

GigaDevice Semiconductor Inc.

GD32F20x
ARM[®] Cortex[®]-M3 32-bit MCU

**固件库
使用指南**

1.6 版本

(2026 年 2 月)

目录

目录	2
图索引	5
表索引	6
1. 介绍	27
1.1. 文档和固件库规则	27
1.1.1. 外设缩写	27
1.1.2. 命名规则	28
2. 固件库概述	29
2.1. 文件组织结构	29
2.1.1. Examples 文件夹	30
2.1.2. Firmware 文件夹	30
2.1.3. Template 文件夹	30
2.1.4. Utilities 文件夹	33
2.2. 固件库文件描述	34
3. 外设固件库	35
3.1. 外设固件库概述	35
3.2. ADC	35
3.2.1. 外设寄存器描述	35
3.2.2. 外设库函数说明	36
3.3. BKP	64
3.3.1. 外设寄存器说明	64
3.3.2. 外设库函数说明	64
3.4. CAN	78
3.4.1. 外设寄存器说明	78
3.4.2. 外设库函数说明	79
3.5. CAU	102
3.5.1. 外设寄存器说明	102
3.5.2. 外设库函数说明	103
3.6. CRC	124
3.6.1. 外设寄存器说明	124
3.6.2. 外设库函数说明	125
3.7. DAC	129
3.7.1. 外设寄存器说明	129
3.7.2. 外设库函数说明	129

3.8. DBG	144
3.8.1. 外设寄存器说明	144
3.8.2. 外设库函数说明	144
3.9. DCI	150
3.9.1. 外设寄存器说明	150
3.9.2. 外设库函数说明	150
3.10. DMA	163
3.10.1. 外设寄存器说明	163
3.10.2. 外设库函数说明	163
3.11. ENET	185
3.11.1. 外设寄存器说明	185
3.11.2. 外设库函数说明	187
3.12. EXMC	266
3.12.1. 外设寄存器说明	266
3.12.2. 外设库函数说明	267
3.13. EXTI.....	302
3.13.1. 外设寄存器说明	302
3.13.2. 外设库函数说明	303
3.14. FMC	310
3.14.1. 外设寄存器说明	310
3.14.2. 外设库函数说明	310
3.15. FWDGT.....	329
3.15.1. 外设寄存器说明	329
3.15.2. 外设库函数说明	329
3.16. GPIO	334
3.16.1. 外设寄存器说明	334
3.16.2. 外设库函数说明	335
3.17. HAU	354
3.17.1. 外设寄存器说明	355
3.17.2. 外设库函数说明	355
3.18. I2C	372
3.18.1. 外设寄存器说明	372
3.18.2. 外设库函数说明	372
3.19. MISC	394
3.19.1. 外设寄存器说明	394
3.19.2. 外设库函数说明	395
3.20. PMU.....	402
3.20.1. 外设寄存器说明	403
3.20.2. 外设库函数说明	403

3.21.	RCU.....	410
3.21.1.	外设寄存器说明	410
3.21.2.	外设库函数说明	411
3.22.	RTC	448
3.22.1.	外设寄存器说明	448
3.22.2.	外设库函数说明	448
3.23.	SDIO.....	456
3.23.1.	外设寄存器说明	456
3.23.2.	外设库函数说明	457
3.24.	SPI.....	488
3.24.1.	外设寄存器说明	488
3.24.2.	外设库函数说明	488
3.25.	TIMER	512
3.25.1.	外设寄存器说明	512
3.25.2.	外设库函数说明	512
3.26.	TLI.....	567
3.26.1.	外设寄存器说明	567
3.26.2.	外设库函数说明	568
3.27.	TRNG.....	586
3.27.1.	外设寄存器说明	586
3.27.2.	外设库函数说明	586
3.28.	USART	592
3.28.1.	外设寄存器说明	592
3.28.2.	外设库函数说明	592
3.29.	WWDGT	625
3.29.1.	外设寄存器说明	625
3.29.2.	外设库函数说明	625
4.	版本历史.....	630

图索引

图 2-1. GD32F20x 固件库文件组织结构	29
图 2-2. 选择外设例程文件	31
图 2-3. 拷贝外设例程文件	32
图 2-4. 打开工程文件	32
图 2-5. 配置工程文件	33
图 2-6. 编译调试下载	33

表索引

表 1-1. 外设缩写	27
表 2-1. 固件函数库文件描述	34
表 3-1. 外设固件库函数描述格式	35
表 3-2. ADC 寄存器	35
表 3-3. ADC 库函数	36
表 3-4. 函数 adc_deinit	37
表 3-5. 函数 adc_mode_config	37
表 3-6. 函数 adc_special_function_config	39
表 3-7. 函数 adc_data_alignment_config	39
表 3-8. 函数 adc_enable	40
表 3-9. 函数 adc_disable	41
表 3-10. 函数 adc_calibration_enable	41
表 3-11. 函数 adc_tempsensor_vrefint_enable	42
表 3-12. 函数 adc_tempsensor_vrefint_disable	42
表 3-13. 函数 adc_dma_mode_enable	43
表 3-14. 函数 adc_dma_mode_disable	43
表 3-15. 函数 adc_discontinuous_mode_config	44
表 3-16. 函数 adc_channel_length_config	44
表 3-17. 函数 adc_regular_channel_config	45
表 3-18. 函数 adc_inserted_channel_config	46
表 3-19. 函数 adc_inserted_channel_offset_config	47
表 3-20. 函数 adc_external_trigger_source_config	48
表 3-21. 函数 adc_external_trigger_config	51
表 3-22. 函数 adc_software_trigger_enable	51
表 3-23. 函数 adc_regular_data_read	52
表 3-24. 函数 adc_inserted_data_read	53
表 3-25. 函数 adc_sync_mode_convert_value_read	53
表 3-26. 函数 adc_watchdog_single_channel_enable	54
表 3-27. 函数 adc_watchdog_group_channel_enable	54
表 3-28. 函数 adc_watchdog_disable	55
表 3-29. 函数 adc_watchdog_threshold_config	56
表 3-30. 函数 adc_resolution_config	56
表 3-31. 函数 adc_oversample_mode_config	57
表 3-32. 函数 adc_oversample_mode_enable	59
表 3-33. 函数 adc_oversample_mode_disable	59
表 3-34. 函数 adc_flag_get	60
表 3-35. 函数 adc_flag_clear	60
表 3-36. 函数 adc_interrupt_enable	61
表 3-37. 函数 adc_interrupt_disable	62
表 3-38. 函数 adc_interrupt_flag_get	62

表 3-39. 函数 adc_interrupt_flag_clear	63
表 3-40. BKP 寄存器	64
表 3-41. BKP 库函数	64
表 3-42. 枚举类型 bkp_data_register_enum	65
表 3-43. 枚举类型 bkp_tamper_enum	66
表 3-44. 函数 bkp_deinit	66
表 3-45. 函数 bkp_data_write	67
表 3-46. 函数 bkp_data_read	67
表 3-47. 函数 bkp_rtc_calibration_output_enable	68
表 3-48. 函数 bkp_rtc_calibration_output_disable	68
表 3-49. 函数 bkp_rtc_signal_output_enable	69
表 3-50. 函数 bkp_rtc_signal_output_disable	69
表 3-51. 函数 bkp_rtc_output_select	70
表 3-52. 函数 bkp_rtc_clock_output_select	70
表 3-53. 函数 bkp_rtc_clock_calibration_direction	71
表 3-54. 函数 bkp_rtc_calibration_value_set	71
表 3-55. 函数 bkp_tamper_detection_enable	72
表 3-56. 函数 bkp_tamper_detection_disable	72
表 3-57. 函数 bkp_tamper_active_level_set	73
表 3-58. 函数 bkp_waveform_detect_config	74
表 3-59. 函数 bkp_flag_get	74
表 3-60. 函数 bkp_flag_clear	75
表 3-61. 函数 bkp_tamper_interrupt_enable	75
表 3-62. 函数 bkp_tamper_interrupt_disable	76
表 3-63. 函数 bkp_interrupt_flag_get	77
表 3-64. 函数 bkp_interrupt_flag_clear	77
表 3-65. CAN 寄存器	78
表 3-66. CAN 库函数	79
表 3-67. 结构体 can_parameter_struct	79
表 3-68. 结构体 can_transmit_message_struct	80
表 3-69. 结构体 can_receive_message_struct	80
表 3-70. 结构体 can_filter_parameter_struct	80
表 3-71. 枚举类型 can_flag_enum	81
表 3-72. 枚举类型 can_interrupt_flag_enum	82
表 3-73. 枚举类型 can_error_enum	82
表 3-74. 枚举类型 can_transmit_state_enum	82
表 3-75. 枚举类型 can_struct_type_enum	83
表 3-76. 函数 can_deinit	83
表 3-77. 函数 can_struct_para_init	83
表 3-78. 函数 can_init	84
表 3-79. 函数 can_filter_init	85
表 3-80. 函数 can1_filter_start_bank	86
表 3-81. 函数 can_debug_freeze_enable	86
表 3-82. 函数 can_debug_freeze_disable	87

表 3-83. 函数 can_time_trigger_mode_enable	87
表 3-84. 函数 can_time_trigger_mode_disable	88
表 3-85. 函数 can_message_transmit	88
表 3-86. 函数 can_transmit_states	89
表 3-87. 函数 can_transmission_stop	90
表 3-88. 函数 can_message_receive	91
表 3-89. 函数 can_fifo_release	91
表 3-90. 函数 can_receive_message_length_get	92
表 3-91. 函数 can_working_mode_set	93
表 3-92. 函数 can_wakeup	93
表 3-93. 函数 can_error_get	94
表 3-94. 函数 can_receive_error_number_get	94
表 3-95. 函数 can_transmit_error_number_get	95
表 3-96. 函数 can_flag_get	95
表 3-97. 函数 can_flag_clear	97
表 3-98. 函数 can_interrupt_enable	98
表 3-99. 函数 can_interrupt_disable	99
表 3-100. 函数 can_interrupt_flag_get	100
表 3-101. 函数 can_interrupt_flag_clear	101
表 3-102. CAU 寄存器	102
表 3-103. CAU 库函数	103
表 3-104. 结构体 cau_key_parameter_struct	104
表 3-105. 结构体 cau_iv_parameter_struct	104
表 3-106. 枚举类型 cau_text_struct	104
表 3-107. 函数 cau_deinit	104
表 3-108. 函数 cau_enable	105
表 3-109. 函数 cau_disable	105
表 3-110. 函数 cau_dma_enable	106
表 3-111. 函数 cau_dma_disable	106
表 3-112. 函数 cau_init	107
表 3-113. 函数 cau_aes_keysize_config	108
表 3-114. 函数 cau_key_init	108
表 3-115. 函数 cau_key_parameter_init	109
表 3-116. 函数 cau_iv_init	110
表 3-117. 函数 cau_iv_parameter_init	110
表 3-118. 函数 cau_fifo_flush	111
表 3-119. 函数 cau_enable_state_get	111
表 3-120. 函数 cau_data_write	112
表 3-121. 函数 cau_data_read	112
表 3-122. 函数 cau_aes_ecb	113
表 3-123. 函数 cau_aes_cbc	114
表 3-124. 函数 cau_aes_ctr	116
表 3-125. 函数 cau_tdes_ecb	117
表 3-126. 函数 cau_tdes_cbc	118

表 3-127. 函数 cau_des_ecb	119
表 3-128. 函数 cau_des_cbc	121
表 3-129. 函数 cau_flag_get	122
表 3-130. 函数 cau_interrupt_enable	123
表 3-131. 函数 cau_interrupt_disable	123
表 3-132. 函数 cau_interrupt_flag_get	124
表 3-133. CRC 寄存器	125
表 3-134. CRC 库函数	125
表 3-135. 函数 crc_deinit	125
表 3-136. 函数 crc_data_register_reset	126
表 3-137. 函数 crc_data_register_read	126
表 3-138. 函数 crc_free_data_register_read	127
表 3-139. 函数 crc_free_data_register_write	127
表 3-140. 函数 crc_single_data_calculate	128
表 3-141. 函数 crc_block_data_calculate	128
表 3-142. DAC 寄存器	129
表 3-143. DAC 库函数	129
表 3-144. 函数 dac_deinit	130
表 3-145. 函数 dac_enable	131
表 3-146. 函数 dac_disable	131
表 3-147. 函数 dac_dma_enable	132
表 3-148. 函数 dac_dma_disable	132
表 3-149. 函数 dac_output_buffer_enable	133
表 3-150. 函数 dac_output_buffer_disable	133
表 3-151. 函数 dac_output_value_get	134
表 3-152. 函数 dac_data_set	135
表 3-153. 函数 dac_trigger_enable	135
表 3-154. 函数 dac_trigger_disable	136
表 3-155. 函数 dac_trigger_source_config	137
表 3-156. 函数 dac_software_trigger_enable	138
表 3-157. 函数 dac_wave_mode_config	138
表 3-158. 函数 dac_lfsr_noise_config	139
表 3-159. 函数 dac_triangle_noise_config	140
表 3-160. 函数 dac_concurrent_enable	140
表 3-161. 函数 dac_concurrent_disable	141
表 3-162. 函数 dac_concurrent_software_trigger_enable	141
表 3-163. 函数 dac_concurrent_output_buffer_enable	142
表 3-164. 函数 dac_concurrent_output_buffer_disable	142
表 3-165. 函数 dac_concurrent_data_set	143
表 3-166. DBG 寄存器	144
表 3-167. DBG 库函数	144
表 3-168. 枚举类型 dbg_periph_enum	144
表 3-169. 函数 dbg_id_get	145
表 3-170. 函数 dbg_low_power_enable	145

表 3-171. 函数 dbg_low_power_disable	146
表 3-172. 函数 dbg_periph_enable	147
表 3-173. 函数 dbg_periph_disable	147
表 3-174. 函数 dbg_trace_pin_enable	148
表 3-175. 函数 dbg_trace_pin_disable	148
表 3-176. 函数 dbg_trace_pin_mode_set	149
表 3-177. DCI 寄存器	150
表 3-178. DCI 库函数	150
表 3-179. 结构体 dci_parameter_struct	151
表 3-180. 函数 dci_deinit	151
表 3-181. 函数 dci_init	152
表 3-182. 函数 dci_enable	152
表 3-183. 函数 dci_disable	153
表 3-184. 函数 dci_capture_enable	153
表 3-185. 函数 dci_capture_disable	154
表 3-186. 函数 dci_jpeg_enable	154
表 3-187. 函数 dci_jpeg_disable	155
表 3-188. 函数 dci_crop_window_enable	155
表 3-189. 函数 dci_crop_window_disable	156
表 3-190. 函数 dci_crop_window_config	156
表 3-191. 函数 dci_embedded_sync_enable	157
表 3-192. 函数 dci_embedded_sync_disable	157
表 3-193. 函数 dci_sync_codes_config	158
表 3-194. 函数 dci_sync_codes_unmask_config	158
表 3-195. 函数 dci_data_read	159
表 3-196. 函数 dci_flag_get	160
表 3-197. 函数 dci_interrupt_enable	160
表 3-198. 函数 dci_interrupt_disable	161
表 3-199. 函数 dci_interrupt_flag_get	161
表 3-200. 函数 dci_interrupt_flag_clear	162
表 3-201. DMA 寄存器	163
表 3-202. DMA 库函数	163
表 3-203. 结构体 dma_parameter_struct	164
表 3-204. 函数 dma_deinit	165
表 3-205. 函数 dma_struct_para_init	165
表 3-206. 函数 dma_init	166
表 3-207. 函数 dma_circulation_enable	167
表 3-208. 函数 dma_circulation_disable	167
表 3-209. 函数 dma_memory_to_memory_enable	168
表 3-210. 函数 dma_memory_to_memory_disable	169
表 3-211. 函数 dma_channel_enable	169
表 3-212. 函数 dma_channel_disable	170
表 3-213. 函数 dma_periph_address_config	171
表 3-214. 函数 dma_memory_address_config	171

表 3-215. 函数 dma_transfer_number_config	172
表 3-216. 函数 dma_transfer_number_get	173
表 3-217. 函数 dma_priority_config.....	174
表 3-218. 函数 dma_memory_width_config	174
表 3-219. 函数 dma_periph_width_config	175
表 3-220. 函数 dma_memory_increase_enable.....	176
表 3-221. 函数 dma_memory_increase_disable.....	177
表 3-222. 函数 dma_periph_increase_enable.....	177
表 3-223. 函数 dma_periph_increase_disable	178
表 3-224. 函数 dma_transfer_direction_config	179
表 3-225. 函数 dma_1_channel_5_fulldata_transfer_enable	179
表 3-226. 函数 dma_1_channel_5_fulldata_transfer_disable	180
表 3-227. 函数 dma_flag_get.....	180
表 3-228. 函数 dma_flag_clear	181
表 3-229. 函数 dma_interrupt_enable	182
表 3-230. 函数 dma_interrupt_disable	183
表 3-231. 函数 dma_interrupt_flag_get.....	184
表 3-232. 函数 dma_interrupt_flag_clear.....	184
表 3-233. ENET 寄存器.....	185
表 3-234. ENET 库函数.....	187
表 3-235. 结构体 enet_descriptors_struct	190
表 3-236. 结构体 enet_ptp_systime_struct	190
表 3-237. 枚举类型 enet_flag_enum	190
表 3-238. 枚举类型 enet_flag_clear_enum.....	192
表 3-239. 枚举类型 enet_int_enum.....	193
表 3-240. 枚举类型 enet_int_flag_enum.....	194
表 3-241. 枚举类型 enet_int_flag_clear_enum.....	196
表 3-242. 枚举类型 enet_desc_reg_enum.....	196
表 3-243. 枚举类型 enet_msc_counter_enum.....	197
表 3-244. 枚举类型 enet_option_enum.....	197
表 3-245. 枚举类型 enet_mediamode_enum	198
表 3-246. 枚举类型 enet_chksumconf_enum	198
表 3-247. 枚举类型 enet_frmrecept_enum.....	198
表 3-248. 枚举类型 enet_registers_type_enum	199
表 3-249. 枚举类型 enet_dmadirection_enum	199
表 3-250. 枚举类型 enet_phydirection_enum	199
表 3-251. 枚举类型 enet_regdirection_enum	199
表 3-252. 枚举类型 enet_macaddress_enum	199
表 3-253. 枚举类型 enet_descstate_enum.....	200
表 3-254. 函数 enet_deinit.....	200
表 3-255. 函数 enet_initpara_config.....	201
表 3-256. 函数 enet_init	204
表 3-257. 函数 enet_software_reset	205
表 3-258. 函数 enet_rxframe_size_get.....	206

表 3-259. 函数 enet_descriptors_chain_init.....	206
表 3-260. 函数 enet_descriptors_ring_init	207
表 3-261. 函数 enet_frame_receive.....	207
表 3-262. 函数 enet_frame_transmit.....	208
表 3-263. 函数 enet_transmit_checksum_config	209
表 3-264. 函数 enet_enable	209
表 3-265. 函数 enet_disable	210
表 3-266. 函数 enet_mac_address_set	210
表 3-267. 函数 enet_mac_address_get	211
表 3-268. 函数 enet_flag_get.....	212
表 3-269. 函数 enet_flag_clear	214
表 3-270. 函数 enet_interrupt_enable	216
表 3-271. 函数 enet_interrupt_disable	217
表 3-272. 函数 enet_interrupt_flag_get.....	219
表 3-273. 函数 enet_interrupt_flag_clear.....	221
表 3-274. 函数 enet_tx_enable.....	222
表 3-275. 函数 enet_tx_disable.....	222
表 3-276. 函数 enet_rx_enable	223
表 3-277. 函数 enet_rx_disable	223
表 3-278. 函数 enet_registers_get	224
表 3-279. 函数 enet_address_filter_enable	224
表 3-280. 函数 enet_address_filter_disable	225
表 3-281. 函数 enet_address_filter_config.....	226
表 3-282. 函数 enet_phy_config.....	227
表 3-283. 函数 enet_phy_write_read	227
表 3-284. 函数 enet_phyloopback_enable.....	228
表 3-285. 函数 enet_phyloopback_disable.....	229
表 3-286. 函数 enet_forward_feature_enable	229
表 3-287. 函数 enet_forward_feature_disable	230
表 3-288. 函数 enet_fliter_feature_enable	230
表 3-289. 函数 enet_fliter_feature_disable	231
表 3-290. 函数 enet_pauseframe_generate	232
表 3-291. 函数 enet_pauseframe_detect_config	233
表 3-292. 函数 enet_pauseframe_config	233
表 3-293. 函数 enet_flowcontrol_threshold_config.....	234
表 3-294. 函数 enet_flowcontrol_feature_enable	236
表 3-295. 函数 enet_flowcontrol_feature_disable.....	236
表 3-296. 函数 enet_dmaprocess_state_get	237
表 3-297. 函数 enet_dmaprocess_resume	238
表 3-298. 函数 enet_rxprocess_check_recovery	238
表 3-299. 函数 enet_txfifo_flush.....	239
表 3-300. 函数 enet_current_desc_address_get.....	239
表 3-301. 函数 enet_desc_information_get.....	240
表 3-302. 函数 enet_missed_frame_counter_get.....	241

表 3-303. 函数 enet_desc_flag_get.....	242
表 3-304. 函数 enet_desc_flag_set.....	244
表 3-305. 函数 enet_desc_flag_clear.....	245
表 3-306. 函数 enet_desc_receive_complete_bit_enable.....	246
表 3-307. 函数 enet_desc_receive_complete_bit_disable.....	247
表 3-308. 函数 enet_rxframe_drop.....	247
表 3-309. 函数 enet_dma_feature_enable	248
表 3-310. 函数 enet_dma_feature_disable	248
表 3-311. 函数 enet_ptp_normal_descriptors_chain_init	249
表 3-312. 函数 enet_ptp_normal_descriptors_ring_init.....	249
表 3-313. 函数 enet_ptpframe_receive_normal_mode.....	250
表 3-314. 函数 enet_ptpframe_transmit_normal_mode.....	251
表 3-315. 函数 enet_wum_filter_register_pointer_reset.....	251
表 3-316. 函数 enet_wum_filter_config.....	252
表 3-317. 函数 enet_wum_feature_enable.....	252
表 3-318. 函数 enet_wum_feature_disable	253
表 3-319. 函数 enet_msc_counters_reset	254
表 3-320. 函数 enet_msc_feature_enable.....	254
表 3-321. 函数 enet_msc_feature_disable.....	255
表 3-322. 函数 enet_msc_counters_get.....	255
表 3-323. 函数 enet_ptp_subsecond_2_nanosecond	256
表 3-324. 函数 enet_ptp_nanosecond_2_subsecond	257
表 3-325. 函数 enet_ptp_feature_enable	257
表 3-326. 函数 enet_ptp_feature_disable	258
表 3-327. 函数 enet_ptp_timestamp_function_config.....	258
表 3-328. 函数 enet_ptp_subsecond_increment_config	259
表 3-329. 函数 enet_ptp_timestamp_addend_config	260
表 3-330. 函数 enet_ptp_timestamp_update_config	260
表 3-331. 函数 enet_ptp_expected_time_config	261
表 3-332. 函数 enet_ptp_system_time_get	261
表 3-333. 函数 enet_ptp_start.....	262
表 3-334. 函数 enet_ptp_finecorrection_adjfreq	263
表 3-335. 函数 enet_ptp_coarsecorrection_systime_update.....	263
表 3-336. 函数 enet_ptp_finecorrection_settime	264
表 3-337. 函数 enet_ptp_flag_get	265
表 3-338. 函数 enet_initpara_reset	265
表 3-339. EXMC 寄存器	266
表 3-340. EXMC 库函数	267
表 3-341. 结构体 exmc_norsram_timing_parameter_struct.....	268
表 3-342. 结构体 exmc_norsram_parameter_struct	268
表 3-343. 结构体 exmc_nand_pccard_timing_parameter_struct	269
表 3-344. 结构体 exmc_nand_parameter_struct	269
表 3-345. 结构体 exmc_pccard_parameter_struct.....	270
表 3-346. 结构体 exmc_sdram_timing_parameter_struct	270

表 3-347. 结构体 exmc_sdram_parameter_struct	270
表 3-348. 结构体 exmc_sdram_command_parameter_struct	271
表 3-349. 结构体 exmc_sqpsram_parameter_struct	271
表 3-350. 函数 exmc_norsram_deinit	272
表 3-351. 函数 exmc_norsram_struct_para_init	272
表 3-352. 函数 exmc_norsram_init	273
表 3-353. 函数 exmc_norsram_enable	274
表 3-354. 函数 exmc_norsram_disable	275
表 3-355. 函数 exmc_nand_deinit	275
表 3-356. 函数 exmc_nand_struct_para_init	276
表 3-357. 函数 exmc_nand_init	276
表 3-358. 函数 exmc_nand_enable	277
表 3-359. 函数 exmc_nand_disable	278
表 3-360. 函数 exmc_nand_ecc_config	278
表 3-361. 函数 exmc_ecc_get	279
表 3-362. 函数 exmc_pccard_deinit	280
表 3-363. 函数 exmc_pccard_struct_para_init	280
表 3-364. 函数 exmc_pccard_init	281
表 3-365. 函数 exmc_pccard_enable	282
表 3-366. 函数 exmc_pccard_disable	282
表 3-367. 函数 exmc_sdram_deinit	282
表 3-368. 函数 exmc_sdram_struct_para_init	283
表 3-369. 函数 exmc_sdram_init	284
表 3-370. 函数 exmc_sdram_struct_command_para_init	285
表 3-371. 函数 exmc_sdram_command_config	285
表 3-372. 函数 exmc_sdram_refresh_count_set	286
表 3-373. 函数 exmc_sdram_autorefresh_number_set	286
表 3-374. 函数 exmc_sdram_write_protection_config	287
表 3-375. 函数 exmc_sdram_bankstatus_get	287
表 3-376. 函数 exmc_sdram_readsample_config	288
表 3-377. 函数 exmc_sdram_readsample_enable	289
表 3-378. 函数 exmc_sdram_readsample_disable	289
表 3-379. 函数 exmc_sqpsram_deinit	290
表 3-380. 函数 exmc_sqpsram_struct_para_init	290
表 3-381. 函数 exmc_sqpsram_init	291
表 3-382. 函数 exmc_sqpsram_read_command_set	292
表 3-383. 函数 exmc_sqpsram_write_command_set	293
表 3-384. 函数 exmc_sqpsram_read_id_command_send	293
表 3-385. 函数 exmc_sqpsram_write_cmd_send	294
表 3-386. 函数 exmc_sqpsram_low_id_get	294
表 3-387. 函数 exmc_sqpsram_high_id_get	295
表 3-388. 函数 exmc_sqpsram_send_command_state_get	295
表 3-389. 函数 exmc_flag_get	296
表 3-390. 函数 exmc_flag_clear	297

表 3-391. 函数 exmc_interrupt_enable	298
表 3-392. 函数 exmc_interrupt_disable	299
表 3-393. 函数 exmc_interrupt_flag_get	300
表 3-394. 函数 exmc_interrupt_flag_clear	301
表 3-395. EXTI 寄存器	302
表 3-396. EXTI 库函数	303
表 3-397. 枚举类型 exti_line_enum	303
表 3-398. 枚举类型 exti_mode_enum	304
表 3-399. 枚举类型 exti_trig_type_enum	304
表 3-400. 函数 exti_deinit	304
表 3-401. 函数 exti_init	304
表 3-402. 函数 exti_interrupt_enable	305
表 3-403. 函数 exti_interrupt_disable	306
表 3-404. 函数 exti_event_enable	306
表 3-405. 函数 exti_event_disable	306
表 3-406. 函数 exti_software_interrupt_enable	307
表 3-407. 函数 exti_software_interrupt_disable	307
表 3-408. 函数 exti_flag_get	308
表 3-409. 函数 exti_flag_clear	308
表 3-410. 函数 exti_interrupt_flag_get	309
表 3-411. 函数 exti_interrupt_flag_clear	309
表 3-412. FMC 寄存器	310
表 3-413. FMC 固件库函数	311
表 3-414. 枚举类型 fmc_state_enum	311
表 3-415. 枚举类型 fmc_interrupt_enum	312
表 3-416. 枚举类型 fmc_flag_enum	312
表 3-417. 枚举类型 fmc_interrupt_flag_enum	312
表 3-418. 函数 fmc_wscnt_set	313
表 3-419. 函数 fmc_unlock	313
表 3-420. 函数 fmc_bank0_unlock	314
表 3-421. 函数 fmc_bank1_unlock	314
表 3-422. 函数 fmc_lock	315
表 3-423. 函数 fmc_bank0_lock	315
表 3-424. 函数 fmc_bank1_lock	316
表 3-425. 函数 fmc_page_erase	316
表 3-426. 函数 fmc_mass_erase	317
表 3-427. 函数 fmc_bank0_erase	317
表 3-428. 函数 fmc_bank1_erase	318
表 3-429. 函数 fmc_word_program	318
表 3-430. 函数 fmc_halfword_program	319
表 3-431. 函数 ob_unlock	319
表 3-432. 函数 ob_lock	320
表 3-433. 函数 ob_erase	320
表 3-434. 函数 ob_write_protection_enable	321

表 3-435. 函数 ob_security_protection_config.....	321
表 3-436. 函数 ob_user_write	322
表 3-437. 函数 ob_data_program	323
表 3-438. 函数 ob_user_get	323
表 3-439. 函数 ob_data_get.....	324
表 3-440. 函数 ob_write_protection_get.....	325
表 3-441. 函数 ob_spc_get.....	325
表 3-442. 函数 fmc_flag_get.....	326
表 3-443. 函数 fmc_flag_clear	326
表 3-444. 函数 fmc_interrupt_enable	327
表 3-445. 函数 fmc_interrupt_disable	327
表 3-446. 函数 fmc_interrupt_flag_get.....	328
表 3-447. 函数 fmc_interrupt_flag_clear.....	328
表 3-448. FWDGT 寄存器.....	329
表 3-449. FWDGT 库函数.....	329
表 3-450. 函数 fwdgt_write_enable.....	330
表 3-451. 函数 fwdgt_write_disable.....	330
表 3-452. 函数 fwdgt_enable.....	331
表 3-453. 函数 fwdgt_prescaler_value_config.....	331
表 3-454. 函数 fwdgt_reload_value_config	332
表 3-455. 函数 fwdgt_counter_reload	332
表 3-456. 函数 fwdgt_config	333
表 3-457. 函数 fwdgt_flag_get	334
表 3-458. GPIO 寄存器.....	334
表 3-459. GPIO 库函数.....	335
表 3-460. 函数 gpio_deinit.....	336
表 3-461. 函数 gpio_afio_deinit.....	336
表 3-462. 函数 gpio_init	337
表 3-463. 函数 gpio_bit_set.....	338
表 3-464. 函数 gpio_bit_reset.....	338
表 3-465. 函数 gpio_bit_write	339
表 3-466. 函数 gpio_port_write	340
表 3-467. 函数 gpio_input_bit_get	340
表 3-468. 函数 gpio_input_port_get	341
表 3-469. 函数 gpio_output_bit_get.....	341
表 3-470. 函数 gpio_output_port_get.....	342
表 3-471. 函数 gpio_pin_remap_config	342
表 3-472. 函数 gpio_pin_remap1_config	345
表 3-473. 函数 gpio_ethernet_phy_select	351
表 3-474. 函数 gpio_exti_source_select.....	352
表 3-475. 函数 gpio_event_output_config.....	352
表 3-476. 函数 gpio_event_output_enable	353
表 3-477. 函数 gpio_event_output_disable	353
表 3-478. 函数 gpio_pin_lock	354

表 3-479. HAU 寄存器.....	355
表 3-480. HAU 库函数.....	355
表 3-481. 结构体 hau_init_parameter_struct.....	356
表 3-482. 结构体 hau_digest_parameter_struct.....	356
表 3-483. 函数 hau_deinit.....	356
表 3-484. 函数 hau_init	357
表 3-485. 函数 hau_init_struct_para_init.....	357
表 3-486. 函数 hau_reset	358
表 3-487. 函数 hau_last_word_validbits_num_config	359
表 3-488. 函数 hau_data_write	359
表 3-489. 函数 hau_infifo_words_num_get.....	359
表 3-490. 函数 hau_digest_read.....	360
表 3-491. 函数 hau_digest_calculation_enable	361
表 3-492. 函数 hau_multiple_single_dma_config	361
表 3-493. 函数 hau_dma_enable	362
表 3-494. 函数 hau_dma_disable	362
表 3-495. 函数 hau_hash_sha_1	363
表 3-496. 函数 hau_hmac_sha_1	363
表 3-497. 函数 hau_hash_sha_224	364
表 3-498. 函数 hau_hmac_sha_224	365
表 3-499. 函数 hau_hash_sha_256	366
表 3-500. 函数 hau_hmac_sha_256	366
表 3-501. 函数 hau_hash_md5	367
表 3-502. 函数 hau_hmac_md5	368
表 3-503. 函数 hau_flag_get.....	368
表 3-504. 函数 hau_flag_clear	369
表 3-505. 函数 hau_interrupt_enable	370
表 3-506. 函数 hau_interrupt_disable	370
表 3-507. 函数 hau_interrupt_flag_get.....	371
表 3-508. 函数 hau_interrupt_flag_clear.....	371
表 3-509. I2C 寄存器.....	372
表 3-510. I2C 库函数.....	372
表 3-511. 函数 i2c_deinit.....	373
表 3-512. 函数 i2c_clock_config	374
表 3-513. 函数 i2c_mode_addr_config.....	374
表 3-514. 函数 i2c_smbus_type_config	375
表 3-515. 函数 i2c_ack_config.....	376
表 3-516. 函数 i2c_ackpos_config	377
表 3-517. 函数 i2c_master_addressing	377
表 3-518. 函数 i2c_dualaddr_enable	378
表 3-519. 函数 i2c_dualaddr_disable	378
表 3-520. 函数 i2c_enable.....	379
表 3-521. 函数 i2c_disable.....	379
表 3-522. 函数 i2c_start_on_bus.....	380

表 3-523. 函数 i2c_stop_on_bus	380
表 3-524. 函数 i2c_data_transmit.....	381
表 3-525. 函数 i2c_data_receive.....	382
表 3-526. 函数 i2c_dma_enable.....	382
表 3-527. 函数 i2c_dma_last_transfer_config	383
表 3-528. 函数 i2c_stretch_scl_low_config.....	383
表 3-529. 函数 i2c_slave_response_to_gcall_config	384
表 3-530. 函数 i2c_software_reset_config	385
表 3-531. 函数 i2c_pec_enable.....	385
表 3-532. 函数 i2c_pec_transfer_enable.....	386
表 3-533. 函数 i2c_pec_value_get	386
表 3-534. 函数 i2c_smbus_issue_alert	387
表 3-535. 函数 i2c_smbus_arp_enable	388
表 3-536. 函数 i2c_flag_get.....	388
表 3-537. 函数 i2c_flag_clear.....	390
表 3-538. 函数 i2c_interrupt_enable.....	390
表 3-539. 函数 i2c_interrupt_disable.....	391
表 3-540. 函数 i2c_interrupt_flag_get	392
表 3-541. 函数 i2c_interrupt_flag_clear	393
表 3-542. NVIC 寄存器.....	394
表 3-543. SysTick 寄存器.....	395
表 3-544. 枚举类型 IRQn_Type.....	395
表 3-545. MISC 库函数.....	398
表 3-546. 函数 nvic_priority_group_set.....	398
表 3-547. 函数 nvic_irq_enable	399
表 3-548. 函数 nvic_irq_disable	399
表 3-549. 函数 nvic_vector_table_set	400
表 3-550. 函数 system_lowpower_set.....	400
表 3-551. 函数 system_lowpower_reset.....	401
表 3-552. 函数 systick_clksource_set.....	402
表 3-553. PMU 寄存器.....	403
表 3-554. PMU 库函数.....	403
表 3-555. 函数 pmu_deinit.....	403
表 3-556. 函数 pmu_lvd_select.....	404
表 3-557. 函数 pmu_lvd_disable	404
表 3-558. 函数 pmu_to_sleepmode	405
表 3-559. 函数 pmu_to_deepsleepmode	405
表 3-560. 函数 pmu_to_standbymode.....	406
表 3-561. 函数 pmu_wakeup_pin_enable	407
表 3-562. 函数 pmu_wakeup_pin_disable	407
表 3-563. 函数 pmu_backup_write_enable.....	408
表 3-564. 函数 pmu_backup_write_disable	408
表 3-565. 函数 pmu_flag_get	409
表 3-566. 函数 pmu_flag_clear.....	409

表 3-567. RCU 寄存器.....	410
表 3-568. RCU 库函数.....	411
表 3-569. 枚举类型 rcu_periph_enum.....	412
表 3-570. 枚举类型 rcu_periph_sleep_enum.....	413
表 3-571. 枚举类型 rcu_periph_reset_enum	413
表 3-572. 枚举类型 rcu_flag_enum	413
表 3-573. 枚举类型 rcu_int_flag_enum.....	414
表 3-574. 枚举类型 rcu_int_flag_clear_enum.....	415
表 3-575. 枚举类型 rcu_int_enum	415
表 3-576. 枚举类型 rcu_osci_type_enum.....	416
表 3-577. 枚举类型 rcu_clock_freq_enum	416
表 3-578. 函数 rcu_deinit.....	416
表 3-579. 函数 rcu_periph_clock_enable	417
表 3-580. 函数 rcu_periph_clock_disable	418
表 3-581. 函数 rcu_periph_clock_sleep_enable.....	419
表 3-582. 函数 rcu_periph_clock_sleep_disable.....	420
表 3-583. 函数 rcu_periph_reset_enable	420
表 3-584. 函数 rcu_periph_reset_disable	421
表 3-585. 函数 rcu_bkp_reset_enable	422
表 3-586. 函数 rcu_bkp_reset_disable	422
表 3-587. 函数 rcu_system_clock_source_config	423
表 3-588. 函数 rcu_system_clock_source_get.....	424
表 3-589. 函数 rcu_ahb_clock_config	424
表 3-590. 函数 rcu_apb1_clock_config	425
表 3-591. 函数 rcu_apb2_clock_config	425
表 3-592. 函数 rcu_ckout0_config	426
表 3-593. 函数 rcu_ckout1_config	427
表 3-594. 函数 rcu_pll_config.....	428
表 3-595. 函数 rcu_predv0_config	429
表 3-596. 函数 rcu_predv1_config	429
表 3-597. 函数 rcu_pll1_config.....	430
表 3-598. 函数 rcu_pll2_config.....	430
表 3-599. 函数 rcu_adc_clock_config	431
表 3-600. 函数 rcu_usbfs_trng_clock_config	432
表 3-601. 函数 rcu_rtc_clock_config	432
表 3-602. 函数 rcu_i2s1_clock_config	433
表 3-603. 函数 rcu_i2s2_clock_config	434
表 3-604. 函数 rcu_pllt_config.....	434
表 3-605. 函数 rcu_pllt_vco_config	435
表 3-606. 函数 rcu_tli_clock_config	435
表 3-607. 函数 rcu_lxtal_drive_capability_config	436
表 3-608. 函数 rcu_osci_stab_wait.....	437
表 3-609. 函数 rcu_osci_on.....	438
表 3-610. 函数 rcu_osci_off	438

表 3-611. 函数 rcu_osc_bypass_mode_enable	439
表 3-612. 函数 rcu_osc_bypass_mode_disable	439
表 3-613. 函数 rcu_hxtal_clock_monitor_enable	440
表 3-614. 函数 rcu_hxtal_clock_monitor_disable	440
表 3-615. 函数 rcu_irc8m_adjust_value_set	441
表 3-616. 函数 rcu_deepsleep_voltage_set	441
表 3-617. 函数 rcu_clock_freq_get	442
表 3-618. 函数 rcu_flag_get	443
表 3-619. 函数 rcu_all_reset_flag_clear	444
表 3-620. 函数 rcu_interrupt_enable	444
表 3-621. 函数 rcu_interrupt_disable	445
表 3-622. 函数 rcu_interrupt_flag_get	446
表 3-623. 函数 rcu_interrupt_flag_clear	447
表 3-624. RTC 寄存器	448
表 3-625. RTC 库函数	448
表 3-626. 函数 rtc_configuration_mode_enter	449
表 3-627. 函数 rtc_configuration_mode_exit	449
表 3-628. 函数 rtc_lwoff_wait	450
表 3-629. 函数 rtc_register_sync_wait	450
表 3-630. 函数 rtc_counter_get	451
表 3-631. 函数 rtc_counter_set	451
表 3-632. 函数 rtc_prescaler_set	452
表 3-633. 函数 rtc_alarm_config	452
表 3-634. 函数 rtc_divider_get	453
表 3-635. 函数 rtc_flag_get	453
表 3-636. 函数 rtc_flag_clear	454
表 3-637. 函数 rtc_interrupt_enable	455
表 3-638. 函数 rtc_interrupt_disable	455
表 3-639. SDIO 寄存器	456
表 3-640. SDIO 库函数	457
表 3-641. 函数 sdio_deinit	458
表 3-642. 函数 sdio_clock_config	458
表 3-643. 函数 sdio_hardware_clock_enable	459
表 3-644. 函数 sdio_hardware_clock_disable	460
表 3-645. 函数 sdio_bus_mode_set	460
表 3-646. 函数 sdio_power_state_set	461
表 3-647. 函数 sdio_power_state_get	461
表 3-648. 函数 sdio_clock_enable	462
表 3-649. 函数 sdio_clock_disable	462
表 3-650. 函数 sdio_command_response_config	463
表 3-651. 函数 sdio_wait_type_set	464
表 3-652. 函数 sdio_csm_enable	464
表 3-653. 函数 sdio_csm_disable	465
表 3-654. 函数 sdio_command_index_get	465

表 3-655. 函数 sdio_response_get	466
表 3-656. 函数 sdio_data_config	466
表 3-657. 函数 sdio_data_transfer_config	468
表 3-658. 函数 sdio_dsm_enable	468
表 3-659. 函数 sdio_dsm_disable	469
表 3-660. 函数 sdio_data_write	469
表 3-661. 函数 sdio_data_read	470
表 3-662. 函数 sdio_data_counter_get	470
表 3-663. 函数 sdio_data_counter_get	471
表 3-664. 函数 sdio_dma_enable	471
表 3-665. 函数 sdio_dma_disable	472
表 3-666. 函数 sdio_readwait_enable	472
表 3-667. 函数 sdio_readwait_disable	473
表 3-668. 函数 sdio_stop_readwait_enable	473
表 3-669. 函数 sdio_stop_readwait_disable	474
表 3-670. 函数 sdio_readwait_type_set	474
表 3-671. 函数 sdio_operation_enable	475
表 3-672. 函数 sdio_operation_disable	475
表 3-673. 函数 sdio_suspend_enable	476
表 3-674. 函数 sdio_suspend_disable	476
表 3-675. 函数 sdio_ceata_command_enable	477
表 3-676. 函数 sdio_ceata_command_disable	477
表 3-677. 函数 sdio_ceata_interrupt_enable	478
表 3-678. 函数 sdio_ceata_interrupt_disable	478
表 3-679. 函数 sdio_ceata_command_completion_enable	479
表 3-680. 函数 sdio_ceata_command_completion_disable	479
表 3-681. 函数 sdio_flag_get	480
表 3-682. 函数 sdio_flag_clear	481
表 3-683. 函数 sdio_interrupt_enable	482
表 3-684. 函数 sdio_interrupt_disable	484
表 3-685. 函数 sdio_interrupt_flag_get	485
表 3-686. 函数 sdio_interrupt_flag_clear	486
表 3-687. SPI/I2S 寄存器	488
表 3-688. SPI/I2S 库函数	488
表 3-689. 结构体 spi_parameter_struct	489
表 3-690. 函数 spi_i2s_deinit	490
表 3-691. 函数 spi_struct_para_init	490
表 3-692. 函数 spi_init	491
表 3-693. 函数 spi_enable	491
表 3-694. 函数 spi_disable	492
表 3-695. 函数 i2s_init	492
表 3-696. 函数 i2s_psc_config	494
表 3-697. 函数 i2s_enable	495
表 3-698. 函数 i2s_disable	495

表 3-699. 函数 spi_nss_output_enable	496
表 3-700. 函数 spi_nss_output_disable	496
表 3-701. 函数 spi_nss_internal_high	497
表 3-702. 函数 spi_nss_internal_low	497
表 3-703. 函数 spi_dma_enable	498
表 3-704. 函数 spi_dma_disable	499
表 3-705. 函数 spi_i2s_data_frame_format_config	499
表 3-706. 函数 spi_bidirectional_transfer_config	500
表 3-707. 函数 spi_i2s_data_transmit	501
表 3-708. 函数 spi_i2s_data_receive	501
表 3-709. 函数 spi_crc_polynomial_set	502
表 3-710. 函数 spi_crc_polynomial_get	502
表 3-711. 函数 spi_crc_on	503
表 3-712. 函数 spi_crc_off	503
表 3-713. 函数 spi_crc_next	504
表 3-714. 函数 spi_crc_get	504
表 3-715. 函数 spi_quad_enable	505
表 3-716. 函数 spi_quad_disable	506
表 3-717. 函数 spi_quad_write_enable	506
表 3-718. 函数 spi_quad_read_enable	507
表 3-719. 函数 spi_quad_io23_output_enable	507
表 3-720. 函数 spi_quad_io23_output_disable	508
表 3-721. 函数 spi_i2s_flag_get	508
表 3-722. 函数 spi_i2s_interrupt_enable	509
表 3-723. 函数 spi_i2s_interrupt_disable	510
表 3-724. 函数 spi_i2s_interrupt_flag_get	510
表 3-725. 函数 spi_crc_error_clear	511
表 3-726. TIMER 寄存器	512
表 3-727. TIMER 库函数	513
表 3-728. 结构体 timer_parameter_struct	515
表 3-729. 结构体 timer_break_parameter_struct	515
表 3-730. 结构体 timer_oc_parameter_struct	515
表 3-731. 结构体 timer_ic_parameter_struct	516
表 3-732. 函数 timer_deinit	516
表 3-733. 函数 timer_struct_para_init	516
表 3-734. 函数 timer_init	517
表 3-735. 函数 timer_enable	518
表 3-736. 函数 timer_disable	518
表 3-737. 函数 timer_auto_reload_shadow_enable	519
表 3-738. 函数 timer_auto_reload_shadow_disable	519
表 3-739. 函数 timer_update_event_enable	520
表 3-740. 函数 timer_update_event_disable	520
表 3-741. 函数 timer_counter_alignment	521
表 3-742. 函数 timer_counter_up_direction	522

表 3-743. 函数 timer_counter_down_direction.....	522
表 3-744. 函数 timer_prescaler_config	523
表 3-745. 函数 timer_repetition_value_config	523
表 3-746. 函数 timer_autoreload_value_config	524
表 3-747. 函数 timer_counter_value_config	524
表 3-748. 函数 timer_counter_read.....	525
表 3-749. 函数 timer_prescaler_read.....	526
表 3-750. 函数 timer_single_pulse_mode_config	526
表 3-751. 函数 timer_update_source_config	527
表 3-752. 函数 timer_dma_enable.....	527
表 3-753. 函数 timer_dma_disable.....	528
表 3-754. 函数 timer_channel_dma_request_source_select	529
表 3-755. 函数 timer_dma_transfer_config	530
表 3-756. 函数 timer_event_software_generate	531
表 3-757. 函数 timer_break_struct_para_init	532
表 3-758. 函数 timer_break_config	533
表 3-759. 函数 timer_break_enable	534
表 3-760. 函数 timer_break_disable	534
表 3-761. 函数 timer_automatic_output_enable.....	535
表 3-762. 函数 timer_automatic_output_disable.....	535
表 3-763. 函数 timer_primary_output_config	536
表 3-764. 函数 timer_channel_control_shadow_config.....	536
表 3-765. 函数 timer_channel_control_shadow_update_config	537
表 3-766. 函数 timer_channel_output_struct_para_init.....	538
表 3-767. 函数 timer_channel_output_config.....	538
表 3-768. 函数 timer_channel_output_mode_config.....	539
表 3-769. 函数 timer_channel_output_pulse_value_config	540
表 3-770. 函数 timer_channel_output_shadow_config.....	541
表 3-771. 函数 timer_channel_output_fast_config	542
表 3-772. 函数 timer_channel_output_clear_config	543
表 3-773. 函数 timer_channel_output_polarity_config	543
表 3-774. 函数 timer_channel_complementary_output_polarity_config	544
表 3-775. 函数 timer_channel_output_state_config	545
表 3-776. 函数 timer_channel_complementary_output_state_config	546
表 3-777. 函数 timer_channel_input_struct_para_init	547
表 3-778. 函数 timer_input_capture_config	547
表 3-779. 函数 timer_channel_input_capture_prescaler_config	548
表 3-780. 函数 timer_channel_capture_value_register_read.....	549
表 3-781. 函数 timer_input_pwm_capture_config	550
表 3-782. 函数 timer_hall_mode_config	551
表 3-783. 函数 timer_input_trigger_source_select.....	551
表 3-784. 函数 timer_master_output_trigger_source_select.....	552
表 3-785. 函数 timer_slave_mode_select	553
表 3-786. 函数 timer_master_slave_mode_config	554

表 3-787. 函数 timer_external_trigger_config	555
表 3-788. 函数 timer_quadrature_decoder_mode_config	556
表 3-789. 函数 timer_internal_clock_config.....	557
表 3-790. 函数 timer_internal_trigger_as_external_clock_config.....	557
表 3-791. 函数 timer_external_trigger_as_external_clock_config.....	558
表 3-792. 函数 timer_external_clock_mode0_config	559
表 3-793. 函数 timer_external_clock_mode1_config	560
表 3-794. 函数 timer_external_clock_mode1_disable	561
表 3-795. 函数 timer_flag_get	562
表 3-796. 函数 timer_flag_clear	563
表 3-797. 函数 timer_interrupt_enable	564
表 3-798. 函数 timer_interrupt_disable	564
表 3-799. 函数 timer_interrupt_flag_get	565
表 3-800. 函数 timer_interrupt_flag_clear	566
表 3-801. TLI 寄存器	567
表 3-802. TLI 库函数	568
表 3-803. 结构体 tli_parameter_struct.....	569
表 3-804. 结构体 tli_layer_parameter_struct.....	569
表 3-805. 结构体 tli_layer_lut_parameter_struct	570
表 3-806. 枚举类型 tli_layer_ppf_enum.....	570
表 3-807. 函数 tli_deinit	570
表 3-808. 函数 tli_struct_para_init	571
表 3-809. 函数 tli_init.....	571
表 3-810. 函数 tli_dither_config	572
表 3-811. 函数 tli_enable.....	573
表 3-812. 函数 tli_disable	573
表 3-813. 函数 tli_reload_config.....	574
表 3-814. 函数 tli_layer_struct_para_init	574
表 3-815. 函数 tli_layer_init.....	575
表 3-816. 函数 tli_layer_window_offset_modify.....	576
表 3-817. 函数 tli_lut_struct_para_init	577
表 3-818. 函数 tli_lut_init.....	578
表 3-819. 函数 tli_color_key_init	578
表 3-820. 函数 tli_layer_enable	579
表 3-821. 函数 tli_layer_disable	579
表 3-822. 函数 tli_color_key_enable.....	580
表 3-823. 函数 tli_color_key_disable.....	581
表 3-824. 函数 tli_lut_enable.....	581
表 3-825. 函数 tli_lut_disable.....	582
表 3-826. 函数 tli_line_mark_set	582
表 3-827. 函数 tli_current_pos_get.....	583
表 3-828. 函数 tli_interrupt_enable.....	583
表 3-829. 函数 tli_interrupt_disable.....	584
表 3-830. 函数 tli_interrupt_flag_get	584

表 3-831. 函数 tli_interrupt_flag_clear	585
表 3-832. 函数 tli_flag_get	585
表 3-833. TRNG 寄存器	586
表 3-834. TRNG 库函数	587
表 3-835. 枚举类型 trng_flag_enum	587
表 3-836. 枚举类型 trng_int_flag_enum	587
表 3-837. 函数 trng_deinit	587
表 3-838. 函数 trng_enable	588
表 3-839. 函数 trng_disable	588
表 3-840. 函数 trng_get_true_random_data	589
表 3-841. 函数 trng_flag_get	589
表 3-842. 函数 trng_interrupt_enable	590
表 3-843. 函数 trng_interrupt_disable	590
表 3-844. 函数 trng_interrupt_flag_get	591
表 3-845. 函数 trng_interrupt_flag_clear	591
表 3-846. USART 寄存器	592
表 3-847. USART 库函数	592
表 3-848. 枚举类型 usart_flag_enum	594
表 3-849. 枚举类型 usart_interrupt_flag_enum	594
表 3-850. 枚举类型 usart_interrupt_enum	595
表 3-851. 枚举类型 usart_invert_enum	595
表 3-852. 函数 usart_deinit	595
表 3-853. 函数 usart_baudrate_set	596
表 3-854. 函数 usart_parity_config	596
表 3-855. 函数 usart_word_length_set	597
表 3-856. 函数 usart_stop_bit_set	598
表 3-857. 函数 usart_enable	598
表 3-858. 函数 usart_disable	599
表 3-859. 函数 usart_transmit_config	599
表 3-860. 函数 usart_receive_config	600
表 3-861. 函数 usart_data_first_config	601
表 3-862. 函数 usart_invert_config	601
表 3-863. 函数 usart_receiver_timeout_enable	602
表 3-864. 函数 usart_receiver_timeout_disable	602
表 3-865. 函数 usart_receiver_timeout_threshold_config	603
表 3-866. 函数 usart_data_transmit	603
表 3-867. 函数 usart_data_receive	604
表 3-868. 函数 usart_address_config	605
表 3-869. 函数 usart_mute_mode_enable	605
表 3-870. 函数 usart_mute_mode_disable	606
表 3-871. 函数 usart_mute_mode_wakeup_config	606
表 3-872. 函数 usart_lin_mode_enable	607
表 3-873. 函数 usart_lin_mode_disable	607
表 3-874. 函数 usart_lin_break_dection_length_config	608

表 3-875. 函数 usart_send_break	609
表 3-876. 函数 usart_halfduplex_enable.....	609
表 3-877. 函数 usart_halfduplex_disable	610
表 3-878. 函数 usart_synchronous_clock_enable	610
表 3-879. 函数 usart_synchronous_clock_disable.....	611
表 3-880. 函数 usart_synchronous_clock_config.....	611
表 3-881. 函数 usart_guard_time_config.....	612
表 3-882. 函数 usart_smartcard_mode_enable	613
表 3-883. 函数 usart_smartcard_mode_disable	613
表 3-884. 函数 usart_smartcard_mode_nack_enable	614
表 3-885. 函数 usart_smartcard_mode_nack_disable	614
表 3-886. 函数 usart_smartcard_autoretry_config	615
表 3-887. 函数 usart_block_length_config.....	615
表 3-888. 函数 usart_irda_mode_enable	616
表 3-889. 函数 usart_irda_mode_disable	616
表 3-890. 函数 usart_prescaler_config	617
表 3-891. 函数 usart_irda_lowpower_config.....	617
表 3-892. 函数 usart_hardware_flow_rts_config.....	618
表 3-893. 函数 usart_hardware_flow_cts_config	619
表 3-894. 函数 usart_dma_receive_config	619
表 3-895. 函数 usart_dma_transmit_config	620
表 3-896. 函数 usart_flag_get	621
表 3-897. 函数 usart_flag_clear	621
表 3-898. 函数 usart_interrupt_enable	622
表 3-899. 函数 usart_interrupt_disable	623
表 3-900. 函数 usart_interrupt_flag_get	623
表 3-901. 函数 usart_interrupt_flag_clear	624
表 3-902. WWDGT 寄存器.....	625
表 3-903. WWDGT 库函数.....	625
表 3-904. 函数 wwdgt_deinit.....	625
表 3-905. 函数 wwdgt_enable	626
表 3-906. 函数 wwdgt_counter_update.....	626
表 3-907. 函数 wwdgt_config.....	627
表 3-908. 函数 wwdgt_flag_get.....	628
表 3-909. 函数 wwdgt_flag_clear	628
表 3-910. 函数 wwdgt_interrupt_enable	629
表 3-911. 函数 wwdgt_interrupt_flag_get	629
表 4-1. 版本历史	630

1. 介绍

本手册介绍了32位基于ARM微控制器GD32F20x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GD32F20x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
CAU	加密处理器
CRC	循环冗余校验计算单元
DAC	数模转换器
DBG	调试模块
DCI	数字摄像头接口
DMA	直接存储器访问控制器
ENET	以太网控制器模块

外设缩写	说明
EXMC	外部存储器控制器
EXTI	外部中断事件控制器
FMC	闪存控制器
FWDGT	独立看门狗
GPIO/AFIO	通用和备用输入/输出接口
HAU	哈希处理器
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
TLI	TFT-LCD接口
TRNG	真随机数生成器
USART	通用同步异步收发器
WWDGT	窗口看门狗
USBFS	通用串行总线全速接口

1.1.2. 命名规则

固件库遵从以下命名规则：

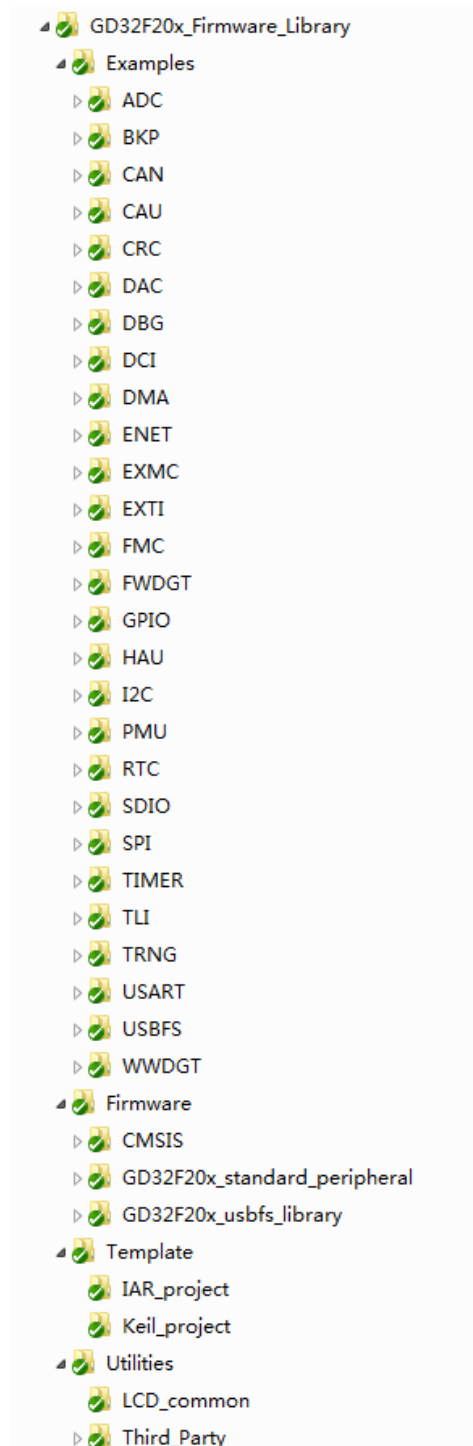
- XXX表示任一外设缩写，例如：ADC。更多缩写相关信息参阅[外设缩写](#)；
- 源文件和头文件命名都以“gd32f20x_”作为开头，例如：gd32f20x_adc.h；
- 常量仅被应用于一个文件的，定义于该文件中；被应用于多个文件的，在对应头文件中定义。所有常量都由英文字母大写书写；
- 寄存器作为常量处理。他们的命名都由英文字母大写书写。在大多数情况下，寄存器缩写规范与本用户手册一致；
- 变量名采用全部小写，有多个单词组成的，在单词之间以下划线分隔；
- 外设函数的命名以该外设的缩写加下划线为开头，有多个单词组成的，在单词之间以下划线分隔，所有外设函数都由英文字母小写书写。

2. 固件库概述

2.1. 文件组织结构

GD32F20x_Firmware_Library，文件组织结构见下图：

图 2-1. GD32F20x 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；
- **gd32f20x_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）；
- **gd32f20x_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）；
- **gd32f20x_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GD32F20x的全局头文件和系统配置文件；
- **GD32F20x_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；
- **GD32F20x_usbfs_driver**子文件夹包含了关于**USBFS**外设的所有文件，用户无需修改该文件夹；

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

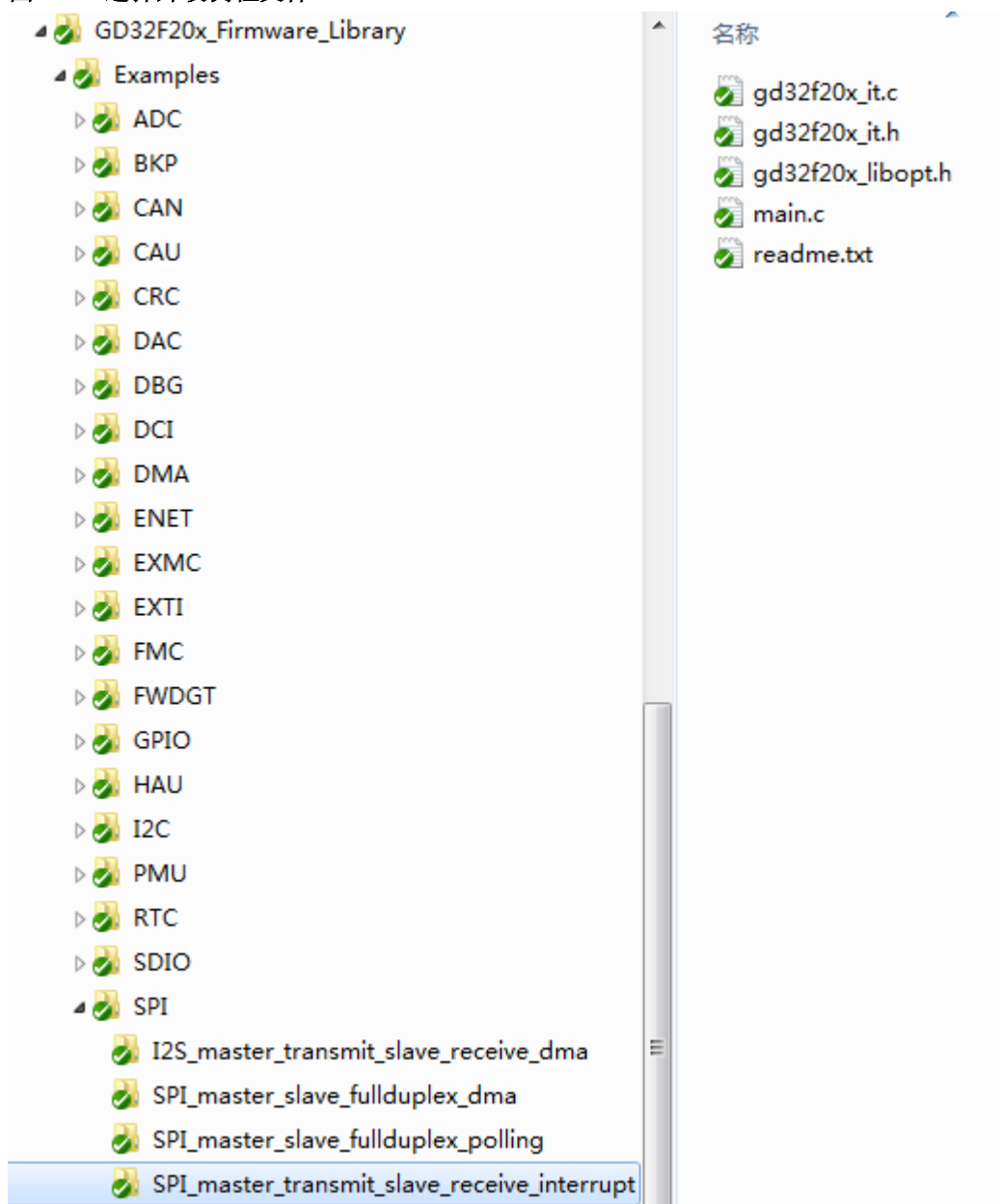
2.1.3. Template 文件夹

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

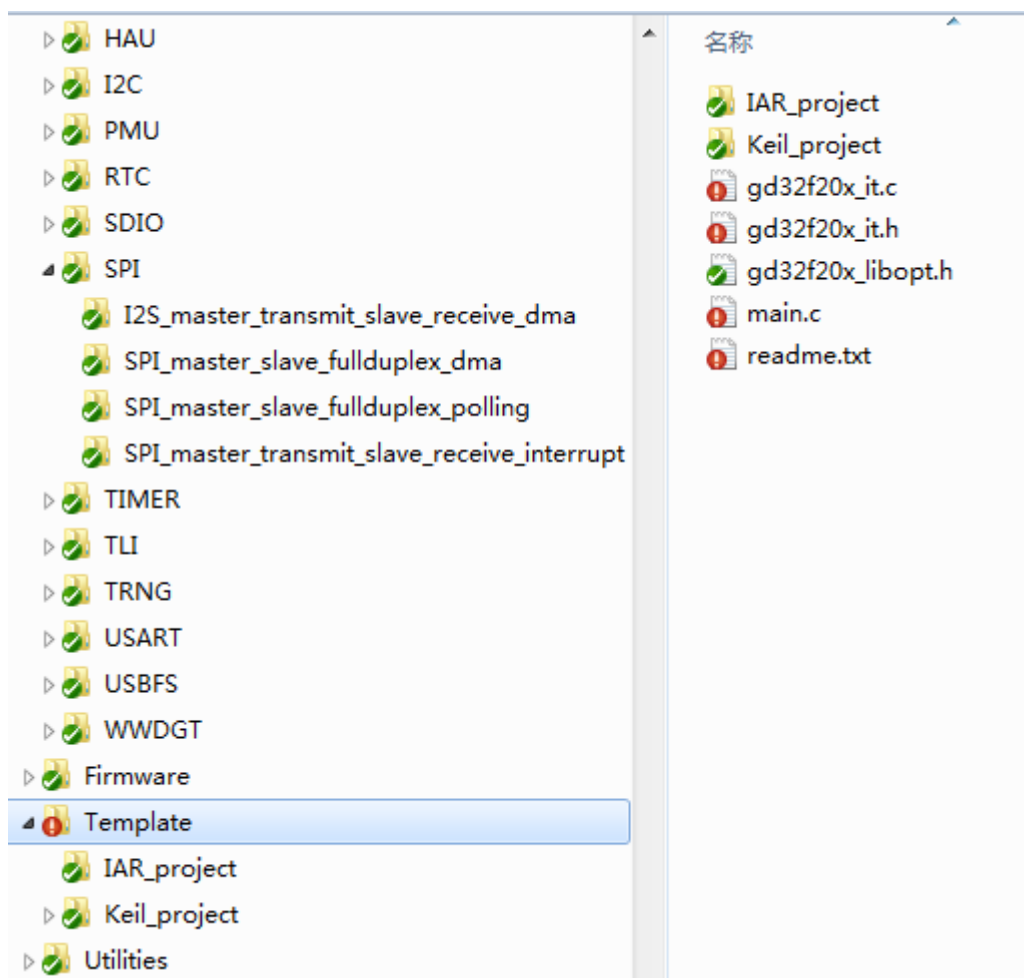
图 2-2. 选择外设例程文件



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：

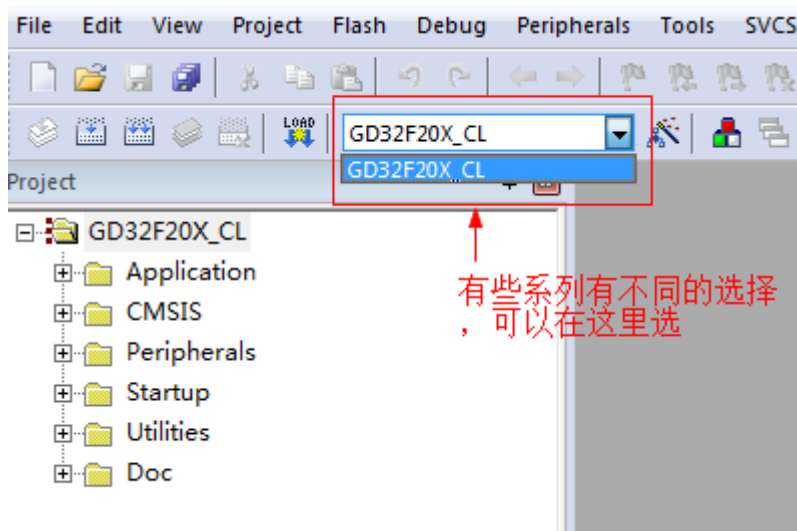
图 2-3. 拷贝外设例程文件



打开工程

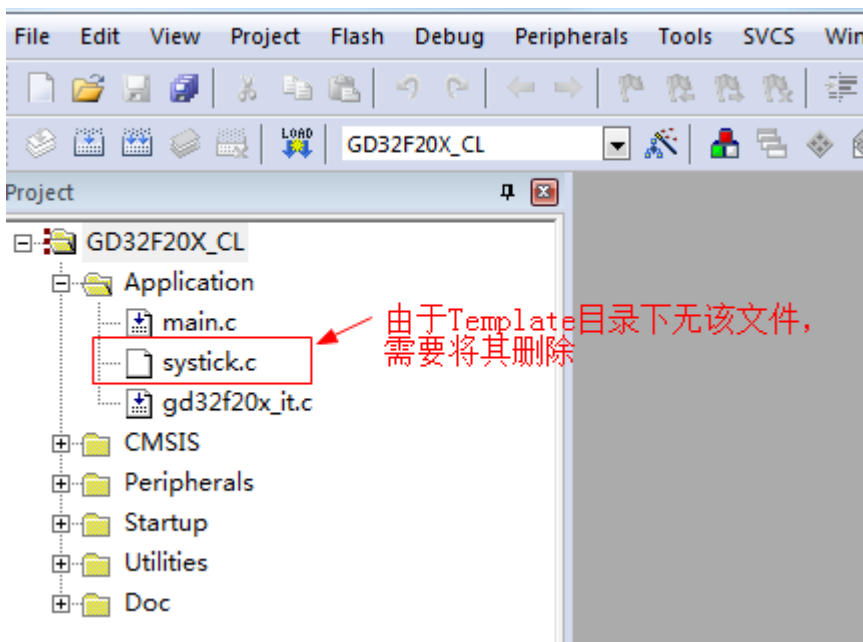
GD提供Keil和IAR两种版本的工程，根据客户所安装的软件，打开不同的project，如“Keil_project”，打开\Template\Keil_project\Project.uvproj，如下图所示：

