GigaDevice Semiconductor Inc.

GD32E11x
ARM® Cortex®-M4 32-bit MCU

固件库

使用指南

1.5 版本

(2026 年 4 月)

目录

目录 2
图索引 5
表索引 6
1. 介绍 20
1.1. 文档和固件库规则 20
1.1.1. 外设缩写 20
1.1.2. 命名规则 21
2. 固件库概述 22
2.1. 文件组织结构 22
2.1.1. Examples 文件夹 23
2.1.2. Firmware 文件夹 23
2.1.3. Template 文件夹 23
2.1.4. Utilities 文件夹 26
2.2. 固件库文件描述 26
3. 外设固件库 28
3.1. 外设固件库概述 28
3.2. ADC 28
3.2.1. 外设寄存器描述 28
3.2.2. 外设库函数说明 29
3.3. BKP 58
3.3.1. 外设寄存器说明 58
3.3.2. 外设库函数说明 58
3.4. CRC 71
3.4.1. 外设寄存器说明 71
3.4.2. 外设库函数说明 71
3.5. CTC 75
3.5.1. 外设寄存器说明 75
3.5.2. 外设库函数说明 76
3.6. DAC 89

3.6.1. 外设寄存器说明 89
3.6.2. 外设库函数说明 90
3.7. DBG 105
3.7.1. 外设寄存器说明 105
3.7.2. 外设库函数说明 105
3.8. DMA 112
3.8.1. 外设寄存器说明 112
3.8.2. 外设库函数说明 112
3.9. EXMC 133
3.9.1. 外设寄存器说明 133
3.9.2. 外设库函数说明 133
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 147
3.12. FWDGT 167
3.12.1. 外设寄存器说明 168
3.12.2. 外设库函数说明 168
3.13. GPIO 173
3.13.1. 外设寄存器说明 173
3.13.2. 外设库函数说明 174
3.14. I2C 188
3.14.1. 外设寄存器说明 188
3.14.2. 外设库函数说明 188
3.15. MISC 215
3.15.1. 外设寄存器说明 215
3.15.2. 外设库函数说明 216
3.16. PMU 223
3.16.1. 外设寄存器说明 223
3.16.2. 外设库函数说明 223
3.17. RCU 231
3.17.1. 外设寄存器说明 231
3.17.2. 外设库函数说明 232
3.18. RTC 265
3.18.1. 外设寄存器描述 265
3.18.2. 外设库函数描述 265
3.19. SPI 275
3.19.1. 外设寄存器说明 275
3.19.2. 外设库函数说明 275
3.20. TIMER 301
3.20.1. 外设寄存器说明 301

3.20.2. 外设库函数说明 302
3.21. USART 360
3.21.1. 外设寄存器说明 360
3.21.2. 外设库函数说明 360
3.22. WWDGT 396
3.22.1. 外设寄存器说明 396
3.22.2. 外设库函数说明 397
4. 版本历史 402

图索引

图 2-1. GD32E11x 固件库文件组织结构 22
图 2-2. 选择外设例程文件 24
图 2-3. 拷贝外设例程文件 25
图 2-4. 打开工程文件 25
图 2-5. 编译调试下载 26

表索引

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

表 3-38. 函数 adc_interrupt_flag_get 55
表 3-39. 函数 adc_interrupt_flag_clear 56
表 3-40. 函数 adc_interrupt_enable 57
表 3-41. 函数 adc_interrupt_disable 57
表 3-42. BKP 寄存器 58
表 3-43. BKP 库函数 59
表 3-44. 函数 bkp_deinit 59
表 3-45. 函数 bkp_data_write 60
表 3-46. 函数 bkp_data_read 60
表 3-47. 函数 bkp_rtc_calibration_output_enable 61
表 3-48. 函数 bkp_rtc_calibration_output_disable 61
表 3-49. 函数 bkp_rtc_signal_output_enable 62
表 3-50. 函数 bkp_rtc_signal_output_disable 62
表 3-51. 函数 bkp_rtc_output_select 63
表 3-52. 函数 bkp_rtc_clock_output_select 63
表 3-53. 函数 bkp_rtc_clock_calibration_direction_select 64
表 3-54. 函数 bkp_rtc_calibration_value_set 65
表 3-55. 函数 bkp_tamper_detection_enable 65
表 3-56. 函数 bkp_tamper_detection_disable 66
表 3-57. 函数 bkp_tamper_active_level_set 66
表 3-58. 函数 bkp_interrupt_enable 67
表 3-59. 函数 bkp_interrupt_disable 67
表 3-60. 函数 bkp_flag_get 68
表 3-61. 函数 bkp_flag_clear 68
表 3-62. 函数 bkp_interrupt_flag_get 69
表 3-63. 函数 bkp_interrupt_flag_clear 69
表 3-64. CRC 寄存器 71
表 3-65. CRC 库函数 71
表 3-66. 函数 crc_deinit 71
表 3-67. 函数 crc_data_register_reset 72
表 3-68. 函数 crc_data_register_read 72
表 3-69. 函数 crc_ free_data_register_read 73
表 3-70. 函数 crc_ free_data_register_write 73
表 3-71. 函数 crc_single_data_calculate 74
表 3-72. 函数 crc_block_data_calculate 74
表 3-73. CTC 寄存器 75
表 3-74. CTC 库函数 76
表 3-75. 函数 ctc_deinit 76
表 3-76. 函数 ctc_counter_enable 77
表 3-77. 函数 ctc_counter_disable 77
表 3-78. 函数 ctc_irc48m_trim_value_config 78
表 3-79. 函数 ctc_software_refsource_pulse_generate 78
表 3-80. 函数 ctc_hardware_trim_mode_config 79

表 3-81. 函数 ctc_refsource_polarity_config 80
表 3-82. 函数 ctc_refsource_signal_select 80
表 3-83. 函数 ctc_refsource_prescaler_config 81
表 3-84. 函数 ctc_clock_limit_value_config 82
表 3-85. 函数 ctc_counter_reload_value_config 82
表 3-86. 函数 ctc_counter_capture_value_read 83
表 3-87. 函数 ctc_counter_direction_read 83
表 3-88. 函数 ctc_counter_reload_value_read 84
表 3-89. 函数 ctc_irc48m_trim_value_read 84
表 3-90. 函数 ctc_interrupt_enable 85
表 3-91. 函数 ctc_interrupt_disable 86
表 3-92. 函数 ctc_interrupt_flag_get 86
表 3-93. 函数 ctc_interrupt_flag_clear 87
表 3-94. 函数 ctc_flag_get 88
表 3-95. 函数 ctc_flag_clear 89
表 3-96. DAC 寄存器 90
表 3-97. DAC 库函数 90
表 3-98. 函数 dac_deinit 91
表 3-99. 函数 dac_enable 91
表 3-100. 函数 dac_disable 92
表 3-101. 函数 dac_dma_enable 92
表 3-102. 函数 dac_dma_disable 93
表 3-103. 函数 dac_output_buffer_enable 94
表 3-104. 函数 dac_output_buffer_disable 94
表 3-105. 函数 dac_output_value_get 95
表 3-106. 函数 dac_data_set 95
表 3-107. 函数 dac_trigger_enable 96
表 3-108. 函数 dac_trigger_disable 97
表 3-109. 函数 dac_trigger_source_config 97
表 3-110. 函数 dac_software_trigger_enable 99
表 3-111. 函数 dac_wave_mode_config 99
表 3-112. 函数 dac_lfsr_noise_config 100
表 3-113. 函数 dac_triangle_noise_config 101
表 3-114. 函数 dac_concurrent_enable 101
表 3-115. 函数 dac_concurrent_disable 102
表 3-116. 函数 dac_concurrent_software_trigger_enable 102
表 3-117. 函数 dac_concurrent_output_buffer_enable 103
表 3-118. 函数 dac_concurrent_output_buffer_disable 103
表 3-119. 函数 dac_concurrent_data_set 104
表 3-120. DBG 寄存器 105
表 3-121. DBG 库函数 105
表 3-122. 枚举类型 dbg_periph_enum 106
表 3-123. 函数 dbg_id_get 106

表 3-124. 函数 dbg_low_power_enable 107
表 3-125. 函数 dbg_low_power_disable 108
表 3-126. 函数 dbg_periph_enable 109
表 3-127. 函数 dbg_periph_disable 110
表 3-128. 函数 dbg_trace_pin_enable 110
表 3-129. 函数 dbg_trace_pin_disable 111
表 3-130. DMA 寄存器 112
表 3-131. DMA 库函数 112
表 3-132. 结构体 dma_parameter_struct 113
表 3-133. 函数 dma_deinit 113
表 3-134. 函数 dma_struct_para_init 114
表 3-135. 函数 dma_init 114
表 3-136. 函数 dma_circulation_enable 115
表 3-137. 函数 dma_circulation_disable 116
表 3-138. 函数 dma_memory_to_memory_enable 117
表 3-139. 函数 dma_memory_to_memory_disable 117
表 3-140. 函数 dma_channel_enable 118
表 3-141. 函数 dma_channel_disable 119
表 3-142. 函数 dma_periph_address_config 119
表 3-143. 函数 dma_memory_address_config 120
表 3-144. 函数 dma_transfer_number_config 121
表 3-145. 函数 dma_transfer_number_get 122
表 3-146. 函数 dma_priority_config 122
表 3-147. 函数 dma_memory_width_config 123
表 3-148. 函数 dma_periph_width_config 124
表 3-149. 函数 dma_memory_increase_enable 125
表 3-150. 函数 dma_memory_increase_disable 125
表 3-151. 函数 dma_periph_increase_enable 126
表 3-152. 函数 dma_periph_increase_disable 127
表 3-153. 函数 dma_transfer_direction_config 127
表 3-154. 函数 dma_flag_get 128
表 3-155. 函数 dma_flag_clear 129
表 3-156. 函数 dma_interrupt_flag_get 130
表 3-157. 函数 dma_interrupt_flag_clear 130
表 3-158. 函数 dma_interrupt_enable 131
表 3-159. 函数 dma_interrupt_disable 132
表 3-160. EXMC 寄存器 133
表 3-161. EXMC 库函数 133
表 3-162. 结构体 exmc_norsram_timing_parameter_struct 134
表 3-163. 结构体 exmc_norsram_parameter_struct 134
表 3-164. 函数 exmc_norsram_deinit 134
表 3-165. 函数 exmc_norsram_init 135
表 3-166. 函数 exmc_norsram_struct_para_init 136

表 3-167. 函数 exmc_norsram_enable 137
表 3-168. 函数 exmc_norsram_disable 137
表 3-169. 函数 exmc_norsram_page_size_config 138
表 3-170. EXTI 寄存器 139
表 3-171. EXTI 库函数 139
表 3-172. 枚举类型 exti_line_enum 140
表 3-173. 枚举类型 exti_mode_enum 140
表 3-174. 枚举类型 exti_trig_type_enum 140
表 3-175. 函数 exti_deinit 141
表 3-176. 函数 exti_init 141
表 3-177. 函数 exti_interrupt_enable 142
表 3-178. 函数 exti_interrupt_disable 142
表 3-179. 函数 exti_event_enable 143
表 3-180. 函数 exti_event_disable 143
表 3-181. 函数 exti_software_interrupt_enable 144
表 3-182. 函数 exti_software_interrupt_disable 144
表 3-183. 函数 exti_flag_get 145
表 3-184. 函数 exti_flag_clear 145
表 3-185. 函数 exti_interrupt_flag_get 146
表 3-186. 函数 exti_interrupt_flag_clear 146
表 3-187. FMC 寄存器 147
表 3-188. FMC 固件库函数 147
表 3-189. 枚举类型 fmc_state_enum 148
表 3-190. 函数 fmc_wscnt_set 149
表 3-191. 函数 fmc_prefetch_enable 149
表 3-192. 函数 fmc_prefetch_disable 150
表 3-193. 函数 fmc_ibus_enable 150
表 3-194. 函数 fmc_ibus_disable 151
表 3-195. 函数 fmc_dbus_enable 151
表 3-196. 函数 fmc_dbus_disable 152
表 3-197. 函数 fmc_ibus_reset 152
表 3-198. 函数 fmc_dbus_reset 153
表 3-199. 函数 fmc_program_width_set 153
表 3-200. 函数 fmc_unlock 154
表 3-201. 函数 fmc_lock 154
表 3-202. 函数 fmc_page_erase 155
表 3-203. 函数 fmc_mass_erase 155
表 3-204. 函数 fmc_doubleword_program 156
表 3-205. 函数 fmc_word_program 156
表 3-206. 函数 ob_unlock 157
表 3-207. 函数 ob_lock 157
表 3-208. 函数 ob_erase 158
表 3-209. 函数 ob_write_protection_enable 158

表 3-210. 函数 ob_security_protection_config 159
表 3-211. 函数 ob_user_write 159
表 3-212. 函数 ob_data_program 160
表 3-213. 函数 ob_user_get 161
表 3-214. 函数 ob_data_get 161
表 3-215. 函数 ob_write_protection_get 162
表 3-216. 函数 ob_security_protection_flag_get 162
表 3-217. 函数 fmc_interrupt_enable 163
表 3-218. 函数 fmc_interrupt_disable 163
表 3-219. 函数 fmc_flag_get 164
表 3-220. 函数 fmc_flag_clear 164
表 3-221. 函数 fmc_interrupt_flag_get 165
表 3-222. 函数 fmc_interrupt_flag_clear 166
表 3-223. 函数 fmc_state_get 166
表 3-224. 函数 fmc_ready_wait 167
表 3-225. FWDGT 寄存器 168
表 3-226. FWDGT 库函数 168
表 3-227. 函数 fwdgt_write_enable 168
表 3-228. 函数 fwdgt_write_disable 169
表 3-229. 函数 fwdgt_enable 169
表 3-230. 函数 fwdgt_prescaler_value_config 170
表 3-231. 函数 fwdgt_reload_value_config 171
表 3-232. 函数 fwdgt_counter_reload 171
表 3-233. 函数 fwdgt_config 172
表 3-234. 函数 fwdgt_flag_get 172
表 3-235. GPIO 寄存器 173
表 3-236. GPIO 库函数 174
表 3-237. 函数 gpio_deinit 175
表 3-238. 函数 gpio_afio_deinit 175
表 3-239. 函数 gpio_init 176
表 3-240. 函数 gpio_bit_set 177
表 3-241. 函数 gpio_bit_reset 177
表 3-242. 函数 gpio_bit_write 178
表 3-243. 函数 gpio_port_write 179
表 3-244. 函数 gpio_input_bit_get 179
表 3-245. 函数 gpio_input_port_get 180
表 3-246. 函数 gpio_output_bit_get 181
表 3-247. 函数 gpio_output_port_get 181
表 3-248. 函数 gpio_pin_remap_config 182
表 3-249. 函数 gpio_exti_source_select 184
表 3-250. 函数 gpio_event_output_config 184
表 3-251. 函数 gpio_event_output_enable 185
表 3-252. 函数 gpio_event_output_disable 185

表 3-253. 函数 gpio_pin_lock 186
表 3-254. 函数 gpio_compensation_config 186
表 3-255. 函数 gpio_compensation_flag_get 187
表 3-256. I2C 寄存器 188
表 3-257. I2C 库函数 188
表 3-258. 枚举类型 i2c_flag_enum 189
表 3-259. 枚举类型 i2c_interrupt_flag_enum 190
表 3-260. 枚举类型 i2c_interrupt_enum 191
表 3-261. 函数 i2c_deinit 191
表 3-262. 函数 i2c_clock_config 191
表 3-263. 函数 i2c_mode_addr_config 192
表 3-264. 函数 i2c_smbus_type_config 193
表 3-265. 函数 i2c_ack_config 194
表 3-266. 函数 i2c_ackpos_config 194
表 3-267. 函数 i2c_master_addressing 195
表 3-268. 函数 i2c_dualaddr_enable 196
表 3-269. 函数 i2c_dualaddr_disable 196
表 3-270. 函数 i2c_enable 197
表 3-271. 函数 i2c_disable 197
表 3-272. 函数 i2c_start_on_bus 198
表 3-273. 函数 i2c_stop_on_bus 198
表 3-274. 函数 i2c_data_transmit 199
表 3-275. 函数 i2c_data_receive 199
表 3-276. 函数 i2c_dma_config 200
表 3-277. 函数 i2c_dma_last_transfer_config 201
表 3-278. 函数 i2c_stretch_scl_low_config 201
表 3-279. 函数 i2c_slave_response_to_gcall_config 202
表 3-280. 函数 i2c_software_reset_config 203
表 3-281. 函数 i2c_pec_config 203
表 3-282. 函数 i2c_pec_transfer_config 204
表 3-283. 函数 i2c_pec_value_get 204
表 3-284. 函数 i2c_smbus_alert_config 205
表 3-285. 函数 i2c_smbus_arp_config 206
表 3-286. 函数 i2c_sam_enable 206
表 3-287. 函数 i2c_sam_disable 207
表 3-288. 函数 i2c_sam_timeout_enable 207
表 3-289. 函数 i2c_sam_timeout_disable 208
表 3-290. 函数 i2c_flag_get 208
表 3-291. 函数 i2c_flag_clear 210
表 3-292. 函数 i2c_interrupt_enable 211
表 3-293. 函数 i2c_interrupt_disable 212
表 3-294. 函数 i2c_interrupt_flag_get 212
表 3-295. 函数 i2c_interrupt_flag_clear 214

表 3-296. NVIC 寄存器 215
表 3-297. Systick 寄存器 216
表 3-298. 枚举类型 IRQn_Type 216
表 3-299. MISC 库函数 218
表 3-300. 函数 nvic_priority_group_set 218
表 3-301. 函数 nvic_irq_enable 219
表 3-302. 函数 nvic_irq_disable 220
表 3-303. 函数 nvic_vector_table_set 220
表 3-304. 函数 system_lowpower_set 221
表 3-305. 函数 system_lowpower_reset 222
表 3-306. 函数 systick_clksource_set 222
表 3-307. PMU 寄存器 223
表 3-308. PMU 库函数 223
表 3-309. 函数 pmu_deinit 224
表 3-310. 函数 pmu_lvd_select 224
表 3-311. 函数 pmu_ldo_output_select 225
表 3-312. 函数 pmu_lvd_disable 226
表 3-313. 函数 pmu_to_sleepmode 226
表 3-314. 函数 pmu_to_deepsleepmode 227
表 3-315. 函数 pmu_to_standbymode 227
表 3-316. 函数 pmu_wakeup_pin_enable 228
表 3-317. 函数 pmu_wakeup_pin_disable 228
表 3-318. 函数 pmu_backup_write_enable 229
表 3-319. 函数 pmu_backup_write_disable 229
表 3-320. 函数 pmu_flag_get 230
表 3-321. 函数 pmu_flag_clear 231
表 3-322. RCU 寄存器 231
表 3-323. RCU 库函数 232
表 3-324. 枚举类型 rcu_periph_enum 233
表 3-325. 枚举类型 rcu_periph_sleep_enum 234
表 3-326. 枚举类型 rcu_periph_reset_enum 234
表 3-327. 枚举类型 rcu_flag_enum 236
表 3-328. 枚举类型 rcu_int_flag_enum 236
表 3-329. 枚举类型 rcu_int_flag_clear_enum 237
表 3-330. 枚举类型 rcu_int_enum 237
表 3-331. 枚举类型 rcu_osci_type_enum 238
表 3-332. 枚举类型 rcu_clock_freq_enum 238
表 3-333. 函数 rcu_deinit 238
表 3-334. 函数 rcu_periph_clock_enable 239
表 3-335. 函数 rcu_periph_clock_disable 239
表 3-336. 函数 rcu_periph_clock_sleep_enable 240
表 3-337. 函数 rcu_periph_clock_sleep_disable 240
表 3-338. 函数 rcu_periph_reset_enable 241

表 3-339. 函数 rcu_periph_reset_disable 241
表 3-340. 函数 rcu_bkp_reset_enable 242
表 3-341. 函数 rcu_bkp_reset_disable 242
表 3-342. 函数 rcu_system_clock_source_config 243
表 3-343. 函数 rcu_system_clock_source_get 243
表 3-344. 函数 rcu_ahb_clock_config 244
表 3-345. 函数 rcu_apb1_clock_config 244
表 3-346. 函数 rcu_apb2_clock_config 245
表 3-347. 函数 rcu_ckout0_config 246
表 3-348. 函数 rcu_pll_config 247
表 3-349. 函数 rcu_pllpresel_config 248
表 3-350. 函数 rcu_predv0_config 248
表 3-351. 函数 rcu_predv1_config 249
表 3-352. 函数 rcu_pll1_config 249
表 3-353. 函数 rcu_pll2_config 250
表 3-354. 函数 rcu_adc_clock_config 250
表 3-355. 函数 rcu_usb_clock_config 251
表 3-356. 函数 rcu_rtc_clock_config 252
表 3-357. 函数 rcu_i2s1_clock_config 253
表 3-358. 函数 rcu_i2s2_clock_config 254
表 3-359. 函数 rcu_ck48m_clock_config 254
表 3-360. 函数 rcu_flag_get 255
表 3-361. 函数 rcu_all_reset_flag_clear 255
表 3-362. 函数 rcu_interrupt_flag_get 256
表 3-363. 函数 rcu_interrupt_flag_clear 256
表 3-364. 函数 rcu_interrupt_enable 257
表 3-365. 函数 rcu_interrupt_disable 258
表 3-366. 函数 rcu_lxtal_drive_capability_config 258
表 3-367. 函数 rcu_osci_stab_wait 259
表 3-368. 函数 rcu_osci_on 260
表 3-369. 函数 rcu_osci_off 260
表 3-370. 函数 rcu_osci_bypass_mode_enable 261
表 3-371. 函数 rcu_osci_bypass_mode_disable 261
表 3-372. 函数 rcu_hxtal_clock_monitor_enable 262
表 3-373. 函数 rcu_hxtal_clock_monitor_disable 262
表 3-374. 函数 rcu_irc8m_adjust_value_set 263
表 3-375. 函数 rcu_deepsleep_voltage_set 263
表 3-376. 函数 rcu_clock_freq_get 264
表 3-377. RTC 寄存器 265
表 3-378. RTC 库函数 265
表 3-379. 函数 rtc_configuration_mode_enter 266
表 3-380. 函数 rtc_configuration_mode_exit 266
表 3-381. 函数 rtc_counter_set 267

表 3-382. 函数 rtc_prescaler_set 267
表 3-383. 函数 rtc_lwoff_wait 268
表 3-384. 函数 rtc_register_sync_wait 268
表 3-385. 函数 rtc_alarm_config 269
表 3-386. 函数 rtc_counter_get 270
表 3-387. 函数 rtc_divider_get 270
表 3-388. 函数 rtc_flag_get 271
表 3-389. 函数 rtc_flag_clear 271
表 3-390. 函数 rtc_flag_get 272
表 3-391. 函数 rtc_interrupt_flag_clear 273
表 3-392. 函数 rtc_interrupt_enable 273
表 3-393. 函数 rtc_interrupt_disable 274
表 3-394. SPI/I2S 寄存器 275
表 3-395. SPI/I2S 库函数 275
表 3-396. 结构体 spi_parameter_struct 276
表 3-397. 函数 spi_i2s_deinit 277
表 3-398. 函数 spi_struct_para_init 277
表 3-399. 函数 spi_init 278
表 3-400. 函数 spi_enable 279
表 3-401. 函数 spi_disable 279
表 3-402. 函数 i2s_init 280
表 3-403. 函数 i2s_psc_config 281
表 3-404. 函数 i2s_enable 282
表 3-405. 函数 i2s_disable 283
表 3-406. 函数 spi_nss_output_enable 283
表 3-407. 函数 spi_nss_output_disable 284
表 3-408. 函数 spi_nss_internal_high 284
表 3-409. 函数 spi_nss_internal_low 285
表 3-410. 函数 spi_dma_enable 285
表 3-411. 函数 spi_dma_disable 286
表 3-412. 函数 spi_i2s_data_frame_format_config 287
表 3-413. 函数 spi_i2s_data_transmit 288
表 3-414. 函数 spi_i2s_data_receive 288
表 3-415. 函数 spi_bidirectional_transfer_config 289
表 3-416. 函数 spi_crc_polynomial_set 289
表 3-417. 函数 spi_crc_polynomial_get 290
表 3-418. 函数 spi_crc_on 291
表 3-419. 函数 spi_crc_off 291
表 3-420. 函数 spi_crc_next 292
表 3-421. 函数 spi_crc_get 292
表 3-422. 函数 spi_ti_mode_enable 293
表 3-423. 函数 spi_ti_mode_disable 293
表 3-424. 函数 spi_nssp_mode_enable 294

表 3-425. 函数 spi_nssp_mode_disable 295
表 3-426. 函数 spi_quad_enable 295
表 3-427. 函数 spi_quad_disable 296
表 3-428. 函数 spi_quad_write_enable 296
表 3-429. 函数 spi_quad_read_enable 297
表 3-430. 函数 spi_i2s_interrupt_enable 297
表 3-431. 函数 spi_i2s_interrupt_disable 298
表 3-432. 函数 spi_i2s_interrupt_flag_get 299
表 3-433. 函数 spi_i2s_flag_get 300
表 3-434. 函数 spi_crc_error_clear 301
表 3-435. TIMER 寄存器 301
表 3-436. TIMER 库函数 302
表 3-437. 结构体 timer_parameter_struct 304
表 3-438. 结构体 timer_break_parameter_struct 305
表 3-439. 结构体 timer_oc_parameter_struct 305
表 3-440. 结构体 timer_ic_parameter_struct 305
表 3-441. 函数 timer_deinit 306
表 3-442. 函数 timer_struct_para_init 306
表 3-443. 函数 timer_init 307
表 3-444. 函数 timer_enable 308
表 3-445. 函数 timer_disable 308
表 3-446. 函数 timer_auto_reload_shadow_enable 309
表 3-447. 函数 timer_auto_reload_shadow_disable 309
表 3-448. 函数 timer_update_event_enable 310
表 3-449. 函数 timer_update_event_disable 310
表 3-450. 函数 timer_counter_alignment 311
表 3-451. 函数 timer_counter_up_direction 312
表 3-452. 函数 timer_counter_down_direction 312
表 3-453. 函数 timer_prescaler_config 313
表 3-454. 函数 timer_repetition_value_config 313
表 3-455. 函数 timer_autoreload_value_config 314
表 3-456. 函数 timer_counter_value_config 315
表 3-457. 函数 timer_counter_read 315
表 3-458. 函数 timer_prescaler_read 316
表 3-459. 函数 timer_single_pulse_mode_config 316
表 3-460. 函数 timer_update_source_config 317
表 3-461. 函数 timer_dma_enable 318
表 3-462. 函数 timer_dma_disable 319
表 3-463. 函数 timer_channel_dma_request_source_select 319
表 3-464. 函数 timer_dma_transfer_config 320
表 3-465. 函数 timer_event_software_generate 322
表 3-466. 函数 timer_break_struct_para_init 323
表 3-467. 函数 timer_break_config 323

表 3-468. 函数 timer_break_enable 324
表 3-469. 函数 timer_break_disable 325
表 3-470. 函数 timer_automatic_output_enable 325
表 3-471. 函数 timer_automatic_output_disable 326
表 3-472. 函数 timer_primary_output_config 326
表 3-473. 函数 timer_channel_control_shadow_config 327
表 3-474. 函数 timer_channel_control_shadow_update_config 328
表 3-475. 函数 timer_channel_output_struct_para_init 328
表 3-476. 函数 timer_channel_output_config 329
表 3-477. 函数 timer_channel_output_mode_config 330
表 3-478. 函数 timer_channel_output_pulse_value_config 331
表 3-479. 函数 timer_channel_output_shadow_config 332
表 3-480. 函数 timer_channel_output_fast_config 333
表 3-481. 函数 timer_channel_output_clear_config 333
表 3-482. 函数 timer_channel_output_polarity_config 334
表 3-483. 函数 timer_channel_complementary_output_polarity_config 335
表 3-484. 函数 timer_channel_output_state_config 336
表 3-485. 函数 timer_channel_complementary_output_state_config 337
表 3-486. 函数 timer_channel_input_struct_para_init 338
表 3-487. 函数 timer_input_capture_config 338
表 3-488. 函数 timer_channel_input_capture_prescaler_config 339
表 3-489. 函数 timer_channel_capture_value_register_read 340
表 3-490. 函数 timer_input_pwm_capture_config 341
表 3-491. 函数 timer_hall_mode_config 342
表 3-492. 函数 timer_input_trigger_source_select 342
表 3-493. 函数 timer_master_output_trigger_source_select 343
表 3-494. 函数 timer_slave_mode_select 344
表 3-495. 函数 timer_master_slave_mode_config 345
表 3-496. 函数 timer_external_trigger_config 346
表 3-497. 函数 timer_quadrature_decoder_mode_config 347
表 3-498. 函数 timer_internal_clock_config 348
表 3-499. 函数 timer_internal_trigger_as_external_clock_config 349
表 3-500. 函数 timer_external_trigger_as_external_clock_config 350
表 3-501. 函数 timer_external_clock_mode0_config 351
表 3-502. 函数 timer_external_clock_mode1_config 352
表 3-503. 函数 timer_external_clock_mode1_disable 353
表 3-504. 函数 timer_write_chxval_register_config 353
表 3-505. 函数 timer_output_value_selection_config 354
表 3-506. 函数 timer_interrupt_enable 354
表 3-507. 函数 timer_interrupt_disable 355
表 3-508. 函数 timer_interrupt_flag_get 356
表 3-509. 函数 timer_interrupt_flag_clear 357
表 3-510. 函数 timer_flag_get 358

表 3-511. 函数 timer_flag_clear 359
表 3-512. USART 寄存器 360
表 3-513. USART 库函数 360
表 3-514. 枚举类型 usart_flag_enum 362
表 3-515. 枚举类型 usart_interrupt_flag_enum 362
表 3-516. 枚举类型 usart_interrupt_enum 363
表 3-517. 枚举类型 usart_invert_enum 363
表 3-518. 函数 usart_deinit 363
表 3-519. 函数 usart_baudrate_set 364
表 3-520. 函数 usart_parity_config 365
表 3-521. 函数 usart_word_length_set 365
表 3-522. 函数 usart_stop_bit_set 366
表 3-523. 函数 usart_enable 367
表 3-524. 函数 usart_disable 367
表 3-525. 函数 usart_transmit_config 368
表 3-526. 函数 usart_receive_config 368
表 3-527. 函数 usart_data_first_config 369
表 3-528. 函数 usart_invert_config 370
表 3-529. 函数 usart_receiver_timeout_enable 371
表 3-530. 函数 usart_receiver_timeout_disable 371
表 3-531. 函数 usart_receiver_timeout_threshold_config 372
表 3-532. 函数 usart_data_transmit 372
表 3-533. 函数 usart_data_receive 373
表 3-534. 函数 usart_address_config 374
表 3-535. 函数 usart_mute_mode_enable 374
表 3-536. 函数 usart_mute_mode_disable 375
表 3-537. 函数 usart_mute_mode_wakeup_config 375
表 3-538. 函数 usart_lin_mode_enable 376
表 3-539. 函数 usart_lin_mode_disable 376
表 3-540. 函数 usart_lin_break_dection_length_config 377
表 3-541. 函数 usart_send_break 378
表 3-542. 函数 usart_halfduplex_enable 378
表 3-543. 函数 usart_halfduplex_disable 379
表 3-544. 函数 usart_synchronous_clock_enable 379
表 3-545. 函数 usart_synchronous_clock_disable 380
表 3-546. 函数 usart_synchronous_clock_config 380
表 3-547. 函数 usart_guard_time_config 381
表 3-548. 函数 usart_smartcard_mode_enable 382
表 3-549. 函数 usart_smartcard_mode_disable 382
表 3-550. 函数 usart_smartcard_mode_nack_enable 383
表 3-551. 函数 usart_smartcard_mode_nack_disable 383
表 3-552. 函数 usart_smartcard_autoretry_config 384
表 3-553. 函数 usart_block_length_config 385

表 3-554. 函数 usart_irda_mode_enable 385
表 3-555. 函数 usart_irda_mode_disable 386
表 3-556. 函数 usart_prescaler_config 386
表 3-557. 函数 usart_irda_lowpower_config 387
表 3-558. 函数 usart_hardware_flow_rts_config 388
表 3-559. 函数 usart_hardware_flow_cts_config 388
表 3-560. 函数 usart_dma_receive_config 389
表 3-561. 函数 usart_dma_transmit_config 390
表 3-562. 函数 usart_hardware_flow_coherence_config 390
表 3-563. 函数 usart_flag_get 391
表 3-564. 函数 usart_flag_clear 392
表 3-565. 函数 usart_interrupt_enable 392
表 3-566. 函数 usart_interrupt_disable 393
表 3-567. 函数 usart_interrupt_flag_get 394
表 3-568. 函数 usart_interrupt_flag_clear 395
表 3-569. WWDGT 寄存器 397
表 3-570. WWDGT 库函数 397
表 3-571. 函数 wwdgt_deinit 397
表 3-572. 函数 wwdgt_enable 398
表 3-573. 函数 wwdgt_counter_update 398
表 3-574. 函数 wwdgt_config 399
表 3-575. 函数 wwdgt_interrupt_enable 399
表 3-576. 函数 wwdgt_flag_get 400
表 3-577. 函数 wwdgt_flag_clear 401
表 4-1. 版本历史 402

1. 介绍

本手册介绍了32位基于ARM微控制器GD32E11x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GD32E11x所有外设的性能特征。该固件库还包括每一个外设的驱动描述和基于评估板的固件库使用例程。通过使用本固件库，用户无需深入掌握细节，也可以轻松应用每一个外设。使用本固件库可以大大减少用户的编程时间，从而降低开发成本。

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

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

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

此份固件库使用手册的整体架构如下：

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

1.1. 文档和固件库规则

1.1.1. 外设缩写

表 1-1. 外设缩写

外设缩写	说明
ADC	模数转换器
BKP	备份寄存器
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
EXMC	外部存储器控制器
EXTI	外部中断事件控制器
FMC	闪存控制器

外设缩写	说明
FWDGT	独立看门狗
GPIO/AFIO	通用和备用输入/输出接口
I2C	内部集成电路总线接口
MISC	嵌套中断向量列表控制器
PMU	电源管理单元
RCU	复位和时钟单元
RTC	实时时钟
SPI/I2S	串行外设接口/片上音频接口
TIMER	定时器
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：

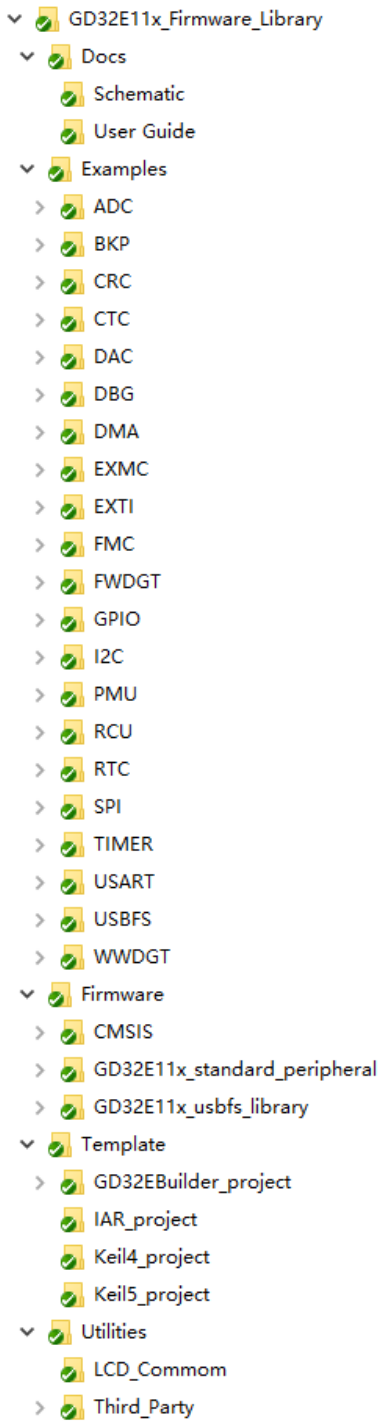
- XXX表示任一外设缩写，例如：ADC。更多缩写相关信息参阅外设缩写；
- 源文件和头文件命名都以"gd32e11x_"作为开头，例如：gd32e11x_adc.h；
- 常量仅被应用于一个文件的，定义于该文件中；被应用于多个文件的，在对应头文件中定义。所有常量都由英文字母大写书写；
- 寄存器作为常量处理。他们的命名都由英文字母大写书写。在大多数情况下，寄存器缩写规范与本用户手册一致；
- 变量名采用全部小写，有多个单词组成的，在单词之间以下划线分隔；
- 外设函数的命名以该外设的缩写加下划线为开头，有多个单词组成的，在单词之间以下划线分隔，所有外设函数都由英文字母小写书写。

2. 固件库概述

2.1. 文件组织结构

GD32E11x_Firmware_Library，文件组织结构见下图：

图 2-1. GD32E11x 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；
- gd32e11x_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)；
- gd32e11x_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)；
- gd32e11x.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GD32E11x的全局头文件和系统配置文件；
- GD32E11x_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；
- GD32E11x_usbfs_driver子文件夹包含了关于USBFS外设的所有文件；用户无需修改该文件夹；

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

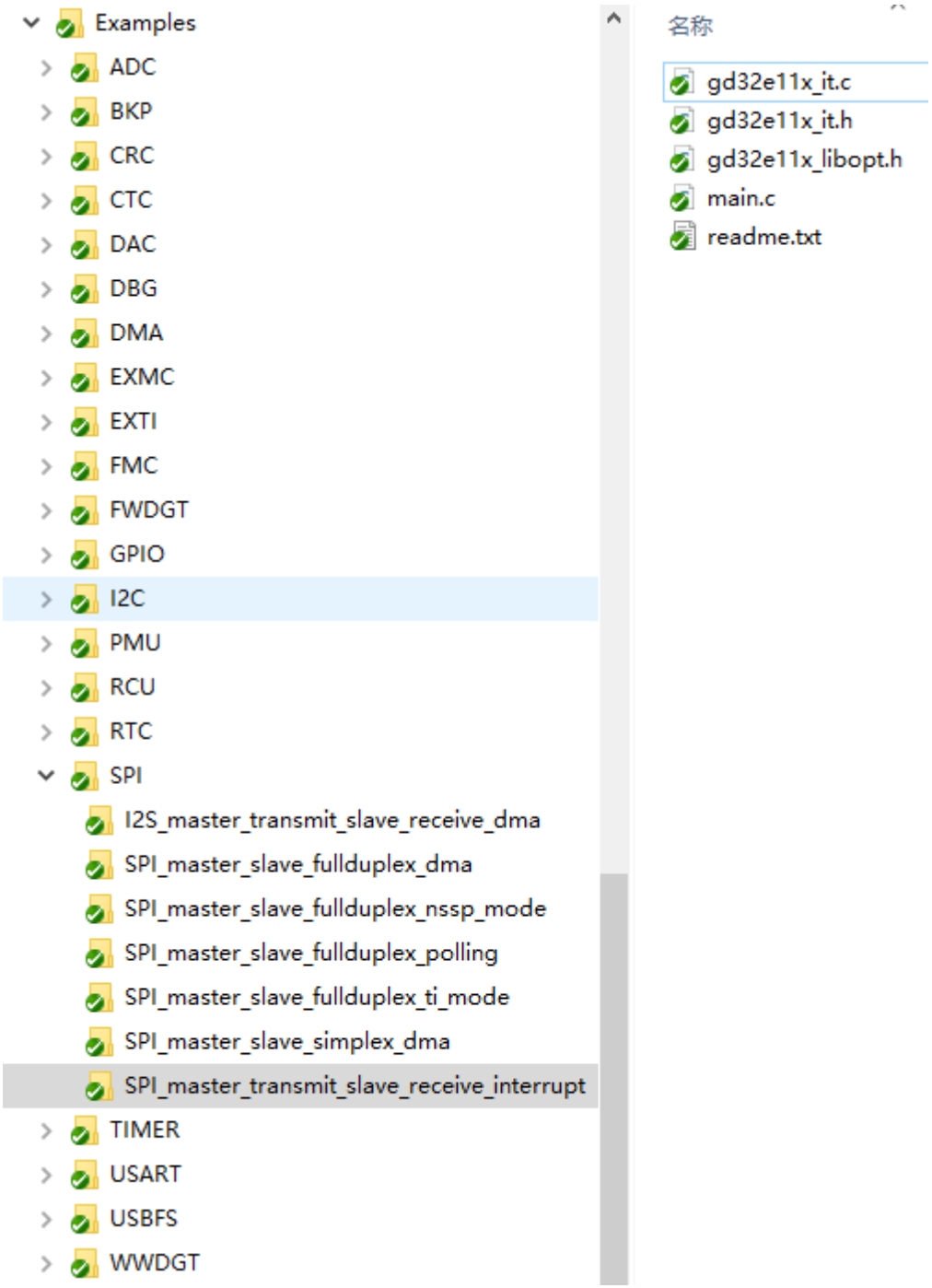
2.1.3. Template 文件夹

Template文件夹包含一个关于使用LED、USART打印、按键控制的简单例程，(IAR_project用于IAR编译环境，Keil4_project用于Keil4编译环境，Keil5_project用于Keil5编译环境，GD32EBuilder_project用于GD32EmbeddedBuilder编译环境)。用户可以使用该工程模板进行固件库例程的移植编译，具体使用方法见下：

选择文件

打开"Examples"文件夹，选择需要测试的模块，如SPI，打开"SPI"文件夹，选择SPI的一个例程，如"SPI_master_transmit_slave_receive_interrupt"，如下图所示：

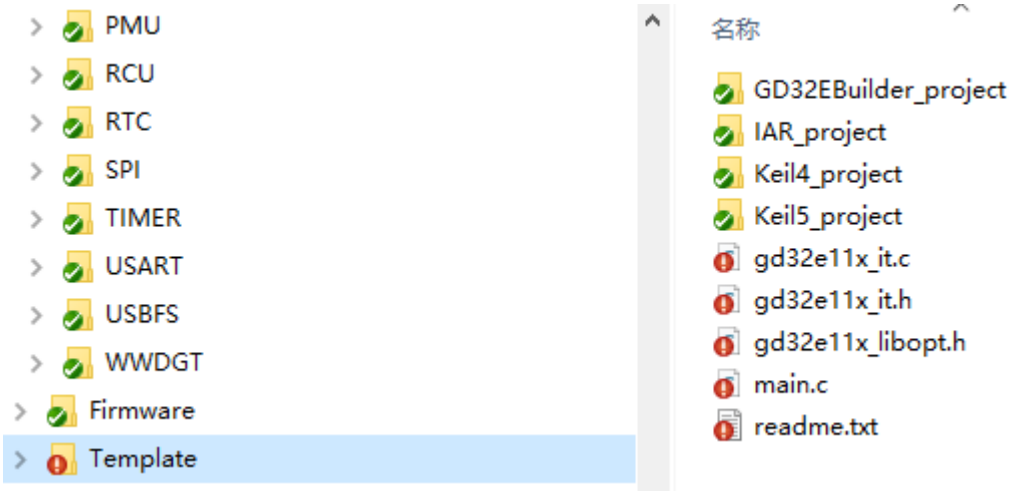
图 2-2. 选择外设例程文件



拷贝文件

打开"Template"文件夹，保留"IAR_project"、"Keil4_project"、"Keil5_project"和"GD32EBuilder_project"文件夹，其他文件都删除，然后将"SPI_master_transmit_slave_receive_interrupt"文件夹中的所有文件拷到"Template"文件夹子目录下，如下图所示:

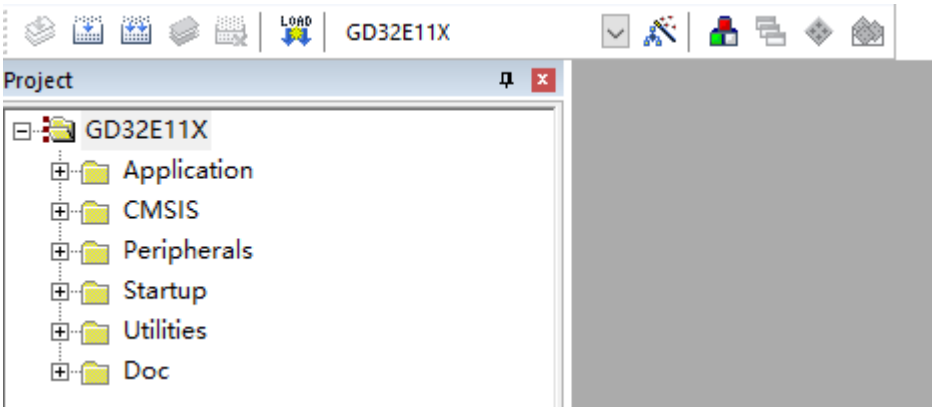
图 2-3. 拷贝外设例程文件



打开工程

GD提供Keil4、Keil5、IAR和GD32EBuilder三种版本的工程，根据客户所安装的软件，打开不同的project，如"Keil4_project"，打开\Template\Keil4_project\Project.uvproj，如下图所示：

