

GigaDevice Semiconductor Inc.

GD32F3x0
Arm[®] Cortex[®]-M4 32-bit MCU

**固件库
使用指南**

1.2 版本

(2022 年 1 月)

目录

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

3.8. DBG	116
3.8.1. 外设寄存器说明	116
3.8.2. 外设库函数说明	117
3.9. DMA	121
3.9.1. 外设寄存器说明	121
3.9.2. 外设库函数说明	122
3.10. EXTI	139
3.10.1. 外设寄存器说明	139
3.10.2. 外设库函数说明	139
3.11. FMC	147
3.11.1. 外设寄存器说明	147
3.11.2. 外设库函数说明	148
3.12. FWDGT	166
3.12.1. 外设寄存器说明	166
3.12.2. 外设库函数说明	167
3.13. GPIO	171
3.13.1. 外设寄存器说明	171
3.13.2. 外设库函数说明	171
3.14. I2C	182
3.14.1. 外设寄存器说明	182
3.14.2. 外设库函数说明	182
3.15. MISC	205
3.15.1. 外设寄存器说明	205
3.15.2. 外设库函数说明	206
3.16. PMU	212
3.16.1. 外设寄存器说明	212
3.16.2. 外设库函数说明	212
3.17. RCU	224
3.17.1. 外设寄存器说明	224
3.17.2. 外设库函数说明	225
3.18. RTC	259
3.18.1. 外设寄存器描述	260
3.18.2. 外设库函数描述	260
3.19. SPI/I2S	280
3.19.1. 外设寄存器说明	280
3.19.2. 外设库函数说明	280
3.20. SYSCFG	306
3.20.1. 外设寄存器说明	306
3.20.2. 外设库函数说明	306

3.21. TIMER.....	313
3.21.1. 外设寄存器说明.....	313
3.21.2. 外设库函数说明.....	314
3.22. TSI	372
3.22.1. 外设寄存器描述.....	372
3.22.2. 外设库函数说明.....	372
3.23. USART.....	394
3.23.1. 外设寄存器说明.....	394
3.23.2. 外设库函数说明.....	395
3.24. WWDGT.....	444
3.24.1. 外设寄存器说明.....	444
3.24.2. 外设库函数说明.....	444
3.25. USBFS.....	449
4. 版本历史	450

图索引

图 2-1. GD32F3x0 固件库文件组织结构	24
图 2-2. 选择外设例程文件	26
图 2-3. 拷贝外设例程文件	27
图 2-4. 打开工程文件	28
图 2-5. 配置工程文件	28
图 2-6. 编译调试下载	29

表索引

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

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

表 3-83. CRC 库函数	82
表 3-84. 函数 <code>crc_deinit</code>	82
表 3-85. 函数 <code>crc_reverse_output_data_enable</code>	83
表 3-86. 函数 <code>crc_reverse_output_data_enable</code>	83
表 3-87. 函数 <code>crc_data_register_reset</code>	84
表 3-88. 函数 <code>crc_data_register_read</code>	84
表 3-89. 函数 <code>crc_free_data_register_read</code>	85
表 3-90. 函数 <code>crc_free_data_register_write</code>	85
表 3-91. 函数 <code>crc_init_data_register_write</code>	86
表 3-92. 函数 <code>crc_input_data_reverse_config</code>	86
表 3-93. 函数 <code>crc_polynomial_size_set</code>	87
表 3-94. 函数 <code>crc_polynomial_set</code>	88
表 3-95. 函数 <code>crc_single_data_calculate</code>	88
表 3-96. 函数 <code>crc_block_data_calculate</code>	89
表 3-97. CTC 寄存器	89
表 3-98. CTC 库函数	90
表 3-99. 函数 <code>ctc_deinit</code>	90
表 3-100. 函数 <code>ctc_refresource_polarity_config</code>	91
表 3-101. 函数 <code>ctc_refresource_signal_select</code>	91
表 3-102. 函数 <code>ctc_refresource_prescaler_config</code>	92
表 3-103. 函数 <code>ctc_clock_limit_value_config</code>	93
表 3-104. 函数 <code>ctc_counter_reload_value_config</code>	93
表 3-105. 函数 <code>ctc_counter_enable</code>	94
表 3-106. 函数 <code>ctc_counter_disable</code>	94
表 3-107. 函数 <code>ctc_irc48m_trim_value_config</code>	95
表 3-108. 函数 <code>ctc_software_refresource_pulse_generate</code>	95
表 3-109. 函数 <code>ctc_hardware_trim_mode_config</code>	96
表 3-110. 函数 <code>ctc_counter_capture_value_read</code>	96
表 3-111. 函数 <code>ctc_counter_direction_read</code>	97
表 3-112. 函数 <code>ctc_counter_reload_value_read</code>	97
表 3-113. 函数 <code>ctc_irc48m_trim_value_read</code>	98
表 3-114. 函数 <code>ctc_interrupt_enable</code>	98
表 3-115. 函数 <code>ctc_interrupt_disable</code>	99
表 3-116. 函数 <code>ctc_flag_get</code>	100
表 3-117. 函数 <code>ctc_flag_clear</code>	100
表 3-118. 函数 <code>ctc_interrupt_flag_get</code>	101
表 3-119. 函数 <code>ctc_interrupt_flag_clear</code>	102
表 3-120. DAC 寄存器	103
表 3-121. DAC 库函数	103
表 3-122. 函数 <code>dac_deinit</code>	104
表 3-123. 函数 <code>dac_enable</code>	104
表 3-124. 函数 <code>dac_disable</code>	105
表 3-125. 函数 <code>dac_dma_enable</code>	105
表 3-126. 函数 <code>dac_dma_disable</code>	106

表 3-127. 函数 dac_output_buffer_enable	106
表 3-128. 函数 dac_output_buffer_disable	107
表 3-129. 函数 dac_trigger_enable	107
表 3-130. 函数 dac_trigger_disable	108
表 3-131. 函数 dac_software_trigger_enable	108
表 3-132. 函数 dac_software_trigger_disable	109
表 3-133. 函数 dac_trigger_source_config	109
表 3-134. 函数 dac_wave_mode_config	110
表 3-135. 函数 dac_wave_bit_width_config	110
表 3-136. 函数 dac_lfsr_noise_config	111
表 3-137. 函数 dac_triangle_noise_config	112
表 3-138. 函数 dac_output_value_get	112
表 3-139. 函数 dac_data_set	113
表 3-140. 函数 dac_flag_get	113
表 3-141. 函数 dac_flag_clear	114
表 3-142. 函数 dac_interrupt_enable	114
表 3-143. 函数 dac_interrupt_disable	115
表 3-144. 函数 dac_interrupt_flag_get	115
表 3-145. 函数 dac_interrupt_flag_clear	116
表 3-146. DBG 寄存器	117
表 3-147. DBG 库函数	117
表 3-148. 枚举类型 dbg_periph_enum	117
表 3-149. 函数 dbg_deinit	117
表 3-150. 函数 dbg_id_get	118
表 3-151. 函数 dbg_low_power_enable	118
表 3-152. 函数 dbg_low_power_disable	119
表 3-153. 函数 dbg_periph_enable	120
表 3-154. 函数 dbg_periph_disable	120
表 3-155. DMA 寄存器	121
表 3-156. DMA 库函数	122
表 3-157. 枚举类型 dma_channel_enum	122
表 3-158. 结构体类型 dma_parameter_struct	123
表 3-159. 函数 dma_deinit	123
表 3-160. 函数 dma_struct_para_init	123
表 3-161. 函数 dma_init	124
表 3-162. 函数 dma_circulation_enable	125
表 3-163. 函数 dma_circulation_disable	125
表 3-164. 函数 dma_memory_to_memory_enable	126
表 3-165. 函数 dma_memory_to_memory_disable	126
表 3-166. 函数 dma_channel_enable	127
表 3-167. 函数 dma_channel_disable	127
表 3-168. 函数 dma_periph_address_config	128
表 3-169. 函数 dma_memory_address_config	128
表 3-170. 函数 dma_transfer_number_config	129

表 3-171. 函数 dma_transfer_number_get	130
表 3-172. 函数 dma_priority_config	130
表 3-173. 函数 dma_memory_width_config	131
表 3-174. 函数 dma_periph_width_config	132
表 3-175. 函数 dma_memory_increase_enable	132
表 3-176. 函数 dma_memory_increase_disable	133
表 3-177. 函数 dma_periph_increase_enable	133
表 3-178. 函数 dma_periph_increase_disable	134
表 3-179. 函数 dma_transfer_direction_config	134
表 3-180. 函数 dma_flag_get.....	135
表 3-181. 函数 dma_flag_clear.....	136
表 3-182. 函数 dma_interrupt_enable	136
表 3-183. 函数 dma_interrupt_disable	137
表 3-184. 函数 dma_interrupt_flag_get.....	137
表 3-185. 函数 dma_interrupt_flag_clear.....	138
表 3-186. EXTI 寄存器	139
表 3-187. EXTI 库函数	139
表 3-188. 枚举类型 exti_line_enum.....	140
表 3-189. 枚举类型 exti_mode_enum	141
表 3-190. 枚举类型 exti_trig_type_enum.....	141
表 3-191. 函数 exti_deinit	141
表 3-192. 函数 exti_init	141
表 3-193. 函数 exti_interrupt_enable	142
表 3-194. 函数 exti_interrupt_disable	143
表 3-195. 函数 exti_event_enable	143
表 3-196. 函数 exti_event_disable	144
表 3-197. 函数 exti_software_interrupt_enable	144
表 3-198. 函数 exti_software_interrupt_disable	145
表 3-199. 函数 exti_flag_get.....	145
表 3-200. 函数 exti_flag_clear	146
表 3-201. 函数 exti_interrupt_flag_get	146
表 3-202. 函数 exti_interrupt_flag_clear	147
表 3-203. FMC 寄存器.....	147
表 3-204. FMC 固件库函数	148
表 3-205. 枚举类型 fmc_state_enum	149
表 3-206. 结构体类型 ob_parm_struct.....	149
表 3-207. 函数 fmc_unlock	149
表 3-208. 函数 fmc_lock	150
表 3-209. 函数 fmc_wscnt_set	150
表 3-210. 函数 fmc_wait_state_enable.....	151
表 3-211. 函数 fmc_wait_state_disable	151
表 3-212. 函数 fmc_page_erase.....	152
表 3-213. 函数 fmc_mass_erase	152
表 3-214. 函数 fmc_word_program	153

表 3-215. 函数 fmc_halfword_program.....	153
表 3-216. 函数 fmc_word_reprogram.....	154
表 3-217. 函数 ob_unlock	154
表 3-218. 函数 ob_lock	155
表 3-219. 函数 ob_reset	155
表 3-220. 函数 ob_erase	156
表 3-221. 函数 ob_write_protection_enable	156
表 3-222. 函数 ob_security_protection_config	157
表 3-223. 函数 ob_user_write.....	157
表 3-224. 函数 ob_data_program	158
表 3-225. 函数 ob_user_get.....	159
表 3-226. 函数 ob_data_get.....	159
表 3-227. 函数 ob_write_protection_get.....	160
表 3-228. 函数 ob_obstat_plevel_get	160
表 3-229. 函数 fmc_interrupt_enable	161
表 3-230. 函数 fmc_interrupt_disable	161
表 3-231. 函数 fmc_flag_get.....	162
表 3-232. 函数 fmc_flag_clear.....	162
表 3-233. 函数 fmc_interrupt_flag_get.....	163
表 3-234. 函数 fmc_interrupt_flag_clear.....	163
表 3-235. 函数 fmc_state_get.....	164
表 3-236. 函数 fmc_ready_wait.....	165
表 3-237. 函数 ob_parm_get	165
表 3-238. 函数 ob_value_modify	166
表 3-239. FWDGT 寄存器	166
表 3-240. FWDGT 库函数	167
表 3-241. 函数 fwdgt_write_enable	167
表 3-242. 函数 fwdgt_write_disable	167
表 3-243. 函数 fwdgt_enable	168
表 3-244. 函数 fwdgt_window_value_config	168
表 3-245. 函数 fwdgt_counter_reload	169
表 3-246. 函数 fwdgt_config.....	169
表 3-247. 函数 fwdgt_flag_get.....	170
表 3-248. GPIO 寄存器	171
表 3-249. GPIO 库函数	171
表 3-250. 函数 gpio_deinit.....	172
表 3-251. 函数 gpio_mode_set.....	172
表 3-252. 函数 gpio_output_options_set.....	173
表 3-253. 函数 gpio_bit_set.....	174
表 3-254. 函数 gpio_bit_reset	175
表 3-255. 函数 gpio_bit_write.....	175
表 3-256. 函数 gpio_port_write	176
表 3-257. 函数 gpio_input_bit_get	177
表 3-258. 函数 gpio_input_port_get	177

表 3-259. 函数 gpio_output_bit_get	178
表 3-260. 函数 gpio_output_port_get.....	178
表 3-261. 函数 gpio_af_set.....	179
表 3-262. 函数 gpio_pin_lock.....	180
表 3-263. 函数 gpio_bit_toggle	180
表 3-264. 函数 gpio_port_toggle	181
表 3-265. I2C 寄存器.....	182
表 3-266. I2C 库函数.....	182
表 3-267. 枚举类型 i2c_flag_enum.....	183
表 3-268. 枚举类型 i2c_interrupt_flag_enum.....	184
表 3-269. 枚举类型 i2c_interrupt_enum	184
表 3-270. 函数 i2c_deinit	184
表 3-271. 函数 i2c_clock_config	185
表 3-272. 函数 i2c_mode_addr_config.....	186
表 3-273. 函数 i2c_smbus_type_config	186
表 3-274. 函数 i2c_ack_config	187
表 3-275. 函数 i2c_ackpos_config.....	188
表 3-276. 函数 i2c_master_addressing	188
表 3-277. 函数 i2c_dualaddr_enable	189
表 3-278. 函数 i2c_dualaddr_disable	189
表 3-279. 函数 i2c_enable.....	190
表 3-280. 函数 i2c_disable.....	190
表 3-281. 函数 i2c_start_on_bus	191
表 3-282. 函数 i2c_stop_on_bus.....	191
表 3-283. 函数 i2c_data_transmit	192
表 3-284. 函数 i2c_data_receive	193
表 3-285. 函数 i2c_dma_enable	193
表 3-286. 函数 i2c_dma_last_transfer_config	194
表 3-287. 函数 i2c_stretch_scl_low_config	194
表 3-288. 函数 i2c_slave_response_to_gcall_config	195
表 3-289. 函数 i2c_software_reset_config.....	196
表 3-290. 函数 i2c_pec_enable	196
表 3-291. 函数 i2c_pec_transfer_enable	197
表 3-292. 函数 i2c_pec_value_get	197
表 3-293. 函数 i2c_smbus_issue_alert	198
表 3-294. 函数 i2c_smbus_arp_enable	199
表 3-295. 函数 i2c_flag_get	199
表 3-296. 函数 i2c_flag_clear	201
表 3-297. 函数 i2c_interrupt_enable.....	201
表 3-298. 函数 i2c_interrupt_disable.....	202
表 3-299. 函数 i2c_interrupt_flag_get	203
表 3-300. 函数 i2c_interrupt_flag_clear	204
表 3-301. NVIC 寄存器.....	205
表 3-302. Systick 寄存器.....	206

表 3-303. MISC 库函数	206
表 3-304. 枚举类型 IRQn_Type	206
表 3-305. 函数 nvic_priority_group_set.....	208
表 3-306. 函数 nvic_irq_enable.....	208
表 3-307. 函数 nvic_irq_disable.....	209
表 3-308. 函数 nvic_vector_table_set	209
表 3-309. 函数 system_lowpower_set.....	210
表 3-310. 函数 system_lowpower_reset	210
表 3-311. 函数 systick_clksource_set.....	211
表 3-312. PMU 寄存器	212
表 3-313. PMU 库函数	212
表 3-314. 函数 pmu_deinit.....	213
表 3-315. 函数 pmu_lvd_select.....	213
表 3-316. 函数 pmu_ldo_output_select	214
表 3-317. 函数 pmu_lvd_disable.....	214
表 3-318. 函数 pmu_lowdriver_mode_enable	215
表 3-319. 函数 pmu_lowdriver_mode_disable	215
表 3-320. 函数 pmu_highdriver_mode_enable.....	216
表 3-321. 函数 pmu_highdriver_mode_disable.....	216
表 3-322. 函数 pmu_highdriver_switch_select	217
表 3-323. 函数 pmu_lowpower_driver_config.....	217
表 3-324. 函数 pmu_normalpower_driver_config.....	218
表 3-325. 函数 pmu_to_sleepmode	218
表 3-326. 函数 pmu_to_deepsleepmode.....	219
表 3-327. 函数 pmu_to_standbymode	220
表 3-328. 函数 pmu_wakeup_pin_enable	220
表 3-329. 函数 pmu_wakeup_pin_disable	221
表 3-330. 函数 pmu_backup_write_enable	222
表 3-331. 函数 pmu_backup_write_disable	222
表 3-332. 函数 pmu_flag_get.....	223
表 3-333. 函数 pmu_flag_clear	223
表 3-334. RCU 寄存器	224
表 3-335. RCU 库函数	225
表 3-336. 枚举类型 reg_idx	226
表 3-337. 枚举类型 rcu_periph_enum	227
表 3-338. 枚举类型 rcu_periph_sleep_enum	227
表 3-339. 枚举类型 rcu_periph_reset_enum.....	228
表 3-340. 枚举类型 rcu_flag_enum	228
表 3-341. 枚举类型 rcu_int_flag_enum.....	229
表 3-342. 枚举类型 rcu_int_flag_clear_enum	230
表 3-343. 枚举类型 rcu_int_enum	230
表 3-344. 枚举类型 rcu_adc_clock_enum	231
表 3-345. 枚举类型 rcu_osc_type_enum	231
表 3-346. 枚举类型 rcu_clock_freq_enum	231

表 3-347. 函数 rcu_deinit.....	232
表 3-348. 函数 rcu_periph_clock_enable.....	232
表 3-349. 函数 rcu_periph_clock_disable.....	233
表 3-350. 函数 rcu_periph_clock_sleep_enable	234
表 3-351. 函数 rcu_periph_clock_sleep_disable	235
表 3-352. 函数 rcu_periph_reset_enable	235
表 3-353. 函数 rcu_periph_reset_disable	236
表 3-354. 函数 rcu_bkp_reset_enable	237
表 3-355. 函数 rcu_bkp_reset_disable	237
表 3-356. 函数 rcu_system_clock_source_config	238
表 3-357. 函数 rcu_system_clock_source_get	238
表 3-358. 函数 rcu_ahb_clock_config.....	239
表 3-359. 函数 rcu_apb1_clock_config.....	239
表 3-360. 函数 rcu_apb2_clock_config.....	240
表 3-361. 函数 rcu_adc_clock_config	240
表 3-362. 函数 rcu_usbfs_clock_config.....	241
表 3-363. 函数 rcu_ckout_config	242
表 3-364. 函数 rcu_pll_preselection_config	243
表 3-365. 函数 rcu_pll_config	243
表 3-366. 函数 rcu_usart_clock_config	244
表 3-367. 函数 rcu_cec_clock_config	245
表 3-368. 函数 rcu_rtc_clock_config	245
表 3-369. 函数 rcu_ck48m_clock_config	246
表 3-370. 函数 rcu_hxtal_prediv_config	246
表 3-371. 函数 rcu_lxtal_drive_capability_config	247
表 3-372. 函数 rcu_flag_get.....	248
表 3-373. 函数 rcu_all_reset_flag_clear.....	249
表 3-374. 函数 rcu_interrupt_flag_get.....	249
表 3-375. 函数 rcu_interrupt_flag_clear.....	250
表 3-376. 函数 rcu_interrupt_enable	251
表 3-377. 函数 rcu_interrupt_disable	252
表 3-378. 函数 rcu_osc1_stab_wait.....	253
表 3-379. 函数 rcu_osc1_on.....	253
表 3-380. 函数 rcu_osc1_off.....	254
表 3-381. 函数 rcu_osc1_bypass_mode_enable.....	255
表 3-382. 函数 rcu_osc1_bypass_mode_disable.....	255
表 3-383. 函数 rcu_hxtal_clock_monitor_enable	256
表 3-384. 函数 rcu_hxtal_clock_monitor_disable	256
表 3-385. 函数 rcu_irc8m_adjust_value_set.....	257
表 3-386. 函数 rcu_irc28m_adjust_value_set.....	257
表 3-387. 函数 rcu_voltage_key_unlock	258
表 3-388. 函数 rcu_deepsleep_voltage_set.....	258
表 3-389. 函数 rcu_clock_freq_get	259
表 3-390. RTC 寄存器.....	260

表 3-391. RTC 库函数.....	260
表 3-392. 结构体类型 <code>rtc_parameter_struct</code>	261
表 3-393. 结构体类型 <code>rtc_alarm_struct</code>	262
表 3-394. 结构体 <code>rtc_timestamp_struct</code>	262
表 3-395. 结构体 <code>rtc_tamper_struct</code>	262
表 3-396. 函数 <code>rtc_deinit</code>	263
表 3-397. 函数 <code>rtc_init</code>	263
表 3-398. 函数 <code>rtc_init_mode_enter</code>	264
表 3-399. 函数 <code>rtc_init_mode_exit</code>	264
表 3-400. 函数 <code>rtc_register_sync_wait</code>	265
表 3-401. 函数 <code>rtc_current_time_get</code>	265
表 3-402. 函数 <code>rtc_subsecond_get</code>	266
表 3-403. 函数 <code>rtc_alarm_config</code>	266
表 3-404. 函数 <code>rtc_alarm_subsecond_config</code>	267
表 3-405. 函数 <code>rtc_alarm_get</code>	268
表 3-406. 函数 <code>rtc_alarm_subsecond_get</code>	268
表 3-407. 函数 <code>rtc_alarm_enable</code>	269
表 3-408. 函数 <code>rtc_alarm_disable</code>	269
表 3-409. 函数 <code>rtc_timestamp_enable</code>	270
表 3-410. 函数 <code>rtc_timestamp_disable</code>	270
表 3-411. 函数 <code>rtc_timestamp_get</code>	271
表 3-412. 函数 <code>rtc_timestamp_subsecond_get</code>	271
表 3-413. 函数 <code>rtc_tamper_enable</code>	272
表 3-414. 函数 <code>rtc_tamper_disable</code>	272
表 3-415. 函数 <code>rtc_interrupt_enable</code>	273
表 3-416. 函数 <code>rtc_interrupt_disable</code>	273
表 3-417. 函数 <code>rtc_flag_get</code>	274
表 3-418. 函数 <code>rtc_flag_clear</code>	274
表 3-419. 函数 <code>rtc_alter_output_config</code>	275
表 3-420. 函数 <code>rtc_calibration_config</code>	276
表 3-421. 函数 <code>rtc_hour_adjust</code>	277
表 3-422. 函数 <code>rtc_second_adjust</code>	277
表 3-423. 函数 <code>rtc_bypass_shadow_enable</code>	278
表 3-424. 函数 <code>rtc_bypass_shadow_disable</code>	278
表 3-425. 函数 <code>rtc_refclock_detection_enable</code>	279
表 3-426. 函数 <code>rtc_refclock_detection_disable</code>	279
表 3-427. SPI/I2S 寄存器	280
表 3-428. SPI/I2S 库函数	280
表 3-429. 结构体类型 <code>spi_parameter_struct</code>	281
表 3-430. 函数 <code>spi_i2s_deinit</code>	282
表 3-431. 函数 <code>spi_struct_para_init</code>	282
表 3-432. 函数 <code>spi_init</code>	283
表 3-433. 函数 <code>spi_enable</code>	284
表 3-434. 函数 <code>spi_disable</code>	284

表 3-435. 函数 i2s_init.....	285
表 3-436. 函数 i2s_psc_config	286
表 3-437. 函数 i2s_enable.....	287
表 3-438. 函数 i2s_disable.....	288
表 3-439. 函数 spi_nss_output_enable	288
表 3-440. 函数 spi_nss_output_disable	289
表 3-441. 函数 spi_nss_internal_high.....	289
表 3-442. 函数 spi_nss_internal_low	290
表 3-443. 函数 spi_dma_enable	290
表 3-444. 函数 spi_dma_disable	291
表 3-445. 函数 spi_i2s_data_frame_format_config	292
表 3-446. 函数 spi_i2s_data_transmit	292
表 3-447. 函数 spi_i2s_data_receive	293
表 3-448. 函数 spi_bidirectional_transfer_config	293
表 3-449. 函数 spi_crc_polynomial_set	294
表 3-450. 函数 spi_crc_polynomial_get	294
表 3-451. 函数 spi_crc_on	295
表 3-452. 函数 spi_crc_off	296
表 3-453. 函数 spi_crc_next	296
表 3-454. 函数 spi_crc_get	297
表 3-455. 函数 spi_ti_mode_enable	297
表 3-456. 函数 spi_ti_mode_disable	298
表 3-457. 函数 spi_nssp_mode_enable	298
表 3-458. 函数 spi_nssp_mode_disable	299
表 3-459. 函数 qspi_enable	299
表 3-460. 函数 qspi_disable	300
表 3-461. 函数 qspi_write_enable	300
表 3-462. 函数 qspi_read_enable	301
表 3-463. 函数 qspi_io23_output_enable	301
表 3-464. 函数 qspi_io23_output_disable	302
表 3-465. 函数 spi_i2s_interrupt_enable	302
表 3-466. 函数 spi_i2s_interrupt_disable	303
表 3-467. 函数 spi_i2s_interrupt_flag_get	303
表 3-468. 函数 spi_i2s_flag_get	304
表 3-469. 函数 spi_crc_error_clear	305
表 3-470. SYSCFG 寄存器	306
表 3-471. SYSCFG 库函数	306
表 3-472. 函数 syscfg_deinit	307
表 3-473. 函数 syscfg_dma_remap_enable	307
表 3-474. 函数 syscfg_dma_remap_disable	308
表 3-475. 函数 syscfg_high_current_enable	309
表 3-476. 函数 syscfg_high_current_disable	309
表 3-477. 函数 syscfg_exti_line_config	310
表 3-478. 函数 syscfg_lock_config	310

表 3-479. 函数 syscfg_flag_get.....	311
表 3-480. 函数 syscfg_flag_clear.....	311
表 3-481. 函数 syscfg_compensation_config	312
表 3-482. 函数 syscfg_cps_rdy_flag_get.....	313
表 3-483. TIMER 寄存器.....	313
表 3-484. TIMER 库函数	314
表 3-485. 结构体类型 timer_parameter_struct.....	316
表 3-486. 结构体类型 timer_break_parameter_struct.....	317
表 3-487. 结构体类型 timer_oc_parameter_struct	317
表 3-488. 结构体类型 timer_ic_parameter_struct.....	317
表 3-489. 函数 timer_deinit.....	318
表 3-490. 函数 timer_struct_para_init	318
表 3-491. 函数 timer_init.....	319
表 3-492. 函数 timer_enable.....	319
表 3-493. 函数 timer_disable.....	320
表 3-494. 函数 timer_auto_reload_shadow_enable.....	321
表 3-495. 函数 timer_auto_reload_shadow_disable.....	321
表 3-496. 函数 timer_update_event_enable	322
表 3-497. 函数 timer_update_event_disable	322
表 3-498. 函数 timer_counter_alignment.....	323
表 3-499. 函数 timer_counter_up_direction	323
表 3-500. 函数 timer_counter_down_direction	324
表 3-501. 函数 timer_prescaler_config	324
表 3-502. 函数 timer_repetition_value_config.....	325
表 3-503. 函数 timer_autoreload_value_config.....	326
表 3-504. 函数 timer_counter_value_config	326
表 3-505. 函数 timer_counter_read	327
表 3-506. 函数 timer_prescaler_read	327
表 3-507. 函数 timer_single_pulse_mode_config.....	328
表 3-508. 函数 timer_update_source_config.....	329
表 3-509. 函数 timer_ocpre_clear_source_config	329
表 3-510. 函数 timer_interrupt_enable	330
表 3-511. 函数 timer_interrupt_disable	331
表 3-512. 函数 timer_interrupt_flag_get	332
表 3-513. 函数 timer_interrupt_flag_clear	333
表 3-514. 函数 timer_flag_get	334
表 3-515. 函数 timer_flag_clear	334
表 3-516. 函数 timer_dma_enable	335
表 3-517. 函数 timer_dma_disable	336
表 3-518. 函数 timer_channel_dma_request_source_select.....	337
表 3-519. 函数 timer_dma_transfer_config	338
表 3-520. 函数 timer_event_software_generate.....	340
表 3-521. 函数 timer_break_struct_para_init	340
表 3-522. 函数 timer_break_config.....	341

表 3-523. 函数 timer_break_enable	342
表 3-524. 函数 timer_break_disable	342
表 3-525. 函数 timer_automatic_output_enable	343
表 3-526. 函数 timer_automatic_output_disable	343
表 3-527. 函数 timer_primary_output_config	344
表 3-528. 函数 timer_channel_control_shadow_config	345
表 3-529. 函数 timer_channel_control_shadow_update_config	345
表 3-530. 函数 timer_channel_output_struct_para_init	346
表 3-531. 函数 timer_channel_output_config	346
表 3-532. 函数 timer_channel_output_mode_config	347
表 3-533. 函数 timer_channel_output_pulse_value_config	349
表 3-534. 函数 timer_channel_output_shadow_config	349
表 3-535. 函数 timer_channel_output_fast_config	350
表 3-536. 函数 timer_channel_output_clear_config	351
表 3-537. 函数 timer_channel_output_polarity_config	352
表 3-538. 函数 timer_channel_complementary_output_polarity_config	353
表 3-539. 函数 timer_channel_output_state_config	353
表 3-540. 函数 timer_channel_complementary_output_state_config	354
表 3-541. 函数 timer_channel_input_struct_para_init	355
表 3-542. 函数 timer_input_capture_config	356
表 3-543. 函数 timer_channel_input_capture_prescaler_config	356
表 3-544. 函数 timer_channel_capture_value_register_read	357
表 3-545. 函数 timer_input_pwm_capture_config	358
表 3-546. 函数 timer_hall_mode_config	359
表 3-547. 函数 timer_input_trigger_source_select	360
表 3-548. 函数 timer_master_output_trigger_source_select	360
表 3-549. 函数 timer_slave_mode_select	362
表 3-550. 函数 timer_master_slave_mode_config	362
表 3-551. 函数 timer_external_trigger_config	363
表 3-552. 函数 timer_quadrature_decoder_mode_config	364
表 3-553. 函数 timer_internal_clock_config	365
表 3-554. 函数 timer_internal_trigger_as_external_clock_config	366
表 3-555. 函数 timer_external_trigger_as_external_clock_config	366
表 3-556. 函数 timer_external_clock_mode0_config	367
表 3-557. 函数 timer_external_clock_mode1_config	368
表 3-558. 函数 timer_external_clock_mode1_disable	369
表 3-559. 函数 timer_channel_remap_config	370
表 3-560. 函数 timer_write_chxval_register_config	371
表 3-561. 函数 timer_output_value_selection_config	371
表 3-562. TSI 寄存器	372
表 3-563. TSI 库函数	373
表 3-564. 函数 tsi_deinit	374
表 3-565. 函数 tsi_init	374
表 3-566. 函数 cec_enable	376

表 3-567. 函数 tsi_disable	376
表 3-568. 函数 tsi_sample_pin_enable	377
表 3-569. 函数 tsi_sample_pin_disable	377
表 3-570. 函数 tsi_channel_pin_enable	378
表 3-571. 函数 tsi_channel_pin_disable	378
表 3-572. 函数 tsi_sofeware_mode_config.....	379
表 3-573. 函数 tsi_software_start	379
表 3-574. 函数 tsi_software_stop	380
表 3-575. 函数 tsi_hardware_mode_config	380
表 3-576. 函数 tsi_pin_mode_config.....	381
表 3-577. 函数 tsi_extend_charge_config	381
表 3-578. 函数 tsi_plus_config	382
表 3-579. 函数 tsi_max_number_config	383
表 3-580. 函数 tsi_hysteresis_on	384
表 3-581. 函数 tsi_hysteresis_off	385
表 3-582. 函数 tsi_analog_on.....	385
表 3-583. 函数 tsi_analog_off.....	386
表 3-584. 函数 tsi_group_enable	386
表 3-585. 函数 tsi_group_disable	387
表 3-586. 函数 tsi_group_status_get	387
表 3-587. 函数 tsi_group0_cycle_get	388
表 3-588. 函数 tsi_group1_cycle_get	388
表 3-589. 函数 tsi_group2_cycle_get	389
表 3-590. 函数 tsi_group3_cycle_get	389
表 3-591. 函数 tsi_group4_cycle_get	390
表 3-592. 函数 tsi_group5_cycle_get	390
表 3-593. 函数 tsi_flag_clear	391
表 3-594. 函数 tsi_flag_get.....	391
表 3-595. 函数 tsi_interrupt_enable	392
表 3-596. 函数 tsi_interrupt_disable	392
表 3-597. 函数 tsi_interrupt_flag_clear	393
表 3-598. 函数 tsi_interrupt_flag_get	394
表 3-599. USART 寄存器.....	394
表 3-600. USART 库函数.....	395
表 3-601. 枚举类型 usart_flag_enum.....	397
表 3-602. 枚举类型 usart_interrupt_flag_enum	398
表 3-603. 枚举类型 usart_interrupt_enum	398
表 3-604. 枚举类型 usart_invert_enum	399
表 3-605. 函数 usart_deinit.....	399
表 3-606. 函数 usart_baudrate_set.....	399
表 3-607. 函数 usart_parity_config	400
表 3-608. 函数 usart_word_length_set	401
表 3-609. 函数 usart_stop_bit_set.....	401
表 3-610. 函数 usart_enable.....	402

表 3-611. 函数 usart_disable	402
表 3-612. 函数 usart_transmit_config	403
表 3-613. 函数 usart_receive_config	404
表 3-614. 函数 usart_data_first_config	404
表 3-615. 函数 usart_invert_config	405
表 3-616. 函数 usart_overrun_enable	406
表 3-617. 函数 usart_overrun_disable	406
表 3-618. 函数 usart_oversample_config	407
表 3-619. 函数 usart_sample_bit_config	407
表 3-620. 函数 usart_receiver_timeout_enable	408
表 3-621. 函数 usart_receiver_timeout_disable	409
表 3-622. 函数 usart_receiver_timeout_threshold_config	409
表 3-623. 函数 usart_data_transmit	410
表 3-624. 函数 usart_data_receive	410
表 3-625. 函数 usart_autobaud_detection_enable	411
表 3-626. 函数 usart_autobaud_detection_disable	411
表 3-627. 函数 usart_autobaud_detection_mode_config	412
表 3-628. 函数 usart_address_config	413
表 3-629. 函数 usart_address_detection_mode_config	413
表 3-630. 函数 usart_mute_mode_enable	414
表 3-631. 函数 usart_mute_mode_disable	414
表 3-632. 函数 usart_mute_mode_wakeup_config	415
表 3-633. 函数 usart_lin_mode_enable	415
表 3-634. 函数 usart_lin_mode_disable	416
表 3-635. 函数 usart_lin_break_detection_length_config	416
表 3-636. 函数 usart_halfduplex_enable	417
表 3-637. 函数 usart_halfduplex_disable	418
表 3-638. 函数 usart_clock_enable	418
表 3-639. 函数 usart_clock_disable	419
表 3-640. 函数 usart_synchronous_clock_config	419
表 3-641. 函数 usart_guard_time_config	420
表 3-642. 函数 usart_smartcard_mode_enable	420
表 3-643. 函数 usart_smartcard_mode_disable	421
表 3-644. 函数 usart_smartcard_mode_nack_enable	421
表 3-645. 函数 usart_smartcard_mode_nack_disable	422
表 3-646. 函数 usart_smartcard_mode_early_nack_enable	422
表 3-647. 函数 usart_smartcard_mode_early_nack_disable	423
表 3-648. 函数 usart_smartcard_autoretry_config	423
表 3-649. 函数 usart_block_length_config	424
表 3-650. 函数 usart_irda_mode_enable	425
表 3-651. 函数 usart_irda_mode_disable	425
表 3-652. 函数 usart_prescaler_config	426
表 3-653. 函数 usart_irda_lowpower_config	426
表 3-654. 函数 usart_hardware_flow_rts_config	427

表 3-655. 函数 usart_hardware_flow_cts_config	427
表 3-656. 函数 usart_rs485_driver_enable	428
表 3-657. 函数 usart_rs485_driver_disable	429
表 3-658. 函数 usart_driver_assertime_config	429
表 3-659. 函数 usart_driver_deassertime_config	430
表 3-660. 函数 usart_depolarity_config	430
表 3-661. 函数 usart_dma_receive_config	431
表 3-662. 函数 usart_dma_transmit_config	431
表 3-663. 函数 usart_reception_error_dma_disable	432
表 3-664. 函数 usart_reception_error_dma_enable	433
表 3-665. 函数 usart_wakeup_enable	433
表 3-666. 函数 usart_wakeup_disable	434
表 3-667. 函数 usart_wakeup_mode_config	434
表 3-668. 函数 usart_command_enable	435
表 3-669. 函数 usart_receive_fifo_enable	436
表 3-670. 函数 usart_receive_fifo_disable	436
表 3-671. 函数 usart_receive_fifo_counter_number	437
表 3-672. 函数 usart_flag_get	437
表 3-673. 函数 usart_flag_clear	439
表 3-674. 函数 usart_interrupt_enable	440
表 3-675. 函数 usart_interrupt_disable	441
表 3-676. 函数 usart_interrupt_flag_get	441
表 3-677. 函数 usart_interrupt_flag_clear	443
表 3-678. WWDGT 寄存器	444
表 3-679. WWDGT 库函数	444
表 3-680. 函数 wwdgt_deinit	445
表 3-681. 函数 wwdgt_enable	445
表 3-682. 函数 wwdgt_counter_update	446
表 3-683. 函数 wwdgt_config	446
表 3-684. 函数 wwdgt_interrupt_enable	447
表 3-685. 函数 wwdgt_flag_get	447
表 3-686. 函数 wwdgt_flag_clear	448
表 4-1. 版本历史	450

1. 介绍

本手册介绍了基于ARM处理器的32位微控制器GD32F3x0的固件库。

该固件库是一个固件函数包，它由程序、数据结构和宏组成，包括了GD32F3x0所有外设的性能特征。该固件库还包括每一个外设的驱动描述和基于评估板的固件库使用例程。通过使用本固件库，用户无需深入掌握细节，也可以轻松应用每一个外设。使用本固件库可以大大减少用户的编程时间，从而降低开发成本。

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

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

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

此份固件库使用手册的整体架构如下：

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

1.1. 文档和固件库规则

1.1.1. 外设缩写

表 1-1. 外设缩写

外设缩写	说明
ADC	模数转换器
CEC	HDMI-CEC控制器
CMP	比较器
CRC	循环冗余校验计算单元
CTC	时钟校准控制器
DAC	数模转换器
DBG	调试模块
DMA	直接存储器访问控制器
EXTI	外部中断事件控制器
FMC	闪存控制器
FWDGT	独立看门狗
GPIO/AFO	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
TSI	触摸传感控制器
USART	通用同步异步收发器
WWDGT	窗口看门狗
USBFS	通用串行总线全速接口

1.1.2. 命名规则

固件库遵从以下命名规则：

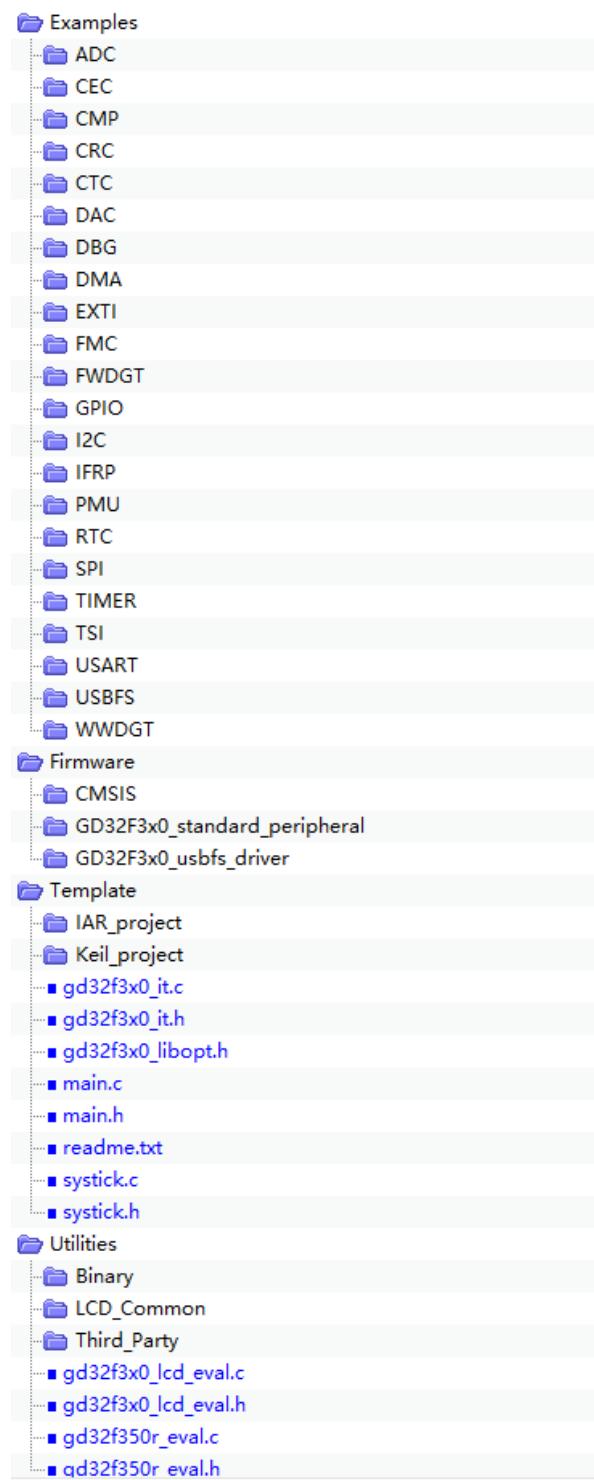
- XXX表示任一外设缩写，例如：ADC。更多缩写相关信息参阅[外设缩写](#)；
- 源文件和头文件命名都以“GD32f3x0_”作为开头，例如：GD32f3x0_adc.h；
- 常量仅被应用于一个文件的，定义于该文件中；被应用于多个文件的，在对应头文件中定义。所有常量都由英文字母大写书写；
- 寄存器作为常量处理。他们的命名都由英文字母大写书写。在大多数情况下，寄存器缩写规范与本用户手册一致；
- 变量名采用全部小写，有多个单词组成的，在单词之间以下划线分隔；
- 外设函数的命名以该外设的缩写加下划线为开头，有多个单词组成的，在单词之间以下划线分隔，所有外设函数都由英文字母小写书写。

2. 固件库概述

2.1. 文件组织结构

GD32F3x0_Firmware_Library, 文件组织结构见下图:

图 2-1. GD32F3x0 固件库文件组织结构



2.1.1. Examples 文件夹

文件夹**Examples**, 对应每一个GD32外设均包含一个子文件夹。每个子文件夹包含了关于本外设的一个或多个例程, 来示范如何使用对应外设。每个例程子文件夹包含如下文件:

- **readme.txt**: 关于本例程的简单描述和使用说明;
- **GD32f3x0_libopt.h**: 该头文件可以设置例程所使用到的外设, 由不同的“**DEFINE**”语句组成(默认情况下, 所有外设均打开);
- **GD32f3x0_it.c**: 该源文件包含了所有的中断处理程序(如果未使用到中断, 则所有的函数体都为空);
- **GD32f3x0_it.h**: 该头文件包含了所有的中断处理程序的原形;
- **systick.c**: 该源文件包含了使用**systick**的精准延时程序;
- **systick.h**: 该头文件包含了使用**systick**的精准延时程序的原形;
- **main.c**: 例程代码注: 所有的例程的使用, 都不受不同软件开发环境的影响。

2.1.2. Firmware 文件夹

Firmware文件夹包含组成固件库核心的所有子文件夹和文件:

- CMSIS子文件夹包含有**Cortex M4**内核的支持文件、基于**Cortex M4**内核处理器的启动代码和库引导文件以及基于GD32F3x0的全局头文件和系统配置文件;
- **GD32F3x0_standard_peripheral**子文件夹:
 - **Include** 子文件夹包含了固件函数库所需的头文件, 用户无需修改该文件夹;
 - **Source** 子文件夹包含了固件函数库所需的源文件, 用户无需修改该文件夹;

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

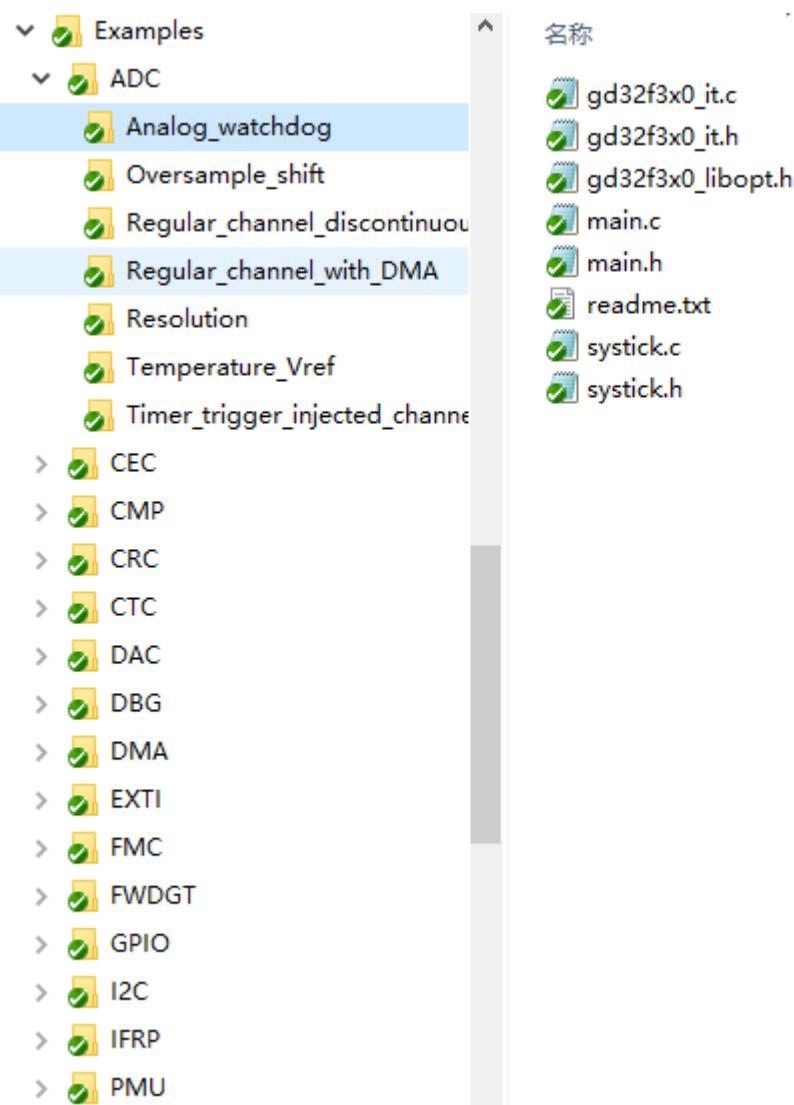
2.1.3. Template 文件夹

Template文件夹包含一个关于使用**LED**、**USART**打印、按键控制的简单例程, (**IAR_project**用于**IAR**编译环境, **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:

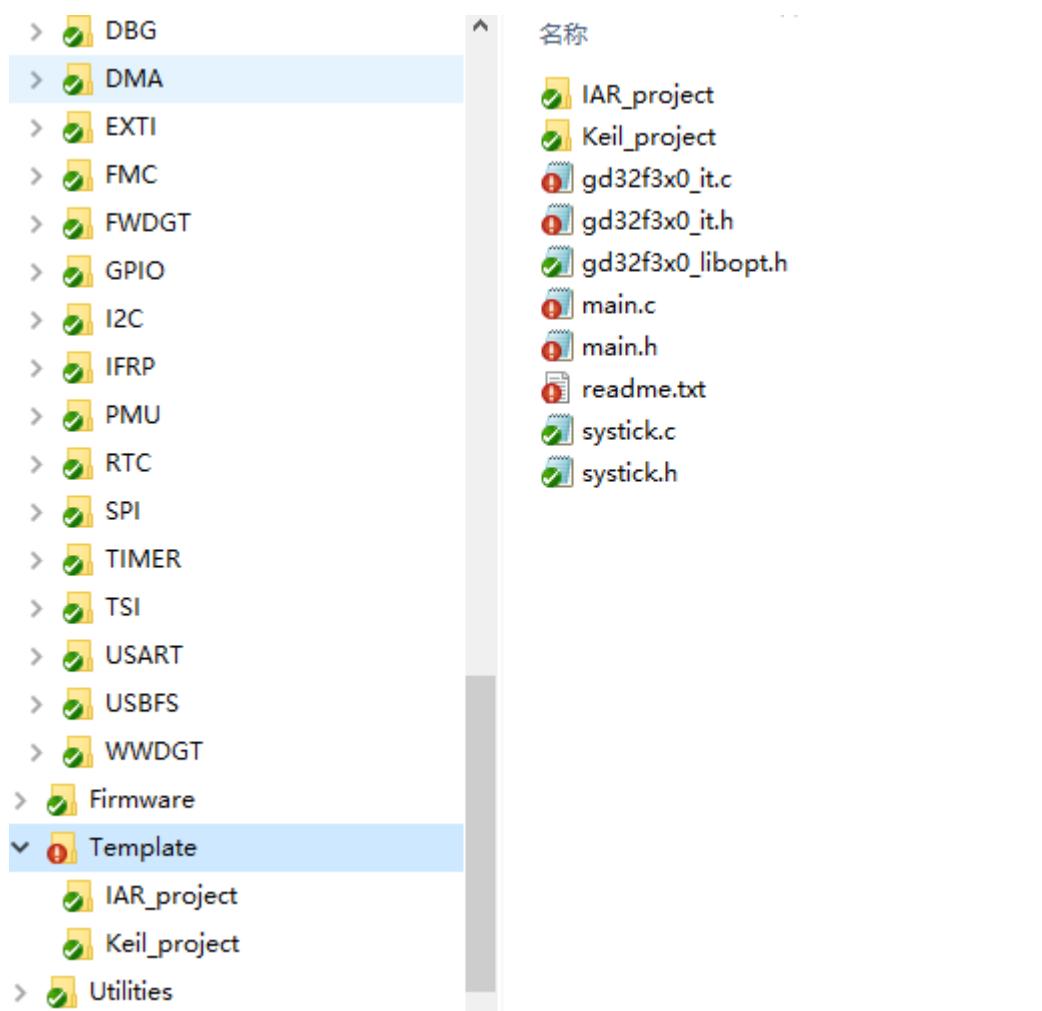
图 2-2. 选择外设例程文件



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：

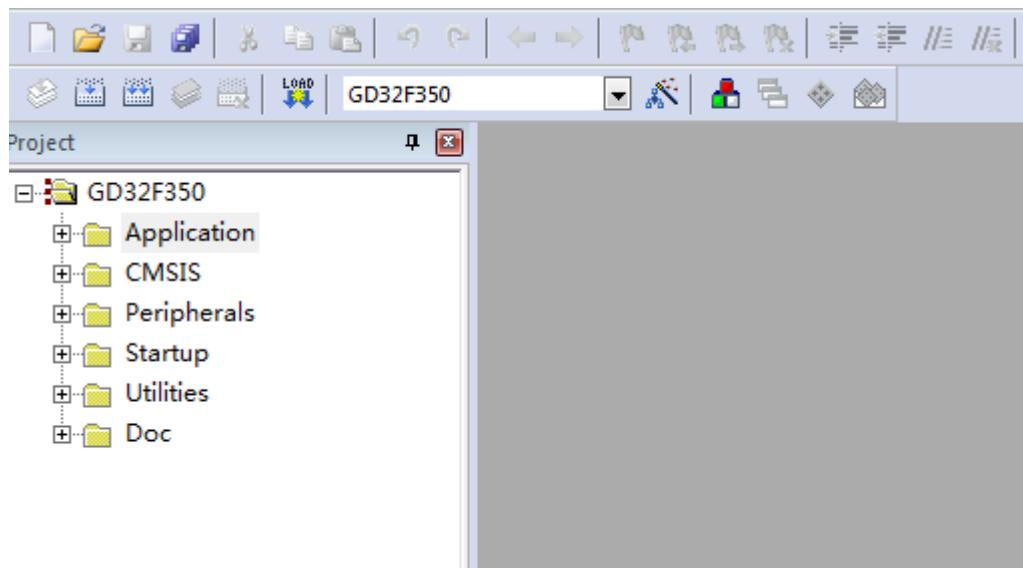
图 2-3. 拷贝外设例程文件



打开工程

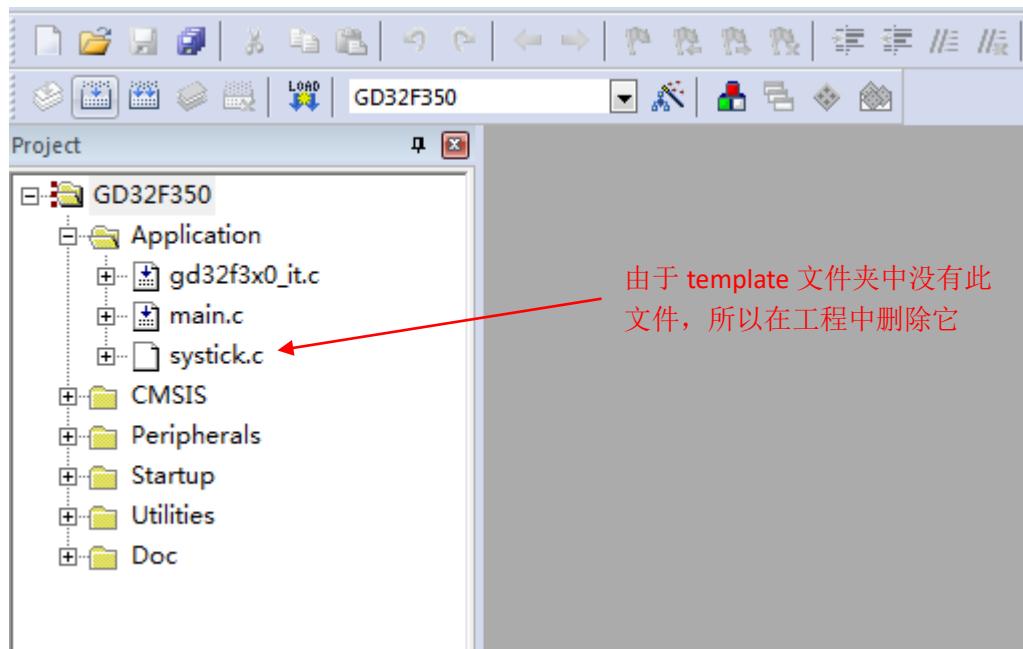
GD 提供 Keil 和 IAR 两种版本的工程，根据客户所安装的软件，打开不同的 project，如“Keil_project”，打开\Template\Keil_project\Project.uvproj，如下图所示：