图 2-4. 打开工程文件



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

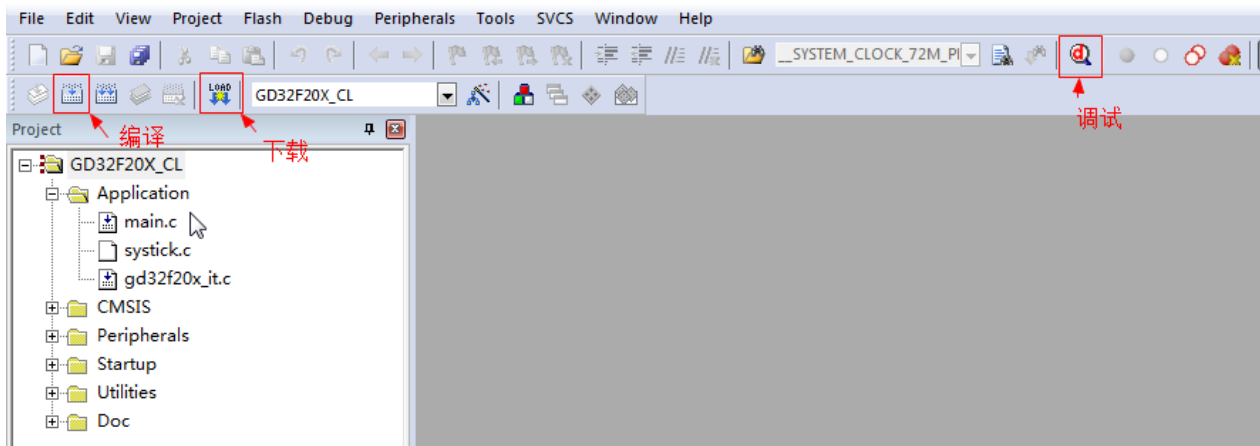
图 2-5. 配置工程文件



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：

- LCD_Commom及Third_Party子文件夹包含有USB测试所需文件；

- gd32f20x_eval.h及gd32f20x_lcd_eval.h文件是运行固件库例程所需关于评估板的头文件；
- gd32f20x_eval.c及gd32f20x_lcd_eval.c文件是运行固件库例程所需关于评估板的源文件。

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

2.2. 固件库文件描述

下表列举和描述了固件库使用的主要文件。

表 2-1. 固件函数库文件描述

文件名	描述
gd32f20x_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。
gd32f20x_it.h	头文件，包含所有中断处理函数原形。
gd32f20x_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。
gd32f20x_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。
gd32f20x_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

寄存器名称	寄存器描述
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
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
ADC_OVSAMPCTL	过采样控制寄存器

3.2.2. 外设库函数说明

ADC库函数列表如下表所示：

表 3-3. ADC 库函数

库函数名称	库函数描述
adc_deinit	复位ADCx外设
adc_mode_config	配置ADCx模式(仅限ADC0)
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	ADCx 禁能
adc_disable	ADCx 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	ADCx DMA请求使能
adc_dma_mode_disable	ADCx 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

库函数名称	库函数描述
e	
adc_watchdog_disable	ADC模拟看门狗禁能
adc_watchdog_threshold_config	配置ADC模拟看门狗阈值
adc_resolution_config	配置ADC分辨率
adc_oversample_mode_config	配置ADC过采样模式
adc_oversample_mode_enable	使能ADC过采样模式
adc_oversample_mode_disable	除能ADC过采样模式
adc_flag_get	获取ADC中断标志位
adc_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
adc_interrupt_flag_get	获取ADC中断标志位
adc_interrupt_flag_clear	清除ADC中断标志位

函数 adc_deinit

函数adc_deinit描述见下表：

表 3-4. 函数 adc_deinit

函数名称	adc_deinit
函数原形	void adc_deinit(uint32_t adc_periph);
功能描述	复位ADCx外设
先决条件	-
被调用函数	rcu_periph_reset_enable / rcu_periph_reset_disable
输入参数{in}	
adc_periph	ADC外设
ADCx	x=0,1,2
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_PARALLEL_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	adc_special_function_config
函数原形	void adc_special_function_config(uint32_t adc_periph, uint32_t function, ControlStatus new_value);
功能描述	使能或除能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}	
newvalue	功能使能/禁能
ENABLE	使能
DISABLE	除能
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_data_alignment_config描述见下表：

表 3-7. 函数 adc_data_alignment_config

函数名称	adc_data_alignment_config
函数原形	void adc_data_alignment_config(uint32_t adc_periph, uint32_t data_alignment);
功能描述	配置ADCx数据对齐方式
先决条件	-

被调用函数	-
输入参数{in}	
adc_periph	ADC外设
ADCx	x=0,1,2
输入参数{in}	
data_alignment	数据对齐方式选择
ADC_DATAALIGN_	LSB 对齐
RIGHT	
ADC_DATAALIGN_	MSB 对齐
LEFT	
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* configure ADC0 data alignment */
```

```
adc_data_alignment_config(ADC0, ADC_DATAALIGN_RIGHT);
```

函数 adc_enable

函数adc_enable描述见下表：

表 3-8. 函数 adc_enable

函数名称	adc_enable
函数原形	void adc_enable(uint32_t adc_periph);
功能描述	使能ADCx外设
先决条件	-
被调用函数	-
输入参数{in}	
adc_periph	ADC外设
ADCx	x=0,1,2
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	除能ADCx外设
先决条件	-
被调用函数	-
输入参数{in}	
adc_periph	ADC外设
ADCx	x=0,1,2
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	ADCx校准复位
先决条件	-
被调用函数	-
输入参数{in}	
adc_periph	ADC外设
ADCx	x=0,1,2
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	ADCx DMA请求使能
先决条件	-
被调用函数	-
输入参数{in}	
adc_periph	ADC外设
ADCx	x=0,1,2
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	ADCx DMA请求禁能
先决条件	-
被调用函数	-
输入参数{in}	
adc_periph	ADC外设
ADCx	x=0,1,2
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_DISCONTINUOUS_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 _41POINT5	41.5 周期
ADC_SAMPLETIME _55POINT5	55.5 周期
ADC_SAMPLETIME _71POINT5	71.5 周期
ADC_SAMPLETIME _239POINT5	239.5 周期
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* configure ADC0 regular channel */
```

```
adc_regular_channel_config(ADC0, 1, ADC_CHANNEL_0, ADC_SAMPLETIME_7POINT5);
```

函数 adc_inserted_channel_config

函数adc_inserted_channel_config描述见下表：

表 3-18. 函数 adc_inserted_channel_config

函数名称	adc_inserted_channel_config
函数原形	void adc_inserted_channel_config(uint32_t adc_periph, uint8_t rank, uint8_t adc_channel, uint32_t sample_time);
功能描述	配置ADC注入通道组
先决条件	-
被调用函数	-
输入参数{in}	
adc_periph	ADC外设
ADCx	x=0,1,2
输入参数{in}	
rank	注入组通道序列，取值范围为0~3
输入参数{in}	
adc_channel	ADC通道选择

ADC_CHANNEL_x	ADC 通道x (x=0..17)(只有ADC0, 可取值x=16和17)
输入参数{in}	
sample_time	采样时间
ADC_SAMPLETIME_1POINT5	1.5 周期
ADC_SAMPLETIME_7POINT5	7.5 周期
ADC_SAMPLETIME_13POINT5	13.5 周期
ADC_SAMPLETIME_28POINT5	28.5 周期
ADC_SAMPLETIME_41POINT5	41.5 周期
ADC_SAMPLETIME_55POINT5	55.5 周期
ADC_SAMPLETIME_71POINT5	71.5 周期
ADC_SAMPLETIME_239POINT5	239.5 周期
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* configure ADC0 inserted channel */
```

```
adc_inserted_channel_config(ADC0, 1, ADC_CHANNEL_0, ADC_SAMPLETIME_7POINT5);
```

函数 adc_inserted_channel_offset_config

函数adc_inserted_channel_offset_config描述见下表:

表 3-19. 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	adc_external_trigger_source_config
函数原形	void adc_external_trigger_source_config(uint32_t adc_periph, uint8_t adc_channel_group, uint32_t external_trigger_source);
功能描述	配置ADC外部触发源
先决条件	-
被调用函数	-
输入参数{in}	
adc_periph	ADC外设
ADCx	x=0,1,2
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)

1	
ADC0_1_EXTTRIG _REGULAR_T0_CH 2	TIMER0 CH2事件（规则组）
ADC0_1_EXTTRIG _REGULAR_T1_CH 1	TIMER1 CH1事件（规则组）
ADC0_1_EXTTRIG _REGULAR_T2_TR GO	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_TRGO	TIMER1 TRGO事件（注入组）
ADC0_1_EXTTRIG_INSERTED_T1_CH0	TIMER1 CH0事件（注入组）
ADC0_1_EXTTRIG_INSERTED_T2_CH3	TIMER2 CH3事件（注入组）
ADC0_1_EXTTRIG_INSERTED_T3_TRGO	TIMER3 TRGO事件（注入组）
ADC0_1_EXTTRIG_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_NON	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	<code>adc_external_trigger_config</code>
函数原形	<code>void adc_external_trigger_config(uint32_t adc_periph, uint8_t adc_channel_group, ControlStatus newvalue);</code>
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}	
<code>adc_periph</code>	ADC外设
<code>ADCx</code>	$x=0,1,2$
输入参数{in}	
<code>adc_channel_group</code>	通道组选择
<code>ADC_REGULAR_CHANNEL</code>	规则通道组
<code>ADC_INSERTED_CHANNEL</code>	注入通道组
输入参数{in}	
<code>newvalue</code>	通道使能禁能
<code>ENABLE</code>	使能
<code>DISABLE</code>	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	<code>adc_software_trigger_enable</code>
------	--

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
ADC_REGULAR_CHANNEL	规则通道组
ADC_INSERTED_CHANNEL	注入通道组
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* enable ADC0 regular channel group software trigger */
```

```
adc_software_trigger_enable(ADC0, ADC_REGULAR_CHANNEL);
```

函数 adc_regular_data_read

函数adc_inserted_regular_data_read描述见下表：

表 3-23. 函数 adc_regular_data_read

函数名称	adc_regular_data_read
函数原形	uint16_t adc_regular_data_read(uint32_t adc_periph);
功能描述	读ADC规则组数据寄存器
先决条件	-
被调用函数	-
输入参数{in}	
adc_periph	ADC外设
ADCx	x=0,1,2
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	
uint32_t	ADC转换值 (0-0xFFFFFFFF)

例如:

```
/* read the last ADC0 and ADC1 conversion result data in sync mode */
```

```
uint32_t adc_value = 0;
```

```
adc_value = adc_sync_mode_convert_value_read ();
```

函数 adc_watchdog_single_channel_enable

函数adc_watchdog_single_channel_enable描述见下表:

表 3-26. 函数 adc_watchdog_single_channel_enable

函数名称	adc_watchdog_single_channel_enable
函数原形	void adc_watchdog_single_channel_enable(uint32_t adc_periph, uint8_t adc_channel);
功能描述	配置ADC模拟看门狗单通道有效
先决条件	-
被调用函数	-
输入参数{in}	
adc_periph	ADC外设
ADCx	x=0,1,2
输入参数{in}	
adc_channel	选择ADC通道
ADC_CHANNEL_x	ADC Channelx(x=0...17) (只有ADC0, 可取值x=16和17)
输出参数{out}	
-	-
返回值	
-	-

例如:

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

```
adc_watchdog_single_channel_enable(ADC0, ADC_CHANNEL_1);
```

函数 adc_watchdog_group_channel_enable

函数adc_watchdog_group_channel_enable描述见下表:

表 3-27. 函数 adc_watchdog_group_channel_enable

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

函数原形	void adc_watchdog_group_channel_enable(uint32_t adc_periph, uint8_t adc_channel_group);
功能描述	配置ADC模拟看门狗在通道组有效
先决条件	-
被调用函数	-
输入参数{in}	
adc_periph	ADC外设
ADCx	x=0,1,2
输入参数{in}	
adc_channel_group	通道组使用模拟看门狗
ADC_REGULAR_CHANNEL	规则通道组
ADC_INSERTED_CHANNEL	注入通道组
ADC_REGULAR_INSERTED_CHANNEL	规则和注入通道组
输出参数{out}	
-	-
返回值	
-	-

例如：

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

```
adc_watchdog_group_channel_enable(ADC0, ADC_REGULAR_CHANNEL);
```

函数 adc_watchdog_disable

函数adc_watchdog_disable描述见下表：

表 3-28. 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
ADCx	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>	x=0,1,2
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_resolution_config`

函数`adc_resolution_config`描述见下表：

表 3-30. 函数 `adc_resolution_config`

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

adc_periph	ADC外设
<i>ADCx</i>	<i>x=0,1,2</i>
输入参数{in}	
resolution	ADC分辨率
<i>ADC_RESOLUTION_12B</i>	12位ADC分辨率
<i>ADC_RESOLUTION_10B</i>	10位ADC分辨率
<i>ADC_RESOLUTION_8B</i>	8位ADC分辨率
<i>ADC_RESOLUTION_6B</i>	6位ADC分辨率
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* config ADC0 resolution */
adc_resolution_config(ADC0, ADC_RESOLUTION_10B);
```

函数 `adc_oversample_mode_config`

函数`adc_oversample_mode_config`描述见下表：

表3-31. 函数`adc_oversample_mode_config`

函数名称	<code>adc_oversample_mode_config</code>
函数原形	<code>void adc_oversample_mode_config(uint32_t adc_periph, uint8_t mode, uint16_t shift, uint8_t ratio);</code>
功能描述	配置ADC过采样模式
先决条件	-
被调用函数	-
输入参数{in}	
adc_periph	ADC外设
<i>ADCx</i>	<i>x=0,1,2</i>
输入参数{in}	
mode	ADC过采样模式
<i>ADC_OVERSAMPLING_ALL_CONVERT</i>	触发后持续地对信道进行过采样转换
<i>ADC_OVERSAMPLING_ONE_CONVERT</i>	信道的每次过采样转换均需触发

输入参数{in}	
shift	ADC过采样移位
ADC_OVERSAMPLING_SHIFT_NONE	没有过采样移位
ADC_OVERSAMPLING_SHIFT_1B	1位过采样移位
ADC_OVERSAMPLING_SHIFT_2B	2位过采样移位
ADC_OVERSAMPLING_SHIFT_3B	3位过采样移位
ADC_OVERSAMPLING_SHIFT_4B	4位过采样移位
ADC_OVERSAMPLING_SHIFT_5B	5位过采样移位
ADC_OVERSAMPLING_SHIFT_6B	6位过采样移位
ADC_OVERSAMPLING_SHIFT_7B	7位过采样移位
ADC_OVERSAMPLING_SHIFT_8B	8位过采样移位
输入参数{in}	
ratio	ADC过采样率
ADC_OVERSAMPLING_RATIO_MUL2	2倍过采样率
ADC_OVERSAMPLING_RATIO_MUL4	4倍过采样率
ADC_OVERSAMPLING_RATIO_MUL8	8倍过采样率
ADC_OVERSAMPLING_RATIO_MUL16	16倍过采样率
ADC_OVERSAMPLING_RATIO_MUL32	32倍过采样率
ADC_OVERSAMPLING_RATIO_MUL64	64倍过采样率
ADC_OVERSAMPLING_RATIO_MUL128	128倍过采样率
ADC_OVERSAMPLING_RATIO_MUL256	256倍过采样率
输出参数{out}	
-	-
返回值	

-	-
---	---

例如：

```
/* config ADC0 oversample mode */

adc_oversample_mode_config (ADC0,      ADC_OVERSAMPLING_ALL_CONVERT,
ADC_OVERSAMPLING_SHIFT_2B, ADC_OVERSAMPLING_RATIO_MUL4);
```

函数 `adc_oversample_mode_enable`

函数`adc_oversample_mode_enable`描述见下表：

表 3-32. 函数 `adc_oversample_mode_enable`

函数名称	<code>adc_oversample_mode_enable</code>
函数原形	<code>void adc_oversample_mode_enable(uint32_t adc_periph)</code>
功能描述	使能ADC过采样模式
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
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* enable ADC0 oversample mode */

adc_oversample_mode_enable(ADC0);
```

函数 `adc_oversample_mode_disable`

函数`adc_oversample_mode_disable`描述见下表：

表 3-33. 函数 `adc_oversample_mode_disable`

函数名称	<code>adc_oversample_mode_disable</code>
函数原形	<code>void adc_oversample_mode_disable(uint32_t adc_periph)</code>
功能描述	除能ADC过采样模式
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
输出参数{out}	
-	-

返回值	
-	-

例如：

```
/* disable ADC0 oversample mode */
adc_oversample_mode_disable(ADC0);
```

函数 adc_flag_get

函数adc_flag_get描述见下表：

表 3-34. 函数 adc_flag_get

函数名称	adc_flag_get
函数原形	FlagStatus adc_flag_get(uint32_t adc_periph , uint32_t adc_flag);
功能描述	获取ADC标志位
先决条件	-
被调用函数	-
输入参数{in}	
adc_periph	ADC外设
ADCx	x=0,1,2
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	
FlagStatus	SET 或 RESET

例如：

```
/* get the ADC0 analog watchdog flag bits */
FlagStatus flag_value;
flag_value = adc_flag_get(ADC0, ADC_FLAG_WDE);
```

函数 adc_flag_clear

函数adc_flag_clear描述见下表：

表 3-35. 函数 adc_flag_clear

函数名称	adc_flag_clear
------	----------------

函数原形	void adc_flag_clear(uint32_t adc_periph, uint32_t adc_flag);
功能描述	清除ADC标志位
先决条件	-
被调用函数	-
输入参数{in}	
adc_periph	ADC外设
ADCx	x=0,1,2
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_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中断标志位
ADC_INT_WDE	模拟看门狗中断标志位
ADC_INT_EOC	组转换结束中断标志位
ADC_INT_EOIC	注入通道组转换结束中断标志位
输出参数{out}	
-	-

返回值	
-	-

例如：

```
/* enable ADC0 analog watchdog interrupt */
adc_interrupt_enable(ADC0, ADC_INT_WDE);
```

函数 adc_interrupt_disable

函数adc_interrupt_disable描述见下表：

表 3-37. 函数 adc_interrupt_disable

函数名称	adc_interrupt_disable
函数原形	void adc_interrupt_enable(uint32_t adc_periph, uint32_t adc_interrupt);
功能描述	ADC中断禁能
先决条件	-
被调用函数	-
输入参数{in}	
adc_periph	ADC外设
ADCx	x=0,1,2
输入参数{in}	
adc_interrupt	ADC中断标志位
ADC_INT_WDE	模拟看门狗中断标志位
ADC_INT_EOC	组转换结束中断标志位
ADC_INT_EOIC	注入通道组转换结束中断标志位
输出参数{out}	
-	-
返回值	
-	-

例如：

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

函数 adc_interrupt_flag_get

函数adc_interrupt_flag_get描述见下表：

表 3-38. 函数 adc_interrupt_flag_get

函数名称	adc_interrupt_flag_get
函数原形	FlagStatus adc_interrupt_flag_get(uint32_t adc_periph, uint32_t adc_interrupt);
功能描述	获取ADC中断标志位
先决条件	-
被调用函数	-

输入参数{in}	
adc_periph	ADC外设
<i>ADCx</i>	x=0,1,2
输入参数{in}	
adc_interrupt	ADC中断标志位
<i>ADC_INT_WDE</i>	模拟看门狗中断标志位
<i>ADC_INT_EOC</i>	组转换结束中断标志位
<i>ADC_INT_EOIC</i>	注入通道组转换结束中断标志位
输出参数{out}	
-	-
返回值	
FlagStatus	SET 或 RESET

例如：

```
/* get the ADC0 analog watchdog interrupt bits*/
```

```
FlagStatus flag_value;
```

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

函数 adc_interrupt_flag_clear

函数adc_interrupt_flag_clear描述见下表：

表 3-39. 函数 adc_interrupt_flag_clear

函数名称	adc_interrupt_flag_clear
函数原形	void adc_interrupt_flag_clear(uint32_t adc_periph, uint32_t adc_interrupt);
功能描述	清除ADC中断标志位
先决条件	-
被调用函数	-
输入参数{in}	
adc_periph	ADC外设
<i>ADCx</i>	x=0,1,2
输入参数{in}	
adc_interrupt	ADC中断标志位
<i>ADC_INT_WDE</i>	模拟看门狗中断标志位
<i>ADC_INT_EOC</i>	组转换结束中断标志位
<i>ADC_INT_EOIC</i>	注入通道组转换结束中断标志位
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* clear the ADC0 analog watchdog interrupt bits*/
```

```
adc_interrupt_flag_clear(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 3-40. BKP 寄存器

寄存器名称	寄存器描述
BKP_DATAx (x=0..41)	备份数据寄存器
BKP_OCTL	RTC信号输出控制寄存器
BKP_TPCTL0	侵入引脚控制寄存器0
BKP_TPCS	侵入控制状态寄存器
BKP_TPCTL1	侵入引脚控制寄存器1

3.3.2. 外设库函数说明

BKP库函数列表如下表所示：

表 3-41. BKP 库函数

库函数名称	库函数描述
bkp_deinit	复位BKP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时钟校准输出禁能
bkp_rtc_signal_output_enable	RTC闹钟或秒信号输出使能
bkp_rtc_signal_output_disable	RTC闹钟或秒信号输出禁能
bkp_rtc_output_select	RTC输出选择
bkp_rtc_clock_output_select	RTC时钟输出选择
bkp_rtc_clock_calibration_direction	RTC时钟校准方向
bkp_rtc_calibration_value_set	RTC时钟校准值
bkp_tamper_detection_enable	TAMPER引脚使能
bkp_tamper_detection_disable	TAMPER引脚禁能
bkp_tamper_active_level_set	TAMPER引脚有效电平设置
bkp_waveform_detect_config	方波检测配置
bkp_flag_get	获取BKP标志位

库函数名称	库函数描述
bkp_flag_clear	清除BKP标志位
bkp_tamper_interrupt_enable	TAMPER中断使能
bkp_tamper_interrupt_disable	TAMPER中断禁能
bkp_interrupt_flag_get	获取BKP中断标志位
bkp_interrupt_flag_clear	清除BKP中断标志位

枚举类型 `bkp_data_register_enum`

表 3-42. 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
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

成员名称	功能描述
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

枚举类型 `bkp_tamper_enum`

表 3-43. 枚举类型 `bkp_tamper_enum`

成员名称	功能描述
TAMPER_0	BKP侵入引脚0
TAMPER_1	BKP侵入引脚1

函数 `bkp_deinit`

函数`bkp_deinit`描述见下表：

表 3-44. 函数 `bkp_deinit`

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

例如：

```
/* reset BKP registers */
bkp_deinit();
```

函数 `bkp_data_write`

函数`bkp_data_write`描述见下表：

表 3-45. 函数 bkp_data_write

函数名称	bkp_data_write
函数原型	void bkp_data_write(bkp_data_register_enum register_number, uint16_t data);
功能描述	写备份数据寄存器
先决条件	-
被调用函数	-
输入参数{in}	
register_number	参考枚举 表3-42. 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
0x0000-0xFFFF	数值
输出参数{out}	
-	-
返回值	
-	-

例如：

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

函数 bkp_data_read

函数bkp_data_read描述见下表：

表 3-46. 函数 bkp_data_read

函数名称	bkp_data_read
函数原型	uint16_t bkp_data_read(bkp_data_register_enum register_number);
功能描述	读备份数据寄存器
先决条件	-
被调用函数	-
输入参数{in}	
register_number	参考枚举 表3-42. 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	备份数据寄存器的数值(0x0000-0xFFFF)

例如：

```
/* read BKP data register */
```

```
uint16_t data;
```

```
data = bkp_data_read(BKP_DATA_0);
```

函数 bkp_rtc_calibration_output_enable

函数bkp_rtc_calibration_output_enable描述见下表：

表 3-47. 函数 bkp_rtc_calibration_output_enable

函数名称	bkp_rtc_calibration_output_enable
函数原型	void bkp_rtc_calibration_output_enable(void);
功能描述	RTC时钟校准输出使能
先决条件	-
被调用函数	-
输入参数{in}	
-	-
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* enable RTC clock calibration output */
```

```
bkp_rtc_calibration_output_enable();
```

函数 bkp_rtc_calibration_output_disable

函数bkp_rtc_calibration_output_disable描述见下表：

表 3-48. 函数 bkp_rtc_calibration_output_disable

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

```
bkp_rtc_calibration_output_disable();
```

函数 bkp_rtc_signal_output_enable

函数bkp_rtc_signal_output_enable描述见下表：

表 3-49. 函数 bkp_rtc_signal_output_enable

函数名称	bkp_rtc_signal_output_enable
函数原型	void bkp_rtc_signal_output_enable (void);
功能描述	RTC闹钟或秒信号输出使能
先决条件	-
被调用函数	-
输入参数{in}	
-	-
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* enable RTC alarm or second signal output */
```

```
bkp_rtc_signal_output_enable();
```

函数 bkp_rtc_signal_output_disable

函数bkp_rtc_signal_output_disable描述见下表：

表 3-50. 函数 bkp_rtc_signal_output_disable

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

```
bkp_rtc_signal_output_disable();
```

函数 bkp_rtc_output_select

函数bkp_rtc_output_select描述见下表：

表 3-51. 函数 bkp_rtc_output_select

函数名称	bkp_rtc_output_select
函数原型	void bkp_rtc_output_select (uint16_t outputsel);
功能描述	RTC输出选择
先决条件	-
被调用函数	-
输入参数{in}	
outputsel	RTC输出选择
RTC_OUTPUT_AL ARM_PULSE	RTC闹钟脉冲被选择为RTC输出
RTC_OUTPUT_SE COND_PULSE	RTC秒脉冲被选择为RTC输出
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* select RTC alarm pulse as the RTC output */
```

```
bkp_rtc_output_select(RTC_OUTPUT_ALARM_PULSE);
```

函数 bkp_rtc_clock_output_select

函数bkp_rtc_clock_output_select描述见下表：

表 3-52. 函数 bkp_rtc_clock_output_select

函数名称	bkp_rtc_clock_output_select
函数原型	void bkp_rtc_clock_output_select(uint16_t clocksel)
功能描述	RTC时钟输出选择
先决条件	-
被调用函数	-
输入参数{in}	
clocksel	RTC时钟输出选择
RTC_CLOCK_DIV6 4	RTC时钟64分频
RTC_CLOCK_DIV1	RTC时钟不分频
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* RTC clock output selection */
```

```
bkp_rtc_clock_output_select(RTC_CLOCK_DIV1);
```

函数 bkp_rtc_clock_calibration_direction

函数bkp_rtc_clock_calibration_direction描述见下表：

表 3-53. 函数 bkp_rtc_clock_calibration_direction

函数名称	bkp_rtc_clock_calibration_direction
函数原型	void bkp_rtc_clock_calibration_direction(uint16_t direction);
功能描述	RTC时钟校准方向
先决条件	-
被调用函数	-
输入参数{in}	
direction	RTC时钟校方向
RTC_CLOCK_SLOW_DOWN	变慢
RTC_CLOCK_SPEED_UP	变快
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* set RTC clock speed up */
```

```
bkp_rtc_clock_calibration_direction(RTC_CLOCK_SPEED_UP);
```

函数 bkp_rtc_calibration_value_set

函数bkp_rtc_calibration_value_set描述见下表：

表 3-54. 函数 bkp_rtc_calibration_value_set

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

-	-
返回值	
-	-

例如：

```
/* set RTC clock calibration value */
```

```
bkp_rtc_calibration_value_set(0x7F);
```

函数 bkp_tamper_detection_enable

函数bkp_tamper_detection_enable描述见下表：

表 3-55. 函数 bkp_tamper_detection_enable

函数名称	bkp_tamper_detection_enable
函数原型	void bkp_tamper_detection_enable(bkp_tamper_enum tamperx);
功能描述	TAMPER引脚使能
先决条件	-
被调用函数	-
输入参数{in}	
tamperx	参考枚举 表3-43. 枚举类型bkp_tamper_enum
TAMPER_0	BKP tamper0
TAMPER_1	BKP tamper1
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* enable tamper0 pin detection */
```

```
bkp_tamper_detection_enable(TAMPER_0);
```

函数 bkp_tamper_detection_disable

函数bkp_tamper_detection_disable描述见下表：

表 3-56. 函数 bkp_tamper_detection_disable

函数名称	bkp_tamper_detection_disable
函数原型	void bkp_tamper_detection_disable(bkp_tamper_enum tamperx);
功能描述	TAMPER引脚禁能
先决条件	-
被调用函数	-
输入参数{in}	
tamperx	参考枚举 表3-43. 枚举类型bkp_tamper_enum

TAMPER_0	BKP tamper0
TAMPER_1	BKP tamper1
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* disable tamper0 pin detection */
```

```
bkp_tamper_detection_disable(TAMPER_0);
```

函数 bkp_tamper_active_level_set

函数bkp_tamper_active_level_set描述见下表：

表 3-57. 函数 bkp_tamper_active_level_set

函数名称	bkp_tamper_active_level_set
函数原型	void bkp_tamper_active_level_set(bkp_tamper_enum tamperx, 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}	
tamperx	参考枚举 表3-43. 枚举类型bkp_tamper_enum
TAMPER_0	BKP tamper0
TAMPER_1	BKP tamper1
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 tamper0 pin active level high */
```

```
bkp_tamper_active_level_set(TAMPER_0, TAMPER_PIN_ACTIVE_HIGH);
```

函数 bkp_waveform_detect_config

函数bkp_waveform_detect_config描述见下表：

表 3-58. 函数 bkp_waveform_detect_config

函数名称	bkp_waveform_detect_config
函数原型	void bkp_waveform_detect_config(uint16_t waveform_detect_mode, ControlStatus newvalue)
功能描述	方波检测配置
先决条件	-
被调用函数	-
输入参数{in}	
waveform_detect_mode	波形检测模式
BKP_WAVEFORM_DETECT_1	方波1检测
BKP_WAVEFORM_DETECT_2	方波2检测
输入参数{in}	
newvalue	功能使能/禁能
ENABLE	使能
DISABLE	除能
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* enable the first waveform detect */
```

```
bkp_waveform_detect_config(BKP_WAVEFORM_DETECT_1, ENABLE);
```

函数 bkp_flag_get

函数bkp_flag_get描述见下表：

表 3-59. 函数 bkp_flag_get

函数名称	bkp_flag_get
函数原型	FlagStatus bkp_flag_get(uint16_t flag);
功能描述	获取BKP标志位
先决条件	-
被调用函数	-
输入参数{in}	
flag	BKP标志位
BKP_FLAG_TAMP_ER0	侵入事件0标志
BKP_FLAG_TAMP_ER1_WAVEDETEC	侵入事件1/方波检测事件标志

<i>T</i>	
输出参数{out}	
-	-
返回值	
FlagStatus	SET或RESET

例如：

```
/* get BKP tamper0 event flag */
```

```
FlagStatus status;
```

```
status = bkp_flag_get(BKP_FLAG_TAMPER0);
```

函数 bkp_flag_clear

函数bkp_flag_clear描述见下表：

表 3-60. 函数 bkp_flag_clear

函数名称	bkp_flag_clear
函数原型	void bkp_flag_clear(uint16_t flag);
功能描述	清除BKP标志位
先决条件	-
被调用函数	-
输入参数{in}	
flag	BKP标志位
BKP_FLAG_TAMP ER0	侵入事件0标志
BKP_FLAG_TAMP ER1_WAVEDETEC T	侵入事件1/方波检测事件标志
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* clear BKP tamper0 event flag */
```

```
bkp_flag_clear(BKP_FLAG_TAMPER0);
```

函数 bkp_tamper_interrupt_enable

函数bkp_tamper_interrupt_enable描述见下表：

表 3-61. 函数 bkp_tamper_interrupt_enable

函数名称	bkp_tamper_interrupt_enable
------	-----------------------------

函数原型	void bkp_tamper_interrupt_enable(uint16_t bkp_interrupt);
功能描述	TAMPER中断使能
先决条件	-
被调用函数	-
输入参数{in}	
bkp_interrupt	BKP中断
<i>BKP_INT_TAMPER</i> <i>0</i>	BKP侵入中断0
<i>BKP_INT_TAMPER</i> <i>1_WAVEDETECT</i>	BKP侵入中断1/方波检测中断
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* enable tamper0 interrupt */
```

```
bkp_tamper_interrupt_enable(BKP_INT_TAMPER0);
```

函数 bkp_tamper_interrupt_disable

函数bkp_tamper_interrupt_disable描述见下表：

表 3-62. 函数 bkp_tamper_interrupt_disable

函数名称	bkp_tamper_interrupt_disable
函数原型	void bkp_tamper_interrupt_disable(uint16_t bkp_interrupt);
功能描述	TAMPER中断禁能
先决条件	-
被调用函数	-
输入参数{in}	
bkp_interrupt	BKP中断
<i>BKP_INT_TAMPER</i> <i>0</i>	BKP侵入中断0
<i>BKP_INT_TAMPER</i> <i>1_WAVEDETECT</i>	BKP侵入中断1/方波检测中断
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* disable tamper0 interrupt */
```

```
bkp_tamper_interrupt_disable(BKP_INT_TAMPER0);
```

函数 bkp_interrupt_flag_get

函数bkp_interrupt_flag_get描述见下表:

表 3-63. 函数 bkp_interrupt_flag_get

函数名称	bkp_interrupt_flag_get
函数原型	FlagStatus bkp_interrupt_flag_get(uint16_t flag);
功能描述	获取BKP中断标志位
先决条件	-
被调用函数	-
输入参数{in}	
flag	BKP中断标志位
BKP_INT_FLAG_T AMPER0	侵入中断0标志
BKP_INT_FLAG_T AMPER1_WAVEDETECT	侵入中断1/方波检测中断标志
输出参数{out}	
-	-
返回值	
FlagStatus	SET或RESET

例如:

```
/* get BKP tamper0 interrupt flag */
```

```
FlagStatus tamper0_status;
```

```
tamper0_status = bkp_interrupt_flag_get(BKP_INT_FLAG_TAMPER0);
```

函数 bkp_interrupt_flag_clear

函数bkp_interrupt_flag_clear描述见下表:

表 3-64. 函数 bkp_interrupt_flag_clear

函数名称	bkp_interrupt_flag_clear
函数原型	void bkp_interrupt_flag_clear(uint16_t flag);
功能描述	清除BKP中断标志位
先决条件	-
被调用函数	-
输入参数{in}	
flag	BKP中断标志位
BKP_INT_FLAG_T AMPER0	侵入中断0标志
BKP_INT_FLAG_T AMPER1_WAVEDETECT	侵入中断1/方波检测中断标志

TECT	
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* clear BKP tamper0 interrupt flag */
```

```
bkp_interrupt_flag_clear(BKP_INT_FLAG_TAMPER0);
```

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 3-65. 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_TMDATA0x	发送邮箱data0寄存器
CAN_TMDATA1x	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

寄存器名称	寄存器描述
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 3-66. CAN 库函数

库函数名称	库函数描述
can_deinit	复位外设CAN
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 3-67. 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_wake_up	自动唤醒
auto_retrans	自动重传
rec_fifo_overwrite	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_transmit_message_struct

表 3-68. 结构体 can_transmit_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 3-69. 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 3-70. 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 3-71. 枚举类型 can_flag_enum

成员名称	功能描述
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_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 3-72. 枚举类型 `can_interrupt_flag_enum`

成员名称	功能描述
<code>CAN_INT_FLAG_SLPIF</code>	进入睡眠工作模式的状态改变中断标志
<code>CAN_INT_FLAG_WUIF</code>	从睡眠工作模式唤醒的状态改变中断标志
<code>CAN_INT_FLAG_ERRIF</code>	错误中断标志
<code>CAN_INT_FLAG_MTF2</code>	邮箱2发送完成中断标志
<code>CAN_INT_FLAG_MTF1</code>	邮箱1发送完成中断标志
<code>CAN_INT_FLAG_MTF0</code>	邮箱0发送完成中断标志
<code>CAN_INT_FLAG_RFO0</code>	接收FIFO0溢出中断标志
<code>CAN_INT_FLAG_RFF0</code>	接收FIFO0满中断标志
<code>CAN_INT_FLAG_RFL0</code>	接收FIFO0非空中断标志
<code>CAN_INT_FLAG_RFO1</code>	接收FIFO1溢出中断标志
<code>CAN_INT_FLAG_RFF1</code>	接收FIFO1满中断标志
<code>CAN_INT_FLAG_RFL1</code>	接收FIFO1非空中断标志
<code>CAN_INT_FLAG_ERRN</code>	错误种类中断标志
<code>CAN_INT_FLAG_BOERR</code>	离线错误中断标志
<code>CAN_INT_FLAG_PERR</code>	被动错误中断标志
<code>CAN_INT_FLAG_WERR</code>	主动错误中断标志

枚举类型 `can_error_enum`

表 3-73. 枚举类型 `can_error_enum`

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

枚举类型 `can_transmit_state_enum`

表 3-74. 枚举类型 `can_transmit_state_enum`

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

枚举类型 `can_struct_type_enum`

表 3-75. 枚举类型 `can_struct_type_enum`

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

函数 `can_deinit`

函数`can_deinit`描述见下表：

表 3-76. 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
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 3-77. 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3-75. 枚举类型<code>can_struct_type_enum</code>
<code>CAN_INIT_STRUC</code>	CAN初始化参数结构体

<i>T</i>	
<i>CAN_FILTER_STRUCTURE</i>	CAN过滤器参数结构体
<i>CAN_TX_MESSAGE_STRUCT</i>	CAN发送信息参数结构体
<i>CAN_RX_MESSAGE_STRUCT</i>	CAN接受信息参数结构体
输出参数{out}	
p_struct	特定结构体的指针
返回值	
-	-

例如:

```
/* CAN parameter struct initialize */
can_parameter_struct can_parameter;
can_struct_para_init(CAN_INIT_STRUCTURE, &can_parameter);
```

函数 can_init

函数can_init描述见下表:

表 3-78. 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
<i>CANx(x=0,1)</i>	CAN外设选择
输入参数{in}	
can_parameter_init	初始化结构体, 结构体成员参考 表3-67. 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_parameter.time_triggered = DISABLE;

can_parameter.auto_bus_off_recovery = DISABLE;

can_parameter.auto_wake_up = DISABLE;

can_parameter.auto_retrans = ENABLE;

can_parameter.rec_fifo_overwrite = DISABLE;

can_parameter.trans_fifo_order = DISABLE;

can_parameter.working_mode = CAN_NORMAL_MODE;

can_parameter.resync_jump_width = CAN_BT_SJW_1TQ;

can_parameter.time_segment_1 = CAN_BT_BS1_8TQ;

can_parameter.time_segment_2 = CAN_BT_BS2_3TQ;

can_parameter.prescaler = 5;

can_init(CAN0, &can_parameter);

```

函数 can_filter_init

函数can_filter_init描述见下表：

表 3-79. 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3-70. 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.filter_number=0;

can_filter.filter_mode = CAN_FILTERMODE_MASK;

can_filter.filter_bits = CAN_FILTERBITS_32BIT;

```

```
can_filter.filter_list_high = 0x3000;

can_filter.filter_list_low = 0x0000;

can_filter.filter_mask_high = 0x3000;

can_filter.filter_mask_low = 0x0000;

can_filter.filter_fifo_number = CAN_FIFO0;

can_filter.filter_enable = ENABLE;

can_filter_init(&can_filter);
```

函数 can1_filter_start_bank

函数can1_filter_start_bank描述见下表：

表 3-80. 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 fliter 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 3-81. 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
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 3-82. 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
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 3-83. 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
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 3-84. 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
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 3-85. 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
输入参数{in}	
transmit_message	报文发送结构体，结构体成员参考 表3-68. 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;

can_transmit_message_struct transmit_message;

transmit_message.tx_sfid = 0x300 >> 1;

transmit_message.tx_efid = 0x00;

transmit_message.tx_ft = CAN_FT_DATA;

transmit_message.tx_ff = CAN_FF_STANDARD;

transmit_message.tx_dlen = 2;

transmit_message.tx_data[0] = 0x55;

transmit_message.tx_data[1] = 0xAA;

transmit_mailbox = can_message_transmit(CAN0, &transmit_message);

```

函数 can_transmit_states

函数can_transmit_states描述见下表：

表 3-86. 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
输入参数{in}	

mailbox_number	邮箱标号
<i>CAN_MAILBOXx</i> (x=0,1,2)	CAN_MAILBOXx
输出参数{out}	
-	-
返回值	
can_transmit_state_enum	请参考 表3-74. 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 3-87. 函数 can_transmission_stop

函数名称	can_transmission_stop
函数原型	ErrStatus 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
<i>CANx</i> (x=0,1)	CAN外设选择
输入参数{in}	
mailbox_number	邮箱标号
<i>CAN_MAILBOXx</i> (x=0,1,2)	CAN_MAILBOXx
输出参数{out}	
-	-
返回值	
ErrStatus	SUCCESS或ERROR

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 3-88. 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
输入参数{in}	
fifo_number	FIFO编号
CAN_FIFOx(x=0,1)	CAN_FIFOx
输入参数{in}	
receive_message	接收报文结构体, 结构体成员参考 表3-69. 结构体 can_receive_message_struct
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* CAN0 FIFO0 receive message */
can_receive_message_struct 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 3-89. 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

输入参数{in}	
fifo_number	FIFO编号
<i>CAN_FIFOx(x=0,1)</i>	CAN_FIFOx
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 3-90. 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
<i>CANx(x=0,1)</i>	CAN外设选择
输入参数{in}	
fifo_number	FIFO编号
<i>CAN_FIFOx(x=0,1)</i>	CAN_FIFOx
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 3-91. 函数 `can_working_mode_set`

函数名称	<code>can_working_mode_set</code>
函数原型	<code>ErrStatus can_working_mode_set(uint32_t can_periph, uint8_t working_mode);</code>
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}	
<code>can_periph</code>	CAN外设
<code>CANx(x=0,1)</code>	CAN外设选择
输入参数{in}	
<code>can_working_mode</code>	模式选择
<code>CAN_MODE_INITIALIZE</code>	初始化模式
<code>CAN_MODE_NORMAL</code>	正常模式
<code>CAN_MODE_SLEEP</code>	睡眠模式
输出参数{out}	
-	-
返回值	
<code>ErrStatus</code>	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 3-92. 函数 `can_wakeup`

函数名称	<code>can_wakeup</code>
函数原型	<code>ErrStatus can_wakeup(uint32_t can_periph);</code>
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}	
<code>can_periph</code>	CAN外设
<code>CANx(x=0,1)</code>	CAN外设选择
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 3-93. 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
输出参数{out}	
-	-
返回值	
can_error_enum	请参考 表3-73. 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 3-94. 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
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 3-95. 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
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_flag_get

函数can_flag_get描述见下表：

表 3-96. 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
<i>CANx(x=0,1)</i>	CAN外设选择
输入参数{in}	
flag	CAN标志位参考 表3-71. 枚举类型can_flag_enum
<i>CAN_FLAG_RXL</i>	RX引脚电平
<i>CAN_FLAG_LASTRX</i>	RX引脚最近一次的采样值
<i>CAN_FLAG_RS</i>	接收状态
<i>CAN_FLAG_TS</i>	发送状态
<i>CAN_FLAG_SLPIF</i>	进入睡眠工作模式的状态改变标志
<i>CAN_FLAG_WUIF</i>	从睡眠工作模式唤醒的状态改变标志
<i>CAN_FLAG_ERRIF</i>	错误标志
<i>CAN_FLAG_SLPWS</i>	睡眠工作状态
<i>CAN_FLAG_IWS</i>	初始化工作状态
<i>CAN_FLAG_TMLS2</i>	在发送FIFO中邮箱2最后发送
<i>CAN_FLAG_TMLS1</i>	在发送FIFO中邮箱1最后发送
<i>CAN_FLAG_TMLS0</i>	在发送FIFO中邮箱0最后发送
<i>CAN_FLAG_TME2</i>	发送邮箱2空
<i>CAN_FLAG_TME1</i>	发送邮箱1空
<i>CAN_FLAG_TME0</i>	发送邮箱0空
<i>CAN_FLAG_MTE2</i>	邮箱2发送错误
<i>CAN_FLAG_MTE1</i>	邮箱1发送错误
<i>CAN_FLAG_MTE0</i>	邮箱0发送错误
<i>CAN_FLAG_MAL2</i>	邮箱2仲裁失败
<i>CAN_FLAG_MAL1</i>	邮箱1仲裁失败
<i>CAN_FLAG_MAL0</i>	邮箱0仲裁失败
<i>CAN_FLAG_MTFNERR2</i>	邮箱2无错发送完成
<i>CAN_FLAG_MTFNERR1</i>	邮箱1无错发送完成
<i>CAN_FLAG_MTFNERR0</i>	邮箱0无错发送完成
<i>CAN_FLAG_MTF2</i>	邮箱2发送完成
<i>CAN_FLAG_MTF1</i>	邮箱1发送完成
<i>CAN_FLAG_MTF0</i>	邮箱0发送完成
<i>CAN_FLAG_RFO0</i>	接收FIFO0溢出
<i>CAN_FLAG_RFF0</i>	接收FIFO0满
<i>CAN_FLAG_RFO1</i>	接收FIFO1溢出
<i>CAN_FLAG_RFF1</i>	接收FIFO1满
<i>CAN_FLAG_BOERR</i>	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 3-97. 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
输入参数{in}	
flag	CAN标志位参考 表3-71. 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_enable

函数can_interrupt_enable描述见下表:

表 3-98. 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
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 3-99. 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
输入参数{in}	
interrupt	中断类型
CAN_INT_TME	发送邮箱空中断
CAN_INT_RFNE0	接收FIFO0非空中断
CAN_INT_RFF0	接收FIFO0满中断
CAN_INT_RFO0	接收FIFO0溢出中断
CAN_INT_RFNE1	接收FIFO1非空中断
CAN_INT_RFF1	接收FIFO1满中断
CAN_INT_RFO1	接收FIFO1溢出中断
CAN_INT_WERR	警告错误中断
CAN_INT_PERR	被动错误中断
CAN_INT_BO	离线中断
CAN_INT_ERRN	错误种类中断
CAN_INT_ERR	错误中断
CAN_INT_WU	唤醒中断
CAN_INT_SLPW	睡眠中断
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_interrupt_flag_get

函数can_interrupt_flag_get描述见下表:

表 3-100. 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
输入参数{in}	
flag	CAN中断标志位参考 表3-72. 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	离线错误中断标志

<i>ERR</i>	
<i>CAN_INT_FLAG_PE</i> <i>RR</i>	被动错误中断标志
<i>CAN_INT_FLAG_WE</i> <i>RR</i>	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 3-101. 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
输入参数{in}	
flag	CAN中断标志位参考 表3-72. 枚举类型can_interrupt_flag_enum
<i>CAN_INT_FLAG_SLP</i> <i>IF</i>	进入睡眠工作模式的状态改变中断标志
<i>CAN_INT_FLAG_WUI</i> <i>F</i>	从睡眠工作模式唤醒的状态改变中断标志
<i>CAN_INT_FLAG_ER</i> <i>RIF</i>	错误中断标志
<i>CAN_INT_FLAG_MT</i> <i>F2</i>	邮箱2发送完成中断标志
<i>CAN_INT_FLAG_MT</i> <i>F1</i>	邮箱1发送完成中断标志
<i>CAN_INT_FLAG_MT</i> <i>F0</i>	邮箱0发送完成中断标志

<code>CAN_INT_FLAG_RF00</code>	接收FIFO0溢出中断标志
<code>CAN_INT_FLAG_RFF0</code>	接收FIFO0满中断标志
<code>CAN_INT_FLAG_RF01</code>	接收FIFO1溢出中断标志
<code>CAN_INT_FLAG_RFF1</code>	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. CAU

加密处理单元支持处理DES，三重DES或AES（128，192或256）算法，对数据进行加密或解密。CAU寄存器列举在章节[3.5.1](#)，CAU固件库函数介绍在章节[3.5.2](#)。

3.5.1. 外设寄存器说明

CAU寄存器列表如下表所示：

表 3-102. CAU 寄存器

寄存器名称	寄存器描述
CAU_CTL	CAU控制寄存器
CAU_STAT0	CAU状态寄存器0
CAU_DI	CAU数据输入寄存器
CAU_DO	CAU数据输出寄存器
CAU_DMAEN	CAU DMA使能寄存器
CAU_INTEN	CAU中断使能寄存器
CAU_STAT1	CAU状态寄存器1
CAU_INTF	CAU中断标志寄存器
CAU_KEY0H	CAU密钥0高位寄存器
CAU_KEY0L	CAU密钥0低位寄存器
CAU_KEY1H	CAU密钥1高位寄存器
CAU_KEY1L	CAU密钥1低位寄存器
CAU_KEY2H	CAU密钥2高位寄存器
CAU_KEY2L	CAU密钥2低位寄存器

寄存器名称	寄存器描述
CAU_KEY3H	CAU密钥3高位寄存器
CAU_KEY3L	CAU密钥3低位寄存器
CAU_IV0H	CAU初始化向量0高位寄存器
CAU_IV0L	CAU初始化向量0低位寄存器
CAU_IV1H	CAU初始化向量1高位寄存器
CAU_IV1L	CAU初始化向量1低位寄存器

3.5.2. 外设库函数说明

CAU库函数列表如下表所示：

表 3-103. CAU 库函数

库函数名称	库函数描述
cau_deinit	复位CAU外设
cau_enable	使能CAU外设
cau_disable	除能CAU外设
cau_dma_enable	使能CAU DMA接口
cau_dma_disable	除能CAU DMA接口
cau_init	初始化CAU
cau_aes_keysize_config	在使用AES算法的情况下配置密钥大小
cau_key_init	初始化密钥参数
cau_key_parameter_init	初始化结构体cau_key_initpara
cau_iv_init	初始化矢量参数
cau_iv_parameter_init	初始化结构体cau_iv_parameter_struct
cau_fifo_flush	清除FIFO内容
cau_enable_state_get	返回CAU外设是否使能的状态值
cau_data_write	将数据写入IN FIFO
cau_data_read	返回最近进入OUT FIFO的数据
cau_aes_ecb	ECB模式下使用AES算法加密和解密
cau_aes_cbc	CBC模式下使用AES算法加密和解密
cau_aes_ctr	CTR模式下使用AES算法加密和解密
cau_tdes_ecb	ECB模式下使用TDES算法加密和解密
cau_tdes_cbc	CBC模式下使用TDES算法加密和解密
cau_des_ecb	ECB模式下使用DES算法加密和解密
cau_des_cbc	CBC模式下使用DES算法加密和解密
cau_flag_get	获取CAU标志状态
cau_interrupt_enable	使能CAU中断
cau_interrupt_disable	除能CAU中断
cau_interrupt_flag_get	获取中断标志

结构体 `cau_key_parameter_struct`

表 3-104. 结构体 `cau_key_parameter_struct`

成员名称	功能描述
<code>key_0_high</code>	密钥0高位
<code>key_0_low</code>	密钥0低位
<code>key_1_high</code>	密钥1高位
<code>key_1_low</code>	密钥1低位
<code>key_2_high</code>	密钥2高位
<code>key_2_low</code>	密钥2低位
<code>key_3_high</code>	密钥3高位
<code>key_3_low</code>	密钥3低位

结构体 `cau_iv_parameter_struct`

表 3-105. 结构体 `cau_iv_parameter_struct`

成员名称	功能描述
<code>iv_0_high</code>	矢量0高位
<code>iv_0_low</code>	矢量0低位
<code>iv_1_high</code>	矢量1高位
<code>iv_1_low</code>	矢量1低位

枚举类型 `cau_text_struct`

表 3-106. 枚举类型 `cau_text_struct`

成员名称	功能描述
<code>input</code>	指向输入缓存区
<code>in_length</code>	输入缓存区长度，必须是8的倍数（DES或TDES）抑或16的倍数（AES）
<code>output</code>	指向返回缓存区

函数 `cau_deinit`

函数`cau_deinit`描述见下表：

表 3-107. 函数 `cau_deinit`

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

返回值	
-	-

例如：

```
/* reset the CAU peripheral */
```

```
cau_deinit();
```

函数 cau_enable

函数cau_enable描述见下表：

表 3-108. 函数 cau_enable

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

例如：

```
/* enable the CAU peripheral */
```

```
cau_enable();
```

函数 cau_disable

函数cau_disable描述见下表：

表 3-109. 函数 cau_disable

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

例如:

```
/* disable the CAU peripheral */
```

```
cau_disable();
```

函数 cau_dma_enable

函数cau_dma_enable描述见下表:

表 3-110. 函数 cau_dma_enable

函数名称	cau_dma_enable
函数原形	void cau_dma_enable(uint32_t dma_req);
功能描述	使能CAU DMA接口
先决条件	-
被调用函数	-
输入参数{in}	
dma_req	使能CAU指定的DMA传输请求方向
CAU_DMA_INFIFO	DMA用于接收数据
CAU_DMA_OUTFIFO	DMA用于发送数据
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* enable the CAU DMA interface */
```

```
cau_dma_enable(CAU_DMA_INFIFO);
```

函数 cau_dma_disable

函数cau_dma_disable描述见下表:

表 3-111. 函数 cau_dma_disable

函数名称	cau_dma_disable
函数原形	void cau_dma_disable(uint32_t dma_req);
功能描述	除能CAU DMA接口
先决条件	-
被调用函数	-
输入参数{in}	
dma_req	除能CAU指定的DMA传输请求方向
CAU_DMA_INFIFO	DMA用于接收数据
CAU_DMA_OUTFIFO	DMA用于发送数据
输出参数{out}	
-	-

返回值	
-	-

例如:

```
/* disable the CAU DMA interface */
```

```
cau_dma_disable(CAU_DMA_INFIFO);
```

函数 cau_init

函数cau_init描述见下表:

表 3-112. 函数 cau_init

函数名称	cau_init
函数原形	void cau_init(uint32_t algo_dir, uint32_t algo_mode, uint32_t swapping);
功能描述	初始化CAU
先决条件	-
被调用函数	-
输入参数{in}	
algo_dir	算法方向
CAU_ENCRYPT	加密
CAU_DECRYPT	解密
输入参数{in}	
algo_mode	算法模式选择
CAU_MODE_TDES_ECB	TDES-ECB
CAU_MODE_TDES_CBC	TDES-CBC
CAU_MODE_DES_ECB	DES-ECB
CAU_MODE_DES_CBC	DES-CBC
CAU_MODE_AES_ECB	AES-ECB
CAU_MODE_AES_CBC	AES-CBC
CAU_MODE_AES_CTR	AES-CTR
CAU_MODE_AES_KEY	AES解密密钥准备模式
输入参数{in}	
swapping	数据交换选择
CAU_SWAPPING_32BIT	无交换

CAU_SWAPPING_16BIT	半字交换
CAU_SWAPPING_8BIT	字节交换
CAU_SWAPPING_1BIT	位交换
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* initialize the CAU peripheral */
```

```
cau_init(CAU_ENCRYPT, CAU_MODE_TDES_ECB, CAU_SWAPPING_32BIT);
```

函数 cau_aes_keysize_config

函数cau_aes_keysize_config描述见下表：

表 3-113. 函数 cau_aes_keysize_config

函数名称	cau_aes_keysize_config
函数原形	void cau_aes_keysize_config(uint32_t key_size);
功能描述	在使用AES算法的情况下配置密钥大小
先决条件	-
被调用函数	-
输入参数{in}	
key_size	密钥长度
CAU_KEYSZIE_128BIT	128位密钥长度
CAU_KEYSZIE_192BIT	192位密钥长度
CAU_KEYSZIE_256BIT	256位密钥长度
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* configure key size if used AES algorithm */
```

```
cau_aes_keysize_config (CAU_KEYSZIE_128BIT);
```

函数 cau_key_init

函数cau_key_init描述见下表：

表 3-114. 函数 cau_key_init

函数名称	cau_key_init
函数原形	void cau_key_init(cau_key_parameter_struct* key_initpara);

功能描述	初始化密钥参数
先决条件	-
被调用函数	-
输入参数{in}	
key_initpara	密钥初始化参数结构体，参考结构体 表3-104. 结构体 cau_key_parameter_struct
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* initialize the key parameters */

cau_key_parameter_struct key_initpara;

uint32_t key_128[4] = {0x2b7e1516U, 0x28aed2a6U, 0xabf71588U, 0x09cf4f3cU};

key_initpara.key_2_high = key_128[0];

key_initpara.key_2_low  = key_128[1];

key_initpara.key_3_high = key_128[2];

key_initpara.key_3_low  = key_128[3];

cau_key_init(&key_initpara);
```

函数 cau_key_parameter_init

函数cau_key_parameter_init描述见下表：

表 3-115. 函数 cau_key_parameter_init

函数名称	cau_key_parameter_init
函数原形	void cau_key_parameter_init(cau_key_parameter_struct* key_initpara);
功能描述	初始化结构体cau_key_initpara
先决条件	-
被调用函数	-
输入参数{in}	
-	-
输出参数{out}	
key_initpara	密钥初始化参数结构体，参考结构体 表3-104. 结构体 cau_key_parameter_struct
返回值	
-	-

例如：

```
/* initialize the sturct cau_key_initpara */
```

```
cau_key_parameter_struct key_initpara;
```

```
cau_key_parameter_init(&key_initpara);
```

函数 cau_iv_init

函数cau_iv_init描述见下表：

表 3-116. 函数 cau_iv_init

函数名称	cau_iv_init
函数原形	void cau_iv_init(cau_iv_parameter_struct* iv_initpara);
功能描述	初始化矢量参数
先决条件	-
被调用函数	-
输入参数{in}	
iv_initpara	矢量初始化参数结构体，参考结构体 表3-105. 结构体 cau_iv_parameter_struct
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* initialize the vectors parameters */
```

```
cau_iv_parameter_struct iv_initpara;
```

```
uint32_t vectors[4] = {0x00010203U, 0x04050607U, 0x08090A0BU, 0x0C0D0E0FU};
```

```
iv_initpara.iv_0_high = vectors[0];
```

```
iv_initpara.iv_0_low = vectors[1];
```

```
iv_initpara.iv_1_high = vectors[2];
```

```
iv_initpara.iv_1_low = vectors[3];
```

```
cau_iv_init(&iv_initpara);
```

函数 cau_iv_parameter_init

函数cau_iv_parameter_init描述见下表：

表 3-117. 函数 cau_iv_parameter_init

函数名称	cau_iv_parameter_init
函数原形	void cau_iv_parameter_init(cau_iv_parameter_struct* iv_initpara);
功能描述	初始化矢量参数结构体

先决条件	-
被调用函数	-
输入参数{in}	
-	-
输出参数{out}	
iv_initpara	矢量初始化参数结构体，参考结构体 表3-105. 结构体 cau_iv_parameter_struct
返回值	
-	-

例如：

```
/* initialize the vectors parameters */
cau_iv_parameter_struct iv_initpara;
cau_iv_parameter_init (&iv_initpara);
```

函数 cau_fifo_flush

函数cau_fifo_flush描述见下表：

表 3-118. 函数 cau_fifo_flush

函数名称	cau_fifo_flush
函数原形	void cau_fifo_flush(void);
功能描述	清除FIFO内容
先决条件	-
被调用函数	-
输入参数{in}	
-	-
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* flush the IN and OUT FIFOs */
cau_fifo_flush();
```

函数 cau_enable_state_get

函数cau_enable_state_get描述见下表：

表 3-119. 函数 cau_enable_state_get

函数名称	cau_enable_state_get
函数原形	ControlStatus cau_enable_state_get(void);

功能描述	返回CAU外设是否使能的状态值
先决条件	-
被调用函数	-
输入参数{in}	
-	-
输出参数{out}	
-	-
返回值	
ControlStatus	ENABLE或DISABLE

例如：

```
/* return whether CAU peripheral is enabled or disabled */
```

```
ControlStatus state;
```

```
state = cau_enable_state_get();
```

函数 cau_data_write

函数cau_data_write描述见下表：

表 3-120. 函数 cau_data_write

函数名称	cau_data_write
函数原形	void cau_data_write(uint32_t data);
功能描述	将数据写入IN FIFO
先决条件	-
被调用函数	-
输入参数{in}	
data	所写的的数据(0x0 - 0xFFFFFFFF)
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* write data to the IN FIFO */
```

```
cau_data_write(0x10);
```

函数 cau_data_read

函数cau_data_read描述见下表：

表 3-121. 函数 cau_data_read

函数名称	cau_data_read
函数原形	uint32_t cau_data_read(void);

功能描述	返回最近进入OUT FIFO的数据
先决条件	-
被调用函数	-
输入参数{in}	
-	-
输出参数{out}	
-	-
返回值	
uint32_t	0x0 – 0xFFFFFFFF

例如：

```
/* return the last data entered into the output FIFO */
```

```
uint32_t data;
```

```
data = cau_data_read();
```

函数 cau_aes_ecb

函数cau_aes_ecb描述见下表：

表 3-122. 函数 cau_aes_ecb

函数名称	cau_aes_ecb
函数原形	ErrStatus cau_aes_ecb(uint32_t algo_dir, uint8_t *key, uint16_t keysize, cau_text_struct *text);
功能描述	ECB模式下使用AES算法加密和解密
先决条件	-
被调用函数	-
输入参数{in}	
algo_dir	算法方向
CAU_ENCRYPT	加密
CAU_DECRYPT	解密
输入参数{in}	
key	用于AES算法的密钥
输入参数{in}	
keysize	密钥长度必须是128, 192或256
输入参数{in}	
text	指向文本信息结构体，参考结构体 表3-106. 枚举类型cau_text_struct
输出参数{out}	
-	-
返回值	
ErrStatus	SUCCESS或ERROR

例如：

```
/* encrypt using AES in ECB mode */
```

```

ErrStatus status;

cau_text_struct text;

uint8_t plaintext[64] = {

    0x6bU, 0xc1U, 0xbeU, 0xe2U, 0x2eU, 0x40U, 0x9fU, 0x96U,

    0xe9U, 0x3dU, 0x7eU, 0x11U, 0x73U, 0x93U, 0x17U, 0x2aU,

    0xaeU, 0x2dU, 0x8aU, 0x57U, 0x1eU, 0x03U, 0xacU, 0x9cU,

    0x9eU, 0xb7U, 0x6fU, 0xacU, 0x45U, 0xafU, 0x8eU, 0x51U,

    0x30U, 0xc8U, 0x1cU, 0x46U, 0xa3U, 0x5cU, 0xe4U, 0x11U,

    0xe5U, 0xfbU, 0xc1U, 0x19U, 0x1aU, 0x0aU, 0x52U, 0xefU,

    0xf6U, 0x9fU, 0x24U, 0x45U, 0xdfU, 0x4fU, 0x9bU, 0x17U,

    0xadU, 0x2bU, 0x41U, 0x7bU, 0xe6U, 0x6cU, 0x37U, 0x10U

};

uint8_t encrypt_result[64];

text.input = plaintext;

text.in_length = 64;

text.output = encrypt_result;

status = cau_aes_ecb(CAU_ENCRYPT, CAU_KEYSIZE_128BIT, 128, &text);

```

函数 cau_aes_cbc

函数cau_aes_cbc描述见下表：

表 3-123. 函数 cau_aes_cbc

函数名称	cau_aes_cbc
函数原形	ErrStatus cau_aes_cbc(uint32_t algo_dir, uint8_t *key, uint16_t keysize, uint8_t iv[16], cau_text_struct *text);
功能描述	CBC模式下使用AES算法加密和解密
先决条件	-
被调用函数	-
输入参数{in}	
algo_dir	算法方向
CAU_ENCRYPT	加密
CAU_DECRYPT	解密
输入参数{in}	
key	用于AES算法的密钥
输入参数{in}	

keysize	密钥长度必须是128, 192或256
输入参数{in}	
iv	用于AES算法的初始化矢量
输入参数{in}	
text	指向文本信息结构体，参考结构体 表3-106. 枚举类型cau_text_struct
输出参数{out}	
-	-
返回值	
ErrStatus	SUCCESS或ERROR

例如：

```

/* encrypt using AES in CBC mode */

ErrStatus status;

cau_text_struct text;

uint8_t plaintext[64] = {
    0x6bU, 0xc1U, 0xbeU, 0xe2U, 0x2eU, 0x40U, 0x9fU, 0x96U,
    0xe9U, 0x3dU, 0x7eU, 0x11U, 0x73U, 0x93U, 0x17U, 0x2aU,
    0xaeU, 0x2dU, 0x8aU, 0x57U, 0x1eU, 0x03U, 0xacU, 0x9cU,
    0x9eU, 0xb7U, 0x6fU, 0xacU, 0x45U, 0xafU, 0x8eU, 0x51U,
    0x30U, 0xc8U, 0x1cU, 0x46U, 0xa3U, 0x5cU, 0xe4U, 0x11U,
    0xe5U, 0xfbU, 0xc1U, 0x19U, 0x1aU, 0x0aU, 0x52U, 0xefU,
    0xf6U, 0x9fU, 0x24U, 0x45U, 0xdfU, 0x4fU, 0x9bU, 0x17U,
    0xadU, 0x2bU, 0x41U, 0x7bU, 0xe6U, 0x6cU, 0x37U, 0x10U
};

uint8_t encrypt_result[64];

text.input = plaintext;

text.in_length = 64;

text.output = encrypt_result;

uint8_t iv[16] = {0x00}

status = cau_aes_cbc(CAU_ENCRYPT, CAU_KEYSIZE_128BIT, 128, iv, &text);

```

函数 cau_aes_ctr

函数cau_aes_ctr描述见下表：

表 3-124. 函数 cau_aes_ctr

函数名称	cau_aes_ctr
函数原形	ErrStatus cau_aes_ctr(uint32_t algo_dir, uint8_t *key, uint16_t keysize, uint8_t iv[16], cau_text_struct *text);
功能描述	CTR模式下使用AES算法加密和解密
先决条件	-
被调用函数	-
输入参数{in}	
algo_dir	算法方向
CAU_ENCRYPT	加密
CAU_DECRYPT	解密
输入参数{in}	
key	用于AES算法的密钥
输入参数{in}	
keysize	密钥长度必须是128, 192或256
输入参数{in}	
iv	用于AES算法的初始化矢量
输入参数{in}	
text	指向文本信息结构体，参考结构体 表3-106. 枚举类型cau_text_struct
输出参数{out}	
-	-
返回值	
ErrStatus	SUCCESS或ERROR

例如：

```
/* encrypt using AES in CTR mode */
```

```
ErrStatus status;
```

```
cau_text_struct text;
```

```
uint8_t plaintext[64] = {
```

```
    0x6bU, 0xc1U, 0xbeU, 0xe2U, 0x2eU, 0x40U, 0x9fU, 0x96U,
    0xe9U, 0x3dU, 0x7eU, 0x11U, 0x73U, 0x93U, 0x17U, 0x2aU,
    0xaeU, 0x2dU, 0x8aU, 0x57U, 0x1eU, 0x03U, 0xacU, 0x9cU,
    0x9eU, 0xb7U, 0x6fU, 0xacU, 0x45U, 0xafU, 0x8eU, 0x51U,
    0x30U, 0xc8U, 0x1cU, 0x46U, 0xa3U, 0x5cU, 0xe4U, 0x11U,
    0xe5U, 0xfbU, 0xc1U, 0x19U, 0x1aU, 0x0aU, 0x52U, 0xefU,
    0xf6U, 0x9fU, 0x24U, 0x45U, 0xdfU, 0x4fU, 0x9bU, 0x17U,
    0xadU, 0x2bU, 0x41U, 0x7bU, 0xe6U, 0x6cU, 0x37U, 0x10U
```