图 2-4. 打开工程文件

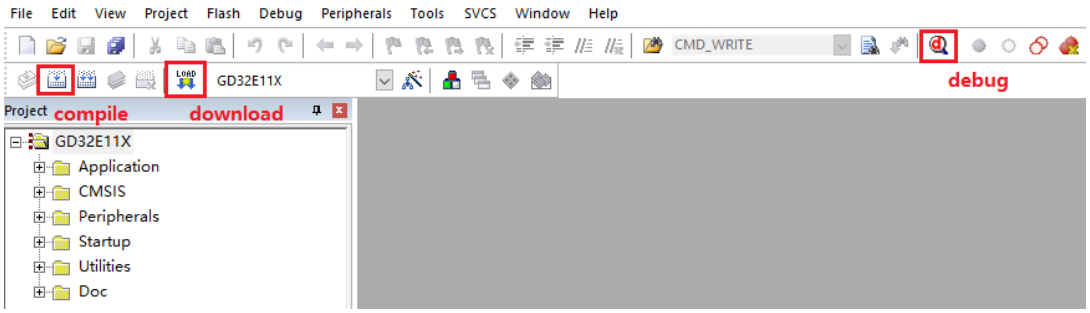


由于不同的模块、不同的功能，会使用到不同的文件，需要根据客户选择拷贝的文件，对工程里的文件进行增加或删除。

编译调试下载

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

图 2-5. 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：

- LCD_Commom及Third_Party子文件夹包含有USB测试所需文件；
- gd32e11x_eval.h及gd32e11x_lcd_eval.h文件是运行固件库例程所需关于评估板的头文件；
- gd32e11x_eval.c及gd32e11x_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
gd32e11x_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。
gd32e11x_it.h	头文件，包含所有中断处理函数原形。
gd32e11x_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。
gd32e11x_xxx.h	外设PPP的头文件。包含外设PPP函数的定义，以及这些函数使用的变量。
gd32e11x_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
XX	输入参数可选宏描述
输出参数{out}	
XXX	输出参数描述
返回值	
XXX	函数的返回值

3.2. ADC

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

3.2.1. 外设寄存器描述

ADC寄存器列表如下表所示：

表 3-2. ADC 寄存器

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

寄存器名称	寄存器描述
ADC_RSQ1	规则序列寄存器1
ADC_RSQ2	规则序列寄存器2
ADC_ISQ	注入序列寄存器
ADC_IDATAx(x=0..3)	注入数据寄存器x
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ADCx外设
adc_mode_config	配置ADC同步模式
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_enable	使能ADCx外设
adc_disable	禁能ADCx外设
adc_calibration_enable	ADCx校准复位
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_resolution_config	配置ADCx分辨率
adc_oversample_mode_config	配置ADCx过采样模式
adc_oversample_mode_enable	使能ADCx过采样
adc_oversample_mode_disable	禁能ADCx过采样
adc_dma_mode_enable	ADCx DMA请求使能
adc_dma_mode_disable	ADCx DMA请求禁能
adc_discontinuous_mode_config	配置ADC间断模式
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_source_config	配置ADC外部触发源
adc_external_trigger_config	配置ADC外部触发
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_sync_mode_convert_value_read	在同步模式下，读ADC0和ADC1最近的一次转换结果
adc_watchdog_single_channel_enabl	配置ADC模拟看门狗单通道有效

库函数名称	库函数描述
e	
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_flag_get	获取ADC标志位
adc_flag_clear	清除ADC标志位
adc_regular_software_startconv_flag_get	获取ADC规则通道组软件触发转换开始位
adc_inserted_software_startconv_flag_get	获取ADC注入通道组软件触发转换开始位
adc_interrupt_flag_get	获取ADC中断标志位
adc_interrupt_flag_clear	清除ADC中断标志位
adc_interrupt_enable	ADC中断使能
adc_interrupt_disable	ADC中断禁能

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(uint32_t adc_periph);
功能描述	复位ADCx外设
先决条件	-
被调用函数	rcu_periph_reset_enable / rcu_periph_reset_disable
输入参数{in}	
adc_periph	ADC外设
ADCx(x=0,1)	ADC外设选择
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* reset ADC0 */
adc_deinit (ADC0);
```

函数 adc_mode_config

函数adc_mode_config描述见下表：

表 3-5. 函数 adc_mode_config

函数名称	adc_mode_config
函数原形	void adc_mode_config(uint32_t mode);
功能描述	配置ADC同步模式
先决条件	-
被调用函数	-
输入参数{in}	
mode	ADC 运行模式
ADC_MODE_FREE	所有ADC运行于独立模式
ADC_DAUL_REGULAL_PARALLEL_INSERTED_PARALLEL	ADC0和ADC1运行在规则并行+注入并行组合模式
ADC_DAUL_REGULAL_PARALLEL_INSERTED_ROTATION	ADC0和ADC1运行在规则并行+交替触发组合模式
ADC_DAUL_INSERTED_PARALLEL_REGULAL_FOLLOWUP_FAST	ADC0和ADC1运行在注入并行+快速交叉组合模式
ADC_DAUL_INSERTED_PARALLEL_REGULAL_FOLLOWUP_SLOW	ADC0和ADC1运行在注入并行+慢速交叉组合模式
ADC_DAUL_INSERTED_PARALLEL	ADC0和ADC1运行在注入并行模式
ADC_DAUL_REGULAL_PARALLEL	ADC0和ADC1运行在规则并行模式
ADC_DAUL_REGULAL_FOLLOWUP_FAST	ADC0和ADC1运行在快速交叉模式
ADC_DAUL_REGULAL_FOLLOWUP_SLOW	ADC0和ADC1运行在慢速交叉模式
ADC_DAUL_INSERTED_TRRIGGER_ROTATION	ADC0和ADC1运行在交替触发模式
输出参数{out}	
-	-
返回值	
-	-

例如：

/* configure the ADC sync mode */

adc_mode_config(ADC_MODE_FREE);

函数 adc_special_function_config

函数adc_special_function_config描述见下表：

表 3-6. 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 adc_periph, 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}	
adc_periph	ADC外设
ADCx(x=0,1)	ADC外设选择
输入参数{in}	
function	功能配置
ADC_SCAN_MODE	扫描模式选择
ADC_INSERTED_CHANNEL_AUTO	注入组自动转换
ADC_CONTINUOUS_MODE	连续模式选择
输入参数{in}	
newvalue	功能使能禁能
ENABLE	使能
DISABLE	禁能
输出参数{out}	
-	-
返回值	
-	-

例如：

/* enable ADC0 scan mode */

adc_special_function_config(ADC0, ADC_SCAN_MODE, ENABLE);

函数 adc_data_alignment_config

函数adc_alignment_config描述见下表：

表 3-7. 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 adc_periph, uint32_t data_alignment);
功能描述	配置ADCx数据对齐方式
先决条件	-
被调用函数	-
输入参数{in}	
adc_periph	ADC外设
ADCx(x=0,1)	ADC外设选择
输入参数{in}	
data_alignment	数据对齐方式选择
ADC_DATAALIGN_RIGHT	LSB 对齐
ADC_DATAALIGN_LEFT	MSB 对齐
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* configure ADC0 data alignment */
adc_data_alignment_config(ADC0, ADC_DATAALIGN_RIGHT);
```

函数 adc_enable

函数adc_enable描述见下表：

表 3-8. 函数 adc_enable

函数名称	adc_enable
函数原形	void adc_enable(uint32_t adc_periph);
功能描述	使能ADCx外设
先决条件	-
被调用函数	-
输入参数{in}	
adc_periph	ADC外设
ADCx(x=0,1)	ADC外设选择
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* enable ADC0 */
adc_enable(ADC0);
```

函数 adc_disable

函数adc_disable描述见下表：

表 3-9. 函数 adc_disable

函数名称	adc_disable
函数原形	void adc_disable(uint32_t adc_periph);
功能描述	禁能ADCx外设
先决条件	-
被调用函数	-
输入参数{in}	
adc_periph	ADC外设
ADCx(x=0,1)	ADC外设选择
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* disable ADC0 */
adc_disable(ADC0);
```

函数 adc_calibration_enable

函数adc_calibration_enable描述见下表：

表 3-10. 函数 adc_calibration_enable

函数名称	adc_calibration_enable
函数原形	void adc_calibration_enable(uint32_t adc_periph);
功能描述	ADCx校准复位
先决条件	-
被调用函数	-
输入参数{in}	
adc_periph	ADC外设
ADCx(x=0,1)	ADC外设选择
输出参数{out}	
-	-

返回值	
-	-

例如：

```
/* ADC0 calibration and reset calibration */
adc_calibration_enable(ADC0);
```

函数 adc_tempsensor_vrefint_enable

函数adc_tempsensor_vrefint_enable描述见下表：

表 3-11. 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 3-12. 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_resolution_config

函数adc_resolution_config描述见下表：

表 3-13. 函数 adc_resolution_config

函数名称	adc_resolution_config
函数原形	void adc_resolution_config(uint32_t adc_periph , uint32_t resolution);
功能描述	配置ADCx分辨率
先决条件	-
被调用函数	-
输入参数{in}	
adc_periph	ADC外设
ADCx(x=0,1)	ADC外设选择
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 ADC0 data alignment */

adc_data_alignment_config(ADC0, ADC_DATAALIGN_RIGHT);

函数 adc_oversample_mode_config

函数adc_oversample_mode_config描述见下表：

表 3-14. 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 adc_periph, uint32_t mode, uint16_t shift, uint8_t ratio);
功能描述	配置ADCx过采样模式
先决条件	-
被调用函数	-
输入参数{in}	
adc_periph	ADC外设
ADCx(x=0,1)	ADC外设选择
输入参数{in}	
mode	ADC过采样触发模式
ADC_OVERSAMPLING_ALL_CONVERT	在一个触发之后，对一个通道连续进行过采样转换
ADC_OVERSAMPLING_ONE_CONVERT	在一个触发之后，对一个通道只进行一次过采样转换
输入参数{in}	
shift	ADC过滤采样移位
ADC_OVERSAMPLING_SHIFT_NONE	不移位
ADC_OVERSAMPLING_SHIFT_1B	移1位
ADC_OVERSAMPLING_SHIFT_2B	移2位
ADC_OVERSAMPLING_SHIFT_3B	移3位
ADC_OVERSAMPLING_SHIFT_4B	移4位
ADC_OVERSAMPLING_SHIFT_5B	移5位
ADC_OVERSAMPLING_SHIFT_6B	移6位
ADC_OVERSAMPLING_SHIFT_7B	移7位
ADC_OVERSAMPLING_SHIFT_8B	移8位

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

Example:

/* configure ADC1 oversample mode: 16 times sample, 4 bits shift */

adc_oversample_mode_config(ADC1, 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 3-15. 函数 adc_oversample_mode_enable

函数名称	adc_oversample_mode_enable
函数原形	void adc_oversample_mode_enable(uint32_t adc_periph);
功能描述	使能ADCx过采样
先决条件	-
被调用函数	-
输入参数{in}	
adc_periph	ADC外设

ADCx(x=0,1)	ADC外设选择
输出参数{out}	
-	-
返回值	
-	-

Example:

/* enable ADC0 oversample mode */

adc_oversample_mode_enable (ADC0);

函数 adc_oversample_mode_disable

函数adc_oversample_mode_disable描述见下表：

表 3-16. 函数 adc_oversample_mode_disable

函数名称	adc_oversample_mode_disable
函数原形	void adc_oversample_mode_disable(uint32_t adc_periph);
功能描述	禁能ADCx过采样
先决条件	-
被调用函数	-
输入参数{in}	
adc_periph	ADC外设
ADCx(x=0,1)	ADC外设选择
输出参数{out}	
-	-
返回值	
-	-

Example:

/* disable ADC0 oversample mode */

adc_oversample_mode_disable (ADC0);

函数 adc_dma_mode_enable

函数adc_dma_mode_enable描述见下表：

表 3-17. 函数 adc_dma_mode_enable

函数名称	adc_dma_mode_enable
函数原形	void adc_dma_mode_enable(uint32_t adc_periph);
功能描述	ADCx DMA请求使能
先决条件	-
被调用函数	-

输入参数{in}	
adc_periph	ADC外设
ADCx(x=0,1)	ADC外设选择
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* enable ADC0 DMA request */
adc_dma_mode_enable(ADC0);
```

函数 adc_dma_mode_disable

函数adc_dma_mode_disable描述见下表：

表 3-18. 函数 adc_dma_mode_disable

函数名称	adc_dma_mode_disable
函数原形	void adc_dma_mode_disable(uint32_t adc_periph);
功能描述	ADCx DMA请求禁能
先决条件	-
被调用函数	-
输入参数{in}	
adc_periph	ADC外设
ADCx(x=0,1)	ADC外设选择
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* disable ADC0 DMA request */
adc_dma_mode_disable(ADC0);
```

函数 adc_discontinuous_mode_config

函数adc_discontinuous_mode_config描述见下表：

表 3-19. 函数 adc_discontinuous_mode_config

函数名称	adc_discontinuous_mode_config
函数原形	void adc_discontinuous_mode_config(uint32_t adc_periph, uint8_t adc_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_periph	ADC外设
ADCx(x=0,1)	ADC外设选择
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 ADC0 discontinuous mode */
adc_discontinuous_mode_config(ADC0, ADC_REGULAR_CHANNEL, 6);
```

函数 adc_channel_length_config

函数adc_channel_length_config描述见下表：

表 3-20. 函数 adc_channel_length_config

函数名称	adc_channel_length_config
函数原形	void adc_channel_length_config(uint32_t adc_periph, uint8_t adc_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}	
adc_periph	ADC外设
ADCx(x=0,1)	ADC外设选择
输入参数{in}	
adc_channel_grou	通道组选择

p	
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 ADC0 regular channel */
adc_channel_length_config(ADC0, ADC_REGULAR_CHANNEL, 4);
```

函数 adc_regular_channel_config

函数adc_regular_channel_config描述见下表：

表 3-21. 函数 adc_regular_channel_config

函数名称	adc_regular_channel_config
函数原形	void adc_regular_channel_config(uint32_t adc_periph, uint8_t rank, uint8_t adc_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}	
adc_periph	ADC外设
ADCx(x=0,1)	ADC外设选择
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}	
adc_channel	ADC通道选择
ADC_CHANNEL_x(x=0..17)	ADC 通道x (x=0..17)(只有ADC0，可取值x=16和17)
输入参数{in}	
sample_time	采样时间
ADC_SAMPLETIME_1POINT5	1.5 周期
ADC_SAMPLETIME_7POINT5	7.5 周期

ADC_SAMPLETIME_13POINT5	13.5 周期
ADC_SAMPLETIME_28POINT5	28.5 周期
ADC_SAMPLETIME_41POINT5	41.5 周期
ADC_SAMPLETIME_55POINT5	55.5 周期
ADC_SAMPLETIME_71POINT5	71.5 周期
ADC_SAMPLETIME_239POINT5	239.5 周期
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* configure ADC0 regular channel */
adc_regular_channel_config(ADC0, 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 3-22. 函数 adc_inserted_channel_config

函数名称	adc_inserted_channel_config
函数原形	void adc_inserted_channel_config(uint32_t adc_periph, uint8_t rank, uint8_t adc_channel, uint32_t sample_time);
功能描述	配置ADC注入通道组
先决条件	-
被调用函数	-
输入参数{in}	
adc_periph	ADC外设
ADCx(x=0,1)	ADC外设选择
输入参数{in}	
rank	注入组通道序列，取值范围为0~3
输入参数{in}	
adc_channel	ADC通道选择
ADC_CHANNEL_x(x=0..17)	ADC 通道x (x=0..17)(只有ADC0，可取值x=16和17)
输入参数{in}	

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

例如：

```
/* configure ADC0 inserted channel */
adc_inserted_channel_config(ADC0, 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 3-23. 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(uint32_t adc_periph, 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}	
adc_periph	ADC外设
ADCx(x=0,1)	ADC外设选择
输入参数{in}	

inserted_channel	注入通道选择
ADC_INSERTED_CHANNEL_x(x=0..3)	注入通道，x=0,1,2,3
输入参数{in}	
offset	数据偏移值，取值范围为0~4095
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* configure ADC0 inserted channel offset */
adc_inserted_channel_offset_config(ADC0, ADC_INSERTED_CHANNEL_0, 100);
```

函数 adc_external_trigger_source_config

函数adc_external_trigger_source_config描述见下表：

表 3-24. 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(uint32_t adc_periph, uint8_t adc_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}	
adc_periph	ADC外设
ADCx(x=0,1)	ADC外设选择
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
输入参数{in}	
external_trigger_source	规则通道组或注入通道组触发源
ADC0_1_EXTTRIG_REGULAR_T0_CH0	TIMER0 CH0事件(规则组)
ADC0_1_EXTTRIG	TIMER0 CH1事件(规则组)

_REGULAR_T0_CH1	
ADC0_1_EXTTRIG_REGULAR_T0_CH2	TIMER0 CH2事件(规则组)
ADC0_1_EXTTRIG_REGULAR_T1_CH1	TIMER1 CH1事件(规则组)
ADC0_1_EXTTRIG_REGULAR_T2_TRGO	TIMER2 TRGO事件(规则组)
ADC0_1_EXTTRIG_REGULAR_T3_CH3	TIMER3 CH3事件(规则组)
ADC0_1_EXTTRIG_REGULAR_T7_TRGO	TIMER7 TRGO事件(规则组)
ADC0_1_EXTTRIG_REGULAR_EXTI_11	外部中断线11(规则组)
ADC2_EXTTRIG_REGULAR_T2_CH0	TIMER2 CH0事件(规则组)
ADC0_1_EXTTRIG_REGULAR_NONE	软件触发(规则组)
ADC0_1_EXTTRIG_INSERTED_T0_TRGO	TIMER0 TRGO事件(注入组)
ADC0_1_EXTTRIG_INSERTED_T0_CH3	TIMER0 CH3事件(注入组)
ADC0_1_EXTTRIG_INSERTED_T1_TRGO	TIMER1 TRGO事件(注入组)
ADC0_1_EXTTRIG_INSERTED_T1_CH0	TIMER1 CH0事件(注入组)
ADC0_1_EXTTRIG_INSERTED_T2_CH3	TIMER2 CH3事件(注入组)
ADC0_1_EXTTRIG_INSERTED_T3_TRGO	TIMER3 TRGO事件(注入组)
ADC0_1_EXTTRIG	外部中断线15 (注入组)

_INSERTED_EXTI_15	
ADC0_1_EXTTRIG_INSERTED_T7_CH3	TIMER7 CH3事件(注入组)
ADC0_1_EXTTRIG_INSERTED_NONE	软件触发(注入组)
输出参数{out}	
-	-
返回值	
-	-

例如：

/* configure ADC0 regular channel external trigger source */

adc_external_trigger_source_config(ADC0,ADC_REGULAR_CHANNEL, ADC0_1_EXTTRIG_REGULAR_T0_CH0);

函数 adc_external_trigger_config

函数adc_external_trigger_config描述见下表：

表 3-25. 函数 adc_external_trigger_config

函数名称	adc_external_trigger_config
函数原形	void adc_external_trigger_config(uint32_t adc_periph, uint8_t adc_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}	
adc_periph	ADC外设
ADCx(x=0,1)	ADC外设选择
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
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 ADC0 inserted channel group external trigger */
adc_external_trigger_config(ADC0, ADC_INSERTED_CHANNEL_0, ENABLE);
```

函数 adc_software_trigger_enable

函数adc_software_trigger_enable描述见下表：

表 3-26. 函数 adc_software_trigger_enable

函数名称	adc_software_trigger_enable
函数原形	void adc_software_trigger_enable(uint32_t adc_periph, uint8_t adc_channel_group);
功能描述	ADC软件触发使能
先决条件	-
被调用函数	-
输入参数{in}	
adc_periph	ADC外设
ADCx(x=0,1)	ADC外设选择
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
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* enable ADC0 regular channel group software trigger */
adc_software_trigger_enable(ADC0, ADC_REGULAR_CHANNEL);
```

函数 adc_regular_data_read

函数adc_inserted_regular_data_read描述见下表：

表 3-27. 函数 adc_regular_data_read

函数名称	adc_regular_data_read
函数原形	uint16_t adc_regular_data_read(uint32_t adc_periph);
功能描述	读ADC规则组数据寄存器
先决条件	-
被调用函数	-
输入参数{in}	
adc_periph	ADC外设
ADCx(x=0,1)	ADC外设选择
输出参数{out}	
-	-
返回值	
uint16_t	ADC转换值 (0-0xFFFF)

例如：

```
/* read ADC0 regular group data register   */
uint16_t adc_value = 0;
adc_value = adc_regular_data_read(ADC0);
```

函数 adc_inserted_data_read

函数adc_inserted_regular_data_read描述见下表：

表 3-28. 函数 adc_inserted_data_read

函数名称	adc_inserted_data_read
函数原形	uint16_t adc_inserted_data_read(uint32_t adc_periph, 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}	
adc_periph	ADC外设
ADCx(x=0,1)	ADC外设选择
输入参数{in}	
inserted_channel	注入通道选择
ADC_INSERTED_CHANNEL_x(x=0..3)	注入通道x，x=0,1,2,3
输出参数{out}	
-	-
返回值	
uint16_t	ADC转换值(0-0xFFFF)

例如：

```
/* read ADC0 inserted group data register   */
uint16_t adc_value = 0;
adc_value = adc_inserted_data_read (ADC0, ADC_INSERTED_CHANNEL_0);
```

函数 adc_sync_mode_convert_value_read

函数adc_sync_mode_convert_value_read描述见下表：

表 3-29. 函数 adc_sync_mode_convert_value_read

函数名称	adc_sync_mode_convert_value_read
函数原形	uint32_t adc_sync_mode_convert_value_read(void);
功能描述	在同步模式下，读ADC0和ADC1最近的一次转换结果
先决条件	-
被调用函数	-
输入参数{in}	
-	-
输出参数{out}	
-	-
返回值	
uint32_t	ADC转换值（0-0xFFFFFFFF）

例如：

```
/* read the last ADC0 and ADC1 conversion result data in sync mode   */
Uint32_t adc_value = 0;
adc_value = adc_sync_mode_convert_value_read ();
```

函数 adc_watchdog_single_channel_enable

函数adc_watchdog_single_channel_enable描述见下表：

表 3-30. 函数 adc_watchdog_single_channel_enable

函数名称	adc_watchdog_single_channel_enable
函数原形	void adc_watchdog_single_channel_enable(uint32_t adc_periph, uint8_t adc_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_periph	ADC外设
ADCx(x=0,1)	ADC外设选择

输入参数{in}	
adc_channel	选择ADC通道
ADC_CHANNEL_x(x=0..17)	ADC Channelx(x=0..17) (只有ADC0，可取值x=16和17)
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* configure ADC0 analog watchdog single channel */
adc_watchdog_single_channel_enable(ADC0, ADC_CHANNEL_1);
```

函数 adc_watchdog_group_channel_enable

函数adc_watchdog_group_channel_enable描述见下表：

表 3-31. 函数 adc_watchdog_group_channel_enable

函数名称	adc_watchdog_group_channel_enable
函数原形	void adc_watchdog_group_channel_enable(uint32_t adc_periph, uint8_t adc_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}	
adc_periph	ADC外设
ADCx(x=0,1)	ADC外设选择
输入参数{in}	
adc_channel_group	通道组使用模拟看门狗
ADC_REGULAR_CHANNEL	规则通道组
ADC_INSERTED_CHANNEL	注入通道组
ADC_REGULAR_INSERTED_CHANNEL	规则和注入通道组
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* configure ADC0 analog watchdog group channel */

adc_watchdog_group_channel_enable(ADC0, ADC_REGULAR_CHANNEL);
```

函数 adc_watchdog_disable

函数adc_watchdog_disable描述见下表：

表 3-32. 函数 adc_watchdog_disable

函数名称	adc_watchdog_disable
函数原形	void adc_watchdog_disable(uint32_t adc_periph);
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}	
adc_periph	ADC外设
ADCx(x=0,1)	ADC外设选择
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* disable ADC0 analog watchdog */

adc_watchdog_disable(ADC0);
```

函数 adc_watchdog_threshold_config

函数adc_watchdog_threshold_config描述见下表：

表 3-33. 函数 adc_watchdog_threshold_config

函数名称	adc_watchdog_threshold_config
函数原形	void adc_watchdog_threshold_config(uint32_t adc_periph, 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}	
adc_periph	ADC外设
ADCx(x=0,1)	ADC外设选择
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 ADC0 analog watchdog threshold */
adc_watchdog_threshold_config(ADC0, 0x0400, 0x0A00);
```

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 adc_periph , uint32_t adc_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}	
adc_periph	ADC外设
ADCx(x=0,1)	ADC外设选择
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
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 ADC0 analog watchdog flag bits*/
FlagStatus flag_value;
flag_value = adc_flag_get(ADC0, 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 adc_periph, uint32_t adc_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_periph	ADC外设
ADCx(x=0,1)	ADC外设选择
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
ADC_FLAG_STRC	规则通道组转换开始标志位
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* clear the ADC0 analog watchdog flag bits*/
adc_flag_clear(ADC0, ADC_FLAG_WDE);
```

函数 adc_regular_software_startconv_flag_get

函数adc_regular_software_startconv_flag_get描述见下表：

表 3-36. 函数 adc_regular_software_startconv_flag_get

函数名称	adc_regular_software_startconv_flag_get
函数原形	FlagStatus adc_regular_software_startconv_flag_get(uint32_t adc_periph);
功能描述	获取ADC规则通道组软件触发转换开始位
先决条件	-
被调用函数	-
输入参数{in}	
adc_periph	ADC外设
ADCx(x=0,1)	ADC外设选择

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

例如：

```
/* get the bit state of ADC0 software regular channel start conversion */
FlagStatus flag_value;
flag_value = adc_regular_software_startconv_flag_get(ADC0);
```

函数 adc_inserted_software_startconv_flag_get

函数adc_inserted_software_startconv_flag_get描述见下表：

表 3-37. 函数 adc_inserted_software_startconv_flag_get

函数名称	adc_inserted_software_startconv_flag_get
函数原形	FlagStatus adc_inserted_software_startconv_flag_get(uint32_t adc_periph);
功能描述	获取ADC规则注入组软件触发转换开始位
先决条件	-
被调用函数	-
输入参数{in}	
adc_periph	ADC外设
ADCx(x=0,1)	ADC外设选择
输出参数{out}	
-	-
返回值	
FlagStatus	SET或RESET

例如：

```
/* get the bit state of ADC0 software inserted channel start conversion */
FlagStatus flag_value;
flag_value = adc_inserted_software_startconv_flag_get(ADC0);
```

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 adc_periph, uint32_t adc_interrupt);
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}	
adc_periph	ADC外设
ADCx(x=0,1)	ADC外设选择
输入参数{in}	
adc_interrupt	ADC中断标志位
ADC_INT_WDE	模拟看门狗中断标志位
ADC_INT_EOC	组转换结束中断标志位
ADC_INT_EOIC	注入通道组转换结束中断标志位
输出参数{out}	
-	-
返回值	
FlagStatus	SET 或 RESET

例如：

```
/* get the ADC0 analog watchdog interrupt bits*/
FlagStatus flag_value;
flag_value = adc_interrupt_flag_get(ADC0, ADC_INT_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 adc_periph, uint32_t adc_interrupt);
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}	
adc_periph	ADC外设
ADCx(x=0,1)	ADC外设选择
输入参数{in}	
adc_interrupt	ADC中断标志位
ADC_INT_WDE	模拟看门狗中断标志位
ADC_INT_EOC	组转换结束中断标志位
ADC_INT_EOIC	注入通道组转换结束中断标志位
输出参数{out}	
-	-
返回值	

-	-

例如：

/* clear the ADC0 analog watchdog interrupt bits*/

adc_interrupt_flag_ clear(ADC0, ADC_INT_WDE);

函数 adc_interrupt_enable

函数adc_interrupt_enable描述见下表：

表 3-40. 函数 adc_interrupt_enable

函数名称	adc_interrupt_enable
函数原形	void adc_interrupt_enable(uint32_t adc_periph, uint32_t adc_interrupt);
功能描述	ADC中断使能
先决条件	-
被调用函数	-
输入参数{in}	
adc_periph	ADC外设
ADCx(x=0,1)	ADC外设选择
输入参数{in}	
adc_interrupt	ADC中断标志位
ADC_INT_WDE	模拟看门狗中断标志位
ADC_INT_EOC	组转换结束中断标志位
ADC_INT_EOIC	注入通道组转换结束中断标志位
输出参数{out}	
-	-
返回值	
-	-

例如：

/* enable ADC0 analog watchdog interrupt */

adc_interrupt_enable(ADC0, ADC_INT_WDE);

函数 adc_interrupt_disable

函数adc_interrupt_disable描述见下表：

表 3-41. 函数 adc_interrupt_disable

函数名称	adc_interrupt_disable
函数原形	void adc_interrupt_enable(uint32_t adc_periph, uint32_t adc_interrupt);
功能描述	ADC中断禁能
先决条件	-

被调用函数	-
输入参数{in}	
adc_periph	ADC外设
ADCx(x=0,1)	ADC外设选择
输入参数{in}	
adc_interrupt	ADC中断标志位
ADC_INT_WDE	模拟看门狗中断标志位
ADC_INT_EOC	组转换结束中断标志位
ADC_INT_EOIC	注入通道组转换结束中断标志位
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* disable ADC0 interrupt */
adc_interrupt_disable(ADC0, ADC_INT_WDE);
```

3.3. BKP

位于备份域中的备份寄存器可在V_{DD}电源关闭时由V_{BAT}供电，备份寄存器有42个16位(84字节)寄存器可用来存储并保护用户应用数据，从待机模式唤醒或系统复位也不会对这些寄存器造成影响。章节*3.3.1*描述了BKP的寄存器列表，章节*3.3.2*对BKP库函数进行说明。

3.3.1. 外设寄存器说明

BKP寄存器列表如下表所示：

表 3-42. BKP 寄存器

寄存器名称	寄存器描述
BKP_DATAx (x= 0..41)	备份数据寄存器
BKP_OCTL	RTC信号输出控制寄存器
BKP_TPCTL	侵入引脚控制寄存器
BKP_TPCS	侵入控制状态寄存器

3.3.2. 外设库函数说明

BKP库函数列表如下表所示：

表 3-43. BKP 库函数

库函数名称	库函数描述
bkp_deinit	复位备份数据寄存器
bkp_data_write	写备份数据寄存器
bkp_data_read	读备份数据寄存器
bkp_rtc_calibration_output_enable	RTC时钟校准输出使能
bkp_rtc_calibration_output_disable	RTC时钟校准输出失能
bkp_rtc_signal_output_enable	RTC闹钟或秒信号输出使能
bkp_rtc_signal_output_disable	RTC闹钟或秒信号输出失能
bkp_rtc_output_select	RTC输出选择，RTC输出可选择为闹钟脉冲或秒脉冲
bkp_rtc_clock_output_select	RTC时钟输出选择
bkp_rtc_clock_calibration_direction_select	RTC时钟校准方向选择
bkp_rtc_calibration_value_set	RTC时钟校准值
bkp_tamper_detection_enable	TAMPER引脚使能
bkp_tamper_detection_disable	TAMPER引脚失能
bkp_tamper_active_level_set	TAMPER引脚有效电平设置
bkp_interrupt_enable	TAMPER中断使能
bkp_interrupt_disable	TAMPER中断失能
bkp_flag_get	获取标志位
bkp_flag_clear	清除标志位
bkp_interrupt_flag_get	获取中断标志位
bkp_interrupt_flag_clear	清除中断标志位

函数 bkp_deinit

函数bkp_deinit描述见下表：

表 3-44. 函数 bkp_deinit

函数名称	bkp_deinit
函数原型	void bkp_deinit(void);
功能描述	复位备份数据寄存器
先决条件	-
被调用函数	rcu_bkp_reset_enable / rcu_bkp_reset_disable
输入参数{in}	
-	-
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* reset BKP registers */
```

bkp_deinit();

函数 bkp_data_write

函数bkp_data_write描述见下表：

表 3-45. 函数 bkp_data_write

函数名称	bkp_data_write
函数原型	void bkp_data_write(bkp_data_register_enum register_number, uint16_t data);
功能描述	写备份数据寄存器
先决条件	-
被调用函数	-
输入参数{in}	
register_number	参考枚举bkp_data_register_enum
BKP_DATA_x(x = 0..41)	BKP数据寄存器x
输入参数{in}	
Data	待写入BKP数据寄存器的数据
0-0xffff	数值
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* write BKP data register */
bkp_data_write(BKP_DATA_0, 0x1226);
```

函数 bkp_data_read

函数bkp_data_read描述见下表：

表 3-46. 函数 bkp_data_read

函数名称	bkp_data_read
函数原型	uint16_t bkp_data_read(bkp_data_register_enum register_number);
功能描述	读备份数据寄存器
先决条件	-
被调用函数	-
输入参数{in}	
register_number	参考枚举bkp_data_register_enum
BKP_DATA_x(x = 0..41)	BKP数据寄存器x

输出参数{out}	
-	-
返回值	
uint16_t	0-0xffff

例如：

```
/* read BKP data register */
uint16_t data;
data = bkp_data_read(BKP_DATA_0);
```

函数 bkp_rtc_calibration_output_enable

函数bkp_rtc_calibration_output_enable描述见下表：

表 3-47. 函数 bkp_rtc_calibration_output_enable

函数名称	bkp_rtc_calibration_output_enable
函数原型	void bkp_rtc_calibration_output_enable(void);
功能描述	RTC时钟校准输出使能
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 RTC clock calibration output */
bkp_rtc_calibration_output_enable();
```

函数 bkp_rtc_calibration_output_disable

函数bkp_rtc_calibration_output_disable描述见下表：

表 3-48. 函数 bkp_rtc_calibration_output_disable

函数名称	bkp_rtc_calibration_output_disable
函数原型	void bkp_rtc_calibration_output_disable(void);
功能描述	RTC时钟校准输出失能
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 RTC clock calibration output */
bkp_rtc_calibration_output_disable();
```

函数 bkp_rtc_signal_output_enable

函数bkp_rtc_signal_output_enable描述见下表：

表 3-49. 函数 bkp_rtc_signal_output_enable

函数名称	bkp_rtc_signal_output_enable
函数原型	void bkp_rtc_signal_output_enable (void);
功能描述	RTC闹钟或秒信号输出使能
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 RTC alarm or second signal output */
bkp_rtc_signal_output_enable();
```

函数 bkp_rtc_signal_output_disable

函数bkp_rtc_signal_output_disable描述见下表：

表 3-50. 函数 bkp_rtc_signal_output_disable

函数名称	bkp_rtc_signal_output_disable
函数原型	void bkp_rtc_signal_output_disable (void);
功能描述	RTC闹钟或秒信号输出失能
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 RTC alarm or second signal output */
bkp_rtc_signal_output_disable();
```

函数 bkp_rtc_output_select

函数bkp_rtc_output_select描述见下表：

表 3-51. 函数 bkp_rtc_output_select

函数名称	bkp_rtc_output_select
函数原型	void bkp_rtc_output_select (uint16_t outputsel);
功能描述	RTC输出选择，RTC输出可选择为闹钟脉冲或秒脉冲
先决条件	-
被调用函数	-
输入参数{in}	
outputsel	RTC输出选择
RTC_OUTPUT_ALARM_PULSE	RTC闹钟脉冲被选择为RTC输出
RTC_OUTPUT_SECOND_PULSE	RTC秒脉冲被选择为RTC输出
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* select RTC output alarm signal output */
bkp_rtc_output_select(RTC_OUTPUT_ALARM_PULSE);
```

函数 bkp_rtc_clock_output_select

函数bkp_rtc_clock_output_select描述见下表：

表 3-52. 函数 bkp_rtc_clock_output_select

函数名称	bkp_rtc_clock_output_select

函数原型	void bkp_rtc_clock_output_select(uint16_t clocksel);
功能描述	RTC时钟输出选择，RTC时钟输出可选择为不分频或64分频
先决条件	-
被调用函数	-
输入参数{in}	
clocksel	RTC时钟输出选择
RTC_CLOCK_DIV_64	RTC时钟输出被选择为64分频
RTC_CLOCK_DIV_1	RTC时钟输出被选择为不分频
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* select RTC clock devided 64 to output */
bkp_rtc_clock_output_select(RTC_CLOCK_DIV_64);
```

函数 bkp_rtc_clock_calibration_direction_select

函数bkp_rtc_clock_calibration_direction_select描述见下表：

表 3-53. 函数 bkp_rtc_clock_calibration_direction_select

函数名称	bkp_rtc_clock_calibration_direction_select
函数原型	void bkp_rtc_clock_calibration_direction_select(uint16_t direction);
功能描述	RTC时钟校准方向选择，RTC时钟校准方向可选择为变快或变慢
先决条件	-
被调用函数	-
输入参数{in}	
direction	RTC时钟校准方向
RTC_CLOCK_SLOWED_DOWN	RTC时钟变慢
RTC_CLOCK_SPEED_UP	RTC时钟变快
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* set RTC clock slowed down */
```

bkp_rtc_clock_calibration_direction_select(RTC_CLOCK_SLOWED_DOWN);

函数 bkp_rtc_calibration_value_set

函数bkp_rtc_calibration_value_set描述见下表：

表 3-54. 函数 bkp_rtc_calibration_value_set

函数名称	bkp_rtc_calibration_value_set
函数原型	void bkp_rtc_calibration_value_set(uint8_t value);
功能描述	RTC时钟校准值
先决条件	-
被调用函数	-
输入参数{in}	
value	RTC时钟校准值
0x00 - 0x7F	校准值
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* set RTC clock calibration value */
bkp_rtc_calibration_value_set(0x7f);
```

函数 bkp_tamper_detection_enable

函数bkp_tamper_detection_enable描述见下表：

表 3-55. 函数 bkp_tamper_detection_enable

函数名称	bkp_tamper_detection_enable
函数原型	void bkp_tamper_detection_enable (void);
功能描述	TAMPER引脚使能
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 tamper pin detection */
bkp_tamper_detection_enable();
```

函数 bkp_tamper_detection_disable

函数bkp_tamper_detection_disable描述见下表：

表 3-56. 函数 bkp_tamper_detection_disable

函数名称	bkp_tamper_detection_disable
函数原型	void bkp_tamper_detection_disable (void);
功能描述	TAMPER引脚失能
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 tamper pin detection */
bkp_tamper_detection_disable();
```

函数 bkp_tamper_active_level_set

函数bkp_tamper_active_level_set描述见下表：

表 3-57. 函数 bkp_tamper_active_level_set

函数名称	bkp_tamper_active_level_set
函数原型	void bkp_tamper_active_level_set(uint16_t level);
功能描述	TAMPER引脚有效电平设置
先决条件	-
被调用函数	-
输入参数{in}	
level	TAMPER引脚有效电平
TAMPER_PIN_ACTIVE_HIGH	TAMPER引脚高电平有效
TAMPER_PIN_ACTIVE_LOW	TAMPER引脚低电平有效
输出参数{out}	
-	-

返回值	
-	-

例如：

```
/* set tamper pin active level high */
bkp_tamper_active_level_set(TAMPER_PIN_ACTIVE_HIGH);
```

函数 bkp_interrupt_enable

函数bkp_interrupt_enable描述见下表：

表 3-58. 函数 bkp_interrupt_enable

函数名称	bkp_interrupt_enable
函数原型	void bkp_interrupt_enable (void);
功能描述	TAMPER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 tamper pin interrupt */
bkp_interrupt_enable();
```

函数 bkp_interrupt_disable

函数bkp_interrupt_disable描述见下表：

表 3-59. 函数 bkp_interrupt_disable

函数名称	bkp_interrupt_disable
函数原型	void bkp_interrupt_disable(void);
功能描述	TAMPER中断失能
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 tamper pin interrupt */
bkp_interrupt_disable();
```

函数 bkp_flag_get

函数bkp_flag_get描述见下表：

表 3-60. 函数 bkp_flag_get

函数名称	bkp_flag_get
函数原型	FlagStatus bkp_flag_get(void);
功能描述	获取标志位
先决条件	-
被调用函数	-
输入参数{in}	
-	-
输出参数{out}	
-	-
返回值	
FlagStatus	SET或RESET

例如：

```
/* get BKP flag state */
FlagStatus status;
status = bkp_flag_get();
```

函数 bkp_flag_clear

函数bkp_flag_clear描述见下表：

表 3-61. 函数 bkp_flag_clear

函数名称	bkp_flag_clear
函数原型	void bkp_flag_clear(void);
功能描述	清除标志位
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 BKP flag state */
bkp_flag_clear();
```

函数 bkp_interrupt_flag_get

函数bkp_interrupt_flag_get描述见下表：

表 3-62. 函数 bkp_interrupt_flag_get

函数名称	bkp_interrupt_flag_get
函数原型	FlagStatus bkp_interrupt_flag_get(void);
功能描述	获取中断标志位
先决条件	-
被调用函数	-
输入参数{in}	
-	-
输出参数{out}	
-	-
返回值	
FlagStatus	SET或RESET

例如：

```
/* get BKP interrupt flag state */
bkp_interrupt_flag_get();
```

函数 bkp_interrupt_flag_clear

函数bkp_interrupt_flag_clear描述见下表：

表 3-63. 函数 bkp_interrupt_flag_clear

函数名称	bkp_interrupt_flag_clear
函数原型	void bkp_interrupt_flag_ clear(void);
功能描述	清除中断标志位
先决条件	-
被调用函数	-
输入参数{in}	
-	-

输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* clear BKP interrupt flag state */
bkp_interrupt_flag_clear();
```

3.4. CRC

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

3.4.1. 外设寄存器说明

CRC寄存器列表如下表所示：

表 3-64. CRC 寄存器

寄存器名称	寄存器描述
CRC_DATA	CRC数据寄存器
CRC_FDATA	CRC独立数据寄存器
CRC_CTL	CRC控制寄存器

3.4.2. 外设库函数说明

CRC库函数列表如下表所示：

表 3-65. CRC 库函数

库函数名称	库函数描述
crc_deinit	复位CRC计算单元
crc_data_register_reset	根据数据寄存器的复位值(0xFFFFFFFF)复位数据寄存器
crc_data_register_read	读数据寄存器
crc_free_data_register_read	读独立数据寄存器
crc_free_data_register_write	写独立数据寄存器
crc_single_data_calculate	CRC计算一个32位数据
crc_block_data_calculate	CRC计算一个32位数组

函数 crc_deinit

函数crc_deinit描述见下表：

表 3-66. 函数 crc_deinit

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

-	-
返回值	
-	-

例如：

```
/* reset crc */
crc_deinit();
```

函数 crc_data_register_reset

函数crc_data_register_reset描述见下表：

表 3-67. 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 3-68. 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 3-69. 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 3-70. 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_single_data_calculate

函数crc_single_data_calculate描述见下表：

表 3-71. 函数 crc_single_data_calculate

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

例如：

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

函数 crc_block_data_calculate

函数crc_block_data_calculate描述见下表：

表 3-72. 函数 crc_block_data_calculate

函数名称	crc_block_data_calculate
函数原形	uint32_t crc_block_data_calculate(uint32_t array[], uint32_t size);

功能描述	CRC计算一个32位数组
先决条件	-
被调用函数	-
输入参数{in}	
array	32位数据数组的指针
输入参数{in}	
size	数据长度
输出参数{out}	
-	-
返回值	
uint32_t	32位CRC计算结果 (0-0xFFFFFFFF)

例如：

