

GigaDevice Semiconductor Inc.

GD32F10x

Arm[®] Cortex[®]-M3 32-bit MCU

**固件库
使用指南**

2.2 版本

(2024 年 1 月)

目录

目录.....	2
图索引.....	5
表索引.....	6
1. 介绍	23
1.1. 文档和固件库规则	23
1.1.1. 外设缩写.....	23
1.1.2. 命名规则.....	24
2. 固件库概述.....	25
2.1. 文件组织结构.....	25
2.1.1. Examples 文件夹.....	25
2.1.2. Firmware 文件夹.....	26
2.1.3. Template 文件夹.....	26
2.1.4. Utilities 文件夹.....	29
2.2. 固件库文件描述.....	29
3. 外设固件库.....	30
3.1. 外设固件库概述.....	30
3.2. ADC.....	30
3.2.1. 外设寄存器描述.....	30
3.2.2. 外设库函数说明.....	31
3.3. BKP.....	56
3.3.1. 外设寄存器说明.....	56
3.3.2. 外设库函数说明.....	57
3.4. CAN.....	68
3.4.1. 外设寄存器说明.....	68
3.4.2. 外设库函数说明.....	68
3.5. CRC.....	91
3.5.1. 外设寄存器说明.....	91
3.5.2. 外设库函数说明.....	91
3.6. DAC.....	95
3.6.1. 外设寄存器说明.....	95
3.6.2. 外设库函数说明.....	96
3.7. DBG.....	110
3.7.1. 外设寄存器说明.....	110
3.7.2. 外设库函数说明.....	110

3.8. DMA.....	116
3.8.1. 外设寄存器说明.....	117
3.8.2. 外设库函数说明.....	117
3.9. ENET.....	137
3.9.1. 外设寄存器描述.....	137
3.9.2. 外设库函数说明.....	139
3.10. EXMC.....	208
3.10.1. 外设寄存器说明.....	208
3.10.2. 外设库函数说明.....	209
3.11. EXTI.....	227
3.11.1. 外设寄存器说明.....	227
3.11.2. 外设库函数说明.....	228
3.12. FMC.....	235
3.12.1. 外设寄存器说明.....	235
3.12.2. 外设库函数说明.....	236
3.13. FWDGT.....	257
3.13.1. 外设寄存器说明.....	257
3.13.2. 外设库函数说明.....	258
3.14. GPIO.....	263
3.14.1. 外设寄存器说明.....	263
3.14.2. 外设库函数说明.....	263
3.15. I2C.....	276
3.15.1. 外设寄存器说明.....	276
3.15.2. 外设库函数说明.....	277
3.16. MISC.....	299
3.16.1. 外设寄存器说明.....	300
3.16.2. 外设库函数说明.....	301
3.17. PMU.....	308
3.17.1. 外设寄存器说明.....	308
3.17.2. 外设库函数说明.....	308
3.18. RCU.....	315
3.18.1. 外设寄存器说明.....	315
3.18.2. 外设库函数说明.....	316
3.19. RTC.....	345
3.19.1. 外设寄存器描述.....	345
3.19.2. 外设库函数描述.....	346
3.20. SDIO.....	355
3.20.1. 外设寄存器说明.....	355
3.20.2. 外设库函数说明.....	355

3.21.	SPI	386
3.21.1.	外设寄存器说明.....	386
3.21.2.	外设库函数说明.....	387
3.22.	TIMER	407
3.22.1.	外设寄存器说明.....	407
3.22.2.	外设库函数说明.....	408
3.23.	USART	463
3.23.1.	外设寄存器说明.....	463
3.23.2.	外设库函数说明.....	464
3.24.	WWDGT	492
3.24.1.	外设寄存器说明.....	492
3.24.2.	外设库函数说明.....	492
4.	版本历史.....	497

图索引

图 2-1. GD32F10x 固件库文件组织结构.....	25
图 2-2. 选择外设例程文件.....	27
图 2-3. 拷贝外设例程文件.....	28
图 2-4. 打开工程文件.....	28
图 2-5. 配置工程文件.....	28
图 2-6. 编译调试下载.....	29

表索引

表 1-1. 外设缩写	23
表 2-1. 固件函数库文件描述.....	29
表 3-1. 外设固件库函数描述格式.....	30
表 3-2. ADC 寄存器.....	30
表 3-3. ADC 库函数.....	31
表 3-4. 函数 adc_deinit.....	32
表 3-5. 函数 adc_mode_config.....	32
表 3-6. 函数 adc_special_function_config.....	33
表 3-7. 函数 adc_data_alignment_config.....	34
表 3-8. 函数 adc_enable.....	35
表 3-9. 函数 adc_disable.....	35
表 3-10. 函数 adc_calibration_enable.....	36
表 3-11. 函数 adc_tempsensor_vrefint_enable.....	36
表 3-12. 函数 adc_tempsensor_vrefint_disable.....	37
表 3-13. 函数 adc_dma_mode_enable.....	37
表 3-14. 函数 adc_dma_mode_disable.....	38
表 3-15. 函数 adc_discontinuous_mode_config.....	38
表 3-16. 函数 adc_channel_length_config.....	39
表 3-17. 函数 adc_regular_channel_config.....	40
表 3-18. 函数 adc_inserted_channel_config.....	41
表 3-19. 函数 adc_inserted_channel_offset_config.....	42
表 3-20. 函数 adc_external_trigger_source_config.....	43
表 3-21. 函数 adc_external_trigger_config.....	45
表 3-22. 函数 adc_software_trigger_enable.....	46
表 3-23. 函数 adc_regular_data_read.....	47
表 3-24. 函数 adc_inserted_data_read.....	48
表 3-25. 函数 adc_sync_mode_convert_value_read.....	48
表 3-26. 函数 adc_watchdog_single_channel_enable.....	49
表 3-27. 函数 adc_watchdog_group_channel_enable.....	49
表 3-28. 函数 adc_watchdog_disable.....	50
表 3-29. 函数 adc_watchdog_threshold_config.....	51
表 3-30. 函数 adc_flag_get.....	51
表 3-31. 函数 adc_flag_clear.....	52
表 3-32. 函数 adc_regular_software_startconv_flag_get.....	53
表 3-33. 函数 adc_inserted_software_startconv_flag_get.....	53
表 3-34. 函数 adc_interrupt_flag_get.....	54
表 3-35. 函数 adc_interrupt_flag_clear.....	54
表 3-36. 函数 adc_interrupt_enable.....	55
表 3-37. 函数 adc_interrupt_disable.....	56
表 3-38. BKP 寄存器.....	56

表 3-39. BKP 库函数.....	57
表 3-40. 枚举类型 bkp_data_register_enum.....	57
表 3-41. 函数 bkp_deinit.....	58
表 3-42. 函数 bkp_data_write.....	59
表 3-43. 函数 bkp_data_read.....	60
表 3-44. 函数 bkp_rtc_calibration_output_enable	60
表 3-45. 函数 bkp_rtc_calibration_output_disable.....	61
表 3-46. 函数 bkp_rtc_signal_output_enable	61
表 3-47. 函数 bkp_rtc_signal_output_disable	62
表 3-48. 函数 bkp_rtc_output_select.....	62
表 3-49. 函数 bkp_rtc_calibration_value_set.....	63
表 3-50. 函数 bkp_tamper_detection_enable	63
表 3-51. 函数 bkp_tamper_detection_disable	64
表 3-52. 函数 bkp_tamper_active_level_set.....	64
表 3-53. 函数 bkp_interrupt_enable	65
表 3-54. 函数 bkp_interrupt_disable.....	65
表 3-55. 函数 bkp_flag_get.....	66
表 3-56. 函数 bkp_flag_clear	66
表 3-57. 函数 bkp_interrupt_flag_get.....	67
表 3-58. 函数 bkp_interrupt_flag_clear.....	67
表 3-59. CAN 寄存器.....	68
表 3-60. CAN 库函数.....	68
表 3-61. 结构体 can_parameter_struct.....	69
表 3-62. 结构体 can_transmit_message_struct	70
表 3-63. 结构体 can_receive_message_struct	70
表 3-64. 结构体 can_filter_parameter_struct	70
表 3-65. 枚举类型 can_flag_enum.....	70
表 3-66. 枚举类型 can_interrupt_flag_enum	71
表 3-67. 枚举类型 can_error_enum	72
表 3-68. 枚举类型 can_transmit_state_enum.....	72
表 3-69. 枚举类型 can_struct_type_enum.....	72
表 3-70. 函数 can_deinit.....	73
表 3-71. 函数 can_struct_para_init.....	73
表 3-72. 函数 can_init.....	74
表 3-73. 函数 can_filter_init.....	74
表 3-74. 函数 can1_filter_start_bank.....	75
表 3-75. 函数 can_debug_freeze_enable.....	76
表 3-76. 函数 can_debug_freeze_disable	76
表 3-77. 函数 can_time_trigger_mode_enable	77
表 3-78. 函数 can_time_trigger_mode_disable.....	77
表 3-79. 函数 can_message_transmit.....	78
表 3-80. 函数 can_transmit_states.....	78
表 3-81. 函数 can_transmission_stop	79
表 3-82. 函数 can_message_receive	79

表 3-83. 函数 can_fifo_release.....	80
表 3-84. 函数 can_receive_message_length_get.....	81
表 3-85. 函数 can_working_mode_set.....	81
表 3-86. 函数 can_wakeup.....	82
表 3-87. 函数 can_error_get.....	82
表 3-88. 函数 can_receive_error_number_get.....	83
表 3-89. 函数 can_transmit_error_number_get.....	84
表 3-90. 函数 can_interrupt_enable	84
表 3-91. 函数 can_interrupt_disable.....	85
表 3-92. 函数 can_flag_get.....	86
表 3-93. 函数 can_flag_clear	87
表 3-94. 函数 can_interrupt_flag_get.....	88
表 3-95. 函数 can_interrupt_flag_clear.....	90
表 3-96. CRC 寄存器.....	91
表 3-97. CRC 库函数.....	91
表 3-98. 函数 crc_deinit.....	91
表 3-99. 函数 crc_data_register_reset.....	92
表 3-100. 函数 crc_data_register_read	92
表 3-101. 函数 crc_free_data_register_read	93
表 3-102. 函数 crc_free_data_register_write	93
表 3-103. 函数 crc_single_data_calculate	94
表 3-104. 函数 crc_block_data_calculate.....	94
表 3-105. DAC 寄存器.....	95
表 3-106. DAC 库函数.....	96
表 3-107. 函数 dac_deinit.....	96
表 3-108. 函数 dac_enable	97
表 3-109. 函数 dac_disable	98
表 3-110. 函数 dac_dma_enable.....	98
表 3-111. 函数 dac_dma_disable.....	99
表 3-112. 函数 dac_output_buffer_enable	99
表 3-113. 函数 dac_output_buffer_disable	100
表 3-114. 函数 dac_output_value_get.....	100
表 3-115. 函数 dac_data_set.....	101
表 3-116. 函数 dac_trigger_enable.....	102
表 3-117. 函数 dac_trigger_disable	102
表 3-118. 函数 dac_trigger_source_config.....	103
表 3-119. 函数 dac_software_trigger_enable.....	104
表 3-120. 函数 dac_wave_mode_config	105
表 3-121. 函数 dac_lfsr_noise_config.....	105
表 3-122. 函数 dac_triangle_noise_config.....	106
表 3-123. 函数 dac_concurrent_enable	107
表 3-124. 函数 dac_concurrent_disable	107
表 3-125. 函数 dac_concurrent_software_trigger_enable	108
表 3-126. 函数 dac_concurrent_output_buffer_enable	108

表 3-127. 函数 dac_concurrent_output_buffer_disable.....	109
表 3-128. 函数 dac_concurrent_data_set.....	109
表 3-129. DBG 寄存器.....	110
表 3-130. DBG 库函数.....	110
表 3-131. 枚举类型 dbg_periph_enum.....	111
表 3-132. 函数 dbg_id_get.....	111
表 3-133. 函数 dbg_low_power_enable.....	112
表 3-134. 函数 dbg_low_power_disable.....	113
表 3-135. 函数 dbg_periph_enable.....	113
表 3-136. 函数 dbg_periph_disable.....	114
表 3-137. 函数 dbg_trace_pin_enable.....	115
表 3-138. 函数 dbg_trace_pin_disable.....	115
表 3-139. 函数 dbg_trace_pin_mode_set.....	116
表 3-140. DMA 寄存器.....	117
表 3-141. DMA 库函数.....	117
表 3-142. 结构体 dma_parameter_struct.....	118
表 3-143. 函数 dma_deinit.....	118
表 3-144. 函数 dma_deinit.....	119
表 3-145. 函数 dma_init.....	119
表 3-146. 函数 dma_circulation_enable.....	120
表 3-147. 函数 dma_circulation_disable.....	121
表 3-148. 函数 dma_memory_to_memory_enable.....	121
表 3-149. 函数 dma_memory_to_memory_disable.....	122
表 3-150. 函数 dma_channel_enable.....	123
表 3-151. 函数 dma_channel_disable.....	123
表 3-152. 函数 dma_periph_address_config.....	124
表 3-153. 函数 dma_memory_address_config.....	125
表 3-154. 函数 dma_transfer_number_config.....	125
表 3-155. 函数 dma_transfer_number_get.....	126
表 3-156. 函数 dma_priority_config.....	127
表 3-157. 函数 dma_memory_width_config.....	127
表 3-158. 函数 dma_periph_width_config.....	128
表 3-159. 函数 dma_memory_increase_enable.....	129
表 3-160. 函数 dma_memory_increase_disable.....	130
表 3-161. 函数 dma_periph_increase_enable.....	130
表 3-162. 函数 dma_periph_increase_disable.....	131
表 3-163. 函数 dma_transfer_direction_config.....	131
表 3-164. 函数 dma_flag_get.....	132
表 3-165. 函数 dma_flag_clear.....	133
表 3-166. 函数 dma_interrupt_flag_get.....	134
表 3-167. 函数 dma_interrupt_flag_clear.....	135
表 3-168. 函数 dma_interrupt_enable.....	135
表 3-169. 函数 dma_interrupt_disable.....	136
表 3-170. ENET 寄存器.....	137

表 3-171. ENET 库函数.....	139
表 3-172. 结构体 enet_descriptors_struct	142
表 3-173. 结构体 enet_ptp_systime_struct.....	142
表 3-174. 函数 enet_deinit.....	143
表 3-175. 函数 enet_initpara_config.....	143
表 3-176. 函数 enet_init.....	147
表 3-177. 函数 enet_software_reset.....	148
表 3-178. 函数 enet_rxframe_size_get.....	149
表 3-179. 函数 enet_descriptors_chain_init.....	149
表 3-180. 函数 enet_descriptors_ring_init.....	150
表 3-181. 函数 enet_frame_receive	150
表 3-182. 函数 enet_frame_transmit.....	151
表 3-183. 函数 enet_transmit_checksum_config	151
表 3-184. 函数 enet_enable	152
表 3-185. 函数 enet_disable	153
表 3-186. 函数 enet_mac_address_set	153
表 3-187. 函数 enet_mac_address_get.....	154
表 3-188. 函数 enet_flag_get.....	155
表 3-189. 函数 enet_flag_clear.....	157
表 3-190. 函数 enet_interrupt_enable.....	158
表 3-191. 函数 enet_interrupt_disable	160
表 3-192. 函数 enet_interrupt_flag_get.....	161
表 3-193. 函数 enet_interrupt_flag_clear	163
表 3-194. 函数 enet_tx_enable	164
表 3-195. 函数 enet_tx_disable.....	165
表 3-196. 函数 enet_rx_enable.....	165
表 3-197. 函数 enet_rx_disable.....	166
表 3-198. 函数 enet_registers_get.....	166
表 3-199. 函数 enet_address_filter_enable	167
表 3-200. 函数 enet_address_filter_disable	168
表 3-201. 函数 enet_address_filter_config.....	168
表 3-202. 函数 enet_phy_config.....	169
表 3-203. 函数 enet_phy_write_read.....	170
表 3-204. 函数 enet_phyloopback_enable	171
表 3-205. 函数 enet_phyloopback_disable	171
表 3-206. 函数 enet_forward_feature_enable	172
表 3-207. 函数 enet_forward_feature_disable	172
表 3-208. 函数 enet_filter_feature_enable	173
表 3-209. 函数 enet_filter_feature_disable	174
表 3-210. 函数 enet_pauseframe_generate	174
表 3-211. 函数 enet_pauseframe_detect_config.....	175
表 3-212. 函数 enet_pauseframe_config.....	176
表 3-213. 函数 enet_flowcontrol_threshold_config.....	176
表 3-214. 函数 enet_flowcontrol_feature_enable	178

表 3-215. 函数 enet_flowcontrol_feature_disable.....	179
表 3-216. 函数 enet_dmaprocess_state_get.....	179
表 3-217. 函数 enet_dmaprocess_resume.....	180
表 3-218. 函数 enet_rxprocess_check_recovery.....	181
表 3-219. 函数 enet_txfifo_flush.....	181
表 3-220. 函数 enet_current_desc_address_get.....	182
表 3-221. 函数 enet_desc_information_get.....	182
表 3-222. 函数 enet_missed_frame_counter_get.....	183
表 3-223. 函数 enet_desc_flag_get.....	184
表 3-224. 函数 enet_desc_flag_set.....	186
表 3-225. 函数 enet_desc_flag_clear.....	187
表 3-226. 函数 enet_desc_receive_complete_bit_enable.....	188
表 3-227. 函数 enet_desc_receive_complete_bit_disable.....	189
表 3-228. 函数 enet_rxframe_drop.....	189
表 3-229. 函数 enet_dma_feature_enable.....	190
表 3-230. 函数 enet_dma_feature_disable.....	190
表 3-231. 函数 enet_ptp_normal_descriptors_chain_init.....	191
表 3-232. 函数 enet_ptp_normal_descriptors_ring_init.....	192
表 3-233. 函数 enet_ptpframe_receive_normal_mode.....	192
表 3-234. 函数 enet_ptpframe_transmit_normal_mode.....	193
表 3-235. 函数 enet_wum_filter_register_pointer_reset.....	194
表 3-236. 函数 enet_wum_filter_config.....	194
表 3-237. 函数 enet_wum_feature_enable.....	195
表 3-238. 函数 enet_wum_feature_disable.....	195
表 3-239. 函数 enet_msc_counters_reset.....	196
表 3-240. 函数 enet_msc_feature_enable.....	196
表 3-241. 函数 enet_msc_feature_disable.....	197
表 3-242. 函数 enet_msc_counters_get.....	198
表 3-243. 函数 enet_ptp_subsecond_2_nanosecond.....	199
表 3-244. 函数 enet_ptp_nanosecond_2_subsecond.....	199
表 3-245. 函数 enet_ptp_feature_enable.....	200
表 3-246. 函数 enet_ptp_feature_disable.....	200
表 3-247. 函数 enet_ptp_timestamp_function_config.....	201
表 3-248. 函数 enet_ptp_subsecond_increment_config.....	202
表 3-249. 函数 enet_ptp_timestamp_addend_config.....	202
表 3-250. 函数 enet_ptp_timestamp_update_config.....	202
表 3-251. 函数 enet_ptp_expected_time_config.....	203
表 3-252. 函数 enet_ptp_system_time_get.....	204
表 3-253. 函数 enet_ptp_start.....	204
表 3-254. 函数 enet_ptp_finecorrection_adjfreq.....	205
表 3-255. 函数 enet_ptp_coarsecorrection_systime_update.....	206
表 3-256. 函数 enet_ptp_finecorrection_settime.....	206
表 3-257. 函数 enet_ptp_flag_get.....	207
表 3-258. 函数 enet_initpara_reset.....	208

表 3-259. EXMC 寄存器.....	208
表 3-260. EXMC 库函数.....	209
表 3-261. 结构体 exmc_norsram_timing_parameter_struct.....	210
表 3-262. 结构体 exmc_norsram_parameter_struct.....	210
表 3-263. 结构体 exmc_nand_pccard_timing_parameter_struct.....	210
表 3-264. 结构体 exmc_nand_parameter_struct.....	211
表 3-265. 结构体 exmc_pccard_parameter_struct.....	211
表 3-266. 函数 exmc_norsram_deinit.....	211
表 3-267. 函数 exmc_norsram_struct_para_init.....	212
表 3-268. 函数 exmc_norsram_init.....	212
表 3-269. 函数 exmc_norsram_enable.....	214
表 3-270. 函数 exmc_norsram_disable.....	214
表 3-271. 函数 exmc_nand_deinit.....	215
表 3-272. 函数 exmc_nand_init.....	215
表 3-273. 函数 exmc_nand_struct_para_init.....	217
表 3-274. 函数 exmc_nand_enable.....	217
表 3-275. 函数 exmc_nand_disable.....	218
表 3-276. 函数 exmc_nand_ecc_config.....	218
表 3-277. 函数 exmc_ecc_get.....	219
表 3-278. 函数 exmc_pccard_deinit.....	219
表 3-279. 函数 exmc_pccard_init.....	220
表 3-280. 函数 exmc_pccard_struct_para_init.....	221
表 3-281. 函数 exmc_pccard_enable.....	221
表 3-282. 函数 exmc_pccard_disable.....	222
表 3-283. 函数 exmc_flag_get.....	222
表 3-284. 函数 exmc_flag_clear.....	223
表 3-285. 函数 exmc_interrupt_enable.....	224
表 3-286. 函数 exmc_interrupt_disable.....	225
表 3-287. 函数 exmc_interrupt_flag_get.....	226
表 3-288. 函数 exmc_interrupt_flag_clear.....	226
表 3-289. EXTI 寄存器.....	227
表 3-290. EXTI 库函数.....	228
表 3-291. 枚举类型 exti_line_enum.....	228
表 3-292. 枚举类型 exti_mode_enum.....	229
表 3-293. 枚举类型 exti_trig_type_enum.....	229
表 3-294. 函数 exti_deinit.....	229
表 3-295. 函数 exti_init.....	230
表 3-296. 函数 exti_interrupt_enable.....	230
表 3-297. 函数 exti_interrupt_disable.....	231
表 3-298. 函数 exti_event_enable.....	231
表 3-299. 函数 exti_event_disable.....	232
表 3-300. 函数 exti_software_interrupt_enable.....	232
表 3-301. 函数 exti_software_interrupt_disable.....	233
表 3-302. 函数 exti_flag_get.....	233

表 3-303. 函数 exti_flag_clear	234
表 3-304. 函数 exti_interrupt_flag_get	234
表 3-305. 函数 exti_interrupt_flag_clear	235
表 3-306. FMC 寄存器	235
表 3-307. FMC 固件库函数	236
表 3-308. 枚举类型 fmc_state_enum	237
表 3-309. 枚举类型 fmc_int_enum	237
表 3-310. 枚举类型 fmc_flag_enum	237
表 3-311. 枚举类型 fmc_interrupt_flag_enum	238
表 3-312. 函数 fmc_wsctl_set	238
表 3-313. 函数 fmc_unlock	239
表 3-314. 函数 fmc_bank0_unlock	239
表 3-315. 函数 fmc_bank1_unlock	240
表 3-316. 函数 fmc_lock	240
表 3-317. 函数 fmc_bank0_lock	241
表 3-318. 函数 fmc_bank1_lock	241
表 3-319. 函数 fmc_page_erase	242
表 3-320. 函数 fmc_mass_erase	242
表 3-321. 函数 fmc_bank0_erase	243
表 3-322. 函数 fmc_bank1_erase	243
表 3-323. 函数 fmc_page_program	244
表 3-324. 函数 fmc_halfword_program	244
表 3-325. 函数 ob_unlock	245
表 3-326. 函数 ob_lock	245
表 3-327. 函数 ob_erase	246
表 3-328. 函数 ob_write_protection_enable	246
表 3-329. 函数 ob_security_protection_config	247
表 3-330. 函数 ob_user_write	247
表 3-331. 函数 ob_data_program	248
表 3-332. 函数 ob_user_get	249
表 3-333. 函数 ob_data_get	249
表 3-334. 函数 ob_write_protection_get	250
表 3-335. 函数 ob_spc_get	250
表 3-336. 函数 fmc_interrupt_enable	251
表 3-337. 函数 fmc_interrupt_disable	251
表 3-338. 函数 fmc_flag_get	252
表 3-339. 函数 fmc_flag_clear	253
表 3-340. 函数 fmc_interrupt_flag_get	254
表 3-341. 函数 fmc_interrupt_flag_clear	254
表 3-342. 函数 fmc_bank0_state_get	255
表 3-343. 函数 fmc_bank1_state_get	256
表 3-344. 函数 fmc_bank0_ready_wait	256
表 3-345. 函数 fmc_bank0_ready_wait	257
表 3-346. FWDGT 寄存器	257

表 3-347. FWDGT 库函数.....	258
表 3-348. 函数 fwdgt_write_enable	258
表 3-349. 函数 fwdgt_write_disable	258
表 3-350. 函数 fwdgt_enable	259
表 3-351. 函数 fwdgt_prescaler_value_config	259
表 3-352. 函数 fwdgt_reload_value_config	260
表 3-353. 函数 fwdgt_counter_reload.....	261
表 3-354. 函数 fwdgt_config.....	261
表 3-355. 函数 fwdgt_flag_get.....	262
表 3-356. GPIO 寄存器.....	263
表 3-357. GPIO 库函数.....	263
表 3-358. 函数 gpio_deinit.....	264
表 3-359. 函数 gpio_afio_deinit.....	264
表 3-360. 函数 gpio_init.....	265
表 3-361. 函数 gpio_bit_set.....	266
表 3-362. 函数 gpio_bit_reset.....	267
表 3-363. 函数 gpio_bit_write.....	267
表 3-364. 函数 gpio_port_write.....	268
表 3-365. 函数 gpio_input_bit_get.....	268
表 3-366. 函数 gpio_input_port_get.....	269
表 3-367. 函数 gpio_output_bit_get.....	270
表 3-368. 函数 gpio_output_port_get.....	270
表 3-369. 函数 gpio_pin_remap_config.....	271
表 3-370. 函数 gpio_exti_source_select.....	273
表 3-371. 函数 gpio_event_output_config.....	274
表 3-372. 函数 gpio_event_output_enable	274
表 3-373. 函数 gpio_event_output_disable	275
表 3-374. 函数 gpio_pin_lock.....	275
表 3-375. 函数 gpio_ethernet_phy_select.....	276
表 3-376. I2C 寄存器.....	276
表 3-377. I2C 库函数.....	277
表 3-378. 枚举 i2c_flag_enum	278
表 3-379. 枚举 i2c_interrupt_enum	278
表 3-380. 枚举 i2c_interrupt_flag_enum.....	278
表 3-381. 函数 i2c_deinit.....	279
表 3-382. 函数 i2c_clock_config.....	279
表 3-383. 函数 i2c_mode_addr_config.....	280
表 3-384. 函数 i2c_smbus_type_config.....	281
表 3-385. 函数 i2c_ack_config	282
表 3-386. 函数 i2c_ackpos_config.....	282
表 3-387. 函数 i2c_master_addressing.....	283
表 3-388. 函数 i2c_dualaddr_enable.....	284
表 3-389. 函数 i2c_dualaddr_disable.....	284
表 3-390. 函数 i2c_enable.....	285

表 3-391. 函数 i2c_disable.....	285
表 3-392. 函数 i2c_start_on_bus.....	286
表 3-393. 函数 i2c_stop_on_bus.....	286
表 3-394. 函数 i2c_data_transmit.....	287
表 3-395. 函数 i2c_data_receive.....	287
表 3-396. 函数 i2c_dma_config.....	288
表 3-397. 函数 i2c_dma_last_transfer_config.....	288
表 3-398. 函数 i2c_stretch_scl_low_config.....	289
表 3-399. 函数 i2c_slave_response_to_gcall_config.....	290
表 3-400. 函数 i2c_software_reset_config.....	290
表 3-401. 函数 i2c_pec_config.....	291
表 3-402. 函数 i2c_pec_transfer_config.....	291
表 3-403. 函数 i2c_pec_value_get.....	292
表 3-404. 函数 i2c_smbus_issue_alert.....	293
表 3-405. 函数 i2c_smbus_arb_config.....	293
表 3-406. 函数 i2c_flag_get.....	294
表 3-407. 函数 i2c_flag_clear.....	295
表 3-408. 函数 i2c_interrupt_enable.....	296
表 3-409. 函数 i2c_interrupt_disable.....	297
表 3-410. 函数 i2c_interrupt_flag_get.....	297
表 3-411. 函数 i2c_interrupt_flag_clear.....	298
表 3-412. NVIC 寄存器.....	300
表 3-413. SysTick 寄存器.....	300
表 3-414. 枚举类型 IRQn_Type.....	301
表 3-415. MISC 库函数.....	303
表 3-416. 函数 nvic_priority_group_set.....	304
表 3-417. 函数 nvic_irq_enable.....	304
表 3-418. 函数 nvic_irq_disable.....	305
表 3-419. 函数 nvic_vector_table_set.....	305
表 3-420. 函数 system_lowpower_set.....	306
表 3-421. 函数 system_lowpower_reset.....	307
表 3-422. 函数 systick_clksource_set.....	307
表 3-423. PMU 寄存器.....	308
表 3-424. PMU 库函数.....	308
表 3-425. 函数 pmu_deinit.....	309
表 3-426. 函数 pmu_lvd_select.....	309
表 3-427. 函数 pmu_lvd_disable.....	310
表 3-428. 函数 pmu_to_sleepmode.....	310
表 3-429. 函数 pmu_to_deepsleepmode.....	311
表 3-430. 函数 pmu_to_standbymode.....	311
表 3-431. 函数 pmu_wakeup_pin_enable.....	312
表 3-432. 函数 pmu_wakeup_pin_disable.....	312
表 3-433. 函数 pmu_backup_write_enable.....	313
表 3-434. 函数 pmu_backup_write_disable.....	313

表 3-435. 函数 pmu_flag_get.....	314
表 3-436. 函数 pmu_flag_clear.....	314
表 3-437. RCU 寄存器（中密度、高密度、超高密度产品）	315
表 3-438. RCU 寄存器（互联型产品）	316
表 3-439. RCU 库函数.....	316
表 3-440. 枚举类型 rcu_periph_enum.....	317
表 3-441. 枚举类型 rcu_periph_sleep_enum.....	318
表 3-442. 枚举类型 rcu_periph_reset_enum.....	319
表 3-443. 枚举类型 rcu_flag_enum.....	320
表 3-444. 枚举类型 rcu_int_flag_enum.....	320
表 3-445. 枚举类型 rcu_int_flag_clear_enum.....	321
表 3-446. 枚举类型 rcu_int_enum	321
表 3-447. 枚举类型 rcu_osci_type_enum	321
表 3-448. 枚举类型 rcu_clock_freq_enum.....	322
表 3-449. 函数 rcu_deinit.....	322
表 3-450. 函数 rcu_periph_clock_enable	322
表 3-451. 函数 rcu_periph_clock_disable	323
表 3-452. 函数 rcu_periph_clock_sleep_enable	323
表 3-453. 函数 rcu_periph_clock_sleep_disable	324
表 3-454. 函数 rcu_periph_reset_enable.....	324
表 3-455. 函数 rcu_periph_reset_disable.....	325
表 3-456. 函数 rcu_bkp_reset_enable	325
表 3-457. 函数 rcu_bkp_reset_disable	326
表 3-458. 函数 rcu_system_clock_source_config	326
表 3-459. 函数 rcu_system_clock_source_get.....	327
表 3-460. 函数 rcu_ahb_clock_config.....	327
表 3-461. 函数 rcu_apb1_clock_config.....	328
表 3-462. 函数 rcu_apb2_clock_config.....	329
表 3-463. 函数 rcu_ckout0_config.....	329
表 3-464. 函数 rcu_pll_config.....	330
表 3-465. 函数 rcu_predv0_config.....	331
表 3-466. 函数 rcu_predv0_config.....	331
表 3-467. 函数 rcu_predv1_config.....	332
表 3-468. 函数 rcu_pll1_config.....	333
表 3-469. 函数 rcu_pll2_config.....	333
表 3-470. 函数 rcu_adc_clock_config	334
表 3-471. 函数 rcu_usb_clock_config.....	334
表 3-472. 函数 rcu_rtc_clock_config	335
表 3-473. 函数 rcu_i2s1_clock_config	336
表 3-474. 函数 rcu_i2s2_clock_config	336
表 3-475. 函数 rcu_osci_stab_wait.....	337
表 3-476. 函数 rcu_osci_on.....	337
表 3-477. 函数 rcu_osci_off	338
表 3-478. 函数 rcu_osci_bypass_mode_enable	338

表 3-479. 函数 rcu_osci_bypass_mode_disable	339
表 3-480. 函数 rcu_hxtal_clock_monitor_enable	339
表 3-481. 函数 rcu_hxtal_clock_monitor_disable	340
表 3-482. 函数 rcu_irc8m_adjust_value_set	340
表 3-483. 函数 rcu_deepsleep_voltage_set	341
表 3-484. 函数 rcu_clock_freq_get	341
表 3-485. 函数 rcu_flag_get	342
表 3-486. 函数 rcu_all_reset_flag_clear	343
表 3-487. 函数 rcu_interrupt_flag_get	343
表 3-488. 函数 rcu_interrupt_flag_clear	344
表 3-489. 函数 rcu_interrupt_enable	344
表 3-490. 函数 rcu_interrupt_disable	345
表 3-491. RTC 寄存器	345
表 3-492. RTC 库函数	346
表 3-493. 函数 rtc_configuration_mode_enter	346
表 3-494. 函数 rtc_configuration_mode_exit	347
表 3-495. 函数 rtc_counter_set	347
表 3-496. 函数 rtc_prescaler_set	348
表 3-497. 函数 rtc_lwoff_wait	348
表 3-498. 函数 rtc_register_sync_wait	349
表 3-499. 函数 rtc_alarm_config	349
表 3-500. 函数 rtc_counter_get	350
表 3-501. 函数 rtc_divider_get	350
表 3-502. 函数 rtc_flag_get	351
表 3-503. 函数 rtc_flag_clear	351
表 3-504. 函数 rtc_interrupt_flag_get	352
表 3-505. 函数 rtc_interrupt_flag_clear	353
表 3-506. 函数 rtc_interrupt_enable	353
表 3-507. 函数 rtc_interrupt_disable	354
表 3-508. SDIO 寄存器	355
表 3-509. SDIO 库函数	355
表 3-510. 函数 sdio_deinit	357
表 3-511. 函数 sdio_clock_config	357
表 3-512. 函数 sdio_hardware_clock_enable	358
表 3-513. 函数 sdio_hardware_clock_disable	359
表 3-514. 函数 sdio_bus_mode_set	359
表 3-515. 函数 sdio_power_state_set	360
表 3-516. 函数 sdio_power_state_get	360
表 3-517. 函数 sdio_clock_enable	361
表 3-518. 函数 sdio_clock_disable	361
表 3-519. 函数 sdio_command_response_config	362
表 3-520. 函数 sdio_wait_type_set	363
表 3-521. 函数 sdio_csm_enable	363
表 3-522. 函数 sdio_csm_disable	364

表 3-523. 函数 sdio_command_index_get.....	364
表 3-524. 函数 sdio_response_get.....	365
表 3-525. 函数 sdio_data_config.....	365
表 3-526. 函数 sdio_data_transfer_config.....	367
表 3-527. 函数 sdio_dsm_enable	367
表 3-528. 函数 sdio_dsm_disable	368
表 3-529. 函数 sdio_data_write.....	368
表 3-530. 函数 sdio_data_read.....	369
表 3-531. 函数 sdio_data_counter_get.....	369
表 3-532. 函数 sdio_data_counter_get.....	370
表 3-533. 函数 sdio_dma_enable	370
表 3-534. 函数 sdio_dma_disable	371
表 3-535. 函数 sdio_flag_get.....	371
表 3-536. 函数 sdio_flag_clear.....	373
表 3-537. 函数 sdio_interrupt_enable.....	374
表 3-538. 函数 sdio_interrupt_disable	375
表 3-539. 函数 sdio_interrupt_flag_get.....	376
表 3-540. 函数 sdio_interrupt_flag_clear	378
表 3-541. 函数 sdio_readwait_enable.....	379
表 3-542. 函数 sdio_readwait_disable.....	379
表 3-543. 函数 sdio_stop_readwait_enable.....	380
表 3-544. 函数 sdio_stop_readwait_disable.....	380
表 3-545. 函数 sdio_readwait_type_set.....	381
表 3-546. 函数 sdio_operation_enable.....	381
表 3-547. 函数 sdio_operation_disable.....	382
表 3-548. 函数 sdio_suspend_enable.....	382
表 3-549. 函数 sdio_suspend_disable	383
表 3-550. 函数 sdio_ceata_command_enable	383
表 3-551. 函数 sdio_ceata_command_disable	384
表 3-552. 函数 sdio_ceata_interrupt_enable.....	384
表 3-553. 函数 sdio_ceata_interrupt_disable	385
表 3-554. 函数 sdio_ceata_command_completion_enable.....	385
表 3-555. 函数 sdio_ceata_command_completion_disable	386
表 3-556. SPI/I2S 寄存器.....	386
表 3-557. SPI/I2S 库函数.....	387
表 3-558. 结构体 spi_parameter_struct.....	387
表 3-559. 函数 spi_i2s_deinit.....	388
表 3-560. 函数 spi_struct_para_init.....	388
表 3-561. 函数 spi_init.....	389
表 3-562. 函数 spi_enable.....	390
表 3-563. 函数 spi_disable	390
表 3-564. 函数 i2s_init.....	391
表 3-565. 函数 i2s_psc_config.....	392
表 3-566. 函数 i2s_enable.....	393

表 3-567. 函数 i2s_disable.....	394
表 3-568. 函数 spi_nss_output_enable.....	394
表 3-569. 函数 spi_nss_output_disable.....	395
表 3-570. 函数 spi_nss_internal_high.....	395
表 3-571. 函数 spi_nss_internal_low.....	396
表 3-572. 函数 spi_dma_enable.....	396
表 3-573. 函数 spi_dma_disable.....	397
表 3-574. 函数 spi_i2s_data_frame_format_config.....	398
表 3-575. 函数 spi_bidirectional_transfer_config.....	398
表 3-576. 函数 spi_i2s_data_transmit.....	399
表 3-577. 函数 spi_i2s_data_receive.....	400
表 3-578. 函数 spi_crc_polynomial_set.....	400
表 3-579. 函数 spi_crc_polynomial_get.....	401
表 3-580. 函数 spi_crc_on.....	401
表 3-581. 函数 spi_crc_off.....	402
表 3-582. 函数 spi_crc_next.....	402
表 3-583. 函数 spi_crc_get.....	403
表 3-584. 函数 spi_i2s_flag_get.....	403
表 3-585. 函数 spi_i2s_interrupt_enable.....	404
表 3-586. 函数 spi_i2s_interrupt_disable.....	405
表 3-587. 函数 spi_i2s_interrupt_flag_get.....	405
表 3-588. 函数 spi_crc_error_clear.....	406
表 3-589. TIMER 寄存器.....	407
表 3-590. TIMER 库函数.....	408
表 3-591. 结构体 timer_parameter_struct.....	410
表 3-592. 结构体 timer_break_parameter_struct.....	410
表 3-593. 结构体 timer_oc_parameter_struct.....	411
表 3-594. 结构体 timer_ic_parameter_struct.....	411
表 3-595. 函数 timer_deinit.....	411
表 3-596. 函数 timer_struct_para_init.....	412
表 3-597. 函数 timer_init.....	413
表 3-598. 函数 timer_enable.....	413
表 3-599. 函数 timer_disable.....	414
表 3-600. 函数 timer_auto_reload_shadow_enable.....	414
表 3-601. 函数 timer_auto_reload_shadow_disable.....	415
表 3-602. 函数 timer_update_event_enable.....	415
表 3-603. 函数 timer_update_event_disable.....	416
表 3-604. 函数 timer_counter_alignment.....	416
表 3-605. 函数 timer_counter_up_direction.....	417
表 3-606. 函数 timer_counter_down_direction.....	418
表 3-607. 函数 timer_prescaler_config.....	418
表 3-608. 函数 timer_repetition_value_config.....	419
表 3-609. 函数 timer_autoreload_value_config.....	419
表 3-610. 函数 timer_counter_value_config.....	420

表 3-611. 函数 timer_counter_read	421
表 3-612. 函数 timer_prescaler_read	421
表 3-613. 函数 timer_single_pulse_mode_config	422
表 3-614. 函数 timer_update_source_config	422
表 3-615. 函数 timer_dma_enable	423
表 3-616. 函数 timer_dma_disable	424
表 3-617. 函数 timer_channel_dma_request_source_select	425
表 3-618. 函数 timer_dma_transfer_config	425
表 3-619. 函数 timer_event_software_generate	427
表 3-620. 函数 timer_break_struct_para_init	428
表 3-621. 函数 timer_break_config	429
表 3-622. 函数 timer_break_enable	430
表 3-623. 函数 timer_break_disable	430
表 3-624. 函数 timer_automatic_output_enable	431
表 3-625. 函数 timer_automatic_output_disable	431
表 3-626. 函数 timer_primary_output_config	432
表 3-627. 函数 timer_channel_control_shadow_config	432
表 3-628. 函数 timer_channel_control_shadow_update_config	433
表 3-629. 函数 timer_channel_output_struct_para_init	434
表 3-630. 函数 timer_channel_output_config	434
表 3-631. 函数 timer_channel_output_mode_config	435
表 3-632. 函数 timer_channel_output_pulse_value_config	436
表 3-633. 函数 timer_channel_output_shadow_config	437
表 3-634. 函数 timer_channel_output_fast_config	438
表 3-635. 函数 timer_channel_output_clear_config	439
表 3-636. 函数 timer_channel_output_polarity_config	440
表 3-637. 函数 timer_channel_complementary_output_polarity_config	440
表 3-638. 函数 timer_channel_output_state_config	441
表 3-639. 函数 timer_channel_complementary_output_state_config	442
表 3-640. 函数 timer_channel_input_struct_para_init	443
表 3-641. 函数 timer_input_capture_config	443
表 3-642. 函数 timer_channel_input_capture_prescaler_config	444
表 3-643. 函数 timer_channel_capture_value_register_read	445
表 3-644. 函数 timer_input_pwm_capture_config	446
表 3-645. 函数 timer_hall_mode_config	447
表 3-646. 函数 timer_input_trigger_source_select	447
表 3-647. 函数 timer_master_output_trigger_source_select	448
表 3-648. 函数 timer_slave_mode_select	449
表 3-649. 函数 timer_master_slave_mode_config	450
表 3-650. 函数 timer_external_trigger_config	451
表 3-651. 函数 timer_quadrature_decoder_mode_config	452
表 3-652. 函数 timer_internal_clock_config	453
表 3-653. 函数 timer_internal_trigger_as_external_clock_config	454
表 3-654. 函数 timer_external_trigger_as_external_clock_config	454

表 3-655. 函数 timer_external_clock_mode0_config.....	455
表 3-656. 函数 timer_external_clock_mode1_config.....	456
表 3-657. 函数 timer_external_clock_mode1_disable	457
表 3-658. 函数 timer_interrupt_enable	458
表 3-659. 函数 timer_interrupt_disable.....	459
表 3-660. 函数 timer_interrupt_flag_get.....	459
表 3-661. 函数 timer_interrupt_flag_clear.....	460
表 3-662. 函数 timer_flag_get.....	461
表 3-663. 函数 timer_flag_clear.....	462
表 3-664. USART 寄存器.....	463
表 3-665. USART 库函数.....	464
表 3-666. 函数 usart_deinit.....	465
表 3-667. 函数 usart_baudrate_set.....	465
表 3-668. 函数 usart_parity_config	466
表 3-669. 函数 usart_word_length_set.....	467
表 3-670. 函数 usart_stop_bit_set.....	467
表 3-671. 函数 usart_enable	468
表 3-672. 函数 usart_disable	468
表 3-673. 函数 usart_transmit_config	469
表 3-674. 函数 usart_receive_config	470
表 3-675. 函数 usart_data_transmit.....	470
表 3-676. 函数 usart_data_receive	471
表 3-677. 函数 usart_address_config.....	471
表 3-678. 函数 usart_mute_mode_enable.....	472
表 3-679. 函数 usart_mute_mode_disable.....	473
表 3-680. 函数 usart_mute_mode_wakeup_config.....	473
表 3-681. 函数 usart_lin_mode_enable.....	474
表 3-682. 函数 usart_lin_mode_disable.....	474
表 3-683. 函数 usart_lin_break_dection_length_config.....	475
表 3-684. 函数 usart_send_break.....	476
表 3-685. 函数 usart_halfduplex_enable	476
表 3-686. 函数 usart_halfduplex_disable	477
表 3-687. 函数 usart_synchronous_clock_enable	477
表 3-688. 函数 usart_synchronous_clock_disable.....	478
表 3-689. 函数 usart_synchronous_clock_config.....	478
表 3-690. 函数 usart_guard_time_config	479
表 3-691. 函数 usart_smartcard_mode_enable.....	480
表 3-692. 函数 usart_smartcard_mode_disable	480
表 3-693. 函数 usart_smartcard_mode_nack_enable.....	481
表 3-694. 函数 usart_smartcard_mode_nack_disable.....	481
表 3-695. 函数 usart_irda_mode_enable	482
表 3-696. 函数 usart_irda_mode_disable	482
表 3-697. 函数 usart_prescaler_config	483
表 3-698. 函数 usart_irda_lowpower_config.....	483

表 3-699. 函数 usart_hardware_flow_rts_config.....	484
表 3-700. 函数 usart_hardware_flow_cts_config.....	485
表 3-701. 函数 usart_dma_receive_config.....	485
表 3-702. 函数 usart_dma_transmit_config.....	486
表 3-703. 函数 usart_flag_get.....	487
表 3-704. 函数 usart_flag_clear.....	488
表 3-705. 函数 usart_interrupt_enable.....	488
表 3-706. 函数 usart_interrupt_disable.....	489
表 3-707. 函数 usart_interrupt_flag_get.....	490
表 3-708. 函数 usart_interrupt_flag_clear.....	491
表 3-709. WWDGT 寄存器.....	492
表 3-710. WWDGT 库函数.....	492
表 3-711. 函数 wwdgt_deinit.....	492
表 3-712. 函数 wwdgt_enable.....	493
表 3-713. 函数 wwdgt_counter_update.....	493
表 3-714. 函数 wwdgt_config.....	494
表 3-715. 函数 wwdgt_interrupt_enable.....	495
表 3-716. 函数 wwdgt_flag_get.....	495
表 3-717. 函数 wwdgt_flag_clear.....	496
表 4-1. 版本历史.....	497

1. 介绍

本手册介绍了32位基于ARM微控制器GD32F10x固件库。

本手册介绍了32位基于ARM微控制器GD32F10x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GD32F10x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
CAN	局域网控制器模块
CRC	循环冗余校验计算单元
DAC	数模转换器
DBG	调试模块
DMA	直接存储器访问控制器
ENET	以太网控制器模块
EXMC	外部存储器控制器
EXTI	外部中断事件控制器

外设缩写	说明
FMC	闪存控制器
FWDGT	独立看门狗
GPIO/AFIO	通用和备用输入/输出接口
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
SDIO	SDIO接口
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
USB_D	通用串行总线全速设备接口
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：

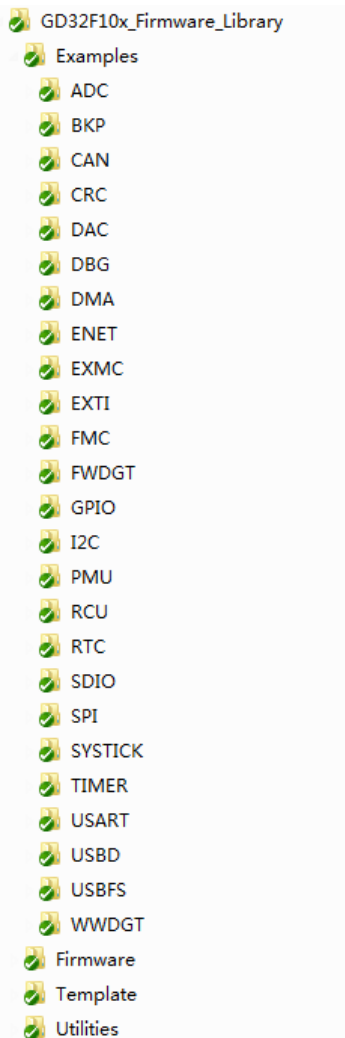
- XXX表示任一外设缩写，例如：ADC。更多缩写相关信息参阅[外设缩写](#)；
- 源文件和头文件命名都以“gd32f10x_”作为开头，例如：gd32f10x_adc.h；
- 常量仅被应用于一个文件的，定义于该文件中；被应用于多个文件的，在对应头文件中定义。所有常量都由英文字母大写书写；
- 寄存器作为常量处理。他们的命名都由英文字母大写书写。在大多数情况下，寄存器缩写规范与本用户手册一致；
- 变量名采用全部小写，有多个单词组成的，在单词之间以下划线分隔；
- 外设函数的命名以该外设的缩写加下划线为开头，有多个单词组成的，在单词之间以下划线分隔，所有外设函数都由英文字母小写书写。

2. 固件库概述

2.1. 文件组织结构

GD32F10x_Firmware_Library，文件组织结构见下图：

图 2-1. GD32F10x 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；
- **gd32f10x_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）；
- **gd32f10x_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）；

- `gd32f10x.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 M3内核的支持文件、基于Cortex M3内核处理器的启动代码和库引导文件以及基于GD32F10x的全局头文件和系统配置文件;
- GD32F10x_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;
- GD32F10x_usbd_driver子文件夹包含了关于USB D外设的所有文件;
 - Include子文件夹包含了USB D外设所需的头文件, 用户无需修改该文件夹;
 - Source子文件夹包含了USB D外设所需的源文件, 用户无需修改该文件夹;
- GD32F10x_usbfs_driver子文件夹包含了关于USB FS外设的所有文件;
 - Include子文件夹包含了USB FS外设所需的头文件, 用户无需修改该文件夹;
 - Source子文件夹包含了USB FS外设所需的源文件, 用户无需修改该文件夹;

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

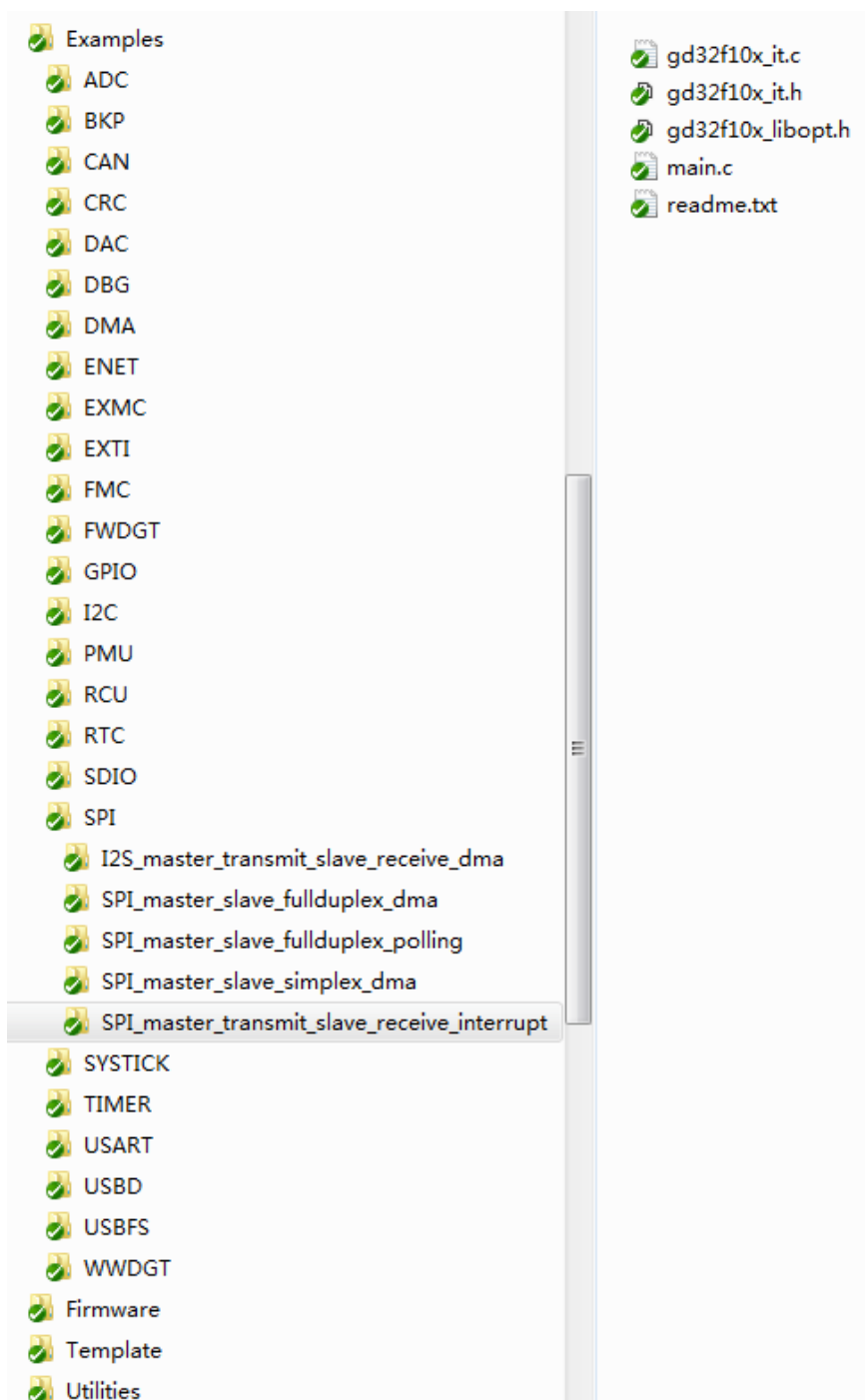
2.1.3. Template 文件夹

Template文件夹包含一个关于使用LED、USART打印、按键控制的简单例程, (IAR_project用于IAR编译环境, Keil_project5用于Keil5编译环境, Keil_project4用于Keil4编译环境)。用户可以使用该工程模板进行固件库例程的移植编译, 具体使用方法见下:

选择文件

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

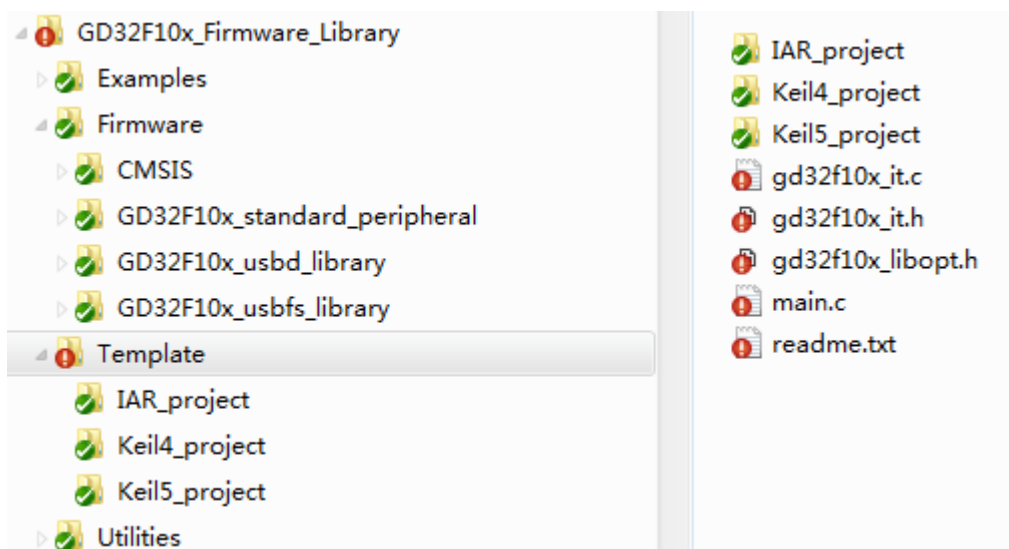
图 2-2. 选择外设例程文件



拷贝文件

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

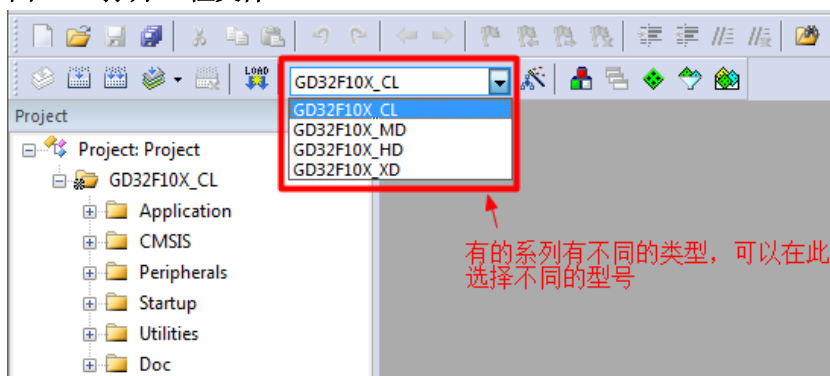
图 2-3. 拷贝外设例程文件



打开工程

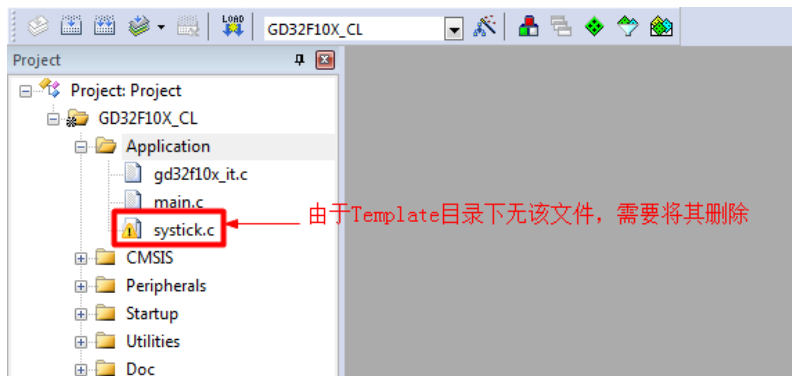
GD 提供 Keil 和 IAR 两种版本的工程，根据客户所安装的软件，打开不同的 project，如“Keil5_project”，打开\Template\Keil5_project\Project.uvprojx，如下图所示：