```
};

uint8_t encrypt_result[64];

text.input = plaintext;

text.in_length = 64;

text.output = encrypt_result;

uint8_t iv[16] = {0x00}

status = cau_aes_ctr(CAU_ENCRYPT, CAU_KEYSIZE_128BIT, 128, iv, &text);
```

函数 cau_tdes_ecb

函数cau_tdes_ecb描述见下表：

表 3-125. 函数 cau_tdes_ecb

函数名称	cau_tdes_ecb
函数原形	ErrStatus cau_tdes_ecb(uint32_t algo_dir, uint8_t key[24], cau_text_struct *text);
功能描述	ECB模式下使用TDES算法加密和解密
先决条件	-
被调用函数	-
输入参数{in}	
algo_dir	算法方向
CAU_ENCRYPT	加密
CAU_DECRYPT	解密
输入参数{in}	
key	用于TDES算法的密钥
输入参数{in}	
text	指向文本信息结构体，参考结构体 表3-106. 枚举类型cau_text_struct
输出参数{out}	
-	-
返回值	
ErrStatus	SUCCESS或ERROR

例如：

```
/* encrypt using TDES in ECB mode */

ErrStatus status;

cau_text_struct text;

uint8_t plaintext[64] = {

    0x6bU, 0xc1U, 0xbeU, 0xe2U, 0x2eU, 0x40U, 0x9fU, 0x96U,

    0xe9U, 0x3dU, 0x7eU, 0x11U, 0x73U, 0x93U, 0x17U, 0x2aU,
```

```

0xaeU, 0x2dU, 0x8aU, 0x57U, 0x1eU, 0x03U, 0xacU, 0x9cU,
0x9eU, 0xb7U, 0x6fU, 0xacU, 0x45U, 0xafU, 0x8eU, 0x51U,
0x30U, 0xc8U, 0x1cU, 0x46U, 0xa3U, 0x5cU, 0xe4U, 0x11U,
0xe5U, 0xfbU, 0xc1U, 0x19U, 0x1aU, 0x0aU, 0x52U, 0xefU,
0xf6U, 0x9fU, 0x24U, 0x45U, 0xdfU, 0x4fU, 0x9bU, 0x17U,
0xadU, 0x2bU, 0x41U, 0x7bU, 0xe6U, 0x6cU, 0x37U, 0x10U
};

```

```

uint8_t encrypt_result[64];

text.input = plaintext;

text.in_length = 64;

text.output = encrypt_result;

uint8_t key[24] = {0x01};

status = cau_tdes_ecb(CAU_ENCRYPT, key, &text);

```

函数 cau_tdes_cbc

函数cau_tdes_cbc描述见下表：

表 3-126. 函数 cau_tdes_cbc

函数名称	cau_tdes_cbc
函数原形	ErrStatus cau_tdes_cbc(uint32_t algo_dir, uint8_t key[24], uint8_t iv[8], cau_text_struct *text);
功能描述	CBC模式下使用TDES算法加密和解密
先决条件	-
被调用函数	-
输入参数{in}	
algo_dir	算法方向
CAU_ENCRYPT	加密
CAU_DECRYPT	解密
输入参数{in}	
key	用于TDES算法的密钥
输入参数{in}	
iv	用于TDES算法的初始化矢量
输入参数{in}	
text	指向文本信息结构体，参考结构体 表3-106. 枚举类型cau_text_struct
输出参数{out}	
-	-
返回值	

ErrStatus	SUCCESS或ERROR
-----------	---------------

例如:

```
/* encrypt using TDES in CBC mode */
```

```
ErrStatus status;
```

```
cau_text_struct text;
```

```
uint8_t plaintext[64] = {
```

```
    0x6bU, 0xc1U, 0xbeU, 0xe2U, 0x2eU, 0x40U, 0x9fU, 0x96U,
    0xe9U, 0x3dU, 0x7eU, 0x11U, 0x73U, 0x93U, 0x17U, 0x2aU,
    0xaeU, 0x2dU, 0x8aU, 0x57U, 0x1eU, 0x03U, 0xacU, 0x9cU,
    0x9eU, 0xb7U, 0x6fU, 0xacU, 0x45U, 0xafU, 0x8eU, 0x51U,
    0x30U, 0xc8U, 0x1cU, 0x46U, 0xa3U, 0x5cU, 0xe4U, 0x11U,
    0xe5U, 0xfbU, 0xc1U, 0x19U, 0x1aU, 0x0aU, 0x52U, 0xefU,
    0xf6U, 0x9fU, 0x24U, 0x45U, 0xdfU, 0x4fU, 0x9bU, 0x17U,
    0xadU, 0x2bU, 0x41U, 0x7bU, 0xe6U, 0x6cU, 0x37U, 0x10U
```

```
};
```

```
uint8_t encrypt_result[64];
```

```
text.input = plaintext;
```

```
text.in_length = 64;
```

```
text.output = encrypt_result;
```

```
uint8_t key[24] = {0x01};
```

```
uint8_t iv[8] = {0x00};
```

```
ErrStatus status = cau_tdes_cbc(CAU_ENCRYPT, key, iv, &text);
```

函数 cau_des_ecb

函数cau_des_ecb描述见下表:

表 3-127. 函数 cau_des_ecb

函数名称	cau_des_ecb
函数原形	ErrStatus cau_des_ecb(uint32_t algo_dir, uint8_t key[24], cau_text_struct *text);
功能描述	ECB模式下使用DES算法加密和解密
先决条件	-
被调用函数	-

输入参数{in}	
algo_dir	算法方向
<i>CAU_ENCRYPT</i>	加密
<i>CAU_DECRYPT</i>	解密
输入参数{in}	
key	用于DES算法的密钥
输入参数{in}	
text	指向文本信息结构体，参考结构体 表3-106. 枚举类型cau_text_struct
输出参数{out}	
-	-
返回值	
ErrStatus	SUCCESS或ERROR

例如：

```

/* encrypt using DES in ECB mode */

ErrStatus status;

cau_text_struct text;

uint8_t plaintext[64] = {
    0x6bU, 0xc1U, 0xbeU, 0xe2U, 0x2eU, 0x40U, 0x9fU, 0x96U,
    0xe9U, 0x3dU, 0x7eU, 0x11U, 0x73U, 0x93U, 0x17U, 0x2aU,
    0xaeU, 0x2dU, 0x8aU, 0x57U, 0x1eU, 0x03U, 0xacU, 0x9cU,
    0x9eU, 0xb7U, 0x6fU, 0xacU, 0x45U, 0xafU, 0x8eU, 0x51U,
    0x30U, 0xc8U, 0x1cU, 0x46U, 0xa3U, 0x5cU, 0xe4U, 0x11U,
    0xe5U, 0xfbU, 0xc1U, 0x19U, 0x1aU, 0x0aU, 0x52U, 0xefU,
    0xf6U, 0x9fU, 0x24U, 0x45U, 0xdfU, 0x4fU, 0x9bU, 0x17U,
    0xadU, 0x2bU, 0x41U, 0x7bU, 0xe6U, 0x6cU, 0x37U, 0x10U
};

uint8_t encrypt_result[64];

text.input = plaintext;

text.in_length = 64;

text.output = encrypt_result;

uint8_t key[24] = {0x01};

status = cau_des_ecb(CAU_ENCRYPT, key, &text);

```


函数 cau_des_cbc

函数cau_des_cbc描述见下表：

表 3-128. 函数 cau_des_cbc

函数名称	cau_des_cbc
函数原形	ErrStatus cau_des_cbc(uint32_t algo_dir, uint8_t key[24], uint8_t iv[8], cau_text_struct *text);
功能描述	CBC模式下使用DES算法加密和解密
先决条件	-
被调用函数	-
输入参数{in}	
algo_dir	算法方向
CAU_ENCRYPT	加密
CAU_DECRYPT	解密
输入参数{in}	
key	用于DES算法的密钥
输入参数{in}	
iv	用于DES算法的初始化矢量
输入参数{in}	
text	指向文本信息结构体，参考结构体 表3-106. 枚举类型cau_text_struct
输出参数{out}	
-	-
返回值	
ErrStatus	SUCCESS或ERROR

例如：

```

/* encrypt using DES in CBC mode */

ErrStatus status;

cau_text_struct text;

uint8_t plaintext[64] = {

    0x6bU, 0xc1U, 0xbeU, 0xe2U, 0x2eU, 0x40U, 0x9fU, 0x96U,
    0xe9U, 0x3dU, 0x7eU, 0x11U, 0x73U, 0x93U, 0x17U, 0x2aU,
    0xaeU, 0x2dU, 0x8aU, 0x57U, 0x1eU, 0x03U, 0xacU, 0x9cU,
    0x9eU, 0xb7U, 0x6fU, 0xacU, 0x45U, 0xafU, 0x8eU, 0x51U,
    0x30U, 0xc8U, 0x1cU, 0x46U, 0xa3U, 0x5cU, 0xe4U, 0x11U,
    0xe5U, 0xfbU, 0xc1U, 0x19U, 0x1aU, 0x0aU, 0x52U, 0xefU,
    0xf6U, 0x9fU, 0x24U, 0x45U, 0xdfU, 0x4fU, 0x9bU, 0x17U,

```

0xadU, 0x2bU, 0x41U, 0x7bU, 0xe6U, 0x6cU, 0x37U, 0x10U

```
};

uint8_t encrypt_result[64];

text.input = plaintext;

text.in_length = 64;

text.output = encrypt_result;

uint8_t key[24] = {0x01};

uint8_t iv[8] = {0x00};

status = cau_des_cbc(CAU_ENCRYPT, key, iv, &text);
```

函数 cau_flag_get

函数cau_flag_get描述见下表:

表 3-129. 函数 cau_flag_get

函数名称	cau_flag_get
函数原形	FlagStatus cau_flag_get(uint32_t flag)
功能描述	获取CAU标志状态
先决条件	-
被调用函数	-
输入参数{in}	
flag	CAU标志状态
CAU_FLAG_INFIFO_EMPTY	输入FIFO空标志
CAU_FLAG_INFIFO_NO_FULL:	输入FIFO未满足标志
CAU_FLAG_OUTFIFO_NO_EMPTY	输出FIFO非空标志
CAU_FLAG_OUTFIFO_FULL	输出FIFO满标志
CAU_FLAG_BUSY	CAU内核忙标志
CAU_FLAG_INFIFO	输入FIFO标志状态
CAU_FLAG_OUTFIFO	输出FIFO标志状态
输出参数{out}	
-	-
返回值	
FlagStatus	SET或RESET

例如:

```
/* get the CAU flag status */
```

```
FlagStatus status;
```

```
status = cau_flag_get(CAU_FLAG_INFIFO_EMPTY);
```

函数 cau_interrupt_enable

函数cau_interrupt_enable描述见下表:

表 3-130. 函数 cau_interrupt_enable

函数名称	cau_interrupt_enable
函数原形	void cau_interrupt_enable(uint32_t interrupt)
功能描述	使能CAU中断
先决条件	-
被调用函数	-
输入参数{in}	
interrupt	使能CAU指定中断源
CAU_INT_INFIFO	输入FIFO中断
CAU_INT_OUTFIFO	输出FIFO中断
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* enable cau interrupt */
```

```
cau_interrupt_enable(CAU_INT_INFIFO);
```

函数 cau_interrupt_disable

函数cau_interrupt_disable描述见下表:

表 3-131. 函数 cau_interrupt_disable

函数名称	cau_interrupt_disable
函数原形	void cau_interrupt_disable(uint32_t interrupt)
功能描述	除能CAU中断
先决条件	-
被调用函数	-
输入参数{in}	
interrupt	使能CAU指定中断源
CAU_INT_INFIFO	输入FIFO中断
CAU_INT_OUTFIFO	输出FIFO中断
输出参数{out}	
-	-
返回值	

-	-
---	---

例如：

```
/* disable cau interrupt */
cau_interrupt_disable(CAU_INT_INFIFO);
```

函数 cau_interrupt_flag_get

函数cau_interrupt_flag_get描述见下表：

表 3-132. 函数 cau_interrupt_flag_get

函数名称	cau_interrupt_flag_get
函数原形	FlagStatus cau_interrupt_flag_get(uint32_t int_flag)
功能描述	获取中断标志
先决条件	-
被调用函数	-
输入参数{in}	
int_flag	CAU中断标志
CAU_INT_FLAG_INFIFO	输入FIFO中断
CAU_INT_FLAG_OUTFIFO	输出FIFO中断
输出参数{out}	
-	-
返回值	
FlagStatus	SET 或 RESET

例如：

```
/* get the CAU interrupt flag status */
FlagStatus status;
status = cau_interrupt_flag_get(CAU_INT_FLAG_INFIFO);
```

3.6. 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[3.6.1](#)描述了CRC的寄存器列表，章节[3.6.2](#)对CRC库函数进行说明。

3.6.1. 外设寄存器说明

CRC寄存器列表如下表所示：

表 3-133. 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

3.6.2. 外设库函数说明

CRC库函数列表如下表所示：

表 3-134. 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	计算CRC一个32位数据
crc_block_data_calculate	计算CRC一个32位数组

函数 crc_deinit

函数crc_deinit描述见下表：

表 3-135. 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 3-136. 函数 `crc_data_register_reset`

函数名称	<code>crc_data_register_reset</code>
函数原形	<code>void crc_data_register_reset(void);</code>
功能描述	根据数据寄存器的复位值（0xFFFFFFFF）复位数据寄存器
先决条件	-
被调用函数	-
输入参数{in}	
-	-
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* reset crc data register */
crc_data_register_reset();
```

函数 `crc_data_register_read`

函数 `crc_data_register_read` 描述见下表：

表 3-137. 函数 `crc_data_register_read`

函数名称	<code>crc_data_register_read</code>
函数原形	<code>uint32_t crc_data_register_read(void);</code>
功能描述	读数据寄存器
先决条件	-
被调用函数	-
输入参数{in}	
-	-
输出参数{out}	
-	-
返回值	
<code>uint32_t</code>	从数据寄存器读取的32位数据 (0-0xFFFFFFFF)

例如：

```
/* read crc data register */
uint32_t crc_value = 0;
crc_value = crc_data_register_read();
```

函数 `crc_free_data_register_read`

函数 `crc_free_data_register_read` 描述见下表：

表 3-138. 函数 `crc_free_data_register_read`

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

例如：

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

uint8_t crc_value = 0;

crc_value = crc_free_data_register_read();
```

函数 `crc_free_data_register_write`

函数 `crc_free_data_register_write` 描述见下表：

表 3-139. 函数 `crc_free_data_register_write`

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

例如：

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

crc_free_data_register_write(0x11);
```

函数 `crc_single_data_calculate`

函数 `crc_single_data_calculate` 描述见下表：

表 3-140. 函数 `crc_single_data_calculate`

函数名称	<code>crc_single_data_calculate</code>
函数原形	<code>uint32_t crc_single_data_calculate(uint32_t sdata);</code>
功能描述	计算CRC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 3-141. 函数 `crc_block_data_calculate`

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

例如：

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



```
#define BUFFER_SIZE 6

uint32_t valcrc = 0;

static const uint32_t data_buffer[BUFFER_SIZE] = {

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

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

3.7. DAC

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

3.7.1. 外设寄存器说明

DAC寄存器列表如下表所示：

表 3-142. DAC 寄存器

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

3.7.2. 外设库函数说明

DAC库函数列表如下表所示：

表 3-143. 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

库函数名称	库函数描述
<code>dac_dma_disable</code>	DAC的DMA功能禁能
<code>dac_output_buffer_enable</code>	DAC输出缓冲区使能
<code>dac_output_buffer_disable</code>	DAC输出缓冲区禁能
<code>dac_output_value_get</code>	DAC输出数据获取
<code>dac_data_set</code>	DAC输出数据设置
<code>dac_trigger_enable</code>	DAC触发使能
<code>dac_trigger_disable</code>	DAC触发禁能
<code>dac_trigger_source_config</code>	DAC触发源配置
<code>dac_software_trigger_enable</code>	DAC软件触发使能
<code>dac_wave_mode_config</code>	DAC噪声波模式配置
<code>dac_lfsr_noise_config</code>	DAC LFSR模式配置
<code>dac_triangle_noise_config</code>	DAC三角波模式配置
<code>dac_concurrent_enable</code>	并发DAC模式使能
<code>dac_concurrent_disable</code>	并发DAC模式禁能
<code>dac_concurrent_software_trigger_enable</code>	并发DAC模式软件触发使能
<code>dac_concurrent_output_buffer_enable</code>	并发DAC模式输出缓冲区使能
<code>dac_concurrent_output_buffer_disable</code>	并发DAC模式输出缓冲区禁能
<code>dac_concurrent_data_set</code>	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 3-144. 函数 `dac_deinit`

函数名称	<code>dac_deinit</code>
函数原型	<code>void dac_deinit(uint32_t dac_periph);</code>
功能描述	DAC外设复位
先决条件	-
被调函数	<code>rcu_periph_reset_enable</code> / <code>rcu_periph_reset_disable</code>
输入参数{in}	
<code>dac_periph</code>	DAC外设
<code>DACx</code>	DAC外设选择（ <code>x = 0</code> ）
输出参数{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 3-145. 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 3-146. 函数 dac_disable

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

-	-
---	---

例如:

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

函数 **dac_dma_enable**

函数dac_dma_enable描述见下表:

表 3-147. 函数 dac_dma_enable

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

例如:

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

函数 **dac_dma_disable**

函数dac_dma_disable描述见下表:

表 3-148. 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
<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 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 3-149. 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
<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:

```
/* 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 3-150. 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
<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 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 3-151. 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);
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 3-152. 函数 `dac_data_set`

函数名称	<code>dac_data_set</code>
函数原型	<code>void dac_data_set(uint32_t dac_periph, uint8_t dac_out, uint32_t dac_align, uint16_t data);</code>
功能描述	DAC输出数据设置
先决条件	-
被调用函数	-
输入参数{in}	
<code>dac_periph</code>	DAC外设
<code>DACx</code>	DAC外设选择（ <code>x = 0</code> ）
输入参数{in}	
<code>dac_out</code>	DAC输出
<code>DAC_OUTx</code>	DAC输出通道选择（ <code>x = 0,1</code> ）
输入参数{in}	
<code>dac_align</code>	DAC对齐模式
<code>DAC_ALIGN_12B_R</code>	12位数据右对齐
<code>DAC_ALIGN_12B_L</code>	12位数据左对齐
<code>DAC_ALIGN_8B_R</code>	8位数据右对齐
输入参数{in}	
<code>data</code>	写入 <code>DAC_OUTx</code> 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 3-153. 函数 `dac_trigger_enable`

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

```
/* 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 3-154. 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 3-155. 函数 dac_trigger_source_config

函数名称	dac_trigger_source_config
函数原型	void dac_trigger_source_config(uint32_t dac_periph, uint8_t dac_out, uint32_t triggersource);
功能描述	DAC触发源配置
先决条件	-
被调用函数	-
输入参数{in}	
dac_periph	DAC外设
DACx	DAC外设选择 (x = 0)
输入参数{in}	
dac_out	DAC输出
DAC_OUTx	DAC输出通道选择 (x = 0,1)
输入参数{in}	
triggersource	DAC触发源
DAC_TRIGGER_T5_TRGO	TIMER5 TRGO
DAC_TRIGGER_T2_TRGO	TIMER2 TRGO
DAC_TRIGGER_T6_TRGO	TIMER6 TRGO
DAC_TRIGGER_T4_TRGO	TIMER4 TRGO
DAC_TRIGGER_T1_TRGO	TIMER1 TRGO
DAC_TRIGGER_T3_TRGO	TIMER3 TRGO
DAC_TRIGGER_EXTI_9	EXTI线9中断
DAC_TRIGGER_SOFTWARE	软件触发
输出参数{out}	
-	-
返回值	
-	-

例如：

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

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

函数 dac_software_trigger_enable

函数dac_software_trigger_enable描述见下表：

表 3-156. 函数 dac_software_trigger_enable

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

例如：

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

函数 dac_wave_mode_config

函数dac_wave_mode_config描述见下表：

表 3-157. 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

<code>DAC_WAVE_DISABLE</code>	噪声波模式禁能
<code>DAC_WAVE_MODE_LFSR</code>	LFSR噪声波模式
<code>DAC_WAVE_MODE_TRIANGLE</code>	三角波噪声波模式
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* configure DAC0_OUT0 wave mode */
dac_wave_mode_config(DAC0, DAC_OUT0, DAC_WAVE_DISABLE);
```

函数 `dac_lfsr_noise_config`

函数 `dac_lfsr_noise_config` 描述见下表:

表 3-158. 函数 `dac_lfsr_noise_config`

函数名称	<code>dac_lfsr_noise_config</code>
函数原型	<code>void dac_lfsr_noise_config(uint32_t dac_periph, uint8_t dac_out, uint32_t unmask_bits);</code>
功能描述	DAC LFSR模式配置
先决条件	-
被调用函数	-
输入参数{in}	
<code>dac_periph</code>	DAC外设
<code>DACx</code>	DAC外设选择 (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)
输入参数{in}	
<code>unmask_bits</code>	噪声波的非屏蔽位宽
<code>DAC_LFSR_BIT0</code>	LFSR模式位0非屏蔽
<code>DAC_LFSR_BITSx_0</code>	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 3-159. 函数 **dac_triangle_noise_config**

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

例如：

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

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

函数 **dac_concurrent_enable**

函数 **dac_concurrent_enable** 描述见下表：

表 3-160. 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
<i>DACx</i>	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 3-161. 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
<i>DACx</i>	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 3-162. 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
<i>DACx</i>	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 3-163. 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
<i>DACx</i>	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 3-164. 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
<i>DACx</i>	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 3-165. 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
<i>DACx</i>	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DACx_OUT0的数据 (0~4095)
输入参数{in}	
data1	写入DACx_OUT1的数据 (0~4095)
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* set DAC0 concurrent mode data holding register value */
dac_concurrent_data_set(DAC0, DAC_ALIGN_8B_R, 0xFF, 0xFF);
```

3.8. DBG

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

3.8.1. 外设寄存器说明

DBG寄存器列表如下表所示：

表 3-166. 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

3.8.2. 外设库函数说明

DBG库函数列表如下表所示：

表 3-167. DBG 库函数

库函数名称	库函数描述
dbg_id_get	读DBG_ID寄存器
dbg_low_power_enable	使能低功耗模式的MCU调试保持功能
dbg_low_power_disable	禁能低功耗模式的MCU调试保持功能
dbg_periph_enable	使能外设的MCU调试保持功能
dbg_periph_disable	禁能外设的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 3-168. 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_TIMER7_HOLD	当内核停止时，保持TIMER7计数器计数值不变

成员名称	功能描述
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_CAN1_HOLD	当内核停止时，CAN1接收寄存器停止接收数据
DBG_TIMER11_HOLD	当内核停止时，保持TIMER11计数器计数值不变
DBG_TIMER12_HOLD	当内核停止时，保持TIMER12计数器计数值不变
DBG_TIMER13_HOLD	当内核停止时，保持TIMER13计数器计数值不变
DBG_TIMER8_HOLD	当内核停止时，保持TIMER8计数器计数值不变
DBG_TIMER9_HOLD	当内核停止时，保持TIMER9计数器计数值不变
DBG_TIMER10_HOLD	当内核停止时，保持TIMER10计数器计数值不变
DBG_I2C2_HOLD	当内核停止时，保持I2C2的SMBUS状态不变，用于调试

函数 dbg_id_get

函数dbg_id_get描述见下表：

表 3-169. 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值（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 3-170. 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 3-171. 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 3-172. 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3-168. 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）
DBG_I2Cx_HOLD	当内核停止时，保持I2Cx的SMBUS状态不变，用于调试（x=0，1，2）
DBG_TIMERx_HOLD	当内核停止时，保持TIMERx计数器计数值不变（x=0~13）
输出参数{out}	
-	-
返回值	
-	-

例如:

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

```
dbg_periph_enable(DBG_TIMER0_HOLD);
```

函数 dbg_periph_disable

函数dbg_periph_disable描述见下表:

表 3-173. 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3-168. 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）
DBG_I2Cx_HOLD	当内核停止时，保持I2Cx的SMBUS状态不变，用于调试（x=0，1，2）
DBG_TIMERx_HOLD	当内核停止时，保持TIMERx计数器计数值不变（x=0~13）
输出参数{out}	
-	-
返回值	
-	-

例如：

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

```
dbg_periph_disable(DBG_TIMER0_HOLD);
```

函数 dbg_trace_pin_enable

函数dbg_trace_pin_enable描述见下表：

表 3-174. 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 3-175. 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 3-176. 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);
```

3.9. DCI

数字摄像头接口是一个同步并行接口，可以从数字摄像头捕获视频和图像信息。DCI寄存器列举在章节[3.9.1](#)，DCI固件库函数列举在章节[3.9.2](#)。

3.9.1. 外设寄存器说明

DCI寄存器列表如下表所示：

表 3-177. DCI 寄存器

寄存器名称	寄存器描述
DCI_CTL	DCI控制寄存器
DCI_STAT0	DCI状态寄存器0
DCI_STAT1	DCI状态寄存器1
DCI_INTEN	DCI中断使能寄存器
DCI_INTF	DCI中断标志寄存器
DCI_INTC	DCI中断标志清除寄存器
DCI_SC	DCI同步码寄存器
DCI_SCUMSK	DCI同步码屏蔽寄存器
DCI_CWSPOS	DCI裁剪窗口起始位置寄存器
DCI_CWSZ	DCI裁剪窗口大小寄存器
DCI_DATA	DCI数据寄存器

3.9.2. 外设库函数说明

DCI库函数列表如下表所示：

表 3-178. DCI 库函数

库函数名称	库函数描述
dci_deinit	复位DCI
dci_init	初始化DCI寄存器
dci_enable	使能DCI功能
dci_disable	除能DCI功能
dci_capture_enable	使能DCI捕获功能
dci_capture_disable	除能DCI捕获功能
dci_jpeg_enable	使能DCI JPEG模式
dci_jpeg_disable	除能DCI JPEG模式
dci_crop_window_enable	使能裁剪窗口功能
dci_crop_window_disable	除能裁剪窗口功能
dci_crop_window_config	配置DCI裁剪窗口功能
dci_embedded_sync_enable	使能内嵌同步模式
dci_embedded_sync_disable	除能内嵌同步模式
dci_sync_codes_config	在内嵌同步模式下配置同步码

库函数名称	库函数描述
dc_i_sync_codes_unmask_config	在内嵌同步模式下配置非屏蔽同步码
dc_i_data_read	读取DCI数据寄存器
dc_i_flag_get	获取指定标志
dc_i_interrupt_enable	使能指定的DCI中断
dc_i_interrupt_disable	除能指定的DCI中断
dc_i_interrupt_flag_get	获取指定的中断标志
dc_i_interrupt_flag_clear	清除指定的中断标志

结构体 dc_i_parameter_struct

表 3-179. 结构体 dc_i_parameter_struct

成员名称	功能描述
capture_mode	DCI获取模式: DCI_CAPTURE_MODE_CONTINUOUS / DCI_CAPTURE_MODE_SNAPSHOT
clock_polarity	时钟极性选择: DCI_CK_POLARITY_FALLING / DCI_CK_POLARITY_RISING
hsync_polarity	水平极性选择: DCI_HSYNC_POLARITY_LOW / DCI_HSYNC_POLARITY_HIGH
vsync_polarity	垂直极性选择: DCI_VSYNC_POLARITY_LOW / DCI_VSYNC_POLARITY_HIGH
frame_rate	帧获取速率: DCI_FRAME_RATE_ALL / DCI_FRAME_RATE_1_2 / DCI_FRAME_RATE_1_4
interface_format	数码相机接口格式: DCI_INTERFACE_FORMAT_8BITS / DCI_INTERFACE_FORMAT_10BITS / DCI_INTERFACE_FORMAT_12BITS / DCI_INTERFACE_FORMAT_14BITS

函数 dc_i_deinit

函数dc_i_deinit描述见下表:

表 3-180. 函数 dc_i_deinit

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

例如:

```
/* DCI deinit */
```

```
dci_deinit();
```

函数 dci_init

函数dci_init描述见下表:

表 3-181. 函数 dci_init

函数名称	dci_init
函数原形	void dci_init(dci_parameter_struct* dci_struct);
功能描述	初始化DCI寄存器, 参考 表3-179. 结构体dci_parameter_struct
先决条件	-
被调用函数	-
输入参数{in}	
dci_struct	DCI参数初始化结构体
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* initialize DCI registers */
```

```
dci_parameter_struct dci_struct;
```

```
dci_struct.capture_mode = DCI_CAPTURE_MODE_CONTINUOUS;
```

```
dci_struct.clock_polarity = DCI_CK_POLARITY_RISING;
```

```
dci_struct.hsync_polarity = DCI_HSYNC_POLARITY_LOW;
```

```
dci_struct.vsync_polarity = DCI_VSYNC_POLARITY_LOW;
```

```
dci_struct.frame_rate = DCI_FRAME_RATE_ALL;
```

```
dci_struct.interface_format = DCI_INTERFACE_FORMAT_8BITS;
```

```
dci_init(&dci_struct);
```

函数 dci_enable

函数dci_enable描述见下表:

表 3-182. 函数 dci_enable

函数名称	dci_enable
函数原形	void dci_enable(void);
功能描述	使能DCI功能

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

例如：

```
/* enable DCI function */
```

```
dc_i_enable( );
```

函数 dci_disable

函数dci_disable描述见下表：

表 3-183. 函数 dci_disable

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

例如：

```
/* disable DCI function */
```

```
dc_i_disable( );
```

函数 dci_capture_enable

函数dci_capture_enable描述见下表：

表 3-184. 函数 dci_capture_enable

函数名称	dci_capture_enable
函数原形	void dci_capture_enable(void);
功能描述	使能DCI捕获功能
先决条件	-
被调用函数	-

输入参数{in}	
-	-
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* enable DCI capture */
```

```
dci_capture_enable( );
```

函数 dci_capture_disable

函数dci_capture_disable描述见下表：

表 3-185. 函数 dci_capture_disable

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

例如：

```
/* disable DCI capture */
```

```
dci_capture_disable( );
```

函数 dci_jpeg_enable

函数dci_jpeg_enable描述见下表：

表 3-186. 函数 dci_jpeg_enable

函数名称	dci_jpeg_enable
函数原形	void dci_jpeg_enable(void);
功能描述	使能DCI JPEG功能
先决条件	-
被调用函数	-
输入参数{in}	
-	-

输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* enable DCI jpeg mode */
dci_jpeg_enable( );
```

函数 dci_jpeg_disable

函数dci_jpeg_disable描述见下表：

表 3-187. 函数 dci_jpeg_disable

函数名称	dci_jpeg_disable
函数原形	void dci_jpeg_disable(void);
功能描述	除能DCI JPEG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 DCI jpeg mode */
dci_jpeg_disable( );
```

函数 dci_crop_window_enable

函数dci_crop_window_enable描述见下表：

表 3-188. 函数 dci_crop_window_enable

函数名称	dci_crop_window_enable
函数原形	void dci_crop_window_enable(void);
功能描述	使能裁剪窗口功能
先决条件	-
被调用函数	-
输入参数{in}	
-	-
输出参数{out}	
-	-

返回值	
-	-

例如：

```
/* enable cropping window function */
```

```
dc_i_crop_window_enable( );
```

函数 dci_crop_window_disable

函数dci_crop_window_disable描述见下表：

表 3-189. 函数 dci_crop_window_disable

函数名称	dc_i_crop_window_disable
函数原形	void dc_i_crop_window_disable(void);
功能描述	除能裁剪窗口功能
先决条件	-
被调用函数	-
输入参数{in}	
-	-
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* disable cropping window function */
```

```
dc_i_crop_window_disable( );
```

函数 dci_crop_window_config

函数dci_crop_window_config描述见下表：

表 3-190. 函数 dci_crop_window_config

函数名称	dc_i_crop_window_config
函数原形	void dc_i_crop_window_config(uint16_t start_x, uint16_t start_y, uint16_t size_width, uint16_t size_height);
功能描述	配置DCI裁剪窗口功能
先决条件	-
被调用函数	-
输入参数{in}	
start_x	窗口水平起始地址
输入参数{in}	
start_y	窗口垂直起始地址
输入参数{in}	

size_width	窗口水平长度大小
输入参数{in}	
size_height	窗口垂直长度大小
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* config DCI cropping window */
dci_crop_window_config (0x800, 0x600, 0x100, 0x100);
```

函数 dci_embedded_sync_enable

函数dci_embedded_sync_enable描述见下表：

表 3-191. 函数 dci_embedded_sync_enable

函数名称	dci_embedded_sync_enable
函数原形	void dci_embedded_sync_enable(void);
功能描述	使能内嵌同步模式
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 embedded synchronous mode */
dci_embedded_sync_enable( );
```

函数 dci_embedded_sync_disable

函数dci_embedded_sync_disable描述见下表：

表 3-192. 函数 dci_embedded_sync_disable

函数名称	dci_embedded_sync_disable
函数原形	void dci_embedded_sync_disable(void);
功能描述	除能内嵌同步模式
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 embedded synchronous mode */
dci_embedded_sync_disable( )
```

函数 dci_sync_codes_config

函数dci_sync_codes_config描述见下表：

表 3-193. 函数 dci_sync_codes_config

函数名称	dci_sync_codes_config
函数原形	void dci_sync_codes_config(uint8_t frame_start, uint8_t line_start, uint8_t line_end, uint8_t frame_end);
功能描述	在内嵌同步模式下配置同步码
先决条件	-
被调用函数	-
输入参数{in}	
frame_start	在内嵌同步模式帧起始码
输入参数{in}	
line_start	在内嵌同步模式行起始码
输入参数{in}	
line_end	在内嵌同步模式行终止码
输入参数{in}	
frame_end	在内嵌同步模式帧终止码
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* config synchronous codes in embedded synchronous mode */
dci_sync_codes_config (0x10, 0x10, 0x20, 0x20)
```

函数 dci_sync_codes_unmask_config

函数dci_sync_codes_unmask_config描述见下表：

表 3-194. 函数 dci_sync_codes_unmask_config

函数名称	dci_sync_codes_unmask_config
函数原形	void dci_sync_codes_unmask_config(uint8_t frame_start, uint8_t line_start,

	uint8_t line_end, uint8_t frame_end);
功能描述	在内嵌同步模式下配置非屏蔽同步码
先决条件	-
被调用函数	-
输入参数{in}	
frame_start	在内嵌同步模式帧起始码
输入参数{in}	
line_start	在内嵌同步模式行起始码
输入参数{in}	
line_end	在内嵌同步模式行终止码
输入参数{in}	
frame_end	在内嵌同步模式帧终止码
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* config synchronous codes unmask in embedded synchronous mode */
dci_sync_codes_unmask_config (0x10, 0x10, 0x20, 0x20)
```

函数 dci_data_read

函数dci_data_read描述见下表：

表 3-195. 函数 dci_data_read

函数名称	dci_data_read
函数原形	uint32_t dci_data_read(void);
功能描述	读取DCI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	0x00 – 0xFFFFFFFF

例如：

```
/* read DCI data register */
uint32_t data = 0;
data = dci_data_read();
```

函数 dci_flag_get

函数dci_flag_get描述见下表:

表 3-196. 函数 dci_flag_get

函数名称	dci_flag_get
函数原形	FlagStatus dci_flag_get(uint32_t flag);
功能描述	获取指定标志
先决条件	-
被调用函数	-
输入参数{in}	
flag	DCI标志
DCI_FLAG_HS	水平同步行状态
DCI_FLAG_VS	垂直同步行状态
DCI_FLAG_FV	FIFO有效
DCI_FLAG_EF	帧结束
DCI_FLAG_OVR	FIFO溢出
DCI_FLAG_ESE	内嵌同步错误
DCI_FLAG_VSYNC	垂直同步
DCI_FLAG_EL	行结束
输出参数{out}	
-	-
返回值	
FlagStatus	SET或RESET

例如:

```
/* get specified flag */
FlagStatus status = RESET;
status = dci_flag_get (DCI_FLAG_HS);
```

函数 dci_interrupt_enable

函数dci_interrupt_enable描述见下表:

表 3-197. 函数 dci_interrupt_enable

函数名称	dci_interrupt_enable
函数原形	void dci_interrupt_enable(uint32_t interrupt);
功能描述	使能指定的DCI中断
先决条件	-
被调用函数	-
输入参数{in}	
interrupt	DCI中断
DCI_INT_EF	帧结束中断标志
DCI_INT_OVR	FIFO溢出中断标志

<i>DCI_INT_ESE</i>	内嵌码同步错误中断标志
<i>DCI_INT_VSYNC</i>	垂直同步中断标志
<i>DCI_INT_EL</i>	行结束中断标志
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* enable specified DCI interrupt */
dci_interrupt_enable (DCI_INT_EF);
```

函数 dci_interrupt_disable

函数dci_interrupt_disable描述见下表：

表 3-198. 函数 dci_interrupt_disable

函数名称	dci_interrupt_disable
函数原形	void dci_interrupt_disable(uint32_t interrupt);
功能描述	除能指定的DCI中断
先决条件	-
被调用函数	-
输入参数{in}	
interrupt	DCI中断
<i>DCI_INT_EF</i>	帧结束中断标志
<i>DCI_INT_OVR</i>	FIFO溢出中断标志
<i>DCI_INT_ESE</i>	内嵌码同步错误中断标志
<i>DCI_INT_VSYNC</i>	垂直同步中断标志
<i>DCI_INT_EL</i>	行结束中断标志
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* disable specified DCI interrupt */
dci_interrupt_disable (DCI_INT_EF);
```

函数 dci_interrupt_flag_get

函数dci_interrupt_flag_get描述见下表：

表 3-199. 函数 dci_interrupt_flag_get

函数名称	dci_interrupt_flag_get
------	------------------------

函数原形	FlagStatus dci_interrupt_flag_get(uint32_t interrupt);
功能描述	获取指定的中断标志
先决条件	-
被调用函数	-
输入参数{in}	
interrupt	DCI中断
DCI_INT_FLAG_EF	帧结束中断标志
DCI_INT_FLAG_OVR	FIFO溢出中断标志
DCI_INT_FLAG_ESE	内嵌码同步错误中断标志
DCI_INT_FLAG_VSYN C	垂直同步中断标志
DCI_INT_FLAG_EL	行结束中断标志
输出参数{out}	
-	-
返回值	
FlagStatus	SET或RESET

例如：

```
/* get specified interrupt flag */
```

```
FlagStatus status = RESET;
```

```
status = dci_interrupt_flag_get(DCI_INT_FLAG_EF);
```

函数 dci_interrupt_flag_clear

函数dci_interrupt_flag_clear描述见下表：

表 3-200. 函数 dci_interrupt_flag_clear

函数名称	dci_interrupt_flag_clear
函数原形	void dci_interrupt_flag_clear(uint32_t interrupt);
功能描述	清除指定的中断标志
先决条件	-
被调用函数	-
输入参数{in}	
interrupt	DCI中断
DCI_INT_FLAG_EF	帧结束中断标志
DCI_INT_FLAG_OVR	FIFO溢出中断标志
DCI_INT_FLAG_ESE	内嵌码同步错误中断标志
DCI_INT_FLAG_VSYN C	垂直同步中断标志
DCI_INT_FLAG_EL	行结束中断标志
输出参数{out}	
-	-

返回值	
-	-

例如：

```
/*clear specified interrupt flag */
```

```
dci_interrupt_flag_clear (DCI_INT_FLAG_EF);
```

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

3.10.1. 外设寄存器说明

DMA寄存器列表如下表所示：

表 3-201. 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
DMA_ACFG	附加配置寄存器

3.10.2. 外设库函数说明

DMA库函数列表如下表所示：

表 3-202. DMA 库函数

库函数名称	库函数描述
dma_deinit	复位外设DMA的通道的所有寄存器
dma_struct_para_init	初始化DMA结构体为默认值
dma_init	初始化外设DMA的通道
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

库函数名称	库函数描述
<code>dma_memory_to_memory_disable</code>	禁能存储器到存储器DMA传输
<code>dma_channel_enable</code>	使能外设DMA的通道传输
<code>dma_channel_disable</code>	禁能外设DMA的通道传输
<code>dma_periph_address_config</code>	配置DMA通道传输的外设基地址
<code>dma_memory_address_config</code>	配置DMA通道传输的存储器基地址
<code>dma_transfer_number_config</code>	配置DMA通道还有多少数据要传输
<code>dma_transfer_number_get</code>	获取DMA通道还有多少数据要传输
<code>dma_priority_config</code>	配置DMA通道的传输软件优先级
<code>dma_memory_width_config</code>	配置DMA通道传输的存储器数据宽度
<code>dma_periph_width_config</code>	配置DMA通道传输的外设数据宽度
<code>dma_memory_increase_enable</code>	使能DMA通道传输的存储器地址生成算法增量模式
<code>dma_memory_increase_disable</code>	禁能DMA通道传输的存储器地址生成算法增量模式
<code>dma_periph_increase_enable</code>	使能DMA通道传输的外设地址生成算法增量模式
<code>dma_periph_increase_disable</code>	禁能DMA通道传输的外设地址生成算法增量模式
<code>dma_transfer_direction_config</code>	配置DMA通道的传输方向
<code>dma_flag_get</code>	获取DMA通道标志位状态
<code>dma_flag_clear</code>	清除DMA通道标志位状态
<code>dma_interrupt_flag_get</code>	获取DMA通道中断标志位状态
<code>dma_interrupt_flag_clear</code>	清除DMA通道中断标志位状态
<code>dma_interrupt_enable</code>	使能DMA通道中断
<code>dma_interrupt_disable</code>	禁能DMA通道中断
<code>dma_1_channel_5_fulldata_transfer_enable</code>	开启DMA1通道5完整数据传输模式
<code>dma_1_channel_5_fulldata_transfer_disable</code>	关闭DMA1通道5完整数据传输模式

结构体 `dma_parameter_struct`

表 3-203. 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 3-204. 函数 dma_deinit

函数名称	dma_deinit
函数原型	void dma_deinit(uint32_t dma_periph, dma_channel_enum channelx);
功能描述	复位外设DMA的通道的所有寄存器
先决条件	无
被调用函数	无
输入参数{in}	
dma_periph	DMA外设
DMA(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..6)	DMA通道选择
输出参数{out}	
-	-
返回值	
-	-

例如：

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

函数 dma_struct_para_init

函数dma_struct_para_deinit描述见下表：

表 3-205. 函数 dma_struct_para_init

函数名称	dma_struct_para_init
函数原型	void dma_struct_para_init(dma_parameter_struct* init_struct);
功能描述	初始化DMA结构体为默认值
先决条件	无
被调用函数	无
输入参数{in}	
-	-
输出参数{out}	
init_struct	DMA通道的默认初始值，参考 表3-203. 结构体dma_parameter_struct
返回值	
-	-

例如：

```
/* initialize the parameters of DMA struct with the default values */

dma_parameter_struct init_struct;

dma_struct_para_init(&init_struct);
```

函数 dma_init

函数dma_init描述见下表：

表 3-206. 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DMA的通道
先决条件	无
被调用函数	无
输入参数{in}	
dma_periph	DMA外设
DMA(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..6)	DMA通道选择
输入参数{in}	
init_struct	初始化结构体，结构体成员参考 表3-203. 结构体dma_parameter_struct
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* initialize DMA0 channel0 */

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 3-207. 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	无
被调用函数	无
输入参数{in}	
dma_periph	DMA外设
DMA(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..6)	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 3-208. 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	无
被调用函数	无
输入参数{in}	
dma_periph	DMA外设
DMA(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..6)	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 3-209. 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	无
被调用函数	无
输入参数{in}	
dma_periph	DMA外设
DMA(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..6)	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 3-210. 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	无
被调用函数	无
输入参数{in}	
dma_periph	DMA外设
DMA(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..6)	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 3-211. 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DMA的通道传输
先决条件	无

被调用函数	无
输入参数{in}	
dma_periph	DMA外设
<i>DMA(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..6)</i>	DMA通道选择
输出参数{out}	
-	-
返回值	
-	-

例如：

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

函数 dma_channel_disable

函数dma_channel_disable描述见下表：

表 3-212. 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	禁能外设DMA的通道传输
先决条件	无
被调用函数	无
输入参数{in}	
dma_periph	DMA外设
<i>DMA(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..6)</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 3-213. 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DMA通道传输的外设基地址
先决条件	无
被调用函数	无
输入参数{in}	
dma_periph	DMA外设
DMA(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..6)	DMA通道选择
输入参数{in}	
address	外设基地址
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* configure DMA0 channel0 address */
```

```
#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 3-214. 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DMA通道传输的存储器基地址

先决条件	无
被调用函数	无
输入参数{in}	
dma_periph	DMA外设
<i>DMA(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..6)</i>	DMA通道选择
输入参数{in}	
address	存储器基地址
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* configure DMA0 channel0 memory address */
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 3-215. 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DMA通道还有多少数据要传输
先决条件	无
被调用函数	无
输入参数{in}	
dma_periph	DMA外设
<i>DMA(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..6)</i>	DMA通道选择
输入参数{in}	
number	数据传输数量（0x0000 – 0xFFFF）

输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* configure DMA0 channel0 transfer number */
#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 3-216. 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DMA通道还有多少数据要传输
先决条件	无
被调用函数	无
输入参数{in}	
dma_periph	DMA外设
DMA(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..6)	DMA通道选择
输出参数{out}	
-	-
返回值	
uint32_t	DMA数据传输剩余数量（0x0000 – 0xFFFF）

例如：

```
/* get DMA0 channel0 transfer number */

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 3-217. 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DMA通道的传输软件优先级
先决条件	无
被调用函数	无
输入参数{in}	
dma_periph	DMA外设
DMA(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..6)	DMA通道选择
输入参数{in}	
priority	DMA通道软件优先级
DMA_PRIORITY_LOW	低优先级
DMA_PRIORITY_MEDIUM	中优先级
DMA_PRIORITY_HIGH	高优先级
DMA_PRIORITY_ULTRA_HIGH	极高优先级
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* configure DMA0 channel0 priority */
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 3-218. 函数 dma_memory_width_config

函数名称	dma_memory_width_config
函数原型	void dma_memory_width_config (uint32_t dma_periph, dma_channel_enum channelx, uint32_t mwidth);
功能描述	配置DMA通道传输的存储器数据宽度

先决条件	无
被调用函数	无
输入参数{in}	
dma_periph	DMA外设
<i>DMA(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..6)</i>	DMA通道选择
输入参数{in}	
mwidth	存储器数据传输宽度
<i>DMA_MEMORY_WIDTH_8BIT</i>	8位数据传输宽度
<i>DMA_MEMORY_WIDTH_16BIT</i>	16位数据传输宽度
<i>DMA_MEMORY_WIDTH_32BIT</i>	32位数据传输宽度
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* configure DMA0 channel0 memory width */
```

```
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 3-219. 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DMA通道传输的外设数据宽度
先决条件	无
被调用函数	无
输入参数{in}	
dma_periph	DMA外设
<i>DMA(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..6)</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：

```
/* configure DMA0 channel0 peripheral width */
```

```
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 3-220. 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DMA通道传输的存储器地址生成算法增量模式
先决条件	无
被调用函数	无
输入参数{in}	
dma_periph	DMA外设
<i>DMA(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..6)</i>	DMA通道选择
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* enable DMA0 channel0 memory increase */
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 3-221. 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DMA通道传输的存储器地址生成算法增量模式
先决条件	无
被调用函数	无
输入参数{in}	
dma_periph	DMA外设
DMA(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..6)	DMA通道选择
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* disable DMA0 channel0 memory increase */
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 3-222. 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DMA通道传输的外设地址生成算法增量模式
先决条件	无
被调用函数	无

输入参数{in}	
dma_periph	DMA外设
<i>DMA(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..6)</i>	DMA通道选择
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* enable DMA0 channel0 peripheral increase */
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 3-223. 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DMA通道传输的外设地址生成算法增量模式
先决条件	无
被调用函数	无
输入参数{in}	
dma_periph	DMA外设
<i>DMA(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..6)</i>	DMA通道选择
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* disable DMA0 channel0 peripheral increase */
```

```
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 3-224. 函数 dma_transfer_direction_config

函数名称	dma_transfer_direction_config
函数原型	void dma_transfer_direction_config(uint32_t dma_periph, dma_channel_enum channelx, uint32_t direction);
功能描述	配置DMA通道的传输方向
先决条件	无
被调用函数	无
输入参数{in}	
dma_periph	DMA外设
DMA(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..6)	DMA通道选择
输入参数{in}	
direction	数据传输方向
DMA_PERIPHERAL_TO_MEMORY	读取外设中数据, 写入存储器
DMA_MEMORY_TO_PERIPHERAL	读取存储器中数据, 写入外设
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* configure DMA0 channel0 peripheral to memory */
```

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

函数 dma_1_channel_5_fulldata_transfer_enable

函数dma_1_channel_5_fulldata_transfer_enable描述见下表:

表 3-225. 函数 dma_1_channel_5_fulldata_transfer_enable

函数名称	dma_1_channel_5_fulldata_transfer_enable
函数原型	void dma_1_channel_5_fulldata_transfer_enable(void);
功能描述	开启DMA1通道5完整数据传输模式

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 DMA1 channel 5 Full_Data transfer mode */
```

```
dma_1_channel_5_fulldata_transfer_enable();
```

函数 dma_1_channel_5_fulldata_transfer_disable

函数dma_1_channel_5_fulldata_transfer_disable描述见下表：

表 3-226. 函数 dma_1_channel_5_fulldata_transfer_disable

函数名称	dma_1_channel_5_fulldata_transfer_disable
函数原型	void dma_1_channel_5_fulldata_transfer_disable(void);
功能描述	关闭DMA1通道5完整数据传输模式
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 DMA1 channel 5 Full_Data transfer mode */
```

```
dma_1_channel_5_fulldata_transfer_disable();
```

函数 dma_flag_get

函数dma_flag_get描述见下表：

表 3-227. 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DMA通道标志位状态
先决条件	无

被调用函数	无
输入参数{in}	
dma_periph	DMA外设
<i>DMA(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..6)</i>	DMA通道选择
输入参数{in}	
flag	DMA标志
<i>DMA_FLAG_G</i>	DMA通道全局中断标志
<i>DMA_FLAG_FTF</i>	DMA通道传输完成标志
<i>DMA_FLAG_HTF</i>	DMA通道半传输完成标志
<i>DMA_FLAG_ERR</i>	DMA通道错误标志
输出参数{out}	
-	-
返回值	
FlagStatus	SET或RESET

例如：

```
/* get DMA0 channel0 flag */
```

```
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 3-228. 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DMA通道标志位状态
先决条件	无
被调用函数	无
输入参数{in}	
dma_periph	DMA外设
<i>DMA(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..6)</i>	DMA通道选择

$x=0..6$)	
输入参数{in}	
flag	DMA标志
<i>DMA_FLAG_G</i>	DMA通道全局中断标志
<i>DMA_FLAG_FTF</i>	DMA通道传输完成标志
<i>DMA_FLAG_HTF</i>	DMA通道半传输完成标志
<i>DMA_FLAG_ERR</i>	DMA通道错误标志
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* clear DMA0 channel0 flag */
```

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

函数 dma_interrupt_enable

函数dma_interrupt_enable描述见下表：

表 3-229. 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DMA通道中断
先决条件	无
被调用函数	无
输入参数{in}	
dma_periph	DMA外设
<i>DMA(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..6)</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 3-230. 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DMA通道中断
先决条件	无
被调用函数	无
输入参数{in}	
dma_periph	DMA外设
DMA(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..6)	DMA通道选择
输入参数{in}	
source	DMA中断源
DMA_INT_FTF	DMA通道传输完成中断
DMA_INT_HTF	DMA通道半传输完成中断
DMA_INT_ERR	DMA通道错误中断
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* DMA0 channel0 interrupt configuration */
```

```
dma_interrupt_disable(DMA0, DMA_CH0, DMA_INT_FTF);
```

函数 dma_interrupt_flag_get

函数dma_interrupt_flag_get描述见下表：

表 3-231. 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DMA通道中断标志位状态
先决条件	无
被调用函数	无
输入参数{in}	
dma_periph	DMA外设
DMA(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..6)	DMA通道选择
输入参数{in}	
flag	DMA标志
DMA_INT_FLAG_G	DMA通道全局中断标志位
DMA_INT_FLAG_TF	DMA通道传输完成中断标志
DMA_INT_FLAG_HTF	DMA通道半传输完成中断标志
DMA_INT_FLAG_ERR	DMA通道错误中断标志
输出参数{out}	
-	-
返回值	
FlagStatus	SET或RESET

例如:

```
/* get DMA0 channel3 interrupt flag */
FlagStatus int_flag_status = RESET;
int_flag_status = dma_interrupt_flag_get(DMA0, DMA_CH3, DMA_INT_FLAG_G);
```

函数 dma_interrupt_flag_clear

函数dma_interrupt_flag_clear描述见下表:

表 3-232. 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DMA通道中断标志位状态

先决条件	无
被调用函数	无
输入参数{in}	
dma_periph	DMA外设
<i>DMA(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..6)</i>	DMA通道选择
输入参数{in}	
flag	DMA标志
<i>DMA_INT_FLAG_G</i>	DMA通道全局中断标志
<i>DMA_INT_FLAG_TF</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	
-	-

例如：

```
/* clear DMA0 channel3 interrupt flag */
dma_interrupt_flag_clear(DMA0, DMA_CH3, DMA_INT_FLAG_G);
```

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

3.11.1. 外设寄存器说明

ENET寄存器列表如下表所示：

表 3-233. ENET 寄存器

寄存器名称	寄存器描述
ENET_MAC_CFG	MAC配置寄存器

寄存器名称	寄存器描述
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
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_ADDT2H	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
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_TSADDEND	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

寄存器名称	寄存器描述
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_RDTADDR	DMA接收描述符列表地址寄存器
ENET_DMA_TDTADDR	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_MFBOCNT	DMA丢失帧和缓存溢出计数器寄存器
ENET_DMA_CTDADDR	DMA当前发送描述符地址寄存器
ENET_DMA_CRDADDR	DMA当前接收描述符地址寄存器
ENET_DMA_CTBADDR	DMA当前发送缓存地址寄存器
ENET_DMA_CRBADDR	DMA当前接收缓存地址寄存器

3.11.2. 外设库函数说明

ENET库函数列表如下表所示：

表 3-234. 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
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/MSC/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/MSC/DMA中断

库函数名称	库函数描述
enet_interrupt_disable	禁能ENET模块MAC/MSC/DMA中断
enet_interrupt_flag_get	获取ENET模块MAC/MSC/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发送功能禁能（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_fliter_feature_enable	使能ENET帧过滤器相关功能
enet_fliter_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
enet_flowcontrol_threshold_config	配置流控阈值
enet_flowcontrol_feature_enable	使能ENET流控相关功能
enet_flowcontrol_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

库函数名称	库函数描述
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_wum_filter_register_pointer_reset	远程唤醒帧过滤器寄存器指针复位
enet_wum_filter_config	配置远程唤醒帧寄存器
enet_wum_feature_enable	使能ENET模块唤醒管理相关功能
enet_wum_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
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

库函数名称	库函数描述
enet_ptp_coarsecorrection_systime_up date	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
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 3-235. 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 3-236. 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

枚举类型 enet_flag_enum

表 3-237. 枚举类型 enet_flag_enum

成员名称	功能描述
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_ MSC	MSC状态标志位
ENET_MAC_FLAG_ MSCR	MSC接收状态标志位
ENET_MAC_FLAG_ MSCT	MSC发送状态标志位
ENET_MAC_FLAG_	时间戳触发状态标志位

成员名称	功能描述
TMST	
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_MSC_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_ ET	早接收状态标志位

成员名称	功能描述
ER	
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_EB_TRANSFER_ERROR	发送错误标志位
ENET_DMA_FLAG_EB_ACCESS_ERROR	DMA访问错误标志位
ENET_DMA_FLAG_MSC	MSC状态标志位
ENET_DMA_FLAG_WUM	WUM状态标志位
ENET_DMA_FLAG_TST	时间戳触发状态标志位

枚举类型 `enet_flag_clear_enum`

表 3-238. 枚举类型 `enet_flag_clear_enum`

成员名称	功能描述
ENET_DMA_FLAG_TS_CLR	发送状态标志位清除
ENET_DMA_FLAG_TPS_CLR	发送流程停止状态标志位清除
ENET_DMA_FLAG_TBU_CLR	发送缓冲区不可用状态标志位清除
ENET_DMA_FLAG_TJT_CLR	发送jabber超时状态标志位清除
ENET_DMA_FLAG_RO_CLR	接收溢出状态标志位清除
ENET_DMA_FLAG_TU_CLR	发送下溢状态标志位清除
ENET_DMA_FLAG_RS_CLR	接收状态标志位清除
ENET_DMA_FLAG_RBU_CLR	接收缓冲区不可用状态标志位清除
ENET_DMA_FLAG_RPS_CLR	接受流程停止状态标志位清除

成员名称	功能描述
<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

枚举类型 `enet_int_enum`

表 3-239. 枚举类型 `enet_int_enum`

成员名称	功能描述
<i>ENET_MAC_INT_WUMIM</i>	WUM中断屏蔽
<i>ENET_MAC_INT_T MSTIM</i>	时间戳触发中断屏蔽
<i>ENET_MSC_INT_R FCEIM</i>	接收帧CRC错误中断屏蔽
<i>ENET_MSC_INT_R FAEIM</i>	接收帧对齐错误中断屏蔽
<i>ENET_MSC_INT_R GUFIM</i>	接收“好”单播帧中断屏蔽
<i>ENET_MSC_INT_T GFSCIM</i>	仅遇到1个冲突后发送“好”帧中断屏蔽
<i>ENET_MSC_INT_T GFMSCIM</i>	遇到1个以上冲突后发送“好”帧中断屏蔽
<i>ENET_MSC_INT_T GFIM</i>	发送“好”的帧的中断屏蔽
<i>ENET_DMA_INT_TIE</i>	发送中断使能
<i>ENET_DMA_INT_T PSIE</i>	发送流程停止中断使能
<i>ENET_DMA_INT_T BUIE</i>	发送缓冲区不可用中断使能
<i>ENET_DMA_INT_T JTIE</i>	发送jabber超时中断使能
<i>ENET_DMA_INT_R</i>	接收溢出中断使能

成员名称	功能描述
OIE	
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

枚举类型 enet_int_flag_enum

表 3-240. 枚举类型 enet_int_flag_enum

成员名称	功能描述
ENET_MAC_INT_FLAG_WUM	WUM中断标志位
ENET_MAC_INT_FLAG_MSC	MSC中断标志位
ENET_MAC_INT_FLAG_MSCR	MSC接收中断标志位
ENET_MAC_INT_FLAG_MSCT	MSC发送中断标志位
ENET_MAC_INT_FLAG_TMST	时间戳触发中断标志位
ENET_MSC_INT_FLAG_RFCE	接收帧CRC错误中断标志位
ENET_MSC_INT_FLAG_RFAE	接收帧对齐错误中断标志位
ENET_MSC_INT_FLAG_RGUF	接收“好”单播帧中断标志位

成员名称	功能描述
ENET_MSC_INT_F LAG_TGFSC	仅遇到1个冲突后发送"好"帧中断标志位
ENET_MSC_INT_F LAG_TGFMSC	遇到1个以上冲突后发送"好"帧中断标志位
ENET_MSC_INT_F LAG_TGF	发送"好"的帧的中断标志位
ENET_DMA_INT_F LAG_TS	发送中断标志位
ENET_DMA_INT_F LAG_TPS	发送流程停止中断标志位
ENET_DMA_INT_F LAG_TBU	发送缓冲区不可用中断标志位
ENET_DMA_INT_F LAG_TJT	发送jabber超时中断标志位
ENET_DMA_INT_F LAG_RO	接收溢出中断标志位
ENET_DMA_INT_F LAG_TU	发送下溢中断标志位
ENET_DMA_INT_F LAG_RS	接收中断标志位
ENET_DMA_INT_F LAG_RBU	接收缓冲区不可用中断标志位
ENET_DMA_INT_F LAG_RPS	接收流程停止中断标志位
ENET_DMA_INT_F LAG_RWT	接收看门狗超时中断标志位
ENET_DMA_INT_F LAG_ET	早发送中断标志位
ENET_DMA_INT_F LAG_FBE	致命总线错误中断标志位
ENET_DMA_INT_F LAG_ER	早接收中断标志位
ENET_DMA_INT_F LAG_AI	异常中断汇总中断标志位
ENET_DMA_INT_F LAG_NI	正常中断汇总中断标志位
ENET_DMA_INT_F LAG_MSC	MSC中断标志位
ENET_DMA_INT_F LAG_WUM	WUM中断标志位
ENET_DMA_INT_F LAG_TST	时间戳触发中断标志位

枚举类型 `enet_int_flag_clear_enum`

表 3-241. 枚举类型 `enet_int_flag_clear_enum`

成员名称	功能描述
<code>ENET_DMA_INT_F LAG_TS_CLR</code>	发送中断标志位清除
<code>ENET_DMA_INT_F LAG_TPS_CLR</code>	发送流程停止中断标志位清除
<code>ENET_DMA_INT_F LAG_TBU_CLR</code>	发送缓冲区不可用中断标志位清除
<code>ENET_DMA_INT_F LAG_TJT_CLR</code>	发送jabber超时中断标志位清除
<code>ENET_DMA_INT_F LAG_RO_CLR</code>	接收溢出中断标志位清除
<code>ENET_DMA_INT_F LAG_TU_CLR</code>	发送下溢中断标志位清除
<code>ENET_DMA_INT_F LAG_RS_CLR</code>	接收中断标志位清除
<code>ENET_DMA_INT_F LAG_RBU_CLR</code>	接收缓冲区不可用中断标志位清除
<code>ENET_DMA_INT_F LAG_RPS_CLR</code>	接收流程停止中断标志位清除
<code>ENET_DMA_INT_F LAG_RWT_CLR</code>	接收看门狗超时中断标志位清除
<code>ENET_DMA_INT_F LAG_ET_CLR</code>	早发送中断标志位清除
<code>ENET_DMA_INT_F LAG_FBE_CLR</code>	致命总线错误中断标志位清除
<code>ENET_DMA_INT_F LAG_ER_CLR</code>	早接收中断标志位清除
<code>ENET_DMA_INT_F LAG_AI_CLR</code>	异常中断汇总中断标志位清除
<code>ENET_DMA_INT_F LAG_NI_CLR</code>	正常中断汇总中断标志位清除

枚举类型 `enet_desc_reg_enum`

表 3-242. 枚举类型 `enet_desc_reg_enum`

成员名称	功能描述
<code>ENET_RX_DESC_T ABLE</code>	接收描述符列表首地址
<code>ENET_RX_CURRE NT_DESC</code>	当前DMA控制器使用的接收描述符地址

成员名称	功能描述
<i>ENET_RX_CURRENT_BUFFER</i>	当前DMA控制器使用的接收描述符缓冲区地址
<i>ENET_TX_DESCRIPTOR_TABLE</i>	发送描述符列表首地址
<i>ENET_TX_CURRENT_DESCRIPTOR</i>	当前DMA控制器使用的发送描述符地址
<i>ENET_TX_CURRENT_BUFFER</i>	当前DMA控制器使用的发送描述符缓冲区地址

枚举类型 `enet_msc_counter_enum`

表 3-243. 枚举类型 `enet_msc_counter_enum`

成员名称	功能描述
<i>ENET_MSC_TX_SCCNT</i>	MSC 1次冲突后发送“好”帧的计数器
<i>ENET_MSC_TX_MSCCNT</i>	MSC 1次以上冲突后发送“好”帧的计数器
<i>ENET_MSC_TX_TGFCNT</i>	MSC发送“好”帧计数器
<i>ENET_MSC_RX_RFCECNT</i>	MSC CRC错误接收帧计数器
<i>ENET_MSC_RX_RFAECNT</i>	MSC对齐错误接收帧计数器
<i>ENET_MSC_RX_RGUFCNT</i>	MSC“好”单播帧接收帧计数器

枚举类型 `enet_option_enum`

表 3-244. 枚举类型 `enet_option_enum`

成员名称	功能描述
<i>FORWARD_OPTION</i>	选择配置帧通过功能相关参数
<i>DMABUS_OPTION</i>	选择配置DMA总线模式相关参数
<i>DMA_MAXBURST_OPTION</i>	选择配置DMA最大突发传输相关参数
<i>DMA_ARBITRATION_OPTION</i>	选择配置DMA仲裁相关参数
<i>STORE_OPTION</i>	选择配置存储转发模式相关参数
<i>DMA_OPTION</i>	选择配置DMA相关参数
<i>VLAN_OPTION</i>	选择配置VLAN相关参数
<i>FLOWCTL_OPTION</i>	选择配置流控相关参数
<i>HASHH_OPTION</i>	选择配置HASH_H相关参数
<i>HASHL_OPTION</i>	选择配置HASH_L相关参数

成员名称	功能描述
<i>FILTER_OPTION</i>	选择配置帧过滤器相关参数
<i>HALFDUPLEX_OPTION</i>	选择配置半双工模式相关参数
<i>TIMER_OPTION</i>	选择配置计数器相关参数
<i>INTERFRAMEGAP_OPTION</i>	选择配置帧间隔相关参数

枚举类型 `enet_mediamode_enum`

表 3-245. 枚举类型 `enet_mediamode_enum`

成员名称	功能描述
<i>ENET_AUTO_NEGOTIATION</i>	PHY自协商
<i>ENET_100M_FULLDUPLEX</i>	100Mbit/s, 全双工
<i>ENET_100M_HALFDUPLEX</i>	100Mbit/s, 半双工
<i>ENET_10M_FULLDUPLEX</i>	10Mbit/s, 全双工
<i>ENET_10M_HALFDUPLEX</i>	10Mbit/s, 半双工
<i>ENET_LOOPBACK_MODE</i>	MII模式下的回环模式

枚举类型 `enet_chksumconf_enum`

表 3-246. 枚举类型 `enet_chksumconf_enum`

成员名称	功能描述
<i>ENET_NO_AUTOCHECKSUM</i>	关闭IP帧校验和功能
<i>ENET_AUTOCHECKSUM_DROP_FAILFRAMES</i>	使能IP帧校验和功能
<i>ENET_AUTOCHECKSUM_ACCEPT_FAILFRAMES</i>	使能IP帧校验和功能，不丢弃仅有载荷错误的帧

枚举类型 `enet_frmrecept_enum`

表 3-247. 枚举类型 `enet_frmrecept_enum`

成员名称	功能描述
<i>ENET_PROMISCUOUS_MODE</i>	使能混杂模式

成员名称	功能描述
<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

枚举类型 `enet_registers_type_enum`

表 3-248. 枚举类型 `enet_registers_type_enum`

成员名称	功能描述
<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_RGUF CNT
<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_CRBADDR

枚举类型 `enet_dmadirection_enum`

表 3-249. 枚举类型 `enet_dmadirection_enum`

成员名称	功能描述
<i>ENET_DMA_TX</i>	DMA Tx描述符
<i>ENET_DMA_RX</i>	DMA Rx描述符

枚举类型 `enet_phydirection_enum`

表 3-250. 枚举类型 `enet_phydirection_enum`

成员名称	功能描述
<i>ENET_PHY_WRITE</i>	向PHY寄存器写数据
<i>ENET_PHY_READ</i>	从PHY寄存器读数据

枚举类型 `enet_regdirection_enum`

表 3-251. 枚举类型 `enet_regdirection_enum`

成员名称	功能描述
<i>ENET_REG_READ</i>	读寄存器
<i>ENET_REG_WRITE</i>	写寄存器

枚举类型 `enet_macaddress_enum`

表 3-252. 枚举类型 `enet_macaddress_enum`

成员名称	功能描述
<i>ENET_MAC_ADDR_ESS0</i>	配置MAC address 0过滤器
<i>ENET_MAC_ADDR</i>	配置MAC address 1过滤器

ESS1	
ENET_MAC_ADDR ESS2	配置MAC address 2过滤器
ENET_MAC_ADDR ESS3	配置MAC address 3过滤器

枚举类型 `enet_descstate_enum`

表 3-253. 枚举类型 `enet_descstate_enum`

成员名称	功能描述
TXDESC_COLLISION_COUNT	帧发送出去前出现的冲突次数
TXDESC_BUFFER_1_ADDR	发送帧的缓冲区地址
RXDESC_FRAME_LENGTH	接收帧长度
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_BUFFER_1_ADDR	接收帧的缓冲区地址

`enet_deinit`

函数`enet_deinit`描述见下表：

表 3-254. 函数 `enet_deinit`

函数名称	<code>enet_deinit</code>
函数原型	<code>void enet_deinit(void);</code>
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	<code>rcu_periph_reset_enable()/rcu_periph_reset_disable()/enet_initpara_reset()</code>
输入参数{in}	
-	-
输出参数{out}	
-	-
返回值	
-	-

例如：

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


enet_initpara_config

函数enet_initpara_config描述见下表：

表 3-255. 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，参考 表3-244. 枚举类型enet_option_enum ，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 参数对应值进行选取)	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_UNDEFSZ_GOODFRAMES_ENABLE / ENET_FORWARD_UNDEFSZ_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
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：