```
/* CRC calculate a 32-bit data array */
#define BUFFER_SIZE       6
uint32_t valcrc = 0;
static const uint32_t data_buffer[BUFFER_SIZE] = {
0x00001111, 0x00002222, 0x00003333, 0x00004444, 0x00005555, 0x00006666};
valcrc = crc_block_data_calculate((uint32_t *) data_buffer, BUFFER_SIZE);
```

3.5. 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*3.5.1*描述了CTC的寄存器列表，章节*3.5.2*对CTC库函数进行说明。

3.5.1. 外设寄存器说明

CTC寄存器列表如下表所示：

表 3-73. CTC 寄存器

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

3.5.2. 外设库函数说明

CTC库函数列表如下表所示：

表 3-74. CTC 库函数

库函数名称	库函数描述
ctc_deinit	复位CTC单元
ctc_counter_enable	使能CTC校准
ctc_counter_disable	禁能CTC校准
ctc_irc48m_trim_value_config	配置IRC48M时钟校准值
ctc_software_refsource_pulse_generate	产生CTC参考时钟源同步脉冲
ctc_hardware_trim_mode_config	CTC硬件自动校准模式配置
ctc_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_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_interrupt_flag_get	CTC中断标志获取
ctc_interrupt_flag_clear	CTC中断标志清除
ctc_flag_get	CTC状态标志获取
ctc_flag_clear	CTC状态标志清除

函数 ctc_deinit

函数ctc_deinit描述见下表：

表 3-75. 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_counter_enable

函数ctc_counter_enable描述见下表：

表 3-76. 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 3-77. 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 3-78. 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 3-79. 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 3-80. 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_refsource_polarity_config

函数ctc_refsource_polarity_config描述见下表：

表 3-81. 函数 ctc_refsource_polarity_config

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

例如：

```
/* set reference source polarity */
ctc_refsource_polarity_config (CTC_REFSOURCE_POLARITY_RISING);
```

函数 ctc_refsource_signal_select

函数ctc_refsource_signal_select描述见下表：

表 3-82. 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
CTC_REFSOURCE_GPIO	选择GPIO输入信号
CTC_REFSOURCE	选择LXTAL时钟

_LXTAL	
输出参数{out}	
-	-
返回值	
-	-

例如：

/* reference signal selection */

ctc_refsource_signal_select (CTC_REFSOURCE_LXTAL);

函数 ctc_refsource_prescaler_config

函数ctc_refsource_prescaler_config描述见下表：

表 3-83. 函数 ctc_refsource_prescaler_config

函数名称	ctc_refsource_prescaler_config
函数原形	void ctc_refsource_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
CTC_REFSOURCE_PSC_OFF	参考信号不分频
CTC_REFSOURCE_PSC_DIV2	参考信号2分频
CTC_REFSOURCE_PSC_DIV4	参考信号4分频
CTC_REFSOURCE_PSC_DIV8	参考信号8分频
CTC_REFSOURCE_PSC_DIV16	参考信号16分频
CTC_REFSOURCE_PSC_DIV32	参考信号32分频
CTC_REFSOURCE_PSC_DIV64	参考信号64分频
CTC_REFSOURCE_PSC_DIV128	参考信号128分频
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* configure reference signal source prescaler */
ctc_refsource_prescaler_config(CTC_REFSOURCE_PSC_DIV2);
```

函数 ctc_clock_limit_value_config

函数ctc_clock_limit_value_config描述见下表：

表 3-84. 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 3-85. 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_capture_value_read

函数ctc_counter_capture_value_read描述见下表：

表 3-86. 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 3-87. 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 3-88. 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 3-89. 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 3-90. 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
CTC_INT_CKOK	时钟校准完成中断
CTC_INT_CKWARN	时钟校准警告中断
CTC_INT_ERR	错误中断
CTC_INT_EREF	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 3-91. 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
CTC_INT_CKOK	时钟校准完成中断
CTC_INT_CKWARN	时钟校准警告中断
CTC_INT_ERR	错误中断
CTC_INT_EREF	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_interrupt_flag_get

函数ctc_interrupt_flag_get描述见下表：

表 3-92. 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 interrupt);
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}	
interrupt	CTC中断标志
CTC_INT_FLAG_CKOK	时钟校准完成中断标志位
CTC_INT_FLAG_CKWARN	时钟校准警告中断标志位
CTC_INT_FLAG_ERR	错误中断标志位
CTC_INT_FLAG_EREF	期望参考信号中断标志位

CTC_INT_FLAG_CKERR	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 3-93. 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 interrupt);
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}	
interrupt	CTC中断标志
CTC_INT_FLAG_CKOK	时钟校准完成中断标志位
CTC_INT_FLAG_CKWARN	时钟校准警告中断标志位
CTC_INT_FLAG_ERR	错误中断标志位
CTC_INT_FLAG_EREF	期望参考信号中断标志位
CTC_INT_FLAG_CKERR	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);

函数 ctc_flag_get

函数ctc_flag_get描述见下表：

表 3-94. 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 3-95. 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);
```

3.6. 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*3.6.1*描述了DAC的寄存器列表，章节*3.6.2*对DAC库函数进行说明。

3.6.1. 外设寄存器说明

DAC寄存器列表如下表所示：

表 3-96. DAC 寄存器

寄存器名称	寄存器描述
DAC_CTL0	DACx控制寄存器
DAC_SWT	DACx软件触发寄存器
DAC_OUT0_R12DH	DACx_OUT0 12位右对齐数据保持寄存器
DAC_OUT0_L12DH	DACx_OUT0 12位左对齐数据保持寄存器
DAC_OUT0_R8DH	DACx_OUT0 8位右对齐数据保持寄存器
DAC_OUT1_R12DH	DACx_OUT1 12位右对齐数据保持寄存器
DAC_OUT1_L12DH	DACx_OUT1 12位左对齐数据保持寄存器
DAC_OUT1_R8DH	DACx_OUT1 8位右对齐数据保持寄存器
DACC_R12DH	DACx并发模式12位右对齐数据保持寄存器
DACC_L12DH	DACx并发模式12位左对齐数据保持寄存器
DACC_R8DH	DACx并发模式8位右对齐数据保持寄存器
DAC_OUT0_DO	DACx_OUT0数据输出寄存器
DAC_OUT1_DO	DACx_OUT1数据输出寄存器

3.6.2. 外设库函数说明

DAC库函数列表如下表所示：

表 3-97. DAC 库函数

库函数名称	库函数描述
dac_deinit	DAC外设复位
dac_enable	DAC使能
dac_disable	DAC禁能
dac_dma_enable	DAC的DMA功能使能
dac_dma_disable	DAC的DMA功能禁能
dac_output_buffer_enable	DAC输出缓冲区使能
dac_output_buffer_disable	DAC输出缓冲区禁能
dac_output_value_get	DAC输出数据获取
dac_data_set	DAC输出数据设置
dac_trigger_enable	DAC触发使能
dac_trigger_disable	DAC触发禁能
dac_trigger_source_config	DAC触发源配置
dac_software_trigger_enable	DAC软件触发使能
dac_wave_mode_config	DAC噪声波模式配置
dac_lfsr_noise_config	DAC LFSR模式配置
dac_triangle_noise_config	DAC三角波模式配置
dac_concurrent_enable	并发DAC模式使能
dac_concurrent_disable	并发DAC模式禁能
dac_concurrent_software_trigger_enable	并发DAC模式软件触发使能
dac_concurrent_output_buffer_enable	并发DAC模式输出缓冲区使能

库函数名称	库函数描述
dac_concurrent_output_buffer_disable	并发DAC模式输出缓冲区禁能
dac_concurrent_data_set	并发DAC模式输出数据设置

函数 dac_deinit

函数dac_deinit描述见下表：

表 3-98. 函数 dac_deinit

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

例如:

```
/* deinitialize DAC0 */
dac_deinit(DAC0);
```

函数 dac_enable

函数dac_enable描述见下表：

表 3-99. 函数 dac_enable

函数名称	dac_enable
函数原型	void dac_enable(uint32_t dac_periph, uint8_t dac_out);
功能描述	DAC使能
先决条件	-
被调用函数	-
输入参数{in}	
dac_periph	DAC外设
DACx	DAC外设选择(x = 0)
输入参数{in}	
dac_out	DAC输出
DAC_OUTx	DAC输出通道选择(x = 0,1)

输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* enable DAC0_OUT0 */
dac_enable(DAC0, DAC_OUT0);
```

函数 dac_disable

函数dac_disable描述见下表:

表 3-100. 函数 dac_disable

函数名称	dac_disable
函数原型	void dac_disable(uint32_t dac_periph, uint8_t dac_out);
功能描述	DAC禁能
先决条件	-
被调用函数	-
输入参数{in}	
dac_periph	DAC外设
DACx	DAC外设选择(x = 0)
输入参数{in}	
dac_out	DAC输出
DAC_OUTx	DAC输出通道选择(x = 0,1)
输出参数{out}	
-	-
返回值	
-	-

例如:

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

函数 dac_dma_enable

函数dac_dma_enable描述见下表:

表 3-101. 函数 dac_dma_enable

函数名称	dac_dma_enable
函数原型	void dac_dma_enable(uint32_t dac_periph, uint8_t dac_out);
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}	
dac_periph	DAC外设
DACx	DAC外设选择(x = 0)
输入参数{in}	
dac_out	DAC输出
DAC_OUTx	DAC输出通道选择(x = 0,1)
输出参数{out}	
-	-
返回值	
-	-

例如:

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

函数 dac_dma_disable

函数dac_dma_disable描述见下表:

表 3-102. 函数 dac_dma_disable

函数名称	dac_dma_disable
函数原型	void dac_dma_disable(uint32_t dac_periph, uint8_t dac_out);
功能描述	DAC的DMA功能禁能
先决条件	-
被调用函数	-
输入参数{in}	
dac_periph	DAC外设
DACx	DAC外设选择(x = 0)
输入参数{in}	
dac_out	DAC输出
DAC_OUTx	DAC输出通道选择(x = 0,1)
输出参数{out}	
-	-
返回值	
-	-

例如:

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

函数 dac_output_buffer_enable

函数dac_output_buffer_enable描述见下表：

表 3-103. 函数 dac_output_buffer_enable

函数名称	dac_output_buffer_enable
函数原型	void dac_output_buffer_enable(uint32_t dac_periph, uint8_t dac_out);
功能描述	DAC输出缓冲区使能
先决条件	-
被调用函数	-
输入参数{in}	
dac_periph	DAC外设
DACx	DAC外设选择(x = 0)
输入参数{in}	
dac_out	DAC输出
DAC_OUTx	DAC输出通道选择(x = 0,1)
输出参数{out}	
-	-
返回值	
-	-

例如:

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

函数 dac_output_buffer_disable

函数dac_output_buffer_disable描述见下表：

表 3-104. 函数 dac_output_buffer_disable

函数名称	dac_output_buffer_disable
函数原型	void dac_output_buffer_disable(uint32_t dac_periph, uint8_t dac_out);
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}	
dac_periph	DAC外设
DACx	DAC外设选择(x = 0)
输入参数{in}	
dac_out	DAC输出
DAC_OUTx	DAC输出通道选择(x = 0,1)
输出参数{out}	

-	-
返回值	
-	-

例如:

/* disable DAC0_OUT0 output buffer */

dac_output_buffer_disable(DAC0, DAC_OUT0);

函数 dac_output_value_get

函数dac_output_value_get描述见下表:

表 3-105. 函数 dac_output_value_get

函数名称	dac_output_value_get
函数原型	uint16_t dac_output_value_get(uint32_t dac_periph, uint8_t dac_out);
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}	
dac_periph	DAC外设
DACx	DAC外设选择(x = 0)
输入参数{in}	
dac_out	DAC输出
DAC_OUTx	DAC输出通道选择(x = 0,1)
输出参数{out}	
-	-
返回值	
uint16_t	外设DACx数据保持寄存器值(0~4095)

例如:

/* get the DAC0_OUT0 last data output value */

uint16_t data=0;

data = dac_output_value_get(DAC0, DAC_OUT0);

函数 dac_data_set

函数dac_data_set描述见下表:

表 3-106. 函数 dac_data_set

函数名称	dac_data_set
函数原型	void dac_data_set(uint32_t dac_periph, uint8_t dac_out, 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}	
dac_periph	DAC外设
DACx	DAC外设选择(x = 0)
输入参数{in}	
dac_out	DAC输出
DAC_OUTx	DAC输出通道选择(x = 0,1)
输入参数{in}	
dac_align	DAC对齐模式
DAC_ALIGN_12B_R	12位数据右对齐
DAC_ALIGN_12B_L	12位数据左对齐
DAC_ALIGN_8B_R	8位数据右对齐
输入参数{in}	
data	写入DAC_OUTx的数据(0~4095)
输出参数{out}	
-	-
返回值	
-	-

例如:

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

函数 dac_trigger_enable

函数dac_trigger_enable描述见下表:

表 3-107. 函数 dac_trigger_enable

函数名称	dac_trigger_enable
函数原型	void dac_trigger_enable(uint32_t dac_periph, uint8_t dac_out);
功能描述	DAC触发使能
先决条件	-
被调用函数	-
输入参数{in}	
dac_periph	DAC外设
DACx	DAC外设选择(x = 0)
输入参数{in}	
dac_out	DAC输出

DAC_OUTx	DAC输出通道选择(x = 0,1)
输出参数{out}	
-	-
返回值	
-	-

例如:

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

函数 dac_trigger_disable

函数dac_trigger_disable描述见下表:

表 3-108. 函数 dac_trigger_disable

函数名称	dac_trigger_disable
函数原型	void dac_trigger_disable(uint32_t dac_periph, uint8_t dac_out);
功能描述	DAC触发禁能
先决条件	-
被调用函数	-
输入参数{in}	
dac_periph	DAC外设
DACx	DAC外设选择(x = 0)
输入参数{in}	
dac_out	DAC输出
DAC_OUTx	DAC输出通道选择(x = 0,1)
输出参数{out}	
-	-
返回值	
-	-

例如:

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

函数 dac_trigger_source_config

函数dac_trigger_source_config描述见下表:

表 3-109. 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 dac_periph, uint8_t dac_out, uint32_t

	triggersource);
功能描述	DAC触发源配置
先决条件	-
被调用函数	-
输入参数{in}	
dac_periph	DAC外设
DACx	DAC外设选择(x = 0)
输入参数{in}	
dac_out	DAC输出
DAC_OUTx	DAC输出通道选择(x = 0,1)
输入参数{in}	
triggersource	DAC触发源
DAC_TRIGGER_T5_TRGO	TIMER5 TRGO
DAC_TRIGGER_T2_TRGO	TIMER2 TRGO
DAC_TRIGGER_T6_TRGO	TIMER6 TRGO
DAC_TRIGGER_T4_TRGO	TIMER4 TRGO
DAC_TRIGGER_T1_TRGO	TIMER1 TRGO
DAC_TRIGGER_T3_TRGO	TIMER3 TRGO
DAC_TRIGGER_EXTI_9	EXTI线9中断
DAC_TRIGGER_SOFTWARE	软件触发
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* configure DAC0_OUT0 trigger source */
dac_trigger_source_config(DAC0, DAC_OUT0, DAC_TRIGGER_T1_TRGO);
```

函数 dac_software_trigger_enable

函数dac_software_trigger_enable描述见下表:

表 3-110. 函数 dac_software_trigger_enable

函数名称	dac_software_trigger_enable
函数原型	void dac_software_trigger_enable(uint32_t dac_periph, uint8_t dac_out);
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}	
dac_periph	DAC外设
DACx	DAC外设选择(x = 0)
输入参数{in}	
dac_out	DAC输出
DAC_OUTx	DAC输出通道选择(x = 0,1)
输出参数{out}	
-	-
返回值	
-	-

例如:

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

函数 dac_wave_mode_config

函数dac_wave_mode_config描述见下表:

表 3-111. 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 dac_periph, uint8_t dac_out, uint32_t wave_mode);
功能描述	DAC噪声波模式配置
先决条件	-
被调用函数	-
输入参数{in}	
dac_periph	DAC外设
DACx	DAC外设选择(x = 0)
输入参数{in}	
dac_out	DAC输出
DAC_OUTx	DAC输出通道选择(x = 0,1)
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 DAC0_OUT0 wave mode */
dac_wave_mode_config(DAC0, DAC_OUT0, DAC_WAVE_DISABLE);
```

函数 dac_lfsr_noise_config

函数dac_lfsr_noise_config描述见下表:

表 3-112. 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 dac_periph, uint8_t dac_out, uint32_t unmask_bits);
功能描述	DAC LFSR模式配置
先决条件	-
被调用函数	-
输入参数{in}	
dac_periph	DAC外设
DACx	DAC外设选择(x = 0)
输入参数{in}	
dac_out	DAC输出
DAC_OUTx	DAC输出通道选择(x = 0,1)
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(x = 1,2,3..11)
输出参数{out}	
-	-
返回值	
-	-

例如:

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

dac_lfsr_noise_config(DAC0, DAC_OUT0, DAC_LFSR_BIT0);

函数 dac_triangle_noise_config

函数dac_triangle_noise_config描述见下表：

表 3-113. 函数 dac_triangle_noise_config

函数名称	dac_triangle_noise_config
函数原型	void dac_triangle_noise_config(uint32_t dac_periph, uint8_t dac_out, uint32_t amplitude);
功能描述	DAC三角波模式配置
先决条件	-
被调用函数	-
输入参数{in}	
dac_periph	DAC外设
DACx	DAC外设选择(x = 0)
输入参数{in}	
dac_out	DAC输出
DAC_OUTx	DAC输出通道选择(x = 0,1)
输入参数{in}	
amplitude	三角波幅值
DAC_TRIANGLE_AMPLITUDE_x	x = 2^n-1(n = 1..12)
输出参数{out}	
-	-
返回值	
-	-

例如:

/* configure DAC0_OUT0 triangle noise mode */

dac_triangle_noise_config(DAC0, DAC_OUT0, DAC_TRIANGLE_AMPLITUDE_1);

函数 dac_concurrent_enable

函数dac_concurrent_enable描述见下表：

表 3-114. 函数 dac_concurrent_enable

函数名称	dac_concurrent_enable
函数原型	void dac_concurrent_enable(uint32_t dac_periph);
功能描述	并发DAC模式使能
先决条件	-
被调用函数	-

输入参数{in}	
dac_periph	DAC外设
DACx	DAC外设选择(x = 0)
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* enable DAC0 concurrent mode */
dac_concurrent_enable(DAC0);
```

函数 dac_concurrent_disable

函数dac_concurrent_disable描述见下表:

表 3-115. 函数 dac_concurrent_disable

函数名称	dac_concurrent_disable
函数原型	void dac_concurrent_disable(uint32_t dac_periph);
功能描述	并发DAC模式禁能
先决条件	-
被调用函数	-
输入参数{in}	
dac_periph	DAC外设
DACx	DAC外设选择(x = 0)
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* disable DAC0 concurrent mode */
dac_concurrent_disable(DAC0);
```

函数 dac_concurrent_software_trigger_enable

函数dac_concurrent_software_trigger_enable描述见下表:

表 3-116. 函数 dac_concurrent_software_trigger_enable

函数名称	dac_concurrent_software_trigger_enable
函数原型	void dac_concurrent_software_trigger_enable(uint32_t dac_periph);
功能描述	并发DAC模式软件触发使能

先决条件	-
被调用函数	-
输入参数{in}	
dac_periph	DAC外设
DACx	DAC外设选择(x = 0)
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* enable DAC0 concurrent software trigger */
dac_concurrent_software_trigger_enable(DAC0);
```

函数 dac_concurrent_output_buffer_enable

函数dac_concurrent_output_buffer_enable描述见下表:

表 3-117. 函数 dac_concurrent_output_buffer_enable

函数名称	dac_concurrent_output_buffer_enable
函数原型	void dac_concurrent_output_buffer_enable(uint32_t dac_periph);
功能描述	并发DAC模式输出缓冲区使能
先决条件	-
被调用函数	-
输入参数{in}	
dac_periph	DAC外设
DACx	DAC外设选择(x = 0)
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* enable DAC0 concurrent buffer function */
dac_concurrent_output_buffer_enable(DAC0);
```

函数 dac_concurrent_output_buffer_disable

函数dac_concurrent_output_buffer_disable描述见下表:

表 3-118. 函数 dac_concurrent_output_buffer_disable

函数名称	dac_concurrent_output_buffer_disable

函数原型	void dac_concurrent_output_buffer_disable(uint32_t dac_periph);
功能描述	并发DAC模式输出缓冲区禁能
先决条件	-
被调用函数	-
输入参数{in}	
dac_periph	DAC外设
DACx	DAC外设选择(x = 0)
输出参数{out}	
-	-
返回值	
-	-

例如:

/* disable DAC0 concurrent buffer function */

dac_concurrent_output_buffer_disable(DAC0);

函数 dac_concurrent_data_set

函数dac_concurrent_data_set描述见下表:

表 3-119. 函数 dac_concurrent_data_set

函数名称	dac_concurrent_data_set
函数原型	void dac_concurrent_data_set(uint32_t dac_periph, uint32_t dac_align, uint16_t data0, uint16_t data1);
功能描述	并发DAC模式输出数据设置
先决条件	-
被调用函数	-
输入参数{in}	
dac_periph	DAC外设
DACx	DAC外设选择(x = 0)
输入参数{in}	
dac_align	DAC对齐模式
DAC_ALIGN_8B_R	8位数据右对齐
DAC_ALIGN_12B_R	12位数据右对齐
DAC_ALIGN_12B_L	12位数据左对齐
输入参数{in}	
data0	写入DACx_OUT0的数据(0~4095)
输入参数{in}	
data1	写入DACx_OUT1的数据(0~4095)
输出参数{out}	
-	-

返回值	
-	-

例如:

/* set DAC0 concurrent mode data holding register value */

dac_concurrent_data_set(DAC0, DAC_ALIGN_8B_R, 0xFF, 0xFF);

3.7. 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*3.7.1*描述了DBG的寄存器列表，章节*3.7.2*对DBG库函数进行说明。

3.7.1. 外设寄存器说明

DBG寄存器列表如下表所示：

表 3-120. DBG 寄存器

寄存器名称	寄存器描述
DBG_ID	DBG ID寄存器
DBG_CTL	DBG控制寄存器

3.7.2. 外设库函数说明

DBG库函数列表如下表所示：

表 3-121. DBG 库函数

库函数名称	库函数描述
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	禁能能外设的MCU调试保持功能
dbg_trace_pin_enable	使能跟踪引脚分配
dbg_trace_pin_disable	禁能跟踪引脚分配

枚举类型 dbg_periph_enum

表 3-122. 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_TIMER3_HOLD	当内核停止时，保持TIMER3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_TIMER4_HOLD	当内核停止时，保持TIMER4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_TIMER6_HOLD	当内核停止时，保持TIMER6计数器计数值不变
DBG_TIMER7_HOLD	当内核停止时，保持TIMER7计数器计数值不变
DBG_TIMER11_HOLD	当内核停止时，保持TIMER11计数器计数值不变
DBG_TIMER12_HOLD	当内核停止时，保持TIMER12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_TIMER8_HOLD	当内核停止时，保持TIMER8计数器计数值不变
DBG_TIMER9_HOLD	当内核停止时，保持TIMER9计数器计数值不变
DBG_TIMER10_HOLD	当内核停止时，保持TIMER10计数器计数值不变

函数 dbg_id_get

函数dbg_id_get描述见下表：

表 3-123. 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	Read DBG_ID code register

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 3-124. 函数 dbg_low_power_enable

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

输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* enable low power behavior when the mcu is in debug mode */
dbg_low_power_enable(DBG_LOW_POWER_SLEEP);
```

函数 dbg_low_power_disable

函数dbg_low_power_disable描述见下表：

表 3-125. 函数 dbg_low_power_disable

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

例如：

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

dbg_low_power_disable(DBG_LOW_POWER_SLEEP);

函数 dbg_periph_enable

函数dbg_periph_enable描述见下表：

表 3-126. 函数 dbg_periph_enable

函数名称	dbg_periph_enable
函数原形	void dbg_periph_enable(dbg_periph_enum dbg_periph);
功能描述	使能外设的MCU调试保持功能
先决条件	-
被调用函数	-
输入参数{in}	
dbg_periph	Peripheral refer to *表3-122. 枚举类型dbg_periph_enum*
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_I2Cx_HOLD	当内核停止时，保持I2Cx(x=0,1)的SMBUS状态不变，用于调试
DBG_TIMERx_HOLD	当内核停止时，保持TIMERx计数器计数值不变 (x=0,1,2,3,4,5,6,7,8,9,10,11,12,13)
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 3-127. 函数 dbg_periph_disable

函数名称	dbg_periph_disable
函数原形	void dbg_periph_disable(dbg_periph_enum dbg_periph);
功能描述	禁能外设的MCU调试保持功能
先决条件	-
被调用函数	-
输入参数{in}	
dbg_periph	Peripheral refer to *表3-122. 枚举类型dbg_periph_enum*
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_I2Cx_HOLD	当内核停止时，保持I2Cx(x=0,1)的SMBUS状态不变，用于调试
DBG_TIMERx_HOLD	当内核停止时，保持TIMERx计数器计数值不变 (x=0,1,2,3,4,5,6,7,8,9,10,11,12,13)
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);
```

函数 dbg_trace_pin_enable

函数dbg_trace_pin_enable描述见下表：

表 3-128. 函数 dbg_trace_pin_enable

函数名称	dbg_trace_pin_enable
函数原形	void dbg_trace_pin_enable(void);
功能描述	使能跟踪引脚分配
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 trace pin assignment */
dbg_trace_pin_enable();
```

函数 dbg_trace_pin_disable

函数dbg_trace_pin_disable描述见下表：

表 3-129. 函数 dbg_trace_pin_disable

函数名称	dbg_trace_pin_disable
函数原形	void dbg_trace_pin_disable(void);
功能描述	禁能跟踪引脚分配
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 trace pin assignment */
dbg_trace_pin_disable();
```

3.8. 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*3.8.1*描述了DMA的寄存器列表，章节*3.8.2*对DMA库函数进行说明。

3.8.1. 外设寄存器说明

DMA寄存器列表如下表所示：

表 3-130. 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外设基地址寄存器
DMA_CHxMADDR (x=0..6)	通道x存储器基地址寄存器

3.8.2. 外设库函数说明

DMA库函数列表如下表所示：

表 3-131. DMA 库函数

库函数名称	库函数描述
dma_deinit	复位外设DMAx的通道y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DMAx的通道y
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	外设DMAx的通道y传输使能
dma_channel_disable	外设DMAx的通道y传输禁能
dma_periph_address_config	DMAx通道y传输的外设基地址配置
dma_memory_address_config	DMAx通道y传输的存储器基地址配置
dma_transfer_number_config	配置DMAx通道y还有多少数据要传输
dma_transfer_number_get	获取DMAx通道y还有多少数据要传输

库函数名称	库函数描述
dma_priority_config	DMAx通道y的传输软件优先级配置
dma_memory_width_config	DMAx通道y传输的存储器数据宽度配置
dma_periph_width_config	DMAx通道y传输的外设数据宽度配置
dma_memory_increase_enable	DMAx通道y传输的存储器地址生成算法增量模式使能
dma_memory_increase_disable	DMAx通道y传输的存储器地址生成算法增量模式禁能
dma_periph_increase_enable	DMAx通道y传输的外设地址生成算法增量模式使能
dma_periph_increase_disable	DMAx通道y传输的外设地址生成算法增量模式禁能
dma_transfer_direction_config	DMAx通道y的传输方向配置
dma_flag_get	获取DMAx通道y标志位状态
dma_flag_clear	清除DMAx通道y标志位状态
dma_interrupt_flag_get	获取DMAx通道y中断标志位状态
dma_interrupt_flag_clear	清除DMAx通道y中断标志位状态
dma_interrupt_enable	DMAx通道y中断使能
dma_interrupt_disable	DMAx通道y中断禁能

结构体 dma_parameter_struct

表 3-132. 结构体 dma_parameter_struct

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

函数 dma_deinit

函数dma_deinit描述见下表：

表 3-133. 函数 dma_deinit

函数名称	dma_deinit
函数原型	void dma_deinit(uint32_t dma_periph, dma_channel_enum channelx);
功能描述	复位外设DMAx的通道y的所有寄存器
先决条件	无
被调用函数	无
输入参数{in}	
dma_periph	DMA外设

DMAx(x=0,1)	DMA外设选择
输入参数{in}	
channelx	DMA通道
DMA_CHx(DMA0:x=0..6; DMA1:x=0..4)	DMA通道选择
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* DMA0 channel0 initialize */
dma_deinit(DMA0, DMA_CH0);
```

函数 dma_struct_para_init

函数dma_struct_para_init描述见下表：

表 3-134. 函数 dma_struct_para_init

函数名称	dma_struct_para_init
函数原型	void dma_struct_para_init(dma_parameter_struct* init_struct);
功能描述	将DMA结构体中所有参数初始化为默认值
先决条件	无
被调用函数	无
输入参数{in}	
***init_struct**	一个已经定义的dma_parameter_struct结构体变量地址
输出参数{out}	
-	-
返回值	
-	-

例如：

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

函数 dma_init

函数dma_init描述见下表：

表 3-135. 函数 dma_init

函数名称	dma_init

函数原型	void dma_init(uint32_t dma_periph, dma_channel_enum channelx, dma_parameter_struct* init_struct);
功能描述	初始化外设DMAx的通道y
先决条件	无
被调用函数	无
输入参数{in}	
dma_periph	DMA外设
DMAx(x=0,1)	DMA外设选择
输入参数{in}	
channelx	DMA通道
DMA_CHx(DMA0:x=0..6; DMA1:x=0..4)	DMA通道选择
输入参数{in}	
init_struct	初始化结构体，结构体成员参考*表3-132. 结构体dma_parameter_struct*
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* DMA0 channel0 initialize */
dma_parameter_struct dma_init_struct;
dma_deinit(DMA0, DMA_CH0);
dma_struct_para_init(&dma_init_struct);

dma_init_struct.direction = DMA_PERIPHERAL_TO_MEMORY;
dma_init_struct.memory_addr = (uint32_t)g_destbuf;
dma_init_struct.memory_inc = DMA_MEMORY_INCREASE_ENABLE;
dma_init_struct.memory_width = DMA_MEMORY_WIDTH_8BIT;
dma_init_struct.number = TRANSFER_NUM;
dma_init_struct.periph_addr = (uint32_t)BANK0_WRITE_START_ADDR;
dma_init_struct.periph_inc = DMA_PERIPH_INCREASE_ENABLE;
dma_init_struct.periph_width = DMA_PERIPHERAL_WIDTH_8BIT;
dma_init_struct.priority = DMA_PRIORITY_ULTRA_HIGH;
dma_init(DMA0, DMA_CH0, &dma_init_struct);
```

函数 dma_circulation_enable

函数dma_circulation_enable描述见下表：

表 3-136. 函数 dma_circulation_enable

函数名称	dma_circulation_enable

函数原型	void dma_circulation_enable(uint32_t dma_periph, 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}	
dma_periph	DMA外设
DMAx(x=0,1)	DMA外设选择
输入参数{in}	
channelx	DMA通道
DMA_CHx(DMA0:x=0..6; DMA1:x=0..4)	DMA通道选择
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* enable DMA0 channel0 circulation mode */
dma_circulation_enable(DMA0, DMA_CH0);
```

函数 dma_circulation_disable

函数dma_circulation_disable描述见下表：

表 3-137. 函数 dma_circulation_disable

函数名称	dma_circulation_disable
函数原型	void dma_circulation_disable(uint32_t dma_periph, 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}	
dma_periph	DMA外设
DMAx(x=0,1)	DMA外设选择
输入参数{in}	
channelx	DMA通道
DMA_CHx(DMA0:x=0..6; DMA1:x=0..4)	DMA通道选择
输出参数{out}	
-	-

返回值	
-	-

例如：

```
/* disable DMA0 channel0 circulation mode */
dma_circulation_disable(DMA0, DMA_CH0);
```

函数 dma_memory_to_memory_enable

函数dma_memory_to_memory_enable描述见下表：

表 3-138. 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(uint32_t dma_periph, 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}	
dma_periph	DMA外设
DMAx(x=0,1)	DMA外设选择
输入参数{in}	
channelx	DMA通道
DMA_CHx(DMA0:x=0..6; DMA1:x=0..4)	DMA通道选择
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* enable DMA0 channel0 memory to memory mode */
dma_memory_to_memory_enable(DMA0, DMA_CH0);
```

函数 dma_memory_to_memory_disable

函数dma_memory_to_memory_disable描述见下表：

表 3-139. 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(uint32_t dma_periph, 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}	
dma_periph	DMA外设
DMAx(x=0,1)	DMA外设选择
输入参数{in}	
channelx	DMA通道
DMA_CHx(DMA0:x=0..6; DMA1:x=0..4)	DMA通道选择
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* disable DMA0 channel0 memory to memory mode */
dma_memory_to_memory_enable(DMA0, DMA_CH0);
```

函数 dma_channel_enable

函数dma_channel_enable描述见下表：

表 3-140. 函数 dma_channel_enable

函数名称	dma_channel_enable
函数原型	void dma_channel_enable(uint32_t dma_periph, dma_channel_enum channelx);
功能描述	外设DMAx的通道y传输使能
先决条件	无
被调用函数	无
输入参数{in}	
dma_periph	DMA外设
DMAx(x=0,1)	DMA外设选择
输入参数{in}	
channelx	DMA通道
DMA_CHx(DMA0:x=0..6; DMA1:x=0..4)	DMA通道选择
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* enable DMA0 channel0 */
dma_channel_enable(DMA0, DMA_CH0);
```

函数 dma_channel_disable

函数dma_channel_disable描述见下表：

表 3-141. 函数 dma_channel_disable

函数名称	dma_channel_disable
函数原型	void dma_channel_disable(uint32_t dma_periph, dma_channel_enum channelx);
功能描述	外设DMAx的通道y传输禁能
先决条件	无
被调用函数	无
输入参数{in}	
dma_periph	DMA外设
DMAx(x=0,1)	DMA外设选择
输入参数{in}	
channelx	DMA通道
DMA_CHx(DMA0:x=0..6; DMA1:x=0..4)	DMA通道选择
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* disable DMA0 channel0 */
dma_channel_disable(DMA0, DMA_CH0);
```

函数 dma_periph_address_config

函数dma_periph_address_config描述见下表：

表 3-142. 函数 dma_periph_address_config

函数名称	dma_periph_address_config
函数原型	void dma_periph_address_config(uint32_t dma_periph, dma_channel_enum channelx, uint32_t address);
功能描述	DMAx通道y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}	
dma_periph	DMA外设
DMAx(x=0,1)	DMA外设选择
输入参数{in}	
channelx	DMA通道
DMA_CHx(DMA0:x=0..6; DMA1:x=0..4)	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：

```
#define BANK0_WRITE_START_ADDR                ((uint32_t)0x08004000)

dma_periph_address_config(DMA0, DMA_CH0, BANK0_WRITE_START_ADDR);
```

函数 dma_memory_address_config

函数dma_memory_address_config描述见下表：

表 3-143. 函数 dma_memory_address_config

函数名称	dma_memory_address_config
函数原型	void dma_memory_address_config(uint32_t dma_periph, dma_channel_enum channelx, uint32_t address);
功能描述	DMAx通道y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}	
dma_periph	DMA外设
DMAx(x=0,1)	DMA外设选择
输入参数{in}	
channelx	DMA通道
DMA_CHx(DMA0:x=0..6; DMA1:x=0..4)	DMA通道选择
输入参数{in}	
address	存储器基地址
输出参数{out}	
-	-

返回值	
-	-

例如：

uint8_t g_destbuf[TRANSFER_NUM];

dma_memory_address_config(DMA0, DMA_CH0, (uint32_t) g_destbuf);

函数 dma_transfer_number_config

函数dma_transfer_number_config描述见下表：

表 3-144. 函数 dma_transfer_number_config

函数名称	dma_transfer_number_config
函数原型	void dma_transfer_number_config(uint32_t dma_periph, dma_channel_enum channelx, uint32_t number);
功能描述	配置DMAx通道y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}	
dma_periph	DMA外设
DMAx(x=0,1)	DMA外设选择
输入参数{in}	
channelx	DMA通道
DMA_CHx(DMA0:x=0..6; DMA1:x=0..4)	DMA通道选择
输入参数{in}	
number	数据传输数量(0x0 - 0xFFFF)
输出参数{out}	
-	-
返回值	
-	-

例如：

#define TRANSFER_NUM 0x400

dma_transfer_number_config(DMA0, DMA_CH0, TRANSFER_NUM);

函数 dma_transfer_number_get

函数dma_transfer_number_get描述见下表：

表 3-145. 函数 dma_transfer_number_get

函数名称	dma_transfer_number_get
函数原型	uint32_t dma_transfer_number_get(uint32_t dma_periph, dma_channel_enum channelx);
功能描述	获取DMAx通道y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}	
dma_periph	DMA外设
DMAx(x=0,1)	DMA外设选择
输入参数{in}	
channelx	DMA通道
DMA_CHx(DMA0:x=0..6; DMA1:x=0..4)	DMA通道选择
输出参数{out}	
-	-
返回值	
uint32_t	DMA数据传输剩余数量(0x0 - 0xFFFF)

例如：

```
uint32_t number = 0;
number = dma_transfer_number_get(DMA0, DMA_CH0);
```

函数 dma_priority_config

函数dma_priority_config描述见下表：

表 3-146. 函数 dma_priority_config

函数名称	dma_priority_config
函数原型	void dma_priority_config(uint32_t dma_periph, dma_channel_enum channelx, uint32_t priority);
功能描述	DMAx通道y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}	
dma_periph	DMA外设
DMAx(x=0,1)	DMA外设选择
输入参数{in}	
channelx	DMA通道
DMA_CHx(DMA0:x=0..6; DMA1:	DMA通道选择

x=0..4)	
输入参数{in}	
priority	DMA通道软件优先级
DMA_PRIORITY_LOW	低优先级
DMA_PRIORITY_MEDIUM	中优先级
DMA_PRIORITY_HIGH	高优先级
DMA_PRIORITY_ULTRA_HIGH	极高优先级
输出参数{out}	
-	-
返回值	
-	-

例如：

dma_priority_config(DMA0, DMA_CH0, DMA_PRIORITY_ULTRA_HIGH);

函数 dma_memory_width_config

函数dma_memory_width_config描述见下表：

表 3-147. 函数 dma_memory_width_config

函数名称	dma_memory_width_config
函数原型	void dma_memory_width_config (uint32_t dma_periph, dma_channel_enum channelx, uint32_t mwidth);
功能描述	DMAx通道y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}	
dma_periph	DMA外设
DMAx(x=0,1)	DMA外设选择
输入参数{in}	
channelx	DMA通道
DMA_CHx(DMA0:x=0..6; DMA1:x=0..4)	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_WI	16位数据传输宽度

DTH_16BIT	
DMA_MEMORY_WIDTH_32BIT	32位数据传输宽度
输出参数{out}	
-	-
返回值	
-	-

例如：

```
dma_memory_width_config(DMA0, DMA_CH0, DMA_MEMORY_WIDTH_8BIT);
```

函数 dma_periph_width_config

函数dma_periph_width_config描述见下表：

表 3-148. 函数 dma_periph_width_config

函数名称	dma_periph_width_config
函数原型	void dma_periph_width_config (uint32_t dma_periph, dma_channel_enum channelx, uint32_t pwidth);
功能描述	DMAx通道y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}	
dma_periph	DMA外设
DMAx(x=0,1)	DMA外设选择
输入参数{in}	
channelx	DMA通道
DMA_CHx(DMA0:x=0..6; DMA1:x=0..4)	DMA通道选择
输入参数{in}	
pwidth	外设数据传输宽度
DMA_PERIPHERAL_WIDTH_8BIT	8位数据传输宽度
DMA_PERIPHERAL_WIDTH_16BIT	16位数据传输宽度
DMA_PERIPHERAL_WIDTH_32BIT	32位数据传输宽度
输出参数{out}	
-	-
返回值	
-	-

例如：

dma_periph_width_config(DMA0, DMA_CH0, DMA_PERIPHERAL_WIDTH_8BIT);

函数 dma_memory_increase_enable

函数dma_memory_increase_enable描述见下表：

表 3-149. 函数 dma_memory_increase_enable

函数名称	dma_memory_increase_enable
函数原型	void dma_memory_increase_enable(uint32_t dma_periph, dma_channel_enum channelx);
功能描述	DMAx通道y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}	
dma_periph	DMA外设
DMAx(x=0,1)	DMA外设选择
输入参数{in}	
channelx	DMA通道
DMA_CHx(DMA0:x=0..6; DMA1:x=0..4)	DMA通道选择
输出参数{out}	
-	-
返回值	
-	-

例如：

dma_memory_increase_enable(DMA0, DMA_CH0);

函数 dma_memory_increase_disable

函数dma_memory_increase_disable描述见下表：

表 3-150. 函数 dma_memory_increase_disable

函数名称	dma_memory_increase_disable
函数原型	void dma_memory_increase_disable(uint32_t dma_periph, dma_channel_enum channelx);
功能描述	DMAx通道y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}	

dma_periph	DMA外设
DMAx(x=0,1)	DMA外设选择
输入参数{in}	
channelx	DMA通道
DMA_CHx(DMA0:x=0..6; DMA1:x=0..4)	DMA通道选择
输出参数{out}	
-	-
返回值	
-	-

例如：

```
dma_memory_increase_disable(DMA0, DMA_CH0);
```

函数 dma_periph_increase_enable

函数dma_periph_increase_enable描述见下表：

表 3-151. 函数 dma_periph_increase_enable

函数名称	dma_periph_increase_enable
函数原型	void dma_periph_increase_enable(uint32_t dma_periph, dma_channel_enum channelx);
功能描述	DMAx通道y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}	
dma_periph	DMA外设
DMAx(x=0,1)	DMA外设选择
输入参数{in}	
channelx	DMA通道
DMA_CHx(DMA0:x=0..6; DMA1:x=0..4)	DMA通道选择
输出参数{out}	
-	-
返回值	
-	-

例如：

```
dma_periph_increase_enable(DMA0, DMA_CH0);
```

函数 dma_periph_increase_disable

函数dma_periph_increase_disable描述见下表：

表 3-152. 函数 dma_periph_increase_disable

函数名称	dma_periph_increase_disable
函数原型	void dma_periph_increase_disable(uint32_t dma_periph, dma_channel_enum channelx);
功能描述	DMAx通道y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}	
dma_periph	DMA外设
DMAx(x=0,1)	DMA外设选择
输入参数{in}	
channelx	DMA通道
DMA_CHx(DMA0:x=0..6; DMA1:x=0..4)	DMA通道选择
输出参数{out}	
-	-
返回值	
-	-

例如：

```
dma_periph_increase_disable(DMA0, DMA_CH0);
```

函数 dma_transfer_direction_config

函数dma_transfer_direction_config描述见下表：

表 3-153. 函数 dma_transfer_direction_config

函数名称	dma_transfer_direction_config
函数原型	void dma_transfer_direction_config(uint32_t dma_periph, dma_channel_enum channelx, uint32_t direction);
功能描述	DMAx通道y的传输方向配置
先决条件	无
被调用函数	无
输入参数{in}	
dma_periph	DMA外设
DMAx(x=0,1)	DMA外设选择
输入参数{in}	
channelx	DMA通道

DMA_CHx(DMA0:x=0..6; DMA1:x=0..4)	DMA通道选择
输入参数{in}	
direction	数据传输方向
DMA_PERIPHERAL_TO_MEMORY	读取外设中数据，写入存储器
DMA_MEMORY_TO_PERIPHERAL	读取存储器中数据，写入外设
输出参数{out}	
-	-
返回值	
-	-

例如：

```
dma_transfer_direction_config(DMA0, DMA_CH0, DMA_PERIPHERAL_TO_MEMORY);
```

函数 dma_flag_get

函数dma_flag_get描述见下表：

表 3-154. 函数 dma_flag_get

函数名称	dma_flag_get
函数原型	FlagStatus dma_flag_get(uint32_t dma_periph, dma_channel_enum channelx, uint32_t flag);
功能描述	获取DMAx通道y标志位状态
先决条件	无
被调用函数	无
输入参数{in}	
dma_periph	DMA外设
DMAx(x=0,1)	DMA外设选择
输入参数{in}	
channelx	DMA通道
DMA_CHx(DMA0:x=0..6; DMA1:x=0..4)	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：