图 2-4. 打开工程文件



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

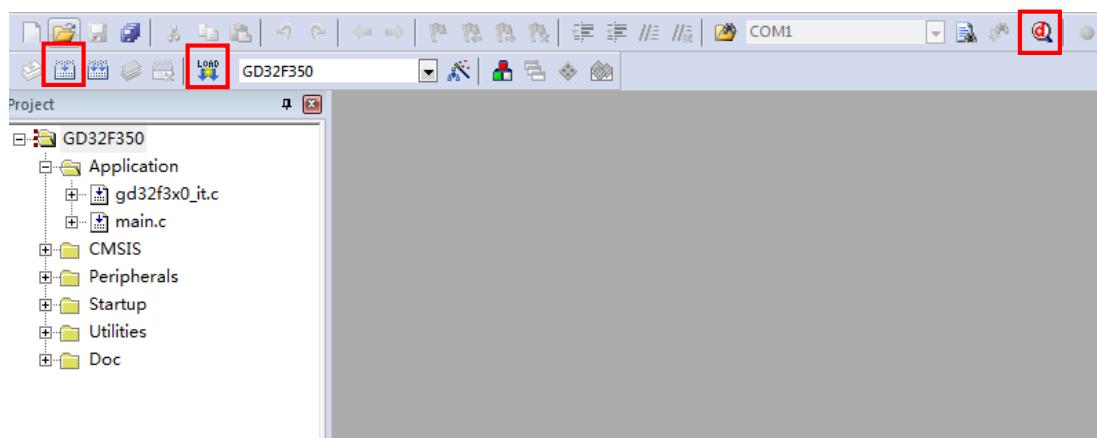
图 2-5. 配置工程文件



编译调试下载

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

- Binary、LCD_Common及Third_Party子文件夹；
- GD32f3x0r_eval.h及GD32f3x0_lcd_eval.h文件是运行固件库例程所需关于评估板的头文件；
- GD32f3x0r_eval.c及GD32f3x0_lcd_eval.c文件是运行固件库例程所需关于评估板的源文件。

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

2.2. 固件库文件描述

下表列举和描述了固件库使用的主要文件。

表 2-1. 固件函数库文件描述

文件名	描述
gd32f3x0_libopt.h	包含了所有外设的头文件的头文件。它是唯一一个用户需要包括在自己应用中的文件，起到应用和库之间界面的作用。
main.c	主函数体示例。
gd32f3x0_it.h	头文件，包含所有中断处理函数原形。
gd32f3x0_it.c	外设中断函数文件。用户可以加入自己的中断程序代码。对于指向同一个中断向量的多个不同中断请求，可以利用函数通过判断外设的中断标志位来确定准确的中断源。固件库提供了这些函数的名称。
gd32f3x0_xxx.h	外设xxx的头文件。包含外设xxx函数的定义，以及这些函数使用的变量。
gd32f3x0_xxx.c	由C语言编写的外设PPP的驱动源程序文件。
systick.h	systick.c的头文件。包含systick配置函数的定义，以及外部用延时函数的定义。
systick.c	systick配置与延时函数源文件。
readme.txt	固件库例程使用及配置说明文档。

3. 外设固件库

3.1. 外设固件库概述

外设固件库函数的描述格式如下表：

表 3-1. 外设固件库函数描述格式

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

3.2. ADC

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

3.2.1. 外设寄存器描述

ADC寄存器列表如下表所示：

表 3-2. ADC 寄存器

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

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

3.2.2. 外设库函数说明

ADC库函数列表如下表所示：

表 3-3. ADC 库函数

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

库函数名称	库函数描述
adc_oversample_mode_enable	使能ADC过采样
adc_oversample_mode_disable	禁能ADC过采样
adc_flag_get	获取ADC标志位
adc_flag_clear	清除ADC标志位
adc_interrupt_enable	ADC中断使能
adc_interrupt_disable	ADC中断禁能
adc_interrupt_flag_get	获取ADC中断标志位
adc_interrupt_flag_clear	清除ADC中断标志位

函数 adc_deinit

函数adc_deinit描述见下表:

表 3-4. 函数 adc_deinit

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

例如:

```
/* reset ADC*/
adc_deinit();
```

函数 adc_enable

函数adc_enable描述见下表:

表 3-5. 函数 adc_enable

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

-	-
返回值	
-	-

例如：

```
/* enable ADC */
adc_enable();
```

函数 **adc_disable**

函数adc_disable描述见下表：

表 3-6. 函数 **adc_disable**

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

例如：

```
/* disable ADC */
adc_disable();
```

函数 **adc_calibration_enable**

函数adc_calibration_enable描述见下表：

表 3-7. 函数 **adc_calibration_enable**

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

-	-
---	---

例如：

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

adc_calibration_enable();
```

函数 **adc_dma_mode_enable**

函数adc_dma_mode_enable描述见下表：

表 3-8. 函数 **adc_dma_mode_enable**

函数名称	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_vbat_enable**

函数adc_vbat_enable描述见下表:

表 3-12. 函数 adc_vbat_enable

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

例如:

```

/* enable the vbat channel */

adc_vbat_enable();

```

函数 **adc_vbat_disable**

函数adc_vbat_disable描述见下表:

表 3-13. 函数 adc_vbat_disable

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

例如:

```

/* disable the vbat channel */

```

```
adc_vbat_disable();
```

函数 **adc_discontinuous_mode_config**

函数adc_discontinuous_mode_config描述见下表:

表 3-14. 函数 adc_discontinuous_mode_config

函数名称	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	-
输入参数{in}	
adc_channel_group	通道组选择
ADC_REGULAR_CHANNEL	规则通道组
ADC_INSERTED_CHANNEL	注入通道组
ADC_CHANNEL_DISCON_DISABLE	规则通道组和注入通道组间断模式禁能
输入参数{in}	
length	间断模式下的转换数目，规则通道组取值为1..8，注入通道组取值无意义
输出参数{out}	
-	-
返回值	
-	-

例如:

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

函数 **adc_special_function_config**

函数adc_special_function_config描述见下表:

表 3-15. 函数 adc_special_function_config

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

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

例如：

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

函数 **adc_data_alignment_config**

函数adc_alignment_config描述见下表：

表 3-16. 函数 adc_data_alignment_config

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

例如：

```
/* configure ADC data alignment */
adc_data_alignment_config(ADC_DATAALIGN_RIGHT);
```

函数 adc_channel_length_config

函数adc_channel_length_config描述见下表:

表 3-17. 函数 adc_channel_length_config

函数名称	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}	
channel_group	通道组选择
ADC_REGULAR_CHANNEL	规则通道组
ADC_INSERTED_CHANNEL	注入通道组
输入参数{in}	
length	通道长度, 规则通道组为1-16, 注入通道组为1-4
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* configure the length of ADC regular channel */
adc_channel_length_config(ADC_REGULAR_CHANNEL, 4);
```

函数 adc_regular_channel_config

函数adc_regular_channel_config描述见下表:

表 3-18. 函数 adc_regular_channel_config

函数名称	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}	
rank	规则组通道序列, 取值范围为0~15
输入参数{in}	
channel	ADC通道选择
ADC_CHANNEL_x	ADC通道x(x=0..18)
输入参数{in}	

sample_time	采样时间
<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
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* configure ADC regular channel */
adc_regular_channel_config(1, ADC_CHANNEL_0, ADC_SAMPLETIME_7POINT5);
```

函数 **adc_inserted_channel_config**

函数adc_inserted_channel_config描述见下表：

表 3-19. 函数 **adc_inserted_channel_config**

函数名称	adc_inserted_channel_config
函数原形	void adc_inserted_channel_config(uint8_t rank, uint8_t channel, uint32_t sample_time);
功能描述	配置ADC注入通道组
先决条件	-
被调用函数	-
输入参数{in}	
rank	注入组通道序列，取值范围为0~3
输入参数{in}	
channel	ADC通道选择
<i>ADC_CHANNEL_x</i>	ADC通道x (x=0..18)
输入参数{in}	

sample_time	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
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* configure ADC inserted channel */
adc_inserted_channel_config(1, ADC_CHANNEL_0, ADC_SAMPLETIME_7POINT5);
```

函数 **adc_inserted_channel_offset_config**

函数adc_inserted_channel_offset_config描述见下表：

表 3-20. 函数 **adc_inserted_channel_offset_config**

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

输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* configure ADC inserted channel offset */
adc_inserted_channel_offset_config(ADC_INSERTED_CHANNEL_0, 100);
```

函数 **adc_external_trigger_config**

函数adc_external_trigger_config描述见下表：

表 3-21. 函数 **adc_external_trigger_config**

函数名称	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}	
channel_group	通道组选择
ADC_REGULAR_CHANNEL	规则通道组
ADC_INSERTED_CHANNEL	注入通道组
输入参数{in}	
newvalue	通道使能禁能
ENABLE	使能
DISABLE	禁能
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* enable ADC inserted channel group external trigger */
adc_external_trigger_config(ADC_INSERTED_CHANNEL_0, ENABLE);
```

函数 **adc_external_trigger_source_config**

函数adc_external_trigger_source_config描述见下表：

表 3-22. 函数 adc_external_trigger_source_config

函数名称	adc_external_trigger_source_config
函数原形	void adc_external_trigger_source_config(uint8_t channel_group, uint32_t external_trigger_source);
功能描述	配置ADC外部触发源
先决条件	-
被调用函数	-
输入参数{in}	
channel_group	通道组选择
ADC_REGULAR_CHANNEL	规则通道组
ADC_INSERTED_CHANNEL	注入通道组
输入参数{in}	
external_trigger_source	规则通道组或注入通道组触发源
ADC_EXTTRIG_REGULAR_T0_CH0	TIMER0 CH0事件（规则组）
ADC_EXTTRIG_REGULAR_T0_CH1	TIMER0 CH1事件（规则组）
ADC_EXTTRIG_REGULAR_T0_CH2	TIMER0 CH2事件（规则组）
ADC_EXTTRIG_REGULAR_T1_CH1	TIMER1 CH1事件（规则组）
ADC_EXTTRIG_REGULAR_T2_TRGO	TIMER2 TRGO事件（规则组）
ADC_EXTTRIG_REGULAR_T14_CH0	TIMER14 CH0事件（规则组）
ADC_EXTTRIG_REGULAR_EXTI_11	外部中断线11（规则组）
ADC_EXTTRIG_REGULAR_NONE	软件触发（规则组）
ADC_EXTTRIG_INSERTED_T0_TRGO	TIMER0 TRGO事件（注入组）
ADC_EXTTRIG_INSERTED_T0_CH3	TIMER0 CH3事件（注入组）
ADC_EXTTRIG_INSERTED_T1_	TIMER1 TRGO事件（注入组）

<i>TRGO</i>	
<i>ADC_EXTTRIG_INSERTED_T1_CH0</i>	TIMER1 CH0事件（注入组）
<i>ADC_EXTTRIG_INSERTED_T2_CH3</i>	TIMER2 CH3事件（注入组）
<i>ADC_EXTTRIG_INSERTED_T14_TRGO</i>	TIMER14 TRGO事件（注入组）
<i>ADC_EXTTRIG_INSERTED EXTI_15</i>	外部中断线15（注入组）
<i>ADC_EXTTRIG_INSERTED_NONE</i>	软件触发（注入组）
输出参数{out}	
-	-
返回值	
-	-

例如：

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

函数 **adc_software_trigger_enable**

函数adc_software_trigger_enable描述见下表：

表 3-23. 函数 **adc_software_trigger_enable**

函数名称	adc_software_trigger_enable
函数原形	void adc_software_trigger_enable(uint8_t channel_group);
功能描述	ADC软件触发使能
先决条件	-
被调用函数	-
输入参数{in}	
channel_group	通道组选择
ADC_REGULAR_CHANNEL	规则通道组
ADC_INSERTED_CHANNEL	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 3-24. 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	-
输出参数{out}	
-	-
返回值	
uint16_t	ADC转换值 (0-0xFFFF)

例如：

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

函数 **adc_inserted_data_read**

函数adc_inserted_regular_data_read描述见下表：

表 3-25. 函数 **adc_inserted_data_read**

函数名称	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	-
输入参数{in}	
inserted_channel	注入通道选择
ADC_INSERTED_CHANNEL_x	注入通道x, x=0,1,2,3
输出参数{out}	
-	-

返回值	
uint16_t	ADC转换值(0-0xFFFF)

例如：

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

uint16_t adc_value = 0;

adc_value = adc_inserted_data_read(ADC_INSERTED_CHANNEL_0);
```

函数 **adc_watchdog_single_channel_enable**

函数adc_watchdog_single_channel_enable描述见下表：

表 3-26. 函数 **adc_watchdog_single_channel_enable**

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

例如：

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

adc_watchdog_single_channel_enable(ADC_CHANNEL_1);
```

函数 **adc_watchdog_group_channel_enable**

函数adc_watchdog_group_channel_enable描述见下表：

表 3-27. 函数 **adc_watchdog_group_channel_enable**

函数名称	adc_watchdog_group_channel_enable
函数原形	void adc_watchdog_group_channel_enable(uint8_t channel_group);
功能描述	配置ADC模拟看门狗在通道组有效
先决条件	-
被调用函数	-
输入参数{in}	
channel_group	通道组使用模拟看门狗
ADC_REGULAR_	规则通道组

<i>CHANNEL</i>	
<i>ADC_INSERTED_CHANNEL</i>	注入通道组
<i>ADC_REGULAR_INSERTED_CHANNEL</i>	规则和注入通道组
输出参数{out}	
-	-
返回值	
-	-

例如：

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

函数 **adc_watchdog_disable**

函数adc_watchdog_disable描述见下表：

表 3-28. 函数 **adc_watchdog_disable**

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

例如：

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

函数 **adc_watchdog_threshold_config**

函数adc_watchdog_threshold_config描述见下表：

表 3-29. 函数 **adc_watchdog_threshold_config**

函数名称	adc_watchdog_threshold_config
函数原形	void adc_watchdog_threshold_config(uint16_t low_threshold, uint16_t high_threshold);

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

例如：

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

函数 **adc_resolution_config**

函数adc_resolution_config描述见下表：

表 3-30. 函数 **adc_resolution_config**

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

Example:

```
/* configure ADC resolution */
```

```
adc_resolution_config(ADC_RESOLUTION_12B);
```

函数 **adc_oversample_mode_config**

函数adc_oversample_mode_config描述见下表：

表 3-31. 函数 adc_oversample_mode_config

函数名称	adc_oversample_mode_config
函数原形	void adc_oversample_mode_config(uint32_t mode, uint16_t shift, uint8_t ratio);
功能描述	配置ADC过采样模式
先决条件	-
被调用函数	-
输入参数{in}	
mode	ADC过采样触发模式
<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}	
shift	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_OVERSAMPLE_SHIFT_2B</i>	移2位
<i>ADC_OVERSAMPLE_SHIFT_3B</i>	移3位
<i>ADC_OVERSAMPLE_SHIFT_4B</i>	移4位
<i>ADC_OVERSAMPLE_SHIFT_5B</i>	移5位
<i>ADC_OVERSAMPLE_SHIFT_6B</i>	移6位
<i>ADC_OVERSAMPLE_SHIFT_7B</i>	移7位

<i>MPLING_SHIFT_7B</i>	
<i>ADC_OVERSA_MPLING_SHIFT_8B</i>	移8位
输入参数{in}	
ratio	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_MUL128</i>	128x
<i>ADC_OVERSA_MPLING_RATIO_MUL256</i>	256x
输出参数{out}	
-	-
返回值	
-	-

Example:

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

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

函数 adc_oversample_mode_enable

函数adc_oversample_mode_enable描述见下表:

表 3-32. 函数 adc_oversample_mode_enable

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

Example:

```
/* enable ADC oversample mode */
adc_oversample_mode_enable();
```

函数 adc_oversample_mode_disable

函数adc_oversample_mode_disable描述见下表:

表 3-33. 函数 adc_oversample_mode_disable

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

Example:

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

函数 adc_flag_get

函数adc_flag_get描述见下表:

表 3-34. 函数 adc_flag_get

函数名称	adc_flag_get
函数原形	FlagStatus adc_flag_get(uint32_t flag);
功能描述	获取ADC标志位
先决条件	-
被调用函数	-
输入参数{in}	
flag	ADC标志位
ADC_FLAG_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
输出参数{out}	
-	-
返回值	
FlagStatus	SET 或 RESET

例如:

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

函数 adc_flag_clear

函数adc_flag_clear描述见下表:

表 3-35. 函数 adc_flag_clear

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

<i>ADC_FLAG_STRC</i>	规则通道组转换开始标志位
输出参数{out}	
-	-
返回值	
-	-

例如：

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

函数 **adc_interrupt_enable**

函数adc_interrupt_enable描述见下表：

表 3-36. 函数 **adc_interrupt_enable**

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

例如：

```
/* enable ADC analog watchdog interrupt */
adc_interrupt_enable(ADC_INT_WDE);
```

函数 **adc_interrupt_disable**

函数adc_interrupt_disable描述见下表：

表 3-37. 函数 **adc_interrupt_disable**

函数名称	adc_interrupt_disable
函数原形	void adc_interrupt_disable(uint32_t interrupt);
功能描述	ADC中断禁能
先决条件	-
被调用函数	-

输入参数{in}	
interrupt	ADC中断标志位
<i>ADC_INT_WDE</i>	模拟看门狗中断标志位
<i>ADC_INT_EOC</i>	组转换结束中断标志位
<i>ADC_INT_EOIC</i>	注入通道组转换结束中断标志位
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* disable ADC interrupt */

adc_interrupt_disable(ADC_INT_WDE);
```

函数 **adc_interrupt_flag_get**

函数adc_interrupt_flag_get描述见下表：

表 3-38. 函数 **adc_interrupt_flag_get**

函数名称	adc_interrupt_flag_get
函数原形	FlagStatus adc_interrupt_flag_get(uint32_t flag);
功能描述	获取ADC中断标志位
先决条件	-
被调用函数	-
输入参数{in}	
flag	ADC中断标志位
<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
输出参数{out}	
-	-
返回值	
FlagStatus	SET 或 RESET

例如：

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

FlagStatus flag_value;

flag_value = adc_interrupt_flag_get(ADC_INT_FLAG_WDE);
```

函数 adc_interrupt_flag_clear

函数adc_interrupt_flag_clear描述见下表:

表 3-39. 函数 adc_interrupt_flag_clear

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

例如:

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

3.3. CEC

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

3.3.1. 外设寄存器描述

CEC寄存器列表如下表所示:

表 3-40. CEC 寄存器

寄存器名称	寄存器描述
CEC_CTL	控制寄存器
CEC_CFG	配置寄存器
CEC_TDATA	数据发送寄存器
CEC_RDATA	数据接收寄存器

寄存器名称	寄存器描述
CEC_INTF	中断标志寄存器
CEC_INTEN	中断使能寄存器

3.3.2. 外设库函数说明

CEC库函数列表如下表所示：

表 3-41. CEC 库函数

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

函数 `cec_deinit`

函数`cec_deinit`描述见下表：

表 3-42. 函数 `cec_deinit`

函数名称	<code>cec_deinit</code>
函数原形	<code>void cec_deinit(void);</code>
功能描述	复位HDMI-CEC外设
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	
-	-

例如：

```
/* reset CEC*/
cec_deinit();
```

函数 **cec_init**

函数 **cec_init** 描述见下表：

表 3-43. 函数 **cec_init**

函数名称	cec_init
函数原形	void cec_init(uint32_t sftmopt, uint32_t sft, uint32_t address);
功能描述	HDMI-CEC 初始化各个参数
先决条件	-
被调用函数	-
输入参数{in}	
sftmopt	SFT开始选项位
CEC_SFT_START_STAOM	STAOM置1后SFT计数器开始计数
CEC_SFT_START_LAST	发收/接收结束后SFT计数器自动启动
输入参数{in}	
sft	信号空闲时间
CEC_SFT_PROTO_COL_PERIOD	SFT时间跟HDMI-CEC协议描述的一样
CEC_SFT_1POINT5_PERIOD	1.5个额定数据位周期
CEC_SFT_2POINT5_PERIOD	2.5个额定数据位周期
CEC_SFT_3POINT5_PERIOD	3.5个额定数据位周期
CEC_SFT_4POINT5_PERIOD	4.5个额定数据位周期
CEC_SFT_5POINT5_PERIOD	5.5个额定数据位周期
CEC_SFT_6POINT5_PERIOD	6.5个额定数据位周期

<i>CEC_SFT_7POINT_5_PERIOD</i>	7.5个额定数据位周期
输入参数{in}	
address	地址设置
<i>CEC_OWN_ADDRESS_CLEAR</i>	清除地址
<i>CEC_OWN_ADDRESS_SSx</i>	自身地址 (x=0-14)
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* init CEC*/
cec_init(CEC_SFT_START_STAOM, CEC_SFT_PROTOCOL_PERIOD,
CEC_OWN_ADDRESS0);
```

函数 **cec_error_config**

函数**cec_error_config**描述见下表：

表 3-44. 函数 **cec_error_config**

函数名称	cec_error_config
函数原形	void cec_error_config(uint32_t broadcast, uint32_t singlecast_lbpe, uint32_t singlecast_bre, uint32_t rxbrestp);
功能描述	检测到错误时进行相关配置
先决条件	-
被调用函数	-
输入参数{in}	
broadcast	广播信息模式下是否产生错误位
<i>CEC_BROADCAST_ERROR_BIT_ON</i>	广播信息模式下产生错误位
<i>CEC_BROADCAST_ERROR_BIT_OFF</i>	广播信息模式下不产生错误位
输入参数{in}	
singlecast_lbpe	在单播模式下监测到BPLE的时候是否产生错误位
<i>CEC_LONG_PERIOD_ERROR_BIT_ON</i>	在单播模式下监测到BPLE时产生错误位
<i>CEC_LONG_PERIOD_ERROR_BIT_OFF</i>	在单播模式下监测到BPLE时不产生错误位
输入参数{in}	

singlecast_bre	在单次传播模式下监测到BRE的时候是否产生错误位
<i>CEC_RISING_PERIOD_N</i>	在单播模式下监测到BRE的时候产生错误位
<i>CEC_RISING_PERIOD_FF</i>	在单播模式下监测到BRE的时候不产生错误位
输入参数{in}	
rxbrestp	监测到BRE时是否停止接收信息
<i>CEC_STOP_RISING_G_ERROR_BIT_ON</i>	停止接收BRE下的信息
<i>CEC_STOP_RISING_G_ERROR_BIT_OF_F</i>	不停止接收BRE下的信息，数据位按额定时间采样（1.05ms）
输出参数{out}	
-	-
返回值	
-	-

例如：

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

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

函数 **cec_enable**

函数**cec_enable**描述见下表：

表 3-45. 函数 cec_enable

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

例如：

```

/* enable HDMI-CEC controller */

cec_enable();

```

函数 **cec_disable**

函数**cec_disable**描述见下表：

表 3-46. 函数 cec_disable

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

例如：

```

/* disable HDMI-CEC controller */

cec_disable ();

```

函数 **cec_transmission_start**

函数**cec_transmission_start**描述见下表：

表 3-47. 函数 cec_transmission_start

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

例如：

```

/* start CEC message transmission */

```

```
cec_transmission_start();
```

函数 **cec_transmission_end**

函数cec_transmission_end描述见下表:

表 3-48. 函数 cec_transmission_end

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

例如:

```
/* end CEC message transmission */

cec_transmission_end();
```

函数 **cec_listen_mode_enable**

函数cec_listen_mode_enable描述见下表:

表 3-49. 函数 cec_listen_mode_enable

函数名称	cec_listen_mode_enable
函数原形	void cec_listen_mode_enable (void);
功能描述	使能CEC监听模式
先决条件	-
被调用函数	-
输入参数{in}	
-	-
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* enable CEC listen mode */

cec_listen_mode_enable();
```

函数 `cec_listen_mode_disable`

函数`cec_listen_mode_disable`描述见下表:

表 3-50. 函数 `cec_listen_mode_disable`

函数名称	cec_listen_mode_disable
函数原形	<code>void cec_listen_mode_disable (void);</code>
功能描述	关闭CEC监听模式
先决条件	-
被调用函数	-
输入参数{in}	
-	-
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* disable CEC listen mode */

cec_listen_mode_disable();
```

函数 `cec_own_address_config`

函数`cec_own_address_config`描述见下表:

表 3-51. 函数 `cec_own_address_config`

函数名称	cec_own_address_config
函数原形	<code>void cec_own_address_config(uint32_t address);</code>
功能描述	CEC自身地址配置
先决条件	-
被调用函数	-
输入参数{in}	
<code>address</code>	地址选择
<code>CEC_OWN_ADDRESS_CLEAR</code>	清除地址
<code>CEC_OWN_ADDRESS_SSx</code>	自身地址 (x=0-14)
输出参数{out}	
-	-
返回值	
-	-

例如:

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

 */

```
cec_own_address_config (CEC_OWN_ADDRESS_CLEAR);
```

函数 cec_sft_config

函数cec_sft_config描述见下表:

表 3-52. 函数 cec_sft_config

函数名称	cec_sft_config
函数原形	void cec_sft_config(uint32_t sftmopt, uint32_t sft);
功能描述	配置CEC信号空闲时间
先决条件	-
被调用函数	-
输入参数{in}	
sftmopt	SFT开始选项位
CEC_SFT_START_STAOM	STAOM置1后SFT计数器开始计数
CEC_SFT_START_LAST	发收/接收结束后SFT计数器自动启动
输入参数{in}	
sft	信号空闲时间
CEC_SFT_PROTO_COL_PERIOD	SFT时间跟HDMI-CEC协议描述的一样
CEC_SFT_1POINT_5_PERIOD	1.5个额定数据位周期
CEC_SFT_2POINT_5_PERIOD	2.5个额定数据位周期
CEC_SFT_3POINT_5_PERIOD	3.5个额定数据位周期
CEC_SFT_4POINT_5_PERIOD	4.5个额定数据位周期
CEC_SFT_5POINT_5_PERIOD	5.5个额定数据位周期
CEC_SFT_6POINT_5_PERIOD	6.5个额定数据位周期
CEC_SFT_7POINT_5_PERIOD	7.5个额定数据位周期
输出参数{out}	
-	-
返回值	
-	-

例如:

```

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

cec_sft_config (CEC_SFT_START_STAOM, CEC_SFT_PROTOCOL_PERIOD);

```

函数 cec_generate_errorbit_config

函数cec_generate_errorbit_config描述见下表:

表 3-53. 函数 cec_generate_errorbit_config

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

例如:

```

/* configure generate Error-bit when detected some abnormal situation or not */

cec_generate_errorbit_config (CEC_BROADCAST_ERROR_BIT_ON,

```

```
CEC_LONG_PERIOD_ERROR_BIT_ON, CEC_RISING_PERIOD_ERROR_BIT_ON);
```

函数 `cec_stop_receive_bre_config`

函数`cec_stop_receive_bre_config`描述见下表：

表 3-54. 函数 `cec_stop_receive_bre_config`

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

例如：

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

函数 `cec_reception_tolerance_enable`

函数`cec_reception_tolerance_enable`描述见下表：

表 3-55. 函数 `cec_reception_tolerance_enable`

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

例如：

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

```
cec_reception_tolerance_enable();
```

函数 **cec_reception_tolerance_disable**

函数 **cec_reception_tolerance_disable** 描述见下表：

表 3-56. 函数 cec_reception_tolerance_disable

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

例如：

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

```
cec_reception_tolerance_disable();
```

函数 **cec_data_send**

函数 **cec_data_send** 描述见下表：

表 3-57. 函数 cec_data_send

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

例如：

```

/* send a data by the CEC peripheral */

cec_data_send (0x55);

```

函数 cec_data_receive

函数cecc_data_receive描述见下表:

表 3-58. 函数 cec_data_receive

函数名称	cec_data_receive
函数原形	uint8_t cec_data_receive(void);
功能描述	CEC数据接收
先决条件	-
被调用函数	-
输入参数{in}	
-	-
输出参数{out}	
-	-
返回值	
Uint8_t	CEC接收到的数据值

例如:

```

/* receive a data by the CEC peripheral */

uint8_t data = 0;

data = cec_data_receive ();

```

函数 cec_interrupt_enable

函数cecc_interrupt_enable描述见下表:

表 3-59. 函数 cec_interrupt_enable

函数名称	cec_interrupt_enable
函数原形	void cec_interrupt_enable(uint32_t flag);
功能描述	CEC中断使能
先决条件	-
被调用函数	-
输入参数{in}	
flag	中断使能位
CEC_INT_BR	字节接收中断使能
CEC_INT_REND	接收结束中断使能
CEC_INT_RO	接收过载中断使能
CEC_INT_BRE	位上升错误中断使能
CEC_INT_BPSE	短位周期错误中断使能
CEC_INT_BPLE	长位周期错误中断使能

<i>CEC_INT_RAE</i>	接收ACK错误中断使能
<i>CEC_INT_ARBF</i>	仲裁失败中断使能
<i>CEC_INT_TBR</i>	发送字节数据请求中断使能
<i>CEC_INT_TEND</i>	发送成功结束中断使能
<i>CEC_INT_TU</i>	发送数据缓冲区欠载中断使能
<i>CEC_INT_TERR</i>	中断使能中断使能
<i>CEC_INT_TAERR</i>	发送ACK错误中断使能
输出参数{out}	
-	-
返回值	
-	-

例如：

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

函数 **cec_interrupt_disable**

函数 **cec_interrupt_disable** 描述见下表：

表 3-60. 函数 **cec_interrupt_disable**

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

返回值	
-	-

例如：

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

函数 cec_flag_get

函数cecc_flag_get描述见下表：

表 3-61. 函数 cec_flag_get

函数名称	cec_flag_get
函数原形	FlagStatus cec_flag_get(uint32_t flag);
功能描述	CEC标志位获取
先决条件	-
被调用函数	-
输入参数{in}	
flag	标志位
CEC_FLAG_BR	字节接收标志
CEC_FLAG_REND	中断使能标志
CEC_FLAG_RO	接收过载标志
CEC_FLAG_BRE	位上升错误标志
CEC_FLAG_BPSE	短位周期错误标志
CEC_FLAG_BPLE	长位周期错误标志
CEC_FLAG_RAE	接收ACK错误标志
CEC_FLAG_ARBF	仲裁失败标志
CEC_FLAG_TBR	发送字节数据请求标志
CEC_FLAG_TEND	发送成功结束标志
CEC_FLAG_TU	发送数据缓冲区欠载标志
CEC_FLAG_TERR	发送错误标志
CEC_FLAG_TAER R	发送ACK错误标志
输出参数{out}	
-	-
返回值	
FlagStatus	SET或者RESET

例如：

```
/* get CEC_FLAG_BR status */
FlagStatus flag = reset;
flag = cec_flag_get(CEC_INT_BR);
```

函数 cec_flag_clear

函数cec_flag_clear描述见下表:

表 3-62. 函数 cec_flag_clear

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

例如:

```
/* clear CEC_FLAG_BR status */
cec_flag_get(CEC_INT_BR);
```

函数 cec_interrupt_flag_get

函数cec_interrupt_flag_get描述见下表:

表 3-63. 函数 cec_interrupt_flag_get

函数名称	cec_interrupt_flag_get
函数原形	FlagStatus cec_interrupt_flag_get(uint32_t flag);
功能描述	CEC中断标志位获取
先决条件	-

被调用函数	-
输入参数{in}	
flag	中断标志位
<i>CEC_INT_FLAG_B_R</i>	字节接收中断标志
<i>CEC_INT_FLAG_R_END</i>	中断使能中断标志
<i>CEC_INT_FLAG_R_O</i>	接收过载中断标志
<i>CEC_INT_FLAG_B_RE</i>	位上升错误中断标志
<i>CEC_INT_FLAG_B_PSE</i>	短位周期错误中断标志
<i>CEC_INT_FLAG_B_PLE</i>	长位周期错误中断标志
<i>CEC_INT_FLAG_R_AE</i>	接收ACK错误中断标志
<i>CEC_INT_FLAG_A_RBF</i>	仲裁失败中断标志
<i>CEC_INT_FLAG_T_BR</i>	发送字节数据请求中断标志
<i>CEC_INT_FLAG_T_END</i>	发送成功结束中断标志
<i>CEC_INT_FLAG_T_U</i>	发送数据缓冲区欠载中断标志
<i>CEC_INT_FLAG_T_ERR</i>	中断使能中断标志
<i>CEC_INT_FLAG_T_AERR</i>	发送ACK错误中断标志
输出参数{out}	
-	-
返回值	
FlagStatus	SET或者RESET

例如：

```
/* get CEC_INT_FLAG_BR status */

FlagStatus flag = reset;

flag = cec_interrupt_flag_get (CEC_INT_FLAG_BR);
```

函数 **cec_interrupt_flag_clear**

函数 **cec_interrupt_flag_clear** 描述见下表：

表 3-64. 函数 cec_interrupt_flag_clear

函数名称	cec_interrupt_flag_clear
函数原形	void cec_interrupt_flag_clear(uint32_t flag);
功能描述	CEC中断标志位清除
先决条件	-
被调用函数	-
输入参数{in}	
flag	中断标志位
CEC_INT_FLAG_B_R	字节接收中断标志
CEC_INT_FLAG_R_END	中断使能中断标志
CEC_INT_FLAG_R_O	接收过载中断标志
CEC_INT_FLAG_B_RE	位上升错误中断标志
CEC_INT_FLAG_B_PSE	短位周期错误中断标志
CEC_INT_FLAG_B_PLE	长位周期错误中断标志
CEC_INT_FLAG_R_AE	接收ACK错误中断标志
CEC_INT_FLAG_A_RBF	仲裁失败中断标志
CEC_INT_FLAG_T_BR	发送字节数据请求中断标志
CEC_INT_FLAG_T_END	发送成功结束中断标志
CEC_INT_FLAG_T_U	发送数据缓冲区欠载中断标志
CEC_INT_FLAG_T_ERR	中断使能中断标志
CEC_INT_FLAG_T_AERR	发送ACK错误中断标志
输出参数{out}	
-	-
返回值	
-	-

例如：

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

3.4. CMP

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[3.4.1](#)描述了CMP的寄存器列表，章节[3.4.2](#)对CMP库函数进行说明

3.4.1. 外设寄存器说明

CMP寄存器列表如下表所示：

表 3-65. CMP 寄存器

寄存器名称	寄存器描述
CMP_CS	控制状态寄存器

3.4.2. 外设库函数说明

CMP库函数列表如下表所示：

表 3-66. CMP 库函数

库函数名称	库函数描述
cmp_deinit	复位CMP
cmp_mode_init	CMP工作模式初始化
cmp_output_init	CMP输出初始化
cmp_enable	使能CMP
cmp_disable	除能CMP
cmp_switch_enable	使能CMP开关
cmp_switch_disable	除能CMP开关
cmp_window_enable	使能CMP窗口模式
cmp_window_disable	禁能CMP窗口模式
cmp_lock_enable	锁定CMP
cmp_output_level_get	获取CMP输出状态

枚举类型 `operating_mode_enum`

表 3-67. 枚举类型 operating_mode_enum

成员名称	功能描述
CMP_HIGHSPEED	高速/全功耗
CMP_Middlespeed	中速/中功耗
CMP_Lowspeed	低速/低功耗
CMP_Verylowspeed	超低速/超低功耗

枚举类型 `inverting_input_enum`

表 3-68. 枚举类型 `inverting_input_enum`

成员名称	功能描述
CMP_1_4VREFINT	选择1/4V _{REFINT} 作为输入源
CMP_1_2VREFINT	选择1/2V _{REFINT} 作为输入源
CMP_3_4VREFINT	选择3/4V _{REFINT} 作为输入源
CMP_VREFINT	选择V _{REFINT} 作为输入源
CMP_DAC	选择PA4 (DAC) 作为输入源
CMP_PA5	选择PA5作为输入源
CMP_PA_0_2	选择PA0或者PA2作为输入源

枚举类型 `cmp_hysteresis_enum`

表 3-69. 枚举类型 `cmp_hysteresis_enum`

成员名称	功能描述
CMP_HYSTERESIS_NO	无迟滞
CMP_HYSTERESIS_LOW	低迟滞
CMP_HYSTERESIS_MIDDLE	中迟滞
CMP_HYSTERESIS_HIGH	高迟滞

枚举类型 `cmp_output_enum`

表 3-70. 枚举类型 `cmp_output_enum`

成员名称	功能描述
CMP_OUTPUT_NONE	无选择
CMP_OUTPUT_TIMER0BKin	TIMER0中止输入
CMP_OUTPUT_TIMER0IC0	TIMER0 channel0输入捕获
CMP_OUTPUT_TIMER0OCPREC_LR	TIMER0 OCPRE_CLR输入
CMP_OUTPUT_TIMER1IC3	TIMER1 channel3输入捕获
CMP_OUTPUT_TIMER1OCPREC_LR	TIMER1 OCPRE_CLR输入
CMP_OUTPUT_TIMER2IC0	TIMER2 channel0输入捕获
CMP_OUTPUT_TIMER2OCPREC_LR	TIMER2 OCPRE_CLR输入

函数 `cmp_deinit`

函数`cmp_deinit`描述见下表：

表 3-71. 函数 `cmp_deinit`

函数名称	<code>cmp_deinit</code>
函数原型	<code>void cmp_deinit(void);</code>
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 3-72. 函数 cmp_mode_init

函数名称	cmp_mode_init
函数原型	void cmp_mode_init(uint32_t cmp_periph, 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}	
cmp_periph	CMP外设
CMP0	比较器0
CMP1	比较器1
输入参数{in}	
operating_mode	CMP模式，具体参考 表3-67. 枚举类型operating_mode_enum
CMP_HIGHSPEED	高速/全功耗
CMP_Middlespe ed	中速/中功耗
CMP_LOWSPEED	低速/低功耗
CMP_VERYLOWSP eed	超低速/超低功耗
输入参数{in}	
inverting_input	CMP_IM输入选择，具体参考 表3-68. 枚举类型inverting_input_enum
CMP_1_4VREFINT	选择1/4V _{REFINT} 作为输入源
CMP_1_2VREFINT	选择1/2V _{REFINT} 作为输入源
CMP_3_4VREFINT	选择3/4V _{REFINT} 作为输入源
CMP_VREFINT	选择V _{REFINT} 作为输入源

<i>CMP_DAC</i>	选择PA4 (DAC) 作为输入源
<i>CMP_PA5</i>	选择PA5作为输入源
<i>CMP_PA_0_2</i>	选择PA0或者PA2作为输入源
输入参数{in}	
<i>output_hysteresis</i>	CMP输出迟滞, 具体参考 表3-69. 枚举类型cmp_hysteresis_enum
<i>CMP_HYSTERESIS_NO</i>	无迟滞
<i>CMP_HYSTERESIS_LOW</i>	低迟滞
<i>CMP_HYSTERESIS_MIDDLE</i>	中迟滞
<i>CMP_HYSTERESIS_HIGH</i>	高迟滞
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* CMP mode initialize*/
cmp_mode_init(CMP0,CMP_HIGHSPED,CMP_1_4VREFINT, CMP_HYSTERESIS_NO);
```

函数 cmp_output_init

函数cmp_output_init描述见下表:

表 3-73. 函数 cmp_output_init

函数名称	cmp_output_init
函数原型	void cmp_output_init(uint32_t cmp_periph, 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	-
输入参数{in}	
<i>cmp_periph</i>	CMP外设
<i>CMP0</i>	比较器0
<i>CMP1</i>	比较器1
输入参数{in}	
<i>output_selection</i>	CMP输出选择, 具体参考 表3-70. 枚举类型cmp_output_enum
<i>CMP_OUTPUT_NO_NE</i>	无选择
<i>CMP_OUTPUT_TIMER0_IN</i>	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>MEROOCPRECLR</i>	TIMER0 OCPRE_CLR输入
<i>CMP_OUTPUT_TI</i> <i>MER1IC3</i>	TIMER1 channel3输入捕获
<i>CMP_OUTPUT_TI</i> <i>MER1OCPRECLR</i>	TIMER1 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
输入参数{in}	
output_polarity	输出极性
<i>CMP_OUTPUT_POLARITY_INVERTED</i>	反相输出
<i>CMP_OUTPUT_POLARITY_NOINVERTED</i>	正相输出
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* CMP output initialize*/
cmp_output_init(CMP0,CMP_OUTPUT_TIMER0BKin,
CMP_OUTPUT_Polarity_NoInverted);
```

函数 cmp_enable

函数cmp_enable描述见下表：

表 3-74. 函数 cmp_enable

函数名称	cmp_enable
函数原型	void cmp_enable(uint32_t cmp_periph)
功能描述	使能CMP
先决条件	-
被调用函数	-
输入参数{in}	
cmp_periph	CMP外设
<i>CMP0</i>	比较器0
<i>CMP1</i>	比较器1

输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* enable CMP*/
cmp_enable(CMP0);
```

函数 cmp_disable

函数cmp_disable描述见下表：

表 3-75. 函数 cmp_disable

函数名称	cmp_disable
函数原型	void cmp_disable(uint32_t cmp_periph);
功能描述	除能CMP
先决条件	-
被调用函数	-
输入参数{in}	
cmp_periph	CMP外设
CMP0	比较器0
CMP1	比较器1
输出参数{out}	
-	-
返回值	
-	-

例如：

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

函数 cmp_switch_enable

函数cmp_switch_enable描述见下表：

表 3-76. 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 3-77. 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_window_enable

函数cmp_window_enable描述见下表：

表 3-78. 函数 cmp_window_enable

函数名称	cmp_window_enable
函数原型	void cmp_window_enable(void)
功能描述	使能CMP窗口模式
先决条件	-
被调用函数	-
输入参数{in}	
-	-
输出参数{out}	
-	-

返回值	
-	-

例如：

```
/* enable the window mode */

cmp_window_enable();
```

函数 cmp_window_disable

函数cmp_window_disable描述见下表：

表 3-79. 函数 cmp_window_disable

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

例如：

```
/* disable the window mode */

cmp_window_disable();
```

函数 cmp_lock_enable

函数cmp_lock_enable描述见下表：

表 3-80. 函数 cmp_lock_enable

函数名称	cmp_lock_enable
函数原型	void cmp_lock_enable(uint32_t cmp_periph)
功能描述	锁定CMP
先决条件	-
被调用函数	-
输入参数{in}	
cmp_periph	CMP外设
CMP0	比较器0
CMP1	比较器1
输出参数{out}	
-	-

返回值	
-	-

例如：

```
/* lock CMP0 register */
cmp_lock_enable(CMP0);
```

函数 cmp_output_level_get

函数cmp_output_level_get描述见下表：

表 3-81. 函数 cmp_output_level_get

函数名称	cmp_output_level_get
函数原型	uint32_t cmp_output_level_get(uint32_t cmp_periph)
功能描述	获取CMP输出状态
先决条件	-
被调用函数	-
输入参数{in}	
cmp_periph	CMP外设
CMP0	比较器0
CMP1	比较器1
输出参数{out}	
-	-
返回值	
uint32_t	CMP输出
CMP_OUTPUTLEV EL_HIGH	相输入端高于反相输入端，输出为高电平
CMP_OUTPUTLEV EL_LOW	相输入端低于反相输入端，输出为低电平

例如：

```
/* get CMP output level */
cmp_output_level_get (CMP0);
```

3.5. CRC

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

3.5.1. 外设寄存器说明

CRC寄存器列表如下表所示：

表 3-82. CRC 寄存器

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

3.5.2. 外设库函数说明

CRC库函数列表如下表所示：

表 3-83. CRC 库函数

库函数名称	库函数描述
crc_deinit	复位CRC计算单元
crc_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 3-84. 函数 `crc_deinit`

函数名称	crc_deinit	
函数原形	void crc_deinit(void);	
功能描述	复位CRC计算单元	
先决条件	-	
被调用函数	-	
输入参数{in}		
-	-	
输出参数{out}		

-	-
返回值	
-	-

例如：

```
/* reset crc */

crc_deinit();
```

函数 **crc_reverse_output_data_enable**

函数crc_reverse_output_data_enable描述见下表：

表 3-85. 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 3-86. 函数 **crc_reverse_output_data_enable**

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 3-87. 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 3-88. 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 3-89. 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		
uint8_t	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 3-90. 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}		
free_data	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 3-91. 函数 crc_init_data_register_write

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

例如：

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

crc_init_data_register_write (0x11223344);
```

函数 **crc_input_data_reverse_config**

函数crc_input_data_reverse_config描述见下表：

表 3-92. 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	-	
输入参数{in}		
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_HWORD	输入数据按半字翻转	

<u>_HALFWORD</u>	
<u>CRC_INPUT_DATA_WORD</u>	输入数据按字翻转
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* configure the crc input data */
crc_input_data_reverse_config (CRC_INPUT_DATA_WORD);
```

函数 **crc_polynomial_size_set**

函数crc_polynomial_size_set描述见下表：

表 3-93. 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}	
poly_size	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 3-94. 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}		
<code>poly</code>	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 3-95. 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}		
<code>sdata</code>	设定的32位数据	
输出参数{out}		
-	-	
返回值		
<code>uint32_t</code>	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 3-96. 函数 `crc_block_data_calculate`

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

例如：

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

#define BUFFER_SIZE      6

uint32_t valcrc = 0;

static const uint32_t data_buffer[BUFFER_SIZE] = {
    0x00001111, 0x00002222, 0x00003333, 0x00004444, 0x00005555, 0x00006666};

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

3.6. CTC

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

3.6.1. 外设寄存器描述

CTC寄存器列表如下表所示：

表 3-97. CTC 寄存器

寄存器名称	寄存器描述
<code>CTC_CTL0</code>	CTC控制寄存器0
<code>CTC_CTL1</code>	CTC控制寄存器1

寄存器名称	寄存器描述
CTC_STAT	CTC状态寄存器
CTC_INTC	CTC中断清除寄存器

3.6.2. 外设库函数说明

CTC库函数列表如下表所示：

表 3-98. CTC 库函数

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

函数 ctc_deinit

函数ctc_deinit描述见下表：

表 3-99. 函数 ctc_deinit

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

-	-
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* reset CTC */

ctc_deinit();
```

函数 **ctc_refsource_polarity_config**

函数**ctc_refsource_polarity_config**描述见下表：

表 3-100. 函数 **ctc_refsource_polarity_config**

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

例如：

```
/* set reference source polarity */

ctc_refsource_polarity_config (CTC_REFRESOURCE_POLARITY_RISING);
```

函数 **ctc_refsource_signal_select**

函数**ctc_refsource_signal_select**描述见下表：

表 3-101. 函数 **ctc_refsource_signal_select**

函数名称	ctc_refsource_signal_select
函数原形	void ctc_refsource_signal_select(uint32_t refs);
功能描述	CTC参考信号源选择
先决条件	-

被调用函数	-
输入参数{in}	
refs	参考信号源
<i>CTC_REFRESOURCE_G PIO</i>	选择GPIO输入信号
<i>CTC_REFRESOURCE_LX TAL</i>	选择LXTAL时钟
<i>CTC_REFRESOURCE_U SBSOF</i>	选择USB SOF信号
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* reference signal selection */

ctc_refresource_signal_select(CTC_REFRESOURCE_LXTAL);
```

函数 **ctc_refresource_prescaler_config**

函数**ctc_refresource_prescaler_config**描述见下表：

表 3-102. 函数 **ctc_refresource_prescaler_config**

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

SC_DIV64	
CTC_REFRESOURCE_P SC_DIV128	参考信号128分频
输出参数{out}	
-	-
返回值	
-	-

例如：

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

ctc_refresource_prescaler_config(CTC_REFRESOURCE_PSC_DIV2);
```

函数 **ctc_clock_limit_value_config**

函数 **ctc_clock_limit_value_config** 描述见下表：

表 3-103. 函数 `ctc_clock_limit_value_config`

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

例如：

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

ctc_clock_limit_value_config (0x1F);
```

函数 **ctc_counter_reload_value_config**

函数 **ctc_counter_reload_value_config** 描述见下表：

表 3-104. 函数 `ctc_counter_reload_value_config`

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

reload_value	0x0000 - 0xFFFF
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* configure CTC counter reload value */
ctc_counter_reload_value_config (0x00FF);
```

函数 **ctc_counter_enable**

函数**ctc_counter_enable**描述见下表：

表 3-105. 函数 ctc_counter_enable

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

例如：

```
/* enable CTC trim counter*/
ctc_counter_enable();
```

函数 **ctc_counter_disable**

函数**ctc_counter_disable**描述见下表：

表 3-106. 函数 ctc_counter_disable

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

-	-
返回值	
-	-

例如：

```
/* disable CTC trim counter */

ctc_counter_disable();
```

函数 **ctc_irc48m_trim_value_config**

函数 **ctc_irc48m_trim_value_config** 描述见下表：

表 3-107. 函数 **ctc_irc48m_trim_value_config**

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

例如：

```
/* IRC48M trim value configuration */

ctc_irc48m_trim_value_config (0x01);
```

函数 **ctc_software_refsource_pulse_generate**

函数 **ctc_software_refsource_pulse_generate** 描述见下表：

表 3-108. 函数 **ctc_software_refsource_pulse_generate**

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

-	-
---	---

例如：

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

ctc_software_refsource_pulse_generate();
```

函数 **ctc_hardware_trim_mode_config**

函数**ctc_hardware_trim_mode_config**描述见下表：

表 3-109. 函数 **ctc_hardware_trim_mode_config**

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

例如：

```
/* enable CTC hardware trim */

ctc_hardware_trim_mode_config(CTC_HARDWARE_TRIM_MODE_ENABLE);
```

函数 **ctc_counter_capture_value_read**

函数**ctc_counter_capture_value_read**描述见下表：

表 3-110. 函数 **ctc_counter_capture_value_read**

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

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

例如：

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

uint16_t ctc_value = 0;

ctc_value = ctc_counter_capture_value_read();
```

函数 **ctc_counter_direction_read**

函数 **ctc_counter_direction_read** 描述见下表：

表 3-111. 函数 **ctc_counter_direction_read**

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

例如：

```
/* read ctc counter direction */

FlagStatus ctc_direction = SET;

ctc_direction = ctc_counter_direction_read();
```

函数 **ctc_counter_reload_value_read**

函数 **ctc_counter_reload_value_read** 描述见下表：

表 3-112. 函数 **ctc_counter_reload_value_read**

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

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

例如：

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

uint16_t ctc_reload_value = 0;

ctc_reload_value = ctc_counter_reload_value_read();
```

函数 **ctc_irc48m_trim_value_read**

函数 **ctc_irc48m_trim_value_read** 描述见下表：

表 3-113. 函数 **ctc_irc48m_trim_value_read**

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

例如：

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

uint8_t ctc_trim_value = 0;

ctc_trim_value = ctc_irc48m_trim_value_read();
```

函数 **ctc_interrupt_enable**

函数 **ctc_interrupt_enable** 描述见下表：

表 3-114. 函数 **ctc_interrupt_enable**

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

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

例如：

```
/* enable CTC clock trim OK interrupt */

ctc_interrupt_enable (CTC_INT_CKOK);
```

函数 **ctc_interrupt_disable**

函数 **ctc_interrupt_disable** 描述见下表：

表 3-115. 函数 **ctc_interrupt_disable**

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

例如：