```
/* config DMA option of the ENET */
```

```
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 3-256. 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，参考 表3-245. 枚举类型enet_mediamode_enum ，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	MII模式下的回环模式
输入参数{in}	
checksum	IP帧数据校验和功能，参考 表3-246. 枚举类型enet_chksumconf_enum ，仅可选择唯一参数
ENET_NO_AUTOCH	关闭IP帧校验和功能

ECKSUM	
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，参考 表3-247. 枚举类型enet_frmrecept_enum ，仅可选择唯一参数
ENET_PROMISCUOUS_MODE	使能混杂模式
ENET_RECEIVEALL	接收所有帧
ENET_BROADCAST_FRAMES_PASS	接收广播帧
ENET_BROADCAST_FRAMES_DROP	禁止接收广播帧
输出参数{out}	
-	-
返回值	
ErrStatus	ERROR 或 SUCCESS

例如：

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

```
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 3-257. 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 或 SUCCESS
-----------	-----------------

例如:

```
/* reset all core internal registers located in CLK_TX and CLK_RX */
```

```
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 3-258. 函数 enet_rxframe_size_get

函数名称	enet_rxframe_size_get
函数原型	uint32_t enet_rxframe_size_get(void);
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	enet_rxframe_drop()
输入参数{in}	
-	-
输出参数{out}	
-	-
返回值	
uint32_t	取值范围: 0x0 - 0x3FFF

例如:

```
/* check receive frame valid */
```

```
uint32_t reval;
```

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

enet_descriptors_chain_init

函数enet_descriptors_chain_init描述见下表:

表 3-259. 函数 enet_descriptors_chain_init

函数名称	enet_descriptors_chain_init
函数原型	void enet_descriptors_chain_init(enet_dmadirection_enum direction);
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, 参考 表3-249. 枚举类型 enet_dmadirection_enum , 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 the DMA Tx/Rx descriptors's parameters 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 3-260. 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，参考 表3-249. 枚举类型 enet_dmadirection_enum ，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 the DMA Tx/Rx descriptors's parameters in ring mode */
```

```
enet_descriptors_ring_init(ENET_DMA_TX);
```

enet_frame_receive

函数enet_frame_receive描述见下表：

表 3-261. 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 或 SUCCESS

例如：

```
/* transfer received frame data to application buffer */
```

```
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 3-262. 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 或 SUCCESS

例如：

```
/* transfer buffer data of application */
```

```
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 3-263. 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3-235. 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 the transmit IP frame checksum offload calculation and insertion */
enet_descriptors_struct rx_desc;
enet_transmit_checksum_config(rx_desc, ENET_CHECKSUM_TCPUDPICMP_FULL);
```

enet_enable

函数enet_enable描述见下表：

表 3-264. 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 the ENET */
```

```
enet_enable();
```

enet_disable

函数enet_disable描述见下表：

表 3-265. 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 the ENET */
```

```
enet_disable();
```

enet_mac_address_set

函数enet_mac_address_set描述见下表：

表 3-266. 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，参考 表3-252. 枚举类型 enet_macaddress_enum ，下列参数仅可选择一个
ENET_MAC_ADDRESSES0	配置MAC address 0过滤器
ENET_MAC_ADDRESSES1	配置MAC address 1过滤器
ENET_MAC_ADDRESSES2	配置MAC address 2过滤器
ENET_MAC_ADDRESSES3	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：

```

/* config 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_ADDRESSES0, netif->hwaddr);

```

enet_mac_address_get

函数enet_mac_address_get描述见下表：

表 3-267. 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，参考 表3-252. 枚举类型enet_macaddress_enum ，下列参数仅可选择一个
ENET_MAC_ADDRESSES0	配置MAC address 0过滤器
ENET_MAC_ADDRESSES1	配置MAC address 1过滤器
ENET_MAC_ADDRESSES2	配置MAC address 2过滤器
ENET_MAC_ADDRESSES3	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 mac address */
```

```
enet_mac_address_get(ENET_MAC_ADDRESSES0, netif->hwaddr);
```

enet_flag_get

函数enet_flag_get描述见下表：

表 3-268. 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/MSC/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，参考 表3-237. 枚举类型enet_flag_enum ，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_MSC	MSC状态标志位
ENET_MAC_FLAG_MSCR	MSC接收状态标志位
ENET_MAC_FLAG_MSCT	MSC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_MSC_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_	接收看门狗超时状态标志位

<i>RWT</i>	
<i>ENET_DMA_FLAG_ET</i>	早发送状态标志位
<i>ENET_DMA_FLAG_FBE</i>	致命总线错误状态标志位
<i>ENET_DMA_FLAG_ER</i>	早接收状态标志位
<i>ENET_DMA_FLAG_AI</i>	异常中断汇总标志位
<i>ENET_DMA_FLAG_NI</i>	正常中断汇总标志位
<i>ENET_DMA_FLAG_EB_DMA_ERROR</i>	DMA错误标志位
<i>ENET_DMA_FLAG_EB_TRANSFER_ERROR</i>	发送错误标志位
<i>ENET_DMA_FLAG_EB_ACCESS_ERROR</i>	DMA访问错误标志位
<i>ENET_DMA_FLAG_MSC</i>	MSC状态标志位
<i>ENET_DMA_FLAG_WUM</i>	WUM状态标志位
<i>ENET_DMA_FLAG_TST</i>	时间戳触发状态标志位
输出参数{out}	
-	-
返回值	
FlagStatus	SET 或 RESET

例如：

```
/* check whether the specified flag bit is set */
```

```
enet_flag_get(ENET_DMA_FLAG_RS);
```

enet_flag_clear

函数enet_flag_clear描述见下表：

表 3-269. 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，参考 表3-238. 枚举类型enet_flag_clear_enum . 下列参数仅可选择一个
ENET_DMA_FLAG_TS_CLR	发送状态标志位清除
ENET_DMA_FLAG_TPS_CLR	发送流程停止状态标志位清除
ENET_DMA_FLAG_TBU_CLR	发送缓冲区不可用状态标志位清除
ENET_DMA_FLAG_TJT_CLR	发送jabber超时状态标志位清除
ENET_DMA_FLAG_RO_CLR	接收溢出状态标志位清除
ENET_DMA_FLAG_TU_CLR	发送下溢状态标志位清除
ENET_DMA_FLAG_RS_CLR	接收状态标志位清除
ENET_DMA_FLAG_RBU_CLR	接收缓冲区不可用状态标志位清除
ENET_DMA_FLAG_RPS_CLR	接受流程停止状态标志位清除
ENET_DMA_FLAG_RWT_CLR	接收看门狗超时状态标志位清除
ENET_DMA_FLAG_ET_CLR	早发送状态标志位清除
ENET_DMA_FLAG_FBE_CLR	致命总线错误状态标志位清除
ENET_DMA_FLAG_ER_CLR	早接收状态标志位清除
ENET_DMA_FLAG_AI_CLR	异常中断汇总标志位清除
ENET_DMA_FLAG_NI_CLR	正常中断汇总标志位清除
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* clear the specified flag bit */
```

```
enet_flag_clear(ENET_DMA_FLAG_RS_CLR);
```

enet_interrupt_enable

函数enet_interrupt_enable描述见下表：

表 3-270. 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/MSC/DMA中断
先决条件	-
被调用函数	-
输入参数{in}	
enet_int	ENET中断，参考 表3-239. 枚举类型enet_int_enum ，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_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_RUIE	接收缓冲区不可用中断使能

<i>BUIE</i>	
<i>ENET_DMA_INT_RPSIE</i>	接收流程停止中断使能
<i>ENET_DMA_INT_RWTIE</i>	接收看门狗超时中断使能
<i>ENET_DMA_INT_ETIE</i>	早发送中断使能
<i>ENET_DMA_INT_FBEIE</i>	致命总线错误中断使能
<i>ENET_DMA_INT_ERIE</i>	早接收中断使能
<i>ENET_DMA_INT_AIE</i>	异常中断汇总使能
<i>ENET_DMA_INT_NIE</i>	正常中断汇总使能
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* enable normal interrupt summary */
```

```
enet_interrupt_enable(ENET_DMA_INT_NIE);
```

enet_interrupt_disable

函数enet_interrupt_disable描述见下表：

表 3-271. 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，参考 表3-239. 枚举类型enet_int_enum ，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_TMSTIM</i>	时间戳触发中断屏蔽
<i>ENET_MSC_INT_RFCCEIM</i>	接收帧CRC错误中断屏蔽
<i>ENET_MSC_INT_RFAIM</i>	接收帧对齐错误中断屏蔽

AEIM	
ENET_MSC_INT_R GUFIM	接收“好”单播帧中断屏蔽
ENET_MSC_INT_T GFSCIM	仅遇到1个冲突后发送“好”帧中断屏蔽
ENET_MSC_INT_T GFMSCIM	遇到1个以上冲突后发送“好”帧中断屏蔽
ENET_MSC_INT_T GFIM	发送“好”的帧的中断屏蔽
ENET_DMA_INT_T E	发送中断使能
ENET_DMA_INT_T SIE	发送流程停止中断使能
ENET_DMA_INT_T UIE	发送缓冲区不可用中断使能
ENET_DMA_INT_T 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_T UIE	发送下溢中断使能
ENET_DMA_INT_R 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 normal interrupt summary */
enet_interrupt_disable(ENET_DMA_INT_NIE);
```

enet_interrupt_flag_get

函数enet_interrupt_flag_get描述见下表：

表 3-272. 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，参考 表3-240. 枚举类型enet_int_flag_enum ，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 AG_RWT	接收看门狗超时中断标志位
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 或 RESET

例如：

```
/* check whether the specified flag bit is set or not */
```

```
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 3-273. 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，参考 表3-241. 枚举类型enet_int_flag_clear_enum ， 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 AG_TBU_CLR	发送缓冲区不可用中断标志位清除
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 receive status flag bit */
enet_interrupt_flag_clear(ENET_DMA_INT_FLAG_RS);
```

enet_tx_enable

函数enet_tx_enable描述见下表：

表 3-274. 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 transport function of MAC and DMA */
enet_tx_enable();
```

enet_tx_disable

函数enet_tx_disable描述见下表：

表 3-275. 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 transport function of MAC and DMA */
```

```
enet_tx_disable();
```

enet_rx_enable

函数enet_rx_enable描述见下表：

表 3-276. 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 reception function of MAC and DMA */
```

```
enet_rx_enable();
```

enet_rx_disable

函数enet_rx_disable描述见下表：

表 3-277. 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 reception function of MAC and DMA */
```

```
enet_rx_disable();
```

enet_registers_get

函数enet_registers_get描述见下表：

表 3-278. 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，参考 表3-248. 枚举类型enet_registers_type_enum ，下列参数 仅可选择一个
ALL_MAC_REG	寄存器范围从ENET_MAC_CFG到ENET_MAC_FCTH
ALL_MSC_REG	寄存器范围从ENET_MSC_CTL到ENET_MSC_RGUF CNT
ALL_PTP_REG	寄存器范围从ENET_PTP_TSCTL到ENET_PTP_PPSCTL
ALL_DMA_REG	寄存器范围从ENET_DMA_BCTL到ENET_DMA_CRBADDR
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 all mac 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 3-279. 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，参考 表3-252. 枚举类型 enet_macaddress_enum ，下列参数仅可选择一个
ENET_MAC_ADDRESSES1	使能MAC地址组1过滤器
ENET_MAC_ADDRESSES2	使能MAC地址组2过滤器
ENET_MAC_ADDRESSES3	使能MAC地址组3过滤器
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* enable the MAC address 1 filter */
```

```
enet_address_filter_enable(ENET_MAC_ADDRESSES1);
```

enet_address_filter_disable

函数enet_address_filter_disable描述见下表：

表 3-280. 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，参考 表3-252. 枚举类型 enet_macaddress_enum ，下列参数仅可选择一个
ENET_MAC_ADDRESSES1	使能MAC地址组1过滤器
ENET_MAC_ADDRESSES2	使能MAC地址组2过滤器
ENET_MAC_ADDRESSES3	使能MAC地址组3过滤器
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* disable the MAC address 1 filter */
```

```
enet_address_filter_disable(ENET_MAC_ADDRESS1);
```

enet_address_filter_config

函数enet_address_filter_config描述见下表:

表 3-281. 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, 参考 表3-252. 枚举类型 enet_macaddress_enum , 下列参数仅可选择一个
ENET_MAC_ADDRESS1	使能MAC地址组1过滤器
ENET_MAC_ADDRESS2	使能MAC地址组2过滤器
ENET_MAC_ADDRESS3	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

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

例如：

```
/* config the MAC address 1 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 3-282. 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 或 SUCCESS

例如：

```
/* config PHY interface */
```

```
enet_phy_config();
```

enet_phy_write_read

函数enet_phy_write_read描述见下表：

表 3-283. 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，参考 表3-250. 枚举类型enet_phydirection_enum
<i>ENET_PHY_WRITE</i>	向PHY寄存器写数据
<i>ENET_PHY_READ</i>	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 或 SUCCESS

例如：

```
/* write 0 to PHY BCR register */

uint16_t temp_phy = 0U;

phy_state = enet_phy_write_read(ENET_PHY_WRITE, PHY_ADDRESS, PHY_REG_BCR,
&temp_phy);
```

enet_phyloopback_enable

函数enet_phyloopback_enable描述见下表：

表 3-284. 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 或 SUCCESS

例如：

```
/* enable the loopback function of PHY chip */

ErrStatus phy_state = ERROR;

phy_state = enet_phyloopback_enable();
```

enet_phyloopback_disable

函数enet_phyloopback_disable描述见下表：

表 3-285. 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 或 SUCCESS

例如：

```
/* disable the loopback function of PHY chip */
```

```
ErrStatus phy_state = ERROR;
```

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

enet_forward_feature_enable

函数enet_forward_feature_enable描述见下表：

表 3-286. 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_UNDERSZ_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 the function that forwarding undersized good frames */
```

```
enet_forward_feature_enable(ENET_FORWARD_UNDERSZ_GOODFRAMES);
```

enet_forward_feature_disable

函数enet_forward_feature_disable描述见下表：

表 3-287. 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_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 the function that forwarding undersized good frames */
```

```
enet_forward_feature_enable(ENET_FORWARD_UNDERSZ_GOODFRAMES);
```

enet_fliter_feature_enable

函数enet_fliter_feature_enable描述见下表：

表 3-288. 函数 enet_fliter_feature_enable

函数名称	enet_fliter_feature_enable
函数原型	void enet_fliter_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
<i>ENET_SRC_FILTER</i>	源地址过滤器
<i>ENET_SRC_FILTER_INVERSE</i>	源地址过滤结果逆转
<i>ENET_DEST_FILTER_INVERSE</i>	目的地址过滤结果逆转
<i>ENET_MULTICAST_FILTER_PASS</i>	接收多播帧
<i>ENET_MULTICAST_FILTER_HASH_MODE</i>	HASH多播过滤器
<i>ENET_UNICAST_FILTER_HASH_MODE</i>	HASH单播过滤器
<i>ENET_FILTER_MODE_EITHER</i>	HASH或完美过滤器功能
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* enable filter source address function */
```

```
enet_fliter_feature_enable(ENET_SRC_FILTER);
```

enet_fliter_feature_disable

函数enet_fliter_feature_disable描述见下表：

表 3-289. 函数 enet_fliter_feature_disable

函数名称	enet_fliter_feature_disable
函数原型	void enet_fliter_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
<i>ENET_SRC_FILTER</i>	源地址过滤器
<i>ENET_SRC_FILTER_INVERSE</i>	源地址过滤结果逆转
<i>ENET_DEST_FILTER_INVERSE</i>	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：

```
enet_fltir_/* enable filter source address function */
enet_fltir_feature_enable(ENET_SRC_FILTER);
```

enet_pauseframe_generate

函数enet_pauseframe_generate描述见下表：

表 3-290. 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 或 SUCCESS

例如：

```
/* generate the pause frame */
ErrStatus reval;
reval = enet_pauseframe_generate();
```

enet_pauseframe_detect_config

函数enet_pauseframe_detect_config描述见下表：

表 3-291. 函数 `enet_pauseframe_detect_config`

函数名称	<code>enet_pauseframe_detect_config</code>
函数原型	<code>void enet_pauseframe_detect_config(uint32_t detect);</code>
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
<code>ENET_MAC0_AND_UNIQUE_ADDRESS_PAUSEDDETECT</code>	除了唯一多播地址的暂停帧，MAC同时还会使用MAC0地址（ <code>ENET_MAC_ADDR0H</code> 寄存器和 <code>ENET_MAC_ADDR0L</code> 寄存器）来检测暂停帧
<code>ENET_UNIQUE_PAUSEDDETECT</code>	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：

```
/* config the pause frame detect type as ENET_UNIQUE_PAUSEDDETECT */
```

```
enet_pauseframe_detect_config(ENET_UNIQUE_PAUSEDDETECT);
```

enet_pauseframe_config

函数`enet_pauseframe_config`描述见下表：

表 3-292. 函数 `enet_pauseframe_config`

函数名称	<code>enet_pauseframe_config</code>
函数原型	<code>void enet_pauseframe_config(uint32_t pausetime, uint32_t pause_threshold);</code>
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
<code>ENET_PAUSETIME_MINUS4</code>	暂停时间 - 4个时间间隙
<code>ENET_PAUSETIME_MINUS28</code>	暂停时间 - 28个时间间隙

ENET_PAUSETIME _MINUS144	暂停时间 – 144个时间间隔
ENET_PAUSETIME _MINUS256	暂停时间 – 256个时间间隔
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* config pause time minus 4 slot times */
```

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

enet_flowcontrol_threshold_config

函数enet_flowcontrol_threshold_config描述见下表：

表 3-293. 函数 enet_flowcontrol_threshold_config

函数名称	enet_flowcontrol_threshold_config
函数原型	void enet_flowcontrol_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	1536字节

HRESHOLD_1536BYTES	
ENET_DEACTIVE_THRESHOLD_1792BYTES	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_THRESHOLD_256BYTES	256字节
ENET_ACTIVE_THRESHOLD_512BYTES	512字节
ENET_ACTIVE_THRESHOLD_768BYTES	768字节
ENET_ACTIVE_THRESHOLD_1024BYTES	1024字节
ENET_ACTIVE_THRESHOLD_1280BYTES	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 the threshold of the flow control */
```

```
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 3-294. 函数 `enet_flowcontrol_feature_enable`

函数名称	<code>enet_flowcontrol_feature_enable</code>
函数原型	<code>void enet_flowcontrol_feature_enable(uint32_t feature);</code>
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
<code>ENET_ZERO_QUANTA_PAUSE</code>	零时间片暂停控制帧自动生成
<code>ENET_TX_FLOWCONTROL</code>	发送流控功能
<code>ENET_RX_FLOWCONTROL</code>	接收流控功能
<code>ENET_BACKPRESSURE</code>	背压功能（仅在半双工模式下）
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* enable the flow control operation in the MAC */
```

```
enet_flowcontrol_feature_enable(ENET_ZERO_QUANTA_PAUSE);
```

`enet_flowcontrol_feature_disable`

函数`enet_flowcontrol_feature_disable`描述见下表：

表 3-295. 函数 `enet_flowcontrol_feature_disable`

函数名称	<code>enet_flowcontrol_feature_disable</code>
函数原型	<code>void enet_flowcontrol_feature_disable(uint32_t feature);</code>
功能描述	禁用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
<code>ENET_ZERO_QUANTA_PAUSE</code>	零时间片暂停控制帧自动生成
<code>ENET_TX_FLOWCONTROL</code>	发送流控功能
<code>ENET_RX_FLOWCONTROL</code>	接收流控功能

<i>ENET_BACK_PRES SURE</i>	背压功能（仅在半双工模式下）
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* disable the automatic zero-quanta generation function */
```

```
enet_flowcontrol_feature_disable(ENET_ZERO_QUANTA_PAUSE);
```

enet_dmaprocess_state_get

函数enet_dmaprocess_state_get描述见下表：

表 3-296. 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
<i>ENET_DMA_TX</i>	DMA发送进程
<i>ENET_DMA_RX</i>	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： <i>ENET_RX_STATE_STOPPED / ENET_RX_STATE_FETCHING /</i> <i>ENET_RX_STATE_WAITING / ENET_RX_STATE_SUSPENDED /</i> <i>ENET_RX_STATE_CLOSING / ENET_RX_STATE_QUEUING /</i> <i>ENET_TX_STATE_STOPPED / ENET_TX_STATE_FETCHING /</i> <i>ENET_TX_STATE_WAITING / ENET_TX_STATE_READING /</i> <i>ENET_TX_STATE_SUSPENDED / ENET_TX_STATE_CLOSING</i>

例如：

```
/* get the 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 3-297. 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：

```
/* enable 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 3-298. 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 recover the Rx process */
```

```
enet_rxprocess_check_recovery();
```

enet_txfifo_flush

函数enet_txfifo_flush描述见下表：

表 3-299. 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 或 SUCCESS

例如：

```
/* flush the ENET transmit 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 3-300. 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，参考 表3-242. 枚举类型enet_desc_reg_enum ，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

<code>T_BUFFER</code>	
<code>ENET_TX_DESC_TABLE</code>	发送描述符列表首地址
<code>ENET_TX_CURRENT_DESC</code>	当前DMA控制器使用的发送描述符地址
<code>ENET_TX_CURRENT_BUFFER</code>	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	
<code>uint32_t</code>	0- 0xFFFFFFFF

例如:

```
/* get the start address of the receive descriptor table */
```

```
uint32_t reval;
```

```
reval = enet_current_desc_address_get(ENET_RX_DESC_TABLE);
```

enet_desc_information_get

函数enet_desc_information_get描述见下表:

表 3-301. 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3-235. 结构体enet_descriptors_struct
输入参数{in}	
info_get	可选择的描述符信息类型, 参考 表3-253. 枚举类型enet_descstate_enum , 下列参数仅可选择一个
<code>RXDESC_BUFFER_1_SIZE</code>	接收缓冲区1大小
<code>RXDESC_BUFFER_2_SIZE</code>	接收缓冲区2大小
<code>RXDESC_FRAME_LENGTH</code>	接收帧长度
<code>TXDESC_COLLISION_COUNT</code>	帧发送出去前出现的冲突次数
<code>RXDESC_BUFFER_1_ADDR</code>	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 reception buffer 1 size */
uint32_t reval;
enet_descriptors_struct rx_desc;
reval = enet_desc_information_get(rx_desc, RXDESC_BUFFER_1_SIZE);
```

enet_missed_frame_counter_get

函数enet_missed_frame_counter_get描述见下表：

表 3-302. 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 rxcnt, txcnt;
enet_missed_frame_counter_get(&rxcnt, &txcnt);
```

enet_desc_flag_get

函数enet_desc_flag_get描述见下表：

表 3-303. 函数 `enet_desc_flag_get`

函数名称	<code>enet_desc_flag_get</code>
函数原型	<code>FlagStatus enet_desc_flag_get(enet_descriptors_struct *desc, uint32_t desc_flag);</code>
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3-235. 结构体 <code>enet_descriptors_struct</code>
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	
<code>ENET_TDES0_DB</code>	顺延位
<code>ENET_TDES0_UFE</code>	数据下溢错误位
<code>ENET_TDES0_EXD</code>	过度顺延位
<code>ENET_TDES0_VFRM</code>	VLAN帧位
<code>ENET_TDES0_ECO</code>	过度冲突位
<code>ENET_TDES0_LCO</code>	延迟冲突位
<code>ENET_TDES0_NCA</code>	无载波位
<code>ENET_TDES0_LCA</code>	载波丢失位
<code>ENET_TDES0_IPPE</code>	IP数据错误位
<code>ENET_TDES0_FRMF</code>	帧清空位
<code>ENET_TDES0_JT</code>	Jabber超时位
<code>ENET_TDES0_ES</code>	错误汇总
<code>ENET_TDES0_IPHE</code>	IP报头错误位
<code>ENET_TDES0_TTMSS</code>	发送时间戳状态位
<code>ENET_TDES0_TCHM</code>	第二地址链表模式位
<code>ENET_TDES0_TERM</code>	环形发送结束模式位
<code>ENET_TDES0_TTSN</code>	使能发送时间戳位
<code>ENET_TDES0_DPAD</code>	不填充位
<code>ENET_TDES0_DCR</code>	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_PCE RR	数据校验和错误
ENET_RDES0_CER R	CRC错误
ENET_RDES0_DBE RR	Dribble位错误
ENET_RDES0_RER R	接收错误
ENET_RDES0_RWD T	接收看门狗超时
ENET_RDES0_FRM T	帧类型
ENET_RDES0_LCO	延迟冲突位
ENET_RDES0_IPHE RR	IP帧报头校验和错误
ENET_RDES0_LDE S	最后一个描述符
ENET_RDES0_FDE S	第一个描述符
ENET_RDES0_VTA G	VLAN标签位
ENET_RDES0_OER R	溢出错误位
ENET_RDES0_LER R	长度错误位
ENET_RDES0_SAF F	未通过源地址过滤器位
ENET_RDES0_DER R	描述符错误位
ENET_RDES0_ERR S	错误汇总位
ENET_RDES0_DAF F	未通过目标地址过滤器位
ENET_RDES0_DAV	描述符可用位
输出参数{out}	
-	-
返回值	
FlagStatus	SET 或 RESET

例如：

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

```
FlagStatus reval;
```

```
enet_descriptors_struct p_txdesc;
```

```
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 3-304. 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3-235. 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_VFRM	VLAN帧位
ENET_TDES0_FRMF	帧清空位
ENET_TDES0_TCHM	第二地址链表模式位
ENET_TDES0_TERM	环形发送结束模式位
ENET_TDES0_TTS	使能发送时间戳位
ENET_TDES0_DPA	不填充位
ENET_TDES0_DCR	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 VLAN frame bit flag of ENET DMA descriptor */
```

```
enet_descriptors_struct p_txdesc;
```

```
enet_desc_flag_set(p_txdesc, ENET_TDES0_VFRM);
```

enet_desc_flag_clear

函数enet_desc_flag_clear描述见下表：

表 3-305. 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3-235. 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_VFRM	VLAN帧位
ENET_TDES0_FRM	帧清空位
ENET_TDES0_TCH	第二地址链表模式位
ENET_TDES0_TER	环形发送结束模式位
ENET_TDES0_TTS	使能发送时间戳位
ENET_TDES0_DPA	不填充位

<i>D</i>	
<i>ENET_TDES0_DCR</i>	不计算CRC位
<i>C</i>	
<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_DAV</i>	描述符可用位
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* clear VLAN frame bit flag of ENET DMA descriptor */
```

```
enet_descriptors_struct p_txdesc;
```

```
enet_desc_flag_clear(p_txdesc, ENET_TDES0_VFRM);
```

enet_desc_receive_complete_bit_enable

函数enet_desc_receive_complete_bit_enable描述见下表：

表 3-306. 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3-235. 结构体 enet_descriptors_struct
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* enable receive complete bit */
```

```
enet_descriptors_struct p_rxdesc;
```

```
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 3-307. 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3-235. 结构体 enet_descriptors_struct
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* disable receive complete bit */
enet_descriptors_struct p_rxdesc;
enet_desc_receive_complete_bit_disable(p_rxdesc);
```

enet_rxframe_drop

函数enet_rxframe_drop描述见下表：

表 3-308. 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();
```

enet_dma_feature_enable

函数enet_dma_feature_enable描述见下表：

表 3-309. 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 RxDMA does not flushes frames function */
```

```
enet_dma_feature_enable(ENET_NO_FLUSH_RXFRAME);
```

enet_dma_feature_disable

函数enet_dma_feature_disable描述见下表：

表 3-310. 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 RxDMA does not flushes frames function */
```

```
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 3-311. 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3-235. 结构体 enet_descriptors_struct
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* initialize the DMA Rx descriptors's parameters in normal 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 3-312. 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3-235. 结构体 enet_descriptors_struct
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* initialize the DMA Rx descriptors's parameters in normal 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 3-313. 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 或 SUCCESS

例如：

```
/* receive a packet data with timestamp values to application buffer in DMA normal mode */
```

```
uint32_t rx_buffer[500];

uint32_t time_stamp[2];

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 3-314. 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 或 SUCCESS

例如：

```
/* send data and timestamp values in application buffer as a transmit packet with DMA normal mode */

uint32_t tx_buffer[500];

uint32_t time_stamp[2];

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 3-315. 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 3-316. 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：

```
/* set 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 3-317. 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 power down mode */
```

```
enet_wum_feature_enable(ENET_WUM_POWER_DOWN);
```

enet_wum_feature_disable

函数enet_wum_feature_disable描述见下表：

表 3-318. 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 power down mode */
```

```
enet_wum_feature_disable(ENET_WUM_POWER_DOWN);
```

enet_msc_counters_reset

函数enet_msc_counters_reset描述见下表：

表 3-319. 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 3-320. 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_COUNTERS_FREEZE	MSC计数器冻结位
输出参数{out}	

-	-
返回值	
-	-

例如：

```
/* enable counter stop rollover function */
```

```
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 3-321. 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_COUNTERS_FREEZE	MSC计数器冻结位
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* disable counter stop rollover function */
```

```
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 3-322. 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，参考 表3-243. 枚举类型enet_msc_counter_enum ，下列参数仅可选择一个
<i>ENET_MSC_TX_SC</i> <i>CNT</i>	MSC 1次冲突后发送”好”帧的计数器
<i>ENET_MSC_TX_MS</i> <i>CCNT</i>	MSC 1次以上冲突后发送”好”帧的计数器
<i>ENET_MSC_TX_TG</i> <i>FCNT</i>	MSC发送”好”帧计数器
<i>ENET_MSC_RX_RF</i> <i>CECNT</i>	MSC CRC错误接收帧计数器
<i>ENET_MSC_RX_RF</i> <i>AECNT</i>	MSC对齐错误接收帧计数器
<i>ENET_MSC_RX_RG</i> <i>UFCNT</i>	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 MSC transmitted good frames after a single collision counter value*/
```

```
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 3-323. 函数 enet_ptp_subsecond_2_nanosecond

函数名称	enet_ptp_subsecond_2_nanosecond
函数原型	uint32_t enet_ptp_subsecond_2_nanosecond(uint32_t subsecond);
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 3-324. 函数 enet_ptp_nanosecond_2_subsecond

函数名称	enet_ptp_nanosecond_2_subsecond
函数原型	uint32_t enet_ptp_nanosecond_2_subsecond(uint32_t nanosecond);
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 3-325. 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	时间戳中断触发

AMP_INT	
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* enable PTP function for all received frames */
```

```
enet_ptp_feature_enable(ENET_RXTX_TIMESTAMP);
```

enet_ptp_feature_disable

函数enet_ptp_feature_disable描述见下表：

表 3-326. 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_RXTX_TIMESTAMP	发送/接收帧时间戳
ENET_PTP_TIMESTAMP_INTERRUPT	时间戳中断触发
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* disable PTP function for all received frames */
```

```
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 3-327. 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 D_UPDATE	加数寄存器更新
ENET_PTP_SYSTIM E_UPDATE	时间戳更新
ENET_PTP_SYSTIM E_INIT	时间戳初始化
ENET_PTP_FINEM ODE	精调模式更新系统时间戳
ENET_PTP_COARS EMODE	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：

```
/* config addend register update 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 3-328. 函数 enet_ptp_subsecond_increment_config

函数名称	enet_ptp_subsecond_increment_config
函数原型	void enet_ptp_subsecond_increment_config(uint32_t subsecond);
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 0x1F as 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 3-329. 函数 enet_ptp_timestamp_addend_config

函数名称	enet_ptp_timestamp_addend_config
函数原型	void enet_ptp_timestamp_addend_config(uint32_t add);
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：

```
/* added 0x1FFF to the accumulator register */
```

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

enet_ptp_timestamp_update_config

函数enet_ptp_timestamp_update_config描述见下表：

表 3-330. 函数 enet_ptp_timestamp_update_config

函数名称	enet_ptp_timestamp_update_config
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_T O_TIME	更新值加到系统时间
ENET_PTP_SUBST RACT_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：

```
/* initialize system time with timestamp update value */
```

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

enet_ptp_expected_time_config

函数enet_ptp_expected_time_config描述见下表：

表 3-331. 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 3-332. 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3-236. 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 3-333. 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_FINEMODE	精调
ENET_PTP_COARSEMODE	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：

```
/* gconfigure and start PTP timestamp counter*/
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 3-334. 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 3-335. 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3-236. 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 3-336. 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}	
systime_struct	PTP系统时间结构体指针，结构体成员介绍请参考 表3-236. 结构体 enet_ptp_systime_struct
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* set system time in fine method */
enet_ptp_systime_struct systime_struct;
systime_struct.second = 0x0000FFFF;
systime_struct.nanosecond = 0;
systime_struct.sign = ENET_PTP_TIME_POSITIVE;
enet_ptp_finecorrection_settime(&systime_struct);
```


enet_ptp_flag_get

函数enet_ptp_flag_get描述见下表：

表 3-337. 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 或 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 3-338. 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();
```

3.12. EXMC

外部存储器控制器EXMC, 用来访问各种片外存储器。章节[3.12.1](#)描述了EXMC的寄存器列表, 章节[3.12.2](#)对EXMC库函数进行说明。

3.12.1. 外设寄存器说明

EXMC寄存器列表如下表所示:

表 3-339. 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 (x=1, 2, 3)	NAND Flash/PC Card中断使能寄存器
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
EXMC_SDCTLx(x=0, 1)	SDRAM控制寄存器
EXMC_SDTCFGx(x=0, 1)	SDRAM时序寄存器
EXMC_SDCMD	SDRAM命令寄存器
EXMC_SDARI	SDRAM自动刷新闻隔寄存器
EXMC_SDSTAT	SDRAM状态寄存器
EXMC_SDRSCTL	SDRAM读采样控制寄存器

寄存器名称	寄存器描述
EXMC_SINIT	SPI初始化寄存器
EXMC_SRCMD	SPI读命令寄存器
EXMC_SWCMD	SPI写命令寄存器
EXMC_SIDL	SPI ID低位寄存器
EXMC_SIDH	SPI ID高位寄存器

3.12.2. 外设库函数说明

EXMC库函数列表如下表所示：

表 3-340. EXMC 库函数

库函数名称	库函数描述
exmc_norsram_deinit	复位EXMC NOR/SRAM region
exmc_norsram_struct_para_init	初始化结构体exmc_norsram_parameter_struct
exmc_norsram_init	初始化EXMC NOR/SRAM region
exmc_norsram_enable	使能EXMC NOR/PSRAM bank中某个region
exmc_norsram_disable	禁能EXMC NOR/PSRAM bank中某个region
exmc_nand_deinit	复位EXMC NAND bank
exmc_nand_struct_para_init	初始化结构体exmc_nand_parameter_struct
exmc_nand_init	初始化EXMC NAND bank
exmc_nand_enable	使能EXMC NAND bank
exmc_nand_disable	禁能EXMC NAND bank
exmc_nand_ecc_config	使能或者禁能EXMC NAND ECC功能
exmc_ecc_get	获取EXMC ECC值
exmc_pccard_deinit	复位EXMC PC卡bank
exmc_pccard_struct_para_init	初始化结构体exmc_pccard_parameter_struct
exmc_pccard_init	初始化EXMC PC卡bank
exmc_pccard_enable	使能PC卡bank
exmc_pccard_disable	禁能PC卡bank
exmc_sdram_deinit	复位EXMC SDRAM设备
exmc_sdram_struct_para_init	初始化结构体exmc_sdram_parameter_init
exmc_sdram_init	初始化EXMC SDRAM设备
exmc_sdram_struct_command_para_init	初始化结构体exmc_sdram_command_init_struct
exmc_sdram_command_config	配置SDRAM内存命令
exmc_sdram_refresh_count_set	设置自动刷新闻隔
exmc_sdram_autorefresh_number_set	设置连续自动刷新命令次数
exmc_sdram_write_protection_config	配置写保护功能
exmc_sdram_bankstatus_get	获取SDRAM device0或device1的状态
exmc_sdram_readsample_config	配置读数据的采样时钟的延迟单元
exmc_sdram_readsample_enable	使能读采样

库函数名称	库函数描述
exmc_sdram_readsample_disable	禁能读采样
exmc_sqpsram_deinit	复位EXMC SQIPSRAM
exmc_sqpsram_struct_para_init	初始化结构体exmc_sqpsram_parameter_struct
exmc_sqpsram_init	初始化EXMC SQIPSRAM
exmc_sqpsram_read_command_set	设置读命令
exmc_sqpsram_write_command_set	设置写命令
exmc_sqpsram_read_id_command_send	发送读SPI ID的命令
exmc_sqpsram_write_cmd_send	发送SPI没有地址和数据阶段的特殊命令
exmc_sqpsram_low_id_get	获取EXMC SPI ID低位数据
exmc_sqpsram_high_id_get	获取EXMC SPI ID高位数据
exmc_sqpsram_send_command_status_get	获取EXMC发送写命令位或读取ID命令的位值
exmc_flag_get	获取EXMC标志位
exmc_flag_clear	清除EXMC标志位
exmc_interrupt_enable	使能EXMC中断
exmc_interrupt_disable	禁能EXMC中断
exmc_interrupt_flag_get	获取EXMC中断标志位
exmc_interrupt_flag_clear	清除EXMC中断标志位

结构体 exmc_norsram_timing_parameter_struct

表 3-341. 结构体 exmc_norsram_timing_parameter_struct

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

结构体 exmc_norsram_parameter_struct

表 3-342. 结构体 exmc_norsram_parameter_struct

成员名称	功能描述
norsram_region	选择EXMC NOR/SRAM bank region
write_mode	写模式，同步模式或者异步模式

extended_mode	使能或者禁用扩展模式
asyn_wait	使能或者禁用异步等待功能
nwait_signal	在同步突发模式中，使能或者禁用NWAIT信号
memory_write	使能或者禁用写操作
nwait_config	配置NWAIT信号，同步访问模式时有效
wrap_burst_mode	使能或者禁用非对齐成组模式
nwait_polarity	指定NWAIT的极性
burst_mode	使能或者禁用突发模式
databus_width	指定外部存储器数据总线宽度
memory_type	指定外部存储器的类型
address_data_mux	数据线/地址线复用是否复用
read_write_timing	未用扩展模式时，读时序参数和写时序参数；或采用扩展模式时，读时序参数，结构体成员参考 表3-341. 结构体exmc_norsram_timing_parameter_struct
write_timing	使用扩展模式时，写时序参数，结构体成员参考 表3-341. 结构体exmc_norsram_timing_parameter_struct

结构体 exmc_nand_pccard_timing_parameter_struct

表 3-343. 结构体 exmc_nand_pccard_timing_parameter_struct

成员名称	功能描述
databus_hiztime	写操作时数据总线高阻时间
holdtime	地址保持时间（写操作时数据保持时间）
waittime	等待时间（保持命令的最小时间）
setuptime	地址信号的建立时间

结构体 exmc_nand_parameter_struct

表 3-344. 结构体 exmc_nand_parameter_struct

成员名称	功能描述
nand_bank	选择EXMC NAND bank
ecc_size	ECC块大小
atr_latency	ALE至RE的延迟
ctr_latency	CLE至RE的延迟
ecc_logic	配置ECC使能或禁用
databus_width	NAND flash数据宽度
wait_feature	配置NWAIT信号使能或禁用
common_space_timing	NAND Flash通用空间时序配置，结构体成员参考 表3-343. 结构体exmc_nand_pccard_timing_parameter_struct
attribute_space_timing	NAND Flash属性空间时序配置，结构体成员参考 表3-343. 结构体exmc_nand_pccard_timing_parameter_struct

结构体 `exmc_pccard_parameter_struct`

表 3-345. 结构体 `exmc_pccard_parameter_struct`

成员名称	功能描述
<code>atr_latency</code>	ALE至RE的延迟
<code>ctr_latency</code>	CLE至RE的延迟
<code>wait_feature</code>	配置NWAIT信号使能或禁用
<code>common_space_timing</code>	PC card通用空间时序配置，结构体成员参考 表3-343. 结构体 <code>exmc_nand_pccard_timing_parameter_struct</code>
<code>attribute_space_timing</code>	PC card属性空间时序配置，结构体成员参考 表3-343. 结构体 <code>exmc_nand_pccard_timing_parameter_struct</code>
<code>io_space_timing</code>	PC card I/O空间时序配置，结构体成员参考 表3-343. 结构体 <code>exmc_nand_pccard_timing_parameter_struct</code>

结构体 `exmc_sdram_timing_parameter_struct`

表 3-346. 结构体 `exmc_sdram_timing_parameter_struct`

成员名称	功能描述
<code>row_to_column_delay</code>	配置行到列延迟
<code>row_precharge_delay</code>	配置行预充电延迟
<code>write_recovery_delay</code>	配置写恢复延迟
<code>auto_refresh_delay</code>	配置自动刷新延迟
<code>row_address_select_delay</code>	配置行地址选择延迟
<code>exit_selfrefresh_delay</code>	配置退出自刷新延迟
<code>load_mode_register_delay</code>	配置加载模式寄存器延迟

结构体 `exmc_sdram_parameter_struct`

表 3-347. 结构体 `exmc_sdram_parameter_struct`

成员名称	功能描述
<code>sdram_device</code>	SDRAM设备
<code>pipeline_read_delay</code>	在CAS延迟之后再延迟多少个HCLK时钟周期才去读数据
<code>brust_read_switch</code>	使能/禁能突发读开关
<code>sdclock_config</code>	配置SDRAM banks的存储器时钟
<code>write_protection</code>	使能/禁能SDRAM banks写保护功能
<code>cas_latency</code>	配置SDRAM CAS延迟时间
<code>internal_bank_number</code>	内部Bank的个数

成员名称	功能描述
data_width	SDRAM存储器数据总线宽度
row_address_width	行地址的比特宽度
column_address_width	列地址的比特宽度
timing	读写SDRAM的时序参数，结构体成员参考 表3-346. 结构体 <code>exmc_sdram_timing_parameter_struct</code>

结构体 `exmc_sdram_command_parameter_struct`

表 3-348. 结构体 `exmc_sdram_command_parameter_struct`

成员名称	功能描述
mode_register_content	SDRAM模式寄存器内容
auto_refresh_number	在CMD=“011”时，发出多少个连续自动刷新周期
bank_select	将会被命令发送的bank
command	将被发送到SDRAM的命令

结构体 `exmc_sqpsram_parameter_struct`

表 3-349. 结构体 `exmc_sqpsram_parameter_struct`

成员名称	功能描述
sample_polarity	读数据时的采样极性
id_length	SPI PSRAM ID长度
address_bits	SPI PSRAM地址位数
command_bits	SPI PSRAM命令位数

函数 `exmc_norsram_deinit`

函数`exmc_norsram_deinit`描述见下表：

表 3-350. 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EXMC NOR/SRAM region
先决条件	-
被调用函数	-
输入参数{in}	
<code>exmc_norsram_region</code>	bank0某个region
<code>EXMC_BANK0_NORSRAM_REGIONx</code> (<code>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_init`描述见下表:

表 3-351. 函数 `exmc_norsram_struct_para_init`

函数名称	<code>exmc_norsram_struct_para_init</code>
函数原型	<code>void exmc_norsram_struct_para_init(exmc_norsram_parameter_struct* exmc_norsram_init_struct);</code>
功能描述	初始化结构体 <code>exmc_norsram_parameter_struct</code>
先决条件	-
被调用函数	-
输入参数{in}	
-	-
输出参数{out}	
<code>exmc_norsram_init_struct</code>	初始化结构体, 结构体成员参考 表3-342. 结构体 <code>exmc_norsram_parameter_struct</code>
返回值	
-	-

例如:

```
/* initialize the structure 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 3-352. 函数 `exmc_norsram_init`

函数名称	<code>exmc_norsram_init</code>
函数原型	<code>void exmc_norsram_init(exmc_norsram_parameter_struct* exmc_norsram_init_struct);</code>
功能描述	初始化EXMC NOR/SRAM region
先决条件	-
被调用函数	-
输入参数{in}	
<code>exmc_norsram_init_struct</code>	初始化结构体，结构体成员参考 表3-342. 结构体 <code>exmc_norsram_parameter_struct</code>
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;

/* configure timing parameter */

nor_timing_init_struct.asyn_access_mode = EXMC_ACCESS_MODE_A;

nor_timing_init_struct.syn_data_latency = EXMC_DATA_LATENCY_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 3-353. 函数 `exmc_norsram_enable`

函数名称	exmc_norsram_enable
函数原型	void exmc_norsram_enable(uint32_t exmc_norsram_region);
功能描述	使能EXMC NOR/SRAM bank中某个region
先决条件	-
被调用函数	-
输入参数{in}	
exmc_norsram_region	NOR/PSRAM bank0某个region
EXMC_BANK0_NORSRAM_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 3-354. 函数 exmc_norsram_disable

函数名称	exmc_norsram_disable
函数原型	void exmc_norsram_disable(uint32_t exmc_norsram_region);
功能描述	禁能EXMC NOR/SRAM bank中某个region
先决条件	-
被调用函数	-
输入参数{in}	
exmc_norsram_region	NOR/PSRAM bank0某个region
EXMC_BANK0_NORSRAM_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 3-355. 函数 exmc_nand_deinit

函数名称	exmc_nand_deinit
函数原型	void exmc_nand_deinit(uint32_t exmc_nand_bank);
功能描述	复位EXMC NAND区
先决条件	-
被调用函数	-
输入参数{in}	
exmc_nand_bank	选择NAND区
EXMC_BANKx_NAND(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_struct_para_init

函数exmc_nand_struct_para_init描述见下表：

表 3-356. 函数 exmc_nand_struct_para_init

函数名称	exmc_nand_struct_para_init
函数原型	void exmc_nand_struct_para_init(exmc_nand_parameter_struct* exmc_nand_init_struct);
功能描述	初始化结构体exmc_nand_parameter_struct
先决条件	-
被调用函数	-
输入参数{in}	
-	-
输出参数{out}	
exmc_nand_init_struct	初始化结构体，结构体成员参考 表3-344. 结构体 exmc_nand_parameter_struct
返回值	
-	-

例如：

```
/* initialize the structure nand_init_struct */  
  
exmc_nand_parameter_struct nand_init_struct;  
  
exmc_nand_struct_para_init(&nand_init_struct);
```

函数 exmc_nand_init

函数exmc_nand_init描述见下表：

表 3-357. 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EXMC NAND bank
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3-344. 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;

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_enable**

函数exmc_nand_enable描述见下表：

表 3-358. 函数 exmc_nand_enable

函数名称	exmc_nand_enable
函数原型	void exmc_nand_enable(uint32_t exmc_nand_bank);
功能描述	使能EXMC NAND bank
先决条件	-
被调用函数	-
输入参数{in}	
exmc_nand_bank	选择NAND区
EXMC_BANKx_NAND(x=1,2)	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 3-359. 函数 exmc_nand_disable

函数名称	exmc_nand_disable
函数原型	void exmc_nand_disable(uint32_t exmc_nand_bank);
功能描述	禁能EXMC NAND bank
先决条件	-
被调用函数	-
输入参数{in}	
exmc_nand_bank	选择NAND区
EXMC_BANKx_NAND(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 3-360. 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 exmc_nand_bank, ControlStatus newvalue);
功能描述	使能或者禁用EXMC NAND ECC功能
先决条件	-
被调用函数	-
输入参数{in}	

exmc_nand_bank	选择NAND区
<i>EXMC_BANKx_NAND(x=1,2)</i>	NAND某个区
输入参数{in}	
newvalue	使能或者禁用
<i>ENABLE</i>	使能
<i>DISABLE</i>	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 3-361. 函数 exmc_ecc_get

函数名称	exmc_ecc_get
函数原型	uint32_t exmc_ecc_get(uint32_t exmc_nand_bank);
功能描述	获取EXMC NAND ECC值
先决条件	-
被调用函数	-
输入参数{in}	
exmc_nand_bank	选择NAND区
<i>EXMC_BANKx_NAND(x=1,2)</i>	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 3-362. 函数 `exmc_pccard_deinit`

函数名称	<code>exmc_pccard_deinit</code>
函数原型	<code>void exmc_pccard_deinit(void);</code>
功能描述	复位EXMC PC卡bank
先决条件	-
被调用函数	-
输入参数{in}	
-	-
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* deinitialize EXMC PC card bank */
exmc_pccard_deinit();
```

函数 `exmc_pccard_struct_para_init`

函数`exmc_pccard_struct_para_init`描述见下表：

表 3-363. 函数 `exmc_pccard_struct_para_init`

函数名称	<code>exmc_pccard_struct_para_init</code>
函数原型	<code>void exmc_pccard_struct_para_init(exmc_pccard_parameter_struct* exmc_pccard_init_struct);</code>
功能描述	初始化结构体 <code>exmc_pccard_parameter_struct</code>
先决条件	-
被调用函数	-
输入参数{in}	
-	-
输出参数{out}	
<code>exmc_pccard_init_struct</code>	初始化结构体，结构体成员参考 表3-345. 结构体 <code>exmc_pccard_parameter_struct</code>
返回值	
-	-

例如：

```
/* initialize the structure pccard_init_struct */
exmc_pccard_parameter_struct pccard_init_struct;
exmc_pccard_struct_para_init(&pccard_init_struct);
```


函数 `exmc_pccard_init`

函数`exmc_pccard_init`描述见下表：

表 3-364. 函数 `exmc_pccard_init`

函数名称	<code>exmc_pccard_init</code>
函数原型	<code>void exmc_pccard_init(exmc_pccard_parameter_struct* exmc_pccard_init_struct);</code>
功能描述	初始化EXMC PC卡bank
先决条件	-
被调用函数	-
输入参数{in}	
<code>exmc_pccard_init_struct</code>	初始化结构体，结构体成员参考 表3-345. 结构体 <code>exmc_pccard_parameter_struct</code>
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;

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_enable`

函数`exmc_pccard_enable`描述见下表：

表 3-365. 函数 `exmc_pccard_enable`

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

例如：

```
/* enable PC card bank */
exmc_pccard_enable();
```

函数 `exmc_pccard_disable`

函数`exmc_pccard_disable`描述见下表：

表 3-366. 函数 `exmc_pccard_disable`

函数名称	<code>exmc_pccard_disable</code>
函数原型	<code>void exmc_pccard_disable(void);</code>
功能描述	禁能PC卡bank
先决条件	-
被调用函数	-
输入参数{in}	
-	-
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* disable PC card bank */
exmc_pccard_disable();
```

函数 `exmc_sdram_deinit`

函数`exmc_sdram_deinit`描述见下表：

表 3-367. 函数 `exmc_sdram_deinit`

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

函数原型	void exmc_sdram_deinit(uint32_t exmc_sdram_device);
功能描述	复位EXMC SDRAM设备
先决条件	-
被调用函数	-
输入参数{in}	
exmc_sdram_device	选择SDRAM设备
EXMC_SDRAM_DEVICEx(x=0, 1)	SDRAM设备x
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* deinitialize EXMC SDRAM device */
exmc_sdram_deinit(EXMC_SDRAM_DEVICE0);
```

函数 exmc_sdram_struct_para_init

函数exmc_sdram_struct_para_init描述见下表：

表 3-368. 函数 exmc_sdram_struct_para_init

函数名称	exmc_sdram_struct_para_init
函数原型	void exmc_sdram_struct_para_init (exmc_sdram_parameter_struct* exmc_sdram_init_struct);
功能描述	初始化结构体exmc_sdram_parameter_struct
先决条件	-
被调用函数	-
输入参数{in}	
-	-
输出参数{out}	
exmc_sdram_init_struct	初始化结构体，结构体成员参考 表3-347. 结构体 exmc_sdram_parameter_struct
返回值	
-	-

例如：

```
/* initialize the structure exmc_sdram_init_struct */
exmc_sdram_parameter_struct exmc_sdram_init_struct;
exmc_sdram_struct_para_init(&exmc_sdram_init_struct);
```

函数 `exmc_sdram_init`

函数`exmc_sdram_init`描述见下表:

表 3-369. 函数 `exmc_sdram_init`

函数名称	<code>exmc_sdram_init</code>
函数原型	<code>void exmc_sdram_init(exmc_sdram_parameter_struct* exmc_sdram_init_struct);</code>
功能描述	初始化EXMC SDRAM设备
先决条件	-
被调用函数	-
输入参数{in}	
<code>exmc_sdram_parameter_struct</code>	初始化结构体，结构体成员参考 表3-347. 结构体 <code>exmc_sdram_parameter_struct</code>
输出参数{out}	
-	-
返回值	
-	-

例如:

```
exmc_sdram_parameter_struct      sdram_init_struct;

sdram_init_struct.sdram_device = sdram_device;

sdram_init_struct.column_address_width = EXMC_SDRAM_COW_ADDRESS_9;

sdram_init_struct.row_address_width = EXMC_SDRAM_ROW_ADDRESS_13;

sdram_init_struct.data_width = EXMC_SDRAM_DATABUS_WIDTH_16B;

sdram_init_struct.internal_bank_number = EXMC_SDRAM_4_INTER_BANK;

sdram_init_struct.cas_latency = EXMC_CAS_LATENCY_3_SDCLK;

sdram_init_struct.write_protection = DISABLE;

sdram_init_struct.sdclk_config = EXMC_SDCLK_PERIODS_2_HCLK;

sdram_init_struct.burst_read_switch = ENABLE;

sdram_init_struct.pipeline_read_delay = EXMC_PIPELINE_DELAY_1_HCLK;

sdram_init_struct.timing = &sdram_timing_init_struct;

exmc_sdram_init(&sdram_init_struct);
```

函数 `exmc_sdram_struct_command_para_init`

函数`exmc_sdram_struct_command_para_init`描述见下表:

表 3-370. 函数 `exmc_sdram_struct_command_para_init`

函数名称	<code>exmc_sdram_struct_command_para_init</code>
函数原型	void <code>exmc_sdram_struct_command_para_init(exmc_sdram_command_parameter_struct *exmc_sdram_command_init_struct);</code>
功能描述	初始化结构体 <code>exmc_sdram_command_init_struct</code>
先决条件	-
被调用函数	-
输入参数{in}	
-	-
输出参数{out}	
<code>exmc_sdram_command_init_struct</code>	初始化结构体，结构体成员参考 表3-348. 结构体 <code>exmc_sdram_command_parameter_struct</code>
返回值	
-	-

例如：

```
/* initialize the structure exmc_sdram_command_init_struct */
exmc_sdram_command_parameter_struct t exmc_sdram_command_init_struct;
exmc_sdram_struct_command_para_init (&exmc_sdram_command_init_struct);
```

函数 `exmc_sdram_command_config`

函数`exmc_sdram_command_config`描述见下表：

表 3-371. 函数 `exmc_sdram_command_config`

函数名称	<code>exmc_sdram_command_config</code>
函数原型	void <code>exmc_sdram_command_config(exmc_sdram_command_parameter_struct* exmc_sdram_command_init_struct);</code>
功能描述	配置SDRAM内存命令
先决条件	-
被调用函数	-
输入参数{in}	
<code>exmc_sdram_command_init_struct</code>	初始化结构体，结构体成员参考 表3-348. 结构体 <code>exmc_sdram_command_parameter_struct</code>
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* configure the SDRAM memory command */
```

```

exmc_sdram_command_parameter_struct exmc_sdram_command_init_struct;

exmc_sdram_command_init_struct.mode_register_content = 0U;

exmc_sdram_command_init_struct.                auto_refresh_number                =
EXMC_SDRAM_AUTO_REFLESH_1_SDCLK;

exmc_sdram_command_init_struct.bank_select = EXMC_SDRAM_DEVICE0_SELECT;

exmc_sdram_command_init_struct.command = EXMC_SDRAM_NORMAL_OPERATION;

exmc_sdram_command_config(&exmc_sdram_command_init_struct);

```

函数 exmc_sdram_refresh_count_set

函数exmc_sdram_refresh_count_set描述见下表：

表 3-372. 函数 exmc_sdram_refresh_count_set

函数名称	exmc_sdram_refresh_count_set
函数原型	void void exmc_sdram_refresh_count_set(uint32_t exmc_count);
功能描述	设置自动刷新闻隔
先决条件	-
被调用函数	-
输入参数{in}	
exmc_count	两个连续的自动刷新命令间的SDRAM时钟个数，0x0000~0x1FFF
输出参数{out}	
-	-
返回值	
-	-

例如：

```

/* set auto-refresh interval */

exmc_sdram_refresh_count_set(0x01);

```

函数 exmc_sdram_autorefresh_number_set

函数exmc_sdram_autorefresh_number_set描述见下表：

表 3-373. 函数 exmc_sdram_autorefresh_number_set

函数名称	exmc_sdram_autorefresh_number_set
函数原型	void exmc_sdram_autorefresh_number_set(uint32_t exmc_number);
功能描述	设置连续自动刷新命令次数
先决条件	-
被调用函数	-
输入参数{in}	
exmc_number	当CMD=自动刷新命令时，将发送多少个自动刷新周期
EXMC_SDRAM_AU	自动刷新周期

<i>TO_REFLESH_x_S</i> <i>DCLK(x=1..15)</i>	
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* set the number of successive auto-refresh command */
```

```
exmc_sdram_autorefresh_number_set(EXMC_SDRAM_AUTO_REFLESH_2_SDCLK);
```

函数 **exmc_sdram_write_protection_config**

函数exmc_sdram_write_protection_config描述见下表:

表 3-374. 函数 exmc_sdram_write_protection_config

函数名称	exmc_sdram_write_protection_config
函数原型	void exmc_sdram_write_protection_config(uint32_t exmc_sdram_device, ControlStatus newvalue);
功能描述	配置写保护功能
先决条件	-
被调用函数	-
输入参数{in}	
exmc_sdram_device	指定SDRAM设备
<i>EXMC_SDRAM_DEVICE</i> <i>VICE(x=0,1)</i>	SDRAM设备
输出参数{out}	
newvalue	功能使能禁能
<i>ENABLE</i>	使能
<i>DISABLE</i>	禁能
返回值	
-	-

例如:

```
/* enable the EXMC SDRAM write protection function */
```

```
exmc_sdram_write_protection_config(EXMC_SDRAM_DEVICE0, ENABLE);
```

函数 **exmc_sdram_bankstatus_get**

函数exmc_sdram_bankstatus_get描述见下表:

表 3-375. 函数 exmc_sdram_bankstatus_get

函数名称	exmc_sdram_bankstatus_get
------	---------------------------

函数原型	uint32_t exmc_sdram_bankstatus_get(uint32_t exmc_sdram_device);
功能描述	获取SDRAM设备0或设备1状态
先决条件	-
被调用函数	-
输入参数{in}	
exmc_sdram_device	指定SDRAM设备
EXMC_SDRAM_DEVICE0 EXMC_SDRAM_DEVICE1	SDRAM设备
输出参数{out}	
-	-
返回值	
uint32_t	SDRAM设备状态
EXMC_SDRAM_DEVICE0 EXMC_SDRAM_DEVICE1	正常状态
EXMC_SDRAM_DEVICE0 EXMC_SDRAM_DEVICE1 EXMC_SDRAM_DEVICE2	自刷新状态
EXMC_SDRAM_DEVICE0 EXMC_SDRAM_DEVICE1 EXMC_SDRAM_DEVICE2 EXMC_SDRAM_DEVICE3	断电状态

例如：

```
/* get the status of SDRAM device0 */
```

```
uint32_t temp;
```

```
temp = exmc_sdram_bankstatus_get(EXMC_SDRAM_DEVICE0);
```

函数 exmc_sdram_readsample_config

函数exmc_sdram_readsample_config描述见下表：

表 3-376. 函数 exmc_sdram_readsample_config

函数名称	exmc_sdram_readsample_config
函数原型	void exmc_sdram_readsample_config(uint32_t delay_cell, uint32_t extra_hclk);
功能描述	配置读数据的采样时钟的延迟单元
先决条件	-
被调用函数	-
输入参数{in}	
delay_cell	读数据的采样时钟的延迟单元
EXMC_SDRAM_DEVICE0 EXMC_SDRAM_DEVICE1 EXMC_SDRAM_DEVICE2 EXMC_SDRAM_DEVICE3	延迟单元
输入参数{in}	

extra_hclk	读数据的采样周期
EXMC_SDRAM_RE ADSAMPLE_x_EXT RAHCLK (x=0, 1)	采样周期
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* configure the delayed sample clock of read data */
exmc_sdram_readsample_config(EXMC_SDRAM_0_DELAY_CELL,
EXMC_SDRAM_READSAMPLE_0_EXTRAHCLK);
```

函数 exmc_sdram_readsample_enable

函数exmc_sdram_readsample_enable描述见下表：

表 3-377. 函数 exmc_sdram_readsample_enable

函数名称	exmc_sdram_readsample_enable
函数原型	void exmc_sdram_readsample_enable(void);
功能描述	使能读采样
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 read sample */
exmc_sdram_readsample_enable();
```

函数 exmc_sdram_readsample_disable

函数exmc_sdram_readsample_disable描述见下表：

表 3-378. 函数 exmc_sdram_readsample_disable

函数名称	exmc_sdram_readsample_disable
函数原型	void exmc_sdram_readsample_disable(void);
功能描述	禁能读采样
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 read sample */
```

```
exmc_sdram_readsample_disable();
```

函数 exmc_sqpsram_deinit

函数exmc_sqpsram_deinit描述见下表：

表 3-379. 函数 exmc_sqpsram_deinit

函数名称	exmc_sqpsram_deinit
函数原型	void exmc_sqpsram_deinit(void);
功能描述	复位EXMC SQIPSRAM
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 SQIPSRAM */
```

```
exmc_sqpsram_deinit();
```

函数 exmc_sqpsram_struct_para_init

函数exmc_sqpsram_struct_para_init描述见下表：

表 3-380. 函数 exmc_sqpsram_struct_para_init

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

输入参数{in}	
-	-
输出参数{out}	
exmc_sqpsram_init_struct	初始化结构体，结构体成员参考 表3-349. 结构体 exmc_sqpsram_parameter_struct
返回值	
-	-

例如：

```
/* initialize the structure exmc_sqpsram_init_struct */
exmc_sqpsram_parameter_struct exmc_sqpsram_init_struct;
exmc_sqpsram_struct_para_init(&exmc_sqpsram_init_struct);
```

函数 exmc_sqpsram_init

函数exmc_sqpsram_init描述见下表：

表 3-381. 函数 exmc_sqpsram_init

函数名称	exmc_sqpsram_init
函数原型	void exmc_sqpsram_init(exmc_sqpsram_parameter_struct* exmc_sqpsram_init_struct);
功能描述	初始化EXMC SQPIPSRAM
先决条件	-
被调用函数	-
输入参数{in}	
exmc_sqpsram_init_struct	初始化结构体，结构体成员参考 表3-349. 结构体 exmc_sqpsram_parameter_struct
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* initialize EXMC SQPIPSRAM */
exmc_sqpsram_parameter_struct exmc_sqpsram_init_struct;
exmc_sqpsram_init_struct->sample_polarity = EXMC_SQPIPSRAM_SAMPLE_RISING_EDGE;
exmc_sqpsram_init_struct->id_length = EXMC_SQPIPSRAM_ID_LENGTH_64B;
exmc_sqpsram_init_struct->address_bits = EXMC_SQPIPSRAM_ADDR_LENGTH_24B;
exmc_sqpsram_init_struct->command_bits = EXMC_SQPIPSRAM_COMMAND_LENGTH_8B;
exmc_sqpsram_init(&exmc_sqpsram_init_struct);
```

函数 `exmc_sqpsram_read_command_set`

函数 `exmc_sqpsram_read_command_set` 描述见下表：

表 3-382. 函数 `exmc_sqpsram_read_command_set`

函数名称	<code>exmc_sqpsram_read_command_set</code>
函数原型	<code>void exmc_sqpsram_read_command_set(uint32_t read_command_mode, uint32_t read_wait_cycle, uint32_t read_command_code);</code>
功能描述	设置读命令
先决条件	-
被调用函数	-
输入参数{in}	
<code>read_command_mode</code>	配置PSRAM读命令模式
<code>EXMC_SQIPSRAM_READ_MODE_DISABLE</code>	非SPI模式
<code>EXMC_SQIPSRAM_READ_MODE_SPI</code>	SPI模式
<code>EXMC_SQIPSRAM_READ_MODE_SQPI</code>	SQPI模式
<code>EXMC_SQIPSRAM_READ_MODE_QPI</code>	QPI模式
输入参数{in}	
<code>read_wait_cycle</code>	地址阶段结束后等待的周期数，0..15
输入参数{in}	
<code>read_command_code</code>	读命令的命令码
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* set the read command */
```

```
exmc_sqpsram_read_command_set(EXMC_SQIPSRAM_READ_MODE_SPI,    0x01,
0x01);
```

函数 `exmc_sqpsram_write_command_set`

函数 `exmc_sqpsram_write_command_set` 描述见下表：

表 3-383. 函数 `exmc_sqpsram_write_command_set`

函数名称	<code>exmc_sqpsram_write_command_set</code>
函数原型	<code>void exmc_sqpsram_write_command_set(uint32_t write_command_mode, uint32_t write_wait_cycle, uint32_t write_command_code);</code>
功能描述	设置写命令
先决条件	-
被调用函数	-
输入参数{in}	
<code>write_command_mode</code>	配置PSRAM写命令模式
<code>EXMC_SQPIPSRAM_WRITE_MODE_DISABLE</code>	非SPI模式
<code>EXMC_SQPIPSRAM_WRITE_MODE_SPI</code>	SPI模式
<code>EXMC_SQPIPSRAM_WRITE_MODE_SQPI</code>	SQPI模式
<code>EXMC_SQPIPSRAM_WRITE_MODE_QPI</code>	QPI模式
输入参数{in}	
<code>write_wait_cycle</code>	地址阶段结束后等待的周期数，0..15
输入参数{in}	
<code>write_command_code</code>	写命令的命令码
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* set the write command */
```

```
exmc_sqpsram_write_command_set(EXMC_SQPIPSRAM_READ_MODE_SPI, 0x01, 0x01);
```

函数 `exmc_sqpsram_read_id_command_send`

函数`exmc_sqpsram_read_id_command_send`描述见下表：

表 3-384. 函数 `exmc_sqpsram_read_id_command_send`

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

函数原型	void exmc_sqpsram_read_id_command_send(void);
功能描述	发送SPI读ID指令
先决条件	-
被调用函数	-
输入参数{in}	
-	-
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* send SPI read ID command */
exmc_sqpsram_read_id_command_send();
```

函数 exmc_sqpsram_write_cmd_send

函数exmc_sqpsram_write_cmd_send描述见下表：

表 3-385. 函数 exmc_sqpsram_write_cmd_send

函数名称	exmc_sqpsram_write_cmd_send
函数原型	void exmc_sqpsram_write_cmd_send(void);
功能描述	发送SPI没有地址和数据阶段的特殊命令
先决条件	-
被调用函数	-
输入参数{in}	
-	-
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* send SPI special command */
exmc_sqpsram_write_cmd_send();
```

函数 exmc_sqpsram_low_id_get

函数exmc_sqpsram_low_id_get描述见下表：

表 3-386. 函数 exmc_sqpsram_low_id_get

函数名称	exmc_sqpsram_low_id_get
函数原型	uint32_t exmc_sqpsram_low_id_get(void);
功能描述	获取EXMC SPI ID低位数据

先决条件	-
被调用函数	-
输入参数{in}	
-	-
输出参数{out}	
-	-
返回值	
uint32_t	ID低位数据

例如：

```
/* get the EXMC SPI ID low data */
```

```
uint32_t temp;
```

```
temp = exmc_sqpsram_low_id_get();
```

函数 exmc_sqpsram_high_id_get

函数exmc_sqpsram_high_id_get描述见下表：

表 3-387. 函数 exmc_sqpsram_high_id_get

函数名称	exmc_sqpsram_high_id_get
函数原型	uint32_t exmc_sqpsram_high_id_get(void);
功能描述	获取EXMC SPI ID高位数据
先决条件	-
被调用函数	-
输入参数{in}	
-	-
输出参数{out}	
-	-
返回值	
uint32_t	ID高位数据

例如：