```
FlagStatus flag = RESET;
flag = dma_flag_get(DMA0, DMA_CH0, DMA_FLAG_FTF);
```

函数 dma_flag_clear

函数dma_flag_clear描述见下表：

表 3-155. 函数 dma_flag_clear

函数名称	dma_flag_clear
函数原型	void dma_flag_clear(uint32_t dma_periph, dma_channel_enum channelx, uint32_t flag);
功能描述	清除DMAx通道y标志位状态
先决条件	无
被调用函数	无
输入参数{in}	
dma_periph	DMA外设
DMAx(x=0,1)	DMA外设选择
输入参数{in}	
channelx	DMA通道
DMA_CHx(DMA0:x=0..6; DMA1:x=0..4)	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	
-	-

例如：

```
dma_flag_clear(DMA0, DMA_CH0, DMA_FLAG_FTF);
```

函数 dma_interrupt_flag_get

函数dma_interrupt_flag_get描述见下表：

表 3-156. 函数 dma_interrupt_flag_get

函数名称	dma_interrupt_flag_get
函数原型	FlagStatus dma_interrupt_flag_get(uint32_t dma_periph, dma_channel_enum channelx, uint32_t flag);
功能描述	获取DMAx通道y中断标志位状态
先决条件	无
被调用函数	无
输入参数{in}	
dma_periph	DMA外设
DMAx(x=0,1)	DMA外设选择
输入参数{in}	
channelx	DMA通道
DMA_CHx(DMA0:x=0..6; DMA1:x=0..4)	DMA通道选择
输入参数{in}	
flag	DMA标志
DMA_INT_FLAG_FTF	DMA通道传输完成中断标志
DMA_INT_FLAG_HTF	DMA通道半传输完成中断标志
DMA_INT_FLAG_ERR	DMA通道错误中断标志
输出参数{out}	
-	-
返回值	
FlagStatus	SET或RESET

例如：

```
if(dma_interrupt_flag_get(DMA0, DMA_CH3, DMA_INT_FLAG_FTF)){
    dma_interrupt_flag_clear(DMA0, DMA_CH3, DMA_INT_FLAG_G);
}
```

函数 dma_interrupt_flag_clear

函数dma_interrupt_flag_clear描述见下表：

表 3-157. 函数 dma_interrupt_flag_clear

函数名称	dma_interrupt_flag_clear

函数原型	void dma_interrupt_flag_clear(uint32_t dma_periph, dma_channel_enum channelx, uint32_t flag);
功能描述	清除DMAx通道y中断标志位状态
先决条件	无
被调用函数	无
输入参数{in}	
dma_periph	DMA外设
DMAx(x=0,1)	DMA外设选择
输入参数{in}	
channelx	DMA通道
DMA_CHx(DMA0:x=0..6; DMA1:x=0..4)	DMA通道选择
输入参数{in}	
flag	DMA标志
DMA_INT_FLAG_G	DMA通道全局中断标志
DMA_INT_FLAG_FTF	DMA通道传输完成中断标志
DMA_INT_FLAG_HTF	DMA通道半传输完成中断标志
DMA_INT_FLAG_ERR	DMA通道错误中断标志
输出参数{out}	
-	-
返回值	
-	-

例如：

```
if(dma_interrupt_flag_get(DMA0, DMA_CH3, DMA_INT_FLAG_FTF)){
    dma_interrupt_flag_clear(DMA0, DMA_CH3, DMA_INT_FLAG_G);
}
```

函数 dma_interrupt_enable

函数dma_interrupt_enable描述见下表：

表 3-158. 函数 dma_interrupt_enable

函数名称	dma_interrupt_enable
函数原型	void dma_interrupt_enable(uint32_t dma_periph, dma_channel_enum channelx, uint32_t source);
功能描述	DMAx通道y中断使能
先决条件	无
被调用函数	无

输入参数{in}	
dma_periph	DMA外设
DMAx(x=0,1)	DMA外设选择
输入参数{in}	
channelx	DMA通道
DMA_CHx(DMA0:x=0..6; DMA1:x=0..4)	DMA通道选择
输入参数{in}	
source	DMA中断源
DMA_INT_FTF	DMA通道传输完成中断
DMA_INT_HTF	DMA通道半传输完成中断
DMA_INT_ERR	DMA通道错误中断
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* DMA0 channel0 interrupt configuration */
dma_interrupt_enable(DMA0, DMA_CH0, DMA_INT_FTF);
```

函数 dma_interrupt_disable

函数dma_interrupt_disable描述见下表：

表 3-159. 函数 dma_interrupt_disable

函数名称	dma_interrupt_disable
函数原型	void dma_interrupt_disable(uint32_t dma_periph, dma_channel_enum channelx, uint32_t source);
功能描述	DMAx通道y中断禁能
先决条件	无
被调用函数	无
输入参数{in}	
dma_periph	DMA外设
DMAx(x=0,1)	DMA外设选择
输入参数{in}	
channelx	DMA通道
DMA_CHx(DMA0:x=0..6; DMA1:x=0..4)	DMA通道选择
输入参数{in}	

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

例如：

/* DMA0 channel0 interrupt configuration */

dma_interrupt_ disable(DMA0, DMA_CH0, DMA_INT_FTF);

3.9. EXMC

外部存储器控制器EXMC，用来访问各种片外存储器。章节*3.9.1*描述了EXMC的寄存器列表，章节*3.9.2*对EXMC库函数进行说明。

3.9.1. 外设寄存器说明

EXMC寄存器列表如下表所示：

表 3-160. EXMC 寄存器

寄存器名称	寄存器描述
EXMC_SNCTL	SRAM/NOR Flash控制寄存器
EXMC_SNTCFG	SRAM/NOR Flash时序寄存器
EXMC_SNWTCFG	SRAM/NOR Flash写时序寄存器

3.9.2. 外设库函数说明

EXMC库函数列表如下表所示：

表 3-161. EXMC 库函数

库函数名称	库函数描述
exmc_norsram_deinit	复位NOR/SRAM bank
exmc_norsram_init	初始化NOR/SRAM bank
exmc_norsram_struct_para_init	初始化结构体exmc_norsram_parameter_struct
exmc_norsram_enable	使能EXMC NOR/SRAM bank
exmc_norsram_disable	禁用EXMC NOR/SRAM bank
exmc_norsram_page_size_config	配置CRAM页大小

结构体 exmc_norsram_timing_parameter_struct

表 3-162. 结构体 exmc_norsram_timing_parameter_struct

成员名称	功能描述
asyn_access_mode	异步访问模式
syn_data_latency	数据延迟
syn_clk_division	同步时钟分频比
bus_latency	总线延迟
asyn_data_setuptime	数据建立时间
asyn_address_holdtime	地址保持时间
asyn_address_setuptime	地址建立时间

结构体 exmc_norsram_parameter_struct

表 3-163. 结构体 exmc_norsram_parameter_struct

成员名称	功能描述
write_mode	写模式，同步模式或者异步模式
extended_mode	使能或者禁用扩展模式
asyn_wait	使能或者禁用异步等待功能
nwait_signal	在同步突发模式中，使能或者禁用NWAIT信号
memory_write	使能或者禁用写操作
nwait_config	配置NWAIT信号
wrap_burst_mode	使能或者禁用非对齐成组模式
nwait_polarity	指定NWAIT的极性
burst_mode	使能或者禁用突发模式
databus_width	指定外部存储器数据总线宽度
memory_type	指定外部存储器的类型
address_data_mux	数据线/地址线复用是否复用
read_write_timing	未用扩展模式时，读时序参数和写时序参数；或采用扩展模式时，读时序参数
write_timing	未用扩展模式时，写时序参数

函数 exmc_norsram_deinit

函数exmc_norsram_deinit描述见下表：

表 3-164. 函数 exmc_norsram_deinit

函数名称	exmc_norsram_deinit
函数原型	void exmc_norsram_deinit(void);
功能描述	复位NOR/SRAM bank

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 EXMC NOR/SRAM bank */
exmc_norsram_deinit();
```

函数 exmc_norsram_init

函数exmc_norsram_init描述见下表：

表 3-165. 函数 exmc_norsram_init

函数名称	exmc_norsram_init
函数原型	void exmc_norsram_init(exmc_norsram_parameter_struct* exmc_norsram_init_struct);
功能描述	初始化NOR/SRAM bank
先决条件	-
被调用函数	-
输入参数{in}	
exmc_norsram_init_struct	初始化结构体，结构体成员参考*表3-163. 结构体 exmc_norsram_parameter_struct*
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* initialize EXMC NOR/SRAM bank */
exmc_norsram_parameter_struct lcd_init_struct;
exmc_norsram_timing_parameter_struct lcd_timing_init_struct;
/* configure timing parameter */
lcd_timing_init_struct.asyn_access_mode = EXMC_ACCESS_MODE_A;
lcd_timing_init_struct.syn_data_latency = EXMC_DATALAT_2_CLK;
```

```
lcd_timing_init_struct.syn_clk_division = EXMC_SYN_CLOCK_RATIO_DISABLE;
lcd_timing_init_struct.bus_latency = 1;
lcd_timing_init_struct.asyn_data_setuptime = 5;
lcd_timing_init_struct.asyn_address_holdtime = 2;
lcd_timing_init_struct.asyn_address_setuptime = 2;
/* configure EXMC bus parameters */
lcd_init_struct.write_mode = EXMC_ASYN_WRITE;
lcd_init_struct.extended_mode = DISABLE;
lcd_init_struct.asyn_wait = DISABLE;
lcd_init_struct.nwait_signal = DISABLE;
lcd_init_struct.memory_write = ENABLE;
lcd_init_struct.nwait_config = EXMC_NWAIT_CONFIG_BEFORE;
lcd_init_struct.wrap_burst_mode = DISABLE;
lcd_init_struct.nwait_polarity = EXMC_NWAIT_POLARITY_LOW;
lcd_init_struct.burst_mode = DISABLE;
lcd_init_struct.databus_width = EXMC_NOR_DATABUS_WIDTH_16B;
lcd_init_struct.memory_type = EXMC_MEMORY_TYPE_SRAM;
lcd_init_struct.address_data_mux = DISABLE;
lcd_init_struct.read_write_timing = &lcd_timing_init_struct;
lcd_init_struct.write_timing = &lcd_timing_init_struct;
exmc_norsram_init(&lcd_init_struct);
```

函数 exmc_norsram_struct_para_init

函数exmc_norsram_struct_para_init描述见下表：

表 3-166. 函数 exmc_norsram_struct_para_init

函数名称	exmc_norsram_struct_para_init
函数原型	void exmc_norsram_struct_para_init(exmc_norsram_parameter_struct* exmc_norsram_init_struct);
功能描述	初始化结构体exmc_norsram_parameter_struct
先决条件	-
被调用函数	-

输入参数{in}	
exmc_norsram_init_struct	初始化结构体，结构体成员参考*表3-163. 结构体 exmc_norsram_parameter_struct*
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* initialize the struct nor_init_struct */
exmc_norsram_parameter_struct nor_init_struct;
exmc_norsram_struct_para_init (&nor_init_struct);
```

函数 exmc_norsram_enable

函数exmc_norsram_enable描述见下表：

表 3-167. 函数 exmc_norsram_enable

函数名称	exmc_norsram_enable
函数原型	void exmc_norsram_enable(void);
功能描述	使能EXMC NOR/SRAM bank
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 EXMC NOR/SRAM bank */
exmc_norsram_enable();
```

函数 exmc_norsram_disable

函数exmc_norsram_disable描述见下表：

表 3-168. 函数 exmc_norsram_disable

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

功能描述	禁用EXMC NOR/SRAM bank
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 EXMC NOR/SRAM bank */
exmc_norsram_disable();
```

函数 exmc_norsram_page_size_config

函数exmc_norsram_page_size_config描述见下表：

表 3-169. 函数 exmc_norsram_page_size_config

函数名称	exmc_norsram_page_size_config
函数原型	void exmc_norsram_page_size_config(uint32_t page_size);
功能描述	配置CRAM页大小
先决条件	-
被调用函数	-
输入参数{in}	
page_size	CRAM页大小
EXMC_CRAM_AUTO_SPLIT	页边界自动突发分割
EXMC_CRAM_PAGE_SIZE_128_BYTES	页大小128字节
EXMC_CRAM_PAGE_SIZE_256_BYTES	页大小256字节
EXMC_CRAM_PAGE_SIZE_512_BYTES	页大小512字节
EXMC_CRAM_PAGE_SIZE_1024_BYTES	页大小1024字节
输出参数{out}	
-	-

返回值	
-	-

例如：

/* configure CRAM page size */

exmc_norsram_page_size_config (EXMC_CRAM_PAGE_SIZE_128_BYTES);

3.10. EXTI

EXTI是MCU中的中断/事件控制器，包括19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 3-170. EXTI 寄存器

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

3.10.2. 外设库函数说明

EXTI库函数列表如下表所示：

表 3-171. EXTI 库函数

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

库函数名称	库函数描述
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 3-172. 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

枚举类型 exti_mode_enum

表 3-173. 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 3-174. 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 3-175. 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
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 3-176. 函数 exti_init

函数名称	exti_init
函数原形	void exti_init(exti_line_enum linex, exti_mode_enum mode, exti_trig_type_enum trig_type);
功能描述	初始化EXTI线x
先决条件	-
被调用函数	-
输入参数{in}	
linex	EXTI线x，参考*表3-172. 枚举类型exti_line_enum*
输入参数{in}	
mode	EXTI模式，参考*表3-173. 枚举类型exti_mode_enum*
输入参数{in}	
trig_type	触发类型，参考*表3-174. 枚举类型exti_trig_type_enum*
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 3-177. 函数 exti_interrupt_enable

函数名称	exti_interrupt_enable
函数原形	void exti_interrupt_enable(exti_line_enum linex);
功能描述	EXTI线x中断使能
先决条件	-
被调用函数	-
输入参数{in}	
linex	EXTI线x，参考*表3-172. 枚举类型exti_line_enum*
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 3-178. 函数 exti_interrupt_disable

函数名称	exti_interrupt_disable
函数原形	void exti_interrupt_disable(exti_line_enum linex);
功能描述	EXTI线x中断禁能
先决条件	-
被调用函数	-
输入参数{in}	
linex	EXTI线x，参考*表3-172. 枚举类型exti_line_enum*
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 3-179. 函数 exti_event_enable

函数名称	exti_event_enable
函数原形	void exti_event_enable(exti_line_enum linex);
功能描述	EXTI线x事件使能
先决条件	-
被调用函数	-
输入参数{in}	
linex	EXTI线x，参考*表3-172. 枚举类型exti_line_enum*
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 3-180. 函数 exti_event_disable

函数名称	exti_event_disable
函数原形	void exti_event_disable(exti_line_enum linex);
功能描述	EXTI线x事件禁能
先决条件	-
被调用函数	-
输入参数{in}	
linex	EXTI线x，参考*表3-172. 枚举类型exti_line_enum*
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 3-181. 函数 exti_software_interrupt_enable

函数名称	exti_software_interrupt_enable
函数原形	void exti_software_interrupt_enable(exti_line_enum linex);
功能描述	EXTI线x软件中断事件使能
先决条件	-
被调用函数	-
输入参数{in}	
linex	EXTI线x，参考*表3-172. 枚举类型exti_line_enum*
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 3-182. 函数 exti_software_interrupt_disable

函数名称	exti_software_interrupt_disable
函数原形	void exti_software_interrupt_disable(exti_line_enum linex);
功能描述	EXTI线x软件中断事件禁能
先决条件	-
被调用函数	-
输入参数{in}	
linex	EXTI线x，参考*表3-172. 枚举类型exti_line_enum*
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 3-183. 函数 exti_flag_get

函数名称	exti_flag_get
函数原形	FlagStatus exti_flag_get(exti_line_enum linex);
功能描述	获取EXTI线x中断标志位
先决条件	-
被调用函数	-
输入参数{in}	
linex	EXTI线x，参考*表3-172. 枚举类型exti_line_enum*
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 3-184. 函数 exti_flag_clear

函数名称	exti_flag_clear
函数原形	void exti_flag_clear(exti_line_enum linex);
功能描述	清除EXTI线x中断标志位
先决条件	-
被调用函数	-
输入参数{in}	
linex	EXTI线x，参考*表3-172. 枚举类型exti_line_enum*
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 3-185. 函数 exti_interrupt_flag_get

函数名称	exti_interrupt_flag_get
函数原形	FlagStatus exti_interrupt_flag_get(exti_line_enum linex);
功能描述	获取EXTI线x中断标志位
先决条件	-
被调用函数	-
输入参数{in}	
linex	EXTI线x，参考*表3-172. 枚举类型exti_line_enum*
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 3-186. 函数 exti_interrupt_flag_clear

函数名称	exti_interrupt_flag_clear
函数原形	void exti_interrupt_flag_clear(exti_line_enum linex);
功能描述	清除EXTI线x中断标志位
先决条件	-
被调用函数	-
输入参数{in}	
linex	EXTI线x，参考*表3-172. 枚举类型exti_line_enum*
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 3-187. FMC 寄存器

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

3.11.2. 外设库函数说明

FMC固件库函数列举如下表：

表 3-188. FMC 固件库函数

函数名称	函数描述
fmc_wscnt_set	设置FMC等待状态计数值
fmc_prefetch_enable	使能pre-fetch
fmc_prefetch_disable	失能pre-fetch
fmc_ibus_enable	使能IBUS缓存区
fmc_ibus_disable	失能IBUS缓存区
fmc_dbus_enable	使能DBUS缓存区
fmc_dbus_disable	失能DBUS缓存区

函数名称	函数描述
fmc_ibus_reset	复位IBUS缓存区
fmc_dbus_reset	复位DBUS缓存区
fmc_program_width_set	设置编程位宽
fmc_unlock	解锁FMC主编程块操作
fmc_lock	锁定FMC主编程块操作
fmc_page_erase	FMC 页擦除
fmc_mass_erase	FMC 全片擦除
fmc_doubleword_program	在相应地址双字编程
fmc_word_program	在相应地址全字编程
ob_unlock	解锁选项字节操作
ob_lock	锁定选项字节操作
ob_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_security_protection_flag_get	获取安全保护选项字节
fmc_interrupt_enable	使能FMC中断
fmc_interrupt_disable	除能FMC中断
fmc_flag_get	检查标志位是否置位
fmc_flag_clear	清除FMC标志
fmc_interrupt_flag_get	获取FMC中断标志状态
fmc_interrupt_flag_clear	清除FMC中断标志状态
fmc_state_get	获取FMC状态
fmc_ready_wait	检查FMC是否准备好

枚举类型 fmc_state_enum

表 3-189. 枚举类型 fmc_state_enum

枚举名称	枚举描述
FMC_READY	操作完成
FMC_BUSY	操作进行中
FMC_PGERR	编程错误
FMC_PGAERR	编程对齐错误
FMC_WPERR	写保护错误
FMC_TOERR	超时错误

函数 fmc_wscnt_set

函数fmc_wscnt_set描述见下表：

表 3-190. 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	rcu_periph_reset_enable / rcu_periph_reset_disable
输入参数{in}	
wscnt	等待状态计数值
FMC_WAIT_STATE_0	FMC 0个等待状态
FMC_WAIT_STATE_1	FMC 1个等待状态
FMC_WAIT_STATE_2	FMC 2个等待状态
FMC_WAIT_STATE_3	FMC 3个等待状态
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* set the wait state counter value */
fmc_wscnt_set (FMC_WAIT_STATE_1);
```

函数 fmc_prefetch_enable

函数fmc_prefetch_enable描述见下表：

表 3-191. 函数 fmc_prefetch_enable

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

返回值	
-	-

例如：

```
/* enable pre-fetch */
fmc_prefetch_enable( );
```

函数 fmc_prefetch_disable

函数fmc_prefetch_disable描述见下表：

表 3-192. 函数 fmc_prefetch_disable

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

例如：

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

函数 fmc_ibus_enable

函数fmc_ibus_enable描述见下表：

表 3-193. 函数 fmc_ibus_enable

函数名称	fmc_ibus_enable
函数原型	void fmc_ibus_enable(void);
功能描述	使能IBUS缓存区
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 IBUS cache */
fmc_ibus_enable( );
```

函数 fmc_ibus_disable

函数fmc_ibus_disable描述见下表：

表 3-194. 函数 fmc_ibus_disable

函数名称	fmc_ibus_disable
函数原型	void fmc_ibus_disable(void);
功能描述	失能IBUS缓存区
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 IBUS cache */
fmc_ibus_disable( );
```

函数 fmc_dbus_enable

函数fmc_dbus_enable描述见下表：

表 3-195. 函数 fmc_dbus_enable

函数名称	fmc_dbus_enable
函数原型	void fmc_dbus_enable(void);
功能描述	使能DBUS缓存区
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 DBUS cache */
fmc_dbus_enable( );
```

函数 fmc_dbus_disable

函数fmc_dbus_disable描述见下表：

表 3-196. 函数 fmc_dbus_disable

函数名称	fmc_dbus_disable
函数原型	void fmc_dbus_disable(void);
功能描述	失能DBUS缓存区
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 DBUS cache */
fmc_dbus_disable( );
```

函数 fmc_ibus_reset

函数fmc_ibus_reset描述见下表：

表 3-197. 函数 fmc_ibus_reset

函数名称	fmc_ibus_reset
函数原型	void fmc_ibus_reset (void);
功能描述	复位IBUS缓存区
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 IBUS cache */
fmc_ibus_reset( );
```

函数 fmc_dbus_reset

函数fmc_dbus_reset描述见下表：

表 3-198. 函数 fmc_dbus_reset

函数名称	fmc_dbus_reset
函数原型	void fmc_dbus_reset(void);
功能描述	复位DBUS缓存区
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 DBUS cache */
fmc_dbus_reset( );
```

函数 fmc_program_width_set

函数fmc_program_width_set描述见下表：

表 3-199. 函数 fmc_program_width_set

函数名称	fmc_program_width_set
函数原型	void fmc_program_width_set(uint32_t pgw);
功能描述	设置编程位宽
先决条件	fmc_unlock
被调用函数	-
输入参数{in}	
pgw	编程宽度
FMC_PROG_W_32B	32-bit编程宽度
FMC_PROG_W_64B	64-bit编程宽度

输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* set program width to flash memory */
fmc_program_width_set(FMC_PROG_W_32B);
```

函数 fmc_unlock

函数fmc_unlock描述见下表：

表 3-200. 函数 fmc_unlock

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

例如：

```
/* unlock the main FMC operation */
fmc_unlock( );
```

函数 fmc_lock

函数fmc_lock描述见下表：

表 3-201. 函数 fmc_lock

函数名称	fmc_lock
函数原型	void fmc_lock(void);
功能描述	锁定flash操作
先决条件	-
被调用函数	-
输入参数{in}	
-	-
输出参数{out}	

-	-
返回值	
-	-

例如：

```
/* lock the main FMC operation */
fmc_lock( );
```

函数 fmc_page_erase

函数fmc_page_erase描述见下表：

表 3-202. 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***

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 3-203. 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*

例如：

```
/* erase whole chip */
fmc_state_enum state = fmc_mass_erase ( );
```

函数 fmc_doubleword_program

函数fmc_doubleword_program描述见下表：

表 3-204. 函数 fmc_doubleword_program

函数名称	fmc_doubleword_program
函数原型	fmc_state_enum fmc_doubleword_program(uint32_t address, uint64_t data);
功能描述	对相应地址双字编程
先决条件	fmc_unlock
被调用函数	fmc_ready_wait
输入参数{in}	
address	编程地址
输入参数{in}	
data	编程数据
输出参数{out}	
-	-
返回值	
fmc_state_enum	*FMC状态*

例如：

```
/* program double word at the corresponding address */
fmc_state_enum state = fmc_word_program(0x08004000, 0xaabbccddeeffgghh);
```

函数 fmc_word_program

函数fmc_word_program描述见下表：

表 3-205. 函数 fmc_word_program

函数名称	fmc_word_program
函数原型	fmc_state_enum fmc_word_program(uint32_t address, uint32_t data);
功能描述	对相应地址全字编程
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*

例如：

```
/* program a word at the corresponding address */
fmc_state_enum state = fmc_word_program (0x08004000, 0xaabbccdd);
```

函数 ob_unlock

函数ob_unlock描述见下表：

表 3-206. 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 3-207. 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_erase

函数ob_erase描述见下表：

表 3-208. 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	
-	-

例如：

```
/* erase the FMC option byte */
ob_ erase ( );
```

函数 ob_write_protection_enable

函数ob_write_protection_enable描述见下表：

表 3-209. 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(uint32_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
OB_WP_x	特定写保护单元（x= 0 …31)

OB_WP_ALL	全片写保护
输出参数{out}	
-	-
返回值	
fmc_state_enum	***FMC状态***

例如：

/* enable write protection */

fmc_state_enum state = ob_write_protection_enable (OB_WP_7);

函数 ob_security_protection_config

函数ob_security_protection_config描述见下表：

表 3-210. 函数 ob_security_protection_config

函数名称	ob_security_protection_config
函数原型	fmc_state_enum ob_security_protection_config (uint8_t ob_spc);
功能描述	配置安全保护
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_USPC	安全保护
输出参数{out}	
-	-
返回值	
fmc_state_enum	***FMC状态***

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 3-211. 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_fwdgt, uint8_t ob_deepsleep, uint8_t ob_stdby);
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_fwdgt	选项字节看门狗数值
OB_FWDGT_SW	软件看门狗
OB_FWDGT_HW	硬件看门狗
输入参数{in}	
ob_deepsleep	选项字节深度睡眠复位值
OB_DEEPSLEEP_NRST	进入深度睡眠时不复位
OB_DEEPSLEEP_RST	进入深度睡眠时产生复位
输入参数{in}	
ob_stdby	选项字节待机复位值
OB_STDBY_NRST	进入待机时不复位
OB_STDBY_RST	进入待机时产生复位
输出参数{out}	
-	-
返回值	
fmc_state_enum	*FMC状态*

例如：

```
/* configure user option byte */
fmc_state_enum state = ob_user_write(OB_FWDGT_HW,OB_DEEPSLEEP_RST,
OB_STDBY_RST);
```

函数 ob_data_program

函数ob_data_program描述见下表：

表 3-212. 函数 ob_data_program

函数名称	ob_data_program
函数原型	fmc_state_enum ob_data_program(uint32_t address, uint8_t data);
功能描述	编程数字选项字节
先决条件	ob_unlock
被调用函数	fmc_ready_wait
输入参数{in}	
address	编程数字选项字节地址
输入参数{in}	
data	所编程数值
输出参数{out}	
-	-
返回值	

fmc_state_enum	FMC状态

例如：

```
/* program option bytes data */
fmc_state_enum state = ob_data_program (0x1ffff804, 0x56);
```

函数 ob_user_get

函数ob_user_get描述见下表：

表 3-213. 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(0xF0 - 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 3-214. 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	
uint8_t	选项字节数据值(0x0 - 0xFF)

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 3-215. 函数 ob_write_protection_get

函数名称	ob_ write_protection_get
函数原型	uint32_t ob_write_protection_get(void);
功能描述	获取选项字节写保护数值
先决条件	-
被调用函数	-
输入参数{in}	
-	-
输出参数{out}	
-	-
返回值	
uint32_t	选项字节写保护数值(0x0 - 0XFFFFFFFF)

例如：

```
/* get the FMC option byte write protection */
uint32_t wp = ob_write_protection_get ( );
```

函数 ob_security_protection_flag_get

函数ob_security_protection_flag_get描述见下表：

表 3-216. 函数 ob_security_protection_flag_get

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

例如：

/* get the FMC option byte security protection */

FlagStatus spc = ob_security_protection_flag_get ( );

函数 fmc_interrupt_enable

函数fmc_interrupt_enable描述见下表：

表 3-217. 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 3-218. 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 3-219. 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_PGERR	FMC操作错误标志
FMC_FLAG_PGAERR	FMC编程对齐错误标志
FMC_FLAG_WPERR	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 3-220. 函数 fmc_flag_clear

函数名称	fmc_flag_clear
函数原型	void fmc_flag_clear(uint32_t flag);
功能描述	清除FMC标志

先决条件	-
被调用函数	-
输入参数{in}	
flag	清除FMC标志
FMC_FLAG_PGERR	FMC操作错误标志
FMC_FLAG_PGAERR	FMC编程对齐错误标志
FMC_FLAG_WPERR	FMC写保护错误标志
FMC_FLAG_END	FMC操作完成标志
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* clear FMC flag */
FlagStatus flag = fmc_flag_clear(FMC_FLAG_END);
```

函数 fmc_interrupt_flag_get

函数fmc_interrupt_flag_get描述见下表：

表 3-221. 函数 fmc_interrupt_flag_get

函数名称	fmc_interrupt_flag_get
函数原型	FlagStatus fmc_interrupt_flag_get(fmc_interrupt_flag_enum flag);
功能描述	获取FMC中断标志状态
先决条件	-
被调用函数	-
输入参数{in}	
flag	中断标志
FMC_INT_FLAG_PGERR	FMC操作错误标志
FMC_INT_FLAG_PGAERR	FMC编程对齐错误标志
FMC_INT_FLAG_WPERR	FMC写保护错误标志
FMC_INT_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 interrupt flag */

FlagStatus flag = fmc_interrupt_flag_get (FMC_INT_FLAG_PGERR);

函数 fmc_interrupt_flag_clear

函数fmc_interrupt_flag_clear描述见下表：

表 3-222. 函数 fmc_interrupt_flag_clear

函数名称	fmc_interrupt_flag_ clear
函数原型	FlagStatus fmc_interrupt_flag_ clear (fmc_interrupt_flag_enum flag);
功能描述	清除FMC中断标志
先决条件	-
被调用函数	-
输入参数{in}	
flag	清除FMC中断标志
FMC_INT_FLAG_PGERR	FMC操作错误标志
FMC_INT_FLAG_PGAERR	FMC编程对齐错误标志
FMC_INT_FLAG_WPERR	FMC写保护错误标志
FMC_INT_FLAG_END	FMC操作完成标志
输出参数{out}	
-	-
返回值	
-	-

例如：

/* clear FMC interrupt flag */

FlagStatus flag = fmc_interrupt_flag_clear (FMC_INT_FLAG_BANK0_PGERR);

函数 fmc_state_get

函数fmc_state_get描述见下表：

表 3-223. 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*

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 3-224. 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	count of loop
输出参数{out}	
-	-
返回值	
fmc_state_enum	*FMC状态*

例如：

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

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 3-225. FWDGT 寄存器

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

3.12.2. 外设库函数说明

FWDGT库函数列表如下表所示：

表 3-226. FWDGT 库函数

库函数名称	库函数描述
fwdgt_write_enable	使能对寄存器FWDGT_PSC和FWDGT_RLD的写操作
fwdgt_write_disable	失能对寄存器FWDGT_PSC和FWDGT_RLD的写操作
fwdgt_enable	使能FWDGT
fwdgt_prescaler_value_config	配置独立看门狗定时器预分频值
fwdgt_reload_value_config	配置独立看门狗定时器重装载值
fwdgt_counter_reload	按照FWDGT_RLD寄存器的值重装载IWDG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 3-227. 函数 fwdgt_write_enable

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

例如：

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

函数 fwdgt_write_disable

函数fwdgt_write_disable描述见下表：

表 3-228. 函数 fwdgt_write_disable

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

例如：

```
/* disable write access to FWDGT_PSC and FWDGT_RLD */
fwdgt_write_disable ( );
```

函数 fwdgt_enable

函数fwdgt_enable描述见下表：

表 3-229. 函数 fwdgt_enable

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

例如：

```
/* start the free watchdog timer counter */

fwdgt_enable ( );
```

函数 fwdgt_prescaler_value_config

函数fwdgt_prescaler_value_config描述见下表：

表 3-230. 函数 fwdgt_prescaler_value_config

函数名称	fwdgt_prescaler_value_config
函数原型	ErrStatus fwdgt_prescaler_value_config(uint16_t prescaler_value);
功能描述	配置独立看门狗定时器计数器窗口值
先决条件	-
输入参数{in}	
prescaler_value	预分频值
FWDGT_PSC_DIV4	FWDGT预分频值设为4
FWDGT_PSC_DIV8	FWDGT预分频值设为8
FWDGT_PSC_DIV16	FWDGT预分频值设为16
FWDGT_PSC_DIV32	FWDGT预分频值设为32
FWDGT_PSC_DIV64	FWDGT预分频值设为64
FWDGT_PSC_DIV128	FWDGT预分频值设为128
FWDGT_PSC_DIV256	FWDGT预分频值设为256
输出参数{out}	
-	-
返回值	
ErrStatus	ERROR / SUCCESS

例如：

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

ErrStatus flag;

flag = fwdgt_prescaler_value_config (FWDGT_PSC_DIV256);
```

函数 fwdgt_reload_value_config

函数fwdgt_reload_value_config描述见下表：

表 3-231. 函数 fwdgt_reload_value_config

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

例如：

```
/* set FWDGT reload value to 0x0FFF */
ErrStatus flag;
flag = fwdgt_reload_value_config (0x0FFF);
```

函数 fwdgt_counter_reload

函数fwdgt_counter_reload描述见下表：

表 3-232. 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	-
被调用函数	-
输入参数{in}	
-	-
输出参数{out}	
-	-
返回值	
-	-

例如：

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

函数 fwdgt_config

函数fwdgt_config描述见下表：

表 3-233. 函数 fwdgt_config

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

例如：

```
/* confiure FWDGT counter clock: 40KHz(IRC40K) / 64 = 0.625 KHz */
fwdgt_config(2*500, FWDGT_PSC_DIV64);
```

函数 fwdgt_flag_get

函数fwdgt_flag_get描述见下表：

表 3-234. 函数 fwdgt_flag_get

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

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

例如：

```
/* test if a prescaler value update is on going */
FlagStatus status;
status = fwdgt_flag_get (FWDGT_FLAG_PUD);
if(status == RESET)
{
...
}else
{
...
}
```

3.13. GPIO

GPIO用来实现各片上设备的逻辑输入/输出功能。章节*3.14.1*描述了GPIO的寄存器列表，章节*3.14.2*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 3-235. GPIO 寄存器

寄存器名称	寄存器描述
GPIOx_CTL0	端口控制寄存器0
GPIOx_CTL1	端口控制寄存器1

寄存器名称	寄存器描述
GPIOx_ISTAT	端口输入状态寄存器
GPIOx_OCTL	端口输出控制寄存器
GPIOx_BOP	端口位操作寄存器
GPIOx_BC	位清除寄存器
GPIOx_LOCK	端口配置锁定寄存器
GPIOx_SPD	端口位速度寄存器
AFIO_EC	事件控制寄存器
AFIO_PCF0	AFIO端口配置寄存器0
AFIO_EXTISS0	EXTI源选择寄存器0寄存器
AFIO_EXTISS1	EXTI源选择寄存器1寄存器
AFIO_EXTISS2	EXTI源选择寄存器2寄存器
AFIO_EXTISS3	EXTI源选择寄存器3寄存器
AFIO_PCF1	AFIO端口配置寄存器1
AFIO_CPSCTL	IO补偿控制寄存器

3.13.2. 外设库函数说明

GPIO库函数列表如下表所示：

表 3-236. GPIO 库函数

库函数名称	库函数描述
gpio_deinit	复位外设GPIOx
gpio_afio_deinit	复位AFIO
gpio_init	GPIO参数初始化
gpio_bit_set	置位引脚值
gpio_bit_reset	复位引脚值
gpio_bit_write	将特定的值写入引脚
gpio_port_write	将特定的值写入一组端口
gpio_input_bit_get	获取引脚的输入值
gpio_input_port_get	获取一组端口的输入值
gpio_output_bit_get	获取引脚的输出值
gpio_output_port_get	获取一组端口的输出值
gpio_pin_remap_config	配置GPIO引脚重映射
gpio_exti_source_select	选择哪个引脚作为EXTI源
gpio_event_output_config	配置事件输出
gpio_event_output_enable	事件输出使能
gpio_event_output_disable	事件输出除能
gpio_pin_lock	相应的引脚配置被锁定
gpio_compensation_config	配置I/O补偿单元
gpio_compensation_flag_get	检测I/O补偿单元是否准备好

函数 gpio_deinit

函数gpio_deinit描述见下表：

表 3-237. 函数 gpio_deinit

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

例如：

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

函数 gpio_afio_deinit

函数gpio_afio_deinit描述见下表：

表 3-238. 函数 gpio_afio_deinit

函数名称	gpio_afio_deinit
函数原型	void gpio_afio_deinit(void);
功能描述	复位AFIO
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 alternate function */
gpio_afio_deinit();
```

函数 gpio_init

函数gpio_init描述见下表：

表 3-239. 函数 gpio_init

函数名称	gpio_init
函数原型	void gpio_init(uint32_t gpio_periph, uint32_t mode, uint32_t speed, uint32_t pin);;
功能描述	GPIO参数初始化
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,E)
输入参数{in}	
gpio_mode	GPIO引脚模式
GPIO_MODE_AIN	模拟输入模式
GPIO_MODE_IN_FLOATING	浮空输入模式
GPIO_MODE_IPD	下拉输入模式
GPIO_MODE_IPU	上拉输入模式
GPIO_MODE_OUT_OD	开漏输出模式
GPIO_MODE_OUT_PP	推挽输出模式
GPIO_MODE_AF_OD	AFIO开漏输出模式
GPIO_MODE_AF_PP	AFIO推挽输出模式
输出参数{out}	
speed	GPIO输出最大速度
GPIO_OSPEED_10MHZ	10MHZ
GPIO_OSPEED_2MHZ	2MHZ
GPIO_OSPEED_50MHZ	50MHZ
GPIO_OSPEED_MAX	>50MHZ
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：

```
/* config PA0 as analog input mode */
gpio_init (GPIOA, GPIO_MODE_AIN, GPIO_OSPEED_50MHZ, GPIO_PIN_0);
```

函数 gpio_bit_set

函数gpio_bit_set描述见下表：

表 3-240. 函数 gpio_bit_set

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

例如：

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

函数 gpio_bit_reset

函数gpio_bit_reset描述见下表：

表 3-241. 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,E)
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 3-242. 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,E)
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 3-243. 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,E)
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 to Port A */

gpio_port_write (GPIOA, 0xA5);

函数 gpio_input_bit_get

函数gpio_input_bit_get描述见下表：

表 3-244. 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,E)
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 3-245. 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,E)
输出参数{out}	
-	-
返回值	
uint16_t	0x00 - 0xFF

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 3-246. 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,E)
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 3-247. 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,E)
输出参数{out}	
-	-

返回值	
uint16_t	0x00 - 0xFF

例如：

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

函数 gpio_pin_remap_config

函数gpio_pin_remap_config描述见下表：

表 3-248. 函数 gpio_pin_remap_config

函数名称	gpio_pin_remap_config
函数原型	void gpio_pin_remap_config(uint32_t gpio_remap, ControlStatus newvalue);
功能描述	配置GPIO引脚重映射
先决条件	-
被调用函数	-
输入参数{in}	
gpio_remap	选择重映射
GPIO_SPI0_REMAP	SPI0重映射
GPIO_I2C0_REMAP	I2C0重映射
GPIO_USART0_REMAP	USART0重映射
GPIO_USART1_REMAP	USART1重映射
GPIO_USART2_PARTIAL_REMAP	USART2部分重映射
GPIO_USART2_FULL_REMAP	USART2全部重映射
GPIO_TIMER0_PARTIAL_REMAP	TIMER0部分重映射
GPIO_TIMER0_FULL_REMAP	TIMER0全部重映射
GPIO_TIMER1_PARTIAL_REMAP1	TIMER1部分重映射
GPIO_TIMER1_PARTIAL_REMAP2	TIMER1部分重映射
GPIO_TIMER1_FULL_REMAP	TIMER1全部重映射
GPIO_TIMER2_PARTIAL_REMAP	TIMER2部分重映射

GPIO_TIMER2_FULL_REMAP	TIMER2全部重映射
GPIO_TIMER3_REMAP	TIMER3重映射
GPIO_PD01_REMAP	PD01重映射
GPIO_TIMER4CH3_IREMAP	TIMER4 channel3内部重映射
GPIO_ADC0_ETRGRT_REMAP	ADC0常规转换外部触发重映射
GPIO_ADC1_ETRGRT_REMAP	ADC1常规转换外部触发重映射
GPIO_SWJ_NONJTRST_REMAP	全部的SWJ(JTAG-DP + SW-DP)，但是不包括NJTRST
GPIO_SWJ_SWDPENABLE_REMAP	JTAG-DP除能，SW-DP使能
GPIO_SWJ_DISABLE_REMAP	JTAG-DP除能，SW-DP除能
GPIO_SPI2_REMAP	SPI2重映射
GPIO_TIMER1ITR0_REMAP	TIMER1内部触发0重映射
GPIO_TIMER8_REMAP	TIMER8重映射
GPIO_EXMC_NADV_REMAP	EXMC_NADV 连接/断开
GPIO_CTC_REMAP0	CTC重映射(PD15)
输入参数{in}	
newvalue	是否使能
ENABLE	使能
DISABLE	除能
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* enable SPI0 remapping */
gpio_pin_remap_config (GPIO_SPI0_REMAP, ENABLE);
```

函数 gpio_exti_source_select

函数gpio_exti_source_select描述见下表：

表 3-249. 函数 gpio_exti_source_select

函数名称	gpio_exti_source_select
函数原型	void gpio_exti_source_select(uint8_t output_port, uint8_t output_pin);
功能描述	选择哪个引脚作为EXTI源
先决条件	-
被调用函数	-
输入参数{in}	
gpio_outputport	EXTI源端口
GPIO_PORT_SOURCE_GPIOx	源端口选择(x = A,B,C,D,E)
输入参数{in}	
gpio_outputpin	源端口引脚
GPIO_PIN_SOURCE_x	引脚选择 (x = 0..15)
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* config PA0 as EXTI source */
gpio_exti_source_select (GPIO_PORT_SOURCE_GPIOA, GPIO_PIN_SOURCE_0);
```

函数 gpio_event_output_config

函数gpio_event_output_config描述见下表:

表 3-250. 函数 gpio_event_output_config

函数名称	gpio_event_output_config
函数原型	void gpio_event_output_config(uint8_t output_port, uint8_t output_pin);
功能描述	配置事件输出
先决条件	-
被调用函数	-
输入参数{in}	
gpio_outputport	GPIO事件输出端口
GPIO_EVENT_PORT_GPIOx	事件输出端口选择(x = A,B,C,D,E)
输入参数{in}	
gpio_outputpin	GPIO事件输出引脚
GPIO_EVENT_PIN_x	引脚选择 (x = 0..15)
输出参数{out}	

-	-
返回值	
-	-

例如：

/* Config PA0 as the output of event */

gpio_event_output_config (GPIO_EVENT_PORT_GPIOA, GPIO_EVENT_PIN_0);

函数 gpio_event_output_enable

函数gpio_event_output_enable描述见下表：

表 3-251. 函数 gpio_event_output_enable

函数名称	gpio_event_output_enable
函数原型	void gpio_event_output_enable(void);
功能描述	事件输出使能
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 GPIO pin event output */

gpio_event_output_enable(void);

函数 gpio_event_output_disable

函数gpio_event_output_disable描述见下表：

表 3-252. 函数 gpio_event_output_disable

函数名称	gpio_event_output_disable
函数原型	void gpio_event_output_ disable(void);
功能描述	事件输出除能
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 GPIO pin event output */

gpio_event_output_disable(void);

函数 gpio_pin_lock

函数gpio_pin_lock描述见下表：

表 3-253. 函数 gpio_pin_lock

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

例如：

/* lock PA0 */

gpio_pin_lock (GPIOA, GPIO_PIN_0);

函数 gpio_compensation_config

函数gpio_compensation_config描述见下表：

表 3-254. 函数 gpio_compensation_config

函数名称	gpio_compensation_config
函数原型	void gpio_compensation_config(uint32_t compensation);
功能描述	事件输出除能

先决条件	配置I/O补偿单元
被调用函数	-
输入参数{in}	
compensation	指定I/O补偿单元模式
GPIO_COMPENSATION_ENABLE	I/O补偿单元使能
GPIO_COMPENSATION_DISABLE	I/O补偿单元失能
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* enabled I/O compensation cell */
gpio_compensation_config (GPIO_COMPENSATION_ENABLE);
```

函数 gpio_compensation_flag_get

函数gpio_compensation_flag_get描述见下表：

表 3-255. 函数 gpio_compensation_flag_get

函数名称	gpio_compensation_flag_get
函数原型	FlagStatus gpio_compensation_flag_get(void);
功能描述	检测I/O补偿单元是否准备好
先决条件	-
被调用函数	-
输入参数{in}	
-	-
输出参数{out}	
-	-
返回值	
FlagStatus	SET or RESET

例如：