```
/* disable CTC clock trim OK interrupt */

ctc_interrupt_disable (CTC_INT_CKOK);
```

函数 **ctc_flag_get**

函数 **ctc_flag_get** 描述见下表：

表 3-116. 函数 ctc_flag_get

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

例如：

```
/* get CTC flag status */
FlagStatus state = ctc_flag_get (CTC_FLAG_CKOK);
```

函数 ctc_flag_clear

函数ctc_flag_clear描述见下表：

表 3-117. 函数 ctc_flag_clear

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

输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* clear CTC flag status */

ctc_flag_clear (CTC_FLAG_CKOK);
```

函数 **ctc_interrupt_flag_get**

函数 **ctc_interrupt_flag_get** 描述见下表：

表 3-118. 函数 **ctc_interrupt_flag_get**

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

例如：

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

FlagStatus state = ctc_interrupt_flag_get (CTC_INT_FLAG_CKOK);
```

函数 ctc_interrupt_flag_clear

函数ctc_interrupt_flag_clear描述见下表：

表 3-119. 函数 ctc_interrupt_flag_clear

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

例如：

```
/*clear CTC interrupt flag status */
ctc_interrupt_flag_clear (CTC_INT_FLAG_CKOK);
```

3.7. DAC

数字/模拟转换器可以将12位的数字数据转换为外部引脚上的电压输出，章节[3.7.1](#)描述了DAC的寄存器列表，章节[3.7.2](#)对DAC库函数进行说明。GD32F330xxxx没有DAC外设。

3.7.1. 外设寄存器说明

DAC寄存器列表如下表所示：

表 3-120. DAC 寄存器

寄存器名称	寄存器描述
DAC_CTL	DAC控制寄存器
DAC_SWT	DAC软件触发寄存器
DAC_R12DH	DAC 12位右对齐数据保持寄存器
DAC_L12DH	DAC 12位左对齐数据保持寄存器
DAC_R8DH	DAC 8位右对齐数据保持寄存器
DAC_DO	DAC数据输出寄存器
DAC_STAT	DAC状态寄存器

3.7.2. 外设库函数说明

DAC库函数列表如下表所示：

表 3-121. DAC 库函数

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

dac_deinit

函数dac_deinit描述见下表:

表 3-122. 函数 dac_deinit

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

例如:

```
/* deinitialize DAC */

dac_deinit();
```

dac_enable

函数dac_enable描述见下表:

表 3-123. 函数 dac_enable

函数名称	dac_enable
函数原型	void dac_enable(void);
功能描述	DAC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 DAC */

dac_enable();
```

dac_disable

函数dac_disable描述见下表:

表 3-124. 函数 dac_disable

函数名称	dac_disable
函数原型	void dac_disable(void);
功能描述	DAC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 DAC */

dac_disable();
```

dac_dma_enable

函数dac_dma_enable描述见下表:

表 3-125. 函数 dac_dma_enable

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

例如:

```
/* enable DAC DMA function */

dac_dma_enable();
```

dac_dma_disable

函数dac_dma_disable描述见下表:

表 3-126. 函数 dac_dma_disable

函数名称	dac_dma_disable
函数原型	void dac_dma_disable(void);
功能描述	DAC的DMA功能禁能
先决条件	-
被调用函数	-
输入参数{in}	
-	-
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* disable DAC DMA function */

dac_dma_disable();
```

dac_output_buffer_enable

函数dac_output_buffer_enable描述见下表:

表 3-127. 函数 dac_output_buffer_enable

函数名称	dac_output_buffer_enable
函数原型	void dac_output_buffer_enable(void);
功能描述	DAC输出缓冲区使能
先决条件	-
被调用函数	-
输入参数{in}	
-	-
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* enable DAC output buffer */

dac_output_buffer_enable();
```

dac_output_buffer_disable

函数dac_output_buffer_disable描述见下表:

表 3-128. 函数 dac_output_buffer_disable

函数名称	dac_output_buffer_disable
函数原型	void dac_output_buffer_disable(void);
功能描述	DAC输出缓冲区禁能
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 DAC output buffer */

dac_output_buffer_disable();
```

dac_trigger_enable

函数dac_trigger_enable描述见下表:

表 3-129. 函数 dac_trigger_enable

函数名称	dac_trigger_enable
函数原型	void dac_trigger_enable(void);
功能描述	DAC触发使能
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 DAC trigger */

dac_trigger_enable();
```

dac_trigger_disable

函数dac_trigger_disable描述见下表:

表 3-130. 函数 dac_trigger_disable

函数名称	dac_trigger_disable
函数原型	void dac_trigger_disable(void);
功能描述	DAC触发禁能
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 DAC trigger */

dac_trigger_disable();
```

dac_software_trigger_enable

函数dac_software_trigger_enable描述见下表:

表 3-131. 函数 dac_software_trigger_enable

函数名称	dac_software_trigger_enable
函数原型	void dac_software_trigger_enable(void);
功能描述	DAC软件触发使能
先决条件	-
被调用函数	-
输入参数{in}	
-	-
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* enable DAC software trigger */

dac_software_trigger_enable();
```

dac_software_trigger_disable

函数dac_software_trigger_disable描述见下表:

表 3-132. 函数 **dac_software_trigger_disable**

函数名称	dac_software_trigger_disable
函数原型	void dac_software_trigger_disable(void);
功能描述	DAC软件触发禁能
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 DAC software trigger */

dac_software_trigger_disable();
```

dac_trigger_source_config

函数dac_trigger_source_config描述见下表:

表 3-133. 函数 **dac_trigger_source_config**

函数名称	dac_trigger_source_config
函数原型	void dac_trigger_source_config(uint32_t triggersource);
功能描述	DAC触发源选择
先决条件	-
被调用函数	-
输入参数{in}	
triggersource	DAC触发源
DAC_TRIGGER_T1 _TRGO	TIMER1 TRGO
DAC_TRIGGER_T2 _TRGO	TIMER2 TRGO
DAC_TRIGGER_T5 _TRGO	TIMER5 TRGO
DAC_TRIGGER_T1 _4_TRGO	TIMER14 TRGO
DAC_TRIGGER _EXTI_9	EXTI线9中断
DAC_TRIGGER	软件触发

<u>_SOFTWARE</u>	
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* configure DAC trigger source */
dac_trigger_source_config(DAC_TRIGGER_T1_TRGO);
```

dac_wave_mode_config

函数dac_wave_mode_config描述见下表：

表 3-134. 函数 dac_wave_mode_config

函数名称	dac_wave_mode_config
函数原型	void dac_wave_mode_config(uint32_t wave_mode);
功能描述	DAC噪声波模式选择
先决条件	-
被调用函数	-
输入参数{in}	
wave_mode	噪声波模式选择
DAC_WAVE_DISABLE	噪声波模式禁能
DAC_WAVE_MODE_LFSR	LFSR噪声波模式
DAC_WAVE_MODE_TRIANGLE	三角波噪声波模式
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* configure DAC wave mode */
dac_wave_mode_config(DAC_WAVE_DISABLE);
```

dac_wave_bit_width_config

函数dac_wave_bit_width_config描述见下表：

表 3-135. 函数 dac_wave_bit_width_config

函数名称	dac_wave_bit_width_config
函数原型	void dac_wave_bit_width_config(uint32_t bit_width);

功能描述	DAC噪声波位宽设置
先决条件	-
被调用函数	-
输入参数{in}	
bit_width	噪声波位宽
DAC_WAVE_BIT_WIDTH_x	噪声波位宽 ($x = 1..12$)
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* configure DAC wave bit width */

dac_wave_bit_width_config(DAC_WAVE_BIT_WIDTH_1);
```

dac_lfsr_noise_config

函数dac_lfsr_noise_config描述见下表：

表 3-136. 函数 dac_lfsr_noise_config

函数名称	dac_lfsr_noise_config
函数原型	void dac_lfsr_noise_config(uint32_t unmask_bits);
功能描述	DAC LFSR模式设置
先决条件	-
被调用函数	-
输入参数{in}	
unmask_bits	噪声波的非屏蔽位宽
DAC_LFSR_BIT0	LFSR模式位0非屏蔽
DAC_LFSR_BITSx_0	LFSR模式位[x:0]非屏蔽
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* configure DAC LFSR noise mode */

dac_lfsr_noise_config(DAC_LFSR_BIT0);
```

dac_triangle_noise_config

函数dac_triangle_noise_config描述见下表：

表 3-137. 函数 `dac_triangle_noise_config`

函数名称	dac_triangle_noise_config
函数原型	void dac_triangle_noise_config(uint32_t amplitude);
功能描述	DAC三角波模式设置
先决条件	-
被调用函数	-
输入参数{in}	
amplitude	三角波幅值
<i>DAC_TRIANGLE_AMPLITUDE_x</i>	$x = 2^n - 1 (n = 1..12)$
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* configure DAC triangle noise mode */
dac_triangle_noise_config(DAC_TRIANGLE_AMPLITUDE_1);
```

`dac_output_value_get`

函数 `dac_output_value_get` 描述见下表:

表 3-138. 函数 `dac_output_value_get`

函数名称	dac_output_value_get
函数原型	uint16_t dac_output_value_get(void);
功能描述	DAC输出数据获取
先决条件	-
被调用函数	-
输入参数{in}	
-	-
输出参数{out}	
-	DAC输出数据
返回值	
uint16_t	外设DACx数据保持寄存器值 (0x000~0x7FF)

例如:

```
/* get the last data output value */
uint16_t data = 0;
data = dac_output_value_get();
```

dac_data_set

函数dac_data_set描述见下表:

表 3-139. 函数 dac_data_set

函数名称	dac_data_set
函数原型	void dac_data_set(uint32_t dac_align, uint16_t data);
功能描述	DAC输出数据设置
先决条件	-
被调用函数	-
输入参数{in}	
-	-
输入参数{in}	
dac_align	DAC对齐模式
<i>DAC_ALIGN_8B_R</i>	8位数据右对齐
<i>DAC_ALIGN_12B_R</i>	12位数据右对齐
<i>DAC_ALIGN_12B_L</i>	12位数据左对齐
输入参数{in}	
data	写入DACx的数据
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* set DAC data holding register value */
dac_data_set(DAC_ALIGN_8B_R, 0xff);
```

dac_flag_get

函数dac_flag_get描述见下表:

表 3-140. 函数 dac_flag_get

函数名称	dac_flag_get
函数原型	FlagStatus dac_flag_get(void)
功能描述	获取DAC标志
先决条件	-
被调用函数	-
输入参数{in}	
-	-
输出参数{out}	
-	-
返回值	

FlagStatus	DAC状态
SET	DMA下溢错误发生
RESET	DMA下溢错误未发生

例如：

```
/* get the specified DAC flag(DAC DMA underrun flag) */

FlagStatus dac_falg = RESET;

dac_falg = dac_flag_get();
```

dac_flag_clear

函数**dac_flag_clear**描述见下表：

表 3-141. 函数 dac_flag_clear

函数名称	dac_flag_clear
函数原型	void dac_flag_clear(void)
功能描述	清除DAC标志
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 the specified DAC flag(DAC DMA underrun flag)*/

dac_flag_clear();
```

dac_interrupt_enable

函数**dac_interrupt_enable**描述见下表：

表 3-142. 函数 dac_interrupt_enable

函数名称	dac_interrupt_enable
函数原型	void dac_interrupt_enable(void)
功能描述	使能DAC中断（DAC的DMA下溢中断使能）
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 DAC interrupt(DAC DMA underrun interrupt) */

dac_interrupt_enable();
```

dac_interrupt_disable

函数dac_interrupt_disable描述见下表：

表 3-143. 函数 dac_interrupt_disable

函数名称	dac_interrupt_disable
函数原型	void dac_interrupt_disable(void)
功能描述	禁能DAC中断（DAC的DMA下溢中断禁能）
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 DAC interrupt(DAC DMA underrun interrupt) */

dac_interrupt_disable();
```

dac_interrupt_flag_get

函数dac_interrupt_flag_get描述见下表：

表 3-144. 函数 dac_interrupt_flag_get

函数名称	dac_interrupt_flag_get
函数原型	FlagStatus dac_interrupt_flag_get(void)
功能描述	获取DAC中断标志
先决条件	-
被调用函数	-
输入参数{in}	
-	-
输出参数{out}	

-	-
返回值	
FlagStatus	DAC中断状态
SET	DMA下溢错误中断发生
RESET	DMA下溢错误中断未发生

例如:

```

/* get the specified DAC flag(DAC DMA underrun flag) */

FlagStatus dac_falg = RESET;

dac_falg = dac_interrupt_flag_get ();

dac_interrupt_flag_clear

```

函数dac_interrupt_flag_clear描述见下表:

表 3-145. 函数 dac_interrupt_flag_clear

函数名称	dac_interrupt_flag_clear
函数原型	void dac_interrupt_flag_clear(void)
功能描述	清除DAC中断标志
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 the specified DAC interrupt flag(DAC DMA underrun interrupt flag)*/

dac_interrupt_flag_clear ();

```

3.8. **DBG**

调试系统帮助调试者在低功耗模式下调试或者进行一些外设调试。章节[3.8.1](#)描述了DBG的寄存器列表，章节[3.8.2](#)对DBG库函数进行说明。

3.8.1. 外设寄存器说明

DBG寄存器列表如下表所示:

表 3-146. DBG 寄存器

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

3.8.2. 外设库函数说明

DBG库函数列表如下表所示：

表 3-147. 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 3-148. 枚举类型 `dbg_periph_enum`

成员名称	功能描述
DBG_FWDGT_HOLD	当内核停止时，保持FWDGT计数器时钟
DBG_WWDGT_HOLD	当内核停止时，保持WWDGT计数器时钟
DBG_TIMER0_HOLD	当内核停止时，保持TIMER0计数器计数值不变
DBG_TIMER1_HOLD	当内核停止时，保持TIMER1计数器计数值不变
DBG_TIMER2_HOLD	当内核停止时，保持TIMER2计数器计数值不变
DBG_TIMER5_HOLD	当内核停止时，保持TIMER5计数器计数值不变
DBG_TIMER13_HOLD	当内核停止时，保持TIMER13计数器计数值不变
DBG_TIMER14_HOLD	当内核停止时，保持TIMER14计数器计数值不变
DBG_TIMER15_HOLD	当内核停止时，保持TIMER15计数器计数值不变
DBG_TIMER16_HOLD	当内核停止时，保持TIMER16计数器计数值不变
DBG_I2C0_HOLD	当内核停止时，保持I2C0的SMBUS状态不变，用于调试
DBG_I2C1_HOLD	当内核停止时，保持I2C1的SMBUS状态不变，用于调试
DBG_RTC_HOLD	当内核停止时，保持RTC计数器，用于调试

函数 `dbg_deinit`

函数`dbg_deinit`描述见下表：

 表 3-149. 函数 `dbg_deinit`

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

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

例如：

```
/* reset DBG register */

dbg_deinit();
```

函数 **dbg_id_get**

函数**dbg_id_get**描述见下表：

表 3-150. 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 3-151. 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
<i>DBG_LOW_POWER_R_SLEEP</i>	在睡眠模式下，保持调试器连接，可进行调试
<i>DBG_LOW_POWER_R_DEEPSLEEP</i>	在深度睡眠模式下，保持调试器连接，可进行调试
<i>DBG_LOW_POWER_R_STANDBY</i>	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 3-152. 函数 **dbg_low_power_disable**

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

例如：

```

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

dbg_low_power_disable(DBG_LOW_POWER_SLEEP);

```

函数 `dbg_periph_enable`

函数`dbg_periph_enable`描述见下表:

表 3-153. 函数 `dbg_periph_enable`

函数名称	dbg_periph_enable
函数原形	void dbg_periph_enable(dbg_periph_enum dbg_periph);
功能描述	使能外设的MCU调试保持功能
先决条件	-
被调用函数	-
输入参数{in}	
<code>dbg_periph</code>	请参考 表3-148. 枚举类型<code>dbg_periph_enum</code>
<code>DBG_FWDGT_HOLD</code> <i>D</i>	当内核停止时，保持FWDGT计数器时钟
<code>DBG_WWDGT_HOLD</code> <i>LD</i>	当内核停止时，保持WWDGTH计数器时钟
<code>DBG_TIMERx_HOLD</code> <i>D</i>	当内核停止时，保持TIMERx计数器计数值不变（ <i>x</i> =0, 1, 2, 5, 13, 14, 15, 16）
<code>DBG_I2Cx_HOLD</code>	当内核停止时，保持I2Cx（ <i>x</i> =0, 1）的SMBUS状态不变，用于调试
<code>DBG_RTC_HOLD</code>	当内核停止时，保持RTC计数器，用于调试
输出参数{out}	
-	-
返回值	
-	-

例如:

```

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

dbg_periph_enable(DBG_TIMER0_HOLD);

```

函数 `dbg_periph_disable`

函数`dbg_periph_disable`描述见下表:

表 3-154. 函数 `dbg_periph_disable`

函数名称	dbg_periph_disable
函数原形	void dbg_periph_disable(dbg_periph_enum dbg_periph);
功能描述	禁能外设的MCU调试保持功能
先决条件	-
被调用函数	-
输入参数{in}	
<code>dbg_periph</code>	请参考 表3-148. 枚举类型<code>dbg_periph_enum</code>

<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WWDGTH计数器时钟
<i>DBG_TIMERx_HOLD</i> <i>D</i>	当内核停止时，保持TIMERx计数器计数值不变 ($x=0, 1, 2, 5, 13, 14, 15, 16$)
<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
输出参数{out}	
-	-
返回值	
-	-

例如：

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

dbg_periph_disable(DBG_TIMER0_HOLD);
```

3.9. DMA

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

3.9.1. 外设寄存器说明

DMA寄存器列表如下表所示：

表 3-155. DMA 寄存器

寄存器名称	寄存器描述
DMA_INTF	中断标志位寄存器
DMA_INTC	中断标志位清除寄存器
DMA_CHxCTL ($x=0..6$)	通道x控制寄存器
DMA_CHxCNT ($x=0..6$)	通道x计数寄存器
DMA_CHxPADDR ($x=0..6$)	通道x外设地址寄存器
DMA_CHxMADDR ($x=0..6$)	通道x存储器地址寄存器

3.9.2. 外设库函数说明

DMA库函数列表如下表所示：

表 3-156. DMA 库函数

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

枚举 dma_channel_enum

表 3-157. 枚举类型 dma_channel_enum

成员名称	功能描述
DMA_CH0	DMA通道0
DMA_CH1	DMA通道1
DMA_CH2	DMA通道2
DMA_CH3	DMA通道3
DMA_CH4	DMA通道4

DMA_CH5	DMA通道5
DMA_CH6	DMA通道6

结构体 **dma_parameter_struct**

表 3-158. 结构体类型 **dma_parameter_struct**

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

函数 **dma_deinit**

函数dma_deinit描述见下表：

表 3-159. 函数 **dma_deinit**

函数名称	dma_deinit
函数原型	void dma_deinit(dma_channel_enum channelx);
功能描述	复位DMA通道x的所有寄存器
先决条件	无
被调用函数	无
输入参数{in}	
channelx	DMA通道, 具体参考 表3-157. 枚举类型dma_channel_enum 。
DMA_CHx(x=0..6)	DMA通道选择
输出参数{out}	
-	-
返回值	
-	-

例如：

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

函数 **dma_struct_para_init**

函数dma_struct_para_init描述见下表：

表 3-160. 函数 **dma_struct_para_init**

函数名称	dma_struct_para_init
------	----------------------

函数原型	void dma_struct_para_init(dma_parameter_struct* init_struct);
功能描述	将DMA结构体中所有参数初始化为默认值
先决条件	无
被调用函数	无
输入参数{in}	
init_struct	初始化结构体, 结构体成员参考 表3-158. 结构体类型dma_parameter_struct
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 3-161. 函数 dma_init

函数名称	dma_init
函数原型	void dma_init(dma_channel_enum channelx, dma_parameter_struct* init_struct);
功能描述	初始化DMA通道x
先决条件	无
被调用函数	无
输入参数{in}	
channelx	DMA通道, 具体参考 表3-157. 枚举类型dma_channel_enum 。
DMA_CHx(x=0..6)	DMA通道选择
输出参数{out}	
-	-
返回值	
-	-

例如:

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

dma_init_struct.direction = DMA_PERIPHERAL_TO_MEMORY;
dma_init_struct.memory_addr = (uint32_t)g_destbuf;
dma_init_struct.memory_inc = DMA_MEMORY_INCREASE_ENABLE;
```

```

dma_init_struct.memory_width = DMA_MEMORY_WIDTH_8BIT;
dma_init_struct.number = TRANSFER_NUM;
dma_init_struct.periph_addr = (uint32_t)BANK0_WRITE_START_ADDR;
dma_init_struct.periph_inc = DMA_PERIPH_INCREASE_ENABLE;
dma_init_struct.periph_width = DMA_PERIPHERAL_WIDTH_8BIT;
dma_init_struct.priority = DMA_PRIORITY_ULTRA_HIGH;
dma_init(DMA_CH0, &dma_init_struct);

```

函数 **dma_circulation_enable**

函数**dma_circulation_enable**描述见下表:

表 3-162. 函数 **dma_circulation_enable**

函数名称	dma_circulation_enable	
函数原型	void dma_circulation_enable(dma_channel_enum channelx);	
功能描述	DMA循环模式使能	
先决条件	无	
被调用函数	无	
输入参数{in}		
channelx	DMA通道, 具体参考 表3-157. 枚举类型dma_channel_enum 。	
DMA_CHx(x=0..6)	DMA通道选择	
输出参数{out}		
-	-	
返回值		
-	-	

例如:

```

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

```

函数 **dma_circulation_disable**

函数**dma_circulation_disable**描述见下表:

表 3-163. 函数 **dma_circulation_disable**

函数名称	dma_circulation_disable	
函数原型	void dma_circulation_disable(dma_channel_enum channelx);	
功能描述	DMA循环模式禁能	
先决条件	无	
被调用函数	无	
输入参数{in}		
channelx	DMA通道, 具体参考 表3-157. 枚举类型dma_channel_enum 。	
DMA_CHx(x=0..6)	DMA通道选择	
输出参数{out}		
-	-	

返回值	
-	-

例如：

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

函数 **dma_memory_to_memory_enable**

函数dma_memory_to_memory_enable描述见下表：

表 3-164. 函数 **dma_memory_to_memory_enable**

函数名称	dma_memory_to_memory_enable
函数原型	void dma_memory_to_memory_enable(dma_channel_enum channelx);
功能描述	存储器到存储器DMA传输使能
先决条件	无
被调用函数	无
输入参数{in}	
channelx	DMA通道, 具体参考 表3-157. 枚举类型dma_channel_enum 。
DMA_CHx(x=0..6)	DMA通道选择
输出参数{out}	
-	-
返回值	
-	-

例如：

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

函数 **dma_memory_to_memory_disable**

函数dma_memory_to_memory_disable描述见下表：

表 3-165. 函数 **dma_memory_to_memory_disable**

函数名称	dma_memory_to_memory_disable
函数原形	void dma_memory_to_memory_disable(dma_channel_enum channelx);
功能描述	存储器到存储器DMA传输禁能
先决条件	无
被调用函数	无
输入参数{in}	
channelx	DMA通道, 具体参考 表3-157. 枚举类型dma_channel_enum 。
DMA_CHx(x=0..6)	DMA通道选择
输出参数{out}	
-	-
返回值	

-	-
---	---

例如：

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

函数 **dma_channel_enable**

函数**dma_channel_enable**描述见下表：

表 3-166. 函数 **dma_channel_enable**

函数名称	dma_channel_enable
函数原型	void dma_channel_enable(dma_channel_enum channelx);
功能描述	DMA通道x传输使能
先决条件	无
被调用函数	无
输入参数{in}	
channelx	DMA通道, 具体参考 表3-157. 枚举类型dma_channel_enum 。
DMA_CHx(x=0..6)	DMA通道选择
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* enable DMA channel0 */
dma_channel_enable(DMA_CH0)
```

函数 **dma_channel_disable**

函数**dma_channel_disable**描述见下表：

表 3-167. 函数 **dma_channel_disable**

函数名称	dma_channel_disable
函数原型	void dma_channel_disable(dma_channel_enum channelx);
功能描述	DMA通道x传输禁能
先决条件	无
被调用函数	无
输入参数{in}	
channelx	DMA通道, 具体参考 表3-157. 枚举类型dma_channel_enum 。
DMA_CHx(x=0..6)	DMA通道选择
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* disable DMA channel0 */
dma_channel_disable(DMA_CH0);
```

函数 `dma_periph_address_config`

函数`dma_periph_address_config`描述见下表：

表 3-168. 函数 `dma_periph_address_config`

函数名称	dma_periph_address_config
函数原型	void dma_periph_address_config(dma_channel_enum channelx, uint32_t address);
功能描述	DMA通道x传输的外设基地址配置
先决条件	无
被调用函数	无
输入参数{in}	
channelx	DMA通道，具体参考 表3-157. 枚举类型dma_channel_enum 。
DMA_CHx(x=0..6)	DMA通道选择
输入参数{in}	
address	外设基地址
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* configure DMA channel0 periph address */

#define BANK0_WRITE_START_ADDR ((uint32_t)0x08004000)

dma_periph_address_config(DMA_CH0, BANK0_WRITE_START_ADDR);
```

函数 `dma_memory_address_config`

函数`dma_memory_address_config`描述见下表：

表 3-169. 函数 `dma_memory_address_config`

函数名称	dma_memory_address_config
函数原型	void dma_memory_address_config(dma_channel_enum channelx, uint32_t address);
功能描述	DMA通道x传输的存储器基地址配置
先决条件	无
被调用函数	无
输入参数{in}	
channelx	DMA通道，具体参考 表3-157. 枚举类型dma_channel_enum 。

DMA_CHx(x=0..6)	DMA通道选择
输入参数{in}	
address	存储器基地址
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* configure DMA channel0 memory address */

uint8_t g_destbuf[TRANSFER_NUM];

dma_memory_address_config(DMA_CH0, (uint32_t)g_destbuf);
```

函数 `dma_transfer_number_config`

函数`dma_transfer_number_config`描述见下表：

表 3-170. 函数 `dma_transfer_number_config`

函数名称	dma_transfer_number_config
函数原型	void dma_transfer_number_config(dma_channel_enum channelx, uint32_t number);
功能描述	配置DMA通道x还有多少数据要传输
先决条件	无
被调用函数	无
输入参数{in}	
channelx	DMA通道，具体参考 表3-157. 枚举类型dma_channel_enum 。
DMA_CHx(x=0..6)	DMA通道选择
输入参数{in}	
number	数据传输数量（0x00000000 – 0x0000FFFF）
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* configure DMA channel0 transfer number */

#define TRANSFER_NUM          0x400

dma_transfer_number_config(DMA_CH0, TRANSFER_NUM);
```

函数 `dma_transfer_number_get`

函数`dma_transfer_number_get`描述见下表：

表 3-171. 函数 `dma_transfer_number_get`

函数名称	<code>dma_transfer_number_get</code>
函数原型	<code>uint32_t dma_transfer_number_get(dma_channel_enum channelx);</code>
功能描述	获取DMA通道x还有多少数据要传输
先决条件	无
被调用函数	无
输入参数{in}	
<code>channelx</code>	DMA通道, 具体参考 表3-157. 枚举类型dma_channel_enum 。
<code>DMA_CHx(x=0..6)</code>	DMA通道选择
输出参数{out}	
-	-
返回值	
<code>uint32_t</code>	DMA数据传输剩余数量 (0x00000000 – 0x0000FFFF)

例如：

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

uint32_t number = 0;

number = dma_transfer_number_get(DMA_CH0);
```

函数 `dma_priority_config`

函数`dma_priority_config`描述见下表：

 表 3-172. 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	无
被调用函数	无
输入参数{in}	
<code>channelx</code>	DMA通道, 具体参考 表3-157. 枚举类型dma_channel_enum 。
<code>DMA_CHx(x=0..6)</code>	DMA通道选择
输入参数{in}	
<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
输出参数{out}	

-	-	-
返回值		
-	-	-

例如：

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

函数 **dma_memory_width_config**

函数**dma_memory_width_config**描述见下表：

表 3-173. 函数 **dma_memory_width_config**

函数名称	dma_memory_width_config	
函数原型	void dma_memory_width_config(dma_channel_enum channelx, uint32_t mwidth);	
功能描述	DMA通道x传输的存储器数据宽度配置	
先决条件	无	
被调用函数	无	
输入参数{in}		
channelx	DMA通道, 具体参考 表3-157. 枚举类型dma_channel_enum 。	
DMA_CHx(x=0..6)	DMA通道选择	
输入参数{in}		
mwidth	存储器数据传输宽度	
DMA_MEMORY_WIDTH_8BIT	8位数据传输宽度	
DMA_MEMORY_WIDTH_16BIT	16位数据传输宽度	
DMA_MEMORY_WIDTH_32BIT	32位数据传输宽度	
输出参数{out}		
-	-	
返回值		
-	-	

例如：

```
/* configure DMA channel0 memory width */
dma_memory_width_config(DMA_CH0, DMA_MEMORY_WIDTH_8BIT);
```

函数 **dma_periph_width_config**

函数**dma_periph_width_config**描述见下表：

表 3-174. 函数 `dma_periph_width_config`

函数名称	<code>dma_periph_width_config</code>
函数原型	<code>void dma_periph_width_config(dma_channel_enum channelx, uint32_t pwidth);</code>
功能描述	DMA通道x传输的外设数据宽度配置
先决条件	无
被调用函数	无
输入参数{in}	
<code>channelx</code>	DMA通道, 具体参考 表3-157. 枚举类型dma_channel_enum 。
<code>DMA_CHx(x=0..6)</code>	DMA通道选择
输入参数{in}	
<code>pwidth</code>	外设数据传输宽度
<code>DMA_PERIPHERAL_WIDTH_8BIT</code>	8位数据传输宽度
<code>DMA_PERIPHERAL_WIDTH_16BIT</code>	16位数据传输宽度
<code>DMA_PERIPHERAL_WIDTH_32BIT</code>	32位数据传输宽度
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* configure DMA channel0 periph width */
dma_periph_width_config(DMA_CH0, DMA_PERIPHERAL_WIDTH_8BIT);
```

函数 `dma_memory_increase_enable`

函数`dma_memory_increase_enable`描述见下表：

表 3-175. 函数 `dma_memory_increase_enable`

函数名称	<code>dma_memory_increase_enable</code>
函数原型	<code>void dma_memory_increase_enable(dma_channel_enum channelx);</code>
功能描述	DMA通道x传输的存储器地址生成算法增量模式使能
先决条件	无
被调用函数	无
输入参数{in}	
<code>channelx</code>	DMA通道, 具体参考 表3-157. 枚举类型dma_channel_enum 。
<code>DMA_CHx(x=0..6)</code>	DMA通道选择
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* enable DMA channel0 memory increase */
```

```
dma_memory_increase_enable(DMA_CH0);
```

函数 **dma_memory_increase_disable**

函数**dma_memory_increase_disable**描述见下表：

表 3-176. 函数 **dma_memory_increase_disable**

函数名称	dma_memory_increase_disable
函数原型	void dma_memory_increase_disable(dma_channel_enum channelx);
功能描述	DMA通道x传输的存储器地址生成算法增量模式禁能
先决条件	无
被调用函数	无
输入参数{in}	
channelx	DMA通道，具体参考 表3-157. 枚举类型dma_channel_enum 。
DMA_CHx(x=0..6)	DMA通道选择
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* disable DMA channel0 memory increase */
```

```
dma_memory_increase_disable(DMA_CH0);
```

函数 **dma_periph_increase_enable**

函数**dma_periph_increase_enable**描述见下表：

表 3-177. 函数 **dma_periph_increase_enable**

函数名称	dma_periph_increase_enable
函数原型	void dma_periph_increase_enable(dma_channel_enum channelx);
功能描述	DMA通道x传输的外设地址生成算法增量模式使能
先决条件	无
被调用函数	无
输入参数{in}	
channelx	DMA通道，具体参考 表3-157. 枚举类型dma_channel_enum 。
DMA_CHx(x=0..6)	DMA通道选择
输出参数{out}	
-	-
返回值	
-	-

例如：

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

函数 **dma_periph_increase_disable**

函数 **dma_periph_increase_disable** 描述见下表：

表 3-178. 函数 **dma_periph_increase_disable**

函数名称	dma_periph_increase_disable
函数原型	void dma_periph_increase_disable(dma_channel_enum channelx);
功能描述	DMA通道x传输的外设地址生成算法增量模式禁能
先决条件	无
被调用函数	无
输入参数{in}	
channelx	DMA通道，具体参考 表3-157. 枚举类型dma_channel_enum 。
DMA_CHx(x=0..6)	DMA通道选择
输出参数{out}	
-	-
返回值	
-	-

例如：

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

函数 **dma_transfer_direction_config**

函数 **dma_transfer_direction_config** 描述见下表：

表 3-179. 函数 **dma_transfer_direction_config**

函数名称	dma_transfer_direction_config
函数原型	void dma_transfer_direction_config(dma_channel_enum channelx, uint32_t direction);
功能描述	DMA通道x的传输方向配置
先决条件	无
被调用函数	无
输入参数{in}	
channelx	DMA通道，具体参考 表3-157. 枚举类型dma_channel_enum 。
DMA_CHx(x=0..6)	DMA通道选择
输入参数{in}	
direction	数据传输方向
DMA_PERIPHERAL	读取外设中数据，写入存储器

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

例如：

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

函数 **dma_flag_get**

函数dma_flag_get描述见下表：

表 3-180. 函数 **dma_flag_get**

函数名称	dma_flag_get
函数原型	FlagStatus dma_flag_get(dma_channel_enum channelx, uint32_t flag);
功能描述	获取DMA通道x标志位状态
先决条件	无
被调用函数	无
输入参数{in}	
channelx	DMA通道，具体参考 表3-157. 枚举类型dma_channel_enum 。
DMA_CHx(x=0..6)	DMA通道选择
输入参数{in}	
flag	DMA标志
DMA_FLAG_G	DMA通道全局中断标志
DMA_FLAG_FTF	DMA通道传输完成标志
DMA_FLAG_HTF	DMA通道半传输完成标志
DMA_FLAG_ERR	DMA通道错误标志
输出参数{out}	
-	-
返回值	
FlagStatus	SET或RESET

例如：

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

函数 `dma_flag_clear`

函数`dma_flag_clear`描述见下表:

表 3-181. 函数 `dma_flag_clear`

函数名称	dma_flag_clear
函数原型	<code>void dma_flag_clear(dma_channel_enum channelx, uint32_t flag);</code>
功能描述	清除DMA通道x标志位状态
先决条件	无
被调用函数	无
输入参数{in}	
<code>channelx</code>	DMA通道, 具体参考 表3-157. 枚举类型dma_channel_enum 。
<code>DMA_CHx(x=0..6)</code>	DMA通道选择
输入参数{in}	
<code>flag</code>	DMA标志
<code>DMA_FLAG_G</code>	DMA通道全局中断标志
<code>DMA_FLAG_FTF</code>	DMA通道传输完成标志
<code>DMA_FLAG_HTF</code>	DMA通道半传输完成标志
<code>DMA_FLAG_ERR</code>	DMA通道错误标志
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* clear DMA channel0 flag*/
dma_flag_clear(DMA_CH0, DMA_FLAG_FTF);
```

函数 `dma_interrupt_enable`

函数`dma_interrupt_enable`描述见下表:

表 3-182. 函数 `dma_interrupt_enable`

函数名称	dma_interrupt_enable
函数原型	<code>void dma_interrupt_enable(dma_channel_enum channelx, uint32_t source);</code>
功能描述	DMA通道x中断使能
先决条件	无
被调用函数	无
输入参数{in}	
<code>channelx</code>	DMA通道, 具体参考 表3-157. 枚举类型dma_channel_enum 。
<code>DMA_CHx(x=0..6)</code>	DMA通道选择
输入参数{in}	
<code>source</code>	DMA中断源
<code>DMA_INT_FTF</code>	DMA通道传输完成中断

DMA_INT_HTF	DMA通道半传输完成中断
DMA_INT_ERR	DMA通道错误中断
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* enable DMA channel0 interrupt */
dma_interrupt_enable(DMA_CH0, DMA_INT_FTF);
```

函数 **dma_interrupt_disable**

函数dma_interrupt_disable描述见下表：

表 3-183. 函数 **dma_interrupt_disable**

函数名称	dma_interrupt_disable
函数原型	void dma_interrupt_disable(dma_channel_enum channelx, uint32_t source);
功能描述	DMA通道x中断禁能
先决条件	无
被调用函数	无
输入参数{in}	
channelx	DMA通道, 具体参考 表3-157. 枚举类型dma_channel_enum 。
DMA_CHx(x=0..6)	DMA通道选择
输入参数{in}	
source	DMA中断源
DMA_INT_FTF	DMA通道传输完成中断
DMA_INT_HTF	DMA通道半传输完成中断
DMA_INT_ERR	DMA通道错误中断
输出参数{out}	
-	-
返回值	
-	-

例如：

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

函数 **dma_interrupt_flag_get**

函数dma_interrupt_flag_get描述见下表：

表 3-184. 函数 **dma_interrupt_flag_get**

函数名称	dma_interrupt_flag_get
------	------------------------

函数原型	FlagStatus dma_interrupt_flag_get(dma_channel_enum channelx, uint32_t flag);
功能描述	获取DMA通道x中断标志位状态
先决条件	无
被调用函数	无
输入参数{in}	
channelx	DMA通道, 具体参考 表3-157. 枚举类型dma_channel_enum 。
DMA_CHx(x=0..6)	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 3-185. 函数 `dma_interrupt_flag_clear`

函数名称	dma_interrupt_flag_clear
函数原型	void dma_interrupt_flag_clear(dma_channel_enum channelx, uint32_t flag);
功能描述	清除DMA通道x中断标志位状态
先决条件	无
被调用函数	无
输入参数{in}	
channelx	DMA通道, 具体参考 表3-157. 枚举类型dma_channel_enum 。
DMA_CHx(x=0..6)	DMA通道选择
输入参数{in}	
flag	DMA标志
DMA_INT_FLAG_G	DMA通道全局中断标志

DMA_INT_FLAG_FTF	DMA通道传输完成中断标志
DMA_INT_FLAG_HT_F	DMA通道半传输完成中断标志
DMA_INT_FLAG_ER_R	DMA通道错误中断标志
输出参数{out}	
-	-
返回值	
-	-

例如：

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

3.10. EXTI

EXTI是MCU中的中断/事件控制器，包括24个相互独立的边沿检测电路并且能够向处理器内核产生中断请求或唤醒事件。章节[3.10.1](#)描述了EXTI的寄存器列表，章节[3.10.2](#)对EXTI库函数进行说明。

3.10.1. 外设寄存器说明

EXTI寄存器列表如下表所示：

表 3-186. EXTI 寄存器

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

3.10.2. 外设库函数说明

EXTI库函数列表如下表所示：

表 3-187. EXTI 库函数

库函数名称	库函数描述
exti_deinit	复位EXTI，将EXTI寄存器的值赋为初始值
exti_init	初始化EXTI线x

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

枚举类型 exti_line_enum

表 3-188. 枚举类型 exti_line_enum

成员名称	功能描述
EXTI_0	EXTI中断线0
EXTI_1	EXTI中断线1
EXTI_2	EXTI中断线2
EXTI_3	EXTI中断线3
EXTI_4	EXTI中断线4
EXTI_5	EXTI中断线5
EXTI_6	EXTI中断线6
EXTI_7	EXTI中断线7
EXTI_8	EXTI中断线8
EXTI_9	EXTI中断线9
EXTI_10	EXTI中断线10
EXTI_11	EXTI中断线11
EXTI_12	EXTI中断线12
EXTI_13	EXTI中断线13
EXTI_14	EXTI中断线14
EXTI_15	EXTI中断线15
EXTI_16	EXTI中断线16
EXTI_17	EXTI中断线17
EXTI_18	EXTI中断线18
EXTI_19	EXTI中断线19
EXTI_20	EXTI中断线20
EXTI_21	EXTI中断线21
EXTI_22	EXTI中断线22
EXTI_23	EXTI中断线23
EXTI_24	EXTI中断线24
EXTI_25	EXTI中断线25
EXTI_26	EXTI中断线26

EXTI_27	EXTI中断线27
---------	-----------

枚举类型 exti_mode_enum

表 3-189. 枚举类型 exti_mode_enum

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

枚举类型 exti_trig_type_enum

表 3-190. 枚举类型 exti_trig_type_enum

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

函数 exti_deinit

函数exti_deinit描述见下表:

表 3-191. 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 3-192. 函数 exti_init

函数名称	exti_init
------	-----------

函数原形	void exti_init(exti_line_enum linex, exti_mode_enum mode, exti_trig_type_enum trig_type);
功能描述	初始化EXTI线x
先决条件	-
被调用函数	-
输入参数{in}	
linex	EXTI线x, 具体参考 表3-188. 枚举类型exti_line_enum
EXTI_x	x=0..19,21,22
输入参数{in}	
mode	EXTI模式, 具体参考 表3-189. 枚举类型exti_mode_enum
EXTI_INTERRUPT	中断模式
EXTI_EVENT	事件模式
输入参数{in}	
trig_type	触发类型, 具体参考 表3-190. 枚举类型exti_trig_type_enum
EXTI_TRIG_RISING	上升沿触发
EXTI_TRIG_FALLING	下降沿触发
G	
EXTI_TRIG_BOTH	上升沿和下降沿均触发
EXTI_TRIG_NONE	上升沿和下降沿均不触发
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* configure EXTI_0 */

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

函数 exti_interrupt_enable

函数exti_interrupt_enable描述见下表:

表 3-193. 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, 具体参考 表3-188. 枚举类型exti_line_enum
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_interrupt_disable

函数exti_interrupt_disable描述见下表：

表 3-194. 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, 具体参考 表3-188. 枚举类型exti_line_enum
EXTI_x	x=0..19,21,22
输出参数{out}	
-	-
返回值	
-	-

例如：

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

函数 exti_event_enable

函数exti_event_enable描述见下表：

表 3-195. 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, 具体参考 表3-188. 枚举类型exti_line_enum
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_event_disable

函数exti_event_disable描述见下表：

表 3-196. 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, 具体参考 表3-188. 枚举类型exti_line_enum
EXTI_x	x=0..19,21,22
输出参数{out}	
-	-
返回值	
-	-

例如：

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

函数 exti_software_interrupt_enable

函数exti_software_interrupt_enable描述见下表：

表 3-197. 函数 exti_software_interrupt_enable

函数名称	exti_software_interrupt_enable
函数原形	void exti_software_interrupt_enable(exti_line_enum linex);
功能描述	EXTI线x软件中断使能
先决条件	-
被调用函数	-
输入参数{in}	
linex	EXTI线x, 具体参考 表3-188. 枚举类型exti_line_enum
EXTI_x	x=0..19,21,22
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 3-198. 函数 exti_software_interrupt_disable

函数名称	exti_software_interrupt_disable
函数原形	void exti_software_interrupt_disable(exti_line_enum linex);
功能描述	EXTI线x软件中断禁能
先决条件	-
被调用函数	-
输入参数{in}	
linex	EXTI线x, 具体参考 表3-188. 枚举类型exti_line_enum
EXTI_x	x=0..19,21,22
输出参数{out}	
-	-
返回值	
-	-

例如：

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

函数 exti_flag_get

函数exti_flag_get描述见下表：

表 3-199. 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, 具体参考 表3-188. 枚举类型exti_line_enum
EXTI_x	x=0..19,21,22
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 3-200. 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, 具体参考 表3-188. 枚举类型exti_line_enum
EXTI_x	x=0..19,21,22
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 3-201. 函数 exti_interrupt_flag_get

函数名称	exti_interrupt_flag_get
函数原形	FlagStatus exti_interrupt_flag_get(exti_line_enum linex);
功能描述	获取EXTI线x中断标志位
先决条件	-
被调用函数	-
输入参数{in}	
linex	EXTI线x, 具体参考 表3-188. 枚举类型exti_line_enum
EXTI_x	x=0..19,21,22
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 3-202. 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, 具体参考 表3-188. 枚举类型exti_line_enum
EXTI_x	x=0..19,21,22
输出参数{out}	
-	-
返回值	
-	-

例如：

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

3.11. FMC

FMC是MCU中的Flash控制器，其中包括存储数据的主编程块和选项字节。章节[3.11.1](#)描述了FMC的寄存器列表，章节[3.11.2](#)对FMC库函数进行说明。

3.11.1. 外设寄存器说明

FMC寄存器列表如下：

表 3-203. 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

寄存器	描述
FMC_STAT	状态寄存器
FMC_CTL	控制寄存器
FMC_ADDR	地址寄存器
FMC_OBSTAT	选项字节状态寄存器
FMC_WP	写保护寄存器
FMC_PID	产品ID寄存器

3.11.2. 外设库函数说明

FMC固件库函数列举如下表：

表 3-204. FMC 固件库函数

函数名称	函数描述
fmc_unlock	解锁FMC主编程块操作
fmc_lock	锁定FMC主编程块操作
fmc_wscnt_set	设置FMC等待状态计数值
fmc_wait_state_enable	使能等待状态功能
fmc_wait_state_disable	失能等待状态功能
fmc_page_erase	FMC 页擦除
fmc_mass_erase	FMC 全片擦除
fmc_word_program	在相应地址全字编程
fmc_halfword_program	在相应地址半字编程
fmc_word_reprogram	在不擦除编程地址的前提下进行字编程
ob_unlock	解锁选项字节操作
ob_lock	锁定选项字节操作
ob_reset	重装载选项字节，并产生一次系统复位
ob_erase	擦除选项字节
ob_write_protection_enable	使能写保护
ob_security_protection_config	配置安全保护
ob_user_write	写用户选项字节
ob_data_program	写数据选项字节
ob_user_get	获取用户选项字节
ob_data_get	获取数据选项字节
ob_write_protection_get	获取写保护选项字节
ob_obstat_plevel_get	在FMC_OBSTAT寄存器中获取FMC可选字节块的安全保护级别值
fmc_interrupt_enable	使能FMC中断
fmc_interrupt_disable	禁能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

函数名称	函数描述
fmc_state_get	获取FMC状态
fmc_ready_wait	检查FMC是否准备好
ob_parm_get	获取当前选项字节数值
ob_value_modify	在当前选项字节值的基础上变更选项字节

枚举 fmc_state_enum

表 3-205. 枚举类型 fmc_state_enum

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

结构体 ob_parm_struct

表 3-206. 结构体类型 ob_parm_struct

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

函数 fmc_unlock

函数fmc_unlock描述见下表:

表 3-207. 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 3-208. 函数 fmc_lock

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

例如：

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

```
fmc_lock( );
```

函数 fmc_wscnt_set

函数fmc_wscnt_set描述见下表：

表 3-209. 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_wait_state_enable**

函数fmc_wait_state_enable描述见下表：

表 3-210. 函数 **fmc_wait_state_enable**

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

例如：

```
/* fmc wait state enable */

fmc_wait_state_enable ( );
```

函数 **fmc_wait_state_disable**

函数fmc_wait_state_disable描述见下表：

表 3-211. 函数 **fmc_wait_state_disable**

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

例如：

```
/* fmc wait state disable */

fmc_wait_state_disable();
```

函数 **fmc_page_erase**

函数fmc_page_erase描述见下表：

表 3-212. 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 表3-205. 枚举类型fmc_state_enum

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 3-213. 函数 fmc_mass_erase

函数名称	fmc_mass_erase
函数原型	fmc_state_enum fmc_mass_erase(void);
功能描述	全片擦除
先决条件	fmc_unlock
被调用函数	fmc_ready_wait
输入参数{in}	
-	-
输出参数{out}	
-	-
返回值	
fmc_state_enum	FMC状态值，详情参考枚举变量 表3-205. 枚举类型fmc_state_enum

例如：

```

/* erase whole chip */

fmc_state_enum state = fmc_mass_erase();

```

函数 fmc_word_program

函数fmc_word_program描述见下表:

表 3-214. 函数 fmc_word_program

函数名称	fmc_word_program
函数原型	fmc_state_enum fmc_word_program(uint32_t address, uint32_t data);
功能描述	对相应地址全字编程
先决条件	fmc_unlock
被调用函数	fmc_ready_wait
输入参数{in}	
address	编程地址
输入参数{in}	
data	编程数据
输出参数{out}	
-	-
返回值	
fmc_state_enum	FMC状态值, 详情参考枚举变量 表3-205. 枚举类型fmc_state_enum

例如:

```

/* program a word at the corresponding address */

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

```

函数 fmc_halfword_program

函数fmc_halfword_program描述见下表:

表 3-215. 函数 fmc_halfword_program

函数名称	fmc_halfword_program
函数原型	fmc_state_enum fmc_halfword_program(uint32_t address, uint16_t data);
功能描述	对相应地址半字编程
先决条件	fmc_unlock
被调用函数	fmc_ready_wait
输入参数{in}	
address	编程地址
输入参数{in}	
data	编程数据
输出参数{out}	
-	-
返回值	
fmc_state_enum	FMC状态值, 详情参考枚举变量 表3-205. 枚举类型fmc_state_enum

例如：

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

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

函数 **fmc_word_reprogram**

函数fmc_word_reprogram描述见下表：

表 3-216. 函数 fmc_word_reprogram

函数名称	fmc_word_reprogram	
函数原型	fmc_state_enum fmc_word_reprogram(uint32_t address, uint32_t data);	
功能描述	对相应地址在不擦除原数据的前提下全字编程	
先决条件	fmc_unlock	
被调用函数	fmc_ready_wait	
输入参数{in}		
address	编程地址	
输入参数{in}		
data	编程数据	
输出参数{out}		
-	-	
返回值		
fmc_state_enum	FMC状态值，详情参考枚举变量 表3-205. 枚举类型fmc_state_enum	

例如：

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

fmc_state_enum state;

state = fmc_word_program (0x08004000, 0x01234567);

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

函数 **ob_unlock**

函数ob_unlock描述见下表：

表 3-217. 函数 ob_unlock

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

-	-
返回值	
-	-

例如：

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

ob_unlock ( );
```

函数 **ob_lock**

函数**ob_lock**描述见下表：

表 3-218. 函数 **ob_lock**

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

例如：

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

ob_lock ( );
```

函数 **ob_reset**

函数**ob_reset**描述见下表：

表 3-219. 函数 **ob_reset**

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

-	-
---	---

例如：

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

ob_reset();
```

函数 **ob_erase**

函数**ob_erase**描述见下表：

表 3-220. 函数 **ob_erase**

函数名称	ob_erase
函数原型	void ob_erase(void);
功能描述	擦除选项字节
先决条件	ob_unlock
被调用函数	fmc_ready_wait
输入参数{in}	
-	-
输出参数{out}	
-	-
返回值	
fmc_state_enum	FMC状态值，详情参考枚举变量 表3-205. 枚举类型fmc_state_enum

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 3-221. 函数 **ob_write_protection_enable**

函数名称	ob_write_protection_enable
函数原型	fmc_state_enum ob_write_protection_enable(uint16_t ob_wp);
功能描述	使能写保护
先决条件	ob_unlock
被调用函数	fmc_ready_wait
输入参数{in}	
ob_wp	写保护单元
输出参数{out}	
-	-
返回值	
fmc_state_enum	FMC状态值，详情参考枚举变量 表3-205. 枚举类型fmc_state_enum

例如：

```
/* enable write protection */

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

函数 **ob_security_protection_config**

函数**ob_security_protection_config**描述见下表：

表 3-222. 函数 **ob_security_protection_config**

函数名称	ob_security_protection_config
函数原型	fmc_state_enum ob_security_protection_config (uint16_t ob_spc);
功能描述	配置安全保护
先决条件	ob_unlock
被调用函数	fmc_ready_wait
输入参数{in}	
ob_spc	安全保护
FMC_NSPC	无安全保护
FMC_LSPC	低保护级别
FMC_HSPC	高保护级别
输出参数{out}	
-	-
返回值	
fmc_state_enum	FMC状态值，详情参考枚举变量 表3-205. 枚举类型fmc_state_enum

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 3-223. 函数 **ob_user_write**

函数名称	ob_user_write
函数原型	fmc_state_enum ob_user_write(uint8_t ob_user);
功能描述	编辑用户选项字节
先决条件	ob_unlock
被调用函数	fmc_ready_wait
输入参数{in}	
ob_user	用户定义的选项字节
OB_FWDGT_HW	硬件看门狗
OB_DEEPSLEEP_RST	进入深度睡眠时不复位

<i>OB_STDBY_RST</i>	进入深度睡眠时产生复位
<i>OB_BOOT1_SET_1</i>	BOOT1位是1
<i>OB_VDDA_DISABLE</i>	失能 <i>V_{DDA}</i> 监视器
<i>OB_SRAM_PARITY_ENABLE</i>	使能SRAM奇偶校验
输出参数{out}	
-	-
返回值	
<i>fmc_state_enum</i>	FMC状态值, 详情参考枚举变量 表3-205. 枚举类型fmc_state_enum

例如:

```
/* configure user option byte */

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

函数 **ob_data_program**

函数**ob_data_program**描述见下表:

表 3-224. 函数 **ob_data_program**

函数名称	ob_data_program
函数原型	<i>fmc_state_enum ob_data_program(uint32_t address, uint8_t data);</i>
功能描述	编程数字选项字节
先决条件	<i>ob_unlock</i>
被调用函数	<i>fmc_ready_wait</i>
输入参数{in}	
address	编程数字选项字节地址
<i>OB_DATA_ADDR0</i>	编程数字选项字节地址0
<i>OB_DATA_ADDR1</i>	编程数字选项字节地址1
输入参数{in}	
address	OB_DATA_ADDR0或OB_DATA_ADDR1
输入参数{in}	
data	所编程数值
输出参数{out}	
-	-
返回值	
<i>fmc_state_enum</i>	FMC状态值, 详情参考枚举变量 表3-205. 枚举类型fmc_state_enum

例如:

```
/* program option bytes data */

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

函数 **ob_user_get**

函数ob_user_get描述见下表:

表 3-225. 函数 ob_user_get

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

例如:

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

uint8_t user = ob_user_get();
```

函数 **ob_data_get**

函数ob_data_get描述见下表:

表 3-226. 函数 ob_data_get

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

例如:

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

Uint16_t data = ob_data_get();
```

函数 **ob_write_protection_get**

函数ob_write_protection_get描述见下表:

表 3-227. 函数 ob_write_protection_get

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

例如:

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

uint32_t wp = ob_write_protection_get();
```

函数 **ob_obstat_plevel_get**

函数ob_security_protection_flag_get描述见下表:

表 3-228. 函数 ob_obstat_plevel_get

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

例如:

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

uint32_t obstat_plevel = ob_obstat_plevel_get();
```

函数 fmc_interrupt_enable

函数fmc_interrupt_enable描述见下表:

表 3-229. 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 3-230. 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 3-231. 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_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 3-232. 函数 fmc_flag_clear

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

<i>R</i>	
<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 3-233. 函数 **fmc_interrupt_flag_get**

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

例如：

```
/* get FMC interrupt flag */

FlagStatus flag = fmc_interrupt_flag_get (FMC_FLAG_PGERR);
```

函数 **fmc_interrupt_flag_clear**

函数fmc_interrupt_flag_clear描述见下表：

表 3-234. 函数 **fmc_interrupt_flag_clear**

函数名称	fmc_interrupt_flag_clear
函数原型	void fmc_interrupt_flag_clear (uint32_t flag);

功能描述	通过写1清除FMC中断标志
先决条件	-
被调用函数	-
输入参数{in}	
flag	清除FMC中断标志
<i>FMC_FLAG_PGERR</i> <i>R</i>	FMC操作错误标志
<i>FMC_FLAG_WPERR</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 interrupt flag */

FlagStatus flag = fmc_interrupt_flag_clear (FMC_FLAG_PGERR);
```

函数 **fmc_state_get**

函数fmc_state_get描述见下表：

表 3-235. 函数 **fmc_state_get**

函数名称	fmc_state_get
函数原型	fmc_state_enum fmc_state_get(void);
功能描述	获取FMC状态
先决条件	-
被调用函数	-
输入参数{in}	
-	-
输出参数{out}	
-	-
返回值	
fmc_state_enum	FMC状态值，详情参考枚举变量 表3-205. 枚举类型fmc_state_enum

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 3-236. 函数 fmc_ready_wait

函数名称	fmc_ready_wait
函数原型	fmc_state_enum fmc_ready_wait(uint32_t timeout);
功能描述	检查FMC是否准备好
先决条件	-
被调用函数	fmc_state_get();
输入参数{in}	
timeout	循环计数次数
输出参数{out}	
-	-
返回值	
fmc_state_enum	FMC状态值, 详情参考枚举变量 表3-205. 枚举类型fmc_state_enum

例如:

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

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

函数 ob_parm_get

函数ob_parm_get描述见下表:

表 3-237. 函数 ob_parm_get

函数名称	ob_parm_get
函数原型	ob_parm_get(ob_parm_struct *ob_parm);
功能描述	获取当前选项字节值
先决条件	-
被调用函数	-
输入参数{in}	
ob_parm	指向选项字节参数结构体 表3-206. 结构体类型ob_parm_struct
输出参数{out}	
ob_parm	指向选项字节参数结构体 表3-206. 结构体类型ob_parm_struct
返回值	
-	-

例如:

```
/* get current option byte value */

ob_parm_struct ob_parm;

ob_parm_get(&ob_parm);
```

函数 ob_value_modify

函数ob_value_modify描述见下表:

表 3-238. 函数 ob_value_modify

函数名称	ob_value_modify	
函数原型	Void ob_value_modify(uint32_t address, uint16_t value,ob_parm_struct *ob_parm)	
功能描述	基于当前选项字节数值按照指定地址和数值修改选项字节值	
先决条件	-	
被调用函数	-	
输入参数{in}		
address	目标地址	
输入参数{in}		
value	目标数值	
输入参数{in}		
ob_parm	指向选项字节参数结构体 表3-206. 结构体类型ob_parm_struct	
输出参数{out}		
-	-	
返回值		
-	-	

例如：

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

ob_parm_struct ob_parm;

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

3.12. FWDGT

独立看门狗定时器（FWDGT）是一个硬件计时电路，用来监测由软件故障导致的系统故障。适合于需要独立环境且对计时精度要求不高的场合。章节[3.12.1](#)描述了FWDGT的寄存器列表，章节[3.12.2](#)对FWDGT库函数进行说明。

3.12.1. 外设寄存器说明

FWDGT寄存器列表如下表所示：

表 3-239. FWDGT 寄存器

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

3.12.2. 外设库函数说明

FWDGT库函数列表如下表所示：

表 3-240. FWDGT 库函数

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

函数 **fwdgt_write_enable**

函数fwdgt_write_enable描述见下表：

表 3-241. 函数 fwdgt_write_enable

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

例如：

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

fwdgt_write_enable();
```

函数 **fwdgt_write_disable**

函数fwdgt_write_disable描述见下表：

表 3-242. 函数 fwdgt_write_disable

函数名称	fwdgt_write_disable
函数原型	void fwdgt_write_disable(void);
功能描述	禁能对寄存器FWDGT_PSC, FWDGT_RLD和FWDGT_WND的写操作

先决条件	-
输入参数{in}	
-	-
输出参数{out}	
-	-
返回值	
-	-

例如：

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

fwdgt_write_disable();
```

函数 **fwdgt_enable**

函数fwdgt_enable描述见下表：

表 3-243. 函数 **fwdgt_enable**

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

例如：

```
/* start the FWDGT counter */

fwdgt_enable();
```

函数 **fwdgt_window_value_config**

函数fwdgt_window_value_config描述见下表：

表 3-244. 函数 **fwdgt_window_value_config**

函数名称	fwdgt_window_value_config
函数原型	ErrStatus fwdgt_window_value_config(uint16_t window_value);
功能描述	配置独立看门狗定时器计数器窗口值
先决条件	-
输入参数{in}	
window_value	窗口值,数值范围为 0x0000 – 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 3-245. 函数 **fwdgt_counter_reload**

函数名称	fwdgt_counter_reload
函数原型	void fwdgt_counter_reload(void);
功能描述	按照FWDGT_RLD寄存器的值重装载IWDG计数器
先决条件	-
输入参数{in}	
-	-
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* reload FWDGT counter */
fwdgt_counter_reload ( );
```

函数 **fwdgt_config**

函数fwdgt_config描述见下表：

表 3-246. 函数 **fwdgt_config**

函数名称	fwdgt_config
函数原型	ErrStatus fwdgt_config(uint16_t reload_value, uint8_t prescaler_div);
功能描述	设置FWDGT重装载值、预分频值
先决条件	-
输入参数{in}	
reload_value	重装载值(0x0000 - 0xFFFF)-
输入参数{in}	
prescaler_div	FWDGT预分频值
FWDGT_PSC_DIV4	FWDGT预分频值设为4

<i>FWDGT_PSC_DIV8</i>	FWDGT预分频值设为8
<i>FWDGT_PSC_DIV16</i>	FWDGT预分频值设为16
<i>FWDGT_PSC_DIV32</i>	FWDGT预分频值设为32
<i>FWDGT_PSC_DIV64</i>	FWDGT预分频值设为64
<i>FWDGT_PSC_DIV128</i>	FWDGT预分频值设为128
<i>FWDGT_PSC_DIV256</i>	FWDGT预分频值设为256
输出参数{out}	
-	-
返回值	
ErrStatus	ERROR or SUCCESS-

例如：

```
/* config 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 3-247. 函数 fwdgt_flag_get

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

例如：

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

```

FlagStatus status;

status = fwdgt_flag_get (FWDGT_FLAG_PUD);

```

3.13. GPIO

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

3.13.1. 外设寄存器说明

GPIO寄存器列表如下表所示：

表 3-248. 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
GPIO_AFSEL0	备用功能选择寄存器0
GPIO_AFSEL1	备用功能选择寄存器1
GPIO_BC	位清除寄存器
GPIO_TG	端口位翻转寄存器
GPIO_OSPEED1	端口输出速度寄存器1

3.13.2. 外设库函数说明

GPIO库函数列表如下表所示：

表 3-249. 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

库函数名称	库函数描述
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 3-250. 函数 gpio_deinit

函数名称	gpio_deinit
函数原型	void gpio_deinit(uint32_t gpio_periph);
功能描述	复位外设GPIOx
先决条件	-
被调用函数	rcu_periph_reset_enable / rcu_periph_reset_disable
输入参数{in}	
gpio_periph	GPIO端口
GPIOx	GPIOx(x = A,B,C,D,F)
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* reset GPIOA */
gpio_deinit(GPIOA);
```

函数 gpio_mode_set

函数gpio_mode_set描述见下表：

表 3-251. 函数 gpio_mode_set

函数名称	gpio_mode_set
函数原型	void gpio_mode_set(uint32_t gpio_periph, uint32_t mode, uint32_t pull_up_down, uint32_t pin);
功能描述	设置GPIO模式
先决条件	-
被调用函数	rcu_periph_reset_enable / rcu_periph_reset_disable
输入参数{in}	
gpio_periph	GPIO端口
GPIOx	GPIOx(x = A,B,C,D,F)

输入参数{in}	
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	GPIO_PIN_x(x=0..15)
GPIO_PIN_ALL	All pins
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* 配置PA0为上拉输入模式*/
gpio_mode_set(GPIOA, GPIO_MODE_INPUT, GPIO_PUPD_PULLUP, GPIO_PIN_0);
```

函数 **gpio_output_options_set**

函数gpio_output_options_set描述见下表：

表 3-252. 函数 **gpio_output_options_set**

函数名称	gpio_output_options_set
函数原型	void gpio_output_options_set(uint32_t gpio_periph, uint8_t otype, uint32_t speed, uint32_t pin);
功能描述	设置GPIO输出模式和速度
先决条件	-
被调用函数	-
输入参数{in}	
gpio_periph	GPIO端口
GPIOx	GPIOx(x = A,B,C,D,F)
输入参数{in}	
otype	GPIO引脚输出模式

<code>GPIO_OTYPE_PP</code>	推挽输出模式
<code>GPIO_OTYPE_OD</code>	开漏输出模式
输入参数{in}	
<code>speed</code>	GPIO引脚输出最大速度
<code>GPIO_OSPEED_2MHz</code>	最大输出速度为2MHz
<code>GPIO_OSPEED_10MHz</code>	最大输出速度为10MHz
<code>GPIO_OSPEED_50MHz</code>	最大输出速度为50MHz
<code>GPIO_OSPEED_MAX</code>	GPIO高输出速度，最大速度超过50MHz
输入参数{in}	
<code>pin</code>	GPIO引脚
<code>GPIO_PIN_x</code>	引脚选择 (x=0..15)
<code>GPIO_PIN_ALL</code>	所有引脚
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* config PA0 as push pull mode */
gpio_output_options_set(GPIOA, GPIO_OTYPE_PP, GPIO_OSPEED_2MHz, GPIO_PIN_0);
```

函数 `gpio_bit_set`

函数`gpio_bit_set`描述见下表：

表 3-253. 函数 `gpio_bit_set`

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

-	-
返回值	
-	-

例如：

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

函数 gpio_bit_reset

函数gpio_bit_reset描述见下表：

表 3-254. 函数 gpio_bit_reset

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

例如：

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

函数 gpio_bit_write

函数gpio_bit_write描述见下表：

表 3-255. 函数 gpio_bit_write

函数名称	gpio_bit_write
函数原型	void gpio_bit_write(uint32_t gpio_periph,uint32_t pin,bit_status bit_value);
功能描述	将特定的值写入引脚
先决条件	-
被调用函数	-

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

例如：

```
/* write 1 to PA0*/
gpio_bit_write (GPIOA, GPIO_PIN_0, SET);
```

函数 **gpio_port_write**

函数gpio_port_write描述见下表：

表 3-256. 函数 **gpio_port_write**

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

例如：

```
/* write 1010 0101 1010 0101 to Port A */
gpio_port_write (GPIOA, 0xA5A5);
```

函数 gpio_input_bit_get

函数gpio_input_bit_get描述见下表:

表 3-257. 函数 gpio_input_bit_get

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

例如:

```
/* get status of PA0*/
FlagStatus bit_state;
bit_state = gpio_input_bit_get (GPIOA, GPIO_PIN_0);
```

函数 gpio_input_port_get

函数gpio_input_port_get描述见下表:

表 3-258. 函数 gpio_input_port_get

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

uint16_t	0x0000-0xFFFF
----------	---------------

例如：

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

uint16_t port_state;

port_state = gpio_input_bit_get(GPIOA);
```

函数 **gpio_output_bit_get**

函数gpio_output_bit_get描述见下表：

表 3-259. 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,D,F)
输入参数{in}	
pin	GPIO引脚
GPIO_PIN_x	引脚选择 (x=0..15)
GPIO_PIN_ALL	所有引脚
输出参数{out}	
-	-
返回值	
FlagStatus	SET / RESET

例如：

```
/* get 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 3-260. 函数 **gpio_output_port_get**

函数名称	gpio_output_port_get
函数原型	uint16_t gpio_output_port_get(uint32_t gpio_periph);
功能描述	获取引脚的输出值
先决条件	-

被调用函数	-
输入参数{in}	
gpio_periph	GPIO端口
GPIOx	端口选择(x = A,B,C,D,F)
输出参数{out}	
-	-
返回值	
uint16_t	0x0000-0xFFFF

例如：

```
/* get output value of Port A */
uint16_t port_state;
port_state = gpio_output_port_get(GPIOA);
```

函数 gpio_af_set

函数gpio_af_set描述见下表：

表 3-261. 函数 gpio_af_set

函数名称	gpio_af_set
函数原型	void gpio_af_set(uint32_t gpio_periph, uint32_t alt_func_num, uint32_t pin);
功能描述	设置GPIO的备用功能
先决条件	-
被调用函数	-
输入参数{in}	
gpio_periph	GPIO 端口
GPIOx	GPIOx(x = A,B,C)
输入参数{in}	
alt_func_num	GPIO 引脚备用功能, 请参见特定设备的数据手册
GPIO_AF_0	TIMER2, TIMER13, TIMER14, TIMER16, SPI0, SPI1, I2S0, CK_OUT, USART0, CEC, IFRP, TSI, CTC, I2C0, I2C1, SWDIO, SWCLK
GPIO_AF_1	USART0, USART1, TIMER2, TIMER14, I2C0, I2C1, IFRP, CEC
GPIO_AF_2	TIMER0, TIMER1, TIMER15, TIMER16, I2S0
GPIO_AF_3	TSI, I2C0, TIMER14
GPIO_AF_4 (port A,B only)	USART1, I2C0, I2C1, TIMER13
GPIO_AF_5 (port A,B only)	TIMER15, TIMER16, USBFS, I2S0
GPIO_AF_6 (port A,B only)	CTC, SPI1
GPIO_AF_7 (port A,B only)	CMP0, CMP1
输入参数{in}	

pin	GPIO引脚
<i>GPIO_PIN_x</i>	引脚选择 (x=0..15)
<i>GPIO_PIN_ALL</i>	所有引脚
输出参数{out}	
-	-
返回值	
-	-

例如：

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

函数 **gpio_pin_lock**

函数gpio_pin_lock描述见下表：

表 3-262. 函数 gpio_pin_lock

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

例如：

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

函数 **gpio_bit_toggle**

函数gpio_bit_toggle描述见下表：

表 3-263. 函数 gpio_bit_toggle

函数名称	gpio_bit_toggle
------	-----------------

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

例如：

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

函数 **gpio_port_toggle**

函数gpio_port_toggle描述见下表：

表 3-264. 函数 **gpio_port_toggle**

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

例如：

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

3.14. I2C

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

3.14.1. 外设寄存器说明

I2C寄存器列表如下表所示：

表 3-265. I2C 寄存器

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

3.14.2. 外设库函数说明

I2C库函数列表如下表所示：

表 3-266. I2C 库函数

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

库函数名称	库函数描述
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_flag_get	获取I2C标志位
i2c_flag_clear	清除I2C标志位
i2c_interrupt_enable	中断使能
i2c_interrupt_disable	中断除能
i2c_interrupt_flag_get	中断标志位获取
i2c_interrupt_flag_clear	中断标志位清除

枚举 i2c_flag_enum

表 3-267. 枚举类型 i2c_flag_enum

成员名称	功能描述
I2C_FLAG_SBSEND	主机模式下发送START起始位
I2C_FLAG_ADDSEND	主机模式下成功发送了地址，从机模式下接收到了地址并且和自身的地址匹配
I2C_FLAG_BTC	字节发送结束
I2C_FLAG_ADD10SEND	主机模式下10位地址的地址头被发送
I2C_FLAG_STPDET	从机模式下监测到STOP结束位
I2C_FLAG_RBNE	接收期间I2C_DATA非空
I2C_FLAG_TBE	发送期间I2C_DATA为空
I2C_FLAG_BERR	总线错误，表示I2C总线上发生了预料之外的START起始位t或STOP结束位
I2C_FLAG_LOSTARB	主机模式下仲裁丢失
I2C_FLAG_AERR	应答错误
I2C_FLAG_OUERR	从机接收模式下，发生了过载或欠载事件
I2C_FLAG_PECERR	接收数据时PEC错误
I2C_FLAG_SMBTO	SMBus模式下超时信号
I2C_FLAG_SMBALT	SMBus警报状态
I2C_FLAG_MASTER	表明I2C时钟在主机模式还是从机模式的标志位
I2C_FLAG_I2CBSY	忙标志
I2C_FLAG_TR	I2C作发送端还是接收端
I2C_FLAG_RXGC	I2C作发送端还是接收端
I2C_FLAG_DEFSMB	从机模式下SMBus主机地址头

成员名称	功能描述
I2C_FLAG_HSTSMB	从机模式下监测到SMBus主机地址头
I2C_FLAG_DUMOD	从机模式下双标志位表明哪个地址和双地址模式匹配

枚举 i2c_interrupt_flag_enum

表 3-268. 枚举类型 i2c_interrupt_flag_enum

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

枚举 i2c_interrupt_enum

表 3-269. 枚举类型 i2c_interrupt_enum

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

函数 i2c_deinit

函数i2c_deinit描述见下表：

表 3-270. 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
<i>I2Cx</i>	(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 3-271. 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
<i>I2Cx</i>	(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
<i>I2C_DTCY_2</i>	T_low/T_high=2
<i>I2C_DTCY_16_9</i>	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 3-272. 函数 i2c_mode_addr_config

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

例如：

```
/* configure I2C0 address as 0x82, using 7 bits */
i2c_mode_addr_config(I2C0, I2C_I2CMODE_ENABLE, I2C_ADDFORMAT_7BITS, 0x82);
```

函数 i2c_smbus_type_config

函数i2c_smbus_type_config描述见下表：

表 3-273. 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 3-274. 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}	
ack	是否发送ACK
I2C_ACK_ENABLE	ACK会被发送
I2C_ACK_DISABLE	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 3-275. 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	配置在接收模式下ACK和PEC的位置
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}	
pos	ACK位置
I2C_ACKPOS_CUR RENT	目前正在接收的字节是否发送ACK
I2C_ACKPOS_NEX T	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 3-276. 函数 i2c_master_addressing

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

输入参数{in}	
trandirection	发送或接收
<i>I2C_TRANSMITTER</i> <i>R</i>	发送
<i>I2C_RECEIVER</i>	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 3-277. 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
<i>I2Cx</i>	(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 3-278. 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 3-279. 函数 i2c_enable

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

例如：

```
/* enable I2C0 */

i2c_enable (I2C0);
```

函数 i2c_disable

函数i2c_disable描述见下表：

表 3-280. 函数 i2c_disable

函数名称	i2c_disable
------	-------------

函数原型	void i2c_disable(uint32_t i2c_periph);
功能描述	禁能I2C模块
先决条件	-
被调用函数	-
输入参数{in}	
i2c_periph	I2C外设
I2Cx	(x=0,1)
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* disable I2C0 */

i2c_disable (I2C0);
```

函数 i2c_start_on_bus

函数i2c_start_on_bus描述见下表：

表 3-281. 函数 i2c_start_on_bus

函数名称	i2c_start_on_bus
函数原型	void i2c_start_on_bus(uint32_t i2c_periph);
功能描述	在I2C总线上生成起始位
先决条件	-
被调用函数	-
输入参数{in}	
i2c_periph	I2C外设
I2Cx	(x=0,1)
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* I2C0 send a start condition to I2C bus */

i2c_start_on_bus (I2C0);
```

函数 i2c_stop_on_bus

函数i2c_stop_on_bus描述见下表：

表 3-282. 函数 i2c_stop_on_bus

函数名称	i2c_stop_on_bus
------	-----------------

函数原型	void i2c_stop_on_bus(uint32_t i2c_periph);
功能描述	在I2C总线上生成停止位
先决条件	-
被调用函数	-
输入参数{in}	
i2c_periph	I2C外设
I2Cx	(x=0,1)
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* I2C0 generate a STOP condition to I2C bus */

i2c_stop_on_bus (I2C0);
```

函数 i2c_data_transmit

函数i2c_data_transmit描述见下表：

表 3-283. 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 3-284. 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 3-285. 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	-
输入参数{in}	
i2c_periph	I2C外设
I2Cx	(x=0,1)
输入参数{in}	
dmastate	开启或关闭
I2C_DMA_ON	DMA模式开启
I2C_DMA_OFF	DMA模式关闭
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* I2C0 DMA mode 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 3-286. 函数 i2c_dma_last_transfer_config

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

例如:

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

i2c_dma_last_transfer_config(I2C0, I2C_DMALST_ON);
```

函数 i2c_stretch_scl_low_config

函数i2c_stretch_scl_low_config描述见下表:

表 3-287. 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	拉低SCL	

<i>I2C_SCLSTRETCH_ENABLE</i>	
<i>I2C_SCLSTRETCH_DISABLE</i>	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 3-288. 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}	
<i>i2c_periph</i>	I2C外设
<i>I2Cx</i>	(x=0,1)
输入参数{in}	
<i>gcallpara</i>	是否响应广播呼叫
<i>I2C_GCEN_ENABLE</i>	从机响应广播呼叫
<i>I2C_GCEN_DISABLE</i>	从机不响应广播呼叫
输出参数{out}	
-	-
返回值	
-	-

例如：

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

i2c_slave_response_to_gcall_config (I2C0, I2C_GCEN_ENABLE);
```

函数 i2c_software_reset_config

函数i2c_software_reset_config描述见下表:

表 3-289. 函数 i2c_software_reset_config

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

例如:

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

函数 i2c_pec_enable

函数i2c_pec_enable描述见下表:

表 3-290. 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	是否使能报文错误校验
先决条件	-
被调用函数	-
输入参数{in}	
i2c_periph	I2C外设
I2Cx	(x=0,1)
输入参数{in}	
pecpara	开启或关闭
I2C_PEC_ENABLE	报文错误校验使能
I2C_PEC_DISABLE	报文错误校验关闭

输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* 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 3-291. 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
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 3-292. 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 3-293. 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}	
i2c_periph	I2C外设
I2Cx	(x=0,1)
输入参数{in}	
smbuspara	是否通过SMBA引脚发送警告
I2C_SALTSEND_E NABLE	通过SMBA引脚发送警告
I2C_SALTSEND_D/ SABLE	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 3-294. 函数 i2c_smbus_arp_enable

函数名称	i2c_smbus_arp_enable
函数原型	void i2c_smbus_arp_enable(uint32_t i2c_periph, uint32_t arpstate);
功能描述	SMBus下ARP协议是否开启
先决条件	-
被调用函数	-
输入参数{in}	
i2c_periph	I2C外设
I2Cx	(x=0,1)
输入参数{in}	
arpstate	SMBus下ARP协议是否开启
I2C_ARP_ENABLE	使能ARP
I2C_ARP_DISABLE	关闭ARP
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* enable I2C0 ARP protocol in SMBus switch */
i2c_smbus_arp_enable (I2C0, I2C_ARP_ENABLE);
```

函数 i2c_flag_get

函数i2c_flag_get描述见下表:

表 3-295. 函数 i2c_flag_get

函数名称	i2c_flag_get
函数原型	FlagStatus i2c_flag_get(uint32_t i2c_periph, i2c_flag_enum flag);
功能描述	标志位获取
先决条件	-
被调用函数	-
输入参数{in}	
i2c_periph	I2C外设
I2Cx	(x=0,1)
输入参数{in}	
flag	需要获取的标志位
I2C_FLAG_SBSEN	起始位是否发送

<i>D</i>	
<i>I2C_FLAG_ADDSEN</i>	主机模式下地址是否发送/从机模式下地址是否匹配
<i>I2C_FLAG_BTCS</i>	字节传输完成
<i>I2C_FLAG_ADD10SEND</i>	主机模式下10位地址地址头发送完成
<i>I2C_FLAG_STPDET</i>	从机模式下监测到STOP结束位
<i>I2C_FLAG_RBNE</i>	接收期间I2C_DATA非空
<i>I2C_FLAG_TBE</i>	发送期间I2C_DATA为空
<i>I2C_FLAG_BERR</i>	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
<i>I2C_FLAG_LOSTARB</i>	主机模式下仲裁丢失
<i>I2C_FLAG_AERR</i>	应答错误
<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_PECER</i>	接收数据时发生PEC错误
<i>I2C_FLAG_SMBTO</i>	SMBus模式下超时信号
<i>I2C_FLAG_SMBALT</i>	SMBus警报状态
<i>I2C_FLAG_MASTERR</i>	表明I2C时钟在主机模式还是从机模式的标志位
<i>I2C_FLAG_I2CBSY</i>	忙标志
<i>I2C_FLAG_TR</i>	I2C作发送端还是接收端
<i>I2C_FLAG_RXGC</i>	是否接收到广播地址(00h)
<i>I2C_FLAG_DEFSMB</i>	从机模式下SMBus主机地址头
<i>I2C_FLAG_HSTSMB</i>	从机模式下监测到SMBus主机地址头
<i>I2C_FLAG_DUMOD</i>	从机模式下双标志位表明哪个地址和双地址模式匹配
输出参数{out}	
-	-
返回值	
FlagStatus	SET / RESET

例如：

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

FlagStatus flag_state = RESET;

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

函数 i2c_flag_clear

函数i2c_flag_clear描述见下表:

表 3-296. 函数 i2c_flag_clear

函数名称	i2c_flag_clear
函数原型	void i2c_flag_clear(uint32_t i2c_periph, i2c_flag_enum flag)
功能描述	清除标志位
先决条件	-
被调用函数	-
输入参数{in}	
i2c_periph	I2C外设
I2Cx	(x=0,1)
输入参数{in}	
flag	标志位类型
I2C_FLAG_SMBAL_T	SMBus警报状态
I2C_FLAG_SMBTO	SMBus模式下超时信号
I2C_FLAG_PECER_R	接收数据时PEC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_AERR	应答错误
I2C_FLAG_LOSTA_RB	主机模式下仲裁丢失
I2C_FLAG_BERR	总线错误
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，通过读I2C_STAT0和I2C_STAT1来清除
输出参数{out}	
-	-
返回值	
-	-

例如:

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

函数 i2c_interrupt_enable

函数i2c_interrupt_enable描述见下表:

表 3-297. 函数 i2c_interrupt_enable

函数名称	i2c_interrupt_enable
函数原型	void i2c_interrupt_enable(uint32_t i2c_periph, i2c_interrupt_enum interrupt);
功能描述	中断使能

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

例如：

```
/* 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 3-298. 函数 i2c_interrupt_disable

函数名称	i2c_interrupt_disable
函数原型	void i2c_interrupt_disable(uint32_t i2c_periph, i2c_interrupt_enum interrupt);
功能描述	中断禁能
先决条件	-
被调用函数	-
输入参数{in}	
i2c_periph	I2C外设
I2Cx	(x=0,1)
输入参数{in}	
interrupt	中断类型
I2C_INT_ERR	错误中断使能
I2C_INT_EV	事件中断使能
I2C_INT_BUF	缓冲区中断使能
输出参数{out}	
-	-
返回值	
-	-

例如：

```

/* disable I2C0 event interrupt */

i2c_interrupt_disable (I2C0, I2C_INT_EV);

```

函数 i2c_interrupt_flag_get

函数i2c_interrupt_flag_get描述见下表：

表 3-299. 函数 i2c_interrupt_flag_get

函数名称	i2c_interrupt_flag_get
函数原型	FlagStatus i2c_interrupt_flag_get(uint32_t i2c_periph, i2c_interrupt_flag_enum int_flag)
功能描述	中断标志位获取
先决条件	-
被调用函数	-
输入参数{in}	
i2c_periph	I2C外设
I2Cx	(x=0,1)
输入参数{in}	
int_flag	中断标志
I2C_INT_FLAG_SB_SEND	主机模式下发送START起始位
I2C_INT_FLAG_AD_DSEND	主机模式下成功发送了地址 / 从机模式下接收到了地址并且和自身的地址匹配
I2C_INT_FLAG_BT_C	字节发送结束
I2C_INT_FLAG_AD_D10SEND	主机模式下10位地址地址头被发送
I2C_INT_FLAG_ST_PDET	从机模式下监测到STOP结束位
I2C_INT_FLAG_RB_NE	接收期间I2C_DATA非空
I2C_INT_FLAG_TB_E	发送期间I2C_DATA为空
I2C_INT_FLAG_BE_RR	总线错误
I2C_INT_FLAG_LO_STARB	主机模式下仲裁丢失
I2C_INT_FLAG_AE_RR	应答错误
I2C_INT_FLAG_OU_ERR	当禁用SCL拉低功能后，在从机模式下发生了过载或欠载事件
I2C_INT_FLAG_PE_CERR	接收数据时PEC错误

<i>I2C_INT_FLAG_SM_BTO</i>	SMBus模式下超时信号
<i>I2C_INT_FLAG_SM_BALT</i>	SMBus警报状态
输出参数{out}	
-	-
返回值	
FlagStatus	SET / RESET

例如：

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

FlagStatus flag_state = RESET;

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

函数 **i2c_interrupt_flag_clear**

函数*i2c_interrupt_flag_clear*描述见下表：

表 3-300. 函数 *i2c_interrupt_flag_clear*

函数名称	<i>i2c_interrupt_flag_clear</i>
函数原型	void i2c_interrupt_flag_clear(uint32_t i2c_periph, i2c_interrupt_flag_enum int_flag);
功能描述	中断标志位清除
先决条件	-
被调用函数	-
输入参数{in}	
i2c_periph	I2C外设
<i>I2Cx</i>	(x=0,1)
输入参数{in}	
int_flag	中断标志
<i>I2C_INT_FLAG_AD_DSEND</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_BE_RR</i>	总线错误
<i>I2C_INT_FLAG_LO_STARB</i>	主机模式下仲裁丢失
<i>I2C_INT_FLAG_AE_RR</i>	应答错误
<i>I2C_INT_FLAG_OU_ERR</i>	当禁用SCL 拉低功能后，在从机模式下发生了过载或欠载事件
<i>I2C_INT_FLAG_PE_CERR</i>	接收数据时PEC错误
<i>I2C_INT_FLAG_SM</i>	SMBus模式下超时信号

BTO	
<i>I2C_INT_FLAG_SM_BALT</i>	SMBus警报状态
输出参数{out}	
-	-
返回值	
-	-

例如：

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

i2c_interrupt_flag_clear (I2C0, I2C_INT_FLAG_AERR);
```

3.15. MISC

MISC 是对嵌套向量中断控制器（NVIC）和系统定时器（SysTick）操作的软件包。章节 [3.15.1](#) 描述了 NVIC 和 SysTick 的寄存器列表，章节 [3.15.2](#) 对 MISC 库函数进行说明。

3.15.1. 外设寄存器说明

表 3-301. NVIC 寄存器

寄存器名称	寄存器描述
ISER ⁽¹⁾	中断使能寄存器
ICER ⁽¹⁾	中断禁能寄存器
ISPR ⁽¹⁾	中断挂起寄存器
ICPR ⁽¹⁾	中断清除寄存器
IABR ⁽¹⁾	中断活动状态寄存器
IP ⁽¹⁾	中断优先级寄存器
STIR ⁽¹⁾	软触发中断寄存器
CPUID ⁽²⁾	CPUID寄存器
ICSR ⁽²⁾	中断控制及状态寄存器
VTOR ⁽²⁾	向量表偏移量寄存器
AIRCR ⁽²⁾	应用程序中断及复位控制寄存器
SCR ⁽²⁾	系统控制寄存器
CCR ⁽²⁾	配置与控制寄存器
SHP ⁽²⁾	系统异常优先级寄存器
SHCSR ⁽²⁾	系统异常控制及状态寄存器
CFSR ⁽²⁾	配置错误状态寄存器
HFSR ⁽²⁾	硬错误状态寄存器
DFSR ⁽²⁾	调试错误状态寄存器
MMFAR ⁽²⁾	存储管理错误地址寄存器

寄存器名称	寄存器描述
BFAR ⁽²⁾	总线错误地址寄存器
AFSR ⁽²⁾	辅助错误地址寄存器
PFR ⁽²⁾	处理器特性寄存器
DFR ⁽²⁾	调试特性寄存器
ADR ⁽²⁾	辅助特性寄存器
MMFR ⁽²⁾	存储模型特性寄存器
ISAR ⁽²⁾	指令设置属性寄存器
CPACR ⁽²⁾	协处理器访问控制寄存器

1. 参考 core_cm4.h 文件中定义的结构体类型 NVIC_Type
2. 参考 core_cm4.h 文件中定义的结构体类型 SCB_Type

表 3-302. Systick 寄存器

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

1. 参考 core_cm4.h 文件中定义的结构体类型 SysTick_Type

3.15.2. 外设库函数说明

MISC 库函数列表如下表所示：

表 3-303. MISC 库函数

库函数名称	库函数描述
nvic_irq_enable	使能NVIC的中断
nvic_irq_disable	禁能NVIC的中断
nvic_system_reset	复位
nvic_vector_table_set	设置向量表地址
system_lowpower_set	设置系统低功耗模式状态
system_lowpower_reset	复位系统低功耗模式状态
systick_clksource_set	设置系统定时器时钟源

枚举类型 IRQn_Type

表 3-304. 枚举类型 IRQn_Type

成员名称	功能描述
WWDGT_IRQn	窗口看门狗中断
LVD_IRQn	连接到 EXTI 线的 LVD 中断
RTC_IRQn	侵入检测中断
FMC_IRQn	FMC 全局中断
RCU_CTC_IRQn	RCU 与 CTC 全局中断
EXTI0_1_IRQn	EXTI 线 0 和线 1 中断

成员名称	功能描述
EXTI2_3_IRQHandler	EXTI 线 2 和线 3 中断
EXTI4_15_IRQHandler	EXTI 线 4 到线 15 中断
TSI_IRQHandler	TSI 中断
DMA_Channel0_IRQHandler	DMA0 通道 0 全局中断
DMA_Channel1_2_IRQHandler	DMA0 通道 1 与通道 2 全局中断
DMA_Channel3_4_IRQHandler	DMA0 通道 3 与通道 4 全局中断
ADC_CMP_IRQHandler	ADC 与 CMP 全局中断
TIMER0_BRK_UP_TRG_COM_IRQHandler	TIMER0 的中止，更新，触发和换相中断
TIMER0_Channel_IRQHandler	TIMER0 的捕获中断
TIMER1_IRQHandler	TIMER1 全局中断
TIMER2_IRQHandler	TIMER2 全局中断
TIMER5_DAC_IRQHandler	TIMER5 与 DAC 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
CEC_IRQHandler	CEC 全局中断
I2C0_ER_IRQHandler	I2C0 错误中断
I2C1_ER_IRQHandler	I2C1 错误中断
DMA_Channel5_6_IRQHandler	DMA0 通道 5 与通道 6 全局中断
USBFS_WKUP_IRQHandler	USBFS 唤醒中断
USBFS_IRQHandler	USBFS 全局中断

函数 **nvic_priority_group_set**

函数nvic_priority_group_set描述见下表：

表 3-305. 函数 nvic_priority_group_set

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

例如：

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

函数 nvic_irq_enable

函数nvic_irq_enable描述见下表：

表 3-306. 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 表3-304. 枚举类型IRQn_Type
输入参数{in}	
nvic_irq_priority	优先级 (0~3)
输出参数{out}	
-	-

返回值	
-	-

例如：

```
/* enable window watchDog timer interrupt , priority is 1 */
nvic_irq_enable(WWDGT_IRQn,1);
```

函数 nvic_irq_disable

函数nvic_irq_disable描述见下表：

表 3-307. 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 表3-304. 枚举类型IRQn_Type	
输出参数{out}		
-	-	
返回值		
-	-	

例如：

```
/* disable window watchDog timer interrupt */
nvic_irq_disable(WWDGT_IRQn);
```

函数 nvic_vector_table_set

函数nvic_vector_table_set描述见下表：

表 3-308. 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	-	
输入参数{in}		
nvic_vict_tab	RAM 或者 FLASH基址	
NVIC_VECTTAB_RAM	RAM 基址	
NVIC_VECTTAB_FLASH	FLASH基址	

输入参数{in}	
offset	向量表偏移量（向量表地址=基地址+偏移量）
输出参数{out}	
-	-
返回值	
-	-

例如：

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

nvic_vector_table_set(NVIC_VECTTAB_FLASH,0x200);
```

函数 **system_lowpower_set**

函数**system_lowpower_set**描述见下表：

表 3-309. 函数 system_lowpower_set

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

例如：

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

system_lowpower_set(SCB_LPM_SLEEP_EXIT_ISR);
```

函数 **system_lowpower_reset**

函数**system_lowpower_reset**描述见下表：

表 3-310. 函数 system_lowpower_reset

函数名称	system_lowpower_reset
------	-----------------------

函数原形	void system_lowpower_reset(uint8_t lowpower_mode);
功能描述	复位系统低功耗模式状态
先决条件	-
被调用函数	-
输入参数{in}	
lowpower_mode	系统低功耗模式的状态
SCB_LPM_SLEEP_EXIT_ISR	系统将通过退出ISR退出低功耗模式
SCB_LPM_DEEPSLEEP_EEP	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 3-311. 函数 systick_clksource_set

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

例如：

```

/* systick clock source is HCLK/8 */

systick_clksource_set(SYSTICK_CLKSOURCE_HCLK_DIV8);

```

3.16. PMU

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

3.16.1. 外设寄存器说明

PMU 寄存器列表如下表所示：

表 3-312. PMU 寄存器

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

3.16.2. 外设库函数说明

PMU 库函数列表如下表所示：

表 3-313. PMU 库函数

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

函数 pmu_deinit

函数pmu_deinit描述见下表：

表 3-314. 函数 pmu_deinit

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

例如：

```
/* reset PMU */
pmu_deinit();
```

函数 pmu_lvd_select

函数pmu_lvd_select描述见下表：

表 3-315. 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
PMU_LVDT_6	电压阈值为3.0V
PMU_LVDT_7	电压阈值为3.1V
输出参数{out}	
-	-
返回值	

-	-
---	---

例如：

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

```
pmu_lvd_select (PMU_LVDT_7);
```

函数 pmu_ldo_output_select

函数pmu_ldo_output_select描述见下表：

表 3-316. 函数 pmu_ldo_output_select

函数名称	pmu_ldo_output_select
函数原型	void pmu_ldo_output_select(uint32_t ldo_output);
功能描述	内部电压调节器（LDO）输出电压选择
先决条件	-
被调用函数	-
输入参数{in}	
ldo_output	输出电压模式
PMU_LDOVS_LOW	输出低电压模式
PMU_LDOVS_MID	输出中等电压模式
PMU_LDOVS_HIG H	输出高电压模式
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* select output low voltage mode */
```

```
pmu_ldo_output_select (PMU_LDOVS_LOW);
```

函数 pmu_lvd_disable

函数pmu_lvd_disable描述见下表：

表 3-317. 函数 pmu_lvd_disable

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

-	-	-
返回值		
-	-	-

例如：

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

函数 pmu_lowdriver_mode_enable

函数pmu_lowdriver_mode_enable描述见下表：

表 3-318. 函数 pmu_lowdriver_mode_enable

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

例如：

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

函数 pmu_lowdriver_mode_disable

函数pmu_lowdriver_mode_disable描述见下表：

表 3-319. 函数 pmu_lowdriver_mode_disable

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

-	-
---	---

例如：

```
/* disable low-driver mode */

pmu_lowdriver_mode_disable();
```

函数 **pmu_highdriver_mode_enable**

函数pmu_highdriver_mode_enable描述见下表：

表 3-320. 函数 pmu_highdriver_mode_enable

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

例如：

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

pmu_highdriver_mode_enable();
```

函数 **pmu_highdriver_mode_disable**

函数pmu_highdriver_mode_disable描述见下表：

表 3-321. 函数 pmu_highdriver_mode_disable

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

例如：

```
/* disable high-driver mode */
pmu_highdriver_mode_disable();
```

函数 pmu_highdriver_switch_select

函数pmu_highdriver_switch_select描述见下表：

表 3-322. 函数 pmu_highdriver_switch_select

函数名称	pmu_highdriver_switch_select
函数原型	void pmu_highdriver_switch_select(uint32_t highdr_switch);
功能描述	高驱动模式切换器选择
先决条件	-
被调用函数	-
输入参数{in}	
highdr_switch	高驱动模式切换器
PMU_HIGHDR_SWITCH_NONE	失能高驱动模式切换器
PMU_HIGHDR_SWITCH_EN	使能高驱动模式切换器
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* enable high-driver mode switch */
pmu_highdriver_switch_select(PMU_HIGHDR_SWITCH_EN);
```

函数 pmu_lowpower_driver_config

函数pmu_lowpower_driver_config描述见下表：

表 3-323. 函数 pmu_lowpower_driver_config

函数名称	pmu_lowpower_driver_config
函数原型	void pmu_lowpower_driver_config(uint32_t mode);
功能描述	使用低驱动LDO时低驱动模式配置
先决条件	-
被调用函数	-
输入参数{in}	
mode	驱动模式
PMU_NORMALDR_LOWPOWER	使用低驱动LDO时工作在正常驱动模式

<i>PMU_LOWDR_LO WPWR</i>	使用低驱动LDO且LDEN为1时低驱动模式使能
输出参数{out}	
-	-
返回值	
-	-

例如：

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

pmu_lowpower_driver_config(PMU_NORMALDR_LOWPWR);
```

函数 pmu_normalpower_driver_config

函数pmu_normalpower_driver_config描述见下表：

表 3-324. 函数 pmu_normalpower_driver_config

函数名称	pmu_normalpower_driver_config
函数原型	void pmu_normalpower_driver_config(uint32_t mode);
功能描述	使用正常驱动LDO时低驱动模式配置
先决条件	-
被调用函数	-
输入参数{in}	
mode	驱动模式
<i>PMU_NORMALDR_ NORMALPWR</i>	使用正常驱动LDO时工作在正常驱动模式
<i>PMU_NORMALDR_ NORMALPWR</i>	使用正常驱动LDO且LDEN为1时时低驱动驱动模式使能
输出参数{out}	
-	-
返回值	
-	-

例如：

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

pmu_normalpower_driver_config(PMU_NORMALDR_LOWPWR);
```

函数 pmu_to_sleepmode

函数pmu_to_sleepmode描述见下表：

表 3-325. 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
<i>WFI_CMD</i>	WFI命令
<i>WFE_CMD</i>	WFE命令
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* PMU work at sleep mode */

pmu_to_sleepmode (WFI_CMD);
```

函数 pmu_to_deepsleepmode

函数pmu_to_deepsleepmode描述见下表：

表 3-326. 函数 pmu_to_deepsleepmode

函数名称	pmu_to_deepsleepmode
函数原型	void pmu_to_deepsleepmode(uint32_t ldo, uint32_t lowdrive, uint8_t deepsleepmodecmd);
功能描述	进入深度睡眠模式
先决条件	-
被调用函数	-
输入参数{in}	
ldo	LDO工作模式
<i>PMU_LDO_NORMA_L</i>	当系统进入深度睡眠模式时，LDO仍正常工作
<i>PMU_LDO_LOWPO_WER</i>	当系统进入深度睡眠模式时，LDO进入低功耗模式
输入参数{in}	
lowdrive	使能或失能低驱动模式
<i>PMU_LOWDRIVER_ENABLE</i>	在深度睡眠模式时，使能低驱动模式
<i>PMU_LOWDRIVER_DISABLE</i>	在深度睡眠模式时，失能低驱动模式
输入参数{in}	
deepsleepmodecmd	进入深度睡眠模式命令
<i>WFI_CMD</i>	WFI命令
<i>WFE_CMD</i>	WFE命令

输出参数{out}	
-	-
返回值	
-	-

例如：

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

pmu_to_deepsleepmode (PMU_LDO_NORMAL, WFI_CMD);
```

函数 pmu_to_standbymode

函数pmu_to_standbymode描述见下表：

表 3-327. 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	-
输入参数{in}	
standbymodecmd	进入待机模式命令
WFI_CMD	WFI命令
WFE_CMD	WFE命令
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* PMU work at 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 3-328. 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

<i>PMU_WAKEUP_PI_N0</i>	WKUP Pin 0 (PA0)
<i>PMU_WAKEUP_PI_N1</i>	WKUP Pin 1 (PC13)
<i>PMU_WAKEUP_PI_N4</i>	WKUP Pin 4 (PC5)
<i>PMU_WAKEUP_PI_N5</i>	WKUP Pin 5 (PB5)
<i>PMU_WAKEUP_PI_N6</i>	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 3-329. 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}	
<i>wakeup_pin</i>	Wakeup pin
<i>PMU_WAKEUP_PI_N0</i>	WKUP Pin 0 (PA0)
<i>PMU_WAKEUP_PI_N1</i>	WKUP Pin 1 (PC13)
<i>PMU_WAKEUP_PI_N4</i>	WKUP Pin 4 (PC5)
<i>PMU_WAKEUP_PI_N5</i>	WKUP Pin 5 (PB5)
<i>PMU_WAKEUP_PI_N6</i>	WKUP Pin 6 (PB15)
输出参数{out}	
-	-

返回值	
-	-

例如：

```
/* disable wakeup pin6 */
pmu_wakeup_pin_disable (PMU_WAKEUP_PIN6);
```

函数 pmu_backup_write_enable

函数pmu_backup_write_enable描述见下表：

表 3-330. 函数 pmu_backup_write_enable

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

例如：

```
/* enable backup domain write */
pmu_backup_write_enable ();
```

函数 pmu_backup_write_disable

函数pmu_backup_write_disable描述见下表：

表 3-331. 函数 pmu_backup_write_disable

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

例如：

```
/* disable backup domain write */
pmu_backup_write_disable();
```

函数 pmu_flag_get

函数pmu_flag_get描述见下表：

表 3-332. 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
PMU_FLAG_LDOVS	LDO电压选择就绪标志
PMU_FLAG_HDR	高驱动就绪标志
PMU_FLAG_HDSR	高驱动切换器就绪标志
PMU_FLAG_LDR	低驱动模式就绪标志
输出参数{out}	
-	-
返回值	
FlagStatus	SET或RESET

例如：

```
/* get flag state */

FlagStatus status;

status = pmu_flag_get (PMU_FLAG_WAKEUP);
```

函数 pmu_flag_clear

函数pmu_flag_clear描述见下表：

表 3-333. 函数 pmu_flag_clear

函数名称	pmu_flag_clear
函数原型	void pmu_flag_clear(uint32_t flag);

功能描述	清除标志位
先决条件	-
被调用函数	-
输入参数{in}	
<code>flag_clear</code>	标志位
<code>PMU_FLAG_RESET_WAKEUP</code>	清除唤醒标志
<code>PMU_FLAG_RESET_STANDBY</code>	清除待机标志
输出参数{out}	
-	
返回值	
-	

例如：

```
/* clear flag bit */

pmu_flag_clear (PMU_FLAG_RESET_WAKEUP);
```

3.17. RCU

RCU 是复位和时钟单元，复位控制包括三种控制方式：电源复位、系统复位和备份域复位。时钟控制单元提供了一系列频率的时钟功能。章节 [3.17.1](#) 描述了 RCU 的寄存器列表，章节 [3.17.2](#) 对 RCU 库函数进行说明。

3.17.1. 外设寄存器说明

RCU 寄存器列表如下表所示：

表 3-334. RCU 寄存器

寄存器名称	寄存器描述
<code>RCU_CTL0</code>	控制寄存器0
<code>RCU_CFG0</code>	配置寄存器0
<code>RCU_INT</code>	中断寄存器
<code>RCU_APB2RST</code>	APB2复位寄存器
<code>RCU_APB1RST</code>	APB1复位寄存器
<code>RCU_AHBEN</code>	AHB使能寄存器
<code>RCU_APB2EN</code>	APB2使能寄存器
<code>RCU_APB1EN</code>	APB1使能寄存器
<code>RCU_BDCTL</code>	备份域控制寄存器
<code>RCU_RSTSCK</code>	复位源/时钟寄存器
<code>RCU_AHBRST</code>	AHB复位寄存器

寄存器名称	寄存器描述
RCU_CFG1	配置寄存器1
RCU_CFG2	配置寄存器2
RCU_CTL1	控制寄存器1
RCU_ADDCTL	附加时钟控制寄存器
RCU_ADDINT	附加时钟中断寄存器
RCU_ADDAPB1EN	APB1附加使能寄存器
RCU_ADDAPB1RST	APB1附加复位寄存器
RCU_VKEY	电源解锁寄存器
RCU_DSV	深度睡眠模式电压寄存器

3.17.2. 外设库函数说明

RCU库函数列表如下表所示：

表 3-335. RCU 库函数

库函数名称	库函数描述
rcu_deinit	复位RCU
rcu_periph_clock_enable	使能外设时钟
rcu_periph_clock_disable	禁能外设时钟
rcu_periph_clock_sleep_enable	在睡眠模式下，使能外设时钟
rcu_periph_clock_sleep_disable	在睡眠模式下，禁能外设时钟
rcu_periph_reset_enable	外设时钟复位使能
rcu_periph_reset_disable	外设时钟复位除能
rcu_bkp_reset_enable	备份域时钟复位使能
rcu_bkp_reset_disable	备份域时钟复位除能
rcu_system_clock_source_config	配置选择系统时钟源
rcu_system_clock_source_get	获取系统时钟源选择状态
rcu_ahb_clock_config	配置AHB时钟预分频选择
rcu_apb1_clock_config	配置APB1时钟预分频选择
rcu_apb2_clock_config	配置APB2时钟预分频选择
rcu_adc_clock_config	配置ADC时钟预分频选择
rcu_usbfs_clock_config	配置USBFS预分频系数
rcu_ckout_config	配置CKOUT时钟源选择及分频系数
rcu_pll_preselection_config	配置主PLL时钟源预选项
rcu_pll_config	配置PLL时钟源及倍频因子
rcu_usart_clock_config	配置串口时钟
rcu_cec_clock_config	配置CEC时钟
rcu_rtc_clock_config	配置RTC时钟
rcu_ck48m_clock_config	配置IRC48M时钟
rcu_hxtal_prediv_config	配置HXTAL作为PLL输入源分频因子
rcu_lxtal_drive_capability_config	配置LXTAL的驱动力

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
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

枚举 reg_idx

表 3-336. 枚举类型 reg_idx

成员名称	功能描述
IDX_AHBEN	AHB使能寄存器地址偏移
IDX_APB2EN	APB2使能寄存器地址偏移
IDX_APB1EN	APB1使能寄存器地址偏移
IDX_ADDAPB1EN	APB1附加使能寄存器地址偏移
IDX_AHBRST	AHB复位寄存器地址偏移
IDX_APB2RST	APB2复位寄存器地址偏移
IDX_APB1RST	APB1复位寄存器地址偏移
IDX_ADDAPB1RST	APB1附加复位寄存器地址偏移
IDX_CTL0	控制寄存器地址偏移
IDX_BDCTL	备份域控制寄存器地址偏移
IDX_CTL1	控制寄存器1地址偏移
IDX_ADDCTL	附加时钟控制寄存器地址偏移
IDX_RSTSCK	复位源/时钟寄存器地址偏移
IDX_INT	中断寄存器地址偏移
IDX_ADDINT	附加时钟中断寄存器地址偏移
IDX_CFG0	控制寄存器0地址偏移
IDX_CFG2	控制寄存器1地址偏移