```
/* get the EXMC SPI ID low data */
```

```
uint32_t temp;
```

```
temp = exmc_sqpsram_high_id_get();
```

函数 exmc_sqpsram_send_command_state_get

函数exmc_sqpsram_send_command_state_get描述见下表：

表 3-388. 函数 exmc_sqpsram_send_command_state_get

函数名称	exmc_sqpsram_send_command_state_get
函数原型	FlagStatus exmc_sqpsram_send_command_state_get(uint32_t

	send_command_flag);
功能描述	获取发送写入命令位或读取ID命令的位值
先决条件	-
被调用函数	-
输入参数{in}	
send_command_flag	发送指令标志
EXMC_SEND_COMMAND_FLAG_RDID	EXMC_SRCMD_RDID标志位
EXMC_SEND_COMMAND_FLAG_SC	EXMC_SWCMD_SC flag标志位
输出参数{out}	
-	-
返回值	
value	新的发送指令标志

例如:

```
/* get the bit value of EXMC send write command bit */
```

```
FlagStatus send_command_flag_status;
```

```
send_command_flag_status=exmc_sqpsram_send_command_state_get(EXMC_SEND_COMMAND_FLAG_SC);
```

函数 exmc_flag_get

函数exmc_flag_get描述见下表:

表 3-389. 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EXMC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_PC_CARD	PC Card区
EXMC_SDRAM_DEVICE0	SDRAM设备0
EXMC_SDRAM_DEVICE1	SDRAM设备1

VICE1	
输入参数{in}	
flag	标志位
EXMC_NAND_PCCARD_FLAG_RISE	中断上升沿状态
EXMC_NAND_PCCARD_FLAG_LEVEL	中断高电平状态
EXMC_NAND_PCCARD_FLAG_FALL	中断下降沿状态
EXMC_NAND_PCCARD_FLAG_FIFOE	FIFO空标志
EXMC_SDRAM_FLAG_REFRESH	刷新错误标志
EXMC_SDRAM_FLAG_NREADY	非就绪状态
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 3-390. 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EXMC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

EXMC_SDRAM_DE VICE0	SDRAM设备0
EXMC_SDRAM_DE VICE1	SDRAM设备1
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
EXMC_SDRAM_FL AG_REFRESH	刷新错误标志
EXMC_SDRAM_FL AG_NREADY	非就绪状态
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 3-391. 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_source);
功能描述	使能EXMC中断
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
EXMC_SDRAM_DE VICE0	SDRAM设备0
EXMC_SDRAM_DE VICE1	SDRAM设备1
输入参数{in}	
interrupt_source	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
EXMC_SDRAM_IN T_REFRESH	中断刷新错误
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 3-392. 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_source);
功能描述	禁能EXMC中断
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

EXMC_SDRAM_DE VICE0	SDRAM设备0
EXMC_SDRAM_DE VICE1	SDRAM设备1
输入参数{in}	
interrupt_source	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
EXMC_SDRAM_IN T_REFRESH	中断刷新错误
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 3-393. 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 interrupt_source);
功能描述	获取EXMC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_NA ND	NAND bank1
EXMC_BANK2_NA ND	NAND bank2
EXMC_BANK3_PC CARD	PC Card区
EXMC_SDRAM_DE VICE0	SDRAM设备0

EXMC_SDRAM_DE VICE1	SDRAM设备1
输入参数{in}	
interrupt_source	中断标志位
EXMC_NAND_PCC ARD_INT_RISE	中断上升沿状态
EXMC_NAND_PCC ARD_INT_LEVEL	中断高电平状态
EXMC_NAND_PCC ARD_INT_FALL	中断下降沿状态
EXMC_SDRAM_IN T_FLAG_REFRESH	中断刷新错误标志
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 3-394. 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 interrupt_source);
功能描述	清除EXMC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_NA ND	NAND bank1
EXMC_BANK2_NA ND	NAND bank2
EXMC_BANK3_PC CARD	PC Card区
EXMC_SDRAM_DE VICE0	SDRAM设备0

EXMC_SDRAM_DE VICE1	SDRAM设备1
输入参数{in}	
interrupt_source	中断标志位
EXMC_NAND_PCC ARD_INT_RISE	中断上升沿状态
EXMC_NAND_PCC ARD_INT_LEVEL	中断高电平状态
EXMC_NAND_PCC ARD_INT_FALL	中断下降沿状态
EXMC_SDRAM_IN T_FLAG_REFRESH	中断刷新错误标志
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);
```

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

3.13.1. 外设寄存器说明

EXTI寄存器列表如下表所示：

表 3-395. EXTI 寄存器

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

3.13.2. 外设库函数说明

EXTI库函数列表如下表所示：

表 3-396. 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 3-397. 枚举类型 `exti_line_enum`

成员名称	功能描述
EXTI_0	EXTI中断线0
EXTI_1	EXTI中断线1
EXTI_2	EXTI中断线2
EXTI_3	EXTI中断线3
EXTI_4	EXTI中断线4
EXTI_5	EXTI中断线5
EXTI_6	EXTI中断线6
EXTI_7	EXTI中断线7
EXTI_8	EXTI中断线8
EXTI_9	EXTI中断线9
EXTI_10	EXTI中断线10
EXTI_11	EXTI中断线11
EXTI_12	EXTI中断线12
EXTI_13	EXTI中断线13
EXTI_14	EXTI中断线14
EXTI_15	EXTI中断线15
EXTI_16	EXTI中断线16
EXTI_17	EXTI中断线17
EXTI_18	EXTI中断线18
EXTI_19	EXTI中断线19

枚举类型 `exti_mode_enum`

表 3-398. 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 3-399. 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 3-400. 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 3-401. 函数 `exti_init`

函数名称	<code>exti_init</code>
函数原形	<code>void exti_init(exti_line_enum linex, exti_mode_enum mode, exti_trig_type_enum trig_type);</code>

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, EXTI线x, 参考 表3-397. 枚举类型exti_line_enum
输入参数{in}	
mode	EXTI模式, 参考 表3-398. 枚举类型exti_mode_enum
输入参数{in}	
trig_type	中断触发类型, 参考 表3-399. 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 3-402. 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3-397. 枚举类型exti_line_enum
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* enable the interrupt 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 3-403. 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3-397. 枚举类型exti_line_enum
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* disable the interrupt 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 3-404. 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3-397. 枚举类型exti_line_enum
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* enable the event 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 3-405. 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}	
Linux	EXTI线x, 参考 表3-397. 枚举类型exti_line_enum
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* disable the event 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 3-406. 函数 exti_software_interrupt_enable

函数名称	exti_software_interrupt_enable
函数原形	void exti_software_interrupt_enable(exti_line_enum linex);
功能描述	EXTI线x软件中断使能
先决条件	-
被调用函数	-
输入参数{in}	
linex	EXTI线x, 参考 表3-397. 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 3-407. 函数 exti_software_interrupt_disable

函数名称	exti_software_interrupt_disable
函数原形	void exti_software_interrupt_disable(exti_line_enum linex);
功能描述	EXTI线x软件中断禁能

先决条件	-
被调用函数	-
输入参数{in}	
linex	EXTI线x, 参考 表3-397. 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 3-408. 函数 exti_flag_get

函数名称	exti_flag_get
函数原形	FlagStatus exti_flag_get(exti_line_enum linex);
功能描述	获取EXTI线x标志位
先决条件	-
被调用函数	-
输入参数{in}	
linex	EXTI线x, 参考 表3-397. 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 = 0;
```

```
state = exti_flag_get(EXTI_0);
```

函数 exti_flag_clear

函数exti_flag_clear描述见下表:

表 3-409. 函数 exti_flag_clear

函数名称	exti_flag_clear
函数原形	void exti_flag_clear(exti_line_enum linex);
功能描述	清除EXTI线x标志位

先决条件	-
被调用函数	-
输入参数{in}	
linex	EXTI线x, 参考 表3-397. 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 3-410. 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3-397. 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 = 0;
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 3-411. 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3-397. 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);
```

3.14. 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[3.14.1](#)描述了FMC的寄存器列表，章节[3.14.2](#)对FMC库函数进行说明。

3.14.1. 外设寄存器说明

FMC寄存器列表如下:

表 3-412. 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	选项字节操作解锁寄存器
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	擦除/编程保护寄存器
FMC_KEY1	解锁寄存器1
FMC_STAT1	状态寄存器1
FMC_CTL1	控制寄存器1
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

3.14.2. 外设库函数说明

FMC固件库函数列举如下表:

表 3-413. 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	页擦除
fmc_mass_erase	全片擦除
fmc_bank0_erase	bank0全片擦除
fmc_bank1_erase	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_flag_get	检查标志位是否置位
fmc_flag_clear	清除FMC标志
fmc_interrupt_enable	使能FMC中断
fmc_interrupt_disable	禁能FMC中断
fmc_interrupt_flag_get	获取FMC中断标志状态
fmc_interrupt_flag_clear	清除FMC中断标志

枚举类型 fmc_state_enum

表 3-414. 枚举类型 fmc_state_enum

成员名称	功能描述
FMC_READY	操作完成
FMC_BUSY	操作进行中
FMC_PGERR	编程错误
FMC_WPERR	擦除/编程保护错误
FMC_TOERR	超时错误

枚举类型 `fmc_interrupt_enum`

表 3-415. 枚举类型 `fmc_interrupt_enum`

成员名称	功能描述
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

枚举类型 `fmc_flag_enum`

表 3-416. 枚举类型 `fmc_flag_enum`

成员名称	功能描述
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擦除/编程保护错误标志
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
FMC_FLAG_BANK1_WPERR	FMC bank1擦除/编程保护错误标志
FMC_FLAG_BANK1_END	FMC bank1操作完成标志

枚举类型 `fmc_interrupt_flag_enum`

表 3-417. 枚举类型 `fmc_interrupt_flag_enum`

成员名称	功能描述
FMC_INT_FLAG_BANK0_PGERR	FMC bank0操作错误中断标志
FMC_INT_FLAG_BANK0_WPERR	FMC bank0擦除/编程保护错误中断标志

成员名称	功能描述
FMC_INT_FLAG_B ANK0_END	FMC bank0操作完成中断标志
FMC_INT_FLAG_B ANK1_PGERR	FMC bank1操作错误中断标志
FMC_INT_FLAG_B ANK1_WPERR	FMC bank1擦除/编程保护错误中断标志
FMC_INT_FLAG_B ANK1_END	FMC bank1操作完成中断标志

函数 fmc_wscnt_set

函数fmc_wscnt_set描述见下表：

表 3-418. 函数 fmc_wscnt_set

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

例如：

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

```
fmc_wscnt_set(WS_WSCNT_1);
```

函数 fmc_unlock

函数fmc_unlock描述见下表：

表 3-419. 函数 fmc_unlock

函数名称	fmc_unlock
函数原型	void fmc_unlock(void);
功能描述	解锁FMC主编程块操作
先决条件	-
被调用函数	-

输入参数{in}	
-	-
输出参数{out}	
-	-
返回值	
-	-

例如：

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

```
fmc_unlock();
```

函数 fmc_bank0_unlock

函数fmc_bank0_unlock描述见下表：

表 3-420. 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FMC主编程块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 3-421. 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FMC主编程块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 3-422. 函数 fmc_lock

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

例如：

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

```
fmc_lock();
```

函数 fmc_bank0_lock

函数fmc_bank0_lock描述见下表：

表 3-423. 函数 fmc_bank0_lock

函数名称	fmc_bank0_lock
函数原型	void fmc_bank0_lock(void);
功能描述	锁定FMC主编程块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 3-424. 函数 fmc_bank1_lock

函数名称	fmc_bank1_lock
函数原型	void fmc_bank1_lock(void);
功能描述	锁定FMC主编程块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 3-425. 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3-414. 枚举类型fmc_state_enum 。

例如:

```
/* erase page */  
  
fmc_state_enum state;  
  
state = fmc_page_erase(0x08004000);
```

函数 fmc_mass_erase

函数fmc_mass_erase描述见下表:

表 3-426. 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3-414. 枚举类型fmc_state_enum 。

例如:

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

函数 fmc_bank0_erase

函数fmc_bank0_erase描述见下表:

表 3-427. 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3-414. 枚举类型fmc_state_enum 。
-----------------------	---

例如：

```
/* erase bank0 whole chip */
```

```
fmc_state_enum state;
```

```
state = fmc_bank0_erase();
```

函数 fmc_bank1_erase

函数fmc_bank1_erase描述见下表：

表 3-428. 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3-414. 枚举类型fmc_state_enum 。

例如：

```
/* erase bank1 whole chip */
```

```
fmc_state_enum state;
```

```
state = fmc_bank1_erase();
```

函数 fmc_word_program

函数fmc_word_program描述见下表：

表 3-429. 函数 fmc_word_program

函数名称	fmc_word_program
函数原型	fmc_state_enum fmc_word_program(uint32_t address, uint32_t data);
功能描述	在相应地址全字编程
先决条件	fmc_unlock
被调用函数	fmc_bank0_ready_wait / fmc_bank1_ready_wait
输入参数{in}	
address	编程地址
输入参数{in}	
data	编程数据

输出参数{out}	
-	-
返回值	
fmc_state_enum	FMC状态，参考 表3-414. 枚举类型fmc_state_enum 。

例如：

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

fmc_state_enum state;

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

函数 fmc_halfword_program

函数fmc_halfword_program描述见下表：

表 3-430. 函数 fmc_halfword_program

函数名称	fmc_halfword_program
函数原型	fmc_state_enum fmc_halfword_program(uint32_t address, uint16_t data);
功能描述	在相应地址半字编程
先决条件	fmc_unlock
被调用函数	fmc_bank0_ready_wait / fmc_bank1_ready_wait
输入参数{in}	
address	编程地址
输入参数{in}	
data	编程数据
输出参数{out}	
-	-
返回值	
fmc_state_enum	FMC状态，参考 表3-414. 枚举类型fmc_state_enum 。

例如：

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

fmc_state_enum state;

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

函数 ob_unlock

函数ob_unlock描述见下表：

表 3-431. 函数 ob_unlock

函数名称	ob_unlock
函数原型	void ob_unlock(void);
功能描述	解锁选项字节操作
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 3-432. 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 3-433. 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 3-434. 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3-414. 枚举类型fmc_state_enum 。

例如：

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

```
fmc_state_enum state;
```

```
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 3-435. 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3-414. 枚举类型fmc_state_enum 。

例如：

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

```
fmc_state_enum state;
```

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

函数 ob_user_write

函数ob_user_write描述见下表：

表 3-436. 函数 ob_user_write

函数名称	ob_user_write
函数原型	fmc_state_enum ob_user_write(uint8_t ob_fwdgt, uint8_t ob_deepsleep, uint8_t ob_stdby, uint8_t ob_boot);
功能描述	写用户选项字节
先决条件	ob_unlock
被调用函数	fmc_bank0_ready_wait
输入参数{in}	
ob_fwdgt	选项字节看门狗数值
OB_FWDGT_SW	软件看门狗
OB_FWDGT_HW	硬件看门狗
输入参数{in}	
ob_deepsleep	选项字节深度睡眠复位值
OB_DEEPSLEEP_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_N_RST	进入待机时不复位
OB_STDBY_R_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

<code>OB_BOOT_B1</code>	从bank1启动
输出参数{out}	
-	-
返回值	
<code>fmc_state_enum</code>	FMC状态，参考 表3-414. 枚举类型fmc_state_enum 。

例如：

```
/* configure user option byte */

fmc_state_enum state;

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 3-437. 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	写数据选项字节
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	
<code>fmc_state_enum</code>	FMC状态，参考 表3-414. 枚举类型fmc_state_enum 。

例如：

```
/* program option bytes data */

fmc_state_enum state;

state = ob_data_program(0x1ffff804, 0x56);
```

函数 ob_user_get

函数ob_user_get描述见下表：

表 3-438. 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（0x00 – 0xFF）

例如：

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

```
uint8_t user;
```

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

函数 ob_data_get

函数ob_data_get描述见下表：

表 3-439. 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;
```

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

函数 ob_write_protection_get

函数ob_write_protection_get描述见下表：

表 3-440. 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	获取写保护选项字节
先决条件	-
被调用函数	-
输入参数{in}	
-	-
输出参数{out}	
-	-
返回值	
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;
```

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

函数 ob_spc_get

函数ob_spc_get描述见下表:

表 3-441. 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	
FlagStatus	SET或RESET

例如:

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

```
FlagStatus spc;
```

```
spc = ob_spc_get();
```

函数 fmc_flag_get

函数fmc_flag_get描述见下表:

表 3-442. 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	检查标志位是否置位
先决条件	-
被调用函数	-
输入参数{in}	
flag	FMC标志, 参考 表3-416. 枚举类型fmc_flag_enum 。
输出参数{out}	
-	-
返回值	
FlagStatus	SET或RESET

例如:

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

```
FlagStatus flag;
```

```
flag = fmc_flag_get(FMC_FLAG_BANK0_END);
```

函数 fmc_flag_clear

函数fmc_flag_clear描述见下表:

表 3-443. 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	FMC标志, 参考 表3-416. 枚举类型fmc_flag_enum 。
FMC_FLAG_BANK0_PGERR	FMC bank0操作错误标志
FMC_FLAG_BANK0_WPERR	FMC bank0擦除/编程保护错误标志
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	FMC bank1擦除/编程保护错误标志

<code>_WPERR</code>	
<code>FMC_FLAG_BANK1_END</code>	FMC bank1操作完成标志
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* clear FMC flag */
fmc_flag_clear(FMC_FLAG_BANK0_END);
```

函数 `fmc_interrupt_enable`

函数 `fmc_interrupt_enable` 描述见下表：

表 3-444. 函数 `fmc_interrupt_enable`

函数名称	<code>fmc_interrupt_enable</code>
函数原型	<code>void fmc_interrupt_enable(fmc_interrupt_enum interrupt);</code>
功能描述	使能FMC中断
先决条件	-
被调用函数	-
输入参数{in}	
<code>interrupt</code>	FMC中断，参考 表3-415. 枚举类型 <code>fmc_interrupt_enum</code> 。
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 3-445. 函数 `fmc_interrupt_disable`

函数名称	<code>fmc_interrupt_disable</code>
函数原型	<code>void fmc_interrupt_disable(fmc_interrupt_enum interrupt);</code>
功能描述	禁能FMC中断
先决条件	-
被调用函数	-
输入参数{in}	

interrupt	FMC中断，参考 表3-415. 枚举类型fmc_interrupt_enum 。
输出参数{out}	
-	-
返回值	
-	-

例如：

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

```
fmc_interrupt_disable(FMC_INT_BANK0_END);
```

函数 fmc_interrupt_flag_get

函数fmc_interrupt_flag_get描述见下表：

表 3-446. 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 int_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}	
int_flag	中断标志，参考 表3-417. 枚举类型fmc_interrupt_flag_enum 。
输出参数{out}	
-	-
返回值	
FlagStatus	SET或RESET

例如：

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

```
FlagStatus flag;
```

```
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 3-447. 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 int_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}	

int_flag	中断标志，参考 表3-417. 枚举类型fmc_interrupt_flag_enum 。
输出参数{out}	
-	-
返回值	
-	-

例如：

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

```
fmc_interrupt_flag_clear(FMC_INT_FLAG_BANK0_PGERR);
```

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

3.15.1. 外设寄存器说明

FWDGT寄存器列表如下表所示：

表 3-448. 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

3.15.2. 外设库函数说明

FWDGT库函数列表如下表所示：

表 3-449. FWDGT 库函数

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

函数 fwdgt_write_enable

函数fwdgt_write_enable描述见下表：

表 3-450. 函数 fwdgt_write_enable

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

例如：

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

```
fwdgt_write_enable();
```

函数 fwdgt_write_disable

函数fwdgt_write_disable描述见下表：

表 3-451. 函数 fwdgt_write_disable

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

例如：

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

```
fwdgt_write_disable();
```

函数 fwdgt_enable

函数fwdgt_enable描述见下表：

表 3-452. 函数 fwdgt_enable

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

例如：

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

```
fwdgt_enable();
```

函数 fwdgt_prescaler_value_config

函数fwdgt_prescaler_value_config描述见下表：

表 3-453. 函数 fwdgt_prescaler_value_config

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

输出参数{out}	
-	-
返回值	
ErrStatus	ERROR / SUCCESS

例如：

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

```
ErrStatus flag;
```

```
flag = fwdgt_prescaler_value_config (FWDGT_PSC_DIV256);
```

函数 fwdgt_reload_value_config

函数fwdgt_reload_value_config描述见下表：

表 3-454. 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 3-455. 函数 fwdgt_counter_reload

函数名称	fwdgt_counter_reload
函数原型	void fwdgt_counter_reload(void);
功能描述	按照FWDGT_RLD寄存器的值重装载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 3-456. 函数 fwdgt_config

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

例如:

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

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

函数 fwdgt_flag_get

函数fwdgt_flag_get描述见下表:

表 3-457. 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);
```

3.16. GPIO

GPIO用来实现各片上设备的逻辑输入/输出功能。章节[3.16.1](#)描述了GPIO的寄存器列表，章节[3.16.2](#)对GPIO库函数进行说明。

3.16.1. 外设寄存器说明

GPIO寄存器列表如下表所示:

表 3-458. 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位清除寄存器

寄存器名称	寄存器描述
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	EXTI源选择寄存器0寄存器
AFIO_EXTISS1	EXTI源选择寄存器1寄存器
AFIO_EXTISS2	EXTI源选择寄存器2寄存器
AFIO_EXTISS3	EXTI源选择寄存器3寄存器
AFIO_PCF1	AFIO端口配置寄存器1
AFIO_PCF2	AFIO端口配置寄存器2
AFIO_PCF3	AFIO端口配置寄存器3
AFIO_PCF4	AFIO端口配置寄存器4
AFIO_PCF5	AFIO端口配置寄存器5

3.16.2. 外设库函数说明

GPIO库函数列表如下表所示：

表 3-459. GPIO 库函数

库函数名称	库函数描述
gpio_deinit	复位外设GPIO
gpio_afio_deinit	复位AFIO
gpio_init	GPIO参数初始化
gpio_bit_set	置位引脚值
gpio_bit_reset	复位引脚值
gpio_bit_write	将特定的值写入引脚
gpio_port_write	将特定的值写入一组端口
gpio_input_bit_get	获取引脚的输入值
gpio_input_port_get	获取一组端口的输入值
gpio_output_bit_get	获取引脚的输出值
gpio_output_port_get	获取一组端口的输出值
gpio_pin_remap_config	配置GPIO引脚重映射
gpio_pin_remap1_config	配置GPIO引脚重映射1
gpio_exti_source_select	选择哪个引脚作为EXTI源
gpio_ethernet_phy_select	以太网MII或RMII PHY选择
gpio_event_output_config	配置事件输出
gpio_event_output_enable	事件输出使能
gpio_event_output_disable	事件输出禁能
gpio_pin_lock	相应的引脚配置被锁定

函数 gpio_deinit

函数gpio_deinit描述见下表：

表 3-460. 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GPIO
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, H, I)
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 3-461. 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 3-462. 函数 `gpio_init`

函数名称	<code>gpio_init</code>
函数原型	<code>void gpio_init(uint32_t gpio_periph, uint32_t mode, uint32_t speed, uint32_t pin);</code>
功能描述	GPIO参数初始化
先决条件	-
被调用函数	-
输入参数{in}	
<code>gpio_periph</code>	GPIO端口
<code>GPIOx</code>	端口选择 (x = A, B, C, D, E, F, G, H, I)
输入参数{in}	
<code>mode</code>	GPIO引脚模式
<code>GPIO_MODE_AIN</code>	模拟输入模式
<code>GPIO_MODE_IN_FLOATING</code>	浮空输入模式
<code>GPIO_MODE_IPD</code>	下拉输入模式
<code>GPIO_MODE_IPU</code>	上拉输入模式
<code>GPIO_MODE_OUT_OD</code>	开漏输出模式
<code>GPIO_MODE_OUT_PP</code>	推挽输出模式
<code>GPIO_MODE_AF_OD</code>	AFIO开漏输出模式
<code>GPIO_MODE_AF_PP</code>	AFIO推挽输出模式
输出参数{out}	
<code>speed</code>	GPIO输出最大速度
<code>GPIO_OSPEED_10MHZ</code>	10MHZ
<code>GPIO_OSPEED_2MHZ</code>	20MHZ
<code>GPIO_OSPEED_50MHZ</code>	50MHZ
输入参数{in}	
<code>pin</code>	GPIO引脚
<code>GPIO_PIN_x</code>	引脚选择 (x=0..15)
<code>GPIO_PIN_ALL</code>	所有引脚
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* config PA0 as 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 3-463. 函数 gpio_bit_set

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

例如:

```
/* set PA0*/
```

```
gpio_bit_set(GPIOA, GPIO_PIN_0);
```

函数 gpio_bit_reset

函数gpio_bit_reset描述见下表:

表 3-464. 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, H, I)
输入参数{in}	
pin	GPIO引脚
GPIO_PIN_x	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：

```
/* reset PA0 */
```

```
gpio_bit_reset(GPIOA, GPIO_PIN_0);
```

函数 **gpio_bit_write**

函数gpio_bit_write描述见下表：

表 3-465. 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
<i>GPIOx</i>	端口选择 (x = A, B, C, D, E, F, G, H, I)
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 3-466. 函数 gpio_port_write

函数名称	gpio_port_write
函数原型	void gpio_port_write(uint32_t gpio_periph, uint16_t data);
功能描述	将特定的值写入端口
先决条件	-
被调用函数	-
输入参数{in}	
gpio_periph	GPIO端口
GPIOx	端口选择 (x = A, B, C, D, E, F, G, H, I)
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 3-467. 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, H, I)
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 3-468. 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, H, I)
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_port_get(GPIOA);
```

函数 gpio_output_bit_get

函数gpio_output_bit_get描述见下表:

表 3-469. 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, H, I)
输入参数{in}	
pin	GPIO引脚
GPIO_PIN_x	引脚选择 (x=0..15)

<i>GPIO_PIN_ALL</i>	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 3-470. 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
<i>GPIOx</i>	端口选择 (x = A, B, C, D, E, F, G, H, I)
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 3-471. 函数 gpio_pin_remap_config

函数名称	gpio_pin_remap_config
函数原型	void gpio_pin_remap_config(uint32_t gpio_remap, ControlStatus newvalue);
功能描述	配置GPIO引脚重映射
先决条件	-

被调用函数	-
输入参数{in}	
gpio_remap	选择重映射
<i>GPIO_SPI0_REMAP</i>	SPI0重映射
<i>GPIO_I2C0_REMAP</i>	I2C0重映射
<i>GPIO_USART0_REMAP</i>	USART0重映射
<i>GPIO_USART1_REMAP</i>	USART1重映射
<i>GPIO_USART2_PARTIAL_REMAP</i>	USART2部分重映射
<i>GPIO_USART2_FULL_REMAP</i>	USART2全部重映射
<i>GPIO_TIMER0_PARTIAL_REMAP</i>	TIMER0部分重映射
<i>GPIO_TIMER0_FULL_REMAP</i>	TIMER0全部重映射
<i>GPIO_TIMER1_PARTIAL_REMAP0</i>	TIMER1部分重映射
<i>GPIO_TIMER1_PARTIAL_REMAP1</i>	TIMER1部分重映射
<i>GPIO_TIMER1_FULL_REMAP</i>	TIMER1全部重映射
<i>GPIO_TIMER2_PARTIAL_REMAP</i>	TIMER2部分重映射
<i>GPIO_TIMER2_FULL_REMAP</i>	TIMER2全部重映射
<i>GPIO_TIMER3_REMAP</i>	TIMER3重映射
<i>GPIO_CAN0_PARTIAL_REMAP</i>	CAN0部分重映射
<i>GPIO_CAN0_FULL_REMAP</i>	CAN0全部重映射
<i>GPIO_PD01_REMAP</i>	PD01重映射
<i>GPIO_TIMER4CH3_I</i>	TIMER4通道3内部重映射
<i>GPIO_ADC0_ETRGIN_REMAP</i>	ADC0外部触发注入转换重映射
<i>GPIO_ADC0_ETRGR</i>	ADC0外部触发规则转换重映射
<i>GPIO_ADC1_ETRGIN_REMAP</i>	ADC1外部触发注入转换重映射
<i>GPIO_ADC1_ETRGR</i>	ADC1外部触发注入转换重映射

<i>EG_REMAP</i>	
<i>GPIO_ENET_REMAP</i>	ENET重映射
<i>GPIO_CAN1_REMAP</i>	CAN1重映射
<i>GPIO_SWJ_NONJTRST_REMAP</i>	全部的SWJ（JTAG-DP + SW-DP），但是不包括NJTRST
<i>GPIO_SWJ_SWDENABLE_REMAP</i>	JTAG-DP禁能，SW-DP使能
<i>GPIO_SWJ_DISABLE_REMAP</i>	JTAG-DP禁能，SW-DP禁能
<i>GPIO_SPI2_REMAP</i>	SPI2重映射
<i>GPIO_TIMER1IT1_REMAP</i>	TIMER1内部触发1重映射
<i>GPIO_PTP_PPS_REMAP</i>	以太网PTP PPS重映射
<i>GPIO_TIMER8_REMAP</i>	TIMER8重映射
<i>GPIO_TIMER9_REMAP</i>	TIMER9重映射
<i>GPIO_TIMER10_REMAP</i>	TIMER10重映射
<i>GPIO_TIMER12_REMAP</i>	TIMER12重映射
<i>GPIO_TIMER13_REMAP</i>	TIMER13重映射
<i>GPIO_EXMC_NADV_REMAP</i>	EXMC_NADV连接/断开
输入参数{in}	
newvalue	是否使能
<i>ENABLE</i>	使能
<i>DISABLE</i>	禁能
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* enable SPI0 remapping */
```

```
gpio_pin_remap_config(GPIO_SPI0_REMAP, ENABLE);
```

函数 **gpio_pin_remap1_config**

函数gpio_pin_remap1_config描述见下表：

表 3-472. 函数 gpio_pin_remap1_config

函数名称	gpio_pin_remap1_config
函数原型	void gpio_pin_remap1_config(uint8_t remap_reg, uint32_t remap, ControlStatus newvalue);
功能描述	配置GPIO引脚重映射1
先决条件	-
被调用函数	-
输入参数{in}	
remap_reg	AFIO端口配置寄存器
GPIO_PCF2	AFIO端口配置寄存器2
GPIO_PCF3	AFIO端口配置寄存器3
GPIO_PCF4	AFIO端口配置寄存器4
GPIO_PCF5	AFIO端口配置寄存器5
输入参数{in}	
remap	选择要重新映射的PIN
GPIO_PCF2_DCI_VS YNC_PG9_REMAP	DCI VSYNC重映射到PG9
GPIO_PCF2_DCI_VS YNC_PI5_REMAP	DCI VSYNC重映射到PI5
GPIO_PCF2_DCI_D0 _PC6_REMAP	DCI D0重映射到PC6
GPIO_PCF2_DCI_D0 _PH9_REMAP	DCI D0重映射到PH9
GPIO_PCF2_DCI_D1 _PC7_REMAP	DCI D1重映射到PC7
GPIO_PCF2_DCI_D1 _PH10_REMAP	DCI D1重映射到PH10
GPIO_PCF2_DCI_D2 _PE0_REMAP	DCI D2重映射到PE0
GPIO_PCF2_DCI_D2 _PG10_REMAP	DCI D2重映射到PG10
GPIO_PCF2_DCI_D2 _PH11_REMAP	DCI D2重映射到PH11
GPIO_PCF2_DCI_D3 _PE1_REMAP	DCI D3重映射到PE1
GPIO_PCF2_DCI_D3 _PG11_REMAP	DCI D3重映射到PG11
GPIO_PCF2_DCI_D3 _PH12_REMAP	DCI D3重映射到PH12
GPIO_PCF2_DCI_D4 _PE4_REMAP	DCI D4重映射到PE4
GPIO_PCF2_DCI_D4 _PH14_REMAP	DCI D4重映射到PH14

GPIO_PCF2_DCI_D5 _PD3_REMAP	DCI D5重映射到PD3
GPIO_PCF2_DCI_D5 _PI4_REMAP	DCI D5重映射到PI4
GPIO_PCF2_DCI_D6 _PE5_REMAP	DCI D6重映射到PE5
GPIO_PCF2_DCI_D6 _PI6_REMAP	DCI D6重映射到PI6
GPIO_PCF2_DCI_D7 _PE6_REMAP	DCI D7重映射到PE6
GPIO_PCF2_DCI_D7 _PI7_REMAP	D7重映射到PI7
GPIO_PCF2_DCI_D8 _PH6_REMAP	D8重映射到PH6
GPIO_PCF2_DCI_D8 _PI1_REMAP	DCI D8重映射到PI1
GPIO_PCF2_DCI_D9 _PH7_REMAP	DCI D9重映射到PH7
GPIO_PCF2_DCI_D9 _PI2_REMAP	DCI D9重映射到PI2
GPIO_PCF2_DCI_D1 0_PD6_REMAP	DCI D10重映射到PD6
GPIO_PCF2_DCI_D1 0_PI3_REMAP	DCI D10重映射到PI3
GPIO_PCF2_DCI_D1 1_PF10_REMAP	DCI D11重映射到PF10
GPIO_PCF2_DCI_D1 1_PH15_REMAP	DCI D11重映射到PH15
GPIO_PCF2_DCI_D1 2_PG6_REMAP	DCI D12重映射到PG6
GPIO_PCF2_DCI_D1 3_PG15_REMAP	DCI D12重映射到PG15
GPIO_PCF2_DCI_D1 3_PI0_REMAP	DCI D13重映射到PI0
GPIO_PCF2_DCI_HS YNC_PH8_REMAP	DCI HSYNC重映射到PH8
GPIO_PCF2_PH01_R EMAP	PH0/PH1重映射
GPIO_PCF3_TLI_B5_ PA3_REMAP	TLI B5重映射到PA3
GPIO_PCF3_TLI_VS YNC_PA4_REMAP	TLI VSYNC重映射到PA4
GPIO_PCF3_TLI_G2	TLI G2重映射到PA6

<code>_PA6_REMAP</code>	
<code>GPIO_PCF3_TLI_R6</code> <code>_PA8_REMAP</code>	TLI R6重映射到PA8
<code>GPIO_PCF3_TLI_R4</code> <code>_PA11_REMAP</code>	TLI R4重映射到PA11
<code>GPIO_PCF3_TLI_R5</code> <code>_PA12_REMAP</code>	TLI R5重映射到PA12
<code>GPIO_PCF3_TLI_R3</code> <code>_PB0_REMAP</code>	TLI R3重映射到PB0
<code>GPIO_PCF3_TLI_R6</code> <code>_PB1_REMAP</code>	TLI R6重映射到PB1
<code>GPIO_PCF3_TLI_B6</code> <code>PB8_REMAP</code>	TLI B6重映射到PB8
<code>GPIO_PCF3_TLI_B7</code> <code>PB9_REMAP</code>	TLI B7重映射到PB9
<code>GPIO_PCF3_TLI_G4</code> <code>_PB10_REMAP</code>	TLI G4重映射到PB10
<code>GPIO_PCF3_TLI_G5</code> <code>_PB11_REMAP</code>	TLI G5重映射到PB11
<code>GPIO_PCF3_TLI_HS</code> <code>YNC_PC6_REMAP</code>	TLI HSYNC重映射到PC6
<code>GPIO_PCF3_TLI_G6</code> <code>_PC7_REMAP</code>	TLI G6重映射到PC7
<code>GPIO_PCF3_TLI_R2</code> <code>_PC10_REMAP</code>	TLI R2重映射到PC10
<code>GPIO_PCF3_TLI_G7</code> <code>_PD3_REMAP</code>	TLI G7重映射到PD3
<code>GPIO_PCF3_TLI_B2</code> <code>PD6_REMAP</code>	TLI B2重映射到PD6
<code>GPIO_PCF3_TLI_B3</code> <code>PD10_REMAP</code>	TLI B3重映射到PD10
<code>GPIO_PCF3_TLI_B0</code> <code>PE4_REMAP</code>	TLI B0重映射到PE4
<code>GPIO_PCF3_TLI_G0</code> <code>_PE5_REMAP</code>	TLI G0重映射到PE5
<code>GPIO_PCF3_TLI_G1</code> <code>_PE6_REMAP</code>	TLI G1重映射到PE6
<code>GPIO_PCF3_TLI_G3</code> <code>_PE11_REMAP</code>	TLI G3重映射到PE11
<code>GPIO_PCF3_TLI_B4</code> <code>PE12_REMAP</code>	TLI B4重映射到PE12
<code>GPIO_PCF3_TLI_DE</code> <code>_PE13_REMAP</code>	TLI DE重映射到PE13

<i>GPIO_PCF3_TLI_CLK_PE14_REMAP</i>	TLI CLK重映射到PE14
<i>GPIO_PCF3_TLI_R7_PE15_REMAP</i>	TLI R7重映射到PE15
<i>GPIO_PCF3_TLI_DE_PF10_REMAP</i>	TLI DE重映射到PF10
<i>GPIO_PCF3_TLI_R7_PG6_REMAP</i>	TLI R7重映射到PG6
<i>GPIO_PCF3_TLI_CLK_PG7_REMAP</i>	TLI CLK重映射到PG7
<i>GPIO_PCF3_TLI_G3_PG10_REMAP</i>	TLI G3重映射到PG10
<i>GPIO_PCF3_TLI_B2_PG10_REMAP</i>	TLI B2重映射到PG10
<i>GPIO_PCF3_TLI_B3_PG11_REMAP</i>	TLI B3重映射到PG11
<i>GPIO_PCF4_TLI_B4_PG12_REMAP</i>	B4重映射到PG12
<i>GPIO_PCF4_TLI_B1_PG12_REMAP</i>	B1重映射到PG12
<i>GPIO_PCF4_TLI_R0_PH2_REMAP2</i>	R0重映射到PH2
<i>GPIO_PCF4_TLI_R1_PH3_REMAP</i>	TLI R1重映射到PH3
<i>GPIO_PCF4_TLI_R2_PH8_REMAP</i>	TLI R2重映射到PH8
<i>GPIO_PCF4_TLI_R3_PH9_REMAP</i>	TLI R3重映射到PH9
<i>GPIO_PCF4_TLI_R4_PH10_REMAP</i>	TLI R4重映射到PH10
<i>GPIO_PCF4_TLI_R5_PH11_REMAP</i>	TLI R5重映射到PH11
<i>GPIO_PCF4_TLI_R6_PH12_REMAP</i>	TLI R6重映射到PH12
<i>GPIO_PCF4_TLI_G2_PH13_REMAP</i>	TLI G2重映射到PH13
<i>GPIO_PCF4_TLI_G3_PH14_REMAP</i>	TLI G3重映射到PH14
<i>GPIO_PCF4_TLI_G4_PH15_REMAP</i>	TLI G4重映射到PH15
<i>GPIO_PCF4_TLI_G5_PI0_REMAP</i>	TLI G5重映射到PI0
<i>GPIO_PCF4_TLI_G6</i>	TLI G6重映射到PI1

<code>_PI1_REMAP</code>	
<code>GPIO_PCF4_TLI_G7</code> <code>_PI2_REMAP</code>	TLI G7重映射到PI2
<code>GPIO_PCF4_TLI_B4</code> <code>PI4_REMAP</code>	TLI B4重映射到PI4
<code>GPIO_PCF4_TLI_B5</code> <code>PI5_REMAP</code>	TLI B5重映射到PI5
<code>GPIO_PCF4_TLI_B6</code> <code>PI6_REMAP</code>	TLI B6重映射到PI6
<code>GPIO_PCF4_TLI_B7</code> <code>PI7_REMAP</code>	TLI B7重映射到PI7
<code>GPIO_PCF4_TLI_VS</code> <code>YNC_PI9_REMAP</code>	TLI VSYNC重映射到PI9
<code>GPIO_PCF4_TLI_HS</code> <code>YNC_PI10_REMAP</code>	TLI HSYNC重映射到PI10
<code>GPIO_PCF4_TLI_R0</code> <code>_PH4_REMAP</code>	TLI R0重映射到PH4
<code>GPIO_PCF4_TLI_R1</code> <code>_PI3_REMAP</code>	TLI R1重映射到PI3
<code>GPIO_PCF4_SPI1_S</code> <code>CK_PD3_REMAP</code>	SPI1 SCK重映射到PD3
<code>GPIO_PCF4_SPI2_M</code> <code>OSI_PD6_REMAP</code>	SPI2 MOSI重映射到PD6
<code>GPIO_PCF5_I2C2_R</code> <code>EMAP0</code>	I2C2重映射0
<code>GPIO_PCF5_I2C2_R</code> <code>EMAP1</code>	I2C2重映射1
<code>GPIO_PCF5_TIMER1</code> <code>_CH0_REMAP</code>	TIMER1 CH0重映射到PA5
<code>GPIO_PCF5_TIMER4</code> <code>_REMAP</code>	TIMER4 CH0重映射
<code>GPIO_PCF5_TIMER7</code> <code>_CHON_REMAP0</code>	TIMER7 CHON重映射0
<code>GPIO_PCF5_TIMER7</code> <code>_CHON_REMAP1</code>	TIMER7 CHON重映射1
<code>GPIO_PCF5_TIMER7</code> <code>_CH_REMAP</code>	TIMER7 CH重映射
<code>GPIO_PCF5_I2C1_R</code> <code>EMAP0</code>	I2C1重映射0
<code>GPIO_PCF5_I2C1_R</code> <code>EMAP1</code>	I2C1重映射1
<code>GPIO_PCF5_SPI1_N</code> <code>SCK_REMAP0</code>	SPI1 NSS/SCK重映射0

GPIO_PCF5_SPI1_N SCK_REMAP1	SPI1 NSS/SCK重映射1
GPIO_PCF5_SPI1_IO _REMAP0	SPI1 MISO/MOSI重映射0
GPIO_PCF5_SPI1_IO _REMAP1	SPI1 MISO/MOSI重映射1
GPIO_PCF5_UART3 _REMAP	UART3重映射
GPIO_PCF5_TIMER1 1_REMAP	TIMER11重映射
GPIO_PCF5_CAN0_ ADD_REMAP	CAN0附加重映射
GPIO_PCF5_ENET_ TXD3_REMAP	ETH_TXD3重映射到PE2
GPIO_PCF5_PPS_HI _REMAP	ETH_PPS_OUT重映射到PG8
GPIO_PCF5_ENET_ TXD01_REMAP	ETH_TX_EN/ETH_TXD0/ETH_TXD1重映射
GPIO_PCF5_ENET_ CRSCOL_REMAP	ETH_MII_CRS/ETH_MII_COL重映射
GPIO_PCF5_ENET_ RX_HI_REMAP	ETH_RXD2/ETH_RXD3/ETH_RX_ER重映射
GPIO_PCF5_UART6 _REMAP	UART6重映射
GPIO_PCF5_USART 5_CK_PG7_REMAP	USART5 CK重映射到PG7
GPIO_PCF5_USART 5_RTS_PG12_REMA P	USART5 RTS重映射到PG12
GPIO_PCF5_USART 5_CTS_PG13_REMA P	USART5 CTS重映射到PG13
GPIO_PCF5_USART 5_TX_PG14_REMAP	USART5 TX重映射到PG14
GPIO_PCF5_USART 5_RX_PG9_REMAP	USART5 RX重映射到PG9
GPIO_PCF5_EXMC_ SDNWE_PC0_REMA P	EXMC SDNWE重映射到PC0
GPIO_PCF5_EXMC_ SDCKE0_PC3_REMA P	EXMC SDCKE0重映射到PC3
GPIO_PCF5_EXMC_ SDCKE1_PC5_REMA P	EXMC SDCKE1重映射到PB5

<i>SDCKE1_PB5_REMAP</i>	
<i>GPIO_PCF5_EXMC_SDNE0_PC2_REMAP</i>	EXMC SDNE0重映射到PC2
<i>GPIO_PCF5_EXMC_SDNE1_PB6_REMAP</i>	EXMC SDNE1重映射到PB6
输入参数{in}	
newvalue	是否使能
<i>ENABLE</i>	使能
<i>DISABLE</i>	禁能
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/*enable GPIO_PCF2 UART3 remapping*/
```

```
gpio_pin_remap_config(GPIO_PCF2, GPIO_PCF5_UART3_REMAP, ENABLE);
```

函数 gpio_ethernet_phy_select

函数gpio_ethernet_phy_select描述见下表：

表 3-473. 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 enet_sel);
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}	
enet_sel	以太网PHY选择
<i>GPIO_ENET_PHY_MII</i>	以太网MAC选择连接MII
<i>GPIO_ENET_PHY_RMII</i>	以太网MAC选择连接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);
```

函数 gpio_exti_source_select

函数gpio_exti_source_select描述见下表：

表 3-474. 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
GPIOT_PORT_SOURCE_GPIOx	源端口选择 (x = A, B, C, D, E, F, G, H, I)
输入参数{in}	
gpio_outputpin	源端口引脚
GPIO_PIN_SOURCE_x	引脚选择 (x=0..15)
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* configure nfig 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 3-475. 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)
输入参数{in}	
gpio_outputpin	GPIO事件输出引脚
GPIO_EVENT_PIN	引脚选择 (x=0..15)

_x	
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* configure 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 3-476. 函数 gpio_event_output_enable

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

例如：

```
/* enable GPIO pin event output */
```

```
gpio_event_output_enable(void);
```

函数 gpio_event_output_disable

函数gpio_event_output_disable描述见下表：

表 3-477. 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 GPIO pin event output */
```

```
gpio_event_output_disable(void);
```

函数 gpio_pin_lock

函数gpio_pin_lock描述见下表:

表 3-478. 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, H, I)
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);
```

3.17. HAU

哈希处理器应用于信息安全。支持应用于多种场合的安全哈希算法 (SHA-1, SHA-224和SHA-256), 消息摘要算法 (MD5) 和哈希运算消息认证码 (HMAC)。HAU寄存器列举在章节[3.17.1](#), HAU固件库函数介绍在章节[3.17.2](#)。

3.17.1. 外设寄存器说明

HAU寄存器列表如下表所示：

表 3-479. HAU 寄存器

寄存器名称	寄存器描述
HAU_CTL	控制寄存器
HAU_DI	数据输入寄存器
HAU_CFG	配置寄存器
HAU_DO0	数据输出寄存器0
HAU_DO1	数据输出寄存器1
HAU_DO2	数据输出寄存器2
HAU_DO3	数据输出寄存器3
HAU_DO4	数据输出寄存器4
HAU_DO5	数据输出寄存器5
HAU_DO6	数据输出寄存器6
HAU_DO7	数据输出寄存器7
HAU_INTEN	中断使能寄存器
HAU_STAT	状态寄存器

3.17.2. 外设库函数说明

HAU库函数列表如下表所示：

表 3-480. HAU 库函数

库函数名称	库函数描述
hau_deinit	复位HAU外设
hau_init	初始化HAU外设参数
hau_init_struct_para_init	初始化结构体hau_initpara
hau_reset	复位HAU内核
hau_last_word_validbits_num_config	配置消息最新字有效位数
hau_data_write	写数据到IN FIFO
hau_infifo_words_num_get	返回已经写入IN FIFO的字数目
hau_digest_read	读消息摘要结果
hau_digest_calculation_enable	使能摘要计算
hau_multiple_single_dma_config	配置多个或单个DMA使用，并在DMA传输结束时进行摘要计算
hau_dma_enable	使能HAU DMA接口
hau_dma_disable	除能HAU DMA接口
hau_hash_sha_1	在HASH模式下使用SHA1计算摘要
hau_hmac_sha_1	在HMAC模式下使用SHA1计算摘要
hau_hash_sha_224	在HASH模式下使用SHA224计算摘要
hau_hmac_sha_224	在HMAC模式下使用SHA224计算摘要

库函数名称	库函数描述
hau_hash_sha_256	在HASH模式下使用SHA256计算摘要
hau_hmac_sha_256	在HMAC模式下使用SHA256计算摘要
hau_hash_md5	在HASH模式下使用MD5计算摘要
hau_hmac_md5	在HMAC模式下使用MD5计算摘要
hau_flag_get	获取HAU标志状态
hau_flag_clear	清除HAU标志状态
hau_interrupt_enable	使能HAU中断
hau_interrupt_disable	除能HAU中断
hau_interrupt_flag_get	获取HAU中断标志状态
hau_interrupt_flag_clear	清除HAU中断标志状态

结构体 hau_init_parameter_struct

表 3-481. 结构体 hau_init_parameter_struct

成员名称	功能描述
algo	算法选择: HAU_ALGO_SHA1, HAU_ALGO_SHA224, HAU_ALGO_SHA256, HAU_ALGO_MD5
mode	HAU模式选择: HAU_MODE_HASH, HAU_MODE_HMAC
datatype	数据类型模式: HAU_SWAPPING_32BIT, HAU_SWAPPING_16BIT, HAU_SWAPPING_8BIT, HAU_SWAPPING_1BIT
keytype	密钥长度模式: HAU_KEY_SHORTER_64, HAU_KEY_LONGGER_64

结构体 hau_digest_parameter_struct

表 3-482. 结构体 hau_digest_parameter_struct

成员名称	功能描述
out[8]	消息摘要结果0-7

函数 hau_deinit

函数hau_deinit描述见下表:

表 3-483. 函数 hau_deinit

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

-	-
返回值	
-	-

例如：

```
/* reset the HAU peripheral */
```

```
hau_deinit();
```

函数 `hau_init`

函数 `hau_init` 描述见下表：

表 3-484. 函数 `hau_init`

函数名称	<code>hau_init</code>
函数原形	<code>void hau_init(hau_init_parameter_struct* initpara);</code>
功能描述	初始化HAU外设参数
先决条件	-
被调用函数	-
输入参数{in}	
<code>initpara</code>	HAU初始化参数结构体，参考结构体 表3-481. 结构体 <code>hau_init_parameter_struct</code>
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* initialize the HAU peripheral parameters */
```

```
hau_init_parameter_struct hau_initpara;
```

```
hau_initpara.algo = HAU_ALGO_MD5;
```

```
hau_initpara.mode = HAU_MODE_HASH;
```

```
hau_initpara.datatype = HAU_SWAPPING_8BIT;
```

```
hau_initpara.keytype = HAU_KEY_SHORTER_64;
```

```
hau_init(&hau_initpara);
```

函数 `hau_init_struct_para_init`

函数 `hau_init_struct_para_init` 描述见下表：

表 3-485. 函数 `hau_init_struct_para_init`

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

函数原形	void hau_init_struct_para_init (hau_init_parameter_struct* initpara);
功能描述	初始化结构体hau_initpara
先决条件	-
被调用函数	-
输入参数{in}	
-	-
输出参数{out}	
initpara	HAU初始化参数结构体，参考结构体 表3-481. 结构体 <u>hau_init_parameter_struct</u>
返回值	
-	-

例如：

```
/* initialize the HAU peripheral parameters */
```

```
hau_init_parameter_struct initpara;
```

```
hau_init_struct_para_init(&initpara);
```

函数 hau_reset

函数hau_reset描述见下表：

表 3-486. 函数 hau_reset

函数名称	hau_reset
函数原形	void hau_reset(void);
功能描述	复位HAU内核
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 HAU processor core */
```

```
hau_reset();
```

函数 hau_last_word_validbits_num_config

函数hau_last_word_validbits_num_config描述见下表：

表 3-487. 函数 hau_last_word_validbits_num_config

函数名称	hau_last_word_validbits_num_config
函数原形	void hau_last_word_validbits_num_config(uint32_t valid_num);
功能描述	配置消息最新字有效位数
先决条件	-
被调用函数	-
输入参数{in}	
valid_num	消息最新字有效位数(0x00 – 0x1F)
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* configure the number of valid bits in last word of the message */
```

```
hau_last_word_validbits_num_config(0x10);
```

函数 hau_data_write

函数hau_data_write描述见下表：

表 3-488. 函数 hau_data_write

函数名称	hau_data_write
函数原形	void hau_data_write(uint32_t data);
功能描述	写数据到IN FIFO
先决条件	-
被调用函数	-
输入参数{in}	
data	所写的的数据(0x0 – 0xFFFFFFFF)
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* write data to the IN FIFO */
```

```
hau_data_write(0x10);
```

函数 hau_infifo_words_num_get

函数hau_infifo_words_num_get描述见下表：

表 3-489. 函数 hau_infifo_words_num_get

函数名称	hau_infifo_words_num_get
------	--------------------------

函数原形	uint32_t hau_infifo_words_num_get(void);
功能描述	返回已经写入IN FIFO的字数目
先决条件	-
被调用函数	-
输入参数{in}	
-	-
输出参数{out}	
-	-
返回值	
uint32_t	0x0 – 0xFFFFFFFF

例如：

```
/* return the number of words already written into the IN FIFO */
```

```
uint32_t num;
```

```
num = hau_infifo_words_num_get();
```

函数 hau_digest_read

函数hau_digest_read描述见下表：

表 3-490. 函数 hau_digest_read

函数名称	hau_digest_read
函数原形	void hau_digest_read(hau_digest_parameter_struct* digestpara);
功能描述	读消息摘要结果
先决条件	-
被调用函数	-
输入参数{in}	
-	-
输出参数{out}	
digestpara	HAU摘要参数结构体，参考结构体 表3-482. 结构体 hau_digest_parameter_struct
返回值	
-	-

例如：

```
/* read the message digest result */
```

```
hau_digest_parameter_struct digestpara;
```

```
hau_digest_read(&digestpara);
```

函数 hau_digest_calculation_enable

函数hau_digest_calculation_enable描述见下表：

表 3-491. 函数 hau_digest_calculation_enable

函数名称	hau_digest_calculation_enable
函数原形	void hau_digest_calculation_enable(void);
功能描述	使能摘要计算
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 digest calculation */
```

```
hau_digest_calculation_enable();
```

函数 hau_multiple_single_dma_config

函数hau_multiple_single_dma_config描述见下表:

表 3-492. 函数 hau_multiple_single_dma_config

函数名称	hau_multiple_single_dma_config
函数原形	void hau_multiple_single_dma_config(uint32_t multi_single);
功能描述	配置使用多DMA或单DMA，并确定是否在DMA传输结束后计算摘要
先决条件	-
被调用函数	-
输入参数{in}	
multi_single	Multiple or single
SINGLE_DMA_AUTO_DIGEST	在DMA传输完成后消息填充和计算摘要
MULTIPLE_DMA_NO_DIGEST	需要多次DMA传输，在DMA传输结束时硬件不自动将CALEN位置1
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* configure multiple or single DMA is used, and digest calculation at the end of a DMA transfer or not */
```

```
hau_multiple_single_dma_config(SINGLE_DMA_AUTO_DIGEST);
```

函数 hau_dma_enable

函数hau_dma_enable描述见下表：

表 3-493. 函数 hau_dma_enable

函数名称	hau_dma_enable
函数原形	void hau_dma_enable(void);
功能描述	使能HAU DMA接口
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 HAU DMA interface */
```

```
hau_dma_enable();
```

函数 hau_dma_disable

函数hau_dma_disable描述见下表：

表 3-494. 函数 hau_dma_disable

函数名称	hau_dma_disable
函数原形	void hau_dma_disable(void);
功能描述	除能HAU DMA接口
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 HAU DMA interface */
```

```
hau_dma_disable();
```

函数 hau_hash_sha_1

函数hau_hash_sha_1描述见下表：

表 3-495. 函数 hau_hash_sha_1

函数名称	hau_hash_sha_1
函数原形	ErrStatus hau_hash_sha_1(uint8_t *input, uint32_t in_length, uint8_t output[20]);
功能描述	在HASH模式下使用SHA1计算摘要
先决条件	-
被调用函数	-
输入参数{in}	
input	指向输入缓存区
输入参数{in}	
in_length	输入缓存区长度
输出参数{out}	
output	摘要结果
返回值	
ErrStatus	SUCCESS或ERROR

例如：

```
/* calculate digest using SHA1 in HASH mode */
ErrStatus status;

const uint8_t input[16] = {0x01};

static uint8_t output[20];

status = hau_hash_sha_1((uint8_t *)input, 0x10, output);
```

函数 hau_hmac_sha_1

函数hau_hmac_sha_1描述见下表：

表 3-496. 函数 hau_hmac_sha_1

函数名称	hau_hmac_sha_1
函数原形	ErrStatus hau_hmac_sha_1(uint8_t *key, uint32_t keysize, uint8_t *input, uint32_t in_length, uint8_t output[20]);
功能描述	在HMAC模式下使用SHA1计算摘要
先决条件	-
被调用函数	-
输入参数{in}	
key	指向用于HMAC模式的密钥
输入参数{in}	
keysize	用于HMAC模式的密钥长度

输入参数{in}	
input	指向输入缓存区
输入参数{in}	
in_length	输入缓存区长度
输出参数{out}	
output	摘要结果
返回值	
ErrStatus	SUCCESS或ERROR

例如:

```
/* calculate digest using SHA1 in HMAC mode */
```

```
ErrStatus status;
```

```
const uint8_t key[16] = {0x01};
```

```
const uint8_t input[16] = {0x01};
```

```
static uint8_t output[20];
```

```
status = hau_hmac_sha_1((uint8_t *)key, 0x10, (uint8_t *)input, 0x10, output);
```

函数 hau_hash_sha_224

函数hau_hash_sha_224描述见下表:

表 3-497. 函数 hau_hash_sha_224

函数名称	hau_hash_sha_224
函数原形	ErrStatus hau_hash_sha_224(uint8_t *input, uint32_t in_length, uint8_t output[28]);
功能描述	在HASH模式下使用SHA224计算摘要
先决条件	-
被调用函数	-
输入参数{in}	
input	指向输入缓存区
输入参数{in}	
in_length	输入缓存区长度
输出参数{out}	
output	摘要结果
返回值	
ErrStatus	SUCCESS或ERROR

例如:

```
/* calculate digest using SHA224 in HASH mode */
```

```
ErrStatus status;
```

```
const uint8_t input[16] = {0x01};

static uint8_t output[20];

status = hau_hash_sha_224((uint8_t *)input, 0x10, output);
```

函数 hau_hmac_sha_224

函数hau_hmac_sha_224描述见下表：

表 3-498. 函数 hau_hmac_sha_224

函数名称	hau_hmac_sha_224
函数原形	ErrStatus hau_hmac_sha_224(uint8_t *key, uint32_t keysize, uint8_t *input, uint32_t in_length, uint8_t output[28]);
功能描述	在HMAC模式下使用SHA224计算摘要
先决条件	-
被调用函数	-
输入参数{in}	
key	指向用于HMAC模式的密钥
输入参数{in}	
keysize	用于HMAC模式的密钥长度
输入参数{in}	
input	指向输入缓存区
输入参数{in}	
in_length	输入缓存区长度
输出参数{out}	
output	摘要结果
返回值	
ErrStatus	SUCCESS或ERROR

例如：

```
/* calculate digest using SHA224 in HMAC mode */

ErrStatus status;

const uint8_t key[16] = {0x01};

const uint8_t input[16] = {0x01};

static uint8_t output[20];

status = hau_hmac_sha_224((uint8_t *)key, 0x10, (uint8_t *)input, 0x10, output);
```

函数 hau_hash_sha_256

函数hau_hash_sha_256描述见下表：

表 3-499. 函数 hau_hash_sha_256

函数名称	hau_hash_sha_256
函数原形	ErrStatus hau_hash_sha_256(uint8_t *input, uint32_t in_length, uint8_t output[32]);
功能描述	在HASH模式下使用SHA256计算摘要
先决条件	-
被调用函数	-
输入参数{in}	
input	指向输入缓存区
输入参数{in}	
in_length	输入缓存区长度
输出参数{out}	
output	摘要结果
返回值	
ErrStatus	SUCCESS或ERROR

例如:

```
/* calculate digest using SHA256 in HASH mode */
ErrStatus status;

const uint8_t input[16] ={0x01};

static uint8_t output[20];

status = hau_hash_sha_256((uint8_t *)input, 0x10, output);
```

函数 hau_hmac_sha_256

函数hau_hmac_sha_256描述见下表:

表 3-500. 函数 hau_hmac_sha_256

函数名称	hau_hmac_sha_256
函数原形	ErrStatus hau_hmac_sha_256(uint8_t *key, uint32_t keysize, uint8_t *input, uint32_t in_length, uint8_t output[32]);
功能描述	在HMAC模式下使用SHA256计算摘要
先决条件	-
被调用函数	-
输入参数{in}	
key	指向用于HMAC模式的密钥
输入参数{in}	
keysize	用于HMAC模式的密钥长度
输入参数{in}	
input	指向输入缓存区
输入参数{in}	
in_length	输入缓存区长度

输出参数{out}	
output	摘要结果
返回值	
ErrStatus	SUCCESS或ERROR

例如:

```
/* calculate digest using SHA256 in HMAC mode */

ErrStatus status;

const uint8_t key[16] ={0x01};

const uint8_t input[16] ={0x01};

static uint8_t output[20];

status = hau_hmac_sha_256((uint8_t *)key, 0x10, (uint8_t *)input, 0x10, output);
```

函数 hau_hash_md5

函数hau_hash_md5描述见下表:

表 3-501. 函数 hau_hash_md5

函数名称	hau_hash_md5
函数原形	ErrStatus hau_hash_md5(uint8_t *input, uint32_t in_length, uint8_t output[16]);
功能描述	在HASH模式下使用MD5计算摘要
先决条件	-
被调用函数	-
输入参数{in}	
input	指向输入缓存区
输入参数{in}	
in_length	输入缓存区长度
输出参数{out}	
output	摘要结果
返回值	
ErrStatus	SUCCESS或ERROR

例如:

```
/* calculate digest using MD5 in HASH mode */

ErrStatus status;

const uint8_t input[16] ={0x01};

static uint8_t output[20];

status = hau_hash_md5((uint8_t *)input, 0x10, output);
```

函数 hau_hmac_md5

函数hau_hmac_md5描述见下表：

表 3-502. 函数 hau_hmac_md5

函数名称	hau_hmac_md5
函数原形	ErrStatus hau_hmac_md5(uint8_t *key, uint32_t keysize, uint8_t *input, uint32_t in_length, uint8_t output[16]);
功能描述	在HMAC模式下使用MD5计算摘要
先决条件	-
被调用函数	-
输入参数{in}	
key	指向用于HMAC模式的密钥
输入参数{in}	
keysize	用于HMAC模式的密钥长度
输入参数{in}	
input	指向输入缓存区
输入参数{in}	
in_length	输入缓存区长度
输出参数{out}	
output	摘要结果
返回值	
ErrStatus	SUCCESS或ERROR

例如：

```
/* calculate digest using MD5 in HMAC mode */
ErrStatus status;

const uint8_t key[16] ={0x01};

const uint8_t input[16] ={0x01};

static uint8_t output[20];

status = hau_hmac_md5((uint8_t *)key, 0x10, (uint8_t *)input, 0x10, output);
```

函数 hau_flag_get

函数hau_flag_get描述见下表：

表 3-503. 函数 hau_flag_get

函数名称	hau_flag_get
函数原形	FlagStatus hau_flag_get(uint32_t flag);
功能描述	获取HAU标志状态
先决条件	-
被调用函数	-

输入参数{in}	
flag	HAU标志状态
HAU_FLAG_DATA_INP UT	输入FIFO有足够空间
HAU_FLAG_CALCULA TION_COMPLETE	摘要计算完整
HAU_FLAG_DMA	DMA被使能或传输正在处理
HAU_FLAG_BUSY	数据块正在处理
HAU_FLAG_INFIFO_N O_EMPTY	输入FIFO非空
输出参数{out}	
-	-
返回值	
FlagStatus	SET或RESET

例如：

```
/* get the HAU flag status */
```

```
FlagStatus status;
```

```
status = hau_flag_get(HAU_FLAG_DMA);
```

函数 hau_flag_clear

函数hau_flag_clear描述见下表：

表 3-504. 函数 hau_flag_clear

函数名称	hau_flag_clear
函数原形	void hau_flag_clear(uint32_t flag);
功能描述	清除HAU标志状态
先决条件	-
被调用函数	-
输入参数{in}	
flag	HAU标志状态
HAU_FLAG_DATA_INP UT	输入FIFO有足够空间
HAU_FLAG_CALCULA TION_COMPLETE	摘要计算完整
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* clear the HAU flag status */
```

```
hau_flag_clear(HAU_FLAG_DATA_INPUT);
```

函数 `hau_interrupt_enable`

函数 `hau_interrupt_enable` 描述见下表：

表 3-505. 函数 `hau_interrupt_enable`

函数名称	<code>hau_interrupt_enable</code>
函数原形	<code>void hau_interrupt_enable(uint32_t interrupt);</code>
功能描述	使能HAU中断
先决条件	-
被调用函数	-
输入参数{in}	
flag	HAU标志状态
HAU_INT_DATA_INPUT	新数据块进入IN缓存区
HAU_INT_CALCULATION_COMPLETE	计算完成
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* enable hau interrupt */
```

```
hau_interrupt_enable(HAU_INT_DATA_INPUT);
```

函数 `hau_interrupt_disable`

函数 `hau_interrupt_disable` 描述见下表：

表 3-506. 函数 `hau_interrupt_disable`

函数名称	<code>hau_interrupt_disable</code>
函数原形	<code>void hau_interrupt_disable(uint32_t interrupt);</code>
功能描述	除能HAU中断
先决条件	-
被调用函数	-
输入参数{in}	
flag	HAU标志状态
HAU_INT_DATA_INPUT	新数据块进入IN缓存区
HAU_INT_CALCULATION_COMPLETE	计算完成
输出参数{out}	

-	-
返回值	
-	-

例如:

```
/* disable hau interrupt */
```

```
hau_interrupt_disable (HAU_INT_DATA_INPUT);
```

函数 `hau_interrupt_flag_get`

函数 `hau_interrupt_flag_get` 描述见下表:

表 3-507. 函数 `hau_interrupt_flag_get`

函数名称	<code>hau_interrupt_flag_get</code>
函数原形	<code>FlagStatus hau_interrupt_flag_get(uint32_t int_flag)</code>
功能描述	获取HAU中断标志状态
先决条件	-
被调用函数	-
输入参数{in}	
<code>int_flag</code>	HAU标志状态
<code>HAU_INT_FLAG_DATA_INPUT</code>	输入FIFO有足够空间
<code>HAU_INT_FLAG_CALCULATION_COMPLETE</code>	摘要计算完整
输出参数{out}	
-	-
返回值	
FlagStatus	SET或RESET

例如:

```
/* get the HAU interrupt flag status */
```

```
FlagStatus status;
```

```
status = hau_interrupt_flag_get(HAU_INT_FLAG_DATA_INPUT);
```

函数 `hau_interrupt_flag_clear`

函数 `hau_interrupt_flag_clear` 描述见下表:

表 3-508. 函数 `hau_interrupt_flag_clear`

函数名称	<code>hau_interrupt_flag_clear</code>
函数原形	<code>void hau_interrupt_flag_clear(uint32_t int_flag)</code>
功能描述	清除HAU中断标志状态
先决条件	-

被调用函数	-
输入参数{in}	
int_flag	HAU标志状态
HAU_INT_FLAG_DATA_INPUT	输入FIFO有足够空间
HAU_INT_FLAG_CALCULATION_COMPLETE	摘要计算完整
输出参数{out}	
-	-
返回值	
-	-