图 2-4. 打开工程文件



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

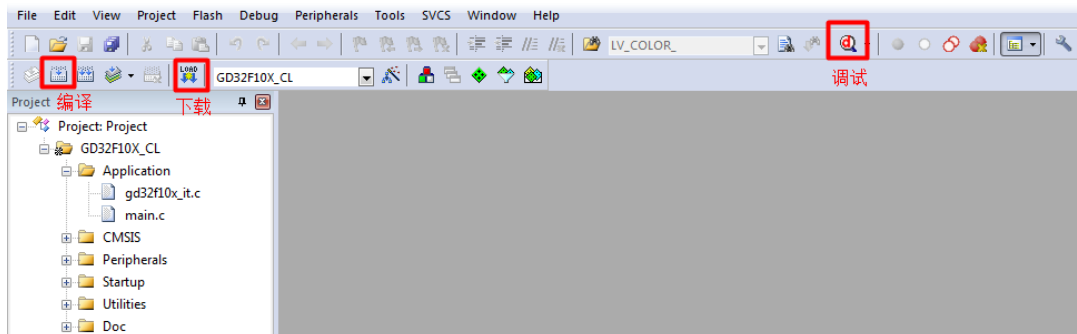
图 2-5. 配置工程文件



编译调试下载

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

图 2-6. 编译调试下载



2.1.4. Utilities 文件夹

Utilities文件夹包含运行固件库例程评估板的文件：

- Binary、LCD_Commom及Third_Party子文件夹包含有USB测试所需文件；
- gd32f10x_eval.h及gd32f10x_lcd_eval.h文件是运行固件库例程所需关于评估板的头文件；
- gd32f10x_eval.c及gd32f10x_lcd_eval.c文件是运行固件库例程所需关于评估板的源文件。

注意：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
gd32f10x_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。
gd32f10x_it.h	头文件，包含所有中断处理函数原形。
gd32f10x_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。
gd32f10x_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。
gd32f10x_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
ADC_RSQ1	规则序列寄存器1

寄存器名称	寄存器描述
ADC_RSQ2	规则序列寄存器2
ADC_ISQ	注入序列寄存器
ADC_IDATAx	注入数据寄存器x
ADC_RDATA	规则数据寄存器

3.2.2. 外设库函数说明

ADC库函数列表如下表所示:

表 3-3. ADC 库函数

库函数名称	库函数描述
adc_deinit	复位ADC外设
adc_mode_config	配置ADC模式
adc_special_function_config	使能或除能ADC特殊功能
adc_data_alignment_config	配置ADC数据对齐方式
adc_enable	ADC使能
adc_disable	ADC禁能
adc_calibration_enable	使能ADC校准功能
adc_tempsensor_vrefint_enable	温度传感器和Vrefint通道使能
adc_tempsensor_vrefint_disable	温度传感器和Vrefint通道禁能
adc_dma_mode_enable	ADC DMA 请求使能
adc_dma_mode_disable	ADC DMA 请求除能
adc_discontinuous_mode_config	配置ADC非连续模式
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_enable	配置ADC模拟看门狗单通道有效
adc_watchdog_group_channel_enable	配置ADC模拟看门狗在通道组有效
adc_watchdog_disable	ADC模拟看门狗禁能
adc_watchdog_threshold_config	配置ADC模拟看门狗阈值
adc_flag_get	获取ADC中断标志位
adc_flag_clear	清除ADC中断标志位
adc_regular_software_startconv_flag_	获取ADC规则通道组软件触发转换开始位

库函数名称	库函数描述
get	
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中断除能

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ADC外设
先决条件	-
被调用函数	rcu_periph_reset_enable / rcu_periph_reset_disable
输入参数{in}	
adc_periph	ADC外设
ADCx	x=0,1,2
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_DUAL_REGULAR_PARALLEL_INSERTED_PARALLEL	ADC0和ADC1运行在规则并行+注入并行组合模式
ADC_DUAL_REGULAR_INSERTED_ROTATION	ADC0和ADC1运行在规则并行+交替触发组合模式
ADC_DUAL_INSERTED_PARALLEL_REGULAR_FOLLOWUP_FAST	ADC0和ADC1运行在注入并行+快速交叉组合模式
ADC_DUAL_INSERTED_PARALLEL_REGULAR_FOLLOWUP_SLOW	ADC0和ADC1运行在注入并行+慢速交叉组合模式
ADC_DUAL_INSERTED_PARALLEL	ADC0和ADC1运行在注入并行模式
ADC_DUAL_REGULAR_PARALLEL	ADC0和ADC1运行在规则并行模式
ADC_DUAL_REGULAR_FOLLOWUP_FAST	ADC0和ADC1运行在快速交叉模式
ADC_DUAL_REGULAR_FOLLOWUP_SLOW	ADC0和ADC1运行在慢速交叉模式
ADC_DUAL_INSERTED_TRIGGER_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	<code>adc_special_function_config</code>
------	--

函数原形	void adc_special_function_config(uint32_t adc_periph, uint32_t function, ControlStatus new_value);
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=0,1,2
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}	
new_value	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ADC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=0,1,2
输入参数{in}	

data_alignment	数据对齐方式选择
<i>ADC_DATAALIGN_RIGHT</i>	LSB对齐
<i>ADC_DATAALIGN_LEFT</i>	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ADC外设
先决条件	-
被调用函数	-
输入参数{in}	
adc_periph	ADC外设
<i>ADCx</i>	x=0,1,2
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ADC外设

先决条件	-
被调用函数	-
输入参数{in}	
adc_periph	ADC外设
ADCx	x=0,1,2
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	ADC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=0,1,2
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_dma_mode_enable**

函数adc_dma_mode_enable描述见下表：

表 3-13. 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	ADC 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=0,1,2
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 3-14. 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	ADC 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=0,1,2
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 3-15. 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=0,1,2
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 3-16. 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=0,1,2
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}	
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 3-17. 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=0,1,2
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	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	41.5周期

<code>_41POINT5</code>	
<code>ADC_SAMPLETIME_55POINT5</code>	55.5周期
<code>ADC_SAMPLETIME_71POINT5</code>	71.5周期
<code>ADC_SAMPLETIME_239POINT5</code>	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 3-18. 函数 `adc_inserted_channel_config`

函数名称	<code>adc_inserted_channel_config</code>
函数原形	<code>void adc_inserted_channel_config(uint32_t adc_periph, uint8_t rank, uint8_t adc_channel, uint32_t sample_time);</code>
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}	
<code>adc_periph</code>	ADC外设
<code>ADCx</code>	$x=0,1,2$
输入参数{in}	
<code>rank</code>	注入组通道序列，取值范围为0~3
输入参数{in}	
<code>adc_channel</code>	ADC通道选择
<code>ADC_CHANNEL_x</code>	ADC 通道 x ($x=0..17$)（只有ADC0，可取值 $x=16$ 和 17 ）
输入参数{in}	
<code>sample_time</code>	采样时间
<code>ADC_SAMPLETIME_1POINT5</code>	1.5周期
<code>ADC_SAMPLETIME_7POINT5</code>	7.5周期
<code>ADC_SAMPLETIME_13POINT5</code>	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 3-19. 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=0,1,2
输入参数{in}	
inserted_channel	注入通道选择
ADC_INSERTED_CHANNEL_x	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 3-20. 函数 `adc_external_trigger_source_config`

函数名称	<code>adc_external_trigger_source_config</code>
函数原形	<code>void adc_external_trigger_source_config(uint32_t adc_periph, uint8_t adc_channel_group, uint32_t external_trigger_source);</code>
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=0,1,2
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_REGULAR_T0_CH1	TIMER0 CH1事件（规则组）
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_CH 3	TIMER3 CH3事件（规则组）
ADC0_1_EXTTRIG _REGULAR_T7_TR GO	TIMER7 TRGO事件（规则组）
ADC0_1_EXTTRIG _REGULAR_EXTI_ 11	外部中断线11（规则组）
ADC2_EXTTRIG_R EGULAR_T2_CH0	TIMER2 CH0事件（规则组）
ADC2_EXTTRIG_R EGULAR_T1_CH2	TIMER1 CH2事件（规则组）
ADC2_EXTTRIG_R EGULAR_T0_CH2	TIMER0 CH2事件（规则组）
ADC2_EXTTRIG_R EGULAR_T7_CH0	TIMER7 CH0事件（规则组）
ADC2_EXTTRIG_R EGULAR_T7_TRG O	TIMER7 TRGO事件（规则组）
ADC2_EXTTRIG_R EGULAR_T4_CH0	TIMER4 CH0事件（规则组）
ADC2_EXTTRIG_R EGULAR_T4_CH2	TIMER4 CH2事件（规则组）
ADC0_1_2_EXTTRI G_REGULAR_NON E	软件触发（规则组）
ADC0_1_EXTTRIG _INSERTED_T0_T RGO	TIMER0 TRGO事件（注入组）
ADC0_1_EXTTRIG _INSERTED_T0_C H3	TIMER0 CH3事件（注入组）
ADC0_1_EXTTRIG _INSERTED_T1_T RGO	TIMER1 TRGO事件（注入组）
ADC0_1_EXTTRIG _INSERTED_T1_C H0	TIMER1 CH0事件（注入组）
ADC0_1_EXTTRIG _INSERTED_T2_C H3	TIMER2 CH3事件（注入组）
ADC0_1_EXTTRIG	TIMER3 TRGO事件（注入组）

_INSERTED_T3_TRGO	
ADC0_1_EXTTRIG_INSERTED_EXTI_15	外部中断线15（注入组）
ADC0_1_EXTTRIG_INSERTED_T7_CH3	TIMER7 CH3事件（注入组）
ADC2_EXTTRIG_INSERTED_T0_TRGO	TIMER0 TRGO事件（注入组）
ADC2_EXTTRIG_INSERTED_T0_CH3	TIMER0 CH3事件（注入组）
ADC2_EXTTRIG_INSERTED_T3_CH2	TIMER3 CH2事件（注入组）
ADC2_EXTTRIG_INSERTED_T7_CH1	TIMER7 CH1事件（注入组）
ADC2_EXTTRIG_INSERTED_T7_CH3	TIMER7 CH3事件（注入组）
ADC2_EXTTRIG_INSERTED_T4_TRGO	TIMER4 TRGO事件（注入组）
ADC2_EXTTRIG_INSERTED_T4_CH3	TIMER4 CH3事件（注入组）
ADC0_1_2_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 3-21. 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 new_value);
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=0,1,2
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}	
new_value	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 3-22. 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=0,1,2
输入参数{in}	

adc_channel_group	通道组选择
<i>ADC_REGULAR_CHANNEL</i>	规则通道组
<i>ADC_INSERTED_CHANNEL</i>	注入通道组
输出参数{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 3-23. 函数 `adc_regular_data_read`

函数名称	<code>adc_regular_data_read</code>
函数原形	<code>uint16_t adc_regular_data_read(uint32_t adc_periph);</code>
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
<i>ADCx</i>	x=0,1,2
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 3-24. 函数 `adc_inserted_data_read`

函数名称	<code>adc_inserted_data_read</code>
函数原形	<code>uint16_t adc_inserted_data_read(uint32_t adc_periph, uint8_t inserted_channel);</code>
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}	
<code>adc_periph</code>	ADC外设
<code>ADCx</code>	<code>x=0,1,2</code>
输入参数{in}	
<code>inserted_channel</code>	注入通道选择
<code>ADC_INSERTED_CHANNEL_x</code>	注入通道 <code>x</code> , <code>x=0,1,2,3</code>
输出参数{out}	
-	-
返回值	
<code>uint16_t</code>	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 3-25. 函数 `adc_sync_mode_convert_value_read`

函数名称	<code>adc_sync_mode_convert_value_read</code>
函数原形	<code>uint32_t adc_sync_mode_convert_value_read(void);</code>
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	
<code>uint32_t</code>	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 3-26. 函数 `adc_watchdog_single_channel_enable`

函数名称	<code>adc_watchdog_single_channel_enable</code>
函数原形	<code>void adc_watchdog_single_channel_enable(uint32_t adc_periph, uint8_t adc_channel);</code>
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}	
<code>adc_periph</code>	ADC外设
<code>ADCx</code>	<code>x=0,1,2</code>
输入参数{in}	
<code>adc_channel</code>	选择ADC通道
<code>ADC_CHANNEL_x</code>	ADC Channelx(<code>x=0..17</code>) (只有ADC0, 可取值 <code>x=16</code> 和 <code>17</code>)
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 3-27. 函数 `adc_watchdog_group_channel_enable`

函数名称	<code>adc_watchdog_group_channel_enable</code>
函数原形	<code>void adc_watchdog_group_channel_enable(uint32_t adc_periph, uint8_t adc_channel_group);</code>
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}	
<code>adc_periph</code>	ADC外设
<code>ADCx</code>	<code>x=0,1,2</code>
输入参数{in}	

adc_channel_group	通道组使用模拟看门狗
<i>ADC_REGULAR_CHANNEL</i>	规则通道组
<i>ADC_INSERTED_CHANNEL</i>	注入通道组
<i>ADC_REGULAR_INSERTED_CHANNEL</i>	规则和注入通道组
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* configure 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 3-28. 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
<i>ADCx</i>	x=0,1,2
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 3-29. 函数 `adc_watchdog_threshold_config`

函数名称	<code>adc_watchdog_threshold_config</code>
函数原形	<code>void adc_watchdog_threshold_config(uint32_t adc_periph, uint16_t low_threshold, uint16_t high_threshold);</code>
功能描述	配置ADC模拟看门狗阈值
先决条件	-
被调用函数	-
输入参数{in}	
<code>adc_periph</code>	ADC外设
<code>ADCx</code>	<code>x=0,1,2</code>
输入参数{in}	
<code>low_threshold</code>	模拟看门狗低阈值, 0..4095
输入参数{in}	
<code>high_threshold</code>	模拟看门狗高阈值, 0..4095
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* configure 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 3-30. 函数 `adc_flag_get`

函数名称	<code>adc_flag_get</code>
函数原形	<code>FlagStatus adc_flag_get(uint32_t adc_periph, uint32_t adc_flag);</code>
功能描述	获取ADC标志位
先决条件	-
被调用函数	-
输入参数{in}	
<code>adc_periph</code>	ADC外设
<code>ADCx</code>	<code>x=0,1,2</code>
输入参数{in}	
<code>adc_flag</code>	ADC标志位
<code>ADC_FLAG_WDE</code>	模拟看门狗事件标志位
<code>ADC_FLAG_EOC</code>	组转换结束标志位
<code>ADC_FLAG_EOIC</code>	注入通道组转换结束标志位
<code>ADC_FLAG_STIC</code>	注入通道组转换开始标志位
<code>ADC_FLAG_STRC</code>	规则通道组转换开始标志位

输出参数{out}	
-	-
返回值	
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 3-31. 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=0,1,2
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 3-32. 函数 `adc_regular_software_startconv_flag_get`

函数名称	<code>adc_regular_software_startconv_flag_get</code>
函数原形	<code>FlagStatus adc_regular_software_startconv_flag_get(uint32_t adc_periph);</code>
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}	
<code>adc_periph</code>	ADC外设
<code>ADCx</code>	<code>x=0,1,2</code>
输出参数{out}	
-	-
返回值	
<code>FlagStatus</code>	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 3-33. 函数 `adc_inserted_software_startconv_flag_get`

函数名称	<code>adc_inserted_software_startconv_flag_get</code>
函数原形	<code>FlagStatus adc_inserted_software_startconv_flag_get(uint32_t adc_periph);</code>
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}	
<code>adc_periph</code>	ADC外设
<code>ADCx</code>	<code>x=0,1,2</code>
输出参数{out}	
-	-
返回值	
<code>FlagStatus</code>	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 3-34. 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=0,1,2
输入参数{in}	
adc_interrupt	ADC中断标志位
ADC_INT_FLAG_WDE	模拟看门狗中断标志位
ADC_INT_FLAG_EOC	组转换结束中断标志位
ADC_INT_FLAG_EOIC	注入通道组转换结束中断标志位
输出参数{out}	
-	-
返回值	
FlagStatus	SET或RESET

例如：

```
/* get the ADC0 analog watchdog interrupt bits */
FlagStatus flag_value;

flag_value = adc_interrupt_flag_get(ADC0, ADC_INT_FLAG_WDE);
```

函数 adc_interrupt_flag_clear

函数adc_interrupt_flag_clear描述见下表：

表 3-35. 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=0,1,2
输入参数{in}	
adc_interrupt	ADC中断标志位
ADC_INT_FLAG_WDE	模拟看门狗中断标志位
ADC_INT_FLAG_EOC	组转换结束中断标志位
ADC_INT_FLAG_EOIC	注入通道组转换结束中断标志位
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* clear the ADC0 analog watchdog interrupt bits */
adc_interrupt_flag_clear(ADC0, ADC_INT_FLAG_WDE);
```

函数 adc_interrupt_enable

函数adc_interrupt_enable描述见下表：

表 3-36. 函数 adc_interrupt_enable

函数名称	adc_interrupt_enable
函数原形	void adc_interrupt_enable(uint32_t 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=0,1,2
输入参数{in}	
adc_interrupt	ADC中断
ADC_INT_WDE	模拟看门狗中断
ADC_INT_EOC	组转换结束中断
ADC_INT_EOIC	注入通道组转换结束中断
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 3-37. 函数 `adc_interrupt_disable`

函数名称	<code>adc_interrupt_disable</code>
函数原形	<code>void adc_interrupt_disable(uint32_t adc_periph, uint32_t adc_interrupt);</code>
功能描述	ADC中断禁能
先决条件	-
被调用函数	-
输入参数{in}	
<code>adc_periph</code>	ADC外设
<code>ADCx</code>	$x=0,1,2$
输入参数{in}	
<code>adc_interrupt</code>	ADC中断标志位
<code>ADC_INT_WDE</code>	模拟看门狗中断标志位
<code>ADC_INT_EOC</code>	组转换结束中断标志位
<code>ADC_INT_EOIC</code>	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 3-38. BKP 寄存器

寄存器名称	寄存器描述
<code>BKP_DATAx</code> ($x=0..41$)	备份数据寄存器

寄存器名称	寄存器描述
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 3-39. BKP 库函数

库函数名称	库函数描述
bkp_deinit	复位备份寄存器
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_calibration_value_set	设置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

枚举类型 bkp_data_register_enum

表 3-40. 枚举类型 bkp_data_register_enum

成员名称	功能描述
BKP_DATA_0	BKP数据寄存器0
BKP_DATA_1	BKP数据寄存器1
BKP_DATA_2	BKP数据寄存器2
BKP_DATA_3	BKP数据寄存器3
BKP_DATA_4	BKP数据寄存器4
BKP_DATA_5	BKP数据寄存器5
BKP_DATA_6	BKP数据寄存器6
BKP_DATA_7	BKP数据寄存器7
BKP_DATA_8	BKP数据寄存器8

成员名称	功能描述
BKP_DATA_9	BKP数据寄存器9
BKP_DATA_10	BKP数据寄存器10
BKP_DATA_11	BKP数据寄存器11
BKP_DATA_12	BKP数据寄存器12
BKP_DATA_13	BKP数据寄存器13
BKP_DATA_14	BKP数据寄存器14
BKP_DATA_15	BKP数据寄存器15
BKP_DATA_16	BKP数据寄存器16
BKP_DATA_17	BKP数据寄存器17
BKP_DATA_18	BKP数据寄存器18
BKP_DATA_19	BKP数据寄存器19
BKP_DATA_20	BKP数据寄存器20
BKP_DATA_21	BKP数据寄存器21
BKP_DATA_22	BKP数据寄存器22
BKP_DATA_23	BKP数据寄存器23
BKP_DATA_24	BKP数据寄存器24
BKP_DATA_25	BKP数据寄存器25
BKP_DATA_26	BKP数据寄存器26
BKP_DATA_27	BKP数据寄存器27
BKP_DATA_28	BKP数据寄存器28
BKP_DATA_29	BKP数据寄存器29
BKP_DATA_30	BKP数据寄存器30
BKP_DATA_31	BKP数据寄存器31
BKP_DATA_32	BKP数据寄存器32
BKP_DATA_33	BKP数据寄存器33
BKP_DATA_34	BKP数据寄存器34
BKP_DATA_35	BKP数据寄存器35
BKP_DATA_36	BKP数据寄存器36
BKP_DATA_37	BKP数据寄存器37
BKP_DATA_38	BKP数据寄存器38
BKP_DATA_39	BKP数据寄存器39
BKP_DATA_40	BKP数据寄存器40
BKP_DATA_41	BKP数据寄存器41

函数 bkp_deinit

函数bkp_deinit描述见下表:

表 3-41. 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 3-42. 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 表3-40. 枚举类型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 3-43. 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 表3-40. 枚举类型 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 3-44. 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 3-45. 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 3-46. 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 3-47. 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 3-48. 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_calibration_value_set

函数bkp_rtc_calibration_value_set描述见下表:

表 3-49. 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 3-50. 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 3-51. 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 3-52. 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 3-53. 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 3-54. 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 3-55. 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 3-56. 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 3-57. 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 3-58. 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. CAN

CAN（Controller Area Network）总线是一种可以在无主机情况下实现微处理器或者设备之间相互通信的总线标准。章节[3.4.1](#)描述了CAN的寄存器列表，章节[3.4.2](#)对CAN库函数进行说明

3.4.1. 外设寄存器说明

CAN寄存器列表如下表所示：

表 3-59. CAN 寄存器

寄存器名称	寄存器描述
CAN_CTL	控制寄存器
CAN_STAT	状态寄存器
CAN_TSTAT	发送状态寄存器
CAN_RFIFO0	接收FIFO0寄存器
CAN_RFIFO1	接收FIFO1寄存器
CAN_INTEN	中断使能寄存器
CAN_ERR	错误寄存器
CAN_BT	位时序寄存器
CAN_TMIx	发送邮箱标识符寄存器
CAN_TMPx	发送邮箱属性寄存器
CAN_TMDA TA0x	发送邮箱data0寄存器
CAN_TMDA TA1x	发送邮箱data1寄存器
CAN_RFIFOMIx	接收FIFO邮箱标识符寄存器
CAN_RFIFOMPx	接收FIFO邮箱属性寄存器
CAN_RFIFOMDAT A0x	接收FIFO邮箱data0寄存器
CAN_RFIFOMDAT A1x	接收FIFO邮箱data1寄存器
CAN_FCTL	过滤器控制寄存器
CAN_FMCFG	过滤器模式配置寄存器
CAN_FSCFG	过滤器位宽配置寄存器
CAN_FAFIFO	过滤器关联FIFO寄存器
CAN_FW	过滤器激活寄存器
CAN_FxDATAy	过滤器(x)数据(y)寄存器

3.4.2. 外设库函数说明

CAN库函数列表如下表所示：

表 3-60. CAN 库函数

库函数名称	库函数描述
can_deinit	复位外设CAN

库函数名称	库函数描述
can_struct_para_init	初始化外设CAN结构体参数
can_init	初始化外设CAN
can_filter_init	CAN过滤器初始化
can1_filter_start_bank	CAN1过滤器序起始编号设置
can_debug_freeze_enable	CAN调试冻结使能
can_debug_freeze_disable	CAN调试冻结关闭
can_time_trigger_mode_enable	CAN时间触发模式使能
can_time_trigger_mode_disable	CAN时间触发模式关闭
can_message_transmit	CAN传输报文
can_transmit_states	获取CAN传输状态
can_transmission_stop	CAN邮箱停止发送
can_message_receive	CAN接收报文
can_fifo_release	CAN释放FIFO
can_receive_message_length_get	获取CAN接收帧的数量
can_working_mode_set	CAN工作模式设置
can_wakeup	从睡眠模式中唤醒CAN
can_error_get	获取CAN总线错误
can_receive_error_number_get	获取CAN接收错误
can_transmit_error_number_get	获取CAN发送错误
can_interrupt_enable	CAN中断使能
can_interrupt_disable	CAN中断关闭
can_flag_get	获取CAN标志位状态
can_flag_clear	清除CAN标志位状态
can_interrupt_flag_get	获取CAN中断标志位状态
can_interrupt_flag_clear	清除CAN中断标志位状态

结构体 can_parameter_struct

表 3-61. 结构体 can_parameter_struct

成员名称	功能描述
working_mode	工作模式
resync_jump_width	再同步补偿宽度
time_segment_1	位段1
time_segment_2	位段2
time_triggered	时间触发通信模式
auto_bus_off_recovery	自动离线恢复
auto_wakeup	自动唤醒
no_auto_retrans	自动重传禁止
rec_fifo_overflow	接收FIFO满时覆盖
trans_fifo_order	发送FIFO顺序
prescaler	波特率分频系数

结构体 `can_trasnmit_message_struct`

表 3-62. 结构体 `can_trasnmit_message_struct`

成员名称	功能描述
tx_sfid	标准格式帧标识符
tx_efid	扩展格式帧标识符
tx_ff	帧格式：标准格式/扩展格式
tx_ft	帧类型：数据帧/远程帧
tx_dlen	数据长度
tx_data[8]	数据值

结构体 `can_receive_message_struct`

表 3-63. 结构体 `can_receive_message_struct`

成员名称	功能描述
rx_sfid	标准格式帧标识符
rx_efid	扩展格式帧标识符
rx_ff	帧格式：标准格式/扩展格式
rx_ft	帧类型：数据帧/远程帧
rx_dlen	数据长度
rx_data[8]	数据值
rx_fi	过滤器索引

结构体 `can_filter_parameter_struct`

表 3-64. 结构体 `can_filter_parameter_struct`

成员名称	功能描述
filter_list_high	过滤器列表数高位
filter_list_low	过滤器列表数低位
filter_mask_high	过滤器掩码数高位
filter_mask_low	过滤器掩码数低位
filter_fifo_number	接收FIFO编号
filter_number	过滤器索引号
filter_mode	过滤模式：列表模式/掩码模式
filter_bits	过滤器位宽
filter_enable	过滤器是否工作

枚举类型 `can_flag_enum`

表 3-65. 枚举类型 `can_flag_enum`

成员名称	功能描述
<code>CAN_FLAG_RXL</code>	RX引脚电平
<code>CAN_FLAG_LASTRX</code>	RX引脚最近一次的采样值
<code>CAN_FLAG_RS</code>	接收状态

成员名称	功能描述
CAN_FLAG_TS	发送状态
CAN_FLAG_SLPIF	进入睡眠工作模式的状态改变标志
CAN_FLAG_WUIF	从睡眠工作模式唤醒的状态改变标志
CAN_FLAG_ERRIF	错误标志
CAN_FLAG_SLPWS	睡眠工作状态
CAN_FLAG_IWS	初始化工作状态
CAN_FLAG_TMLS2	在发送FIFO中邮箱2最后发送
CAN_FLAG_TMLS1	在发送FIFO中邮箱1最后发送
CAN_FLAG_TMLS0	在发送FIFO中邮箱0最后发送
CAN_FLAG_TME2	发送邮箱2空
CAN_FLAG_TME1	发送邮箱1空
CAN_FLAG_TME0	发送邮箱0空
CAN_FLAG_MTE2	邮箱2发送错误
CAN_FLAG_MTE1	邮箱1发送错误
CAN_FLAG_MTE0	邮箱0发送错误
CAN_FLAG_MAL2	邮箱2仲裁失败
CAN_FLAG_MAL1	邮箱1仲裁失败
CAN_FLAG_MAL0	邮箱0仲裁失败
CAN_FLAG_MTFNERR2	邮箱2无错发送完成
CAN_FLAG_MTFNERR1	邮箱1无错发送完成
CAN_FLAG_MTFNERR0	邮箱0无错发送完成
CAN_FLAG_MTF2	邮箱2发送完成
CAN_FLAG_MTF1	邮箱1发送完成
CAN_FLAG_MTF0	邮箱0发送完成
CAN_FLAG_RFO0	接收FIFO0溢出
CAN_FLAG_RFF0	接收FIFO0满
CAN_FLAG_RFO1	接收FIFO1溢出
CAN_FLAG_RFF1	接收FIFO1满
CAN_FLAG_BOERR	离线错误
CAN_FLAG_PERR	被动错误
CAN_FLAG_WERR	警告错误

枚举类型 can_interrupt_flag_enum

表 3-66. 枚举类型 can_interrupt_flag_enum

成员名称	功能描述
CAN_INT_FLAG_SLPIF	进入睡眠工作模式的状态改变中断标志
CAN_INT_FLAG_WUIF	从睡眠工作模式唤醒的状态改变中断标志
CAN_INT_FLAG_ERRIF	错误中断标志
CAN_INT_FLAG_MTF2	邮箱2发送完成中断标志
CAN_INT_FLAG_MTF1	邮箱1发送完成中断标志
CAN_INT_FLAG_MTF0	邮箱0发送完成中断标志

成员名称	功能描述
CAN_INT_FLAG_RFO0	接收FIFO0溢出中断标志
CAN_INT_FLAG_RFF0	接收FIFO0满中断标志
CAN_INT_FLAG_RFL0	接收FIFO0非空中断标志
CAN_INT_FLAG_RFO1	接收FIFO1溢出中断标志
CAN_INT_FLAG_RFF1	接收FIFO1满中断标志
CAN_INT_FLAG_RFL1	接收FIFO1非空中断标志
CAN_INT_FLAG_ERRN	错误种类中断标志
CAN_INT_FLAG_BOERR	离线错误中断标志
CAN_INT_FLAG_PERR	被动错误中断标志
CAN_INT_FLAG_WERR	主动错误中断标志

枚举类型 can_error_enum

表 3-67. 枚举类型 can_error_enum

成员名称	功能描述
CAN_ERROR_NONE	无错误
CAN_ERROR_FILL	填充错误
CAN_ERROR_FORMATE	格式错误
CAN_ERROR_ACK	ACK错误
CAN_ERROR_BITRECESSIVE	位隐性错
CAN_ERROR_BITDOMINANTER	位显性错误
CAN_ERROR_CRC	CRC错误
CAN_ERROR_SOFTWARECFG	软件设置值

枚举类型 can_transmit_state_enum

表 3-68. 枚举类型 can_transmit_state_enum

成员名称	功能描述
CAN_TRANSMIT_FAILED	CAN发送失败
CAN_TRANSMIT_OK	CAN发送成功
CAN_TRANSMIT_PENDING	CAN发送挂起
CAN_TRANSMIT_NOMAILBOX	CAN发送时未选到邮箱

枚举类型 can_struct_type_enum

表 3-69. 枚举类型 can_struct_type_enum

成员名称	功能描述
CAN_INIT_STRUCT	CAN初始化结构体
CAN_FILTER_STRUCT	CAN过滤器结构体
CAN_TX_MESSAGE_STRUCT	CAN发送消息结构体
CAN_RX_MESSAGE_STRUCT	CAN接收消息结构体

函数 `can_deinit`

函数`can_deinit`描述见下表:

表 3-70. 函数 `can_deinit`

函数名称	<code>can_deinit</code>
函数原型	<code>void can_deinit(uint32_t can_periph);</code>
功能描述	复位外设CAN
先决条件	-
被调用函数	<code>rcu_periph_reset_enable/ rcu_periph_reset_disable</code>
输入参数{in}	
<code>can_periph</code>	CAN外设
<code>CANx(x=0,1)</code>	CAN外设选择, CAN1仅适用于GD32F10x_CL
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* CAN0 deinitialize*/
```

```
can_deinit(CAN0);
```

函数 `can_struct_para_init`

函数`can_struct_para_init`描述见下表:

表 3-71. 函数 `can_struct_para_init`

函数名称	<code>can_struct_para_init</code>
函数原型	<code>void can_struct_para_init(can_struct_type_enum type, void* p_struct)</code>
功能描述	初始化外设CAN结构体参数
先决条件	-
被调用函数	-
输入参数{in}	
<code>type</code>	CAN结构体参考 表3-69. 枚举类型can_struct_type_enum
<code>CAN_INIT_STRUCTURE</code>	CAN初始化参数结构体
<code>CAN_FILTER_STRUCTURE</code>	CAN过滤器参数结构体
<code>CAN_TX_MESSAGE_STRUCTURE</code>	CAN发送信息参数结构体
<code>CAN_RX_MESSAGE_STRUCTURE</code>	CAN接受信息参数结构体
输入参数{in}	
<code>p_struct</code>	特定结构体的指针

输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* CAN parameter initialize*/
```

```
can_init(CAN_INIT_STRUCT, &can_parameter);
```

函数 can_init

函数can_init描述见下表：

表 3-72. 函数 can_init

函数名称	can_init
函数原型	ErrStatus can_init(uint32_t can_periph, can_parameter_struct* can_parameter_init);
功能描述	初始化外设CAN
先决条件	-
被调用函数	-
输入参数{in}	
can_periph	CAN外设
CANx(x=0,1)	CAN外设选择，CAN1仅适用于GD32F10x_CL
输入参数{in}	
can_parameter_init	初始化结构体，结构体成员参考 表3-61. 结构体can_parameter_struct
输出参数{out}	
-	-
返回值	
ErrStatus	SUCCESS / ERROR

例如：

```
/* CAN0 initialize*/
```

```
can_parameter_struct can_parameter;
```

```
can_init(CAN0, &can_parameter);
```

函数 can_filter_init

函数can_filter_init描述见下表：

表 3-73. 函数 can_filter_init

函数名称	can_filter_init
函数原型	void can_filter_init(can_filter_parameter_struct* can_filter_parameter_init);

功能描述	CAN过滤器初始化
先决条件	-
被调用函数	-
输入参数{in}	
can_filter_parameter_init	过滤器初始化结构体，结构体成员参考 表3-64. 结构体 can_filter_parameter_struct
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* initialize CAN filter */
can_filter_parameter_struct can_filter;
can_filter_init(&can_filter);
```

函数 can1_filter_start_bank

函数can1_filter_start_bank描述见下表：

表 3-74. 函数 can1_filter_start_bank

函数名称	can1_filter_start_bank
函数原型	void can1_filter_start_bank(uint8_t start_bank);
功能描述	CAN1过滤器序起始编号设置
先决条件	-
被调用函数	-
输入参数{in}	
start_bank	CAN1过滤器序起始编号
1..27	可选的编号
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* set CAN1 filter start bank number 15*/
can1_filter_start_bank(15);
```

函数 can_debug_freeze_enable

函数can_debug_freeze_enable描述见下表：

表 3-75. 函数 can_debug_freeze_enable

函数名称	can_debug_freeze_enable
函数原型	void can_debug_freeze_enable(uint32_t can_periph);
功能描述	CAN调试冻结使能
先决条件	-
被调用函数	dbg_periph_enable
输入参数{in}	
can_periph	CAN外设
CANx(x=0,1)	CAN外设选择, CAN1仅适用于GD32F10x_CL
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* enable CAN0 debug freeze */
can_debug_freeze_enable(CAN0);
```

函数 can_debug_freeze_disable

函数can_debug_freeze_disable描述见下表:

表 3-76. 函数 can_debug_freeze_disable

函数名称	can_debug_freeze_disable
函数原型	void can_debug_freeze_disable(uint32_t can_periph);
功能描述	CAN调试冻结关闭
先决条件	-
被调用函数	dbg_periph_disable
输入参数{in}	
can_periph	CAN外设
CANx(x=0,1)	CAN外设选择, CAN1仅适用于GD32F10x_CL
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* disable CAN0 debug freeze */
can_debug_freeze_disable(CAN0);
```

函数 can_time_trigger_mode_enable

函数can_time_trigger_mode_enable描述见下表:

表 3-77. 函数 can_time_trigger_mode_enable

函数名称	can_time_trigger_mode_enable
函数原型	void can_time_trigger_mode_enable(uint32_t can_periph);
功能描述	CAN时间触发模式使能
先决条件	-
被调用函数	-
输入参数{in}	
can_periph	CAN外设
CANx(x=0,1)	CAN外设选择, CAN1仅适用于GD32F10x_CL
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* enable CAN0 time trigger mode */
can_time_trigger_mode_enable(CAN0);
```

函数 can_time_trigger_mode_disable

函数can_time_trigger_mode_disable描述见下表:

表 3-78. 函数 can_time_trigger_mode_disable

函数名称	can_time_trigger_mode_disable
函数原型	void can_time_trigger_mode_disable(uint32_t can_periph);
功能描述	CAN时间触发模式关闭
先决条件	-
被调用函数	-
输入参数{in}	
can_periph	CAN外设
CANx(x=0,1)	CAN外设选择, CAN1仅适用于GD32F10x_CL
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* disable CAN0 time trigger mode */
can_time_trigger_mode_disable(CAN0);
```

函数 can_message_transmit

函数can_message_transmit描述见下表:

表 3-79. 函数 can_message_transmit

函数名称	can_message_transmit
函数原型	uint8_t can_message_transmit(uint32_t can_periph, can_transmit_message_struct* transmit_message);
功能描述	CAN传输报文
先决条件	-
被调用函数	-
输入参数{in}	
can_periph	CAN外设
CANx(x=0,1)	CAN外设选择, CAN1仅适用于GD32F10x_CL
输入参数{in}	
transmit_message	报文发送结构体, 结构体成员参考 表3-62. 结构体 can_transmit_message_struct
输出参数{out}	
-	-
返回值	
uint8_t	0x00-0x03

例如:

```
/* CAN0 transmit message and return the mailbox number */
```

```
uint8_t transmit_mailbox = 0;
```

```
transmit_mailbox = can_message_transmit(CAN0, &transmit_message);
```

函数 can_transmit_states

函数can_transmit_states描述见下表:

表 3-80. 函数 can_transmit_states

函数名称	can_transmit_states
函数原型	can_transmit_state_enum can_transmit_states(uint32_t can_periph, uint8_t mailbox_number);
功能描述	获取CAN传输状态
先决条件	-
被调用函数	-
输入参数{in}	
can_periph	CAN外设
CANx(x=0,1)	CAN外设选择, CAN1仅适用于GD32F10x_CL
输入参数{in}	
mailbox_number	邮箱标号
CAN_MAILBOXx	CAN_MAILBOXx(x=0,1,2)
输出参数{out}	
-	-
返回值	

can_transmit_state_enum	请参考 表3-68. 枚举类型can_transmit_state_enum
-------------------------	--

例如：

```
/* CAN0 mailbox0 transmit state */
can_transmit_state_enum transmit_state = CAN_TRANSMIT_FAILED;
transmit_state = can_transmit_states(CAN0, CAN_MAILBOX0);
```

函数 can_transmission_stop

函数can_transmission_stop描述见下表：

表 3-81. 函数 can_transmission_stop

函数名称	can_transmission_stop
函数原型	void can_transmission_stop(uint32_t can_periph, uint8_t mailbox_number);
功能描述	CAN邮箱停止发送
先决条件	-
被调用函数	-
输入参数{in}	
can_periph	CAN外设
CANx(x=0,1)	CAN外设选择，CAN1仅适用于GD32F10x_CL
输入参数{in}	
mailbox_number	邮箱标号
CAN_MAILBOXx	CAN_MAILBOXx(x=0,1,2)
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* stop CAN0 mailbox0 transmission */
can_transmission_stop(CAN0, CAN_MAILBOX0);
```

函数 can_message_receive

函数can_message_receive描述见下表：

表 3-82. 函数 can_message_receive

函数名称	can_message_receive
函数原型	void can_message_receive(uint32_t can_periph, uint8_t fifo_number, can_receive_message_struct* receive_message);
功能描述	CAN接收报文
先决条件	-

被调用函数	-
输入参数{in}	
can_periph	CAN外设
CANx(x=0,1)	CAN外设选择, CAN1仅适用于GD32F10x_CL
输入参数{in}	
fifo_number	FIFO编号
CAN_FIFOx	CAN_FIFOx(x=0,1)
输入参数{in}	
receive_message	接收报文结构体, 结构体成员参考 表3-63. 结构体 can_receive_message_struct
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* CAN0 FIFO0 receive message */
```

```
can_message_receive(CAN0, CAN_FIFO0, &receive_message);
```

函数 can_fifo_release

函数can_fifo_release描述见下表:

表 3-83. 函数 can_fifo_release

函数名称	can_fifo_release
函数原型	void can_fifo_release(uint32_t can_periph, uint8_t fifo_number);
功能描述	CAN释放FIFO
先决条件	-
被调用函数	-
输入参数{in}	
can_periph	CAN外设
CANx(x=0,1)	CAN外设选择, CAN1仅适用于GD32F10x_CL
输入参数{in}	
fifo_number	FIFO编号
CAN_FIFOx	CAN_FIFOx(x=0,1)
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* CAN0 release FIFO0 */
```

```
can_fifo_release(CAN0, CAN_FIFO0);
```

函数 can_receive_message_length_get

函数can_receive_message_length_get描述见下表：

表 3-84. 函数 can_receive_message_length_get

函数名称	can_receive_message_length_get
函数原型	uint8_t can_receive_message_length_get(uint32_t can_periph, uint8_t fifo_number);
功能描述	获取CAN接收帧的数量
先决条件	-
被调用函数	-
输入参数{in}	
can_periph	CAN外设
CANx(x=0,1)	CAN外设选择，CAN1仅适用于GD32F10x_CL
输入参数{in}	
fifo_number	FIFO编号
CAN_FIFOx	CAN_FIFOx(x=0,1)
输出参数{out}	
-	-
返回值	
uint8_t	0..3

例如：

```
/* CAN0 FIFO0 receive message length */
```

```
uint8_t frame_number = 0;
```

```
frame_number = can_receive_message_length_get(CAN0, CAN_FIFO0);
```

函数 can_working_mode_set

函数can_working_mode_set描述见下表：

表 3-85. 函数 can_working_mode_set

函数名称	can_working_mode_set
函数原型	ErrStatus can_working_mode_set(uint32_t can_periph, uint8_t working_mode);
功能描述	CAN工作模式设置
先决条件	-
被调用函数	-
输入参数{in}	
can_periph	CAN外设
CANx(x=0,1)	CAN外设选择，CAN1仅适用于GD32F10x_CL
输入参数{in}	
can_working_mode	模式选择

CAN_MODE_INITIALIZE	初始化模式
CAN_MODE_NORMAL	正常模式
CAN_MODE_SLEEP	睡眠模式
输出参数{out}	
-	-
返回值	
ErrStatus	SUCCESS / ERROR

例如：

```
/* set CAN0 working at initialize mode */
can_working_mode_set(CAN0, CAN_MODE_INITIALIZE);
```

函数 can_wakeup

函数can_wakeup描述见下表：

表 3-86. 函数 can_wakeup

函数名称	can_wakeup
函数原型	ErrStatus can_wakeup(uint32_t can_periph);
功能描述	从睡眠模式中唤醒CAN
先决条件	-
被调用函数	-
输入参数{in}	
can_periph	CAN外设
CANx(x=0,1)	CAN外设选择，CAN1仅适用于GD32F10x_CL
输出参数{out}	
-	-
返回值	
ErrStatus	SUCCESS / ERROR

例如：

```
/* wake up CAN0 */
can_wakeup(CAN0);
```

函数 can_error_get

函数can_error_get描述见下表：

表 3-87. 函数 can_error_get

函数名称	can_error_get
函数原型	can_error_enum can_error_get(uint32_t can_periph);

功能描述	获取CAN总线错误
先决条件	-
被调用函数	-
输入参数{in}	
can_periph	CAN外设
CANx(x=0,1)	CAN外设选择, CAN1仅适用于GD32F10x_CL
输出参数{out}	
-	-
返回值	
can_error_enum	请参考 表3-67. 枚举类型can_error_enum

例如:

```
/* get CAN0 error type */
can_error_enum err_type;
err_type = can_error_get(CAN0);
```

函数 can_receive_error_number_get

函数can_receive_error_number_get描述见下表:

表 3-88. 函数 can_receive_error_number_get

函数名称	can_receive_error_number_get
函数原型	uint8_t can_receive_error_number_get(uint32_t can_periph);
功能描述	获取CAN接收错误
先决条件	-
被调用函数	-
输入参数{in}	
can_periph	CAN外设
CANx(x=0,1)	CAN外设选择, CAN1仅适用于GD32F10x_CL
输出参数{out}	
-	-
返回值	
uint8_t	0..255

例如:

```
/* get CAN0 receive error number */
uint8_t error_num;
error_num = can_receive_error_number_get(CAN0);
```

函数 can_transmit_error_number_get

函数can_transmit_error_number_get描述见下表:

表 3-89. 函数 can_transmit_error_number_get

函数名称	can_transmit_error_number_get
函数原型	uint8_t can_transmit_error_number_get(uint32_t can_periph);
功能描述	获取CAN发送错误
先决条件	-
被调用函数	-
输入参数{in}	
can_periph	CAN外设
CANx(x=0,1)	CAN外设选择, CAN1仅适用于GD32F10x_CL
输出参数{out}	
-	-
返回值	
uint8_t	0..255

例如:

```
/* get CAN0 transmit error number */
```

```
uint8_t error_num;
```

```
error_num = can_transmit_error_number_get(CAN0);
```

函数 can_interrupt_enable

函数can_interrupt_enable描述见下表:

表 3-90. 函数 can_interrupt_enable

函数名称	can_interrupt_enable
函数原型	void can_interrupt_enable(uint32_t can_periph, uint32_t interrupt);
功能描述	CAN中断使能
先决条件	-
被调用函数	-
输入参数{in}	
can_periph	CAN外设
CANx(x=0,1)	CAN外设选择, CAN1仅适用于GD32F10x_CL
输入参数{in}	
interrupt	中断类型
CAN_INT_TME	发送邮箱空中断使能
CAN_INT_RFNE0	接收FIFO0非空中断使能
CAN_INT_RFF0	接收FIFO0满中断使能
CAN_INT_RFO0	接收FIFO0溢出中断使能
CAN_INT_RFNE1	接收FIFO1非空中断使能
CAN_INT_RFF1	接收FIFO1满中断使能
CAN_INT_RFO1	接收FIFO1溢出中断使能
CAN_INT_WERR	警告错误中断使能

CAN_INT_PERR	被动错误中断使能
CAN_INT_BO	离线中断使能
CAN_INT_ERRN	错误种类中断使能
CAN_INT_ERR	错误中断使能
CAN_INT_WU	唤醒中断使能
CAN_INT_SLPW	睡眠中断使能
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* CAN0 transmit mailbox empty interrupt enable */
```

```
can_interrupt_enable(CAN0, CAN_INT_TME);
```

函数 can_interrupt_disable

函数can_interrupt_disable描述见下表:

表 3-91. 函数 can_interrupt_disable

函数名称	can_interrupt_disable
函数原型	void can_interrupt_disable(uint32_t can_periph, uint32_t interrupt);
功能描述	CAN中断关闭
先决条件	-
被调用函数	-
输入参数{in}	
can_periph	CAN外设
CANx(x=0,1)	CAN外设选择, CAN1仅适用于GD32F10x_CL
输入参数{in}	
interrupt	中断类型
CAN_INT_TME	发送邮箱空中断使能
CAN_INT_RFNE0	接收FIFO0非空中断使能
CAN_INT_RFF0	接收FIFO0满中断使能
CAN_INT_RFO0	接收FIFO0溢出中断使能
CAN_INT_RFNE1	接收FIFO1非空中断使能
CAN_INT_RFF1	接收FIFO1满中断使能
CAN_INT_RFO1	接收FIFO1溢出中断使能
CAN_INT_WERR	警告错误中断使能
CAN_INT_PERR	被动错误中断使能
CAN_INT_BO	离线中断使能
CAN_INT_ERRN	错误种类中断使能
CAN_INT_ERR	错误中断使能
CAN_INT_WU	唤醒中断使能

CAN_INT_SLPW	睡眠中断使能
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* CAN0 transmit mailbox empty interrupt disable */
```

```
can_interrupt_disable(CAN0, CAN_INT_TME);
```

函数 can_flag_get

函数can_flag_get描述见下表:

表 3-92. 函数 can_flag_get

函数名称	can_flag_get
函数原型	FlagStatus can_flag_get(uint32_t can_periph, can_flag_enum flag);
功能描述	获取CAN标志位状态
先决条件	-
被调用函数	-
输入参数{in}	
can_periph	CAN外设
CANx(x=0,1)	CAN外设选择, CAN1仅适用于GD32F10x_CL
输入参数{in}	
flag	CAN标志位参考 表3-65. 枚举类型can_flag_enum
CAN_FLAG_RXL	RX引脚电平
CAN_FLAG_LASTRX	RX引脚最近一次的采样值
CAN_FLAG_RS	接收状态
CAN_FLAG_TS	发送状态
CAN_FLAG_SLPWF	进入睡眠工作模式的状态改变标志
CAN_FLAG_WUWF	从睡眠工作模式唤醒的状态改变标志
CAN_FLAG_ERRIF	错误标志
CAN_FLAG_SLPWS	睡眠工作状态
CAN_FLAG_IWS	初始化工作状态
CAN_FLAG_TMLS2	在发送FIFO中邮箱2最后发送
CAN_FLAG_TMLS1	在发送FIFO中邮箱1最后发送
CAN_FLAG_TMLS0	在发送FIFO中邮箱0最后发送
CAN_FLAG_TME2	发送邮箱2空
CAN_FLAG_TME1	发送邮箱1空
CAN_FLAG_TME0	发送邮箱0空
CAN_FLAG_MTE2	邮箱2发送错误

CAN_FLAG_MTE1	邮箱1发送错误
CAN_FLAG_MTE0	邮箱0发送错误
CAN_FLAG_MAL2	邮箱2仲裁失败
CAN_FLAG_MAL1	邮箱1仲裁失败
CAN_FLAG_MAL0	邮箱0仲裁失败
CAN_FLAG_MTFN ERR2	邮箱2无错发送完成
CAN_FLAG_MTFN ERR1	邮箱1无错发送完成
CAN_FLAG_MTFN ERR0	邮箱0无错发送完成
CAN_FLAG_MTF2	邮箱2发送完成
CAN_FLAG_MTF1	邮箱1发送完成
CAN_FLAG_MTF0	邮箱0发送完成
CAN_FLAG_RFO0	接收FIFO0溢出
CAN_FLAG_RFF0	接收FIFO0满
CAN_FLAG_RFO1	接收FIFO1溢出
CAN_FLAG_RFF1	接收FIFO1满
CAN_FLAG_BOER R	离线错误
CAN_FLAG_PERR	被动错误
CAN_FLAG_WERR	警告错误
输出参数{out}	
-	-
返回值	
FlagStatus	SET / RESET

例如：

```
/* get CAN0 mailbox 0 transmit finished flag */
```

```
can_flag_get(CAN0, CAN_FLAG_MTF0);
```

函数 can_flag_clear

函数can_flag_clear描述见下表：

表 3-93. 函数 can_flag_clear

函数名称	can_flag_clear
函数原型	void can_flag_clear(uint32_t can_periph, can_flag_enum flag);
功能描述	清除CAN标志位状态
先决条件	-
被调用函数	-
输入参数{in}	
can_periph	CAN外设

CANx(x=0,1)	CAN外设选择, CAN1仅适用于GD32F10x_CL
输入参数{in}	
flag	CAN标志位参考 表3-65. 枚举类型can_flag_enum
CAN_FLAG_SLPIF	进入睡眠工作模式的状态改变标志
CAN_FLAG_WUIF	从睡眠工作模式唤醒的状态改变标志
CAN_FLAG_ERRIF	错误标志
CAN_FLAG_MTE2	邮箱2发送错误
CAN_FLAG_MTE1	邮箱1发送错误
CAN_FLAG_MTE0	邮箱0发送错误
CAN_FLAG_MAL2	邮箱2仲裁失败
CAN_FLAG_MAL1	邮箱1仲裁失败
CAN_FLAG_MAL0	邮箱0仲裁失败
CAN_FLAG_MTFNERR2	邮箱2无错发送完成
CAN_FLAG_MTFNERR1	邮箱1无错发送完成
CAN_FLAG_MTFNERR0	邮箱0无错发送完成
CAN_FLAG_MTF2	邮箱2发送完成
CAN_FLAG_MTF1	邮箱1发送完成
CAN_FLAG_MTF0	邮箱0发送完成
CAN_FLAG_RFO0	接收FIFO0溢出
CAN_FLAG_RFF0	接收FIFO0满
CAN_FLAG_RFO1	接收FIFO1溢出
CAN_FLAG_RFF1	接收FIFO1满
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* clear CAN0 mailbox 0 transmit error flag */
can_flag_clear(CAN0, CAN_FLAG_MTE0);
```

函数 can_interrupt_flag_get

函数can_interrupt_flag_get描述见下表:

表 3-94. 函数 can_interrupt_flag_get

函数名称	can_interrupt_flag_get
函数原型	FlagStatus can_interrupt_flag_get(uint32_t can_periph, can_interrupt_flag_enum flag);
功能描述	获取CAN中断标志位状态

先决条件	-
被调用函数	-
输入参数{in}	
can_periph	CAN外设
CANx(x=0,1)	CAN外设选择, CAN1仅适用于GD32F10x_CL
输入参数{in}	
flag	CAN中断标志位参考 表3-66. 枚举类型can_interrupt_flag_enum
CAN_INT_FLAG_SLP IF	进入睡眠工作模式的状态改变中断标志
CAN_INT_FLAG_WUI F	从睡眠工作模式唤醒的状态改变中断标志
CAN_INT_FLAG_ER RIF	错误中断标志
CAN_INT_FLAG_MT F2	邮箱2发送完成中断标志
CAN_INT_FLAG_MT F1	邮箱1发送完成中断标志
CAN_INT_FLAG_MT F0	邮箱0发送完成中断标志
CAN_INT_FLAG_RF O0	接收FIFO0溢出中断标志
CAN_INT_FLAG_RFF 0	接收FIFO0满中断标志
CAN_INT_FLAG_RF O1	接收FIFO1溢出中断标志
CAN_INT_FLAG_RFF 1	接收FIFO1满中断标志
CAN_INT_FLAG_ER RN	错误种类中断标志
CAN_INT_FLAG_BO ERR	离线错误中断标志
CAN_INT_FLAG_PE RR	被动错误中断标志
CAN_INT_FLAG_WE RR	主动错误中断标志
输出参数{out}	
-	-
返回值	
FlagStatus	SET / RESET

例如:

```
/* get CAN0 mailbox 0 transmit finished interrupt flag */
```

```
FlagStatus Status;
```

```
Status = can_interrupt_flag_get(CAN0, CAN_INT_FLAG_MTF0);
```

函数 can_interrupt_flag_clear

函数can_interrupt_flag_clear描述见下表:

表 3-95. 函数 can_interrupt_flag_clear

函数名称	can_interrupt_flag_clear
函数原型	void can_interrupt_flag_clear(uint32_t can_periph, can_interrupt_flag_enum flag);
功能描述	清除CAN中断标志位状态
先决条件	-
被调用函数	-
输入参数{in}	
can_periph	CAN外设
CANx(x=0,1)	CAN外设选择, CAN1仅适用于GD32F10x_CL
输入参数{in}	
flag	CAN中断标志位参考 表3-66. 枚举类型can_interrupt_flag_enum
CAN_INT_FLAG_SLP IF	进入睡眠工作模式的状态改变中断标志
CAN_INT_FLAG_WUI F	从睡眠工作模式唤醒的状态改变中断标志
CAN_INT_FLAG_ER RIF	错误中断标志
CAN_INT_FLAG_MT F2	邮箱2发送完成中断标志
CAN_INT_FLAG_MT F1	邮箱1发送完成中断标志
CAN_INT_FLAG_MT F0	邮箱0发送完成中断标志
CAN_INT_FLAG_RF O0	接收FIFO0溢出中断标志
CAN_INT_FLAG_RFF 0	接收FIFO0满中断标志
CAN_INT_FLAG_RF O1	接收FIFO1溢出中断标志
CAN_INT_FLAG_RFF 1	接收FIFO1满中断标志
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* clear CAN0 mailbox 0 transmit finished interrupt flag */
can_interrupt_flag_clear(CAN0, CAN_INT_FLAG_MTF0);
```

