, prescaler divider value to
8 */

wwdgt_config(127, 80, WWDGT_CFG_PSC_DIV8);
```

### 函数 wwdgt\_interrupt\_enable

函数wwdgt\_interrupt\_enable描述见下表:

**表 3-546. 函数 wwdgt\_interrupt\_enable**

函数名称	wwdgt_interrupt_enable
函数原型	void wwdgt_interrupt_enable(void);
功能描述	使能WWDGT提前唤醒中断
先决条件	-
被调用函数	-
输入参数{in}	
-	-
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* enable early wakeup interrupt of WWDGT */
```

```
wwdgt_interrupt_enable ( );
```

### 函数 wwdgt\_flag\_get

函数wwdgt\_flag\_get描述见下表:

**表 3-547. 函数 wwdgt\_flag\_get**

函数名称	wwdgt_flag_get
函数原型	FlagStatus wwdgt_flag_get(void);
功能描述	检查WWDGT提前唤醒中断标志位是否置位
先决条件	-
被调用函数	-
输入参数{in}	
-	-
输出参数{out}	
-	-
返回值	
FlagStatus	SET or RESET

例如:

```
/* test if the counter value update has reached the 0x40 */

FlagStatus status;

status = wwdgt_flag_get ( );

if(status == RESET)
```

### 函数 wwdgt\_flag\_clear

函数wwdgt\_flag\_clear描述见下表:

**表 3-548. 函数 wwdgt\_flag\_clear**

函数名称	wwdgt_flag_clear
函数原型	void wwdgt_flag_clear(void);
功能描述	清除WWDGT提前唤醒中断标志位状态
先决条件	-
被调用函数	-
输入参数{in}	
-	-
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* clear early wakeup interrupt state of WWDGT */
```

```
wwdgt_flag_clear( );
```

## 4. 版本历史

表 4-1. 版本历史

版本号.	说明	日期
1.0	初稿发布	2020 年 12 月 7 日

## Important Notice

This document is the property of GigaDevice Semiconductor Inc. and its subsidiaries (the "Company"). This document, including any product of the Company described in this document (the "Product"), is owned by the Company under the intellectual property laws and treaties of the People's Republic of China and other jurisdictions worldwide. The Company reserves all rights under such laws and treaties and does not grant any license under its patents, copyrights, trademarks, or other intellectual property rights. The names and brands of third party referred thereto (if any) are the property of their respective owner and referred to for identification purposes only.

The Company makes no warranty of any kind, express or implied, with regard to this document or any Product, including, but not limited to, the implied warranties of merchantability and fitness for a particular purpose. The Company does not assume any liability arising out of the application or use of any Product described in this document. Any information provided in this document is provided only for reference purposes. It is the responsibility of the user of this document to properly design, program, and test the functionality and safety of any application made of this information and any resulting product. Except for customized products which has been expressly identified in the applicable agreement, the Products are designed, developed, and/or manufactured for ordinary business, industrial, personal, and/or household applications only. The Products are not designed, intended, or authorized for use as components in systems designed or intended for the operation of weapons, weapons systems, nuclear installations, atomic energy control instruments, combustion control instruments, airplane or spaceship instruments, transportation instruments, traffic signal instruments, life-support devices or systems, other medical devices or systems (including resuscitation equipment and surgical implants), pollution control or hazardous substances management, or other uses where the failure of the device or Product could cause personal injury, death, property or environmental damage ("Unintended Uses"). Customers shall take any and all actions to ensure using and selling the Products in accordance with the applicable laws and regulations. The Company is not liable, in whole or in part, and customers shall and hereby do release the Company as well as it's suppliers and/or distributors from any claim, damage, or other liability arising from or related to all Unintended Uses of the Products. Customers shall indemnify and hold the Company as well as it's suppliers and/or distributors harmless from and against all claims, costs, damages, and other liabilities, including claims for personal injury or death, arising from or related to any Unintended Uses of the Products.

Information in this document is provided solely in connection with the Products. The Company reserves the right to make changes, corrections, modifications or improvements to this document and Products and services described herein at any time, without notice.