```
/* clear the HAU interrupt flag status */
```

```
hau_interrupt_flag_clear (HAU_INT_FLAG_DATA_INPUT);
```

3.18. 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[3.18.1](#)描述了I2C的寄存器列表，章节[3.18.2](#)对I2C库函数进行说明。

3.18.1. 外设寄存器说明

I2C寄存器列表如下表所示：

表 3-509. I2C 寄存器

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

3.18.2. 外设库函数说明

I2C库函数列表如下表所示：

表 3-510. I2C 库函数

库函数名称	库函数描述
i2c_deinit	复位外设I2C

库函数名称	库函数描述
i2c_clock_config	配置I2C时钟
i2c_mode_addr_config	配置I2C地址
i2c_smbus_type_config	SMBus类型选择
i2c_ack_config	是否发送ACK
i2c_ackpos_config	ACK位置配置
i2c_master_addressing	主机发送从机地址
i2c_dualaddr_enable	双地址模式使能
i2c_enable	使能I2C模块
i2c_disable	关闭I2C模块
i2c_start_on_bus	在I2C总线上生成起始位
i2c_stop_on_bus	在I2C总线上生成停止位
i2c_data_transmit	发送数据
i2c_data_receive	接收数据
i2c_dma_enable	I2C DMA模式使能
i2c_dma_last_transfer_config	使能下一个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_enable	报文错误校验使能
i2c_pec_transfer_enable	传输PEC值使能
i2c_pec_value_get	获取报文错误校验值
i2c_smbus_issue_alert	通过SMBA引脚发送警告
i2c_smbus_arp_enable	SMBus下ARP协议是否开启
i2c_flag_get	标志位获取
i2c_flag_clear	清除标志位
i2c_interrupt_enable	中断使能
i2c_interrupt_disable	中断除能
i2c_interrupt_flag_get	中断标志位获取
i2c_interrupt_flag_clear	中断标志位清除

函数 i2c_deinit

函数i2c_deinit描述见下表：

表 3-511. 函数 i2c_deinit

函数名称	i2c_deinit
函数原型	void i2c_deinit(uint32_t i2c_periph);
功能描述	复位外设I2C
先决条件	-
被调用函数	rcu_periph_reset_enable / rcu_periph_reset_disable
输入参数{in}	
i2c_periph	I2C外设

<i>I2Cx</i>	(x=0,1,2)
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 3-512. 函数 i2c_clock_config

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

例如：

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

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

函数 i2c_mode_addr_config

函数i2c_mode_addr_config描述见下表：

表 3-513. 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=0,1,2)
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 3-514. 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

<i>I2Cx</i>	(x=0,1,2)
输入参数{in}	
type	主机或从机
<i>I2C_SMBUS_DEVICE</i>	从机
<i>I2C_SMBUS_HOST</i>	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 3-515. 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
<i>I2Cx</i>	(x=0,1,2)
输入参数{in}	
ack	是否发送ACK
<i>I2C_ACK_ENABLE</i>	ACK 会被发送
<i>I2C_ACK_DISABLE</i>	ACK 不会发送
输出参数{out}	
-	-
返回值	
-	-

例如：

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

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


函数 i2c_ackpos_config

函数i2c_ackpos_config描述见下表：

表 3-516. 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=0,1,2)
输入参数{in}	
pos	ACK位置
I2C_ACKPOS_CURRENT	当前正在接收的字节是否发送ACK
I2C_ACKPOS_NEXT	下一个接收的字节是否发送ACK
输出参数{out}	
-	-
返回值	
-	-

例如：

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

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

函数 i2c_master_addressing

函数i2c_master_addressing描述见下表：

表 3-517. 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=0,1,2)
输入参数{in}	
addr	从机地址

输入参数{in}	
trandirection	发送或接收
<i>I2C_TRANSMITTER</i>	发送
<i>I2C_RECEIVER</i>	接收
输出参数{out}	
-	-
返回值	
-	-

例如：

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

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

函数 i2c_dualaddr_enable

函数i2c_dualaddr_enable描述见下表：

表 3-518. 函数 i2c_dualaddr_enable

函数名称	i2c_dualaddr_enable
函数原型	void i2c_dualaddr_enable(uint32_t i2c_periph, uint32_t addr);
功能描述	双地址模式使能
先决条件	-
被调用函数	-
输入参数{in}	
i2c_periph	I2C外设
<i>I2Cx</i>	(x=0,1,2)
输入参数{in}	
addr	双地址模式下第二个地址
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* configure i2c second slave address */
```

```
i2c_dualaddr_enable(I2C0, 0x82);
```

函数 i2c_dualaddr_disable

函数i2c_dualaddr_disable描述见下表：

表 3-519. 函数 i2c_dualaddr_disable

函数名称	i2c_dualaddr_disable
------	----------------------

函数原型	void i2c_dualaddr_disable(uint32_t i2c_periph);
功能描述	双地址模式除能
先决条件	-
被调用函数	-
输入参数{in}	
i2c_periph	I2C外设
I2Cx	(x=0,1,2)
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* disable I2C0 dual-address */
```

```
i2c_dualaddr_disable(I2C0);
```

函数 i2c_enable

函数i2c_enable描述见下表：

表 3-520. 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=0,1,2)
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 3-521. 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	关闭I2C模块
先决条件	-
被调用函数	-
输入参数{in}	
i2c_periph	I2C外设
I2Cx	(x=0,1,2)
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 3-522. 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=0,1,2)
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 3-523. 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=0,1,2)
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 3-524. 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=0,1,2)
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 3-525. 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=0,1,2)
输出参数{out}	
-	-
返回值	
uint8_t	0x00..0xFF

例如：

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

```
uint8_t i2c_receiver;
```

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

函数 i2c_dma_enable

函数i2c_dma_enable描述见下表：

表 3-526. 函数 i2c_dma_enable

函数名称	i2c_dma_enable
函数原型	void i2c_dma_enable(uint32_t i2c_periph, uint32_t dmastate);
功能描述	I2C DMA模式使能
先决条件	-
被调用函数	-
输入参数{in}	
i2c_periph	I2C外设
I2Cx	(x=0,1,2)
输入参数{in}	
dmastate	开启或关闭
I2C_DMA_ON	DMA模式开启
I2C_DMA_OFF	DMA模式关闭
输出参数{out}	
-	-
返回值	
-	-

例如：

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

```
i2c_dma_enable(I2C0, I2C_DMA_ON);
```

函数 i2c_dma_last_transfer_config

函数i2c_dma_last_transfer_config描述见下表:

表 3-527. 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	使能下一个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=0,1,2)
输入参数{in}	
dmalast	是否使能下一个DMA EOT是最后传输
I2C_DMALST_ON	使能
I2C_DMALST_OFF	不使能
输出参数{out}	
-	-
返回值	
-	-

例如:

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

```
i2c_dma_last_transfer_enable(I2C0, I2C_DMALST_ON);
```

函数 i2c_stretch_scl_low_config

函数i2c_stretch_scl_low_config描述见下表:

表 3-528. 函数 i2c_stretch_scl_low_config

函数名称	i2c_stretch_scl_low_config
函数原型	void i2c_stretch_scl_low_config(uint32_t i2c_periph, uint32_t stretchpara);
功能描述	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
I2Cx	(x=0,1,2)
输入参数{in}	
stretchpara	是否拉低SCL
I2C_SCLSTRETCH	拉低SCL

<code>_ENABLE</code>	
<code>I2C_SCLSTRETCH_DISABLE</code>	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 3-529. 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=0,1,2)
输入参数{in}	
gcallpara	是否响应广播呼叫
I2C_GCEN_ENABLE	从机响应广播呼叫
I2C_GCEN_DISABLE	从机不响应广播呼叫
输出参数{out}	
-	-
返回值	
-	-

例如：

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

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


函数 i2c_software_reset_config

函数i2c_software_reset_config描述见下表:

表 3-530. 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=0,1,2)
输入参数{in}	
sreset	是否复位
I2C_SRESET_SET	复位
I2C_SRESET_RESET	没有复位
输出参数{out}	
-	-
返回值	
-	-

例如:

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

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

函数 i2c_pec_enable

函数i2c_pec_enable描述见下表:

表 3-531. 函数 i2c_pec_enable

函数名称	i2c_pec_enable
函数原型	void i2c_pec_enable(uint32_t i2c_periph, uint32_t pecstate);
功能描述	报文错误校验使能
先决条件	-
被调用函数	-
输入参数{in}	
i2c_periph	I2C外设
I2Cx	(x=0,1,2)
输入参数{in}	
pecpara	开启或关闭
I2C_PEC_ENABLE	报文错误校验使能
I2C_PEC_DISABLE	报文错误校验关闭

输出参数{out}	
-	-
返回值	
-	-

例如：

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

函数 i2c_pec_transfer_enable

函数i2c_pec_transfer_enable描述见下表：

表 3-532. 函数 i2c_pec_transfer_enable

函数名称	i2c_pec_transfer_enable
函数原型	void i2c_pec_transfer_enable(uint32_t i2c_periph, uint32_t pecpara);
功能描述	传输PEC值使能
先决条件	-
被调用函数	-
输入参数{in}	
i2c_periph	I2C外设
I2Cx	(x=0,1,2)
输入参数{in}	
pecpara	是否传输PEC
I2C_PECTRANS_ENABLE	传输PEC
I2C_PECTRANS_DISABLE	不传输PEC
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* I2C0 transfer PEC */
i2c_pec_transfer_enable(I2C0, I2C_PECTRANS_ENABLE);
```

函数 i2c_pec_value_get

函数i2c_pec_value_get描述见下表：

表 3-533. 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=0,1,2)
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_issue_alert

函数i2c_smbus_issue_alert描述见下表：

表 3-534. 函数 i2c_smbus_issue_alert

函数名称	i2c_smbus_issue_alert
函数原型	void i2c_smbus_issue_alert(uint32_t i2c_periph, uint32_t smbuspara);
功能描述	通过SMBA引脚发送警告
先决条件	-
被调用函数	-
输入参数{in}	
i2c_periph	I2C外设
I2Cx	(x=0,1,2)
输入参数{in}	
smbuspara	是否通过SMBA引脚发送警告
I2C_SALTSEND_ENABLE	通过SMBA引脚发送警告
I2C_SALTSEND_DISABLE	不通过SMBA引脚发送警告
输出参数{out}	
-	-
返回值	
-	-

例如：

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

```
i2c_smbus_issue_alert(I2C0, I2C_SALTSEND_ENABLE);
```

函数 i2c_smbus_arp_enable

函数i2c_smbus_arp_enable描述见下表:

表 3-535. 函数 i2c_smbus_arp_enable

函数名称	i2c_smbus_arp_enable
函数原型	void i2c_smbus_arp_enable(uint32_t i2c_periph, uint32_t arpstate);
功能描述	SMBus下ARP协议是否开启
先决条件	-
被调用函数	-
输入参数{in}	
i2c_periph	I2C外设
I2Cx	(x=0,1,2)
输入参数{in}	
arpstate	SMBus下ARP协议是否开启
I2C_ARP_ENABLE	使能ARP
I2C_ARP_DISABLE	关闭ARP
输出参数{out}	
-	-
返回值	
-	-

例如:

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

```
i2c_smbus_arp_enable(I2C0, I2C_ARP_ENABLE);
```

函数 i2c_flag_get

函数i2c_flag_get描述见下表:

表 3-536. 函数 i2c_flag_get

函数名称	i2c_flag_get
函数原型	FlagStatus i2c_flag_get(uint32_t i2c_periph, uint32_t flag);
功能描述	标志位获取
先决条件	-
被调用函数	-
输入参数{in}	
i2c_periph	I2C外设
I2Cx	(x=0,1,2)
输入参数{in}	
flag	需要获取的标志位
I2C_FLAG_SBSEN	起始位是否发送

<i>D</i>	
<i>I2C_FLAG_ADDSEND</i>	主机模式下地址是否发送/从机模式下地址是否匹配
<i>I2C_FLAG_BTC</i>	字节传输完成
<i>I2C_FLAG_ADD10SEND</i>	主机模式下10位地址地址头发送完成
<i>I2C_FLAG_STPDET</i>	从机模式下监测到STOP结束位
<i>I2C_FLAG_RBNE</i>	接收期间I2C_DATA非空
<i>I2C_FLAG_TBE</i>	发送期间I2C_DATA为空
<i>I2C_FLAG_BERR</i>	总线错误，表示I2C总线上发生了预料之外的START起始位或STOP结束位
<i>I2C_FLAG_LOSTARB</i>	主机模式下仲裁丢失
<i>I2C_FLAG_AERR</i>	应答错误
<i>I2C_FLAG_OUERR</i>	当禁用SCL拉低功能后，在从机模式下发生了过载或欠载事件
<i>I2C_FLAG_PECERR</i>	接收数据时PEC错误
<i>I2C_FLAG_SMBTO</i>	SMBus模式下超时信号
<i>I2C_FLAG_SMBALT</i>	SMBus警报状态
<i>I2C_FLAG_MASTERR</i>	表明I2C时钟在主机模式还是从机模式的标志位
<i>I2C_FLAG_I2CBSY</i>	忙标志
<i>I2C_FLAG_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_DEFSMB</i>	从机模式下SMBus主机地址头
<i>I2C_FLAG_HSTSMB</i>	从机模式下监测到SMBus主机地址头
<i>I2C_FLAG_DUMOD</i>	从机模式下双标志位表明哪个地址和双地址模式匹配
输出参数{out}	
-	-
返回值	
FlagStatus	SET / RESET

例如：

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

```
FlagStatus flag_state = RESET;
```

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

函数 i2c_flag_clear

函数i2c_flag_clear描述见下表：

表 3-537. 函数 i2c_flag_clear

函数名称	i2c_flag_clear
函数原型	void i2c_flag_clear(uint32_t i2c_periph, 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}	
i2c_periph	I2C外设
I2Cx	(x=0,1,2)
输入参数{in}	
flag	标志位类型
I2C_FLAG_SMBAL T	SMBus警报状态
I2C_FLAG_SMBTO	SMBus模式下超时信号
I2C_FLAG_PECER R	接收数据时PEC错误
I2C_FLAG_OUERR	当禁用SCL拉低功能后，在从机模式下发生了过载或欠载事件
I2C_FLAG_AERR	应答错误
I2C_FLAG_LOSTA RB	主机模式下仲裁丢失
I2C_FLAG_BERR	总线错误
I2C_FLAG_ADDSE ND	主机模式下地址是否发送/从机模式下地址是否匹配，通过读I2C_STAT0和I2C_STAT1来清除
输出参数{out}	
-	-
返回值	
-	-

例如：

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

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

函数 i2c_interrupt_enable

函数i2c_interrupt_enable描述见下表：

表 3-538. 函数 i2c_interrupt_enable

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

先决条件	-
被调用函数	-
输入参数{in}	
i2c_periph	I2C外设
I2Cx	(x=0,1,2)
输入参数{in}	
interrupt	中断类型
I2C_INT_ERR	错误中断使能
I2C_INT_EV	事件中断使能
I2C_INT_BUF	缓冲区中断使能
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* enable I2C0 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 3-539. 函数 i2c_interrupt_disable

函数名称	i2c_interrupt_disable
函数原型	void i2c_interrupt_disable(uint32_t i2c_periph, i2c_interrupt_enum interrupt);
功能描述	中断除能
先决条件	-
被调用函数	-
输入参数{in}	
i2c_periph	I2C外设
I2Cx	(x=0,1,2)
输入参数{in}	
interrupt	中断类型
I2C_INT_ERR	错误中断使能
I2C_INT_EV	事件中断使能
I2C_INT_BUF	缓冲区中断使能
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* disable I2C0 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 3-540. 函数 i2c_interrupt_flag_get

函数名称	i2c_interrupt_flag_get
函数原型	FlagStatus i2c_interrupt_flag_get(uint32_t i2c_periph, i2c_interrupt_flag_enum int_flag);
功能描述	中断标志位获取
先决条件	-
被调用函数	-
输入参数{in}	
i2c_periph	I2C外设
I2Cx	(x=0,1,2)
输入参数{in}	
int_flag	中断标志
I2C_INT_FLAG_SB SEND	主机模式下发送START起始位
I2C_INT_FLAG_AD DSEND	主机模式下成功发送了地址 / 从机模式下接收到了地址并且和自身的地址匹配
I2C_INT_FLAG_BT C	字节发送结束
I2C_INT_FLAG_AD D10SEND	主机模式下10位地址地址头被发送
I2C_INT_FLAG_ST PDET	从机模式下监测到STOP结束位
I2C_INT_FLAG_RB NE	接收期间I2C_DATA非空
I2C_INT_FLAG_TB E	发送期间I2C_DATA为空
I2C_INT_FLAG_BE RR	总线错误
I2C_INT_FLAG_LO STARB	主机模式下仲裁丢失
I2C_INT_FLAG_AE RR	应答错误
I2C_INT_FLAG_OU ERR	当禁用SCL拉低功能后，在从机模式下发生了过载或欠载事件
I2C_INT_FLAG_PE CERR	接收数据时PEC错误

<i>I2C_INT_FLAG_SM BTO</i>	SMBus模式下超时信号
<i>I2C_INT_FLAG_SM BALT</i>	SMBus警报状态
输出参数{out}	
-	-
返回值	
FlagStatus	SET / RESET

例如：

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

```
FlagStatus flag_state = RESET;
```

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

函数 i2c_interrupt_flag_clear

函数i2c_interrupt_flag_clear描述见下表：

表 3-541. 函数 i2c_interrupt_flag_clear

函数名称	i2c_interrupt_flag_clear
函数原型	void i2c_interrupt_flag_clear(uint32_t i2c_periph, i2c_interrupt_flag_enum int_flag);
功能描述	中断标志位清除
先决条件	-
被调用函数	-
输入参数{in}	
i2c_periph	I2C外设
I2Cx	(x=0,1,2)
输入参数{in}	
int_flag	中断标志
<i>I2C_INT_FLAG_AD DSEND</i>	主机模式下成功发送了地址 / 从机模式下接收到了地址并且和自身的地址匹配
<i>I2C_INT_FLAG_BE RR</i>	总线错误
<i>I2C_INT_FLAG_LO STARB</i>	主机模式下仲裁丢失
<i>I2C_INT_FLAG_AE RR</i>	应答错误
<i>I2C_INT_FLAG_OU ERR</i>	当禁用SCL 拉低功能后，在从机模式下发生了过载或欠载事件
<i>I2C_INT_FLAG_PE CERR</i>	接收数据时PEC错误
<i>I2C_INT_FLAG_SM</i>	SMBus模式下超时信号

<i>BTO</i>	
<i>I2C_INT_FLAG_SM BALT</i>	SMBus警报状态
输出参数{out}	
-	-
返回值	
-	-

例如：

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

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

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

3.19.1. 外设寄存器说明

表 3-542. 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

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

1. 参考 core_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 3-543. 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

3.19.2. 外设库函数说明

枚举类型 IRQn_Type

表 3-544. 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_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
CAN0_TX_IRQn	CAN0 发送中断
CAN0_RX0_IRQn	CAN0 接收 0 中断
CAN0_RX1_IRQn	CAN0 接收 1 中断
CAN0_EWMC_IRQn	CAN0 EWMC 中断
EXTI5_9_IRQn	EXTI 线[9:5] 中断
TIMER0_BRK_TIMER8_IRQn	TIMER0 中止中断和 TIMER8 全局中断
TIMER0_UP_TIMER9_IRQn	TIMER0 更新中断 TIMER 全局中断
TIMER0_TRG_CMT_TIMER10_IRQn	TIMER0 触发与通道换相中断以及 TIMER10 全局中断
TIMER0_Channel_IRQn	TIMER0 通道捕获比较中断
TIMER1_IRQn	TIMER1 全局中断
TIMER2_IRQn	TIMER2 全局中断
TIMER3_IRQn	TIMER3 全局中断
I2C0_EV_IRQn	I2C0 事件中断
I2C0_ER_IRQn	I2C0 错误中断
I2C1_EV_IRQn	I2C1 事件中断
I2C1_ER_IRQn	I2C1 错误中断
SPI0_IRQn	SPI0 全局中断
SPI1_IRQn	SPI1 全局中断
USART0_IRQn	USART0 全局中断
USART1_IRQn	USART1 全局中断
USART2_IRQn	USART2 全局中断
EXTI10_15_IRQn	EXTI 线[15:10] 中断
RTC_Alarm_IRQn	连接 EXTI 线的 RTC 闹钟中断
USBFS_WKUP_IRQn	连接 EXTI 线的 USBFS 唤醒中断
TIMER7_BRK_TIMER11_IRQn	TIMER7 中止中断 TIMER11 全局中断
TIMER7_UP_TIMER12_IRQn	TIMER7 更新中断和 TIMER12 全局中断

TIMER7_TRG_CMT_TIMER13_IRQn	TIMER7 触发与通道换相中断以及 TIMER13 全局中断
TIMER7_Channel_I RQn	TIMER7 通道比较捕获中断
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_IR Qn	DMA1 通道 0 全局中断
DMA1_Channel1_IR Qn	DMA1 通道 1 全局中断
DMA1_Channel2_IR Qn	DMA1 通道 2 全局中断
DMA1_Channel3_IR Qn	DMA1 通道 3 全局中断
DMA1_Channel4_IR Qn	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_IRQ n	CAN1 EWMC 中断
USBFS_IRQn	USBFS 全局中断
DMA1_Channel5_IR Qn	DMA1 通道 5 全局中断
DMA1_Channel6_IR Qn	DMA1 通道 6 全局中断
USART5_IRQn	USART5 全局中断
I2C2_EV_IRQn	I2C2 事件中断
I2C2_ER_IRQn	I2C2 错误中断
DCI_IRQn	DCI 全局中断
CAU_IRQn	CAU 全局中断
HAU_TRNG_IRQn	HAU 和 TRNG 全局中断
UART6_IRQn	UART6 全局中断
UART7_IRQn	UART 全局中断

TLI_IRQn	TLI 全局中断
TLI_ER_IRQn	TLI 全局错误中断

MISC库函数列表如下表所示:

表 3-545. 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	设置向量表地址
system_lowpower_set	设置系统低功耗模式状态
system_lowpower_reset	复位系统低功耗模式状态
systick_clksource_set	设置Systick定时器时钟源

函数 nvic_priority_group_set

函数nvic_priority_group_set描述见下表:

表 3-546. 函数 nvic_priority_group_set

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

例如:

```
/* priority group configuration , 0 bits for pre-emption priority 4 bits for subpriority */
```

```
nvic_priority_group_set(NVIC_PRIGROUP_PRE0_SUB4);
```

函数 nvic_irq_enable

函数nvic_irq_enable描述见下表:

表 3-547. 函数 nvic_irq_enable

函数名称	nvic_irq_enable
函数原形	void nvic_irq_enable(uint8_t nvic_irq, uint8_t nvic_irq_pre_priority, uint8_t nvic_irq_sub_priority);
功能描述	使能NVIC的中断
先决条件	-
被调用函数	nvic_priority_group_set
输入参数{in}	
nvic_irq	NVIC中断, 参考枚举类型 表3-544. 枚举类型IRQn_Type
输入参数{in}	
nvic_irq_pre_priority	抢占优先级
输入参数{in}	
nvic_irq_sub_priority	响应优先级
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* enable window watchDog timer interrupt , pre-emption priority is 1, subpriority is 1 */
```

```
nvic_irq_enable(WWDGT_IRQn,1,1);
```

函数 nvic_irq_disable

函数nvic_irq_disable描述见下表:

表 3-548. 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3-544. 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 3-549. 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 */
```

```
#define NVIC_VECTTAB_FLASH ((uint32_t)0x08000000)
```

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

函数 system_lowpower_set

函数system_lowpower_set描述见下表：

表 3-550. 函数 system_lowpower_set

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

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

例如：

```
/* the system always enter low power mode by exiting from ISR */
#define SCB_SCR_SLEEPONEXIT ((uint8_t)0x02)
#define SCB_LPM_SLEEP_EXIT_ISR SCB_SCR_SLEEPONEXIT
system_lowpower_set(SCB_LPM_SLEEP_EXIT_ISR);
```

函数 system_lowpower_reset

函数system_lowpower_reset描述见下表：

表 3-551. 函数 system_lowpower_reset

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

-	-
---	---

例如：

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

#define SCB_SCR_SLEEPONEXIT ((uint8_t)0x02)

#define SCB_LPM_SLEEP_EXIT_ISR SCB_SCR_SLEEPONEXIT

system_lowpower_reset(SCB_LPM_SLEEP_EXIT_ISR);
```

函数 `systick_clksource_set`

函数 `systick_clksource_set` 描述见下表：

表 3-552. 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 */

#define SYSTICK_CLKSOURCE_HCLK_DIV8 ((uint32_t)0xFFFFFFFBU)

systick_clksource_set(SYSTICK_CLKSOURCE_HCLK_DIV8);
```

3.20. 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 [3.20.1](#) 描述了 PMU 的寄存器列表，章节 [3.20.2](#) 对 PMU 库函数进行说明。

3.20.1. 外设寄存器说明

PMU 寄存器列表如下表所示：

表 3-553. 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

3.20.2. 外设库函数说明

PMU 库函数列表如下表所示：

表 3-554. PMU 库函数

库函数名称	库函数描述
pmu_deinit	复位PMU寄存器
pmu_lvd_select	选择低压检测阈值
pmu_lvd_disable	低压检测器禁能
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引脚唤醒禁能
pmu_backup_write_enable	备份域寄存器写访问使能
pmu_backup_write_disable	备份域寄存器写访问禁能
pmu_flag_get	获取PMU标志位
pmu_flag_clear	清除PMU标志位

函数 pmu_deinit

函数pmu_deinit描述见下表：

表 3-555. 函数 pmu_deinit

函数名称	pmu_deinit
函数原型	void pmu_deinit(void);
功能描述	复位PMU寄存器
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 3-556. 函数 pmu_lvd_select

函数名称	pmu_lvd_select
函数原型	void pmu_lvd_select(uint32_t lvdn);
功能描述	选择低压检测阈值
先决条件	-
被调用函数	-
输入参数{in}	
lvdn	电压阈值
PMU_LVDT_0	电压阈值为2.2V
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 3-557. 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	低压检测器禁能
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 PMU lvd */
pmu_lvd_disable ();
```

函数 pmu_to_sleepmode

函数pmu_to_sleepmode描述见下表：

表 3-558. 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 3-559. 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
<i>PMU_LDO_NORMAL</i>	当系统进入深度睡眠模式时，LDO正常工作
<i>PMU_LDO_LOWPOWER</i>	当系统进入深度睡眠模式时，LDO进入低功耗模式
输入参数{in}	
deepsleepmodecmd	进入深度睡眠模式命令
<i>WFI_CMD</i>	WFI命令
<i>WFE_CMD</i>	WFE命令
输出参数{out}	
-	-
返回值	
-	-

例如：

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

```
pmu_to_deepsleepmode (PMU_LDO_NORMAL, WFI_CMD);
```

函数 pmu_to_standbymode

函数pmu_to_standbymode描述见下表：

表 3-560. 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_standby ();
```

函数 pmu_wakeup_pin_enable

函数pmu_wakeup_pin_enable描述见下表：

表 3-561. 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 3-562. 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 wakeup pin */
pmu_wakeup_pin_disable ();
```

函数 pmu_backup_write_enable

函数pmu_backup_write_enable描述见下表：

表 3-563. 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	备份域寄存器写访问使能
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 write access to the registers in backup domain */
```

```
pmu_backup_write_enable ();
```

函数 pmu_backup_write_disable

函数pmu_backup_write_disable描述见下表：

表 3-564. 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	备份域寄存器写访问禁能
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 write access to the registers in backup domain */
```

```
pmu_backup_write_disable ();
```


函数 pmu_flag_get

函数pmu_flag_get描述见下表:

表 3-565. 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PMU标志位
先决条件	-
被调用函数	-
输入参数{in}	
flag	PMU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 PMU wakeup flag */
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 3-566. 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PMU标志位
先决条件	-
被调用函数	-
输入参数{in}	
flag	PMU标志位
PMU_FLAG_RESET_WAKEUP	清除唤醒标志
PMU_FLAG_RESET_STANDBY	清除待机标志

输出参数{out}	
-	
返回值	
-	

例如：

```
/* clear PMU wakeup flag */
```

```
pmu_flag_clear (PMU_FLAG_RESET_WAKEUP);
```

3.21. 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 [3.21.1](#) 描述了 RCU 的寄存器列表，章节 [3.21.2](#) 对 RCU 库函数进行说明。

3.21.1. 外设寄存器说明

RCU寄存器列表如下表所示：

表 3-567. RCU 寄存器

寄存器名称	寄存器描述
RCU_CTL	控制寄存器
RCU_CFG0	时钟配置寄存器 0
RCU_INT	时钟中断寄存器
RCU_APB2RST	APB2 复位寄存器
RCU_APB1RST	APB1 复位寄存器
RCU_AHB1EN	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_AHB1RST	AHB1 复位寄存器
RCU_CFG1	配置寄存器 1
RCU_DSV	深度睡眠模式电压寄存器
RCU_AHB2EN	AHB2 使能寄存器
RCU_ADDAPB2EN	APB2 附加使能寄存器
RCU_ADDAPB1EN	APB1 附加使能寄存器
RCU_AHB2RST	AHB2 复位寄存器
RCU_ADDAPB2RS T	APB2 附加复位寄存器
RCU_ADDAPB1RS T	APB1 附加复位寄存器

寄存器名称	寄存器描述
RCU_CFG2	配置寄存器 2
RCU_PLLTCTL	PLLT 控制寄存器
RCU_PLLTINT	PLLT 中断寄存器
RCU_PLLTCFG	PLLT 配置寄存器

3.21.2. 外设库函数说明

RCU库函数列表如下表所示：

表 3-568. 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_ckout1_config	配置 CKOUT1 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
rcu_usbfs_trng_clock_config	配置 USBFS/TRNG 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_pllt_config	配置 PLLT 的时钟源选择
rcu_pllt_vco_config	配置 PLLT 时钟的倍频和分频因子
rcu_tli_clock_config	配置 TLI 的时钟分频系数
rcu_lxtal_drive_capability_config	配置 LXTAL 驱动能力
rcu_osci_stab_wait	等待振荡器稳定标志位置位或振荡器起振超时

库函数名称	库函数描述
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
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_enable	使能时钟稳定中断
rcu_interrupt_disable	除能时钟稳定中断
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

枚举类型 rcu_periph_enum

表 3-569. 枚举类型 rcu_periph_enum

成员名称	功能描述
RCU_GPIOx	GPIOx时钟(x=A,B,C,D,E,F,G,H,I)
RCU_AF	复用功能时钟
RCU_CRC	CRC时钟
RCU_DMA	DMA时钟(x=0,1)
RCU_ENET	以太网ENET时钟
RCU_ENETTX	ENETTX时钟
RCU_ENETRX	ENETRX时钟
RCU_USBFS	USBFS时钟
RCU_EXMC	EXMC时钟
RCU_TIMERx	TIMERx时钟(x=0,1,2,3,4,5,6,7,8,9,10,11,12,13)
RCU_WWDGT	WWDGT时钟
RCU_SPIx	SPIx时钟(x=0,1,2)
RCU_USARTx	USARTx时钟(x=0,1,2,5)
RCU_UARTx	UARTx时钟(x=3,4,6,7)
RCU_I2Cx	I2Cx时钟(x=0,1,2)
RCU_CANx	CANx时钟(x=0,1)
RCU_PMU	PMU时钟
RCU_DAC	DAC时钟
RCU_RTC	RTC时钟
RCU_ADCx	ADCx时钟(x=0,1,2)
RCU_SDIO	SDIO时钟
RCU_BKPI	BKP接口时钟

成员名称	功能描述
RCU_TLI	TLI时钟
RCU_DCI	DCI时钟
RCU_CAU	CAU时钟
RCU_HAU	HAU时钟
RCU_TRNG	TRNG时钟

枚举类型 rcu_periph_sleep_enum

表 3-570. 枚举类型 rcu_periph_sleep_enum

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

枚举类型 rcu_periph_reset_enum

表 3-571. 枚举类型 rcu_periph_reset_enum

成员名称	功能描述
RCU_GPIOxRST	复位GPIOx时钟(x=A,B,C,D,E,F,G)
RCU_AFRST	复位功能时钟
RCU_ENETRST	复用以太网时钟
RCU_USBFSRST	复位USBFS时钟
RCU_TIMERxRST	复位TIMERx时钟(x=0,1,2,3,4,5,6,7,8,9,10,11,12,13)
RCU_WWDGTRST	复位WWDGT时钟
RCU_SPIxRST	复位SPIx时钟(x=0,1,2)
RCU_USARTxRST	复位USARTx时钟(x=0,1,2,5)
RCU_UARTxRST	复位UARTx时钟(x=3,4,6,7)
RCU_I2CxRST	复位I2Cx时钟(x=0,1,2)
RCU_CANxRST	复位CANx时钟(x=0,1)
RCU_PMURST	复位PMU时钟
RCU_DACRST	复位DAC时钟
RCU_ADCxRST	复位ADCx时钟(x=0,1,2)
RCU_BKPIRST	复位BKP接口时钟
RCU_TLIRST	复位TLI时钟
RCU_DCIRST	复位DCI时钟
RCU_CAURST	复位CAU时钟
RCU_HAURST	复位HAU时钟
RCU_TRNGRST	复位TRNG时钟

枚举类型 rcu_flag_enum

表 3-572. 枚举类型 rcu_flag_enum

成员名称	功能描述
RCU_FLAG_IRC8M	IRC8M稳定标志

成员名称	功能描述
STB	
RCU_FLAG_HXTALSTB	HXTAL稳定标志
RCU_FLAG_PLLSTB	PLL稳定标志
RCU_FLAG_PLL1STB	PLL1稳定标志
RCU_FLAG_PLL2STB	PLL2稳定标志
RCU_FLAG_PLLTSTB	PLLT稳定标志
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	独立看门狗定时器复位标志
RCU_FLAG_WWDGTRST	窗口看门狗定时器复位标志
RCU_FLAG_LPRST	low-power复位标志

枚举类型 rcu_int_flag_enum

表 3-573. 枚举类型 rcu_int_flag_enum

成员名称	功能描述
RCU_INT_FLAG_IRC40KSTB	IRC40K稳定中断标志
RCU_INT_FLAG_LXTALSTB	LXTAL稳定中断标志
RCU_INT_FLAG_IRC8MSTB	IRC8M稳定中断标志
RCU_INT_FLAG_HXTALSTB	HXTAL稳定中断标志
RCU_INT_FLAG_PLLSTB	PLL稳定中断标志
RCU_INT_FLAG_PLL1STB	PLL1稳定中断标志

成员名称	功能描述
LL1STB	
RCU_INT_FLAG_P LL2STB	PLL2稳定中断标志
RCU_INT_FLAG_C KM	HXTAL时钟阻塞中断标志
RCU_INT_FLAG_P LLTSTB	PLLT稳定中断标志

枚举类型 `rcu_int_flag_clear_enum`

表 3-574. 枚举类型 `rcu_int_flag_clear_enum`

成员名称	功能描述
RCU_INT_FLAG_IR C40KSTB_CLR	IRC40K稳定中断标志清除
RCU_INT_FLAG_L XTALSTB_CLR	LXTAL稳定中断标志清除
RCU_INT_FLAG_IR C8MSTB_CLR	IRC8M稳定中断标志清除
RCU_INT_FLAG_H XTALSTB_CLR	HXTAL稳定中断标志清除
RCU_INT_FLAG_P LLSTB_CLR	PLL稳定中断标志清除
RCU_INT_FLAG_P LL1STB_CLR	PLL1稳定中断标志清除
RCU_INT_FLAG_P LL2STB_CLR	PLL2稳定中断标志清除
RCU_INT_FLAG_C KM_CLR	时钟阻塞中断标志清除
RCU_INT_FLAG_P LLTSTB_CLR	PLLT稳定中断标志清除

枚举类型 `rcu_int_enum`

表 3-575. 枚举类型 `rcu_int_enum`

成员名称	功能描述
RCU_INT_IRC40KS TB	IRC40K稳定中断
RCU_INT_LXTALS TB	LXTAL稳定中断
RCU_INT_IRC8MS TB	IRC8M稳定中断使能
RCU_INT_HXTALS TB	HXTAL稳定中断

成员名称	功能描述
RCU_INT_PLLSTB	PLL稳定中断
RCU_INT_PLL1STB	PLL1稳定中断
RCU_INT_PLL2STB	PLL2稳定中断
RCU_INT_PLLTSTB	PLLT稳定中断

枚举类型 `rcu_osci_type_enum`

表 3-576. 枚举类型 `rcu_osci_type_enum`

成员名称	功能描述
RCU_IRC8M	内部8M RC振荡器
RCU_HXTAL	高速晶体振荡器
RCU_PLL_CK	锁相环
RCU_PLL1_CK	锁相环1
RCU_PLL2_CK	锁相环2
RCU_LXTAL	低速晶体振荡器
RCU_IRC40K	内部40K RC振荡器
RCU_PLLT_CK	TLI锁相环

枚举类型 `rcu_clock_freq_enum`

表 3-577. 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 3-578. 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
先决条件	-
被调用函数	-
输入参数{in}	
-	-
输出参数{out}	
-	-
返回值	

-	-
---	---

例如:

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

```
rcu_deinit();
```

函数 rcu_periph_clock_enable

函数rcu_periph_clock_enable描述见下表:

表 3-579. 函数 rcu_periph_clock_enable

函数名称	rcu_periph_clock_enable
函数原形	void rcu_periph_clock_enable(rcu_periph_enum periph);
功能描述	使能外设时钟
先决条件	-
被调用函数	-
输入参数{in}	
periph	RCU 外设，具体参考 表 3-569. 枚举类型 rcu_periph_enum
RCU_GPIOx	GPIOx 时钟(x=A,B,C,D,E,F,G ,H,I)
RCU_AF	复用功能时钟
RCU_CRC	CRC 时钟
RCU_DMA	DMA 时钟(x=0,1)
RCU_ENET	以太网 ENET 时钟
RCU_ENETTX	ENETTX 时钟
RCU_ENETRX	ENETRX 时钟
RCU_USBFS	USBFS 时钟
RCU_EXMC	EXMC 时钟
RCU_TIMERx	TIMERx 时钟(x=0,1,2,3,4,5,6,7,8,9,10,11,12,13)
RCU_WWDGT	WWDGT 时钟
RCU_SPIx	SPIx 时钟(x=0,1,2)
RCU_USARTx	USARTx 时钟(x=0,1,2,5)
RCU_UARTx	UARTx 时钟(x=3,4 ,6,7)
RCU_I2Cx	I2Cx 时钟(x=0,1,2)
RCU_CANx	CANx 时钟(x=0,1)
RCU_PMU	PMU 时钟
RCU_DAC	DAC 时钟
RCU_RTC	RTC 时钟
RCU_ADCx	ADCx 时钟(x=0,1,2)
RCU_SDIO	SDIO 时钟
RCU_BKPI	BKP 接口时钟
RCU_TLI	TLI 时钟
RCU_DCI	DCI 时钟
RCU_CAU	CAU 时钟

<i>RCU_HAU</i>	HAU 时钟
<i>RCU_TRNG</i>	TRNG 时钟
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* enable the USART0 clock */
```

```
rcu_periph_clock_enable(RCU_USART0);
```

函数 `rcu_periph_clock_disable`

函数 `rcu_periph_clock_disable` 描述见下表:

表 3-580. 函数 `rcu_periph_clock_disable`

函数名称	<code>rcu_periph_clock_disable</code>
函数原形	<code>void rcu_periph_clock_disable(rcu_periph_enum periph);</code>
功能描述	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 3-569. 枚举类型 <code>rcu_periph_enum</code>
<i>RCU_GPIOx</i>	GPIOx 时钟(x=A,B,C,D,E,F,G ,H,I)
<i>RCU_AF</i>	复用功能时钟
<i>RCU_CRC</i>	CRC 时钟
<i>RCU_DMA</i>	DMA 时钟(x=0,1)
<i>RCU_ENET</i>	以太网 ENET 时钟
<i>RCU_ENETTX</i>	ENETTX 时钟
<i>RCU_ENETRX</i>	ENETRX 时钟
<i>RCU_USBFS</i>	USBFS 时钟
<i>RCU_EXMC</i>	EXMC 时钟
<i>RCU_TIMERx</i>	TIMERx 时钟(x=0,1,2,3,4,5,6,7,8,9,10,11,12,13)
<i>RCU_WWDGT</i>	WWDGT 时钟
<i>RCU_SPIx</i>	SPIx 时钟(x=0,1,2)
<i>RCU_USARTx</i>	USARTx 时钟(x=0,1,2,5)
<i>RCU_UARTx</i>	UARTx 时钟(x=3,4 ,6,7)
<i>RCU_I2Cx</i>	I2Cx 时钟(x=0,1,2)
<i>RCU_CANx</i>	CANx 时钟(x=0,1)
<i>RCU_PMU</i>	PMU 时钟
<i>RCU_DAC</i>	DAC 时钟
<i>RCU_RTC</i>	RTC 时钟
<i>RCU_ADCx</i>	ADCx 时钟(x=0,1,2)

<i>RCU_SDIO</i>	SDIO 时钟
<i>RCU_BKPI</i>	BKP 接口时钟
<i>RCU_TLI</i>	TLI 时钟
<i>RCU_DCI</i>	DCI 时钟
<i>RCU_CAU</i>	CAU 时钟
<i>RCU_HAU</i>	HAU 时钟
<i>RCU_TRNG</i>	TRNG 时钟
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* disable the USART0 clock */
```

```
rcu_periph_clock_disable(RCU_USART0);
```

函数 `rcu_periph_clock_sleep_enable`

函数 `rcu_periph_clock_sleep_enable` 描述见下表:

表 3-581. 函数 `rcu_periph_clock_sleep_enable`

函数名称	<code>rcu_periph_clock_sleep_enable</code>
函数原形	<code>void rcu_periph_clock_sleep_enable(rcu_periph_sleep_enum periph);</code>
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 3-570. 枚举类型 <code>rcu_periph_sleep_enum</code>
<i>RCU_FMC_SLP</i>	FMC 时钟
<i>RCU_SRAM_SLP</i>	SRAM 时钟
输出参数{out}	
-	-
返回值	
-	-

例如:

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

```
rcu_periph_clock_sleep_enable(RCU_FMC_SLP);
```

函数 `rcu_periph_clock_sleep_disable`

函数 `rcu_periph_clock_sleep_disable` 描述见下表:

表 3-582. 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 3-570. 枚举类型 rcu_periph_sleep_enum
RCU_FMC_SLP	FMC 时钟
RCU_SRAM_SLP	SRAM 时钟
输出参数{out}	
-	-
返回值	
-	-

例如：

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

```
rcu_periph_clock_sleep_disable(RCU_FMC_SLP);
```

函数 rcu_periph_reset_enable

函数rcu_periph_reset_enable描述见下表：

表 3-583. 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 3-571. 枚举类型 rcu_periph_reset_enum
RCU_GPIOxRST	复位 GPIOx 时钟(x=A,B,C,D,E,F,G)
RCU_AFRST	复位复用功能时钟
RCU_ENETRST	复位以太网时钟
RCU_USBFIRST	复位 USBFS 时钟
RCU_TIMERxRST	复位 TIMERx 时钟(x=0,1,2,3,4,5,6,7,8,9,10,11,12,13)
RCU_WWDGTRST	复位 WWDGT 时钟
RCU_SPIxRST	复位 SPIx 时钟(x=0,1,2)
RCU_USARTxRST	复位 USARTx 时钟(x=0,1,2,5)
RCU_UARTxRST	复位 UARTx 时钟(x=3,4,6,7)
RCU_I2CxRST	复位 I2Cx 时钟(x=0,1,2)
RCU_CANxRST	复位 CANx 时钟(x=0,1)
RCU_PMURST	复位 PMU 时钟

<i>RCU_DACRST</i>	复位 DAC 时钟
<i>RCU_ADCxRST</i>	复位 ADCx 时钟(x=0,1,2)
<i>RCU_BKPIRST</i>	复位 BKP 接口时钟
<i>RCU_TLIRST</i>	复位 TLI 时钟
<i>RCU_DCIRST</i>	复位 DCI 时钟
<i>RCU_CAURST</i>	复位 CAU 时钟
<i>RCU_HAURST</i>	复位 HAU 时钟
<i>RCU_TRNGRST</i>	复位 TRNG 时钟
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* enable SPI0 reset */
```

```
rcu_periph_reset_enable(RCU_SPI0RST);
```

函数 `rcu_periph_reset_disable`

函数 `rcu_periph_reset_disable` 描述见下表:

表 3-584. 函数 `rcu_periph_reset_disable`

函数名称	<code>rcu_periph_reset_disable</code>
函数原形	<code>void rcu_periph_reset_disable(rcu_periph_reset_enum periph_reset);</code>
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 3-571. 枚举类型 <code>rcu_periph_reset_enum</code>
<i>RCU_GPIOxRST</i>	复位 GPIOx 时钟(x=A,B,C,D,E,F,G)
<i>RCU_AFRST</i>	复位复用功能时钟
<i>RCU_ENETRST</i>	复用以太网时钟
<i>RCU_USBFSRST</i>	复位 USBFS 时钟
<i>RCU_TIMERxRST</i>	复位 TIMERx 时钟(x=0,1,2,3,4,5,6,7,8,9,10,11,12,13)
<i>RCU_WWDGTRST</i>	复位 WWDGT 时钟
<i>RCU_SPIxRST</i>	复位 SPIx 时钟(x=0,1,2)
<i>RCU_USARTxRST</i>	复位 USARTx 时钟(x=0,1,2,5)
<i>RCU_UARTxRST</i>	复位 UARTx 时钟(x=3,4,6,7)
<i>RCU_I2CxRST</i>	复位 I2Cx 时钟(x=0,1,2)
<i>RCU_CANxRST</i>	复位 CANx 时钟(x=0,1)
<i>RCU_PMURST</i>	复位 PMU 时钟
<i>RCU_DACRST</i>	复位 DAC 时钟
<i>RCU_ADCxRST</i>	复位 ADCx 时钟(x=0,1,2)

<i>RCU_BKPIRST</i>	复位 BKP 接口时钟
<i>RCU_TLIRST</i>	复位 TLI 时钟
<i>RCU_DCIRST</i>	复位 DCI 时钟
<i>RCU_CAURST</i>	复位 CAU 时钟
<i>RCU_HAURST</i>	复位 HAU 时钟
<i>RCU_TRNGRST</i>	复位 TRNG 时钟
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* disable SPI0 reset */
rcu_periph_reset_disable(RCU_SPI0RST);
```

函数 rcu_bkp_reset_enable

函数rcu_bkp_reset_enable描述见下表:

表 3-585. 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 3-586. 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 3-587. 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
<i>RCU_CKSYSSRC_I RC8M</i>	选择 CK_IRC8M 时钟作为 CK_SYS 时钟源
<i>RCU_CKSYSSRC_ HXTAL</i>	选择 CK_HXTAL 时钟作为 CK_SYS 时钟源
<i>RCU_CKSYSSRC_ PLL</i>	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 3-588. 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：

```
uint32_t temp_cksys_status;

/* get the CK_SYS source */

temp_cksys_status = rcu_system_clock_source_get();
```

函数 rcu_ahb_clock_config

函数rcu_ahb_clock_config描述见下表：

表 3-589. 函数 rcu_ahb_clock_config

函数名称	rcu_ahb_clock_config
函数原形	void rcu_ahb_clock_config(uint32_t ck_ahb);
功能描述	配置 AHB 时钟预分频选择
先决条件	-
被调用函数	-
输入参数{in}	
ck_ahb	AHB 预分频选择
RCU_AHB_CKSYS	选择 CK_SYS 时钟 x 分频 (x=1, 2, 4, 8, 16, 64, 128, 256, 512)
_DIVx	
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 3-590. 函数 rcu_apb1_clock_config

函数名称	rcu_apb1_clock_config
函数原形	void rcu_apb1_clock_config(uint32_t ck_apb1);
功能描述	配置 APB1 时钟预分频选择
先决条件	-
被调用函数	-
输入参数{in}	
ck_apb1	APB1 预分频选择
RCU_APB1_CKAHB_DIV1	选择 CK_AHB 为 CK_APB1
RCU_APB1_CKAHB_DIV2	选择 CK_AHB/2 为 CK_APB1
RCU_APB1_CKAHB_DIV4	选择 CK_AHB/4 为 CK_APB1
RCU_APB1_CKAHB_DIV8	选择 CK_AHB/8 为 CK_APB1
RCU_APB1_CKAHB_DIV16	选择 CK_AHB/16 为 CK_APB1
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* configure CK_AHB/16 as CK_APB1 */
```

```
rcu_apb1_clock_config(RCU_APB1_CKAHB_DIV16);
```

函数 rcu_apb2_clock_config

函数rcu_apb2_clock_config描述见下表：

表 3-591. 函数 rcu_apb2_clock_config

函数名称	rcu_apb2_clock_config
函数原形	void rcu_apb2_clock_config(uint32_t ck_apb2);
功能描述	配置 APB2 时钟预分频选择
先决条件	-
被调用函数	-
输入参数{in}	

ck_apb2	APB2 预分频选择
RCU_APB2_CKAHB_DIV1	选择 CK_AHB 为 CK_APB2
RCU_APB2_CKAHB_DIV2	选择 CK_AHB/2 为 CK_APB2
RCU_APB2_CKAHB_DIV4	选择 CK_AHB/4 为 CK_APB2
RCU_APB2_CKAHB_DIV8	选择 CK_AHB/8 为 CK_APB2
RCU_APB2_CKAHB_DIV16	选择 CK_AHB/16 为 CK_APB2
输出参数{out}	
-	-
返回值	
-	-

例如:

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

函数 rcu_ckout0_config

函数rcu_ckout0_config描述见下表:

表 3-592. 函数 rcu_ckout0_config

函数名称	rcu_ckout0_config
函数原形	void rcu_ckout0_config(uint32_t ckout0_src, uint32_t ckout0_div);
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_NONE	无时钟输出
RCU_CKOUT0SRC_CKSYS	选择系统时钟 CK_SYS
RCU_CKOUT0SRC_IRC8M	选择内部 8M RC 振荡器时钟
RCU_CKOUT0SRC_HXTAL	选择高速晶体振荡器时钟 (HXTAL)
RCU_CKOUT0SRC_CKPLL_DIV2	选择 (CK_PLL / 2) 时钟
RCU_CKOUT0SRC	选择 CK_PLL1 时钟

<code>_CKPLL1</code>	
<code>RCU_CKOUT0SRC</code> <code>_CKPLL2_DIV2</code>	选择 (CK_PLL2 / 2) 时钟
<code>RCU_CKOUT0SRC</code> <code>_EXT1</code>	选择EXT1时钟
<code>RCU_CKOUT0SRC</code> <code>_CKPLL2</code>	选择CK_PLL2时钟
输入参数{out}	
<code>ckout0_div</code>	CK_OUT0 分频器
<code>RCU_CKOUT0_DIV</code> <code>x(x=1..64)</code>	CK_OUT0 分频系数
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* configure the HXTAL as CK_OUT0 clock source */
```

```
rcu_ckout0_config(RCU_CKOUT0SRC_HXTAL, RCU_CKOUT0_DIV1);
```

函数 `rcu_ckout1_config`

函数`rcu_ckout1_config`描述见下表:

表 3-593. 函数 `rcu_ckout1_config`

函数名称	<code>rcu_ckout1_config</code>
函数原形	<code>void rcu_ckout1_config(uint32_t ckout1_src, uint32_t ckout1_div);</code>
功能描述	配置 CKOUT1 时钟源选择
先决条件	-
被调用函数	-
输入参数{in}	
<code>ckout1_src</code>	CK_OUT1 时钟源选择
<code>RCU_CKOUT1SRC</code> <code>_NONE</code>	无时钟输出
<code>RCU_CKOUT1SRC</code> <code>_CKSYS</code>	选择系统时钟 CK_SYS
<code>RCU_CKOUT1SRC</code> <code>_IRC8M</code>	选择内部 8M RC 振荡器时钟
<code>RCU_CKOUT1SRC</code> <code>_HXTAL</code>	选择高速晶体振荡器时钟 (HXTAL)
<code>RCU_CKOUT1SRC</code> <code>_CKPLL_DIV2</code>	选择 (CK_PLL / 2) 时钟
<code>RCU_CKOUT0SRC</code>	选择 CK_PLL1 时钟

<code>_CKPLL1</code>	
<code>RCU_CKOUT1SRC</code> <code>_CKPLL2</code>	选择 (CK_PLL2 / 2) 时钟
<code>RCU_CKOUT1SRC</code> <code>_EXT1</code>	选择EXT1时钟
<code>RCU_CKOUT1SRC</code> <code>_CKPLL2</code>	选择CK_PLL2时钟
输入参数{out}	
<code>ckout1_div</code>	CK_OUT1 分频器
<code>RCU_CKOUT1_DIV</code> $x(x=1..64)$	CK_OUT1 分频系数
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* configure the HXTAL as CK_OUT1 clock source */
```

```
rcu_ckout1_config(RCU_CKOUT1SRC_HXTAL, RCU_CKOUT1_DIV1);
```

函数 rcu_pll_config

函数rcu_pll_config描述见下表:

表 3-594. 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}	
<code>pll_src</code>	PLL 时钟源选择
<code>RCU_PLLSRC_IRC</code> <code>8M_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}	
<code>pll_mul</code>	PLL 时钟倍频因子
<code>RCU_PLL_MULx</code>	$x = 2..14, 16..32, 65$
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

函数rcu_predv0_config描述见下表:

表 3-595. 函数 rcu_predv0_config

函数名称	rcu_predv0_config
函数原形	void rcu_predv0_config(uint32_t predv0_source, uint32_t predv0_div);
功能描述	配置 PREDV0 分频因子
先决条件	-
被调用函数	-
输入参数{in}	
predv0_source	PREDV0 输入时钟源
RCU_PREDV0SRC_HXTAL	HXTAL 被选择为 PREDV0 的时钟源
RCU_PREDV0SRC_CKPLL1	CK_PLL1 被选择为 PREDV0 的时钟源
输入参数{in}	
predv0_div	PREDV0 分频因子
RCU_PREDV0_DIV_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

函数rcu_predv1_config描述见下表:

表 3-596. 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

函数rcu_pll1_config描述见下表:

表 3-597. 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
<i>RCU_PLL1_MULx</i>	PLL1 源时钟*x, (x = 8..16, 20)
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* configure the PLL1 clock */
```

```
rcu_pll1_config(RCU_PLL1_MUL8);
```

函数 rcu_pll2_config

函数rcu_pll2_config描述见下表:

表 3-598. 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
<i>RCU_PLL2_MULx</i>	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 3-599. 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
<i>RCU_CKADC_CKA</i> <i>PB2_DIV2</i>	CK_ADC = CK_APB2 / 2
<i>RCU_CKADC_CKA</i> <i>PB2_DIV4</i>	CK_ADC = CK_APB2 / 4
<i>RCU_CKADC_CKA</i> <i>PB2_DIV6</i>	CK_ADC = CK_APB2 / 6
<i>RCU_CKADC_CKA</i> <i>PB2_DIV8</i>	CK_ADC = CK_APB2 / 8
<i>RCU_CKADC_CKA</i> <i>PB2_DIV12</i>	CK_ADC = CK_APB2 / 12
<i>RCU_CKADC_CKA</i> <i>PB2_DIV16</i>	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_usbfs_trng_clock_config

函数rcu_usbfs_trng_clock_config描述见下表：

表 3-600. 函数 rcu_usbfs_trng_clock_config

函数名称	rcu_usbfs_trng_clock_config
函数原形	void rcu_usbfs_trng_clock_config(uint32_t usbfs_trng_psc);
功能描述	配置 USBFS 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 3-601. 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

<i>RCU_RTCSRC_NO NE</i>	没有时钟
<i>RCU_RTCSRC_LX TAL</i>	选择 CK_LXTAL 时钟作为 RTC 的时钟源
<i>RCU_RTCSRC_IRC 40K</i>	选择 CK_IRC40K 时钟作为 RTC 的时钟源
<i>RCU_RTCSRC_HX TAL_DIV_128</i>	选择 CK_HXTAL / 128 时钟作为 RTC 的时钟源
输出参数{out}	
-	-
返回值	
-	-

例如：

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

函数 rcu_i2s1_clock_config

函数rcu_i2s1_clock_config描述见下表：

表 3-602. 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}	
<i>i2s_clock_source</i>	I2S 时钟源选择
<i>RCU_I2S1SRC_CK SYS</i>	系统时钟被选择为 I2S1 时钟的时钟源
<i>RCU_I2S1SRC_CK PLL2_MUL2</i>	(CK_PLL2 * 2) 被选择为I2S1时钟的时钟源
输出参数{out}	
-	-
返回值	
-	-

例如：

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

函数 rcu_i2s2_clock_config

函数rcu_i2s2_clock_config描述见下表：

表 3-603. 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_CKSYS	系统时钟被选择为 I2S2 时钟的时钟源
RCU_I2S2SRC_CKPLL2_MUL2	(CK_PLL2 * 2) 被选择为 I2S2 时钟的时钟源
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* configure the I2S2 clock source selection */
```

```
rcu_i2s2_clock_config(RCU_I2S2SRC_CKPLL2_MUL2);
```

函数 rcu_pllt_config

函数rcu_pllt_config描述见下表：

表 3-604. 函数 rcu_pllt_config

函数名称	rcu_pllt_config
函数原形	void rcu_pllt_config(uint32_t pllt_src);
功能描述	配置 PLLT 的时钟源选择
先决条件	-
被调用函数	
输入参数{in}	
pllt_src	PLLT 时钟源选择
RCU_PLLTSRC_IRC8M	IRC8M 被选择为 PLLT 时钟的时钟源
RCU_PLLTSRC_HXTAL	HXTAL 被选择为 PLLT 时钟的时钟源
输出参数{out}	
-	-
返回值	

-	-
---	---

例如:

```
/* configure the PLLT clock selection */
rcu_pllt_config(RCU_PLLTSRC_IRC8M);
```

函数 rcu_pllt_vco_config

函数rcu_pllt_vco_config描述见下表:

表 3-605. 函数 rcu_pllt_vco_config

函数名称	rcu_pllt_vco_config
函数原形	ErrStatus rcu_pllt_vco_config(uint32_t pllt_psc, uint32_t pllt_mul, uint32_t ppltr_psc);
功能描述	配置 PLLT 时钟的倍频和分频因子
先决条件	-
被调用函数	
输入参数{in}	
pllt_src	PLLT VCO 输入时钟分频因子
parameter	2~63
输入参数{in}	
pllt_mul	PLLT VCO 输出时钟倍频因子
parameter	49~432
输入参数{in}	
ppltr_psc	PLLTR 分频因子
parameter	2~7
输出参数{out}	
-	-
返回值	
ErrStatus	SUCCESS 或 ERROR

例如:

```
/* configure the PLLT clock multiplication and division factors */
if(SUCCESS == rcu_pllt_vco_config(0x4, 0x64, 0x4)){
}
}
```

函数 rcu_tli_clock_config

函数rcu_tli_clock_config描述见下表:

表 3-606. 函数 rcu_tli_clock_config

函数名称	rcu_tli_clock_config
函数原形	void rcu_tli_clock_config(uint32_t tli_psc);

功能描述	配置 TLI 分频系数
先决条件	-
被调用函数	
输入参数{in}	
tli_psc	TLI 分频系数
RCU_CKTLI_CKPL LTR_DIV2	CK_PLLTR/2
RCU_CKTLI_CKPL LTR_DIV4	CK_PLLTR/4
RCU_CKTLI_CKPL LTR_DIV8	CK_PLLTR/8
RCU_CKTLI_CKPL LTR_DIV16	CK_PLLTR/16
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* configure the TLI prescaler factor from PLLTR clock */
```

```
rcu_tli_clock_config(RCU_CKTLI_CKPLLTR_DIV2);
```

函数 rcu_lxtal_drive_capability_config

函数rcu_lxtal_drive_capability_config描述见下表：

表 3-607. 函数 rcu_lxtal_drive_capability_config

函数名称	rcu_lxtal_drive_capability_config
函数原形	void rcu_lxtal_drive_capability_config(uint32_t lxtal_dricap);
功能描述	配置 LXTAL 驱动能力
先决条件	-
被调用函数	-
输入参数{in}	
lxtal_dricap	LXTAL 驱动能力
RCU_LXTAL_LOW DRI	弱驱动能力
RCU_LXTAL_MED_ LOWDRI	中低驱动能力
RCU_LXTAL_MED_ HIGHDRI	中高驱动能力
RCU_LXTAL_HIGH DRI	强驱动能力
输出参数{out}	

-	-
返回值	
-	-

例如：

```
/* configure the LXTAL drive capability */
```

```
rcu_lxtal_drive_capability_config(RCU_LXTAL_LOWDRI);
```

函数 rcu_osci_stab_wait

函数rcu_osci_stab_wait描述见下表：

表 3-608. 函数 rcu_osci_stab_wait

函数名称	rcu_osci_stab_wait
函数原形	ErrStatus rcu_osci_stab_wait(rcu_osci_type_enum osci);
功能描述	等待振荡器稳定标志位置位或振荡器起振超时
先决条件	-
被调用函数	rcu_flag_get
输入参数{in}	
osci	振荡器类型，参考 表 3-576. 枚举类型 rcu_osci_type_enum
RCU_IRC8M	内部 8M RC 振荡器
RCU_HXTAL	高速晶体振荡器
RCU_PLL_CK	锁相环
RCU_PLL1_CK	锁相环 1
RCU_PLL2_CK	锁相环 2
RCU_LXTAL	低速晶体振荡器
RCU_IRC40K	内部 40K RC 振荡器
RCU_PLLT_CK	TLI 锁相环
输出参数{out}	
-	-
返回值	
ErrStatus	SUCCESS 或 ERROR

例如：

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

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

函数 rcu_osci_on

函数rcu_osci_on描述见下表：

表 3-609. 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 3-576. 枚举类型 rcu_osc_i_type_enum
RCU_IRC8M	内部 8M RC 振荡器
RCU_HXTAL	高速晶体振荡器
RCU_PLL_CK	锁相环
RCU_PLL1_CK	锁相环 1
RCU_PLL2_CK	锁相环 2
RCU_LXTAL	低速晶体振荡器
RCU_IRC40K	内部 40K RC 振荡器
RCU_PLLT_CK	TLI 锁相环
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 3-610. 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 3-576. 枚举类型 rcu_osc_i_type_enum
RCU_IRC8M	内部 8M RC 振荡器
RCU_HXTAL	高速晶体振荡器
RCU_PLL_CK	锁相环
RCU_PLL1_CK	锁相环 1
RCU_PLL2_CK	锁相环 2
RCU_LXTAL	低速晶体振荡器

<i>RCU_IRC40K</i>	内部 40K RC 振荡器
<i>RCU_PLLT_CK</i>	TLI 锁相环
输出参数{out}	
-	-
返回值	
-	-

例如：

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

```
rcu_osci_off(RCU_HXTAL);
```

函数 `rcu_osci_bypass_mode_enable`

函数 `rcu_osci_bypass_mode_enable` 描述见下表：

表 3-611. 函数 `rcu_osci_bypass_mode_enable`

函数名称	<code>rcu_osci_bypass_mode_enable</code>
函数原形	<code>void rcu_osci_bypass_mode_enable(rcu_osci_type_enum osci);</code>
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 3-576. 枚举类型 <code>rcu_osci_type_enum</code>
<i>RCU_HXTAL</i>	高速晶体振荡器
<i>RCU_LXTAL</i>	低速晶体振荡器
输出参数{out}	
-	-
返回值	
-	-

例如：

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

```
rcu_osci_bypass_mode_enable(RCU_HXTAL);
```

函数 `rcu_osci_bypass_mode_disable`

函数 `rcu_osci_bypass_mode_disable` 描述见下表：

表 3-612. 函数 `rcu_osci_bypass_mode_disable`

函数名称	<code>rcu_osci_bypass_mode_disable</code>
函数原形	<code>void rcu_osci_bypass_mode_disable(rcu_osci_type_enum osci);</code>
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 3-576. 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 3-613. 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 3-614. 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 3-615. 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 3-616. 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

<code>RCU_DEEPSLEEP_V_1_2</code>	在深度睡眠模式下内核电压为 1.2V
<code>RCU_DEEPSLEEP_V_1_1</code>	在深度睡眠模式下内核电压为 1.1V
<code>RCU_DEEPSLEEP_V_1_0</code>	在深度睡眠模式下内核电压为 1.0V
<code>RCU_DEEPSLEEP_V_0_9</code>	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 3-617. 函数 `rcu_clock_freq_get`

函数名称	<code>rcu_clock_freq_get</code>
函数原形	<code>uint32_t rcu_clock_freq_get(rcu_clock_freq_enum clock);</code>
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 3-577. 枚举类型 <code>rcu_clock_freq_enum</code>
<code>CK_SYS</code>	系统时钟
<code>CK_AHB</code>	AHB 时钟
<code>CK_APB1</code>	APB1 时钟
<code>CK_APB2</code>	APB2 时钟
输出参数{out}	
-	-
返回值	
ck_freq	系统时钟/AHB 时钟/APB1 时钟/APB2 时钟频率

例如：

```
uint32_t temp_freq;

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

函数 rcu_flag_get

函数rcu_flag_get描述见下表：

表 3-618. 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 3-572. 枚举类型 rcu_flag_enum
RCU_FLAG_IRC8M STB	IRC8M 稳定标志
RCU_FLAG_HXTAL STB	HXTAL 稳定标志
RCU_FLAG_PLLST B	PLL 稳定标志
RCU_FLAG_PLL1S TB	PLL1 稳定标志
RCU_FLAG_PLL2S TB	PLL2 稳定标志
RCU_FLAG_PLLTS TB	PLLT 稳定标志
RCU_FLAG_LXTAL STB	LXTAL 稳定标志
RCU_FLAG_IRC40 KSTB	IRC40K 稳定标志
RCU_FLAG_EPRS T	外部引脚复位标志
RCU_FLAG_PORR ST	电源复位标志
RCU_FLAG_SWRS T	软件复位标志
RCU_FLAG_FWDG TRST	独立看门狗定时器复位标志
RCU_FLAG_WWD GTRST	窗口看门狗定时器复位标志
RCU_FLAG_LPRST	low-power 复位标志
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 3-619. 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_enable

函数rcu_interrupt_enable描述见下表：

表 3-620. 函数 rcu_interrupt_enable

函数名称	rcu_interrupt_enable
函数原形	void rcu_interrupt_enable(rcu_int_enum stab_int);
功能描述	使能时钟稳定中断
先决条件	-
被调用函数	-
输入参数{in}	
stab_int	时钟稳定中断
RCU_INT_IRC40KS TB	IRC40K 稳定中断
RCU_INT_LXTALS TB	LXTAL 稳定中断
RCU_INT_IRC8MS	IRC8M 稳定中断使能

<i>TB</i>	
<i>RCU_INT_HXTALS</i> <i>TB</i>	HXTAL 稳定中断
<i>RCU_INT_PLLSTB</i>	PLL 稳定中断
<i>RCU_INT_PLL1STB</i>	PLL1 稳定中断
<i>RCU_INT_PLL2STB</i>	PLL2 稳定中断
<i>RCU_INT_PLLTST</i> <i>B</i>	PLLT 稳定中断
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* enable the HXTAL stabilization interrupt */
rcu_interrupt_enable(RCU_INT_HXTALSTB);
```

函数 rcu_interrupt_disable

函数rcu_interrupt_disable描述见下表：

表 3-621. 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
<i>RCU_INT_IRC40KS</i> <i>TB</i>	IRC40K 稳定中断
<i>RCU_INT_LXTALS</i> <i>TB</i>	LXTAL 稳定中断
<i>RCU_INT_IRC8MS</i> <i>TB</i>	IRC8M 稳定中断使能
<i>RCU_INT_HXTALS</i> <i>TB</i>	HXTAL 稳定中断
<i>RCU_INT_PLLSTB</i>	PLL 稳定中断
<i>RCU_INT_PLL1STB</i>	PLL1 稳定中断
<i>RCU_INT_PLL2STB</i>	PLL2 稳定中断
<i>RCU_INT_PLLTST</i> <i>B</i>	PLLT 稳定中断
输出参数{out}	

-	-
返回值	
-	-

例如：

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

函数 rcu_interrupt_flag_get

函数rcu_interrupt_flag_get描述见下表：

表 3-622. 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 3-573. 枚举类型 rcu_int_flag_enum
RCU_INT_FLAG_IRC40KSTB	IRC40K 稳定中断标志
RCU_INT_FLAG_LXTALSTB	LXTAL 稳定中断标志
RCU_INT_FLAG_IRC8MSTB	IRC8M 稳定中断标志
RCU_INT_FLAG_HXTALSTB	HXTAL 稳定中断标志
RCU_INT_FLAG_PLLSTB	PLL 稳定中断标志
RCU_INT_FLAG_PLL1STB	PLL1 稳定中断标志
RCU_INT_FLAG_PLL2STB	PLL2 稳定中断标志
RCU_INT_FLAG_CKMKM	HXTAL 时钟阻塞中断标志
RCU_INT_FLAG_PLLTSTB	PLLT 稳定中断标志
输出参数{out}	
-	-
返回值	
FlagStatus	SET 或 RESET

例如：

```

/* get the clock stabilization interrupt flag */

if(SET == rcu_interrupt_flag_get(RCU_INT_FLAG_HXTALSTB)){

}

```

函数 rcu_interrupt_flag_clear

函数rcu_interrupt_flag_clear描述见下表：

表 3-623. 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 3-574. 枚举类型 rcu_int_flag_clear_enum
RCU_INT_FLAG_IRC40KSTB_CLR	IRC40K 稳定中断标志清除
RCU_INT_FLAG_LX TALSTB_CLR	LXTAL 稳定中断标志清除
RCU_INT_FLAG_IRC8MSTB_CLR	IRC8M 稳定中断标志清除
RCU_INT_FLAG_HXTALSTB_CLR	HXTAL 稳定中断标志清除
RCU_INT_FLAG_PLLSTB_CLR	PLL 稳定中断标志清除
RCU_INT_FLAG_PLL1STB_CLR	PLL1 稳定中断标志清除
RCU_INT_FLAG_PLL2STB_CLR	PLL2 稳定中断标志清除
RCU_INT_FLAG_CKM_CLR	时钟阻塞中断标志清除
RCU_INT_FLAG_PLLTSTB_CLR	PLL T 稳定中断标志清除
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);
```

3.22. 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[3.22.1](#)描述了RTC的寄存器列表，章节[3.22.2](#)对FWDGT库函数进行说明。

3.22.1. 外设寄存器说明

RTC寄存器列表如下表所示：

表 3-624. 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

3.22.2. 外设库函数说明

RTC库函数列表如下表所示：

表 3-625. 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_lwoff_wait	等待最近一次对RTC寄存器的写操作完成
rtc_register_sync_wait	等待RTC寄存器(RTC_CNTx、RTC_ALRMx和RTC_PSCx)与RTC的APB时钟同步
rtc_counter_get	获取RTC计数器的值
rtc_counter_set	设置RTC计数器的值
rtc_prescaler_set	设置RTC预分频值
rtc_alarm_config	设置RTC闹钟值
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