3.5. CRC

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

3.5.1. 外设寄存器说明

CRC寄存器列表如下表所示：

表 3-96. 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

3.5.2. 外设库函数说明

CRC库函数列表如下表所示：

表 3-97. 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 3-98. 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 3-99. 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 3-100. 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 3-101. 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 3-102. 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 3-103. 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;
rcu_periph_clock_enable(RCU_CRC);
valcrc = crc_single_data_calculate(val);
```

函数 **crc_block_data_calculate**

函数crc_block_data_calculate描述见下表:

表 3-104. 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};

Rcu_periph_clock_enable(RCU_CRC);

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

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 3-105. DAC 寄存器

寄存器名称	寄存器描述
DAC_CTL0	DACx控制寄存器0
DAC_SWT	DACx软件触发寄存器
DAC_OUT0_R12DH	DAC_OUT0 12位右对齐数据保持寄存器
DAC_OUT0_L12DH	DAC_OUT0 12位左对齐数据保持寄存器
DAC_OUT0_R8DH	DAC_OUT0 8位右对齐数据保持寄存器
DAC_OUT1_R12DH	DAC_OUT1 12位右对齐数据保持寄存器
DAC_OUT1_L12DH	DAC_OUT1 12位左对齐数据保持寄存器
DAC_OUT1_R8DH	DAC_OUT1 8位右对齐数据保持寄存器
DACC_R12DH	DAC并发模式12位右对齐数据保持寄存器

寄存器名称	寄存器描述
DACC_L12DH	DAC并发模式12位左对齐数据保持寄存器
DACC_R8DH	DAC并发模式8位右对齐数据保持寄存器
DAC_OUT0_DO	DAC_OUT0数据输出寄存器
DAC_OUT1_DO	DAC_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 3-106. 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
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 3-107. 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 3-108. 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 3-109. 函数 `dac_disable`

函数名称	<code>dac_disable</code>
函数原型	<code>void dac_disable(uint32_t dac_periph, uint8_t dac_out);</code>
功能描述	DAC禁能
先决条件	-
被调用函数	-
输入参数{in}	
<code>dac_periph</code>	DAC外设
<code>DACx</code>	DAC外设选择 ($x = 0$)
输入参数{in}	
<code>dac_out</code>	DAC输出
<code>DAC_OUTx</code>	DAC输出通道选择 ($x = 0,1$)
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 3-110. 函数 `dac_dma_enable`

函数名称	<code>dac_dma_enable</code>
函数原型	<code>void dac_dma_enable(uint32_t dac_periph, uint8_t dac_out);</code>
功能描述	DAC的DMA 功能使能
先决条件	-
被调用函数	-
输入参数{in}	
<code>dac_periph</code>	DAC外设
<code>DACx</code>	DAC外设选择 ($x = 0$)
输入参数{in}	
<code>dac_out</code>	DAC输出
<code>DAC_OUTx</code>	DAC输出通道选择 ($x = 0,1$)
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 3-111. 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 3-112. 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 3-113. 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 3-114. 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
<i>DACx</i>	DAC外设选择 (x = 0)
输入参数{in}	
dac_out	DAC输出
<i>DAC_OUTx</i>	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 3-115. 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
<i>DACx</i>	DAC外设选择 (x = 0)
输入参数{in}	
dac_out	DAC输出
<i>DAC_OUTx</i>	DAC输出通道选择 (x = 0,1)
输入参数{in}	
dac_align	DAC对齐模式
<i>DAC_ALIGN_12B_R</i>	12位数据右对齐
<i>DAC_ALIGN_12B_L</i>	12位数据左对齐
<i>DAC_ALIGN_8B_R</i>	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 3-116. 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 3-117. 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
<i>DACx</i>	DAC外设选择 (x = 0)
输入参数{in}	
dac_out	DAC输出
<i>DAC_OUTx</i>	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 3-118. 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
<i>DACx</i>	DAC外设选择 (x = 0)
输入参数{in}	
dac_out	DAC输出
<i>DAC_OUTx</i>	DAC输出通道选择 (x = 0,1)
输入参数{in}	
triggersource	DAC触发源
<i>DAC_TRIGGER_T5_TRGO</i>	TIMER5 TRGO
<i>DAC_TRIGGER_T2_TRGO</i>	TIMER2 TRGO, 只对GD32F10X_CL系列有效
<i>DAC_TRIGGER_T7_TRGO</i>	TIMER7 TRGO, 只对GD32F10X_MD, GD32F10X_HD, GD32F10X_XD系列有效
<i>DAC_TRIGGER_T6_TRGO</i>	TIMER6 TRGO
<i>DAC_TRIGGER_T4_TRGO</i>	TIMER4 TRGO

<code>DAC_TRIGGER_T1_TRGO</code>	TIMER1 TRGO
<code>DAC_TRIGGER_T3_TRGO</code>	TIMER3 TRGO
<code>DAC_TRIGGER_EXTI_9</code>	EXTI线9中断
<code>DAC_TRIGGER_SOFTWARE</code>	软件触发
输出参数{out}	
-	-
返回值	
-	-

例如:

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

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

函数 `dac_software_trigger_enable`

函数 `dac_software_trigger_enable` 描述见下表:

表 3-119. 函数 `dac_software_trigger_enable`

函数名称	<code>dac_software_trigger_enable</code>
函数原型	<code>void dac_software_trigger_enable(uint32_t dac_periph, uint8_t dac_out);</code>
功能描述	DAC软件触发使能
先决条件	-
被调用函数	-
输入参数{in}	
<code>dac_periph</code>	DAC外设
<code>DACx</code>	DAC外设选择 (x = 0)
输入参数{in}	
<code>dac_out</code>	DAC输出
<code>DAC_OUTx</code>	DAC输出通道选择 (x = 0,1)
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 3-120. 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 3-121. 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
<i>DACx</i>	DAC外设选择 (x = 0)
输入参数{in}	
dac_out	DAC输出
<i>DAC_OUTx</i>	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
<i>DAC_LFSR_BIT0</i>	LFSR模式位0非屏蔽
<i>DAC_LFSR_BITSx_0</i>	LFSR模式位[x:0]非屏蔽 (x = 1,2,3..11)
输出参数{out}	
-	-
返回值	
-	-

例如:

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

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

函数 **dac_triangle_noise_config**

函数dac_triangle_noise_config描述见下表:

表 3-122. 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
<i>DACx</i>	DAC外设选择 (x = 0)
输入参数{in}	
dac_out	DAC输出
<i>DAC_OUTx</i>	DAC输出通道选择 (x = 0,1)
输入参数{in}	
amplitude	三角波幅值
<i>DAC_TRIANGLE_AMPLITUDE_x</i>	$x = 2^n - 1$ (n = 1..12)
输出参数{out}	

-	-
返回值	
-	-

例如:

```
/* configure 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 3-123. 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 3-124. 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 3-125. 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 3-126. 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 3-127. 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 3-128. 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
<i>DAC_ALIGN_12B_R</i>	12位数据右对齐
<i>DAC_ALIGN_12B_L</i>	12位数据左对齐
<i>DAC_ALIGN_8B_R</i>	8位数据右对齐
输入参数{in}	
data0	写入DAC_OUT0的数据（0~4095）
输入参数{in}	
data1	写入DAC_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 3-129. 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 3-130. 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

库函数名称	库函数描述
dbg_periph_disable	禁能外设的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
dbg_trace_pin_mode_set	配置跟踪引脚分配模式

枚举类型 dbg_periph_enum

表 3-131. 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_CAN0_HOLD	当内核停止时，CAN0接收寄存器停止接收数据
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_CAN1_HOLD	当内核停止时，CAN1接收寄存器停止接收数据（只适用于GD32F10x CL系列）
DBG_TIMER11_HOLD	当内核停止时，保持TIMER11计数器计数值不变（只适用于GD32F10x CL和XD系列）
DBG_TIMER12_HOLD	当内核停止时，保持TIMER12计数器计数值不变（只适用于GD32F10x CL和XD系列）
DBG_TIMER13_HOLD	当内核停止时，保持TIMER13计数器计数值不变（只适用于GD32F10x CL和XD系列）
DBG_TIMER8_HOLD	当内核停止时，保持TIMER8计数器计数值不变（只适用于GD32F10x CL和XD系列）
DBG_TIMER9_HOLD	当内核停止时，保持TIMER9计数器计数值不变（只适用于GD32F10x CL和XD系列）
DBG_TIMER10_HOLD	当内核停止时，保持TIMER10计数器计数值不变（只适用于GD32F10x CL和XD系列）

函数 dbg_id_get

函数dbg_id_get描述见下表:

表 3-132. 函数 dbg_id_get

函数名称	dbg_id_get
------	------------

函数原形	uint32_t dbg_id_get(void);
功能描述	读DBG_ID寄存器
先决条件	-
被调用函数	-
输入参数{in}	
-	-
输出参数{out}	
-	-
返回值	
uint32_t	DBG ID (0-0xFFFFFFFF)

例如:

```
/* read DBG_ID code register */
```

```
uint32_t id_value = 0;
```

```
id_value = dbg_id_get();
```

函数 dbg_low_power_enable

函数dbg_low_power_enable描述见下表:

表 3-133. 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 3-134. 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 3-135. 函数 dbg_periph_enable

函数名称	dbg_periph_enable
函数原形	void dbg_periph_enable(dbg_periph_enum dbg_periph);
功能描述	使能外设的MCU调试保持功能
先决条件	-
被调用函数	-
输入参数{in}	
dbg_periph	请参考 表3-131. 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_CANx_HOLD	当内核停止时, CANx接收寄存器停止接收数据 (x=0,1, 但CAN1只适用于

GD32F10x CL系列)	
<i>DBG_I2Cx_HOLD</i>	当内核停止时，保持I2Cx (x=0,1) 的SMBUS状态不变，用于调试
<i>DBG_TIMERx_HOLD</i>	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, 但TIMER8..13只适用于GD32F10x CL和XD系列)
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 3-136. 函数 dbg_periph_disable

函数名称	dbg_periph_disable
函数原形	void dbg_periph_disable(dbg_periph_enum dbg_periph);
功能描述	禁能外设的MCU调试保持功能
先决条件	-
被调用函数	-
输入参数{in}	
dbg_periph	请参考 表3-131. 枚举类型dbg_periph_enum
<i>DBG_FWDGT_HOLD</i>	当内核停止时，保持FWDGT计数器时钟
<i>DBG_WWDGT_HOLD</i>	当内核停止时，保持WWDGT计数器时钟
<i>DBG_CANx_HOLD</i>	当内核停止时，CANx接收寄存器停止接收数据 (x=0,1, 但CAN1只适用于GD32F10x CL系列)
<i>DBG_I2Cx_HOLD</i>	当内核停止时，保持I2Cx (x=0,1) 的SMBUS状态不变，用于调试
<i>DBG_TIMERx_HOLD</i>	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, 但TIMER8..13只适用于GD32F10x CL和XD系列)
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 3-137. 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 3-138. 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();
```

函数 dbg_trace_pin_mode_set

函数dbg_trace_pin_mode_set描述见下表：

表 3-139. 函数 dbg_trace_pin_mode_set

函数名称	dbg_trace_pin_mode_set
函数原形	void dbg_trace_pin_mode_set(uint32_t trace_mode);
功能描述	配置跟踪引脚分配模式
先决条件	-
被调用函数	-
输入参数{in}	
trace_mode	跟踪引脚分配模式选择
TRACE_MODE_ASYNC	跟踪引脚用于异步模式
TRACE_MODE_SYNC_DATASIZE_1	跟踪引脚用于同步模式且数据长度为1
TRACE_MODE_SYNC_DATASIZE_2	跟踪引脚用于同步模式且数据长度为2
TRACE_MODE_SYNC_DATASIZE_4	跟踪引脚用于同步模式且数据长度为4
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* trace pin mode selection */
```

```
dbg_trace_pin_mode_set(TRACE_MODE_ASYNC);
```

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 3-140. 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 3-141. 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	初始化DMA结构体为默认值
dma_init	初始化外设DMAx的通道y
dma_circulation_enable	使能DMA循环模式
dma_circulation_disable	禁能DMA循环模式
dma_memory_to_memory_enable	使能存储器到存储器DMA传输
dma_memory_to_memory_disable	禁能存储器到存储器DMA传输
dma_channel_enable	使能外设DMAx的通道y传输
dma_channel_disable	使能外设DMAx的通道y传输禁能
dma_periph_address_config	配置DMAx通道y传输的外设基地址
dma_memory_address_config	配置DMAx通道y传输的存储器基地址
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
dma_priority_config	配置DMAx通道y的传输软件优先级
dma_memory_width_config	配置DMAx通道y传输的存储器数据宽度
dma_periph_width_config	配置DMAx通道y传输的外设数据宽度
dma_memory_increase_enable	使能DMAx通道y传输的存储器地址生成算法增量模式
dma_memory_increase_disable	禁能DMAx通道y传输的存储器地址生成算法增量模式
dma_periph_increase_enable	使能DMAx通道y传输的外设地址生成算法增量模式
dma_periph_increase_disable	禁能DMAx通道y传输的外设地址生成算法增量模式

库函数名称	库函数描述
<code>dma_transfer_direction_config</code>	配置DMAx通道y的传输方向
<code>dma_flag_get</code>	获取DMAx通道y标志位状态
<code>dma_flag_clear</code>	清除DMAx通道y标志位状态
<code>dma_interrupt_flag_get</code>	获取DMAx通道y中断标志位状态
<code>dma_interrupt_flag_clear</code>	清除DMAx通道y中断标志位状态
<code>dma_interrupt_enable</code>	使能DMAx通道y中断
<code>dma_interrupt_disable</code>	禁能DMAx通道y中断

结构体 `dma_parameter_struct`

表 3-142. 结构体 `dma_parameter_struct`

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

函数 `dma_deinit`

函数`dma_deinit`描述见下表：

表 3-143. 函数 `dma_deinit`

函数名称	<code>dma_deinit</code>
函数原型	<code>void dma_deinit(uint32_t dma_periph, dma_channel_enum channelx);</code>
功能描述	复位外设DMAx的通道y的所有寄存器
先决条件	-
被调用函数	-
输入参数{in}	
<code>dma_periph</code>	DMA外设
<code>DMAx(x=0,1)</code>	DMA外设选择
输入参数{in}	
<code>channelx</code>	DMA通道
<code>DMA_CHx(DMA0:x=0..6; DMA1:x=0..4)</code>	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_deinit

函数dma_struct_para_deinit描述见下表：

表 3-144. 函数 dma_deinit

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	初始化DMA结构体为默认值
先决条件	-
被调用函数	-
输入参数{in}	
init_struct	DMA通道的默认初始值，结构体成员参考 表3-142. 结构体 dma_parameter_struct
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* initialize the parameters of DMA struct with the default values */
dma_parameter_struct init_struct;
dma_struct_para_deinit(&init_struct);
```

函数 dma_init

函数dma_init描述见下表：

表 3-145. 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	-
被调用函数	-
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

<i>DMA_CHx(DMA0:x =0..6; DMA1: x=0..4)</i>	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3-142. 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_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 3-146. 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	使能DMA循环模式
先决条件	-
被调用函数	-
输入参数{in}	
dma_periph	DMA外设
<i>DMAx(x=0,1)</i>	DMA外设选择
输入参数{in}	
channelx	DMA通道
<i>DMA_CHx(DMA0:x =0..6; DMA1: x=0..4)</i>	DMA通道选择

输出参数{out}	
-	-
返回值	
-	-

例如：

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

函数 dma_circulation_disable

函数dma_circulation_disable描述见下表：

表 3-147. 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	禁能DMA循环模式
先决条件	-
被调用函数	-
输入参数{in}	
dma_periph	DMA外设
<i>DMAx(x=0,1)</i>	DMA外设选择
输入参数{in}	
channelx	DMA通道
<i>DMA_CHx(DMA0:x=0..6; DMA1:x=0..4)</i>	DMA通道选择
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* DMA0 channel0 mode configuration */
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 3-148. 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	使能存储器到存储器DMA传输
先决条件	-
被调用函数	-
输入参数{in}	
dma_periph	DMA外设
<i>DMAx(x=0,1)</i>	DMA外设选择
输入参数{in}	
channelx	DMA通道
<i>DMA_CHx(DMA0:x =0..6; DMA1: x=0..4)</i>	DMA通道选择
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* DMA0 channel0 mode configuration */
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 3-149. 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	禁能存储器到存储器DMA传输
先决条件	-
被调用函数	-
输入参数{in}	
dma_periph	DMA外设
<i>DMAx(x=0,1)</i>	DMA外设选择
输入参数{in}	
channelx	DMA通道
<i>DMA_CHx(DMA0:x =0..6; DMA1: x=0..4)</i>	DMA通道选择
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* DMA0 channel0 mode configuration */
dma_memory_to_memory_enable(DMA0, DMA_CH0);
```

函数 dma_channel_enable

函数dma_channel_enable描述见下表：

表 3-150. 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	使能外设DMAx的通道y传输
先决条件	-
被调用函数	-
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 transfer */
dma_channel_enable(DMA0, DMA_CH0);
```

函数 dma_channel_disable

函数dma_channel_disable描述见下表：

表 3-151. 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	禁用外设DMAx的通道y传输
先决条件	-
被调用函数	-
输入参数{in}	

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

例如：

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

函数 dma_periph_address_config

函数dma_periph_address_config描述见下表：

表 3-152. 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	配置DMAx通道y传输的外设基地址
先决条件	-
被调用函数	-
输入参数{in}	
dma_periph	DMA外设
<i>DMAx(x=0,1)</i>	DMA外设选择
输入参数{in}	
channelx	DMA通道
<i>DMA_CHx(DMA0:x=0..6; DMA1:x=0..4)</i>	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 3-153. 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	配置DMAx通道y传输的存储器基地址
先决条件	-
被调用函数	-
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 3-154. 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	-
被调用函数	-
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 3-155. 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	-
被调用函数	-
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 3-156. 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	配置DMAx通道y的传输软件优先级
先决条件	-
被调用函数	-
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}	
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 3-157. 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 mw idth);
功能描述	配置DMAx通道y传输的存储器数据宽度
先决条件	-
被调用函数	-
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_WIDTH_16BIT	16位数据传输宽度
DMA_MEMORY_WIDTH_32BIT	32位数据传输宽度
输出参数{out}	
-	-
返回值	
-	-

例如:

```
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 3-158. 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	配置DMAx通道y传输的外设数据宽度
先决条件	-
被调用函数	-
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

<i>DMA_CHx(DMA0:x =0..6; DMA1: x=0..4)</i>	DMA通道选择
输入参数{in}	
pwidth	外设数据传输宽度
<i>DMA_PERIPHERAL _WIDTH_8BIT</i>	8位数据传输宽度
<i>DMA_PERIPHERAL _WIDTH_16BIT</i>	16位数据传输宽度
<i>DMA_PERIPHERAL _WIDTH_32BIT</i>	32位数据传输宽度
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 3-159. 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	使能DMAx通道y传输的存储器地址生成算法增量模式
先决条件	-
被调用函数	-
输入参数{in}	
dma_periph	DMA外设
<i>DMAx(x=0,1)</i>	DMA外设选择
输入参数{in}	
channelx	DMA通道
<i>DMA_CHx(DMA0:x =0..6; DMA1: x=0..4)</i>	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 3-160. 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	禁能DMAx通道y传输的存储器地址生成算法增量模式
先决条件	-
被调用函数	-
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 3-161. 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	使能DMAx通道y传输的外设地址生成算法增量模式
先决条件	-
被调用函数	-
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

<i>DMA_CHx(DMA0:x =0..6; DMA1: x=0..4)</i>	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 3-162. 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	禁能DMAx通道y传输的外设地址生成算法增量模式
先决条件	-
被调用函数	-
输入参数{in}	
dma_periph	DMA 外设
<i>DMAx(x=0,1)</i>	DMA 外设选择
输入参数{in}	
channelx	DMA通道
<i>DMA_CHx(DMA0:x =0..6; DMA1: x=0..4)</i>	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 3-163. 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, uint8_t direction);
功能描述	配置DMAx通道y的传输方向
先决条件	-
被调用函数	-
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_T O_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 3-164. 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	-
被调用函数	-
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

$x=0..4$	
输入参数{in}	
flag	DMA标志
<i>DMA_FLAG_G</i>	DMA通道全局中断标志
<i>DMA_FLAG_FTF</i>	DMA通道传输完成标志
<i>DMA_FLAG_HTF</i>	DMA通道半传输完成标志
<i>DMA_FLAG_ERR</i>	DMA通道错误标志
输出参数{out}	
-	-
返回值	
FlagStatus	SET或RESET

例如：

```
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 3-165. 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	-
被调用函数	-
输入参数{in}	
dma_periph	DMA外设
<i>DMAx(x=0,1)</i>	DMA外设选择
输入参数{in}	
channelx	DMA通道
<i>DMA_CHx(DMA0:x=0..6; DMA1:x=0..4)</i>	DMA通道选择
输入参数{in}	
flag	DMA标志
<i>DMA_FLAG_G</i>	DMA通道全局中断标志
<i>DMA_FLAG_FTF</i>	DMA通道传输完成标志
<i>DMA_FLAG_HTF</i>	DMA通道半传输完成标志
<i>DMA_FLAG_ERR</i>	DMA通道错误标志
输出参数{out}	
-	-

返回值	
-	-

例如：

```
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 3-166. 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	-
被调用函数	-
输入参数{in}	
dma_periph	DMA外设
<i>DMAx(x=0,1)</i>	DMA外设选择
输入参数{in}	
channelx	DMA通道
<i>DMA_CHx(DMA0:x=0..6; DMA1:x=0..4)</i>	DMA通道选择
输入参数{in}	
flag	DMA标志
<i>DMA_INT_FLAG_FTF</i>	DMA通道传输完成中断标志
<i>DMA_INT_FLAG_HTF</i>	DMA通道半传输完成中断标志
<i>DMA_INT_FLAG_ERR</i>	DMA通道错误中断标志
输出参数{out}	
-	-
返回值	
FlagStatus	SET或RESET

例如：

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

函数 dma_interrupt_flag_clear

函数dma_interrupt_flag_clear描述见下表:

表 3-167. 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	-
被调用函数	-
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_F TF	DMA通道传输完成中断标志
DMA_INT_FLAG_H TF	DMA通道半传输完成中断标志
DMA_INT_FLAG_E RR	DMA通道错误中断标志
输出参数{out}	
-	-
返回值	
-	-

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 3-168. 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	使能DMAx通道y中断
先决条件	-
被调用函数	-
输入参数{in}	
dma_periph	DMA外设
<i>DMAx(x=0,1)</i>	DMA外设选择
输入参数{in}	
channelx	DMA通道
<i>DMA_CHx(DMA0:x=0..6; DMA1:x=0..4)</i>	DMA通道选择
输入参数{in}	
source	DMA中断源
<i>DMA_INT_FTF</i>	DMA通道传输完成中断
<i>DMA_INT_HTF</i>	DMA通道半传输完成中断
<i>DMA_INT_ERR</i>	DMA通道错误中断
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 3-169. 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	禁能DMAx通道y中断
先决条件	-
被调用函数	-
输入参数{in}	
dma_periph	DMA外设
<i>DMAx(x=0,1)</i>	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. ENET

以太网模块包含10/100Mbps以太网MAC（媒体访问控制器），采用DMA优化数据帧的发送与接收性能，支持MII（媒体独立接口）与RMII（简化的媒体独立接口）两种与物理层(PHY)通讯的标准接口，实现以太网数据帧的发送与接收。章节[3.9.1](#)描述了ENET的寄存器列表，章节[3.9.2](#)对ENET库函数进行说明。

3.9.1. 外设寄存器描述

ENET寄存器列表如下表所示:

表 3-170. ENET 寄存器

寄存器名称	寄存器描述
ENET_MAC_CFG	MAC配置寄存器
ENET_MAC_FRMF	MAC帧过滤器寄存器
ENET_MAC_HLH	MAC hash列表高寄存器
ENET_MAC_HLL	MAC hash列表低寄存器
ENET_MAC_PHY_CTL	MAC PHY控制寄存器
ENET_MAC_PHY_DATA	MAC PHY数据寄存器
ENET_MAC_FCTL	MAC流控寄存器
ENET_MAC_FCTH	MAC流控阈值寄存器
ENET_MAC_VLT	MAC VLAN标签寄存器
ENET_MAC_RWFF	MAC远程唤醒帧过滤器寄存器
ENET_MAC_WUM	MAC唤醒管理寄存器

寄存器名称	寄存器描述
ENET_MAC_INTF	MAC中断状态寄存器
ENET_MAC_INTMSK	MAC中断屏蔽寄存器
ENET_MAC_ADDR0H	MAC地址0高寄存器
ENET_MAC_ADDR0L	MAC地址0低寄存器
ENET_MAC_ADDR1H	MAC地址1高寄存器
ENET_MAC_ADDR1L	MAC地址1低寄存器
ENET_MAC_ADDR2H	MAC地址2高寄存器
ENET_MAC_ADDR2L	MAC地址2低寄存器
ENET_MAC_ADDR3H	MAC地址3高寄存器
ENET_MAC_ADDR3L	MAC地址3低寄存器
ENET_MSC_CTL	MSC控制寄存器
ENET_MSC_RINTF	MSC接收中断状态寄存器
ENET_MSC_TINTF	MSC发送中断状态寄存器
ENET_MSC_RINTMSK	MSC接收中断屏蔽寄存器
ENET_MSC_TINTMSK	MSC发送中断屏蔽寄存器
ENET_MSC_SCCNT	MSC 1次冲突后发送”好”帧的计数器寄存器
ENET_MSC_MSCCNT	MSC 1次以上冲突后发送”好”帧的计数器寄存器
ENET_MSC_TGFCNT	MSC发送”好”帧计数器寄存器
ENET_MSC_RFCECNT	MSC CRC错误接收帧计数器寄存器
ENET_MSC_RFAECNT	MSC对齐错误接收帧计数器寄存器
ENET_MSC_RGUFcnt	MSC “好”单播帧接收帧计数器寄存器
ENET_PTP_TSCTL	PTP时间戳控制寄存器
ENET_PTP_SSINC	PTP亚秒递增寄存器
ENET_PTP_TSH	PTP时间戳高寄存器
ENET_PTP_TSL	PTP时间戳低寄存器

寄存器名称	寄存器描述
ENET_PTP_TSUH	PTP时间戳高更新寄存器
ENET_PTP_TSUL	PTP时间戳低更新寄存器
ENET_PTP_TSADD END	PTP时间戳加数寄存器
ENET_PTP_ETH	PTP期望时间高寄存器
ENET_PTP_ETL	PTP期望时间低寄存器
ENET_DMA_BCTL	DMA总线控制寄存器
ENET_DMA_TPEN	DMA发送查询使能寄存器
ENET_DMA_RPEN	DMA接收查询使能寄存器
ENET_DMA_RDTA DDR	DMA接收描述符列表地址寄存器
ENET_DMA_TDTA DDR	DMA发送描述符列表地址寄存器
ENET_DMA_STAT	DMA状态寄存器
ENET_DMA_CTL	DMA控制寄存器
ENET_DMA_INTEN	DMA中断使能寄存器
ENET_DMA_MFBO CNT	DMA丢失帧和缓存溢出计数器寄存器
ENET_DMA_CTDA DDR	DMA当前发送描述符地址寄存器
ENET_DMA_CRDA DDR	DMA当前接收描述符地址寄存器
ENET_DMA_CTBA DDR	DMA当前发送缓存地址寄存器
ENET_DMA_CRBA DDR	DMA当前接收缓存地址寄存器

3.9.2. 外设库函数说明

ENET库函数列表如下表所示:

表 3-171. ENET 库函数

库函数名称	库函数描述
常用函数	
enet_deinit	复位ENET模块及相关软件初始化所需结构体
enet_initpara_config	配置ENET模块的各类不常用功能。当enet_init()函数无法满足所需实现功能时调用，必须在enet_init()函数之前调用
enet_init	ENET模块初始化，配置用户最关心的功能
enet_software_reset	复位ENET寄存器，并检测CLK_TX/CLK_RX信号
enet_rxfraze_size_get	检测接收帧是否有错误，正确时返回帧长度
enet_descriptors_chain_init	初始化DMA接收/发送描述符为链模式
enet_descriptors_ring_init	初始化DMA接收/发送描述符为环模式

库函数名称	库函数描述
enet_frame_receive	处理当前接收到的帧，并将当前描述符中存储的接收帧数据拷贝到指定区域
enet_frame_transmit	将指定区域内的数据拷贝到当前发送描述符中，并发送
enet_transmit_checksum_config	配置发送帧校验和模式
enet_enable	ENET Tx/Rx功能使能（包括ENET外设内的MAC和DMA模块）
enet_disable	ENET Tx/Rx功能禁能（包括ENET外设内的MAC和DMA模块）
enet_mac_address_set	配置MAC地址
enet_mac_address_get	获取MAC地址
enet_flag_get	获取ENET模块MAC/MSR/PTP/DMA状态标志位
enet_flag_clear	清除ENET状态标志位
enet_interrupt_enable	使能ENET模块MAC/MSR/DMA中断
enet_interrupt_disable	禁能ENET模块MAC/MSR/DMA中断
enet_interrupt_flag_get	获取ENET模块MAC/MSR/DMA中断标志位
enet_interrupt_flag_clear	禁能ENET中断标志位
MAC功能函数	
enet_tx_enable	ENET发送功能使能（包括ENET外设内的MAC和DMA模块）
enet_tx_disable	ENET发送功能禁能（包括ENET外设内的MAC和DMA模块）
enet_rx_enable	ENET接收功能使能（包括ENET外设内的MAC和DMA模块）
enet_rx_disable	ENET接收功能禁能（包括ENET外设内的MAC和DMA模块）
enet_registers_get	获取指定范围ENET寄存器值
enet_address_filter_enable	MAC地址过滤器使能
enet_address_filter_disable	MAC地址过滤器禁能
enet_address_filter_config	配置MAC地址过滤器模式
enet_phy_config	PHY接口配置（配置SMI时钟并复位PHY芯片）
enet_phy_write_read	写/读PHY寄存器
enet_phyloopback_enable	使能PHY芯片回环模式
enet_phyloopback_disable	禁能PHY芯片回环模式
enet_forward_feature_enable	使能ENET帧通过相关功能
enet_forward_feature_disable	禁能ENET帧通过相关功能
enet_filter_feature_enable	使能ENET帧过滤器相关功能
enet_filter_feature_disable	禁能ENET帧过滤器相关功能
流控功能函数	
enet_pauseframe_generate	生成暂停帧，使能发送流控功能后ENET模块将发送暂停帧
enet_pauseframe_detect_config	配置暂停帧检测类型
enet_pauseframe_config	配置暂停帧参数

库函数名称	库函数描述
enet_flow control_threshold_config	配置流控阈值
enet_flow control_feature_enable	使能ENET流控相关功能
enet_flow control_feature_disable	禁能ENET流控相关功能
DMA 功能函数	
enet_dmaprocess_state_get	获取DMA发送/接收流程状态
enet_dmaprocess_resume	DMA发送/接收查询使能
enet_rxprocess_check_recovery	检测并恢复接收流程
enet_txfifo_flush	刷新ENET发送FIFO，并等待刷新操作完成
enet_current_desc_address_get	获取当前发送/接收描述符地址、当前缓冲区地址、描述符列表首地址
enet_desc_information_get	获取发送/接收描述符详细信息
enet_missed_frame_counter_get	获取接收丢弃帧数
描述符功能函数	
enet_desc_flag_get	获取ENET模块DMA描述符标志位
enet_desc_flag_set	设置ENET模块DMA描述符标志位
enet_desc_flag_clear	清除ENET模块DMA描述符标志位
enet_desc_receive_complete_bit_enable	当接收完成时，ENET_DMA_STAT寄存器的RS位将被置位
enet_desc_receive_complete_bit_disable	当接收完成时，ENET_DMA_STAT寄存器的RS位将不会被置位
enet_rxframe_drop	丢弃当前接收到的帧
enet_dma_feature_enable	使能ENET模块DMA相关功能
enet_dma_feature_disable	禁能ENET模块DMA相关功能
enet_ptp_normal_descriptors_chain_init	初始化具有PTP功能的DMA接收/发送描述符为链模式
enet_ptp_normal_descriptors_ring_init	初始化具有PTP功能的DMA接收/发送描述符为环模式
enet_ptpframe_receive_normal_mode	在PTP模式下处理当前接收到的帧，并将当前描述符中存储的接收帧数据和时间戳拷贝到指定区域
enet_ptpframe_transmit_normal_mode	在PTP模式下将指定区域内的数据拷贝到当前发送描述符中，并同时时间戳一起发送
WUM功能函数	
enet_w um_filter_register_pointer_reset	远程唤醒帧过滤器寄存器指针复位
enet_w um_filter_config	配置远程唤醒帧寄存器
enet_w um_feature_enable	使能ENET模块唤醒管理相关功能
enet_w um_feature_disable	禁能ENET模块唤醒管理相关功能
MSC功能函数	
enet_msc_counters_reset	复位MAC统计计数器组
enet_msc_feature_enable	使能MAC统计计数器相关功能
enet_msc_feature_disable	禁能MAC统计计数器相关功能

库函数名称	库函数描述
enet_msc_counters_get	获取MAC相关统计计数器值
PTP功能函数	
enet_ptp_subsecond_2_nanosecond	亚秒到纳秒的转换
enet_ptp_nanosecond_2_subsecond	纳秒到亚秒的转换
enet_ptp_feature_enable	使能PTP相关功能
enet_ptp_feature_disable	禁能PTP相关功能
enet_ptp_timestamp_function_config	配置PTP时间戳相关功能
enet_ptp_subsecond_increment_config	配置PTP系统时间亚秒增加值
enet_ptp_timestamp_addend_config	精调模式下PTP时钟频率校准配置
enet_ptp_timestamp_update_config	初始化时用于替换系统时间，在更新时表示在系统时间上加上或减去的秒值
enet_ptp_expected_time_config	配置PTP期望时间
enet_ptp_system_time_get	获取PTP当前系统时间
enet_ptp_start	配置并启动PTP时间戳计数器
enet_ptp_finecorrection_adjfreq	在精调模式下通过配置加数寄存器校准频率
enet_ptp_coarsecorrection_systime_update	粗调模式下更新系统时间
enet_ptp_finecorrection_settime	精调模式下配置系统时间
enet_ptp_flag_get	获取PTP标志位状态
其它	
enet_initpara_reset	复位 ENET initpara struct, 需在enet_initpara_config()函数前调用

结构体 enet_descriptors_struct

表 3-172. 结构体 enet_descriptors_struct

成员名称	功能描述
status	描述符状态位
control_buffer_size	描述符控制位及缓冲区1、2长度
buffer1_addr	缓冲区1地址指针/时间戳低
buffer2_next_desc_addr	缓冲区2或下一描述符地址指针/时间戳高

结构体 enet_ptp_systime_struct

表 3-173. 结构体 enet_ptp_systime_struct

成员名称	功能描述
------	------

second	系统时间（单位秒）
nanosecond	系统时间（单位纳秒）
sign	系统时间符号位

enet_deinit

函数enet_deinit描述见下表：

表 3-174. 函数 enet_deinit

函数名称	enet_deinit
函数原型	void enet_deinit(void);
功能描述	复位ENET模块及相关软件初始化所需结构体
先决条件	-
被调用函数	rcu_periph_reset_enable()/rcu_periph_reset_disable()/enet_initpara_reset()
输入参数{in}	
-	-
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* ENET deinitialize */
enet_deinit();
```

enet_initpara_config

函数enet_initpara_config描述见下表：

表 3-175. 函数 enet_initpara_config

函数名称	enet_initpara_config
函数原型	void enet_initpara_config(enet_option_enum option, uint32_t para)
功能描述	配置ENET模块的各类不常用功能。当enet_init()函数无法满足所需实现功能时调用，必须在enet_init()函数之前调用
先决条件	enet_initpara_reset(void)
被调用函数	-
输入参数{in}	
option	ENET模块功能选项，根据选择需要使用不同参数进行配置，下列参数仅可选择一个
FORWARD_OPTION	选择配置帧通过功能相关参数
DMABUS_OPTION	选择配置DMA总线模式相关参数
DMA_MAXBURST_OPTION	选择配置DMA最大突发传输相关参数

DMA_ARBITRATION_OPTION	选择配置DMA仲裁相关参数
STORE_OPTION	选择配置存储转发模式相关参数
DMA_OPTION	选择配置DMA相关参数
VLAN_OPTION	选择配置VLAN相关参数
FLOWCTL_OPTION	选择配置流控相关参数
HASHH_OPTION	选择配置HASH_H相关参数
HASHL_OPTION	选择配置HASH_L相关参数
FILTER_OPTION	选择配置帧过滤器相关参数
HALFDUPLEX_OPTION	选择配置半双工模式相关参数
TIMER_OPTION	选择配置计数器相关参数
INTERFRAMEGAP_OPTION	选择配置帧间隔相关参数
输入参数{in}	
para (该参数值需根据option参数对应值进行选择)	下列参数值均可以被配置 例如: para = (value1 value2 value3...)
当option参数值为FORWARD_OPTION时	
value1	ENET_AUTO_PADCRC_DROP_ENABLE / ENET_AUTO_PADCRC_DROP_DISABLE
value2	ENET_FORWARD_ERRFRAMES_ENABLE / ENET_FORWARD_ERRFRAMES_DISABLE
value3	ENET_FORWARD_UNDERSZ_GOODFRAMES_ENABLE / ENET_FORWARD_UNDERSZ_GOODFRAMES_DISABLE
当option参数值为DMABUS_OPTION时	
value1	ENET_ADDRESS_ALIGN_ENABLE / ENET_ADDRESS_ALIGN_DISABLE
value2	ENET_FIXED_BURST_ENABLE / ENET_FIXED_BURST_DISABLE
当option参数值为DMA_MAXBURST_OPTION时	
value1	ENET_RXDP_1BEAT / ENET_RXDP_2BEAT / ENET_RXDP_4BEAT / ENET_RXDP_8BEAT / ENET_RXDP_16BEAT / ENET_RXDP_32BEAT / ENET_RXDP_4xPGBL_4BEAT / ENET_RXDP_4xPGBL_8BEAT / ENET_RXDP_4xPGBL_16BEAT / ENET_RXDP_4xPGBL_32BEAT / ENET_RXDP_4xPGBL_64BEAT / ENET_RXDP_4xPGBL_128BEAT
value2	ENET_PGBL_1BEAT / ENET_PGBL_2BEAT / ENET_PGBL_4BEAT / ENET_PGBL_8BEAT / ENET_PGBL_16BEAT / ENET_PGBL_32BEAT / ENET_PGBL_4xPGBL_4BEAT / ENET_PGBL_4xPGBL_8BEAT / ENET_PGBL_4xPGBL_16BEAT / ENET_PGBL_4xPGBL_32BEAT / ENET_PGBL_4xPGBL_64BEAT / ENET_PGBL_4xPGBL_128BEAT
value3	ENET_RXTX_DIFFERENT_PGBL / ENET_RXTX_SAME_PGBL
当option参数值为DMA_ARBITRATION_OPTION时	
value1	ENET_ARBITRATION_RXPRIORTX / ENET_ARBITRATION_RXTX_1_1 /

	ENET_ARBITRATION_RXTX_2_1 / ENET_ARBITRATION_RXTX_3_1 / ENET_ARBITRATION_RXTX_4_1
当 option 参数值为STORE_OPTION时	
value1	ENET_RX_MODE_STOREFORWARD / ENET_RX_MODE_CUTTHROUGH
value2	ENET_TX_MODE_STOREFORWARD / ENET_TX_MODE_CUTTHROUGH
value3	ENET_RX_THRESHOLD_64BYTES / ENET_RX_THRESHOLD_32BYTES / ENET_RX_THRESHOLD_96BYTES / ENET_RX_THRESHOLD_128BYTES
value4	ENET_TX_THRESHOLD_64BYTES / ENET_TX_THRESHOLD_128BYTES / ENET_TX_THRESHOLD_192BYTES / ENET_TX_THRESHOLD_256BYTES / ENET_TX_THRESHOLD_40BYTES / ENET_TX_THRESHOLD_32BYTES / ENET_TX_THRESHOLD_24BYTES / ENET_TX_THRESHOLD_16BYTES
当 option 参数值为DMA_OPTION时	
value1	ENET_FLUSH_RXFRAME_ENABLE / ENET_FLUSH_RXFRAME_DISABLE
value2	ENET_SECONDFRAME_OPT_ENABLE / ENET_SECONDFRAME_OPT_DISABLE
当 option 参数值为VLAN_OPTION时	
value1	ENET_VLANTAGCOMPARISON_12BIT/ ENET_VLANTAGCOMPARISON_16BIT
value2	MAC_VLT_VLTI(regval)
当 option 参数值为FLOWCTL_OPTION时	
value1	MAC_FCTL_PTM(regval)
value2	ENET_ZERO_QUANTA_PAUSE_ENABLE / ENET_ZERO_QUANTA_PAUSE_DISABLE
value3	ENET_PAUSETIME_MINUS4 / ENET_PAUSETIME_MINUS28 / ENET_PAUSETIME_MINUS144/ENET_PAUSETIME_MINUS256
value4	ENET_MAC0_AND_UNIQUE_ADDRESS_PAUSEDetect / ENET_UNIQUE_PAUSEDetect
value5	ENET_RX_FLOWCONTROL_ENABLE / ENET_RX_FLOWCONTROL_DISABLE
value6	ENET_TX_FLOWCONTROL_ENABLE / ENET_TX_FLOWCONTROL_DISABLE
value7	ENET_ACTIVE_THRESHOLD_256BYTES / ENET_ACTIVE_THRESHOLD_512BYTES / ENET_ACTIVE_THRESHOLD_768BYTES / ENET_ACTIVE_THRESHOLD_1024BYTES / ENET_ACTIVE_THRESHOLD_1280BYTES / ENET_ACTIVE_THRESHOLD_1536BYTES / ENET_ACTIVE_THRESHOLD_1792BYTES
value8	ENET_DEACTIVE_THRESHOLD_256BYTES / ENET_DEACTIVE_THRESHOLD_512BYTES / ENET_DEACTIVE_THRESHOLD_768BYTES / ENET_DEACTIVE_THRESHOLD_1024BYTES / ENET_DEACTIVE_THRESHOLD_1280BYTES /

	ENET_DEACTIVE_THRESHOLD_1536BYTES / ENET_DEACTIVE_THRESHOLD_1792BYTES
当option参数值为HASHH_OPTION时	
value1	0x0~0xFFFF FFFFU
当option参数值为HASHL_OPTION时	
value1	0x0~0xFFFF FFFFU
当option参数值为FILTER_OPTION时	
value1	ENET_SRC_FILTER_NORMAL_ENABLE / ENET_SRC_FILTER_INVERSE_ENABLE / ENET_SRC_FILTER_DISABLE
value2	ENET_DEST_FILTER_INVERSE_ENABLE / ENET_DEST_FILTER_INVERSE_DISABLE
value3	ENET_MULTICAST_FILTER_HASH_OR_PERFECT / ENET_MULTICAST_FILTER_HASH / ENET_MULTICAST_FILTER_PERFECT / ENET_MULTICAST_FILTER_NONE
value4	ENET_UNICAST_FILTER_EITHER / ENET_UNICAST_FILTER_HASH / ENET_UNICAST_FILTER_PERFECT
value5	ENET_PCFRM_PREVENT_ALL / ENET_PCFRM_PREVENT_PAUSEFRAME / ENET_PCFRM_FORWARD_ALL / ENET_PCFRM_FORWARD_FILTERED
当option参数值为HALFDUPLEX_OPTION时	
value1	ENET_CARRIERSENSE_ENABLE / ENET_CARRIERSENSE_DISABLE
value2	ENET_RECEIVEOWN_ENABLE / ENET_RECEIVEOWN_DISABLE
value3	ENET_RETRYTRANSMISSION_ENABLE / ENET_RETRYTRANSMISSION_DISABLE
value4	ENET_BACKOFFLIMIT_10 / ENET_BACKOFFLIMIT_8 / ENET_BACKOFFLIMIT_4 / ENET_BACKOFFLIMIT_1
value5	ENET_DEFERRALCHECK_ENABLE / ENET_DEFERRALCHECK_DISABLE
当option参数值为TIMER_OPTION时	
value1	ENET_WATCHDOG_ENABLE / ENET_WATCHDOG_DISABLE
value2	ENET_JABBER_ENABLE / ENET_JABBER_DISABLE
当option参数值为INTERFRAMEGAP_OPTION时	
value1	ENET_INTERFRAMEGAP_96BIT / ENET_INTERFRAMEGAP_88BIT / ENET_INTERFRAMEGAP_80BIT / ENET_INTERFRAMEGAP_72BIT / ENET_INTERFRAMEGAP_64BIT / ENET_INTERFRAMEGAP_56BIT / ENET_INTERFRAMEGAP_48BIT / ENET_INTERFRAMEGAP_40BIT
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* configure ENET initialization parameters */
```

```
enet_initpara_reset();
```

```
enet_initpara_config(DMA_OPTION, ENET_FLUSH_RXFRAME_ENABLE | ENET_SECONDFRAME_OPT_ENABLE | ENET_NORMAL_DESCRIPTOR);
```

enet_init

函数enet_init描述见下表:

表 3-176. 函数 enet_init

函数名称	enet_init
函数原型	ErrStatus enet_init(enet_mediamode_enum mediamode, enet_chksumconf_enum checksum, enet_frmrecept_enum recept);
功能描述	ENET模块初始化，配置用户最关心的功能
先决条件	enet_deinit ()
被调用函数	enet_phy_config()/enet_phy_write_read()
输入参数{in}	
mediamode	ENET通讯方式配置，仅可选择唯一参数
ENET_AUTO_NEGOTIATION	PHY自协商
ENET_100M_FULLDUPLEX	100Mbit/s, 全双工
ENET_100M_HALFDUPLEX	100Mbit/s, 半双工
ENET_10M_FULLDUPLEX	10Mbit/s, 全双工
ENET_10M_HALFDUPLEX	10Mbit/s, 半双工
ENET_LOOPBACKMODE	MI模式下的回环模式
输入参数{in}	
checksum	IP帧数据校验和功能，仅可选择唯一参数
ENET_NO_AUTOCHECKSUM	关闭IP帧校验和功能
ENET_AUTOCHECKSUM_DROP_FAILFRAMES	使能IP帧校验和功能
ENET_AUTOCHECKSUM_ACCEPT_FAILFRAMES	使能IP帧校验和功能，不丢弃仅有载荷错误的帧
输入参数{in}	
recept	帧过滤功能，仅可选择唯一参数
ENET_PROMISCUOUS_MODE	使能混杂模式

<i>ENET_RECEIVEALL</i>	接收所有帧
<i>ENET_BROADCAST _FRAMES_PASS</i>	接收广播帧
<i>ENET_BROADCAST _FRAMES_DROP</i>	禁止接收广播帧
输出参数{out}	
-	-
返回值	
ErrStatus	ERROR or SUCCESS

例如：

```
/* initialize ENET */
```

```
ErrStatus enet_init_status;
```

```
enet_init_status = enet_init(ENET_AUTO_NEGOTIATION, ENET_AUTOCHECKSUM_DROP_FAILFRAMES, ENET_BROADCAST_FRAMES_PASS);
```

enet_software_reset

函数enet_software_reset描述见下表：

表 3-177. 函数 enet_software_reset

函数名称	enet_software_reset
函数原型	ErrStatus enet_software_reset(void);
功能描述	复位ENET寄存器，并检测CLK_TX/CLK_RX信号
先决条件	-
被调用函数	-
输入参数{in}	
-	-
输出参数{out}	
-	-
返回值	
ErrStatus	ERROR or SUCCESS

例如：

```
/* reset ENET registers by software */
```

```
ErrStatus reval_state = ERROR;
```

```
reval_state = enet_software_reset();
```

enet_rxframe_size_get

函数enet_rxframe_size_get描述见下表：

表 3-178. 函数 `enet_rxframe_size_get`

函数名称	<code>enet_rxframe_size_get</code>
函数原型	<code>uint32_t enet_rxframe_size_get(void);</code>
功能描述	检测接收帧是否有错误，正确时返回帧长度
先决条件	-
被调用函数	<code>enet_rxframe_drop()</code>
输入参数{in}	
-	-
输出参数{out}	
-	-
返回值	
<code>uint32_t</code>	取值范围：0x0 - 0x3FFF

例如：

```
/* get received frame size */
```

```
uint32_t reval;
```

```
reval = enet_rxframe_size_get();
```

`enet_descriptors_chain_init`

函数`enet_descriptors_chain_init`描述见下表：

表 3-179. 函数 `enet_descriptors_chain_init`

函数名称	<code>enet_descriptors_chain_init</code>
函数原型	<code>void enet_descriptors_chain_init(enet_dmadirection_enum direction);</code>
功能描述	初始化DMA接收/发送描述符为链模式
先决条件	-
被调用函数	-
输入参数{in}	
direction	想要初始化的描述符类型，下列参数仅可选择一个
<code>ENET_DMA_TX</code>	DMA Tx描述符
<code>ENET_DMA_RX</code>	DMA Rx描述符
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* initialize descriptors in chain mode */
```

```
enet_descriptors_chain_init(ENET_DMA_TX);
```

enet_descriptors_ring_init

函数enet_descriptors_ring_init描述见下表：

表 3-180. 函数 enet_descriptors_ring_init

函数名称	enet_descriptors_ring_init
函数原型	void enet_descriptors_ring_init(enet_dmadirection_enum direction);
功能描述	初始化DMA接收/发送描述符为环模式
先决条件	-
被调用函数	-
输入参数{in}	
direction	想要初始化的描述符类型，下列参数仅可选择一个
ENET_DMA_TX	DMA Tx描述符
ENET_DMA_RX	DMA Rx描述符
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* initialize descriptors in ring mode */
enet_descriptors_ring_init(ENET_DMA_TX)
```

enet_frame_receive

函数enet_frame_receive描述见下表：

表 3-181. 函数 enet_frame_receive

函数名称	enet_frame_receive
函数原型	ErrStatus enet_frame_receive(uint8_t *buffer, uint32_t bufsize);
功能描述	处理当前接收到的帧，并将当前描述符中存储的接收帧数据拷贝到指定区域
先决条件	-
被调用函数	-
输入参数{in}	
bufsize	缓冲区长度，范围(0~1524)
输出参数{out}	
buffer-	接收帧数据的缓冲区地址指针。如果输入NULL，用户需要在调用该函数之前将数据拷贝到用户缓冲区内
返回值	
ErrStatus	ERROR or SUCCESS

例如：

```
/* receive frame from descriptor */
```

```
uint8_t data_buffer[1500];

uint32_t data_size;

enet_frame_receive(data_buffer, &data_size);
```

enet_frame_transmit

函数enet_frame_transmit描述见下表:

表 3-182. 函数 enet_frame_transmit

函数名称	enet_frame_transmit
函数原型	ErrStatus enet_frame_transmit(uint8_t *buffer, uint32_t length);
功能描述	将指定区域内的数据拷贝到当前发送描述符中，并发送
先决条件	-
被调用函数	-
输入参数{in}	
buffer	等待发送的帧数据缓冲区地址指针。如果输入NULL，用户需要在调用该函数之前将待发送数据拷贝到描述符指定的位置
输入参数{in}	
length	待发送数据长度，范围(0~1524)
输出参数{out}	
-	-
返回值	
ErrStatus	ERROR or SUCCESS

例如:

```
/* transmit ENET frame */

uint8_t data_buffer[1500];

uint32_t data_size = 800;

enet_frame_transmit (data_buffer, data_size);
```

enet_transmit_checksum_config

函数enet_transmit_checksum_config描述见下表:

表 3-183. 函数 enet_transmit_checksum_config

函数名称	enet_transmit_checksum_config
函数原型	void enet_transmit_checksum_config(enet_descriptors_struct *desc, uint32_t checksum);
功能描述	配置发送帧校验和模式
先决条件	-
被调用函数	-
输入参数{in}	

desc	需要配置的描述符地址指针，结构体成员介绍请参考 表3-172. 结构体 enet_descriptors_struct
输入参数{in}	
checksum	IP帧校验和配置，下列参数仅可选择一个
ENET_CHECKSUM_DISABLE	禁能校验和自动插入
ENET_CHECKSUM_IPV4HEADER	仅使能IP头校验和计算和插入
ENET_CHECKSUM_TCPUDPICMP_SEGMENT	TCP/UDP/ICMP校验和（除去伪报头）计算和插入
ENET_CHECKSUM_TCPUDPICMP_FULL	TCP/UDP/ICMP校验和计算和插入
输出参数{out}	
返回值	

例如：

```
/* configure IP frame checksum offload */
enet_descriptors_struct tx_desc;
enet_transmit_checksum_config(tx_desc, ENET_CHECKSUM_TCPUDPICMP_FULL);
```

enet_enable

函数enet_enable描述见下表：

表 3-184. 函数 enet_enable

函数名称	enet_enable
函数原型	void enet_enable(void);
功能描述	ENET Tx/Rx功能使能（包括ENET外设内的MAC和DMA模块）
先决条件	-
被调用函数	enet_tx_enable()/enet_rx_enable()
输入参数{in}	
-	-
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* enable ENET */
```

```
enet_enable();
```

enet_disable

函数enet_disable描述见下表:

表 3-185. 函数 enet_disable

函数名称	enet_disable
函数原型	void enet_disable(void);
功能描述	ENET Tx/Rx功能禁能（包括ENET外设内的MAC和DMA模块）
先决条件	-
被调用函数	enet_tx_disable()/enet_rx_disable()
输入参数{in}	
-	-
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* disable ENET */
```

```
enet_disable();
```

enet_mac_address_set

函数enet_mac_address_set描述见下表:

表 3-186. 函数 enet_mac_address_set

函数名称	enet_mac_address_set
函数原型	void enet_mac_address_set(enet_macaddress_enum mac_addr, uint8_t paddr[]);
功能描述	配置MAC地址
先决条件	-
被调用函数	-
输入参数{in}	
mac_addr	选择何组MAC地址将被配置，下列参数仅可选择一个
ENET_MAC_ADDRE SS0	配置MAC address 0过滤器
ENET_MAC_ADDRE SS1	配置MAC address 1过滤器
ENET_MAC_ADDRE SS2	配置MAC address 2过滤器
ENET_MAC_ADDRE SS3	配置MAC address 3过滤器

输入参数{in}	
paddr	存储MAC地址的缓冲区指针，小端存储 例如MAC地址为aa:bb:cc:dd:ee:22，缓冲区内数据为{22, ee, dd, cc, bb, aa}
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* set ENET MAC address */
```

```
netif->hwaddr[0] = 0x02;
```

```
netif->hwaddr[1] = 0xaa;
```

```
netif->hwaddr[2] = 0xbb;
```

```
netif->hwaddr[3] = 0xcc;
```

```
netif->hwaddr[4] = 0xdd;
```

```
netif->hwaddr[5] = 0xee;
```

```
enet_mac_address_set(ENET_MAC_ADDRESS0, netif->hwaddr);
```

enet_mac_address_get

函数enet_mac_address_get描述见下表：

表 3-187. 函数 enet_mac_address_get

函数名称	enet_mac_address_get
函数原型	void enet_mac_address_get(enet_macaddress_enum mac_addr, uint8_t paddr[])
功能描述	获取MAC地址
先决条件	-
被调用函数	-
输入参数{in}	
mac_addr	选择何组MAC地址将被配置，下列参数仅可选择一个
ENET_MAC_ADDRESS0	配置MAC address 0过滤器
ENET_MAC_ADDRESS1	配置MAC address 1过滤器
ENET_MAC_ADDRESS2	配置MAC address 2过滤器
ENET_MAC_ADDRESS3	配置MAC address 3过滤器
输出参数{out}	

paddr	存储MAC地址的缓冲区指针，小端存储 例如MAC地址为aa:bb:cc:dd:ee:22，缓冲区内数据为{22, ee, dd, cc, bb, aa}
返回值	
-	-

例如：