```
/* check the I/O compensation cell state */
FlagStatus cell_state;
cell_state = gpio_compensation_flag_get (void);
```

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 3-256. I2C 寄存器

寄存器名称	寄存器描述
I2C_CTL0	控制寄存器0
I2C_CTL1	控制寄存器1
I2C_SADDR0	从机地址寄存器0
I2C_SADDR1	从机地址寄存器1
I2C_DATA	传输缓冲区寄存器
I2C_STAT0	传输状态寄存器0
I2C_STAT1	传输状态寄存器1
I2C_CKCFG	时钟配置寄存器
I2C_RT	上升时间寄存器
I2C_SAMCS	SAM控制状态寄存器
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 3-257. 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POAP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

库函数名称	库函数描述
i2c_data_transmit	发送数据
i2c_data_receive	接收数据
i2c_dma_config	I2C DMA模式配置
i2c_dma_last_transfer_config	配置下一个DMA EOT是否最后一次传输
i2c_stretch_scl_low_config	当从机数据没有准备好时是否拉低SCL
i2c_slave_response_to_gcall_config	从机是否响应广播呼叫
i2c_software_reset_config	配置I2C软件复位
i2c_pec_config	报文错误校验配置
i2c_pec_transfer_config	传输PEC值配置
i2c_pec_value_get	获取报文错误校验值
i2c_smbus_alert_config	通过SMBA引脚发送警告
i2c_smbus_arp_config	SMBus下ARP协议是否开启
i2c_sam_enable	使能SAM_V接口
i2c_sam_disable	禁止SAM_V接口
i2c_sam_timeout_enable	使能SAM_V接口超时检测
i2c_sam_timeout_disable	禁止SAM_V接口超时检测
i2c_flag_get	获取I2C标志位
i2c_flag_clear	清除I2C标志位
i2c_interrupt_enable	中断使能
i2c_interrupt_disable	中断除能
i2c_interrupt_flag_get	中断标志位获取
i2c_interrupt_flag_clear	中断标志位清除

枚举 i2c_flag_enum

表 3-258. 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

成员名称	功能描述
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	是否接收广播寻址
I2C_FLAG_DEFSMB	从机模式下SMBus主机地址头
I2C_FLAG_HSTSMB	从机模式下监测到SMBus主机地址头
I2C_FLAG_DUMOD	从机模式下双标志位表明哪个地址和双地址模式匹配
I2C_FLAG_TFF	SAM_V模式下发送帧下降沿标志
I2C_FLAG_TFR	SAM_V模式下发送帧上升沿标志
I2C_FLAG_RFF	SAM_V模式下接收帧下降沿标志
I2C_FLAG_RFR	SAM_V模式下接收帧上升沿标志

枚举 i2c_interrupt_flag_enum

表 3-259. 枚举类型 i2c_interrupt_flag_enum

成员名称	功能描述
I2C_INT_FLAG_SBSEND	主机模式下发送START起始位中断标志
I2C_INT_FLAG_ADDSEND	主机模式下成功发送了地址，从机模式下接收到了地址并且和自身的地址匹配中断标志
I2C_INT_FLAG_BTC	字节发送结束中断标志
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t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
I2C_INT_FLAG_TFF	SAM_V模式下发送帧下降沿中断标志
I2C_INT_FLAG_TFR	SAM_V模式下发送帧上升沿中断标志
I2C_INT_FLAG_RFF	SAM_V模式下接收帧下降沿中断标志
I2C_INT_FLAG_RFR	SAM_V模式下接收帧上升沿中断标志

枚举 i2c_interrupt_enum

表 3-260. 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
I2C_INT_TFF	SAM_V模式下发送帧下降沿中断使能
I2C_INT_TFR	SAM_V模式下发送帧上升沿中断使能
I2C_INT_RFF	SAM_V模式下接收帧下降沿中断使能
I2C_INT_RFR	SAM_V模式下接收帧上升沿中断使能

函数 i2c_deinit

函数i2c_deinit描述见下表：

表 3-261. 函数 i2c_deinit

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

例如：

```
/* reset I2C0 */

i2c_deinit(I2C0);
```

函数 i2c_clock_config

函数i2c_clock_config描述见下表：

表 3-262. 函数 i2c_clock_config

函数名称	i2c_clock_config
函数原型	void i2c_clock_config(uint32_t i2c_periph, uint32_t clkspeed, uint32_t dutycyc);
功能描述	配置I2C时钟

先决条件	-
被调用函数	rcu_clock_freq_get
输入参数{in}	
i2c_periph	I2C外设
I2Cx	(x=0,1)
输入参数{in}	
clkspeed	i2c时钟速率
输入参数{in}	
dutycyc	快速模式下占空比
I2C_DTCY_2	T_low/T_high=2
I2C_DTCY_16_9	T_low/T_high=16/9
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* configure I2C0 clock speed as 100KHz */
i2c_clock_config(I2C0, 100000, I2C_DTCY_2);
```

函数 i2c_mode_addr_config

函数i2c_mode_addr_config描述见下表：

表 3-263. 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 3-264. 函数 i2c_smbus_type_config

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

例如：

```
/* config I2C0 as SMBUS host type */
i2c_smbus_type_config (I2C0, I2C_SMBUS_HOST);
```

函数 i2c_ack_config

函数i2c_ack_config描述见下表：

表 3-265. 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 3-266. 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	当前正在接收的字节是否发送ACK

RENT	
I2C_ACKPOS_NEXT	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 3-267. 函数 i2c_master_addressing

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

例如：

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

i2c_master_addressing(I2C0, 0x82, I2C_RECEIVER);

函数 i2c_dualaddr_enable

函数i2c_dualaddr_enable描述见下表：

表 3-268. 函数 i2c_dualaddr_enable

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

例如：

```
/* enable I2C0 dual-address */
i2c_dualaddr_enable (I2C0, 0x80);
```

函数 i2c_dualaddr_disable

函数i2c_dualaddr_disable描述见下表：

表 3-269. 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 3-270. 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 3-271. 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 3-272. 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 3-273. 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 3-274. 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 3-275. 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_config

函数i2c_dma_config描述见下表：

表 3-276. 函数 i2c_dma_config

函数名称	i2c_dma_config
函数原型	void i2c_dma_config(uint32_t i2c_periph, uint32_t dmastate);
功能描述	I2C DMA模式配置
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}	
dmastate	开启或关闭
I2C_DMA_ON	DMA模式开启
I2C_DMA_OFF	DMA模式关闭
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* I2C0 DMA mode config */
i2c_dma_conifg (I2C0, I2C_DMA_ON);
```

函数 i2c_dma_last_transfer_congig

函数i2c_dma_last_transfer_config描述见下表：

表 3-277. 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 3-278. 函数 i2c_stretch_scl_low_config

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

-	-
返回值	
-	-

例如：

```
/* stretch SCL low when data is not ready in slave mode */
i2c_stretch_scl_low_config (I2C0, I2C_SCLSTRETCH_ENABLE);
```

函数 i2c_slave_response_to_gcall_config

函数i2c_slave_response_to_gcall_config描述见下表：

表 3-279. 函数 i2c_slave_response_to_gcall_config

函数名称	i2c_slave_response_to_gcall_config
函数原型	void i2c_slave_response_to_gcall_config(uint32_t i2c_periph, uint32_t gcallpara);
功能描述	从机是否响应广播呼叫
先决条件	-
被调用函数	-
输入参数{in}	
i2c_periph	I2C外设
I2Cx	(x=0,1)
输入参数{in}	
gcallpara	是否响应广播呼叫
I2C_GCEN_ENABLE	从机响应广播呼叫
I2C_GCEN_DISABLE	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 3-280. 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_RESET	没有复位
输出参数{out}	
-	-
返回值	
-	-

例如：

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

函数 i2c_pec_config

函数i2c_pec_config描述见下表：

表 3-281. 函数 i2c_pec_config

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

返回值	
-	-

例如：

/* config I2C PEC calculation */

i2c_pec_config (I2C0, I2C_PEC_ENABLE);

函数 i2c_pec_transfer_config

函数i2c_pec_transfer_config描述见下表：

表 3-282. 函数 i2c_pec_transfer_config

函数名称	i2c_pec_transfer_config
函数原型	void i2c_pec_transfer_conifg(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_config (I2C0, I2C_PECTRANS_ENABLE);

函数 i2c_pec_value_get

函数i2c_pec_value_get描述见下表：

表 3-283. 函数 i2c_pec_value_get

函数名称	i2c_pec_value_get
函数原型	uint8_t i2c_pec_value_get(uint32_t i2c_periph);

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

例如：

```
/* I2C0 get packet error checking value */
uint8_t pec_value;
pec_value = i2c_pec_value_get (I2C0);
```

函数 i2c_smbus_alert_config

函数i2c_smbus_alert_config描述见下表：

表 3-284. 函数 i2c_smbus_alert_config

函数名称	i2c_smbus_alert_config
函数原型	void i2c_smbus_alert_config (uint32_t i2c_periph, uint32_t smbuspara);
功能描述	通过SMBA引脚发送警告
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_ENABLE	通过SMBA引脚发送警告
I2C_SALTSEND_DISABLE	不通过SMBA引脚发送警告
输出参数{out}	
-	-
返回值	
-	-

例如：

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

i2c_smbus_alert_config (I2C0, I2C_SALTSEND_ENABLE);

函数 i2c_smbus_arp_config

函数i2c_smbus_arp_config描述见下表：

表 3-285. 函数 i2c_smbus_arp_config

函数名称	i2c_smbus_arp_config
函数原型	void i2c_smbus_arp_config(uint32_t i2c_periph, uint32_t arpstate);
功能描述	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：

/* config I2C0 ARP protocol in SMBus switch */

i2c_smbus_arp_config (I2C0, I2C_ARP_ENABLE);

函数 i2c_sam_enable

函数i2c_sam_enable描述见下表：

表 3-286. 函数 i2c_sam_enable

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

-	-
返回值	
-	-

例如：

/* enable I2C0 SAM_V interface */

i2c_sam_enable (I2C0);

函数 i2c_sam_disable

函数i2c_sam_disable描述见下表：

表 3-287. 函数 i2c_sam_disable

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

例如：

/* disable I2C0 SAM_V interface */

i2c_sam_disable (I2C0);

函数 i2c_sam_timeout_enable

函数i2c_sam_timeout_enable描述见下表：

表 3-288. 函数 i2c_sam_timeout_enable

函数名称	i2c_sam_timeout_enable
函数原型	void i2c_sam_timeout_enable(uint32_t i2c_periph);
功能描述	使能SAM_V接口超时检测
先决条件	-
被调用函数	-
输入参数{in}	
i2c_periph	I2C外设

I2Cx	(x=0,1)
输出参数{out}	
-	-
返回值	
-	-

例如：

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

函数 i2c_sam_timeout_disable

函数i2c_sam_timeout_disable描述见下表：

表 3-289. 函数 i2c_sam_timeout_disable

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

例如：

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

函数 i2c_flag_get

函数i2c_flag_get描述见下表：

表 3-290. 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_SBSEND	起始位是否发送
I2C_FLAG_ADDSEND	主机模式下地址是否发送/从机模式下地址是否匹配
I2C_FLAG_BTC	字节传输完成
I2C_FLAG_ADD10SEND	主机模式下10位地址地址头发送完成
I2C_FLAG_STPDET	从机模式下监测到STOP结束位
I2C_FLAG_RBNE	接收期间I2C_DATA非空
I2C_FLAG_TBE	发送期间I2C_DATA为空
I2C_FLAG_BERR	总线错误，表示I2C总线上发生了预料之外的START起始位或STOP结束位
I2C_FLAG_LOSTARB	主机模式下仲裁丢失
I2C_FLAG_AERR	应答错误
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_PECERR	接收数据时发生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	是否接收到广播地址(00h)
I2C_FLAG_DEFSMB	从机模式下SMBus主机地址头
I2C_FLAG_HSTSMB	从机模式下监测到SMBus主机地址头
I2C_FLAG_DUMOD	从机模式下双标志位表明哪个地址和双地址模式匹配
I2C_FLAG_TFF	SAM_V模式下发送帧下降沿标志
I2C_FLAG_TFR	SAM_V模式下发送帧上升沿标志
I2C_FLAG_RFF	SAM_V模式下接收帧下降沿标志
I2C_FLAG_RFR	SAM_V模式下接收帧上升沿标志
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 3-291. 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_SMBALT	SMBus警报状态
I2C_FLAG_SMBTO	SMBus模式下超时信号
I2C_FLAG_PECERR	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_LOSTARB	主机模式下仲裁丢失
I2C_FLAG_BERR	总线错误
I2C_FLAG_ADDSEND	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
I2C_FLAG_TFF	SAM_V模式下发送帧下降沿标志
I2C_FLAG_TFR	SAM_V模式下发送帧上升沿标志
I2C_FLAG_RFF	SAM_V模式下接收帧下降沿标志
I2C_FLAG_RFR	SAM_V模式下接收帧上升沿标志
输出参数{out}	
-	-
返回值	

-	-

例如：

/* clear a bus error flag*/

i2c_flag_clear (I2C0, I2C_FLAG_BERR);

函数 i2c_interrupt_enable

函数i2c_interrupt_enable描述见下表：

表 3-292. 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}	
inttype	中断类型
I2C_INT_ERR	错误中断使能
I2C_INT_EV	事件中断使能
I2C_INT_BUF	缓冲区中断使能
I2C_INT_TFF	SAM_V模式下发送帧下降沿中断使能
I2C_INT_TFR	SAM_V模式下发送帧上升沿中断使能
I2C_INT_RFF	SAM_V模式下接收帧下降沿中断使能
I2C_INT_RFR	SAM_V模式下接收帧上升沿中断使能
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 3-293. 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}	
inttype	中断类型
I2C_INT_ERR	错误中断禁能
I2C_INT_EV	事件中断禁能
I2C_INT_BUF	缓冲区中断禁能
I2C_INT_TFF	SAM_V模式下发送帧下降沿中断禁能
I2C_INT_TFR	SAM_V模式下发送帧上升沿中断禁能
I2C_INT_RFF	SAM_V模式下接收帧下降沿中断禁能
I2C_INT_RFR	SAM_V模式下接收帧上升沿中断禁能
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 3-294. 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_SBSEND	主机模式下发送START起始位
I2C_INT_FLAG_ADDSEND	主机模式下成功发送了地址 / 从机模式下接收到了地址并且和自身的地址匹配
I2C_INT_FLAG_BTC	字节发送结束中断使能
I2C_INT_FLAG_ADD10SEND	主机模式下10位地址地址头被发送
I2C_INT_FLAG_STPDET	从机模式下监测到STOP结束位
I2C_INT_FLAG_RBNE	接收期间I2C_DATA非空
I2C_INT_FLAG_TBE	发送期间I2C_DATA为空
I2C_INT_FLAG_BERR	总线错误
I2C_INT_FLAG_LOSTARB	主机模式下仲裁丢失
I2C_INT_FLAG_AERR	应答错误
I2C_INT_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_INT_FLAG_PECERR	接收数据时PEC错误
I2C_INT_FLAG_SMBTO	SMBus模式下超时信号
I2C_INT_FLAG_SMBALT	SMBus警报状态
I2C_INT_FLAG_TFF	SAM_V模式下发送帧下降沿标志
I2C_INT_FLAG_TFR	SAM_V模式下发送帧上升沿标志
I2C_INT_FLAG_RFF	SAM_V模式下接收帧下降沿标志
I2C_INT_FLAG_RFR	SAM_V模式下接收帧上升沿标志
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 3-295. 函数 i2c_interrupt_flag_clear

函数名称	i2c_interrupt_flag_clear
函数原型	void i2c_interrupt_flag_clear(uint32_t i2c_periph, i2c_interrupt_flag_enum int_flag);
功能描述	中断标志位清除
先决条件	-
被调用函数	-
输入参数{in}	
i2c_periph	I2C外设
I2Cx	(x=0,1)
输入参数{in}	
int_flag	中断标志
I2C_INT_FLAG_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_INT_FLAG_BERR	总线错误
I2C_INT_FLAG_LOSTARB	主机模式下仲裁丢失
I2C_INT_FLAG_AERR	应答错误
I2C_INT_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_INT_FLAG_PECERR	接收数据时PEC错误
I2C_INT_FLAG_SMBTO	SMBus模式下超时信号
I2C_INT_FLAG_SMBALT	SMBus警报状态
I2C_INT_FLAG_TFF	SAM_V模式下发送帧下降沿标志
I2C_INT_FLAG_TFR	SAM_V模式下发送帧上升沿标志
I2C_INT_FLAG_RFF	SAM_V模式下接收帧下降沿标志

I2C_INT_FLAG_RFR	SAM_V模式下接收帧上升沿标志
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*错误!未找到引用源。*描述了 NVIC 和 SysTick 的寄存器列表，章节*错误!未找到引用源。*对 MISC 库函数进行说明。

3.15.1. 外设寄存器说明

表 3-296. 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
BFAR(2)	总线错误地址寄存器

寄存器名称	寄存器描述
AFSR(2)	辅助错误状态寄存器
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 3-297. 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

枚举类型 IRQn_Type

表 3-298. 枚举类型 IRQn_Type

成员名称	功能描述
WWDGT_IRQn	窗口看门狗定时器中断
LVD_IRQn	连接到 EXTI 线的 LVD 中断
TAMPER_IRQn	侵入检测中断
RTC_IRQn	RTC 全局中断
FMC_IRQn	FMC 全局中断
RCU_CTC_IRQn	RCU 和 CTC 中断
EXTI0_IRQn	EXTI 线 0 中断
EXTI1_IRQn	EXTI 线 1 中断
EXTI2_IRQn	EXTI 线 2 中断
EXTI3_IRQn	EXTI 线 3 中断
EXTI4_IRQn	EXTI 线 4 中断
DMA0_Channel0_IRQn	DMA0 通道 0 全局中断

成员名称	功能描述
DMA0_Channel1_IRQn	DMA0 通道 1 全局中断
DMA0_Channel2_IRQn	DMA0 通道 2 全局中断
DMA0_Channel3_IRQn	DMA0 通道 3 全局中断
DMA0_Channel4_IRQn	DMA0 通道 4 全局中断
DMA0_Channel5_IRQn	DMA0 通道 5 全局中断
DMA0_Channe6_IRQn	DMA0 通道 6 全局中断
ADC0_1_IRQn	ADC0 和 ADC1 全局中断
EXTI5_9_IRQn	EXTI 线[9:5]中断
TIMER0_BRK_TIMER8_IRQn	TIMER0 中止中断和 TIMER8 全局中断
TIMER0_UP_TIMER9_IRQn	TIMER0 更新中断和 TIMER9 全局中断
TIMER0_TRG_CMT_TIMER10_IRQn	TIMER0 触发与通道换相中断和 TIMER10 全局中断
TIMER0_Channel_IRQn	TIMER0 通道捕获比较中断
TIMER1_IRQn	TIMER1 全局中断
TIMER2_IRQn	TIMER2 全局中断
TIMER3_IRQn	TIMER3 全局中断
I2C0_EV_IRQn	I2C0 事件中断
I2C0_ER_IRQn	I2C0 错误中断
I2C1_EV_IRQn	I2C1 事件中断
I2C1_ER_IRQn	I2C1 错误中断
SPI0_IRQn	SPI0 全局中断
SPI1_IRQn	SPI1 全局中断
USART0_IRQn	USART0 全局中断
USART1_IRQn	USART1 全局中断
USART2_IRQn	USART2 全局中断
EXTI10_15_IRQn	EXTI 线[15:10]中断
RTC_Alarm_IRQn	连接到 EXTI 线的 RTC 闹钟中断
USBFS_WKUP_IRQn	连接到 EXTI 线的 USBFS 唤醒中断
TIMER7_BRK_TIMER11_IRQn	TIMER7 中止中断和 TIMER11 全局中断

成员名称	功能描述
TIMER7_UP_TIMER12_IRQn	TIMER7 更新中断和 TIMER12 全局中断
TIMER7_TRG_CMT_TIMER13_IRQn	TIMER7 触发与通道换相中断和 TIMER13 全局中断
TIMER7_Channel_IRQn	TIMER7 通道捕获比较中断
EXMC_IRQn	EXMC 全局中断
TIMER4_IRQn	TIMER4 全局中断
SPI2_IRQn	SPI2 全局中断
UART3_IRQn	UART3 全局中断
UART4_IRQn	UART4 全局中断
TIMER5_IRQn	TIMER5 全局中断
TIMER6_IRQn	TIMER6 全局中断
DMA1_Channel0_IRQn	DMA1 通道 0 全局中断
DMA1_Channel1_IRQn	DMA1 通道 1 全局中断
DMA1_Channel2_IRQn	DMA1 通道 2 全局中断
DMA1_Channel3_IRQn	DMA1 通道 3 全局中断
DMA1_Channel4_IRQn	DMA1 通道 4 全局中断
USBFS_IRQn	USBFS 全局中断

MISC库函数列表如下表所示：

表 3-299. MISC 库函数

库函数名称	库函数描述
nvic_priority_group_set	设置优先级组
nvic_irq_enable	使能NVIC的中断
nvic_irq_disable	除能NVIC的中断
nvic_vector_table_set	设置向量表基地址
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SysTick时钟源

函数 nvic_priority_group_set

函数nvic_priority_group_set描述见下表：

表 3-300. 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	设置优先级组
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 3-301. 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_pre_priority, uint8_t nvic_irq_sub_priority);
功能描述	使能NVIC的中断
先决条件	-
被调用函数	nvic_priority_group_set
输入参数{in}	
nvic_irq	NVIC中断，参考枚举类型*错误!未找到引用源。*
输入参数{in}	
nvic_irq_pre_priority	抢占优先级
输入参数{in}	

nvic_irq_sub_priority	响应优先级
输出参数{out}	
-	-
返回值	
-	-

例如：

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

函数 nvic_irq_disable

函数nvic_irq_disable描述见下表：

表 3-302. 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	除能NVIC的中断
先决条件	-
被调用函数	-
输入参数{in}	
nvic_irq	NVIC中断，参考枚举类型*错误!未找到引用源。*
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 3-303. 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	设置向量表基地址
先决条件	-
被调用函数	-

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

例如：

```
/* set vector table address = NVIC_VECTTAB_FLASH +0x200 */
nvic_vector_table_set (NVIC_VECTTAB_FLASH,0x200);
```

函数 system_lowpower_set

函数system_lowpower_set描述见下表：

表 3-304. 函数 system_lowpower_set

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

例如：

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

system_lowpower_set (SCB_LPM_SLEEP_EXIT_ISR);

函数 system_lowpower_reset

函数system_lowpower_reset描述见下表：

表 3-305. 函数 system_lowpower_reset

函数名称	system_lowpower_reset
函数原形	void system_lowpower_reset(uint8_t lowpower_mode);
功能描述	复位系统低功耗模式状态
先决条件	-
被调用函数	-
输入参数{in}	
lowpower_mode	系统低功耗模式的状态
SCB_LPM_SLEEP_EXIT_ISR	该位为0时，退出ISR时退出低功耗模式
SCB_LPM_DEEPSLEEP	该位为0时，系统进入sleep模式
SCB_LPM_WAKE_BY_ALL_INT	该位为0时，系统只能被使能的中断唤醒
输出参数{out}	
-	-
返回值	
-	-

例如：

/* the system will exit low power mode by exiting from ISR */

system_lowpower_reset (SCB_LPM_SLEEP_EXIT_ISR);

函数 systick_clksource_set

函数systick_clksource_set描述见下表：

表 3-306. 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	设置SysTick时钟源
SYSTICK_CLKSOU	SysTick时钟源为AHB时钟

RCE_HCLK	
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 3-307. PMU 寄存器

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

3.16.2. 外设库函数说明

PMU 库函数列表如下表所示：

表 3-308. 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_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

库函数名称	库函数描述
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 3-309. 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 3-310. 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2.2V
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2.5V
PMU_LVDT_4	电压阈值为2.6V
PMU_LVDT_5	电压阈值为2.7V
PMU_LVDT_6	电压阈值为2.8V
PMU_LVDT_7	电压阈值为2.9V
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* select low voltage detector threshold as 2.9V */
pmu_lvd_select(PMU_LVDT_7);
```

函数 pmu_ldo_output_select

函数pmu_ldo_output_select描述见下表：

表 3-311. 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_NORMAL	输出正常电压模式
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 3-312. 函数 pmu_lvd_disable

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

例如：

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

函数 pmu_to_sleepmode

函数pmu_to_sleepmode描述见下表：

表 3-313. 函数 pmu_to_sleepmode

函数名称	pmu_to_sleepmode
函数原型	void pmu_to_sleepmode(uint8_t sleepmodecmd);
功能描述	进入睡眠模式
先决条件	-
被调用函数	-
输入参数{in}	
sleepmodecmd	进入睡眠模式命令
WFI_CMD	WFI命令
WFE_CMD	WFE命令
输出参数{out}	
-	-
返回值	
-	-

例如：

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

pmu_to_sleepmode(WFI_CMD);

函数 pmu_to_deepsleepmode

函数pmu_to_deepsleepmode描述见下表：

表 3-314. 函数 pmu_to_deepsleepmode

函数名称	pmu_to_deepsleepmode
函数原型	void pmu_to_deepsleepmode(uint32_t ldo,uint8_t deepsleepmodecmd);
功能描述	进入深度睡眠模式
先决条件	-
被调用函数	-
输入参数{in}	
ldo	LDO工作模式
PMU_LDO_NORMAL	当系统进入深度睡眠模式时，LDO仍正常工作
PMU_LDO_LOWPOWER	当系统进入深度睡眠模式时，LDO进入低功耗模式
输入参数{in}	
deepsleepmodecmd	进入深度睡眠模式命令
WFI_CMD	WFI命令
WFE_CMD	WFE命令
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* PMU work at deepsleep mode */
pmu_to_deepsleepmode(PMU_LDO_NORMAL, WFI_CMD);
```

函数 pmu_to_standbymode

函数pmu_to_standbymode描述见下表：

表 3-315. 函数 pmu_to_standbymode

函数名称	pmu_to_standbymode
函数原型	void pmu_to_standbymode(void);
功能描述	进入待机模式
先决条件	-
被调用函数	-

输入参数{in}	
-	-
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* PMU work at standby mode */
pmu_to_standbymode();
```

函数 pmu_wakeup_pin_enable

函数pmu_wakeup_pin_enable描述见下表：

表 3-316. 函数 pmu_wakeup_pin_enable

函数名称	pmu_wakeup_pin_enable
函数原型	void pmu_wakeup_pin_enable(void);
功能描述	WKUP引脚唤醒使能
先决条件	-
被调用函数	-
输入参数{in}	
-	-
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* enable wakeup pin */
pmu_wakeup_pin_enable();
```

函数 pmu_wakeup_pin_disable

函数pmu_wakeup_pin_disable描述见下表：

表 3-317. 函数 pmu_wakeup_pin_disable

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

输入参数{in}	
-	-
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* disable wakeup pin */
pmu_wakeup_pin_disable();
```

函数 pmu_backup_write_enable

函数pmu_backup_write_enable描述见下表：

表 3-318. 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 3-319. 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 3-320. 函数 pmu_flag_get

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

例如：

```
/* get flag state */
FlagStatus status;
status = pmu_flag_get(PMU_FLAG_WAKEUP);
```

函数 pmu_flag_clear

函数pmu_flag_clear描述见下表：

表 3-321. 函数 pmu_flag_clear

函数名称	pmu_flag_clear
函数原型	void pmu_flag_clear(uint32_t flag_reset);
功能描述	清除标志位
先决条件	-
被调用函数	-
输入参数{in}	
flag_reset	标志位
PMU_FLAG_RESET_WAKEUP	清除唤醒标志
PMU_FLAG_RESET_STANDBY	清除待机标志
输出参数{out}	
-	
返回值	
-	

例如：

```
/* clear flag bit */
pmu_flag_clear(PMU_FLAG_RESET_WAKEUP);
```

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 3-322. RCU 寄存器

寄存器名称	寄存器描述
RCU_CTL	控制寄存器
RCU_CFG0	时钟配置寄存器0
RCU_INT	时钟中断寄存器
RCU_APB2RST	APB2复位寄存器
RCU_APB1RST	APB1复位寄存器
RCU_AHBEN	AHB使能寄存器
RCU_APB2EN	APB2使能寄存器
RCU_APB1EN	APB1使能寄存器
RCU_BDCTL	备份域控制寄存器

寄存器名称	寄存器描述
RCU_RSTSCK	复位源/时钟寄存器
RCU_AHBRST	AHB复位寄存器
RCU_CFG1	时钟配置寄存器1
RCU_DSV	深度睡眠模式电压寄存器
RCU_ADDCTL	附加时钟控制寄存器
RCU_ADDINT	附加时钟中断寄存器
RCU_ADDAPB1RST	APB1附加复位寄存器
RCU_ADDAPB1EN	APB1附加使能寄存器

3.17.2. 外设库函数说明

RCU库函数列表如下表所示：

表 3-323. 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	使能外设复位
rcu_periph_reset_disable	禁能外设复位
rcu_bkp_reset_enable	使能BKP复位
rcu_bkp_reset_disable	禁能BKP复位
rcu_system_clock_source_config	配置选择系统时钟源
rcu_system_clock_source_get	获取系统时钟源选择状态
rcu_ahb_clock_config	配置AHB时钟预分频选择
rcu_apb1_clock_config	配置APB1时钟预分频选择
rcu_apb2_clock_config	配置APB2时钟预分频选择
rcu_ckout0_config	配置CKOUT0时钟源选择
rcu_pll_config	配置主PLL时钟
rcu_pllpresel_config	配置PLL时钟源选择
rcu_predv0_config	配置PREDV0分频因子
rcu_predv1_config	配置PREDV1分频因子
rcu_pll1_config	配置PLL1时钟
rcu_pll2_config	配置PLL2时钟
rcu_adc_clock_config	配置ADC的时钟分频系数
rcu_usb_clock_config	配置USB的时钟分频系数
rcu_rtc_clock_config	配置RTC的时钟源选择

库函数名称	库函数描述
rcu_i2s1_clock_config	配置I2S1的时钟源选择
rcu_i2s2_clock_config	配置I2S2的时钟源选择
rcu_ck48m_clock_config	配置CK48M时钟选择
rcu_flag_get	获取时钟稳定和外设复位标志
rcu_all_reset_flag_clear	清除所有复位标志位
rcu_interrupt_flag_get	获取时钟稳定中断和时钟阻塞中断标志
rcu_interrupt_flag_clear	清除中断标志
rcu_interrupt_enable	使能时钟稳定中断
rcu_interrupt_disable	禁能时钟稳定中断
rcu_lxtal_drive_capability_config	配置LXTAL驱动能力
rcu_osci_stab_wait	等待振荡器稳定标志位置位或振荡器起振超时
rcu_osci_on	打开振荡器
rcu_osci_off	关闭振荡器
rcu_osci_bypass_mode_enable	使能振荡器时钟旁路模式
rcu_osci_bypass_mode_disable	禁能振荡器时钟旁路模式
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_deepsleep_voltage_set	设置深度睡眠模式电压值
rcu_clock_freq_get	获取系统时钟、总线频率

枚举类型 rcu_periph_enum

表 3-324. 枚举类型 rcu_periph_enum

成员名称	功能描述
RCU_DMA0	DMA0时钟
RCU_DMA1	DMA1时钟
RCU_CRC	CRC时钟
RCU_EXMC	EXMC时钟
RCU_USBFS	USBFS时钟
RCU_TIMER1	TIMER1时钟
RCU_TIMER2	TIMER2时钟
RCU_TIMER3	TIMER3时钟
RCU_TIMER4	TIMER4时钟
RCU_TIMER5	TIMER5时钟
RCU_TIMER6	TIMER6时钟
RCU_TIMER11	TIMER11时钟
RCU_TIMER12	TIMER12时钟
RCU_TIMER13	TIMER13时钟
RCU_WWDGT	WWDGT时钟
RCU_SPI1	SPI1时钟

成员名称	功能描述
RCU_SPI2	SPI2时钟
RCU_USART1	USART1时钟
RCU_USART2	USART2时钟
RCU_UART3	UART3时钟
RCU_UART4	UART4时钟
RCU_I2C0	I2C0时钟
RCU_I2C1	I2C1时钟
RCU_BKPI	BKPI时钟
RCU_PMU	PMU时钟
RCU_DAC	DAC时钟
RCU_RTC	RTC时钟
RCU_CTC	CTC时钟
RCU_AF	复用功能时钟
RCU_GPIOA	GPIOA时钟
RCU_GPIOB	GPIOB时钟
RCU_GPIOC	GPIOC时钟
RCU_GPIOD	GPIOD时钟
RCU_GPIOE	GPIOE时钟
RCU_ADC0	ADC0时钟
RCU_ADC1	ADC1时钟
RCU_TIMER0	TIMER0时钟
RCU_SPI0	SPI0时钟
RCU_TIMER7	TIMER7时钟
RCU_USART0	USART0时钟
RCU_ADC2	ADC2时钟
RCU_TIMER8	TIMER8时钟
RCU_TIMER9	TIMER9时钟
RCU_TIMER10	TIMER10时钟

枚举类型 rcu_periph_sleep_enum

表 3-325. 枚举类型 rcu_periph_sleep_enum

成员名称	功能描述
RCU_SRAM_SLP	睡眠模式下SRAM时钟
RCU_FMC_SLP	睡眠模式下FMC时钟

枚举类型 rcu_periph_reset_enum

表 3-326. 枚举类型 rcu_periph_reset_enum

成员名称	功能描述
RCU_USBFSRST	USBFS时钟复位

成员名称	功能描述
RCU_TIMER1RST	TIMER1时钟复位
RCU_TIMER2RST	TIMER2时钟复位
RCU_TIMER3RST	TIMER3时钟复位
RCU_TIMER4RST	TIMER4时钟复位
RCU_TIMER5RST	TIMER5时钟复位
RCU_TIMER6RST	TIMER6时钟复位
RCU_TIMER11RST	TIMER11时钟复位
RCU_TIMER12RST	TIMER12时钟复位
RCU_TIMER13RST	TIMER13时钟复位
RCU_WWDGTRST	WWDGT时钟复位
RCU_SPI1RST	SPI1时钟复位
RCU_SPI2RST	SPI2时钟复位
RCU_USART1RST	USART1时钟复位
RCU_USART2RST	USART2时钟复位
RCU_UART3RST	UART3时钟复位
RCU_UART4RST	UART4时钟复位
RCU_I2C0RST	I2C0时钟复位
RCU_I2C1RST	I2C1时钟复位
RCU_USBDRST	USBD时钟复位
RCU_I2C2RST	I2C2时钟复位
RCU_BKPIRST	BKPI时钟复位
RCU_PMURST	PMU时钟复位
RCU_DACRST	DAC时钟复位
RCU_CTCRST	CTC时钟复位
RCU_AFRST	复用功能时钟复位
RCU_GPIOARST	GPIOA时钟复位
RCU_GPIOBRST	GPIOB时钟复位
RCU_GPIOCRST	GPIOC时钟复位
RCU_GPIODRST	GPIOD时钟复位
RCU_GPIOERST	GPIOE时钟复位
RCU_ADC0RST	ADC0时钟复位
RCU_ADC1RST	ADC1时钟复位
RCU_TIMER0RST	TIMER0时钟复位
RCU_SPI0RST	SPI0时钟复位
RCU_TIMER7RST	TIMER7时钟复位
RCU_USART0RST	USART0时钟复位
RCU_TIMER8RST	TIMER8时钟复位
RCU_TIMER9RST	TIMER9时钟复位
RCU_TIMER10RST	TIMER10时钟复位

枚举类型 rcu_flag_enum

表 3-327. 枚举类型 rcu_flag_enum

成员名称	功能描述
RCU_FLAG_IRC8MSTB	IRC8M振荡器稳定标志
RCU_FLAG_HXTALSTB	外部高速晶振稳定标志
RCU_FLAG_PLLSTB	PLL稳定标志
RCU_FLAG_PLL1STB	PLL1稳定标志
RCU_FLAG_PLL2STB	PLL2稳定标志
RCU_FLAG_LXTALSTB	LXTAL稳定标志
RCU_FLAG_IRC40KSTB	IRC40K稳定标志
RCU_FLAG_IRC48MSTB	IRC48M稳定标志
RCU_FLAG_EPRST	外部引脚复位标志
RCU_FLAG_PORRST	电源复位标志
RCU_FLAG_SWRST	软件复位标志
RCU_FLAG_FWDGTRST	独立看门狗复位标志
RCU_FLAG_WWDGTRST	窗口看门狗复位标志
RCU_FLAG_LPRST	低功耗复位标志

枚举类型 rcu_int_flag_enum

表 3-328. 枚举类型 rcu_int_flag_enum

成员名称	功能描述
RCU_INT_FLAG_IRC40KSTB	IRC40K时钟稳定中断标志
RCU_INT_FLAG_LXTALSTB	外部低速晶振时钟稳定中断标志
RCU_INT_FLAG_IRC8MSTB	IRC8M时钟稳定中断标志
RCU_INT_FLAG_HXTALSTB	外部高速晶振时钟稳定中断标志
RCU_INT_FLAG_PLLSTB	PLL时钟稳定中断标志
RCU_INT_FLAG_PLL1STB	PLL1时钟稳定中断标志

成员名称	功能描述
RCU_INT_FLAG_PLL2STB	PLL2时钟稳定中断标志
RCU_INT_FLAG_CKM	外部高速晶振时钟阻塞中断标志
RCU_INT_FLAG_IRC48MSTB	IRC48M时钟稳定中断标志

枚举类型 rcu_int_flag_clear_enum

表 3-329. 枚举类型 rcu_int_flag_clear_enum

成员名称	功能描述
RCU_INT_FLAG_IRC40KSTB_CLR	IRC40K时钟稳定中断清除标志
RCU_INT_FLAG_LXTALSTB_CLR	外部低速晶振时钟稳定中断清除标志
RCU_INT_FLAG_IRC8MSTB_CLR	IRC8M时钟稳定中断清除标志
RCU_INT_FLAG_HXTALSTB_CLR	外部高速晶振时钟稳定中断清除标志
RCU_INT_FLAG_PLLSTB_CLR	PLL时钟稳定中断清除标志
RCU_INT_FLAG_PLL1STB_CLR	PLL1时钟稳定中断清除标志
RCU_INT_FLAG_PLL2STB_CLR	PLL2时钟稳定中断清除标志
RCU_INT_FLAG_CKM_CLR	外部高速晶振时钟阻塞中断清除标志
RCU_INT_FLAG_IRC48MSTB_CLR	IRC48M时钟稳定中断清除标志

枚举类型 rcu_int_enum

表 3-330. 枚举类型 rcu_int_enum

成员名称	功能描述
RCU_INT_IRC40KSTB	IRC40K时钟稳定中断
RCU_INT_LXTALSTB	外部低速晶振时钟稳定中断
RCU_INT_IRC8MSTB	IRC8M时钟稳定中断
RCU_INT_HXTALSTB	外部高速晶振时钟稳定中断
RCU_INT_PLLSTB	PLL时钟稳定中断
RCU_INT_PLL1STB	PLL1时钟稳定中断
RCU_INT_PLL2STB	PLL2时钟稳定中断
RCU_INT_IRC48MSTB	IRC48M时钟稳定中断

枚举类型 rcu_osci_type_enum

表 3-331. 枚举类型 rcu_osci_type_enum

成员名称	功能描述
RCU_HXTAL	外部高速振荡器
RCU_LXTAL	外部低速振荡器
RCU_IRC8M	IRC8M振荡器
RCU_IRC48M	IRC48M振荡器
RCU_IRC40K	IRC40K振荡器
RCU_PLL_CK	锁相环时钟
RCU_PLL1_CK	锁相环1时钟
RCU_PLL2_CK	锁相环2时钟

枚举类型 rcu_clock_freq_enum

表 3-332. 枚举类型 rcu_clock_freq_enum

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

函数 rcu_deinit

函数rcu_deinit描述见下表：

表 3-333. 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	rcu_osci_stab_wait
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 3-334. 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3-324. 枚举类型rcu_periph_enum*
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 3-335. 函数 rcu_periph_clock_disable

函数名称	rcu_periph_clock_disable
函数原形	void rcu_periph_clock_disable(rcu_periph_enum periph);
功能描述	禁能外设时钟
先决条件	-
被调用函数	-
输入参数{in}	
periph	RCU外设，具体参考*表3-324. 枚举类型rcu_periph_enum*
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 3-336. 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，参考*表3-325. 枚举类型rcu_periph_sleep_enum*
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 3-337. 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，参考*表3-325. 枚举类型rcu_periph_sleep_enum*
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 3-338. 函数 rcu_periph_reset_enable

函数名称	rcu_periph_reset_enable
函数原形	void rcu_periph_reset_enable(rcu_periph_reset_enum periph_reset);
功能描述	使能外设复位
先决条件	-
被调用函数	-
输入参数{in}	
periph_reset	RCU外设复位，参考*表3-326. 枚举类型rcu_periph_reset_enum*
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 3-339. 函数 rcu_periph_reset_disable

函数名称	rcu_periph_reset_disable
函数原形	void rcu_periph_reset_disable(rcu_periph_reset_enum periph_reset);
功能描述	禁能外设复位
先决条件	-
被调用函数	-
输入参数{in}	
periph_reset	RCU外设复位，参考*表3-326. 枚举类型rcu_periph_reset_enum*
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 3-340. 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 3-341. 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 3-342. 函数 rcu_system_clock_source_config

函数名称	rcu_system_clock_source_config
函数原形	void rcu_system_clock_source_config(uint32_t ck_sys);
功能描述	配置选择系统时钟源
先决条件	-
被调用函数	-
输入参数{in}	
ck_sys	系统时钟源选择
RCU_CKSYSSRC_IRC8M	选择CK_IRC8M时钟作为CK_SYS时钟源
RCU_CKSYSSRC_HXTAL	选择CK_HXTAL时钟作为CK_SYS时钟源
RCU_CKSYSSRC_PLL	选择CK_PLL时钟作为CK_SYS时钟源
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* configure the CK_HXTAL as the CK_SYS source */
rcu_system_clock_source_config(RCU_CKSYSSRC_HXTAL);
```

函数 rcu_system_clock_source_get

函数rcu_system_clock_source_get描述见下表：

表 3-343. 函数 rcu_system_clock_source_get

函数名称	rcu_system_clock_source_get
函数原形	uint32_t rcu_system_clock_source_get(void);
功能描述	获取系统时钟源选择状态
先决条件	-
被调用函数	-
输入参数{in}	
-	-
输出参数{out}	

-	-
返回值	
uint32_t	RCU_SCSS_IRC8M/RCU_SCSS_HXTAL/RCU_SCSS_PLL

例如：

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

函数 rcu_ahb_clock_config

函数rcu_ahb_clock_config描述见下表：

表 3-344. 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 3-345. 函数 rcu_apb1_clock_config

函数名称	rcu_apb1_clock_config
函数原形	void rcu_apb1_clock_config(uint32_t ck_apb1);
功能描述	配置APB1时钟预分频选择
先决条件	-

被调用函数	-
输入参数{in}	
ck_apb1	APB1预分频选择
RCU_APB1_CKAHB_DIV1	选择CK_AHB为CK_APB1
RCU_APB1_CKAHB_DIV2	选择CK_AHB/2为CK_APB1
RCU_APB1_CKAHB_DIV4	选择CK_AHB/4为CK_APB1
RCU_APB1_CKAHB_DIV8	选择CK_AHB/8为CK_APB1
RCU_APB1_CKAHB_DIV16	选择CK_AHB/16为CK_APB1
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* configure CK_AHB/16 as CK_APB1 */
rcu_apb1_clock_config(RCU_APB1_CKAHB_DIV16);
```

函数 rcu_apb2_clock_config

函数rcu_apb2_clock_config描述见下表：

表 3-346. 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}	
ck_apb2	APB2预分频选择
RCU_APB2_CKAHB_DIV1	选择CK_AHB为CK_APB2
RCU_APB2_CKAHB_DIV2	选择CK_AHB/2为CK_APB2
RCU_APB2_CKAHB_DIV4	选择CK_AHB/4为CK_APB2
RCU_APB2_CKAHB_DIV8	选择CK_AHB/8为CK_APB2

RCU_APB2_CKAHB_DIV16	选择CK_AHB/16为CK_APB2
输出参数{out}	
-	-
返回值	
-	-

例如：

/* configure CK_AHB/8 as CK_APB2 */

rcu_apb2_clock_config(RCU_APB2_CKAHB_DIV8);