枚举 rcu_periph_enum

表 3-337. 枚举类型 rcu_periph_enum

成员名称	功能描述
RCU_DMA	DMA时钟
RCU_CRC	CRC时钟
RCU_GPIOA	GPIOA时钟
RCU_GPIOB	GPIOB时钟
RCU_GPIOC	GPIOC时钟
RCU_GPIOD	GPIOD时钟
RCU_GPIOF	GPIOF时钟
RCU_TSI	TSI时钟
RCU_CFGCMP	CGFCMP时钟
RCU_ADC	ADC时钟
RCU_TIMER0	TIMER0时钟
RCU_SPI0	SPI0时钟
RCU_USART0	USART0时钟
RCU_TIMER14	TIMER14时钟
RCU_TIMER15	TIMER15时钟
RCU_TIMER16	TIMER16时钟
RCU_TIMER1	TIMER1时钟
RCU_TIMER2	TIMER2时钟
RCU_TIMER13	TIMER13时钟
RCU_WWDGT	WWDGT时钟
RCU_SPI1	SPI1时钟
RCU_USART1	USART1时钟
RCU_I2C0	I2C0时钟
RCU_I2C1	I2C1时钟
RCU_PMU	PMU时钟
RCU_DAC	DAC时钟（仅对GD32F350适用）
RCU_CEC	CEC时钟（仅对GD32F350适用）
RCU_TIMER5	TIMER5时钟（仅对GD32F350适用）
RCU_USBFS	USBFS时钟（仅对GD32F350适用）
RCU_RTC	RTC时钟
RCU_CTC	CTC时钟

枚举 rcu_periph_sleep_enum

表 3-338. 枚举类型 rcu_periph_sleep_enum

成员名称	功能描述
RCU_SRAM_SLP	SRAM接口时钟
RCU_FMC_SLP	FMC时钟

枚举 rcu_periph_reset_enum

表 3-339. 枚举类型 rcu_periph_reset_enum

成员名称	功能描述
RCU_GPIOARST	复位GPIOA时钟
RCU_GPIOBRST	复位GPIOB时钟
RCU_GPIOCRST	复位GPIOC时钟
RCU_GPIODRST	复位GPIOD时钟
RCU_GPIOFRST	复位GPIOF时钟
RCU_TSIRST	复位TSI时钟
RCU_CFGCMRST	复位CFGCM时钟
RCU_ADCRST	复位ADC时钟
RCU_TIMER0RST	复位TIMER0时钟
RCU_SPI0RST	复位SPI0时钟
RCU_USART0RST	复位USART0时钟
RCU_TIMER14RST	复位TIMER14时钟
RCU_TIMER15RST	复位TIMER15时钟
RCU_TIMER16RST	复位TIMER16时钟
RCU_TIMER1RST	复位TIMER1时钟
RCU_TIMER2RST	复位TIMER2时钟
RCU_TIMER13RST	复位TIMER13时钟
RCU_WWDGTRST	复位WWDG时钟
RCU_SPI1RST	复位SPI1时钟
RCU_USART1RST	复位USART1时钟
RCU_I2C0RST	复位I2C0时钟
RCU_I2C1RST	复位I2C1时钟
RCU_PMURST	复位PMU时钟
RCU_DACRST	复位DAC时钟（仅对GD32F350适用）
RCU_CECRST	复位CEC时钟（仅对GD32F350适用）
RCU_TIMER5RST	复位TIMER5时钟（仅对GD32F350适用）
RCU_USBFSRST	复位USBFS时钟（仅对GD32F350适用）
RCU_CTCRST	复位CTC时钟

枚举 rcu_flag_enum

表 3-340. 枚举类型 rcu_flag_enum

成员名称	功能描述
RCU_FLAG_IRC40K_STB	IRC40K振荡器稳定标志
RCU_FLAG_LXTAL_STB	外部低速晶振稳定标志
RCU_FLAG_IRC8M_STB	IRC8M振荡器稳定标志

成员名称	功能描述
RCU_FLAG_HXTAL_STB	外部高速晶振稳定标志
RCU_FLAG_PLLST_B	PLL稳定标志
RCU_FLAG_IRC28_MSTB	IRC28M振荡器稳定标志
RCU_FLAG_IRC48_MSTB	IRC48M振荡器稳定标志
RCU_FLAG_V12RS_T	1.2V电源域复位标志
RCU_FLAG_OBLR_ST	可选字节装载器复位标志
RCU_FLAG_EPRS_T	外部引脚复位标志
RCU_FLAG_PORR_ST	电源复位标志
RCU_FLAG_SWRS_T	软件复位标志
RCU_FLAG_FWDG_TRST	独立看门狗复位标志
RCU_FLAG_WWD_GTRST	窗口看门狗复位标志
RCU_FLAG_LPRST	低功耗复位标志

枚举 `rcu_int_flag_enum`

表 3-341. 枚举类型 `rcu_int_flag_enum`

成员名称	功能描述
RCU_INT_FLAG_IR_C40KSTB	IRC40K时钟稳定中断标志
RCU_INT_FLAG_L_XTALSTB	外部低速晶振时钟稳定中断标志
RCU_INT_FLAG_IR_C8MSTB	IRC8M时钟稳定中断标志
RCU_INT_FLAG_H_XTALSTB	外部高速晶振时钟稳定中断标志
RCU_INT_FLAG_P_LLSTB	PLL时钟稳定中断标志
RCU_INT_FLAG_IR_C28MSTB	IRC28M时钟稳定中断标志
RCU_INT_FLAG_C_KM	外部高速晶振时钟阻塞中断标志

成员名称	功能描述
RCU_INT_FLAG_IR_C48MSTB	IRC48M时钟稳定中断标志

枚举 `rcu_int_flag_clear_enum`

表 3-342. 枚举类型 `rcu_int_flag_clear_enum`

成员名称	功能描述
RCU_INT_FLAG_IR_C40KSTB_CLR	IRC40K时钟稳定中断清除标志
RCU_INT_FLAG_L_XTALSTB_CLR	外部低速晶振时钟稳定中断清除标志
RCU_INT_FLAG_IR_C8MSTB_CLR	IRC8M时钟稳定中断清除标志
RCU_INT_FLAG_H_XTALSTB_CLR	外部高速晶振时钟稳定中断清除标志
RCU_INT_FLAG_P_LLSTB_CLR	PLL时钟稳定中断清除标志
RCU_INT_FLAG_IR_C28MSTB_CLR	IRC28M时钟稳定中断清除标志
RCU_INT_FLAG_C_KM_CLR	外部高速晶振时钟阻塞中断清除标志
RCU_INT_FLAG_IR_C48MSTB_CLR	IRC48M时钟稳定中断清除标志

枚举 `rcu_int_enum`

表 3-343. 枚举类型 `rcu_int_enum`

成员名称	功能描述
RCU_INT_IRC40KS_TB	IRC40K时钟稳定中断
RCU_INT_LXTALS_TB	外部低速晶振时钟稳定中断
RCU_INT_IRC8MS_TB	IRC8M时钟稳定中断
RCU_INT_HXTALS_TB	外部高速晶振时钟稳定中断
RCU_INT_PLLSTB	PLL时钟稳定中断
RCU_INT_IRC28M_STB	IRC28M时钟稳定中断
RCU_INT_IRC48M_STB	IRC48M时钟稳定中断

枚举 rcu_adc_clock_enum

表 3-344. 枚举类型 rcu_adc_clock_enum

成员名称	功能描述
RCU_ADCCK_IRC2_8M_DIV2	选择IRC28M的2分频作为ADC的时钟源
RCU_ADCCK_IRC2_8M	选择IRC28M作为ADC的时钟源
RCU_ADCCK_APB_2_DIV2	选择APB的2分频作为ADC的时钟源
RCU_ADCCK_AHB_DIV3	选择AHB的3分频作为ADC的时钟源
RCU_ADCCK_APB_2_DIV4	选择APB的4分频作为ADC的时钟源
RCU_ADCCK_AHB_DIV5	选择AHB的5分频作为ADC的时钟源
RCU_ADCCK_APB_2_DIV6	选择APB的6分频作为ADC的时钟源
RCU_ADCCK_AHB_DIV7	选择AHB的7分频作为ADC的时钟源
RCU_ADCCK_APB_2_DIV8	选择APB的8分频作为ADC的时钟源
RCU_ADCCK_AHB_DIV9	选择AHB的9分频作为ADC的时钟源

枚举 rcu_osc_type_enum

表 3-345. 枚举类型 rcu_osc_type_enum

成员名称	功能描述
RCU_HXTAL	外部高速振荡器
RCU_LXTAL	外部低速振荡器
RCU_IRC8M	IRC8M振荡器
RCU_IRC28M	IRC28M振荡器
RCU_IRC48M	IRC48M振荡器
RCU_IRC40K	IRC40K振荡器
RCU_PLL_CK	锁相环时钟

枚举 rcu_clock_freq_enum

表 3-346. 枚举类型 rcu_clock_freq_enum

成员名称	功能描述
CK_SYS	系统时钟
CK_AHB	AHB时钟
CK_APB1	APB1时钟

成员名称	功能描述
CK_APB2	APB2时钟
CK_ADC	ADC时钟
CK_CEC	CEC时钟
CK_USART	USART时钟

函数 rcu_deinit

函数rcu_deinit描述见下表:

表 3-347. 函数 rcu_deinit

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

例如:

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

函数 rcu_periph_clock_enable

函数rcu_periph_clock_enable描述见下表:

表 3-348. 函数 rcu_periph_clock_enable

函数名称	rcu_periph_clock_enable
函数原形	void rcu_periph_clock_enable(rcu_periph_enum periph);
功能描述	使能外设时钟
先决条件	-
被调用函数	-
输入参数{in}	
periph	RCU外设，具体参考 表3-337. 枚举类型rcu_periph_enum 。
RCU_GPIOx	GPIOx时钟(x=A,B,C,D,F)
RCU_DMA	DMA时钟
RCU_CRC	CRC时钟
RCU_TSI	TSI时钟
RCU_CFGCMP	CFGCMP时钟

<i>RCU_ADC</i>	ADC时钟
<i>RCU_TIMERx</i>	TIMERx时钟(x=0,1,2,5,13,14,15,16), RCU_TIMER5仅对GD32F350适用
<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_USBFS</i>	USBFS时钟 (仅对GD32F350适用)
<i>RCU_PMU</i>	PMU时钟
<i>RCU_DAC</i>	DAC时钟 (仅对GD32F350适用)
<i>RCU_CEC</i>	CEC时钟 (仅对GD32F350适用)
<i>RCU_CTC</i>	CTC时钟
<i>RCU_RTC</i>	RTC时钟
输出参数{out}	
-	-
返回值	
-	-

例如：

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

函数 `rcu_periph_clock_disable`

函数`rcu_periph_clock_disable`描述见下表：

表 3-349. 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}	
<i>periph</i>	RCU外设, 具体参考 表3-337. 枚举类型rcu_periph_enum 。
<i>RCU_GPIOx</i>	GPIOx时钟(x=A,B,C,D,F)
<i>RCU_DMA</i>	DMA时钟
<i>RCU_CRC</i>	CRC时钟
<i>RCU_TSI</i>	TSI时钟
<i>RCU_CFGCMP</i>	CFGCMP时钟
<i>RCU_ADC</i>	ADC时钟
<i>RCU_TIMERx</i>	TIMERx时钟(x=0,1,2,5,13,14,15,16), RCU_TIMER5仅对GD32F350适用
<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_USBFS</i>	USBFS时钟（仅对GD32F350适用）
<i>RCU_PMU</i>	PMU时钟
<i>RCU_DAC</i>	DAC时钟（仅对GD32F350适用）
<i>RCU_CEC</i>	CEC时钟（仅对GD32F350适用）
<i>RCU_CTC</i>	CTC时钟
<i>RCU_RTC</i>	RTC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 3-350. 函数 **rcu_periph_clock_sleep_enable**

函数名称	rcu_periph_clock_sleep_enable
函数原形	void rcu_periph_clock_sleep_enable(rcu_periph_sleep_enum periph);
功能描述	在睡眠模式下，使能外设时钟
先决条件	-
被调用函数	-
输入参数{in}	
periph	RCU外设，具体参考 表3-338. 枚举类型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 3-351. 函数 rcu_periph_clock_sleep_disable

函数名称	rcu_periph_clock_sleep_disable
函数原形	void rcu_periph_clock_sleep_disable(rcu_periph_sleep_enum periph);
功能描述	在睡眠模式下，禁能外设时钟
先决条件	-
被调用函数	-
输入参数{in}	
periph	RCU外设，具体参考 表3-338. 枚举类型rcu_periph_sleep_enum 。
RCU_FMC_SLP	FMC时钟
RCU_SRAM_SLP	SRAM时钟
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* disable the FMC clock when in sleep mode */

rcu_periph_clock_sleep_disable(RCU_FMC_SLP);
```

函数 rcu_periph_reset_enable

函数rcu_periph_reset_enable描述见下表：

表 3-352. 函数 rcu_periph_reset_enable

函数名称	rcu_periph_reset_enable
函数原形	void rcu_periph_reset_enable(rcu_periph_reset_enum periph_reset);
功能描述	使能外设复位
先决条件	-
被调用函数	-
输入参数{in}	
periph_reset	RCU外设复位，具体参考 表3-339. 枚举类型rcu_periph_reset_enum 。
RCU_GPIOxRST	复位GPIOx时钟(x=A,B,C,D,F)
RCU_TSIRST	复位TSI时钟
RCU_CFGCMRST	复位CFGCMP时钟
RCU_ADCRST	复位ADC时钟
RCU_TIMERxRST	复位TIMERx时钟(x=0,1,2,5,13,14,15,16)，RCU_TIMER5仅对GD32F350适用
RCU_SPIxRST	复位SPIx时钟(x=0,1)
RCU_USARTxRST	复位USARTx时钟(x=0,1)
RCU_WWDGTRST	复位WWDGTRST时钟
RCU_I2CxRST	复位I2Cx时钟(x=0,1)
RCU_USBFSRST	复位USBFS时钟（仅对GD32F350适用）
RCU_PMURST	复位PMU时钟
RCU_DACRST	复位DAC时钟（仅对GD32F350适用）

<i>RCU_CECRST</i>	复位CEC时钟（仅对GD32F350适用）
<i>RCU_CTCRST</i>	复位CTC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 3-353. 函数 rcu_periph_reset_disable

函数名称	rcu_periph_reset_disable
函数原形	void rcu_periph_reset_disable(<i>rcu_periph_reset_enum periph_reset</i>);
功能描述	禁能外设复位
先决条件	-
被调用函数	-
输入参数{in}	
<i>periph_reset</i>	RCU外设复位，具体参考 表3-339. 枚举类型rcu_periph_reset_enum 。
<i>RCU_GPIOxRST</i>	复位GPIOx时钟(x=A,B,C,D,F)
<i>RCU_TSIRST</i>	复位TSI时钟
<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=0,1,2,5,13,14,15,16)，RCU_TIMER5仅对GD32F350适用
<i>RCU_SPIxRST</i>	复位SPIx时钟(x=0,1)
<i>RCU_USARTxRST</i>	复位USARTx时钟(x=0,1)
<i>RCU_WWDGTRST</i>	复位WWDGT时钟
<i>RCU_I2CxRST</i>	复位I2Cx时钟(x=0,1)
<i>RCU_USBFSRST</i>	复位USBFS时钟（仅对GD32F350适用）
<i>RCU_PMURST</i>	复位PMU时钟
<i>RCU_DACRST</i>	复位DAC时钟（仅对GD32F350适用）
<i>RCU_CECRST</i>	复位CEC时钟（仅对GD32F350适用）
<i>RCU_CTCRST</i>	复位CTC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 3-354. 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 3-355. 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 3-356. 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 3-357. 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	

uint32_t	RCU_SCSS_IRC8M/RCU_SCSS_HXTAL/RCU_SCSS_PLL
----------	--

例如：

```
uint32_t temp_cksys_status;  
  
/* get the CK_SYS source */  
  
temp_cksys_status = rcu_system_clock_source_get();
```

函数 **rcu_ahb_clock_config**

函数rcu_ahb_clock_config描述见下表：

表 3-358. 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 3-359. 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	选择CK_AHB时钟x分频作为CK_APB1时钟 (x=1,2,4,8,16)

<i>B_DIVx</i>	
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 3-360. 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}	
<i>ck_apb2</i>	APB2预分频选择
<i>RCU_APB2_CKAH_B_DIVx</i>	选择CK_AHB时钟x分频作为CK_APB2时钟 (x=1,2,4,8,16)
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* configure CK_AHB/8 as CK_APB2 */
rcu_apb2_clock_config(RCU_APB2_CKAHB_DIV8);
```

函数 **rcu_adc_clock_config**

函数rcu_adc_clock_config描述见下表：

表 3-361. 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}	

ck_adc	ADC预分频选择，具体参考 表3-344. 枚举类型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)
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* select CK_IRC28M as CK_ADC */

rcu_adc_clock_config(RCU_ADCCK_IRC28M);
```

函数 rcu_usbfs_clock_config

函数rcu_usbfs_clock_config描述见下表：

表 3-362. 函数 rcu_usbfs_clock_config

函数名称	rcu_usbfs_clock_config
函数原形	void rcu_usbfs_clock_config(uint32_t ck_usbfs);
功能描述	配置USBFS预分频系数
先决条件	-
被调用函数	-
输入参数{in}	
ck_usbfs	USBFS时钟预分频系数
<i>RCU_USBFS_CKP_LL_DIV1</i>	选择 (CK_PLL) 作为USBFS时钟
<i>RCU_USBFS_CKP_LL_DIV1_5</i>	选择 (CK_PLL / 1.5) 作为USBFS时钟
<i>RCU_USBFS_CKP_LL_DIV2</i>	选择 (CK_PLL / 2) 作为USBFS时钟
<i>RCU_USBFS_CKP_LL_DIV2_5</i>	选择 (CK_PLL / 2.5) 作为USBFS时钟
<i>RCU_USBFS_CKP_LL_DIV3</i>	选择 (CK_PLL / 3) 作为USBFS时钟
<i>RCU_USBFS_CKP_LL_DIV3_5</i>	选择 (CK_PLL / 3.5) 作为USBFS时钟
输出参数{out}	

-	-
返回值	
-	-

例如：

```
/* configure the CK_PLL as CK_USBFS */
rcu_usbfs_clock_config(RCU_USBFS_CKPLL_DIV1);
```

函数 rcu_ckout_config

函数rcu_ckout_config描述见下表：

表 3-363. 函数 rcu_ckout_config

函数名称	rcu_ckout_config
函数原形	void rcu_ckout_config(uint32_t ckout_src, uint32_t ckout_div);
功能描述	配置CKOUT时钟源选择及分频系数
先决条件	-
被调用函数	-
输入参数{in}	
ckout_src	CKOUT时钟源选择
RCU_CKOUTSRC_NONE	无时钟输出
RCU_CKOUTSRC_IRC28M	选择内部28M RC振荡器时钟
RCU_CKOUTSRC_IRC40K	选择内部40K RC振荡器时钟
RCU_CKOUTSRC_LXTAL	选择外部低速晶体振荡器时钟（LXTAL）
RCU_CKOUTSRC_CKSYS	选择系统时钟CK_SYS
RCU_CKOUTSRC_IRC8M	选择内部8M RC振荡器时钟
RCU_CKOUTSRC_HXTAL	选择外部高速晶体振荡器时钟（HXTAL）
RCU_CKOUTSRC_CKPLL_DIV1	选择CK_PLL时钟
RCU_CKOUTSRC_CKPLL_DIV2	选择(CK_PLL / 2)时钟
输入参数{in}	
ckout_div	CKOUT分频系数
RCU_CKOUT_DIVx	将CKOUT所选时钟x分频(x=1,2,4,8,16,32,64,128)
输出参数{out}	
-	-

返回值	
-	-

例如：

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

函数 rcu_pll_preselection_config

函数rcu_pll_preselection_config描述见下表：

表 3-364. 函数 rcu_pll_preselection_config

函数名称	rcu_pll_preselection_config
函数原形	void rcu_pll_preselection_config(uint32_t pll_presel);
功能描述	配置主PLL时钟源预选项
先决条件	-
被调用函数	-
输入参数{in}	
pll_presel	PLL时钟源预选项选择
RCU_PLLPRESEL_IRC48M	预选IRC48M作为PLL时钟的时钟源
RCU_PLLPRESEL_HXTAL	预选HXTAL作为PLL时钟的时钟源
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* select HXTAL as PLL preselection clock */
rcu_pll_preselection_config(RCU_PLLPRESEL_HXTAL);
```

函数 rcu_pll_config

函数rcu_pll_config描述见下表：

表 3-365. 函数 rcu_pll_config

函数名称	rcu_pll_config
函数原形	void rcu_pll_config(uint32_t pll_src, uint32_t pll_mul);
功能描述	配置主PLL时钟及倍频因子
先决条件	-
被调用函数	-
输入参数{in}	
pll_src	PLL时钟源选择

<i>RCU_PLLSRC_IRC8M_DIV2</i>	(IRC8M / 2)被选择为PLL时钟的时钟源
<i>RCU_PLLSRC_HXTAL_IRC48M</i>	选择HXTAL时钟或IRC48M时钟作为PLL时钟的时钟源
输入参数{in}	
pll_mul	PLL时钟倍频因子
<i>RCU_PLL_MULx</i>	PLL源时钟 * x (x = 2..64)
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* configure the PLL */
rcu_pll_config(RCU_PLLSRC_HXTAL_IRC48M, RCU_PLL_MUL10);
```

函数 rcu_usart_clock_config

函数rcu_usart_clock_config描述见下表：

表 3-366. 函数 rcu_usart_clock_config

函数名称	rcu_usart_clock_config
函数原形	void rcu_usart_clock_config(uint32_t ck_usart);
功能描述	配置串口时钟
先决条件	-
被调用函数	-
输入参数{in}	
ck_usart	USART0输入时钟源
<i>RCU_USART0SRC_CKAPB2</i>	选择CK_APB2时钟作为CK_USART0时钟
<i>RCU_USART0SRC_CKSYS</i>	选择CK_SYS时钟作为CK_USART0时钟
<i>RCU_USART0SRC_LXTAL</i>	选择CK_LXTAL时钟作为CK_USART0时钟
<i>RCU_USART0SRC_IRC8M</i>	选择CK_IRC8M时钟作为CK_USART0时钟
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* configure the USART */
```

```
rcu_usart_clock_config(RCU_USART0SRC_CKAPB2);
```

函数 `rcu_cec_clock_config`

函数`rcu_cec_clock_config`描述见下表：

表 3-367. 函数 `rcu_cec_clock_config`

函数名称	<code>rcu_cec_clock_config</code>
函数原形	<code>void rcu_cec_clock_config(uint32_t ck_cec);</code>
功能描述	配置CEC时钟
先决条件	-
被调用函数	-
输入参数{in}	
<code>ck_cec</code>	CEC时钟源选择
<code>RCU_CECSRC_IR C8M_DIV244</code>	选择CK_IRC8M / 244作为CEC时钟源
<code>RCU_CECSRC_LX TAL</code>	选择CK_LXTAL作为CEC时钟源
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* configure the CEC clock source selection */

rcu_cec_clock_config(RCU_CECSRC_LXTAL);
```

函数 `rcu_rtc_clock_config`

函数`rcu_rtc_clock_config`描述见下表：

表 3-368. 函数 `rcu_rtc_clock_config`

函数名称	<code>rcu_rtc_clock_config</code>
函数原形	<code>void rcu_rtc_clock_config(uint32_t rtc_clock_source);</code>
功能描述	配置RTC时钟
先决条件	-
被调用函数	-
输入参数{in}	
<code>rtc_clock_source</code>	RTC时钟源选择
<code>RCU_RTCSRC_NO NE</code>	未选择时钟
<code>RCU_RTCSRC_LX TAL</code>	选择CK_LXTAL作为RTC时钟源
<code>RCU_RTCSRC_IRC</code>	选择内部40K RC振荡器时钟作为RTC时钟源

40K	
RCU_RTC_SRC_HX_TAL_DIV32	选择外部高速晶振32分频作为RTC时钟源
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* configure the RTC clock source selection */

rcu_rtc_clock_config(RCU_RTC_SRC_IRC40K);
```

函数 rcu_ck48m_clock_config

函数rcu_ck48m_clock_config描述见下表：

表 3-369. 函数 rcu_ck48m_clock_config

函数名称	rcu_ck48m_clock_config
函数原形	void rcu_ck48m_clock_config(uint32_t ck48m_clock_source);
功能描述	配置CK48M时钟源
先决条件	-
被调用函数	-
输入参数{in}	
ck48m_clock_sour ce	CK48M时钟源选择
RCU_CK48MSRC_PLL48M	选择CK_PLL48M作为CK48M时钟源
RCU_CK48MSRC_IRC48M	选择CK_IRC48M作为CK48M时钟源
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* configure the CK_48M clock source selection */

rcu_ck48m_clock_config (RCU_CK48MSRC_IRC48M);
```

函数 rcu_hxtal_prediv_config

函数rcu_hxtal_prediv_config描述见下表：

表 3-370. 函数 rcu_hxtal_prediv_config

函数名称	rcu_hxtal_prediv_config
------	-------------------------

函数原形	void rcu_hxtal_prediv_config(uint32_t hxtal_prediv);
功能描述	配置HXTAL作为PLL输入源分频因子
先决条件	-
被调用函数	-
输入参数{in}	
hxtal_prediv	PLL时钟源分频因子选择
RCU_PLL_PREDIVx	HXTAL或TRC48M的x分频作为PLL时钟 (x=1..16)
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* configure the PLL clock source selection */

rcu_hxtal_prediv_config(RCU_PLL_PREDIV2);
```

函数 **rcu_lxtal_drive_capability_config**

函数rcu_lxtal_drive_capability_config描述见下表：

表 3-371. 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_drcap);
功能描述	配置LXTAL驱动能力
先决条件	-
被调用函数	-
输入参数{in}	
lxtal_drcap	LXTAL驱动能力
RCU_LXTAL_LOW_DRI	低驱动力
RCU_LXTAL_MED_LOWDRI	中低驱动力
RCU_LXTAL_MED_HIGHDRI	中高驱动力
RCU_LXTAL_HIGH_DRI	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 3-372. 函数 rcu_flag_get

函数名称	rcu_flag_get
函数原形	FlagStatus rcu_flag_get (rcu_flag_enum flag);
功能描述	获取时钟稳定和外设复位标志
先决条件	-
被调用函数	-
输入参数{in}	
flag	时钟稳定和外设复位标志, 参考 表3-340. 枚举类型rcu_flag_enum 。
RCU_FLAG_IRC40K_STB	IRC40K稳定标志
RCU_FLAG_LXTAL_STB	LXTAL稳定标志
RCU_FLAG_IRC8M_STB	IRC8M稳定标志
RCU_FLAG_HXTAL_STB	HXTAL稳定标志
RCU_FLAG_PLLST_B	PLL时钟稳定标志
RCU_FLAG_IRC28M_STB	IRC28M稳定标志
RCU_FLAG_IRC48M_STB	IRC48M稳定标志
RCU_FLAG_V12RS_T	1.2V电压域复位标志
RCU_FLAG_OBLR_ST	选项字节复位标志
RCU_FLAG_EPRS_T	外部引脚复位标志
RCU_FLAG_PORR_ST	电源复位标志
RCU_FLAG_SWRS_T	软件复位标志
RCU_FLAG_FWDG_TRST	独立看门狗复位标志
RCU_FLAG_WWD_GTRST	窗口看门狗复位标志
RCU_FLAG_LPRST	低电压复位标志
输出参数{out}	

-	-
返回值	
-	SET或RESET

例如：

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

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

函数 **rcu_all_reset_flag_clear**

函数rcu_all_reset_flag_clear描述见下表：

表 3-373. 函数 **rcu_all_reset_flag_clear**

函数名称	rcu_all_reset_flag_clear
函数原形	void rcu_all_reset_flag_clear(void);
功能描述	清除所有复位标志位
先决条件	-
被调用函数	-
输入参数{in}	
-	-
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* clear all the reset flag */

rcu_all_reset_flag_clear();
```

函数 **rcu_interrupt_flag_get**

函数rcu_interrupt_flag_get描述见下表：

表 3-374. 函数 **rcu_interrupt_flag_get**

函数名称	rcu_interrupt_flag_get
函数原形	FlagStatus rcu_interrupt_flag_get(rcu_int_flag_enum int_flag);
功能描述	获取时钟稳定中断和时钟阻塞中断标志
先决条件	-
被调用函数	-
输入参数{in}	
Int_flag	中断以及CKM标志，参考 表3-341. 枚举类型rcu_int_flag_enum 。
RCU_INT_FLAG_IR	IRC40K稳定中断标志

C40KSTB	
RCU_INT_FLAG_L_XTALSTB	LXTAL稳定中断标志
RCU_INT_FLAG_IR_C8MSTB	IRC8M稳定中断标志
RCU_INT_FLAG_H_XTALSTB	HXTAL稳定中断标志
RCU_INT_FLAG_P_LLSTB	PLL稳定中断标志
RCU_INT_FLAG_IR_C28MSTB	IRC28M稳定中断标志
RCU_INT_FLAG_IR_C48MSTB	IRC48M稳定中断标志
RCU_INT_FLAG_C_KM	HXTAL时钟阻塞中断标志
输出参数{out}	
-	-
返回值	
FlagStatus	SET 或 RESET

例如：

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

if(SET == rcu_interrupt_flag_get(RCU_INT_FLAG_HXTALSTB)){
}
```

函数 `rcu_interrupt_flag_clear`

函数`rcu_interrupt_flag_clear`描述见下表：

表 3-375. 函数 `rcu_interrupt_flag_clear`

函数名称	rcu_interrupt_flag_clear
函数原形	void rcu_interrupt_flag_clear (rcu_int_flag_clear_enum int_flag_clear);
功能描述	清除中断标志和时钟阻塞中断标志
先决条件	-
被调用函数	-
输入参数{in}	
int_flag_clear	时钟稳定和阻塞中断标志清除，参考 表3-342. 枚举类型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	清除IRC8M稳定中断标志

<i>C8MSTB_CLR</i>	
<i>RCU_INT_FLAG_H_XTALSTB_CLR</i>	清除HXTAL稳定中断标志
<i>RCU_INT_FLAG_P_LLSTB_CLR</i>	清除PLL稳定中断标志
<i>RCU_INT_FLAG_IR_C28MSTB_CLR</i>	清除IRC28M稳定中断标志
<i>RCU_INT_FLAG_IR_C48MSTB_CLR</i>	清除IRC48M稳定中断标志
<i>RCU_INT_FLAG_C_KM_CLR</i>	清除HXTAL时钟阻塞中断标志
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* clear the interrupt HXTAL stabilization interrupt flag */

rcu_interrupt_flag_clear(RCU_INT_FLAG_HXTALSTB_CLR);
```

函数 **rcu_interrupt_enable**

函数rcu_interrupt_enable描述见下表：

表 3-376. 函数 **rcu_interrupt_enable**

函数名称	rcu_interrupt_enable
函数原形	void rcu_interrupt_enable (rcu_int_enum stab_int);
功能描述	使能时钟稳定中断
先决条件	-
被调用函数	-
输入参数{in}	
stab_int	时钟稳定中断，具体参考 表3-343. 枚举类型rcu_int enum 。
<i>RCU_INT_IRC40KS_TB</i>	使能IRC40K稳定中断
<i>RCU_INT_LXTALS_TB</i>	使能LXTAL稳定中断
<i>RCU_INT_IRC8MS_TB</i>	使能IRC8M稳定中断
<i>RCU_INT_HXTALS_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_STB</i>	使能IRC28M稳定中断

<i>RCU_INT_IRC48M STB</i>	使能IRC48M稳定中断
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* enable the HXTAL stabilization interrupt */

rcu_interrupt_enable(RCU_INT_HXTALSTB);
```

函数 `rcu_interrupt_disable`

函数`rcu_interrupt_disable`描述见下表：

表 3-377. 函数 `rcu_interrupt_disable`

函数名称	rcu_interrupt_disable
函数原形	void rcu_interrupt_disable (rcu_int_enum stab_int);
功能描述	除能时钟稳定中断
先决条件	-
被调用函数	-
输入参数{in}	
stb_int	时钟稳定中断，具体参考 表3-343. 枚举类型rcu_int_enum 。
<i>RCU_INT_IRC40KS TB</i>	除能IRC40K稳定中断
<i>RCU_INT_LXTALS TB</i>	除能LXTAL稳定中断
<i>RCU_INT_IRC8MS TB</i>	除能IRC8M稳定中断
<i>RCU_INT_HXTALS TB</i>	除能HXTAL稳定中断
<i>RCU_INT_PLLSTB</i>	除能PLL稳定中断
<i>RCU_INT_IRC28M STB</i>	除能IRC28M稳定中断
<i>RCU_INT_IRC48M STB</i>	除能IRC48M稳定中断
输出参数{out}	
-	-
返回值	
-	-

例如：

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

```
rcu_interrupt_disable(RCU_INT_HXTALSTB);
```

函数 rcu_osc_i_stab_wait

函数rcu_osc_i_stab_wait描述见下表:

表 3-378. 函数 rcu_osc_i_stab_wait

函数名称	rcu_osc_i_stab_wait
函数原形	ErrStatus rcu_osc_i_stab_wait(rcu_osc_i_type_enum osci);
功能描述	等待振荡器稳定标志位置位或振荡器起振超时
先决条件	-
被调用函数	rcu_flag_get
输入参数{in}	
osci	振荡器类型, 具体参考 表3-345. 枚举类型rcu_osc_i_type_enum 。
RCU_HXTAL	高速晶体振荡器
RCU_LXTAL	低速晶体振荡器
RCU_IRC8M	内部8M RC振荡器
RCU_IRC28M	内部28M RC振荡器
RCU_IRC48M	内部48M RC振荡器
RCU_IRC40K	内部40K RC振荡器
RCU_PLL_CK	锁相环
输出参数{out}	
-	-
返回值	
ErrStatus	SUCCESS 或 ERROR

例如:

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

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

函数 rcu_osc_i_on

函数rcu_osc_i_on描述见下表:

表 3-379. 函数 rcu_osc_i_on

函数名称	rcu_osc_i_on
函数原形	void rcu_osc_i_on(rcu_osc_i_type_enum osci);
功能描述	打开振荡器
先决条件	-
被调用函数	-
输入参数{in}	
osci	振荡器类型, 具体参考 表3-345. 枚举类型rcu_osc_i_type_enum 。
RCU_HXTAL	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_IRC48M</i>	内部48M RC振荡器
<i>RCU_IRC40K</i>	内部40K RC振荡器
<i>RCU_PLL_CK</i>	锁相环
输出参数{out}	
-	-
返回值	
-	-

例如：

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

函数 **rcu_osc_off**

函数rcu_osc_off描述见下表：

表 3-380. 函数 **rcu_osc_off**

函数名称	rcu_osc_off
函数原形	void rcu_osc_off(rcu_osc_type_enum osci);
功能描述	关闭振荡器
先决条件	-
被调用函数	-
输入参数{in}	
<i>osci</i>	振荡器类型，具体参考 表3-345. 枚举类型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_IRC48M</i>	内部48M RC振荡器
<i>RCU_IRC40K</i>	内部40K RC振荡器
<i>RCU_PLL_CK</i>	锁相环
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* turn off the high speed crystal oscillator */
rcu_osc_off(RCU_HXTAL);
```

函数 rcu_oscibypass_mode_enable

函数rcu_oscibypass_mode_enable描述见下表:

表 3-381. 函数 rcu_oscibypass_mode_enable

函数名称	rcu_oscibypass_mode_enable
函数原形	void rcu_oscibypass_mode_enable(rcu_oscib_type_enum osci);
功能描述	使能振荡器时钟旁路模式
先决条件	HXTALEN或LXTALEN应在使能振荡器时钟旁路模式前先复位
被调用函数	-
输入参数{in}	
osci	振荡器类型, 具体参考 表3-345. 枚举类型rcu_oscib_type_enum 。
RCU_HXTAL	高速晶体振荡器
RCU_LXTAL	低速晶体振荡器
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* enable the high speed crystal oscillator bypass mode */
rcu_oscibypass_mode_enable(RCU_HXTAL);
```

函数 rcu_oscibypass_mode_disable

函数rcu_oscibypass_mode_disable描述见下表:

表 3-382. 函数 rcu_oscibypass_mode_disable

函数名称	rcu_oscibypass_mode_disable
函数原形	void rcu_oscibypass_mode_disable(rcu_oscib_type_enum osci);
功能描述	除能振荡器时钟旁路模式
先决条件	HXTALEN或LXTALEN应在使能振荡器时钟旁路模式前先复位
被调用函数	-
输入参数{in}	
osci	振荡器类型, 具体参考 表3-345. 枚举类型rcu_oscib_type_enum 。
RCU_HXTAL	高速晶体振荡器
RCU_LXTAL	低速晶体振荡器
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* disable the 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 3-383. 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 3-384. 函数 rcu_hxtal_clock_monitor_disable

函数名称	rcu_hxtal_clock_monitor_disable
函数原形	void rcu_hxtal_clock_monitor_disable(void);
功能描述	除能HXTAL时钟监视器
先决条件	-
被调用函数	-
输入参数{in}	
-	-
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* disable the HXTAL clock monitor */
```

```
rcu_hxtal_clock_monitor_disable();
```

函数 rcu_irc8m_adjust_value_set

函数rcu_irc8m_adjust_value_set描述见下表:

表 3-385. 函数 rcu_irc8m_adjust_value_set

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

例如:

```
/* set the IRC8M adjust value */
rcu_irc8m_adjust_value_set(0x10);
```

函数 rcu_irc28m_adjust_value_set

函数rcu_irc28m_adjust_value_set描述见下表:

表 3-386. 函数 rcu_irc28m_adjust_value_set

函数名称	rcu_irc28m_adjust_value_set
函数原形	void rcu_irc28m_adjust_value_set(uint32_t irc28m_adjval);
功能描述	设置内部28MHz RC振荡器时钟调整值
先决条件	-
被调用函数	-
输入参数{in}	
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 3-387. 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 3-388. 函数 rcu_deepsleep_voltage_set

函数名称	rcu_deepsleep_voltage_set
函数原形	void rcu_deepsleep_voltage_set(uint32_t dsvol);
功能描述	设置深度睡眠模式电压值
先决条件	-
被调用函数	-
输入参数{in}	
dsvol	深度睡眠模式电压值
<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（不推荐客户使用）
<i>RCU_DEEPSLEEP_V_0_8</i>	在深度睡眠模式下内核电压为0.8V（不推荐客户使用）
<i>RCU_DEEPSLEEP_V_0_7</i>	在深度睡眠模式下内核电压为0.7V（不推荐客户使用）
输出参数{out}	
-	-
返回值	

-	-
---	---

例如：

```
/* set the deep-sleep mode voltage */

rcu_deepsleep_voltage_set(RCU_DEEPSLEEP_V_1_0);
```

函数 `rcu_clock_freq_get`

函数`rcu_clock_freq_get`描述见下表：

表 3-389. 函数 `rcu_clock_freq_get`

函数名称	rcu_clock_freq_get
函数原形	uint32_t rcu_clock_freq_get(rcu_clock_freq_enum clock);
功能描述	获取系统、总线以及外设时钟频率
先决条件	-
被调用函数	-
输入参数{in}	
<code>clock</code>	要获取的时钟频率，具体参考 <code>rcu_clock_freq_enum</code>
<code>CK_SYS</code>	系统时钟频率
<code>CK_AHB</code>	AHB时钟频率
<code>CK_APB1</code>	APB1时钟频率
<code>CK_APB2</code>	APB2时钟频率
<code>CK_ADC</code>	ADC时钟频率
<code>CK_CEC</code>	CEC时钟频率
<code>CK_USART</code>	USART时钟频率
输出参数{out}	
-	-
返回值	
<code>uint32_t</code>	系统时钟/AHB时钟/APB1时钟/APB2时钟/ADC时钟/USART时钟频率

例如：

```
uint32_t temp_freq;

/* get the system clock frequency */

temp_freq = rcu_clock_freq_get(CK_SYS);
```

3.18. RTC

实时时钟RTC通常被用作时钟日历。位于备份域中的RTC电路，包含一个32位的累加计数器、一个闹钟、一个预分频器、一个分频器以及RTC时钟配置寄存器。章节[3.18.1](#)描述了FWDGT的寄存器列表，章节[3.18.2](#)对FWDGT库函数进行说明。

3.18.1. 外设寄存器描述

RTC寄存器列表如下表所示：

表 3-390. RTC 寄存器

寄存器名称	寄存器描述
RTC_TIME	RT时间寄存器
RTC_DATE	RTC日期寄存器
RTC_CTL	RTC控制寄存器
RTC_STAT	RTC状态寄存器
RTC_PSC	RTC预分频寄存器
RTC_ALRM0TD	RTC闹钟0时间日期寄存器
RTC_WPK	RTC写保护钥匙寄存器
RTC_SS	RTC亚秒寄存器
RTC_SHIFTCTL	RTC移位控制寄存器
RTC_TTS	RTC时间戳时间寄存器
RTC_DTS	RTC时间戳日期寄存器
RTC_SSTS	RTC时间戳亚秒寄存器
RTC_HRFC	RTC高精度频率补偿寄存器
RTC_TAMP	RTC侵入寄存器
RTC_ALRM0SS	RTC闹钟0亚秒寄存器
RTC_BKP0	RTC备份域寄存器0
RTC_BKP1	RTC备份域寄存器1
RTC_BKP2	RTC备份域寄存器2
RTC_BKP3	RTC备份域寄存器3
RTC_BKP4	RTC备份域寄存器4

3.18.2. 外设库函数描述

RTC库函数列表如下表所示：

表 3-391. 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

库函数名称	库函数描述
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 3-392. 结构体类型 `rtc_parameter_struct`

Member name	Function description
rtc_year	RTC年份值: 0x0 - 0x99 (BCD格式)
rtc_month	RTC月份值
rtc_date	RTC日期值: 0x1 - 0x31 (BCD格式)
rtc_day_of_week	RTC星期值
rtc_hour	RTC小时值
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 - 0xFFFF
rtc_am_pm	RTC AM / PM值
rtc_display_format	RTC时间格式

结构体 `rtc_alarm_struct`

表 3-393. 结构体类型 `rtc_alarm_struct`

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

结构体 `rtc_timestamp_struct`

表 3-394. 结构体 `rtc_timestamp_struct`

Member name	Function description
<code>rtc_timestamp_month_h</code>	RTC时间戳月份值
<code>rtc_timestamp_date</code>	RTC 时间戳日期值: 0x1 - 0x31 (BCD格式)
<code>rtc_timestamp_day</code>	RTC时间戳星期值
<code>rtc_timestamp_hour</code>	RTC 时间戳小时值
<code>rtc_timestamp_minute</code>	RTC时间戳分钟值: 0x0 - 0x59 (BCD格式)
<code>rtc_timestamp_second</code>	RTC时间戳秒数值: 0x0 - 0x59 (BCD格式)
<code>rtc_am_pm</code>	RTC时间戳AM/PM数值

结构体 `rtc_tamper_struct`

表 3-395. 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_precharge_enable</code>	RTC在电压电平检测期间的预充电功能
<code>rtc_tamper_precharge_time</code>	RTC侵入事件电平检测采样预充电时间, 如果预充电功能使能
<code>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 3-396. 函数 rtc_deinit

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

例如:

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

函数 rtc_init

函数rtc_init描述见下表:

表 3-397. 函数 rtc_init

函数名称	rtc_init
函数原型	ErrStatus rtc_init(rtc_parameter_struct* rtc_initpara_struct);
功能描述	初始化RTC寄存器
先决条件	-
输入参数{in}	
rtc_initpara_struct	初始化结构体, 结构体成员参考 表3-392. 结构体类型rtc_parameter_struct
输出参数{out}	
-	-
返回值	
ErrStatus	ERROR或SUCCESS

例如:

```
rtc_parameter_struct rtc_initpara_struct;  
  
ErrStatus error_status;  
  
/* initialize RTC registers */  
  
ErrStatus error_status = rtc_init(& rtc_initpara_struct);
```

函数 `rtc_init_mode_enter`

函数`rtc_init_mode_enter`描述见下表:

表 3-398. 函数 `rtc_init_mode_enter`

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

例如:

```
ErrStatus error_status

/* enter RTC init mode */

ErrStatus error_status = rtc_init_mode_enter();
```

函数 `rtc_init_mode_exit`

函数`rtc_init_mode_exit`描述见下表:

表 3-399. 函数 `rtc_init_mode_exit`

函数名称	rtc_init_mode_exit
函数原型	void rtc_init_mode_exit(void);
功能描述	退出RTC初始化模式
先决条件	-
输入参数{in}	
-	-
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* exit RTC init mode */

rtc_init_mode_exit();
```

函数 `rtc_register_sync_wait`

函数`rtc_register_sync_wait`描述见下表:

表 3-400. 函数 `rtc_register_sync_wait`

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

例如:

```
ErrStatus error_status;

/* wait until RTC_TIME and RTC_DATE registers are synchronized with APB clock, and the
shadow registers are updated */

ErrStatus error_status = rtc_register_sync_wait ( );
```

函数 `rtc_current_time_get`

函数`rtc_current_time_get`描述见下表:

表 3-401. 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	-
输入参数{in}	
-	-
输出参数{out}	
rtc_initpara_struct	初始化结构体，结构体成员参考 表3-392. 结构体类型<code>rtc_parameter_struct</code>
返回值	
-	-

例如:

```
rtc_parameter_struct rtc_initpara_struct;

/* get current time and date */

rtc_current_time_get (& rtc_initpara_struct);
```

函数 `rtc_subsecond_get`

函数`rtc_subsecond_get`描述见下表:

表 3-402. 函数 `rtc_subsecond_get`

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

例如:

```
uint32_t sub_second;

/* get current subsecond value */

uint32_t sub_second = rtc_subsecond_get();
```

函数 `rtc_alarm_config`

函数`rtc_alarm_config`描述见下表:

表 3-403. 函数 `rtc_alarm_config`

函数名称	rtc_alarm_config
函数原型	<code>void rtc_alarm_config(rtc_alarm_struct* rtc_alarm_time);</code>
功能描述	配置RTC闹钟
先决条件	-
输入参数{in}	
<code>rtc_alarm_time</code>	闹钟结构体, 结构体成员参考 表3-393. 结构体类型<code>rtc_alarm_struct</code>
输出参数{out}	
-	-
返回值	
-	-

例如:

```
rtc_alarm_struct rtc_alarm_time;

/* rtc_alarm_config */

rtc_alarm_config (&rtc_alarm_time);
```

函数 `rtc_alarm_subsecond_config`

函数`rtc_alarm_subsecond_config`描述见下表：

表 3-404. 函数 `rtc_alarm_subsecond_config`

函数名称	<code>rtc_alarm_subsecond_config</code>
函数原型	<code>void rtc_alarm_subsecond_config(uint32_t mask_subsecond, uint32_t subsecond);</code>
功能描述	配置RTC闹钟的亚秒值
先决条件	-
输入参数{in}	
<code>mask_subsecond</code>	闹钟亚秒屏蔽位
<code>RTC_MASKSSC_0_14</code>	屏蔽闹钟亚秒设置
<code>RTC_MASKSSC_1_14</code>	屏蔽RTC_ALRM0SS_SSC[14:1], SSC[0]位用于时间匹配
<code>RTC_MASKSSC_2_14</code>	屏蔽RTC_ALRM0SS_SSC[14:2], SSC[1:0]位用于时间匹配
<code>RTC_MASKSSC_3_14</code>	屏蔽RTC_ALRM0SS_SSC[14:3], SSC[2:0]位用于时间匹配
<code>RTC_MASKSSC_4_14</code>	屏蔽RTC_ALRM0SS_SSC[14:4], SSC[3:0]位用于时间匹配
<code>RTC_MASKSSC_5_14</code>	屏蔽RTC_ALRM0SS_SSC[14:5], SSC[4:0]位用于时间匹配
<code>RTC_MASKSSC_6_14</code>	屏蔽RTC_ALRM0SS_SSC[14:6], SSC[5:0]位用于时间匹配
<code>RTC_MASKSSC_7_14</code>	屏蔽RTC_ALRM0SS_SSC[14:7], SSC[6:0]位用于时间匹配
<code>RTC_MASKSSC_8_14</code>	屏蔽RTC_ALRM0SS_SSC[14:8], SSC[7:0]位用于时间匹配
<code>RTC_MASKSSC_9_14</code>	屏蔽RTC_ALRM0SS_SSC[14:9], SSC[8:0]位用于时间匹配
<code>RTC_MASKSSC_10_14</code>	屏蔽RTC_ALRM0SS_SSC[14:10], SSC[9:0]位用于时间匹配
<code>RTC_MASKSSC_11_14</code>	屏蔽RTC_ALRM0SS_SSC[14:11], SSC[10:0]位用于时间匹配
<code>RTC_MASKSSC_12_14</code>	屏蔽RTC_ALRM0SS_SSC[14:12], SSC[11:0]位用于时间匹配
<code>RTC_MASKSSC_13_14</code>	屏蔽RTC_ALRM0SS_SSC[14:13], SSC[12:0]位用于时间匹配
<code>RTC_MASKSSC_NON_E</code>	无屏蔽, SSC[14:0]位用于时间匹配
输入参数{in}	
<code>subsecond</code>	闹钟亚秒值(0x000 - 0x7FFF)
输出参数{out}	
-	-
返回值	
-	-

例如：

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

函数 `rtc_alarm_get`

函数`rtc_alarm_get`描述见下表:

表 3-405. 函数 `rtc_alarm_get`

函数名称	<code>rtc_alarm_get</code>
函数原型	<code>void rtc_alarm_get(rtc_alarm_struct* rtc_alarm_time);</code>
功能描述	获取RTC闹钟
先决条件	-
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 表3-393. 结构体类型<code>rtc_alarm_struct</code>
返回值	
-	-

例如:

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

函数 `rtc_alarm_subsecond_get`

函数`rtc_alarm_subsecond_get`描述见下表:

表 3-406. 函数 `rtc_alarm_subsecond_get`

函数名称	<code>rtc_alarm_subsecond_get</code>
函数原型	<code>uint32_t rtc_alarm_subsecond_get(void);</code>
功能描述	获取RTC闹钟亚秒值
先决条件	-
输入参数{in}	
-	-
输出参数{out}	
-	-
返回值	
<code>uint32_t</code>	RTC 闹钟亚秒值(0x0 - 0x7FFF)

例如:

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

函数 `rtc_alarm_enable`

函数`rtc_alarm_enable`描述见下表:

表 3-407. 函数 `rtc_alarm_enable`

函数名称	<code>rtc_alarm_enable</code>
函数原型	<code>void rtc_alarm_enable(void);</code>
功能描述	使能RTC闹钟
先决条件	-
输入参数{in}	
-	-
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* enable RTC alarm */

rtc_alarm_enable();
```

函数 `rtc_alarm_disable`

函数`rtc_alarm_disable`描述见下表:

表 3-408. 函数 `rtc_alarm_disable`

函数名称	<code>rtc_alarm_disable</code>
函数原型	<code>ErrStatus rtc_alarm_disable(void);</code>
功能描述	失能RTC闹钟
先决条件	-
输入参数{in}	
-	-
输出参数{out}	
-	-
返回值	
<code>ErrStatus</code>	ERROR或SUCCESS

例如:

```
ErrStatus error_status;

/* disable RTC alarm */

ErrStatus error_status = rtc_alarm_disable();
```

函数 `rtc_timestamp_enable`

函数can_init描述见下表:

表 3-409. 函数 `rtc_timestamp_enable`

函数名称	rtc_timestamp_enable
函数原型	<code>void rtc_timestamp_enable(uint32_t edge);</code>
功能描述	使能RTC时间戳
先决条件	-
输入参数{in}	
<code>edge</code>	选定哪种边沿触发时间戳检测
<code>RTC_TIMESTAMP_RISING_EDGE</code>	上升沿是时间戳事件有效检测沿
<code>RTC_TIMESTAMP_FALLING_EDGE</code>	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 3-410. 函数 `rtc_timestamp_disable`

函数名称	rtc_timestamp_disable
函数原型	<code>void rtc_timestamp_disable(void);</code>
功能描述	失能RTC时间戳
先决条件	-
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 3-411. 函数 `rtc_timestamp_get`

函数名称	<code>rtc_timestamp_get</code>
函数原型	<code>void rtc_timestamp_get(rtc_timestamp_struct* rtc_timestamp);</code>
功能描述	获取RTC时间戳时间和日期
先决条件	-
输入参数{in}	
-	-
输出参数{out}	
<code>rtc_timestamp</code>	时间戳结构体，结构体成员参考 表3-394. 结构体<code>rtc_timestamp_struct</code>
返回值	
-	-

例如：

```
rtc_timestamp_struct rtc_timestamp;
/* get RTC timestamp time and date */
rtc_timestamp_get (& rtc_timestamp);
```

函数 `rtc_timestamp_subsecond_get`

函数`rtc_timestamp_subsecond_get`描述见下表：

表 3-412. 函数 `rtc_timestamp_subsecond_get`

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

例如：