```
/* get ENET MAC address */
```

```
enet_mac_address_get (ENET_MAC_ADDRESS0, netif->hwaddr);
```

enet_flag_get

函数enet_flag_get描述见下表：

表 3-188. 函数 enet_flag_get

函数名称	enet_flag_get
函数原型	FlagStatus enet_flag_get(enet_flag_enum enet_flag);
功能描述	获取ENET模块MAC/MSR/PTP/DMA状态标志位
先决条件	-
被调用函数	-
输入参数{in}	
enet_flag	ENET状态标志位，下列参数仅可选择一个
ENET_MAC_FLAG_MPKR	接收到魔术帧标志位
ENET_MAC_FLAG_WUFR	接收到唤醒帧标志位
ENET_MAC_FLAG_FLOWCONTROL	流控状态标志位
ENET_MAC_FLAG_WUM	WUM状态标志位
ENET_MAC_FLAG_MSR	MSR状态标志位
ENET_MAC_FLAG_MSCR	MSR接收状态标志位
ENET_MAC_FLAG_MSCT	MSR发送状态标志位
ENET_MAC_FLAG_TMST	时间戳触发状态标志位
ENET_PTP_FLAG_TSSCO	时间戳秒计数溢出标志位
ENET_PTP_FLAG_TTM	目标时间匹配标志位
ENET_MSR_FLAG_RFCE	接收帧CRC错误标志位

ENET_MSC_FLAG_ RFAE	接收帧对齐错误标志位
ENET_MSC_FLAG_ RGUF	接收到“好”的单播帧标志位
ENET_MSC_FLAG_ TGFSC	发送“好”的帧时仅遇到1个冲突标志位
ENET_MSC_FLAG_ TGFMSC	发送“好”的帧时遇到1个以上冲突
ENET_MSC_FLAG_ TGF	发送“好”的帧标志位
ENET_DMA_FLAG_ TS	发送状态标志位
ENET_DMA_FLAG_ TPS	发送流程停止状态标志位
ENET_DMA_FLAG_ TBU	发送缓冲区不可用状态标志位
ENET_DMA_FLAG_ TJT	发送jabber超时状态标志位
ENET_DMA_FLAG_ RO	接收溢出状态标志位
ENET_DMA_FLAG_ TU	发送下溢状态标志位
ENET_DMA_FLAG_ RS	接收状态标志位
ENET_DMA_FLAG_ RBU	接收缓冲区不可用状态标志位
ENET_DMA_FLAG_ RPS	接受流程停止状态标志位
ENET_DMA_FLAG_ RWT	接收看门狗超时状态标志位
ENET_DMA_FLAG_ ET	早发送状态标志位
ENET_DMA_FLAG_ FBE	致命总线错误状态标志位
ENET_DMA_FLAG_ ER	早接收状态标志位
ENET_DMA_FLAG_ AI	异常中断汇总标志位
ENET_DMA_FLAG_ NI	正常中断汇总标志位
ENET_DMA_FLAG_ EB_DMA_ERROR	DMA 错误标志位
ENET_DMA_FLAG_ 	发送错误标志位

<code>EB_TRANSFER_ERROR</code>	
<code>ENET_DMA_FLAG_EB_ACCESS_ERROR</code>	DMA访问错误标志位
<code>ENET_DMA_FLAG_MSC</code>	MSC状态标志位
<code>ENET_DMA_FLAG_WUM</code>	WUM状态标志位
<code>ENET_DMA_FLAG_TST</code>	时间戳触发状态标志位
输出参数{out}	
-	-
返回值	
FlagStatus	SET or RESET

例如：

```
/* get ENET flag */
```

```
enet_flag_get (ENET_DMA_FLAG_RS);
```

enet_flag_clear

函数enet_flag_clear描述见下表：

表 3-189. 函数 enet_flag_clear

函数名称	enet_flag_clear
函数原型	void enet_flag_clear(enet_flag_clear_enum enet_flag);
功能描述	清除ENET状态标志位
先决条件	-
被调用函数	-
输入参数{in}	
enet_flag	ENET模块DMA标志位清除，下列参数仅可选择一个
<code>ENET_DMA_FLAG_TS_CLR</code>	发送状态标志位清除
<code>ENET_DMA_FLAG_TPS_CLR</code>	发送流程停止状态标志位清除
<code>ENET_DMA_FLAG_TBU_CLR</code>	发送缓冲区不可用状态标志位清除
<code>ENET_DMA_FLAG_TJT_CLR</code>	发送jabber超时状态标志位清除
<code>ENET_DMA_FLAG_RO_CLR</code>	接收溢出状态标志位清除
<code>ENET_DMA_FLAG_</code>	发送下溢状态标志位清除

<i>TU_CLR</i>	
<i>ENET_DMA_FLAG_RS_CLR</i>	接收状态标志位清除
<i>ENET_DMA_FLAG_RBU_CLR</i>	接收缓冲区不可用状态标志位清除
<i>ENET_DMA_FLAG_RPS_CLR</i>	接受流程停止状态标志位清除
<i>ENET_DMA_FLAG_RWT_CLR</i>	接收看门狗超时状态标志位清除
<i>ENET_DMA_FLAG_ET_CLR</i>	早发送状态标志位清除
<i>ENET_DMA_FLAG_FBE_CLR</i>	致命总线错误状态标志位清除
<i>ENET_DMA_FLAG_ER_CLR</i>	早接收状态标志位清除
<i>ENET_DMA_FLAG_AI_CLR</i>	异常中断汇总标志位清除
<i>ENET_DMA_FLAG_NI_CLR</i>	正常中断汇总标志位清除
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* clear ENET flag */
```

```
enet_flag_clear(ENET_DMA_FLAG_RS_CLR);
```

enet_interrupt_enable

函数enet_interrupt_enable描述见下表：

表 3-190. 函数 enet_interrupt_enable

函数名称	enet_interrupt_enable
函数原型	void enet_interrupt_enable(enet_int_enum enet_int);
功能描述	使能ENET模块MAC/MSD/DMA中断
先决条件	-
被调用函数	-
输入参数{in}	
enet_int	ENET中断，下列参数仅可选择一个
<i>ENET_MAC_INT_WUMIM</i>	WUM中断屏蔽
<i>ENET_MAC_INT_T</i>	时间戳触发中断屏蔽

MSTIM	
ENET_MSC_INT_RFCEIM	接收帧CRC错误中断屏蔽
ENET_MSC_INT_RFAEIM	接收帧对齐错误中断屏蔽
ENET_MSC_INT_RGUFIM	接收“好”单播帧中断屏蔽
ENET_MSC_INT_TGFSCIM	仅遇到1个冲突后发送“好”帧中断屏蔽
ENET_MSC_INT_TGFMSCIM	遇到1个以上冲突后发送“好”帧中断屏蔽
ENET_MSC_INT_TGFIM	发送“好”的帧的中断屏蔽
ENET_DMA_INT_TIE	发送中断使能
ENET_DMA_INT_TPSIE	发送流程停止中断使能
ENET_DMA_INT_TBUIE	发送缓冲区不可用中断使能
ENET_DMA_INT_TJTIE	发送jabber超时中断使能
ENET_DMA_INT_ROIE	接收溢出中断使能
ENET_DMA_INT_TUIE	发送下溢中断使能
ENET_DMA_INT_RIE	接收中断使能
ENET_DMA_INT_RBUIE	接收缓冲区不可用中断使能
ENET_DMA_INT_RPSIE	接收流程停止中断使能
ENET_DMA_INT_RWTIE	接收看门狗超时中断使能
ENET_DMA_INT_ETIE	早发送中断使能
ENET_DMA_INT_FBEIE	致命总线错误中断使能
ENET_DMA_INT_ERIE	早接收中断使能
ENET_DMA_INT_AIE	异常中断汇总使能
ENET_DMA_INT_NIE	正常中断汇总使能

输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* enable ENET interrupt */
```

```
enet_interrupt_enable(ENET_DMA_INT_NIE);
```

enet_interrupt_disable

函数enet_interrupt_disable描述见下表：

表 3-191. 函数 enet_interrupt_disable

函数名称	enet_interrupt_disable
函数原型	void enet_interrupt_disable(enet_int_enum enet_int);
功能描述	禁能ENET模块MAC/MSC/DMA中断
先决条件	-
被调用函数	-
输入参数{in}	
enet_int	ENET中断，下列参数仅可选择一个
ENET_MAC_INT_WUMIM	WUM中断屏蔽
ENET_MAC_INT_TSTMIM	时间戳触发中断屏蔽
ENET_MSC_INT_RFCEIM	接收帧CRC错误中断屏蔽
ENET_MSC_INT_RFAEIM	接收帧对齐错误中断屏蔽
ENET_MSC_INT_RGUFIM	接收“好”单播帧中断屏蔽
ENET_MSC_INT_TGFSCIM	仅遇到1个冲突后发送“好”帧中断屏蔽
ENET_MSC_INT_TGFMSCIM	遇到1个以上冲突后发送“好”帧中断屏蔽
ENET_MSC_INT_TGFIM	发送“好”的帧的中断屏蔽
ENET_DMA_INT_TIE	发送中断使能
ENET_DMA_INT_TPSIE	发送流程停止中断使能
ENET_DMA_INT_TBUIE	发送缓冲区不可用中断使能

ENET_DMA_INT_TJ TIE	发送jabber超时中断使能
ENET_DMA_INT_R OIE	接收溢出中断使能
ENET_DMA_INT_TU IE	发送下溢中断使能
ENET_DMA_INT_RI E	接收中断使能
ENET_DMA_INT_R BUIE	接收缓冲区不可用中断使能
ENET_DMA_INT_R PSIE	接收流程停止中断使能
ENET_DMA_INT_R WTIE	接收看门狗超时中断使能
ENET_DMA_INT_ET IE	早发送中断使能
ENET_DMA_INT_FB EIE	致命总线错误中断使能
ENET_DMA_INT_E RIE	早接收中断使能
ENET_DMA_INT_AI E	异常中断汇总使能
ENET_DMA_INT_NI E	正常中断汇总使能
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* disable ENET interrupt */
```

```
enet_interrupt_disable(ENET_DMA_INT_NIE);
```

enet_interrupt_flag_get

函数enet_interrupt_flag_get描述见下表：

表 3-192. 函数 enet_interrupt_flag_get

函数名称	enet_interrupt_flag_get
函数原型	FlagStatus enet_interrupt_flag_get(enet_int_flag_enum int_flag);
功能描述	获取ENET模块MAC/MSD/DMA中断标志位
先决条件	-
被调用函数	-

输入参数{in}	
int_flag	ENET中断标志位，下列参数仅可选择一个
ENET_MAC_INT_FL AG_WUM	WUM中断标志位
ENET_MAC_INT_FL AG_MSC	MSC中断标志位
ENET_MAC_INT_FL AG_MSCR	MSC接收中断标志位
ENET_MAC_INT_FL AG_MSCT	MSC发送中断标志位
ENET_MAC_INT_FL AG_TMST	时间戳触发中断标志位
ENET_MSC_INT_FL AG_RFCE	接收帧CRC错误中断标志位
ENET_MSC_INT_FL AG_RFAE	接收帧对齐错误中断标志位
ENET_MSC_INT_FL AG_RGUF	接收“好”单播帧中断标志位
ENET_MSC_INT_FL AG_TGFSC	仅遇到1个冲突后发送“好”帧中断标志位
ENET_MSC_INT_FL AG_TGFMSC	遇到1个以上冲突后发送“好”帧中断标志位
ENET_MSC_INT_FL AG_TGF	发送“好”的帧的中断标志位
ENET_DMA_INT_FL AG_TS	发送中断标志位
ENET_DMA_INT_FL AG_TPS	发送流程停止中断标志位
ENET_DMA_INT_FL AG_TBU	发送缓冲区不可用中断标志位
ENET_DMA_INT_FL AG_TJT	发送jabber超时中断标志位
ENET_DMA_INT_FL AG_RO	接收溢出中断标志位
ENET_DMA_INT_FL AG_TU	发送下溢中断标志位
ENET_DMA_INT_FL AG_RS	接收中断标志位
ENET_DMA_INT_FL AG_RBU	接收缓冲区不可用中断标志位
ENET_DMA_INT_FL AG_RPS	接收流程停止中断标志位
ENET_DMA_INT_FL	接收看门狗超时中断标志位

AG_RWT	
ENET_DMA_INT_FL AG_ET	早发送中断标志位
ENET_DMA_INT_FL AG_FBE	致命总线错误中断标志位
ENET_DMA_INT_FL AG_ER	早接收中断标志位
ENET_DMA_INT_FL AG_AI	异常中断汇总中断标志位
ENET_DMA_INT_FL AG_NI	正常中断汇总中断标志位
ENET_DMA_INT_FL AG_MSC	MSC中断标志位
ENET_DMA_INT_FL AG_WUM	WUM中断标志位
ENET_DMA_INT_FL AG_TST	时间戳触发中断标志位
输出参数{out}	
-	-
返回值	
FlagStatus	SET or RESET

例如：

```
/* get ENET interrupt flag */
```

```
enet_interrupt_flag_get(ENET_DMA_INT_FLAG_RS);
```

enet_interrupt_flag_clear

函数enet_interrupt_flag_clear描述见下表：

表 3-193. 函数 enet_interrupt_flag_clear

函数名称	enet_interrupt_flag_clear
函数原型	void enet_interrupt_flag_clear(enet_int_flag_clear_enum int_flag_clear);
功能描述	禁能ENET中断标志位
先决条件	-
被调用函数	-
输入参数{in}	
int_flag_clear	ENET中断标志位清除，下列参数仅可选择一个
ENET_DMA_INT_FL AG_TS_CLR	发送中断标志位清除
ENET_DMA_INT_FL AG_TPS_CLR	发送流程停止中断标志位清除
ENET_DMA_INT_FL	发送缓冲区不可用中断标志位清除

AG_TBU_CLR	
ENET_DMA_INT_FL AG_TJT_CLR	发送jabber超时中断标志位清除
ENET_DMA_INT_FL AG_RO_CLR	接收溢出中断标志位清除
ENET_DMA_INT_FL AG_TU_CLR	发送下溢中断标志位清除
ENET_DMA_INT_FL AG_RS_CLR	接收中断标志位清除
ENET_DMA_INT_FL AG_RBU_CLR	接收缓冲区不可用中断标志位清除
ENET_DMA_INT_FL AG_RPS_CLR	接收流程停止中断标志位清除
ENET_DMA_INT_FL AG_RWT_CLR	接收看门狗超时中断标志位清除
ENET_DMA_INT_FL AG_ET_CLR	早发送中断标志位清除
ENET_DMA_INT_FL AG_FBE_CLR	致命总线错误中断标志位清除
ENET_DMA_INT_FL AG_ER_CLR	早接收中断标志位清除
ENET_DMA_INT_FL AG_AI_CLR	异常中断汇总中断标志位清除
ENET_DMA_INT_FL AG_NI_CLR	正常中断汇总中断标志位清除
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* clear ENET interrupt flag */
```

```
enet_interrupt_flag_clear(ENET_DMA_INT_FLAG_RS);
```

enet_tx_enable

函数enet_tx_enable描述见下表：

表 3-194. 函数 enet_tx_enable

函数名称	enet_tx_enable
函数原型	void enet_tx_enable(void);
功能描述	ENET发送功能使能（包括ENET外设内的MAC和DMA模块）
先决条件	-

被调用函数	enet_txfifo_flush()
输入参数{in}	
-	-
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* enable ENET tx function */
```

```
enet_tx_enable();
```

enet_tx_disable

函数enet_tx_disable描述见下表：

表 3-195. 函数 enet_tx_disable

函数名称	enet_tx_disable
函数原型	void enet_tx_disable(void);
功能描述	ENET发送功能禁能（包括ENET外设内的MAC和DMA模块）
先决条件	-
被调用函数	enet_txfifo_flush()
输入参数{in}	
-	-
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* disable ENET tx function */
```

```
enet_tx_disable();
```

enet_rx_enable

函数enet_rx_enable描述见下表：

表 3-196. 函数 enet_rx_enable

函数名称	enet_rx_enable
函数原型	void enet_rx_enable(void);
功能描述	ENET接收功能使能（包括ENET外设内的MAC和DMA模块）
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 ENET rx function */
```

```
enet_rx_enable();
```

enet_rx_disable

函数enet_rx_disable描述见下表：

表 3-197. 函数 enet_rx_disable

函数名称	enet_rx_disable
函数原型	void enet_rx_disable(void);
功能描述	ENET发送功能禁能（包括ENET外设内的MAC和DMA模块）
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 ENET rx function */
```

```
enet_rx_disable();
```

enet_registers_get

函数enet_registers_get描述见下表：

表 3-198. 函数 enet_registers_get

函数名称	enet_registers_get
函数原型	void enet_registers_get(enet_registers_type_enum type, uint32_t *preg, uint32_t num);
功能描述	获取指定范围ENET寄存器值
先决条件	-
被调用函数	-
输入参数{in}	
type	寄存器类型，下列参数仅可选择一个

<i>ALL_MAC_REG</i>	寄存器范围从ENET_MAC_CFG到ENET_MAC_FCTH
<i>ALL_MSC_REG</i>	寄存器范围从ENET_MSC_CTL到ENET_MSC_RGUFCNT
<i>ALL_PTP_REG</i>	寄存器范围从ENET_PTP_TSCTL到ENET_PTP_PPSCTL
<i>ALL_DMA_REG</i>	寄存器范围从ENET_DMA_BCTL到ENET_DMA_CRBA DDR
输入参数{in}	
num	想要获取的寄存器个数，范围(0~54)
输出参数{out}	
preg	存储寄存器值的应用缓冲区指针
返回值	
-	-

例如：

```
/* get ENET registers value */
uint32_t register_buffer[5];

enet_registers_get(ALL_MAC_REG, 5, register_buffer);
```

enet_address_filter_enable

函数enet_address_filter_enable描述见下表：

表 3-199. 函数 enet_address_filter_enable

函数名称	enet_address_filter_enable
函数原型	void enet_address_filter_enable(enet_macaddress_enum mac_addr);
功能描述	MAC地址过滤器使能
先决条件	-
被调用函数	--
输入参数{in}	
mac_addr	选择被使能的MAC地址组，下列参数仅可选择一个
<i>ENET_MAC_ADDRE</i> <i>SS1</i>	使能MAC地址组1过滤器
<i>ENET_MAC_ADDRE</i> <i>SS2</i>	使能MAC地址组2过滤器
<i>ENET_MAC_ADDRE</i> <i>SS3</i>	使能MAC地址组3过滤器
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* enable ENET MAC address filter */

enet_address_filter_enable(ENET_MAC_ADDRESS1);
```

enet_address_filter_disable

函数enet_address_filter_disable描述见下表:

表 3-200. 函数 enet_address_filter_disable

函数名称	enet_address_filter_disable
函数原型	void enet_address_filter_disable(enet_macaddress_enum mac_addr);
功能描述	MAC地址过滤器禁能
先决条件	-
被调用函数	-
输入参数{in}	
mac_addr	选择被使能的MAC地址组，下列参数仅可选择一个
ENET_MAC_ADDRES S1	使能MAC地址组1过滤器
ENET_MAC_ADDRES S2	使能MAC地址组2过滤器
ENET_MAC_ADDRES S3	使能MAC地址组3过滤器
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* disable ENET MAC address filter */
enet_address_filter_disable(ENET_MAC_ADDRESS1);
```

enet_address_filter_config

函数enet_address_filter_config描述见下表:

表 3-201. 函数 enet_address_filter_config

函数名称	enet_address_filter_config
函数原型	void enet_address_filter_config(enet_macaddress_enum mac_addr, uint32_t addr_mask, uint32_t filter_type);
功能描述	配置MAC地址过滤器模式
先决条件	-
被调用函数	-
输入参数{in}	
mac_addr	选择被使能的MAC地址组，下列参数仅可选择一个
ENET_MAC_ADDRES S1	使能MAC地址组1过滤器
ENET_MAC_ADDRES S2	使能MAC地址组2过滤器

ENET_MAC_ADDRESSES3	使能MAC地址组3过滤器
输入参数{in}	
addr_mask	选择MAC地址哪些字节将被屏蔽，下列参数可以选择多个
ENET_ADDRESS_MASK_BYTE0	屏蔽ENET_MAC_ADDR1L[7:0] bits
ENET_ADDRESS_MASK_BYTE1	屏蔽ENET_MAC_ADDR1L[15:8] bits
ENET_ADDRESS_MASK_BYTE2	屏蔽ENET_MAC_ADDR1L[23:16] bits
ENET_ADDRESS_MASK_BYTE3	屏蔽ENET_MAC_ADDR1L [31:24] bits
ENET_ADDRESS_MASK_BYTE4	屏蔽ENET_MAC_ADDR1H [7:0] bits
ENET_ADDRESS_MASK_BYTE5	屏蔽ENET_MAC_ADDR1H [15:8] bits
输入参数{in}	
filter_type	选择MAC地址过滤器类型， 下列参数仅可选择一个
ENET_ADDRESS_FILTER_SA	过滤器比对接收帧MAC地址的源地址域
ENET_ADDRESS_FILTER_DA	过滤器比对接收帧MAC地址的目的地址域
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* configure ENET MAC address filter */
```

```
enet_address_filter_config(ENET_MAC_ADDRESS1, ENET_ADDRESS_MASK_BYTE0 |
ENET_ADDRESS_MASK_BYTE1 | ENET_ADDRESS_MASK_BYTE2, ENET_ADDRESS_FILTER_DA);
```

enet_phy_config

函数enet_phy_config描述见下表：

表 3-202. 函数 enet_phy_config

函数名称	enet_phy_config
函数原型	ErrStatus enet_phy_config(void);
功能描述	PHY接口配置（配置SMI时钟并复位PHY芯片）
先决条件	-

被调用函数	rcu_clock_freq_get()/enet_phy_write_read()
输入参数{in}	
-	-
输出参数{out}	
-	-
返回值	
ErrStatus	ERROR or SUCCESS

例如:

```
/* initialize PHY */
enet_phy_config();
```

enet_phy_write_read

函数enet_phy_write_read描述见下表:

表 3-203. 函数 enet_phy_write_read

函数名称	enet_phy_write_read
函数原型	ErrStatus enet_phy_write_read(enet_phydirection_enum direction, uint16_t phy_address, uint16_t phy_reg, uint16_t *pvalue);
功能描述	写/读PHY寄存器
先决条件	-
被调用函数	-
输入参数{in}	
direction	下列参数仅可选择一个
ENET_PHY_WRITE	向PHY寄存器写数据
ENET_PHY_READ	从PHY寄存器读数据
输入参数{in}	
phy_address	0x0 - 0x1F
输入参数{in}	
phy_reg	0x0 - 0x1F
输入参数{in}	
pvalue	当direction选择ENET_PHY_WRITE时, 表示将写入PHY寄存器的值
输出参数{out}	
pvalue-	当direction选择ENET_PHY_READ时, 表示将存储PHY寄存器读出的值
返回值	
ErrStatus	ERROR or SUCCESS

例如:

```
/* read the specified PHY register value */
uint16_t temp_phy = 0U;
phy_state = enet_phy_write_read(ENET_PHY_READ, PHY_ADDRESS, PHY_REG_BCR,
```

```
&temp_phy);
```

enet_phyloopback_enable

函数enet_phyloopback_enable描述见下表:

表 3-204. 函数 enet_phyloopback_enable

函数名称	enet_phyloopback_enable
函数原型	ErrStatus enet_phyloopback_enable(void);
功能描述	使能PHY芯片回环模式
先决条件	-
被调用函数	enet_phy_write_read()
输入参数{in}	
-	-
输出参数{out}	
-	-
返回值	
ErrStatus	ERROR or SUCCESS

例如:

```
/* enable PHY loopback function */
```

```
phy_state = enet_phyloopback_enable();
```

enet_phyloopback_disable

函数enet_phyloopback_disable描述见下表:

表 3-205. 函数 enet_phyloopback_disable

函数名称	enet_phyloopback_disable
函数原型	ErrStatus enet_phyloopback_disable(void);
功能描述	禁能PHY芯片回环模式
先决条件	-
被调用函数	enet_phy_write_read
输入参数{in}	
-	-
输出参数{out}	
-	-
返回值	
ErrStatus	ERROR or SUCCESS

例如:

```
/* disable PHY loopback function */
```

```
phy_state = enet_phyloopback_disable();
```

enet_forward_feature_enable

函数enet_forward_feature_enable描述见下表：

表 3-206. 函数 enet_forward_feature_enable

函数名称	enet_forward_feature_enable
函数原型	void enet_forward_feature_enable(uint32_t feature);
功能描述	使能ENET帧通过相关功能
先决条件	-
被调用函数	-
输入参数{in}	
feature	ENET帧通过功能，下列参数可以选择多个
ENET_AUTO_PADCRC_DROP	自动去除接收帧填充字节和FCS域
ENET_FORWARD_ERRFRAMES	除了过短帧外的其他错误帧都会转发给应用
ENET_FORWARD_UNDEFSZ_GOODFRAMES	帧长小于64字节但没有错误的帧将转发给应用
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* enable ENET forward feature */
```

```
enet_forward_feature_enable(ENET_AUTO_PADCRC_DROP);
```

enet_forward_feature_disable

函数enet_forward_feature_disable描述见下表：

表 3-207. 函数 enet_forward_feature_disable

函数名称	enet_forward_feature_disable
函数原型	void enet_forward_feature_disable(uint32_t feature);
功能描述	禁能ENET帧通过相关功能
先决条件	-
被调用函数	-
输入参数{in}	
feature	ENET帧通过功能，下列参数可以选择多个
ENET_AUTO_PADCRC_DROP	自动去除接收帧填充字节和FCS域
ENET_FORWARD_ERRFRAMES	除了过短帧外的其他错误帧都会转发给应用

ENET_FORWARD_ UNDERSZ_GOODF RAMES	帧长小于64字节但没有错误的帧将转发给应用
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* disable ENET forward feature */
```

```
enet_forward_feature_disable (ENET_AUTO_PADCRC_DROP);
```

enet_filter_feature_enable

函数enet_filter_feature_enable描述见下表：

表 3-208. 函数 enet_filter_feature_enable

函数名称	enet_filter_feature_enable
函数原型	void enet_filter_feature_enable(uint32_t feature)
功能描述	使能ENET帧过滤器相关功能
先决条件	-
被调用函数	-
输入参数{in}	
feature	ENET过滤器功能，下列参数可以选择多个
ENET_SRC_FILTER	源地址过滤器
ENET_SRC_FILTER _INVERSE	源地址过滤结果逆转
ENET_DEST_FILTE R_INVERSE	目的地址过滤结果逆转
ENET_MULTICAST_ FILTER_PASS	接收多播帧
ENET_MULTICAST_ FILTER_HASH_MO DE	HASH多播过滤器
ENET_UNICAST_FI LTER_HASH_MODE	HASH单播过滤器
ENET_FILTER_MO DE_EITHER	HASH或完美过滤器功能
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* enable ENET filter feature */
```

```
enet_filter_feature_enable(ENET_SRC_FILTER);
```

enet_filter_feature_disable

函数enet_filter_feature_disable描述见下表:

表 3-209. 函数 enet_filter_feature_disable

函数名称	enet_filter_feature_disable
函数原型	void enet_filter_feature_disable(uint32_t feature);
功能描述	禁能ENET帧过滤器相关功能
先决条件	-
被调用函数	-
输入参数{in}	
feature	ENET过滤器功能，下列参数可以选择多个
ENET_SRC_FILTER	源地址过滤器
ENET_SRC_FILTER_INVERSE	源地址过滤结果逆转
ENET_DEST_FILTER_INVERSE	目的地址过滤结果逆转
ENET_MULTICAST_FILTER_PASS	接收多播帧
ENET_MULTICAST_FILTER_HASH_MODE	HASH多播过滤器
ENET_UNICAST_FILTER_HASH_MODE	HASH单播过滤器
ENET_FILTER_MODE_EITHER	HASH或完美过滤器功能
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* disable ENET filter feature */
```

```
enet_filter_feature_disable(ENET_SRC_FILTER_INVERSE);
```

enet_pauseframe_generate

函数enet_pauseframe_generate描述见下表:

表 3-210. 函数 enet_pauseframe_generate

函数名称	enet_pauseframe_generate
------	--------------------------

函数原型	ErrStatus enet_pauseframe_generate(void);
功能描述	生成暂停帧，使能发送流控功能后ENET模块将发送暂停帧
先决条件	-
被调用函数	-
输入参数{in}	
-	-
输出参数{out}	
-	-
返回值	
ErrStatus	ERROR or SUCCESS

例如：

```
/* generate ENET pause frame */
ErrStatus reval;

reval = enet_pauseframe_generate();
```

enet_pauseframe_detect_config

函数enet_pauseframe_detect_config描述见下表：

表 3-211. 函数 enet_pauseframe_detect_config

函数名称	enet_pauseframe_detect_config
函数原型	void enet_pauseframe_detect_config(uint32_t detect);
功能描述	配置暂停帧检测类型
先决条件	-
被调用函数	-
输入参数{in}	
detect	暂停帧检测，下列参数仅可选择一个
ENET_MAC0_AND_UNIQUE_ADDRESS_PAUSEDTECT	除了唯一多播地址的暂停帧，MAC同时还会使用MAC0地址（ENET_MAC_ADDR0H寄存器和ENET_MAC_ADDR0L寄存器）来检测暂停帧
ENET_UNIQUE_PAUSEDTECT	MAC只接收符合IEEE802.3规范定义的唯一多播地址的暂停帧
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* configure ENET pause frame detect type */

enet_pauseframe_detect_config(ENET_UNIQUE_PAUSEDTECT);
```

enet_pauseframe_config

函数enet_pauseframe_config描述见下表:

表 3-212. 函数 enet_pauseframe_config

函数名称	enet_pauseframe_config
函数原型	void enet_pauseframe_config(uint32_t pausetime, uint32_t pause_threshold);
功能描述	配置暂停帧参数
先决条件	-
被调用函数	-
输入参数{in}	
pausetime	暂停控制帧时间域, 范围(0~0xFFFF)
输入参数{in}	
pause_threshold	设置了自动重发暂停帧的定时器阈值。这个阈值应当大于0, 小于位[31:16]定义的暂停时间。低阈值的计算公式为PTM-PLTS。例如, PTM = 0x80 (128个时间间隙), PLTS = 0x1 (28个时间间隙), 那么在第一个暂停帧发出100(128-28)个时间间隙后, 将自动重发第二个暂停帧, 下列参数仅可选择一个
ENET_PAUSETIME_MINUS4	暂停时间 - 4个时间间隙
ENET_PAUSETIME_MINUS28	暂停时间 - 28个时间间隙
ENET_PAUSETIME_MINUS144	暂停时间 - 144个时间间隙
ENET_PAUSETIME_MINUS256	暂停时间 - 256个时间间隙
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* configure ENET pause frame */
```

```
enet_pauseframe_config(30, ENET_PAUSETIME_MINUS4);
```

enet_flowcontrol_threshold_config

函数enet_flowcontrol_threshold_config描述见下表:

表 3-213. 函数 enet_flowcontrol_threshold_config

函数名称	enet_flow control_threshold_config
函数原型	void enet_flow control_threshold_config(uint32_t deactive, uint32_t active);
功能描述	配置流控阈值
先决条件	-
被调用函数	-

输入参数{in}	
deactive	流控失效的阈值。这个值应当小于位[2:0]定义的流控激活阈值。当RxFIFO中未处理的数据低于这些位所设置的值，流控功能将自动失效，下列参数仅可选择一个
ENET_DEACTIVE_T HRESHOLD_256BY TES	256字节
ENET_DEACTIVE_T HRESHOLD_512BY TES	512字节
ENET_DEACTIVE_T HRESHOLD_768BY TES	768字节
ENET_DEACTIVE_T HRESHOLD_1024B YTES	1024字节
ENET_DEACTIVE_T HRESHOLD_1280B YTES	1280字节
ENET_DEACTIVE_T HRESHOLD_1536B YTES	1536字节
ENET_DEACTIVE_T HRESHOLD_1792B YTES	1792字节
输入参数{in}	
active	流控激活的阈值。若使能了流控功能，当RxFIFO中未处理的数据超过了这些位所设置的值，流控功能将被激活，下列参数仅可选择一个
ENET_ACTIVE_THR ESHOLD_256BYTE S	256字节
ENET_ACTIVE_THR ESHOLD_512BYTE S	512字节
ENET_ACTIVE_THR ESHOLD_768BYTE S	768字节
ENET_ACTIVE_THR ESHOLD_1024BYTE S	1024字节
ENET_ACTIVE_THR ESHOLD_1280BYTE S	1280字节

ENET_ACTIVE_THRESHOLD_1536BYTES	1536字节
ENET_ACTIVE_THRESHOLD_1792BYTES	1792字节
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* configure ENET flowcontrol threshold value */
```

```
enet_flowcontrol_threshold_config(ENET_DEACTIVE_THRESHOLD_256BYTES, ENET_ACTIVE_THRESHOLD_256BYTES);
```

enet_flowcontrol_feature_enable

函数enet_flowcontrol_feature_enable描述见下表：

表 3-214. 函数 enet_flowcontrol_feature_enable

函数名称	enet_flow control_feature_enable
函数原型	void enet_flow control_feature_enable(uint32_t feature);
功能描述	使能ENET流控相关功能
先决条件	-
被调用函数	-
输入参数{in}	
feature	ENET流控功能模式，下列参数可以选择多个
ENET_ZERO_QUANTA_PAUSE	零时间片暂停控制帧自动生成
ENET_TX_FLOWCONTROL	发送流控功能
ENET_RX_FLOWCONTROL	接收流控功能
ENET_BACKPRESSURE	背压功能（仅在半双工模式下）
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* enable ENET flowcontrol feature */
```

```
enet_flowcontrol_feature_enable(ENET_ZERO_QUANTA_PAUSE);
```

enet_flowcontrol_feature_disable

函数enet_flowcontrol_feature_disable描述见下表：

表 3-215. 函数 enet_flowcontrol_feature_disable

函数名称	enet_flow control_feature_disable
函数原型	void enet_flow control_feature_disable(uint32_t feature);
功能描述	禁能ENET流控相关功能
先决条件	-
被调用函数	-
输入参数{in}	
feature	ENET流控功能模式，下列参数可以选择多个
ENET_ZERO_QUANTA_PAUSE	零时间片暂停控制帧自动生成
ENET_TX_FLOWCONTROL	发送流控功能
ENET_RX_FLOWCONTROL	接收流控功能
ENET_BACKPRESSURE	背压功能（仅在半双工模式下）
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* disable ENET flowcontrol feature */
```

```
enet_flowcontrol_feature_disable(ENET_ZERO_QUANTA_PAUSE);
```

enet_dmaprocess_state_get

函数enet_dmaprocess_state_get描述见下表：

表 3-216. 函数 enet_dmaprocess_state_get

函数名称	enet_dmaprocess_state_get
函数原型	uint32_t enet_dmaprocess_state_get(enet_dmadirection_enum direction);
功能描述	获取DMA发送/接收流程状态
先决条件	-
被调用函数	-
输入参数{in}	
direction	DMA传输方向
ENET_DMA_TX	DMA发送进程

ENET_DMA_RX	DMA接收进程
输出参数{out}	
-	-
返回值	
uint32_t	DMA流程状态，可取值如下： ENET_RX_STATE_STOPPED / ENET_RX_STATE_FETCHING / ENET_RX_STATE_WAITING / ENET_RX_STATE_SUSPENDED / ENET_RX_STATE_CLOSING / ENET_RX_STATE_QUEUING / ENET_TX_STATE_STOPPED / ENET_TX_STATE_FETCHING / ENET_TX_STATE_WAITING / ENET_TX_STATE_READING / ENET_TX_STATE_SUSPENDED / ENET_TX_STATE_CLOSING

例如：

```

/* get ENET dma receive process state */

uint32_t reval;

reval = enet_dmaprocess_state_get(ENET_DMA_RX);

if(ENET_RX_STATE_SUSPENDED == reval){

    do...

}

```

enet_dmaprocess_resume

函数enet_dmaprocess_resume描述见下表：

表 3-217. 函数 enet_dmaprocess_resume

函数名称	enet_dmaprocess_resume
函数原型	void enet_dmaprocess_resume(enet_dmadirection_enum direction);
功能描述	DMA发送/接收查询使能
先决条件	-
被调用函数	-
输入参数{in}	
direction	描述符类型，下列参数仅可选择一个
ENET_DMA_TX	DMA发送进程
ENET_DMA_RX	DMA接收进程
输出参数{out}	
-	-
返回值	
-	-

例如：

```

/* resume ENET dma receive process */

```



```
enet_dmaprocess_resume(ENET_DMA_RX);
```

enet_rxprocess_check_recovery

函数enet_rxprocess_check_recovery描述见下表:

表 3-218. 函数 enet_rxprocess_check_recovery

函数名称	enet_rxprocess_check_recovery
函数原型	void enet_rxprocess_check_recovery(void);
功能描述	检测并恢复接收流程
先决条件	-
被调用函数	-
输入参数{in}	
-	-
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* check and recovery ENET dma receive process */
```

```
enet_rxprocess_check_recovery();
```

enet_txfifo_flush

函数enet_txfifo_flush描述见下表:

表 3-219. 函数 enet_txfifo_flush

函数名称	enet_txfifo_flush
函数原型	ErrStatus enet_txfifo_flush(void);
功能描述	刷新ENET发送FIFO，并等待刷新操作完成
先决条件	-
被调用函数	-
输入参数{in}	
-	-
输出参数{out}	
-	-
返回值	
ErrStatus	ERROR or SUCCESS

例如:

```
/* flush ENET dma transmission FIFO */
```

```
ErrStatus reval;
```

```
reval = enet_txfifo_flush();
```

enet_current_desc_address_get

函数enet_current_desc_address_get描述见下表:

表 3-220. 函数 enet_current_desc_address_get

函数名称	enet_current_desc_address_get
函数原型	uint32_t enet_current_desc_address_get(enet_desc_reg_enum addr_get);
功能描述	获取当前发送/接收描述符地址、当前缓冲区地址、描述符列表首地址
先决条件	-
被调用函数	-
输入参数{in}	
addr_get	可获取的描述符地址类型，下列参数仅可选择一个
ENET_RX_DESC_TABLE	接收描述符列表首地址
ENET_RX_CURRENT_DESC	当前DMA控制器使用的接收描述符地址
ENET_RX_CURRENT_BUFFER	当前DMA控制器使用的接收描述符缓冲区地址
ENET_TX_DESC_TABLE	发送描述符列表首地址
ENET_TX_CURRENT_DESC	当前DMA控制器使用的发送描述符地址
ENET_TX_CURRENT_BUFFER	当前DMA控制器使用的发送描述符缓冲区地址
输出参数{out}	
-	-
返回值	
uint32_t	0- 0xFFFFFFFF

例如:

```
/* get current receive descriptor address */
```

```
uint32_t reval;
```

```
reval = enet_current_desc_address_get(ENET_RX_CURRENT_DESC);
```

enet_desc_information_get

函数enet_desc_information_get描述见下表:

表 3-221. 函数 enet_desc_information_get

函数名称	enet_desc_information_get
函数原型	uint32_t enet_desc_information_get(enet_descriptors_struct *desc, enet_descstate_enum info_get);

功能描述	获取发送/接收描述符详细信息
先决条件	-
被调用函数	-
输入参数{in}	
desc	描述符指针，结构体成员介绍请参考 表3-172. 结构体 <i>enet_descriptors_struct</i>
输入参数{in}	
info_get	可选择的描述符信息类型，下列参数仅可选择一个
RXDESC_BUFFER_1_SIZE	接收缓冲区1大小
RXDESC_BUFFER_2_SIZE	接收缓冲区2大小
RXDESC_FRAME_LENGTH	接收帧长度
TXDESC_COLLISION_COUNT	帧发送出去前出现的冲突次数
RXDESC_BUFFER_1_ADDR	接收帧的缓冲区地址
TXDESC_BUFFER_1_ADDR	发送帧的缓冲区地址
输出参数{out}	
-	-
返回值	
uint32_t	描述符信息，如果返回值为0xFFFFFFFFU，说明输入参数有误

例如：

```
/* get the information of specific receive descriptor */
```

```
uint32_t reval;
```

```
reval = enet_desc_information_get(rx_desc, RXDESC_FRAME_LENGTH);
```

enet_missed_frame_counter_get

函数enet_missed_frame_counter_get描述见下表：

表 3-222. 函数 enet_missed_frame_counter_get

函数名称	enet_missed_frame_counter_get
函数原型	void enet_missed_frame_counter_get(uint32_t *rxfifo_drop, uint32_t *rxdma_drop);
功能描述	获取接收丢弃帧数
先决条件	-
被调用函数	-
输入参数{in}	
-	-

输出参数{out}	
rxfifo_drop	存储由于过短帧或接收FIFO溢出而丢弃帧数的指针
输出参数{out}	
rxdma_drop	存储由于接收描述符不可用而丢弃帧数的指针
返回值	
-	-

例如：

```
/* get the number of missed frames during receiving */
```

```
uint32_t fifo_cnt, dma_cnt;
```

```
enet_missed_frame_counter_get(&fifo_cnt, &dma_cnt);
```

enet_desc_flag_get

函数enet_desc_flag_get描述见下表：

表 3-223. 函数 enet_desc_flag_get

函数名称	enet_desc_flag_get
函数原型	FlagStatus enet_desc_flag_get(enet_descriptors_struct *desc, uint32_t desc_flag);
功能描述	获取ENET模块DMA描述符标志位
先决条件	-
被调用函数	-
输入参数{in}	
desc	描述符指针，结构体成员介绍请参考 表3-172. 结构体 enet_descriptors_struct
输入参数{in}	
desc_flag (the value according to the parameter desc)	描述符标志位，下列参数仅可选择一个
当desc参数为发送描述符时	
ENET_TDES0_DB	顺延位
ENET_TDES0_UFE	数据下溢错误位
ENET_TDES0_EXD	过度顺延位
ENET_TDES0_VFRM	VLAN帧位
ENET_TDES0_ECO	过度冲突位
ENET_TDES0_LCO	延迟冲突位
ENET_TDES0_NCA	无载波位
ENET_TDES0_LCA	载波丢失位
ENET_TDES0_IPPE	IP数据错误位
ENET_TDES0_FRM	帧清空位

<i>F</i>	
<i>ENET_TDES0_JT</i>	Jabber超时位
<i>ENET_TDES0_ES</i>	错误汇总
<i>ENET_TDES0_IPHE</i>	IP报头错误位
<i>ENET_TDES0_TTM</i> <i>SS</i>	发送时间戳状态位
<i>ENET_TDES0_TCH</i> <i>M</i>	第二地址链表模式位
<i>ENET_TDES0_TER</i> <i>M</i>	环形发送结束模式位
<i>ENET_TDES0_TTS</i> <i>EN</i>	使能发送时间戳位
<i>ENET_TDES0_DPA</i> <i>D</i>	不填充位
<i>ENET_TDES0_DCR</i> <i>C</i>	不计算CRC位
<i>ENET_TDES0_FSG</i>	第一分块位
<i>ENET_TDES0_LSG</i>	最后分块位
<i>ENET_TDES0_INTC</i>	完成时中断位
<i>ENET_TDES0_DAV</i>	DAV位
当 <i>desc</i> 参数为接收描述符时	
<i>ENET_RDES0_PCE</i> <i>RR</i>	数据校验和错误
<i>ENET_RDES0_CER</i> <i>R</i>	CRC错误
<i>ENET_RDES0_DBE</i> <i>RR</i>	Dribble位错误
<i>ENET_RDES0_RER</i> <i>R</i>	接收错误
<i>ENET_RDES0_RWD</i> <i>T</i>	接收看门狗超时
<i>ENET_RDES0_FRM</i> <i>T</i>	帧类型
<i>ENET_RDES0_LCO</i>	延迟冲突位
<i>ENET_RDES0_IPHE</i> <i>RR</i>	IP帧报头校验和错误
<i>ENET_RDES0_LDE</i> <i>S</i>	最后一个描述符
<i>ENET_RDES0_FDE</i> <i>S</i>	第一个描述符
<i>ENET_RDES0_VTA</i> <i>G</i>	VLAN标签位
<i>ENET_RDES0_OER</i>	溢出错误位

<i>R</i>	
<i>ENET_RDES0_LER</i> <i>R</i>	长度错误位
<i>ENET_RDES0_SAF</i> <i>F</i>	未通过源地址过滤器位
<i>ENET_RDES0_DER</i> <i>R</i>	描述符错误位
<i>ENET_RDES0_ERR</i> <i>S</i>	错误汇总位
<i>ENET_RDES0_DAF</i> <i>F</i>	未通过目标地址过滤器位
<i>ENET_RDES0_DAV</i>	描述符可用位
输出参数{out}	
-	-
返回值	
FlagStatus	SET or RESET

例如：

```
/* get the bit flag of ENET DMA transimission descriptor */
```

```
FlagStatus reval;
```

```
reval = enet_desc_flag_get(p_txdesc, ENET_TDES0_TCHM);
```

enet_desc_flag_set

函数enet_desc_flag_set描述见下表：

表 3-224. 函数 enet_desc_flag_set

函数名称	enet_desc_flag_set
函数原型	void enet_desc_flag_set(enet_descriptors_struct *desc, uint32_t desc_flag);
功能描述	设置ENET模块DMA描述符标志位
先决条件	-
被调用函数	-
输入参数{in}	
desc	描述符指针，结构体成员介绍请参考 表3-172. 结构体 enet_descriptors_struct
输入参数{in}	
desc_flag (the value according to the parameter desc)	描述符标志位，下列参数仅可选择一个
当desc参数为发送描述符时	
<i>ENET_TDES0_VFR</i> <i>M</i>	VLAN帧位

ENET_TDES0_FRM F	帧清空位
ENET_TDES0_TCH M	第二地址链表模式位
ENET_TDES0_TER M	环形发送结束模式位
ENET_TDES0_TTS EN	使能发送时间戳位
ENET_TDES0_DPA D	不填充位
ENET_TDES0_DCR C	不计算CRC位
ENET_TDES0_FSG	第一分块位
ENET_TDES0_LSG	最后分块位
ENET_TDES0_INTC	完成时中断位
ENET_TDES0_DAV	DAV位
当desc参数为接收描述符时	
ENET_RDES0_DAV	描述符可用位
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* set the bit flag of ENET DMA transimission descriptor */
```

```
enet_desc_flag_set(p_txdesc, ENET_TDES0_TCHM);
```

enet_desc_flag_clear

函数enet_desc_flag_clear描述见下表：

表 3-225. 函数 enet_desc_flag_clear

函数名称	enet_desc_flag_clear
函数原型	void enet_desc_flag_clear(enet_descriptors_struct *desc, uint32_t desc_flag);
功能描述	清除ENET模块DMA描述符标志位
先决条件	-
被调用函数	-
输入参数{in}	
desc	描述符指针，结构体成员介绍请参考 表3-172. 结构体 enet_descriptors_struct
输入参数{in}	
desc_flag (the value according	描述符标志位，下列参数仅可选择一个

to the parameter desc)	
当 desc 参数为发送描述符时	
ENET_TDES0_VFR M	VLAN帧位
ENET_TDES0_FRM F	帧清空位
ENET_TDES0_TCH M	第二地址链表模式位
ENET_TDES0_TER M	环形发送结束模式位
ENET_TDES0_TTS EN	使能发送时间戳位
ENET_TDES0_DPA D	不填充位
ENET_TDES0_DCR C	不计算CRC位
ENET_TDES0_FSG	第一分块位
ENET_TDES0_LSG	最后分块位
ENET_TDES0_INTC	完成时中断位
ENET_TDES0_DAV	DAV位
当 desc 参数为接收描述符时	
ENET_RDES0_DAV	描述符可用位
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* clear the bit flag of ENET DMA transimission descriptor */
```

```
enet_desc_flag_clear(p_txdesc, ENET_TDES0_TCHM);
```

enet_desc_receive_complete_bit_enable

函数enet_desc_receive_complete_bit_enable描述见下表：

表 3-226. 函数 enet_desc_receive_complete_bit_enable

函数名称	enet_desc_receive_complete_bit_enable
函数原型	void enet_desc_receive_complete_bit_enable(enet_descriptors_struct *desc);
功能描述	当接收完成时，ENET_DMA_STAT寄存器的RS位将被置位
先决条件	-
被调用函数	-
输入参数{in}	

desc	描述符指针，结构体成员介绍请参考 表3-172. 结构体 <i>enet_descriptors_struct</i>
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* when receiving the completed, RS bit in ENET_DMA_STAT register will set */
```

```
enet_desc_receive_complete_bit_enable(p_rxdesc);
```

enet_desc_receive_complete_bit_disable

函数enet_desc_receive_complete_bit_disable描述见下表：

表 3-227. 函数 enet_desc_receive_complete_bit_disable

函数名称	enet_desc_receive_complete_bit_disable
函数原型	void enet_desc_receive_complete_bit_disable(enet_descriptors_struct *desc);
功能描述	当接收完成时，ENET_DMA_STAT寄存器的RS位将不会被置位
先决条件	-
被调用函数	-
输入参数{in}	
desc	描述符指针，结构体成员介绍请参考 表3-172. 结构体 <i>enet_descriptors_struct</i>
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* when receiving the completed, RS bit in ENET_DMA_STAT register will not set */
```

```
enet_desc_receive_complete_bit_disable(p_rxdesc);
```

enet_rxframe_drop

函数enet_rxframe_drop描述见下表：

表 3-228. 函数 enet_rxframe_drop

函数名称	enet_rxframe_drop
函数原型	void enet_rxframe_drop(void);
功能描述	丢弃当前接收到的帧
先决条件	-
被调用函数	-
输入参数{in}	

-	-
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* drop current receive frame */
```

```
enet_rxframe_drop(ADC0);
```

enet_dma_feature_enable

函数enet_dma_feature_enable描述见下表：

表 3-229. 函数 enet_dma_feature_enable

函数名称	enet_dma_feature_enable
函数原型	void enet_dma_feature_enable(uint32_t feature);
功能描述	使能ENET模块DMA相关功能
先决条件	-
被调用函数	-
输入参数{in}	
feature	DMA功能，下列参数可以选择多个
ENET_NO_FLUSH_RXFRAME	描述符不可用时，RxDMA控制器清空接收帧功能
ENET_SECONDFRAME_OPT	TxDMA控制器第二帧功能
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* enable ENET dma feature */
```

```
enet_dma_feature_enable(ENET_NO_FLUSH_RXFRAME);
```

enet_dma_feature_disable

函数enet_dma_feature_disable描述见下表：

表 3-230. 函数 enet_dma_feature_disable

函数名称	enet_dma_feature_disable
函数原型	void enet_dma_feature_disable(uint32_t feature);
功能描述	禁能ENET模块DMA相关功能
先决条件	-

被调用函数	-
输入参数{in}	
feature	DMA功能，下列参数可以选择多个
ENET_NO_FLUSH_RXFRAME	描述符不可用时，RxDMA控制器清空接收帧功能
ENET_SECONDFRAME_OPT	TxDMA控制器第二帧功能
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* disable ENET dma feature */
```

```
enet_dma_feature_disable(ENET_NO_FLUSH_RXFRAME);
```

enet_ptp_normal_descriptors_chain_init

函数enet_ptp_normal_descriptors_chain_init描述见下表：

表 3-231. 函数 enet_ptp_normal_descriptors_chain_init

函数名称	enet_ptp_normal_descriptors_chain_init
函数原型	void enet_ptp_normal_descriptors_chain_init(enet_dmadirection_enum direction, enet_descriptors_struct *desc_ptptab);
功能描述	初始化具有PTP功能的DMA接收/发送描述符为链模式
先决条件	-
被调用函数	-
输入参数{in}	
direction	描述符类型，下列参数仅可选择一个
ENET_DMA_TX	DMA Tx描述符
ENET_DMA_RX	DMA Rx描述符
输入参数{in}	
desc_ptptab	PTP接收描述符列表首地址指针，结构体成员介绍请参考 表3-172. 结构体 enet_descriptors_struct
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* initialize received descriptors in chain mode with PTP function */
```

```
enet_descriptors_struct ptp_rxdesc_tab[ENET_RXBUF_NUM];
```

```
enet_ptp_normal_descriptors_chain_init(ENET_DMA_RX, ptp_rxdesc_tab);
```

enet_ptp_normal_descriptors_ring_init

函数enet_ptp_normal_descriptors_ring_init描述见下表:

表 3-232. 函数 enet_ptp_normal_descriptors_ring_init

函数名称	enet_ptp_normal_descriptors_ring_init
函数原型	void enet_ptp_normal_descriptors_ring_init(enet_dmadirection_enum direction, enet_descriptors_struct *desc_ptptab);
功能描述	初始化具有PTP功能的DMA接收/发送描述符为环模式
先决条件	-
被调用函数	-
输入参数{in}	
direction	描述符类型, 下列参数仅可选择一个
ENET_DMA_TX	DMA Tx描述符
ENET_DMA_RX	DMA Rx描述符
输入参数{in}	
desc_ptptab	PTP接收描述符列表首地址指针, 结构体成员介绍请参考 表3-172. 结构体 enet_descriptors_struct
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* initialize received descriptors in ring mode with PTP function */
enet_descriptors_struct ptp_rxdesc_tab[ENET_RXBUF_NUM];
enet_ptp_normal_descriptors_ring_init(ENET_DMA_RX, ptp_rxdesc_tab);
```

enet_ptpframe_receive_normal_mode

函数enet_ptpframe_receive_normal_mode描述见下表:

表 3-233. 函数 enet_ptpframe_receive_normal_mode

函数名称	enet_ptpframe_receive_normal_mode
函数原型	ErrStatus enet_ptpframe_receive_normal_mode(uint8_t *buffer, uint32_t bufsize, uint32_t timestamp[]);
功能描述	在PTP模式下处理当前接收到的帧, 并将当前描述符中存储的接收帧数据和时戳拷贝到指定区域
先决条件	-
被调用函数	-
输入参数{in}	
bufsize	缓冲区大小

输出参数{out}	
timestamp	存放时间戳指针
输出参数{out}	
buffer	存放数据的用户缓冲区指针，如果输入NULL，用户需要在调用该函数之前将数据拷贝到自己指定的位置
返回值	
ErrStatus	ERROR or SUCCESS

例如：

```
/* receive frame with timestamp from descriptor */
```

```
uint32_t rx_buffer[500];
```

```
uint32_t time_stamp[2];
```

```
ErrStatus status;
```

```
status = enet_ptpframe_receive_normal_mode (rx_buffer, 500, time_stamp);
```

enet_ptpframe_transmit_normal_mode

函数enet_ptpframe_transmit_normal_mode描述见下表：

表 3-234. 函数 enet_ptpframe_transmit_normal_mode

函数名称	enet_ptpframe_transmit_normal_mode
函数原型	ErrStatus enet_ptpframe_transmit_normal_mode(uint8_t *buffer, uint32_t length, uint32_t timestamp[]);
功能描述	在PTP模式下将指定区域内的数据拷贝到当前发送描述符中，并同时时间戳一起发送
先决条件	-
被调用函数	--
输入参数{in}	
buffer	存放数据的用户缓冲区指针，如果输入NULL，用户需要在调用该函数之前将数据拷贝到指定的位置
输入参数{in}	
length	发送数据大小
输出参数{out}	
timestamp	存放时间戳的指针，如果输入为NULL，则忽略时间戳
返回值	
ErrStatus	ERROR or SUCCESS

例如：

```
/* transmit frame with timestamp */
```

```
uint32_t tx_buffer[500];
```

```
uint32_t time_stamp[2];
```

```
ErrStatus status;
```

```
status = enet_ptpframe_transmit_normal_mode(tx_buffer, 500, time_stamp);
```

enet_wum_filter_register_pointer_reset

函数enet_wum_filter_register_pointer_reset描述见下表:

表 3-235. 函数 enet_wum_filter_register_pointer_reset

函数名称	enet_wum_filter_register_pointer_reset
函数原型	void enet_wum_filter_register_pointer_reset(void);
功能描述	远程唤醒帧过滤器寄存器指针复位
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 wakeup frame filter register pointer */
```

```
enet_wum_filter_register_pointer_reset ();
```

enet_wum_filter_config

函数enet_wum_filter_config描述见下表:

表 3-236. 函数 enet_wum_filter_config

函数名称	enet_wum_filter_config
函数原型	void enet_wum_filter_config(uint32_t pdata[]);
功能描述	配置远程唤醒帧寄存器
先决条件	-
被调用函数	-
输入参数{in}	
pdata	存放将写入远程唤醒帧寄存器组的数据指针（总共8字节）
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* configure the remote wakeup frame registers */
```

```
uint32_t wum_data[8];
```

```
enet_wum_filter_config(wum_data);
```

enet_wum_feature_enable

函数enet_wum_feature_enable描述见下表：

表 3-237. 函数 enet_wum_feature_enable

函数名称	enet_wum_feature_enable
函数原型	void enet_wum_feature_enable(uint32_t feature);
功能描述	使能ENET模块唤醒管理相关功能
先决条件	-
被调用函数	-
输入参数{in}	
feature	下列参数可以选择多个
ENET_WUM_POWER_DOWN	掉电模式
ENET_WUM_MAGIC_PACKET_FRAME	使能接收到魔法帧的唤醒事件
ENET_WUM_WAKEUP_FRAME	使能接收到唤醒帧的唤醒事件
ENET_WUM_GLOBAL_UNICAST	任何通过过滤器的单播帧均作为唤醒帧
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* enable wakeup management features */
```

```
enet_wum_feature_enable(ENET_WUM_POWER_DOWN);
```

enet_wum_feature_disable

函数enet_wum_feature_disable描述见下表：

表 3-238. 函数 enet_wum_feature_disable

函数名称	enet_wum_feature_disable
函数原型	void enet_wum_feature_disable(uint32_t feature)
功能描述	禁能ENET模块唤醒管理相关功能
先决条件	-
被调用函数	-
输入参数{in}	
feature	下列参数可以选择多个