函数 rcu_ckout0_config

函数rcu_ckout0_config描述见下表：

表 3-347. 函数 rcu_ckout0_config

函数名称	rcu_ckout0_config
函数原形	void rcu_ckout0_config(uint32_t ckout0_src);
功能描述	配置CKOUT0时钟源选择
先决条件	-
被调用函数	-
输入参数{in}	
ckout0_src	CKOUT0时钟源选择
RCU_CKOUT0SRC_NONE	无时钟输出
RCU_CKOUT0SRC_CKSYS	选择系统时钟CK_SYS
RCU_CKOUT0SRC_IRC8M	选择内部8M RC振荡器时钟
RCU_CKOUT0SRC_HXTAL	选择高速晶体振荡器时钟(HXTAL)
RCU_CKOUT0SRC_CKPLL_DIV2	选择(CK_PLL / 2)时钟
RCU_CKOUT0SRC_CKPLL1	选择CK_PLL1时钟
RCU_CKOUT0SRC_CKPLL2_DIV2	选择(CK_PLL2 / 2)时钟
RCU_CKOUT0SRC_CKPLL2	选择CK_PLL2时钟
RCU_CKOUT0SRC_IRC48M	选择IRC48M时钟
RCU_CKOUT0SRC	选择(IRC48M / 8)时钟

_IRC48M_DIV8	
输出参数{out}	
-	-
返回值	
-	-

例如：

/* configure the HXTAL as CK_OUT0 clock source */

rcu_ckout0_config(RCU_CKOUT0SRC_HXTAL);

函数 rcu_pll_config

函数rcu_pll_config描述见下表：

表 3-348. 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
先决条件	-
被调用函数	-
输入参数{in}	
pll_src	PLL时钟源选择
RCU_PLLSRC_IRC8M_DIV2	(IRC8M / 2)被选择为PLL时钟的时钟源
RCU_PLLSRC_HXTAL_IRC48M	HXTAL时钟或者IRC48M时钟被选择为PLL时钟的时钟源
输入参数{in}	
pll_mul	PLL时钟倍频因子
RCU_PLL_MULx	PLL源时钟 * x (x = 2..14, 6.5, 16..31)
输出参数{out}	
-	-
返回值	
-	-

例如：

/* configure the PLL */

rcu_pll_config(RCU_PLLSRC_HXTAL, RCU_PLL_MUL10);

函数 rcu_pllpresel_config

函数rcu_pllpresel_config描述见下表：

表 3-349. 函数 rcu_pllpresel_config

函数名称	rcu_pllpresel_config
函数原形	void rcu_pllpresel_config(uint32_t pll_presel);
功能描述	配置PLL时钟源预选
先决条件	-
被调用函数	-
输入参数{in}	
pll_presel	PLL时钟源
RCU_PLLPRESRC_HXTAL	PLL输入为HXTAL
RCU_PLLPRESRC_IRC48M	PLL输入为IRC48M
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* configure the PLL preselection */
rcu_pllpresel_config (RCU_PLLPRESRC_HXTAL);
```

函数 rcu_predv0_config

函数rcu_predv0_config描述见下表：

表 3-350. 函数 rcu_predv0_config

函数名称	rcu_predv0_config
函数原形	void rcu_predv0_config(uint32_t predv0_source, uint32_t predv0_div);
功能描述	配置PREDV0分频因子
先决条件	-
被调用函数	-
输入参数{in}	
predv0_source	PREDV0输入时钟源
RCU_PREDV0SRC_HXTAL_IRC48M	HXTAL或者IRC48M被选择为PREDV0的时钟源
RCU_PREDV0SRC_CKPLL1	CK_PLL1被选择为PREDV0的时钟源
输入参数{in}	
predv0_div	PREDV0分频因子
RCU_PREDV0_DIVx	PREDV0输入源时钟x分频(x = 1..16)
输出参数{out}	

-	-
返回值	
-	-

例如：

/* configure the PREDV0 division factor */

rcu_predv0_config(RCU_PREDV0SRC_HXTAL_IRC48M, RCU_PREDV0_DIV4);

函数 rcu_predv1_config

函数rcu_predv1_config描述见下表：

表 3-351. 函数 rcu_predv1_config

函数名称	rcu_predv1_config
函数原形	void rcu_predv1_config(uint32_t predv1_div);
功能描述	配置PREDV1分频因子
先决条件	-
被调用函数	-
输入参数{in}	
predv1_div	PREDV1分频因子
RCU_PREDV1_DIVx	PREDV1输入源时钟x分频(x = 1..16)
输出参数{out}	
-	-
返回值	
-	-

例如：

/* configure the PREDV1 division factor */

rcu_predv1_config(RCU_PREDV1_DIV8);

函数 rcu_pll1_config

函数rcu_pll1_config描述见下表：

表 3-352. 函数 rcu_pll1_config

函数名称	rcu_pll1_config
函数原形	void rcu_pll1_config(uint32_t pll_mul);
功能描述	配置PLL1时钟
先决条件	-
被调用函数	-
输入参数{in}	

pll_mul	PLL1时钟倍频因子
RCU_PLL1_MULx	PLL1源时钟*x，(x = 8..16, 20)
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* configure the PLL1 clock */
rcu_pll1_config(RCU_PLL1_MUL8);
```

函数 rcu_pll2_config

函数rcu_pll2_config描述见下表：

表 3-353. 函数 rcu_pll2_config

函数名称	rcu_pll2_config
函数原形	void rcu_pll2_config(uint32_t pll_mul);
功能描述	配置PLL2时钟
先决条件	-
被调用函数	-
输入参数{in}	
pll_mul	PLL2时钟倍频因子
RCU_PLL2_MULx	PLL2源时钟*x，(x = 8..16, 20)
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* configure the PLL2 clock */
rcu_pll2_config(RCU_PLL2_MUL8);
```

函数 rcu_adc_clock_config

函数rcu_adc_clock_config描述见下表：

表 3-354. 函数 rcu_adc_clock_config

函数名称	rcu_adc_clock_config
函数原形	void rcu_adc_clock_config(uint32_t adc_psc);
功能描述	配置ADC的时钟分频系数
先决条件	-

被调用函数	-
输入参数{in}	
adc_psc	ADC分频因子
RCU_CKADC_CKAPB2_DIV2	CK_ADC = CK_APB2 / 2
RCU_CKADC_CKAPB2_DIV4	CK_ADC = CK_APB2 / 4
RCU_CKADC_CKAPB2_DIV6	CK_ADC = CK_APB2 / 6
RCU_CKADC_CKAPB2_DIV8	CK_ADC = CK_APB2 / 8
RCU_CKADC_CKAPB2_DIV12	CK_ADC = CK_APB2 / 12
RCU_CKADC_CKAPB2_DIV16	CK_ADC = CK_APB2 / 16
RCU_CKADC_CKAHB_DIV3	CK_ADC = CK_AHB / 3
RCU_CKADC_CKAHB_DIV5	CK_ADC = CK_ AHB / 5
RCU_CKADC_CKAHB_DIV7	CK_ADC = CK_ AHB / 7
RCU_CKADC_CKAHB_DIV9	CK_ADC = CK_ AHB / 9
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* configure the ADC prescaler factor */
rcu_adc_clock_config(RCU_CKADC_CKAPB2_DIV8);
```

函数 rcu_usb_clock_config

函数rcu_usb_clock_config描述见下表：

表 3-355. 函数 rcu_usb_clock_config

函数名称	rcu_usb_clock_config
函数原形	void rcu_usb_clock_config(uint32_t usb_psc);
功能描述	配置USB的时钟分频系数
先决条件	-
被调用函数	-

输入参数{in}	
usb_psc	USB时钟分频系数
RCU_CKUSB_CKPLL_DIV1_5	CK_USBFS = CK_PLL / 1.5
RCU_CKUSB_CKPLL_DIV1	CK_USBFS = CK_PLL / 1
RCU_CKUSB_CKPLL_DIV2_5	CK_USBFS = CK_PLL / 2.5
RCU_CKUSB_CKPLL_DIV2	CK_USBFS = CK_PLL / 2
RCU_CKUSB_CKPLL_DIV3	CK_USBFS = CK_PLL / 3
RCU_CKUSB_CKPLL_DIV3_5	CK_USBFS = CK_PLL / 3.5
RCU_CKUSB_CKPLL_DIV4	CK_USBFS = CK_PLL / 4
输出参数{out}	
-	-
返回值	
-	-

例如：

/* configure the USB prescaler factor */

rcu_usb_clock_config(RCU_CKUSB_CKPLL_DIV2_5);

函数 rcu_rtc_clock_config

函数rcu_rtc_clock_config描述见下表：

表 3-356. 函数 rcu_rtc_clock_config

函数名称	rcu_rtc_clock_config
函数原形	void rcu_rtc_clock_config(uint32_t rtc_clock_source);
功能描述	配置RTC的时钟源选择
先决条件	-
被调用函数	-
输入参数{in}	
rtc_clock_source	RTC时钟源选择
RCU_RTCSRC_NONE	没有时钟
RCU_RTCSRC_LXTAL	选择CK_LXTAL时钟作为RTC的时钟源
RCU_RTCSRC_IRC	选择CK_IRC40K时钟作为RTC的时钟源

40K	
RCU_RTCSRC_HXTAL_DIV_128	选择CK_HXTAL / 128时钟作为RTC的时钟源
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* configure the RTC clock source selection */
rcu_rtc_clock_config(RCU_RTCSRC_IRC40K);
```

函数 rcu_i2s1_clock_config

函数rcu_i2s1_clock_config描述见下表：

表 3-357. 函数 rcu_i2s1_clock_config

函数名称	rcu_i2s1_clock_config
函数原形	void rcu_i2s1_clock_config(uint32_t i2s_clock_source);
功能描述	配置I2S1的时钟源选择
先决条件	-
被调用函数	-
输入参数{in}	
i2s_clock_source	I2S1时钟源选择
RCU_I2S1SRC_CKSYS	系统时钟被选择为I2S1时钟的时钟源
RCU_I2S1SRC_CKPLL2_MUL2	(CK_PLL2 * 2) 被选择为I2S1时钟的时钟源
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* configure the I2S1 clock source selection */
rcu_i2s1_clock_config(RCU_I2S1SRC_CKPLL2_MUL2);
```

函数 rcu_i2s2_clock_config

函数rcu_i2s2_clock_config描述见下表：

表 3-358. 函数 rcu_i2s2_clock_config

函数名称	rcu_i2s2_clock_config
函数原形	void rcu_i2s2_clock_config(uint32_t i2s_clock_source);
功能描述	配置I2S2的时钟源选择
先决条件	-
被调用函数	
输入参数{in}	
i2s_clock_source	I2S2时钟源选择
RCU_I2S2SRC_CKSYS	系统时钟被选择为I2S2时钟的时钟源
RCU_I2S2SRC_CKPLL2_MUL2	(CK_PLL2 * 2) 被选择为I2S2时钟的时钟源
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* configure the I2S2 clock source selection */
rcu_i2s2_clock_config(RCU_I2S2SRC_CKPLL2_MUL2);
```

函数 rcu_ck48m_clock_config

函数rcu_ck48m_clock_config描述见下表:

表 3-359. 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时钟源选择
先决条件	-
被调用函数	-
输入参数{in}	
ck48m_clock_source	CK48M时钟源
RCU_CK48MSRC_CKPLL	选择CK_PLL为CK_48M时钟源
RCU_CK48MSRC_IRC48M	选择CK_IRC48M为CK48M时钟源
输出参数{out}	
-	-
返回值	
FlagStatus	SET 或 RESET

例如：

```
/* configure the CK48M clock source selection */
rcu_ck48m_clock_config (RCU_CK48MSRC_IRC48M);
```

函数 rcu_flag_get

函数rcu_flag_get描述见下表：

表 3-360. 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3-327. 枚举类型rcu_flag_enum*
输出参数{out}	
-	-
返回值	
FlagStatus	SET 或 RESET

例如：

```
/* get the clock stabilization flag */
if(RESET != rcu_flag_get(RCU_FLAG_LXTALSTB)){
}
```

函数 rcu_all_reset_flag_clear

函数rcu_all_reset_flag_clear描述见下表：

表 3-361. 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 3-362. 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3-328. 枚举类型rcu_int_flag_enum*
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 3-363. 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3-329. 枚举类型 rcu_int_flag_clear_enum*

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 3-364. 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
RCU_INT_IRC40KSTB	IRC40K稳定中断
RCU_INT_LXTALSTB	LXTAL稳定中断
RCU_INT_IRC8MSTB	IRC8M稳定中断使能
RCU_INT_HXTALSTB	HXTAL稳定中断
RCU_INT_PLLSTB	PLL稳定中断
RCU_INT_PLL1STB	PLL1稳定中断
RCU_INT_PLL2STB	PLL2稳定中断
RCU_INT_IRC48MSTB	IRC48M稳定中断使能
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 3-365. 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	禁能时钟稳定中断
先决条件	-
被调用函数	-
输入参数{in}	
stab_int	时钟稳定中断
RCU_INT_IRC40KSTB	IRC40K稳定中断
RCU_INT_LXTALSTB	LXTAL稳定中断
RCU_INT_IRC8MSTB	IRC8M稳定中断使能
RCU_INT_HXTALSTB	HXTAL稳定中断
RCU_INT_PLLSTB	PLL稳定中断
RCU_INT_PLL1STB	PLL1稳定中断
RCU_INT_PLL2STB	PLL2稳定中断
RCU_INT_IRC48MSTB	IRC48M稳定中断使能
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* disable the HXTAL stabilization interrupt */
rcu_interrupt_disable(RCU_INT_HXTALSTB);
```

函数 rcu_lxtal_drive_capability_config

函数rcu_lxtal_drive_capability_config描述见下表：

表 3-366. 函数 rcu_lxtal_drive_capability_config

函数名称	rcu_lxtal_drive_capability_config
函数原形	void rcu_lxtal_drive_capability_config(uint32_t lxtal_dricap);
功能描述	配置LXTAL驱动能力

先决条件	-
被调用函数	-
输入参数{in}	
lxtal_dricap	LXTAL驱动能力
RCU_LXTAL_LOWDRI	低驱动能力
RCU_LXTAL_MED_LOWDRI	中低驱动能力
RCU_LXTAL_MED_HIGHDRI	中高驱动能力
RCU_LXTAL_HIGHDRI	高驱动能力
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* configure the LXTAL drive capability */
rcu_lxtal_drive_capability_config (RCU_LXTAL_LOWDRI);
```

函数 rcu_osci_stab_wait

函数rcu_osci_stab_wait描述见下表：

表 3-367. 函数 rcu_osci_stab_wait

函数名称	rcu_osci_stab_wait
函数原形	ErrStatus rcu_osci_stab_wait(rcu_osci_type_enum osci);
功能描述	等待振荡器稳定标志位置位或振荡器起振超时
先决条件	-
被调用函数	rcu_flag_get
输入参数{in}	
osci	振荡器类型，参考*表3-331. 枚举类型rcu_osci_type_enum*
输出参数{out}	
-	-
返回值	
ErrStatus	SUCCESS 或 ERROR

例如：

```
/* wait for oscillator stabilization flag */
if(SUCCESS == rcu_osci_stab_wait(RCU_HXTAL)){
```

}

函数 rcu_osci_on

函数rcu_osci_on描述见下表：

表 3-368. 函数 rcu_osci_on

函数名称	rcu_osci_on
函数原形	void rcu_osci_on(rcu_osci_type_enum osci);
功能描述	打开振荡器
先决条件	-
被调用函数	-
输入参数{in}	
osci	振荡器类型，参考*表3-331. 枚举类型rcu_osci_type_enum*
输出参数{out}	
-	-
返回值	
-	-

例如：

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

函数 rcu_osci_off

函数rcu_osci_off描述见下表：

表 3-369. 函数 rcu_osci_off

函数名称	rcu_osci_off
函数原形	void rcu_osci_off(rcu_osci_type_enum osci);
功能描述	关闭振荡器
先决条件	-
被调用函数	-
输入参数{in}	
osci	振荡器类型，参考*表3-331. 枚举类型rcu_osci_type_enum*
输出参数{out}	
-	-
返回值	
-	-

例如：

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

rcu_osci_off(RCU_HXTAL);

函数 rcu_osci_bypass_mode_enable

函数rcu_osci_bypass_mode_enable描述见下表：

表 3-370. 函数 rcu_osci_bypass_mode_enable

函数名称	rcu_osci_bypass_mode_enable
函数原形	void rcu_osci_bypass_mode_enable(rcu_osci_type_enum osci);
功能描述	使能振荡器时钟旁路模式
先决条件	HXTALEN或LXTALEN应在使能振荡器时钟旁路模式前先复位
被调用函数	-
输入参数{in}	
osci	振荡器类型，参考*表3-331. 枚举类型rcu_osci_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_osci_bypass_mode_enable(RCU_HXTAL);
```

函数 rcu_osci_bypass_mode_disable

函数rcu_osci_bypass_mode_disable描述见下表：

表 3-371. 函数 rcu_osci_bypass_mode_disable

函数名称	rcu_osci_bypass_mode_disable
函数原形	void rcu_osci_bypass_mode_disable(rcu_osci_type_enum osci);
功能描述	禁能振荡器时钟旁路模式
先决条件	HXTALEN或LXTALEN应在使能振荡器时钟旁路模式前先复位
被调用函数	-
输入参数{in}	
osci	振荡器类型，参考*表3-331. 枚举类型rcu_osci_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 high speed crystal oscillator bypass mode */

rcu_osci_bypass_mode_disable(RCU_HXTAL);

函数 rcu_hxtal_clock_monitor_enable

函数rcu_hxtal_clock_monitor_enable描述见下表：

表 3-372. 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 3-373. 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	禁能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 3-374. 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_deepsleep_voltage_set

函数rcu_deepsleep_voltage_set描述见下表：

表 3-375. 函数 rcu_deepsleep_voltage_set

函数名称	rcu_deepsleep_voltage_set
函数原形	void rcu_deepsleep_voltage_set(uint32_t dsvol);
功能描述	设置深度睡眠模式电压值
先决条件	-
被调用函数	-
输入参数{in}	
dsvol	深度睡眠模式电压值
RCU_DEEPSLEEP_V_1_0	在深度睡眠模式下内核电压为1.0V
RCU_DEEPSLEEP	在深度睡眠模式下内核电压为0.9V

_V_0_9	
RCU_DEEPSLEEP_V_0_8	在深度睡眠模式下内核电压为0.8V
RCU_DEEPSLEEP_V_1_2	在深度睡眠模式下内核电压为1.2V
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 3-376. 函数 rcu_clock_freq_get

函数名称	rcu_clock_freq_get
函数原形	uint32_t rcu_clock_freq_get(rcu_clock_freq_enum clock);
功能描述	获取系统时钟、总线频率
先决条件	-
被调用函数	-
输入参数{in}	
clock	要获取的时钟频率
CK_SYS	系统时钟
CK_AHB	AHB时钟
CK_APB1	APB1时钟
CK_APB2	APB2时钟
输出参数{out}	
-	-
返回值	
ck_freq	系统时钟/AHB时钟/APB1时钟/APB2时钟频率

例如：

```
uint32_t temp_freq;
/* get the system clock frequency */
temp_freq = rcu_clock_freq_get(CK_SYS);
```

3.18. RTC

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

3.18.1. 外设寄存器描述

RTC寄存器列表如下表所示：

表 3-377. RTC 寄存器

寄存器名称	寄存器描述
RTC_INTEN	中断使能寄存器
RTC_CTL	控制寄存器
RTC_PSCH	预分频寄存器高位
RTC_PSCL	预分频寄存器低位
RTC_DIVH	分频寄存器高位
RTC_DIVL	分频寄存器低位
RTC_CNTH	计数寄存器高位
RTC_CNTL	计数寄存器低位
RTC_ALRMH	闹钟寄存器高位
RTC_ALRML	闹钟寄存器低位

3.18.2. 外设库函数描述

RTC库函数列表如下表所示：

表 3-378. RTC 库函数

库函数名称	库函数描述
rtc_configuration_mode_enter	进入RTC配置模式
rtc_configuration_mode_exit	退出RTC配置模式
rtc_counter_set	设置RTC计数器的值
rtc_prescaler_set	设置RTC预分频值
rtc_lwoff_wait	等待最近一次对RTC寄存器的写操作完成
rtc_register_sync_wait	等待RTC寄存器(RTC_CNTx、RTC_ALRMx和RTC_PSCx)与RTC的APB时钟同步
rtc_alarm_config	设置RTC闹钟值
rtc_counter_get	获取RTC计数器的值
rtc_divider_get	获取RTC分频值
rtc_flag_get	获取RTC标志位状态
rtc_flag_clear	清除RTC标志位状态
rtc_interrupt_flag_get	获取RTC中断标志位状态

库函数名称	库函数描述
rtc_interrupt_flag_clear	清除RTC中断标志位状态
rtc_interrupt_enable	使能RTC中断
rtc_interrupt_disable	失能RTC中断

函数 rtc_configuration_mode_enter

函数rtc_configuration_mode_enter描述见下表：

表 3-379. 函数 rtc_configuration_mode_enter

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

例如：

```
/* enter RTC configuration mode */

rtc_configuration_mode_enter( );
```

函数 rtc_configuration_mode_exit

函数rtc_configuration_mode_exit描述见下表：

表 3-380. 函数 rtc_configuration_mode_exit

函数名称	rtc_configuration_mode_exit
函数原型	void rtc_configuration_mode_exit(void);
功能描述	退出RTC配置模式
先决条件	-
被调用函数	-
输入参数{in}	
-	-
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* exit RTC configuration mode */
rtc_configuration_mode_ exit ( );
```

函数 rtc_counter_set

函数rtc_counter_set描述见下表：

表 3-381. 函数 rtc_counter_set

函数名称	rtc_counter_set
函数原型	void rtc_counter_set(uint32_t cnt);
功能描述	设置RTC计数器的值
先决条件	-
被调用函数	-
输入参数{in}	
cnt	RTC计数器的值(0-0xFFFF FFFF)
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* wait until last write operation on RTC registers has finished */
rtc_lwoff_wait( );
/* set counter value to 0xFFFF */
rtc_counter_set (0xFFFF);
```

函数 rtc_prescaler_set

函数rtc_prescaler_set描述见下表：

表 3-382. 函数 rtc_prescaler_set

函数名称	rtc_interrupt_rtc_prescaler_set
函数原型	void rtc_prescaler_set(uint32_t psc);
功能描述	设置RTC预分频值
先决条件	调用此函数之前，必须调用函数rtc_lwoff_wait ( )(等待标志位LWOFF置位)
被调用函数	rtc_configuration_mode_enter / rtc_configuration_mode_exit
输入参数{in}	
psc	RTC预分频值(0-0x000F FFFF)
输出参数{out}	
-	-

返回值	
-	-

例如：

```
/* wait until last write operation on RTC registers has finished */
rtc_lwoff_wait( );
/* set RTC prescaler value to 0x7FFFF */
rtc_prescaler_set (0x7FFFF);
```

函数 rtc_lwoff_wait

函数rtc_lwoff_wait描述见下表：

表 3-383. 函数 rtc_lwoff_wait

函数名称	rtc_lwoff_wait
函数原型	void rtc_lwoff_wait(void);
功能描述	等待最近一次对RTC寄存器的写操作完成
先决条件	-
被调用函数	-
输入参数{in}	
-	-
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* wait until last write operation on RTC registers has finished */
rtc_lwoff_wait( );
/* enable the RTC second interrupt */
rtc_interrupt_enable(RTC_INT_SECOND);
```

函数 rtc_register_sync_wait

函数rtc_register_sync_wait描述见下表：

表 3-384. 函数 rtc_register_sync_wait

函数名称	rtc_register_sync_wait
函数原型	void rtc_register_sync_wait(void);
功能描述	等待RTC寄存器(RTC_CNTx、RTC_ALRMx和RTC_PSCx)与RTC的APB时钟

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

例如：

```
/* wait for RTC registers synchronization */
rtc_register_sync_wait( );
```

函数 rtc_alarm_config

函数rtc_alarm_config描述见下表：

表 3-385. 函数 rtc_alarm_config

函数名称	rtc_alarm_config
函数原型	void rtc_alarm_config(uint32_t alarm);
功能描述	设置RTC闹钟值
先决条件	调用此函数之前，必须调用函数rtc_lwoff_wait ( )(等待标志位LWOFF置位)
被调用函数	rtc_configuration_mode_enter / rtc_configuration_mode_exit
输入参数{in}	
alarm	RTC闹钟值(0-0xFFFF FFFF)
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* wait until last write operation on RTC registers has finished */
rtc_lwoff_wait( );
/* set alarm value to 0xFFFF */
rtc_alarm_config (0xFFFF);
```

函数 rtc_counter_get

函数rtc_counter_get描述见下表：

表 3-386. 函数 rtc_counter_get

函数名称	rtc_counter_get
函数原型	uint32_t rtc_counter_get(void);
功能描述	获取RTC计时器的值
先决条件	-
被调用函数	-
输入参数{in}	
-	-
输出参数{out}	
RTC counter value	RTC计时器的值
返回值	
-	-

例如：

```
/* get the counter value */
uint32_t rtc_counter_value;
rtc_counter_value = rtc_counter_get ( );
```

函数 rtc_divider_get

函数rtc_divider_get描述见下表：

表 3-387. 函数 rtc_divider_get

函数名称	rtc_divider_get
函数原型	uint32_t rtc_divider_get(void);
功能描述	获取RTC分频值
先决条件	-
被调用函数	-
输入参数{in}	
-	-
输出参数{out}	
-	-
返回值	
RTC divider value	RTC分频值

例如：

```
/* get the current RTC divider value */
uint32_t rtc_divider_value;
rtc_divider_value = rtc_divider_get ( );
```

函数 rtc_flag_get

函数rtc_flag_get描述见下表：

表 3-388. 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RTC标志位状态
先决条件	-
被调用函数	-
输入参数{in}	
flag	需要获取的RTC标志位
RTC_FLAG_SECOND	秒中断标志位
RTC_FLAG_ALARM	闹钟中断标志位
RTC_FLAG_OVERFLOW	溢出中断标志位
RTC_FLAG_RSYN	寄存器同步标志位
RTC_FLAG_LWOF	最近一次对RTC寄存器的写操作完成标志位
输出参数{out}	
-	-
返回值	
FlagStatus	SET 或 RESET

例如：

```
/* get the RTC overflow interrupt status */
FlagStatus alarm_status;
alarm_status = rtc_flag_get (RTC_FLAG_ALARM);
```

函数 rtc_flag_clear

函数rtc_flag_clear描述见下表：

表 3-389. 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RTC标志位状态
先决条件	-
被调用函数	-
输入参数{in}	
flag	待清除的RTC标志位
RTC_FLAG_SECON	秒中断标志位

D	
RTC_FLAG_ALARM	闹钟中断标志位
RTC_FLAG_OVERFLOW	溢出中断标志位
RTC_FLAG_RSYN	寄存器同步标志位
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* clear the RTC alarm flag */
rtc_flag_clear (RTC_FLAG_ALARM);
```

函数 rtc_ interrupt_flag_get

函数rtc_ interrupt_flag_get描述见下表：

表 3-390. 函数 rtc_flag_get

函数名称	rtc_ interrupt_flag_get
函数原型	FlagStatus rtc_ interrupt_flag_get(uint32_t flag);
功能描述	获取RTC中断标志位状态
先决条件	-
被调用函数	-
输入参数{in}	
flag	需要获取的RTC中断标志位
RTC_INT_FLAG_SECOND	秒中断标志位
RTC_INT_FLAG_ALARM	闹钟中断标志位
RTC_INT_FLAG_OVERFLOW	溢出中断标志位
输出参数{out}	
-	-
返回值	
FlagStatus	SET 或 RESET

例如：

```
/* get the RTC alarm interrupt status */
FlagStatus alarm_status;
alarm_status = rtc_interrupt_flag_get (RTC_INT_FLAG_ALARM);
```

函数 rtc_interrupt_flag_clear

函数rtc_interrupt_flag_clear描述见下表：

表 3-391. 函数 rtc_interrupt_flag_clear

函数名称	rtc_interrupt_flag_clear
函数原型	void rtc_interrupt_flag_clear(uint32_t flag);
功能描述	清除RTC中断标志位状态
先决条件	-
被调用函数	-
输入参数{in}	
flag	待清除的RTC中断标志位
RTC_INT_FLAG_SECOND	秒中断标志位
RTC_INT_FLAG_ALARM	闹钟中断标志位
RTC_INT_FLAG_OVERFLOW	溢出中断标志位
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* clear the RTC alarm interrupt flag */
rtc_interrupt_flag_clear (RTC_INT_FLAG_ALARM);
```

函数 rtc_interrupt_enable

函数rtc_interrupt_enable描述见下表：

表 3-392. 函数 rtc_interrupt_enable

函数名称	rtc_interrupt_enable
函数原型	void rtc_interrupt_enable(uint32_t interrupt);
功能描述	使能RTC中断
先决条件	调用此函数之前，必须调用函数rtc_lwoff_wait ( )(等待标志位LWOFF置位)
被调用函数	-
输入参数{in}	
interrupt	待使能的RTC中断源
RTC_INT_SECOND	秒中断
RTC_INT_ALARM	闹钟中断
RTC_INT_OVERFL	溢出中断

OW	
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* wait until last write operation on RTC registers has finished */
rtc_lwoff_wait( );
/* enable the RTC second interrupt */
rtc_interrupt_enable(RTC_INT_SECOND);
```

函数 rtc_interrupt_disable

函数rtc_interrupt_disable描述见下表：

表 3-393. 函数 rtc_interrupt_disable

函数名称	rtc_interrupt_disable
函数原型	void rtc_interrupt_disable(uint32_t interrupt);
功能描述	失能RTC中断
先决条件	调用此函数之前，必须调用函数rtc_lwoff_wait ( )(等待标志位LWOFF置位)
被调用函数	-
输入参数{in}	
interrupt	待失能的RTC中断源
RTC_INT_SECOND	秒中断
RTC_INT_ALARM	闹钟中断
RTC_INT_OVERFLOW	溢出中断
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* wait until last write operation on RTC registers has finished */
rtc_lwoff_wait( );
/* disable the RTC second interrupt */
rtc_interrupt_disable(RTC_INT_SECOND);
```

3.19. SPI

SPI/I2S模块可以通过SPI协议或I2S音频协议与外部设备进行通信。章节*3.19.1*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 3-394. 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	四线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 3-395. SPI/I2S 库函数

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

库函数名称	库函数描述
spi_dma_disable	失能外设SPIx的DMA功能
spi_i2s_data_frame_format_config	配置外设SPIx/I2Sx数据帧格式
spi_i2s_data_transmit	发送数据
spi_i2s_data_receive	接收数据
spi_bidirectional_transfer_config	配置外设SPIx的数据传输方向
spi_crc_polynomial_set	设置外设SPIx的CRC多项式值
spi_crc_polynomial_get	获取外设SPIx的CRC多项式值
spi_crc_on	打开外设SPIx的CRC功能
spi_crc_off	关闭外设SPIx的CRC功能
spi_crc_next	设置外设SPIx下一次传输数据为CRC值
spi_crc_get	外设SPIx获取CRC值
spi_ti_mode_enable	使能SPI TI模式
spi_ti_mode_disable	禁能SPI TI模式
spi_nssp_mode_enable	使能SPI NSS脉冲模式
spi_nssp_mode_disable	禁能SPI NSS脉冲模式
spi_quad_enable	使能四线SPI模式
spi_quad_disable	禁能四线SPI模式
spi_quad_write_enable	使能四线SPI写
spi_quad_read_enable	使能四线SPI读
spi_i2s_interrupt_enable	使能外设SPIx/I2Sx中断
spi_i2s_interrupt_disable	失能外设SPIx/I2Sx中断
spi_i2s_interrupt_flag_get	获取外设SPIx/I2Sx中断状态
spi_i2s_flag_get	获取外设SPIx/I2Sx标志状态
spi_crc_error_clear	清除SPIx CRC错误标志状态

结构体 spi_parameter_struct

表 3-396. 结构体 spi_parameter_struct

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

e	(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 3-397. 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=0,1,2
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 3-398. 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	
先决条件	-
被调用函数	-
输入参数{in}	
***spi_struct**	一个已经定义的spi_parameter_struct结构体变量地址
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 3-399. 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=0,1,2
输入参数{in}	
spi_struct	初始化结构体，结构体成员参考*表3-396. 结构体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 3-400. 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=0,1,2
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 3-401. 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=0,1,2
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 3-402. 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I2Sx
先决条件	-
被调用函数	-
输入参数{in}	
spi_periph	外设I2Sx
SPIx	x=1,2
输入参数{in}	
mode	I2S运行模式
I2S_MODE_SLAVETX	I2S从机发送模式
I2S_MODE_SLAVERX	I2S从机接收模式
I2S_MODE_MASTERTX	I2S主机发送模式
I2S_MODE_MASTERRX	I2S主机接收模式
输入参数{in}	
standard	I2S标准选择
I2S_STD_PHILIPS	I2S飞利浦标准
I2S_STD_MSB	I2S MSB对齐标准
I2S_STD_LSB	I2S LSB对齐标准
I2S_STD_PCMSHORT	I2S PCM短帧标准
I2S_STD_PCMLONG	I2S PCM长帧标准
输入参数{in}	
ckpl	I2S空闲状态时钟极性
I2S_CKPL_LOW	I2S_CK空闲状态为低电平
I2S_CKPL_HIGH	I2S_CK空闲状态为高电平
输出参数{out}	
-	-
返回值	
-	-

例如：

/* initialize I2S1 */

i2s_init(SPI1, I2S_MODE_MASTERTX, I2S_STD_PHILIPS, I2S_CKPL_LOW);

函数 i2s_psc_config

函数i2s_psc_config描述见下表：

表 3-403. 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I2Sx预分频器
先决条件	-
被调用函数	-
输入参数{in}	
spi_periph	外设I2Sx
SPIx	x=1,2
输入参数{in}	
audiosample	I2S音频采样频率
I2S_AUDIOSAMPLE_8K	音频采样频率为8KHz
I2S_AUDIOSAMPLE_11K	音频采样频率为11KHz
I2S_AUDIOSAMPLE_16K	音频采样频率为16KHz
I2S_AUDIOSAMPLE_22K	音频采样频率为22KHz
I2S_AUDIOSAMPLE_32K	音频采样频率为32KHz
I2S_AUDIOSAMPLE_44K	音频采样频率为44KHz
I2S_AUDIOSAMPLE_48K	音频采样频率为48KHz
I2S_AUDIOSAMPLE_96K	音频采样频率为96KHz
I2S_AUDIOSAMPLE_192K	音频采样频率为192KHz
输入参数{in}	
frameformat	I2S数据长度和通道长度
I2S_FRAMEFORMAT_DT16B_CH16B	I2S数据长度为16位，通道长度为16位

I2S_FRAMEFORMAT_DT16B_CH32B	I2S数据长度为16位，通道长度为32位
I2S_FRAMEFORMAT_DT24B_CH32B	I2S数据长度为24位，通道长度为32位
I2S_FRAMEFORMAT_DT32B_CH32B	I2S数据长度为32位，通道长度为32位
输入参数{in}	
mckout	2S_MCK输出使能
I2S_MCKOUT_ENABLE	I2S_MCK输出使能
I2S_MCKOUT_DISABLE	I2S_MCK输出禁止
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* configure I2S1 prescaler */
i2s_psc_config(SPI1, I2S_AUDIOSAMPLE_44K, I2S_FRAMEFORMAT_DT16B_CH16B, I2S_MCKOUT_DISABLE);
```

函数 i2s_enable

函数i2s_enable描述见下表：

表 3-404. 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I2Sx
先决条件	-
被调用函数	-
输入参数{in}	
spi_periph	外设SPI/I2Sx
SPIx	x=1,2
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* enable I2S1 */
i2s_enable(SPI1);
```

函数 i2s_disable

函数i2s_disable描述见下表：

表 3-405. 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I2Sx
先决条件	-
被调用函数	-
输入参数{in}	
spi_periph	外设SPI/I2Sx
SPIx	x=1,2
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* disable I2S1 */
i2s_disable(SPI1);
```

函数 spi_nss_output_enable

函数spi_nss_output_enable描述见下表：

表 3-406. 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=0,1,2
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 3-407. 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=0,1,2
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 3-408. 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=0,1,2
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 3-409. 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=0,1,2
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 3-410. 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=0,1,2
输入参数{in}	
dma	SPI DMA模式
SPI_DMA_TRANSMIT	SPI发送缓冲区DMA使能
SPI_DMA_RECEIVE	SPI接收缓冲区DMA使能
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* enable 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 3-411. 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=0,1,2
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 3-412. 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/I2Sx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=0,1,2
输入参数{in}	
frame_format	SPI帧大小
SPI_FRAMESIZE_16BIT	SPI 16位数据帧格式
SPI_FRAMESIZE_8BIT	SPI 8位数据帧格式
输出参数{out}	
-	-
返回值	
-	-

例如：

/* configure SPI1/I2S1 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 3-413. 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=0,1,2
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 3-414. 函数 spi_i2s_data_receive

函数名称	spi_i2s_data_receive
函数原形	uint16_t spi_i2s_data_receive(uint32_t spi_periph);
功能描述	SPI接收数据
先决条件	-
被调用函数	-
输入参数{in}	
spi_periph	外设SPIx
SPIx	x=0,1,2
输出参数{out}	
-	-
返回值	
-	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 3-415. 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
SPIx	x=0,1,2
输入参数{in}	
transfer_direction	SPI双向传输输出使能
SPI_BIDIRECTIONAL_TRANSMIT	SPI工作在只发送模式
SPI_BIDIRECTIONAL_RECEIVE	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 3-416. 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=0,1,2
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 3-417. 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=0,1,2
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 3-418. 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=0,1,2
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 3-419. 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=0,1,2
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 3-420. 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=0,1,2
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 3-421. 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=0,1,2
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 3-422. 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=0,1,2
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 3-423. 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=0,1,2
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 3-424. 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=0,1,2
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 3-425. 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=0,1,2
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* disable SPI0 NSS pulse mode */
spi_nssp_mode_disable(SPI0);
```

函数 spi_quad_enable

函数spi_quad_enable描述见下表：

表 3-426. 函数 spi_quad_enable

函数名称	spi_quad_enable
函数原形	void spi_quad_enable(uint32_t spi_periph);
功能描述	使能四线SPI模式
先决条件	-
被调用函数	-
输入参数{in}	
spi_periph	外设SPIx
SPIx	x=0
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* enable SPI0 quad wire mode */
spi_quad_enable(SPI0);
```

函数 spi_quad_disable

函数spi_quad_disable描述见下表：

表 3-427. 函数 spi_quad_disable

函数名称	spi_quad_disable
函数原形	spi_quad_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=0
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* disable SPI0 quad wire mode */
spi_quad_disable(SPI0);
```

函数 spi_quad_write_enable

函数spi_quad_write_enable描述见下表：

表 3-428. 函数 spi_quad_write_enable

函数名称	spi_quad_write_enable
函数原形	void spi_quad_write_enable(uint32_t spi_periph);
功能描述	使能四线SPI写
先决条件	-
被调用函数	-
输入参数{in}	
spi_periph	外设SPIx
SPIx	x=0
输出参数{out}	
-	-
返回值	
-	-

例如：

/* enable SPI0 quad wire write */

spi_quad_write_enable(SPI0);

函数 spi_quad_read_enable

函数spi_quad_read_enable描述见下表：

表 3-429. 函数 spi_quad_read_enable

函数名称	spi_quad_read_enable
函数原形	void spi_quad_read_enable(uint32_t spi_periph);
功能描述	使能四线SPI读
先决条件	-
被调用函数	-
输入参数{in}	
spi_periph	外设SPIx
SPIx	x=0
输出参数{out}	
-	-
返回值	
-	-

例如：

/* enable SPI0 quad wire read */

spi_quad_read_enable(SPI0);

函数 spi_i2s_interrupt_enable

函数spi_i2s_interrupt_enable描述见下表：

表 3-430. 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/I2Sx中断
先决条件	-
被调用函数	-
输入参数{in}	
spi_periph	外设SPIx
SPIx	x=0,1,2
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 3-431. 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/I2Sx中断
先决条件	-
被调用函数	-
输入参数{in}	
spi_periph	外设SPIx
SPIx	x=0,1,2
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 3-432. 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/I2Sx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=0,1,2
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_CONFERR	配置错误中断
SPI_INT_FLAG_CRCERR	CRC错误中断
I2S_INT_FLAG_TXURERR	发送欠载错误中断
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 3-433. 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/I2Sx标志状态
先决条件	-
被调用函数	-
输入参数{in}	
spi_periph	外设SPIx
SPIx	x=0,1,2
输入参数{in}	
flag	SPI/I2S标志状态
SPI_FLAG_TBE	发送缓冲区空标志
SPI_FLAG_RBNE	接收缓冲区非空标志
SPI_FLAG_TRANS	通信进行中标志
SPI_I2S_INT_FLAG_RXORERR	接收过载错误标志
SPI_FLAG_CONFERR	配置错误标志
SPI_FLAG_CRCERR	CRC错误标志
I2S_FLAG_RXORERR	接收过载错误标志
I2S_FLAG_TXURERR	发送欠载错误标志
I2S_FLAG_CH	通道标志
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 3-434. 函数 spi_crc_error_clear

函数名称	spi_crc_error_clear
函数原形	void spi_crc_error_clear(uint32_t spi_periph);
功能描述	清除SPIx CRC错误标志状态
先决条件	-
被调用函数	-
输入参数{in}	
spi_periph	外设SPIx
SPIx	x=0,1,2
输出参数{out}	
-	-
返回值	
-	-

例如：

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

3.20. 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(TIMERx, x=0, 7)，通用定时器L0(TIMERx, x=1, 2, 3, 4)，通用定时器L1(TIMERx, x=8, 11)，通用定时器L2(TIMERx, x=9, 10, 12, 13)，基本定时器(TIMERx, x=5, 6)，不同类型的定时器具体功能有所差别。章节*3.20.1*描述了TIMER的寄存器列表，章节*3.20.2*对TIMER库函数进行说明。

3.20.1. 外设寄存器说明

TIMER寄存器列表如下表所示：

表 3-435. TIMER 寄存器

寄存器名称	寄存器描述
TIMER_CTL0	控制寄存器0
TIMER_CTL1	控制寄存器1
TIMER_SMCFG	从模式配置寄存器
TIMER_DMAINTEN	DMA和中断使能寄存器
TIMER_INTF	中断标志寄存器

寄存器名称	寄存器描述
TIMER_SWEVG	软件事件产生寄存器
TIMER_CHCTL0	通道控制寄存器0
TIMER_CHCTL1	通道控制寄存器1
TIMER_CHCTL2	通道控制寄存器2
TIMER_CNT	计数器寄存器
TIMER_PSC	预分频寄存器
TIMER_CAR	计数器自动重载寄存器
TIMER_CREP	重复计数寄存器
TIMER_CH0CV	通道0捕获/比较寄存器
TIMER_CH1CV	通道1捕获/比较寄存器
TIMER_CH2CV	通道2捕获/比较寄存器
TIMER_CH3CV	通道3捕获/比较寄存器
TIMER_CCHP	互补通道保护寄存器
TIMER_DMACFG	DMA配置寄存器
TIMER_DMATB	DMA发送缓冲区寄存器
TIMER_CFG	配置寄存器

3.20.2. 外设库函数说明

TIMER库函数列表如下表所示：

表 3-436. TIMER 库函数

库函数名称	库函数描述
timer_deinit	复位外设TIMERx
timer_struct_para_init	将TIMER初始化结构体中所有参数初始化为默认值
timer_init	初始化外设TIMERx
timer_enable	使能外设TIMERx
timer_disable	禁能外设TIMERx
timer_auto_reload_shadow_enable	TIMERx自动重载影子使能
timer_auto_reload_shadow_disable	TIMERx自动重载影子禁能
timer_update_event_enable	TIMERx更新使能
timer_update_event_disable	TIMERx更新禁能
timer_counter_alignment	设置外设TIMERx的对齐模式
timer_counter_up_direction	设置外设TIMERx向上计数
timer_counter_down_direction	设置外设TIMERx向下计数
timer_prescaler_config	配置外设TIMERx预分频器
timer_repetition_value_config	配置外设TIMERx的重复计数器
timer_autoreload_value_config	配置外设TIMERx的自动重载寄存器
timer_counter_value_config	配置外设TIMERx的计数器值
timer_counter_read	读取外设TIMERx的计数器值
timer_prescaler_read	读取外设TIMERx的预分频器值