```
uint32_t subsecond;
/* get RTC time-stamp subsecond */
uint32_t subsecond = rtc_timestamp_subsecond_get ( );
```

函数 `rtc_tamper_enable`

函数`rtc_tamper_enable`描述见下表：

表 3-413. 函数 `rtc_tamper_enable`

函数名称	rtc_tamper_enable
函数原型	<code>void rtc_tamper_enable(rtc_tamper_struct* rtc_tamper);</code>
功能描述	使能RTC侵入检测
先决条件	-
输入参数{in}	
<code>rtc_tamper</code>	tamper化结构体，结构体成员参考 表3-395. 结构体<code>rtc_tamper_struct</code>
输出参数{out}	
-	-
返回值	
-	-

例如：

```
rtc_tamper_struct rtc_tamper

/* enable RTC tamper */

rtc_tamper_enable (& rtc_tamper);
```

函数 `rtc_tamper_disable`

函数`rtc_tamper_disable`描述见下表：

表 3-414. 函数 `rtc_tamper_disable`

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

例如：

```
/* disable RTC tamper */

rtc_tamper_disable (RTC_TAMPER0);
```

函数 `rtc_interrupt_enable`

函数`rtc_interrupt_enable`描述见下表:

表 3-415. 函数 `rtc_interrupt_enable`

函数名称	<code>rtc_interrupt_enable</code>
函数原型	<code>void rtc_interrupt_enable(uint32_t interrupt);</code>
功能描述	使能RTC指定的中断
先决条件	-
输入参数{in}	
<code>interrupt</code>	选定被使能的中断源
<code>RTC_INT_TIMESTAMP</code>	时间戳中断
<code>RTC_INT_ALARM</code>	闹钟中断
<code>RTC_INT_TAMP</code>	侵入检测中断
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* 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 3-416. 函数 `rtc_interrupt_disable`

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

例如:

```
/* 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 3-417. 函数 `rtc_flag_get`

函数名称	rtc_flag_get
函数原型	FlagStatus rtc_flag_get(uint32_t flag);
功能描述	获取指定中断标志位
先决条件	-
输入参数{in}	
flag	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_TAMPO</i>	tamper 0事件标志
<i>RTC_FLAG_TIMESTAMPOVERFLOW</i>	时间戳事件溢出标志
<i>RTC_FLAG_TIMESTAMPMP</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_ALARM0WRITTEN</i>	Alarm0配置可写标志
输出参数{out}	
-	-
返回值	
FlagStatus	SET 或 RESET

例如：

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

函数 **rtc_flag_clear**

函数`rtc_flag_clear`描述见下表：

表 3-418. 函数 `rtc_flag_clear`

函数名称	rtc_flag_clear
函数原型	void rtc_flag_clear(uint32_t flag);

功能描述	清除指定中断标志位
先决条件	-
输入参数{in}	
flag	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_OVERFLOW</i>	时间戳事件溢出标志
<i>RTC_FLAG_TIMESTAMP_MP</i>	时间戳事件标志
<i>RTC_FLAG_ALARM0</i>	Alarm0发生标志
<i>RTC_FLAG_RSYN</i>	寄存器同步标志
输出参数{out}	
-	-
返回值	
-	-

例如：

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

rtc_flag_clear (RTC_FLAG_TIMESTAMP);
```

函数 **rtc_alter_output_config**

函数**rtc_alter_output_config**描述见下表：

表 3-419. 函数 rtc_alter_output_config

函数名称	rtc_alter_output_config
函数原型	void rtc_alter_output_config(uint32_t source, uint32_t mode);
功能描述	配置RTC备用输出源
先决条件	-
输入参数{in}	
source	指定输出信号
<i>RTC_CALIBRATION_512HZ</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
输入参数{in}	
mode	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

<i>T_PP</i>	
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* configure rtc alternate output source */

rtc_alter_output_config (RTC_ALARM_LOW, RTC_ALARM_OUTPUT_PP);
```

函数 **rtc_calibration_config**

函数 **rtc_calibration_config** 描述见下表：

表 3-420. 函数 **rtc_calibration_config**

Function name	rtc_calibration_config
Function prototype	ErrStatus rtc_calibration_config(uint32_t window, uint32_t plus, uint32_t minus);
Function descriptions	配置RTC校准寄存器
Precondition	-
输入参数{in}	
<i>window</i>	选择校准窗口
<i>RTC_CALIBRATION_WINDOW_32S</i>	如果RTCCLK = 32768Hz在32秒校准窗增加2exp20 RTCCLK周期
<i>RTC_CALIBRATION_WINDOW_16S</i>	如果RTCCLK = 32768 Hz在16秒校准窗增加2exp19 RTCCLK周期
<i>RTC_CALIBRATION_WINDOW_8S</i>	如果RTCCLK = 32768 Hz在8秒校准窗增加2exp18 RTCCLK周期
输入参数{in}	
<i>plus</i>	增加或者不增加RTC脉冲
<i>RTC_CALIBRATION_PLUS_SET</i>	每2048个RTC脉冲增加一个RTC脉冲
<i>RTC_CALIBRATION_PLUS_RESET</i>	无影响
输入参数{in}	
<i>minus</i>	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：

```
ErrStatus error_status;
```

```

/* configure RTC calibration register */

ErrStatus error_status = rtc_calibration_config(RTC_CALIBRATION_WINDOW_32S,
    RTC_CALIBRATION_PLUS_SET, 0x1FF);
  
```

函数 `rtc_hour_adjust`

函数`rtc_hour_adjust`描述见下表:

表 3-421. 函数 `rtc_hour_adjust`

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

例如:

```

/* adjust the daylight saving time by adding one hour from the current time */

rtc_hour_adjust (RTC_CTL_A1H);
  
```

函数 `rtc_second_adjust`

函数`rtc_second_adjust`描述见下表:

表 3-422. 函数 `rtc_second_adjust`

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

-	-
返回值	
-	-

例如：

```
ErrStatus error_status;

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

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

函数 **rtc_bypass_shadow_enable**

函数`rtc_bypass_shadow_enable`描述见下表：

表 3-423. 函数 `rtc_bypass_shadow_enable`

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

例如：

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

rtc_bypass_shadow_enable ( );
```

函数 **rtc_bypass_shadow_disable**

函数`rtc_bypass_shadow_disable`描述见下表：

表 3-424. 函数 `rtc_bypass_shadow_disable`

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

-	-
---	---

例如：

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

rtc_bypass_shadow_disable();
```

函数 **rtc_refclock_detection_enable**

函数`rtc_refclock_detection_enable`描述见下表：

表 3-425. 函数 **rtc_refclock_detection_enable**

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

例如：

```
ErrStatus error_status;

/* enable RTC reference clock detection function */

ErrStatus error_status = rtc_refclock_detection_enable();
```

函数 **rtc_refclock_detection_disable**

函数`rtc_refclock_detection_disable`描述见下表：

表 3-426. 函数 **rtc_refclock_detection_disable**

函数名称	rtc_refclock_detection_disable
函数原型	ErrStatus rtc_refclock_detection_disable(void);
功能描述	失能RTC参考时钟检测功能
先决条件	-
输入参数{in}	
-	-
输出参数{out}	
-	-
返回值	
ErrStatus	ERROR或SUCCESS

例如：

```

ErrStatus error_status;

/* disableRTC reference clock detection function */

ErrStatus error_status = rtc_refclock_detection_disable();

```

3.19. SPI/I2S

SPI/I2S模块可以通过SPI协议或I2S音频协议与外部设备进行通信。章节[3.19.1](#)描述了SPI/I2S的寄存器列表，章节[3.19.2](#)对SPI/I2S库函数进行说明。

3.19.1. 外设寄存器说明

SPI/I2S寄存器列表如下表所示：

表 3-427. SPI/I2S 寄存器

寄存器名称	寄存器描述
SPI_CTL0	控制寄存器0
SPI_CTL1	控制寄存器1
SPI_STAT	状态寄存器
SPI_DATA	数据寄存器
SPI_CRCPOLY	CRC多项式寄存器
SPI_RCRC	接收CRC寄存器
SPI_TCRC	发送CRC寄存器
SPI_I2SCTL	I2S控制寄存器
SPI_I2SPSC	I2S时钟分频寄存器
SPI_QCTL	四路SPI控制寄存器

3.19.2. 外设库函数说明

SPI/I2S库函数列表如下表所示：

表 3-428. SPI/I2S 库函数

库函数名称	库函数描述
spi_i2s_deinit	复位外设SPIx/I2Sx
spi_struct_para_init	将SPI结构体中所有参数初始化为默认值
spi_init	初始化外设SPIx
spi_enable	使能外设SPIx
spi_disable	禁能外设SPIx
i2s_init	初始化外设I2Sx
i2s_psc_config	配置I2Sx预分频器
i2s_enable	使能外设I2Sx
i2s_disable	禁能外设I2Sx
spi_nss_output_enable	使能外设SPIx NSS输出

库函数名称	库函数描述
spi_nss_output_disable	禁能外设SPIx NSS输出
spi_nss_internal_high	NSS软件模式下NSS引脚拉高
spi_nss_internal_low	NSS软件模式下NSS引脚拉低
spi_dma_enable	使能外设SPIx的DMA功能
spi_dma_disable	禁能外设SPIx的DMA功能
spi_i2s_data_frame_format_config	配置外设SPIx/I2Sx数据帧格式
spi_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	禁能外设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

结构体 **spi_parameter_struct**

表 3-429. 结构体类型 **spi_parameter_struct**

成员名称	功能描述
device_mode	主机或设备模式配置 (SPI_MASTER, SPI_SLAVE)
trans_mode	传输模式 (SPI_TRANSMODE_FULLDUPLEX, SPI_TRANSMODE_RECEIVEONLY, SPI_TRANSMODE_BDRECEIVE, SPI_TRANSMODE_BDTRANSMIT)
frame_size	数据帧格式配置 (SPI_FRAMESIZE_8BIT, SPI_FRAMESIZE_16BIT)

成员名称	功能描述
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 3-430. 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 3-431. 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 表3-429. 结构体类型 spi_parameter_struct 。
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 3-432. 函数 spi_init

函数名称	spi_init
函数原形	void spi_init(uint32_t spi_periph, spi_parameter_struct* spi_struct);
功能描述	初始化外设SPIx
先决条件	-
被调用函数	-
输入参数{in}	
spi_periph	外设SPIx
SPIx	x=0,1
输入参数{in}	
spi_struct	初始化结构体，结构体成员参考 表3-429. 结构体类型 spi_parameter_struct 。
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* initialize SPI0 */

spi_parameter_struct spi_init_struct;

spi_init_struct.trans_mode      = SPI_TRANSMODE_BDTRANSMIT;
spi_init_struct.device_mode    = SPI_MASTER;
spi_init_struct.frame_size     = SPI_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;
spi_init(SPI0, &spi_init_struct);
  
```

函数 spi_enable

函数spi_enable描述见下表：

表 3-433. 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 3-434. 函数 spi_disable

函数名称	spi_disable
函数原形	void spi_disable(uint32_t spi_periph);
功能描述	禁能外设SPIx
先决条件	-
被调用函数	-
输入参数{in}	
spi_periph	外设SPIx
SPIx	x=0,1
输出参数{out}	
-	-
返回值	

-	-
---	---

例如：

```
/* disable SPI0 */
spi_disable(SPI0);
```

函数 i2s_init

函数i2s_init描述见下表：

表 3-435. 函数 i2s_init

函数名称	i2s_init
函数原形	void i2s_init(uint32_t spi_periph,uint32_t mode, uint32_t standard, uint32_t ckpl);
功能描述	初始化外设I2S0
先决条件	-
被调用函数	-
输入参数{in}	
spi_periph	外设I2S0
SPIx	x=0
输入参数{in}	
mode	I2S运行模式
I2S_MODE_SLAVE_TX	I2S从机发送模式
I2S_MODE_SLAVE_RX	I2S从机接收模式
I2S_MODE_MASTER_RT	I2S主机发送模式
I2S_MODE_MASTER_RRX	I2S主机接收模式
输入参数{in}	
standard	I2S标准选择
I2S_STD_PHILLIPS	I2S飞利浦标准
I2S_STD_MSB	I2S MSB对齐标准
I2S_STD_LSB	I2S LSB对齐标准
I2S_STD_PCMSHORTFRAME_RT	I2S PCM短帧标准
I2S_STD_PCMLONG_G	I2S PCM长帧标准
输入参数{in}	
ckpl	I2S空闲状态时钟极性
I2S_CKPL_LOW	I2S_CK空闲状态为低电平
I2S_CKPL_HIGH	I2S_CK空闲状态为高电平

输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* initialize I2S0 */

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

函数 i2s_psc_config

函数i2s_psc_config描述见下表：

表 3-436. 函数 i2s_psc_config

函数名称	i2s_psc_config
函数原形	void i2s_psc_config(uint32_t spi_periph, uint32_t audiosample, uint32_t frameformat, uint32_t mckout);
功能描述	配置I2S0预分频器
先决条件	-
被调用函数	rcu_clock_freq_get
输入参数{in}	
spi_periph	外设I2S0
SPIx	x=0
输入参数{in}	
audiosample	I2S音频采样频率
I2S_AUDIOSAMPL_E_8K	音频采样频率为8KHz
I2S_AUDIOSAMPL_E_11K	音频采样频率为11KHz
I2S_AUDIOSAMPL_E_16K	音频采样频率为16KHz
I2S_AUDIOSAMPL_E_22K	音频采样频率为22KHz
I2S_AUDIOSAMPL_E_32K	音频采样频率为32KHz
I2S_AUDIOSAMPL_E_44K	音频采样频率为44KHz
I2S_AUDIOSAMPL_E_48K	音频采样频率为48KHz
I2S_AUDIOSAMPL_E_96K	音频采样频率为96KHz
I2S_AUDIOSAMPL_E_192K	音频采样频率为192KHz

输入参数{in}	
frameformat	I2S数据长度和通道长度
<i>I2S_FRAMEFORMAT_DT16B_CH16B</i>	I2S数据长度为16位，通道长度为16位
<i>I2S_FRAMEFORMAT_DT16B_CH32B</i>	I2S数据长度为16位，通道长度为32位
<i>I2S_FRAMEFORMAT_DT24B_CH32B</i>	I2S数据长度为24位，通道长度为32位
<i>I2S_FRAMEFORMAT_DT32B_CH32B</i>	I2S数据长度为32位，通道长度为32位
输入参数{in}	
mckout	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
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* configure I2S0 prescaler */

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

函数 i2s_enable

函数i2s_enable描述见下表：

表 3-437. 函数 i2s_enable

函数名称	i2s_enable
函数原形	void i2s_enable(uint32_t spi_periph);
功能描述	使能外设I2S0
先决条件	-
被调用函数	-
输入参数{in}	
spi_periph	外设I2S0
<i>SPIx</i>	x=0
输出参数{out}	
-	-
返回值	
-	-

例如：

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

```
i2s_enable(SPI0);
```

函数 i2s_disable

函数i2s_disable描述见下表：

表 3-438. 函数 i2s_disable

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

例如：

```
/* disable I2S0 */
```

```
i2s_disable(SPI0);
```

函数 spi_nss_output_enable

函数spi_nss_output_enable描述见下表：

表 3-439. 函数 spi_nss_output_enable

函数名称	spi_nss_output_enable
函数原形	void spi_nss_output_enable(uint32_t spi_periph);
功能描述	使能外设SPIx NSS输出
先决条件	-
被调用函数	-
输入参数{in}	
spi_periph	外设SPIx
SPIx	x=0,1
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* enable SPI0 NSS output */

spi_nss_output_enable(SPI0);
```

函数 **spi_nss_output_disable**

函数spi_nss_output_disable描述见下表：

表 3-440. 函数 spi_nss_output_disable

函数名称	spi_nss_output_disable
函数原形	void spi_nss_output_disable(uint32_t spi_periph);
功能描述	禁能外设SPIx NSS输出
先决条件	-
被调用函数	-
输入参数{in}	
spi_periph	外设SPIx
SPIx	x=0,1
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* disable SPI0 NSS output */

spi_nss_output_disable(SPI0);
```

函数 **spi_nss_internal_high**

函数spi_nss_internal_high描述见下表：

表 3-441. 函数 spi_nss_internal_high

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

例如：

```
/* SPI0 NSS pin is pulled high level in software mode */
```

```
spi_nss_internal_high(SPI0);
```

函数 **spi_nss_internal_low**

函数spi_nss_internal_low描述见下表：

表 3-442. 函数 spi_nss_internal_low

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

例如：

```
/* SPI0 NSS pin is pulled low level in software mode */
```

```
spi_nss_internal_low(SPI0);
```

函数 **spi_dma_enable**

函数spi_dma_enable描述见下表：

表 3-443. 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	-
输入参数{in}	
spi_periph	外设SPIx
SPIx	x=0,1
输入参数{in}	
dma	SPI DMA模式
SPI_DMA_TRANSMIT	SPI发送缓冲区DMA使能

<i>SPI_DMA_RECEIVE</i>	SPI接收缓冲区DMA使能
输出参数{out}	
-	-
返回值	
-	-

例如：

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

函数 **spi_dma_disable**

函数spi_dma_disable描述见下表：

表 3-444. 函数 **spi_dma_disable**

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

例如：

```
/* disable SPI0 transmit data DMA function */
spi_dma_disable(SPI0, SPI_DMA_TRANSMIT);
```

函数 **spi_i2s_data_frame_format_config**

函数spi_i2s_data_frame_format_config描述见下表：

表 3-445. 函数 spi_i2s_data_frame_format_config

函数名称	spi_i2s_data_frame_format_config
函数原形	void spi_i2s_data_frame_format_config(uint32_t spi_periph, uint16_t frame_format);
功能描述	配置外设SPIx/I2S0数据帧格式
先决条件	-
被调用函数	-
输入参数{in}	
spi_periph	外设SPIx
SPIx	x=0,1
输入参数{in}	
frame_format	SPI帧大小
SPI_FRAMESIZE_8BIT	SPI 8位数据帧格式
SPI_FRAMESIZE_16BIT	SPI 16位数据帧格式
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* configure SPI0/I2S0 data frame format size is 16 bits */
spi_i2s_data_frame_format_config(SPI1, SPI_FRAMESIZE_16BIT);
```

函数 spi_i2s_data_transmit

函数spi_i2s_data_transmit描述见下表：

表 3-446. 函数 spi_i2s_data_transmit

函数名称	spi_i2s_data_transmit
函数原形	void spi_i2s_data_transmit(uint32_t spi_periph, uint16_t data);
功能描述	SPI发送数据
先决条件	-
被调用函数	-
输入参数{in}	
spi_periph	外设SPIx
SPIx	x=0,1
输入参数{in}	
data	16位数据
输出参数{out}	
-	-
返回值	

-	-
---	---

例如：

```
/* SPI0 transmit data */
spi_i2s_data_transmit(SPI0, spi0_send_array[send_n]);
```

函数 **spi_i2s_data_receive**

函数spi_i2s_data_receive描述见下表：

表 3-447. 函数 **spi_i2s_data_receive**

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

例如：

```
/* SPI0 receive data */
spi0_receive_array[receive_n] = spi_i2s_data_receive(SPI0);
```

函数 **spi_bidirectional_transfer_config**

函数spi_bidirectional_transfer_config描述见下表：

表 3-448. 函数 **spi_bidirectional_transfer_config**

函数名称	spi_bidirectional_transfer_config
函数原形	void spi_bidirectional_transfer_config(uint32_t spi_periph, uint32_t transfer_direction);
功能描述	配置外设SPIx的数据传输方向
先决条件	-
被调用函数	-
输入参数{in}	
spi_periph	外设SPIx
<i>SPIx</i>	x=0,1
输入参数{in}	
transfer_direction	SPI双向传输输出使能

<i>SPI_BIDIRECTIONAL_AL_TRANSMIT</i>	SPI工作在只发送模式
<i>SPI_BIDIRECTIONAL_AL_RECEIVE</i>	SPI工作在只接收模式
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* SPI0 works in transmit-only mode */

spi_bidirectional_transfer_config(SPI0, SPI_BIDIRECTIONAL_TRANSMIT);
```

函数 **spi_crc_polynomial_set**

函数spi_crc_polynomial_set描述见下表：

表 3-449. 函数 **spi_crc_polynomial_set**

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

例如：

```
/* set SPI0 CRC polynomial */

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 3-450. 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 3-451. 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 3-452. 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 3-453. 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 3-454. 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 3-455. 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 3-456. 函数 spi_ti_mode_disable

函数名称	spi_ti_mode_disable
函数原形	void spi_ti_mode_disable(uint32_t spi_periph);
功能描述	禁能SPI TI模式
先决条件	-
被调用函数	-
输入参数{in}	
spi_periph	外设SPIx
SPIx	x=0,1
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* disable SPI0 TI mode */
spi_ti_mode_disable(SPI0);
```

函数 **spi_nssp_mode_enable**

函数spi_nssp_mode_enable描述见下表：

表 3-457. 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 3-458. 函数 spi_nssp_mode_disable

函数名称	spi_nssp_mode_disable
函数原形	void spi_nssp_mode_disable(uint32_t spi_periph);
功能描述	禁能SPI NSS脉冲模式
先决条件	-
被调用函数	-
输入参数{in}	
spi_periph	外设SPIx
SPIx	x=0,1
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* disable SPI0 NSS pulse mode */
spi_nssp_mode_disable(SPI0);
```

函数 **qspi_enable**

函数qspi_enable描述见下表:

表 3-459. 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
SPIx	x=1
输出参数{out}	
-	-
返回值	
-	-

例如:

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

```
qspi_enable(SPI1);
```

函数 qspi_disable

函数qspi_disable描述见下表:

表 3-460. 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	禁能四线SPI模式	
先决条件	-	
被调用函数	-	
输入参数{in}		
spi_periph	外设SPIx	
SPIx	x=1	
输出参数{out}		
-	-	
返回值		
-	-	

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 3-461. 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	-	
输入参数{in}		
spi_periph	外设SPIx	
SPIx	x=1	
输出参数{out}		
-	-	
返回值		
-	-	

例如:

```
/* enable SPI1 quad wire write */
```

```
qspi_write_enable(SPI1);
```

函数 qspi_read_enable

函数qspi_read_enable描述见下表:

表 3-462. 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	-
输入参数{in}	
spi_periph	外设SPIx
SPIx	x=1
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* enable SPI1 quad wire read */

qspi_read_enable(SPI1);
```

函数 qspi_io23_output_enable

函数qspi_io23_output_enable描述见下表:

表 3-463. 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}	
spi_periph	外设SPIx
SPIx	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 3-464. 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	禁能SPI_IO2和SPI_IO3输出	
先决条件	-	
被调用函数	-	
输入参数{in}		
spi_periph	外设SPIx	
SPIx	x=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 3-465. 函数 spi_i2s_interrupt_enable

函数名称	spi_i2s_interrupt_enable	
函数原形	void spi_i2s_interrupt_enable(uint32_t spi_periph, uint8_t interrupt);	
功能描述	使能外设SPIx/I2S0中断	
先决条件	-	
被调用函数	-	
输入参数{in}		
spi_periph	外设SPIx	
SPIx	x=0,1	
输入参数{in}		
interrupt	SPI/I2S中断	
SPI_I2S_INT_TBE	发送缓冲区空中断使能	
SPI_I2S_INT_RBNE	接收缓冲区非空中断使能	
SPI_I2S_INT_ERR	错误中断使能	
输出参数{out}		
-	-	

返回值	
-	-

例如：

```
/* 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 3-466. 函数 **spi_i2s_interrupt_disable**

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

例如：

```
/* disable SPI0 transmit buffer empty interrupt */
spi_i2s_interrupt_disable(SPI0, SPI_I2S_INT_TBE);
```

函数 **spi_i2s_interrupt_flag_get**

函数spi_i2s_interrupt_flag_get描述见下表：

表 3-467. 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 3-468. 函数 **spi_i2s_flag_get**

函数名称	spi_i2s_flag_get
函数原形	FlagStatus spi_i2s_flag_get(uint32_t spi_periph, uint32_t flag);
功能描述	获取外设SPIx/I2S0标志状态
先决条件	-
被调用函数	-
输入参数{in}	

spi_periph	外设SPIx
<i>SPIx</i>	x=0,1
输入参数{in}	
flag	SPI/I2S标志状态
<i>SPI_FLAG_TBE</i>	SPI发送缓冲区空标志
<i>SPI_FLAG_RBNE</i>	SPI接收缓冲区非空标志
<i>SPI_FLAG_TRANS</i>	SPI通信进行中标志
<i>SPI_FLAG_RXORE</i> <i>RR</i>	SPI接收过载错误标志
<i>SPI_FLAG_CONFE</i> <i>RR</i>	SPI配置错误标志
<i>SPI_FLAG_CRCER</i> <i>R</i>	SPI CRC错误标志
<i>SPI_FLAG_FERR</i>	SPI 格式错误标志
<i>I2S_FLAG_TBE</i>	I2S发送缓冲区空标志
<i>I2S_FLAG_RBNE</i>	I2S接收缓冲区非空标志
<i>I2S_FLAG_TRANS</i>	I2S通信进行中标志
<i>I2S_FLAG_RXORE</i> <i>RR</i>	I2S接收过载错误标志
<i>I2S_FLAG_TXURE</i> <i>RR</i>	I2S发送欠载错误标志
<i>I2S_FLAG_CH</i>	I2S通道标志
<i>I2S_FLAG_FERR</i>	I2S格式错误标志
输出参数{out}	
-	-
返回值	
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 3-469. 函数 **spi_crc_error_clear**

函数名称	spi_crc_error_clear
函数原形	void spi_crc_error_clear(uint32_t spi_periph);
功能描述	清除SPIx CRC错误标志
先决条件	-
被调用函数	-

输入参数{in}	
spi_periph	外设SPIx
SPIx	x=0,1
输出参数{out}	
-	-
返回值	
-	-

例如：

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

3.20. SYSCFG

章节[3.20.1](#)描述了SYSCFG的寄存器列表，章节[3.20.2](#)对SYSCFG库函数进行说明。

3.20.1. 外设寄存器说明

SYSCFG寄存器列表如下表所示：

表 3-470. SYSCFG 寄存器

寄存器名称	寄存器描述
SYSCFG_CFG0	配置寄存器0
SYSCFG_EXTI_S0	EXTI源选择寄存器0
SYSCFG_EXTI_S1	EXTI源选择寄存器1
SYSCFG_EXTI_S2	EXTI源选择寄存器2
SYSCFG_EXTI_S3	EXTI源选择寄存器3
SYSCFG_CFG2	系统配置寄存器2
SYSCFG_CPU_IRQ_LAT	IRQ延迟寄存器

3.20.2. 外设库函数说明

SYSCFG库函数列表如下表所示：

表 3-471. SYSCFG 库函数

库函数名称	库函数描述
syscfg_deinit	复位SYSCFG寄存器
syscfg_dma_remap_enable	使能DMA通道重映射
syscfg_dma_remap_disable	失能DMA通道重映射
syscfg_high_current_enable	使能PB9引脚大电流能力
syscfg_high_current_disable	失能PB9引脚大电流能力
syscfg_exti_line_config	配置GPIO引脚作为EXTI
syscfg_lock_config	将TIMER 0/14/15/16中断输入连接到所选参数

库函数名称	库函数描述
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 3-472. 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 3-473. 函数 syscfg_dma_remap_enable

函数名称	syscfg_dma_remap_enable
函数原形	void syscfg_dma_remap_enable(uint32_t syscfg_dma_remap);
功能描述	使能DMA通道重映射
先决条件	-
被调用函数	-
输入参数{in}	
syscfg_dma_rema p	指定要重新映射的DMA通道
SYSCFG_DMA_RE MAP_TIMER16	重新映射TIMER 16通道0和向通道1发送DMA请求
SYSCFG_DMA_RE MAP_TIMER15	重新映射TIMER 15通道5和向通道3发送DMA请求
SYSCFG_DMA_RE	将AUARTARO RX DMA请求重新映射到通道4

<i>MAP_USART0RX</i>	
<i>SYSCFG_DMA_RE</i> <i>MAP_USART0TX</i>	将UART0 TX DMA请求重新映射到通道3
<i>SYSCFG_DMA_RE</i> <i>MAP_ADC</i>	从通道0重新映射ADC DMA请求到通道1
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* enable DMA channel remap*/
syscfg_dma_remap_enable(SYSCFG_DMA_REMAP_TIMER16);
```

函数 **syscfg_dma_remap_disable**

函数**syscfg_dma_remap_disable**描述见下表：

表 3-474. 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}	
<i>syscfg_dma_rema</i> <i>p</i>	指定要重新映射的DMA通道
<i>SYSCFG_DMA_RE</i> <i>MAP_TIMER16</i>	重新映射TIMER 16通道0和向通道1发送DMA请求
<i>SYSCFG_DMA_RE</i> <i>MAP_TIMER15</i>	重新映射TIMER 15通道5和向通道3发送DMA请求
<i>SYSCFG_DMA_RE</i> <i>MAP_USART0RX</i>	将UART0 RX DMA请求重新映射到通道4
<i>SYSCFG_DMA_RE</i> <i>MAP_USART0TX</i>	将UART0 TX DMA请求重新映射到通道3
<i>SYSCFG_DMA_RE</i> <i>MAP_ADC</i>	从通道0重新映射ADC DMA请求到通道1
输出参数{out}	
-	-
返回值	
-	-

例如：

```

/* disable DMA channel remap*/

syscfg_dma_remap_disable(SYSCFG_DMA_REMAP_TIMER16);

```

函数 **syscfg_high_current_enable**

函数**syscfg_high_current_enable**描述见下表：

表 3-475. 函数 syscfg_high_current_enable

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

例如：

```

/* enable PB9 high current capability */

syscfg_high_current_enable();

```

函数 **syscfg_high_current_disable**

函数**syscfg_high_current_disable**描述见下表：

表 3-476. 函数 syscfg_high_current_disable

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

例如：

```

/* disable PB9 high current capability */

```

```
syscfg_high_current_disable();
```

函数 **syscfg_exti_line_config**

函数**syscfg_exti_line_config**描述见下表:

表 3-477. 函数 syscfg_exti_line_config

函数名称	syscfg_exti_line_config
函数原形	void syscfg_exti_line_config(uint8_t exti_port, uint8_t exti_pin);
功能描述	配置GPIO引脚作为EXTI
先决条件	-
被调用函数	-
输入参数{in}	
exti_port	指定EXTI使用的GPIO端口
EXTI_SOURCE_GP /Ox	x=A,B,C,D,F
输入参数{in}	
exti_pin	EXTI引脚
EXTI_SOURCE_PI Nx	x=0..15(GPIOA, GPIOB, GPIOC, GPIOD, GPIOF)
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* configure the GPIO pin as EXTI Line */
syscfg_exti_line_config(EXTI_SOURCE_GPIOA, EXTI_SOURCE_PIN0);
```

函数 **syscfg_lock_config**

函数**syscfg_lock_config**描述见下表:

表 3-478. 函数 syscfg_lock_config

函数名称	syscfg_lock_config
函数原形	void syscfg_lock_config(uint32_t syscfg_lock);
功能描述	将TIMER 0/14/15/16中断输入连接到所选参数
先决条件	-
被调用函数	-
输入参数{in}	
exti_port	指定EXTI使用的GPIO端口
SYSCFG_LOCK_L OCKUP	Cortex-M4锁定输出连接到断开输入
SYSCFG_LOCK_S	SRAM_PARITY校验错误连接到断开输入

<i>RAM_PARITY_ERR</i>	
OR	
<i>SYSCFG_LOCK_LV</i>	LVD中断连接到断开输入
D	
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* configure syscfg lock*/
syscfg_lock_config(SYSCFG_LOCK_LOCKUP);
```

函数 **syscfg_flag_get**

函数**syscfg_flag_get**描述见下表：

表 3-479. 函数 **syscfg_flag_get**

函数名称	syscfg_flag_get
函数原形	FlagStatus syscfg_flag_get(uint32_t syscfg_flag);
功能描述	校验SYSCFG_CFG2寄存器中指定的标志位是否置位
先决条件	-
被调用函数	-
输入参数{in}	
syscfg_flag	中断标志
<i>SYSCFG_SRAM_P_CEF</i>	SRAM奇偶校验错误标志
输出参数{out}	
-	-
返回值	
FlagStatus	SET或者RESET

例如：

```
/* get syscfg flag */
FlagStatus status;
status = syscfg_flag_get(SYSCFG_SRAM_PCEF);
```

函数 **syscfg_flag_clear**

函数**syscfg_flag_clear**描述见下表：

表 3-480. 函数 **syscfg_flag_clear**

函数名称	syscfg_flag_gclear
------	--------------------

函数原形	void syscfg_flag_clear(uint32_t syscfg_flag);
功能描述	清除SYSCFG_CFG2的标志位
先决条件	-
被调用函数	-
输入参数{in}	
syscfg_flag	中断标志
SYSCFG_SRAM_P CEF	SRAM奇偶校验错误标志
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* clear syscfg flag */
syscfg_flag_clear(SYSCFG_SRAM_PCEF);
```

函数 syscfg_compensation_config

函数syscfg_compensation_config描述见下表：

表 3-481. 函数 syscfg_compensation_config

函数名称	syscfg_compensation_config
函数原形	void syscfg_compensation_config(uint32_t syscfg_compensation);
功能描述	IO补偿单元设置
先决条件	-
被调用函数	-
输入参数{in}	
syscfg_compensation	IO补偿单元模式
SYSCFG_COMPENSATION_ENABLE	IO补偿单元打开
SYSCFG_COMPENSATION_DISABLE	IO补偿单元关闭
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* enable the I/O compensation cell */
syscfg_compensation_config (SYSCFG_COMPENSATION_ENABLE);
```

函数 syscfg_cps_rdy_flag_get

函数syscfg_cps_rdy_flag_get描述见下表:

表 3-482. 函数 syscfg_cps_rdy_flag_get

函数名称	syscfg_cps_rdy_flag_get	
函数原形	FlagStatus syscfg_cps_rdy_flag_get(void);	
功能描述	IO补偿单元是否准备就绪	
先决条件	-	
被调用函数	-	
输入参数{in}		
-	-	
输出参数{out}		
-	-	
返回值		
FlagStatus	SET或者RESET	

例如:

```
/* check if the I/O compensation cell ready flag is set or not */

FlagStatus ready_flag;

ready_flag = syscfg_cps_rdy_flag_get();
```

3.21. TIMER

定时器含有可编程的一个无符号计数器，支持输入捕获和输出比较，分为五种类型：高级定时器(TIMER0)，通用定时器L0(TIMERx, x=1, 2)，通用定时器L2(TIMER13)，通用定时器L3(TIMER14)，通用定时器L4(TIMERx, x=15, 16)，基本定时器(TIMER5)，不同类型的定时器具体功能有所差别。章节[3.21.1](#)描述了TIMER的寄存器列表，章节[3.21.2](#)对TIMER库函数进行说明。

3.21.1. 外设寄存器说明

TIMER寄存器列表如下表所示:

表 3-483. TIMER 寄存器

寄存器名称	寄存器描述
TIMERx_CTL0(TIMERx, x=0, 1, 2, 5, 13, 14, 15, 16)	控制寄存器 0
TIMERx_CTL1(TIMERx, x=0, 1, 2, 5, 13, 14, 15, 16)	控制寄存器 1
TIMERx_SMCFG(TIMERx, x=0, 1, 2, 14)	从模式配置寄存器
TIMERx_DMAINTEN(TIMERx, x=0, 1, 2, 5, 13, 14, 15, 16)	DMA 和中断使能寄存器
TIMERx_INTF(TIMERx, x=0, 1, 2, 5, 13, 14, 15, 16)	中断标志寄存器

寄存器名称	寄存器描述
TIMERx_SWEVG(TIMERx, x=0, 1, 2, 5, 13, 14, 15, 16)	软件事件产生寄存器
TIMERx_CHCTL0(TIMERx, x=0, 1, 2, 13, 14, 15, 16)	通道控制寄存器 0
TIMERx_CHCTL1(TIMERx, x=0,1,2)	通道控制寄存器 1
TIMERx_CHCTL2(TIMERx, x=0, 1, 2, 13, 14, 15, 16)	通道控制寄存器 2
TIMERx_CNT(TIMERx, x=0, 1, 2, 5, 13, 14, 15, 16)	计数器寄存器
TIMERx_PSC(TIMERx, x=0, 1, 2, 5, 13, 14, 15, 16)	预分频寄存器
TIMERx_CAR(TIMERx, x=0, 1, 2, 5, 13, 14, 15, 16)	计数器自动重载寄存器
TIMERx_CREP(TIMERx, x=0, 5, 14, 15, 16)	重复计数寄存器
TIMERx_CH0CV(TIMERx, x=0, 1, 2, 13, 14, 15, 16)	通道 0 捕获/比较寄存器
TIMERx_CH1CV(TIMERx, x=0, 1, 2, 14)	通道 1 捕获/比较寄存器
TIMERx_CH2CV(TIMERx, x=0, 1, 2)	通道 2 捕获/比较寄存器
TIMERx_CH3CV(TIMERx, x=0, 1, 2)	通道 3 捕获/比较寄存器
TIMERx_IRMP(TIMERx, x=13)	通道输入重映射寄存器
TIMERx_CCHP(TIMERx, x=0, 1, 2, 14, 15, 16)	互补通道保护寄存器
TIMERx_DMACFG(TIMERx, x=0, 1, 2, 14, 15, 16)	DMA 配置寄存器
TIMERx_DMATB(TIMERx, x=0, 1, 2, 14, 15, 16)	DMA 发送缓冲区寄存器
TIMERx_CFG(TIMERx, x=0, 1, 2, 13, 14, 15, 16)	配置寄存器

3.21.2. 外设库函数说明

TIMER库函数列表如下表所示：

表 3-484. 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

库函数名称	库函数描述
timer_single_pulse_mode_config	配置外设 TIMERx 的单脉冲模式
timer_update_source_config	配置外设 TIMERx 的更新源
timer_ocpre_clear_source_config	配置 TIMERx 的 OCPRE 清除源选择
timer_flag_get	外设 TIMERx 标志位获取
timer_flag_clear	外设 TIMERx 标志位清除
timer_interrupt_enable	外设 TIMERx 的中断使能
timer_interrupt_disable	外设 TIMERx 的中断禁能
timer_interrupt_flag_get	外设 TIMERx 中断标志获取
timer_interrupt_flag_clear	外设 TIMERx 中断标志清除
timer_dma_enable	使能 TIMERx 的 DMA 功能
timer_dma_disable	禁能 TIMERx 的 DMA 功能
timer_channel_dma_request_source_select	外设 TIMERx 的通道 DMA 请求源选择
timer_dma_transfer_config	配置外设 TIMERx 的 DMA 模式
timer_event_software_generate	软件产生事件
timer_break_struct_para_init	将 TIMER 中止功能参数结构体中所有参数初始化为默认值
timer_break_config	配置中止功能
timer_break_enable	使能 TIMERx 的中止功能
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	配置互补通道输出状态

库函数名称	库函数描述
_state_config	
timer_channel_input_struct _para_init	将 TIMER 通道输入参数结构体中所有参数初始化为默认值
timer_input_capture_config	配置 TIMERx 输入捕获参数
timer_channel_input_capture_prescaler_config	配置 TIMERx 通道输入捕获预分频值
timer_channel_capture_value_register_read	读取通道输入捕获值
timer_input_pwm_capture_config	配置 TIMERx 捕获 PWM 输入参数
timer_hall_mode_config	配置 TIMERx 的 HALL 接口功能
timer_input_trigger_source_select	TIMERx 的输入触发源选择
timer_master_output_trigger_source_select	选择 TIMERx 主模式输出触发源
timer_slave_mode_select	TIMERx 从模式配置
timer_master_slave_mode_config	TIMERx 主从模式配置
timer_external_trigger_config	配置 TIMERx 外部触发输入
timer_quadrature_decoder_mode_config	TIMERx 配置为编码器模式
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 3-485. 结构体类型 timer_parameter_struct

成员名称	功能描述
prescaler	预分频值 (0~65535)
alignedmode	对齐模式 (TIMER_COUNTER_EDGE, TIMER_COUNTER_CENTER_DOWN, TIMER_COUNTER_CENTER_UP, TIMER_COUNTER_CENTER_BOTH)
counterdirection	计数方向 (TIMER_COUNTER_UP, TIMER_COUNTER_DOWN)
period	周期 (0~65535)
clockdivision	时钟分频因子 (TIMER_CKDIV_DIV1, TIMER_CKDIV_DIV2, TIMER_CKDIV_DIV4)
repetitioncounter	重复计数器值 (0~255)

结构体 timer_break_parameter_struct

表 3-486. 结构体类型 timer_break_parameter_struct

成员名称	功能描述
runoffstate	运行模式下“关闭状态”配置 (TIMER_ROS_STATE_ENABLE, TIMER_ROS_STATE_DISABLE)
ideloffstate	空闲模式下“关闭状态”配置 (TIMER_IOS_STATE_ENABLE, TIMER_IOS_STATE_DISABLE)
deadtime	死区时间 (0~255)
breakpolarity	中止信号极性 (TIMER_BREAK_POLARITY_LOW, TIMER_BREAK_POLARITY_HIGH)
outputautostate	自动输出使能 (TIMER_OUTAUTO_ENABLE, TIMER_OUTAUTO_DISABLE)
protectmode	互补寄存器保护控制 (TIMER_CCHP_PROT_OFF, TIMER_CCHP_PROT_0, TIMER_CCHP_PROT_1, TIMER_CCHP_PROT_2)
breakstate	中止使能 (TIMER_BREAK_ENABLE, TIMER_BREAK_DISABLE)

结构体 timer_oc_parameter_struct

表 3-487. 结构体类型 timer_oc_parameter_struct

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

结构体 timer_ic_parameter_struct

表 3-488. 结构体类型 timer_ic_parameter_struct

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

函数 timer_deinit

函数timer_deinit描述见下表:

表 3-489. 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 3-490. 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 表 3-485. 结构体类型 timer_parameter_struct
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 3-491. 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 表 3-485. 结构体类型 timer_parameter_struct
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 3-492. 函数 timer_enable

函数名称	timer_enable
------	--------------

函数原型	void timer_enable(uint32_t timer_periph);
功能描述	使能外设 TIMERx
先决条件	-
被调用函数	-
输入参数{in}	
timer_periph	TIMER 外设
<i>TIMERx(x=0..2, 5, 13..16)</i>	TIMER 外设选择
输出参数{out}	
-	-
返回值	
-	-

例如：

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

函数 **timer_disable**

函数timer_disable描述见下表：

表 3-493. 函数 **timer_disable**

函数名称	timer_disable
函数原型	void timer_disable(uint32_t timer_periph);
功能描述	禁能外设 TIMERx
先决条件	-
被调用函数	-
输入参数{in}	
timer_periph	TIMER 外设
<i>TIMERx(x=0..2, 5, 13..16)</i>	TIMER 外设选择
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* disable TIMER0 */
timer_disable (TIMER0);
```

函数 **timer_auto_reload_shadow_enable**

函数timer_auto_reload_shadow_enable描述见下表：

表 3-494. 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 3-495. 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 3-496. 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 3-497. 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 3-498. 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 3-499. 函数 timer_counter_up_direction

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

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

例如：

```
/* set TIMER0 counter up direction */
```

```
timer_counter_up_direction (TIMER0);
```

函数 timer_counter_down_direction

函数timer_counter_down_direction描述见下表：

表 3-500. 函数 timer_counter_down_direction

函数名称	timer_counter_down_direction
函数原型	void timer_counter_down_direction(uint32_t timer_periph);
功能描述	设置外设 TIMERx 向下计数
先决条件	计数器设置为无中央对齐计数模式（边沿对齐模式）
被调用函数	-
输入参数{in}	
timer_periph	TIMER 外设
TIMERx(x=0..2)	TIMER 外设选择
输出参数{out}	
-	-
返回值	
-	-

例如：

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

```
timer_counter_down_direction (TIMER0);
```

函数 timer_prescaler_config

函数timer_prescaler_config描述见下表：

表 3-501. 函数 timer_prescaler_config

函数名称	timer_prescaler_config
函数原型	void timer_prescaler_config(uint32_t timer_periph, uint16_t prescaler, uint8_t

	pscreload);
功能描述	配置外设 TIMERx 预分频器
先决条件	-
被调用函数	-
输入参数{in}	
timer_periph	TIMER 外设
<i>TIMERx(x=0..2, 5, 13..16)</i>	TIMER 外设选择
输入参数{in}	
prescaler	预分频值, 0~65535
输入参数{in}	
pscreload	预分频值加载模式
<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
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* configure TIMER0 prescaler */
timer_prescaler_config(TIMER0, 3000, TIMER_PSC_RELOAD_NOW);
```

函数 **timer_repetition_value_config**

函数**timer_repetition_value_config**描述见下表：

表 3-502. 函数 **timer_repetition_value_config**

函数名称	timer_repetition_value_config
函数原型	void timer_repetition_value_config(uint32_t timer_periph, uint16_t repetition);
功能描述	配置外设 TIMERx 的重复计数器
先决条件	-
被调用函数	-
输入参数{in}	
timer_periph	TIMER 外设
<i>TIMERx(x=0, 15, 16)</i>	TIMER 外设选择
输入参数{in}	
repetition	重复计数器值, 取值范围 0~255
输出参数{out}	
-	-
返回值	

-	-
---	---

例如：

```
/* configure TIMER0 repetition register value */
timer_repetition_value_config (TIMER0, 98);
```

函数 timer_autoreload_value_config

函数timer_autoreload_value_config描述见下表：

表 3-503. 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
<i>TIMERx(x=0..2, 5, 13..16)</i>	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 3-504. 函数 timer_counter_value_config

函数名称	timer_counter_value_config
函数原型	void timer_counter_value_config(uint32_t timer_periph, uint16_t counter);
功能描述	配置外设 TIMERx 的计数器值
先决条件	-
被调用函数	-
输入参数{in}	
timer_periph	TIMER 外设
<i>TIMERx(x=0..2, 5,</i>	TIMER 外设选择

13..16)	
输入参数{in}	
counter	计数器值 (0-65535)
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* configure TIMER0 counter register value */

timer_counter_value_config (TIMER0, 3000);
```

函数 **timer_counter_read**

函数timer_counter_read描述见下表：

表 3-505. 函数 timer_counter_read

函数名称	timer_counter_read
函数原型	uint32_t timer_counter_read(uint32_t timer_periph);
功能描述	读取外设 TIMERx 的计数器值
先决条件	-
被调用函数	-
输入参数{in}	
timer_periph	TIMER 外设
TIMERx(x=0..2, 5, 13..16)	TIMER 外设选择
输出参数{out}	
-	-
返回值	
uint32_t	外设 TIMERx 的计数器值 (0~65535)

例如：

```
/* read TIMER0 counter value */

uint32_t i = 0;

i = timer_counter_read (TIMER0);
```

函数 **timer_prescaler_read**

函数timer_prescaler_read描述见下表：

表 3-506. 函数 timer_prescaler_read

函数名称	timer_prescaler_read
函数原型	uint16_t timer_prescaler_read(uint32_t timer_periph);

功能描述	读取外设 TIMERx 的预分频器值
先决条件	-
被调用函数	-
输入参数{in}	
timer_periph	TIMER 外设
<i>TIMERx(x=0..2, 5, 13..16)</i>	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 3-507. 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 spemode);
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
<i>TIMERx(x=0..2, 5, 14..16)</i>	TIMER 外设选择
输入参数{in}	
spemode	脉冲模式
<i>TIMER_SP_MODE_SINGLE</i>	单脉冲模式计数
<i>TIMER_SP_MODE_REPETITIVE</i>	重复模式计数
输出参数{out}	
-	-
返回值	
-	-

例如：

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

timer_single_pulse_mode_config (TIMER0, TIMER_SP_MODE_SINGLE);
```

函数 timer_update_source_config

函数timer_update_source_config描述见下表:

表 3-508. 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 3-509. 函数 timer_ocpre_clear_source_config

函数名称	timer_ocpre_clear_source_config
函数原型	void timer_ocpre_clear_source_config (uint32_t timer_periph, uint8_t ocpreclear);
功能描述	配置外设 TIMERx 的 OCPRE 清除源选择
先决条件	-

被调用函数	-
输入参数{in}	
timer_periph	TIMER 外设
TIMERx(x=0..2)	TIMER 外设选择
输入参数{in}	
ocpreclear	清除源
TIMER_OCPRE_CL_R	OCPRE_CLR_INT 连接到 OCPRE_CLR 输入
TIMER_OCPRE_CL_F	OCPRE_CLR_INT 连接到 ETIF
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* configure TIMER0 OCPRE_CLR_INT is connected to the OCPRE_CLR input */
timer_ocpre_clear_source_config(TIMER0, TIMER_OCPRE_CLEAR_SOURCE_CLR);
```

函数 timer_interrupt_enable

函数timer_interrupt_enable描述见下表：

表 3-510. 函数 timer_interrupt_enable

函数名称	timer_interrupt_enable
函数原型	void timer_interrupt_enable(uint32_t timer_periph, uint32_t interrupt);
功能描述	外设 TIMERx 中断使能
先决条件	-
被调用函数	-
输入参数{in}	
timer_periph	TIMER 外设
TIMERx	参考具体参数
输入参数{in}	
interrupt	中断源
TIMER_INT_UP	更新中断, TIMERx (x=0..2, 5, 13..16)
TIMER_INT_CH0	通道0比较/捕获中断, TIMERx(x=0..2, 13..16)
TIMER_INT_CH1	通道 1 比较/捕获中断, TIMERx(x=0..2, 14)
TIMER_INT_CH2	通道 2 比较/捕获中断, TIMERx(x=0..2)
TIMER_INT_CH3	通道 3 比较/捕获中断, TIMERx(x=0..2)
TIMER_INT_CMT	换相更新中断, TIMERx (x=0, 14..16)
TIMER_INT_TRG	触发中断, TIMERx(x=0..2, 14)

<i>TIMER_INT_BRK</i>	中止中断, TIMERx(x=0, 14..16)
输出参数{out}	
-	-
返回值	
-	-

例如:

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

timer_interrupt_enable (TIMER0, TIMER_INT_UP);
```

函数 **timer_interrupt_disable**

函数**timer_interrupt_disable**描述见下表:

表 3-511. 函数 **timer_interrupt_disable**

函数名称	timer_interrupt_disable
函数原型	void timer_interrupt_disable (uint32_t timer_periph, uint32_t interrupt);
功能描述	外设 TIMERx 中断禁能
先决条件	-
被调用函数	-
输入参数{in}	
<i>timer_periph</i>	TIMER 外设
<i>TIMERx</i>	参考具体参数
输入参数{in}	
<i>interrupt</i>	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)
输出参数{out}	
-	-
返回值	
-	-

例如:

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

timer_interrupt_disable (TIMER0, TIMER_INT_UP);
```

函数 timer_interrupt_flag_get

函数timer_interrupt_flag_get描述见下表:

表 3-512. 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 3-513. 函数 timer_interrupt_flag_clear

函数名称	timer_interrupt_flag_clear
函数原型	void timer_interrupt_flag_clear(uint32_t timer_periph, uint32_t interrupt);
功能描述	清除外设 TIMERx 的中断标志
先决条件	-
被调用函数	-
输入参数{in}	
timer_periph	TIMER 外设
TIMERx	参考具体参数
输入参数{in}	
interrupt	中断源
TIMER_INT_FLAG_UP	更新中断, 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	
-	-

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 3-514. 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 3-515. 函数 timer_flag_clear

函数名称	timer_flag_clear
------	------------------

函数原型	void timer_flag_clear(uint32_t timer_periph, uint32_t flag);
功能描述	清除外设 TIMERx 状态标志
先决条件	-
被调用函数	-
输入参数{in}	
timer_periph	TIMER 外设
TIMERx	参考具体参数
输入参数{in}	
flag	状态标志
TIMER_FLAG_UP	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)
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* clear TIMER0 update flags */

timer_flag_clear(TIMER0, TIMER_FLAG_UP);
```

函数 timer_dma_enable

函数timer_dma_enable描述见下表：

表 3-516. 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 3-517. 函数 **timer_dma_disable**

函数名称	timer_dma_disable
函数原型	void timer_dma_disable (uint32_t timer_periph, uint16_t dma);
功能描述	外设 TIMERx 的 DMA 禁能
先决条件	-
被调用函数	-
输入参数{in}	
timer_periph	TIMER 外设
TIMERx	参考具体参数
输入参数{in}	
dma	DMA 源

<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 3-518. 函数 timer_channel_dma_request_source_select

函数名称	timer_channel_dma_request_source_select
函数原型	void timer_channel_dma_request_source_select(uint32_t timer_periph, uint32_t dma_request);
功能描述	外设 TIMERx 的通道 DMA 请求源选择
先决条件	-
被调用函数	-
输入参数{in}	
timer_periph	TIMER 外设
<i>TIMERx(x=0..2, 14..16)</i>	TIMER 外设选择
输入参数{in}	
dma_request	通道的 DMA 请求源选择
<i>TIMER_DMAREQU EST_CHANNELEV ENT</i>	当通道捕获/比较事件发生时, 发送通道 n 的 DMA 请求
<i>TIMER_DMAREQU</i>	当更新事件发生, 发送通道 n 的 DMA 请求