ENET_WUM_MAGIC_PACKET_FRAME	使能接收到魔法帧的唤醒事件
ENET_WUM_WAKEUP_FRAME	使能接收到唤醒帧的唤醒事件
ENET_WUM_GLOBAL_UNICAST	任何通过过滤器的单播帧均作为唤醒帧
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* disable wakeup management features */
```

```
enet_wum_feature_disable(ENET_WUM_POWER_DOWN);
```

enet_msc_counters_reset

函数enet_msc_counters_reset描述见下表：

表 3-239. 函数 enet_msc_counters_reset

函数名称	enet_msc_counters_reset
函数原型	void enet_msc_counters_reset(void)
功能描述	复位MAC统计计数器组
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 MAC statistics counters */
```

```
enet_msc_counters_reset();
```

enet_msc_feature_enable

函数enet_msc_feature_enable描述见下表：

表 3-240. 函数 enet_msc_feature_enable

函数名称	enet_msc_feature_enable
函数原型	void enet_msc_feature_enable(uint32_t feature);
功能描述	使能MAC统计计数器相关功能

先决条件	-
被调用函数	-
输入参数{in}	
feature	下列参数可以选择多个
ENET_MSC_COUNTER_STOP_ROLLOVER	计数器停止回转
ENET_MSC_RESET_ON_READ	读时复位
ENET_MSC_COUNTER_FREEZE	MSC计数器冻结位
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* enable the MAC statistics counter features */
```

```
enet_msc_feature_enable(ENET_MSC_COUNTER_STOP_ROLLOVER);
```

enet_msc_feature_disable

函数enet_msc_feature_disable描述见下表：

表 3-241. 函数 enet_msc_feature_disable

函数名称	enet_msc_feature_disable
函数原型	void enet_msc_feature_disable(uint32_t feature);
功能描述	禁能MAC统计计数器相关功能
先决条件	-
被调用函数	-
输入参数{in}	
feature	下列参数可以选择多个
ENET_MSC_COUNTER_STOP_ROLLOVER	计数器停止回转
ENET_MSC_RESET_ON_READ	读时复位
ENET_MSC_COUNTER_FREEZE	MSC计数器冻结位
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* disable the MAC statistics counter features */
```

```
enet_msc_feature_disable(ENET_MSC_COUNTER_STOP_ROLLOVER);
```

enet_msc_counters_get

函数enet_msc_counters_get描述见下表：

表 3-242. 函数 enet_msc_counters_get

函数名称	enet_msc_counters_get
函数原型	uint32_t enet_msc_counters_get(enet_msc_counter_enum counter);
功能描述	获取MAC相关统计计数器值
先决条件	-
被调用函数	-
输入参数{in}	
counter	MSC计数器，下列参数仅可选择一个
ENET_MSC_TX_SC CNT	MSC 1次冲突后发送”好”帧的计数器
ENET_MSC_TX_MS CCNT	MSC 1次以上冲突后发送”好”帧的计数器
ENET_MSC_TX_TG FCNT	MSC发送”好”帧计数器
ENET_MSC_RX_RF CECNT	MSC CRC错误接收帧计数器
ENET_MSC_RX_RF AECNT	MSC对齐错误接收帧计数器
ENET_MSC_RX_RG UFCNT	MSC “好”单播帧接收帧计数器
输出参数{out}	
-	-
返回值	
uint32_t	MSC计数器值

例如：

```
/* get MAC statistics counter */
```

```
uint32_t reval;
```

```
reval = enet_msc_counters_get(ENET_MSC_TX_SCCNT);
```

enet_ptp_subsecond_2_nanosecond

函数enet_ptp_subsecond_2_nanosecond描述见下表：

表 3-243. 函数 `enet_ptp_subsecond_2_nanosecond`

函数名称	<code>enet_ptp_subsecond_2_nanosecond</code>
函数原型	<code>uint32_t enet_ptp_subsecond_2_nanosecond(uint32_t subsecond);</code>
功能描述	亚秒到纳秒的转换
先决条件	-
被调用函数	-
输入参数{in}	
subsecond	亚秒值，范围(0~0x7FFF FFFF)
输出参数{out}	
-	-
返回值	
uint32_t	纳秒值，范围(0~499999999)

例如：

```
/* change subsecond to nanosecond */
```

```
uint32_t reval;
```

```
reval = enet_ptp_subsecond_2_nanosecond(50);
```

`enet_ptp_nanosecond_2_subsecond`

函数`enet_ptp_nanosecond_2_subsecond`描述见下表：

表 3-244. 函数 `enet_ptp_nanosecond_2_subsecond`

函数名称	<code>enet_ptp_nanosecond_2_subsecond</code>
函数原型	<code>uint32_t enet_ptp_nanosecond_2_subsecond(uint32_t nanosecond);</code>
功能描述	纳秒到亚秒的转换
先决条件	-
被调用函数	-
输入参数{in}	
nanosecond	纳秒值，范围(0~499999999)
输出参数{out}	
-	-
返回值	
uint32_t	亚秒值，范围(0~0x7FFF FFFF)

例如：

```
/* change nanosecond to subsecond */
```

```
uint32_t reval;
```

```
reval = enet_ptp_nanosecond_2_subsecond(50);
```

enet_ptp_feature_enable

函数enet_ptp_feature_enable描述见下表：

表 3-245. 函数 enet_ptp_feature_enable

函数名称	enet_ptp_feature_enable
函数原型	void enet_ptp_feature_enable(uint32_t feature);
功能描述	使能PTP相关功能
先决条件	-
被调用函数	-
输入参数{in}	
feature	ENET模块PTP功能，下列参数可以选择多个
ENET_RTX_TIMES TAMP	发送/接收帧时间戳
ENET_PTP_TIMEST AMP_INT	时间戳中断触发
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* enable the PTP features */
```

```
enet_ptp_feature_enable(ENET_RTX_TIMESTAMP);
```

enet_ptp_feature_disable

函数enet_ptp_feature_disable描述见下表：

表 3-246. 函数 enet_ptp_feature_disable

函数名称	enet_ptp_feature_disable
函数原型	void enet_ptp_feature_disable(uint32_t feature);
功能描述	禁能PTP相关功能
先决条件	-
被调用函数	-
输入参数{in}	
feature	ENET模块PTP功能，下列参数可以选择多个
ENET_RTX_TIMES TAMP	发送/接收帧时间戳
ENET_PTP_TIMEST AMP_INT	时间戳中断触发
输出参数{out}	
-	-
返回值	

-	-
---	---

例如:

```
/* disable the PTP features */
```

```
enet_ptp_feature_disable(ENET_RXTX_TIMESTAMP);
```

enet_ptp_timestamp_function_config

函数enet_ptp_timestamp_function_config描述见下表:

表 3-247. 函数 enet_ptp_timestamp_function_config

函数名称	enet_ptp_timestamp_function_config
函数原型	ErrStatus enet_ptp_timestamp_function_config(enet_ptp_function_enum func);
功能描述	配置PTP时间戳相关功能
先决条件	-
被调用函数	-
输入参数{in}	
func	下列参数仅可选择一个
ENET_PTP_ADDEND_UPDATE	加数寄存器更新
ENET_PTP_SYSTIME_UPDATE	时间戳更新
ENET_PTP_SYSTIME_INIT	时间戳初始化
ENET_PTP_FINEMODE	精调模式更新系统时间戳
ENET_PTP_COARSEMODE	粗调模式更新系统时间戳
输出参数{out}	
-	-
返回值	
ErrStatus	SUCCESS or ERROR

例如:

```
/* configure the PTP timestamp function */
```

```
ErrStatus reval;
```

```
reval = enet_ptp_timestamp_function_config(ENET_PTP_ADDEND_UPDATE);
```

enet_ptp_subsecond_increment_config

函数enet_ptp_subsecond_increment_config描述见下表:

表 3-248. 函数 `enet_ptp_subsecond_increment_config`

函数名称	<code>enet_ptp_subsecond_increment_config</code>
函数原型	<code>void enet_ptp_subsecond_increment_config(uint32_t subsecond);</code>
功能描述	配置PTP系统时间亚秒增加值
先决条件	-
被调用函数	-
输入参数{in}	
subsecond	该值将被加到系统时间的亚秒值，范围（0~0xFF）
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* configure system time subsecond increment value */
```

```
enet_ptp_subsecond_increment_config(0x1F);
```

`enet_ptp_timestamp_addend_config`

函数`enet_ptp_timestamp_addend_config`描述见下表：

表 3-249. 函数 `enet_ptp_timestamp_addend_config`

函数名称	<code>enet_ptp_timestamp_addend_config</code>
函数原型	<code>void enet_ptp_timestamp_addend_config(uint32_t add);</code>
功能描述	精调模式下PTP时钟频率校准配置
先决条件	-
被调用函数	-
输入参数{in}	
add	通过将该值加到累加器用于时间同步，范围(0~0xFFFF FFFF)
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* adjusting the clock frequency only in fine update mode */
```

```
enet_ptp_timestamp_addend_config(0x1FFF);
```

`enet_ptp_timestamp_update_config`

函数`enet_ptp_timestamp_update_config`描述见下表：

表 3-250. 函数 `enet_ptp_timestamp_update_config`

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

函数原型	void enet_ptp_timestamp_update_config(uint32_t sign, uint32_t second, uint32_t subsecond);
功能描述	初始化时用于替换系统时间，在更新时表示在系统时间上加上或减去的秒值
先决条件	-
被调用函数	-
输入参数{in}	
sign	时间戳更新正或负符号位，下列参数仅可选择一个
ENET_PTP_ADD_TO_TIME	更新值加到系统时间
ENET_PTP_SUBSTRACT_FROM_TIME	将系统时间减去更新值
输入参数{in}	
second	秒值，范围(0~0xFFFF FFFF)
输入参数{in}	
subsecond	亚秒值，精度为0.46 ns，范围(0~0x7FFF FFFF)
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* update new value to system time */
```

```
enet_ptp_timestamp_update_config(ENET_PTP_ADD_TO_TIME, 0x10, 0);
```

enet_ptp_expected_time_config

函数enet_ptp_expected_time_config描述见下表：

表 3-251. 函数 enet_ptp_expected_time_config

函数名称	enet_ptp_expected_time_config
函数原型	void enet_ptp_expected_time_config(uint32_t second, uint32_t nanosecond);
功能描述	配置PTP期望时间
先决条件	-
被调用函数	-
输入参数{in}	
second	目标时间秒值，范围(0~0xFFFF FFFF)
输入参数{in}	
nanosecond	目标时间纳秒值，范围(0~0xFFFF FFFF)
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* configure the expected target time */
```

```
enet_ptp_expected_time_config(2000, 0);
```

enet_ptp_system_time_get

函数enet_ptp_system_time_get描述见下表:

表 3-252. 函数 enet_ptp_system_time_get

函数名称	enet_ptp_system_time_get
函数原型	void enet_ptp_system_time_get(enet_ptp_systime_struct *systime_struct);
功能描述	获取PTP当前系统时间
先决条件	-
被调用函数	-
输入参数{in}	
-	-
输出参数{out}	
systime_struct	PTP系统时间结构体指针，结构体成员介绍请参考 表3-173. 结构体 enet_ptp_systime_struct
返回值	
-	-

例如:

```
/* get the current system time */
```

```
enet_ptp_systime_struct systime;
```

```
enet_ptp_system_time_get(&systime);
```

enet_ptp_start

函数enet_ptp_start描述见下表:

表 3-253. 函数 enet_ptp_start

函数名称	enet_ptp_start
函数原型	void enet_ptp_start(int32_t updatemethod, uint32_t init_sec, uint32_t init_subsec, uint32_t carry_cfg, uint32_t accuracy_cfg);
功能描述	配置并启动PTP时间戳计数器
先决条件	-
被调用函数	enet_interrupt_disable/ enet_ptp_feature_enable/ enet_ptp_subsecond_increment_config/ enet_ptp_timestamp_addend_config/ enet_ptp_timestamp_function_config/ enet_ptp_flag_get/ enet_ptp_timestamp_update_config
输入参数{in}	
updatemethod	更新方式
ENET_PTP_FINEM	精调

ODE	
ENET_PTP_COARS EMODE	粗调
输入参数{in}	
init_sec	秒值，范围(0~0xFFFF FFFF)
输入参数{in}	
init_subsec	亚秒值，范围(0~0x7FFF FFFF)
输入参数{in}	
carry_cfg	该值将加到累加器中（精调模式下使用），范围(0~0xFFFF FFFF)
输入参数{in}	
accuracy_cfg	该值将加到系统时间的亚秒值，范围(0~0xFF)
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* start PTP function */
```

```
enet_ptp_start(ENET_PTP_FINEMODE, 0, 0, 50, 0);
```

enet_ptp_finecorrection_adjfreq

函数enet_ptp_finecorrection_adjfreq描述见下表：

表 3-254. 函数 enet_ptp_finecorrection_adjfreq

函数名称	enet_ptp_finecorrection_adjfreq
函数原型	void enet_ptp_finecorrection_adjfreq(int32_t carry_cfg);
功能描述	在精调模式下通过配置加数寄存器校准频率
先决条件	-
被调用函数	enet_ptp_timestamp_addend_config/ enet_ptp_timestamp_function_config
输入参数{in}	
carry_cfg	该值将加到累加器中（精调模式下使用），范围(0~0xFFFF FFFF)
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* adjust frequency in fine method by configure addend register */
```

```
enet_ptp_finecorrection_adjfreq(50);
```

enet_ptp_coarsecorrection_systime_update

函数enet_ptp_coarsecorrection_systime_update描述见下表:

表 3-255. 函数 enet_ptp_coarsecorrection_systime_update

函数名称	enet_ptp_coarsecorrection_systime_update
函数原型	void enet_ptp_coarsecorrection_systime_update(enet_ptp_systime_struct *systime_struct);
功能描述	粗调模式下更新系统时间
先决条件	-
被调用函数	enet_ptp_nanosecond_2_subsecond/ enet_ptp_timestamp_update_config/ enet_ptp_timestamp_function_config/ enet_ptp_flag_get/ enet_ptp_timestamp_addend_config
输入参数{in}	
systime_struct	PTP系统时间结构体指针，结构体成员介绍请参考 表3-173. 结构体 enet_ptp_systime_struct
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* update system time in coarse method */
enet_ptp_systime_struct systime_struct;
systime_struct.second = 0x0000FFFF;
systime_struct.nanosecond = 0;
systime_struct.sign = ENET_PTP_TIME_POSITIVE;
enet_ptp_coarsecorrection_systime_update (&systime_struct);
```

enet_ptp_finecorrection_settime

函数enet_ptp_finecorrection_settime描述见下表:

表 3-256. 函数 enet_ptp_finecorrection_settime

函数名称	enet_ptp_finecorrection_settime
函数原型	void enet_ptp_finecorrection_settime(enet_ptp_systime_struct *systime_struct);
功能描述	精调模式下配置系统时间
先决条件	-
被调用函数	enet_ptp_nanosecond_2_subsecond/ enet_ptp_timestamp_update_config/ enet_ptp_timestamp_function_config/ enet_ptp_flag_get
输入参数{in}	

sysptime_struct	PTP系统时间结构体指针，结构体成员介绍请参考 表3-173. 结构体 <i>enet_ptp_sysptime_struct</i>
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* set system time in fine method */
enet_ptp_sysptime_struct sysptime_struct;

sysptime_struct.second = 0x0000FFFF;

sysptime_struct.nanosecond = 0;

sysptime_struct.sign = ENET_PTP_TIME_POSITIVE;

enet_ptp_finecorrection_settime(&sysptime_struct);
```

enet_ptp_flag_get

函数enet_ptp_flag_get描述见下表：

表 3-257. 函数 enet_ptp_flag_get

函数名称	enet_ptp_flag_get
函数原型	FlagStatus enet_ptp_flag_get(uint32_t flag);
功能描述	获取PTP标志位状态
先决条件	-
被调用函数	-
输入参数{in}	
flag	PTP状态标志位
ENET_PTP_ADDEND_UPDATE	加数寄存器更新
ENET_PTP_SYSTIME_UPDATE	时间戳更新
ENET_PTP_SYSTIME_INIT	时间戳初始化
输出参数{out}	
-	-
返回值	
FlagStatus	SET or RESET

例如：

```
/* get the ptp flag status */

FlagStatus reval;
```

```
reval = enet_ptp_flag_get(ENET_PTP_ADDEND_UPDATE);
```

enet_initpara_reset

函数enet_initpara_reset描述见下表:

表 3-258. 函数 enet_initpara_reset

函数名称	enet_initpara_reset
函数原型	void enet_initpara_reset(void);
功能描述	复位 ENET initpara struct, 需在enet_initpara_config()函数前调用
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 ENET initpara struct */
```

```
enet_initpara_reset();
```

3.10. EXMC

外部存储器控制器EXMC, 用来访问各种片外存储器。章节[3.10.1](#)描述了EXMC的寄存器列表, 章节[3.10.2](#)对EXMC库函数进行说明。

3.10.1. 外设寄存器说明

EXMC寄存器列表如下表所示:

表 3-259. EXMC 寄存器

寄存器名称	寄存器描述
EXMC_SNCTLx (x=0, 1, 2, 3)	SRAM/NOR Flash控制寄存器
EXMC_SNTCFGx (x=0, 1, 2, 3)	SRAM/NOR Flash时序寄存器
EXMC_SNWTCFGx (x=0, 1, 2, 3)	SRAM/NOR Flash写时序寄存器
EXMC_NPCTLx (x=1, 2, 3)	NAND Flash/PC Card控制寄存器
EXMC_NPINTENx	NAND Flash/PC Card中断使能寄存器

寄存器名称	寄存器描述
(x=1, 2, 3)	
EXMC_NPCTCFGx (x=1, 2, 3)	NAND Flash/PC Card通用空间时序寄存器
EXMC_NPATCFGx (x=1, 2, 3)	NAND Flash/PC Card属性空间时序寄存器
EXMC_PIOTCFG3	PC Card I/O空间时序寄存器
EXMC_NECCx (x=1, 2)	NAND Flash ECC结果寄存器

3.10.2. 外设库函数说明

EXMC库函数列表如下表所示：

表 3-260. EXMC 库函数

库函数名称	库函数描述
exmc_norsram_deinit	复位NOR/SRAM region
exmc_norsram_struct_para_init	初始化结构体exmc_norsram_parameter_struct
exmc_norsram_init	初始化NOR/SRAM region
exmc_norsram_enable	使能bank0中某个region
exmc_norsram_disable	禁用bank0中某个region
exmc_nand_deinit	复位NAND区
exmc_nand_init	初始化NAND区
exmc_nand_struct_para_init	初始化结构体exmc_nand_parameter_struct
exmc_nand_enable	使能NAND区某个bank
exmc_nand_disable	禁用NAND区某个bank
exmc_nand_ecc_config	使能或者禁用NAND ECC功能
exmc_ecc_get	获取ECC值
exmc_pccard_deinit	复位PC Card区
exmc_pccard_init	初始化PC Card区
exmc_pccard_struct_para_init	初始化结构体exmc_pccard_parameter_struct
exmc_pccard_enable	使能PC Card区
exmc_pccard_disable	禁用PC Card区
exmc_interrupt_enable	中断使能
exmc_interrupt_disable	中断失能
exmc_interrupt_flag_get	获取中断标志位
exmc_interrupt_flag_clear	清除中断标志位
exmc_flag_get	获取标志位
exmc_flag_clear	清除标志位

结构体 `exmc_norsram_timing_parameter_struct`

表 3-261. 结构体 `exmc_norsram_timing_parameter_struct`

成员名称	功能描述
<code>asyn_access_mode</code>	异步访问模式
<code>syn_data_latency</code>	数据延迟
<code>syn_clk_division</code>	同步时钟分频比
<code>bus_latency</code>	总线延迟
<code>asyn_data_setup_time</code>	数据建立时间
<code>asyn_address_hold_time</code>	地址保持时间
<code>asyn_address_setup_time</code>	地址建立时间

结构体 `exmc_norsram_parameter_struct`

表 3-262. 结构体 `exmc_norsram_parameter_struct`

成员名称	功能描述
<code>norsram_region</code>	NOR/SRAM块的具体region
<code>write_mode</code>	写模式，同步模式或者异步模式
<code>extended_mode</code>	使能或者禁用扩展模式
<code>asyn_wait</code>	使能或者禁用异步等待功能
<code>nwait_signal</code>	在同步突发模式中，使能或者禁用NWAIT信号
<code>memory_write</code>	使能或者禁用写操作
<code>nwait_config</code>	配置NWAIT信号
<code>wrap_burst_mode</code>	使能或者禁用非对齐成组模式
<code>nwait_polarity</code>	指定NWAIT的极性
<code>burst_mode</code>	使能或者禁用突发模式
<code>databus_width</code>	指定外部存储器数据总线宽度
<code>memory_type</code>	指定外部存储器的类型
<code>address_data_mux</code>	数据线/地址线复用是否复用
<code>read_write_timing</code>	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
<code>write_timing</code>	未用扩展模式时，写时序参数

结构体 `exmc_nand_pccard_timing_parameter_struct`

表 3-263. 结构体 `exmc_nand_pccard_timing_parameter_struct`

成员名称	功能描述
<code>databus_hiztime</code>	写时序中空闲数据总线的高阻时间
<code>holdtime</code>	地址建立时间（或写时序中数据保持时间）
<code>waittime</code>	最小等待时间
<code>setup_time</code>	地址建立时间

结构体 `exmc_nand_parameter_struct`

表 3-264. 结构体 `exmc_nand_parameter_struct`

成员名称	功能描述
<code>nand_bank</code>	选择NAND块
<code>ecc_size</code>	ECC计算页大小
<code>atr_latency</code>	ALE低到RB低延迟时间
<code>ctr_latency</code>	CLE低到RB低延迟时间
<code>ecc_logic</code>	使能或禁用ECC计算逻辑
<code>databus_width</code>	NAND数据宽度
<code>wait_feature</code>	使能或禁用等待功能
<code>common_space_timing</code>	通用空间时间参数
<code>attribute_space_timing</code>	属性空间时间参数

结构体 `exmc_pccard_parameter_struct`

表 3-265. 结构体 `exmc_pccard_parameter_struct`

成员名称	功能描述
<code>atr_latency</code>	ALE低到RB低延迟时间
<code>ctr_latency</code>	CLE低到RB低延迟时间
<code>wait_feature</code>	使能或禁用等待功能
<code>common_space_timing</code>	通用空间时间参数
<code>attribute_space_timing</code>	属性空间时间参数
<code>io_space_timing</code>	IO空间时间参数

函数 `exmc_norsram_deinit`

函数`exmc_norsram_deinit`描述见下表：

表 3-266. 函数 `exmc_norsram_deinit`

函数名称	<code>exmc_norsram_deinit</code>
函数原型	<code>void exmc_norsram_deinit(uint32_t norsram_region);</code>
功能描述	复位NOR/SRAM region
先决条件	-
被调用函数	-
输入参数{in}	
<code>norsram_region</code>	bank0某个region
<code>EXMC_BANK0_NO RSRAM_REGIONx(x=0..3)</code>	bank0的region x

输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* deinitialize EXMC NOR/SRAM region 0 */
```

```
exmc_norsram_deinit(EXMC_BANK0_NORSRAM_REGION0);
```

函数 exmc_norsram_struct_para_init

函数exmc_norsram_struct_para_init描述见下表：

表 3-267. 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3-262. 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_init

函数exmc_norsram_init描述见下表：

表 3-268. 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 region
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3-262. 结构体 exmc_norsram_parameter_struct
输出参数{out}	
-	-
返回值	
-	-

例如：

```

/* initialize EXMC NOR/SRAM region */

exmc_norsram_parameter_struct nor_init_struct;

exmc_norsram_timing_parameter_struct nor_timing_init_struct;

exmc_norsram_struct_para_init(&nor_init_struct);

/* configure timing parameter */

nor_timing_init_struct.asyn_access_mode = EXMC_ACCESS_MODE_A;

nor_timing_init_struct.syn_data_latency = EXMC_DATA_LAT_2_CLK;

nor_timing_init_struct.syn_clk_division = EXMC_SYN_CLOCK_RATIO_DISABLE;

nor_timing_init_struct.bus_latency = 1;

nor_timing_init_struct.asyn_data_setup_time = 7;

nor_timing_init_struct.asyn_address_hold_time = 2;

nor_timing_init_struct.asyn_address_setup_time = 5;

/* configure EXMC bus parameters */

nor_init_struct.norsram_region = EXMC_BANK0_NORSRAM_REGION0;

nor_init_struct.write_mode = EXMC_ASYN_WRITE;

nor_init_struct.extended_mode = DISABLE;

nor_init_struct.asyn_wait = DISABLE;

nor_init_struct.nwait_signal = DISABLE;

nor_init_struct.memory_write = ENABLE;

nor_init_struct.nwait_config = EXMC_NWAIT_CONFIG_BEFORE;

nor_init_struct.wrap_burst_mode = DISABLE;

nor_init_struct.nwait_polarity = EXMC_NWAIT_POLARITY_LOW;

```

```

nor_init_struct.burst_mode = DISABLE;

nor_init_struct.databus_width = EXMC_NOR_DATABUS_WIDTH_16B;

nor_init_struct.memory_type = EXMC_MEMORY_TYPE_NOR;

nor_init_struct.address_data_mux = ENABLE;

nor_init_struct.read_write_timing = &nor_timing_init_struct;

nor_init_struct.write_timing = &nor_timing_init_struct;

exmc_norsram_init(&nor_init_struct);

```

函数 **exmc_norsram_enable**

函数exmc_norsram_enable描述见下表：

表 3-269. 函数 exmc_norsram_enable

函数名称	exmc_norsram_enable
函数原型	void exmc_norsram_enable(uint32_t norsram_region);
功能描述	使能bank0中某个region
先决条件	-
被调用函数	-
输入参数{in}	
norsram_region	NOR/PSRAM bank0某个region
EXMC_BANK0_NO RSRAM_REGIONx(x=0..3)	bank0的region x
输出参数{out}	
-	-
返回值	
-	-

例如：

```

/* enable region 0 of bank0 */

exmc_norsram_enable(EXMC_BANK0_NORSRAM_REGION0);

```

函数 **exmc_norsram_disable**

函数exmc_norsram_disable描述见下表：

表 3-270. 函数 exmc_norsram_disable

函数名称	exmc_norsram_disable
函数原型	void exmc_norsram_disable(uint32_t norsram_region);
功能描述	禁用bank0中某个region
先决条件	-

被调用函数	-
输入参数{in}	
norsram_region	NOR/PSRAM bank0某个region
EXMC_BANK0_NO RSRAM_REGIONx(x=0..3)	bank0的region x
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* disable region 0 of bank0 */
```

```
exmc_norsram_disable(EXMC_BANK0_NORSRAM_REGION0);
```

函数 exmc_nand_deinit

函数exmc_nand_deinit描述见下表：

表 3-271. 函数 exmc_nand_deinit

函数名称	exmc_nand_deinit
函数原型	void exmc_nand_deinit(uint32_t nand_bank);
功能描述	复位NAND区
先决条件	-
被调用函数	-
输入参数{in}	
nand_bank	选择NAND区
EXMC_BANKx_NA ND(x=1,2)	NAND某个区
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* deinitialize bank1 */
```

```
exmc_nand_deinit(EXMC_BANK1_NAND);
```

函数 exmc_nand_init

函数exmc_nand_init描述见下表：

表 3-272. 函数 exmc_nand_init

函数名称	exmc_nand_init
------	----------------

函数原型	void exmc_nand_init(exmc_nand_parameter_struct* exmc_nand_init_struct);
功能描述	初始化NAND区
先决条件	-
被调用函数	-
输入参数{in}	
exmc_nand_init_struct	初始化结构体，结构体成员参考 表3-264. 结构体 exmc_nand_parameter_struct
输出参数{out}	
-	-
返回值	
-	-

例如：

```

/* initialize EXMC NAND bank */

exmc_nand_parameter_struct nand_init_struct;

exmc_nand_pccard_timing_parameter_struct nand_timing_init_struct;

exmc_nand_struct_para_init(&nand_init_struct);

nand_timing_init_struct.setuptime = 1;
nand_timing_init_struct.waittime = 3;
nand_timing_init_struct.holdtime = 2;
nand_timing_init_struct.databus_hiztime = 2;

nand_init_struct.nand_bank = EXMC_BANK1_NAND;
nand_init_struct.ecc_size = EXMC_ECC_SIZE_2048BYTES;
nand_init_struct.atr_latency = EXMC_ALE_RE_DELAY_1_HCLK;
nand_init_struct.ctr_latency = EXMC_CLE_RE_DELAY_1_HCLK;
nand_init_struct.ecc_logic = ENABLE;

nand_init_struct.databus_width = EXMC_NAND_DATABUS_WIDTH_8B;
nand_init_struct.wait_feature = ENABLE;

nand_init_struct.common_space_timing = &nand_timing_init_struct;
nand_init_struct.attribute_space_timing = &nand_timing_init_struct;

exmc_nand_init(&nand_init_struct);

```

函数 exmc_nand_struct_para_init

函数exmc_nand_struct_para_init描述见下表：

表 3-273. 函数 `exmc_nand_struct_para_init`

函数名称	<code>exmc_nand_struct_para_init</code>
函数原型	<code>void exmc_nand_struct_para_init(exmc_nand_parameter_struct* exmc_nand_init_struct);</code>
功能描述	初始化结构体 <code>exmc_nand_parameter_struct</code>
先决条件	-
被调用函数	-
输入参数{in}	
<code>exmc_nand_parameter_struct</code>	初始化结构体，结构体成员参考 表3-262. 结构体 <code>exmc_norsram_parameter_struct</code>
输出参数{out}	
-	-
返回值	
-	-

例如：

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

函数 `exmc_nand_enable`

函数`exmc_nand_enable`描述见下表：

表 3-274. 函数 `exmc_nand_enable`

函数名称	<code>exmc_nand_enable</code>
函数原型	<code>void exmc_nand_enable(uint32_t nand_bank);</code>
功能描述	使能NAND区某个bank
先决条件	-
被调用函数	-
输入参数{in}	
<code>nand_bank</code>	选择NAND区
<code>EXMC_BANKx_NAND(x=1,2)</code>	NAND某个区
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* enable NAND bank */
exmc_nand_enable(EXMC_BANK1_NAND);
```

函数 exmc_nand_disable

函数exmc_nand_disable描述见下表：

表 3-275. 函数 exmc_nand_disable

函数名称	exmc_nand_disable
函数原型	void exmc_nand_disable(uint32_t nand_bank);
功能描述	禁用NAND区某个bank
先决条件	-
被调用函数	-
输入参数{in}	
nand_bank	选择NAND区
EXMC_BANKx_NA ND(x=1,2)	NAND某个区
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* disable NAND bank */
```

```
exmc_nand_disable(EXMC_BANK1_NAND);
```

函数 exmc_nand_ecc_config

函数exmc_nand_ecc_config描述见下表：

表 3-276. 函数 exmc_nand_ecc_config

函数名称	exmc_nand_ecc_config
函数原型	void exmc_nand_ecc_config(uint32_t nand_bank, ControlStatus new_value);
功能描述	使能或者禁用NAND ECC功能
先决条件	-
被调用函数	-
输入参数{in}	
nand_bank	选择NAND区
EXMC_BANKx_NA ND(x=1,2)	NAND某个区
输入参数{in}	
new_value	使能或者禁用
ENABLE	使能
DISABLE	禁用
输出参数{out}	
-	-
返回值	

-	-
---	---

例如：

```
/* enable the EXMC NAND ECC function */
exmc_nand_ecc_config(EXMC_BANK1_NAND, ENABLE);
```

函数 **exmc_ecc_get**

函数exmc_ecc_get描述见下表：

表 3-277. 函数 exmc_ecc_get

函数名称	exmc_ecc_get
函数原型	uint32_t exmc_ecc_get(uint32_t nand_bank);
功能描述	获取ECC值
先决条件	-
被调用函数	-
输入参数{in}	
nand_bank	选择NAND区
EXMC_BANKx_NAND(x=1,2)	NAND某个区
输出参数{out}	
-	-
返回值	
uint32_t	0-0xFFFFFFFF

例如：

```
/* get the EXMC ECC value */
uint32_t value;
value = exmc_ecc_get(EXMC_BANK1_NAND);
```

函数 **exmc_pccard_deinit**

函数exmc_pccard_deinit描述见下表：

表 3-278. 函数 exmc_pccard_deinit

函数名称	exmc_pccard_deinit
函数原型	void exmc_pccard_deinit(void);
功能描述	复位PC Card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：

```
/* deinitialize EXMC PC card bank */
```

```
exmc_pccard_deinit();
```

函数 **exmc_pccard_init**

函数exmc_pccard_init描述见下表：

表 3-279. 函数 exmc_pccard_init

函数名称	exmc_pccard_init
函数原型	void exmc_pccard_init(exmc_pccard_parameter_struct* exmc_pccard_init_struct);
功能描述	初始化PC Card区
先决条件	-
被调用函数	-
输入参数{in}	
exmc_pccard_init_struct	初始化结构体，结构体成员参考 表3-265. 结构体 exmc_pccard_parameter_struct
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* initialize EXMC PC card bank */
```

```
exmc_pccard_parameter_struct pccard_init_struct;
```

```
exmc_nand_pccard_timing_parameter_struct pccard_timing_init_struct;
```

```
exmc_pccard_struct_para_init(pccard_init_struct);
```

```
pccard_timing_init_struct.databus_hiztime = 2;
```

```
pccard_timing_init_struct.holdtime = 2;
```

```
pccard_timing_init_struct.waittime = 3;
```

```
pccard_timing_init_struct.setuptime = 1;
```

```
pccard_init_struct.atr_latency = EXMC_ALE_RE_DELAY_1_HCLK;
```

```
pccard_init_struct.ctr_latency = EXMC_CLE_RE_DELAY_1_HCLK;
```

```
pccard_init_struct.wait_feature = DISABLE;
```



```

pccard_init_struct.common_space_timing = &pccard_timing_init_struct;

pccard_init_struct.attribute_space_timing = &pccard_timing_init_struct;

pccard_init_struct.io_space_timing = &pccard_timing_init_struct;

exmc_pccard_init(&pccard_init_struct);

```

函数 exmc_pccard_struct_para_init

函数exmc_pccard_struct_para_init描述见下表:

表 3-280. 函数 exmc_pccard_struct_para_init

函数名称	exmc_pccard_struct_para_init
函数原型	void exmc_pccard_struct_para_init(exmc_pccard_parameter_struct* exmc_pccard_init_struct);
功能描述	初始化结构体exmc_pccard_parameter_struct
先决条件	-
被调用函数	-
输入参数{in}	
exmc_pccard_init_struct	初始化结构体，结构体成员参考 表3-265. 结构体 exmc_pccard_parameter_struct
输出参数{out}	
-	-
返回值	
-	-

例如:

```

/* initialize the struct exmc_pccard_parameter_struct */

exmc_pccard_parameter_struct pccard_init_struct;

exmc_pccard_struct_para_init(&pccard_init_struct);

```

函数 exmc_pccard_enable

函数exmc_pccard_enable描述见下表:

表 3-281. 函数 exmc_pccard_enable

函数名称	exmc_pccard_enable
函数原型	void exmc_pccard_enable(void);
功能描述	使能PC Card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 PC Card bank */
```

```
exmc_pccard_enable();
```

函数 **exmc_pccard_disable**

函数exmc_pccard_disable描述见下表：

表 3-282. 函数 exmc_pccard_disable

函数名称	exmc_pccard_disable
函数原型	void exmc_pccard_disable(void);
功能描述	禁用PC Card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 PC Card bank */
```

```
exmc_pccard_disable();
```

函数 **exmc_flag_get**

函数exmc_flag_get描述见下表：

表 3-283. 函数 exmc_flag_get

函数名称	exmc_flag_get
函数原型	FlagStatus exmc_flag_get(uint32_t bank, 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}	
bank	选择NAND区或PC Card区
EXMC_BANK1_NAND	NAND bank1
EXMC_BANK2_NAND	NAND bank2

ND	
EXMC_BANK3_PC CARD	PC Card区
输入参数{in}	
flag	标志位
EXMC_NAND_PCC ARD_FLAG_RISE	中断上升沿状态
EXMC_NAND_PCC ARD_FLAG_LEVEL	中断高电平状态
EXMC_NAND_PCC ARD_FLAG_FALL	中断下降沿状态
EXMC_NAND_PCC ARD_FLAG_FIFOE	FIFO空标志
输出参数{out}	
-	-
返回值	
FlagStatus	SET或RESET

例如：

```
/* check EXMC flag is set or not */
```

```
FlagStatus status;
```

```
status = exmc_flag_get(EXMC_BANK1_NAND, EXMC_NAND_PCCARD_FLAG_RISE);
```

函数 exmc_flag_clear

函数exmc_flag_clear描述见下表：

表 3-284. 函数 exmc_flag_clear

函数名称	exmc_flag_clear
函数原型	void exmc_flag_clear(uint32_t bank, uint32_t flag);
功能描述	清除标志位
先决条件	-
被调用函数	-
输入参数{in}	
bank	选择NAND区或PC Card区
EXMC_BANK1_NA ND	NAND bank1
EXMC_BANK2_NA ND	NAND bank2
EXMC_BANK3_PC CARD	PC Card区
输入参数{in}	
flag	标志位

EXMC_NAND_PCC ARD_FLAG_RISE	中断上升沿状态
EXMC_NAND_PCC ARD_FLAG_LEVEL	中断高电平状态
EXMC_NAND_PCC ARD_FLAG_FALL	中断下降沿状态
EXMC_NAND_PCC ARD_FLAG_FIFOE	FIFO空标志
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* clear EXMC flag */
```

```
exmc_flag_clear(EXMC_BANK1_NAND, EXMC_NAND_PCCARD_FLAG_RISE);
```

函数 exmc_interrupt_enable

函数exmc_interrupt_enable描述见下表：

表 3-285. 函数 exmc_interrupt_enable

函数名称	exmc_interrupt_enable
函数原型	void exmc_interrupt_enable(uint32_t bank, uint32_t interrupt);
功能描述	中断使能
先决条件	-
被调用函数	-
输入参数{in}	
bank	选择NAND区或PC Card区
EXMC_BANK1_NAND	NAND bank1
EXMC_BANK2_NAND	NAND bank2
EXMC_BANK3_PC_CARD	PC Card区
输入参数{in}	
interrupt	中断源
EXMC_NAND_PCC ARD_INT_RISE	中断上升沿检测
EXMC_NAND_PCC ARD_INT_LEVEL	中断高电平检测
EXMC_NAND_PCC ARD_INT_FALL	中断下降沿检测

输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* enable EXMC interrupt */
```

```
exmc_interrupt_enable(EXMC_BANK1_NAND, EXMC_NAND_PCCARD_INT_RISE);
```

函数 **exmc_interrupt_disable**

函数exmc_interrupt_disable描述见下表：

表 3-286. 函数 exmc_interrupt_disable

函数名称	exmc_interrupt_disable
函数原型	void exmc_interrupt_disable(uint32_t bank, uint32_t interrupt);
功能描述	中断失能
先决条件	-
被调用函数	-
输入参数{in}	
bank	选择NAND区或PC Card区
EXMC_BANK1_NAND	NAND bank1
EXMC_BANK2_NAND	NAND bank2
EXMC_BANK3_PCCARD	PC Card区
输入参数{in}	
interrupt	中断源
EXMC_NAND_PCCARD_INT_RISE	中断上升沿检测
EXMC_NAND_PCCARD_INT_LEVEL	中断高电平检测
EXMC_NAND_PCCARD_INT_FALL	中断下降沿检测
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* disable EXMC interrupt */
```

```
exmc_interrupt_disable(EXMC_BANK1_NAND, EXMC_NAND_PCCARD_INT_RISE);
```

函数 exmc_interrupt_flag_get

函数exmc_interrupt_flag_get描述见下表:

表 3-287. 函数 exmc_interrupt_flag_get

函数名称	exmc_interrupt_flag_get
函数原型	FlagStatus exmc_interrupt_flag_get(uint32_t bank, uint32_t int_flag);
功能描述	获取中断标志位
先决条件	-
被调用函数	-
输入参数{in}	
bank	选择NAND区或PC Card区
EXMC_BANK1_NAND	NAND bank1
EXMC_BANK2_NAND	NAND bank2
EXMC_BANK3_PCCARD	PC Card区
输入参数{in}	
int_flag	中断标志位
EXMC_NAND_PCCARD_INT_RISE	中断上升沿状态
EXMC_NAND_PCCARD_INT_LEVEL	中断高电平状态
EXMC_NAND_PCCARD_INT_FALL	中断下降沿状态
输出参数{out}	
-	-
返回值	
FlagStatus	SET或RESET

例如:

```
/* check EXMC interrupt flag is set or not */
```

```
FlagStatus status;
```

```
status = exmc_interrupt_flag_get(EXMC_BANK1_NAND,  
EXMC_NAND_PCCARD_INT_RISE);
```

函数 exmc_interrupt_flag_clear

函数exmc_interrupt_flag_clear描述见下表:

表 3-288. 函数 exmc_interrupt_flag_clear

函数名称	exmc_interrupt_flag_clear
------	---------------------------

函数原型	void exmc_interrupt_flag_clear(uint32_t bank, uint32_t int_flag);
功能描述	清除中断标志位
先决条件	-
被调用函数	-
输入参数{in}	
bank	选择NAND区或PC Card区
EXMC_BANK1_NAND	NAND bank1
EXMC_BANK2_NAND	NAND bank2
EXMC_BANK3_PCCARD	PC Card区
输入参数{in}	
int_flag	中断标志位
EXMC_NAND_PCCARD_INT_RISE	中断上升沿状态
EXMC_NAND_PCCARD_INT_LEVEL	中断高电平状态
EXMC_NAND_PCCARD_INT_FALL	中断下降沿状态
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* clear EXMC interrupt flag */
```

```
exmc_interrupt_flag_clear(EXMC_BANK1_NAND, EXMC_NAND_PCCARD_INT_RISE);
```

3.11. EXTI

EXTI是MCU中的中断/事件控制器，包括20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[3.11.1](#)描述了EXTI的寄存器列表，章节[3.11.2](#)对EXTI库函数进行说明。

3.11.1. 外设寄存器说明

EXTI寄存器列表如下表所示：

表 3-289. EXTI 寄存器

寄存器名称	寄存器描述
EXTI_INTEN	中断使能寄存器

寄存器名称	寄存器描述
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

3.11.2. 外设库函数说明

EXTI库函数列表如下表所示：

表 3-290. 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
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 3-291. 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

成员名称	功能描述
EXTI_13	EXTI中断线13
EXTI_14	EXTI中断线14
EXTI_15	EXTI中断线15
EXTI_16	EXTI中断线16
EXTI_17	EXTI中断线17
EXTI_18	EXTI中断线18
EXTI_19	EXTI中断线19

枚举类型 `exti_mode_enum`

表 3-292. 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 3-293. 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 3-294. 函数 `exti_deinit`

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

例如：

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

```
exti_deinit();
```

函数 exti_init

函数exti_init描述见下表:

表 3-295. 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3-291. 枚举类型exti_line_enum
输入参数{in}	
mode	EXTI模式, 参考 表3-292. 枚举类型exti_mode_enum
输入参数{in}	
trig_type	触发类型, 参考 表3-293. 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 3-296. 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3-291. 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 3-297. 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3-291. 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 3-298. 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3-291. 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 3-299. 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3-291. 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 3-300. 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3-291. 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 3-301. 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3-291. 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 3-302. 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3-291. 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 3-303. 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3-291. 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 3-304. 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3-291. 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 3-305. 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3-291. 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);
```

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

3.12.1. 外设寄存器说明

FMC寄存器列表如下：

表 3-306. FMC 寄存器

寄存器	描述
FMC_WS	等待状态寄存器
FMC_KEY0	解锁寄存器0
FMC_OBKEY	选项字节解锁寄存器
FMC_STAT0	状态寄存器0
FMC_CTL0	控制寄存器0
FMC_ADDR0	地址寄存器0
FMC_OBSTAT	选项字节状态寄存器
FMC_WP	写保护寄存器
FMC_KEY1	解锁寄存器1
FMC_STAT1	状态寄存器1
FMC_CTL1	控制寄存器1

寄存器	描述
FMC_ADDR1	地址寄存器1
FMC_WSEN	等待状态使能寄存器
FMC_PID	产品ID寄存器

3.12.2. 外设库函数说明

FMC固件库函数列举如下表:

表 3-307. 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_unlock	解锁FMC主编程块操作
fmc_bank0_unlock	解锁FMC主编程块bank0操作
fmc_bank1_unlock	解锁FMC主编程块bank1操作
fmc_lock	锁定FMC主编程块操作
fmc_bank0_lock	锁定FMC主编程块bank0操作
fmc_bank1_lock	锁定FMC主编程块bank1操作
fmc_page_erase	FMC 页擦除
fmc_mass_erase	FMC 全片擦除
fmc_bank0_erase	FMC bank0 全片擦除
fmc_bank1_erase	FMC bank1 全片擦除
fmc_word_program	在相应地址全字编程
fmc_halfword_program	在相应地址半字编程
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_spc_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_bank0_state_get	获取bank0状态
fmc_bank1_state_get	获取bank1状态

函数名称	函数描述
fmc_bank0_ready_wait	检查bank0是否准备好
fmc_bank1_ready_wait	检查bank1是否准备好

枚举 fmc_state_enum

表 3-308. 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_WPERR	写保护错误
FMC_TOERR	超时错误

枚举 fmc_int_enum

表 3-309. 枚举类型 fmc_int_enum

枚举名称	枚举描述
FMC_INT_BANK0_END	使能FMC bank0编程完成中断
FMC_INT_BANK0_ERR	使能FMC bank0错误中断
FMC_INT_BANK1_END	使能FMC bank1编程完成中断
FMC_INT_BANK1_ERR	使能FMC bank1错误中断

枚举 fmc_flag_enum

表 3-310. 枚举类型 fmc_flag_enum

枚举名称	枚举描述
FMC_FLAG_BANK0_BUSY	FMC bank0忙碌标志
FMC_FLAG_BANK0_PGERR	FMC bank0操作错误标志
FMC_FLAG_BANK0_WPERR	FMC bank0写保护错误标志
FMC_FLAG_BANK0_END	FMC bank0操作完成标志
FMC_FLAG_OBERR	FMC选项字节错误标志
FMC_FLAG_BANK1_BUSY	FMC bank1忙碌标志
FMC_FLAG_BANK1_PGERR	FMC bank1操作错误标志

枚举名称	枚举描述
_PGERR	
FMC_FLAG_BANK1_WPERR	FMC bank1写保护错误标志
FMC_FLAG_BANK1_END	FMC bank1操作完成标志

枚举 `fmc_interrupt_flag_enum`

表 3-311. 枚举类型 `fmc_interrupt_flag_enum`

枚举名称	枚举描述
FMC_INT_FLAG_BANK0_PGERR	FMC bank0操作错误中断标志
FMC_INT_FLAG_BANK0_WPERR	FMC bank0写保护错误中断标志
FMC_INT_FLAG_BANK0_END	FMC bank0操作完成中断标志
FMC_INT_FLAG_BANK1_PGERR	FMC bank1操作错误中断标志
FMC_INT_FLAG_BANK1_WPERR	FMC bank1写保护错误中断标志
FMC_INT_FLAG_BANK1_END	FMC bank1操作完成中断标志

函数 `fmc_wscent_set`

函数 `fmc_wscent_set` 描述见下表:

表 3-312. 函数 `fmc_wscent_set`

函数名称	<code>fmc_wscent_set</code>
函数原型	<code>void fmc_wscent_set(uint32_t wscent);</code>
功能描述	设置等待状态计数值
先决条件	-
被调用函数	<code>rcu_periph_reset_enable</code> / <code>rcu_periph_reset_disable</code>
输入参数{in}	
wscent	等待状态计数值
<code>WS_WSCNT_0</code>	FMC 0个等待状态
<code>WS_WSCNT_1</code>	FMC 1个等待状态
<code>WS_WSCNT_2</code>	FMC 2个等待状态
输出参数{out}	
-	-
返回值	
-	-

例如:

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

```
fmc_wsclnt_set (WS_WSCNT_1);
```

函数 fmc_unlock

函数fmc_unlock描述见下表：

表 3-313. 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_bank0_unlock

函数fmc_bank0_unlock描述见下表：

表 3-314. 函数 fmc_bank0_unlock

函数名称	fmc_bank0_unlock
函数原型	void fmc_bank0_unlock(void);
功能描述	解锁Flash bank0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 bank0 operation */
```

```
fmc_bank0_unlock( );
```

函数 fmc_bank1_unlock

函数fmc_bank1_unlock描述见下表：

表 3-315. 函数 fmc_bank1_unlock

函数名称	fmc_bank1_unlock
函数原型	void fmc_bank1_unlock(void);
功能描述	解锁Flash bank1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 bank1 operation */
```

```
fmc_bank1_unlock( );
```

函数 fmc_lock

函数fmc_lock描述见下表：

表 3-316. 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_bank0_lock`

函数 `fmc_bank0_lock` 描述见下表：

表 3-317. 函数 `fmc_bank0_lock`

函数名称	<code>fmc_bank0_lock</code>
函数原型	<code>void fmc_bank0_lock(void);</code>
功能描述	锁定flash bank0操作
先决条件	-
被调用函数	-
输入参数{in}	
-	-
输出参数{out}	
-	-
返回值	
-	-

Example:

```
/* lock the main FMC bank0 operation */
```

```
fmc_bank0_lock( );
```

函数 `fmc_bank1_lock`

函数 `fmc_bank1_lock` 描述见下表：

表 3-318. 函数 `fmc_bank1_lock`

函数名称	<code>fmc_bank1_lock</code>
函数原型	<code>void fmc_bank1_lock(void);</code>
功能描述	锁定flash bank1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 bank1 operation */
```

```
fmc_bank1_lock( );
```

函数 fmc_page_erase

函数fmc_page_erase描述见下表：

表 3-319. 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_bank0_ready_wait/ fmc_bank1_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，参考 表3-308. 枚举类型 fmc_state_enum

例如：

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

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

函数 fmc_mass_erase

函数fmc_mass_erase描述见下表：

表 3-320. 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_bank0_ready_wait/fmc_bank1_ready_wait
输入参数{in}	
-	-
输出参数{out}	
-	-
返回值	
fmc_state_enum	FMC状态，参考 表3-308. 枚举类型 fmc_state_enum

例如：

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

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

函数 **fmc_bank0_erase**

函数fmc_bank0_erase描述见下表：

表 3-321. 函数 **fmc_bank0_erase**

函数名称	fmc_bank0_erase
函数原型	fmc_state_enum fmc_bank0_erase(void);
功能描述	bank0全片擦除
先决条件	fmc_unlock
被调用函数	fmc_bank0_ready_wait
输入参数{in}	
-	-
输出参数{out}	
-	-
返回值	
fmc_state_enum	FMC状态，参考 表3-308. 枚举类型 fmc_state_enum

例如：

```
/* erase bank0 whole chip */
fmc_state_enum state = fmc_bank0_erase( );
```

函数 **fmc_bank1_erase**

函数fmc_bank1_erase描述见下表：

表 3-322. 函数 **fmc_bank1_erase**

函数名称	fmc_bank1_erase
函数原型	fmc_state_enum fmc_bank1_erase(void);
功能描述	bank1全片擦除
先决条件	fmc_unlock
被调用函数	fmc_bank1_ready_wait
输入参数{in}	
-	-
输出参数{out}	
-	-
返回值	
fmc_state_enum	FMC状态，参考 表3-308. 枚举类型 fmc_state_enum

例如：

```
/* erase bank1 whole chip */
fmc_state_enum state = fmc_bank1_erase( );
```

函数 `fmc_word_program`

函数 `fmc_word_program` 描述见下表:

表 3-323. 函数 `fmc_word_program`

函数名称	<code>fmc_word_program</code>
函数原型	<code>fmc_state_enum fmc_word_program(uint32_t address, uint32_t data);</code>
功能描述	对相应地址全字编程
先决条件	<code>fmc_unlock</code>
被调用函数	<code>fmc_bank0_ready_wait/fmc_bank1_ready_wait</code>
输入参数{in}	
<code>address</code>	编程地址
输入参数{in}	
<code>data</code>	编程数据
输出参数{out}	
-	-
返回值	
<code>fmc_state_enum</code>	FMC状态, 参考 表3-308. 枚举类型 <code>fmc_state_enum</code>

例如:

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

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

函数 `fmc_halfword_program`

函数 `fmc_halfword_program` 描述见下表:

表 3-324. 函数 `fmc_halfword_program`

函数名称	<code>fmc_halfword_program</code>
函数原型	<code>fmc_state_enum fmc_halfword_program(uint32_t address, uint16_t data);</code>
功能描述	对相应地址半字编程
先决条件	<code>fmc_unlock</code>
被调用函数	<code>fmc_bank0_ready_wait /fmc_bank1_ready_wait</code>
输入参数{in}	
<code>address</code>	编程地址
输入参数{in}	
<code>data</code>	编程数据
输出参数{out}	
-	-
返回值	
<code>fmc_state_enum</code>	FMC状态, 参考 表3-308. 枚举类型 <code>fmc_state_enum</code>

例如:

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



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

函数 ob_unlock

函数ob_unlock描述见下表:

表 3-325. 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 3-326. 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 3-327. 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_bank0_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 3-328. 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_bank0_ready_wait
输入参数{in}	
ob_wp	写保护单元
OB_WPx	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, 参考 表3-308. 枚举类型 fmc_state_enum

例如:

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

```
fmc_state_enum state = ob_write_protection_enable(OB_WP7);
```

函数 **ob_security_protection_config**

函数ob_security_protection_config描述见下表:

表 3-329. 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_bank0_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, 参考 表3-308. 枚举类型 fmc_state_enum

例如:

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

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

函数 **ob_user_write**

函数ob_user_write描述见下表:

表 3-330. 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_fw_dgt, uint8_t ob_deepsleep, uint8_t ob_stdby, uint8_t ob_boot);
功能描述	编辑用户选项字节
先决条件	ob_unlock
被调用函数	fmc_bank0_ready_wait
输入参数{in}	
ob_fw_dgt	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_N_RST	进入深度睡眠时不复位
OB_DEEPSLEEP_R_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
输入参数{in}	
ob_boot	选项字节bank启动值
OB_BOOT_B0	从bank0启动
OB_BOOT_B1	从bank1启动
输出参数{out}	
-	-
返回值	
fmc_state_enum	FMC状态, 参考 表3-308. 枚举类型 fmc_state_enum

例如:

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

```
fmc_state_enum state = ob_user_write(OB_FWDGT_HW,OB_DEEPSLEEP_RST,
OB_STDBY_RST, OB_BOOT_B1);
```

函数 ob_data_program

函数ob_data_program描述见下表:

表 3-331. 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_bank0_ready_wait
输入参数{in}	
address	0x1ffff804 / 0x1ffff806
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, 参考 表3-308. 枚举类型 fmc_state_enum

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 3-332. 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 3-333. 函数 ob_data_get

函数名称	ob_data_get
函数原型	uint8_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 */
```

```
uint8_t data = ob_data_get( );
```

函数 ob_write_protection_get

函数ob_write_protection_get描述见下表：

表 3-334. 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_spc_get

函数ob_spc_get描述见下表：

表 3-335. 函数 ob_spc_get

函数名称	ob_spc_get
函数原型	FlagStatus ob_spc_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_spc_get( );
```

函数 fmc_interrupt_enable

函数fmc_interrupt_enable描述见下表：

表 3-336. 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_BANK0_END	FMC bank0编程完成中断
FMC_INT_BANK0_ERR	FMC bank0错误中断
FMC_INT_BANK1_END	FMC bank1编程完成中断
FMC_INT_BANK1_ERR	FMC bank1错误中断
输出参数{out}	
-	-
返回值	
-	-

例如：

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

```
fmc_interrupt_enable(FMC_INT_BANK0_END);
```

函数 fmc_interrupt_disable

函数fmc_interrupt_disable描述见下表：

表 3-337. 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_BANK0_END	FMC bank0编程完成中断
FMC_INT_BANK0_ERR	FMC bank0错误中断

<i>ERR</i>	
<i>FMC_INT_BANK1_END</i>	FMC bank1编程完成中断
<i>FMC_INT_BANK1_ERR</i>	FMC bank1错误中断
输出参数{out}	
-	-
返回值	
-	-

例如:

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

```
fmc_interrupt_disable(FMC_INT_BANK0_END);
```

函数 **fmc_flag_get**

函数fmc_flag_get描述见下表:

表 3-338. 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
<i>FMC_FLAG_BANK0_BUSY</i>	FMC bank0忙碌标志
<i>FMC_FLAG_BANK0_PGERR</i>	FMC bank0操作错误标志
<i>FMC_FLAG_BANK0_WPERR</i>	FMC bank0写保护错误标志
<i>FMC_FLAG_BANK0_END</i>	FMC bank0操作完成标志
<i>FMC_FLAG_BANK1_BUSY</i>	FMC bank1忙碌标志
<i>FMC_FLAG_BANK1_PGERR</i>	FMC bank1操作错误标志
<i>FMC_FLAG_BANK1_WPERR</i>	FMC bank1写保护错误标志
<i>FMC_FLAG_BANK1_END</i>	FMC bank1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_BANK0_END);
```

函数 fmc_flag_clear

函数fmc_flag_clear描述见下表:

表 3-339. 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_BANK0_PGERR	FMC bank0操作错误标志
FMC_FLAG_BANK0_WPERR	FMC bank0写保护错误标志
FMC_FLAG_BANK0_END	FMC bank0操作完成标志
FMC_FLAG_BANK1_PGERR	FMC bank1操作错误标志
FMC_FLAG_BANK1_WPERR	FMC bank1写保护错误标志
FMC_FLAG_BANK1_END	FMC bank1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_BANK0_END);
```

函数 **fmc_interrupt_flag_get**

函数fmc_interrupt_flag_get描述见下表：

表 3-340. 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_BANK0_PGERR	FMC bank0操作错误标志
FMC_INT_FLAG_BANK0_WPERR	FMC bank0写保护错误标志
FMC_INT_FLAG_BANK0_END	FMC bank0操作完成标志
FMC_INT_FLAG_BANK1_PGERR	FMC bank1操作错误标志
FMC_INT_FLAG_BANK1_WPERR	FMC bank1写保护错误标志
FMC_INT_FLAG_BANK1_END	FMC bank1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_BANK0_PGERR);
```

函数 **fmc_interrupt_flag_clear**

函数fmc_interrupt_flag_clear描述见下表：

表 3-341. 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_BANK0_PGERR	FMC bank0操作错误标志
FMC_INT_FLAG_BANK0_WPERR	FMC bank0写保护错误标志
FMC_INT_FLAG_BANK0_END	FMC bank0操作完成标志
FMC_INT_FLAG_BANK1_PGERR	FMC bank1操作错误标志
FMC_INT_FLAG_BANK1_WPERR	FMC bank1写保护错误标志
FMC_INT_FLAG_BANK1_END	FMC bank1操作完成标志
输出参数{out}	
-	-
返回值	
-	-

例如：

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

```
fmc_interrupt_flag_clear(FMC_INT_FLAG_BANK0_PGERR);
```

函数 fmc_bank0_state_get

函数fmc_bank0_state_get描述见下表：

表 3-342. 函数 fmc_bank0_state_get

函数名称	fmc_bank0_state_get
函数原型	fmc_state_enum fmc_bank0_state_get(void);
功能描述	获取FMC bank0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, 参考 表3-308. 枚举类型 fmc_state_enum

例如:

```
/* get the FMC bank0 state */
```

```
fmc_state_enum state = fmc_bank0_state_get( );
```

函数 fmc_bank1_state_get

函数fmc_bank1_state_get描述见下表:

表 3-343. 函数 fmc_bank1_state_get

函数名称	fmc_bank1_state_get
函数原型	fmc_state_enum fmc_bank1_state_get(void);
功能描述	获取FMC bank1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, 参考 表3-308. 枚举类型 fmc_state_enum

例如:

```
/* get the FMC bank1 state */
```

```
fmc_state_enum state = fmc_bank1_state_get( );
```

函数 fmc_bank0_ready_wait

函数 fmc_bank0_ready_wait描述见下表:

表 3-344. 函数 fmc_bank0_ready_wait

函数名称	fmc_bank0_ready_wait
函数原型	fmc_state_enum fmc_bank0_ready_wait(uint32_t timeout);
功能描述	检查bank0是否准备好
先决条件	-
被调用函数	fmc_bank0_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，参考 表3-308. 枚举类型 fmc_state_enum
-----------------------	--

例如：

```
/* check whether FMC bank0 is ready or not */
fmc_state_enum state = fmc_bank0_ready_wait(0x00001000);
```

函数 fmc_bank1_ready_wait

函数fmc_bank1_ready_wait描述见下表：

表 3-345. 函数 fmc_bank0_ready_wait

函数名称	fmc_bank1_ready_wait
函数原型	fmc_state_enum fmc_bank1_ready_wait(uint32_t timeout);
功能描述	检查bank1是否准备好
先决条件	-
被调用函数	fmc_bank1_state_get
输入参数{in}	
timeout	循环次数
输出参数{out}	
-	-
返回值	
fmc_state_enum	FMC状态，参考 表3-308. 枚举类型 fmc_state_enum

例如：

```
/* check whether FMC bank1 is ready or not */
fmc_state_enum state = fmc_bank1_ready_wait(0x00001000 );
```

3.13. 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[3.13.1](#)描述了FWDGT的寄存器列表，章节[3.13.2](#)对FWDGT库函数进行说明。

3.13.1. 外设寄存器说明

FWDGT寄存器列表如下表所示：

表 3-346. 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

3.13.2. 外设库函数说明

FWDGT库函数列表如下表所示：

表 3-347. FWDGT 库函数

库函数名称	库函数描述
fw_dgt_write_enable	使能对寄存器FWDGT_PSC和FWDGT_RLD的写操作
fw_dgt_write_disable	失能对寄存器FWDGT_PSC和FWDGT_RLD的写操作
fw_dgt_enable	使能FWDGT
fw_dgt_prescaler_value_config	配置独立看门狗定时器预分频值
fw_dgt_reload_value_config	配置独立看门狗定时器重装载值
fw_dgt_counter_reload	按照FWDGT_RLD寄存器的值重装载FWDGT计数器
fw_dgt_config	设置FWDGT重装载值、预分频值
fw_dgt_flag_get	获取FWDGT标志位状态

函数 fw_dgt_write_enable

函数fw_dgt_write_enable描述见下表：

表 3-348. 函数 fw_dgt_write_enable

函数名称	fw_dgt_write_enable
函数原型	void fw_dgt_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 */
```

```
fw_dgt_write_enable();
```

函数 fw_dgt_write_disable

函数fw_dgt_write_disable描述见下表：

表 3-349. 函数 fw_dgt_write_disable

函数名称	fw_dgt_write_disable
函数原型	void fw_dgt_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 3-350. 函数 fwdgt_enable

函数名称	fw dgt_enable
函数原型	void fw dgt_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 3-351. 函数 fwdgt_prescaler_value_config

函数名称	fw dgt_prescaler_value_config
函数原型	ErrStatus fw dgt_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
<i>FWDGT_PSC_DIV4</i>	FWDGT预分频值设为4
<i>FWDGT_PSC_DIV8</i>	FWDGT预分频值设为8
<i>FWDGT_PSC_DIV16</i>	FWDGT预分频值设为16
<i>FWDGT_PSC_DIV32</i>	FWDGT预分频值设为32
<i>FWDGT_PSC_DIV64</i>	FWDGT预分频值设为64
<i>FWDGT_PSC_DIV128</i>	FWDGT预分频值设为128
<i>FWDGT_PSC_DIV256</i>	FWDGT预分频值设为256
输出参数{out}	
-	-
返回值	
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 3-352. 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 3-353. 函数 fwdgt_counter_reload

函数名称	fw dgt_counter_reload
函数原型	void fw dgt_counter_reload(void);
功能描述	按照FWDGT_RLD寄存器的值重装载FWDGT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 3-354. 函数 fwdgt_config

函数名称	fw dgt_config
函数原型	ErrStatus fw dgt_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

4	
FWDGT_PSC_DIV1 28	FWDGT prescaler set to 128
FWDGT_PSC_DIV2 56	FWDGT prescaler set to 256
输出参数{out}	
-	-
返回值	
ErrStatus	ERROR or SUCCESS-

例如:

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

```
ErrStatus status;
```

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

函数 fwdgt_flag_get

函数fwdgt_flag_get描述见下表:

表 3-355. 函数 fwdgt_flag_get

函数名称	fwdgt_flag_get
函数原型	FlagStatus fwdgt_flag_get(uint16_t flag);
功能描述	获取FWDGT标志位 状态
先决条件	-
被调用函数	-
输入参数{in}	
flag	需要获取状态的FWDGT标志 位
FWDGT_FLAG_PUD	预分频值更新进行中
FWDGT_FLAG_RU D	重装载值更新进行中
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);
```

3.14. 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。

3.14.1. 外设寄存器说明

GPIO寄存器列表如下表所示：

表 3-356. GPIO 寄存器

寄存器名称	寄存器描述
GPIOx_CTL0	GPIO端口控制寄存器0
GPIOx_CTL1	GPIO端口控制寄存器1
GPIOx_ISTAT	GPIO端口输入状态寄存器
GPIOx_OCTL	GPIO端口输出控制寄存器
GPIOx_BOP	GPIO端口位操作寄存器
GPIOx_BC	GPIO端口位清除寄存器
GPIOx_LOCK	GPIO端口配置锁定寄存器
AFIO_EC	AFIO事件控制寄存器
AFIO_PCF0	AFIO端口配置寄存器0
AFIO_EXTISS0	AFIO EXTI源选择寄存器0寄存器
AFIO_EXTISS1	AFIO EXTI源选择寄存器1寄存器
AFIO_EXTISS2	AFIO EXTI源选择寄存器2寄存器
AFIO_EXTISS3	AFIO EXTI源选择寄存器3寄存器
AFIO_PCF1	AFIO端口配置寄存器1

3.14.2. 外设库函数说明

GPIO库函数列表如下表所示：

表 3-357. 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	将特定的值写入指定的引脚
gpio_port_write	将特定的值写入指定的一组端口
gpio_input_bit_get	获取引脚的输入值
gpio_input_port_get	获取一组端口的输入值
gpio_output_bit_get	获取引脚的输出值
gpio_output_port_get	获取一组端口的输出值

库函数名称	库函数描述
gpio_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_ethernet_phy_select	以太网MII或RMII PHY选择

函数 gpio_deinit

函数gpio_deinit描述见下表:

表 3-358. 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,F,G)
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 3-359. 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 3-360. 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,F,G)
输入参数{in}	
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
输入参数{out}	
speed	GPIO输出最大速度
GPIO_OSPEED_10_MHZ	输出最大速度为10MHz
GPIO_OSPEED_2_MHZ	输出最大速度为2MHz

<i>GPIO_OSPEED_50MHZ</i>	输出最大速度为50MHz
输入参数{in}	
pin	GPIO引脚
<i>GPIO_PIN_x</i>	引脚选择 (x=0..15)
<i>GPIO_PIN_ALL</i>	所有引脚
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* 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 3-361. 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
<i>GPIOx</i>	端口选择(x = A,B,C,D,E,F,G)
输入参数{in}	
pin	GPIO引脚
<i>GPIO_PIN_x</i>	引脚选择 (x=0..15)
<i>GPIO_PIN_ALL</i>	所有引脚
输出参数{out}	
-	-
返回值	
-	-

例如:

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

函数 gpio_bit_reset

函数gpio_bit_reset描述见下表:

表 3-362. 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,F,G)
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 3-363. 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,F,G)
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
<i>RESET</i>	清除引脚值
<i>SET</i>	设置引脚值
输出参数{out}	
-	-
返回值	
-	-

例如:

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

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

函数 gpio_port_write

函数gpio_port_write描述见下表:

表 3-364. 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
<i>GPIOx</i>	端口选择(x = A,B,C,D,E,F,G)
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 3-365. 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,F,G)
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 3-366. 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,F,G)
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 3-367. 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,F,G)
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 3-368. 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,F,G)
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 3-369. 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 new value);
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

<code>GPIO_TIMER3_REMAP</code>	TIMER3 重映射
<code>GPIO_CAN_PARTIAL_REMAP</code>	CAN部分重映射（仅适用于GD32F10X_MD, GD32F10X_HD和GD32F10X_XD）
<code>GPIO_CAN_FULL_REMAP</code>	CAN全部重映射（仅适用于GD32F10X_MD, GD32F10X_HD和GD32F10X_XD）
<code>GPIO_CAN0_PARTIAL_REMAP</code>	CAN0部分重映射（仅适用于GD32F10X_CL）
<code>GPIO_CAN0_FULL_REMAP</code>	CAN0全部重映射（仅适用于GD32F10X_CL）
<code>GPIO_PD01_REMAP</code>	PD01重映射
<code>GPIO_TIMER4CH3_1_REMAP</code>	TIMER4 channel3内部重映射（仅适用于GD32F10X_CL和GD32F10X_HD）
<code>GPIO_ADC0_ETRGR_T_REMAP</code>	ADC0外部触发常规转换重映射（仅适用于GD32F10X_MD, GD32F10X_HD和GD32F10X_XD）
<code>GPIO_ADC1_ETRGR_T_REMAP</code>	ADC1外部触发常规转换重映射（仅适用于GD32F10X_MD, GD32F10X_HD和GD32F10X_XD）
<code>GPIO_ENET_REMAP</code>	ENET重映射（仅适用于GD32F10X_CL）
<code>GPIO_CAN1_REMAP</code>	CAN1重映射（仅适用于GD32F10X_CL）
<code>GPIO_SWJ_NONJTRST_REMAP</code>	全部的SWJ(JTAG-DP + SW-DP)，但是不包括NJTRST
<code>GPIO_SWJ_SWDENABLE_REMAP</code>	JTAG-DP除能，SW-DP使能
<code>GPIO_SWJ_DISABLE_REMAP</code>	JTAG-DP除能，SW-DP除能
<code>GPIO_SPI2_REMAP</code>	SPI2重映射（仅适用于GD32F10X_CL）
<code>GPIO_TIMER1_IT1_REMAP</code>	TIMER1 内部触发1重映射（仅适用于GD32F10X_CL）
<code>GPIO_PTP_PPS_REMAP</code>	以太网PTP PPS重映射（仅适用于GD32F10X_CL）
<code>GPIO_TIMER8_REMAP</code>	TIMER8 重映射
<code>GPIO_TIMER9_REMAP</code>	TIMER9 重映射
<code>GPIO_TIMER10_REMAP</code>	TIMER10重映射
<code>GPIO_TIMER12_REMAP</code>	TIMER12重映射
<code>GPIO_TIMER13_REMAP</code>	TIMER13重映射
<code>GPIO_EXMC_NADV_REMAP</code>	EXMC_NADV 连接/断开
输入参数{in}	

new value	是否使能
<i>ENABLE</i>	使能
<i>DISABLE</i>	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 3-370. 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 gpio_outputport, uint8_t gpio_outputpin);
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
<i>GPIO_EVENT_PORT_GPIOx</i>	源端口选择(x = A,B,C,D,E,F,G)
输入参数{in}	
gpio_outputpin	源端口引脚
<i>GPIO_EVENT_PIN_x</i>	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 3-371. 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 gpio_outputport, uint8_t gpio_outputpin);
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,F,G)
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 3-372. 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();
```

函数 gpio_event_output_disable

函数gpio_event_output_disable描述见下表:

表 3-373. 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();
```

函数 gpio_pin_lock

函数gpio_pin_lock描述见下表:

表 3-374. 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,F,G)
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_ethernet_phy_select

函数gpio_ethernet_phy_select描述见下表:

表 3-375. 函数 gpio_ethernet_phy_select

函数名称	gpio_ethernet_phy_select
函数原型	void gpio_ethernet_phy_select(uint32_t gpio_enetssel);
功能描述	以太网MII或RMII PHY选择
先决条件	-
被调用函数	-
输入参数{in}	
gpio_enetssel	以太网PHY选择
GPIO_ENET_PHY_MII	选择MII
GPIO_ENET_PHY_RMII	选择RMII
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* configure ethernet MAC for connection with an RMII PHY */
```

```
gpio_ethernet_phy_select(GPIO_ENET_PHY_RMII);
```

3.15. 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[3.15.1](#)描述了I2C的寄存器列表，章节[3.15.2](#)对I2C库函数进行说明。

3.15.1. 外设寄存器说明

I2C寄存器列表如下表所示:

表 3-376. 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

寄存器名称	寄存器描述
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

3.15.2. 外设库函数说明

I2C库函数列表如下表所示:

表 3-377. I2C 库函数

库函数名称	库函数描述
i2c_deinit	复位外设I2C
i2c_clock_config	配置I2C时钟
i2c_mode_addr_config	配置I2C地址
i2c_smbus_type_config	SMBus类型选择
i2c_ack_config	是否发送ACK
i2c_ackpos_config	ACK位置配置
i2c_master_addressing	主机发送从机地址
i2c_dualaddr_enable	使能双地址模式
i2c_dualaddr_disable	禁止双地址模式
i2c_enable	使能I2C模块
i2c_disable	禁止I2C模块
i2c_start_on_bus	在I2C总线上生成起始位
i2c_stop_on_bus	在I2C总线上生成停止位
i2c_data_transmit	发送数据
i2c_data_receive	接收数据
i2c_dma_config	I2C DMA模式使能
i2c_dma_last_transfer_config	配置I2C下一个DMA EOT是最后传输
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	配置报文错误校验
i2c_pec_transfer_config	配置I2C是否传输PEC值
i2c_pec_value_get	获取报文错误校验值
i2c_smbus_alert_config	配置SMBus引脚发送警告
i2c_smbus_arp_config	SMBus下ARP协议是否开启
i2c_flag_get	获取标志位
i2c_flag_clear	清除标志位
i2c_interrupt_enable	使能中断
i2c_interrupt_disable	禁止中断
i2c_interrupt_flag_get	获取中断标志位

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

枚举 i2c_flag_enum

表3-378. 枚举i2c_flag_enum

成员名称	功能描述
I2C_FLAG_SBSEND	发送起始位
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
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	I2C通信忙标志
I2C_FLAG_TRS	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

枚举 i2c_interrupt_enum

表3-379. 枚举i2c_interrupt_enum

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

枚举 i2c_interrupt_flag_enum

表3-380. 枚举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
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

成员名称	功能描述
I2C_INT_FLAG_BTC	字节发送结束
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

函数 i2c_deinit

函数i2c_deinit描述见下表:

表 3-381. 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 3-382. 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 3-383. 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}	
mode	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

7BITS	
I2C_ADDFORMAT_ 10BITS	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 3-384. 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 3-385. 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 3-386. 函数 i2c_ackpos_config

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

<i>T</i>	
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/*设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 3-387. 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 3-388. 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}	
dualaddr	双地址模式下的第二个地址
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* enable I2C0 dual-address */
i2c_dualaddr_enable(I2C0, 0xA0);
```

函数 i2c_dualaddr_disable

函数i2c_dualaddr_disable描述见下表：

表 3-389. 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	使能双地址模式
先决条件	-
被调用函数	-
输入参数{in}	
i2c_periph	I2C外设
I2Cx	(x=0,1)
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* disable I2C0 dual-address */
i2c_dualaddr_disable();
```


函数 i2c_enable

函数i2c_enable描述见下表:

表 3-390. 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 3-391. 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	禁止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 3-392. 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 3-393. 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 3-394. 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);
```

函数 i2c_data_receive

函数i2c_data_receive描述见下表:

表 3-395. 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 3-396. 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	使能I2C DMA 模式
先决条件	-
被调用函数	-
输入参数{in}	
i2c_periph	I2C外设
I2Cx	(x=0,1)
输入参数{in}	
dmastate	开启或关闭
I2C_DMA_ON	DMA 模式开启
I2C_DMA_OFF	DMA 模式关闭
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* I2C0 DMA mode enable */
```

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

函数 i2c_dma_last_transfer_config

函数i2c_dma_last_transfer_config描述见下表:

表 3-397. 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I2C下一个DMA EOT是最后传输
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	是否使能下一个DMA EOT是最后传输

<i>I2C_DMALST_ON</i>	使能
<i>I2C_DMALST_OFF</i>	不使能
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 3-398. 函数 *i2c_stretch_scl_low_config*

函数名称	<i>i2c_stretch_scl_low_config</i>
函数原型	<code>void i2c_stretch_scl_low_config(uint32_t i2c_periph, uint32_t stretchpara);</code>
功能描述	当从机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
<i>I2Cx</i>	(x=0,1)
输入参数{in}	
stretchpara	是否拉低SCL
<i>I2C_SCLSTRETCH_ENABLE</i>	拉低SCL
<i>I2C_SCLSTRETCH_DISABLE</i>	不拉低SCL
输出参数{out}	
-	-
返回值	
-	-

例如：

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

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

函数 **i2c_slave_response_to_gcall_config**

函数*i2c_slave_response_to_gcall_config*描述见下表：

表 3-399. 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_ENABL E	从机响应广播呼叫
I2C_GCEN_DISABL E	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 3-400. 函数 i2c_software_reset_config

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

输出参数{out}	
-	-
返回值	
-	-

例如:

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

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

函数 i2c_pec_config

函数i2c_pec_config描述见下表:

表 3-401. 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	配置报文错误校验
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}	
pecstate	开启或关闭
I2C_PEC_ENABLE	报文错误校验使能
I2C_PEC_DISABLE	报文错误校验关闭
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* enable I2C PEC calculation */
```

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

函数 i2c_pec_transfer_config

函数i2c_pec_transfer_config描述见下表:

表 3-402. 函数 i2c_pec_transfer_config

函数名称	i2c_pec_transfer_config
函数原型	void i2c_pec_transfer_config (uint32_t i2c_periph, uint32_t pecpara);
功能描述	配置I2C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 3-403. 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(0..255)

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 3-404. 函数 `i2c_smbus_issue_alert`

函数名称	<code>i2c_smbus_alert_config</code>
函数原型	<code>void i2c_smbus_alert_config (uint32_t i2c_periph, uint32_t smbuspara);</code>
功能描述	配置SMBA引脚发送警告
先决条件	-
被调用函数	-
输入参数{in}	
<code>i2c_periph</code>	I2C外设
<code>I2Cx</code>	(x=0,1)
输入参数{in}	
<code>smbuspara</code>	是否通过SMBA引脚发送警告
<code>I2C_SALTSEND_ENABLE</code>	通过SMBA引脚发送警告
<code>I2C_SALTSEND_DISABLE</code>	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 3-405. 函数 `i2c_smbus_arp_config`

函数名称	<code>i2c_smbus_arp_config</code>
函数原型	<code>void i2c_smbus_arp_config(uint32_t i2c_periph, uint32_t arpstate);</code>
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}	
<code>i2c_periph</code>	I2C外设
<code>I2Cx</code>	(x=0,1)
输入参数{in}	
<code>arpstate</code>	SMBus下ARP协议是否开启
<code>I2C_ARP_ENABLE</code>	使能ARP

I2C_ARP_DISABLE	关闭ARP
输出参数{out}	
-	-
返回值	
-	-

例如：

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

```
i2c_smbus_arp_config(I2C0, I2C_ARP_ENABLE);
```

函数 i2c_flag_get

函数i2c_flag_get描述见下表：

表 3-406. 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	获取标志位
先决条件	-
被调用函数	-
输入参数{in}	
i2c_periph	I2C外设
I2Cx	(x=0,1)
输入参数{in}	
flag	I2C标志，参考 表3-378. 枚举i2c_flag_enum
I2C_FLAG_SBSEN D	起始位是否发送
I2C_FLAG_ADDSE ND	主机模式下地址是否发送/从机模式下地址是否匹配
I2C_FLAG_BTC	字节传输完成
I2C_FLAG_ADD10 SEND	主机模式下10位地址地址头发送完成
I2C_FLAG_STPDE T	从机模式下监测到STOP结束位
I2C_FLAG_RBNE	接收期间I2C_DATA非空
I2C_FLAG_TBE	发送期间I2C_DATA为空
I2C_FLAG_BERR	总线错误，表示I2C总线上发生了预料之外的START起始位或STOP结束位
I2C_FLAG_LOSTA RB	主机模式下仲裁丢失
I2C_FLAG_AERR	应答错误
I2C_FLAG_OUERR	当禁用SCL拉低功能后，在从机模式下发生了过载或欠载事件
I2C_FLAG_PECER R	接收数据时PEC错误

<i>I2C_FLAG_SMBTO</i>	SMBus模式下超时信号
<i>I2C_FLAG_SMBAL</i> <i>T</i>	SMBus警报状态
<i>I2C_FLAG_MASTE</i> <i>R</i>	表明I2C时钟在主机模式还是从机模式的标志位
<i>I2C_FLAG_I2CBSY</i>	I2C通信忙标志
<i>I2C_FLAG_TRS</i>	I2C作发送端还是接收端
<i>I2C_FLAG_RXGC</i>	是否接收到广播地址(00h)
<i>I2C_FLAG_DEFSM</i> <i>B</i>	从机模式下SMBus主机地址头
<i>I2C_FLAG_HSTSM</i> <i>B</i>	从机模式下监测到SMBus主机地址头
<i>I2C_FLAG_DUMOD</i>	从机模式下双标志位表明哪个地址和双地址模式匹配
输出参数{out}	
-	-
返回值	
FlagStatus	SET / RESET

例如:

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

```
FlagStatus flag_state = RESET;
```

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

函数 i2c_flag_clear

函数i2c_flag_clear描述见下表:

表 3-407. 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	I2C标志位
<i>I2C_FLAG_SMBAL</i> <i>T</i>	SMBus警报状态
<i>I2C_FLAG_SMBTO</i>	SMBus模式下超时信号
<i>I2C_FLAG_PECER</i> <i>R</i>	接收数据时PEC错误

<i>I2C_FLAG_OUERR</i>	当禁用SCL拉低功能后，在从机模式下发生了过载或欠载事件
<i>I2C_FLAG_AERR</i>	应答错误
<i>I2C_FLAG_LOSTARB</i>	主机模式下仲裁丢失
<i>I2C_FLAG_BERR</i>	总线错误
<i>I2C_FLAG_ADDSEN</i>	主机模式下地址是否发送/从机模式下地址是否匹配，通过读 <i>I2C_STAT0</i> 和 <i>I2C_STAT1</i> 来清除
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* clear a bus error flag*/
```

```
i2c_flag_clear(I2C0, I2C_FLAG_BERR);
```

函数 i2c_interrupt_enable

函数i2c_interrupt_enable描述见下表：

表 3-408. 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	使能中断
先决条件	-
被调用函数	-
输入参数{in}	
<i>i2c_periph</i>	I2C外设
<i>I2Cx</i>	(x=0,1)
输入参数{in}	
<i>interrupt</i>	I2C中断，参考 表3-379. 枚举i2c_interrupt_enum
<i>I2C_INT_ERR</i>	错误中断
<i>I2C_INT_EV</i>	事件中断
<i>I2C_INT_BUF</i>	缓冲区中断
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* enable I2C0 error interrupt */
```

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

函数 i2c_interrupt_disable

函数i2c_interrupt_disable描述见下表：

表 3-409. 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	禁止中断
先决条件	-
被调用函数	-
输入参数{in}	
i2c_periph	I2C外设
I2Cx	(x=0,1)
输入参数{in}	
interrupt	I2C中断, 参考表3-379. 枚举i2c_interrupt_enum
I2C_INT_ERR	错误中断
I2C_INT_EV	事件中断
I2C_INT_BUF	缓冲区中断
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* disable I2C0 error 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 3-410. 函数 i2c_interrupt_flag_get

函数名称	i2c_interrupt_flag_get
函数原型	FlagStatus i2c_interrupt_flag_get(uint32_t i2c_periph, i2c_interrupt_flag_enum int_flag);
功能描述	获取中断标志位
先决条件	-
被调用函数	-
输入参数{in}	
i2c_periph	I2C外设
I2Cx	(x=0,1)
输入参数{in}	
int_flag	I2C中断标志, 参考 表3-380. 枚举i2c_interrupt_flag_enum
I2C_INT_FLAG_SB	主机模式下发送START起始位

<i>SEND</i>	
<i>I2C_INT_FLAG_AD DSEND</i>	主机模式下成功发送了地址 / 从机模式下接收到了地址并且和自身的地址匹配
<i>I2C_INT_FLAG_BT C</i>	字节发送结束
<i>I2C_INT_FLAG_AD D10SEND</i>	主机模式下10位地址地址头被发送
<i>I2C_INT_FLAG_ST PDET</i>	从机模式下监测到STOP结束位
<i>I2C_INT_FLAG_RB NE</i>	接收期间I2C_DATA非空
<i>I2C_INT_FLAG_TB E</i>	发送期间I2C_DATA为空
<i>I2C_INT_FLAG_BE RR</i>	总线错误
<i>I2C_INT_FLAG_LO STARB</i>	主机模式下仲裁丢失
<i>I2C_INT_FLAG_AE RR</i>	应答错误
<i>I2C_INT_FLAG_OU ERR</i>	当禁用SCL拉低功能后，在从机模式下发生了过载或欠载事件
<i>I2C_INT_FLAG_PE CERR</i>	接收数据时PEC错误
<i>I2C_INT_FLAG_SM BTO</i>	SMBus模式下超时信号
<i>I2C_INT_FLAG_SM BALT</i>	SMBus警报状态
输出参数{out}	
-	-
返回值	
FlagStatus	SET / RESET

例如：

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

```
FlagStatus flag_state = RESET;
```

```
flag_state = i2c_interrupt_flag_get(I2C0, I2C_INT_FLAG_BT);
```

函数 i2c_interrupt_flag_clear

函数i2c_interrupt_flag_clear描述见下表：

表 3-411. 函数 i2c_interrupt_flag_clear

函数名称	i2c_interrupt_flag_clear
------	--------------------------

函数原型	void i2c_interrupt_flag_clear(uint32_t i2c_periph, i2c_interrupt_flag_enum int_flag);
功能描述	清除中断标志位
先决条件	-
被调用函数	-
输入参数{in}	
i2c_periph	I2C外设
I2Cx	(x=0,1)
输入参数{in}	
int_flag	I2C中断标志
I2C_INT_FLAG_AD DSEND	主机模式下成功发送了地址 / 从机模式下接收到了地址并且和自身的地址匹配
I2C_INT_FLAG_BE RR	总线错误
I2C_INT_FLAG_LO STARB	主机模式下仲裁丢失
I2C_INT_FLAG_AE RR	应答错误
I2C_INT_FLAG_OU ERR	当禁用SCL 拉低功能后，在从机模式下发生了过载或欠载事件
I2C_INT_FLAG_PE CERR	接收数据时PEC错误
I2C_INT_FLAG_SM BTO	SMBus模式下超时信号
I2C_INT_FLAG_SM BALT	SMBus警报状态
输出参数{out}	
-	-
返回值	
-	-

例如：

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

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

3.16. MISC

MISC 是对嵌套向量中断控制器（NVIC）和系统定时器（SysTick）操作的软件包。章节 [3.16.1](#) 描述了 NVIC 和 SysTick 的寄存器列表，章节 [3.16.2](#) 对 MISC 库函数进行说明。

3.16.1. 外设寄存器说明

表 3-412. 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⁾	中断除能寄存器
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
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_cm3.h 文件中定义的结构体类型 NVIC_Type

2 参考 core_cm3.h 文件中定义的结构体类型 SCB_Type

表 3-413. 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_cm3.h 文件中定义的结构体类型 SysTick_Type

3.16.2. 外设库函数说明

枚举类型 IRQn_Type

表 3-414. 枚举类型 IRQn_Type

成员名称	功能描述
WWDGT_IRQn	窗口看门狗中断
LVD_IRQn	连接到 EXTI 线的 LVD 中断
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 全局中断
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
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_Channel6_IRQn	DMA0 通道 6 全局中断
ADC0_1_IRQn	ADC0 和 ADC1 全局中断
USBD_HP_CAN0_TX_IRQn / CAN0_TX_IRQn	USBD 高优先级或 CAN0 发送中断/ CAN0 发送中断
USBD_LP_CAN0_RX0_IRQn / CAN0_RX0_IRQn	USBD 低优先级或 CAN0 接收 0 中断/ CAN0 接收 0 中断
CAN0_RX1_IRQn	CAN0 接收 1 中断
CAN0_EWMC_IRQn	CAN0 EMMC 中断
EXTI5_9_IRQn	EXTI 线[9:5]中断

TIMER0_BRK_TIM ER8_IRQn / TIMER0_BRK_IRQn	TIMER0 中止中断和 TIMER8 全局中断/TIMER0 中止中断
TIMER0_UP_TIME R9_IRQn / TIMER0_UP_IRQn	TIMER0 更新中断和 TIMER9 全局中断/TIMER0 更新中断
TIMER0_TRG_CMT _TIMER10_IRQn / TIMER0_TRG_CMT _IRQn	TIMER0 触发与通道换相中断和 TIMER10 全局中断/TIMER0 触发与通道换相中 断
TIMER0_Channel_I RQn	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
EXTI0_15_IRQn	EXTI 线[15:0]中断
RTC_Alarm_IRQn	连接 EXTI 线的 RTC 闹钟中断
USBD_WKUP_IRQ n / USBFS_WKUP_IR Qn	连接 EXTI 线的 USBD 唤醒中断/连接 EXTI 线的 USBFS 唤醒中断
TIMER7_BRK_TIM ER11_IRQn / TIMER7_BRK_IRQn	TIMER7 中止中断和 TIMER11 全局中断/TIMER7 中止中断
TIMER7_UP_TIME R12_IRQn / TIMER7_UP_IRQn	TIMER7 更新中断和 TIMER12 全局中断/TIMER7 更新中断
TIMER7_TRG_CMT _TIMER13_IRQn / TIMER7_TRG_CMT _IRQn	TIMER7 触发与通道换相中断和 TIMER13 全局中断/TIMER7 触发与通道换相中 断
TIMER7_Channel_I RQn	TIMER7 通道捕获比较中断
ADC2_IRQn	ADC2 全局中断

EXMC_IRQn	EXMC 全局中断
SDIO_IRQn	SDIO 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_Channel4_IRQn / DMA1_Channel3_IRQn	DMA1 通道 3 全局中断和 DMA1 通道 4 全局中断/DMA1 通道 3 全局中断
DMA1_Channel4_IRQn	DMA1 通道 4 全局中断
ENET_IRQn	以太网全局中断
ENET_WKUP_IRQn	连接到 EXTI 线的以太网唤醒中断
CAN1_TX_IRQn	CAN1 发送中断
CAN1_RX0_IRQn	CAN1 接收 0 中断
CAN1_RX1_IRQn	CAN1 接收 1 中断
CAN1_EWMC_IRQn	CAN1 EWMC 中断
USBFS_IRQn	USBFS 全局中断

MISC库函数列表如下表所示:

表 3-415. 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_low power_set	设置系统低功耗模式状态
system_low power_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 3-416. 函数 `nvic_priority_group_set`

函数名称	<code>nvic_priority_group_set</code>
函数原形	<code>void nvic_priority_group_set(uint32_t nvic_prigroup);</code>
功能描述	设置优先级组
先决条件	-
被调用函数	-
输入参数{in}	
<code>nvic_prigroup</code>	优先级组
<code>NVIC_PRIGROUP_PRE0_SUB4</code>	0位用于抢占优先级，4位用于响应优先级
<code>NVIC_PRIGROUP_PRE1_SUB3</code>	1位用于抢占优先级，3位用于响应优先级
<code>NVIC_PRIGROUP_PRE2_SUB2</code>	2位用于抢占优先级，2位用于响应优先级
<code>NVIC_PRIGROUP_PRE3_SUB1</code>	3位用于抢占优先级，1位用于响应优先级
<code>NVIC_PRIGROUP_PRE4_SUB0</code>	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 3-417. 函数 `nvic_irq_enable`

函数名称	<code>nvic_irq_enable</code>
函数原形	<code>void nvic_irq_enable(uint8_t nvic_irq, uint8_t nvic_irq_pre_priority, uint8_t nvic_irq_sub_priority);</code>
功能描述	使能NVIC的中断
先决条件	-
被调用函数	<code>nvic_priority_group_set</code>
输入参数{in}	
<code>nvic_irq</code>	NVIC中断， 参考枚举类型 表3-414. 枚举类型IRQn_Type
输入参数{in}	
<code>nvic_irq_pre_priority</code>	抢占优先级（0~4）

输入参数{in}	
nvic_irq_sub_priority	响应优先级（0~4）
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 3-418. 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 表3-414. 枚举类型IRQn_Type
输出参数{out}	
-	-
返回值	
-	-

例如：

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

```
nvic_irq_disable(WWDGT_IRQn);
```

函数 nvic_vector_table_set

函数nvic_vector_table_set描述见下表：

表 3-419. 函数 nvic_vector_table_set

函数名称	nvic_vector_table_set
函数原形	void nvic_vector_table_set(uint32_t nvic_vect_tab, uint32_t offset);
功能描述	设置向量表基地址
先决条件	-
被调用函数	-
输入参数{in}	

nvic_vect_tab	RAM或者FLASH基地址
<i>NVIC_VECTTAB_RAM</i>	RAM 基地址
<i>NVIC_VECTTAB_FLASH</i>	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 3-420. 函数 system_lowpower_set

函数名称	system_low power_set
函数原形	void system_low power_set(uint8_t low power_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
<i>SCB_LPM_SLEEP_EXIT_ISR</i>	该位为1时，退出ISR时一直处于低功耗模式
<i>SCB_LPM_DEEPSLEEP</i>	该位为1时，系统处于deep sleep模式
<i>SCB_LPM_WAKE_BY_ALL_INT</i>	该位为1时，低功耗模式可以被所有中断唤醒（无论中断是否被使能）
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 3-421. 函数 `system_lowpower_reset`

函数名称	<code>system_low power_reset</code>
函数原形	<code>void system_low pow er_reset(uint8_t low pow er_mode);</code>
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}	
<code>lowpower_mode</code>	系统低功耗模式的状态
<code>SCB_LPM_SLEEP_EXIT_ISR</code>	该位为0时, 退出ISR时退出低功耗模式
<code>SCB_LPM_DEEPSLEEP</code>	该位为0时, 系统进入sleep模式
<code>SCB_LPM_WAKEUP_BY_ALL_INT</code>	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 3-422. 函数 `systick_clksource_set`

函数名称	<code>systick_clksource_set</code>
函数原形	<code>void systick_clksource_set(uint32_t systick_clksource);</code>
功能描述	设置SysTick时钟源
先决条件	-
被调用函数	-
输入参数{in}	
<code>systick_clksource</code>	systick时钟源
<code>SYSTICK_CLKSOURCE_HCLK</code>	systick时钟源为HCLK
<code>SYSTICK_CLKSOURCE_HCLK_DIV8</code>	systick时钟源为HCLK/8
输出参数{out}	

-	-
返回值	
-	-

例如:

```
/* systick clock source is HCLK/8 */
```

```
systick_clksource_set(SYSTICK_CLKSOURCE_HCLK_DIV8);
```

3.17. 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 [3.17.1](#) 描述了 PMU 的寄存器列表，章节 [3.17.2](#) 对 PMU 库函数进行说明。

3.17.1. 外设寄存器说明

PMU 寄存器列表如下表所示:

表 3-423. 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

3.17.2. 外设库函数说明

PMU 库函数列表如下表所示:

表 3-424. 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_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
pmu_wakeup_pin_disable	WKUP引脚唤醒除能
pmu_backup_write_enable	备份域写使能
pmu_backup_write_disable	备份域写除能
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 3-425. 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 3-426. 函数 pmu_lvd_select

函数名称	pmu_lvd_select
函数原型	void pmu_lvd_select(uint32_t lvd_t_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}	
lvd_t_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_lvd_disable

函数pmu_lvd_disable描述见下表:

表 3-427. 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 3-428. 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 3-429. 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 3-430. 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_standbymod();
```

函数 pmu_wakeup_pin_enable

函数pmu_wakeup_pin_enable描述见下表：

表 3-431. 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 3-432. 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 wakeup pin */
```

```
pmu_wakeup_pin_disable();
```

函数 pmu_backup_write_enable

函数pmu_backup_write_enable描述见下表：

表 3-433. 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 3-434. 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 backup domain write */
pmu_backup_write_disable();
```

函数 pmu_flag_get

函数pmu_flag_get描述见下表:

表 3-435. 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 3-436. 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_RESE T_WAKEUP	清除唤醒标志
PMU_FLAG_RESE T_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);
```

3.18. 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 [3.18.1](#) 描述了 RCU 的寄存器列表，章节 [3.18.2](#) 对 RCU 库函数进行说明。

3.18.1. 外设寄存器说明

RCU寄存器（中密度MD、高密度HD、超高密度XD产品）列表如下表所示：

表 3-437. RCU 寄存器（中密度、高密度、超高密度产品）

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
RCU_RSTSCK	复位源/时钟寄存器
RCU_DSV	深度睡眠模式电压寄存器

RCU寄存器（互联型CL产品）列表如下表所示：

表 3-438. RCU 寄存器（互联型产品）

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
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

3.18.2. 外设库函数说明

RCU库函数列表如下表所示：

表 3-439. 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	除能外设时钟
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，除能外设时钟
rcu_periph_reset_enable	使能外设复位
rcu_periph_reset_disable	除能外设复位
rcu_bkp_reset_enable	使能BKP复位
rcu_bkp_reset_disable	除能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_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

库函数名称	库函数描述
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
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_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	除能时钟稳定中断
rcu_oscstabilization_wait	等待振荡器稳定标志位置位或振荡器起振超时
rcu_osc_on	打开振荡器
rcu_osc_off	关闭振荡器
rcu_osc_bypass_mode_enable	使能振荡器时钟旁路模式
rcu_osc_bypass_mode_disable	除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	除能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 3-440. 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_SDIO	SDIO时钟(仅适用于HD、XD、MD系列)
RCU_USBFS	USBFS时钟(仅适用于CL系列)
RCU_ENET	ENET时钟(仅适用于CL系列)
RCU_ENETTX	ENETTX时钟(仅适用于CL系列)
RCU_ENETRX	ENETRX时钟(仅适用于CL系列)
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(仅适用于XD系列)
RCU_TIMER12	TIMER12时钟(仅适用于XD系列)

成员名称	功能描述
RCU_TIMER13	TIMER13时钟(仅适用于XD系列)
RCU_WWDGT	WWDGT时钟
RCU_SPI1	SPI1时钟
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_USBD	USBD时钟(仅适用于XD、HD、MD系列)
RCU_CAN0	CAN0时钟
RCU_CAN1	CAN1时钟(仅适用于CL系列)
RCU_BKPI	BKPI时钟
RCU_PMU	PMU时钟
RCU_DAC	DAC时钟
RCU_RTC	RTC时钟
RCU_AF	alternate function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_GPIOF	GPIOF时钟
RCU_GPIOG	GPIOG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(仅适用于CL系列)
RCU_TIMER8	TIMER8时钟(仅适用于XD系列)
RCU_TIMER9	TIMER9时钟(仅适用于XD系列)
RCU_TIMER10	TIMER10时钟(仅适用于XD系列)

枚举类型 rcu_periph_sleep_enum

表 3-441. 枚举类型 rcu_periph_sleep_enum

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

枚举类型 `rcu_periph_reset_enum`

表 3-442. 枚举类型 `rcu_periph_reset_enum`

成员名称	功能描述
<code>RCU_USBFSRST</code>	复位USBFS时钟(仅适用于CL系列)
<code>RCU_ENETRST</code>	复位ENET时钟(仅适用于CL系列)
<code>RCU_TIMER1RST</code>	复位TIMER1时钟
<code>RCU_TIMER2RST</code>	复位TIMER2时钟
<code>RCU_TIMER3RST</code>	复位TIMER3时钟
<code>RCU_TIMER4RST</code>	复位TIMER4时钟
<code>RCU_TIMER5RST</code>	复位TIMER5时钟
<code>RCU_TIMER6RST</code>	复位TIMER6时钟
<code>RCU_TIMER11RST</code>	复位TIMER11时钟(仅适用于XD系列)
<code>RCU_TIMER12RST</code>	复位TIMER12时钟(仅适用于XD系列)
<code>RCU_TIMER13RST</code>	复位TIMER13时钟(仅适用于XD系列)
<code>RCU_WWDGTRST</code>	复位WWDGT时钟
<code>RCU_SPI1RST</code>	复位SPI1时钟
<code>RCU_SPI2RST</code>	复位SPI2时钟
<code>RCU_USART1RST</code>	复位USART1时钟
<code>RCU_USART2RST</code>	复位USART2时钟
<code>RCU_UART3RST</code>	复位UART3时钟
<code>RCU_UART4RST</code>	复位UART4时钟
<code>RCU_I2C0RST</code>	复位I2C0时钟
<code>RCU_I2C1RST</code>	复位I2C1时钟
<code>RCU_USBD RST</code>	复位USB D时钟(仅适用于XD、HD、MD系列)
<code>RCU_CAN0RST</code>	复位CAN0时钟
<code>RCU_CAN1RST</code>	复位CAN1时钟
<code>RCU_BKPIRST</code>	复位BKPI时钟
<code>RCU_PMURST</code>	复位PMU时钟
<code>RCU_DACRST</code>	复位DAC时钟
<code>RCU_AFRST</code>	复位alternate function时钟
<code>RCU_GPIOARST</code>	复位GPIOA时钟
<code>RCU_GPIOBRST</code>	复位GPIOB时钟
<code>RCU_GPIOCRST</code>	复位GPIOC时钟
<code>RCU_GPIODRST</code>	复位GPIOD时钟
<code>RCU_GPIOERST</code>	复位GPIOE时钟
<code>RCU_GPIOFRST</code>	复位GPIOF时钟
<code>RCU_GPIOGRST</code>	复位GPIOG时钟
<code>RCU_ADC0RST</code>	复位ADC0时钟
<code>RCU_ADC1RST</code>	复位ADC1时钟
<code>RCU_TIMER0RST</code>	复位TIMER0时钟
<code>RCU_SPI0RST</code>	复位SPI0时钟

成员名称	功能描述
RCU_TIMER7RST	复位TIMER7时钟
RCU_USART0RST	复位USART0时钟
RCU_ADC2RST	复位ADC2时钟(仅适用于CL系列)
RCU_TIMER8RST	复位TIMER8时钟(仅适用于XD系列)
RCU_TIMER9RST	复位TIMER9时钟(仅适用于XD系列)
RCU_TIMER10RST	复位TIMER10时钟(仅适用于XD系列)
RCU_USART5RST	复位USART5时钟

枚举类型 `rcu_flag_enum`

表 3-443. 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(仅适用于CL系列)
RCU_FLAG_PLL2STB	PLL2 稳定标志(仅适用于CL系列)
RCU_FLAG_LXTALSTB	LXTAL稳定标志
RCU_FLAG_IRC40KSTB	IRC40K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 3-444. 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(仅适用于CL系列)
RCU_INT_FLAG_PLL2STB	PLL2时钟稳定中断标志(仅适用于CL系列)
RCU_INT_FLAG_CKM	外部高速晶振时钟阻塞中断标志

枚举类型 `rcu_int_flag_clear_enum`

表 3-445. 枚举类型 `rcu_int_flag_clear_enum`

成员名称	功能描述
<code>RCU_INT_FLAG_IRC40KSTB_CLR</code>	IRC40K时钟稳定中断清除标志
<code>RCU_INT_FLAG_LXTALSTB_CLR</code>	外部低速晶振时钟稳定中断清除标志
<code>RCU_INT_FLAG_IRC8MSTB_CLR</code>	IRC8M时钟稳定中断清除标志
<code>RCU_INT_FLAG_HXTALSTB_CLR</code>	外部高速晶振时钟稳定中断清除标志
<code>RCU_INT_FLAG_PLLSTB_CLR</code>	PLL时钟稳定中断清除标志
<code>RCU_INT_FLAG_PLL1STB_CLR</code>	PLL1时钟稳定中断清除标志(仅适用于CL系列)
<code>RCU_INT_FLAG_PLL2STB_CLR</code>	PLL2时钟稳定中断清除标志(仅适用于CL系列)
<code>RCU_INT_FLAG_CKM_CLR</code>	外部高速晶振时钟阻塞中断清除标志

枚举类型 `rcu_int_enum`

表 3-446. 枚举类型 `rcu_int_enum`

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

枚举类型 `rcu_osci_type_enum`

表 3-447. 枚举类型 `rcu_osci_type_enum`

成员名称	功能描述
<code>RCU_HXTAL</code>	外部高速振荡器
<code>RCU_LXTAL</code>	外部低速振荡器
<code>RCU_IRC8M</code>	IRC8M振荡器
<code>RCU_IRC40K</code>	IRC40K振荡器
<code>RCU_PLL_CK</code>	锁相环时钟
<code>RCU_PLL1_CK</code>	锁相环1时钟(仅适用于CL系列)
<code>RCU_PLL2_CK</code>	锁相环2时钟(仅适用于CL系列)

枚举类型 `rcu_clock_freq_enum`表 3-448. 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 3-449. 函数 `rcu_deinit`

函数名称	<code>rcu_deinit</code>
函数原形	<code>void rcu_deinit(void);</code>
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	<code>rcu_osc_i_stab_wait</code>
输入参数{in}	
-	-
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* deinitialize RCU */
```

```
rcu_deinit();
```

函数 `rcu_periph_clock_enable`

函数`rcu_periph_clock_enable`描述见下表:

表 3-450. 函数 `rcu_periph_clock_enable`

函数名称	<code>rcu_periph_clock_enable</code>
函数原形	<code>void rcu_periph_clock_enable(rcu_periph_enum periph);</code>
功能描述	使能外设时钟
先决条件	-
被调用函数	-
输入参数{in}	
<code>periph</code>	RCU外设, 具体参考 表3-440. 枚举类型<code>rcu_periph_enum</code>
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 3-451. 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	除能外设时钟
先决条件	-
被调用函数	-
输入参数{in}	
periph	RCU外设，具体参考 表3-440. 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 3-452. 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3-441. 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 3-453. 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，除能外设时钟
先决条件	-
被调用函数	-
输入参数{in}	
periph	RCU外设，参考 表3-441. 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 3-454. 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3-442. 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 3-455. 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	除能外设复位
先决条件	-
被调用函数	-
输入参数{in}	
periph_reset	RCU外设复位, 参考 表3-442. 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 3-456. 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 3-457. 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	除能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 3-458. 函数 rcu_system_clock_source_config

函数名称	rcu_system_clock_source_config
函数原形	void rcu_system_clock_source_config(uint32_t ck_sys);
功能描述	配置选择系统时钟源
先决条件	-
被调用函数	-
输入参数{in}	
ck_sys	系统时钟源选择
RCU_CKSYSSRC_I RC8M	选择CK_IRC8M时钟作为CK_SYS时钟源
RCU_CKSYSSRC_ HXTAL	选择CK_HXTAL时钟作为CK_SYS时钟源
RCU_CKSYSSRC_ PLL	选择CK_PLL时钟作为CK_SYS时钟源
输出参数{out}	
-	-
返回值	

-	-
---	---

例如：

```
/* configure the CK_HXTAL as the CK_SYS source */
```

```
rcu_system_clock_source_config(RCU_CKSYSSRC_HXTAL);
```

函数 rcu_system_clock_source_get

函数rcu_system_clock_source_get描述见下表：

表 3-459. 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	选择一个时钟作为CK_SYS时钟源
RCU_SCSS_IRC8M	选择CK_IRC8M作为CK_SYS时钟源
RCU_SCSS_HXTAL	选择CK_HXTAL作为CK_SYS时钟源
L	
RCU_SCSS_PLL	选择CK_PLL作为CK_SYS时钟源

例如：

```
/* get the CK_SYS source */
```

```
uint32_t temp_cksys_status;
```

```
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 3-460. 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
<i>RCU_AHB_CKSYS_DIVx</i>	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 3-461. 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
<i>RCU_APB1_CKAHB_DIV1</i>	选择CK_AHB为CK_APB1
<i>RCU_APB1_CKAHB_DIV2</i>	选择CK_AHB/2为CK_APB1
<i>RCU_APB1_CKAHB_DIV4</i>	选择CK_AHB/4为CK_APB1
<i>RCU_APB1_CKAHB_DIV8</i>	选择CK_AHB/8为CK_APB1
<i>RCU_APB1_CKAHB_DIV16</i>	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 3-462. 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_CK_AHB B_DIV1	选择CK_AHB为CK_APB2
RCU_APB2_CK_AHB B_DIV2	选择CK_AHB/2为CK_APB2
RCU_APB2_CK_AHB B_DIV4	选择CK_AHB/4为CK_APB2
RCU_APB2_CK_AHB B_DIV8	选择CK_AHB/8为CK_APB2
RCU_APB2_CK_AHB B_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_CK_AHB_DIV8);
```

函数 rcu_ckout0_config

函数rcu_ckout0_config描述见下表:

表 3-463. 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	CK_OUT0时钟源选择
RCU_CKOUT0SRC	无时钟输出

<code>_NONE</code>	
<code>RCU_CKOUT0SRC_CKSYS</code>	选择系统时钟CK_SYS
<code>RCU_CKOUT0SRC_IRC8M</code>	选择内部8M RC振荡器时钟
<code>RCU_CKOUT0SRC_HXTAL</code>	选择高速晶体振荡器时钟（HXTAL）
<code>RCU_CKOUT0SRC_CKPLL_DIV2</code>	选择（CK_PLL / 2）时钟
<code>RCU_CKOUT0SRC_CKPLL1</code>	选择CK_PLL1时钟
<code>RCU_CKOUT0SRC_CKPLL2_DIV2</code>	选择（CK_PLL2 / 2）时钟
<code>RCU_CKOUT0SRC_EXT1</code>	选择提供给ENET的EXT1时钟
<code>RCU_CKOUT0SRC_CKPLL2</code>	选择CK_PLL2时钟
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 3-464. 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
<code>RCU_PLLSRC_IRC8M_DIV2</code>	(IRC8M / 2)被选择为PLL时钟的时钟源
<code>RCU_PLLSRC_HXTAL</code>	HXTAL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
<i>RCU_PLL_MULx</i>	PLL源时钟 * x (在XD系列中x = 2..32, 在CL系列中, x = 2..14, 6.5, 16..32)
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_predv0_config (MD、HD、XD 产品)

函数rcu_predv0_config描述见下表:

表 3-465. 函数 rcu_predv0_config

函数名称	rcu_predv0_config
函数原形	void rcu_predv0_config(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_div	PREDV0分频因子
<i>RCU_PREDV0_DIV</i> x	PREDV0输入源时钟x分频 (x = 1,2)
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* configure the PREDV0 division factor */
```

```
rcu_predv0_config(RCU_PREDV0_DIV2);
```

函数 rcu_predv0_config (CL 型产品)

函数rcu_predv0_config描述见下表:

表 3-466. 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
<i>RCU_PREDV0SRC_HXTAL</i>	HXTAL被选择为PREDV0的时钟源
<i>RCU_PREDV0SRC_CKPLL1</i>	CK_PLL1被选择为PREDV0的时钟源
输入参数{in}	
predv0_div	PREDV0分频因子
<i>RCU_PREDV0_DIV</i> x	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, RCU_PREDV0_DIV4);
```

函数 **rcu_predv1_config** (CL 型产品)

函数rcu_predv1_config描述见下表:

表 3-467. 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
<i>RCU_PREDV1_DIV</i> x	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** (CL 型产品)

函数 **rcu_pll1_config** 描述见下表:

表 3-468. 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	PLL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** (CL 型产品)

函数 **rcu_pll2_config** 描述见下表:

表 3-469. 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	PLL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 3-470. 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	$CK_ADC = CK_APB2 / 16$
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 3-471. 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_CKP LL_DIV1_5	$CK_USBFS = CK_PLL / 1.5$
RCU_CKUSB_CKP LL_DIV1	$CK_USBFS = CK_PLL / 1$
RCU_CKUSB_CKP LL_DIV2_5	$CK_USBFS = CK_PLL / 2.5$
RCU_CKUSB_CKP LL_DIV2	$CK_USBFS = CK_PLL / 2$
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 3-472. 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_NO NE	没有时钟
RCU_RTCSRC_LX TAL	选择CK_LXTAL时钟作为RTC的时钟源
RCU_RTCSRC_IRC 40K	选择CK_IRC40K时钟作为RTC的时钟源
RCU_RTCSRC_HX TAL_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 (CL 型产品)

函数rcu_i2s1_clock_config描述见下表:

表 3-473. 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	I2S时钟源选择
RCU_I2S1SRC_CK SYS	系统时钟被选择为I2S1时钟的时钟源
RCU_I2S1SRC_CK PLL2_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 (CL 型产品)

函数rcu_i2s2_clock_config描述见下表:

表 3-474. 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	I2S时钟源选择
RCU_I2S2SRC_CK SYS	系统时钟被选择为I2S2时钟的时钟源
RCU_I2S2SRC_CK PLL2_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_osc_i_stab_wait

函数rcu_osc_i_stab_wait描述见下表：

表 3-475. 函数 rcu_osc_i_stab_wait

函数名称	rcu_osc_i_stab_wait
函数原形	ErrStatus rcu_osc_i_stab_wait(rcu_osc_i_type_enum osci);
功能描述	等待振荡器稳定标志位置位或振荡器起振超时
先决条件	-
被调用函数	rcu_flag_get
输入参数{in}	
osci	振荡器类型，参考 表3-447. 枚举类型rcu_osc_i_type_enum
输出参数{out}	
-	-
返回值	
ErrStatus	SUCCESS 或 ERROR