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_dma_enable	外设TIMERx的DMA使能
timer_dma_disable	外设TIMERx的DMA禁能
timer_channel_dma_request_source_select	外设TIMERx的通道DMA请求源选择
timer_dma_transfer_config	配置外设TIMERx的DMA模式
timer_event_software_generate	软件产生事件
timer_break_struct_para_init	将TIMER中止功能参数结构体中所有参数初始化为默认值
timer_break_config	配置中止功能
timer_break_enable	使能TIMERx的中止功能
timer_break_disable	禁能TIMERx的中止功能
timer_automatic_output_enable	自动输出使能
timer_automatic_output_disable	自动输出禁能
timer_primary_output_config	所有的通道输出使能
timer_channel_control_shadow_config	通道换相控制影子寄存器配置
timer_channel_control_shadow_update_config	通道换相控制影子寄存器更新控制
timer_channel_output_struct_para_init	将TIMER通道输出参数结构体中所有参数初始化为默认值
timer_channel_output_config	外设TIMERx的通道输出配置
timer_channel_output_mode_config	配置外设TIMERx通道输出比较模式
timer_channel_output_pulse_value_config	配置外设TIMERx的通道输出比较值
timer_channel_output_shadow_config	配置TIMERx通道输出比较影子寄存器功能
timer_channel_output_fast_config	配置TIMERx通道输出比较快速功能
timer_channel_output_clear_config	配置TIMERx的通道输出比较清0功能
timer_channel_output_polarity_config	通道输出极性配置
timer_channel_complementary_output_polarity_config	互补通道输出极性配置
timer_channel_output_state_config	配置通道状态
timer_channel_complementary_output_state_config	配置互补通道输出状态
timer_channel_input_struct_para_init	将TIMER通道输入参数结构体中所有参数初始化为默认值
timer_input_capture_config	配置TIMERx输入捕获参数
timer_channel_input_capture_prescaler_config	配置TIMERx通道输入捕获预分频值

库函数名称	库函数描述
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
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	TIMERx外部时钟模式1禁能
timer_write_chxval_register_config	配置TIMERx写CHxVAL选择位
timer_output_value_selection_config	配置TIMERx输出值选择位
timer_interrupt_enable	外设TIMERx中断使能
timer_interrupt_disable	外设TIMERx中断禁能
timer_interrupt_flag_get	获取外设TIMERx中断标志
timer_interrupt_flag_clear	清除外设TIMERx的中断标志
timer_flag_get	获取外设TIMERx的状态标志
timer_flag_clear	清除外设TIMERx状态标志

结构体 timer_parameter_struct

表 3-437. 结构体 timer_parameter_struct

成员名称	功能描述
prescaler	预分频值(0~65535)
alignedmode	对齐模式(TIMER_COUNTER_EDGE, TIMER_COUNTER_CENTER_DOWN, TIMER_COUNTER_CENTER_UP, TIMER_COUNTER_CENTER_BOTH)
counterdirection	计数方向(TIMER_COUNTER_UP, TIMER_COUNTER_DOWN)
period	周期(0~65535)
clockdivision	时钟分频因子(TIMER_CKDIV_DIV1, TIMER_CKDIV_DIV2,

成员名称	功能描述
	TIMER_CKDIV_DIV4)
repetitioncounter	重复计数器值(0~255)

结构体 timer_break_parameter_struct

表 3-438. 结构体 timer_break_parameter_struct

成员名称	功能描述
runoffstate	运行模式下"关闭状态"配置 (TIMER_ROS_STATE_ENABLE, TIMER_ROS_STATE_DISABLE)
ideloffstate	空闲模式下"关闭状态"配置(TIMER_IOS_STATE_ENABLE, TIMER_IOS_STATE_DISABLE)
deadtime	死区时间(0~255)
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 3-439. 结构体 timer_oc_parameter_struct

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

结构体 timer_ic_parameter_struct

表 3-440. 结构体 timer_ic_parameter_struct

成员名称	功能描述
icpolarity	通道输入极性(TIMER_IC_POLARITY_RISING, TIMER_IC_POLARITY_FALLING, TIMER_IC_POLARITY_BOTH_EDGE)
icselection	通道输入模式选择(TIMER_IC_SELECTION_DIRECTTI, TIMER_IC_SELECTION_INDIRECTTI, TIMER_IC_SELECTION_ITS)

成员名称	功能描述
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 3-441. 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..13)	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 3-442. 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3-437. 结构体 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 3-443. 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..13)	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3-437. 结构体 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 3-444. 函数 timer_enable

函数名称	timer_enable
函数原型	void timer_enable(uint32_t timer_periph);
功能描述	使能外设TIMERx
先决条件	-
被调用函数	-
输入参数{in}	
timer_periph	TIMER外设
TIMERx(x=0..13)	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 3-445. 函数 timer_disable

函数名称	timer_disable
函数原型	void timer_disable(uint32_t timer_periph);
功能描述	禁能外设TIMERx
先决条件	-
被调用函数	-
输入参数{in}	
timer_periph	TIMER外设
TIMERx(x=0..13)	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 3-446. 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..13)	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 3-447. 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..13)	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 3-448. 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更新使能
先决条件	-
被调用函数	-
输入参数{in}	
timer_periph	TIMER外设
TIMERx(x=0..13)	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 3-449. 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更新禁能
先决条件	-
被调用函数	-
输入参数{in}	
timer_periph	TIMER外设
TIMERx(x=0..13)	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 3-450. 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..4,7..13)	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 3-451. 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..4,7..13)	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 3-452. 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..4,7..13)	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 3-453. 函数 timer_prescaler_config

函数名称	timer_prescaler_config
函数原型	void timer_prescaler_config(uint32_t timer_periph, uint16_t prescaler, uint8_t pscreload);
功能描述	配置外设TIMERx预分频器
先决条件	-
被调用函数	-
输入参数{in}	
timer_periph	TIMER外设
TIMERx(x=0..13)	TIMER外设选择
输入参数{in}	
prescaler	预分频值，0~65535
输入参数{in}	
pscreload	预分频值加载模式
TIMER_PSC_RELOAD_NOW	预分频值立即加载
TIMER_PSC_RELOAD_UPDATE	预分频值在下次更新事件发生时加载
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* configure TIMER0 prescaler */
timer_prescaler_config (TIMER0, 3000, TIMER_PSC_RELOAD_NOW);
```

函数 timer_repetition_value_config

函数timer_repetition_value_config描述见下表：

表 3-454. 函数 timer_repetition_value_config

函数名称	timer_repetition_value_config
函数原型	void timer_repetition_value_config(uint32_t timer_periph, uint16_t repetition);
功能描述	配置外设TIMERx的重复计数器

先决条件	-
被调用函数	-
输入参数{in}	
timer_periph	TIMER外设
TIMERx(x=0,7)	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 3-455. 函数 timer_autoreload_value_config

函数名称	timer_autoreload_value_config
函数原型	void timer_autoreload_value_config(uint32_t timer_periph, uint16_t autoreload);
功能描述	配置外设TIMERx的自动重载寄存器
先决条件	-
被调用函数	-
输入参数{in}	
timer_periph	TIMER外设
TIMERx(x=0..13)	TIMER外设选择
输入参数{in}	
autoreload	计数器自动重载值(0-0xFFFF)
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 3-456. 函数 timer_counter_value_config

函数名称	timer_counter_value_config
函数原型	void timer_counter_value_config(uint32_t timer_periph, uint16_t counter);
功能描述	配置外设TIMERx的计数器值
先决条件	-
被调用函数	-
输入参数{in}	
timer_periph	TIMER外设
TIMERx(x=0..13)	TIMER外设选择
输入参数{in}	
counter	计数器值(0-0xFFFF)
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* configure TIMER0 counter register value */
timer_counter_value_config (TIMER0);
```

函数 timer_counter_read

函数timer_counter_read描述见下表：

表 3-457. 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..13)	TIMER外设选择
输出参数{out}	
-	-
返回值	
uint32_t	外设TIMERx的计数器值(0x0000~0xFFFF)

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 3-458. 函数 timer_prescaler_read

函数名称	timer_prescaler_read
函数原型	uint16_t timer_prescaler_read(uint32_t timer_periph);
功能描述	读取外设TIMERx的预分频器值
先决条件	-
被调用函数	-
输入参数{in}	
timer_periph	TIMER外设
TIMERx(x=0..13)	TIMER外设选择
输出参数{out}	
-	-
返回值	
uint16_t	外设TIMERx的预分频器值(0x0000~0xFFFF)

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 3-459. 函数 timer_single_pulse_mode_config

函数名称	timer_single_pulse_mode_config
函数原型	void timer_single_pulse_mode_config(uint32_t timer_periph, uint8_t spmode);
功能描述	配置外设TIMERx的单脉冲模式
先决条件	-
被调用函数	-
输入参数{in}	
timer_periph	TIMER外设
TIMERx(x=0..8,11)	TIMER外设选择
输入参数{in}	

spmode	脉冲模式
TIMER_SP_MODE_SINGLE	单脉冲模式计数
TIMER_SP_MODE_REPETITIVE	重复模式计数
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* configure TIMER0 single pulse mode */
timer_single_pulse_mode_config (TIMER0, TIMER_SP_MODE_SINGLE);
```

函数 timer_update_source_config

函数timer_update_source_config描述见下表：

表 3-460. 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..13)	TIMER外设选择
输入参数{in}	
update	更新源
TIMER_UPDATE_SRC_GLOBAL	下述任一事件产生更新中断或DMA请求： - UPG位被置1 - 计数器溢出/下溢 - 从模式控制器产生的更新
TIMER_UPDATE_SRC_REGULAR	只有计数器溢出/ 下溢才产生更新中断或DMA请求
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* configure TIMER update only by counter overflow/underflow */
```

timer_update_source_config (TIMER0, TIMER_UPDATE_SRC_REGULAR);

函数 timer_dma_enable

函数timer_dma_enable描述见下表：

表 3-461. 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..7)
TIMER_DMA_CH0D	通道0比较/捕获 DMA请求，TIMERx(x=0..4,7)
TIMER_DMA_CH1D	通道1比较/捕获 DMA请求，TIMERx(x=0..4,7)
TIMER_DMA_CH2D	通道2比较/捕获 DMA请求，TIMERx(x=0..4,7)
TIMER_DMA_CH3D	通道3比较/捕获 DMA请求，TIMERx(x=0..4,7)
TIMER_DMA_CMTD	换相DMA更新请求，TIMERx(x=0,7)
TIMER_DMA_TRGD	触发DMA请求使能，TIMERx(x=0..4,7)
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 3-462. 函数 timer_dma_disable

函数名称	timer_dma_disable
函数原型	void timer_dma_disable (uint32_t timer_periph, uint16_t dma);
功能描述	外设TIMERx的DMA禁能
先决条件	-
被调用函数	-
输入参数{in}	
timer_periph	TIMER外设
TIMERx	参考具体参数
输入参数{in}	
dma	DMA源
TIMER_DMA_UPD	更新DMA请求，TIMERx(x=0..7)
TIMER_DMA_CH0D	通道0比较/捕获 DMA请求，TIMERx(x=0..4,7)
TIMER_DMA_CH1D	通道1比较/捕获 DMA请求，TIMERx(x=0..4,7)
TIMER_DMA_CH2D	通道2比较/捕获 DMA请求，TIMERx(x=0..4,7)
TIMER_DMA_CH3D	通道3比较/捕获 DMA请求，TIMERx(x=0..4,7)
TIMER_DMA_CMTD	换相DMA更新请求，TIMERx(x=0,7)
TIMER_DMA_TRGD	触发DMA请求使能，TIMERx(x=0..4,7)
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* disablethe 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 3-463. 函数 timer_channel_dma_request_source_select

函数名称	timer_channel_dma_request_source_select
函数原型	void timer_channel_dma_request_source_select(uint32_t timer_periph, uint32_t dma_request);
功能描述	外设TIMERx的通道DMA请求源选择

先决条件	-
被调用函数	-
输入参数{in}	
timer_periph	TIMER外设
TIMERx(x=0..4,7)	TIMER外设选择
输入参数{in}	
dma_request	通道的DMA请求源选择
TIMER_DMAREQUEST_CHANNELEVENT	当通道捕获/比较事件发生时，发送通道n的DMA请求
TIMER_DMAREQUEST_UPDATEEVENT	当更新事件发生，发送通道n的DMA请求
输出参数{out}	
-	-
返回值	
-	-

例如：

/* 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 3-464. 函数 timer_dma_transfer_config

函数名称	timer_dma_transfer_config
函数原型	void timer_dma_transfer_config(uint32_t timer_periph, uint32_t dma_baseaddr, uint32_t dma_lenth);
功能描述	配置外设TIMERx的DMA模式
先决条件	-
被调用函数	-
输入参数{in}	
timer_periph	TIMER外设
TIMERx	参考具体参数
输入参数{in}	
dma_baseaddr	DMA传输起始地址
TIMER_DMACFG_DMATA_CTL0	DMA传输起始地址：TIMER_DMACFG_DMATA_CTL0，TIMERx(x=0..4,7)
*TIMER_DMACFG_*	DMA传输起始地址：TIMER_DMACFG_DMATA_CTL1，TIMERx(x=0..4,7)

DMATA_CTL1	
TIMER_DMACFG_DMATA_SMCFG	DMA传输起始地址：TIMER_DMACFG_DMATA_SMCFG，TIMERx(x=0..4,7)
TIMER_DMACFG_DMATA_DMAINTEN	DMA传输起始地址：TIMER_DMACFG_DMATA_DMAINTEN，TIMERx(x=0..4,7)
TIMER_DMACFG_DMATA_INTF	DMA传输起始地址：TIMER_DMACFG_DMATA_INTF，TIMERx(x=0..4,7)
TIMER_DMACFG_DMATA_SWEVG	DMA传输起始地址：TIMER_DMACFG_DMATA_SWEVG，TIMERx(x=0..4,7)
TIMER_DMACFG_DMATA_CHCTL0	DMA传输起始地址：TIMER_DMACFG_DMATA_CHCTL0，TIMERx(x=0..4,7)
TIMER_DMACFG_DMATA_CHCTL1	DMA传输起始地址：TIMER_DMACFG_DMATA_CHCTL1，TIMERx(x=0..4,7)
TIMER_DMACFG_DMATA_CHCTL2	DMA传输起始地址：TIMER_DMACFG_DMATA_CHCTL2，TIMERx(x=0..4,7)
TIMER_DMACFG_DMATA_CNT	DMA传输起始地址：TIMER_DMACFG_DMATA_CNT，TIMERx(x=0..4,7)
TIMER_DMACFG_DMATA_PSC	DMA传输起始地址：TIMER_DMACFG_DMATA_PSC，TIMERx(x=0..4,7)
TIMER_DMACFG_DMATA_CAR	DMA传输起始地址：TIMER_DMACFG_DMATA_CAR，TIMERx(x=0..4,7)
TIMER_DMACFG_DMATA_CREP	DMA传输起始地址：TIMER_DMACFG_DMATA_CREP，TIMERx(x=0,7)
TIMER_DMACFG_DMATA_CH0CV	DMA传输起始地址：TIMER_DMACFG_DMATA_CH0CV，TIMERx(x=0..4,7)
TIMER_DMACFG_DMATA_CH1CV	DMA传输起始地址：TIMER_DMACFG_DMATA_CH1CV，TIMERx(x=0..4,7)
TIMER_DMACFG_DMATA_CH2CV	DMA传输起始地址：TIMER_DMACFG_DMATA_CH2CV，TIMERx(x=0..4,7)
TIMER_DMACFG_DMATA_CH3CV	DMA传输起始地址：TIMER_DMACFG_DMATA_CH3CV，TIMERx(x=0..4,7)
TIMER_DMACFG_DMATA_CCHP	DMA传输起始地址：TIMER_DMACFG_DMATA_CCHP，TIMERx(x=0,7)
TIMER_DMACFG_DMATA_DMACFG	DMA传输起始地址：TIMER_DMACFG_DMATA_DMACFG，TIMERx(x=0..4,7)
TIMER_DMACFG_DMATA_DMATB	DMA传输起始地址：TIMER_DMACFG_DMATA_DMATB，TIMERx(x=0..4,7)
输入参数{in}	
dma_lenth	DMA传输长度
TIMER_DMACFG_DMATC_xTRANSFER	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 3-465. 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_SRC_UPG	更新事件产生，TIMERx(x=0..13)
TIMER_EVENT_SRC_CH0G	通道0捕获或比较事件发生，TIMERx(x=0..4,7..13)
TIMER_EVENT_SRC_CH1G	通道1捕获或比较事件发生，TIMERx(x=0..4,7,8,11)
TIMER_EVENT_SRC_CH2G	通道2捕获或比较事件发生，TIMERx(x=0..4,7)
TIMER_EVENT_SRC_CH3G	通道3捕获或比较事件发生，TIMERx(x=0..4,7)
TIMER_EVENT_SRC_CMTG	通道换相更新事件发生，TIMERx(x=0,7)
TIMER_EVENT_SRC_TRGG	触发事件产生，TIMERx(x=0..4,7,8,11)
TIMER_EVENT_SRC_BRKG	产生中止事件，TIMERx(x=0,7)
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 3-466. 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3-438. 结构体 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 3-467. 函数 timer_break_config

函数名称	timer_break_config
函数原型	void timer_break_config(uint32_t timer_periph, timer_break_parameter_struct* breakpara);
功能描述	配置中止功能
先决条件	-
被调用函数	-

输入参数{in}	
timer_periph	TIMER外设
TIMERx(x=0,7)	TIMER外设选择
输入参数{in}	
breakpara	中止功能配置结构体，详见*表3-438. 结构体 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 3-468. 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=0,7)	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 3-469. 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=0,7)	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 3-470. 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=0,7)	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 3-471. 函数 timer_automatic_output_disable

函数名称	timer_automatic_output_ disable
函数原型	void timer_automatic_output_ disable (uint32_t timer_periph);
功能描述	自动输出禁能
先决条件	只有在TIMERx_CCHP寄存器的PROT [1:0] = 00时，才可修改
被调用函数	-
输入参数{in}	
timer_periph	TIMER外设
TIMERx(x=0,7)	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 3-472. 函数 timer_primary_output_config

函数名称	timer_primary_output_config
函数原型	void timer_primary_output_config(uint32_t timer_periph, ControlStatus newvalue);
功能描述	所有的通道输出使能
先决条件	-
被调用函数	-

输入参数{in}	
timer_periph	TIMER外设
TIMERx(x=0,7)	TIMER外设选择
输入参数{in}	
newvalue	控制状态
ENABLE	使能
DISABLE	禁能
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* enable TIMER0 primary output function */
timer_primary_output_config (TIMER0, ENABLE);
```

函数 timer_channel_control_shadow_config

函数timer_channel_control_shadow_config描述见下表：

表 3-473. 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=0,7)	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 3-474. 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=0,7)	TIMER外设选择
输入参数{in}	
ccuctl	通道换相控制影子寄存器更新控制
TIMER_UPDATECTL_CCU	CMTG位被置1时更新影子寄存器
TIMER_UPDATECTL_CCUTRI	当CMTG位被置1或检测到TRIGI上升沿时，影子寄存器更新
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 3-475. 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3-439. 结构体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 3-476. 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..4,7..13)
TIMER_CH_1	通道1，TIMERx(x=0..4,7,8,11)
TIMER_CH_2	通道2，TIMERx(x=0..4,7)
TIMER_CH_3	通道3，TIMERx(x=0..4,7)
输入参数{in}	
ocpara	输出通道结构体，详见*表3-439. 结构体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 3-477. 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..4,7..13)
TIMER_CH_1	通道1，TIMERx (x=0..4,7,8,11)
TIMER_CH_2	通道2，TIMERx (x=0..4,7)
TIMER_CH_3	通道3，TIMERx (x=0..4,7)
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	强制为高

_HIGH	
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 3-478. 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..4,7..13)
TIMER_CH_1	通道1，TIMERx (x=0..4,7,8,11)
TIMER_CH_2	通道2，TIMERx (x=0..4,7)
TIMER_CH_3	通道3，TIMERx (x=0..4,7)
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 3-479. 函数 timer_channel_output_shadow_config

函数名称	timer_channel_output_shadow_config
函数原型	void timer_channel_output_shadow_config(uint32_t timer_periph, uint16_t channel, uint16_t ocshadow);
功能描述	配置TIMERx通道输出比较影子寄存器功能
先决条件	-
被调用函数	-
输入参数{in}	
timer_periph	TIMER外设
TIMERx	参考具体参数
输入参数{in}	
channel	待配置通道
TIMER_CH_0	通道0，TIMERx (x=0..4,7..13)
TIMER_CH_1	通道1，TIMERx (x=0..4,7,8,11)
TIMER_CH_2	通道2，TIMERx (x=0..4,7)
TIMER_CH_3	通道3，TIMERx (x=0..4,7)
输入参数{in}	
ocshadow	输出比较影子寄存器功能状态
TIMER_OC_SHADOW_ENABLE	使能输出比较影子寄存器
TIMER_OC_SHADOW_DISABLE	禁能输出比较影子寄存器
输出参数{out}	
-	-
返回值	
-	-

例如：

/*configure 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 3-480. 函数 timer_channel_output_fast_config

函数名称	timer_channel_output_fast_config
函数原型	void timer_channel_output_fast_config(uint32_t timer_periph, uint16_t channel, uint16_t ocfast);
功能描述	配置TIMERx通道输出比较快速功能
先决条件	-
被调用函数	-
输入参数{in}	
timer_periph	TIMER外设
TIMERx	参考具体参数
输入参数{in}	
channel	待配置通道
TIMER_CH_0	通道0，TIMERx (x=0..4,7..13)
TIMER_CH_1	通道1，TIMERx (x=0..4,7,8,11)
TIMER_CH_2	通道2，TIMERx (x=0..4,7)
TIMER_CH_3	通道3，TIMERx (x=0..4,7)
输入参数{in}	
ocfast	通道输出比较快速功能状态
TIMER_OC_FAST_ENABLE	通道输出比较快速功能使能
TIMER_OC_FAST_DISABLE	通道输出比较快速功能禁能
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* configure TIMER0 channel 0 output fast function */
timer_channel_output_fast_config (TIMER0, TIMER_CH_0, TIMER_OC_FAST_ENABLE);
```

函数 timer_channel_output_clear_config

函数timer_channel_output_clear_config描述见下表：

表 3-481. 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}	
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..4,7..13)
TIMER_CH_1	通道1，TIMERx (x=0..4,7,8,11)
TIMER_CH_2	通道2，TIMERx (x=0..4,7)
TIMER_CH_3	通道3，TIMERx (x=0..4,7)
输入参数{in}	
occlear	通道比较输出清0功能状态
TIMER_OC_CLEAR_ENABLE	通道比较输出清0功能使能
TIMER_OC_CLEAR_DISABLE	通道比较输出清0功能禁能
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* configure TIMER0 channel 0 output clear function */
timer_channel_output_clear_config (TIMER0, TIMER_CH_0,
TIMER_OC_CLEAR_ENABLE);
```

函数 timer_channel_output_polarity_config

函数timer_channel_output_polarity_config描述见下表：

表 3-482. 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..4,7..13)
TIMER_CH_1	通道1，TIMERx (x=0..4,7,8,11)
TIMER_CH_2	通道2，TIMERx (x=0..4,7)
TIMER_CH_3	通道3，TIMERx (x=0..4,7)
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 3-483. 函数 timer_channel_complementary_output_polarity_config

函数名称	timer_channel_complementary_output_polarity_config
函数原型	void timer_channel_complementary_output_polarity_config(uint32_t timer_periph, uint16_t channel, uint16_t ocnpolarity);
功能描述	互补通道输出极性配置
先决条件	-
被调用函数	-
输入参数{in}	
timer_periph	TIMER外设
TIMERx(x=0,7)	TIMER外设选择
输入参数{in}	
channel	待配置通道
TIMER_CH_0	通道0
TIMER_CH_1	通道1
TIMER_CH_2	通道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 3-484. 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..4,7..13)
TIMER_CH_1	通道1，TIMERx (x=0..4,7,8,11)
TIMER_CH_2	通道2，TIMERx (x=0..4,7)
TIMER_CH_3	通道3，TIMERx (x=0..4,7)
输入参数{in}	
state	通道状态
TIMER_CCX_ENABLE	通道使能
TIMER_CCX_DISABLE	通道禁能

输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* configure TIMER0 channel 0 enable state */
timer_channel_output_state_config (TIMER0, TIMER_CH_0, TIMER_CCX_ENABLE);
```

函数 timer_channel_complementary_output_state_config

函数timer_channel_complementary_output_state_config描述见下表：

表 3-485. 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(x=0,7)	TIMER外设选择
输入参数{in}	
channel	待配置通道
TIMER_CH_0	通道0
TIMER_CH_1	通道1
TIMER_CH_2	通道2
输入参数{in}	
state	互补通道状态
TIMER_CCXN_ENABLE	互补通道使能
TIMER_CCXN_DISABLE	互补通道禁能
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* configure TIMER0 channel 0 complementary output enable state */
timer_channel_complementary_output_state_config (TIMER0, TIMER_CH_0,
```

TIMER_CCXN_ENABLE);

函数 timer_channel_input_struct_para_init

函数timer_channel_input_struct_para_init描述见下表：

表 3-486. 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3-440. 结构体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 3-487. 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..4,7..13)
TIMER_CH_1	通道1，TIMERx (x=0..4,7,8,11)

TIMER_CH_2	通道2，TIMERx (x=0..4,7)
TIMER_CH_3	通道3，TIMERx (x=0..4,7)
输入参数{in}	
icpara	输入捕获结构体，详见*表3-440. 结构体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 3-488. 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..4,7..13)
TIMER_CH_1	通道1，TIMERx (x=0..4,7,8,11)
TIMER_CH_2	通道2，TIMERx (x=0..4,7)
TIMER_CH_3	通道3，TIMERx (x=0..4,7)
输入参数{in}	
prescaler	通道输入捕获预分频值

TIMER_IC_PSC_DIV1	不分频
TIMER_IC_PSC_DIV2	2分频
TIMER_IC_PSC_DIV4	4分频
TIMER_IC_PSC_DIV8	8分频
输出参数{out}	
-	-
返回值	
-	-

例如：

/* configure TIMER0 channel 0 input capture prescaler value */

timer_channel_input_capture_prescaler_config (TIMER0, TIMER_CH_0, TIMER_IC_PSC_DIV2);

函数 timer_channel_capture_value_register_read

函数timer_channel_capture_value_register_read描述见下表：

表 3-489. 函数 timer_channel_capture_value_register_read

函数名称	timer_channel_capture_value_register_read
函数原型	uint32_t timer_channel_capture_value_register_read(uint32_t timer_periph, uint16_t channel);
功能描述	读取通道捕获值
先决条件	-
被调用函数	-
输入参数{in}	
timer_periph	TIMER外设
TIMERx	参考具体参数
输入参数{in}	
channel	待配置通道
TIMER_CH_0	通道0，TIMERx (x=0..4,7..13)
TIMER_CH_1	通道1，TIMERx (x=0..4,7,8,11)
TIMER_CH_2	通道2，TIMERx (x=0..4,7)
TIMER_CH_3	通道3，TIMERx (x=0..4,7)
输出参数{out}	
-	-
返回值	
uint32_t	通道输入捕获值，(0x0000~0xFFFF)

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 3-490. 函数 timer_input_pwm_capture_config

函数名称	timer_input_pwm_capture_config
函数原型	void timer_input_pwm_capture_config(uint32_t timer_periph, uint16_t channel, timer_ic_parameter_struct* icpwm);
功能描述	配置TIMERx捕获PWM输入参数
先决条件	-
被调用函数	timer_channel_input_capture_prescaler_config
输入参数{in}	
timer_periph	TIMER外设
TIMERx(x=0..4,7,8, 11)	TIMER外设选择
输入参数{in}	
channel	待配置通道
TIMER_CH_0	通道0
TIMER_CH_1	通道1
输入参数{in}	
icpwm	输入捕获结构体，详见*表3-440. 结构体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 3-491. 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..4,7)	TIMER外设选择
输入参数{in}	
hallmode	HALL接口功能状态
TIMER_HALLINTERFACE_ENABLE	HALL接口使能
TIMER_HALLINTERFACE_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 3-492. 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..4,7,8,11)	TIMER外设选择
输入参数{in}	
intrigger	待选择的触发源
TIMER_SMCFG_TRGSEL_ITI0	内部触发输入0(ITI0，TIMERx(x=0..4,7,8,11))
TIMER_SMCFG_TRGSEL_ITI1	内部触发输入1(ITI1，TIMERx(x=0..4,7,8,11))
TIMER_SMCFG_TRGSEL_ITI2	内部触发输入2(ITI2，TIMERx(x=0..4,7,8,11))
TIMER_SMCFG_TRGSEL_ITI3	内部触发输入3(ITI3，TIMERx(x=0..4,7,8,11))
TIMER_SMCFG_TRGSEL_CI0F_ED	CI0的边沿标志位（CI0F_ED，TIMERx(x=0..4,7,8,11))
TIMER_SMCFG_TRGSEL_CI0FE0	滤波后的通道0输入（CI0FE0，TIMERx(x=0..4,7,8,11))
TIMER_SMCFG_TRGSEL_CI1FE1	滤波后的通道1输入(CI1FE1，TIMERx(x=0..4,7,8,11))
TIMER_SMCFG_TRGSEL_ETIFP	滤波后的外部触发输入(ETIFP，TIMERx(x=0..4,7))
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 3-493. 函数 timer_master_output_trigger_source_select

函数名称	timer_master_output_trigger_source_select
函数原型	void timer_master_output_trigger_source_select(uint32_t timer_periph, uint32_t outrigger);
功能描述	选择TIMERx主模式输出触发
先决条件	-
被调用函数	-
输入参数{in}	

timer_periph	TIMER外设
TIMERx(x=0..7)	TIMER外设选择
输入参数{in}	
outrigger	主模式输出触发
TIMER_TRI_OUT_SRC_RESET	复位。TIMERx_SWEVG寄存器的UPG位被置1或从模式控制器产生复位触发一次TRGO脉冲，后一种情况下，TRGO上的信号相对实际的复位会有一个延迟。
TIMER_TRI_OUT_SRC_ENABLE	使能。此模式可用于同时启动多个定时器或控制在一段时间内使能从定时器。主模式控制器选择计数器使能信号作为触发输出TRGO。当CEN控制位被置1或者暂停模式下触发输入为高电平时，计数器使能信号被置1。在暂停模式下，计数器使能信号受控于触发输入，在触发输入和TRGO上会有一个延迟，除非选择了主/ 从模式。
TIMER_TRI_OUT_SRC_UPDATE	更新。主模式控制器选择更新事件作为TRGO。
TIMER_TRI_OUT_SRC_CC0	捕获/比较脉冲.通道0在发生一次捕获或一次比较成功时，主模式控制器产生一个TRGO脉冲
TIMER_TRI_OUT_SRC_O0CPRE	比较。在这种模式下主模式控制器选择O0CPRE信号被用于作为触发输出TRGO
TIMER_TRI_OUT_SRC_O1CPRE	比较。在这种模式下主模式控制器选择O1CPRE信号被用于作为触发输出TRGO
TIMER_TRI_OUT_SRC_O2CPRE	比较。在这种模式下主模式控制器选择O2CPRE信号被用于作为触发输出TRGO
TIMER_TRI_OUT_SRC_O3CPRE	比较。在这种模式下主模式控制器选择O3CPRE信号被用于作为触发输出TRGO
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* select TIMER0 master mode output trigger source */
timer_master_output_trigger_source_select (TIMER0, TIMER_TRI_OUT_SRC_RESET);
```

函数 timer_slave_mode_select

函数timer_slave_mode_select描述见下表：

表 3-494. 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..4,7,8,11*)*	TIMER外设选择
输入参数{in}	
slavemode	从模式
TIMER_SLAVE_MODE_DISABLE	关闭从模式
TIMER_QUAD_DECODER_MODE0	正交译码器模式0
TIMER_QUAD_DECODER_MODE1	正交译码器模式1
TIMER_QUAD_DECODER_MODE2	正交译码器模式2
TIMER_SLAVE_MODE_RESTART	复位模式
TIMER_SLAVE_MODE_PAUSE	暂停模式
TIMER_SLAVE_MODE_EVENT	事件模式
TIMER_SLAVE_MODE_EXTERNAL0	外部时钟模式0
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* select TIMER0 slave mode */
timer_slave_mode_select (TIMER0, TIMER_QUAD_DECODER_MODE0);
```

函数 timer_master_slave_mode_config

函数timer_master_slave_mode_config描述见下表：

表 3-495. 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..4,7,8,11*)*	TIMER外设选择
输入参数{in}	
masterslave	主从模式使能状态
TIMER_MASTER_SLAVE_MODE_ENABLE	主从模式使能
TIMER_MASTER_SLAVE_MODE_DISABLE	主从模式禁能
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* configure TIMER0 master slave mode  */
timer_master_slave_mode_config (TIMER0, TIMER_MASTER_SLAVE_MODE_ENABLE);
```

函数 timer_external_trigger_config

函数timer_external_trigger_config描述见下表：

表 3-496. 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..4,7*)*	TIMER外设选择
输入参数{in}	
extprescaler	外部触发预分频
TIMER_EXT_TRI_PSC_OFF	不分频
TIMER_EXT_TRI_PSC_DIV2	2分频
TIMER_EXT_TRI_P	4分频

SC_DIV4	
TIMER_EXT_TRI_PSC_DIV8	8分频
输入参数{in}	
expolarity	外部触发输入极性
TIMER_ETP_FALLING	低电平或者下降沿有效
TIMER_ETP_RISING	高电平或者上升沿有效
输入参数{in}	
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 3-497. 函数 timer_quadrature_decoder_mode_config

函数名称	timer_quadrature_decoder_mode_config
函数原型	void timer_quadrature_decoder_mode_config(uint32_t timer_periph, uint32_t decomode,uint16_t ic0polarity, uint16_t ic1polarity);
功能描述	TIMERx配置为编码器模式
先决条件	-
被调用函数	-
输入参数{in}	
timer_periph	TIMER外设
TIMERx(x=0..4,7)	TIMER外设选择
输入参数{in}	
decomode	编码器模式
TIMER_QUAD_DECODER_MODE0	根据CI0FE0的电平，计数器在CI1FE1的边沿向上/下计数
TIMER_QUAD_DECODER_MODE1	根据CI1FE1的电平，计数器在CI0FE0的边沿向上/下计数
TIMER_QUAD_DE	根据另一个信号的输入电平，计数器在CI0FE0和CI1FE1的

CODER_MODE2	边沿向上/ 下计数
输入参数{in}	
ic0polarity	IC0极性
TIMER_IC_POLARITY_RISING	捕获上升边沿
TIMER_IC_POLARITY_FALLING	捕获下降边沿
输入参数{in}	
ic1polarity	IC1极性
TIMER_IC_POLARITY_RISING	捕获上升边沿
TIMER_IC_POLARITY_FALLING	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_QUAD_DECODER_MODE0,
TIMER_IC_POLARITY_RISING, TIMER_IC_POLARITY_RISING);
```

函数 timer_internal_clock_config

函数timer_internal_clock_config描述见下表：

表 3-498. 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}	
timer_periph	TIMER外设
TIMERx(x=0..4,7,8,11)	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 3-499. 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..4,7,8, 11*)*	TIMER外设选择
输入参数{in}	
intrigger	被选择的内部触发源
TIMER_SMCFG_TRGSEL_ITI0	选择内部触发0 (ITI0)为时钟源
TIMER_SMCFG_TRGSEL_ITI1	选择内部触发1 (ITI1)为时钟源
TIMER_SMCFG_TRGSEL_ITI2	选择内部触发2 (ITI2)为时钟源
TIMER_SMCFG_TRGSEL_ITI3	选择内部触发3 (ITI3)为时钟源
输出参数{out}	
-	-
返回值	
-	-

例如：

/* configure TIMER0 the internal trigger ITI0 as external clock input */

timer_internal_trigger_as_external_clock_config (TIMER0, TIMER_SMCFG_TRGSEL_ITI0);

函数 timer_external_trigger_as_external_clock_config

函数timer_external_trigger_as_external_clock_config描述见下表：

表 3-500. 函数 timer_external_trigger_as_external_clock_config

函数名称	timer_external_trigger_as_external_clock_config
函数原型	void timer_external_trigger_as_external_clock_config(uint32_t timer_periph, uint32_t extrigger, uint16_t expolarity, uint32_t extfilter);
功能描述	配置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..4,7,8, 11*)*	TIMER外设选择
输入参数{in}	
extrigger	外部触发源
TIMER_SMCFG_TRGSEL_CI0F_ED	CI0的边沿标志(CI0F_ED)
TIMER_SMCFG_TRGSEL_CI0FE0	滤波后的通道0输入(CI0FE0)
TIMER_SMCFG_TRGSEL_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
输入参数{in}	
extfilter	滤波参数(0~15)
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* configure TIMER0 the external trigger CI0FE0 as external clock input */
timer_external_trigger_as_external_clock_config (TIMER0,
TIMER_SMCFG_TRGSEL_CI0FE0, TIMER_IC_POLARITY_RISING, 0);
```

函数 timer_external_clock_mode0_config

函数timer_external_clock_mode0_config描述见下表：

表 3-501. 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..4,7,8,11*)*	TIMER外设选择
输入参数{in}	
extprescaler	ETI触发源预分频值
TIMER_EXT_TRI_PSC_OFF	不分频
TIMER_EXT_TRI_PSC_DIV2	2分频
TIMER_EXT_TRI_PSC_DIV4	4分频
TIMER_EXT_TRI_PSC_DIV8	8分频
输入参数{in}	
expolarity	ETI触发源极性
TIMER_ETP_FALLING	下降沿或者低电平有效
TIMER_ETP_RISING	上升沿或者高电平有效
输入参数{in}	
extfilter	ETI触发源滤波参数(0~15)
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* configure 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 3-502. 函数 timer_external_clock_mode1_config

函数名称	timer_external_clock_mode1_config
函数原型	void timer_external_clock_mode1_config(uint32_t timer_periph, uint32_t extprescaler, uint32_t expolarity, uint32_t extfilter);
功能描述	配置TIMERx外部时钟模式1
先决条件	-
被调用函数	timer_external_trigger_config
输入参数{in}	
timer_periph	TIMER外设
TIMERx(x=0..4,7)	TIMER外设选择
输入参数{in}	
extprescaler	ETI触发源预分频值
TIMER_EXT_TRI_PSC_OFF	不分频
TIMER_EXT_TRI_PSC_DIV2	2分频
TIMER_EXT_TRI_PSC_DIV4	4分频
TIMER_EXT_TRI_PSC_DIV8	8分频
输入参数{in}	
expolarity	ETI触发源极性
TIMER_ETP_FALLING	下降沿或者低电平有效
TIMER_ETP_RISING	上升沿或者高电平有效
输入参数{in}	
extfilter	ETI触发源滤波参数(0~15)
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* configure TIMER0 the external clock mode1 */
timer_external_clock_mode1_config (TIMER0, TIMER_EXT_TRI_PSC_DIV2, TIMER_ETP_FALLING, 0);
```

函数 timer_external_clock_mode1_disable

函数timer_external_clock_mode1_disable描述见下表：

表 3-503. 函数 timer_external_clock_mode1_disable

函数名称	timer_external_clock_mode1_disable
函数原型	void timer_external_clock_mode1_disable(uint32_t timer_periph);
功能描述	TIMERx外部时钟模式1禁能
先决条件	-
被调用函数	-
输入参数{in}	
timer_periph	TIMER外设
TIMERx(x=0..4,7)	TIMER外设选择
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* disable TIMER0 the external clock mode1 */
timer_external_clock_mode1_disable (TIMER0);
```

函数 timer_write_chxval_register_config

函数timer_write_chxval_register_config描述见下表：

表 3-504. 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..4,7..13)	TIMER外设选择
输入参数{in}	
ccsel	写CHxVAL寄存器选择位
TIMER_CHVSEL_DISABLE	无影响
TIMER_CHVSEL_ENABLE	当写入捕获比较寄存器的值与寄存器当前值相等时，写入操作无效。
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* configure TIMER0 write CHxVAL register selection */
timer_write_chxval_register_config(TIMER0, TIMER_CHVSEL_ENABLE);
```

函数 timer_output_value_selection_config

函数timer_output_value_selection_config描述见下表：

表 3-505. 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=0,7)	TIMER外设选择
输入参数{in}	
ccsel	输出值选择位
TIMER_OUTSEL_DISABLE	无影响
TIMER_OUTSEL_ENABLE	如果POEN位与IOS位均为0，则输出无效。
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* configure TIMER output value selection */
timer_output_value_selection_config(TIMER0, TIMER_OUTSEL_ENABLE);
```

函数 timer_interrupt_enable

函数timer_interrupt_enable描述见下表：

表 3-506. 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..13)
TIMER_INT_CH0	通道0比较/捕获中断，TIMERx(x=0..4,7..13)
TIMER_INT_CH1	通道1比较/捕获中断，TIMERx(x=0..4,7,8,11)
TIMER_INT_CH2	通道2比较/捕获中断，TIMERx(x=0..4,7)
TIMER_INT_CH3	通道3比较/捕获中断，TIMERx(x=0..4,7)
TIMER_INT_CMT	换相更新中断，TIMERx(x=0,7)
TIMER_INT_TRG	触发中断，TIMERx(x=0..4,7,8,11)
TIMER_INT_BRK	中止中断，TIMERx(x=0,7)
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 3-507. 函数 timer_interrupt_disable

函数名称	timer_interrupt_ disable
函数原型	void timer_interrupt_ disable (uint32_t timer_periph, uint32_t interrupt);
功能描述	外设TIMERx中断禁能
先决条件	-
被调用函数	-
输入参数{in}	
timer_periph	TIMER外设
TIMERx	参考具体参数
输入参数{in}	
interrupt	中断源
TIMER_INT_UP	更新中断，TIMERx(x=0..13)
TIMER_INT_CH0	通道0比较/捕获中断，TIMERx(x=0..4,7..13)
TIMER_INT_CH1	通道1比较/捕获中断，TIMERx(x=0..4,7,8,11)
TIMER_INT_CH2	通道2比较/捕获中断，TIMERx(x=0..4,7)

TIMER_INT_CH3	通道3比较/捕获中断，TIMERx(x=0..4,7)
TIMER_INT_CMT	换相更新中断，TIMERx(x=0,7)
TIMER_INT_TRG	触发中断，TIMERx(x=0..4,7,8,11)
TIMER_INT_BRK	中止中断，TIMERx(x=0,7)
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 3-508. 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..13)
TIMER_INT_FLAG_CH0	通道0比较/捕获中断，TIMERx(x=0..4,7..13)
TIMER_INT_FLAG_CH1	通道1比较/捕获中断，TIMERx(x=0..4,7,8,11)
TIMER_INT_FLAG_CH2	通道2比较/捕获中断，TIMERx(x=0..4,7)
TIMER_INT_FLAG_CH3	通道3比较/捕获中断，TIMERx(x=0..4,7)
TIMER_INT_FLAG_CMT	换相更新中断，TIMERx(x=0,7)
TIMER_INT_FLAG_TRG	触发中断，TIMERx(x=0..4,7,8,11)

TIMER_INT_FLAG_BRK	中止中断，TIMERx(x=0,7)
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 3-509. 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..13)
TIMER_INT_FLAG_CH0	通道0比较/捕获中断，TIMERx(x=0..4,7..13)
TIMER_INT_FLAG_CH1	通道1比较/捕获中断，TIMERx(x=0..4,7,8,11)
TIMER_INT_FLAG_CH2	通道2比较/捕获中断，TIMERx(x=0..4,7)
TIMER_INT_FLAG_CH3	通道3比较/捕获中断，TIMERx(x=0..4,7)
TIMER_INT_FLAG_CMT	换相更新中断，TIMERx(x=0,7)
TIMER_INT_FLAG_TRG	触发中断，TIMERx(x=0..4,7,8,11)
*TIMER_INT_FLAG_*	中止中断，TIMERx(x=0,7)