库函数名称	库函数描述
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 3-626. 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 3-627. 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_lwoff_wait

函数rtc_lwoff_wait描述见下表:

表 3-628. 函数 rtc_lwoff_wait

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

例如:

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

```
rtc_lwoff_wait();
```

函数 rtc_register_sync_wait

函数rtc_register_sync_wait描述见下表:

表 3-629. 函数 rtc_register_sync_wait

函数名称	rtc_register_sync_wait
函数原型	void rtc_register_sync_wait(void);
功能描述	等待RTC寄存器(RTC_CNTx、RTC_ALRMx和RTC_PSCx)与RTC的APB时钟同步
先决条件	-
被调用函数	-
输入参数{in}	
-	-
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* wait for RTC registers synchronization */
```

```
rtc_register_sync_wait();
```

函数 rtc_counter_get

函数rtc_counter_get描述见下表：

表 3-630. 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_counter_set

函数rtc_counter_set描述见下表：

表 3-631. 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_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}	
cnt	RTC计数器的值（0-0xFFFF FFFF）
输出参数{out}	
-	-
返回值	
-	-

例如：

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

rtc_lwoff_wait();

/* set counter value to 0xFFFF */
```

```
rtc_counter_set(0xFFFF);
```

函数 rtc_prescaler_set

函数rtc_prescaler_set描述见下表：

表 3-632. 函数 rtc_prescaler_set

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

例如：

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

```
rtc_lwoff_wait();
```

```
/* set RTC prescaler value to 0x7FFFF */
```

```
rtc_prescaler_set (0x7FFFF);
```

函数 rtc_alarm_config

函数rtc_alarm_config描述见下表：

表 3-633. 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_divider_get

函数rtc_divider_get描述见下表:

表 3-634. 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 3-635. 函数 rtc_flag_get

函数名称	rtc_flag_get
函数原型	FlagStatus rtc_flag_get(uint32_t flag);
功能描述	获取RTC标志位状态
先决条件	-
被调用函数	-
输入参数{in}	
flag	需要获取的RTC标志位
RTC_FLAG_SECON D	秒中断标志位
RTC_FLAG_ALARM	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
<i>RTC_FLAG_LWOF</i>	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 3-636. 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_lwof_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
<i>RTC_FLAG_SECON</i> <i>D</i>	秒中断标志位
<i>RTC_FLAG_ALARM</i>	闹钟中断标志位
<i>RTC_FLAG_OVERFLOW</i> <i>LOW</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_lwof_wait();

/* clear the RTC alarm flag */
```

```
rtc_flag_clear(RTC_FLAG_ALARM);
```

函数 rtc_interrupt_enable

函数rtc_interrupt_enable描述见下表：

表 3-637. 函数 rtc_interrupt_enable

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

例如：

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

```
rtc_lwoff_wait();
```

```
/* enable the RTC second interrupt */
```

```
rtc_interrupt_enable(RTC_INT_SECOND);
```

函数 rtc_interrupt_disable

函数rtc_interrupt_disable描述见下表：

表 3-638. 函数 rtc_interrupt_disable

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

RTC_INT_OVERFLOW	溢出中断
输出参数{out}	
-	-
返回值	
-	-

例如：

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

```
rtc_lwoff_wait();
```

```
/* disable the RTC second interrupt */
```

```
rtc_interrupt_disable(RTC_INT_SECOND);
```

3.23. 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[3.23.1](#)描述了SDIO的寄存器列表，章节[3.23.2](#)对SDIO库函数进行说明。

3.23.1. 外设寄存器说明

SDIO寄存器列表如下表所示：

表 3-639. SDIO 寄存器

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

3.23.2. 外设库函数说明

SDIO库函数列表如下表所示：

表 3-640. SDIO 库函数

库函数名称	库函数描述
sdio_deinit	复位SDIO
sdio_clock_config	配置SDIO时钟
sdio_hardware_clock_enable	使能硬件时钟控制
sdio_hardware_clock_disable	禁能硬件时钟控制
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	禁能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	禁能命令状态机
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	禁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	禁能SDIO的DMA请求
sdio_readwait_enable	使能读等待模式（仅限SD I/O模式）
sdio_readwait_disable	禁能读等待模式（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	禁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	禁能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	禁能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	禁能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)

库函数名称	库函数描述
sdio_ceata_interrupt_disable	禁能CE-ATA中断(仅限CE-ATA模式)
sdio_ceata_command_completion_enable	使能CE-ATA命令完成信号(仅限CE-ATA模式)
sdio_ceata_command_completion_disable	禁能CE-ATA命令完成信号(仅限CE-ATA模式)
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	禁能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

函数 sdio_deinit

函数sdio_deinit描述见下表:

表 3-641. 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 3-642. 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_SDIOLCKED GE_RISING	选择SDIOCLK的上升沿产生SDIO_CLK
SDIO_SDIOLCKED GE_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 SS_ENABLE	使能旁路时钟
SDIO_CLOCKBYPASS SS_DISABLE	禁能旁路时钟
输入参数{in}	
clock_powersave	SDIO_CLK时钟动态开启/关闭以节省功耗
SDIO_CLOCKPWR SAVE_ENABLE	SDIO_CLK时钟在总线空闲时关闭
SDIO_CLOCKPWR SAVE_DISABLE	SDIO_CLK时钟总是开启
输入参数{in}	
clock_division	时钟分频，小于512
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 3-643. 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 3-644. 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	禁能硬件时钟控制
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 3-645. 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

<i>SDIO_BUSMODE_4</i> <i>BIT</i>	4位SDIO卡总线模式
<i>SDIO_BUSMODE_8</i> <i>BIT</i>	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 3-646. 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
<i>SDIO_POWER_ON</i>	SDIO上电
<i>SDIO_POWER_OFF</i>	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 3-647. 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 3-648. 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 3-649. 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	禁能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 3-650. 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 3-651. 函数 sdio_wait_type_set

函数名称	sdio_wait_type_set
函数原形	void sdio_wait_type_set(uint32_t wait_type);
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}	
wait_type	等待类型
SDIO_WAITTYPE_NO	不等待中断
SDIO_WAITTYPE_INTERRUPT	等待中断
SDIO_WAITTYPE_DATAEND	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 3-652. 函数 sdio_csm_enable

函数名称	sdio_csm_enable
函数原形	void sdio_csm_enable(void);
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 3-653. 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	禁能命令状态机
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 3-654. 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 3-655. 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[4] = {0, 0, 0, 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 3-656. 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 3-657. 函数 sdio_data_transfer_config

函数名称	sdio_data_transfer_config
函数原形	void sdio_data_transfer_config(uint32_t transfer_mode, uint32_t transfer_direction);
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}	
transfer_mode	数据传输模式
SDIO_TRANSMODE_BLOCK	块传输模式
SDIO_TRANSMODE_STREAM	流传输或SDIO多字节传输模式
输入参数{in}	
transfer_direction	数据传输方向
SDIO_TRANSDIRECTION_TOCARD	写数据到卡上
SDIO_TRANSDIRECTION_TOSDIO	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 3-658. 函数 sdio_dsm_enable

函数名称	sdio_dsm_enable
函数原形	void sdio_dsm_enable(void);
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 3-659. 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	禁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 3-660. 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 3-661. 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 3-662. 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 3-663. 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 3-664. 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 3-665. 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	禁能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_readwait_enable

函数sdio_readwait_enable描述见下表：

表 3-666. 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 3-667. 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	禁能读等待模式（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 3-668. 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 3-669. 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	禁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 3-670. 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 3-671. 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 3-672. 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	禁能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 3-673. 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 3-674. 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	禁能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 3-675. 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 3-676. 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	禁能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 3-677. 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 3-678. 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	禁能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 3-679. 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 3-680. 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	禁能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();
```

函数 sdio_flag_get

函数sdio_flag_get描述见下表:

表 3-681. 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_CCRCERR	命令响应已接收（CRC检测失败）
SDIO_FLAG_DTCRCERR	数据块已发送/已接收（CRC检测失败）
SDIO_FLAG_CMDTMO	命令响应超时
SDIO_FLAG_DTTMO	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_DATACNT为零）
SDIO_FLAG_STBIT	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_CMDRUN	正在传输命令
SDIO_FLAG_TXRUN	正在传输数据
SDIO_FLAG_RXRUN	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

<i>SDIO_FLAG_RFF</i>	接收FIFO为满
<i>SDIO_FLAG_TFE</i>	发送FIFO为空
<i>SDIO_FLAG_RFE</i>	接收FIFO为空
<i>SDIO_FLAG_TXDT VAL</i>	发送FIFO中的数据有效
<i>SDIO_FLAG_RXDT VAL</i>	接收FIFO中的数据有效
<i>SDIO_FLAG_SDIOI NT</i>	SD I/O中断已接收
<i>SDIO_FLAG_ATAE ND</i>	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 3-682. 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
<i>SDIO_FLAG_CCRC ERR</i>	命令响应已接收（CRC检测失败）
<i>SDIO_FLAG_DTCR CERR</i>	数据块已发送/已接收（CRC检测失败）
<i>SDIO_FLAG_CMDT MOUT</i>	命令响应超时
<i>SDIO_FLAG_DTTM OUT</i>	数据超时
<i>SDIO_FLAG_TXUR E</i>	发送FIFO下溢错误发生

<i>SDIO_FLAG_RXOR E</i>	接收FIFO上溢错误发生
<i>SDIO_FLAG_CMDR ECV</i>	命令响应已接收（CRC检测通过）
<i>SDIO_FLAG_CMDS END</i>	命令已发送（不需响应）
<i>SDIO_FLAG_DTEN D</i>	数据结束（数据计数器，SDIO_DATACNT为零）
<i>SDIO_FLAG_STBIT E</i>	总线上起始位错误
<i>SDIO_FLAG_DTB LKEND</i>	数据块已发送/已接收（CRC检测通过）
<i>SDIO_FLAG_SDIOI NT</i>	SD I/O中断已接收
<i>SDIO_FLAG_ATAE ND</i>	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 3-683. 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
<i>SDIO_INT_CCRCE RR</i>	命令响应CRC错误中断
<i>SDIO_INT_DTCRC ERR</i>	数据CRC错误中断
<i>SDIO_INT_CMDTM OUT</i>	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_DTLKE 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：

```
/* 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 3-684. 函数 `sdio_interrupt_disable`

函数名称	<code>sdio_interrupt_disable</code>
函数原形	<code>void sdio_interrupt_disable(uint32_t int_flag);</code>
功能描述	禁能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
<code>SDIO_INT_CCRCE</code> <code>RR</code>	命令响应CRC错误中断
<code>SDIO_INT_DTCRC</code> <code>ERR</code>	数据CRC错误中断
<code>SDIO_INT_CMDTM</code> <code>OUT</code>	命令响应超时中断
<code>SDIO_INT_DTTMO</code> <code>UT</code>	数据超时中断
<code>SDIO_INT_TXURE</code>	发送FIFO下溢错误中断
<code>SDIO_INT_RXORE</code>	接收FIFO上溢错误中断
<code>SDIO_INT_CMDRE</code> <code>CV</code>	命令响应已接收中断
<code>SDIO_INT_CMDSE</code> <code>ND</code>	命令已发送中断
<code>SDIO_INT_DTEND</code>	数据结束中断
<code>SDIO_INT_STBITE</code>	起始位错误中断
<code>SDIO_INT_DTBLKE</code> <code>ND</code>	数据块已发送/已接收中断
<code>SDIO_INT_CMDRU</code> <code>N</code>	正在传输命令中断
<code>SDIO_INT_TXRUN</code>	正在传输数据中断
<code>SDIO_INT_RXRUN</code>	正在接收数据中断
<code>SDIO_INT_TFH</code>	发送FIFO半满中断
<code>SDIO_INT_RFH</code>	接收FIFO半满中断
<code>SDIO_INT_TFF</code>	发送FIFO满中断
<code>SDIO_INT_RFF</code>	接收FIFO满中断
<code>SDIO_INT_TFE</code>	发送FIFO空中断
<code>SDIO_INT_RFE</code>	接收FIFO空中断
<code>SDIO_INT_TXDTVA</code> <code>L</code>	发送FIFO中的数据有效中断
<code>SDIO_INT_RXDTV</code> <code>AL</code>	接收FIFO中的数据有效中断
<code>SDIO_INT_SDIOIN</code> <code>T</code>	SD I/O中断已接收中断
<code>SDIO_INT_ATAEN</code>	CE-ATA命令完成信号已接收中断

<i>D</i>	
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 3-685. 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_FLAG_C CRCERR	命令响应CRC错误中断
SDIO_INT_FLAG_D TCRCERR	数据CRC错误中断
SDIO_INT_FLAG_C MDTMOUT	命令响应超时中断
SDIO_INT_FLAG_D TTMOUT	数据超时中断
SDIO_INT_FLAG_T XURE	发送FIFO下溢错误中断
SDIO_INT_FLAG_R XORE	接收FIFO上溢错误中断
SDIO_INT_FLAG_C MDRECV	命令响应已接收中断
SDIO_INT_FLAG_C MDSEND	命令已发送中断
SDIO_INT_FLAG_D TEND	数据结束中断
SDIO_INT_FLAG_S TBITE	起始位错误中断
SDIO_INT_FLAG_D	数据块已发送/已接收中断

<i>TBLKEND</i>	
<i>SDIO_INT_FLAG_CMDRUN</i>	正在传输命令中断
<i>SDIO_INT_FLAG_TXRUN</i>	正在传输数据中断
<i>SDIO_INT_FLAG_RXRUN</i>	正在接收数据中断
<i>SDIO_INT_FLAG_TXFH</i>	发送FIFO半满中断
<i>SDIO_INT_FLAG_RXFH</i>	接收FIFO半满中断
<i>SDIO_INT_FLAG_TXFF</i>	发送FIFO满中断
<i>SDIO_INT_FLAG_RXFF</i>	接收FIFO满中断
<i>SDIO_INT_FLAG_TXFE</i>	发送FIFO空中断
<i>SDIO_INT_FLAG_RXFE</i>	接收FIFO空中断
<i>SDIO_INT_FLAG_TXDVAL</i>	发送FIFO中的数据有效中断
<i>SDIO_INT_FLAG_RXDVAL</i>	接收FIFO中的数据有效中断
<i>SDIO_INT_FLAG_SDIOINT</i>	SD I/O中断已接收中断
<i>SDIO_INT_FLAG_ATAEND</i>	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 3-686. 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_FLAG_C CRCERR	命令响应CRC错误中断
SDIO_INT_FLAG_D TCRCERR	数据CRC错误中断
SDIO_INT_FLAG_C MDTMOUT	命令响应超时中断
SDIO_INT_FLAG_D TTMOUT	数据超时中断
SDIO_INT_FLAG_T XURE	发送FIFO下溢错误中断
SDIO_INT_FLAG_R XORE	接收FIFO上溢错误中断
SDIO_INT_FLAG_C MDRECV	命令响应已接收中断
SDIO_INT_FLAG_C MDSEND	命令已发送中断
SDIO_INT_FLAG_D TEND	数据结束中断
SDIO_INT_FLAG_S TBITE	起始位错误中断
SDIO_INT_FLAG_D TBLKEND	数据块已发送/已接收中断
SDIO_INT_FLAG_S DIOINT	SD I/O中断已接收中断
SDIO_INT_FLAG_A TAEND	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_FLAG_DTEND);
```

3.24. 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[3.24.1](#)描述了SPI/I2S的寄存器列表，章节[3.24.2](#)对SPI/I2S库函数进行说明。

3.24.1. 外设寄存器说明

SPI/I2S寄存器列表如下表所示：

表 3-687. SPI/I2S 寄存器

寄存器名称	寄存器描述
SPI_CTL0	控制寄存器0
SPI_CTL1	控制寄存器1
SPI_STAT	状态寄存器
SPI_DATA	数据寄存器
SPI_CRCPOLY	CRC多项式寄存器
SPI_RCRC	接收CRC寄存器
SPI_TCRC	发送CRC寄存器
SPI_I2SCTL	I2S控制寄存器
SPI_I2SPSC	I2S时钟分频寄存器
SPI_QCTL	SPI0四路SPI控制寄存器

3.24.2. 外设库函数说明

SPI/I2S库函数列表如下表所示：

表 3-688. 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PI NSS输出
spi_nss_output_disable	禁能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	禁能SPI DMA功能

库函数名称	库函数描述
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_quad_enable	使能四线SPI模式
spi_quad_disable	禁能四线SPI模式
spi_quad_write_enable	使能四线SPI写
spi_quad_read_enable	使能四线SPI读
spi_quad_io23_output_enable	使能SPI_IO2和SPI_IO3输出
spi_quad_io23_output_disable	禁能SPI_IO2和SPI_IO3输出
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 3-689. 结构体 spi_parameter_struct

成员名称	功能描述
device_mode	主机或设备模式配置 (SPI_MASTER, SPI_SLAVE)
trans_mode	传输模式 (SPI_TRANSMODE_FULLDUPLEX, SPI_TRANSMODE_RECEIVEONLY, SPI_TRANSMODE_BDRECEIVE, SPI_TRANSMODE_BDTRANSMIT)
frame_size	数据帧格式配置 (SPI_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_CK_PL_LOW_PH_1EDGE, SPI_CK_PL_HIGH_PH_1EDGE, SPI_CK_PL_LOW_PH_2EDGE, SPI_CK_PL_HIGH_PH_2EDGE)
prescale	预分频器配置

(SPI_PSC_n (n=2,4,8,16,32,64,128,256))

函数 spi_i2s_deinit

函数spi_i2s_deinit描述见下表:

表 3-690. 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)	SPI外设选择
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 3-691. 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3-689. 结构体spi_parameter_struct
返回值	
-	-

例如:

/* initialize the parameters of SPI */

spi_parameter_struct spi_init_struct;

```
spi_struct_para_init(&spi_init_struct);
```

函数 spi_init

函数spi_init描述见下表：

表 3-692. 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)	SPI外设选择
输入参数{in}	
spi_struct	初始化结构体，结构体成员参考 表3-689. 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 3-693. 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)	SPI外设选择
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 3-694. 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SPIx
SPIx(x=0,1,2)	SPI外设选择
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 3-695. 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)	I2S外设选择
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_MASTE RTX	I2S主机发送模式
I2S_MODE_MASTE RRX	I2S主机接收模式
输入参数{in}	
standard	I2S标准选择
I2S_STD_PHILIPS	I2S飞利浦标准
I2S_STD_MSB	I2S MSB对齐标准
I2S_STD_LSB	I2S LSB对齐标准
I2S_STD_PCMSHO RT	I2S PCM短帧标准
I2S_STD_PCMLON G	I2S PCM长帧标准
输入参数{in}	
ckpl	I2S空闲状态时钟极性
I2S_CKPL_LOW	2S_CK空闲状态为低电平
I2S_CKPL_HIGH	2S_CK空闲状态为高电平
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* initialize I2S1 */
```

```
i2s_init(SPI1, I2S_MODE_MASTERTX, I2S_STD_PHILIPS, I2S_CKPL_LOW);
```

函数 i2s_psc_config

函数i2s_psc_config描述见下表:

表 3-696. 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
SPIx(x=1,2)	I2S外设选择
输入参数{in}	
audiosample	I2S音频采样频率
I2S_AUDIOSAMPL E_8K	音频采样频率为8KHz
I2S_AUDIOSAMPL E_11K	音频采样频率为11KHz
I2S_AUDIOSAMPL E_16K	音频采样频率为16KHz
I2S_AUDIOSAMPL E_22K	音频采样频率为22KHz
I2S_AUDIOSAMPL E_32K	音频采样频率为32KHz
I2S_AUDIOSAMPL E_44K	音频采样频率为44KHz
I2S_AUDIOSAMPL E_48K	音频采样频率为48KHz
I2S_AUDIOSAMPL E_96K	音频采样频率为96KHz
I2S_AUDIOSAMPL E_192K	音频采样频率为192KHz
输入参数{in}	
frameformat	I2S数据长度和通道长度
I2S_FRAMEFORMA T_DT16B_CH16B	I2S数据长度为16位，通道长度为16位
I2S_FRAMEFORMA T_DT16B_CH32B	I2S数据长度为16位，通道长度为32位
I2S_FRAMEFORMA T_DT24B_CH32B	I2S数据长度为24位，通道长度为32位
I2S_FRAMEFORMA T_DT32B_CH32B	I2S数据长度为32位，通道长度为32位
输入参数{in}	
mckout	2S_MCK输出使能
I2S_MCKOUT_ENA	I2S_MCK输出使能

<i>BLE</i>	
<i>I2S_MCKOUT_DISABLE</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 3-697. 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
SPIx(x=1,2)	I2S外设选择
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 3-698. 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
<i>SPIx</i> (x=1,2)	I2S外设选择
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 3-699. 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
<i>SPIx</i> (x=0,1,2)	SPI外设选择
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 3-700. 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SPI NSS输出
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)	SPI外设选择
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 3-701. 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
<i>SPIx</i> (x=0,1,2)	SPI外设选择
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 3-702. 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
<i>SPIx</i> (x=0,1,2)	SPI外设选择
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 3-703. 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
<i>SPIx</i> (x=0,1,2)	SPI外设选择
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 DMA function */
```

```
spi_dma_enable(SPI0, SPI_DMA_TRANSMIT);
```

函数 spi_dma_disable

函数spi_dma_disable描述见下表:

表 3-704. 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
SPIx(x=0,1,2)	SPI外设选择
输入参数{in}	
dma	SPI DMA模式
SPI_DMA_TRANSMIT	SPI发送缓冲区DMA使能
SPI_DMA_RECEIVE	SPI接收缓冲区DMA使能
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* disable SPI0 transmit data DMA function */
```

```
spi_dma_disable(SPI0, SPI_DMA_TRANSMIT);
```

函数 spi_i2s_data_frame_format_config

函数spi_i2s_data_frame_format_config描述见下表:

表 3-705. 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)	SPI外设选择
输入参数{in}	
frame_format	SPI帧大小

<i>SPI_FRAME_SIZE_16BIT</i>	SPI 16位数据帧格式
<i>SPI_FRAME_SIZE_8BIT</i>	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 3-706. 函数 `spi_bidirectional_transfer_config`

函数名称	<code>spi_bidirectional_transfer_config</code>
函数原形	<code>void spi_bidirectional_transfer_config(uint32_t spi_periph, uint32_t transfer_direction);</code>
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
<i>SPIx</i> (x=0,1,2)	SPI外设选择
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 3-707. 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
SPIx(x=0,1,2)	SPI外设选择
输入参数{in}	
data	16位数据
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* SPI0 transmit data */
uint16_t spi0_send_array[10];
uint8_t send_n = 1;
spi_i2s_data_transmit(SPI0, spi0_send_array[send_n]);
```

函数 spi_i2s_data_receive

函数spi_i2s_data_receive描述见下表：

表 3-708. 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)	SPI外设选择
输出参数{out}	
-	-
返回值	
uint16_t	16位数据

例如：

```
/* SPI0 receive data */

uint16_t spi0_receive_array[10];

uint8_t receive_n = 1;

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

函数 spi_crc_polynomial_set

函数spi_crc_polynomial_set描述见下表：

表 3-709. 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)	SPI外设选择
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 */

uint16_t CRC_VALUE = 0x8;

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 3-710. 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
<i>SPIx(x=0,1,2)</i>	SPI外设选择
输出参数{out}	
-	-
返回值	
uint16_t	16位CRC多项式值（0-0xFFFF）

例如：

```
/* get SPI0 CRC polynomial */
uint16_t CRC_VALUE;
CRC_VALUE = spi_crc_polynomial_get(SPI0);
```

函数 spi_crc_on

函数spi_crc_on描述见下表：

表 3-711. 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
<i>SPIx(x=0,1,2)</i>	SPI外设选择
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 3-712. 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
<i>SPIx</i> (x=0,1,2)	SPI外设选择
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 3-713. 函数 **spi_crc_next**

函数名称	spi_crc_next
函数原形	void spi_crc_next(uint32_t spi_periph);
功能描述	设置外设SPIx下一次传输数据为CRC值
先决条件	-
被调用函数	-
输入参数{in}	
spi_periph	外设SPIx
<i>SPIx</i> (x=0,1,2)	SPI外设选择
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 3-714. 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
<i>SPIx</i> (x=0,1,2)	SPI外设选择
输入参数{in}	
crc	SPI CRC值
<i>SPI_CRC_TX</i>	获取发送CRC寄存器值
<i>SPI_CRC_RX</i>	获取接收CRC寄存器值
输出参数{out}	
-	-
返回值	
uint16_t	16位CRC值（0-0xFFFF）

例如：

```
/* get SPI0 CRC send value */
uint16_t value;
value = spi_crc_get(SPI0, SPI_CRC_TX);
```

函数 spi_quad_enable

函数spi_quad_enable描述见下表：

表 3-715. 函数 spi_quad_enable

函数名称	spi_quad_enable
函数原形	void spi_quad_enable(uint32_t spi_periph);
功能描述	使能四线SPI模式
先决条件	-
被调用函数	-
输入参数{in}	
spi_periph	外设SPIx
<i>SPIx</i> (x=0,1,2)	SPI外设选择
输出参数{out}	
-	-
返回值	
-	-

例如：

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

函数 spi_quad_disable

函数spi_quad_disable描述见下表：

表 3-716. 函数 spi_quad_disable

函数名称	spi_quad_disable
函数原形	void spi_quad_disable(uint32_t spi_periph);
功能描述	禁能四线SPI模式
先决条件	-
被调用函数	-
输入参数{in}	
spi_periph	外设SPIx
SPIx(x=0,1,2)	SPI外设选择
输出参数{out}	
-	-
返回值	
-	-

例如：

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

函数 spi_quad_write_enable

函数spi_quad_write_enable描述见下表：

表 3-717. 函数 spi_quad_write_enable

函数名称	spi_quad_write_enable
函数原形	void spi_quad_write_enable(uint32_t spi_periph);
功能描述	使能四线SPI写
先决条件	-
被调用函数	-
输入参数{in}	
spi_periph	外设SPIx
SPIx(x=0,1,2)	SPI外设选择
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* enable quad wire SPI write */
spi_quad_write_enable(SPI0);
```

函数 spi_quad_read_enable

函数spi_quad_read_enable描述见下表：

表 3-718. 函数 `spi_quad_read_enable`

函数名称	<code>spi_quad_read_enable</code>
函数原形	<code>void spi_quad_read_enable(uint32_t spi_periph);</code>
功能描述	使能四线SPI读
先决条件	-
被调用函数	-
输入参数{in}	
<code>spi_periph</code>	外设SPIx
<code>SPIx(x=0,1,2)</code>	SPI外设选择
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* enable quad wire SPI read */
spi_quad_read_enable(SPI0);
```

函数 `spi_quad_io23_output_enable`

函数`spi_quad_io23_output_enable`描述见下表：

表 3-719. 函数 `spi_quad_io23_output_enable`

函数名称	<code>spi_quad_io23_output_enable</code>
函数原形	<code>void spi_quad_io23_output_enable(uint32_t spi_periph);</code>
功能描述	使能SPI_IO2和SPI_IO3输出
先决条件	-
被调用函数	-
输入参数{in}	
<code>spi_periph</code>	外设SPIx
<code>SPIx(x=0)</code>	SPI外设选择
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* enable SPI_IO2 and SPI_IO3 pin output */
spi_quad_io23_output_enable(SPI0);
```

函数 `spi_quad_io23_output_disable`

函数`spi_quad_io23_output_disable`描述见下表：

表 3-720. 函数 `spi_quad_io23_output_disable`

函数名称	<code>spi_quad_io23_output_disable</code>
函数原形	<code>void spi_quad_io23_output_disable(uint32_t spi_periph);</code>
功能描述	禁能SPI_IO2和SPI_IO3输出
先决条件	-
被调用函数	-
输入参数{in}	
<code>spi_periph</code>	外设SPIx
<code>SPIx(x=0)</code>	SPI外设选择
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* disable SPI_IO2 and SPI_IO3 pin output */
spi_quad_io23_output_disable(SPI0);
```

函数 `spi_i2s_flag_get`

函数`spi_i2s_flag_get`描述见下表:

表 3-721. 函数 `spi_i2s_flag_get`

函数名称	<code>spi_i2s_flag_get</code>
函数原形	<code>FlagStatus spi_i2s_flag_get(uint32_t spi_periph, uint32_t flag);</code>
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}	
<code>spi_periph</code>	外设SPIx
<code>SPIx(x=0,1,2)</code>	SPI外设选择
输入参数{in}	
<code>flag</code>	SPI/I2S标志状态
<code>SPI_FLAG_TBE</code>	发送缓冲区空标志
<code>SPI_FLAG_RBNE</code>	接收缓冲区非空标志
<code>SPI_FLAG_TRANS</code>	通信进行中标志
<code>SPI_I2S_INT_FLAG_RXORERR</code>	接收过载错误标志
<code>SPI_FLAG_CONFERR</code>	配置错误标志
<code>SPI_FLAG_CRCERR</code>	CRC错误标志
<code>I2S_FLAG_TBE</code>	发生缓冲区空标志

<i>I2S_FLAG_RBNE</i>	收缓冲区非空标志
<i>I2S_FLAG_TRANS</i>	通信进行中标志
<i>I2S_FLAG_RXORE</i> <i>RR</i>	接收过载错误标志
<i>I2S_FLAG_TXURE</i> <i>RR</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 3-722. 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
<i>SPIx</i> (x=0,1,2)	SPI外设选择
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 3-723. 函数 spi_i2s_interrupt_disable

函数名称	spi_i2s_interrupt_disable
函数原形	void spi_i2s_interrupt_disable(uint32_t spi_periph, uint8_t interrupt);
功能描述	禁能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)	SPI外设选择
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 3-724. 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)	SPI外设选择
输入参数{in}	
interrupt	SPI/I2S中断状态

<code>SPI_I2S_INT_FLAG_TBE</code>	发送缓冲区空中断
<code>SPI_I2S_INT_FLAG_RBNE</code>	接收缓冲区非空中断
<code>SPI_I2S_INT_FLAG_RXOERR</code>	接收过载错误中断
<code>SPI_INT_FLAG_CONFERR</code>	配置错误中断
<code>SPI_INT_FLAG_CRCERR</code>	CRC错误中断
<code>I2S_INT_FLAG_TXURERR</code>	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 3-725. 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
<code>SPIx(x=0,1,2)</code>	SPI外设选择
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* clear SPI0 CRC error flag status */
```

```
spi_crc_error_clear(SPI0);
```

3.25. 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[3.25.1](#)描述了TIMER的寄存器列表，章节[3.25.2](#)对TIMER库函数进行说明。

3.25.1. 外设寄存器说明

TIMER寄存器列表如下表所示：

表 3-726. 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_DMACFG	DMA配置寄存器
TIMER_DMATB	DMA发送缓冲区寄存器

3.25.2. 外设库函数说明

TIMER库函数列表如下表所示：

表 3-727. TIMER 库函数

库函数名称	库函数描述
timer_deinit	复位外设TIMER
timer_struct_para_init	初始化外设TIMER结构体参数
timer_init	初始化外设TIMER
timer_enable	使能外设TIMER
timer_disable	禁能外设TIMER
timer_auto_reload_shadow_enable	TIMER自动重载影子寄存器使能
timer_auto_reload_shadow_disable	TIMER自动重载影子寄存器禁能
timer_update_event_enable	TIMER更新使能
timer_update_event_disable	TIMER更新禁能
timer_counter_alignment	设置外设TIMER的对齐模式
timer_counter_up_direction	设置外设TIMER向上计数
timer_counter_down_direction	设置外设TIMER向下计数
timer_prescaler_config	配置外设TIMER预分频器
timer_repetition_value_config	配置外设TIMER的重复计数器
timer_autoreload_value_config	配置外设TIMER的自动重载寄存器
timer_counter_value_config	配置外设TIMER的计数器值
timer_counter_read	读取外设TIMER的计数器值
timer_prescaler_read	读取外设TIMER的预分频器值
timer_single_pulse_mode_config	配置外设TIMER的单脉冲模式
timer_update_source_config	配置外设TIMER的更新源
timer_dma_enable	外设TIMER的DMA使能
timer_dma_disable	外设TIMER的DMA禁能
timer_channel_dma_request_source_select	外设TIMER的通道DMA请求源选择
timer_dma_transfer_config	配置外设TIMER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TIMER中止功能
timer_break_enable	使能TIMER的中止功能
timer_break_disable	禁能TIMER的中止功能
timer_automatic_output_enable	TIMER自动输出使能
timer_automatic_output_disable	TIMER自动输出禁能
timer_primary_output_config	TIMER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
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TIMER的通道输出配置
timer_channel_output_mode_config	配置外设TIMER通道输出比较模式

库函数名称	库函数描述
timer_channel_output_pulse_value_config	配置外设TIMER的通道输出比较值
timer_channel_output_shadow_config	配置TIMER通道输出比较影子寄存器功能
timer_channel_output_fast_config	配置TIMER通道输出比较快速功能
timer_channel_output_clear_config	配置TIMER的通道输出比较清0功能
timer_channel_output_polarity_config	通道输出极性配置
timer_channel_complementary_output_polarity_config	互补通道输出极性配置
timer_channel_output_state_config	配置通道状态
timer_channel_complementary_output_state_config	配置互补通道输出状态
timer_channel_input_struct_para_init	初始化外设TIMER通道输入结构体参数
timer_input_capture_config	配置TIMER输入捕获参数
timer_channel_input_capture_prescaler_config	配置TIMER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TIMER捕获PWM输入参数
timer_hall_mode_config	配置TIMER的HALL接口功能
timer_input_trigger_source_select	TIMER的输入触发源选择
timer_master_output_trigger_source_select	选择TIMER主模式输出触发
timer_slave_mode_select	TIMER从模式选择
timer_master_slave_mode_config	TIMER主从模式配置
timer_external_trigger_config	配置TIMER外部触发输入
timer_quadrature_decoder_mode_config	TIMER配置为编码器模式
timer_internal_clock_config	TIMER配置为内部时钟模式
timer_internal_trigger_as_external_clock_config	配置TIMER的内部触发作为时钟源
timer_external_trigger_as_external_clock_config	配置TIMER的外部触发作为时钟源
timer_external_clock_mode0_config	配置TIMER外部时钟模式0
timer_external_clock_mode1_config	配置TIMER外部时钟模式1
timer_external_clock_mode1_disable	TIMER外部时钟模式1禁能
timer_flag_get	获取外设TIMER的状态标志
timer_flag_clear	清除外设TIMER状态标志
timer_interrupt_enable	外设TIMER中断使能
timer_interrupt_disable	外设TIMER中断禁能
timer_interrupt_flag_get	获取外设TIMER中断标志
timer_interrupt_flag_clear	清除外设TIMER的中断标志

结构体 timer_parameter_struct

表 3-728. 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 3-729. 结构体 timer_break_parameter_struct

成员名称	功能描述
runoffstate	运行模式下“关闭状态”配置（TIMER_ROS_STATE_ENABLE, TIMER_ROS_STATE_DISABLE）
ideloffstate	空闲模式下“关闭状态”配置（TIMER_IOS_STATE_ENABLE, TIMER_IOS_STATE_DISABLE）
deadtime	死区时间（0~255）
breakpolarity	中止信号极性（TIMER_BREAK_POLARITY_LOW, TIMER_BREAK_POLARITY_HIGH）
outputautostate	自动输出使能（TIMER_OUTAUTO_ENABLE, TIMER_OUTAUTO_DISABLE）
protectmode	互补寄存器保护控制（TIMER_CCHP_PROT_OFF, TIMER_CCHP_PROT_0, TIMER_CCHP_PROT_1, TIMER_CCHP_PROT_2）
breakstate	中止使能（TIMER_BREAK_ENABLE, TIMER_BREAK_DISABLE）

结构体 timer_oc_parameter_struct

表 3-730. 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 3-731. 结构体 timer_ic_parameter_struct

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

函数 timer_deinit

函数timer_deinit描述见下表:

表 3-732. 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TIMER
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 3-733. 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}	
-	-
输出参数{out}	
initpara	初始化结构体，结构体成员参考 表3-728. 结构体timer_parameter_struct
返回值	
-	-

例如：

```
/* initialize the TIMER structure */

timer_parameter_struct initpara;

timer_struct_para_init(&initpara);
```

函数 timer_init

函数timer_init描述见下表：

表 3-734. 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TIMER
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3-728. 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(TIMERO,&timer_initpara);
```

函数 timer_enable

函数timer_enable描述见下表:

表 3-735. 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TIMER
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 TIMERO */
```

```
timer_enable(TIMERO);
```

函数 timer_disable

函数timer_disable描述见下表:

表 3-736. 函数 timer_disable

函数名称	timer_disable
函数原型	void timer_disable(uint32_t timer_periph);
功能描述	禁能外设TIMER
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 3-737. 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	TIMER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 3-738. 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	TIMER自动重载影子禁能
先决条件	-
被调用函数	-
输入参数{in}	
timer_periph	TIMER外设
TIMERx(x=0..13)	TIMER外设选择
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* disable the TIMER0 auto reload shadow function */
```

```
timer_auto_reload_shadow_disable(TIMER0);
```

函数 timer_update_event_enable

函数timer_update_event_enable描述见下表:

表 3-739. 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	TIMER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 3-740. 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	TIMER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 3-741. 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TIMER的对齐模式
先决条件	-
被调用函数	-
输入参数{in}	
timer_periph	TIMER外设
TIMERx(x=0..4, 7..13)	TIMER外设选择
输入参数{in}	
aligned	对齐模式
TIMER_COUNTER_EDGE	无中央对齐计数模式（边沿对齐模式），DIR位指定了计数方向
TIMER_COUNTER_CENTER_DOWN	中央对齐向下计数置1模式。计数器在中央计数模式计数，通道被配置在输出模式（TIMERx_CHCTL0寄存器中CHxMS=00），只有在向下计数时，通道的比较中断标志置1
TIMER_COUNTER_CENTER_UP	中央对齐向上计数置1模式。计数器在中央计数模式计数，通道被配置在输出模式（TIMERx_CHCTL0寄存器中CHxMS=00），只有在向上计数时，通道的比较中断标志置1
TIMER_COUNTER_CENTER_BOTH	中央对齐上下计数置1模式。计数器在中央计数模式计数，通道被配置在输出模式（TIMERx_CHCTL0寄存器中CHxMS=00），在向上和向下计数时，通道的比较中断标志都会置1
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* set TIMER0 counter center-aligned and counting up 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 3-742. 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TIMER向上计数
先决条件	计数器设置为无中央对齐计数模式(边沿对齐模式)
被调用函数	-
输入参数{in}	
timer_periph	TIMER外设
TIMERx(x=0..4, 7..13)	TIMER外设选择
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* set TIMER0 counter up direction */
```

```
timer_counter_up_direction(TIMER0);
```

函数 timer_counter_down_direction

函数timer_counter_down_direction描述见下表:

表 3-743. 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TIMER向下计数
先决条件	计数器设置为无中央对齐计数模式(边沿对齐模式)
被调用函数	-
输入参数{in}	
timer_periph	TIMER外设
TIMERx(x=0..4, 7..13)	TIMER外设选择
输出参数{out}	
-	-
返回值	
-	-

例如:

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

```
timer_counter_down_direction(TIMER0);
```

函数 timer_prescaler_config

函数timer_prescaler_config描述见下表：

表 3-744. 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TIMER预分频器
先决条件	-
被调用函数	-
输入参数{in}	
timer_periph	TIMER外设
TIMERx(x=0..13)	TIMER外设选择
输入参数{in}	
prescaler	预分频值，0~65535
输入参数{in}	
pscreload	预分频值加载模式
TIMER_PSC_RELOAD_NOW	预分频值立即加载
TIMER_PSC_RELOAD_UPDATE	预分频值在下次更新事件加载
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 3-745. 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TIMER的重复计数器
先决条件	-
被调用函数	-
输入参数{in}	
timer_periph	TIMER外设
TIMERx(x=0,7)	TIMER外设选择

输入参数{in}	
repetition	重复计数器值，取值范围0~255
输出参数{out}	
-	-
返回值	
-	-

例如：

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

```
timer_repetition_value_config(TIMER0, 98);
```

函数 timer_autoreload_value_config

函数timer_autoreload_value_config描述见下表：

表 3-746. 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TIMER的自动重载寄存器
先决条件	-
被调用函数	-
输入参数{in}	
timer_periph	TIMER外设
TIMERx(x=0..13)	TIMER外设选择
输入参数{in}	
autoreload	计数器自动重载值，0~65535
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* configure TIMER autoreload register value */
```

```
timer_autoreload_value_config(TIMER0,3000);
```

函数 timer_counter_value_config

函数timer_counter_value_config描述见下表：

表 3-747. 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TIMER的计数器值
先决条件	-

被调用函数	-
输入参数{in}	
timer_periph	TIMER外设
TIMERx(x=0..13)	TIMER外设选择
输入参数{in}	
counter	计数器值，0~65535
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 3-748. 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TIMER的计数器值
先决条件	-
被调用函数	-
输入参数{in}	
timer_periph	TIMER外设
TIMERx(x=0..13)	TIMER外设选择
输出参数{out}	
-	-
返回值	
uint32_t	外设TIMER的计数器值

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 3-749. 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TIMER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
TIMER(x=0..13)	TIMER外设选择
输出参数{out}	
-	-
返回值	
uint16_t	外设TIMER的预分频器值，0~65535

例如：

```
/* read TIMER0 prescaler value */
```

```
uint16_t i = 0;
```

```
i = timer_prescaler_read(TIMER0);
```

函数 timer_single_pulse_mode_config

函数timer_single_pulse_mode_config描述见下表：

表 3-750. 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TIMER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
TIMER(x=0..8,11)	TIMER外设选择
输入参数{in}	
spmode	脉冲模式
TIMER_SP_MODE_SINGLE	单脉冲模式计数
TIMER_SP_MODE_REPETITIVE	重复模式计数
输出参数{out}	
-	-
返回值	
-	-

例如：

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

函数 timer_update_source_config

函数timer_update_source_config描述见下表：

表 3-751. 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TIMER的更新源
先决条件	-
被调用函数	-
输入参数{in}	
timer_periph	TIMER外设
TIMERx(x=0..13)	TIMER外设选择
输入参数{in}	
update	更新源
TIMER_UPDATE_SRC_GLOBAL	下述任一事件产生更新中断或DMA请求： – UPG位被置1 – 计数器溢出/下溢 – 从模式控制器产生的更新
TIMER_UPDATE_SRC_REGULAR	只有计数器溢出/ 下溢才产生更新中断或DMA请求
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* configure TIMER update only by counter overflow/underflow */
timer_update_source_config(TIMER0, TIMER_UPDATE_SRC_REGULAR);
```

函数 timer_dma_enable

函数timer_dma_enable描述见下表：

表 3-752. 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TIMER的DMA使能
先决条件	-

被调用函数	-
输入参数{in}	
timer_periph	TIMER外设
TIMERx	外设选择, x参考具体参数
输入参数{in}	
dma	DMA源
TIMER_DMA_UPD	更新DMA请求, TIMERx (x=0..7)
TIMER_DMA_CH0 D	通道0比较/捕获DMA请求, TIMERx (x=0..4, 7)
TIMER_DMA_CH1 D	通道1比较/捕获DMA请求, TIMERx (x=0..4, 7)
TIMER_DMA_CH2 D	通道2比较/捕获DMA请求, TIMERx (x=0..4, 7)
TIMER_DMA_CH3 D	通道3比较/捕获DMA请求, TIMERx (x=0..4, 7)
TIMER_DMA_CMT D	换相DMA更新请求, TIMERx (x=0, 7)
TIMER_DMA_TRG D	触发DMA请求使能, TIMERx (x=0..4, 7)
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* enable the TIMER0 update DMA */
```

```
timer_dma_enable(TIMER0, TIMER_DMA_UPD);
```

函数 timer_dma_disable

函数timer_dma_disable描述见下表:

表 3-753. 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TIMER的DMA禁能
先决条件	-
被调用函数	-
输入参数{in}	
timer_periph	TIMER外设
TIMERx	外设选择, x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_CH0D</i>	通道0比较/捕获DMA请求, <i>TIMERx</i> (<i>x</i> =0..4, 7)
<i>TIMER_DMA_CH1D</i>	通道1比较/捕获DMA请求, <i>TIMERx</i> (<i>x</i> =0..4, 7)
<i>TIMER_DMA_CH2D</i>	通道2比较/捕获DMA请求, <i>TIMERx</i> (<i>x</i> =0..4, 7)
<i>TIMER_DMA_CH3D</i>	通道3比较/捕获DMA请求, <i>TIMERx</i> (<i>x</i> =0..4, 7)
<i>TIMER_DMA_CMTD</i>	换相DMA更新请求, <i>TIMERx</i> (<i>x</i> =0, 7)
<i>TIMER_DMA_TRGD</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 3-754. 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TIMER的通道DMA请求源选择
先决条件	-
被调用函数	-
输入参数{in}	
timer_periph	TIMER外设
<i>TIMERx</i> (<i>x</i> =0..4, 7)	TIMER外设选择
输入参数{in}	
dma_request	通道的DMA请求源选择
<i>TIMER_DMAREQUEST_CHANNELEVENT</i>	当通道捕获/比较事件发生时, 发送通道 <i>x</i> 的DMA请求
<i>TIMER_DMAREQUEST_UPDATEEVENT</i>	当更新事件发生, 发送通道 <i>x</i> 的DMA请求

NT	
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 3-755. 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TIMER的DMA模式
先决条件	-
被调用函数	-
输入参数{in}	
timer_periph	TIMER外设
TIMERx	外设选择，x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)

<i>DMATA_CHCTL1</i>	7)
<i>TIMER_DMACFG_DMATA_CHCTL2</i>	DMA传输起始地址: <i>TIMER_DMACFG_DMATA_CHCTL2</i> , <i>TIMERx</i> (<i>x</i> =0..4, 7)
<i>TIMER_DMACFG_DMATA_CNT</i>	DMA传输起始地址: <i>TIMER_DMACFG_DMATA_CNT</i> , <i>TIMERx</i> (<i>x</i> =0..4, 7)
<i>TIMER_DMACFG_DMATA_PSC</i>	DMA传输起始地址: <i>TIMER_DMACFG_DMATA_PSC</i> , <i>TIMERx</i> (<i>x</i> =0..4, 7)
<i>TIMER_DMACFG_DMATA_CAR</i>	DMA传输起始地址: <i>TIMER_DMACFG_DMATA_CAR</i> , <i>TIMERx</i> (<i>x</i> =0..4, 7)
<i>TIMER_DMACFG_DMATA_CREP</i>	DMA传输起始地址: <i>TIMER_DMACFG_DMATA_CREP</i> , <i>TIMERx</i> (<i>x</i> =0, 7)
<i>TIMER_DMACFG_DMATA_CH0CV</i>	DMA传输起始地址: <i>TIMER_DMACFG_DMATA_CH0CV</i> , <i>TIMERx</i> (<i>x</i> =0..4, 7)
<i>TIMER_DMACFG_DMATA_CH1CV</i>	DMA传输起始地址: <i>TIMER_DMACFG_DMATA_CH1CV</i> , <i>TIMERx</i> (<i>x</i> =0..4, 7)
<i>TIMER_DMACFG_DMATA_CH2CV</i>	DMA传输起始地址: <i>TIMER_DMACFG_DMATA_CH2CV</i> , <i>TIMERx</i> (<i>x</i> =0..4, 7)
<i>TIMER_DMACFG_DMATA_CH3CV</i>	DMA传输起始地址: <i>TIMER_DMACFG_DMATA_CH3CV</i> , <i>TIMERx</i> (<i>x</i> =0..4, 7)
<i>TIMER_DMACFG_DMATA_CCHP</i>	DMA传输起始地址: <i>TIMER_DMACFG_DMATA_CCHP</i> , <i>TIMERx</i> (<i>x</i> =0, 7)
<i>TIMER_DMACFG_DMATA_DMACFG</i>	DMA传输起始地址: <i>TIMER_DMACFG_DMATA_DMACFG</i> , <i>TIMERx</i> (<i>x</i> =0..4, 7)
<i>TIMER_DMACFG_DMATA_DMATB</i>	DMA传输起始地址: <i>TIMER_DMACFG_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_DMACFG_DMATC_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 3-756. 函数 timer_event_software_generate

函数名称	timer_event_software_generate
------	-------------------------------

函数原型	void timer_event_software_generate(uint32_t timer_periph, uint16_t event);
功能描述	软件产生事件
先决条件	-
被调用函数	-
输入参数{in}	
timer_periph	TIMER外设
TIMERx	外设选择, x参考具体参数
输入参数{in}	
event	事件源
TIMER_EVENT_SRC_UPG	更新事件产生, TIMERx (x=0..13)
TIMER_EVENT_SRC_CH0G	通道0捕获或比较事件发生, TIMERx (x=0..4, 7..13)
TIMER_EVENT_SRC_CH1G	通道1捕获或比较事件发生, TIMERx (x=0..4, 7, 8, 11)
TIMER_EVENT_SRC_CH2G	通道2捕获或比较事件发生, TIMERx (x=0..4, 7)
TIMER_EVENT_SRC_CH3G	通道3捕获或比较事件发生, TIMERx (x=0..4, 7)
TIMER_EVENT_SRC_CMTG	通道换相更新事件发生, TIMERx (x=0, 7)
TIMER_EVENT_SRC_TRGG	触发事件产生, TIMERx (x=0..4, 7, 8, 11)
TIMER_EVENT_SRC_BRKG	产生中止事件, TIMERx (x=0, 7)
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* software generate update event*/
```

```
timer_event_software_generate(TIMERO, TIMER_EVENT_SRC_UPG);
```

函数 timer_break_struct_para_init

函数timer_break_struct_para_init描述见下表:

表 3-757. 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	初始化break结构体参数
先决条件	-

被调用函数	-
输入参数{in}	
-	-
输出参数{out}	
breakpara	中止功能配置结构体，详见 表3-729. 结构体timer_break_parameter_struct
返回值	
-	-

例如：

```
/* initialize the TIMER break parameter structure */
```

```
timer_break_parameter_struct breakpara;
```

```
timer_break_struct_para_init(&breakpara);
```

函数 timer_break_config

函数timer_break_config描述见下表：

表 3-758. 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3-729. 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 3-759. 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TIMER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 3-760. 函数 timer_break_disable

函数名称	timer_break_disable
函数原型	void timer_break_disable(uint32_t timer_periph);
功能描述	禁能TIMER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 3-761. 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 3-762. 函数 timer_automatic_output_disable

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

-	-
返回值	
-	-

例如：

```
/* disable TIMER0 output automatic function */
```

```
timer_automatic_output_disable(TIMER0);
```

函数 timer_primary_output_config

函数timer_primary_output_config描述见下表：

表 3-763. 函数 timer_primary_output_config

函数名称	timer_primary_output_config
函数原型	void timer_primary_output_config(uint32_t timer_periph, ControlStatus newvalue);
功能描述	所有的通道输出使能
先决条件	-
被调用函数	-
输入参数{in}	
timer_periph	TIMER外设
TIMERx(x=0, 7)	TIMER外设选择
输入参数{in}	
newvalue	控制状态
ENABLE	使能
DISABLE	禁能
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* enable TIMER0 primary output function */
```

```
timer_primary_output_config(TIMER0, ENABLE);
```

函数 timer_channel_control_shadow_config

函数timer_channel_control_shadow_config描述见下表：

表 3-764. 函数 timer_channel_control_shadow_config

函数名称	timer_channel_control_shadow_config
函数原型	void timer_channel_control_shadow_config(uint32_t timer_periph, ControlStatus newvalue);
功能描述	通道换相控制影子配置

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

例如：

```
/* channel capture/compare control shadow register enable */
timer_channel_control_shadow_config(TIMER0, ENABLE);
```

函数 timer_channel_control_shadow_update_config

函数timer_channel_control_shadow_update_config描述见下表：

表 3-765. 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_UPDATECT L_CCUC	CMTG位被置1时更新影子寄存器
TIMER_UPDATECT L_CCUTRI	当CMTG位被置1或检测到TRIGI上升沿时，影子寄存器更新
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* configure TIMER0 channel control shadow register update when CMTG bit is set */
```

```
timer_channel_control_shadow_update_config(TIMER0, TIMER_UPDATECTL_CCU);
```

函数 timer_channel_output_struct_para_init

函数timer_channel_output_struct_para_init描述见下表：

表 3-766. 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* ocpa);
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}	
-	-
输出参数{out}	
ocpara	输出通道结构体，详见 表3-730. 结构体timer_oc_parameter_struct
返回值	
-	-

例如：

```
/* initialize the TIMER channel output parameter structure */
```

```
timer_oc_parameter_struct ocpa;
```

```
timer_channel_output_struct_para_init(&ocpa);
```

函数 timer_channel_output_config

函数timer_channel_output_config描述见下表：

表 3-767. 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* ocpa);
功能描述	外设TIMER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	外设选择，x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）

<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, TIMERx (x=0..4, 7)
输入参数{in}	
ocpara	输出通道结构体, 详见 表3-730. 结构体timer_oc_parameter_struct
输出参数{out}	
-	-
返回值	
-	-

例如:

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

timer_oc_parameter_struct timer_ocintpara;

timer_ocintpara.outputstate = TIMER_CCX_ENABLE;

timer_ocintpara.outputnstate = TIMER_CCXN_ENABLE;

timer_ocintpara.ocpolarity = TIMER_OC_POLARITY_HIGH;

timer_ocintpara.ocnpolarity = TIMER_OCN_POLARITY_HIGH;

timer_ocintpara.ocidlestate = TIMER_OC_IDLE_STATE_HIGH;

timer_ocintpara.ocnidlestate = TIMER_OCN_IDLE_STATE_LOW;

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

函数 timer_channel_output_mode_config

函数timer_channel_output_mode_config描述见下表:

表 3-768. 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TIMER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>	外设选择, x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> (x=0..4, 7)
输入参数{in}	
ocmode	通道输出比较模式
<i>TIMER_OC_MODE_TIMING</i>	时基模式
<i>TIMER_OC_MODE_ACTIVE</i>	匹配时设置为高
<i>TIMER_OC_MODE_INACTIVE</i>	匹配时设置为低
<i>TIMER_OC_MODE_TOGGLE</i>	匹配时翻转
<i>TIMER_OC_MODE_LOW</i>	强制为低
<i>TIMER_OC_MODE_HIGH</i>	强制为高
<i>TIMER_OC_MODE_PWM0</i>	PWM模式0
<i>TIMER_OC_MODE_PWM1</i>	PWM模式1
输出参数{out}	
-	-
返回值	
-	-

例如:

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

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

函数 timer_channel_output_pulse_value_config

函数timer_channel_output_pulse_value_config描述见下表:

表 3-769. 函数 timer_channel_output_pulse_value_config

函数名称	timer_channel_output_pulse_value_config
函数原型	void timer_channel_output_pulse_value_config(uint32_t timer_periph, uint16_t channel, uint16_t pulse);
功能描述	配置外设TIMER的通道输出比较值
先决条件	-
被调用函数	-
输入参数{in}	
timer_periph	TIMER外设
<i>TIMERx</i>	外设选择, x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, TIMERx (x=0..4, 7)
输入参数{in}	
pulse	通道输出比较值, 0~65535
输出参数{out}	
-	-
返回值	
-	-

例如:

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

```
timer_channel_output_pulse_value_config(TIMER0, TIMER_CH_0, 399);
```

函数 timer_channel_output_shadow_config

函数timer_channel_output_shadow_config描述见下表:

表 3-770. 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TIMER通道输出比较影子寄存器功能
先决条件	-
被调用函数	-
输入参数{in}	
timer_periph	TIMER外设
<i>TIMERx</i>	外设选择, x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, TIMERx (x=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
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 3-771. 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TIMER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	外设选择，x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 3-772. 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TIMER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 3-773. 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	外设选择, x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 3-774. 函数 timer_channel_complementary_output_polarity_config

函数名称	timer_channel_complementary_output_polarity_config
函数原型	void timer_channel_complementary_output_polarity_config(uint32_t timer_periph, uint16_t channel, uint16_t ocnpolarity);
功能描述	互补通道输出极性配置
先决条件	-
被调用函数	-
输入参数{in}	
timer_periph	TIMER外设
TIMERx	外设选择, x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 3-775. 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>	外设选择, x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, TIMERx (x=0..4, 7)
输入参数{in}	
state	通道状态
<i>TIMER_CCX_ENAB</i>	通道使能

LE	
TIMER_CCX_DISABLE	通道禁能
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* configure TIMER0 channel 0 enable state */
```

```
timer_channel_output_state_config(TIMER0, TIMER_CH_0, TIMER_CCX_ENABLE);
```

函数 timer_channel_complementary_output_state_config

函数timer_channel_complementary_output_state_config描述见下表：

表 3-776. 函数 timer_channel_complementary_output_state_config

函数名称	timer_channel_complementary_output_state_config
函数原型	void timer_channel_complementary_output_state_config(uint32_t timer_periph, uint16_t channel, uint16_t ocnstate);
功能描述	配置互补通道输出状态
先决条件	-
被调用函数	-
输入参数{in}	
timer_periph	TIMER外设
TIMERx(x=0, 7)	TIMER外设选择
输入参数{in}	
channel	待配置通道
TIMER_CH_0	通道0
TIMER_CH_1	通道1
TIMER_CH_2	通道2
输入参数{in}	
state	互补通道状态
TIMER_CCXN_ENABLE	互补通道使能
TIMER_CCXN_DISABLE	互补通道禁能
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* configure TIMER0 channel 0 complementary output enable state */
```

```
timer_channel_complementary_output_state_config(TIMER0, TIMER_CH_0,
TIMER_CCXN_ENABLE);
```

函数 timer_channel_input_struct_para_init

函数timer_channel_input_struct_para_init描述见下表：

表 3-777. 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}	
-	-
输出参数{out}	
icpara	输入捕获结构体，详见 表3-731. 结构体timer_ic_parameter_struct
返回值	
-	-

例如：

```
/* initialize the TIMER channel input parameter structure */
```

```
timer_ic_parameter_struct icpara;
```

```
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 3-778. 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TIMER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	外设选择，x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）

<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}	
icpara	输入捕获结构体, 详见 表3-731. 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 3-779. 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TIMER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>	外设选择, <i>x</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}	
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 3-780. 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>	外设选择, x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)
输出参数{out}	
-	-
返回值	
uint32_t	通道输入捕获值, 0~65535

例如:

```
/* read TIMER0 channel 0 capture compare register value */
```

```
uint32_t CH0_value = 0;
```

```
CH0_value = timer_channel_capture_value_register_read(TIMER0, TIMER_CH_0);
```

函数 timer_input_pwm_capture_config

函数timer_input_pwm_capture_config描述见下表:

表 3-781. 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TIMER捕获PWM输入参数
先决条件	-
被调用函数	timer_channel_input_capture_prescaler_config
输入参数{in}	
timer_periph	TIMER外设
TIMERx(x=0..4, 7, 8, 11)	TIMER外设选择
输入参数{in}	
channel	待配置通道
TIMER_CH_0	通道0
TIMER_CH_1	通道1
输入参数{in}	
icpwm	输入捕获结构体, 详见 表3-731. 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.icselecion = TIMER_IC_SELECTION_DIRECTTTI;
```

```
timer_icinitpara.icprescaler = TIMER_IC_PSC_DIV1;
```

```
timer_icinitpara.icfilter = 0x0;
```

```
timer_input_pwm_capture_config(TIMER0, TIMER_CH_0, &timer_icinitpara);
```

函数 timer_hall_mode_config

函数timer_hall_mode_config描述见下表:

表 3-782. 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TIMER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_HALLINTE RFACE_ENABLE	使能HALL接口
TIMER_HALLINTE RFACE_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 3-783. 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	TIMER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	外设选择, x参考具体参数
输入参数{in}	
intrigger	待选择的触发源

<i>TIMER_SMCFG_T RGSEL_ITI0</i>	内部触发输入0 (ITI0), TIMERx (x=0..4, 7, 8, 11)
<i>TIMER_SMCFG_T RGSEL_ITI1</i>	内部触发输入1 (ITI1), TIMERx (x=0..4, 7, 8, 11)
<i>TIMER_SMCFG_T RGSEL_ITI2</i>	内部触发输入2 (ITI2), TIMERx (x=0..4, 7, 8, 11)
<i>TIMER_SMCFG_T RGSEL_ITI3</i>	内部触发输入3 (ITI3), TIMERx (x=0..4, 7, 8, 11)
<i>TIMER_SMCFG_T RGSEL_CIOF_ED</i>	CI0的边沿标志位 (CIOF_ED), TIMERx (x=0..4, 7, 8, 11)
<i>TIMER_SMCFG_T RGSEL_CIOFE0</i>	滤波后的通道0输入 (CIOFE0), TIMERx (x=0..4, 7, 8, 11)
<i>TIMER_SMCFG_T RGSEL_C1FE1</i>	滤波后的通道1输入 (C1FE1), TIMERx (x=0..4, 7, 8, 11)
<i>TIMER_SMCFG_T RGSEL_ETIFP</i>	滤波后的外部触发输入 (ETIFP), TIMERx (x=0..4, 7)
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* select TIMER0 input trigger source */
```

```
timer_input_trigger_source_select(TIMER0, TIMER_SMCFG_TRGSEL_ITI0);
```

函数 timer_master_output_trigger_source_select

函数timer_master_output_trigger_source_select描述见下表:

表 3-784. 函数 timer_master_output_trigger_source_select

函数名称	timer_master_output_trigger_source_select
函数原型	void timer_master_output_trigger_source_select(uint32_t timer_periph, uint32_t outrigger);
功能描述	选择TIMER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	外设选择, x参考具体参数
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。

<code>TIMER_TRI_OUT_SRC_ENABLE</code>	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。（TIMERx(x=0..7)）
<code>TIMER_TRI_OUT_SRC_UPDATE</code>	更新。主模式控制器选择更新事件作为TRGO。（TIMERx(x=0..7)）
<code>TIMER_TRI_OUT_SRC_CC0</code>	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。（TIMERx(x=0..4, 7)）
<code>TIMER_TRI_OUT_SRC_O0CPRE</code>	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。（TIMERx(x=0..4, 7)）
<code>TIMER_TRI_OUT_SRC_O1CPRE</code>	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。（TIMERx(x=0..4, 7)）
<code>TIMER_TRI_OUT_SRC_O2CPRE</code>	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。（TIMERx(x=0..4, 7)）
<code>TIMER_TRI_OUT_SRC_O3CPRE</code>	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。（TIMERx(x=0..4, 7)）
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 3-785. 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	TIMER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_MO	关闭从模式

DE_DISABLE	
TIMER_QUAD_DE CODER_MODE0	正交译码器模式0
TIMER_QUAD_DE CODER_MODE1	正交译码器模式1
TIMER_QUAD_DE CODER_MODE2	正交译码器模式2
TIMER_SLAVE_MO DE_RESTART	复位模式
TIMER_SLAVE_MO DE_PAUSE	暂停模式
TIMER_SLAVE_MO DE_EVENT	事件模式
TIMER_SLAVE_MO DE_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 3-786. 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	TIMER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_S LAVE_MODE_ENA	主从模式使能

<i>BLE</i>	
<i>TIMER_MASTER_SLAVE_MODE_DISABLE</i>	主从模式禁能
<i>BLE</i>	
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 3-787. 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TIMER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_SC_DIV4	4分频
TIMER_EXT_TRIP_SC_DIV8	8分频
输入参数{in}	
expolarity	外部触发输入极性
TIMER_ETP_FALLING	低电平或者下降沿有效
TIMER_ETP_RISING	高电平或者上升沿有效

输入参数{in}	
extfilter	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 3-788. 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	TIMER配置为编码器模式
先决条件	-
被调用函数	-
输入参数{in}	
timer_periph	TIMER外设
TIMERx(x=0..4, 7, 8, 11)	TIMER外设选择
输入参数{in}	
decomode	编码器模式
TIMER_QUAD_DECODER_MODE0	根据CI0FE0的电平, 计数器在CI1FE1的边沿向上/下计数
TIMER_QUAD_DECODER_MODE1	根据CI1FE1的电平, 计数器在CI0FE0的边沿向上/下计数
TIMER_QUAD_DECODER_MODE2	根据另一个信号的输入电平, 计数器在CI0FE0和CI1FE1的边沿向上/ 下计数
输入参数{in}	
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

<code>TIMER_IC_POLARITY_RISING</code>	捕获上升边沿
<code>TIMER_IC_POLARITY_FALLING</code>	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 3-789. 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	TIMER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
<code>TIMERx</code> (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 3-790. 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TIMER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 3-791. 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TIMER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
<i>TIMER_SMCFG_T RGSEL_CIOF_ED</i>	CIO的边沿标志（CIOF_ED）
<i>TIMER_SMCFG_T RGSEL_CIOFE0</i>	滤波后的通道0输入（CIOFE0）
<i>TIMER_SMCFG_T RGSEL_CIIFE1</i>	滤波后的通道1输入（CIIFE1）
输入参数{in}	
expolarity	外部触发源极性
<i>TIMER_IC_POLARI TY_RISING</i>	外部触发源高电平或者上升沿有效
<i>TIMER_IC_POLARI TY_FALLING</i>	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 3-792. 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TIMER外部时钟模式0
先决条件	-
被调用函数	timer_external_trigger_config
输入参数{in}	
timer_periph	TIMER外设
<i>TIMERx(x=0..4, 7, 8, 11)</i>	TIMER外设选择
输入参数{in}	
extprescaler	ETI触发源预分频值
<i>TIMER_EXT_TRI_P</i>	不分频

SC_OFF	
TIMER_EXT_TRI_P SC_DIV2	2分频
TIMER_EXT_TRI_P SC_DIV4	4分频
TIMER_EXT_TRI_P SC_DIV8	8分频
输入参数{in}	
expolarity	ETI触发源极性
TIMER_ETP_FALLI NG	下降沿或者低电平有效
TIMER_ETP_RISIN G	上升沿或者高电平有效
输入参数{in}	
extfilter	ETI触发源滤波参数（0~15）
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* configure TIMER0 the external clock mode0 */
```

```
timer_external_clock_mode0_config(TIMER0, TIMER_EXT_TRI_PSC_DIV2,  
TIMER_ETP_FALLING, 0);
```

函数 timer_external_clock_mode1_config

函数timer_external_clock_mode1_config描述见下表：

表 3-793. 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TIMER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

<i>TIMER_EXT_TRI_P</i> <i>SC_DIV2</i>	2分频
<i>TIMER_EXT_TRI_P</i> <i>SC_DIV4</i>	4分频
<i>TIMER_EXT_TRI_P</i> <i>SC_DIV8</i>	8分频
输入参数{in}	
expolarity	ETI触发源极性
<i>TIMER_ETP_FALLI</i> <i>NG</i>	下降沿或者低电平有效
<i>TIMER_ETP_RISIN</i> <i>G</i>	上升沿或者高电平有效
输入参数{in}	
extfilter	ETI触发源滤波参数（0~15）
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* configure TIMER0 the external clock mode1 */
```

```
timer_external_clock_mode1_config(TIMER0, TIMER_EXT_TRI_PSC_DIV2,  
TIMER_ETP_FALLING, 0);
```

函数 timer_external_clock_mode1_disable

函数timer_external_clock_mode1_disable描述见下表：

表 3-794. 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	TIMER外部时钟模式1禁能
先决条件	-
被调用函数	-
输入参数{in}	
timer_periph	TIMER外设
<i>TIMERx(x=0..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_flag_get

函数timer_flag_get描述见下表:

表 3-795. 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TIMER的状态标志
先决条件	-
被调用函数	-
输入参数{in}	
timer_periph	TIMER外设
TIMERx	外设选择, x参考具体参数
输入参数{in}	
flag	状态标志
TIMER_FLAG_UP	更新标志, TIMERx (x=0..13)
TIMER_FLAG_CH0	通道0比较/捕获标志, TIMERx (x=0..4, 7..13)
TIMER_FLAG_CH1	通道1比较/捕获标志, TIMERx (x=0..4, 7, 8, 11)
TIMER_FLAG_CH2	通道2比较/捕获标志, TIMERx (x=0..4, 7)
TIMER_FLAG_CH3	通道3比较/捕获标志, TIMERx (x=0..4, 7)
TIMER_FLAG_CMT	通道换相更新标志, TIMERx (x=0, 7)
TIMER_FLAG_TRG	触发标志, TIMERx (x=0, 7, 8, 11)
TIMER_FLAG_BRK	中止标志位, TIMERx (x=0, 7)
TIMER_FLAG_CH0 O	通道0捕获溢出标志, TIMERx (x=0..4, 7..11)
TIMER_FLAG_CH1 O	通道1捕获溢出标志, TIMERx (x=0..4, 7, 8, 11)
TIMER_FLAG_CH2 O	通道2捕获溢出标志, TIMERx (x=0..4, 7)
TIMER_FLAG_CH3 O	通道3捕获溢出标志, TIMERx (x=0..4, 7)
输出参数{out}	
-	-
返回值	
FlagStatus	SET或RESET

例如:

```
/* get TIMER0 update flags */
```

```
FlagStatus Flag_status = RESET;
```

```
Flag_status = timer_flag_get(TIMERO0, TIMER_FLAG_UP);
```

函数 timer_flag_clear

函数timer_flag_clear描述见下表:

表 3-796. 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TIMER状态标志
先决条件	-
被调用函数	-
输入参数{in}	
timer_periph	TIMER外设
TIMERx	外设选择, x参考具体参数
输入参数{in}	
flag	状态标志
TIMER_FLAG_UP	更新标志, TIMERx (x=0..13)
TIMER_FLAG_CH0	通道0比较/捕获标志, TIMERx (x=0..4, 7..13)
TIMER_FLAG_CH1	通道1比较/捕获标志, TIMERx (x=0..4, 7, 8, 11)
TIMER_FLAG_CH2	通道2比较/捕获标志, TIMERx (x=0..4, 7)
TIMER_FLAG_CH3	通道3比较/捕获标志, TIMERx (x=0..4, 7)
TIMER_FLAG_CMT	通道换相更新标志, TIMERx (x=0, 7)
TIMER_FLAG_TRG	触发标志, TIMERx (x=0, 7, 8, 11)
TIMER_FLAG_BRK	中止标志位, TIMERx (x=0, 7)
TIMER_FLAG_CH0 O	通道0捕获溢出标志, TIMERx (x=0..4, 7..11)
TIMER_FLAG_CH1 O	通道1捕获溢出标志, TIMERx (x=0..4, 7, 8, 11)
TIMER_FLAG_CH2 O	通道2捕获溢出标志, TIMERx (x=0..4, 7)
TIMER_FLAG_CH3 O	通道3捕获溢出标志, TIMERx (x=0..4, 7)
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* clear TIMERO0 update flags */
```

```
timer_flag_clear(TIMERO0, TIMER_FLAG_UP);
```

函数 timer_interrupt_enable

函数timer_interrupt_enable描述见下表：

表 3-797. 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TIMER中断使能
先决条件	-
被调用函数	-
输入参数{in}	
timer_periph	TIMER外设
TIMERx	外设选择，x参考具体参数
输入参数{in}	
interrupt	中断源
TIMER_INT_UP	更新中断，TIMERx (x=0..13)
TIMER_INT_CH0	通道0比较/捕获中断，TIMERx (x=0..4, 7..13)
TIMER_INT_CH1	通道1比较/捕获中断，TIMERx (x=0..4, 7, 8, 11)
TIMER_INT_CH2	通道2比较/捕获中断，TIMERx (x=0..4, 7)
TIMER_INT_CH3	通道3比较/捕获中断，TIMERx (x=0..4, 7)
TIMER_INT_CMT	换相更新中断，TIMERx (x=0, 7)
TIMER_INT_TRG	触发中断，TIMERx (x=0..4, 7, 8, 11)
TIMER_INT_BRK	中止中断，TIMERx (x=0, 7)
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* enable the TIMER0 update interrupt */
timer_interrupt_enable(TIMER0, TIMER_INT_UP);
```

函数 timer_interrupt_disable

函数timer_interrupt_disable描述见下表：

表 3-798. 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TIMER中断禁能
先决条件	-
被调用函数	-
输入参数{in}	
timer_periph	TIMER外设

<i>TIMERx</i>	外设选择, x参考具体参数
输入参数{in}	
interrupt	中断源
<i>TIMER_INT_UP</i>	更新中断, <i>TIMERx</i> (x=0..13)
<i>TIMER_INT_CH0</i>	通道0比较/捕获中断, <i>TIMERx</i> (x=0..4, 7..13)
<i>TIMER_INT_CH1</i>	通道1比较/捕获中断, <i>TIMERx</i> (x=0..4, 7, 8, 11)
<i>TIMER_INT_CH2</i>	通道2比较/捕获中断, <i>TIMERx</i> (x=0..4, 7)
<i>TIMER_INT_CH3</i>	通道3比较/捕获中断, <i>TIMERx</i> (x=0..4, 7)
<i>TIMER_INT_CMT</i>	换相更新中断, <i>TIMERx</i> (x=0, 7)
<i>TIMER_INT_TRG</i>	触发中断, <i>TIMERx</i> (x=0..4, 7, 8, 11)
<i>TIMER_INT_BRK</i>	中止中断, <i>TIMERx</i> (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 3-799. 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TIMER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>	外设选择, x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> (x=0..13)
<i>TIMER_INT_FLAG_CH0</i>	通道0比较/捕获中断, <i>TIMERx</i> (x=0..4, 7..13)
<i>TIMER_INT_FLAG_CH1</i>	通道1比较/捕获中断, <i>TIMERx</i> (x=0..4, 7, 8, 11)
<i>TIMER_INT_FLAG_CH2</i>	通道2比较/捕获中断, <i>TIMERx</i> (x=0..4, 7)

<i>TIMER_INT_FLAG_</i> <i>CH3</i>	通道3比较/捕获中断, <i>TIMERx</i> (<i>x</i> =0..4, 7)
<i>TIMER_INT_FLAG_</i> <i>CMT</i>	换相更新中断, <i>TIMERx</i> (<i>x</i> =0, 7)
<i>TIMER_INT_FLAG_</i> <i>TRG</i>	触发中断, <i>TIMERx</i> (<i>x</i> =0..4, 7, 8, 11)
<i>TIMER_INT_FLAG_</i> <i>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 3-800. 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TIMER的中断标志
先决条件	-
被调用函数	-
输入参数{in}	
timer_periph	TIMER外设
<i>TIMERx</i>	外设选择, <i>x</i> 参考具体参数
输入参数{in}	
interrupt	中断源
<i>TIMER_INT_FLAG_</i> <i>UP</i>	更新中断, <i>TIMERx</i> (<i>x</i> =0..13)
<i>TIMER_INT_FLAG_</i> <i>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_</i> <i>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_</i> <i>CH2</i>	通道2比较/捕获中断, <i>TIMERx</i> (<i>x</i> =0..4, 7)
<i>TIMER_INT_FLAG_</i> <i>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, TIMERx (x=0, 7)
<i>TIMER_INT_FLAG_TRG</i>	触发中断, TIMERx (x=0..4, 7, 8, 11)
<i>TIMER_INT_FLAG_BRK</i>	中止中断, TIMERx (x=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);
```

3.26. TLI

TLI(TFT-LCD接口)连接同步的LCD接口, 并且为无源LCD显示屏提供提供像素数据, 时钟以及时序信号。TLI计算器列举在章节[3.26.1](#), TLI固件库函数介绍在章节[3.26.2](#)。

3.26.1. 外设寄存器说明

TLI寄存器列表如下表所示:

表 3-801. TLI 寄存器

寄存器名称	寄存器描述
TLI_SPSZ	TLI同步脉冲宽度寄存器
TLI_BPSZ	TLI后沿宽度寄存器
TLI_ASZ	TLI有效宽度寄存器
TLI_TSZ	TLI总宽度寄存器
TLI_CTL	TLI控制寄存器
TLI_RL	TLI重载配置寄存器
TLI_BGC	TLI背景色配置寄存器
TLI_INTEN	TLI中断使能寄存器
TLI_INTF	TLI中断标志寄存器
TLI_INTC	TLI中断标志清除寄存器
TLI_LM	TLI行标记寄存器
TLI_CPPOS	TLI当前像素寄存器
TLI_STAT	TLI状态寄存器
TLI_LxCTL	TLI x层控制寄存器
TLI_LxHPOS	TLI x层水平位置参数寄存器
TLI_LxVPOS	TLI x层垂直位置参数寄存器

寄存器名称	寄存器描述
TLI_LxCKEY	TLI x层色键值寄存器
TLI_LxPPF	TLI x层像素格式寄存器
TLI_LxSA	TLI x层特定alpha寄存器
TLI_LxDC	TLI x层默认颜色寄存器
TLI_LxBLEND	TLI x层混合寄存器
TLI_LxFBADDR	TLI x层帧基地址寄存器
TLI_LxFLEN	TLI x层行数寄存器
TLI_LxFTLN	TLI x层帧总行数寄存器
TLI_LxLUT	TLI x层颜色查找表寄存器

3.26.2. 外设库函数说明

TLI库函数列表如下表所示：

表 3-802. TLI 库函数

库函数名称	库函数描述
tli_deinit	复位TLI
tli_struct_para_init	用默认值初始化TLI参数结构体，建议在定义一个tli_parameter_struct结构体后调用该接口实现对结构体的初始化
tli_init	初始化TLI
tli_dither_config	配置TLI抖动功能
tli_enable	使能TLI
tli_disable	禁能TLI
tli_reload_config	配置TLI重载模式
tli_layer_struct_para_init	用默认参数初始化TLI层结构体，建议在定义一个tli_layer_parameter_struct结构体后调用该接口实现对结构体的初始化
tli_layer_init	初始化TLI层
tli_layer_window_offset_modify	重新配置窗口位置
tli_lut_struct_para_init	用默认参数初始化TLI层LUT结构体，建议在定义一个tli_layer_lut_parameter_struct结构体后调用该接口实现对结构体的初始化
tli_lut_init	初始化TLI层颜色查表
tli_color_key_init	初始化TLI层色键
tli_layer_enable	使能TLI层
tli_layer_disable	禁能TLI层
tli_color_key_enable	使能TLI层色键
tli_color_key_disable	禁能TLI层色键
tli_lut_enable	使能TLI层颜色查找
tli_lut_disable	禁能TLI层颜色查找
tli_line_mark_set	设置行标记值

库函数名称	库函数描述
tli_current_pos_get	获取当前像素位置
tli_interrupt_enable	使能TLI中断
tli_interrupt_disable	禁能TLI中断
tli_interrupt_flag_get	获取TLI中断标志
tli_interrupt_flag_clear	清除TLI中断标志
tli_flag_get	从TLI_INTF寄存器或TLI_STAT寄存器获取TLI标志或状态

结构体 tli_parameter_struct

表 3-803. 结构体 tli_parameter_struct

成员名称	功能描述
synpsz_vpsz	垂直同步脉冲宽度
synpsz_hpsz	水平同步脉冲宽度
backpsz_vbpsz	垂直后沿加同步脉冲的宽度
backpsz_hbpsz	水平后沿加同步脉冲的宽度
activesz_vasz	垂直有效宽度加后沿像素和水平同步像素宽度
activesz_hasz	水平有效宽度加后沿像素和垂直同步像素宽度
totalsz_vtsz	显示器的垂直总宽度
totalsz_htsz	显示器的水平总宽度
backcolor_red	背景色红色值
backcolor_green	背景色绿色值
backcolor_blue	背景色蓝色值
signalpolarity_hs	水平脉冲极性选择
signalpolarity_vs	垂直脉冲极性选择
signalpolarity_de	数据使能极性选择
signalpolarity_pixelck	像素时钟极性选择

结构体 tli_layer_parameter_struct

表 3-804. 结构体 tli_layer_parameter_struct

成员名称	功能描述
layer_window_rightpos	窗口右边位置
layer_window_leftpos	窗口左边位置
layer_window_bottompos	窗口下边位置
layer_window_toppos	窗口上边位置
layer_ppf	包像素格式
layer_sa	特定alpha
layer_default_alpha	默认颜色alpha
layer_default_red	默认红色值
layer_default_green	默认绿色值
layer_default_blue	默认蓝色值
layer_acf1	Alpha混合计算因子1

layer_acf2	Alpha混合计算因子2
layer_frame_bufaddr	帧缓存区起始地址
layer_frame_buf_stride_offset	帧缓存区步幅偏移
layer_frame_line_length	帧行长度
layer_frame_total_line_number	帧总行数

结构体 tli_layer_lut_parameter_struct

表 3-805. 结构体 tli_layer_lut_parameter_struct

成员名称	功能描述
layer_table_addr	查表写地址
layer_lut_channel_red	颜色查找表条目红色通道
layer_lut_channel_green	颜色查找表条目绿色通道
layer_lut_channel_blue	颜色查找表条目蓝色通道

枚举类型 tli_layer_ppf_enum

表 3-806. 枚举类型 tli_layer_ppf_enum

成员名称	功能描述
LAYER_PPF_ARGB8888	X层包像素格式ARGB8888
LAYER_PPF_RGB888	X层包像素格式RGB888
LAYER_PPF_RGB565	X层包像素格式RGB565
LAYER_PPF_ARGB1555	X层包像素格式ARGB1555
LAYER_PPF_ARGB4444	X层包像素格式ARGB4444
LAYER_PPF_L8	X层包像素格式L8
LAYER_PPF_AL44	X层包像素格式AL44
LAYER_PPF_AL88	X层包像素格式AL88

函数 tli_deinit

函数tli_deinit描述见下表：

表 3-807. 函数 tli_deinit

函数名称	tli_deinit
函数原形	void tli_deinit(void);
功能描述	复位TLI
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：

```
/* deinitialize TLI registers */
```

```
tli_deinit();
```

函数 tli_struct_para_init

函数tli_struct_para_init描述见下表：

表 3-808. 函数 tli_struct_para_init

函数名称	tli_struct_para_init
函数原形	void tli_struct_para_init(tli_parameter_struct *tli_struct);
功能描述	用默认值初始化 TLI 参数结构体，建议在定义一个 tli_parameter_struct 结构体后调用该接口实现对结构体的初始化
先决条件	-
被调用函数	-
输入参数{in}	
-	-
输出参数{out}	
*tli_struct	指向 TLI 参数结构体的指针
返回值	
-	-

例如：

```
tli_parameter_struct tli_init_struct;
```

```
/* initialize the parameters of TLI parameter structure with the default values */
```

```
tli_struct_para_init(&tli_init_struct);
```

函数 tli_init

函数tli_init描述见下表：

表 3-809. 函数 tli_init

函数名称	tli_init
函数原形	void tli_init(tli_parameter_struct *tli_struct);
功能描述	初始化TLI
先决条件	-
被调用函数	-
输入参数{in}	
tli_struct	TLI参数结构体
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* initialize TLI display timing parameters */

tli_parameter_struct tli_init_struct;

/* TLI initialization */

tli_init_struct.signalpolarity_hs = TLI_HSYN_ACTLIVE_LOW;
tli_init_struct.signalpolarity_vs = TLI_VSYN_ACTLIVE_LOW;
tli_init_struct.signalpolarity_de = TLI_DE_ACTLIVE_LOW;
tli_init_struct.signalpolarity_pixelck = TLI_PIXEL_CLOCK_TLI;

/* LCD display timing configuration */

tli_init_struct.synpsz_hpsz = 40;
tli_init_struct.synpsz_vpsz = 9;
tli_init_struct.backpsz_hbpsz = 42;
tli_init_struct.backpsz_vbpsz = 11;
tli_init_struct.activesz_hasz = 42 + 480;
tli_init_struct.activesz_vasz = 11 + 272;
tli_init_struct.totalsz_htsz = 42 + 480 + 2;
tli_init_struct.totalsz_vtsz = 11 + 272 + 2;

/* LCD background color configure */

tli_init_struct.backcolor_red = 0xFF;
tli_init_struct.backcolor_green = 0xFF;
tli_init_struct.backcolor_blue = 0xFF;

tli_init(&tli_init_struct);
```

函数 tli_dither_config

函数tli_dither_config描述见下表:

表 3-810. 函数 tli_dither_config

函数名称	tli_dither_config
函数原形	void tli_dither_config(uint8_t ditherstat);
功能描述	配置TLI抖动功能
先决条件	-
被调用函数	-
输入参数{in}	

ditherstat	抖动状态
TLI_DITHER_ENABLE	使能TLI抖动
TLI_DITHER_DISABLE	除能TLI抖动
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* configure TLI dither function */
tli_dither_config(TLI_DITHER_ENABLE);
```

函数 tli_enable

函数tli_enable描述见下表：

表 3-811. 函数 tli_enable

函数名称	tli_enable
函数原形	void tli_enable(void);
功能描述	使能TLI
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 TLI */
tli_enable( );
```

函数 tli_disable

函数tli_disable描述见下表：

表 3-812. 函数 tli_disable

函数名称	tli_disable
函数原形	void tli_disable(void);
功能描述	除能TLI
先决条件	-
被调用函数	-
输入参数{in}	

-	-
Output parameter{out}	
-	-
返回值	
-	-

例如：

```
/* disable TLI */
```

```
tli_disable( );
```

函数 tli_reload_config

函数tli_reload_config描述见下表：

表 3-813. 函数 tli_reload_config

函数名称	tli_reload_config
函数原形	void tli_reload_config(uint8_t reloadmode);
功能描述	configure TLI reload mode
先决条件	-
被调用函数	-
输入参数{in}	
reloadmode	Reload mode
TLI_FRAME_BLANK_RELOAD_EN	层配置在帧间隔被重载
TLI_REQUEST_RELOAD_EN	层配置在置位后被重载
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* configure TLI reload mode */
```

```
tli_reload_config(TLI_FRAME_BLANK_RELOAD_EN);
```

函数 tli_layer_struct_para_init

函数tli_layer_struct_para_init描述见下表：

表 3-814. 函数 tli_layer_struct_para_init

函数名称	tli_layer_struct_para_init
函数原形	void tli_layer_struct_para_init(tli_layer_parameter_struct *layer_struct);
功能描述	用默认参数初始化 TLI 层结构体，建议在定义一个 tli_layer_parameter_struct 结构体后调用该接口实现对结构体的初始化

先决条件	-
被调用函数	-
输入参数{in}	
-	-
输出参数{out}	
*layer_struct	指向 tli_layer_parameter_struct 结构体的指针
返回值	
-	-

例如：

```
tli_layer_parameter_struct tli_layer_init_struct;
```

```
/* initialize the parameters of TLI layer structure with the default values */
```

```
tli_layer_struct_para_init(&tli_layer_init_struct);
```

函数 tli_layer_init

函数tli_layer_init描述见下表：

表 3-815. 函数 tli_layer_init

函数名称	tli_layer_init
函数原形	void tli_layer_init(uint32_t layerx,tli_layer_parameter_struct *layer_struct)
功能描述	初始化TLI层
先决条件	-
被调用函数	-
输入参数{in}	
layerx	层基地址
LAYER0	0层基地址
LAYER1	1层基地址
输入参数{in}	
layer_struct	TLI Layer parameter struct
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* TLI layer initialize */
```

```
tli_layer_parameter_struct tli_layer_init_struct;
```

```
/* tli layer1 configuration */
```

```
/* tli window size configuration */
```

```
tli_layer_init_struct.layer_window_leftpos = 20 + 43;
```

```

tli_layer_init_struct.layer_window_rightpos = (20 + 140 + 43 - 1);

tli_layer_init_struct.layer_window_toppos = 30 + 12;

tli_layer_init_struct.layer_window_bottompos = (30 + 60 + 12 - 1);

/* tli window pixel format configuration */

tli_layer_init_struct.layer_ppf = LAYER_PPF_RGB565;

/* tli window specified alpha configuration */

tli_layer_init_struct.layer_sa = 255;

/* tli window blend configuration */

tli_layer_init_struct.layer_acf1 = LAYER_ACF1_PASA;

tli_layer_init_struct.layer_acf2 = LAYER_ACF2_PASA;

/* tli layer default alpha R,G,B value configuration */

tli_layer_init_struct.layer_default_alpha = 0;

tli_layer_init_struct.layer_default_blue = 0xFF;

tli_layer_init_struct.layer_default_green = 0xFF;

tli_layer_init_struct.layer_default_red = 0xFF;

/* tli layer frame buffer base address configuration */

tli_layer_init_struct.layer_frame_bufaddr = (uint32_t)&image_0;

tli_layer_init_struct.layer_frame_line_length = ((480 * 2) + 3);

tli_layer_init_struct.layer_frame_buf_stride_offset = (480 * 2);

tli_layer_init_struct.layer_frame_total_line_number = 60;

tli_layer_init(LAYER1, &tli_layer_init_struct);

```

函数 tli_layer_window_offset_modify

函数tli_layer_window_offset_modify描述见下表:

表 3-816. 函数 tli_layer_window_offset_modify

函数名称	tli_layer_window_offset_modify
函数原形	void tli_layer_window_offset_modify(uint32_t layerx,uint32_t offset_x,uint32_t offset_y)
功能描述	重新配置窗口位置
先决条件	-
被调用函数	-
输入参数{in}	

layerx	层基地址
<i>LAYER0</i>	0层基地址
<i>LAYER1</i>	1层基地址
输入参数{in}	
offset_x	新水平偏移
输入参数{in}	
offset_y	新垂直偏移
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* TLI layer initialize */
```

```
tli_layer_window_offset_modify(LAYER0, 0x40, 0x40);
```

函数 tli_lut_struct_para_init

函数tli_lut_struct_para_init描述见下表：

表 3-817. 函数 tli_lut_struct_para_init

函数名称	tli_lut_struct_para_init
函数原形	void tli_lut_struct_para_init(tli_layer_lut_parameter_struct *lut_struct);
功能描述	用默认参数初始化 TLI 层 LUT 结构体，建议在定义一个 tli_layer_lut_parameter_struct 结构体后调用该接口实现对结构体的初始化
先决条件	-
被调用函数	-
输入参数{in}	
-	-
输出参数{out}	
*lut_struct	指向 tli_layer_lut_parameter_struct 结构体的指针
返回值	
-	-

例如：

```
tli_layer_lut_parameter_struct tli_lut_struct;
```

```
/* initialize the parameters of TLI layer LUT structure with the default values */
```

```
tli_lut_struct_para_init(&tli_lut_struct);
```

函数 tli_lut_init

函数tli_lut_init描述见下表：

表 3-818. 函数 tli_lut_init

函数名称	tli_lut_init
函数原形	void tli_lut_init(uint32_t layerx,tli_layer_lut_parameter_struct *lut_struct)
功能描述	初始化TLI层颜色查表
先决条件	-
被调用函数	-
输入参数{in}	
layerx	层基地址
LAYER0	0层基地址
LAYER1	1层基地址
输入参数{in}	
*lut_struct	TLI层LUT参数结构体
输出参数{out}	
-	-
返回值	
-	-

例如：

```

/* TLI layer initialize */

tli_layer_lut_parameter_struct lut_struct;

lut_struct.layer_table_addr = 0x20000400;

lut_struct.layer_lut_channel_red = 0x20;

lut_struct.layer_lut_channel_green = 0x40;

lut_struct.layer_lut_channel_blue = 0x40;

tli_layer_init(LAYER0, &lut_struct);

```

函数 tli_color_key_init

函数tli_color_key_init描述见下表：

表 3-819. 函数 tli_color_key_init

函数名称	tli_color_key_init
函数原形	void tli_color_key_init(uint32_t layerx,uint32_t redkey,uint32_t greenkey,uint32_t bluekey)
功能描述	初始化TLI层色键
先决条件	-
被调用函数	-
输入参数{in}	
layerx	层基地址
LAYER0	0层基地址
LAYER1	1层基地址

输入参数{in}	
redkey	红色键
输入参数{in}	
greenkey	绿色键
输入参数{in}	
bluekey	蓝色键
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* TLI layer color key initialize */
tli_color_key_init(LAYER0, 0xAA, 0xFF, 0x00);
```

函数 tli_layer_enable

函数tli_layer_enable描述见下表:

表 3-820. 函数 tli_layer_enable

函数名称	tli_layer_enable
函数原形	void tli_layer_enable(uint32_t layerx);
功能描述	使能TLI层
先决条件	-
被调用函数	-
输入参数{in}	
layerx	层基地址
LAYER0	0层基地址
LAYER1	1层基地址
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* TLI layer enable */
tli_layer_enable(LAYER0);
```

函数 tli_layer_disable

函数tli_layer_disable描述见下表:

表 3-821. 函数 tli_layer_disable

函数名称	tli_layer_disable
------	-------------------

函数原形	void tli_layer_disable(uint32_t layerx);
功能描述	除能TLI层
先决条件	-
被调用函数	-
输入参数{in}	
layerx	层基地址
<i>LAYER0</i>	0层基地址
<i>LAYER1</i>	1层基地址
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* TLI layer disable */
```

```
tli_layer_disable(LAYER0);
```

函数 tli_color_key_enable

函数tli_color_key_enable描述见下表：

表 3-822. 函数 tli_color_key_enable

函数名称	tli_color_key_enable
函数原形	void tli_color_key_enable(uint32_t layerx);
功能描述	使能TLI层色键
先决条件	-
被调用函数	-
输入参数{in}	
layerx	层基地址
<i>LAYER0</i>	0层基地址
<i>LAYER1</i>	1层基地址
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* TLI layer color keying enable */
```

```
tli_color_key_enable(LAYER0);
```

函数 tli_color_key_disable

函数tli_color_key_disable描述见下表：

表 3-823. 函数 tli_color_key_disable

函数名称	tli_color_key_disable
函数原形	void tli_color_key_disable(uint32_t layerx);
功能描述	除能TLI层色键
先决条件	-
被调用函数	-
输入参数{in}	
layerx	层基地址
LAYER0	0层基地址
LAYER1	1层基地址
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* TLI layer color keying disable */
```

```
tli_color_key_disable(LAYER0);
```

函数 tli_lut_enable

函数tli_lut_enable描述见下表:

表 3-824. 函数 tli_lut_enable

函数名称	tli_lut_enable
函数原形	void tli_lut_enable(uint32_t layerx);
功能描述	使能TLI层颜色查找
先决条件	-
被调用函数	-
输入参数{in}	
layerx	层基地址
LAYER0	0层基地址
LAYER1	1层基地址
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* TLI layer LUT enable */
```

```
tli_lut_enable(LAYER0);
```

函数 tli_lut_disable

函数tli_lut_disable描述见下表：

表 3-825. 函数 tli_lut_disable

函数名称	tli_lut_disable
函数原形	void tli_lut_disable(uint32_t layerx);
功能描述	除能TLI层颜色查找
先决条件	-
被调用函数	-
输入参数{in}	
layerx	层基地址
LAYER0	0层基地址
LAYER1	1层基地址
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* TLI layer LUT disable */
```

```
tli_lut_disable(LAYER0);
```

函数 tli_line_mark_set

函数tli_line_mark_set描述见下表：

表 3-826. 函数 tli_line_mark_set

函数名称	tli_line_mark_set
函数原形	void tli_line_mark_set(uint32_t line_num)
功能描述	设置行标记值
先决条件	-
被调用函数	-
输入参数{in}	
line_num	行编号
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* set line mark value */
```

```
tli_line_mark_set( 0x40);
```

函数 tli_current_pos_get

函数tli_current_pos_get描述见下表：

表 3-827. 函数 tli_current_pos_get

函数名称	tli_current_pos_get
函数原形	uint32_t tli_current_pos_get(void);
功能描述	获取当前像素位置
先决条件	-
被调用函数	-
输入参数{in}	
-	-
输出参数{out}	
-	-
返回值	
uint32_t	0x0 – 0xFFFFFFFF

例如：

```
/* get current displayed position */
uint32_t pos = tli_current_pos_get( );
```

函数 tli_interrupt_enable

函数tli_interrupt_enable描述见下表：

表 3-828. 函数 tli_interrupt_enable

函数名称	tli_interrupt_enable
函数原形	void tli_interrupt_enable(uint32_t int_flag)
功能描述	使能TLI中断
先决条件	-
被调用函数	-
输入参数{in}	
int_flag	TLI中断标志
TLI_INT_LM	行标记中断
TLI_INT_FE	FIFO错误中断
TLI_INT_TE:	事务错误中断
TLI_INT_LCR	层配置重载中断
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* enable TLI interrupt */
```

tli_interrupt_enable (TLI_INT_LM);

函数 tli_interrupt_disable

函数tli_interrupt_disable描述见下表：

表 3-829. 函数 tli_interrupt_disable

函数名称	tli_interrupt_disable
函数原形	void tli_interrupt_disable(uint32_t int_flag)
功能描述	除能TLI中断
先决条件	-
被调用函数	-
输入参数{in}	
int_flag	TLI中断标志
TLI_INT_LM	行标记中断
TLI_INT_FE	FIFO错误中断
TLI_INT_TE:	事务错误中断
TLI_INT_LCR	层配置重载中断
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* disable TLI interrupt */
```

```
tli_interrupt_disable(TLI_INT_LM);
```

函数 tli_interrupt_flag_get

函数tli_interrupt_flag_get描述见下表：

表 3-830. 函数 tli_interrupt_flag_get

函数名称	tli_interrupt_flag_get
函数原形	FlagStatus tli_interrupt_flag_get(uint32_t int_flag);
功能描述	获取TLI中断标志
先决条件	-
被调用函数	-
输入参数{in}	
int_flag	TLI中断标志
TLI_INT_FLAG_LM	行标记中断
TLI_INT_FLAG_FE	FIFO错误中断
TLI_INT_FLAG_TE	事务错误中断
TLI_INT_FLAG_LCR:	层配置重载中断
输出参数{out}	

-	-
返回值	
FlagStatus	SET或RESET

例如:

```
/* get TLI flag state */
```

```
FlagStatus status;
```

```
status = tli_interrupt_flag_get(TLI_INT_FLAG_LM);
```

函数 tli_interrupt_flag_clear

函数tli_interrupt_flag_clear描述见下表:

表 3-831. 函数 tli_interrupt_flag_clear

函数名称	tli_interrupt_flag_clear
函数原形	void tli_interrupt_flag_clear(uint32_t int_flag)
功能描述	清除TLI中断标志
先决条件	-
被调用函数	-
输入参数{in}	
int_flag	TLI中断标志
TLI_INT_FLAG_LM	行标记中断
TLI_INT_FLAG_FE	FIFO错误中断
TLI_INT_FLAG_TE	事务错误中断
TLI_INT_FLAG_LCR:	层配置重载中断
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* clear TLI flag state */
```

```
tli_interrupt_flag_clear(TLI_INT_FLAG_LM);
```

函数 tli_flag_get

函数tli_flag_get描述见下表:

表 3-832. 函数 tli_flag_get

函数名称	tli_flag_get
函数原形	FlagStatus tli_flag_get(uint32_t flag);
功能描述	从TLI_INTF寄存器或TLI_STAT寄存器获取TLI标志或状态
先决条件	-

被调用函数	-
输入参数{in}	
flag	TLI flags
<i>TLI_FLAG_VDE</i>	当前VDE状态
<i>TLI_FLAG_HDE</i>	当前HDE状态
<i>TLI_FLAG_VS</i>	当前vs状态
<i>TLI_FLAG_HS</i>	当前hs状态
<i>TLI_FLAG_LM</i>	行标记中断标志
<i>TLI_FLAG_FE</i>	FIFO错误中断标志
<i>TLI_FLAG_TE</i>	事务错误中断标志
<i>TLI_FLAG_LCR</i>	层配置重载中断标志
输出参数{out}	
-	-
返回值	
FlagStatus	SET或RESET

例如:

```
/* get TLI flag state */
FlagStatus status;

status = tli_flag_get (TLI_FLAG_VDE);
```

3.27. TRNG

真随机数发生器模块（TRNG）能够通过连续模拟噪声生成一个32位的随机数值。TRNG 寄存器列举在章节[3.27.1](#)，TRNG固件库函数介绍在章节[3.27.2](#)。

3.27.1. 外设寄存器说明

TRNG寄存器列表如下表所示:

表 3-833. TRNG 寄存器

寄存器名称	寄存器描述
TRNG_CTL	控制寄存器
TRNG_STAT	状态寄存器
TRNG_DATA	数据寄存器

3.27.2. 外设库函数说明

TRNG库函数列表如下表所示:

表 3-834. TRNG 库函数

库函数名称	库函数描述
trng_deinit	复位 TRNG
trng_enable	使能 TRNG
trng_disable	除能 TRNG
trng_get_true_random_data	获取真随机值
trng_flag_get	获取 TRNG 状态标志
trng_interrupt_enable	使能 TRNG 中断
trng_interrupt_disable	除能 TRNG 中断
trng_interrupt_flag_get	获取 TRNG 中断标志
trng_interrupt_flag_clear	清除 TRNG 中断标志

枚举类型 trng_flag_enum

表 3-835. 枚举类型 trng_flag_enum

成员名称	功能描述
TRNG_FLAG_DRDY	随机数据就绪状态
TRNG_FLAG_CECS	时钟错误目前状态
TRNG_FLAG_SECS	种子错误目前状态

枚举类型 trng_int_flag_enum

表 3-836. 枚举类型 trng_int_flag_enum

成员名称	功能描述
TRNG_INT_FLAG_CE	时钟错误中断标志
TRNG_INT_FLAG_SE	种子错误中断标志

函数 trng_deinit

函数trng_deinit描述见下表：

表 3-837. 函数 trng_deinit

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

例如：

```
/* TRNG deinit */
```

```
trng_deinit();
```

函数 trng_enable

函数trng_enable描述见下表:

表 3-838. 函数 trng_enable

函数名称	trng_enable
函数原形	void trng_enable(void);
功能描述	使能 TRNG
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 TRNG interface */
```

```
trng_enable();
```

函数 trng_disable

函数trng_disable描述见下表:

表 3-839. 函数 trng_disable

函数名称	trng_disable
函数原形	void trng_disable(void);
功能描述	除能 TRNG
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 TRNG interface */
```

```
trng_disable();
```

函数 trng_get_true_random_data

函数trng_get_true_random_data描述见下表:

表 3-840. 函数 trng_get_true_random_data

函数名称	trng_get_true_random_data
函数原形	uint32_t trng_get_true_random_data(void);
功能描述	获取真随机值
先决条件	-
被调用函数	-
输入参数{in}	
-	-
输出参数{out}	
-	-
返回值	
uint32_t	0x0 – 0xFFFFFFFF

例如:

```
/* get the true random data */
```

```
uint32_t data = trng_get_true_random_data();
```

函数 trng_flag_get

函数trng_flag_get描述见下表:

表 3-841. 函数 trng_flag_get

函数名称	trng_flag_get
函数原形	FlagStatus trng_flag_get(trng_flag_enum flag);
功能描述	获取 TRNG 状态标志
先决条件	-
被调用函数	-
输入参数{in}	
flag	TRNG 状态标志, 参阅 表 3-835. 枚举类型 trng_flag_enum
TRNG_FLAG_DRDY	随机数据就绪状态
TRNG_FLAG_CECS	时钟错误目前状态
TRNG_FLAG_SECS	种子错误目前状态
输出参数{out}	
-	-
返回值	
FlagStatus	SET 或 RESET

例如:

```

/* get the trng status flags */

FlagStatus trng_flag = RESET;

trng_flag = trng_flag_get(TRNG_FLAG_DRDY);

```

函数 trng_interrupt_enable

函数trng_interrupt_enable描述见下表：

表 3-842. 函数 trng_interrupt_enable

函数名称	trng_interrupt_enable
函数原形	void trng_interrupt_enable(void);
功能描述	使能 TRNG 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 the TRNG interrupt */

trng_interrupt_enable();

```

函数 trng_interrupt_disable

函数trng_interrupt_disable描述见下表：

表 3-843. 函数 trng_interrupt_disable

函数名称	trng_interrupt_disable
函数原形	void trng_interrupt_disable(void);
功能描述	除能 TRNG 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：

```
/* disable the TRNG interrupt */
```

```
trng_interrupt_disable();
```

函数 trng_interrupt_flag_get

函数trng_interrupt_flag_get描述见下表：

表 3-844. 函数 trng_interrupt_flag_get

函数名称	trng_interrupt_flag_get
函数原形	FlagStatus trng_interrupt_flag_get(trng_int_flag_enum int_flag);
功能描述	获取 TRNG 中断标志
先决条件	-
被调用函数	-
输入参数{in}	
int_flag	TRNG 中断标志,参阅 表 3-836. 枚举类型 trng_int_flag_enum
TRNG_INT_FLAG_CE	时钟错误中断标志
TRNG_INT_FLAG_SE	种子错误中断标志
输出参数{out}	
-	-
返回值	
FlagStatus	SET 或 RESET

例如：

```
/* get the trng interrupt flag */
```

```
FlagStatus trng_flag = RESET;
```

```
trng_flag = trng_interrupt_flag_get(TRNG_INT_FLAG_CE);
```

函数 trng_interrupt_flag_clear

函数trng_interrupt_flag_clear描述见下表：

表 3-845. 函数 trng_interrupt_flag_clear

函数名称	trng_interrupt_flag_clear
函数原形	void trng_interrupt_flag_clear(trng_int_flag_enum int_flag);
功能描述	清除 TRNG 中断标志
先决条件	-
被调用函数	-
输入参数{in}	
int_flag	TRNG 中断标志,参阅 表 3-836. 枚举类型 trng_int_flag_enum
TRNG_INT_FLAG_CE	时钟错误中断标志
TRNG_INT_FLAG_SE	种子错误中断标志
输出参数{out}	
-	-

返回值	
-	-

例如:

```
/*clear the trng interrupt flag */
```

```
trng_interrupt_flag_clear(TRNG_INT_FLAG_CE);
```

3.28. 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[3.28.1](#)描述了USART的寄存器列表, 章节[3.28.2](#)对USART库函数进行说明。

3.28.1. 外设寄存器说明

USART寄存器列表如下表所示:

表 3-846. USART 寄存器

寄存器名称	寄存器描述
USART_STAT0	状态寄存器0
USART_DATA	数据寄存器
USART_BAUD	波特率寄存器
USART_CTL0	控制寄存器0
USART_CTL1	控制寄存器1
USART_CTL2	控制寄存器2
USART_GP	保护时间和预分频器寄存器
USART_CTL3	控制寄存器3
USART_RT	接收超时寄存器
USART_STAT1	状态寄存器1

3.28.2. 外设库函数说明

USART库函数列表如下表所示:

表 3-847. USART 库函数

库函数名称	库函数描述
usart_deinit	复位外设USART/UART
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	禁能USART
usart_transmit_config	配置USART发送器

库函数名称	库函数描述
usart_receive_config	配置USART接收器
usart_data_first_config	数据发送/接收，采用低位/高位在前
usart_invert_config	配置USART反转
usart_receiver_timeout_enable	接收器超时使能
usart_receiver_timeout_disable	接收器超时禁能
usart_receiver_timeout_threshold_config	设置接收器超时阈值
usart_data_transmit	USART发送数据功能
usart_data_receive	USART接收数据功能
usart_address_config	配置在地址匹配唤醒模式下USART地址
usart_mute_mode_enable	使能静默模式
usart_mute_mode_disable	禁能静默模式
usart_mute_mode_wakeup_config	配置静默模式唤醒方式
usart_lin_mode_enable	使能LIN模式
usart_lin_mode_disable	禁能LIN模式
usart_lin_break_dection_length_config	配置LIN模式中断帧长度
usart_send_break	发送断开帧
usart_halfduplex_enable	使能半双工模式
usart_halfduplex_disable	禁能半双工模式
usart_synchronous_clock_enable	在同步通讯模式下使能CK引脚
usart_synchronous_clock_disable	在同步通讯模式下禁能CK引脚
usart_synchronous_clock_config	配置USART同步通讯模式参数
usart_guard_time_config	在智能卡模式下配置保护时间值
usart_smartcard_mode_enable	使能智能卡模式
usart_smartcard_mode_disable	禁能智能卡模式
usart_smartcard_mode_nack_enable	在智能卡模式下使能NACK
usart_smartcard_mode_nack_disable	在智能卡模式下禁能NACK
usart_smartcard_autoretry_config	配置智能卡自动重试次数
usart_block_length_config	配置块长度
usart_irda_mode_enable	使能串行红外编解码功能模块
usart_irda_mode_disable	禁能串行红外编解码功能模块
usart_prescaler_config	配置外设时钟分频系数
usart_irda_lowpower_config	配置IrDA低功耗模式
usart_hardware_flow_rts_config	配置RTS硬件控制流
usart_hardware_flow_cts_config	配置CTS硬件控制流
usart_dma_receive_config	配置USART DMA接收
usart_dma_transmit_config	配置USART DMA发送
usart_flag_get	获取STAT0/STAT1标志位
usart_flag_clear	清除STAT0/STAT1标志位
usart_interrupt_enable	使能USART中断

库函数名称	库函数描述
usart_interrupt_disable	禁能USART中断
usart_interrupt_flag_get	获取USART中断标志位状态
usart_interrupt_flag_clear	清除USART中断标志位状态

枚举类型 usart_flag_enum

表 3-848. 枚举类型 usart_flag_enum

成员名称	功能描述
USART_FLAG_CTS	CTS变化标志
USART_FLAG_LBD	LIN断开检测标志
USART_FLAG_TBE	发送数据寄存器空
USART_FLAG_TC	发送完成
USART_FLAG_RBNE	读数据缓冲区非空
USART_FLAG_IDLE	空闲帧检测标志
USART_FLAG_ORERR	溢出错误标志
USART_FLAG_NERR	噪声错误标志
USART_FLAG_FERR	帧错误标志
USART_FLAG_PERR	校验错误标志
USART_FLAG_BSY	忙标志
USART_FLAG_EB	块结束标志
USART_FLAG_RT	接收超时标志

枚举类型 usart_interrupt_flag_enum

表 3-849. 枚举类型 usart_interrupt_flag_enum

成员名称	功能描述
USART_INT_FLAG_PERR	奇偶校验错误中断标志
USART_INT_FLAG_TBE	发送寄存器空中断标志
USART_INT_FLAG_TC	发送完成中断标志
USART_INT_FLAG_RBNE	读缓冲区非空中断标志
USART_INT_FLAG_RBNE_ORERR	读缓冲区非空和溢出中断标志
USART_INT_FLAG_IDLE	空闲帧检测中断标志
USART_INT_FLAG_LBD	LIN断开检测中断标志
USART_INT_FLAG_CTS	CTS中断标志
USART_INT_FLAG_ERR_ORERR	溢出错误中断标志
USART_INT_FLAG_ERR_NERR	噪声错误中断标志
USART_INT_FLAG_ERR_FERR	帧错误中断标志
USART_INT_FLAG_EB	块结束中断标志
USART_INT_FLAG_RT	接收超时中断标志

枚举类型 `usart_interrupt_enum`

表 3-850. 枚举类型 `usart_interrupt_enum`

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

枚举类型 `usart_invert_enum`

表 3-851. 枚举类型 `usart_invert_enum`

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

函数 `usart_deinit`

函数`usart_deinit`描述见下表:

表 3-852. 函数 `usart_deinit`

函数名称	<code>usart_deinit</code>
函数原型	<code>void usart_deinit(uint32_t usart_periph);</code>
功能描述	复位外设USART/UART
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}	
<code>usart_periph</code>	外设USARTx/UARTx
<code>USARTx</code>	x=0,1,2,5
<code>UARTx</code>	x=3,4,6,7
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 3-853. 函数 usart_baudrate_set

函数名称	usart_baudrate_set
函数原型	void usart_baudrate_set(uint32_t usart_periph, uint32_t baudval);
功能描述	配置USART波特率
先决条件	-
被调用函数	rcu_clock_freq_get
输入参数{in}	
usart_periph	外设USARTx/UARTx
USARTx	x=0,1,2,5
UARTx	x=3,4,6,7
输入参数{in}	
baudval	波特率值
输出参数{out}	
-	-
返回值	
-	-

例如:

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

函数 usart_parity_config

函数usart_parity_config描述见下表:

表 3-854. 函数 usart_parity_config

函数名称	usart_parity_config
函数原型	void usart_parity_config(uint32_t usart_periph, uint32_t paritycfg);
功能描述	配置USART奇偶校验
先决条件	-
被调用函数	-
输入参数{in}	
usart_periph	外设USARTx/UARTx
USARTx	x=0,1,2,5
UARTx	x=3,4,6,7

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 3-855. 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
<i>USARTx</i>	x=0,1,2,5
<i>UARTx</i>	x=3,4,6,7
输入参数{in}	
wlen	配置USART字长
<i>USART_WL_8BIT</i>	8位
<i>USART_WL_9BIT</i>	9位
输出参数{out}	
-	-
返回值	
-	-

例如：

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

函数 usart_stop_bit_set

函数usart_stop_bit_set描述见下表:

表 3-856. 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=0,1,2,5
UARTx	x=3,4,6,7
输入参数{in}	
stblen	配置USART停止位
USART_STB_1BIT	1位
USART_STB_0_5BIT	0.5位, 该位对UARTx(x=3,4,6,7)无效
USART_STB_2BIT	2位
USART_STB_1_5BIT	1.5位, 该位对UARTx(x=3,4,6,7)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 3-857. 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=0,1,2,5

<i>USARTx</i>	x=3,4,6,7
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 3-858. 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	禁能USART
先决条件	-
被调用函数	-
输入参数{in}	
usart_periph	外设USARTx/UARTx
<i>USARTx</i>	x=0,1,2,5
<i>UARTx</i>	x=3,4,6,7
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 3-859. 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	配置USART发送器
先决条件	-
被调用函数	-
输入参数{in}	

usart_periph	外设USARTx/UARTx
<i>USARTx</i>	x=0,1,2,5
<i>UARTx</i>	x=3,4,6,7
输入参数{in}	
txconfig	使能/禁能USART发送器
<i>USART_TRANSMIT_ENABLE</i>	使能USART发送
<i>USART_TRANSMIT_DISABLE</i>	禁能USART发送
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* configure USART0 transmitter */
```

```
usart_transmit_config(USART0,USART_TRANSMIT_ENABLE);
```

函数 usart_receive_config

函数usart_receive_config描述见下表：

表 3-860. 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	配置USART接收器
先决条件	-
被调用函数	-
输入参数{in}	
usart_periph	外设USARTx/UARTx
<i>USARTx</i>	x=0,1,2,5
<i>UARTx</i>	x=3,4,6,7
输入参数{in}	
rxconfig	使能/禁能USART接收器
<i>USART_RECEIVE_ENABLE</i>	使能USART接收
<i>USART_RECEIVE_DISABLE</i>	禁能USART接收
输出参数{out}	
-	-
返回值	
-	-

例如：


```
/* configure USART0 receiver */
```

```
usart_receive_config(USART0, USART_RECEIVE_ENABLE);
```

函数 usart_data_first_config

函数usart_data_first_config描述见下表：

表 3-861. 函数 usart_data_first_config

函数名称	usart_data_first_config
函数原型	void usart_data_first_config(uint32_t usart_periph, uint32_t msbf);
功能描述	数据发送/接收，采用低位/高位在前
先决条件	-
被调用函数	-
输入参数{in}	
usart_periph	外设USARTx
USARTx	x=0,1,2,5
输入参数{in}	
msbf	低位/高位
USART_MSBF_LSB	低位在前
USART_MSBF_MSB	高位在前
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* data is transmitted/received with the LSB first */
```

```
usart_data_first_config(USART_MSBF_LSB);
```

函数 usart_invert_config

函数usart_invert_config描述见下表：

表 3-862. 函数 usart_invert_config

函数名称	usart_invert_config
函数原型	void usart_invert_config(uint32_t usart_periph, usart_invert_enum invertpara);
功能描述	配置USART反转
先决条件	-
被调用函数	-
输入参数{in}	
usart_periph	外设USARTx
USARTx	x=0,1,2,5

输入参数{in}	
invertpara	反转参数，参考 表3-851. 枚举类型usart invert enum 。
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* configure USART0 inversion */
usart_invert_config(USART0, USART_DINV_ENABLE);
```

函数 usart_receiver_timeout_enable

函数usart_receiver_timeout_enable描述见下表：

表 3-863. 函数 usart_receiver_timeout_enable

函数名称	usart_receiver_timeout_enable
函数原型	void usart_receiver_timeout_enable(uint32_t usart_periph);
功能描述	接收器超时使能
先决条件	-
被调用函数	-
输入参数{in}	
usart_periph	外设USARTx
USARTx	x=0,1,2,5
输出参数{out}	
-	-
返回值	
-	-

例如：

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

函数 usart_receiver_timeout_disable

函数usart_receiver_timeout_disable描述见下表：

表 3-864. 函数 usart_receiver_timeout_disable

函数名称	usart_receiver_timeout_disable
函数原型	void usart_receiver_timeout_disable(uint32_t usart_periph);
功能描述	接收器超时进行能
先决条件	-
被调用函数	-
输入参数{in}	

usart_periph	外设USARTx
<i>USARTx</i>	x=0,1,2,5
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* disable USART0 receiver timeout */
```

```
usart_receiver_timeout_disable(USART0);
```

函数 usart_receiver_timeout_threshold_config

函数usart_receiver_timeout_threshold_config描述见下表：

表 3-865. 函数 usart_receiver_timeout_threshold_config

函数名称	usart_receiver_timeout_threshold_config
函数原型	void usart_receiver_timeout_threshold_config(uint32_t usart_periph, uint32_t rtimeout);
功能描述	设置接收器超时阈值
先决条件	-
被调用函数	-
输入参数{in}	
usart_periph	外设USARTx
<i>USARTx</i>	x=0,1,2,5
输入参数{in}	
rtimeout	0-0xFFFFFFFF
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* set the receiver timeout threshold of USART0 */
```

```
usart_receiver_timeout_threshold_config(USART0, 0xFFFF);
```

函数 usart_data_transmit

函数usart_data_transmit描述见下表：

表 3-866. 函数 usart_data_transmit

函数名称	usart_data_transmit
函数原型	void usart_data_transmit(uint32_t usart_periph, uint16_t data);
功能描述	USART发送数据功能

先决条件	-
被调用函数	-
输入参数{in}	
usart_periph	外设USARTx/UARTx
<i>USARTx</i>	x=0,1,2,5
<i>UARTx</i>	x=3,4,6,7
输入参数{in}	
data	发送的数据
<i>0-0xFF</i>	发送的数据
输出参数{out}	
-	-
返回值	
-	-

例如：

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

函数 usart_data_receive

函数usart_data_receive描述见下表：

表 3-867. 函数 usart_data_receive

函数名称	usart_data_receive
函数原型	uint16_t usart_data_receive(uint32_t usart_periph);
功能描述	USART接收数据功能
先决条件	-
被调用函数	-
输入参数{in}	
usart_periph	外设USARTx/UARTx
<i>USARTx</i>	x=0,1,2,5
<i>UARTx</i>	x=3,4,6,7
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 3-868. 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	配置在地址匹配唤醒模式下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=0,1,2,5
UARTx	x=3,4,6,7
输入参数{in}	
addr	USART/UART地址
0-0xFF	USART/UART地址
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* configure address of the USART0 */
```

```
usart_address_config(USART0, 0x00);
```

函数 usart_mute_mode_enable

函数usart_mute_mode_enable描述见下表：

表 3-869. 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=0,1,2,5
UARTx	x=3,4,6,7
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 3-870. 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	禁能静默模式
先决条件	-
被调用函数	-
输入参数{in}	
usart_periph	外设USARTx/UARTx
USARTx	x=0,1,2,5
UARTx	x=3,4,6,7
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 3-871. 函数 usart_mute_mode_wakeup_config

函数名称	usart_mute_mode_wakeup_config
函数原型	void usart_mute_mode_wakeup_config(uint32_t usart_periph, uint32_t wmethod);
功能描述	配置静默模式唤醒方式
先决条件	-
被调用函数	-
输入参数{in}	
usart_periph	外设USARTx/UARTx
USARTx	x=0,1,2,5
UARTx	x=3,4,6,7
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>	地址掩码唤醒
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* configure USART0 wakeup method in mute mode */
usart_mute_mode_wakeup_config(USART0, USART_WM_IDLE);
```

函数 usart_lin_mode_enable

函数usart_lin_mode_enable描述见下表：

表 3-872. 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=0,1,2,5
<i>UARTx</i>	x=3,4,6,7
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 3-873. 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	禁能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=0,1,2,5
UARTx	x=3,4,6,7
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 3-874. 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=0,1,2,5
UARTx	x=3,4,6,7
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位
USART_LBLEN_11 B	断开帧长度为11位
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 3-875. 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	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=0,1,2,5
UARTx	x=3,4,6,7
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 3-876. 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=0,1,2,5
UARTx	x=3,4,6,7
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 3-877. 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	禁能半双工模式
先决条件	-
被调用函数	-
输入参数{in}	
usart_periph	外设USARTx/UARTx
USARTx	x=0,1,2,5
UARTx	x=3,4,6,7
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 3-878. 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=0,1,2,5
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 3-879. 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	在同步通讯模式下禁能CK引脚
先决条件	-
被调用函数	-
输入参数{in}	
usart_periph	外设USARTx
USARTx	x=0,1,2,5
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 3-880. 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=0,1,2,5
输入参数{in}	
clen	CK信号长度
USART_CLEN_NO NE	8位数据帧中有7个CK脉冲，9位数据帧中有8个CK脉冲
USART_CLEN_EN	8位数据帧中有8个CK脉冲，9位数据帧中有9个CK脉冲

输入参数{in}	
cph	时钟相位
<i>USART_CPH_1CK</i>	在首个时钟边沿采样第一个数据
<i>USART_CPH_2CK</i>	在第二个时钟边沿采样第一个数据
输入参数{in}	
cpl	时钟极性
<i>USART_CPL_LOW</i>	CK引脚不对外发送时保持为低电平
<i>USART_CPL_HIGH</i>	CK引脚不对外发送时保持为高电平
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* configure USART0 synchronous mode parameters */

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 3-881. 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
<i>USARTx</i>	x=0,1,2,5
输入参数{in}	
guat	保护时间值
<i>0-0x000000FF</i>	保护时间值
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* configure USART0 guard time value in smartcard mode */

usart_guard_time_config(USART0, 0x00000055);
```

函数 usart_smartcard_mode_enable

函数usart_smartcard_mode_enable描述见下表：

表 3-882. 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=0,1,2,5
输出参数{out}	
-	-
返回值	
-	-

例如：

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

函数 usart_smartcard_mode_disable

函数usart_smartcard_mode_disable描述见下表：

表 3-883. 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	禁能智能卡模式
先决条件	-
被调用函数	-
输入参数{in}	
usart_periph	外设USARTx
USARTx	x=0,1,2,5
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* disable USART0 smartcard mode */
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 3-884. 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=0,1,2,5
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 3-885. 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	在智能卡模式下禁能NACK
先决条件	-
被调用函数	-
输入参数{in}	
usart_periph	外设USARTx
USARTx	x=0,1,2,5
输出参数{out}	
-	-
返回值	
-	-

例如：

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

函数 usart_smartcard_autoretry_config

函数usart_smartcard_autoretry_config描述见下表：

表 3-886. 函数 usart_smartcard_autoretry_config

函数名称	usart_smartcard_autoretry_config
函数原型	void usart_smartcard_autoretry_config(uint32_t usart_periph, uint32_t scrtnum);
功能描述	配置智能卡自动重试次数
先决条件	-
被调用函数	-
输入参数{in}	
usart_periph	外设USARTx/UARTx
USARTx	x=0,1,2,5
输入参数{in}	
scrtnum	智能卡自动重试次数
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* configure USART0 smartcard auto-retry number */
usart_smartcard_autoretry_config(USART0, 0x00000005);
```

函数 usart_block_length_config

函数usart_block_length_config描述见下表：

表 3-887. 函数 usart_block_length_config

函数名称	usart_block_length_config
函数原型	void usart_block_length_config(uint32_t usart_periph, uint32_t bl);
功能描述	配置块长度
先决条件	-
被调用函数	-
输入参数{in}	
usart_periph	外设USARTx/UARTx
USARTx	x=0,1,2,5
输入参数{in}	
bl	块长度
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* configure USART0 block length in Smartcard T=1 reception */
usart_block_length_config(USART0, 0x00000005);
```

函数 usart_irda_mode_enable

函数usart_irda_mode_enable描述见下表：

表 3-888. 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=0,1,2,5
UARTx	x=3,4,6,7
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 3-889. 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	禁能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=0,1,2,5
UARTx	x=3,4,6,7
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 3-890. 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	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=0,1,2,5
UARTx	x=3,4,6,7
输入参数{in}	
psc	时钟分频系数
0-0xFF	时钟分频系数
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* configure the USART0 peripheral clock prescaler in USART IrDA low-power mode */
```

```
usart_prescaler_config(USART0, 0x00);
```

函数 usart_irda_lowpower_config

函数usart_irda_lowpower_config描述见下表：

表 3-891. 函数 usart_irda_lowpower_config

函数名称	usart_irda_lowpower_config
函数原型	void usart_irda_lowpower_config(uint32_t usart_periph, uint32_t irlp);
功能描述	配置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
<i>USARTx</i>	x=0,1,2,5
<i>UARTx</i>	x=3,4,6,7
输入参数{in}	
irlp	IrDA低功耗模式或正常模式
<i>USART_IRLP_LOW</i>	低功耗模式
<i>USART_IRLP_NORMAL</i>	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 3-892. 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
<i>USARTx</i>	x=0,1,2,5
输入参数{in}	
rtsconfig	使能/禁能RTS
<i>USART_RTS_ENABLE</i>	使能RTS
<i>USART_RTS_DISABLE</i>	禁能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 3-893. 函数 usart_hardware_flow_cts_config

函数名称	usart_hardware_flow_cts_config
函数原型	void usart_hardware_flow_cts_config(uint32_t usart_periph, uint32_t ctsconfig);
功能描述	配置CTS硬件控制流
先决条件	-
被调用函数	-
输入参数{in}	
usart_periph	外设USARTx
USARTx	x=0,1,2,5
输入参数{in}	
ctsconfig	使能/禁能CTS
USART_CTS_ENABLE	使能CTS
USART_CTS_DISABLE	禁能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 3-894. 函数 usart_dma_receive_config

函数名称	usart_dma_receive_config
函数原型	void usart_dma_receive_config (uint32_t usart_periph, uint8_t dmaconfig);
功能描述	配置USART DMA接收
先决条件	-
被调用函数	-
输入参数{in}	
usart_periph	外设USARTx/UARTx
USARTx	x=0,1,2,5
UARTx	x=3,4,6,7

输入参数{in}	
dmaconfig	USART DMA模式
<i>USART_RECEIVE_DMA_ENABLE</i>	使能DMA接收功能
<i>USART_RECEIVE_DMA_DISABLE</i>	禁能DMA接收功能
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* enable USART0 DMA for reception */
```

```
usart_dma_receive_config(USART0, USART_RECEIVE_DMA_ENABLE);
```

函数 usart_dma_transmit_config

函数usart_dma_transmit_config描述见下表：

表 3-895. 函数 usart_dma_transmit_config

函数名称	usart_dma_transmit_config
函数原型	void usart_dma_transmit_config(uint32_t usart_periph, uint8_t dmaconfig);
功能描述	配置USART DMA发送
先决条件	-
被调用函数	-
输入参数{in}	
usart_periph	外设USARTx/UARTx
<i>USARTx</i>	x=0,1,2,5
<i>UARTx</i>	x=3,4,6,7
输入参数{in}	
dmaconfig	USART DMA模式
<i>USART_TRANSMIT_DMA_ENABLE</i>	使能DMA发送功能
<i>USART_TRANSMIT_DMA_DISABLE</i>	禁能DMA发送功能
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* enable USART0 DMA for transmission */
```

```
usart_dma_transmit_config(USART0, USART_TRANSMIT_DMA_ENABLE);
```

函数 usart_flag_get

函数usart_flag_get描述见下表:

表 3-896. 函数 usart_flag_get

函数名称	usart_flag_get
函数原型	FlagStatus usart_flag_get(uint32_t usart_periph, usart_flag_enum flag);
功能描述	获取USART状态寄存器标志位
先决条件	-
被调用函数	-
输入参数{in}	
usart_periph	外设USARTx/UARTx
USARTx	x=0,1,2,5
UARTx	x=3,4,6,7
输入参数{in}	
flag	USART标志位, 参考 表3-848. 枚举类型usart_flag_enum 。
输出参数{out}	
-	-
返回值	
FlagStatus	SET或RESET

例如:

```
/* get flag USART0 state */
```

```
FlagStatus status;
```

```
status = usart_flag_get(USART0,USART_FLAG_TBE);
```

函数 usart_flag_clear

函数usart_flag_clear描述见下表:

表 3-897. 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=0,1,2,5
UARTx	x=3,4,6,7
输入参数{in}	
flag	USART标志位, 参考 表3-848. 枚举类型usart_flag_enum 。
USART_FLAG_CTS	CTS变化标志

USART_FLAG_LBD	LIN断开检测标志
USART_FLAG_TC	发送完成
USART_FLAG_RBNE	读数据缓冲区非空
USART_FLAG_EB	块结束标志
USART_FLAG_RT	接收超时标志
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 3-898. 函数 usart_interrupt_enable

函数名称	usart_interrupt_enable
函数原型	void usart_interrupt_enable(uint32_t usart_periph, usart_interrupt_enum interrupt);
功能描述	使能USART中断
先决条件	-
被调用函数	-
输入参数{in}	
usart_periph	外设USARTx/UARTx
USARTx	x=0,1,2,5
UARTx	x=3,4,6,7
输入参数{in}	
interrupt	USART中断标志，参考 表3-850. 枚举类型usart_interrupt_enum 。
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 3-899. 函数 usart_interrupt_disable

函数名称	usart_interrupt_disable
函数原型	void usart_interrupt_disable(uint32_t usart_periph, usart_interrupt_enum interrupt);
功能描述	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=0,1,2,5
UARTx	x=3,4,6,7
输入参数{in}	
interrupt	USART中断标志, 表3-850. 枚举类型usart_interrupt_enum 。
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 3-900. 函数 usart_interrupt_flag_get

函数名称	usart_interrupt_flag_get
函数原型	FlagStatus usart_interrupt_flag_get(uint32_t usart_periph, usart_interrupt_flag_enum int_flag);
功能描述	获取USART中断标志位状态
先决条件	-
被调用函数	-
输入参数{in}	
usart_periph	外设USARTx/UARTx
USARTx	x=0,1,2,5
UARTx	x=3,4,6,7
输入参数{in}	
int_flag	USART中断标志, 参考 表3-849. 枚举类型usart_interrupt_flag_enum 。
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 3-901. 函数 usart_interrupt_flag_clear

函数名称	usart_interrupt_flag_clear
函数原型	void usart_interrupt_flag_clear(uint32_t usart_periph, usart_interrupt_flag_enum int_flag);
功能描述	清除USART中断标志位状态
先决条件	-
被调用函数	-
输入参数{in}	
usart_periph	外设USARTx/UARTx
USARTx	x=0,1,2
UARTx	x=3,4
输入参数{in}	
int_flag	USART中断标志, 参考 表3-849. 枚举类型usart_interrupt_flag_enum 。
USART_INT_FLAG_CTS	CTS中断和标志
USART_INT_FLAG_LBD	LIN断开检测中断和标志
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_EB	块结束标志中断标志位
USART_INT_FLAG_RT	接收超时标志中断标志位
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);
```

3.29. 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[3.29.1](#)描述了WWDGT的寄存器列表，章节[3.29.2](#)对WWDGT库函数进行说明。

3.29.1. 外设寄存器说明

WWDGT寄存器列表如下表所示:

表 3-902. 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

3.29.2. 外设库函数说明

WWDGT库函数列表如下表所示:

表 3-903. WWDGT 库函数

库函数名称	库函数说明
wwdgt_deinit	将WWDGT寄存器重设为缺省值
wwdgt_enable	使能WWDGT
wwdgt_counter_update	设置WWDGT计数器更新值
wwdgt_config	设置WWDGT计数器值、窗口值和预分频值
wwdgt_flag_get	检查WWDGT提前唤醒中断标志位是否置位
wwdgt_flag_clear	清除WWDGT提前唤醒中断标志位状态
wwdgt_interrupt_enable	使能WWDGT提前唤醒中断
wwdgt_interrupt_flag_get	检查WWDGT提前唤醒中断标志位是否置位

函数 wwdgt_deinit

函数wwdgt_deinit描述见下表:

表 3-904. 函数 wwdgt_deinit

函数名称	wwdgt_deinit
函数原型	void wwdgt_deinit(void);
功能描述	将WWDGT寄存器重设为缺省值
先决条件	-

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

例如：

```
/* reset the window watchdog timer configuration */
```

```
wwdgt_deinit();
```

函数 wwdgt_enable

函数wwdgt_enable描述见下表：

表 3-905. 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 3-906. 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 3-907. 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_flag_get

函数wwdgt_flag_get描述见下表:

表 3-908. 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 reaches the 0x40 or refreshes before it reaches the window value */
FlagStatus status;

status = wwdgt_flag_get();
```

函数 wwdgt_flag_clear

函数wwdgt_flag_clear描述见下表:

表 3-909. 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();
```

函数 wwdgt_interrupt_enable

函数wwdgt_interrupt_enable描述见下表:

表 3-910. 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_interrupt_flag_get

函数wwdgt_interrupt_flag_get描述见下表:

表 3-911. 函数 wwdgt_interrupt_flag_get

函数名称	wwdgt_interrupt_flag_get
函数原型	FlagStatus wwdgt_interrupt_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 reaches the 0x40 or refreshes before it reaches the window value */
```

```
FlagStatus status;
```

```
status = wwdgt_interrupt_flag_gett ();
```

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 10 月 31 日
1.1	修订内容: GD32F20x 无 USB, 目录中删除 USB 章节; 修改 2.1.2 节文件夹描述; 更换 2.1 节文件夹结构图; 修改 2.1.4 节文件夹描述; 修改部分图表头格式	2020 年 9 月 30 日
1.2	Rebase 版本发布	2021 年 10 月 31 日
1.3	<ol style="list-style-type: none"> 1. 添加 exmc_sdram_struct_command_para_init 函数 2. 修改 void pmu_to_standbymode(uint8_t standbymodecmd)函数 3. 所有 qspi_xxx 函数都改为 spi_quad_xxx 4. 添加 fwdgt_prescaler_value_config 与 fwdgt_reload_value_config 函数 5. 删除 USBFS 章节 	2022 年 7 月 31 日
1.4	<ol style="list-style-type: none"> 1. 修改 usart_dma_enable() 和 usart_dma_disable()函数为 usart_dma_receive_config()和 usart_dma_transmit_config() 2. 修改 tli_ckey_init() 函数为 tli_color_key_init() 3. 添加函数 tli_struct_para_init() tli_layer_struct_para_init() tli_lut_struct_para_init() 	2023 年 6 月 30 日
1.5	修改 DAC 整个章节	2023 年 12 月 31 日
1.6	更新函数 can_transmission_stop()	2026 年 2 月 6 日

Important Notice

This document is the property of GigaDevice Semiconductor Inc. and its subsidiaries (the "Company"). This document, including any product of the Company described in this document (the "Product"), is owned by the Company according to the laws of the People's Republic of China and other applicable laws. The Company reserves all rights under such laws and no Intellectual Property Rights are transferred (either wholly or partially) or licensed by the Company (either expressly or impliedly) herein. The names and brands of third party referred thereto (if any) are the property of their respective owner and referred to for identification purposes only.

To the maximum extent permitted by applicable law, the Company makes no representations or warranties of any kind, express or implied, with regard to the merchantability and the fitness for a particular purpose of the Product, nor does the Company assume any liability arising out of the application or use of any Product. Any information provided in this document is provided only for reference purposes. It is the sole responsibility of the user of this document to determine whether the Product is suitable and fit for its applications and products planned, and properly design, program, and test the functionality and safety of its applications and products planned using the Product. The Product is designed, developed, and/or manufactured for ordinary business, industrial, personal, and/or household applications only, and the Product is not designed or intended for use in (i) safety critical applications such as weapons systems, nuclear facilities, atomic energy controller, combustion controller, aeronautic or aerospace applications, traffic signal instruments, pollution control or hazardous substance management; (ii) life-support systems, other medical equipment or systems (including life support equipment and surgical implants); (iii) automotive applications or environments, including but not limited to applications for active and passive safety of automobiles (regardless of front market or aftermarket), for example, EPS, braking, ADAS (camera/fusion), EMS, TCU, BMS, BSG, TPMS, Airbag, Suspension, DMS, ICMS, Domain, ESC, DCDC, e-clutch, advanced-lighting, etc.. Automobile herein means a vehicle propelled by a self-contained motor, engine or the like, such as, without limitation, cars, trucks, motorcycles, electric cars, and other transportation devices; and/or (iv) other uses where the failure of the device or the Product can reasonably be expected to result in personal injury, death, or severe property or environmental damage (collectively "Unintended Uses"). Customers shall take any and all actions to ensure the Product meets the applicable laws and regulations. The Company is not liable for, in whole or in part, and customers shall hereby release the Company as well as its suppliers and/or distributors from, any claim, damage, or other liability arising from or related to all Unintended Uses of the Product. Customers shall indemnify and hold the Company, and its officers, employees, subsidiaries, affiliates as well as its suppliers and/or distributors harmless from and against all claims, costs, damages, and other liabilities, including claims for personal injury or death, arising from or related to any Unintended Uses of the Product.

Information in this document is provided solely in connection with the Product. The Company reserves the right to make changes, corrections, modifications or improvements to this document and the Product described herein at any time without notice. The Company shall have no responsibility whatsoever for conflicts or incompatibilities arising from future changes to them. Information in this document supersedes and replaces information previously supplied in any prior versions of this document.