例如：

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

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

函数 rcu_osc_i_on

函数rcu_osc_i_on描述见下表：

表 3-476. 函数 rcu_osc_i_on

函数名称	rcu_osc_i_on
函数原形	void rcu_osc_i_on(rcu_osc_i_type_enum osci);
功能描述	打开振荡器
先决条件	-
被调用函数	-
输入参数{in}	
osci	振荡器类型，参考 表3-447. 枚举类型rcu_osc_i_type_enum

输出参数{out}	
-	-
返回值	
-	-

例如：

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

```
rcu_osc_i_on(RCU_HXTAL);
```

函数 rcu_osc_i_off

函数rcu_osc_i_off描述见下表：

表 3-477. 函数 rcu_osc_i_off

函数名称	rcu_osc_i_off
函数原形	void rcu_osc_i_off(rcu_osc_i_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3-447. 枚举类型rcu_osc_i_type_enum
输出参数{out}	
-	-
返回值	
-	-

例如：

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

```
rcu_osc_i_off(RCU_HXTAL);
```

函数 rcu_osc_i_bypass_mode_enable

函数rcu_osc_i_bypass_mode_enable描述见下表：

表 3-478. 函数 rcu_osc_i_bypass_mode_enable

函数名称	rcu_osc_i_bypass_mode_enable
函数原形	void rcu_osc_i_bypass_mode_enable(rcu_osc_i_type_enum osci);
功能描述	使能振荡器时钟旁路模式
先决条件	HXTALEN或LXTALEN应在使能振荡器时钟旁路模式前先复位
被调用函数	-
输入参数{in}	
osci	振荡器类型，参考 表3-447. 枚举类型rcu_osc_i_type_enum
RCU_HXTAL	高速晶体振荡器
RCU_LXTAL	低速晶体振荡器

输出参数{out}	
-	-
返回值	
-	-

例如：

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

```
rcu_osc_bypass_mode_enable(RCU_HXTAL);
```

函数 rcu_osc_bypass_mode_disable

函数rcu_osc_bypass_mode_disable描述见下表：

表 3-479. 函数 rcu_osc_bypass_mode_disable

函数名称	rcu_osc_bypass_mode_disable
函数原形	void rcu_osc_bypass_mode_disable(rcu_osc_type_enum osci);
功能描述	除能振荡器时钟旁路模式
先决条件	HXTALEN或LXTALEN应在使能振荡器时钟旁路模式前先复位
被调用函数	-
输入参数{in}	
osci	振荡器类型，参考 表3-447. 枚举类型rcu_osc_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_osc_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 3-480. 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 3-481. 函数 rcu_hxtal_clock_monitor_disable

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

例如：

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

```
rcu_hxtal_clock_monitor_disable();
```

函数 rcu_irc8m_adjust_value_set

函数rcu_irc8m_adjust_value_set描述见下表：

表 3-482. 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(uint8_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 3-483. 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_2	在深度睡眠模式下内核电压为1.2V
RCU_DEEPSLEEP_V_1_1	在深度睡眠模式下内核电压为1.1V
RCU_DEEPSLEEP_V_1_0	在深度睡眠模式下内核电压为1.0V
RCU_DEEPSLEEP_V_0_9	在深度睡眠模式下内核电压为0.9V
输出参数{out}	
-	-
返回值	
-	-

例如：

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

```
rcu_deepsleep_voltage_set(RCU_DEEPSLEEP_V_1_1);
```

函数 rcu_clock_freq_get

函数rcu_clock_freq_get描述见下表：

表 3-484. 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，参考 表3-448. 枚举类型rcu clock freq enum
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);
```

函数 rcu_flag_get

函数rcu_flag_get描述见下表：

表 3-485. 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3-443. 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 3-486. 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 3-487. 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3-444. 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 3-488. 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3-445. 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 3-489. 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，参考 表3-446. 枚举类型 rcu_int_enum
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 3-490. 函数 rcu_interrupt_disable

函数名称	rcu_interrupt_disable
函数原形	void rcu_interrupt_disable(rcu_int_enum stab_int);
功能描述	除能时钟稳定中断
先决条件	-
被调用函数	-
输入参数{in}	
stab_int	时钟稳定中断, 参考 表3-446. 枚举类型rcu_int_enum
输出参数{out}	
-	-
返回值	
-	-

例如:

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

3.19. 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[3.19.1](#)描述了RTC的寄存器列表, 章节[3.19.2](#)对RTC库函数进行说明。

3.19.1. 外设寄存器描述

RTC寄存器列表如下表所示:

表 3-491. 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

3.19.2. 外设库函数描述

RTC库函数列表如下表所示：

表 3-492. 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_lw_off_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寄存器同步标志位置位
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
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	除能RTC中断

函数 rtc_configuration_mode_enter

函数rtc_configuration_mode_enter描述见下表：

表 3-493. 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 3-494. 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 3-495. 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	调用此函数之前, 必须调用函数rtc_lw_off_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}	
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_lw_off_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 3-496. 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_lw_off_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_lw_off_wait( );

/* set RTC prescaler value to 0x7FFFF */
rtc_prescaler_set (0x7FFFF);
```

函数 rtc_lw_off_wait

函数rtc_lw_off_wait描述见下表:

表 3-497. 函数 rtc_lw_off_wait

函数名称	rtc_lw_off_wait
函数原型	void rtc_lw_off_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_lw_off_wait( );
```


函数 rtc_register_sync_wait

函数rtc_register_sync_wait描述见下表：

表 3-498. 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寄存器寄存器同步标志位置位
先决条件	-
被调用函数	-
输入参数{in}	
-	-
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* 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 3-499. 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 3-500. 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}	
-	-
返回值	
uint32_t	RTC计时器的值

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 3-501. 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	
uint32_t	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 3-502. 函数 `rtc_flag_get`

函数名称	<code>rtc_flag_get</code>
函数原型	<code>FlagStatus rtc_flag_get(uint32_t flag);</code>
功能描述	获取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
<code>RTC_FLAG_SECON</code> <code>D</code>	秒中断标志位
<code>RTC_FLAG_ALARM</code>	闹钟中断标志位
<code>RTC_FLAG_OVERFLOW</code> <code>LOW</code>	溢出中断标志位
<code>RTC_FLAG_RSYN</code>	寄存器同步标志位
<code>RTC_FLAG_LWOF</code>	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 alarm 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 3-503. 函数 `rtc_flag_clear`

函数名称	<code>rtc_flag_clear</code>
函数原型	<code>void rtc_flag_clear(uint32_t flag);</code>
功能描述	清除RTC标志位状态
先决条件	调用此函数之前, 必须调用函数 <code>rtc_lw_off_wait()</code> (等待标志位LWOF置位)
被调用函数	-
输入参数{in}	
flag	待清除的RTC标志位
<code>RTC_FLAG_SECON</code> <code>D</code>	秒中断标志位

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

例如:

```
/* wait until last write operation on RTC registers has finished */
```

```
rtc_lwoff_wait( );
```

```
/* 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 3-504. 函数 `rtc_interrupt_flag_get`

函数名称	<code>rtc_interrupt_flag_get</code>
函数原型	<code>FlagStatus rtc_interrupt_flag_get(uint32_t flag);</code>
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
<i>RTC_INT_FLAG_SECOND</i>	秒中断标志位
<i>RTC_INT_FLAG_ALARM</i>	闹钟中断标志位
<i>RTC_INT_FLAG_OVERFLOW</i>	溢出中断标志位
输出参数{out}	
-	-
返回值	
FlagStatus	SET或RESET

例如:

```
/* get the RTC overflow interrupt flag status */
```

```
FlagStatus overflow_status;
```

```
overflow_status = rtc_interrupt_flag_get(RTC_INT_FLAG_OVERFLOW);
```

函数 rtc_interrupt_flag_clear

函数rtc_interrupt_flag_clear描述见下表:

表 3-505. 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	调用此函数之前, 必须调用函数rtc_lw off_w ait () (等待标志位LWOFF置位)
被调用函数	-
输入参数{in}	
flag	需要清除的RTC中断标志位
RTC_INT_FLAG_SE COND	秒中断标志位
RTC_INT_FLAG_AL ARM	闹钟中断标志位
RTC_INT_FLAG_OV ERFLOW	溢出中断标志位
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* wait until last write operation on RTC registers has finished */
```

```
rtc_lw off_w ait( );
```

```
/* 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 3-506. 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_lw off_w ait () (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

<i>RTC_INT_FLAG_SECOND</i>	秒中断
<i>RTC_INT_FLAG_ALARM</i>	闹钟中断
<i>RTC_INT_FLAG_OVERFLOW</i>	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( );
/* enable the RTC second interrupt */
rtc_interrupt_enable(RTC_INT_FLAG_SECOND);
```

函数 `rtc_interrupt_disable`

函数`rtc_interrupt_disable`描述见下表:

表 3-507. 函数 `rtc_interrupt_disable`

函数名称	<code>rtc_interrupt_disable</code>
函数原型	<code>void rtc_interrupt_disable(uint32_t interrupt);</code>
功能描述	失能RTC中断
先决条件	调用此函数之前, 必须调用函数 <code>rtc_lwoff_wait()</code> (等待标志位LWOFF置位)
被调用函数	-
输入参数{in}	
interrupt	待除能的RTC中断源
<i>RTC_INT_SECOND</i>	秒中断
<i>RTC_INT_ALARM</i>	闹钟中断
<i>RTC_INT_OVERFLOW</i>	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_FLAG_SECOND);
```

3.20. SDIO

安全的数字输入/输出接口（SDIO）定义了SD卡、SD I/O卡、多媒体卡（MMC）和CE-ATA卡主机接口，提供AHB系统总线与SD存储卡、SD I/O卡、MMC和CE-ATA设备之间的数据传输。章节[3.20.1](#)描述了SDIO的寄存器列表，章节[3.20.2](#)对SDIO库函数进行说明。

3.20.1. 外设寄存器说明

SDIO寄存器列表如下表所示：

表 3-508. SDIO 寄存器

寄存器名称	寄存器描述
SDIO_PWRCTL	电源控制寄存器
SDIO_CLKCTL	时钟控制寄存器
SDIO_CMDAGMT	命令参数寄存器
SDIO_CMDCTL	命令控制寄存器
SDIO_RSPCMDIDX	命令索引响应寄存器
SDIO_RESPx x=0..3	响应寄存器
SDIO_DATA TO	数据超时寄存器
SDIO_DATALEN	数据长度寄存器
SDIO_DATACTL	数据控制寄存器
SDIO_DATACNT	数据计数寄存器
SDIO_STAT	状态寄存器
SDIO_INTC	中断清除寄存器
SDIO_INTEN	中断使能寄存器
SDIO_FIFOCNT	FIFO计数寄存器
SDIO_FIFO	FIFO数据寄存器

3.20.2. 外设库函数说明

SDIO库函数列表如下表所示：

表 3-509. SDIO 库函数

库函数名称	库函数描述
sdio_deinit	复位SDIOx
sdio_clock_config	配置SDIO时钟
sdio_hardware_clock_enable	使能硬件时钟控制
sdio_hardware_clock_disable	除能硬件时钟控制

库函数名称	库函数描述
sdio_bus_mode_set	设置多种SDIO卡总线模式
sdio_power_state_set	设置SDIO电源状态
sdio_power_state_get	获取SDIO电源状态
sdio_clock_enable	使能SDIO_CLK时钟
sdio_clock_disable	除能SDIO_CLK时钟
sdio_command_response_config	配置命令和响应
sdio_wait_type_set	设置命令状态机等待类型
sdio_csm_enable	使能命令状态机
sdio_csm_disable	除能命令状态机
sdio_command_index_get	获取上一次响应的命令索引
sdio_response_get	获取上一次响应的接收命令
sdio_data_config	配置数据超时、数据长度和数据块大小
sdio_data_transfer_config	配置数据传输模式和方向
sdio_dsm_enable	使能数据传输的数据状态机
sdio_dsm_disable	除能数据传输的数据状态机
sdio_data_write	在发送FIFO里写入数据（一个字）
sdio_data_read	在接收FIFO里读取数据（一个字）
sdio_data_counter_get	获取要传输到卡的剩余数据字节的数目
sdio_fifo_counter_get	从FIFO中获取要写入或读取的字数
sdio_dma_enable	使能SDIO的DMA请求
sdio_dma_disable	除能SDIO的DMA请求
sdio_flag_get	获取SDIO的标志位状态
sdio_flag_clear	清除SDIO的标志位状态
sdio_interrupt_enable	使能SDIO中断
sdio_interrupt_disable	除能SDIO中断
sdio_interrupt_flag_get	获取SDIO的中断标志位状态
sdio_interrupt_flag_clear	清除SDIO的中断标志位状态
sdio_readwait_enable	使能读等待模式（仅限SD I/O模式）
sdio_readwait_disable	除能读等待模式（仅限SD I/O模式）
sdio_stop_readwait_enable	使能停止读等待过程的功能（仅限SD I/O模式）
sdio_stop_readwait_disable	除能停止读等待过程的功能（仅限SD I/O模式）
sdio_readwait_type_set	设置读等待类型（仅限SD I/O模式）
sdio_operation_enable	使能SD I/O模式特定操作（仅限SD I/O模式）
sdio_operation_disable	除能SD I/O模式特定操作（仅限SD I/O模式）
sdio_suspend_enable	使能SD I/O休眠模式（仅限SD I/O模式）
sdio_suspend_disable	除能SD I/O休眠模式（仅限SD I/O模式）
sdio_ceata_command_enable	使能CE-ATA命令(仅限CE-ATA模式)
sdio_ceata_command_disable	除能CE-ATA命令(仅限CE-ATA模式)
sdio_ceata_interrupt_enable	使能CE-ATA中断(仅限CE-ATA模式)
sdio_ceata_interrupt_disable	除能CE-ATA中断(仅限CE-ATA模式)
sdio_ceata_command_completion_en	使能CE-ATA命令完成信号(仅限CE-ATA模式)

库函数名称	库函数描述
able	
sdio_ceata_command_completion_disable	除能CE-ATA命令完成信号(仅限CE-ATA模式)

函数 sdio_deinit

函数sdio_deinit描述见下表:

表 3-510. 函数 sdio_deinit

函数名称	sdio_deinit
函数原形	void sdio_deinit(void);
功能描述	复位SDIO
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 SDIO */
```

```
sdio_deinit();
```

函数 sdio_clock_config

函数sdio_clock_config描述见下表:

表 3-511. 函数 sdio_clock_config

函数名称	sdio_clock_config
函数原形	void sdio_clock_config(uint32_t clock_edge, uint32_t clock_bypass, uint32_t clock_powersave, uint16_t clock_division);
功能描述	配置SDIO时钟
先决条件	-
被调用函数	-
输入参数{in}	
clock_edge	SDIO_CLK时钟边沿选择
SDIO_SDIOLCKEDGE_RISING	选择SDIOCLK的上升沿产生SDIO_CLK
SDIO_SDIOLCKEDGE_FALLING	选择SDIOCLK的下降沿产生SDIO_CLK
输入参数{in}	

clock_bypass	旁路时钟使能
SDIO_CLOCKBYPASS_ENABLE	使能旁路时钟
SDIO_CLOCKBYPASS_DISABLE	失能旁路时钟
输入参数{in}	
clock_powersave	SDIO_CLK时钟动态开启/关闭以节省功耗
SDIO_CLOCKPWRSAVE_ENABLE	SDIO_CLK时钟在总线空闲时关闭
SDIO_CLOCKPWRSAVE_DISABLE	SDIO_CLK时钟总是开启
输入参数{in}	
clock_division	时钟分频，小于256
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* configure the SDIO clock */
```

```
sdio_clock_config(SDIO_SDIOLCKEDGE_RISING, SDIO_CLOCKBYPASS_DISABLE,
SDIO_CLOCKPWRSAVE_DISABLE, SD_CLK_DIV_TRANS);
```

函数 sdio_hardware_clock_enable

函数sdio_hardware_clock_enable描述见下表：

表 3-512. 函数 sdio_hardware_clock_enable

函数名称	sdio_hardware_clock_enable
函数原形	void sdio_hardware_clock_enable(void);
功能描述	使能硬件时钟控制
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 hardware clock control */
```

```
sdio_hardware_clock_enable();
```

函数 sdio hardware_clock_disable

函数sdio hardware_clock_disable描述见下表:

表 3-513. 函数 sdio hardware_clock_disable

函数名称	sdio hardware_clock_disable
函数原形	void sdio hardware_clock_disable(void);
功能描述	除能硬件时钟控制
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 hardware clock control */
sdio hardware_clock_disable();
```

函数 sdio_bus_mode_set

函数sdio_bus_mode_set描述见下表:

表 3-514. 函数 sdio_bus_mode_set

函数名称	sdio_bus_mode_set
函数原形	void sdio_bus_mode_set(uint32_t bus_mode);
功能描述	设置多种SDIO卡总线模式
先决条件	-
被调用函数	-
输入参数{in}	
bus_mode	SDIO卡总线模式
SDIO_BUSMODE_1 BIT	1位SDIO卡总线模式
SDIO_BUSMODE_4 BIT	4位SDIO卡总线模式
SDIO_BUSMODE_8 BIT	8位SDIO卡总线模式
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* set SDIO bus mode */
sdio_bus_mode_set(SDIO_BUSMODE_1BIT);
```

函数 **sdio_power_state_set**

函数sdio_power_state_set描述见下表：

表 3-515. 函数 sdio_power_state_set

函数名称	sdio_power_state_set
函数原形	void sdio_power_state_set(uint32_t power_state);
功能描述	设置SDIO电源状态
先决条件	-
被调用函数	-
输入参数{in}	
power_state	SDIO电源状态
SDIO_POWER_ON	SDIO上电
SDIO_POWER_OFF	SDIO断电
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* set SDIO power state */
sdio_power_state_set(SDIO_POWER_ON);
```

函数 **sdio_power_state_get**

函数sdio_power_state_get描述见下表：

表 3-516. 函数 sdio_power_state_get

函数名称	sdio_power_state_get
函数原形	uint32_t sdio_power_state_get(void);
功能描述	获取SDIO电源状态
先决条件	-
被调用函数	-
输入参数{in}	
-	-
输出参数{out}	
-	-
返回值	

uint32_t	SDIO_POWER_ON / SDIO_POWER_OFF
----------	--------------------------------

例如:

```
/* get the SDIO power state */
uint32_t sdio_power_value;

sdio_power_value = sdio_power_state_get();
```

函数 sdio_clock_enable

函数sdio_clock_enable描述见下表:

表 3-517. 函数 sdio_clock_enable

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

例如:

```
/* enable SDIO_CLK clock output */
sdio_clock_enable();
```

函数 sdio_clock_disable

函数sdio_clock_disable描述见下表:

表 3-518. 函数 sdio_clock_disable

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

-	-
---	---

例如：

```
/* disable SDIO_CLK clock output */
sdio_clock_disable();
```

函数 **sdio_command_response_config**

函数sdio_command_response_config描述见下表：

表 3-519. 函数 sdio_command_response_config

函数名称	sdio_command_response_config
函数原形	void sdio_command_response_config(uint32_t cmd_index, uint32_t cmd_argument, uint32_t response_type);
功能描述	配置命令和响应
先决条件	-
被调用函数	-
输入参数{in}	
cmd_index	命令索引，请参阅相关规范
输入参数{in}	
cmd_argument	命令参数，请参阅相关规范
输入参数{in}	
response_type	命令响应类型
SDIO_RESPONSETYPE_NO	无响应
SDIO_RESPONSETYPE_SHORT	短响应
SDIO_RESPONSETYPE_LONG	长响应
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* CMD2(SD_CMD_ALL_SEND_CID) command response config*/
sdio_command_response_config(SD_CMD_ALL_SEND_CID, (uint32_t)0x0,
SDIO_RESPONSETYPE_LONG);
```

函数 **sdio_wait_type_set**

函数sdio_wait_type_set描述见下表：

表 3-520. 函数 `sdio_wait_type_set`

函数名称	<code>sdio_wait_type_set</code>
函数原形	<code>void sdio_wait_type_set(uint32_t wait_type);</code>
功能描述	设置命令状态机等待类型
先决条件	-
被调用函数	-
输入参数{in}	
<code>wait_type</code>	等待类型
<code>SDIO_WAITTYPE_NO</code>	不等待中断
<code>SDIO_WAITTYPE_INTERRUPT</code>	等待中断
<code>SDIO_WAITTYPE_DATAEND</code>	等待数据传输结束
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* set the command state machine wait type */
sdio_wait_type_set(SDIO_WAITTYPE_NO);
```

函数 `sdio_csm_enable`

函数`sdio_csm_enable`描述见下表：

表 3-521. 函数 `sdio_csm_enable`

函数名称	<code>sdio_csm_enable</code>
函数原形	<code>void sdio_csm_enable(void);</code>
功能描述	使能命令状态机
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 CSM(command state machine) */
sdio_csm_enable();
```

函数 sdio_csm_disable

函数sdio_csm_disable描述见下表:

表 3-522. 函数 sdio_csm_disable

函数名称	sdio_csm_disable
函数原形	void sdio_csm_disable(void);
功能描述	除能命令状态机
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 CSM(command state machine) */
sdio_csm_disable();
```

函数 sdio_command_index_get

函数sdio_command_index_get描述见下表:

表 3-523. 函数 sdio_command_index_get

函数名称	sdio_command_index_get
函数原形	uint8_t sdio_command_index_get(void);
功能描述	获取上一次响应的命令索引
先决条件	-
被调用函数	-
输入参数{in}	
-	-
输出参数{out}	
-	-
返回值	
uint8_t	上一次响应的命令索引

例如:

```
/* get SDIO command index */
uint8_t sdio_commond_value;
sdio_commond_value = sdio_command_index_get();
```


函数 **sdio_response_get**

函数sdio_response_get描述见下表:

表 3-524. 函数 **sdio_response_get**

函数名称	sdio_response_get
函数原形	uint32_t sdio_response_get(uint32_t responsex);
功能描述	获取上一次响应的接收命令
先决条件	-
被调用函数	-
输入参数{in}	
responsex	SDIO响应
SDIO_RESPONSE0	卡响应 [31:0]/卡响应 [127:96]
SDIO_RESPONSE1	卡响应 [95:64]
SDIO_RESPONSE2	卡响应 [63:32]
SDIO_RESPONSE3	卡响应 [31:1], 加上位0
输出参数{out}	
-	-
返回值	
uint32_t	上一次响应的接收命令

例如:

```
/* store the CID0 numbers */
uint32_t sdio_cid[0];
sdio_cid[0] = sdio_response_get(SDIO_RESPONSE0);
```

函数 **sdio_data_config**

函数sdio_data_config描述见下表:

表 3-525. 函数 **sdio_data_config**

函数名称	sdio_data_config
函数原形	void sdio_data_config(uint32_t data_timeout, uint32_t data_length, uint32_t data_blocksize);
功能描述	配置数据超时、数据长度和数据块大小
先决条件	-
被调用函数	-
输入参数{in}	
data_timeout	卡总线时钟周期中的数据超时周期
输入参数{in}	
data_length	要传输的数据字节数
输入参数{in}	
data_blocksize	块传输中数据块的大小

SDIO_DATABLOCK SIZE_1BYTE	块大小 = 1字节
SDIO_DATABLOCK SIZE_2BYTES	块大小 = 2字节
SDIO_DATABLOCK SIZE_4BYTES	块大小 = 4字节
SDIO_DATABLOCK SIZE_8BYTES	块大小 = 8字节
SDIO_DATABLOCK SIZE_16BYTES	块大小 = 16字节
SDIO_DATABLOCK SIZE_32BYTES	块大小 = 32字节
SDIO_DATABLOCK SIZE_64BYTES	块大小 = 64字节
SDIO_DATABLOCK SIZE_128BYTES	块大小 = 128字节
SDIO_DATABLOCK SIZE_256BYTES	块大小 = 256字节
SDIO_DATABLOCK SIZE_512BYTES	块大小 = 512字节
SDIO_DATABLOCK SIZE_1024BYTES	块大小 = 1024字节
SDIO_DATABLOCK SIZE_2048BYTES	块大小 = 2048字节
SDIO_DATABLOCK SIZE_4096BYTES	块大小 = 4096字节
SDIO_DATABLOCK SIZE_8192BYTES	块大小 = 8192字节
SDIO_DATABLOCK SIZE_16384BYTES	块大小 = 16384字节
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* configure SDIO data */
```

```
sdio_data_config(0, 0, SDIO_DATABLOCKSIZE_1BYTE);
```

函数 **sdio_data_transfer_config**

函数sdio_data_transfer_config描述见下表:

表 3-526. 函数 `sdio_data_transfer_config`

函数名称	<code>sdio_data_transfer_config</code>
函数原形	<code>void sdio_data_transfer_config(uint32_t transfer_mode, uint32_t transfer_direction);</code>
功能描述	配置数据传输模式和方向
先决条件	-
被调用函数	-
输入参数{in}	
<code>transfer_mode</code>	数据传输模式
<code>SDIO_TRANSMODE_BLOCK</code>	块传输模式
<code>SDIO_TRANSMODE_STREAM</code>	流传输或SDIO多字节传输模式
输入参数{in}	
<code>transfer_direction</code>	数据传输方向
<code>SDIO_TRANSDIRECTION_TOCARD</code>	写数据到卡上
<code>SDIO_TRANSDIRECTION_TOSDIO</code>	从卡中读取数据
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* configure SDIO data transmisson */
```

```
sdio_data_transfer_config(SDIO_TRANSDIRECTION_TOSDIO,  
SDIO_TRANSMODE_BLOCK);
```

函数 `sdio_dsm_enable`

函数`sdio_dsm_enable`描述见下表：

表 3-527. 函数 `sdio_dsm_enable`

函数名称	<code>sdio_dsm_enable</code>
函数原形	<code>void sdio_dsm_enable(void);</code>
功能描述	使能数据传输的数据状态机
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 DSM(data state machine) */
```

```
sdio_dsm_enable();
```

函数 **sdio_dsm_disable**

函数sdio_dsm_disable描述见下表：

表 3-528. 函数 sdio_dsm_disable

函数名称	sdio_dsm_disable
函数原形	void sdio_dsm_disable(void);
功能描述	除能数据传输的数据状态机
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 DSM(data state machine) */
```

```
sdio_dsm_disable();
```

函数 **sdio_data_write**

函数sdio_data_write描述见下表：

表 3-529. 函数 sdio_data_write

函数名称	sdio_data_write
函数原形	void sdio_data_write(uint32_t data);
功能描述	在发送FIFO里写入数据（一个字）
先决条件	-
被调用函数	-
输入参数{in}	
data	往卡里写入32位数据
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* write data(one word) to the transmit FIFO */
```

```
sdio_data_write(0x0000 0001);
```

函数 **sdio_data_read**

函数sdio_data_read描述见下表：

表 3-530. 函数 sdio_data_read

函数名称	sdio_data_read
函数原形	uint32_t sdio_data_read(void);
功能描述	在接收FIFO里读取数据（一个字）
先决条件	-
被调用函数	-
输入参数{in}	
-	-
输出参数{out}	
-	-
返回值	
uint32_t	接收的数据

例如：

```
/* read data(one word) from the receive FIFO */
```

```
sdio_data_read();
```

函数 **sdio_data_counter_get**

函数sdio_data_counter_get描述见下表：

表 3-531. 函数 sdio_data_counter_get

函数名称	sdio_data_counter_get
函数原形	uint32_t sdio_data_counter_get(void);
功能描述	获取要传输到卡的剩余数据字节的数目
先决条件	-
被调用函数	-
输入参数{in}	
-	-
输出参数{out}	
-	-
返回值	
uint32_t	要传输的剩余数据字节数

例如：

```
/* get the number of remaining data bytes to be transferred to card */
```

```
uint32_t sdio_data_value;
```

```
sdio_data_value = sdio_data_counter_get();
```

函数 **sdio_fifo_counter_get**

函数sdio_fifo_counter_get描述见下表:

表 3-532. 函数 sdio_data_counter_get

函数名称	sdio_fifo_counter_get
函数原形	uint32_t sdio_fifo_counter_get(void);
功能描述	从FIFO中获取要写入或读取的字数
先决条件	-
被调用函数	-
输入参数{in}	
-	-
输出参数{out}	
-	-
返回值	
uint32_t	剩余字数

例如:

```
/* get the number of words remaining to be written or read from FIFO */
```

```
uint32_t sdio_fifo_value;
```

```
sdio_fifo_value = sdio_fifo_counter_get();
```

函数 **sdio_dma_enable**

函数sdio_dma_enable描述见下表:

表 3-533. 函数 sdio_dma_enable

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

例如：

```
/* enable the SDIO DMA */
```

```
sdio_dma_enable();
```

函数 **sdio_dma_disable**

函数sdio_dma_disable描述见下表：

表 3-534. 函数 sdio_dma_disable

函数名称	sdio_dma_disable
函数原形	void sdio_dma_disable(void);
功能描述	除能SDIO的DMA 请求
先决条件	-
被调用函数	-
输入参数{in}	
-	-
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* disable the SDIO DMA */
```

```
sdio_dma_disable();
```

函数 **sdio_flag_get**

函数sdio_flag_get描述见下表：

表 3-535. 函数 sdio_flag_get

函数名称	sdio_flag_get
函数原形	FlagStatus sdio_flag_get(uint32_t flag);
功能描述	获取SDIO的标志位状态
先决条件	-
被调用函数	-
输入参数{in}	
flag	SDIO标志位状态
SDIO_FLAG_CCRC ERR	命令响应已接收（CRC检测失败）
SDIO_FLAG_DTCR CERR	数据块已发送/已接收（CRC检测失败）
SDIO_FLAG_CMDT MOUT	命令响应超时

SDIO_FLAG_DTTM OUT	数据超时
SDIO_FLAG_TXUR E	发送FIFO下溢错误发生
SDIO_FLAG_RXOR E	接收FIFO上溢错误发生
SDIO_FLAG_CMDR ECV	命令响应已接收（CRC检测通过）
SDIO_FLAG_CMDS END	命令已发送（不需响应）
SDIO_FLAG_DTEN D	数据结束（数据计数器，SDIO_DATA CNT为零）
SDIO_FLAG_STBIT E	总线上起始位错误
SDIO_FLAG_DTBL KEND	数据块已发送/已接收（CRC检测通过）
SDIO_FLAG_CMDR UN	正在传输命令
SDIO_FLAG_TXRU N	正在传输数据
SDIO_FLAG_RXRU N	正在接收数据
SDIO_FLAG_TFH	发送FIFO半空：至少还有8个字可被写入到FIFO中
SDIO_FLAG_RFH	接收FIFO半满：FIFO中至少还有8个字可被读取
SDIO_FLAG_TFF	发送FIFO为满
SDIO_FLAG_RFF	接收FIFO为满
SDIO_FLAG_TFE	发送FIFO为空
SDIO_FLAG_RFE	接收FIFO为空
SDIO_FLAG_TXDT VAL	发送FIFO中的数据有效
SDIO_FLAG_RXDT VAL	接收FIFO中的数据有效
SDIO_FLAG_SDIOI NT	SD IO中断已接收
SDIO_FLAG_ATAE ND	CE-ATA命令完成信号已接收（仅用于CMD61）
输出参数{out}	
-	-
返回值	
FlagStatus	SET 或 RESET

例如：

```
/* get the flags state of SDIO */
```



```
FlagStatus flag_value;
```

```
flag_value = sdio_flag_get(SDIO_FLAG_RFH);
```

函数 **sdio_flag_clear**

函数sdio_flag_clear描述见下表:

表 3-536. 函数 sdio_flag_clear

函数名称	sdio_flag_clear
函数原形	void sdio_flag_clear(uint32_t flag);
功能描述	清除SDIO的标志位状态
先决条件	-
被调用函数	-
输入参数{in}	
flag	SDIO标志位状态
SDIO_FLAG_CCRCERR	命令响应已接收（CRC检测失败）
SDIO_FLAG_DTCCRERR	数据块已发送/已接收（CRC检测失败）
SDIO_FLAG_CMDTMOU	命令响应超时
SDIO_FLAG_DTTMOU	数据超时
SDIO_FLAG_TXUR	发送FIFO下溢错误发生
SDIO_FLAG_RXOR	接收FIFO上溢错误发生
SDIO_FLAG_CMDRECV	命令响应已接收（CRC检测通过）
SDIO_FLAG_CMDS	命令已发送（不需响应）
SDIO_FLAG_DTEND	数据结束（数据计数器，SDIO_DATA_CNT为零）
SDIO_FLAG_STBITE	总线上起始位错误
SDIO_FLAG_DTBLKEND	数据块已发送/已接收（CRC检测通过）
SDIO_FLAG_SDI	SD IO中断已接收
SDIO_FLAG_ATAEND	CE-ATA命令完成信号已接收（仅用于CMD61）
输出参数{out}	
-	-

返回值	
-	-

例如：

```
/* clear the pending flags of SDIO */
```

```
sdio_flag_clear(SDIO_FLAG_DTCRCERR);
```

函数 **sdio_interrupt_enable**

函数sdio_interrupt_enable描述见下表：

表 3-537. 函数 sdio_interrupt_enable

函数名称	sdio_interrupt_enable
函数原形	void sdio_interrupt_enable(uint32_t int_flag);
功能描述	使能SDIO中断
先决条件	-
被调用函数	-
输入参数{in}	
int_flag	SDIO中断标志位状态
SDIO_INT_CCRCE RR	命令响应CRC错误中断
SDIO_INT_DTCRC ERR	数据CRC错误中断
SDIO_INT_CMDTM OUT	命令响应超时中断
SDIO_INT_DTTMO UT	数据超时中断
SDIO_INT_TXURE	发送FIFO下溢错误中断
SDIO_INT_RXORE	接收FIFO上溢错误中断
SDIO_INT_CMDRE CV	命令响应已接收中断
SDIO_INT_CMDSE ND	命令已发送中断
SDIO_INT_DTEND	数据结束中断
SDIO_INT_STBITE	起始位错误中断
SDIO_INT_DTBLKE ND	数据块已发送/已接收中断
SDIO_INT_CMDRU N	正在传输命令中断
SDIO_INT_TXRUN	正在传输数据中断
SDIO_INT_RXRUN	正在接收数据中断
SDIO_INT_TFH	发送FIFO半满中断
SDIO_INT_RFH	接收FIFO半满中断

SDIO_INT_TFF	发送FIFO满中断
SDIO_INT_RFF	接收FIFO满中断
SDIO_INT_TFE	发送FIFO空中断
SDIO_INT_RFE	接收FIFO空中断
SDIO_INT_TXDTVAL	发送FIFO中的数据有效中断
SDIO_INT_RXDTV AL	接收FIFO中的数据有效中断
SDIO_INT_SDIOINT	SD IO中断已接收中断
SDIO_INT_ATAEND	CE-ATA 命令完成信号已接收中断
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* enable the SDIO corresponding interrupts */
```

```
sdio_interrupt_enable(SDIO_INT_CCCRERR | SDIO_INT_DTTMOUT | SDIO_INT_RXORE  
| SDIO_INT_DTEND | SDIO_INT_STBITE);
```

函数 sdio_interrupt_disable

函数sdio_interrupt_disable描述见下表：

表 3-538. 函数 sdio_interrupt_disable

函数名称	sdio_interrupt_disable
函数原形	void sdio_interrupt_disable(uint32_t int_flag);
功能描述	除能SDIO中断
先决条件	-
被调用函数	-
输入参数{in}	
int_flag	SDIO中断标志位状态
SDIO_INT_CCCRERR	命令响应CRC错误中断
SDIO_INT_DTCRCERR	数据CRC错误中断
SDIO_INT_CMDTMO OUT	命令响应超时中断
SDIO_INT_DTTMO UT	数据超时中断
SDIO_INT_TXURE	发送FIFO下溢错误中断

SDIO_INT_RXORE	接收FIFO上溢错误中断
SDIO_INT_CMDRE CV	命令响应已接收中断
SDIO_INT_CMDSE ND	命令已发送中断
SDIO_INT_DTEND	数据结束中断
SDIO_INT_STBITE	起始位错误中断
SDIO_INT_DTBKE ND	数据块已发送/已接收中断
SDIO_INT_CMDRU N	正在传输命令中断
SDIO_INT_TXRUN	正在传输数据中断
SDIO_INT_RXRUN	正在接收数据中断
SDIO_INT_TFH	发送FIFO半满中断
SDIO_INT_RFH	接收FIFO半满中断
SDIO_INT_TFF	发送FIFO满中断
SDIO_INT_RFF	接收FIFO满中断
SDIO_INT_TFE	发送FIFO空中断
SDIO_INT_RFE	接收FIFO空中断
SDIO_INT_TXDTVA L	发送FIFO中的数据有效中断
SDIO_INT_RXDTV AL	接收FIFO中的数据有效中断
SDIO_INT_SDIOIN T	SD I/O中断已接收中断
SDIO_INT_ATAEN D	CE-ATA 命令完成信号已接收中断
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* disable the SDIO interrupt */
```

```
sdio_interrupt_disable(SDIO_INT_DTCRCERR);
```

函数 sdio_interrupt_flag_get

函数sdio_interrupt_flag_get描述见下表：

表 3-539. 函数 sdio_interrupt_flag_get

函数名称	sdio_interrupt_flag_get
函数原形	FlagStatus sdio_interrupt_flag_get(uint32_t int_flag);

功能描述	获取SDIO的中断标志位状态
先决条件	-
被调用函数	-
输入参数{in}	
int_flag	SDIO中断标志位状态
SDIO_INT_CCRCE RR	命令响应CRC错误中断
SDIO_INT_DTCRC ERR	数据CRC错误中断
SDIO_INT_CMDTM OUT	命令响应超时中断
SDIO_INT_DTTMO UT	数据超时中断
SDIO_INT_TXURE	发送FIFO下溢错误中断
SDIO_INT_RXORE	接收FIFO上溢错误中断
SDIO_INT_CMDRE CV	命令响应已接收中断
SDIO_INT_CMDSE ND	命令已发送中断
SDIO_INT_DTEND	数据结束中断
SDIO_INT_STBITE	起始位错误中断
SDIO_INT_DTBKE ND	数据块已发送/已接收中断
SDIO_INT_CMDRU N	正在传输命令中断
SDIO_INT_TXRUN	正在传输数据中断
SDIO_INT_RXRUN	正在接收数据中断
SDIO_INT_TFH	发送FIFO半满中断
SDIO_INT_RFH	接收FIFO半满中断
SDIO_INT_TFF	发送FIFO满中断
SDIO_INT_RFF	接收FIFO满中断
SDIO_INT_TFE	发送FIFO空中断
SDIO_INT_RFE	接收FIFO空中断
SDIO_INT_TXDTVA L	发送FIFO中的数据有效中断
SDIO_INT_RXDTV AL	接收FIFO中的数据有效中断
SDIO_INT_SDIOIN T	SD IO中断已接收中断
SDIO_INT_ATAEN D	CE-ATA 命令完成信号已接收中断
输出参数{out}	
-	-

返回值	
FlagStatus	SET 或 RESET

例如:

```
/* get the interrupt flags state of SDIO */
```

```
FlagStatus flag_value;
```

```
flag_value = sdio_interrupt_flag_get(SDIO_INT_FLAG_DTEND);
```

函数 sdio_interrupt_flag_clear

函数sdio_interrupt_flag_clear描述见下表:

表 3-540. 函数 sdio_interrupt_flag_clear

函数名称	sdio_interrupt_flag_clear
函数原形	void sdio_interrupt_flag_clear(uint32_t int_flag);
功能描述	清除SDIO的中断标志位状态
先决条件	-
被调用函数	-
输入参数{in}	
int_flag	SDIO中断标志位状态
SDIO_INT_CCRCE RR	命令响应CRC错误中断
SDIO_INT_DTCRC ERR	数据CRC错误中断
SDIO_INT_CMDTM OUT	命令响应超时中断
SDIO_INT_DTTMO UT	数据超时中断
SDIO_INT_TXURE	发送FIFO下溢错误中断
SDIO_INT_RXORE	接收FIFO上溢错误中断
SDIO_INT_CMDRE CV	命令响应已接收中断
SDIO_INT_CMDSE ND	命令已发送中断
SDIO_INT_DTEND	数据结束中断
SDIO_INT_STBITE	起始位错误中断
SDIO_INT_DTBKE ND	数据块已发送/已接收中断
SDIO_INT_SDIOIN T	SD IO中断已接收中断
SDIO_INT_ATAEN D	CE-ATA命令完成信号已接收中断
输出参数{out}	

-	-
返回值	
-	-

例如：

```
/* clear the interrupt pending flags of SDIO */
```

```
sdio_interrupt_flag_clear(SDIO_INT_DTEND);
```

函数 **sdio_readwait_enable**

函数sdio_readwait_enable描述见下表：

表 3-541. 函数 sdio_readwait_enable

函数名称	sdio_readwait_enable
函数原形	void sdio_readwait_enable(void);
功能描述	使能读等待模式（仅限SD I/O模式）
先决条件	-
被调用函数	-
输入参数{in}	
-	-
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* enable the read wait mode(SD I/O only) */
```

```
sdio_readwait_enable();
```

函数 **sdio_readwait_disable**

函数sdio_readwait_disable描述见下表：

表 3-542. 函数 sdio_readwait_disable

函数名称	sdio_readwait_disable
函数原形	void sdio_readwait_disable(void);
功能描述	除能读等待模式（仅限SD I/O模式）
先决条件	-
被调用函数	-
输入参数{in}	
-	-
输出参数{out}	
-	-
返回值	

-	-
---	---

例如：

```
/* disable the read wait mode(SD I/O only) */
```

```
sdio_readwait_disable();
```

函数 **sdio_stop_readwait_enable**

函数sdio_stop_readwait_enable描述见下表：

表 3-543. 函数 sdio_stop_readwait_enable

函数名称	sdio_stop_readwait_enable
函数原形	void sdio_stop_readwait_enable(void);
功能描述	使能停止读等待过程的功能（仅限SD I/O模式）
先决条件	-
被调用函数	-
输入参数{in}	
-	-
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* enable the function that stop the read wait process(SD I/O only) */
```

```
sdio_stop_readwait_enable();
```

函数 **sdio_stop_readwait_disable**

函数sdio_stop_readwait_disable描述见下表：

表 3-544. 函数 sdio_stop_readwait_disable

函数名称	sdio_stop_readwait_disable
函数原形	void sdio_stop_readwait_disable(void);
功能描述	除能停止读等待过程的功能（仅限SD I/O模式）
先决条件	-
被调用函数	-
输入参数{in}	
-	-
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* disable the function that stop the read wait process(SD I/O only) */
```

```
sdio_stop_readwait_disable();
```

函数 **sdio_readwait_type_set**

函数sdio_readwait_type_set描述见下表：

表 3-545. 函数 sdio_readwait_type_set

函数名称	sdio_readwait_type_set
函数原形	void sdio_readwait_type_set(uint32_t readwait_type);
功能描述	设置读等待类型（仅限SD I/O模式）
先决条件	-
被调用函数	-
输入参数{in}	
readwait_type	SD I/O 读等待模式
SDIO_READWAITTYPE_CLK	通过停止SDIO_CLK控制读等待
SDIO_READWAITTYPE_DAT2	使用SDIO_DAT[2] 控制读等待
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* set the read wait type(SD I/O only) */
```

```
sdio_readwait_type_set(SDIO_READWAITTYPE_CLK);
```

函数 **sdio_operation_enable**

函数sdio_operation_enable描述见下表：

表 3-546. 函数 sdio_operation_enable

函数名称	sdio_operation_enable
函数原形	void sdio_operation_enable(void);
功能描述	使能SD I/O模式特定操作（仅限SD I/O模式）
先决条件	-
被调用函数	-
输入参数{in}	
-	-
输出参数{out}	
-	-

返回值	
-	-

例如：

```
/* enable the SD I/O mode specific operation(SD I/O only) */
```

```
sdio_operation_enable();
```

函数 **sdio_operation_disable**

函数sdio_operation_disable描述见下表：

表 3-547. 函数 sdio_operation_disable

函数名称	sdio_operation_disable
函数原形	void sdio_operation_disable(void);
功能描述	除能SD I/O模式特定操作（仅限SD I/O模式）
先决条件	-
被调用函数	-
输入参数{in}	
-	-
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* disable the SD I/O mode specific operation(SD I/O only) */
```

```
void sdio_operation_disable();
```

函数 **sdio_suspend_enable**

函数sdio_suspend_enable描述见下表：

表 3-548. 函数 sdio_suspend_enable

函数名称	sdio_suspend_enable
函数原形	void sdio_suspend_enable(void);
功能描述	使能SD I/O休眠模式（仅限SD I/O模式）
先决条件	-
被调用函数	-
输入参数{in}	
-	-
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* enable the SD I/O suspend operation(SD I/O only) */
```

```
sdio_suspend_enable();
```

函数 **sdio_suspend_disable**

函数sdio_suspend_disable描述见下表：

表 3-549. 函数 **sdio_suspend_disable**

函数名称	sdio_suspend_disable
函数原形	void sdio_suspend_disable(void);
功能描述	除能SD I/O休眠模式（仅限SD I/O模式）
先决条件	-
被调用函数	-
输入参数{in}	
-	-
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* disable the SD I/O suspend operation(SD I/O only) */
```

```
sdio_suspend_disable();
```

函数 **sdio_ceata_command_enable**

函数sdio_ceata_command_enable描述见下表：

表 3-550. 函数 **sdio_ceata_command_enable**

函数名称	sdio_ceata_command_enable
函数原形	void sdio_ceata_command_enable(void);
功能描述	使能CE-ATA命令(仅限CE-ATA模式)
先决条件	-
被调用函数	-
输入参数{in}	
-	-
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* enable the CE-ATA command(CE-ATA only) */
```

```
sdio_ceata_command_enable();
```

函数 **sdio_ceata_command_disable**

函数sdio_ceata_command_disable描述见下表:

表 3-551. 函数 sdio_ceata_command_disable

函数名称	sdio_ceata_command_disable
函数原形	void sdio_ceata_command_disable(void);
功能描述	除能CE-ATA命令(仅限CE-ATA模式)
先决条件	-
被调用函数	-
输入参数{in}	
-	-
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* disable the CE-ATA command(CE-ATA only) */
```

```
sdio_ceata_command_disable();
```

函数 **sdio_ceata_interrupt_enable**

函数sdio_ceata_interrupt_enable描述见下表:

表 3-552. 函数 sdio_ceata_interrupt_enable

函数名称	sdio_ceata_interrupt_enable
函数原形	void sdio_ceata_interrupt_enable(void);
功能描述	使能CE-ATA中断(仅限CE-ATA模式)
先决条件	-
被调用函数	-
输入参数{in}	
-	-
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* enable the CE-ATA interrupt(CE-ATA only) */
```

```
sdio_ceata_interrupt_enable();
```

函数 **sdio_ceata_interrupt_disable**

函数sdio_ceata_interrupt_disable描述见下表:

表 3-553. 函数 sdio_ceata_interrupt_disable

函数名称	sdio_ceata_interrupt_disable
函数原形	void sdio_ceata_interrupt_disable(void);
功能描述	除能CE-ATA中断(仅限CE-ATA模式)
先决条件	-
被调用函数	-
输入参数{in}	
-	-
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* disable the CE-ATA interrupt(CE-ATA only) */
```

```
sdio_ceata_interrupt_disable();
```

函数 **sdio_ceata_command_completion_enable**

函数sdio_ceata_command_completion_enable描述见下表:

表 3-554. 函数 sdio_ceata_command_completion_enable

函数名称	sdio_ceata_command_completion_enable
函数原形	void sdio_ceata_command_completion_enable(void);
功能描述	使能CE-ATA命令完成信号(仅限CE-ATA模式)
先决条件	-
被调用函数	-
输入参数{in}	
-	-
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* enable the CE-ATA command completion signal(CE-ATA only) */
```

```
sdio_ceata_command_completion_enable();
```

函数 sdio_ceata_command_completion_disable

函数sdio_ceata_command_completion_disable描述见下表：

表 3-555. 函数 sdio_ceata_command_completion_disable

函数名称	sdio_ceata_command_completion_disable
函数原形	void sdio_ceata_command_completion_disable(void);
功能描述	除能CE-ATA命令完成信号(仅限CE-ATA模式)
先决条件	-
被调用函数	-
输入参数{in}	
-	-
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* disable the CE-ATA command completion signal(CE-ATA only) */
sdio_ceata_command_completion_disable();
```

3.21. 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[3.21.1](#)描述了SPI/I2S的寄存器列表，章节[3.22.2](#)对SPI/I2S库函数进行说明。

3.21.1. 外设寄存器说明

SPI/I2S寄存器列表如下表所示：

表 3-556. 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

3.21.2. 外设库函数说明

SPI/I2S 库函数列表如下表所示：

表 3-557. SPI/I2S 库函数

库函数名称	库函数描述
spi_i2s_deinit	复位SPI/I2S
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SPI
spi_enable	使能SPI
spi_disable	禁能SPI
i2s_init	初始化I2S
i2s_psc_config	配置I2S预分频器
i2s_enable	使能I2S
i2s_disable	禁能I2S
spi_nss_output_enable	使能SPI NSS输出
spi_nss_output_disable	除能SPI NSS输出
spi_nss_internal_high	NSS软件模式下NSS引脚拉高
spi_nss_internal_low	NSS软件模式下NSS引脚拉低
spi_dma_enable	使能SPI DMA 功能
spi_dma_disable	除能SPI DMA 功能
spi_i2s_data_frame_format_config	配置SPI数据帧格式
spi_bidirectional_transfer_config	配置SPI的数据传输方向
spi_i2s_data_transmit	发送数据
spi_i2s_data_receive	接收数据
spi_crc_polynomial_set	设置SPI的CRC多项式值
spi_crc_polynomial_get	获取SPI的CRC多项式值
spi_crc_on	打开SPI的CRC功能
spi_crc_off	关闭SPI的CRC功能
spi_crc_next	设置SPI下一次传输数据为CRC值
spi_crc_get	SPI获取CRC值
spi_i2s_flag_get	获取SPI/I2S标志状态
spi_i2s_interrupt_enable	使能SPI/I2S中断
spi_i2s_interrupt_disable	禁能SPI/I2S中断
spi_i2s_interrupt_flag_get	获取SPI/I2S中断状态
spi_crc_error_clear	清除SPI CRC错误标志状态

结构体 spi_parameter_struct

表 3-558. 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)

成员名称	功能描述
trans_mode	传输模式 (SPI_TRANSMODE_FULLDUPLEX, SPI_TRANSMODE_RECEIVEONLY, SPI_TRANSMODE_BDRECEIVE, SPI_TRANSMODE_BDTRANSMIT)
frame_size	数据帧格式配置 (SPI_FRAME_SIZE_16BIT, SPI_FRAME_SIZE_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_phase	相位和极性配置 (SPI_CLOCK_PL_LOW_PH1EDGE, SPI_CLOCK_PL_HIGH_PH1EDGE, SPI_CLOCK_PL_LOW_PH2EDGE, SPI_CLOCK_PL_HIGH_PH2EDGE)
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 3-559. 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SPI/I2S
先决条件	-
被调用函数	rcu_periph_reset_enable / rcu_periph_reset_disable
输入参数{in}	
spi_periph	外设SPIx
SPIx	x=0,1,2
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 3-560. 函数 spi_struct_para_init

函数名称	spi_struct_para_init
------	----------------------

函数原形	void spi_struct_para_init(spi_parameter_struct* spi_struct);
功能描述	将SPI结构体参数初始化为默认值
先决条件	-
被调用函数	-
输入参数{in}	
-	-
输出参数{out}	
spi_struct	初始化结构体，结构体成员参考 表3-558. 结构体spi_parameter_struct
返回值	
-	-

例如：

```
/* initialize the parameters of SPI struct */
```

```
spi_parameter_struct spi_struct;
```

```
spi_struct_para_init(&spi_struct);
```

函数 spi_init

函数spi_init描述见下表：

表 3-561. 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SPI
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3-558. 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_FRAME_SIZE_8BIT;

spi_init_struct.clock_polarity_phase = SPI_CLOCK_POLARITY_HIGH_PHASE_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 3-562. 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SPI
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 3-563. 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SPI
先决条件	-
被调用函数	-
输入参数{in}	
spi_periph	外设SPI
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 3-564. 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I2S
先决条件	-
被调用函数	-
输入参数{in}	
spi_periph	外设SPIx
SPIx	x=1,2
输入参数{in}	
mode	I2S运行模式
I2S_MODE_SLAVE_TX	I2S从机发送模式
I2S_MODE_SLAVE_RX	I2S从机接收模式
I2S_MODE_MASTERTX	I2S主机发送模式
I2S_MODE_MASTERRX	I2S主机接收模式
输入参数{in}	
standard	I2S标准选择
I2S_STD_PHILLIPS	I2S飞利浦标准
I2S_STD_MSB	I2S MSB对齐标准
I2S_STD_LSB	I2S LSB对齐标准
I2S_STD_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
<i>I2S_CKPL_LOW</i>	I2S_CK空闲状态为低电平
<i>I2S_CKPL_HIGH</i>	I2S_CK空闲状态为高电平
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* initialize I2S1 */
```

```
i2s_init(SPI1, I2S_MODE_MASTERTX, I2S_STD_PHILLIPS, I2S_CKPL_LOW);
```

函数 i2s_psc_config

函数i2s_psc_config描述见下表:

表 3-565. 函数 i2s_psc_config

函数名称	i2s_psc_config
函数原形	void i2s_psc_config(uint32_t spi_periph, uint32_t audiosample, uint32_t frameformat, uint32_t mckout);
功能描述	配置I2S预分频器
先决条件	-
被调用函数	rcu_clock_freq_get
输入参数{in}	
spi_periph	外设SPIx
<i>SPIx</i>	x=1,2
输入参数{in}	
audiosample	I2S音频采样频率
<i>I2S_AUDIOSAMPL E_8K</i>	音频采样频率为8KHz
<i>I2S_AUDIOSAMPL E_11K</i>	音频采样频率为11KHz
<i>I2S_AUDIOSAMPL E_16K</i>	音频采样频率为16KHz
<i>I2S_AUDIOSAMPL E_22K</i>	音频采样频率为22KHz
<i>I2S_AUDIOSAMPL E_32K</i>	音频采样频率为32KHz
<i>I2S_AUDIOSAMPL E_44K</i>	音频采样频率为44KHz
<i>I2S_AUDIOSAMPL E_48K</i>	音频采样频率为48KHz
<i>I2S_AUDIOSAMPL</i>	音频采样频率为96KHz

<i>E_96K</i>	
<i>I2S_AUDIOSAMPL</i> <i>E_192K</i>	音频采样频率为192KHz
输入参数{in}	
frameformat	I2S数据长度和通道长度
<i>I2S_FRAMEFORMA</i> <i>T_DT16B_CH16B</i>	I2S数据长度为16位，通道长度为16位
<i>I2S_FRAMEFORMA</i> <i>T_DT16B_CH32B</i>	I2S数据长度为16位，通道长度为32位
<i>I2S_FRAMEFORMA</i> <i>T_DT24B_CH32B</i>	I2S数据长度为24位，通道长度为32位
<i>I2S_FRAMEFORMA</i> <i>T_DT32B_CH32B</i>	I2S数据长度为32位，通道长度为32位
输入参数{in}	
mckout	I2S_MCK输出使能
<i>I2S_MCKOUT_ENA</i> <i>BLE</i>	I2S_MCK输出使能
<i>I2S_MCKOUT_DIS</i> <i>ABLE</i>	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 3-566. 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I2S
先决条件	-
被调用函数	-
输入参数{in}	
spi_periph	外设SPIx
<i>SPIx</i>	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 3-567. 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I2S
先决条件	-
被调用函数	-
输入参数{in}	
spi_periph	外设SPIx
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 3-568. 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SPI 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 3-569. 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 3-570. 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 3-571. 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 3-572. 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SPI 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
<i>SPI_DMA_TRANSMIT</i>	SPI发送缓冲区DMA使能
<i>SPI_DMA_RECEIVE</i>	SPI接收缓冲区DMA使能
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* enable SPI0 transmit data DMAfunction */
```

```
spi_dma_enable(SPI0, SPI_DMA_TRANSMIT);
```

函数 spi_dma_disable

函数spi_dma_disable描述见下表：

表 3-573. 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SPI DMA 功能
先决条件	-
被调用函数	-
输入参数{in}	
spi_periph	外设SPIx
<i>SPIx</i>	x=0,1,2
输入参数{in}	
dma	SPI DMA 模式
<i>SPI_DMA_TRANSMIT</i>	SPI发送缓冲区DMA使能
<i>SPI_DMA_RECEIVE</i>	SPI接收缓冲区DMA使能
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* disable SPI0 transmit data DMAfunction */
```

```
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 3-574. 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SPI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_FRAME_SIZE_16BIT	SPI 16位数据帧格式
SPI_FRAME_SIZE_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_FRAME_SIZE_16BIT);
```

函数 spi_bidirectional_transfer_config

函数spi_bidirectional_transfer_config描述见下表：

表 3-575. 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SPI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
<i>SPI_BIDIRECTIONAL_TRANSMIT</i>	SPI工作在只发送模式
<i>SPI_BIDIRECTIONAL_RECEIVE</i>	SPI工作在只接收模式
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* SPI0 works in transmit-only mode */
```