BRK	
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 3-510. 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..13)
TIMER_FLAG_CH0	通道0比较/捕获标志， TIMERx(x=0..4,7..13)
TIMER_FLAG_CH1	通道1比较/捕获标志，TIMERx(x=0..4,7,8,11)
TIMER_FLAG_CH2	通道2比较/捕获标志，TIMERx(x=0..4,7)
TIMER_FLAG_CH3	通道3比较/捕获标志，TIMERx(x=0..4,7)
TIMER_FLAG_CMT	通道换相更新标志，TIMERx(x=0,7)
TIMER_FLAG_TRG	触发标志，TIMERx(x=0,7,8,11)
TIMER_FLAG_BRK	中止标志位，TIMERx(x=0,7)
TIMER_FLAG_CH0O	通道0捕获溢出标志，TIMERx(x=0..4,7..11)
TIMER_FLAG_CH1O	通道1捕获溢出标志，TIMERx(x=0..4,7,8,11)
TIMER_FLAG_CH2O	通道2捕获溢出标志，TIMERx(x=0..4,7)
TIMER_FLAG_CH3O	通道3捕获溢出标志，TIMERx(x=0..4,7)
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 3-511. 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..13)
TIMER_FLAG_CH0	通道0比较/捕获标志， TIMERx(x=0..4,7..13)
TIMER_FLAG_CH1	通道1比较/捕获标志，TIMERx(x=0..4,7,8,11)
TIMER_FLAG_CH2	通道2比较/捕获标志，TIMERx(x=0..4,7)
TIMER_FLAG_CH3	通道3比较/捕获标志，TIMERx(x=0..4,7)
TIMER_FLAG_CMT	通道换相更新标志，TIMERx(x=0,7)
TIMER_FLAG_TRG	触发标志，TIMERx(x=0,7,8,11)
TIMER_FLAG_BRK	中止标志位，TIMERx(x=0,7)
TIMER_FLAG_CH0O	通道0捕获溢出标志，TIMERx(x=0..4,7..11)
TIMER_FLAG_CH1O	通道1捕获溢出标志，TIMERx(x=0..4,7,8,11)
TIMER_FLAG_CH2O	通道2捕获溢出标志，TIMERx(x=0..4,7)
TIMER_FLAG_CH3O	通道3捕获溢出标志，TIMERx(x=0..4,7)
输出参数{out}	
-	-

返回值	
-	-

例如：

```
/* clear TIMER0 update flags */
timer_flag_clear (TIMER0, TIMER_FLAG_UP);
```

3.21. 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*3.21.1*描述了USART的寄存器列表，章节*3.21.2*对USART库函数进行说明。

3.21.1. 外设寄存器说明

USART寄存器列表如下表所示：

表 3-512. USART 寄存器

寄存器名称	寄存器描述
USART_STAT0	状态寄存器0
USART_DATA	数据寄存器
USART_BAUD	波特率寄存器
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_GP	保护时间和预分频器寄存器
USART_CTL3	控制寄存器3
USART_RT	接收超时寄存器
USART_STAT1	状态寄存器1
USART_CHC	兼容性控制寄存器

3.21.2. 外设库函数说明

USART库函数列表如下表所示：

表 3-513. USART 库函数

库函数名称	库函数描述
usart_deinit	复位外设USART
usart_baudrate_set	配置USART波特率
usart_parity_config	配置USART奇偶校验
usart_word_length_set	配置USART字长
usart_stop_bit_set	配置USART停止位
usart_enable	使能USART

库函数名称	库函数描述
usart_disable	失能USART
usart_transmit_config	USART发送配置
usart_receive_config	USART接收配置
usart_data_first_config	配置数据传输时低位在前或高位在前
usart_invert_config	配置USART反转功能
usart_receiver_timeout_enable	使能USART接收超时
usart_receiver_timeout_disable	失能USART接收超时
usart_receiver_timeout_threshold_config	设置USART接收超时阈值
usart_data_transmit	USART发送数据功能
usart_data_receive	USART接收数据功能
usart_address_config	在地址掩码唤醒模式下配置USART地址
usart_mute_mode_enable	使能USART静默模式
usart_mute_mode_disable	失能USART静默模式
usart_mute_mode_wakeup_config	配置USART静默模式唤醒方式
usart_lin_mode_enable	使能USART LIN模式
usart_lin_mode_disable	失能USART LIN模式
usart_lin_break_detection_length_config	配置USART LIN模式中断帧长度
usart_send_break	配置USART发送断开帧
usart_halfduplex_enable	使能USART半双工模式
usart_halfduplex_disable	失能USART半双工模式
usart_synchronous_clock_enable	在USART同步通讯模式下使能CK引脚
usart_synchronous_clock_disable	在USART同步通讯模式下失能CK引脚
usart_synchronous_clock_config	配置USART同步通讯模式参数
usart_guard_time_config	在USART智能卡模式下配置保护时间值
usart_smartcard_mode_enable	使能USART智能卡模式
usart_smartcard_mode_disable	失能USART智能卡模式
usart_smartcard_mode_nack_enable	在USART智能卡模式下使能NACK
usart_smartcard_mode_nack_disable	在USART智能卡模式下失能NACK
usart_smartcard_autoretry_config	配置智能卡自动重试次数
usart_block_length_config	配置智能卡T=1的接收时块的长度
usart_irda_mode_enable	使能USART串行红外编解码功能模块
usart_irda_mode_disable	失能USART串行红外编解码功能模块
usart_prescaler_config	在USART IrDA低功耗模式下配置外设时钟分频系数
usart_irda_lowpower_config	配置USART IrDA低功耗模式
usart_hardware_flow_rts_config	配置USART RTS硬件控制流
usart_hardware_flow_cts_config	配置USART CTS硬件控制流
usart_dma_receive_config	配置USART DMA接收功能
usart_dma_transmit_config	配置USART DMA发送功能
usart_hardware_flow_coherence_conf	配置硬件流控兼容模式

库函数名称	库函数描述
ig	
usart_flag_get	获取USART状态寄存器标志位
usart_flag_clear	清除USART状态寄存器标志位
usart_interrupt_enable	使能USART中断
usart_interrupt_disable	失能USART中断
usart_interrupt_flag_get	获取USART中断标志位状态
usart_interrupt_flag_clear	清除USART中断标志位状态

枚举类型 usart_flag_enum

表 3-514. 枚举类型 usart_flag_enum

成员名称	功能描述
USART_FLAG_CTS	CTS变化标志
USART_FLAG_LBD	LIN断开检测标志
USART_FLAG_TBE	发送数据寄存器空
USART_FLAG_TC	发送完成
USART_FLAG_RBNE	读数据缓冲区非空
USART_FLAG_IDLE	空闲帧检测标志
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_BSY	忙标志
USART_FLAG_EB	块结束标志
USART_FLAG_RT	接收超时标志
USART_FLAG_EPERR	校验错误超前检测标志

枚举类型 usart_interrupt_flag_enum

表 3-515. 枚举类型 usart_interrupt_flag_enum

成员名称	功能描述
USART_INT_FLAG_PERR	奇偶校验错误中断标志
USART_INT_FLAG_TBE	发送寄存器空中断标志
USART_INT_FLAG_TC	发送完成中断标志
USART_INT_FLAG_RBNE	读缓冲区非空中断标志
USART_INT_FLAG_RBNE_ORERR	读缓冲区非空和溢出中断标志
USART_INT_FLAG_IDLE	空闲帧检测中断标志
USART_INT_FLAG_LBD	LIN断开检测中断标志
USART_INT_FLAG_CTS	CTS中断标志
USART_INT_FLAG_ERR_ORER	溢出错误中断标志

成员名称	功能描述
R	
USART_INT_FLAG_ERR_NERR	噪声错误中断标志
USART_INT_FLAG_ERR_FERR	帧错误中断标志
USART_INT_FLAG_EB	块结束中断标志
USART_INT_FLAG_RT	接收超时中断标志

枚举类型 usart_interrupt_enum

表 3-516. 枚举类型 usart_interrupt_enum

成员名称	功能描述
USART_INT_PERR	奇偶校验错误中断
USART_INT_TBE	发送寄存器空中断
USART_INT_TC	发送完成中断
USART_INT_RBNE	读缓冲区非空中断和溢出错误中断
USART_INT_IDLE	空闲线检测中断
USART_INT_LBD	LIN断开检测中断
USART_INT_CTS	CTS中断
USART_INT_ERR	错误中断
USART_INT_EB	块结束中断
USART_INT_RT	接收超时中断

枚举类型 usart_invert_enum

表 3-517. 枚举类型 usart_invert_enum

成员名称	功能描述
USART_DINV_ENABLE	数据位反转
USART_DINV_DISABLE	数据位不反转
USART_TXPIN_ENABLE	TX管脚电平反转
USART_TXPIN_DISABLE	TX管脚电平不反转
USART_RXPIN_ENABLE	RX管脚电平反转
USART_RXPIN_DISABLE	RX管脚电平不反转

函数 usart_deinit

函数usart_deinit描述见下表：

表 3-518. 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/UARTx
先决条件	-

被调用函数	rcu_periph_reset_enable / rcu_periph_reset_disable
输入参数{in}	
usart_periph	外设USARTx/UARTx
USARTx	x=0,1,2
UARTx	x=3,4
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 3-519. 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/UARTx
USARTx	x=0,1,2
UARTx	x=3,4
输入参数{in}	
baudval	波特率值
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* configure USART0 baud rate value */
usart_baudrate_set(USART0, 115200);
```

函数 usart_parity_config

函数usart_parity_config描述见下表：

表 3-520. 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/UARTx
USARTx	x=0,1,2
UARTx	x=3,4
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 USART parity */
usart_parity_config(USART0, USART_PM_EVEN);
```

函数 usart_word_length_set

函数usart_word_length_set描述见下表：

表 3-521. 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/UARTx
USARTx	x=0,1,2
UARTx	x=3,4

输入参数{in}	
wlen	配置USART字长
USART_WL_8BIT	8 bits
USART_WL_9BIT	9 bits
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* configure USART0 word length */
usart_word_length_set(USART0, USART_WL_9BIT);
```

函数 usart_stop_bit_set

函数usart_stop_bit_set描述见下表：

表 3-522. 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/UARTx
USARTx	x=0,1,2
UARTx	x=3,4
输入参数{in}	
stblen	配置USART停止位
USART_STB_1BIT	1 bit
USART_STB_0_5BIT	0.5 bit,该位对UARTx(x=3,4)无效
USART_STB_2BIT	2 bits
USART_STB_1_5BIT	1.5 bits,该位对UARTx(x=3,4)无效
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* configure USART0 stop bit length */
```

usart_stop_bit_set(USART0, USART_STB_1_5BIT);

函数 usart_enable

函数usart_enable描述见下表：

表 3-523. 函数 usart_enable

函数名称	usart_enable
函数原型	void usart_enable(uint32_t usart_periph);
功能描述	使能USART
先决条件	-
被调用函数	-
输入参数{in}	
usart_periph	外设USARTx/UARTx
USARTx	x=0,1,2
UARTx	x=3,4
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* enable USART0 */
usart_enable(USART0);
```

函数 usart_disable

函数usart_disable描述见下表：

表 3-524. 函数 usart_disable

函数名称	usart_disable
函数原型	void usart_disable(uint32_t usart_periph);
功能描述	失能USART
先决条件	-
被调用函数	-
输入参数{in}	
usart_periph	外设USARTx/UARTx
USARTx	x=0,1,2
UARTx	x=3,4
输出参数{out}	
-	-
返回值	

-	-

例如：

/* disable USART0 */

usart_disable(USART0);

函数 usart_transmit_config

函数usart_transmit_config描述见下表：

表 3-525. 函数 usart_transmit_config

函数名称	usart_transmit_config
函数原型	void usart_transmit_config(uint32_t usart_periph, uint32_t txconfig);
功能描述	USART发送器配置
先决条件	-
被调用函数	-
输入参数{in}	
usart_periph	外设USARTx/UARTx
USARTx	x=0,1,2
UARTx	x=3,4
输入参数{in}	
txconfig	使能/失能USART发送器
USART_TRANSMIT_ENABLE	使能USART发送
USART_TRANSMIT_DISABLE	失能USART发送
输出参数{out}	
-	-
返回值	
-	-

例如：

/* configure USART0 transmitter */

usart_transmit_config(USART0,USART_TRANSMIT_ENABLE);

函数 usart_receive_config

函数usart_receive_config描述见下表：

表 3-526. 函数 usart_receive_config

函数名称	usart_receive_config
函数原型	void usart_receive_config(uint32_t usart_periph, uint32_t rxconfig);

功能描述	USART接收器配置
先决条件	-
被调用函数	-
输入参数{in}	
usart_periph	外设USARTx/UARTx
USARTx	x=0,1,2
UARTx	x=3,4
输入参数{in}	
rxconfig	使能/失能USART接收器
USART_RECEIVE_ENABLE	使能USART接收
USART_RECEIVE_DISABLE	失能USART接收
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* configure USART0 receiver */
usart_receive_config(USART0, USART_RECEIVE_ENABLE);
```

函数 usart_data_first_config

函数usart_data_first_config描述见下表：

表 3-527. 函数 usart_data_first_config

函数名称	usart_data_first_config
函数原型	void usart_data_first_config(uint32_t usart_periph, uint32_t msbf);
功能描述	配置数据传输时低位在前或高位在前
先决条件	-
被调用函数	-
输入参数{in}	
usart_periph	外设USARTx
USARTx	x=0,1,2
输入参数{in}	
msbf	数据传输时低位在前/高位在前
USART_MSBF_LSB	数据传输时低位在前
USART_MSBF_MSB	数据传输时高位在前
输出参数{out}	

-	-
返回值	
-	-

例如：

```
/* configure LSB of data first */
usart_data_first_config(USART0, USART_MSBF_LSB);
```

函数 usart_invert_config

函数usart_invert_config描述见下表：

表 3-528. 函数 usart_invert_config

函数名称	usart_invert_config
函数原型	void usart_invert_config(uint32_t usart_periph, usart_invert_enum invertpara);
功能描述	配置USART反转功能
先决条件	-
被调用函数	-
输入参数{in}	
usart_periph	外设USARTx
USARTx	x=0,1,2
输入参数{in}	
invertpara	参考*表3-517. 枚举类型usart_invert_enum*
USART_DINV_ENABLE	数据位电平反转
USART_DINV_DISABLE	数据位电平不反转
USART_TXPIN_ENABLE	TX引脚电平反转
USART_TXPIN_DISABLE	TX引脚电平不反转
USART_RXPIN_ENABLE	RX引脚电平反转
USART_RXPIN_DISABLE	RX引脚电平不反转
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* configure USART inversion */
```

usart_invert_config(USART0, USART_DINV_ENABLE);

函数 usart_receiver_timeout_enable

函数usart_receiver_timeout_enable描述见下表：

表 3-529. 函数 usart_receiver_timeout_enable

函数名称	usart_receiver_timeout_enable
函数原型	void usart_receiver_timeout_enable(uint32_t usart_periph);
功能描述	使能USART接收超时
先决条件	-
被调用函数	-
输入参数{in}	
usart_periph	外设USARTx
USARTx	x=0,1,2
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* enable receiver timeout of USART */
usart_receiver_timeout_enable(USART0);
```

函数 usart_receiver_timeout_disable

函数usart_receiver_timeout_disable描述见下表：

表 3-530. 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=0,1,2
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* disable receiver timeout of USART */
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 3-531. 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=0,1,2
输入参数{in}	
rtimeout	超时时间
0-0xFFFFFF	超时时间值
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* set the receiver timeout threshold of USART0 */
usart_receiver_timeout_threshold_config(USART0, 115200*3);
```

函数 usart_data_transmit

函数usart_data_transmit描述见下表：

表 3-532. 函数 usart_data_transmit

函数名称	usart_data_transmit
函数原型	void usart_data_transmit(uint32_t usart_periph, uint16_t data);
功能描述	USART发送数据功能
先决条件	-
被调用函数	-
输入参数{in}	
usart_periph	外设USARTx/UARTx
USARTx	x=0,1,2

UARTx	x=3,4
输入参数{in}	
data	发送的数据
0-0x1FF	发送的数据
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* USART0 transmit data */
usart_data_transmit(USART0, 0xAA);
```

函数 usart_data_receive

函数usart_data_receive描述见下表：

表 3-533. 函数 usart_data_receive

函数名称	usart_data_receive
函数原型	uint16_t usart_data_receive(uint32_t usart_periph);
功能描述	USART接收数据功能
先决条件	-
被调用函数	-
输入参数{in}	
usart_periph	外设USARTx/UARTx
USARTx	x=0,1,2
UARTx	x=3,4
输出参数{out}	
-	-
返回值	
Uint16_t	接收的数据(0-0x1FF)

例如：

```
/* USART0 receive data */
uint16_t temp;
temp = usart_data_receive(USART0);
```

函数 usart_address_config

函数usart_address_config描述见下表：

表 3-534. 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USARTx/UARTx
USARTx	x=0,1,2
UARTx	x=3,4
输入参数{in}	
addr	USART/UART地址
0-0xFF	USART/UART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_mute_mode_enable

函数usart_mute_mode_enable描述见下表：

表 3-535. 函数 usart_mute_mode_enable

函数名称	usart_mute_mode_enable
函数原型	void usart_mute_mode_enable(uint32_t usart_periph);
功能描述	使能USART静默模式
先决条件	-
被调用函数	-
输入参数{in}	
usart_periph	外设USARTx/UARTx
USARTx	x=0,1,2
UARTx	x=3,4
输出参数{out}	
-	-
返回值	
-	-

例如：

/* enable USART0 receiver in mute mode */

usart_mute_mode_enable(USART0);

函数 usart_mute_mode_disable

函数usart_mute_mode_disable描述见下表：

表 3-536. 函数 usart_mute_mode_disable

函数名称	usart_mute_mode_disable
函数原型	void usart_mute_mode_disable (uint32_t usart_periph);
功能描述	失能USART静默模式
先决条件	-
被调用函数	-
输入参数{in}	
usart_periph	外设USARTx/UARTx
USARTx	x=0,1,2
UARTx	x=3,4
输出参数{out}	
-	-
返回值	
-	-

例如：

/* disable USART0 receiver in mute mode */

usart_mute_mode_disable(USART0);

函数 usart_mute_mode_wakeup_config

函数usart_mute_mode_wakeup_config描述见下表：

表 3-537. 函数 usart_mute_mode_wakeup_config

函数名称	usart_mute_mode_wakeup_config
函数原型	void usart_mute_mode_wakeup_config(uint32_t usart_periph, uint32_t wmethod);
功能描述	配置USART静默模式唤醒方式
先决条件	-
被调用函数	-
输入参数{in}	
usart_periph	外设USARTx/UARTx
USARTx	x=0,1,2
UARTx	x=3,4
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 */
usart_mute_mode_wakeup_config(USART0, USART_WM_IDLE);
```

函数 usart_lin_mode_enable

函数usart_lin_mode_enable描述见下表：

表 3-538. 函数 usart_lin_mode_enable

函数名称	usart_lin_mode_enable
函数原型	void usart_lin_mode_enable(uint32_t usart_periph);
功能描述	使能USART LIN模式
先决条件	-
被调用函数	-
输入参数{in}	
usart_periph	外设USARTx/UARTx
USARTx	x=0,1,2
UARTx	x=3,4
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* USART0 LIN mode enable */
usart_lin_mode_enable(USART0);
```

函数 usart_lin_mode_disable

函数usart_lin_mode_disable描述见下表：

表 3-539. 函数 usart_lin_mode_disable

函数名称	usart_lin_mode_disable
函数原型	void usart_lin_mode_disable(uint32_t usart_periph);

功能描述	失能USART LIN模式
先决条件	-
被调用函数	-
输入参数{in}	
usart_periph	外设USARTx/UARTx
USARTx	x=0,1,2
UARTx	x=3,4
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* USART0 LIN mode disable */
usart_lin_mode_disable(USART0);
```

函数 usart_lin_break_dection_length_config

函数usart_lin_break_dection_length_config描述见下表：

表 3-540. 函数 usart_lin_break_dection_length_config

函数名称	usart_lin_break_dection_length_config
函数原型	void usart_lin_break_dection_length_config(uint32_t usart_periph, uint32_t lblen);
功能描述	配置USART LIN模式中断帧长度
先决条件	-
被调用函数	-
输入参数{in}	
usart_periph	外设USARTx/UARTx
USARTx	x=0,1,2
UARTx	x=3,4
输入参数{in}	
lblen	LIN模式中断帧长度
USART_LBLEN_10B	断开帧长度为10 bits
USART_LBLEN_11B	断开帧长度为11 bits
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* configure LIN break frame length */

usart_lin_break_dection_length_config(USART0, USART_LBLEN_10B);
```

函数 usart_send_break

函数usart_send_break描述见下表：

表 3-541. 函数 usart_send_break

函数名称	usart_send_break
函数原型	void usart_send_break(uint32_t usart_periph);
功能描述	配置USART发送断开帧
先决条件	-
被调用函数	-
输入参数{in}	
usart_periph	外设USARTx/UARTx
USARTx	x=0,1,2
UARTx	x=3,4
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* USART0 send break frame */

usart_send_break(USART0);
```

函数 usart_halfduplex_enable

函数usart_halfduplex_enable描述见下表：

表 3-542. 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}	
usart_periph	外设USARTx/UARTx
USARTx	x=0,1,2
UARTx	x=3,4
输出参数{out}	
-	-

返回值	
-	-

例如：

```
/* enable USART0 half duplex mode*/
usart_halfduplex_enable(USART0);
```

函数 usart_halfduplex_disable

函数usart_halfduplex_disable描述见下表：

表 3-543. 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/UARTx
USARTx	x=0,1,2
UARTx	x=3,4
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* disable USART0 half duplex mode*/
usart_halfduplex_disable(USART0);
```

函数 usart_synchronous_clock_enable

函数usart_synchronous_clock_enable描述见下表：

表 3-544. 函数 usart_synchronous_clock_enable

函数名称	usart_synchronous_clock_enable
函数原型	void usart_synchronous_clock_enable(uint32_t usart_periph);
功能描述	在USART同步通讯模式下使能CK引脚
先决条件	-
被调用函数	-
输入参数{in}	
usart_periph	外设USARTx

USARTx	x=0,1,2
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* enable USART0 CK pin in synchronous mode */
usart_synchronous_clock_enable(USART0);
```

函数 usart_synchronous_clock_disable

函数usart_synchronous_clock_disable描述见下表：

表 3-545. 函数 usart_synchronous_clock_disable

函数名称	usart_synchronous_clock_disable
函数原型	void usart_synchronous_clock_disable(uint32_t usart_periph);
功能描述	在USART同步通讯模式下失能CK引脚
先决条件	-
被调用函数	-
输入参数{in}	
usart_periph	外设USARTx
USARTx	x=0,1,2
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* disable USART0 CK pin in synchronous mode */
usart_synchronous_clock_disable(USART0);
```

函数 usart_synchronous_clock_config

函数usart_synchronous_clock_config描述见下表：

表 3-546. 函数 usart_synchronous_clock_config

函数名称	usart_synchronous_clock_config
函数原型	void usart_synchronous_clock_config(uint32_t usart_periph, uint32_t clen, uint32_t cph, uint32_t cpl);
功能描述	配置USART同步通讯模式参数
先决条件	-

被调用函数	-
输入参数{in}	
usart_periph	外设USARTx
USARTx	x=0,1,2
输入参数{in}	
clen	CK信号长度
USART_CLEN_NONE	8位数据帧中有7个CK脉冲，9位数据帧中有8个CK脉冲
USART_CLEN_EN	8位数据帧中有8个CK脉冲，9位数据帧中有9个CK脉冲
输入参数{in}	
cph	时钟相位
USART_CPH_1CK	在首个时钟边沿采样第一个数据
USART_CPH_2CK	在第二个时钟边沿采样第一个数据
输入参数{in}	
cpl	时钟极性
USART_CPL_LOW	CK引脚不对外发送时保持为低电平
USART_CPL_HIGH	CK引脚不对外发送时保持为高电平
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* configure USART0 synchronous mode parameters */
usart_synchronous_clock_config(USART0,USART_CLEN_EN,USART_CPH_2CK,
USART_CPL_HIGH);
```

函数 usart_guard_time_config

函数usart_guard_time_config描述见下表：

表 3-547. 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,uint8_t gaut);
功能描述	在USART智能卡模式下配置保护时间值
先决条件	-
被调用函数	-
输入参数{in}	
usart_periph	外设USARTx
USARTx	x=0,1,2
输入参数{in}	
gaut	保护时间值

0-0xFF	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, 0x55);
```

函数 usart_smartcard_mode_enable

函数usart_smartcard_mode_enable描述见下表：

表 3-548. 函数 usart_smartcard_mode_enable

函数名称	usart_smartcard_mode_enable
函数原型	void usart_smartcard_mode_enable(uint32_t usart_periph);
功能描述	使能USART智能卡模式
先决条件	-
被调用函数	-
输入参数{in}	
usart_periph	外设USARTx
USARTx	x=0,1,2
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* USART0 smartcard mode enable */
usart_smartcard_mode_enable(USART0);
```

函数 usart_smartcard_mode_disable

函数usart_smartcard_mode_disable描述见下表：

表 3-549. 函数 usart_smartcard_mode_disable

函数名称	usart_smartcard_mode_disable
函数原型	void usart_smartcard_mode_disable(uint32_t usart_periph);
功能描述	失能USART智能卡模式
先决条件	-
被调用函数	-

输入参数{in}	
usart_periph	外设USARTx
USARTx	x=0,1,2
输出参数{out}	
-	-
返回值	
-	-

例如：

/* USART0 smartcard mode disable */

usart_smartcard_mode_disable(USART0);

函数 usart_smartcard_mode_nack_enable

函数usart_smartcard_mode_nack_enable描述见下表：

表 3-550. 函数 usart_smartcard_mode_nack_enable

函数名称	usart_smartcard_mode_nack_enable
函数原型	void usart_smartcard_mode_nack_enable(uint32_t usart_periph);
功能描述	在USART智能卡模式下使能NACK
先决条件	-
被调用函数	-
输入参数{in}	
usart_periph	外设USARTx
USARTx	x=0,1,2
输出参数{out}	
-	-
返回值	
-	-

例如：

/* enable USART0 NACK in smartcard mode */

usart_smartcard_mode_nack_enable(USART0);

函数 usart_smartcard_mode_nack_disable

函数usart_smartcard_mode_nack_disable描述见下表：

表 3-551. 函数 usart_smartcard_mode_nack_disable

函数名称	usart_smartcard_mode_nack_disable
函数原型	void usart_smartcard_mode_nack_disable(uint32_t usart_periph);
功能描述	在USART智能卡模式下失能NACK

先决条件	-
被调用函数	-
输入参数{in}	
usart_periph	外设USARTx
USARTx	x=0,1,2
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* disable USART0 NACK in smartcard mode */
usart_smartcard_mode_nack_disable(USART0);
```

函数 usart_smartcard_autoretry_config

函数usart_smartcard_autoretry_config描述见下表：

表 3-552. 函数 usart_smartcard_autoretry_config

函数名称	usart_smartcard_autoretry_config
函数原型	void usart_smartcard_autoretry_config(uint32_t usart_periph, uint8_t scrtnum);
功能描述	配置智能卡自动重试次数
先决条件	-
被调用函数	-
输入参数{in}	
usart_periph	外设USARTx
USARTx	x=0,1,2
输入参数{in}	
scrtnum	智能卡自动重试次数
0-0xFF	自动重试次数
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* configure smartcard auto-retry number */
usart_smartcard_autoretry_config (USART0, 0xFF);
```

函数 usart_block_length_config

函数usart_block_length_config描述见下表：

表 3-553. 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, uint8_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=0,1,2
输入参数{in}	
bl	块长度
0-0xFF	块长度
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* configure block length in Smartcard T=1 reception */
usart_block_length_config(USART0, 0xFF);
```

函数 usart_irda_mode_enable

函数usart_irda_mode_enable描述见下表：

表 3-554. 函数 usart_irda_mode_enable

函数名称	usart_irda_mode_enable
函数原型	void usart_irda_mode_enable(uint32_t usart_periph);
功能描述	使能USART串行红外编解码功能模块
先决条件	-
被调用函数	-
输入参数{in}	
usart_periph	外设USARTx
USARTx	x=0,1,2
UARTx	x=3,4
输出参数{out}	

-	-
返回值	
-	-

例如：

```
/* enable USART0 IrDA mode */
usart_irda_mode_enable(USART0);
```

函数 usart_irda_mode_disable

函数usart_irda_mode_disable描述见下表：

表 3-555. 函数 usart_irda_mode_disable

函数名称	usart_irda_mode_disable
函数原型	void usart_irda_mode_disable(uint32_t usart_periph);
功能描述	失能USART串行红外编解码功能模块
先决条件	-
被调用函数	-
输入参数{in}	
usart_periph	外设USARTx/UARTx
USARTx	x=0,1,2
UARTx	x=3,4
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* disable USART0 IrDA mode */
usart_irda_mode_disable(USART0);
```

函数 usart_prescaler_config

函数usart_prescaler_config描述见下表：

表 3-556. 函数 usart_prescaler_config

函数名称	usart_prescaler_config
函数原型	void usart_prescaler_config(uint32_t usart_periph, uint8_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/UARTx
USARTx	x=0,1,2
UARTx	x=3,4
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 3-557. 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/UARTx
USARTx	x=0,1,2
UARTx	x=3,4
输入参数{in}	
irlp	IrDA低功耗模式或正常模式
USART_IRLP_LOW	低功耗模式
USART_IRLP_NORMAL	正常模式
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* configure USART0 IrDA low-power */
usart_irda_lowpower_config(USART0, USART_IRLP_LOW);
```

函数 usart_hardware_flow_rts_config

函数usart_hardware_flow_rts_config描述见下表：

表 3-558. 函数 usart_hardware_flow_rts_config

函数名称	usart_hardware_flow_rts_config
函数原型	void usart_hardware_flow_rts_config(uint32_t usart_periph, uint32_t rtsconfig);
功能描述	配置USART RTS硬件控制流
先决条件	-
被调用函数	-
输入参数{in}	
usart_periph	外设USARTx
USARTx	x=0,1,2
输入参数{in}	
rtsconfig	使能/失能RTS
USART_RTS_ENABLE	使能RTS
USART_RTS_DISABLE	失能RTS
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* configure USART0 hardware flow control RTS */
usart_hardware_flow_cts_config(USART0, USART_RTS_ENABLE);
```

函数 usart_hardware_flow_cts_config

函数usart_hardware_flow_cts_config描述见下表：

表 3-559. 函数 usart_hardware_flow_cts_config

函数名称	usart_hardware_flow_cts_config
函数原型	void usart_hardware_flow_cts_config(uint32_t usart_periph, uint32_t ctsconfig);
功能描述	配置USART CTS硬件控制流
先决条件	-
被调用函数	-
输入参数{in}	
usart_periph	外设USARTx
USARTx	x=0,1,2
输入参数{in}	

ctsconfig	使能/失能CTS
USART_CTS_ENABLE	使能CTS
USART_CTS_DISABLE	失能CTS
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* configure USART0 hardware flow control CTS */
usart_hardware_flow_cts_config(USART0, USART_CTS_ENABLE);
```

函数 usart_dma_receive_config

函数usart_dma_receive_config描述见下表：

表 3-560. 函数 usart_dma_receive_config

函数名称	usart_dma_receive_config
函数原型	void usart_dma_receive_config(uint32_t usart_periph, uint32_t dmaconfig);
功能描述	配置USART DMA接收功能
先决条件	-
被调用函数	-
输入参数{in}	
usart_periph	外设USARTx/UARTx
USARTx	x=0,1,2
UARTx	x=3,4
输入参数{in}	
dmaconfig	USART DMA模式
USART_RECEIVE_DMA_ENABLE	使能DMA接收功能
USART_RECEIVE_DMA_DISABLE	失能DMA接收功能
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* USART0 DMA enable for reception */
usart_dma_receive_config(USART0, USART_RECEIVE_DMA_ENABLE);
```

函数 usart_dma_transmit_config

函数usart_dma_transmit_config描述见下表：

表 3-561. 函数 usart_dma_transmit_config

函数名称	usart_dma_transmit_config
函数原型	void usart_dma_transmit_config(uint32_t usart_periph, uint32_t dmaconfig);
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/UARTx
USARTx	x=0,1,2
UARTx	x=3,4
输入参数{in}	
dmaconfig	USART DMA模式
USART_TRANSMIT_DMA_ENABLE	使能DMA发送功能
USART_TRANSMIT_DMA_DISABLE	失能DMA发送功能
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* USART0 DMA enable for transmission */
usart_dma_transmit_config(USART0, USART_TRANSMIT_DMA_ENABLE);
```

函数 usart_hardware_flow_coherence_config

函数usart_hardware_flow_coherence_config描述见下表：

表 3-562. 函数 usart_hardware_flow_coherence_config

函数名称	usart_hardware_flow_coherence_config
函数原型	void usart_hardware_flow_coherence_config(uint32_t usart_periph, uint32_t hcm);
功能描述	配置硬件流控兼容模式
先决条件	-
被调用函数	-
输入参数{in}	
usart_periph	外设USARTx/UARTx

USARTx	x=0,1,2
UARTx	x=3
输入参数{in}	
hcm	硬件流控制兼容模式
USART_RTS_NONE_COHERENCE	nRTS信号与USART_STAT0寄存器中RBNE位相同
USART_RTS_COHERENCE	nRTS信号在最后一个数据位被采样后被置位
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* configure hardware flow control coherence mode */
usart_hardware_flow_coherence_config(USART0, USART_RTS_COHERENCE);
```

函数 usart_flag_get

函数usart_flag_get描述见下表：

表 3-563. 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/UARTx
USARTx	x=0,1,2
UARTx	x=3,4
输入参数{in}	
flag	USART标志位，参考*表3-514. 枚举类型usart_flag_enum*
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 3-564. 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/UARTx
USARTx	x=0,1,2
UARTx	x=3,4
输入参数{in}	
flag	USART标志位，参考*表3-514. 枚举类型usart_flag_enum*
USART_FLAG_CTS	CTS变化标志
USART_FLAG_LBD	LIN断开检测标志
USART_FLAG_TC	发送完成
USART_FLAG_RBNE	读数据缓冲区非空
USART_FLAG_EB	块结束标志
USART_FLAG_RT	接收超时标志
USART_FLAG_EPERR	校验错误超前检测标志
输出参数{out}	
-	-
返回值	
-	-

例如：

/* clear USART0 flag */

usart_flag_clear(USART0,USART_FLAG_TC);

函数 usart_interrupt_enable

函数usart_interrupt_enable描述见下表：

表 3-565. 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/UARTx
USARTx	x=0,1,2
UARTx	x=3,4
输入参数{in}	
interrupt	USART中断，参考*表3-516. 枚举类型usart_interrupt_enum*
USART_INT_PERR	校验错误中断
USART_INT_TBE	发送缓冲区空中断
USART_INT_TC	发送完成中断
USART_INT_RBNE	读数据缓冲区非空中断和过载错误中断
USART_INT_IDLE	IDLE线检测中断
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_RT	接收超时事件中断
USART_INT_EB	块结束事件中断
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 3-566. 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/UARTx
USARTx	x=0,1,2
UARTx	x=3,4
输入参数{in}	
interrupt	USART中断，参考*表3-516. 枚举类型usart_interrupt_enum*
USART_INT_PERR	校验错误中断
USART_INT_TBE	发送缓冲区空中断
USART_INT_TC	发送完成中断
USART_INT_RBNE	读数据缓冲区非空中断和过载错误中断
USART_INT_IDLE	IDLE线检测中断
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_RT	接收超时事件中断
USART_INT_EB	块结束事件中断
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 3-567. 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/UARTx
USARTx	x=0,1,2
UARTx	x=3,4
输入参数{in}	
int_flag	USART中断标志，参考*表3-515. 枚举类型usart_interrupt_flag_enum*
USART_INT_FLAG	校验错误中断标志

_PERR	
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_CTS	CTS中断标志
USART_INT_FLAG_ERR_ORERR	过载错误中断标志
USART_INT_FLAG_ERR_NERR	噪声错误中断标志
USART_INT_FLAG_ERR_FERR	帧错误中断标志
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
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 3-568. 函数 usart_interrupt_flag_clear

函数名称	usart_interrupt_flag_clear
函数原型	void usart_interrupt_flag_clear(uint32_t usart_periph,

	usart_interrupt_flag_enum int_flag);
功能描述	清除USART中断标志位状态
先决条件	-
被调用函数	-
输入参数{in}	
usart_periph	外设USARTx/UARTx
USARTx	x=0,1,2
UARTx	x=3,4
输入参数{in}	
int_flag	USART中断标志，参考*表3-515. 枚举类型usart_interrupt_flag_enum*
USART_INT_FLAG_CTS	CTS变化中断标志
USART_INT_FLAG_LBD	LIN断开检测中断标志
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_EB	块结束事件中断标志
USART_INT_FLAG_RT	超时事件中断标志
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* clear the USART0 interrupt flag */
usart_interrupt_flag_clear(USART0, USART_INT_FLAG_RBNE);
```

3.22. 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*3.22.1*描述了WWDGT的寄存器列表，章节*3.22.2*对WWDGT库函数进行说明。

3.22.1. 外设寄存器说明

WWDGT寄存器列表如下表所示：

表 3-569. WWDGT 寄存器

寄存器名称	寄存器描述
WWDGT_CTL	控制寄存器
WWDGT_CFG	配置寄存器
WWDGT_STAT	状态寄存器

3.22.2. 外设库函数说明

WWDGT库函数列表如下表所示：

表 3-570. WWDGT 库函数

库函数名称	库函数说明
wwdgt_deinit	将WWDGT寄存器重设为缺省值
wwdgt_enable	使能WWDGT
wwdgt_counter_update	设置WWDGT计数器更新值
wwdgt_config	设置WWDGT计数器值、窗口值和预分频值
wwdgt_interrupt_enable	使能WWDGT提前唤醒中断
wwdgt_flag_get	检查WWDGT提前唤醒中断标志位是否置位
wwdgt_flag_clear	清除WWDGT提前唤醒中断标志位状态

函数 wwdgt_deinit

函数wwdgt_deinit描述见下表：

表 3-571. 函数 wwdgt_deinit

函数名称	wwdgt_deinit
函数原型	void wwdgt_deinit(void);
功能描述	将WWDGT寄存器重设为缺省值
先决条件	-
被调用函数	-
输入参数{in}	
-	-
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* reset the window watchdog timer configuration */
wwdgt_deinit ( );
```

函数 wwdgt_enable

函数wwdgt_enable描述见下表：

表 3-572. 函数 wwdgt_enable

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

例如：

```
/* start the window watchdog timer counter */
wwdgt_enable ( );
```

函数 wwdgt_counter_update

函数wwdgt_counter_update描述见下表：

表 3-573. 函数 wwdgt_counter_update

函数名称	wwdgt_counter_update
函数原型	void wwdgt_counter_update(uint16_t counter_value);
功能描述	设置WWDGT计数器更新值
先决条件	-
被调用函数	-
输入参数{in}	
counter_value	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 3-574. 函数 wwdgt_config

函数名称	wwdgt_config
函数原型	void wwdgt_config(uint16_t counter, uint16_t window, uint32_t prescaler);
功能描述	设置WWDGT计数器值、窗口值和预分频值
先决条件	-
被调用函数	-
输入参数{in}	
counter	0x00 - 0x7F
输入参数{in}	
window	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 3-575. 函数 wwdgt_interrupt_enable

函数名称	wwdgt_interrupt_enable
函数原型	void wwdgt_interrupt_enable(void);
功能描述	使能WWDGT提前唤醒中断

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

例如：

```
/* enable early wakeup interrupt of WWDGT */
wwdgt_interrupt_enable ( );
```

函数 wwdgt_flag_get

函数wwdgt_flag_get描述见下表：

表 3-576. 函数 wwdgt_flag_get

函数名称	wwdgt_flag_get
函数原型	FlagStatus wwdgt_flag_get(void);
功能描述	检查WWDGT提前唤醒中断标志位是否置位
先决条件	-
被调用函数	-
输入参数{in}	
-	-
输出参数{out}	
-	-
返回值	
FlagStatus	SET or RESET

例如：

```
/* test if the counter value update has reached the 0x40 */
FlagStatus status;
status = wwdgt_flag_get ( );
if(status == RESET)
{
...
}else
{
```

...

}

函数 wwdgt_flag_clear

函数wwdgt_flag_clear描述见下表：

表 3-577. 函数 wwdgt_flag_clear

函数名称	wwdgt_flag_clear
函数原型	void wwdgt_flag_clear(void);
功能描述	清除WWDGT提前唤醒中断标志位状态
先决条件	-
被调用函数	-
输入参数{in}	
-	-
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* clear early wakeup interrupt state of WWDGT */
wwdgt_flag_clear ( );
```

4. 版本历史

表 4-1. 版本历史

版本号.	说明	日期
1.0	初稿发布	2022 年 8 月 15 日
1.1	1. 修改 Usart 函数 usart_data_transmit usart_guard_time_config usart_smartcard_autoretry_config usart_block_length_config usart_dma_receive_config usart_dma_transmit_config 2. 修改 Timer 编码器名称	2022 年 12 月 31 日
1.2	修改整个 DAC 章节	2023 年 12 月 31 日
1.3	1. 参数宏 I2S_STD_PHILLIPS 修改为 I2S_STD_PHILIPS 2. 更新版权信息	2025 年 8 月 8 日
1.4	1. 移除函数 ***spi_quad_io23_output_enable*** 和 ***spi_quad_io23_output_disable***.	2026 年 2 月 4 日
1.5	1. 章节 ***2.固件库概述***: 增加 GD32EBuilder_project、GD32EBuilder 环境描述。	2026 年 4 月 14 日

Important Notice

This document is the property of GigaDevice Semiconductor Inc. and its subsidiaries (the "Company"). This document, including any product of the Company described in this document (the "Product"), is owned by the Company according to the laws of the People's Republic of China and other applicable laws. The Company reserves all rights under such laws and no Intellectual Property Rights are transferred (either wholly or partially) or licensed by the Company (either expressly or impliedly) herein. The names and brands of third party referred thereto (if any) are the property of their respective owner and referred to for identification purposes only.

To the maximum extent permitted by applicable law, the Company makes no representations or warranties of any kind, express or implied, with regard to the merchantability and the fitness for a particular purpose of the Product, nor does the Company assume any liability arising out of the application or use of any Product. Any information provided in this document is provided only for reference purposes. It is the sole responsibility of the user of this document to determine whether the Product is suitable and fit for its applications and products planned, and properly design, program, and test the functionality and safety of its applications and products planned using the Product. The Product is designed, developed, and/or manufactured for ordinary business, industrial, personal, and/or household applications only, and the Product is not designed or intended for use in (i) safety critical applications such as weapons systems, nuclear facilities, atomic energy controller, combustion controller, aeronautic or aerospace applications, traffic signal instruments, pollution control or hazardous substance management; (ii) life-support systems, other medical equipment or systems (including life support equipment and surgical implants); (iii) automotive applications or environments, including but not limited to applications for active and passive safety of automobiles (regardless of front market or aftermarket), for example, EPS, braking, ADAS (camera/fusion), EMS, TCU, BMS, BSG, TPMS, Airbag, Suspension, DMS, ICMS, Domain, ESC, DCDC, e-clutch, advanced-lighting, etc.. Automobile herein means a vehicle propelled by a self-contained motor, engine or the like, such as, without limitation, cars, trucks, motorcycles, electric cars, and other transportation devices; and/or (iv) other uses where the failure of the device or the Product can reasonably be expected to result in personal injury, death, or severe property or environmental damage (collectively "Unintended Uses"). Customers shall take any and all actions to ensure the Product meets the applicable laws and regulations. The Company is not liable for, in whole or in part, and customers shall hereby release the Company as well as its suppliers and/or distributors from, any claim, damage, or other liability arising from or related to all Unintended Uses of the Product. Customers shall indemnify and hold the Company, and its officers, employees, subsidiaries, affiliates as well as its suppliers and/or distributors harmless from and against all claims, costs, damages, and other liabilities, including claims for personal injury or death, arising from or related to any Unintended Uses of the Product.

Information in this document is provided solely in connection with the Product. The Company reserves the right to make changes, corrections, modifications or improvements to this document and the Product described herein at any time without notice. The Company shall have no responsibility whatsoever for conflicts or incompatibilities arising from future changes to them. Information in this document supersedes and replaces information previously supplied in any prior versions of this document.

© 2026 GigaDevice Semiconductor Inc. - All rights reserved