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

例如：

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

timer_channel_dma_request_source_select(TIMER0,
  TIMER_DMAREQUEST_CHANNELEVENT);
```

函数 timer_dma_transfer_config

函数timer_dma_transfer_config描述见下表：

表 3-519. 函数 timer_dma_transfer_config

函数名称	timer_dma_transfer_config
函数原型	void timer_dma_transfer_config(uint32_t timer_periph, uint32_t dma_baseaddr, uint32_t dma_lenth);
功能描述	配置外设 TIMERx 的 DMA 模式
先决条件	-
被调用函数	-
输入参数{in}	
timer_periph	TIMER 外设
<i>TIMERx(x=0..2, 14..16)</i>	定时器外设选择
输入参数{in}	
dma_baseaddr	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 N</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_</i>	DMA 传输起始地址: TIMER_DMACFG_DMATA_CHCTL0, TIMERx(x=0..2,

<i>DMATA_CHCTL0</i>	14..16)
<i>TIMER_DMACFG_DMATA_CHCTL1</i>	DMA 传输起始地址: <i>TIMER_DMACFG_DMATA_CHCTL1</i> , <i>TIMERx(x=0..2)</i>
<i>TIMER_DMACFG_DMATA_CHCTL2</i>	DMA 传输起始地址: <i>TIMER_DMACFG_DMATA_CHCTL2</i> , <i>TIMERx (x=0..2, 14..16)</i>
<i>TIMER_DMACFG_DMATA_CNT</i>	DMA 传输起始地址: <i>TIMER_DMACFG_DMATA_CNT</i> , <i>TIMERx (x=0..2, 14..16)</i>
<i>TIMER_DMACFG_DMATA_PSC</i>	DMA 传输起始地址: <i>TIMER_DMACFG_DMATA_PSC</i> , <i>TIMERx (x=0..2, 14..16)</i>
<i>TIMER_DMACFG_DMATA_CAR</i>	DMA 传输起始地址: <i>TIMER_DMACFG_DMATA_CAR</i> , <i>TIMERx (x=0..2, 14..16)</i>
<i>TIMER_DMACFG_DMATA_CREP</i>	DMA 传输起始地址: <i>TIMER_DMACFG_DMATA_CREP</i> , <i>TIMERx (x=0, 14..16)</i>
<i>TIMER_DMACFG_DMATA_CH0CV</i>	DMA 传输起始地址: <i>TIMER_DMACFG_DMATA_CH0CV</i> , <i>TIMERx (x=0..2, 14..16)</i>
<i>TIMER_DMACFG_DMATA_CH1CV</i>	DMA 传输起始地址: <i>TIMER_DMACFG_DMATA_CH1CV</i> , <i>TIMERx(x=0..2, 14)</i>
<i>TIMER_DMACFG_DMATA_CH2CV</i>	DMA 传输起始地址: <i>TIMER_DMACFG_DMATA_CH2CV</i> , <i>TIMERx(x=0..2)</i>
<i>TIMER_DMACFG_DMATA_CH3CV</i>	DMA 传输起始地址: <i>TIMER_DMACFG_DMATA_CH3CV</i> , <i>TIMERx(x=0..2)</i>
<i>TIMER_DMACFG_DMATA_CCHP</i>	DMA 传输起始地址: <i>TIMER_DMACFG_DMATA_CCHP</i> , <i>TIMERx (x=0, 14..16)</i>
<i>TIMER_DMACFG_DMATA_DMACFG</i>	DMA 传输起始地址: <i>TIMER_DMACFG_DMATA_DMACFG</i> , <i>TIMERx (x=0..2, 14..16)</i>
输入参数{in}	
<i>dma_lenth</i>	DMA传输长度
<i>TIMER_DMACFG_DMATC_xTRANSFER</i>	x=1..18, DMA传输 x 次
输出参数{out}	
-	-
返回值	
-	-

例如:

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

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

函数 **timer_event_software_generate**

函数**timer_event_software_generate**描述见下表:

表 3-520. 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 3-521. 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 表 3-486. 结构体类型 timer_break_parameter_struct
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 3-522. 函数 `timer_break_config`

函数名称	<code>timer_break_config</code>
函数原型	<code>void timer_break_config(uint32_t timer_periph, timer_break_parameter_struct* breakpara);</code>
功能描述	配置中止功能
先决条件	-
被调用函数	-
输入参数{in}	
<code>timer_periph</code>	TIMER 外设
<code>TIMERx(x=0, 14..16)</code>	TIMER 外设选择
输入参数{in}	
breakpara	中止功能配置结构体, 详见 表 3-486. 结构体类型 timer_break_parameter_struct
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 3-523. 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 3-524. 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 3-525. 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 3-526. 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 <code>TIMERx_CCHP</code> 寄存器的 <code>PROT [1:0] = 00</code> 时，才可修改
被调用函数	-
输入参数{in}	
<code>timer_periph</code>	TIMER 外设
<code>TIMERx(x=0, 14..16)</code>	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 3-527. 函数 `timer_primary_output_config`

函数名称	timer_primary_output_config
函数原型	<code>void timer_primary_output_config(uint32_t timer_periph, ControlStatus newvalue);</code>
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}	
<code>timer_periph</code>	TIMER 外设
<code>TIMERx(x=0,14..16)</code>	TIMER 外设选择
输入参数{in}	
<code>newvalue</code>	控制状态
<code>ENABLE</code>	使能
<code>DISABLE</code>	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 3-528. 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 3-529. 函数 timer_channel_control_shadow_update_config

函数名称	timer_channel_control_shadow_update_config
函数原型	void timer_channel_control_shadow_update_config(uint32_t timer_periph, uint8_t ccuctl);
功能描述	通道换相控制影子寄存器更新控制
先决条件	-
被调用函数	-
输入参数{in}	
timer_periph	TIMER 外设
TIMERx(x=0, 14..16)	TIMER 外设选择
输入参数{in}	

ccuctl	通道换相控制影子寄存器更新控制
<i>TIMER_UPDATECTL_CCU</i>	CMTG 位被置 1 时更新影子寄存器
<i>TIMER_UPDATECTL_CCUTRI</i>	当 CMTG 位被置 1 或检测到 TRIGI 上升沿时，影子寄存器更新
输出参数{out}	
-	-
返回值	
-	-

例如：

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

函数 **timer_channel_output_struct_para_init**

函数timer_channel_output_struct_para_init描述见下表：

表 3-530. 函数 **timer_channel_output_struct_para_init**

函数名称	timer_channel_output_struct_para_init
函数原型	void timer_channel_output_struct_para_init(timer_oc_parameter_struct* ocpara);
功能描述	将 TIMER 通道输出参数结构体中所有参数初始化为默认值
先决条件	-
被调用函数	-
输入参数{in}	
ocpara	输出通道结构体，详见 表 3-487. 结构体类型 timer_oc_parameter_struct 。
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* initialize TIMER channel output parameter struct with a default value */

timer_oc_parameter_struct timer_ocinitpara;

timer_channel_output_struct_para_init(timer_ocinitpara);
```

函数 **timer_channel_output_config**

函数timer_channel_output_config描述见下表：

表 3-531. 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 表 3-487. 结构体类型 timer_oc_parameter_struct
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 3-532. 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 3-533. 函数 timer_channel_output_pulse_value_config

函数名称	timer_channel_output_pulse_value_config
函数原型	void timer_channel_output_pulse_value_config(uint32_t timer_periph, uint16_t channel, uint32_t pulse);
功能描述	配置外设 TIMERx 的通道输出比较值
先决条件	-
被调用函数	-
输入参数{in}	
timer_periph	TIMER 外设
TIMERx	参考具体参数
输入参数{in}	
channel	待配置通道
TIMER_CH_0	通道 0, TIMERx (x=0..2, 13..16)
TIMER_CH_1	通道 1, TIMERx (x=0..2, 14)
TIMER_CH_2	通道 2, TIMERx (x=0..2)
TIMER_CH_3	通道 3, TIMERx (x=0..2)
输入参数{in}	
pulse	通道输出比较值 (0~65535)
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* configure TIMER0 channel 0 output pulse value */
timer_channel_output_pulse_value_config(TIMER0, TIMER_CH_0, 399);
```

函数 timer_channel_output_shadow_config

函数timer_channel_output_shadow_config描述见下表：

表 3-534. 函数 timer_channel_output_shadow_config

函数名称	timer_channel_output_shadow_config
函数原型	void timer_channel_output_shadow_config(uint32_t timer_periph, uint16_t channel, uint16_t ocshadow);
功能描述	配置 TIMERx 通道输出比较影子寄存器功能
先决条件	-
被调用函数	-
输入参数{in}	
timer_periph	TIMER 外设
TIMERx	参考具体参数
输入参数{in}	
channel	待配置通道

<i>TIMER_CH_0</i>	通道 0, TIMERx (x=0..2, 13..16)
<i>TIMER_CH_1</i>	通道 1, TIMERx (x=0..2, 14)
<i>TIMER_CH_2</i>	通道 2, TIMERx (x=0..2)
<i>TIMER_CH_3</i>	通道 3, TIMERx (x=0..2)
输入参数{in}	
ocshadow	输出比较影子寄存器功能状态
<i>TIMER_OC_SHAD_OW_ENABLE</i>	使能输出比较影子寄存器
<i>TIMER_OC_SHAD_OW_DISABLE</i>	禁能输出比较影子寄存器
输出参数{out}	
-	-
返回值	
-	-

例如：

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

timer_channel_output_shadow_config (TIMER0, TIMER_CH_0,
TIMER_OC_SHADOW_ENABLE);
```

函数 **timer_channel_output_fast_config**

函数**timer_channel_output_fast_config**描述见下表：

表 3-535. 函数 timer_channel_output_fast_config

函数名称	timer_channel_output_fast_config
函数原型	void timer_channel_output_fast_config(uint32_t timer_periph, uint16_t channel, uint16_t octfast);
功能描述	配置 TIMERx 通道输出比较快速功能
先决条件	-
被调用函数	-
输入参数{in}	
timer_periph	TIMER 外设
<i>TIMERx</i>	参考具体参数
输入参数{in}	
channel	待配置通道
<i>TIMER_CH_0</i>	通道 0, TIMERx (x=0..2, 13..16)
<i>TIMER_CH_1</i>	通道 1, TIMERx (x=0..2, 14)
<i>TIMER_CH_2</i>	通道 2, TIMERx (x=0..2)
<i>TIMER_CH_3</i>	通道 3, TIMERx (x=0..2)
输入参数{in}	
octfast	通道输出比较快速功能状态
<i>TIMER_OC_FAST_</i>	通道输出比较快速功能使能

<i>ENABLE</i>	
<i>TIMER_OC_FAST_DISABLE</i>	通道输出比较快速功能禁能
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* configure 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 3-536. 函数 timer_channel_output_clear_config

函数名称	timer_channel_output_clear_config
函数原型	void timer_channel_output_clear_config(uint32_t timer_periph, uint16_t channel, uint16_t occlear);
功能描述	配置 TIMERx 的通道输出比较清 0 功能
先决条件	-
被调用函数	-
输入参数{in}	
<i>timer_periph</i>	TIMER 外设
<i>TIMERx</i>	参考具体参数
输入参数{in}	
<i>channel</i>	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)
输入参数{in}	
<i>occlear</i>	通道比较输出清 0 功能状态
<i>TIMER_OC_CLEAR_ENABLE</i>	通道比较输出清 0 功能使能
<i>TIMER_OC_CLEAR_DISABLE</i>	通道比较输出清 0 功能禁能
输出参数{out}	
-	-
返回值	
-	-

例如：

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

```
timer_channel_output_clear_config (TIMER0, TIMER_CH_0,
    TIMER_OC_CLEAR_ENABLE);
```

函数 timer_channel_output_polarity_config

函数timer_channel_output_polarity_config描述见下表：

表 3-537. 函数 timer_channel_output_polarity_config

函数名称	timer_channel_output_polarity_config
函数原型	void timer_channel_output_polarity_config(uint32_t timer_periph, uint16_t channel, uint16_t ocpolarity);
功能描述	通道输出极性配置
先决条件	-
被调用函数	-
输入参数{in}	
timer_periph	TIMER 外设
TIMERx	参考具体参数
输入参数{in}	
channel	待配置通道
TIMER_CH_0	通道 0, TIMERx (x=0..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}	
ocpolarity	通道输出极性
TIMER_OC_POLARITY_HIGH	通道输出极性高电平有效
TIMER_OC_POLARITY_LOW	通道输出极性低电平有效
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* configure TIMER0 channel 0 output polarity */
timer_channel_output_polarity_config (TIMER0, TIMER_CH_0,
    TIMER_OC_POLARITY_HIGH);
```

函数 timer_channel_complementary_output_polarity_config

函数timer_channel_complementary_output_polarity_config描述见下表：

表 3-538. 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 3-539. 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 <i>LE</i>	通道使能
TIMER_CCX_DISA <i>BLE</i>	通道禁能
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* configure TIMER0 channel 0 enable state */
timer_channel_output_state_config(TIMER0, TIMER_CH_0, TIMER_CCX_ENABLE);
```

函数 **timer_channel_complementary_output_state_config**

函数**timer_channel_complementary_output_state_config**描述见下表：

表 3-540. 函数 **timer_channel_complementary_output_state_config**

函数名称	timer_channel_complementary_output_state_config
函数原型	void timer_channel_complementary_output_state_config(uint32_t timer_periph, uint16_t channel, uint16_t ocnstate);
功能描述	配置互补通道输出状态
先决条件	-
被调用函数	-
输入参数{in}	
timer_periph	TIMER 外设
TIMERx	参考具体参数
输入参数{in}	
channel	待配置通道
TIMER_CH_0	通道 0, TIMERx (x=0, 14..16)
TIMER_CH_1	通道 1, TIMERx (x=0)
TIMER_CH_2	通道 2, TIMERx (x=0)

输入参数{in}	
state	互补通道状态
<i>TIMER_CCXN_ENA BLE</i>	互补通道使能
<i>TIMER_CCXN_DIS ABLE</i>	互补通道禁能
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* configure TIMER0 channel 0 complementary output enable state */

timer_channel_complementary_output_state_config (TIMER0, TIMER_CH_0,
TICKER_CCXN_ENABLE);
```

函数 **timer_channel_input_struct_para_init**

函数**timer_channel_input_struct_para_init**描述见下表：

表 3-541. 函数 **timer_channel_input_struct_para_init**

函数名称	timer_channel_input_struct_para_init
函数原型	void timer_channel_input_struct_para_init(timer_ic_parameter_struct* icpara);
功能描述	将 TIMER 通道输入参数结构体中所有参数初始化为默认值
先决条件	-
被调用函数	-
输入参数{in}	
icpara	通道输入结构体，详见 表 3-488. 结构体类型 timer_ic_parameter_struct 。
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* initialize TIMER channel input parameter struct with a default value */

timer_ic_parameter_struct timer_icinitpara;

timer_channel_input_struct_para_init(&timer_icinitpara);
```

函数 **timer_input_capture_config**

函数**timer_input_capture_config**描述见下表：

表 3-542. 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 表 3-488. 结构体类型 timer_ic_parameter_struct 。
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 3-543. 函数 timer_channel_input_capture_prescaler_config

函数名称	timer_channel_input_capture_prescaler_config
函数原型	void timer_channel_input_capture_prescaler_config(uint32_t timer_periph, uint16_t channel, uint16_t prescaler);
功能描述	配置 TIMERx 通道输入捕获预分频值

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

例如：

```
/* configure TIMER0 channel 0 input capture prescaler value */

timer_channel_input_capture_prescaler_config(TIMER0, TIMER_CH_0,
TIMER_IC_PSC_DIV2);
```

函数 **timer_channel_capture_value_register_read**

函数**timer_channel_capture_value_register_read**描述见下表：

表 3-544. 函数 **timer_channel_capture_value_register_read**

函数名称	timer_channel_capture_value_register_read
函数原型	uint32_t timer_channel_capture_value_register_read(uint32_t timer_periph, uint16_t channel);
功能描述	读取通道捕获值
先决条件	-
被调用函数	-
输入参数{in}	

timer_periph	TIMER 外设
<i>TIMERx</i>	参考具体参数
输入参数{in}	
channel	待配置通道
<i>TIMER_CH_0</i>	通道 0, TIMERx (x=0..2, 13..16)
<i>TIMER_CH_1</i>	通道 1, TIMERx (x=0..2, 14)
<i>TIMER_CH_2</i>	通道 2, TIMERx (x=0..2)
<i>TIMER_CH_3</i>	通道 3, TIMERx (x=0..2)
输出参数{out}	
-	-
返回值	
uint32_t	通道输入捕获值, (0~65535)

例如:

```
/* read TIMER0 channel 0 capture compare register value */

uint32_t ch0_value = 0;

ch0_value = timer_channel_capture_value_register_read (TIMER0, TIMER_CH_0);
```

函数 **timer_input_pwm_capture_config**

函数**timer_input_pwm_capture_config**描述见下表:

表 3-545. 函数 **timer_input_pwm_capture_config**

函数名称	timer_input_pwm_capture_config
函数原型	void timer_input_pwm_capture_config(uint32_t timer_periph, uint16_t channel, timer_ic_parameter_struct* icpwm);
功能描述	配置 TIMERx 捕获 PWM 输入参数
先决条件	-
被调用函数	timer_channel_input_capture_prescaler_config
输入参数{in}	
timer_periph	TIMER 外设
<i>TIMERx(x=0..2, 14)</i>	TIMER 外设选择
输入参数{in}	
channel	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}	
icpwm	输入捕获结构体, 详见 表 3-488. 结构体类型 timer_ic_parameter_struct
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 3-546. 函数 timer_hall_mode_config

函数名称	timer_hall_mode_config
函数原型	void timer_hall_mode_config(uint32_t timer_periph, uint8_t hallmode);
功能描述	配置 TIMERx 的 HALL 接口功能
先决条件	-
被调用函数	-
输入参数{in}	
timer_periph	TIMER 外设
TIMERx(x=0..2)	TIMER 外设选择
输入参数{in}	
hallmode	HALL 接口功能状态
TIMER_HALLINTERRUPT_ENABLE	HALL 接口使能
TIMER_HALLINTERRUPT_DISABLE	HALL 接口禁能
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* configure TIMER0 hall sensor mode */
timer_hall_mode_config(TIMER0, TIMER_HALLINTERFACE_ENABLE);
```

函数 **timer_input_trigger_source_select**

函数timer_input_trigger_source_select描述见下表：

表 3-547. 函数 timer_input_trigger_source_select

函数名称	timer_input_trigger_source_select
函数原型	void timer_input_trigger_source_select(uint32_t timer_periph, uint32_t intrigger);
功能描述	TIMERx 的输入触发源选择
先决条件	SMC[2:0] = 000
被调用函数	-
输入参数{in}	
timer_periph	TIMER 外设
TIMERx(x=0, 2, 14)	TIMER 外设选择
输入参数{in}	
intrigger	待选择的触发源
TIMER_SMCFG_T RGSEL_ITI0	内部触发输入 0(ITI0, TIMERx(x=0..2, 14))
TIMER_SMCFG_T RGSEL_ITI1	内部触发输入 1(ITI1, TIMERx(x=0..2, 14))
TIMER_SMCFG_T RGSEL_ITI2	内部触发输入 2(ITI2, TIMERx(x=0..2))
TIMER_SMCFG_T RGSEL_ITI3	内部触发输入 3(ITI3, TIMERx(x=0..2, 14))
TIMER_SMCFG_T RGSEL_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_ITI0);
```

函数 timer_master_output_trigger_source_select

函数timer_master_output_trigger_source_select描述见下表：

表 3-548. 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
<i>TIMERx(x=0..2, 5, 14)</i>	TIMER 外设选择
输入参数{in}	
outrigger	主模式输出触发
<i>TIMER_TRI_OUT_SRC_RESET</i>	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。
<i>TIMER_TRI_OUT_SRC_ENABLE</i>	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。
<i>TIMER_TRI_OUT_SRC_UPDATE</i>	更新。主模式控制器选择更新事件作为 TRGO。
<i>TIMER_TRI_OUT_SRC_CH0</i>	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
<i>TIMER_TRI_OUT_SRC_O0CPRE</i>	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
<i>TIMER_TRI_OUT_SRC_O1CPRE</i>	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
<i>TIMER_TRI_OUT_SRC_O2CPRE</i>	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
<i>TIMER_TRI_OUT_SRC_O3CPRE</i>	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 3-549. 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 3-550. 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_S LAVE_MODE_ENA BLE	主从模式使能
TIMER_MASTER_S LAVE_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 3-551. 函数 timer_external_trigger_config

函数名称	timer_external_trigger_config
函数原型	void timer_external_trigger_config(uint32_t timer_periph, uint32_t extprescaler, uint32_t expolarity, uint32_t extfilter);
功能描述	配置 TIMERx 外部触发输入
先决条件	-
被调用函数	-
输入参数{in}	
timer_periph	TIMER 外设
TIMERx(x=0..2)	TIMER 外设选择
输入参数{in}	
extprescaler	外部触发预分频
TIMER_EXT_TRI_P_SC_OFF	不分频
TIMER_EXT_TRI_P	2 分频

<i>SC_DIV2</i>	
<i>TIMER_EXT_TRI_P</i> <i>SC_DIV4</i>	4 分频
<i>TIMER_EXT_TRI_P</i> <i>SC_DIV8</i>	8 分频
输入参数{in}	
expolarity	外部触发输入极性
<i>TIMER_ETP_FALLING</i>	低电平或者下降沿有效
<i>TIMER_ETP_RISING</i>	高电平或者上升沿有效
输入参数{in}	
extfilter	外部触发滤波控制 (0~15)
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* configure TIMER0 external trigger input */

timer_external_trigger_config (TIMER0, TIMER_EXT_TRI_PSC_DIV2,
    TIMER_ETP_FALLING, 10);
```

函数 `timer_quadrature_decoder_mode_config`

函数`timer_quadrature_decoder_mode_config`描述见下表：

表 3-552. 函数 `timer_quadrature_decoder_mode_config`

函数名称	<code>timer_quadrature_decoder_mode_config</code>
函数原型	<code>void timer_quadrature_decoder_mode_config(uint32_t timer_periph, uint32_t decomode,uint16_t ic0polarity, uint16_t ic1polarity);</code>
功能描述	<code>TIMERx</code> 配置为编码器模式
先决条件	-
被调用函数	-
输入参数{in}	
timer_periph	<code>TIMER</code> 外设
<code>TIMERx(x=0..2)</code>	<code>TIMER</code> 外设选择
输入参数{in}	
decomode	编码器模式
<i>TIMER_ENCODER_MODE0</i>	根据 <code>CI0FE0</code> 的电平，计数器在 <code>CI1FE1</code> 的边沿向上/下计数
<i>TIMER_ENCODER_MODE1</i>	根据 <code>CI1FE1</code> 的电平，计数器在 <code>CI0FE0</code> 的边沿向上/下计数

<i>TIMER_ENCODER_MODE2</i>	根据另一个信号的输入电平，计数器在 CI0FE0 和 CI1FE1 的边沿向上/下计数
输入参数{in}	
<i>ic0polarity</i>	IC0 极性
<i>TIMER_IC_POLARITY_RISING</i>	捕获上升边沿
<i>TIMER_IC_POLARITY_FALLING</i>	捕获下降边沿
输入参数{in}	
<i>ic1polarity</i>	IC1 极性
<i>TIMER_IC_POLARITY_RISING</i>	捕获上升边沿
<i>TIMER_IC_POLARITY_FALLING</i>	捕获下降边沿
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* configure TIMER0 quadrature decoder mode */
timer_quadrature_decoder_mode_config (TIMER0, TIMER_ENCODER_MODE0,
TICKER_IC_POLARITY_RISING, TICKER_IC_POLARITY_RISING);
```

函数 **timer_internal_clock_config**

函数**timer_internal_clock_config**描述见下表：

表 3-553. 函数 timer_internal_clock_config

函数名称	timer_internal_clock_config
函数原型	void timer_internal_clock_config(uint32_t timer_periph);
功能描述	TIMERx 配置为内部时钟模式
先决条件	-
被调用函数	-
输入参数{in}	
<i>timer_periph</i>	TIMER 外设
<i>TIMERx(x=0..2, 14)</i>	TIMER 外设选择
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* configure TIMER0 internal clock mode */
```

```
timer_internal_clock_config(TIMER0);
```

函数 timer_internal_trigger_as_external_clock_config

函数timer_internal_trigger_as_external_clock_config描述见下表:

表 3-554. 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 3-555. 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}	
extrigger	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_POLARITY_RISING	外部触发源高电平或者上升沿有效
TIMER_IC_POLARITY_FALLING	外部触发源低电平或者下降沿有效
TIMER_IC_POLARITY_BOTH_EDGE	下降沿或者上升沿有效
输入参数{in}	
extfilter	滤波参数(0~15)
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* configure TIMER0 the external trigger CI0FE0 as external clock input */

timer_external_trigger_as_external_clock_config(TIMER0,
TMR_SMCFG_TRGSEL_CI0FE0, TMR_IC_POLARITY_RISING, 0);
```

函数 timer_external_clock_mode0_config

函数timer_external_clock_mode0_config描述见下表：

表 3-556. 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,
TIMER_ETP_FALLING, 0);
```

函数 **timer_external_clock_mode1_config**

函数**timer_external_clock_mode1_config**描述见下表：

表 3-557. 函数 **timer_external_clock_mode1_config**

函数名称	timer_external_clock_mode1_config
函数原型	void timer_external_clock_mode1_config(uint32_t timer_periph, uint32_t extprescaler, uint32_t expolarity, uint32_t extfilter);
功能描述	配置 TIMERx 外部时钟模式 1
先决条件	-
被调用函数	timer_external_trigger_config
输入参数{in}	

timer_periph	TIMER 外设
<i>TIMERx(x=0..2)</i>	TIMER 外设选择
输入参数{in}	
extprescaler	ETI 触发源预分频值
<i>TIMER_EXT_TRI_P</i> <i>SC_OFF</i>	不分频
<i>TIMER_EXT_TRI_P</i> <i>SC_DIV2</i>	2 分频
<i>TIMER_EXT_TRI_P</i> <i>SC_DIV4</i>	4 分频
<i>TIMER_EXT_TRI_P</i> <i>SC_DIV8</i>	8 分频
输入参数{in}	
expolarity	ETI 触发源极性
<i>TIMER_ETP_FALLING</i>	下降沿或者低电平有效
<i>TIMER_ETP_RISING</i>	上升沿或者高电平有效
输入参数{in}	
extfilter	ETI 触发源滤波参数 (0~15)
输出参数{out}	
-	-
返回值	
-	-

例如：

```

/* configure TIMER0 the external clock mode1 */

timer_external_clock_mode1_config (TIMER0, TIMER_EXT_TRI_PSC_DIV2,
TMR_ETP_FALLING, 0);
  
```

函数 **timer_external_clock_mode1_disable**

函数**timer_external_clock_mode1_disable**描述见下表：

表 3-558. 函数 **timer_external_clock_mode1_disable**

函数名称	timer_external_clock_mode1_disable
函数原型	void timer_external_clock_mode1_disable(uint32_t timer_periph);
功能描述	TIMERx 外部时钟模式 1 禁能
先决条件	-
被调用函数	-
输入参数{in}	
timer_periph	TIMER 外设
<i>TIMERx(x=0..2)</i>	TIMER 外设选择

输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* disable TIMER0 the external clock mode1 */
timer_external_clock_mode1_disable (TIMER0);
```

函数 timer_channel_remap_config

函数timer_channel_remap_config描述见下表：

表 3-559. 函数 timer_channel_remap_config

函数名称	timer_channel_remap_config
函数原型	void timer_channel_remap_config (uint32_t timer_periph, uint32_t remap);
功能描述	配置 TIMERxt 通道重映射功能
先决条件	-
被调用函数	-
输入参数{in}	
timer_periph	TIMER 外设
TIMERx(x=13)	TIMER 外设选择
输入参数{in}	
remap	重映射功能选择
TIMER13_CI0_RMP_GPIO	通道 0 连接到 GPIO
TIMER13_CI0_RMP_RTCCLK	通道 0 连接到 RTCCLK
TIMER13_CI0_RMP_HXTAL_DIV32	通道 0 连接到 HXTAL/32
TIMER13_CI0_RMP_CKOUTSEL	通道 0 连接到 CKOUTSEL
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* configure TIMER13 channel 0 input is connected to GPIO */
timer_channel_remap_config (TIMER13, TIMER13_CI0_RMP_GPIO);
```

函数 timer_write_chxval_register_config

函数timer_write_chxval_register_config描述见下表:

表 3-560. 函数 timer_write_chxval_register_config

函数名称	timer_write_chxval_register_config
函数原型	void timer_write_chxval_register_config(uint32_t timer_periph, uint16_t ccsel);
功能描述	配置 TIMERx 写 CHxVAL 选择位
先决条件	-
被调用函数	-
输入参数{in}	
timer_periph	TIMER 外设
TIMERx(x=0..2, 13..16)	TIMER 外设选择
输入参数{in}	
ccsel	写 CHxVAL 寄存器选择位
TIMER_CHVSEL_D ISABLE	无影响
TIMER_CHVSEL_E NABLE	当写入捕获比较寄存器的值与寄存器当前值相等时，写入操作无效。
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* configure TIMER0 write CHxVAL register selection */

timer_write_chxval_register_config(TIMER0, TIMER_CHVSEL_ENABLE);
```

函数 timer_output_value_selection_config

函数timer_output_value_selection_config描述见下表:

表 3-561. 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);
```

3.22. TSI

触摸传感控制器（TSI）为触摸按键、滑块、电容近距感测等应用提供了简易的解决方案。章节 [3.22.1](#) 描述了 TSI 的寄存器列表，章节 [3.22.2](#) 对 TSI 库函数进行说明。

3.22.1. 外设寄存器描述

TSI 寄存器列表如下表所示：

表 3-562. TSI 寄存器

寄存器名称	寄存器描述
TSI_CTL0	控制寄存器 0
TSI_INTEN	中断使能寄存器
TSI_INTC	中断标志位清除寄存器
TSI_INTF	中断标志位寄存器
TSI_PHM	引脚迟滞模式寄存器
TSI_ASW	模拟开关寄存器
TSI_SAMPCFG	采样配置寄存器
TSI_CHCFG	通道配置寄存器
TSI_GCTL	组控制寄存器
TSI_GxCYCN (x=0..5)	组 x 周期数寄存器
TSI_CTL1	控制寄存器 1

3.22.2. 外设库函数说明

TSI 库函数列表如下表所示：

表 3-563. TSI 库函数

库函数名称	库函数描述
tsi_deinit	复位TSI外设
tsi_init	TSI初始化
tsi_enable	使能TSI模块
tsi_disable	禁用TSI模块
tsi_sample_pin_enable	使能TSI采样引脚
tsi_sample_pin_disable	禁用TSI采样引脚
tsi_channel_pin_enable	使能TSI通道引脚
tsi_channel_pin_disable	禁用TSI通道引脚
tsi_sofeware_mode_config	配置TSI为软件触发模式
tsi_software_start	软件启动一次电荷转移序列
tsi_software_stop	软件停止已启动的电荷转移序列
tsi_hardware_mode_config	配置TSI为硬件触发模式
tsi_pin_mode_config	状态机为空闲状态时TSI引脚的模式
tsi_extend_charge_config	配置TSI扩展充电状态最大持续时间
tsi_plus_config	配置TSI的电荷转移状态和充电状态的持续时间
tsi_max_number_config	配置TSI电荷转移序列的最大充电、转移周期数
tsi_hysteresis_on	引脚施密特触发迟滞模式使能
tsi_hysteresis_off	引脚施密特触发迟滞模式禁用
tsi_analog_on	模拟开关闭合
tsi_analog_off	模拟开关断开
tsi_group_enable	TSI引脚组使能
tsi_group_disable	TSI引脚组禁止
tsi_group_status_get	电荷转移完成状态获取
tsi_group0_cycle_get	电荷转移序列完成时组0执行的周期数
tsi_group1_cycle_get	电荷转移序列完成时组1执行的周期数
tsi_group2_cycle_get	电荷转移序列完成时组2执行的周期数
tsi_group3_cycle_get	电荷转移序列完成时组3执行的周期数
tsi_group4_cycle_get	电荷转移序列完成时组4执行的周期数
tsi_group5_cycle_get	电荷转移序列完成时组5执行的周期数
tsi_flag_clear	TSI标志位清除
tsi_flag_get	TSI标志位获取
tsi_interrupt_enable	TSI中断使能
tsi_interrupt_disable	TSI中断禁止
tsi_interrupt_flag_clear	TSI中断标志位清除
tsi_interrupt_flag_get	TSI中断标志位获取

函数 tsi_deinit

函数tsi_deinit描述见下表：

表 3-564. 函数 tsi_deinit

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

例如：

```
/* reset TSI*/
tsi_deinit();
```

函数 tsi_init

函数tsi_init描述见下表：

表 3-565. 函数 tsi_init

函数名称	tsi_init
函数原形	void tsi_init(uint32_t prescaler,uint32_t charge_duration,uint32_t transfer_duration,uint32_t max_number);
功能描述	TSI初始化各个参数
先决条件	-
被调用函数	-
输入参数{in}	
prescaler	CTCLK时钟分频系数
<i>TSI_CTCDIV_DIV1</i>	CTCLK等于HCLK
<i>TSI_CTCDIV_DIV2</i>	HCLK 2分频
<i>TSI_CTCDIV_DIV4</i>	HCLK 4分频
<i>TSI_CTCDIV_DIV8</i>	HCLK 8分频
<i>TSI_CTCDIV_DIV16</i>	HCLK 16分频
<i>TSI_CTCDIV_DIV32</i>	HCLK 32分频
<i>TSI_CTCDIV_DIV64</i>	HCLK 64分频
<i>TSI_CTCDIV_DIV128</i>	HCLK 128分频
<i>TSI_CTCDIV_DIV256</i>	HCLK 256分频
<i>TSI_CTCDIV_DIV512</i>	HCLK 512分频

<i>TSI_CTCDIV_DIV10</i> 24	HCLK 1024分频
<i>TSI_CTCDIV_DIV20</i> 48	HCLK 2048分频
<i>TSI_CTCDIV_DIV40</i> 96	HCLK 4096分频
<i>TSI_CTCDIV_DIV81</i> 92	HCLK 8192分频
<i>TSI_CTCDIV_DIV16</i> 384	HCLK 16384分频
<i>TSI_CTCDIV_DIV32</i> 768	HCLK 32768分频
输入参数{in}	
charge_duration	充电状态持续时间
<i>TSI_CHARGE_1CT</i> <i>CLK(x=1..16)</i>	时间位X*tCTCLK(x=1..16)
输入参数{in}	
transfer_duration	电荷转移状态持续时间
<i>TSI_TRANSFER_x</i> <i>CTCLK(x=1..16)SS_</i> <i>CLEAR</i>	时间位X*tCTCLK(x=1..16)
输入参数{in}	
max_number	电荷转移序列的最大充电、转移周期数
<i>TSI_MAXNUM255</i>	255个周期
<i>TSI_MAXNUM511</i>	511个周期
<i>TSI_MAXNUM1023</i>	1023个周期
<i>TSI_MAXNUM2047</i>	2047个周期
<i>TSI_MAXNUM4095</i>	4095个周期
<i>TSI_MAXNUM8191</i>	8191个周期
<i>TSI_MAXNUM1638</i> 3	16383个周期
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* init TSI*/
tsi_init(TSI_CTCDIV_DIV4096, TSI_CHARGE_10CTCLK, TSI_TRANSFER_8CTCLK,
TSI_MAXNUM511);
```

函数 **tsi_enable**

函数tsi_enable描述见下表:

表 3-566. 函数 cec_enable

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

例如:

```
/* enable TSI module */

tsi_enable();
```

函数 **tsi_disable**

函数tsi_disable描述见下表:

表 3-567. 函数 tsi_disable

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

例如:

```
/* disable TSI module */

tsi_disable();
```

函数 tsi_sample_pin_enable

函数tsi_sample_pin_enable描述见下表:

表 3-568. 函数 tsi_sample_pin_enable

函数名称	tsi_sample_pin_enable
函数原形	void tsi_sample_pin_enable(uint32_t sample);
功能描述	使能TSI采样引脚
先决条件	-
被调用函数	-
输入参数{in}	
sample	采样引脚
TSI_SAMPCFG_Gx Py(x=0..5,y=0..3)	GxPy引脚是采样引脚
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* enable G5P3 sample pin */
tsi_sample_pin_enable (TSI_SAMPCFG_G5P3);
```

函数 tsi_sample_pin_disable

函数tsi_sample_pin_disable描述见下表:

表 3-569. 函数 tsi_sample_pin_disable

函数名称	tsi_sample_pin_disable
函数原形	void tsi_sample_pin_disable(uint32_t sample);
功能描述	禁用TSI采样引脚
先决条件	-
被调用函数	-
输入参数{in}	
sample	采样引脚
TSI_SAMPCFG_Gx Py(x=0..5,y=0..3)	GxPy引脚不是采样引脚
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* disable G5P3 sample pin */
```

```
tsi_sample_pin_disable (TSI_SAMPCFG_G5P3);
```

函数 tsi_channel_pin_enable

函数tsi_channel_pin_enable描述见下表:

表 3-570. 函数 tsi_channel_pin_enable

函数名称	tsi_channel_pin_enable
函数原形	void tsi_channel_pin_enable(uint32_t channel);
功能描述	使能TSI通道引脚
先决条件	-
被调用函数	-
输入参数{in}	
channel	通道引脚
TSI_CHCFG_GxPy(x=0..5,y=0..3)	GxPy引脚是通道引脚
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* enable G5P3 channel pin */
tsi_channel_pin_enable (TSI_CHCFG_G5P3);
```

函数 tsi_channel_pin_disable

函数tsi_channel_pin_disable描述见下表:

表 3-571. 函数 tsi_channel_pin_disable

函数名称	tsi_channel_pin_disable
函数原形	void tsi_channel_pin_disable(uint32_t channel);
功能描述	禁用TSI通道引脚
先决条件	-
被调用函数	-
输入参数{in}	
channel	通道引脚
TSI_CHCFG_GxPy(x=0..5,y=0..3)	GxPy引脚不是通道引脚
输出参数{out}	
-	-
返回值	
-	-

例如:

```

/* disable G5P3 channel pin */

tsi_channel_pin_disable (TSI_CHCFG_G5P3);

```

函数 tsi_sofeware_mode_config

函数tsi_sofeware_mode_config描述见下表:

表 3-572. 函数 tsi_sofeware_mode_config

函数名称	tsi_sofeware_mode_config
函数原形	void tsi_sofeware_mode_config (void);
功能描述	配置TSI为软件触发模式
先决条件	-
被调用函数	-
输入参数{in}	
-	-
输出参数{out}	
-	-
返回值	
-	-

例如:

```

/* configure TSI triggering by software */

tsi_sofeware_mode_config ();

```

函数 tsi_software_start

函数tsi_software_start描述见下表:

表 3-573. 函数 tsi_software_start

函数名称	tsi_software_start
函数原形	void tsi_software_start (void);
功能描述	软件启动一次电荷转移序列
先决条件	-
被调用函数	-
输入参数{in}	
-	-
输出参数{out}	
-	-
返回值	
-	-

例如:

```

/* start a charge-transfer sequence when TSI is in software trigger mode */

```

```
tsi_software_start();
```

函数 tsi_software_stop

函数tsi_software_stop描述见下表:

表 3-574. 函数 tsi_software_stop

函数名称	tsi_software_stop
函数原形	void tsi_software_stop (void);
功能描述	软件停止已启动的电荷转移序列
先决条件	-
被调用函数	-
输入参数{in}	
-	-
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* stop a charge-transfer sequence when TSI is in software trigger mode */
```

```
tsi_software_stop();
```

函数 tsi_hardware_mode_config

函数tsi_hardware_mode_config描述见下表:

表 3-575. 函数 tsi_hardware_mode_config

函数名称	tsi_hardware_mode_config
函数原形	void tsi_hardware_mode_config(uint8_t trigger_edge);
功能描述	配置TSI为硬件触发模式
先决条件	-
被调用函数	-
输入参数{in}	
trigger_edge	触发边沿
TSI_FALLING_TRIGGER	下降沿触发
TSI_RISING_TRIGGER	上升沿触发
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* configure TSI triggering by hardware */

tsi_hardware_mode_config(TSI_FALLING_TRIGGER);
```

函数 tsi_pin_mode_config

函数tsi_pin_mode_config描述见下表：

表 3-576. 函数 tsi_pin_mode_config

函数名称	tsi_pin_mode_config	
函数原形	void tsi_pin_mode_config(uint8_t pin_mode);	
功能描述	状态机为空闲状态时TSI引脚的模式	
先决条件	-	
被调用函数	-	
输入参数{in}		
pin_mode	引脚模式	
TSI_OUTPUT_LOW	TSI引脚输出低电平	
TSI_INPUT_FLOATING	TSI引脚保持悬空输入模式	
输出参数{out}		
-	-	
返回值		
-	-	

例如：

```
/* configure TSI pin mode when charge-transfer sequence is IDLE */

tsi_pin_mode_config(TSI_OUTPUT_LOW);
```

函数 tsi_extend_charge_config

函数tsi_extend_charge_config描述见下表：

表 3-577. 函数 tsi_extend_charge_config

函数名称	tsi_extend_charge_config	
函数原形	void tsi_extend_charge_config(ControlStatus extend,uint8_t prescaler,uint32_t max_duration);	
功能描述	配置TSI扩展充电状态最大持续时间	
先决条件	-	
被调用函数	-	
输入参数{in}		
extend	扩展充电状态是否使能	
ENABLE	使能	
DISABLE	不使能	
输入参数{in}		
prescaler	ECCLK时钟分频系数	

<i>TSI_EXTEND_DIV1</i>	HCLK不分频
<i>TSI_EXTEND_DIV2</i>	HCLK2分频
<i>TSI_EXTEND_DIV3</i>	HCLK3分频
<i>TSI_EXTEND_DIV4</i>	HCLK4分频
<i>TSI_EXTEND_DIV5</i>	HCLK5分频
<i>TSI_EXTEND_DIV6</i>	HCLK6分频
<i>TSI_EXTEND_DIV7</i>	HCLK7分频
<i>TSI_EXTEND_DIV8</i>	HCLK8分频
输入参数{in}	
max_duration	扩展充电状态最大持续时间
<i>value range 1...128</i>	时间位X*t _{ECCLK} (x=1..128)
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* configure extend charge state */

tsi_extend_charge_config (ENABLE, TSI_EXTEND_DIV2, 10);
```

函数 **tsi_plus_config**

函数tsi_plus_config描述见下表：

表 3-578. 函数 **tsi_plus_config**

函数名称	tsi_plus_config
函数原形	void tsi_plus_config(uint32_t prescaler,uint32_t charge_duration,uint32_t transfer_duration);
功能描述	配置TSI的电荷转移状态和充电状态的持续时间
先决条件	-
被调用函数	-
输入参数{in}	
prescaler	ECCLK时钟分频系数
<i>TSI_CTCDIV_DIV1</i>	HCLK不分频
<i>TSI_CTCDIV_DIV2</i>	HCLK2分频
<i>TSI_CTCDIV_DIV4</i>	HCLK4分频
<i>TSI_CTCDIV_DIV8</i>	HCLK8分频
<i>TSI_CTCDIV_DIV16</i>	HCLK16分频
<i>TSI_CTCDIV_DIV32</i>	HCLK32分频
<i>TSI_CTCDIV_DIV64</i>	HCLK64分频
<i>TSI_CTCDIV_DIV128</i>	HCLK128分频
<i>TSI_CTCDIV_DIV256</i>	HCLK256分频

6	
<i>TSI_CTCDIV_DIV512</i>	HCLK512分频
<i>TSI_CTCDIV_DIV1024</i>	HCLK1024分频
<i>TSI_CTCDIV_DIV2048</i>	HCLK2048分频
<i>TSI_CTCDIV_DIV4096</i>	HCLK4096分频
<i>TSI_CTCDIV_DIV8192</i>	HCLK8192分频
<i>TSI_CTCDIV_DIV16384</i>	HCLK16384分频
<i>TSI_CTCDIV_DIV32768</i>	HCLK32768分频
输入参数{in}	
charge_duration	充电状态持续时间
<i>TSI_CHARGE_1CTCLK(x=1..16)</i>	时间位X*tCTCLK(x=1..16)
输入参数{in}	
transfer_duration	电荷转移状态持续时间
<i>TSI_TRANSFER_xCTCLK(x=1..16)</i>	时间位X*tCTCLK(x=1..16)
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* configure charge plus and transfer plus */
tsi_plus_config (TSI_CTCDIV_DIV1024, TSI_CHARGE_10CTCLK,
TSI_TRANSFER_8CTCLK);
```

函数 **tsi_max_number_config**

函数tsi_max_number_config描述见下表：

表 3-579. 函数 **tsi_max_number_config**

函数名称	tsi_max_number_config
函数原形	void tsi_max_number_config(uint32_t max_number);
功能描述	配置TSI电荷转移序列的最大充电、转移周期数
先决条件	-
被调用函数	-

输入参数{in}	
max_number	电荷转移序列的最大充电、转移周期数
<i>TSI_MAXNUM255</i>	255个周期
<i>TSI_MAXNUM511</i>	511个周期
<i>TSI_MAXNUM1023</i>	1023个周期
<i>TSI_MAXNUM2047</i>	2047个周期
<i>TSI_MAXNUM4095</i>	4095个周期
<i>TSI_MAXNUM8191</i>	8191个周期
<i>TSI_MAXNUM16383</i>	16383个周期
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* configure the max cycle number of a charge-transfer sequence */
tsi_max_number_config (TSI_MAXNUM1023);
```

函数 tsi_hysteresis_on

函数tsi_hysteresis_on描述见下表：

表 3-580. 函数 tsi_hysteresis_on

输入参数{in}	
group_pin	引脚迟滞模式使能
<i>TSI_PHM_GxPy(x=0..5,y=0..3)</i>	引脚GxPy施密特触发迟滞模式使能(x=0..5,y=0..3)
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* switch on hysteresis pin */
tsi_hysteresis_on (TSI_PHM_G5P3);
```

函数 tsi_hysteresis_off

函数tsi_hysteresis_off描述见下表:

表 3-581. 函数 tsi_hysteresis_off

函数名称	tsi_hysteresis_off
函数原形	void tsi_hysteresis_off(uint32_t group_pin);
功能描述	引脚施密特触发迟滞模式禁用
先决条件	-
被调用函数	-
输入参数{in}	
group_pin	引脚迟滞模式禁用
TSI_PHM_GxPy(x=0..5,y=0..3)	引脚GxPy施密特触发迟滞模式禁用(x=0..5,y=0..3)
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* switch off hysteresis pin */
tsi_hysteresis_off (TSI_PHM_G5P3);
```

函数 tsi_analog_on

函数tsi_analog_on描述见下表:

表 3-582. 函数 tsi_analog_on

函数名称	tsi_analog_on
函数原形	void tsi_analog_on(uint32_t group_pin);
功能描述	模拟开关闭合
先决条件	-
被调用函数	-
输入参数{in}	
group_pin	模拟开关状态
TSI_ASW_GxPy (x=0..5,y=0..3)	引脚GxPy 的模拟开关闭合(x=0..5,y=0..3)
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* switch on analog pin */
```

```
tsi_analog_on (TSI_ASW_G5P3);
```

函数 tsi_analog_off

函数tsi_analog_off描述见下表:

表 3-583. 函数 tsi_analog_off

函数名称	tsi_analog_off
函数原形	void tsi_analog_off(uint32_t group_pin);
功能描述	模拟开关断开
先决条件	-
被调用函数	-
输入参数{in}	
group_pin	模拟开关状态
TSI_ASW_GxPy (x=0..5,y=0..3)	引脚GxPy 的模拟开关断开(x=0..5,y=0..3)
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* switch off analog pin */
tsi_analog_off (TSI_ASW_G5P3);
```

函数 tsi_group_enable

函数tsi_group_enable描述见下表:

表 3-584. 函数 tsi_group_enable

函数名称	tsi_group_enable
函数原形	void tsi_group_enable(uint32_t group);
功能描述	TSI引脚组使能
先决条件	-
被调用函数	-
输入参数{in}	
group	引脚组
TSI_GCTL_GEx(x=0..5)	引脚组x使能
输出参数{out}	
-	-
返回值	
-	-

例如:

```

/* enable group 5 */

tsi_group_enable (TSI_GCTL_GE5);

```

函数 tsi_group_disable

函数tsi_group_disable描述见下表:

表 3-585. 函数 tsi_group_disable

函数名称	tsi_group_disable
函数原形	void tsi_group_disable(uint32_t group);
功能描述	TSI引脚组禁用
先决条件	-
被调用函数	-
输入参数{in}	
group	引脚组
TSI_GCTL_GEx(x=0.5)	引脚组x禁用
输出参数{out}	
-	-
返回值	
-	-

例如:

```

/* disable group 5 */

tsi_group_disable (TSI_GCTL_GE5);

```

函数 tsi_group_status_get

函数tsi_group_status_get描述见下表:

表 3-586. 函数 tsi_group_status_get

函数名称	tsi_group_status_get
函数原形	FlagStatus tsi_group_status_get(uint32_t group);
功能描述	电荷转移完成状态获取
先决条件	-
被调用函数	-
输入参数{in}	
group	引脚组
TSI_GCTL_GCx(x=0.5)	组x电荷转移完成
输出参数{out}	
-	-
返回值	
FlagStatus	SET或RESET

例如：

```
/* get group complete status */

FlagStatus flag = RESET;

flag = tsi_flag_get (TSI_GCTL_GC5);
```

函数 tsi_group0_cycle_get

函数tsi_group0_cycle_get描述见下表：

表 3-587. 函数 tsi_group0_cycle_get

函数名称	tsi_group0_cycle_get
函数原形	uint16_t tsi_group0_cycle_get(void);
功能描述	电荷转移序列完成时组0执行的周期数
先决条件	-
被调用函数	-
输入参数{in}	
-	-
输出参数{out}	
-	-
返回值	
uint16_t	0-8192

例如：

```
/* get the cycle number for group0 as soon as a charge-transfer sequence completes */

uint16_t flag = 0;

flag = tsi_group0_cycle_get();
```

函数 tsi_group1_cycle_get

函数tsi_group1_cycle_get描述见下表：

表 3-588. 函数 tsi_group1_cycle_get

函数名称	tsi_group1_cycle_get
函数原形	uint16_t tsi_group1_cycle_get(void);
功能描述	电荷转移序列完成时组1执行的周期数
先决条件	-
被调用函数	-
输入参数{in}	
-	-
输出参数{out}	
-	-
返回值	

例如：

```
/* get the cycle number for group1 as soon as a charge-transfer sequence completes */

uint16_t flag = 0;

flag = tsi_group1_cycle_get();
```

函数 **tsi_group2_cycle_get**

函数tsi_group2_cycle_get描述见下表：

表 3-589. 函数 tsi_group2_cycle_get

函数名称	tsi_group2_cycle_get	
函数原形	uint16_t tsi_group2_cycle_get(void);	
功能描述	电荷转移序列完成时组2执行的周期数	
先决条件	-	
被调用函数	-	
输入参数{in}		
-	-	
输出参数{out}		
-	-	
返回值		
uint16_t	0-8192	

例如：

```
/* get the cycle number for group2 as soon as a charge-transfer sequence completes */

uint16_t flag = 0;

flag = tsi_group2_cycle_get();
```

函数 **tsi_group3_cycle_get**

函数tsi_group3_cycle_get描述见下表：

表 3-590. 函数 tsi_group3_cycle_get

函数名称	tsi_group3_cycle_get	
函数原形	uint16_t tsi_group3_cycle_get(void);	
功能描述	电荷转移序列完成时组3执行的周期数	
先决条件	-	
被调用函数	-	
输入参数{in}		
-	-	
输出参数{out}		
-	-	

返回值	
uint16_t	0-8192

例如：

```
/* get the cycle number for group3 as soon as a charge-transfer sequence completes */

uint16_t flag = 0;

flag = tsi_group3_cycle_get();
```

函数 **tsi_group4_cycle_get**

函数tsi_group4_cycle_get描述见下表：

表 3-591. 函数 **tsi_group4_cycle_get**

函数名称	tsi_group4_cycle_get
函数原形	uint16_t tsi_group4_cycle_get(void);
功能描述	电荷转移序列完成时组4执行的周期数
先决条件	-
被调用函数	-
输入参数{in}	
-	-
输出参数{out}	
-	-
返回值	
uint16_t	0-8192

例如：

```
/* get the cycle number for group4 as soon as a charge-transfer sequence completes */

uint16_t flag = 0;

flag = tsi_group4_cycle_get();
```

函数 **tsi_group5_cycle_get**

函数tsi_group5_cycle_get描述见下表：

表 3-592. 函数 **tsi_group5_cycle_get**

函数名称	tsi_group5_cycle_get
函数原形	uint16_t tsi_group5_cycle_get(void);
功能描述	电荷转移序列完成时组5执行的周期数
先决条件	-
被调用函数	-
输入参数{in}	
-	-
输出参数{out}	

-	-
返回值	
uint16_t	0-8192

例如：

```
/* get the cycle number for group5 as soon as a charge-transfer sequence completes */

uint16_t flag = 0;

flag = tsi_group5_cycle_get();
```