```
spi_bidirectional_transfer_config(SPI0, SPI_BIDIRECTIONAL_TRANSMIT);
```

函数 spi_i2s_data_transmit

函数spi_i2s_data_transmit描述见下表：

表 3-576. 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	发送数据
先决条件	-
被调用函数	-
输入参数{in}	
spi_periph	外设SPIx
<i>SPIx</i>	x=0,1,2
输入参数{in}	
data	16位数据
输出参数{out}	
-	-
返回值	
-	-

例如：

```
uint16_t spi0_send_data = 0xAA;
```

```
/* SPI0 transmit data */
```

```
spi_i2s_data_transmit(SPI0, spi0_send_data);
```

函数 spi_i2s_data_receive

函数spi_i2s_data_receive描述见下表：

表 3-577. 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	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	
uint16_t	16位数据

例如：

```
uint16_t spi0_receive_data;

/* SPI0 receive data */

spi0_receive_data = spi_i2s_data_receive(SPI0);
```

函数 spi_crc_polynomial_set

函数spi_crc_polynomial_set描述见下表：

表 3-578. 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SPI的CRC多项式值
先决条件	-
被调用函数	-
输入参数{in}	
spi_periph	外设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 3-579. 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SPI的CRC多项式值
先决条件	-
被调用函数	-
输入参数{in}	
spi_periph	外设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：

```
uint16_t crc_data;

/* get SPI0 CRC polynomial */

crc_data = spi_crc_polynomial_get(SPI0);
```

函数 spi_crc_on

函数spi_crc_on描述见下表：

表 3-580. 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SPI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 3-581. 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SPI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 3-582. 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SPI下一次传输数据为CRC值
先决条件	-
被调用函数	-
输入参数{in}	
spi_periph	外设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 3-583. 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	SPI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:

```
uint16_t crc_value;

/* get SPI0 CRC send value */

crc_value = spi_crc_get(SPI0, SPI_CRC_TX);
```

函数 spi_i2s_flag_get

函数spi_i2s_flag_get描述见下表:

表 3-584. 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SPI/I2S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

<i>SPI_FLAG_TBE</i>	发送缓冲区空标志
<i>SPI_FLAG_RBNE</i>	接收缓冲区非空标志
<i>SPI_FLAG_TRANS</i>	通信进行中标志
<i>SPI_I2S_INT_FLAG_RXORERR</i>	接收过载错误标志
<i>SPI_FLAG_CONFERR</i>	配置错误标志
<i>SPI_FLAG_CRCERR</i>	CRC错误标志
<i>I2S_FLAG_RXORERR</i>	接收过载错误标志
<i>I2S_FLAG_TXURERR</i>	发送欠载错误标志
<i>I2S_FLAG_CH</i>	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 */
```

```
FlagStatus Flag = RESET;
```

```
Flag = spi_i2s_flag_get(SPI0, SPI_FLAG_TBE);
```

函数 spi_i2s_interrupt_enable

函数spi_i2s_interrupt_enable描述见下表:

表 3-585. 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SPI/I2S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
<i>SPI_I2S_INT_TBE</i>	发送缓冲区空中断使能
<i>SPI_I2S_INT_RBNE</i>	接收缓冲区非空中断使能
<i>SPI_I2S_INT_ERR</i>	错误中断使能
输出参数{out}	

-	-
返回值	
-	-

例如：

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

```
spi_i2s_interrupt_enable(SPI0, SPI_I2S_INT_TBE);
```

函数 spi_i2s_interrupt_disable

函数spi_i2s_interrupt_disable描述见下表：

表 3-586. 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	除能SPI/I2S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 3-587. 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SPI/I2S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 */
```

```
FlagStatus Flag_interrupt = RESET;
```

```
Flag_interrupt = spi_i2s_interrupt_flag_get(SPI0, SPI_I2S_INT_FLAG_TBE);
```

函数 spi_crc_error_clear

函数spi_crc_error_clear描述见下表：

表 3-588. 函数 spi_crc_error_clear

函数名称	spi_crc_error_clear
函数原形	void spi_crc_error_clear(uint32_t spi_periph);
功能描述	清除SPI CRC错误标志状态
先决条件	-
被调用函数	-
输入参数{in}	
spi_periph	外设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);
```

3.22. 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[3.22.1](#)描述了TIMER的寄存器列表，章节[3.22.2](#)对TIMER库函数进行说明。

3.22.1. 外设寄存器说明

TIMER寄存器列表如下表所示:

表 3-589. 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
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_DMCFG	DMA 配置寄存器
TIMER_DMATB	DMA 发送缓冲区寄存器

3.22.2. 外设库函数说明

TIMER库函数列表如下表所示：

表 3-590. 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	初始化外设TIMER结构体参数
timer_init	初始化外设TIMERx
timer_enable	使能外设TIMERx
timer_disable	除能外设TIMERx
timer_auto_reload_shadow_enable	TIMERx自动重载影子使能
timer_auto_reload_shadow_disable	TIMERx自动重载影子除能
timer_update_event_enable	TIMERx更新使能
timer_update_event_disable	TIMERx更新除能
timer_counter_alignment	设置外设TIMERx的对齐模式
timer_counter_up_direction	设置外设TIMERx向上计数
timer_counter_down_direction	设置外设TIMERx向下计数
timer_prescaler_config	配置外设TIMERx预分频器
timer_repetition_value_config	配置外设TIMERx的重复计数器
timer_autoreload_value_config	配置外设TIMERx的自动重载寄存器
timer_counter_value_config	配置外设TIMERx的计数器值
timer_counter_read	读取外设TIMERx的计数器值
timer_prescaler_read	读取外设TIMERx的预分频器值
timer_single_pulse_mode_config	配置外设TIMERx的单脉冲模式
timer_update_source_config	配置外设TIMERx的更新源
timer_dma_enable	外设TIMERx的DMA使能
timer_dma_disable	外设TIMERx的DMA除能
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	初始化外设TIMER中止功能结构体参数
timer_break_config	配置中止功能
timer_break_enable	使能TIMERx的中止功能
timer_break_disable	除能TIMERx的中止功能
timer_automatic_output_enable	自动输出使能
timer_automatic_output_disable	自动输出除能
timer_primary_output_config	所有的通道输出使能
timer_channel_control_shadow_config	通道换相控制影子配置
timer_channel_control_shadow_update_config	通道换相控制影子寄存器更新控制

库函数名称	库函数描述
timer_channel_output_struct_para_init	初始化外设TIMER通道输出结构体参数
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_parameter_init	初始化外设TIMER通道输入结构体参数
timer_input_capture_config	配置TIMERx输入捕获参数
timer_channel_input_capture_prescaler_config	配置TIMERx通道输入捕获预分频值
timer_channel_capture_value_register_read	读取通道捕获值
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配置为正交译码器模式
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禁用

库函数名称	库函数描述
e	
timer_interrupt_enable	外设TIMERx中断使能
timer_interrupt_disable	外设TIMERx中断除能
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 3-591. 结构体 timer_parameter_struct

成员名称	功能描述
prescaler	预分频值（0~65535）
alignedmode	对齐模式 (TIMER_COUNTER_EDGE, TIMER_COUNTER_CENTER_DOWN, TIMER_COUNTER_CENTER_UP, TIMER_COUNTER_CENTER_BOTH)
counterdirection	计数方向（TIMER_COUNTER_UP, TIMER_COUNTER_DOWN）
period	周期（0~65535）
clockdivision	时钟分频因子 (TIMER_CKDIV_DIV1, TIMER_CKDIV_DIV2, TIMER_CKDIV_DIV4)
repetitioncounter	重复计数器值（0~255）

结构体 timer_break_parameter_struct

表 3-592. 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)
idloffstate	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 3-593. 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 3-594. 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)
icselelection	通道输入模式选择 (TIMER_IC_SELECTION_DIRECTTI, TIMER_IC_SELECTION_INDIRECTTI, TIMER_IC_SELECTION_ITS)
icprescaler	通道输入捕获预分频 (TIMER_IC_PSC_DIV1, TIMER_IC_PSC_DIV2, TIMER_IC_PSC_DIV4, TIMER_IC_PSC_DIV8)
icfilter	通道输入捕获滤波 (0~15)

函数 timer_deinit

函数 timer_deinit 描述见下表:

表 3-595. 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 3-596. 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	初始化外设TIMER结构体参数
先决条件	-
被调用函数	-
输入参数{in}	
initpara	初始化结构体,结构体成员参考 表3-591. 结构体timer_parameter_struct
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* initialize the TIMER structure */
```

```
timer_parameter_struct initpara;
```

```
initpara->prescaler = 0U;
```

```
initpara->alignedmode = TIMER_COUNTER_EDGE;
```

```
initpara->counterdirection = TIMER_COUNTER_UP;
```

```
initpara->period = 65535U;
```

```
initpara->clockdivision = TIMER_CKDIV_DIV1;
```

```
initpara->repetitioncounter = 0U;
```

```
timer_struct_para_init(&initpara);
```

函数 timer_init

函数timer_init描述见下表：

表 3-597. 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	初始化结构体,结构体成员参考 表3-591. 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 3-598. 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 3-599. 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	除能外设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 3-600. 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 3-601. 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自动重载影子除能
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 3-602. 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 3-603. 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 3-604. 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..1)	TIMER外设选择

3)	
输入参数{in}	
aligned	对齐模式
<i>TIMER_COUNTER_EDGE</i>	无中央对齐计数模式(边沿对齐模式)，DIR位指定了计数方向
<i>TIMER_COUNTER_CENTER_DOWN</i>	中央对齐向下计数置1模式。计数器在中央计数模式计数，通道被配置在输出模式（TIMERx_CHCTL0寄存器中CHxMS=00），只有在向下计数时，通道的比较中断标志置1
<i>TIMER_COUNTER_CENTER_UP</i>	中央对齐向上计数置1模式。计数器在中央计数模式计数，通道被配置在输出模式（TIMERx_CHCTL0寄存器中CHxMS=00），只有在向上计数时，通道的比较中断标志置1
<i>TIMER_COUNTER_CENTER_BOTH</i>	中央对齐上下计数置1模式。计数器在中央计数模式计数，通道被配置在输出模式（TIMERx_CHCTL0寄存器中CHxMS=00），在向上和向下计数时，通道的比较中断标志都会置1
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 3-605. 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
<i>TIMERx(x=0..4,7..13)</i>	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 3-606. 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 counterdown direction */
```

```
timer_counter_down_direction(TIMER0);
```

函数 timer_prescaler_config

函数timer_prescaler_config描述见下表：

表 3-607. 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

<i>TIMER_PSC_RELO AD_NOW</i>	预分频值立即加载
<i>TIMER_PSC_RELO AD_UPDATE</i>	预分频值在下次更新事件加载
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 3-608. 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, uint8_t repetition);
功能描述	配置外设TIMERx的重复计数器
先决条件	-
被调用函数	-
输入参数{in}	
timer_periph	TIMER外设
<i>TIMERx(x=0,7)</i>	TIMER外设选择
输入参数{in}	
repetition	重复计数器值,取值范围0~255
输出参数{out}	
-	-
返回值	
-	-

例如:

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

```
timer_repetition_value_config(TIMER0, 98);
```

函数 timer_autoreload_value_config

函数timer_autoreload_value_config描述见下表:

表 3-609. 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, uint32_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
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 3-610. 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, uint32_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
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 3-611. 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	
uint16_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 3-612. 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 3-613. 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..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 3-614. 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, uint8_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
<i>TIMER_UPDATE_S RC_GLOBAL</i>	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
<i>TIMER_UPDATE_S RC_REGULAR</i>	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 3-615. 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
<i>TIMERx</i>	参考具体参数
输入参数{in}	
dma	DMA源
<i>TIMER_DMA_UPD</i>	更新DMA请求，TIMERx(x=0..7)
<i>TIMER_DMA_CH0 D</i>	通道0比较/捕获 DMA请求，TIMERx(x=0..4,7)
<i>TIMER_DMA_CH1 D</i>	通道1比较/捕获 DMA请求，TIMERx(x=0..4,7)
<i>TIMER_DMA_CH2 D</i>	通道2比较/捕获 DMA请求，TIMERx(x=0..4,7)
<i>TIMER_DMA_CH3 D</i>	通道3比较/捕获 DMA请求，TIMERx(x=0..4,7)
<i>TIMER_DMA_CMT D</i>	换相DMA更新请求，TIMERx(x=0,7)

<i>TIMER_DMA_TRG</i> <i>D</i>	触发DMA请求使能, <i>TIMERx</i> (<i>x</i> =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 3-616. 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 <i>TIMERx</i> 的DMA除能
先决条件	-
被调用函数	-
输入参数{in}	
timer_periph	<i>TIMER</i> 外设
<i>TIMERx</i>	参考具体参数
输入参数{in}	
dma	DMA源
<i>TIMER_DMA_UPD</i>	更新DMA请求, <i>TIMERx</i> (<i>x</i> =0..7)
<i>TIMER_DMA_CH0</i> <i>D</i>	通道0比较/捕获 DMA请求, <i>TIMERx</i> (<i>x</i> =0..4,7)
<i>TIMER_DMA_CH1</i> <i>D</i>	通道1比较/捕获 DMA请求, <i>TIMERx</i> (<i>x</i> =0..4,7)
<i>TIMER_DMA_CH2</i> <i>D</i>	通道2比较/捕获 DMA请求, <i>TIMERx</i> (<i>x</i> =0..4,7)
<i>TIMER_DMA_CH3</i> <i>D</i>	通道3比较/捕获 DMA请求, <i>TIMERx</i> (<i>x</i> =0..4,7)
<i>TIMER_DMA_CMT</i> <i>D</i>	换相DMA更新请求, <i>TIMERx</i> (<i>x</i> =0,7)
<i>TIMER_DMA_TRG</i> <i>D</i>	触发DMA请求使能, <i>TIMERx</i> (<i>x</i> =0..4,7)
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* disable the TIMER0 update DMA */
timer_dma_disable(TIMER0, TIMER_DMA_UPD);
```

函数 timer_channel_dma_request_source_select

函数timer_channel_dma_request_source_select描述见下表:

表 3-617. 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, uint8_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x的DMA请求
TIMER_DMAREQUEST_UPDATEEVENT	当更新事件发生, 发送通道x的DMA请求
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* TIMER0 channel DMA request of channel y 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 3-618. 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_DMATA_CTL1	DMA传输起始地址: TIMER_DMACFG_DMATA_CTL1, TIMERx(x=0..4,7)
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)

<i>DMATA_CH3CV</i>	
<i>TIMER_DMACFG_DMATA_CCHP</i>	DMA 传输起始地址: <i>TIMER_DMA_CFG_DMATA_CCHP</i> , <i>TIMERx</i> (<i>x</i> =0..7)
<i>TIMER_DMACFG_DMATA_DMACFG</i>	DMA 传输起始地址: <i>TIMER_DMA_CFG_DMATA_DMACFG</i> , <i>TIMERx</i> (<i>x</i> =0..4,7)
<i>TIMER_DMACFG_DMATA_DMATB</i>	DMA 传输起始地址: <i>TIMER_DMA_CFG_DMATA_DMATB</i> , <i>TIMERx</i> (<i>x</i> =0..4,7)
输入参数{in}	
dma_lenth	DMA 传输长度, <i>TIMER_DMA_CFG_DMA_TC_xTRANSFER</i> (<i>x</i> =1..18)
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 3-619. 函数 *timer_event_software_generate*

函数名称	<i>timer_event_software_generate</i>
函数原型	<code>void timer_event_software_generate(uint32_t timer_periph, uint16_t event);</code>
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
<i>TIMERx</i>	参考具体参数
输入参数{in}	
event	事件源
<i>TIMER_EVENT_SRC_UPG</i>	更新事件产生, <i>TIMERx</i> (<i>x</i> =0..13)
<i>TIMER_EVENT_SRC_CH0G</i>	通道0捕获或比较事件发生, <i>TIMERx</i> (<i>x</i> =0..4,7..13)
<i>TIMER_EVENT_SRC_CH1G</i>	通道1捕获或比较事件发生, <i>TIMERx</i> (<i>x</i> =0..4,7,8,11)
<i>TIMER_EVENT_SRC_CH2G</i>	通道2捕获或比较事件发生, <i>TIMERx</i> (<i>x</i> =0..4,7)
<i>TIMER_EVENT_SRC</i>	通道3捕获或比较事件发生, <i>TIMERx</i> (<i>x</i> =0..4,7)

C_CH3G	
TIMER_EVENT_SR C_CMTG	通道换相更新事件发生, TIMERx(x=0,7)
TIMER_EVENT_SR C_TRGG	触发事件产生, TIMERx(x=0..4,7,8,11)
TIMER_EVENT_SR C_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 3-620. 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	配置中止功能
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3-592. 结构体timer_break_parameter_struct
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* initialize the TIMER break parameter structure */
```

```
timer_break_parameter_struct breakpara;
```

```
breakpara->runoffstate = TIMER_ROS_STATE_DISABLE;
```

```
breakpara->idelloffstate = TIMER_IOS_STATE_DISABLE;
```

```
breakpara->deadtime = 0U;
```

```
breakpara->breakpolarity = TIMER_BREAK_POLARITY_LOW;
```



```
breakpara->outputautostate = TIMER_OUTAUTO_DISABLE;

breakpara->protectmode = TIMER_CCHP_PROT_OFF;

breakpara->breakstate = TIMER_BREAK_DISABLE;

timer_break_struct_para_init(&breakpara);
```

函数 timer_break_config

函数timer_break_config描述见下表:

表 3-621. 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3-592. 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 3-622. 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 3-623. 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	除能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 t描述见下表：

表 3-624. 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 t描述见下表：

表 3-625. 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	自动输出除能
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 3-626. 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 new value);
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	除能
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 3-627. 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 new value);
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	除能
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 3-628. 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_CCUC	CMTG位被置1时更新影子寄存器
TIMER_UPDATECTL_CCUTRI	当CMTG位被置1或检测到TRIG1上升沿时，影子寄存器更新
输出参数{out}	
-	-
返回值	
-	-

例如:

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

函数 timer_channel_output_struct_para_init

函数timer_channel_output_struct_para_init描述见下表:

表 3-629. 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	初始化外设TIMER通道输出结构体参数
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3-593. 结构体timer_oc_parameter_struct
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* initialize the TIMER channel output parameter structure */
timer_oc_parameter_struct ocpara;
ocpara->outputstate = TIMER_CCX_DISABLE;
ocpara->outputnstate = TIMER_CCXN_DISABLE;
ocpara->ocpolarity = TIMER_OC_POLARITY_HIGH;
ocpara->ocnpolarity = TIMER_OCN_POLARITY_HIGH;
ocpara->ocidlestate = TIMER_OC_IDLE_STATE_LOW;
ocpara->ocnidlestate = TIMER_OCN_IDLE_STATE_LOW;
timer_channel_output_struct_para_init(&ocpara);
```

函数 timer_channel_output_config

函数timer_channel_output_config描述见下表:

表 3-630. 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
<i>TIMER_CH_0</i>	通道0, <i>TIMERx</i> (<i>x</i> =0..4,7..13)
<i>TIMER_CH_1</i>	通道1, <i>TIMERx</i> (<i>x</i> =0..4,7,8,11)
<i>TIMER_CH_2</i>	通道2, <i>TIMERx</i> (<i>x</i> =0..4,7)
<i>TIMER_CH_3</i>	通道3, <i>TIMERx</i> (<i>x</i> =0..4,7)
输入参数{in}	
ocpara	输出通道结构体, 详见 表3-593. 结构体timer_oc_parameter_struct
输出参数{out}	
-	-
返回值	
-	-

例如:

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

t 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 3-631. 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
<i>TIMERx</i>	参考具体参数
输入参数{in}	
channel	待配置通道
<i>TIMER_CH_0</i>	通道0, <i>TIMERx</i> (<i>x</i> =0..4,7..13)

<code>TIMER_CH_1</code>	通道1, <code>TIMERx(x=0..4,7,8,11)</code>
<code>TIMER_CH_2</code>	通道2, <code>TIMERx(x=0..4,7)</code>
<code>TIMER_CH_3</code>	通道3, <code>TIMERx(x=0..4,7)</code>
输入参数{in}	
<code>ocmode</code>	通道输出比较模式
<code>TIMER_OC_MODE_TIMING</code>	冻结模式
<code>TIMER_OC_MODE_ACTIVE</code>	匹配时设置为高
<code>TIMER_OC_MODE_INACTIVE</code>	匹配时设置为低
<code>TIMER_OC_MODE_TOGGLE</code>	匹配时翻转
<code>TIMER_OC_MODE_LOW</code>	强制为低
<code>TIMER_OC_MODE_HIGH</code>	强制为高
<code>TIMER_OC_MODE_PWM0</code>	PWM模式0
<code>TIMER_OC_MODE_PWM1</code>	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 3-632. 函数 `timer_channel_output_pulse_value_config`

函数名称	<code>timer_channel_output_pulse_value_config</code>
函数原型	<code>void timer_channel_output_pulse_value_config(uint32_t timer_periph, uint16_t channel, uint16_t pulse);</code>
功能描述	配置外设 <code>TIMERx</code> 的通道输出比较值
先决条件	-
被调用函数	-
输入参数{in}	
<code>timer_periph</code>	<code>TIMER</code> 外设

<i>TIMERx</i>	参考具体参数
输入参数{in}	
channel	待配置通道
<i>TIMER_CH_0</i>	通道0, <i>TIMERx</i> (<i>x</i> =0..4, 7..13)
<i>TIMER_CH_1</i>	通道1, <i>TIMERx</i> (<i>x</i> =0..4, 7, 8, 11)
<i>TIMER_CH_2</i>	通道2, <i>TIMERx</i> (<i>x</i> =0..4, 7)
<i>TIMER_CH_3</i>	通道3, <i>TIMERx</i> (<i>x</i> =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 3-633. 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 <i>TIMERx</i> 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	<i>TIMER</i> 外设
<i>TIMERx</i>	参考具体参数
输入参数{in}	
channel	待配置通道
<i>TIMER_CH_0</i>	通道0, <i>TIMERx</i> (<i>x</i> =0..4, 7..13)
<i>TIMER_CH_1</i>	通道1, <i>TIMERx</i> (<i>x</i> =0..4, 7, 8, 11)
<i>TIMER_CH_2</i>	通道2, <i>TIMERx</i> (<i>x</i> =0..4, 7)
<i>TIMER_CH_3</i>	通道3, <i>TIMERx</i> (<i>x</i> =0..4, 7)
输入参数{in}	
ocshadow	输出比较影子寄存器功能状态
<i>TIMER_OC_SHADOW_ENABLE</i>	使能输出比较影子寄存器
<i>TIMER_OC_SHADOW_DISABLE</i>	禁能输出比较影子寄存器

OW_DISABLE	
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 3-634. 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 3-635. 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(x=0..4,7)	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
TIMER_CH_3	通道3
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 3-636. 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 3-637. 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
<i>TIMERx</i>	参考具体参数
输入参数{in}	
channel	待配置通道
<i>TIMER_CH_0</i>	通道0, TIMERx(x=0..4, 7..13)
<i>TIMER_CH_1</i>	通道1, TIMERx(x=0..4, 7, 8, 11)
<i>TIMER_CH_2</i>	通道2, TIMERx(x=0..4, 7)
输入参数{in}	
ocpolarity	互补通道输出极性
<i>TIMER_OCN_POLARITY_HIGH</i>	互补通道输出极性高电平有效
<i>TIMER_OCN_POLARITY_LOW</i>	互补通道输出极性低电平有效
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* configure TIMER0 channel 0 complementary output polarity */
```

```
timer_channel_complementary_output_polarity_config(TIMER0, TIMER_CH_0,  
TIMER_OCN_POLARITY_HIGH);
```

函数 timer_channel_output_state_config

函数timer_channel_output_state_config描述见下表:

表 3-638. 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
<i>TIMERx</i>	参考具体参数
输入参数{in}	
channel	待配置通道
<i>TIMER_CH_0</i>	通道0, TIMERx(x=0..4, 7..13)
<i>TIMER_CH_1</i>	通道1, TIMERx(x=0..4, 7, 8, 11)
<i>TIMER_CH_2</i>	通道2, TIMERx(x=0..4, 7)

<i>TIMER_CH_3</i>	通道3, <i>TIMERx</i> (<i>x</i> =0..4,7)
输入参数{in}	
state	通道状态
<i>TIMER_CCX_ENABLE</i>	通道使能
<i>TIMER_CCX_DISABLE</i>	通道禁能
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* configure TIMER0 channel 0 enable state */
```

```
timer_channel_output_state_config(TIMER0, TIMER_CH_0, TIMER_CCX_ENABLE);
```

函数 timer_channel_complementary_output_state_config

函数timer_channel_complementary_output_state_config描述见下表:

表 3-639. 函数 timer_channel_complementary_output_state_config

函数名称	timer_channel_complementary_output_state_config
函数原型	void timer_channel_complementary_output_state_config(uint32_t timer_periph, uint16_t channel, uint16_t ocnstate);
功能描述	配置互补通道输出状态
先决条件	-
被调用函数	-
输入参数{in}	
timer_periph	TIMER外设
<i>TIMERx</i> (<i>x</i> =0,7)	TIMER外设选择
输入参数{in}	
channel	待配置通道
<i>TIMER_CH_0</i>	通道0
<i>TIMER_CH_1</i>	通道1
<i>TIMER_CH_2</i>	通道2
输入参数{in}	
state	互补通道状态
<i>TIMER_CCXN_ENABLE</i>	互补通道使能
<i>TIMER_CCXN_DISABLE</i>	互补通道禁能
输出参数{out}	
-	-

返回值	
-	-

例如：

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

函数 timer_channel_input_struct_para_init

函数timer_channel_input_struct_para_init描述见下表：

表 3-640. 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	初始化外设TIMER通道输入结构体参数
先决条件	-
被调用函数	timer_channel_input_capture_prescaler_config
输入参数{in}	
icpara	输入捕获结构体，详见 表3-594. 结构体timer_ic_parameter_struct
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* initialize the TIMER channel input parameter structure */
timer_ic_parameter_struct icpara;
icpara->icpolarity = TIMER_IC_POLARITY_RISING;
icpara->icselection = TIMER_IC_SELECTION_DIRECTTI;
icpara->icprescaler = TIMER_IC_PSC_DIV1;
icpara->icfilter = 0U;
timer_channel_input_struct_para_init(&icpara);
```

函数 timer_input_capture_config

函数timer_input_capture_config描述见下表：

表 3-641. 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3-594. 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 3-642. 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
<i>TIMERx</i>	参考具体参数
输入参数{in}	
channel	待配置通道
<i>TIMER_CH_0</i>	通道0, <i>TIMERx</i> (x=0..4,7..13)
<i>TIMER_CH_1</i>	通道1, <i>TIMERx</i> (x=0..4,7,8,11)
<i>TIMER_CH_2</i>	通道2, <i>TIMERx</i> (x=0..4,7)
<i>TIMER_CH_3</i>	通道3, <i>TIMERx</i> (x=0..4,7)
输入参数{in}	
prescaler	通道输入捕获预分频值
<i>TIMER_IC_PSC_DIV1</i>	不分频
<i>TIMER_IC_PSC_DIV2</i>	2分频
<i>TIMER_IC_PSC_DIV4</i>	4分频
<i>TIMER_IC_PSC_DIV8</i>	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 3-643. 函数 timer_channel_capture_value_register_read

函数名称	timer_channel_capture_value_register_read
函数原型	uint32_t timer_channel_capture_value_register_read(uint32_t timer_periph, uint16_t channel);
功能描述	读取通道捕获值
先决条件	-
被调用函数	-
输入参数{in}	
timer_periph	TIMER外设
<i>TIMERx</i>	参考具体参数
输入参数{in}	

channel	待配置通道
<i>TIMER_CH_0</i>	通道0, <i>TIMERx</i> (<i>x</i> =0..4,7..13)
<i>TIMER_CH_1</i>	通道1, <i>TIMERx</i> (<i>x</i> =0..4,7,8,11)
<i>TIMER_CH_2</i>	通道2, <i>TIMERx</i> (<i>x</i> =0..4,7)
<i>TIMER_CH_3</i>	通道3, <i>TIMERx</i> (<i>x</i> =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 3-644. 函数 timer_input_pwm_capture_config

函数名称	timer_input_pwm_capture_config
函数原型	void timer_input_pwm_capture_config(uint32_t timer_periph, uint16_t channel, timer_ic_parameter_struct* icpwm);
功能描述	配置TIMERx捕获PWM输入参数
先决条件	-
被调用函数	timer_channel_input_capture_prescaler_config
输入参数{in}	
timer_periph	TIMER外设
<i>TIMERx</i> (<i>x</i> =0..4,7,8,11)	TIMER外设选择
输入参数{in}	
channel	待配置通道
<i>TIMER_CH_0</i>	通道0
<i>TIMER_CH_1</i>	通道1
输入参数{in}	
icpwm	输入捕获结构体, 详见 表3-594. 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 3-645. 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	使能HALL接口
TIMER_HALLINTERFACE_DISABLE	禁能HALL接口
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 3-646. 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	参考具体参数
输入参数{in}	
intrigger	待选择的触发源
TIMER_SMCFG_T RGSEL_ITI0	内部触发输入0(ITI0), TIMERx(x=0..4,7,8,11)
TIMER_SMCFG_T RGSEL_ITI1	内部触发输入1(ITI1), TIMERx(x=0..4,7,8,11)
TIMER_SMCFG_T RGSEL_ITI2	内部触发输入2(ITI2), TIMERx(x=0..4,7,8,11)
TIMER_SMCFG_T RGSEL_ITI3	内部触发输入3(ITI3), TIMERx(x=0..4,7,8,11)
TIMER_SMCFG_T RGSEL_CIOF_ED	TIO边沿检测 , TIMERx(x=0..4,7,8,11)
TIMER_SMCFG_T RGSEL_CIOFE0	滤波后的通道0输入 (CIOFE0) , TIMERx(x=0..4,7,8,11)
TIMER_SMCFG_T RGSEL_C1FE1	滤波后的通道1输入(C1FE1) , TIMERx(x=0..4,7,8,11)
TIMER_SMCFG_T RGSEL_ETIFP	滤波后的外部触发输入(ETIFP)
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 3-647. 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 outtrigger);
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 3-648. 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 3-649. 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 3-650. 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, uint8_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_TRIP_SC_OFF	不分频
TIMER_EXT_TRIP_SC_DIV2	2分频
TIMER_EXT_TRIP_P	4分频

SC_DIV4	
TIMER_EXT_TRI_P SC_DIV8	8分频
输入参数{in}	
expolarity	外部触发输入极性
TIMER_ETP_FALLI NG	低电平或者下降沿有效
TIMER_ETP_RISIN G	高电平或者上升沿有效
输入参数{in}	
extfilter	滤波 (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 3-651. 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配置为正交译码器模式
先决条件	-
被调用函数	-
输入参数{in}	
timer_periph	TIMER外设
TIMERx(x=0..4,7,8, 11)	TIMER外设选择
输入参数{in}	
decomode	正交译码器模式
TIMER_QUAD_DE CODER_MODE0	根据CI0FE0的电平, 计数器在CI1FE1的边沿向上/下计数
TIMER_QUAD_DE CODER_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 3-652. 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 3-653. 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 3-654. 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, uint8_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_CIOF_ED	CIO的边沿标志(CIOF_ED)
TIMER_SMCFG_TRGSEL_CIOFE0	滤波后的通道0输入(CIOFE0)
TIMER_SMCFG_TRGSEL_C1FE1	滤波后的通道1输入(C1FE1)
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 CIOFE0 as external clock input */
```

```
timer_external_trigger_as_external_clock_config(TIMER0,
TIMER_SMCFG_TRGSEL_CIOFE0, TIMER_IC_POLARITY_RISING, 0);
```

函数 timer_external_clock_mode0_config

函数timer_external_clock_mode0_config描述见下表:

表 3-655. 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, uint8_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 3-656. 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, uint8_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_P SC_OFF	不分频
TIMER_EXT_TRI_P SC_DIV2	2分频
TIMER_EXT_TRI_P SC_DIV4	4分频
TIMER_EXT_TRI_P SC_DIV8	8分频
输入参数{in}	
expolarity	ETI触发源极性
TIMER_ETP_FALLI NG	下降沿或者低电平有效
TIMER_ETP_RISIN G	上升沿或者高电平有效
输入参数{in}	
extfilter	ETI触发源滤波参数（0~15）
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* configure TIMER0 the external clock mode1 */
```

```
timer_external_clock_mode1_config(TIMER0, TIMER_EXT_TRI_PSC_DIV2,  
TIMER_ETP_FALLING, 0);
```

函数 timer_external_clock_mode1_disable

函数timer_external_clock_mode1_disable描述见下表：

表 3-657. 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除能
先决条件	-
被调用函数	-

输入参数{in}	
timer_periph	TIMER外设
<i>TIMERx(x=0..4,7)</i>	TIMER外设选择
输出参数{out}	
-	-
返回值	
-	-

例如:

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

函数 timer_interrupt_enable

函数timer_interrupt_enable描述见下表:

表 3-658. 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
<i>TIMERx</i>	参考具体参数
输入参数{in}	
interrupt	中断源
<i>TIMER_INT_UP</i>	更新中断, TIMERx(x=0..13)
<i>TIMER_INT_CH0</i>	通道0比较/捕获中断, TIMERx(x=0..4,7..13)
<i>TIMER_INT_CH1</i>	通道1比较/捕获中断, TIMERx(x=0..4,7,8,11)
<i>TIMER_INT_CH2</i>	通道2比较/捕获中断, TIMERx(x=0..4,7)
<i>TIMER_INT_CH3</i>	通道3比较/捕获中断, TIMERx(x=0..4,7)
<i>TIMER_INT_CMT</i>	换相更新中断, TIMERx(x=0,7)
<i>TIMER_INT_TRG</i>	触发中断, TIMERx(x=0..4,7,8,11)
<i>TIMER_INT_BRK</i>	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 3-659. 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中断除能
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, TIME R _x (x=0..4,7..13)
TIMER_INT_CH1	通道1比较/捕获中断, TIME R _x (x=0..4,7,8,11)
TIMER_INT_CH2	通道2比较/捕获中断, TIME R _x (x=0..4,7)
TIMER_INT_CH3	通道3比较/捕获中断, TIME R _x (x=0..4,7)
TIMER_INT_CMT	换相更新中断, TIME R _x (x=0,7)
TIMER_INT_TRG	触发中断, TIME R _x (x=0..4,7,8,11)
TIMER_INT_BRK	中止中断, TIME R _x (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 3-660. 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

<i>TIMERx</i>	参考具体参数
输入参数{in}	
interrupt	中断源
<i>TIMER_INT_FLAG_UP</i>	更新中断, <i>TIMERx</i> (<i>x</i> =0..13)
<i>TIMER_INT_FLAG_CH0</i>	通道0比较/捕获中断, <i>TIMERx</i> (<i>x</i> =0..4,7..13)
<i>TIMER_INT_FLAG_CH1</i>	通道1比较/捕获中断, <i>TIMERx</i> (<i>x</i> =0..4,7,8,11)
<i>TIMER_INT_FLAG_CH2</i>	通道2比较/捕获中断, <i>TIMERx</i> (<i>x</i> =0..4,7)
<i>TIMER_INT_FLAG_CH3</i>	通道3比较/捕获中断, <i>TIMERx</i> (<i>x</i> =0..4,7)
<i>TIMER_INT_FLAG_CMT</i>	换相更新中断, <i>TIMERx</i> (<i>x</i> =0,7)
<i>TIMER_INT_FLAG_TRG</i>	触发中断, <i>TIMERx</i> (<i>x</i> =0..4,7,8,11)
<i>TIMER_INT_FLAG_BRK</i>	中止中断, <i>TIMERx</i> (<i>x</i> =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 3-661. 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 <i>TIMERx</i> 的中断标志
先决条件	-
被调用函数	-
输入参数{in}	
timer_periph	<i>TIMER</i> 外设
<i>TIMERx</i>	参考具体参数
输入参数{in}	

interrupt	中断源
<i>TIMER_INT_FLAG_UP</i>	更新中断, <i>TIMERx</i> (<i>x</i> =0..13)
<i>TIMER_INT_FLAG_CH0</i>	通道0比较/捕获中断, <i>TIMERx</i> (<i>x</i> =0..4,7..13)
<i>TIMER_INT_FLAG_CH1</i>	通道1比较/捕获中断, <i>TIMERx</i> (<i>x</i> =0..4,7,8,11)
<i>TIMER_INT_FLAG_CH2</i>	通道2比较/捕获中断, <i>TIMERx</i> (<i>x</i> =0..4,7)
<i>TIMER_INT_FLAG_CH3</i>	通道3比较/捕获中断, <i>TIMERx</i> (<i>x</i> =0..4,7)
<i>TIMER_INT_FLAG_CMT</i>	换相更新中断, <i>TIMERx</i> (<i>x</i> =0,7)
<i>TIMER_INT_FLAG_TRG</i>	触发中断, <i>TIMERx</i> (<i>x</i> =0..4,7,8,11)
<i>TIMER_INT_FLAG_BRK</i>	中止中断, <i>TIMERx</i> (<i>x</i> =0,7)
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 3-662. 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 <i>TIMERx</i> 的状态标志
先决条件	-
被调用函数	-
输入参数{in}	
timer_periph	<i>TIMER</i> 外设
<i>TIMERx</i>	参考具体参数
输入参数{in}	
flag	状态标志
<i>TIMER_FLAG_UP</i>	更新标志, <i>TIMERx</i> (<i>x</i> =0..13)
<i>TIMER_FLAG_CH0</i>	通道0比较/捕获标志, <i>TIMERx</i> (<i>x</i> =0..4,7..13)

<code>TIMER_FLAG_CH1</code>	通道1比较/捕获标志, <code>TIMERx(x=0..4,7,8,11)</code>
<code>TIMER_FLAG_CH2</code>	通道2比较/捕获标志, <code>TIMERx(x=0..4,7)</code>
<code>TIMER_FLAG_CH3</code>	通道3比较/捕获标志, <code>TIMERx(x=0..4,7)</code>
<code>TIMER_FLAG_CMT</code>	通道换相更新标志, <code>TIMERx(x=0,7)</code>
<code>TIMER_FLAG_TRG</code>	触发标志, <code>TIMERx(x=0,7,8,11)</code>
<code>TIMER_FLAG_BRK</code>	中止标志位, <code>TIMERx(x=0,7)</code>
<code>TIMER_FLAG_CH0</code> O	通道0捕获溢出标志, <code>TIMERx(x=0..4,7..11)</code>
<code>TIMER_FLAG_CH1</code> O	通道1捕获溢出标志, <code>TIMERx(x=0..4,7,8,11)</code>
<code>TIMER_FLAG_CH2</code> O	通道2捕获溢出标志, <code>TIMERx(x=0..4,7)</code>
<code>TIMER_FLAG_CH3</code> O	通道3捕获溢出标志, <code>TIMERx(x=0..4,7)</code>
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 3-663. 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, <code>TIMERx(x=0..13)</code>
TIMER_FLAG_CH0	通道0比较/捕获标志, <code>TIMERx(x=0..4,7..13)</code>
TIMER_FLAG_CH1	通道1比较/捕获标志, <code>TIMERx(x=0..4,7,8,11)</code>
TIMER_FLAG_CH2	通道2比较/捕获标志, <code>TIMERx(x=0..4,7)</code>

<code>TIMER_FLAG_CH3</code>	通道3比较/捕获标志, <code>TIMERx(x=0..4,7)</code>
<code>TIMER_FLAG_CMT</code>	通道换相更新标志, <code>TIMERx(x=0,7)</code>
<code>TIMER_FLAG_TRG</code>	触发标志, <code>TIMERx(x=0,7,8,11)</code>
<code>TIMER_FLAG_BRK</code>	中止标志位, <code>TIMERx(x=0,7)</code>
<code>TIMER_FLAG_CH0</code> 0	通道0捕获溢出标志, <code>TIMERx(x=0..4,7..11)</code>
<code>TIMER_FLAG_CH1</code> 0	通道1捕获溢出标志, <code>TIMERx(x=0..4,7,8,11)</code>
<code>TIMER_FLAG_CH2</code> 0	通道2捕获溢出标志, <code>TIMERx(x=0..4,7)</code>
<code>TIMER_FLAG_CH3</code> 0	通道3捕获溢出标志, <code>TIMERx(x=0..4,7)</code>
输出参数{out}	
-	-
返回值	
-	-

例如:

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

3.23. USART

通用同步异步收发器(USART)提供了一个灵活方便的串行数据交换接口, 章节[3.23.1](#)描述了USART的寄存器列表, 章节[3.23.2](#)对USART库函数进行说明。

3.23.1. 外设寄存器说明

USART寄存器列表如下表所示:

表 3-664. USART 寄存器

寄存器名称	寄存器描述
USART_STAT	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

3.23.2. 外设库函数说明

USART库函数列表如下表所示:

表 3-665. USART 库函数

库函数名称	库函数描述
usart_deinit	复位外设USART
usart_baudrate_set	配置USART波特率
usart_parity_config	配置USART奇偶校验
usart_word_length_set	配置USART字长
usart_stop_bit_set	配置USART停止位
usart_enable	使能USART
usart_disable	除能USART
usart_transmit_config	USART发送配置
usart_receive_config	USART接收配置
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	除能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	除能USART LIN模式
usart_lin_break_dection_length_config	配置USART LIN模式中中断帧长度
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	除能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同步通讯模式下除能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	除能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智能卡模式下除能NACK
usart_irda_mode_enable	使能USART串行红外编解码功能模块
usart_irda_mode_disable	除能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_low power_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

库函数名称	库函数描述
usart_dma_transmit_config	配置USART DMA发送功能
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	除能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

函数 usart_deinit

函数usart_deinit描述见下表:

表 3-666. 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USART
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 3-667. 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

<i>USARTx</i>	x=0,1,2
<i>UARTx</i>	x=3,4
输入参数{in}	
baudval	波特率值
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* configuration 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 3-668. 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
<i>USARTx</i>	x=0,1,2
<i>UARTx</i>	x=3,4
输入参数{in}	
paritycfg	配置USART奇偶校验
<i>USART_PM_NONE</i>	无校验
<i>USART_PM_ODD</i>	奇校验
<i>USART_PM_EVEN</i>	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 3-669. 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 3-670. 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 T	0.5 bit, 该位对UARTx(x=3,4)无效
USART_STB_2BIT	2 bits
USART_STB_1_5BIT T	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 3-671. 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 3-672. 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	除能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 3-673. 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/UART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 transmit */
```

```
usart_transmit_config(USART0, USART_TRANSMIT_ENABLE);
```

函数 usart_receive_config

函数usart_receive_config描述见下表：

表 3-674. 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/UART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/除能USART接收器
USART_RECEIVE_ENABLE	使能USART接收
USART_RECEIVE_DISABLE	除能USART接收
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* Configure USART0 receive */
```

```
usart_receive_config(USART0, USART_RECEIVE_ENABLE);
```

函数 usart_data_transmit

函数usart_data_transmit描述见下表：

表 3-675. 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/UART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

<i>UARTx</i>	<i>x</i> =3,4
输入参数{in}	
data	发送的数据（0-0xFF）
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 3-676. 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/UART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
<i>USARTx</i>	<i>x</i> =0,1,2
<i>UARTx</i>	<i>x</i> =3,4
输出参数{out}	
-	-
返回值	
uint16_t	接收的数据（0-0xFF）

例如：

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

函数 usart_address_config

函数usart_address_config描述见下表：

表 3-677. 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）
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 3-678. 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 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 3-679. 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	除能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 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 3-680. 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 w method);
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
<i>USART_WM_IDLE</i>	空闲线唤醒
<i>USART_WM_ADDR</i>	地址匹配唤醒
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* configure USART0 wakeup method in mute mode */
usart_mute_mode_wakeup_config(USART0, USART_WM_IDLE);
```

函数 usart_lin_mode_enable

函数usart_lin_mode_enable描述见下表：

表 3-681. 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
<i>USARTx</i>	x=0,1,2
<i>UARTx</i>	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 3-682. 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	除能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 3-683. 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模式中中断帧长度
USART_LBLEN_10 B	断开帧长度为10 bits
USART_LBLEN_11 B	断开帧长度为11 bits
输出参数{out}	
-	-
返回值	
-	-

例如：

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

```
usart_lin_break_dection_length_config(USART0, USART_LBLEN_10B);
```

函数 usart_send_break

函数usart_send_break描述见下表:

表 3-684. 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 3-685. 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 3-686. 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	除能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 3-687. 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 3-688. 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同步通讯模式下除能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 3-689. 函数 usart_synchronous_clock_config

函数名称	usart_synchronous_clock_config
函数原型	void usart_synchronous_clock_config(uint32_t usart_periph, uint32_t clen, uint32_t cph, uint32_t cpl);
功能描述	配置USART同步通讯模式参数
先决条件	-
被调用函数	-
输入参数{in}	
usart_periph	外设USARTx
USARTx	x=0,1,2
输入参数{in}	
clen	CK信号长度
USART_CLEN_NO	8位数据帧中有7个CK脉冲，9位数据帧中有8个CK脉冲

NE	
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 3-690. 函数 usart_guard_time_config

函数名称	usart_guard_time_config
函数原型	void usart_guard_time_config(uint32_t usart_periph,uint32_t guat);
功能描述	在USART智能卡模式下配置保护时间值
先决条件	-
被调用函数	-
输入参数{in}	
usart_periph	外设USARTx
USARTx	x=0,1,2
输入参数{in}	
guat	保护时间值(0-0x000000FF)
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* configure USART0 guard time value in smartcard mode */
```

```
usart_guard_time_config(USART0, 0x0000 0055);
```

函数 usart_smartcard_mode_enable

函数usart_smartcard_mode_enable描述见下表：

表 3-691. 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 3-692. 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	除能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 3-693. 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 3-694. 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智能卡模式下除能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_irda_mode_enable

函数usart_irda_mode_enable描述见下表：

表 3-695. 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/UARTx
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 3-696. 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	除能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 3-697. 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）
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 3-698. 函数 usart_irda_lowpower_config

函数名称	usart_irda_low power_config
函数原型	void usart_irda_low power_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 3-699. 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/除能RTS
USART_RTS_ENABLE	使能RTS
USART_RTS_DISABLE	除能RTS
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* configure USART0 hardware flow control RTS */
```

```
usart_hardware_flow_rts_config(USART0, USART_RTS_ENABLE);
```

函数 usart_hardware_flow_cts_config

函数usart_hardware_flow_cts_config描述见下表：

表 3-700. 函数 `usart hardware_flow_cts_config`

函数名称	<code>usart hardware_flow_cts_config</code>
函数原型	<code>void usart hardware_flow_cts_config(uint32_t usart_periph, uint32_t ctsconfig);</code>
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}	
<code>usart_periph</code>	外设USARTx
<code>USARTx</code>	x=0,1,2
输入参数{in}	
<code>ctsconfig</code>	使能/除能CTS
<code>USART_CTS_ENA BLE</code>	使能CTS
<code>USART_CTS_DISA BLE</code>	除能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 3-701. 函数 `usart_dma_receive_config`

函数名称	<code>usart_dma_receive_config</code>
函数原型	<code>void usart_dma_receive_config(uint32_t usart_periph, uint32_t dmamcmd);</code>
功能描述	配置 USART DMA接收功能
先决条件	-
被调用函数	-
输入参数{in}	
<code>usart_periph</code>	外设USARTx/UARTx
<code>USARTx</code>	x=0,1,2
<code>UARTx</code>	x=3,4
输入参数{in}	
<code>dmamcmd</code>	使能/除能DMA接收功能
<code>USART_DENR_EN ABLE</code>	使能DMA接收功能
<code>USART_DENR_DIS</code>	除能DMA接收功能

ABLE	
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* USART0 DMA enable for reception */
```

```
usart_dma_receive_config(USART0, USART_DENR_ENABLE);
```

函数 usart_dma_transmit_config

函数usart_dma_transmit_config描述见下表：

表 3-702. 函数 usart_dma_transmit_config

函数名称	usart_dma_transmit_config
函数原型	void usart_dma_transmit_config(uint32_t usart_periph, uint32_t dmacmd);
功能描述	配置 USART DMA发送功能
先决条件	-
被调用函数	-
输入参数{in}	
usart_periph	外设USARTx/UARTx
USARTx	x=0,1,2
UARTx	x=3
输入参数{in}	
dmacmd	使能/除能DMA发送功能
USART_DENT_EN ABLE	使能DMA发送功能
USART_DENT_DIS ABLE	除能DMA发送功能
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* USART0 DMA enable for transmission */
```

```
usart_dma_transmit_config(USART0, USART_DENT_ENABLE);
```

函数 usart_flag_get

函数usart_flag_get描述见下表：

表 3-703. 函数 `usart_flag_get`

函数名称	<code>usart_flag_get</code>
函数原型	<code>FlagStatus usart_flag_get(uint32_t usart_periph, usart_flag_enum flag);</code>
功能描述	获取USART状态寄存器标志位
先决条件	-
被调用函数	-
输入参数{in}	
<code>usart_periph</code>	外设USARTx/UARTx
<code>USARTx</code>	x=0,1,2
<code>UARTx</code>	x=3,4
输入参数{in}	
<code>flag</code>	USART标志位, 参考枚举 <code>usart_flag_enum</code>
<code>USART_FLAG_CTS</code> <code>F</code>	CTS变化标志
<code>USART_FLAG_LBD</code> <code>F</code>	LIN断开检测标志
<code>USART_FLAG_TBE</code>	发送数据缓冲区空
<code>USART_FLAG_TC</code>	发送完成
<code>USART_FLAG_RB</code> <code>NE</code>	读数据缓冲区非空
<code>USART_FLAG_IDLE</code> <code>EF</code>	空闲线检测标志
<code>USART_FLAG_OR</code> <code>ERR</code>	溢出错误
<code>USART_FLAG_NE</code> <code>RR</code>	噪声错误标志
<code>USART_FLAG_FER</code> <code>R</code>	帧错误标志
<code>USART_FLAG_PE</code> <code>RR</code>	校验错误标志
输出参数{out}	
-	-
返回值	
<code>FlagStatus</code>	SET or 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 3-704. 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, 参考枚举usart_flag_enum
USART_FLAG_CTS F	CTS变化标志
USART_FLAG_LBD F	LIN断开检测标志
USART_FLAG_TC	发送完成
USART_FLAG_RB NE	读数据缓冲区非空
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 3-705. 函数 usart_interrupt_enable

函数名称	usart_interrupt_enable
函数原型	void usart_interrupt_enable(uint32_t usart_periph, uint32_t 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中断标志
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
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 3-706. 函数 usart_interrupt_disable

函数名称	usart_interrupt_disable
函数原型	void usart_interrupt_disable(uint32_t usart_periph, uint32_t interrupt);
功能描述	除能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中断标志
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
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 3-707. 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, uint32_t 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, 参考枚举usart_interrupt_flag_enum
USART_INT_FLAG_PERR	校验错误中断和标志
USART_INT_FLAG_TBE	发送缓冲区空中断和标志
USART_INT_FLAG_TC	发送完成中断和标志
USART_INT_FLAG_RBNE	读数据缓冲区非空中断和标志
USART_INT_FLAG_RBNE_ORERR	读数据缓冲区非空中断和过载错误中断标志
USART_INT_FLAG_IDLE	IDLE线检测中断和标志
USART_INT_FLAG_LBD	LIN断开检测中断和标志
USART_INT_FLAG_CTS	CTS中断和标志

USART_INT_FLAG _ERR_ORERR	错误中断和过载错误
USART_INT_FLAG _ERR_NERR	错误中断和噪声错误标志
USART_INT_FLAG _ERR_FERR	错误中断和帧错误标志
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 3-708. 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, uint32_t 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
USART_INT_FLAG _CTS	CTS变化标志
USART_INT_FLAG _LBD	LIN断开检测标志
USART_INT_FLAG _TC	发送完成
USART_INT_FLAG _RBNE	读数据缓冲区非空
输出参数{out}	
-	-
返回值	

-	-
---	---

例如:

```
/* clear the USART0 interrupt enable bit status */
usart_interrupt_flag_clear(USART0, USART_INT_FLAG_RBNE);
```

3.24. WWDGT

窗口看门狗定时器(WWDGT)用来监测由软件故障导致的系统故障。章节[3.24.1](#)描述了WWDGT的寄存器列表，章节[3.24.2](#)对WWDGT库函数进行说明。

3.24.1. 外设寄存器说明

WWDGT寄存器列表如下表所示:

表 3-709. 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

3.24.2. 外设库函数说明

WWDGT库函数列表如下表所示:

表 3-710. WWDGT 库函数

库函数名称	库函数说明
w w d g t _ d e i n i t	将WWDGT寄存器重设为缺省值
w w d g t _ e n a b l e	使能WWDGT
w w d g t _ c o u n t e r _ u p d a t e	设置WWDGT计数器更新值
w w d g t _ c o n f i g	设置WWDGT计数器值、窗口值和预分频值
w w d g t _ i n t e r r u p t _ e n a b l e	使能WWDGT提前唤醒中断
w w d g t _ f l a g _ g e t	检查WWDGT提前唤醒中断标志位是否置位
w w d g t _ f l a g _ c l e a r	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 3-711. 函数 wwdgt_deinit

函数名称	w w d g t _ d e i n i t
函数原型	void w w d g t _ d e i n i t(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	rcu_periph_reset_enable / rcu_periph_reset_disable
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 3-712. 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 3-713. 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 3-714. 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:

```
/* configure WWDGT counter value to 0x7F, window value to 0x50, prescaler divider value to 8 */
```

```
wwdgt_config(127, 80, WWDGT_CFG_PSC_DIV8);
```

函数 wwdgt_interrupt_enable

函数wwdgt_interrupt_enable描述见下表:

表 3-715. 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 3-716. 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( );
```

函数 wwdgt_flag_clear

函数wwdgt_flag_clear描述见下表:

表 3-717. 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	2018 年 3 月 26 日
2.0	根据最新规范及固件库修改	2021 年 6 月 1 日
2.1	<ol style="list-style-type: none"> 在 3.13.2 中增加函数接口 fw_dgt_prescaler_value_config 和 fw_dgt_reload_value_config。 修改 3.15.2 中函数接口 i2c_dma_enable / i2c_pec_enable / i2c_pec_transfer_enable / i2c_smbus_issue_alert / i2c_smbus_arp_enable 为 i2c_dma_config / i2c_pec_config / i2c_pec_transfer_config / i2c_smbus_alert_config / i2c_smbus_arp_config。 修改 3.17.2 中函数接口 pmu_to_standbymode(WFI_CMD)为 pmu_to_standbymode()。 修改 3.18.2 中函数顺序。 	2022 年 7 月 11 日
2.2	<ol style="list-style-type: none"> 修改 3.6 章节，DAC 一致性更新 	2024 年 1 月 5 日

Important Notice

This document is the property of GigaDevice Semiconductor Inc. and its subsidiaries (the "Company"). This document, including any product of the Company described in this document (the "Product"), is owned by the Company under the intellectual property laws and treaties of the People's Republic of China and other jurisdictions worldwide. The Company reserves all rights under such laws and treaties and does not grant any license under its patents, copyrights, trademarks, or other intellectual property rights. The names and brands of third party referred thereto (if any) are the property of their respective owner and referred to for identification purposes only.

The Company makes no warranty of any kind, express or implied, with regard to this document or any Product, including, but not limited to, the implied warranties of merchantability and fitness for a particular purpose. The Company does not assume any liability arising out of the application or use of any Product described in this document. Any information provided in this document is provided only for reference purposes. It is the responsibility of the user of this document to properly design, program, and test the functionality and safety of any application made of this information and any resulting product. Except for customized products which has been expressly identified in the applicable agreement, the Products are designed, developed, and/or manufactured for ordinary business, industrial, personal, and/or household applications only. The Products are not designed, intended, or authorized for use as components in systems designed or intended for the operation of weapons, weapons systems, nuclear installations, atomic energy control instruments, combustion control instruments, airplane or spaceship instruments, transportation instruments, traffic signal instruments, life-support devices or systems, other medical devices or systems (including resuscitation equipment and surgical implants), pollution control or hazardous substances management, or other uses where the failure of the device or Product could cause personal injury, death, property or environmental damage ("Unintended Uses"). Customers shall take any and all actions to ensure using and selling the Products in accordance with the applicable laws and regulations. The Company is not liable, in whole or in part, and customers shall and hereby do release the Company as well as its suppliers and/or distributors from any claim, damage, or other liability arising from or related to all Unintended Uses of the Products. Customers shall indemnify and hold the Company as well as its suppliers and/or distributors harmless from and against all claims, costs, damages, and other liabilities, including claims for personal injury or death, arising from or related to any Unintended Uses of the Products.

Information in this document is provided solely in connection with the Products. The Company reserves the right to make changes, corrections, modifications or improvements to this document and Products and services described herein at any time, without notice.