函数 **tsi_flag_clear**

函数tsi_flag_clear描述见下表：

表 3-593. 函数 **tsi_flag_clear**

函数名称	tsi_flag_clear
函数原形	void tsi_flag_clear(uint32_t flag);
功能描述	TSI标志位清除
先决条件	-
被调用函数	-
输入参数{in}	
flag	标志位
TSI_FLAG_CTCF	电荷转移完成标志
TSI_FLAG_MNERR	最大循环次数错误标志
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* clear TSI_FLAG_CTCF_CLR flag */

tsi_flag_clear (TSI_FLAG_CTCF_CLR);
```

函数 **tsi_flag_get**

函数tsi_flag_get描述见下表：

表 3-594. 函数 **tsi_flag_get**

函数名称	tsi_flag_get
函数原形	FlagStatus tsi_flag_get(uint32_t flag);
功能描述	TSI标志位获取
先决条件	-
被调用函数	-
输入参数{in}	

flag	标志位
<i>TSI_FLAG_CTCF</i>	电荷转移完成标志
<i>TSI_FLAG_MNERR</i>	最大循环次数错误标志
输出参数{out}	
-	-
返回值	
FlagStatus	SET或RESET

例如：

```
/* get TSI_FLAG_CTCF flag */

FlagStatus flag = RESET;

flag = tsi_flag_get (TSI_FLAG_CTCF);
```

函数 **tsi_interrupt_enable**

函数tsi_interrupt_enable描述见下表：

表 3-595. 函数 **tsi_interrupt_enable**

函数名称	tsi_interrupt_enable
函数原形	void tsi_interrupt_enable(uint32_t source);
功能描述	TSI中断使能
先决条件	-
被调用函数	-
输入参数{in}	
source	中断使能位
<i>TSI_INT_CCTCF</i>	电荷转移完成标志中断使能
<i>TSI_INT_MNERR</i>	最大循环次数错误中断使能
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* enable TSI_INT_CCTCF interrupt */

tsi_interrupt_enable (TSI_INT_CCTCF);
```

函数 **tsi_interrupt_disable**

函数tsi_interrupt_disable描述见下表：

表 3-596. 函数 **tsi_interrupt_disable**

函数名称	tsi_interrupt_disable
函数原形	void tsi_interrupt_disable(uint32_t source);

功能描述	TSI中断禁止
先决条件	-
被调用函数	-
输入参数{in}	
source	中断使能位
<i>TSI_INT_CCTCF</i>	电荷转移完成标志中断禁止
<i>TSI_INT_MNERR</i>	最大循环次数错误中断禁止
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* disable TSI_INT_CCTCF interrupt */

tsi_interrupt_disable (TSI_INT_CCTCF);
```

函数 **tsi_interrupt_flag_clear**

函数 **tsi_interrupt_flag_clear** 描述见下表：

表 3-597. 函数 **tsi_interrupt_flag_clear**

函数名称	tsi_interrupt_flag_clear
函数原形	void tsi_interrupt_flag_clear(uint32_t flag);
功能描述	TSI中断标志位清除
先决条件	-
被调用函数	-
输入参数{in}	
flag	中断标志位
<i>TSI_INT_FLAG_CTF</i>	电荷转移完成标志中断
<i>TSI_INT_FLAG_MNERR</i>	最大循环次数错误标志中断
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* clear TSI_INT_FLAG_CTCF interrupt flag */

tsi_interrupt_flag_clear (TSI_INT_FLAG_CTCF);
```

函数 tsi_interrupt_flag_get

函数tsi_interrupt_flag_get描述见下表:

表 3-598. 函数 tsi_interrupt_flag_get

函数名称	tsi_interrupt_flag_get
函数原形	FlagStatus tsi_interrupt_flag_get(uint32_t flag);
功能描述	TSI中断标志位获取
先决条件	-
被调用函数	-
输入参数{in}	
flag	中断标志位
TSI_INT_FLAG_CTCF	电荷转移完成标志中断
TSI_INT_FLAG_MNERR	最大循环次数错误标志中断
输出参数{out}	
-	-
返回值	
FlagStatus	SET或RESET

例如:

```
/* get TSI_INT_FLAG_CTCF interrupt flag */
FlagStatus flag = RESET;
flag = tsi_interrupt_flag_get (TSI_INT_FLAG_CTCF);
```

3.23. USART

通用同步异步收发器(USART)提供了一个灵活方便的串行数据交换接口，章节[3.23.1](#)描述了USART的寄存器列表，章节[3.23.2](#)对USART库函数进行说明。

3.23.1. 外设寄存器说明

USART寄存器列表如下表所示:

表 3-599. 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_RFCS	接收FIFO控制和状态寄存器

3.23.2. 外设库函数说明

USART库函数列表如下表所示：

表 3-600. 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
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低功耗模式下或者SmartCard模式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_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
uart_wakeup_enable	使能USART唤醒
uart_wakeup_disable	失能USART唤醒

库函数名称	库函数描述
uart_wakeup_mode_config	配置USART唤醒模式
uart_command_enable	使能USART请求
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/RFC5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_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

Enum usart_flag_enum

表 3-601. 枚举类型 usart_flag_enum

Member name	Function description
USART_FLAG_REA	接收使能通知标志
USART_FLAG_TEA	发送使能通知标志
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	断开信号发送标识
USART_FLAG_AM	ADDR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	快结束标志
USART_FLAG_RT	接收超时标志
USART_FLAG_CTS	CTS电平
USART_FLAG_CTSF	CTS变化标志
USART_FLAG_LBD	LIN断开检测标志
USART_FLAG_TBE	发送数据寄存器空
USART_FLAG_TC	发送完成标志
USART_FLAG_RBNE	读数据缓冲区非空标志
USART_FLAG_IDLE	空闲线检测标志
USART_FLAG_ORERR	溢出错误标志
USART_FLAG_NERR	噪声错误标志
USART_FLAG_FERR	帧错误标志
USART_FLAG_PERR	校验错误标志
USART_FLAG_RFF	接收FIFO满标志
USART_FLAG_RFE	接收FIFO空标志

Enum usart_interrupt_flag_enum
表 3-602. 枚举类型 usart_interrupt_flag_enum

Member name	Function description
USART_INT_FLAG_EB	快结束中断标志
USART_INT_FLAG_RT	接收超时中断标志
USART_INT_FLAG_AM	ADDR匹配中断标志
USART_INT_FLAG_PERR	校验错误中断标志
USART_INT_FLAG_TBE	发送数据寄存器空中断标志
USART_INT_FLAG_TC	发送完成中断标志
USART_INT_FLAG_RBNE	读数据缓冲区非空中断标志
USART_INT_FLAG_RBNE_ORE RR	溢出错误中断标志
USART_INT_FLAG_IDLE	空闲线检测中断标志
USART_INT_FLAG_LBD	LIN断开检测中断标志
USART_INT_FLAG_WU	从深度睡眠模式唤醒中断标志
USART_INT_FLAG_CTS	CTS中断中断标志
USART_INT_FLAG_ERR_NERR	噪声错误中断标志
USART_INT_FLAG_ERR_ORER R	溢出错误中断标志
USART_INT_FLAG_ERR_FERR	帧错误中断标志
USART_INT_FLAG_RFFINT	接收FIFO满中断标志

Enum usart_interrupt_enum
表 3-603. 枚举类型 usart_interrupt_enum

Member name	Function description
USART_INT_EB	块尾中断使能
USART_INT_RT	接收超时中断使能
USART_INT_AM	ADDR字符匹配中断使能
USART_INT_PERR	校验错误中断使能
USART_INT_TBE	发送寄存器空中断使能
USART_INT_TC	发送完成中断使能
USART_INT_RBNE	读数据缓冲区非空中断和过载错误中断使能
USART_INT_IDLE	IDLE线检测中断使能
USART_INT_LBD	LIN断开信号检测中断使能
USART_INT_WU	从深度睡眠模式唤醒中断使能
USART_INT_CTS	CTS中断使能
USART_INT_ERR	错误中断使能
USART_INT_RFF	接收FIFO满中断使能

Enum usart_invert_enum
表 3-604. 枚举类型 usart_invert_enum

Member name	Function description
USART_DINV_ENABLE	数据位反转
USART_DINV_DISABLE	数据位不反转
USART_TXPIN_ENABLE	TX管脚电平反转
USART_TXPIN_DISABLE	TX管脚电平不反转
USART_RXPIN_ENABLE	RX管脚电平反转
USART_RXPIN_DISABLE	RX管脚电平不反转
USART_SWAP_ENABLE	交换TX/RX管脚
USART_SWAP_DISABLE	不交换TX/RX管脚

函数 usart_deinit

函数usart_deinit描述见下表:

表 3-605. 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 3-606. 函数 usart_baudrate_set

函数名称	usart_baudrate_set
函数原型	void usart_baudrate_set(uint32_t usart_periph, uint32_t baudval);
功能描述	配置USART波特率
先决条件	-

被调用函数	rcu_clock_freq_get
输入参数{in}	
usart_periph	外设USARTx
USARTx	x=0,1
输入参数{in}	
baudval	波特率值
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* configure USART0 baud rate value */
uart_baudrate_set(USART0, 115200);
```

函数 **usart_parity_config**

函数usart_parity_config描述见下表：

表 3-607. 函数 **usart_parity_config**

函数名称	usart_parity_config
函数原型	void usart_parity_config(uint32_t usart_periph, uint32_t paritycfg);
功能描述	配置USART奇偶校验
先决条件	-
被调用函数	-
输入参数{in}	
usart_periph	外设USARTx
USARTx	x=0,1
输入参数{in}	
paritycfg	配置USART奇偶校验
USART_PM_NONE	无校验
USART_PM_ODD	奇校验
USART_PM EVEN	偶校验
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* configure USART0 parity */
uart_parity_config(USART0, USART_PM EVEN);
```

函数 usart_word_length_set

函数usart_word_length_set描述见下表:

表 3-608. 函数 usart_word_length_set

函数名称	usart_word_length_set
函数原型	void usart_word_length_set(uint32_t usart_periph, uint32_t wlen);
功能描述	配置USART字长
先决条件	-
被调用函数	-
输入参数{in}	
usart_periph	外设USARTx
USARTx	x=0,1
输入参数{in}	
wlen	配置USART字长
USART_WL_8BIT	8位
USART_WL_9BIT	9位
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* configure USART0 word length */
usart_word_length_set(USART0, USART_WL_9BIT);
```

函数 usart_stop_bit_set

函数usart_stop_bit_set描述见下表:

表 3-609. 函数 usart_stop_bit_set

函数名称	usart_stop_bit_set
函数原型	void usart_stop_bit_set(uint32_t usart_periph, uint32_t stblen);
功能描述	配置USART停止位
先决条件	-
被调用函数	-
输入参数{in}	
usart_periph	外设USARTx
USARTx	x=0,1
输入参数{in}	
stblen	配置USART停止位
USART_STB_1BIT	1位
USART_STB_0_5BIT	0.5位

<i>USART_STB_2BIT</i>	2位
<i>USART_STB_1_5BIT</i> <i>T</i>	1.5位
输出参数{out}	
-	-
返回值	
-	-

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 3-610. 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	-
输入参数{in}	
usart_periph	外设USARTx
<i>USARTx</i>	x=0,1
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* enable USART0 */

uart_enable(USART0);
```

函数 **uart_disable**

函数uart_disable描述见下表：

表 3-611. 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	-
输入参数{in}	
usart_periph	外设USARTTx
USARTx	x=0,1
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* disable USART0 */
uart_disable(USART0);
```

函数 **usart_transmit_config**

函数**usart_transmit_config**描述见下表：

表 3-612. 函数 usart_transmit_config

函数名称	usart_transmit_config
函数原型	void usart_transmit_config(uint32_t usart_periph, uint32_t txconfig);
功能描述	USART发送器配置
先决条件	-
被调用函数	-
输入参数{in}	
usart_periph	外设USARTTx
USARTx	x=0,1
输入参数{in}	
txconfig	使能/失能USART发送器
USART_TRANSMIT_ENABLE	使能USART发送
USART_TRANSMIT_DISABLE	失能USART发送
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* configure USART0 transmitter */
uart_transmit_config(USART0, USART_TRANSMIT_ENABLE);
```

函数 **uart_receive_config**

函数uart_receive_config描述见下表:

表 3-613. 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	-
输入参数{in}	
usart_periph	外设USARTx
USARTx	x=0,1
输入参数{in}	
rxconfig	使能/失能USART接收器
USART_RECEIVE_ENABLE	使能USART接收
USART_RECEIVE_DISABLE	失能USART接收
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* configure USART0 receiver */

uart_receive_config(USART0, USART_RECEIVE_ENABLE);
```

函数 **uart_data_first_config**

函数uart_data_first_config描述见下表:

表 3-614. 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	-
输入参数{in}	
usart_periph	外设USARTx
USARTx	x=0,1
输入参数{in}	
msbf	数据传输时低位在前/高位在前
USART_MSBF_LS	数据传输时低位在前

<i>B</i>	
<i>USART_MSBF_MS_B</i>	数据传输时高位在前
输出参数{out}	
-	-
返回值	
-	-

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 3-615. 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	-
输入参数{in}	
usart_periph	外设USARTx
USARTx	x=0,1
输入参数{in}	
invertpara	参考枚举usart_invert_enum
USART_DINV_ENA_BLE	数据位电平反转
USART_DINV_DISABLE	数据位电平不反转
USART_TXPIN_ENABLE	TX引脚电平反转
USART_TXPIN_DISABLE	TX引脚电平不反转
USART_RXPIN_ENABLE	RX引脚电平反转
USART_RXPIN_DISABLE	RX引脚电平不反转
USART_SWAP_ENABLE	TX和RX管脚功能被交换
USART_SWAP_DISABLE	TX和RX管脚功能不被交换

输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* configure USART0 inversion */
uart_invert_config(USART0, USART_DINV_ENABLE);
```

函数 **uart_overrun_enable**

函数 **uart_overrun_enable** 描述见下表：

表 3-616. 函数 **uart_overrun_enable**

函数名称	uart_overrun_enable
函数原型	void usart_overrun_enable (uint32_t usart_periph);
功能描述	使能USART溢出禁止功能
先决条件	-
被调用函数	-
输入参数{in}	
usart_periph	外设USARTx
USARTx	x=0,1
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* enable USART0 overrun */
uart_overrun_enable (USART0);
```

函数 **uart_overrun_disable**

函数 **uart_overrun_disable** 描述见下表：

表 3-617. 函数 **uart_overrun_disable**

函数名称	uart_overrun_disable
函数原型	void usart_overrun_disable (uint32_t usart_periph);
功能描述	失能USART溢出禁止功能
先决条件	-
被调用函数	-
输入参数{in}	
usart_periph	外设USARTx
USARTx	x=0,1

输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* disable USART0 overrun */

uart_overrun_disable (USART0);
```

函数 **uart_oversample_config**

函数uart_oversample_config描述见下表：

表 3-618. 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	-
输入参数{in}	
usart_periph	外设USARTx
USARTx	x=0,1
输入参数{in}	
oversamp	过采样值
USART_OVSMOD_8	8倍过采样
USART_OVSMOD_16	16倍过采样
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* configure 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 3-619. 函数 **uart_sample_bit_config**

函数名称	uart_sample_bit_config
函数原型	void usart_sample_bit_config(uint32_t usart_periph, uint32_t osb);

功能描述	配置USART单次采样方式
先决条件	-
被调用函数	-
输入参数{in}	
usart_periph	外设USARTx
USARTx	x=0,1
输入参数{in}	
osb	单次采样方式
USART_OSB_1BIT	1次采样方法
USART_OSB_3BIT	3次采样方法
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* config USART0 1 bit sample mode */

uart_sample_bit_config(USART0, USART_OSB_1BIT);
```

函数 **uart_receiver_timeout_enable**

函数**uart_receiver_timeout_enable**描述见下表：

表 3-620. 函数 **uart_receiver_timeout_enable**

函数名称	uart_receiver_timeout_enable
函数原型	void usart_receiver_timeout_enable(uint32_t usart_periph);
功能描述	使能USART接收超时
先决条件	-
被调用函数	-
输入参数{in}	
usart_periph	外设USARTx
USARTx	x=0
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* enable USART0 receiver timeout */

uart_receiver_timeout_enable(USART0);
```

函数 usart_receiver_timeout_disable

函数usart_receiver_timeout_disable描述见下表:

表 3-621. 函数 usart_receiver_timeout_disable

函数名称	usart_receiver_timeout_disable	
函数原型	void usart_receiver_timeout_disable(uint32_t usart_periph);	
功能描述	失能USART接收超时	
先决条件	-	
被调用函数	-	
输入参数{in}		
usart_periph	外设USARTx	
USARTx	x=0	
输出参数{out}		
-	-	
返回值		
-	-	

例如:

```
/* disable USART0 receiver timeout */

usart_receiver_timeout_disable(USART0);
```

函数 usart_receiver_timeout_threshold_config

函数usart_receiver_timeout_threshold_config描述见下表:

表 3-622. 函数 usart_receiver_timeout_threshold_config

函数名称	usart_receiver_timeout_threshold_config	
函数原型	void usart_receiver_timeout_threshold_config(uint32_t usart_periph, uint32_t rtimeout);	
功能描述	设置USART接收超时阈值	
先决条件	-	
被调用函数	-	
输入参数{in}		
usart_periph	外设USARTx	
USARTx	x=0	
输入参数{in}		
rtimeout	超时时间	
0x00000000-0x00FFFFFF	超时时间值	
输出参数{out}		
-	-	
返回值		
-	-	

例如：

```
/* set the receiver timeout threshold of USART0 */

uart_receiver_timeout_threshold_config(USART0, 115200*3);
```

函数 `uart_data_transmit`

函数`uart_data_transmit`描述见下表：

表 3-623. 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	-
输入参数{in}	
usart_periph	外设USARTx
USARTx	x=0,1
输入参数{in}	
data	发送的数据
0-0xFF	发送的数据
输出参数{out}	
-	-
返回值	
-	-

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 3-624. 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	-
输入参数{in}	
usart_periph	外设USARTx
USARTx	x=0,1
输出参数{out}	

-	-
返回值	
uint32_t	接收到的数据 (0-0xFF)

例如：

```
/* USART0 receive data */

uint16_t temp ;

temp = usart_data_receive(USART0);
```

函数 **usart_autobaud_detection_enable**

函数usart_autobaud_detection_enable描述见下表：

表 3-625. 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}	
usart_periph	外设USARTTx
USARTTx	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 3-626. 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}	
usart_periph	外设USARTTx

USARTx	x=0
输出参数{out}	
-	-
返回值	

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 3-627. 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	-
输入参数{in}	
usart_periph	外设USARTx
USARTx	x=0
输入参数{in}	
abdmmod	自动波特率检测模式
USART_ABDM_FT OR	下降沿到上升沿的测量
USART_ABDM_FT OF	下降沿对下降沿的测量
输出参数{out}	
-	-
返回值	

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 3-628. 函数 usart_address_config

函数名称	usart_address_config
函数原型	void usart_address_config(uint32_t usart_periph, uint8_t addr);
功能描述	在地址掩码唤醒模式下配置USART地址
先决条件	-
被调用函数	-
输入参数{in}	
usart_periph	外设USARTTx
USARTTx	x=0,1
输入参数{in}	
addr	USART地址
0-0xFF	USART地址
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* configure address of the USART0 */
usart_address_config(USART0, 0x00);
```

函数 usart_address_detection_mode_config

函数usart_address_detection_mode_config描述见下表：

表 3-629. 函数 usart_address_detection_mode_config

函数名称	usart_address_detection_mode_config
函数原型	void usart_address_detection_mode_config(uint32_t usart_periph, uint32_t addmod);
功能描述	配置USART地址检测模式
先决条件	-
被调用函数	-
输入参数{in}	
usart_periph	外设USARTTx
USARTTx	x=0,1
输入参数{in}	
addmod	地址检测模式
USART_ADDM_4BIT	4位地址检测
USART_ADDM_FULLBIT	全位地址检测
输出参数{out}	
-	-

返回值	
-	-

例如：

```
/*configure address detection mode */
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 3-630. 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	-
输入参数{in}	
usart_periph	外设USARTx
USARTx	x=0,1
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* enable USART0 receiver in mute mode */

uart_mute_mode_enable(USART0);
```

函数 **uart_mute_mode_disable**

函数uart_mute_mode_disable描述见下表：

表 3-631. 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	-
输入参数{in}	
usart_periph	外设USARTx
USARTx	x=0,1
-	-
返回值	

-	-
---	---

例如：

```
/* disable USART0 receiver in mute mode */

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 3-632. 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	-
输入参数{in}	
usart_periph	外设USARTx
USARTx	x=0,1
输入参数{in}	
wmethod	两种方法用于进入或退出静默模式
USART_WM_IDLE	空闲线唤醒
USART_WM_ADDR	地址掩码唤醒
输出参数{out}	
-	-
返回值	
-	-

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 3-633. 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}	

usart_periph	外设USARTx
<i>USARTx</i>	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 3-634. 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}	
usart_periph	外设USARTx
<i>USARTx</i>	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_decton_length_config**

函数uart_lin_break_decton_length_config描述见下表：

表 3-635. 函数 **uart_lin_break_decton_length_config**

函数名称	uart_lin_break_decton_length_config
函数原型	void uart_lin_break_decton_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	-

输入参数{in}	
uart_periph	外设USARTx
USARTx	x=0
输入参数{in}	
Iblen	LIN模式中断帧长度
USART_LBLEN_10 B	断开帧长度为10 bits
USART_LBLEN_11 B	断开帧长度为11 bits
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* configure LIN break frame length */

uart_lin_break_decton_length_config(USART0, USART_LBLEN_10B);
```

函数 usart_halfduplex_enable

函数usart_halfduplex_enable描述见下表：

表 3-636. 函数 usart_halfduplex_enable

函数名称	usart_halfduplex_enable
函数原型	void usart_halfduplex_enable(uint32_t usart_periph);
功能描述	使能USART半双工模式
先决条件	-
被调用函数	-
输入参数{in}	
uart_periph	外设USARTx
USARTx	x=0,1
输出参数{out}	
-	-
返回值	
-	-

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 3-637. 函数 usart_halfduplex_disable

函数名称	usart_halfduplex_disable
函数原型	void usart_halfduplex_disable(uint32_t usart_periph);
功能描述	失能USART半双工模式
先决条件	-
被调用函数	-
输入参数{in}	
usart_periph	外设USARTx
USARTx	x=0,1
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* disable USART0 half duplex mode*/
usart_halfduplex_disable(USART0);
```

函数 usart_clock_enable

函数usart_clock_enable描述见下表：

表 3-638. 函数 usart_clock_enable

函数名称	usart_clock_enable
函数原型	void usart_clock_enable(uint32_t usart_periph);
功能描述	使能USART CK引脚
先决条件	-
被调用函数	-
输入参数{in}	
usart_periph	外设USARTx
USARTx	x=0
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* enable USART0 CK pin */
usart_synchronous_clock_enable(USART0);
```

函数 usart_clock_disable

函数usart_clock_disable描述见下表：

表 3-639. 函数 usart_clock_disable

函数名称	usart_clock_disable
函数原型	void usart_clock_disable(uint32_t usart_periph);
功能描述	失能USART CK引脚
先决条件	-
被调用函数	-
输入参数{in}	
usart_periph	外设USARTx
USARTx	x=0
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* disable USART0 CK pin */
usart_clock_disable(USART0);
```

函数 usart_synchronous_clock_config

函数usart_synchronous_clock_config描述见下表：

表 3-640. 函数 usart_synchronous_clock_config

函数名称	usart_synchronous_clock_config
函数原型	void usart_synchronous_clock_config(uint32_t usart_periph, uint32_t clen, uint32_t cph, uint32_t cpl);
功能描述	配置USART同步通讯模式参数
先决条件	-
被调用函数	-
输入参数{in}	
usart_periph	外设USARTx
USARTx	x=0,1
输入参数{in}	
clen	CK信号长度
USART_CLEN_NO NE	8位数据帧中有7个CK脉冲，9位数据帧中有8个CK脉冲
USART_CLEN_EN	8位数据帧中有8个CK脉冲，9位数据帧中有9个CK脉冲
输入参数{in}	
cph	时钟相位
USART_CPH_1CK	在首个时钟边沿采样第一个数据
USART_CPH_2CK	在第二个时钟边沿采样第一个数据
输入参数{in}	
cpl	时钟极性

<i>USART_CPL_LOW</i>	CK引脚不对外发送时保持为低电平
<i>USART_CPL_HIGH</i>	CK引脚不对外发送时保持为高电平
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* configure USART0 synchronous mode parameters */

uart_synchronous_clock_config(USART0,USART_CLEN_EN,USART_CPH_2CK,
USART_CPL_HIGH);
```

函数 **usart_guard_time_config**

函数usart_guard_time_config描述见下表：

表 3-641. 函数 usart_guard_time_config

函数名称	usart_guard_time_config
函数原型	void usart_guard_time_config(uint32_t usart_periph,uint32_t guat);
功能描述	在USART智能卡模式下配置保护时间值
先决条件	-
被调用函数	-
输入参数{in}	
<i>usart_periph</i>	外设USARTx
<i>USARTx</i>	x=0
输入参数{in}	
<i>guat</i>	保护时间值
0-0x000000FF	保护时间值
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* configure USART0 guard time value in smartcard mode */

usart_guard_time_config(USART0, 0x0000 0055);
```

函数 **usart_smartcard_mode_enable**

函数usart_smartcard_mode_enable描述见下表：

表 3-642. 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}	
uart_periph	外设USARTx
USARTx	x=0
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* USART0 smartcard mode enable */
uart_smartcard_mode_enable(USART0);
```

函数 **uart_smartcard_mode_disable**

函数uart_smartcard_mode_disable描述见下表：

表 3-643. 函数 **uart_smartcard_mode_disable**

函数名称	uart_smartcard_mode_disable
函数原型	void uart_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}	
uart_periph	外设USARTx
USARTx	x=0
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* USART0 smartcard mode disable */
uart_smartcard_mode_disable(USART0);
```

函数 **uart_smartcard_mode_nack_enable**

函数uart_smartcard_mode_nack_enable描述见下表：

表 3-644. 函数 **uart_smartcard_mode_nack_enable**

函数名称	uart_smartcard_mode_nack_enable
函数原型	void uart_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}	
uart_periph	外设USARTx
USARTx	x=0
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* enable USART0 NACK in smartcard mode */

uart_smartcard_mode_nack_enable(USART0);
```

函数 **uart_smartcard_mode_nack_disable**

函数uart_smartcard_mode_nack_disable描述见下表：

表 3-645. 函数 **uart_smartcard_mode_nack_disable**

函数名称	uart_smartcard_mode_nack_disable
函数原型	void uart_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}	
uart_periph	外设USARTx
USARTx	x=0
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* disable USART0 NACK in smartcard mode */

uart_smartcard_mode_nack_disable(USART0);
```

函数 **uart_smartcard_mode_early_nack_enable**

函数uart_smartcard_mode_early_nack_enable描述见下表：

表 3-646. 函数 **uart_smartcard_mode_early_nack_enable**

函数名称	uart_smartcard_mode_early_nack_enable
函数原型	void uart_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}	
uart_periph	外设USARTx
USARTx	x=0
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* enable USART0 early NACK in smartcard mode */

uart_smartcard_mode_early_nack_enable (USART0);
```

函数 **uart_smartcard_mode_early_nack_disable**

函数uart_smartcard_mode_early_nack_disable描述见下表：

表 3-647. 函数 **uart_smartcard_mode_early_nack_disable**

函数名称	uart_smartcard_mode_early_nack_disable
函数原型	void uart_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}	
uart_periph	外设USARTx
USARTx	x=0
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* disable USART0 early NACK in smartcard mode */

uart_smartcard_mode_early_nack_disable(USART0);
```

函数 **uart_smartcard_autoretry_config**

函数uart_smartcard_autoretry_config描述见下表：

表 3-648. 函数 **uart_smartcard_autoretry_config**

函数名称	uart_smartcard_autoretry_config
函数原型	void uart_smartcard_autoretry_config(uint32_t usart_periph, uint32_t

	scrnum);
功能描述	配置智能卡自动重试次数
先决条件	-
被调用函数	-
输入参数{in}	
usart_periph	外设USARTx
USARTx	x=0
输入参数{in}	
scrnum	智能卡自动重试次数
0-0x00000007	自动重试次数
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* configure smartcard auto-retry number */
uart_smartcard_autoretry_config (USART0, 0x00000007);
```

函数 usart_block_length_config

函数usart_block_length_config描述见下表：

表 3-649. 函数 usart_block_length_config

函数名称	usart_block_length_config
函数原型	void usart_block_length_config(uint32_t usart_periph, uint32_t bl);
功能描述	配置智能卡T=1的接收时块的长度
先决条件	-
被调用函数	-
输入参数{in}	
usart_periph	外设USARTx
USARTx	x=0
输入参数{in}	
bl	块长度
0-0x000000FF	块长度
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 `uart_irda_mode_enable`

函数`uart_irda_mode_enable`描述见下表：

表 3-650. 函数 `uart_irda_mode_enable`

函数名称	uart_irda_mode_enable	
函数原型	void uart_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 */

uart_irda_mode_enable(USART0);
```

函数 `uart_irda_mode_disable`

函数`uart_irda_mode_disable`描述见下表：

表 3-651. 函数 `uart_irda_mode_disable`

函数名称	uart_irda_mode_disable	
函数原型	void uart_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 usart_prescaler_config

函数usart_prescaler_config描述见下表:

表 3-652. 函数 usart_prescaler_config

函数名称	usart_prescaler_config
函数原型	void usart_prescaler_config(uint32_t usart_periph, uint32_t psc);
功能描述	在USART IrDA低功耗模式下配置外设时钟分频系数
先决条件	-
被调用函数	-
输入参数{in}	
usart_periph	外设USARTx
USARTx	x=0
输入参数{in}	
psc	时钟分频系数
0-0xFF	时钟分频系数
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* configure the USART0 peripheral clock prescaler in USART IrDA low-power mode */
usart_prescaler_config(USART0, 0x00);
```

函数 usart_irda_lowpower_config

函数usart_irda_lowpower_config描述见下表:

表 3-653. 函数 usart_irda_lowpower_config

函数名称	usart_irda_lowpower_config
函数原型	void usart_irda_lowpower_config(uint32_t usart_periph, uint32_t irlp);
功能描述	配置USART IrDA低功耗模式
先决条件	-
被调用函数	-
输入参数{in}	
usart_periph	外设USARTx
USARTx	x=0
输入参数{in}	
irlp	IrDA低功耗模式或正常模式
USART_IRLP_LOW	低功耗模式
USART_IRLP_NORM	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 3-654. 函数 **uart_hardware_flow_rts_config**

函数名称	uart_hardware_flow_rts_config
函数原型	void usart_hardware_flow_rts_config(uint32_t usart_periph, uint32_t rtsconfig);
功能描述	配置USART RTS硬件控制流
先决条件	-
被调用函数	-
输入参数{in}	
usart_periph	外设USARTx
USARTx	x=0,1
输入参数{in}	
rtsconfig	使能/失能RTS
USART_RTS_ENA BLE	使能RTS
USART_RTS_DISA BLE	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 3-655. 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 <i>BLE</i>	使能CTS
USART_CTS_DISA <i>BLE</i>	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_rs485_driver_enable**

函数**uart_rs485_driver_enable**描述见下表：

表 3-656. 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	-
输入参数{in}	
usart_periph	外设USARTx
USARTx	x=0,1
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* enable USART0 RS485 driver */
uart_rs485_driver_enable(USART0);
```

函数 `uart_rs485_driver_disable`

函数`uart_rs485_driver_disable`描述见下表：

表 3-657. 函数 `uart_rs485_driver_disable`

函数名称	uart_rs485_driver_disable	
函数原型	void uart_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USARTTx	
USARTx	x=0,1	
输出参数{out}		
-	-	
返回值		
-	-	

例如：

```
/* disable USART0 RS485 driver */

uart_rs485_driver_disable(USART0);
```

函数 `uart_driver_assertime_config`

函数`uart_driver_assertime_config`描述见下表：

表 3-658. 函数 `uart_driver_assertime_config`

函数名称	uart_driver_assertime_config	
函数原型	void uart_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USARTTx	
USARTx	x=0,1	
输入参数{in}		
deatime	驱动使能置位时间	
0-0x0000001F	驱动使能置位时间	
输出参数{out}		
-	-	
返回值		
-	-	

例如：

```
/* set USART0 driver assertime */

uart_driver_assertime_config(USART0,0x0000001F);
```

函数 **uart_driver_deassertime_config**

函数uart_driver_deassertime_config描述见下表:

表 3-659. 函数 **uart_driver_deassertime_config**

函数名称	uart_driver_deassertime_config
函数原型	void usart_driver_deassertime_config(uint32_t usart_periph, uint32_t dedtime);
功能描述	配置USART驱动使能置低时间
先决条件	-
被调用函数	-
输入参数{in}	
usart_periph	外设USARTx
USARTx	x=0,1
输入参数{in}	
dedtime	驱动使能置低时间
0-0x0000001F	驱动使能置低时间
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* set USART0 driver deassertime */

uart_driver_deassertime_config(USART0,0x0000001F);
```

函数 **uart_depolarity_config**

函数uart_depolarity_config描述见下表:

表 3-660. 函数 **uart_depolarity_config**

函数名称	uart_depolarity_config
函数原型	void usart_depolarity_config(uint32_t usart_periph, uint32_t dep);
功能描述	配置USART驱动使能极性模式
先决条件	-
被调用函数	-
输入参数{in}	
usart_periph	外设USARTx
USARTx	x=0,1
输入参数{in}	
dep	驱动使能的极性选择模式
USART_DEP_HIGH	DE信号高有效

<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 3-661. 函数 `uart_dma_receive_config`

函数名称	uart_dma_receive_config
函数原型	<code>void usart_dma_receive_config(uint32_t usart_periph, uint32_t dmacmd);</code>
功能描述	配置USART DMA接收功能
先决条件	-
被调用函数	-
输入参数{in}	
<code>usart_periph</code>	外设USARTx
<code>USARTx</code>	x=0,1
输入参数{in}	
<code>dmacmd</code>	DMA使能/失能DMA接收功能
<code>USART_DENR_ENABLE</code>	使能DMA接收功能
<code>USART_DENR_DISABLE</code>	失能DMA接收功能
输出参数{out}	
-	-
返回值	
-	-

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 3-662. 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	-
输入参数{in}	
usart_periph	外设USARTx
USARTx	x=0,1
输入参数{in}	
dmacmd	使能/失能DMA发送功能
USART_DENT_EN_ABLE	使能DMA发送功能
USART_DENT_DIS_ABLE	失能DMA发送功能
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* USART0 DMA enable for transmission */

usart_dma_transmit_config(USART0, USART_DENT_ENABLE);
```

函数 **usart_reception_error_dma_disable**

函数usart_reception_error_dma_disable描述见下表：

表 3-663. 函数 **usart_reception_error_dma_disable**

函数名称	usart_reception_error_dma_disable
函数原型	void usart_reception_error_dma_disable (uint32_t usart_periph);
功能描述	USART接收错误时失能DMA
先决条件	-
被调用函数	-
输入参数{in}	
usart_periph	外设USARTx
USARTx	x=0,1
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* disable DMA on reception error */

usart_reception_error_dma_disable (USART0);
```

函数 usart_reception_error_dma_enable

函数usart_reception_error_dma_enable描述见下表:

表 3-664. 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 3-665. 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 3-666. 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 3-667. 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_command_enable**

函数**uart_command_enable**描述见下表：

表 3-668. 函数 **uart_command_enable**

函数名称	uart_command_enable
函数原型	void usart_command_enable(uint32_t usart_periph, uint32_t cmdtype);
功能描述	使能USART请求
先决条件	-
被调用函数	-
输入参数{in}	
usart_periph	外设USARTx
USARTx	x=0,1
输入参数{in}	
cmdtype	请求类型
USART_CMD_ABD CMD	自动波特率检测请求
USART_CMD_SBK CMD	发送断开帧请求
USART_CMD_MM CMD	静模式请求
USART_CMD_RXF CMD	接收数据清空请求
USART_CMD_TXF CMD	发送数据清空请求
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* enable USART0 command */

uart_command_enable(USART0, USART_CMD_ABDCMD);
```

函数 **usart_receive_fifo_enable**

函数usart_receive_fifo_enable描述见下表:

表 3-669. 函数 usart_receive_fifo_enable

函数名称	usart_receive_fifo_enable	
函数原型	void usart_receive_fifo_enable(uint32_t usart_periph);	
功能描述	使能接收FIFO	
先决条件	-	
被调用函数	-	
输入参数{in}		
usart_periph	外设USARTx	
USARTx	x=0,1	
输出参数{out}		
-	-	
返回值		
-	-	

例如:

```
/* enable receive FIFO */

usart_receive_fifo_enable (USART0);
```

函数 **usart_receive_fifo_disable**

函数usart_receive_fifo_disable描述见下表:

表 3-670. 函数 usart_receive_fifo_disable

函数名称	usart_receive_fifo_disable	
函数原型	void usart_receive_fifo_disable(uint32_t usart_periph);	
功能描述	失能接收FIFO	
先决条件	-	
被调用函数	-	
输入参数{in}		
usart_periph	外设USARTx	
USARTx	x=0,1	
输出参数{out}		
-	-	
返回值		
-	-	

例如:

```
/* disable receive FIFO */

usart_receive_fifo_disable(USART0);
```

函数 usart_receive_fifo_counter_number

函数usart_receive_fifo_counter_number描述见下表:

表 3-671. 函数 usart_receive_fifo_counter_number

函数名称	usart_receive_fifo_counter_number
函数原型	uint8_t usart_receive_fifo_counter_number(uint32_t usart_periph);
功能描述	读取接收FIFO计数器的值
先决条件	-
被调用函数	-
输入参数{in}	
usart_periph	外设USARTx
USARTx	x=0,1
输出参数{out}	
-	-
返回值	
uint8_t	接收FIFO计数器的值

例如:

```
/* read receive FIFO counter number */

uint8_t temp;

temp = usart_receive_fifo_counter_number(USART0);
```

函数 usart_flag_get

函数usart_flag_get描述见下表:

表 3-672. 函数 usart_flag_get

函数名称	usart_flag_get
函数原型	FlagStatus usart_flag_get(uint32_t usart_periph, usart_flag_enum flag);
功能描述	获取USART STAT/RFCs寄存器标志位
先决条件	-
被调用函数	-
输入参数{in}	
usart_periph	外设USARTx
USARTx	x=0,1
输入参数{in}	
flag	USART标志位
USART_FLAG_PE RR	校验错误标志
USART_FLAG_FER R	帧错误标志
USART_FLAG_NE	噪声错误标志

<i>RR</i>	
<i>USART_FLAG_OR</i>	
<i>ERR</i>	溢出错误标志
<i>USART_FLAG_IDLE</i>	
<i>E</i>	空闲线检测标志
<i>USART_FLAG_RBNE</i>	读数据缓冲区非空标志
<i>USART_FLAG_TC</i>	发送完成标志
<i>USART_FLAG_TBE</i>	发送数据缓冲区空标志
<i>USART_FLAG_LBD</i>	LIN断开检测标志
<i>USART_FLAG_CTSF</i>	CTS变化标志
<i>USART_FLAG_CTS</i>	CTS电平
<i>USART_FLAG_RT</i>	接收超时标志
<i>USART_FLAG_EB</i>	块结束标志
<i>USART_FLAG_ABDE</i>	自动波特率检测错误
<i>USART_FLAG_AB_D</i>	自动波特率检测标志
<i>USART_FLAG_BSY</i>	忙状态标志
<i>USART_FLAG_AM</i>	ADDR匹配标志
<i>USART_FLAG_SB</i>	断开信号发送标识
<i>USART_FLAG_RWU</i>	接收器从静默模式唤醒
<i>USART_FLAG_WU</i>	从深度睡眠模式唤醒标志
<i>USART_FLAG_TEA</i>	发送使能通知标志
<i>USART_FLAG_REA</i>	接收使能通知标志
<i>USART_FLAG_EPE_RR</i>	校验错误超前检测标志
<i>USART_FLAG_RFE</i>	接收FIFO空标志
<i>USART_FLAG_RFF</i>	接收FIFO满标志
<i>USART_FLAG_RFF_INT</i>	接收FIFO满中断标志
输出参数{out}	
-	-
返回值	
FlagStatus	SET或RESET

例如：

```
/* get flag USART0 state */
```

```
FlagStatus status;
```

```
status = usart_flag_get(USART0,USART_FLAG_TBE);
```

函数 usart_flag_clear

函数usart_flag_clear描述见下表:

表 3-673. 函数 usart_flag_clear

函数名称	usart_flag_clear
函数原型	void usart_flag_clear(uint32_t usart_periph, usart_flag_enum flag);
功能描述	清除USART状态寄存器标志位
先决条件	-
被调用函数	-
输入参数{in}	
usart_periph	外设USARTx
USARTx	x=0,1
输入参数{in}	
flag	USART标志位
USART_FLAG_PE RR	校验错误标志
USART_FLAG_FER R	帧错误标志
USART_FLAG_NE RR	噪声错误标志
USART_FLAG_OR ERR	溢出错误标志
USART_FLAG_IDL E	空闲线检测标志
USART_FLAG_TC	发送完成标志
USART_FLAG_LBD	LIN断开检测标志
USART_FLAG_CTS F	CTS变化标志
USART_FLAG_RT	接收超时标志
USART_FLAG_EB	块结束标志
USART_FLAG_AM	ADDR匹配标志
USART_FLAG_WU	从深度睡眠模式唤醒标志
USART_FLAG_EPE RR	校验错误超前检测标志
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* clear USART0 flag */
```

```
uart_flag_clear(USART0, USART_FLAG_TC);
```

函数 **usart_interrupt_enable**

函数usart_interrupt_enable描述见下表:

表 3-674. 函数 usart_interrupt_enable

函数名称	usart_interrupt_enable
函数原型	void usart_interrupt_enable(uint32_t usart_periph, usart_interrupt_enum interrupt);
功能描述	使能USART中断
先决条件	-
被调用函数	-
输入参数{in}	
usart_periph	外设USARTx
USARTx	x=0,1
输入参数{in}	
interrupt	USART中断
USART_INT_IDLE	IDLE线检测中断
USART_INT_RBNE	读数据缓冲区非空中断和过载错误中断
USART_INT_TC	发送完成中断
USART_INT_TBE	发送缓冲区空中断
USART_INT_PERR	校验错误中断
USART_INT_AM	ADDR匹配中断
USART_INT_RT	接收超时事件中断
USART_INT_EB	块结束事件中断
USART_INT_LBD	LIN断开信号检测中断
USART_INT_ERR	错误中断
USART_INT_CTS	CTS中断
USART_INT_WU	从深度睡眠模式唤醒中断
USART_INT_RFF	接收FIFO满中断
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* enable USART0 TBE interrupt */

usart_interrupt_enable(USART0, USART_INT_TBE);
```

函数 **usart_interrupt_disable**

函数usart_interrupt_disable描述见下表:

表 3-675. 函数 usart_interrupt_disable

函数名称	usart_interrupt_disable
函数原型	void usart_interrupt_disable(uint32_t usart_periph, usart_interrupt_enum interrupt);
功能描述	失能USART中断
先决条件	-
被调用函数	-
输入参数{in}	
usart_periph	外设USARTx
USARTx	x=0,1
输入参数{in}	
interrupt	USART中断
USART_INT_IDLE	IDLE线检测中断
USART_INT_RBNE	读数据缓冲区非空中断和过载错误中断
USART_INT_TC	发送完成中断
USART_INT_TBE	发送缓冲区空中断
USART_INT_PERR	校验错误中断
USART_INT_AM	ADDR匹配中断
USART_INT_RT	接收超时事件中断
USART_INT_EB	块结束事件中断
USART_INT_LBD	LIN断开信号检测中断
USART_INT_ERR	错误中断
USART_INT_CTS	CTS中断
USART_INT_WU	从深度睡眠模式唤醒中断
USART_INT_RFF	接收FIFO满中断
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* disable USART0 TBE interrupt */
usart_interrupt_disable(USART0, USART_INT_TBE);
```

函数 usart_interrupt_flag_get

函数usart_interrupt_flag_get描述见下表：

表 3-676. 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, 参考枚举usart_interrupt_flag_enum
USART_INT_FLAG_EB	块结束事件中断标志
USART_INT_FLAG_RT	超时事件中断标志
USART_INT_FLAG_AM	ADDR匹配中断标志
USART_INT_FLAG_PERR	校验错误中断标志
USART_INT_FLAG_TBE	发送缓冲区空中断标志
USART_INT_FLAG_TC	发送完成中断标志
USART_INT_FLAG_RBNE	读数据缓冲区非空中断标志
USART_INT_FLAG_RBNE_ORERR	读数据缓冲区非空中断和溢出错误中断标志
USART_INT_FLAG_IDLE	IDLE线检测中断标志
USART_INT_FLAG_LBD	LIN断开检测中断标志
USART_INT_FLAG_WU	从深度睡眠模式唤醒中断标志
USART_INT_FLAG_CTS	CTS中断标志
USART_INT_FLAG_ERR_NERR	噪声错误中断标志
USART_INT_FLAG_ERR_ORERR	过载错误中断标志
USART_INT_FLAG_ERR_FERR	帧错误中断标志
USART_INT_FLAG_RFF	接收FIFO满中断标志
输出参数{out}	
-	-
返回值	
FlagStatus	SET或RESET

例如：

```

/* get the USART0 interrupt flag status */

FlagStatus status;

status = usart_interrupt_flag_get(USART0, USART_INT_FLAG_RBNE);
  
```

函数 `usart_interrupt_flag_clear`

函数`usart_interrupt_flag_clear`描述见下表：

表 3-677. 函数 `usart_interrupt_flag_clear`

函数名称	<code>usart_interrupt_flag_clear</code>
函数原型	<code>void usart_interrupt_flag_clear(uint32_t usart_periph, usart_interrupt_flag_enum int_flag);</code>
功能描述	清除USART中断标志位状态
先决条件	-
被调用函数	-
输入参数{in}	
<code>usart_periph</code>	外设USARTx
<code>USARTx</code>	x=0,1
输入参数{in}	
<code>int_flag</code>	USART中断标志
<code>USART_INT_FLAG_PERR</code>	校验错误中断标志
<code>USART_INT_FLAG_ERR_FERR</code>	帧错误中断标志
<code>USART_INT_FLAG_ERR_NERR</code>	噪声错误中断标志
<code>USART_INT_FLAG_RBNE_ORERR</code>	读数据缓冲区非空中断和溢出错误中断标志
<code>USART_INT_FLAG_ERR_ORERR</code>	过载错误中断标志
<code>USART_INT_FLAG_IDLE</code>	IDLE线检测中断标志
<code>USART_INT_FLAG_TC</code>	发送完成中断标志
<code>USART_INT_FLAG_LBD</code>	LIN断开检测中断标志
<code>USART_INT_FLAG_CTS</code>	CTS变化中断标志
<code>USART_INT_FLAG_RT</code>	接收超时事件中断标志
<code>USART_INT_FLAG</code>	块结束事件中断标志

<i>_EB</i>	
<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
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* clear the USART0 interrupt flag */

uart_interrupt_flag_clear(USART0, USART_INT_FLAG_TC);
```

3.24. WWDGT

窗口看门狗定时器(WWDGT)用来监测由软件故障导致的系统故障。章节[3.24.1](#)描述了WWDGT的寄存器列表，章节[3.24.2](#)对WWDGT库函数进行说明。

3.24.1. 外设寄存器说明

WWDGT寄存器列表如下表所示：

表 3-678. WWDGT 寄存器

寄存器名称	寄存器描述
WWDGT_CTL	WWDGT控制寄存器
WWDGT_CFG	WWDGT配置寄存器
WWDGT_STAT	WWDGT状态寄存器

3.24.2. 外设库函数说明

WWDGT库函数列表如下表所示：

表 3-679. 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

库函数名称	库函数说明
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 3-680. 函数 wwdgt_deinit

函数名称	wwdgt_deinit
函数原型	void wwdgt_deinit(void);
功能描述	将WWDGT寄存器重设为缺省值
先决条件	-
输入参数{in}	
-	-
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* reset the window watchdog timer configuration */

wwdgt_deinit();
```

函数 wwdgt_enable

函数wwdgt_enable描述见下表:

表 3-681. 函数 wwdgt_enable

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

例如:

```
/* start the WWDGT counter */

wwdgt_enable();
```

函数 wwdgt_counter_update

函数wwdgt_counter_update描述见下表:

表 3-682. 函数 wwdgt_counter_update

函数名称	wwdgt_counter_update
函数原型	void wwdgt_counter_update(uint16_t counter_value);
功能描述	设置WWDGT计数器更新值
先决条件	-
输入参数{in}	
counter_value	计数器值, 数值范围为0x00 - 0x7F
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* update WWDGT counter to 0x7F */

wwdgt_counter_update (127);
```

函数 wwdgt_config

函数wwdgt_config描述见下表:

表 3-683. 函数 wwdgt_config

函数名称	wwdgt_config
函数原型	void wwdgt_config(uint16_t counter, uint16_t window, uint32_t prescaler);
功能描述	设置WWDGT计数器值、窗口值和预分频值
先决条件	-
输入参数{in}	
counter	定时器计数值, 数值范围0x00 - 0x7F
输入参数{in}	
window	窗口值, 数值范围0x00 - 0x7F
输入参数{in}	
prescaler	WWDGT预分频值
WWDGT_CFG_PSC_DIV1	WWDGT计数器时钟为 (PCLK/4096) /1
WWDGT_CFG_PSC_DIV2	WWDGT计数器时钟为 (PCLK/4096) /2
WWDGT_CFG_PSC_DIV4	WWDGT计数器时钟为 (PCLK/4096) /4
WWDGT_CFG_PSC_DIV8	WWDGT计数器时钟为 (PCLK/4096) /8
输出参数{out}	

-	-
Return value	
-	-

例如：

```
/* 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 3-684. 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	-
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 3-685. 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	-
输入参数{in}	
-	-
输出参数{out}	
-	-
返回值	
FlagStatus	SET or RESET

例如：

```
/* test if the counter value update has reached the 0x40 */  
  
FlagStatus status;  
  
status = wwdgt_flag_get ( );
```

函数 wwdgt_flag_clear

函数wwdgt_flag_clear描述见下表：

表 3-686. 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WWDGTR提前唤醒中断标志位状态
先决条件	-
输入参数{in}	
-	-
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* clear early wakeup interrupt state of WWDGT */  
  
wwdgt_flag_clear ( );
```

3.25. USBFS

USBFS 的固件库函数介绍请参考文档 **GD32F3x0-Firmware-Library-USB User Manual_V1.0.**

4. 版本历史

表 4-1. 版本历史

版本号.	说明	日期
1.0	初稿发布	2019 年 6 月 1 日
1.1	1. 修改章节 3.10.2 与 3.15.2 的库函数表格 2. 章节 3.17.2 增加函数 <code>rcu_deepsleep_voltage_set</code> 的输入变量 <code>dsvol</code> 的描述	2021 年 6 月 1 日
1.2	1. 新增 GD32F310 系列产品	2022 年 1 月 6 日

Important Notice

This document is the property of GigaDevice Semiconductor Inc. and its subsidiaries (the "Company"). This document, including any product of the Company described in this document (the "Product"), is owned by the Company under the intellectual property laws and treaties of the People's Republic of China and other jurisdictions worldwide. The Company reserves all rights under such laws and treaties and does not grant any license under its patents, copyrights, trademarks, or other intellectual property rights. The names and brands of third party referred thereto (if any) are the property of their respective owner and referred to for identification purposes only.

The Company makes no warranty of any kind, express or implied, with regard to this document or any Product, including, but not limited to, the implied warranties of merchantability and fitness for a particular purpose. The Company does not assume any liability arising out of the application or use of any Product described in this document. Any information provided in this document is provided only for reference purposes. It is the responsibility of the user of this document to properly design, program, and test the functionality and safety of any application made of this information and any resulting product. Except for customized products which has been expressly identified in the applicable agreement, the Products are designed, developed, and/or manufactured for ordinary business, industrial, personal, and/or household applications only. The Products are not designed, intended, or authorized for use as components in systems designed or intended for the operation of weapons, weapons systems, nuclear installations, atomic energy control instruments, combustion control instruments, airplane or spaceship instruments, transportation instruments, traffic signal instruments, life-support devices or systems, other medical devices or systems (including resuscitation equipment and surgical implants), pollution control or hazardous substances management, or other uses where the failure of the device or Product could cause personal injury, death, property or environmental damage ("Unintended Uses"). Customers shall take any and all actions to ensure using and selling the Products in accordance with the applicable laws and regulations. The Company is not liable, in whole or in part, and customers shall and hereby do release the Company as well as its suppliers and/or distributors from any claim, damage, or other liability arising from or related to all Unintended Uses of the Products. Customers shall indemnify and hold the Company as well as its suppliers and/or distributors harmless from and against all claims, costs, damages, and other liabilities, including claims for personal injury or death, arising from or related to any Unintended Uses of the Products.

Information in this document is provided solely in connection with the Products. The Company reserves the right to make changes, corrections, modifications or improvements to this document and Products and services described herein at any time, without notice.