

GigaDevice Semiconductor Inc.

GD32F50x

Arm[®] Cortex[®]-M33 32-bit MCU

**固件库
使用指南**

1.1 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	32
2.2. 固件库文件描述	32
3. 外设固件库	34
3.1. 外设固件库概述	34
3.2. ADC	34
3.2.1. 外设寄存器描述	34
3.2.2. 外设库函数说明	35
3.3. BKP	61
3.3.1. 外设寄存器说明	61
3.3.2. 外设库函数说明	61
3.4. CAN	75
3.4.1. 外设寄存器说明	75
3.4.2. 外设库函数说明	76
3.5. CAU	99
3.5.1. 外设寄存器说明	100
3.5.2. 外设库函数说明	100
3.6. CMP	112
3.6.1. 外设寄存器说明	112
3.6.2. 外设库函数说明	112
3.7. CRC	125
3.7.1. 外设寄存器说明	125
3.7.2. 外设库函数说明	125

3.8. CTC	133
3.8.1. 外设寄存器说明	133
3.8.2. 外设库函数说明	133
3.9. DAC	146
3.9.1. 外设寄存器说明	146
3.9.2. 外设库函数说明	146
3.10. DBG	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. DMA / DMAMUX	169
3.11.1. 外设寄存器说明	169
3.11.2. 外设库函数说明	170
3.12. EXMC	215
3.12.1. 外设寄存器说明	215
3.12.2. 外设库函数说明	215
3.13. EXTI	229
3.13.1. 外设寄存器说明	229
3.13.2. 外设库函数说明	229
3.14. FMC	237
3.14.1. 外设寄存器描述	237
3.14.2. 外设库函数说明	237
3.15. FWDGT	266
3.15.1. 外设寄存器说明	266
3.15.2. 外设库函数说明	266
3.16. GPIO	272
3.16.1. 外设寄存器说明	272
3.16.2. 外设库函数说明	272
3.17. HAU	283
3.17.1. 外设寄存器说明	283
3.17.2. 外设库函数说明	283
3.18. I2C	294
3.18.1. 外设寄存器说明	294
3.18.2. 外设库函数说明	294
3.19. MISC	332
3.19.1. 外设寄存器说明	332
3.19.2. 外设库函数说明	333
3.20. PMU	340
3.20.1. 外设寄存器说明	340
3.20.2. 外设库函数说明	340

3.21.	RCU.....	362
3.21.1.	外设寄存器描述	362
3.21.2.	外设库函数说明	362
3.22.	RTC	408
3.22.1.	外设寄存器描述	408
3.22.2.	外设库函数描述	408
3.23.	SYSCFG	416
3.23.1.	外设寄存器说明	416
3.23.2.	外设库函数说明	418
3.24.	SPI.....	432
3.24.1.	外设寄存器说明	432
3.24.2.	外设库函数说明	432
3.25.	TIMER	457
3.25.1.	外设寄存器说明	458
3.25.2.	外设库函数说明	459
3.26.	TRIGSEL.....	550
3.26.1.	外设寄存器说明	550
3.26.2.	外设库函数说明	551
3.27.	TRNG.....	557
3.27.1.	外设寄存器说明	557
3.27.2.	外设库函数说明	557
3.28.	USART	563
3.28.1.	外设寄存器说明	563
3.28.2.	外设库函数说明	563
3.29.	WWDGT	599
3.29.1.	外设寄存器说明	599
3.29.2.	外设库函数说明	599
4.	版本历史.....	605

图索引

图 2-1. GD32F50x 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	31
图 2-4. 打开工程文件	31
图 2-5. 配置工程文件	32
图 2-6. 编译调试下载	32

表索引

表 1-1. 外设缩写.....	27
表 2-1. 固件函数库文件描述	32
表 3-1. 外设固件库函数描述格式.....	34
表 3-2. ADC 寄存器	34
表 3-3. ADC 库函数	35
表 3-4. 函数 adc_deinit	36
表 3-5. 函数 adc_enable.....	36
表 3-6. 函数 adc_disable	37
表 3-7. 函数 adc_dma_mode_enable	37
表 3-8. 函数 adc_dma_mode_disable	38
表 3-9. 函数 adc_discontinuous_mode_config	39
表 3-10. 函数 adc_special_function_config	39
表 3-11. 函数 adc_internal_channel_config	40
表 3-12. 函数 adc_sync_mode_config	41
表 3-13. 函数 adc_sync_routine_data_read	42
表 3-14. 函数 adc_data_alignment_config	43
表 3-15. 函数 adc_channel_length_config	43
表 3-16. 函数 adc_routine_channel_config	44
表 3-17. 函数 adc_inserted_channel_config	45
表 3-18. 函数 adc_inserted_channel_offset_config	46
表 3-19. 函数 adc_external_trigger_config	47
表 3-20. 函数 adc_software_trigger_enable	47
表 3-21. 函数 adc_routine_data_read.....	48
表 3-22. 函数 adc_inserted_data_read.....	49
表 3-23. 函数 adc_latch_data_read.....	49
表 3-24. 函数 adc_latch_data_souce_config	50
表 3-25. 函数 adc_watchdog0_single_channel_enable	51
表 3-26. 函数 adc_watchdog0_sequence_channel_enable	51
表 3-27. 函数 adc_watchdog0_disable	52
表 3-28. 函数 adc_watchdog0_threshold_config	52
表 3-29. 函数 adc_resolution_config.....	53
表 3-30. 函数 adc_oversample_mode_config.....	54
表 3-31. 函数 adc_oversample_mode_enable	55
表 3-32. 函数 adc_oversample_mode_disable	56
表 3-33. 函数 adc_flag_get.....	57
表 3-34. 函数 adc_flag_clear.....	57
表 3-35. 函数 adc_interrupt_enable	58
表 3-36. 函数 adc_interrupt_disable.....	59
表 3-37. 函数 adc_interrupt_flag_get	59
表 3-38. 函数 adc_interrupt_flag_clear	60
表 3-39. BKP 寄存器	61

表 3-40. BKP 库函数	61
表 3-41. 函数 bkp_deinit	62
表 3-42. 函数 bkp_write_data	63
表 3-43. 函数 bkp_read_data	63
表 3-44. 函数 bkp_rtc_calibration_output_enable	64
表 3-45. 函数 bkp_rtc_calibration_output_disable	65
表 3-46. 函数 bkp_rtc_signal_output_enable	65
表 3-47. 函数 bkp_rtc_signal_output_disable	66
表 3-48. 函数 bkp_rtc_output_select	66
表 3-49. 函数 bkp_rtc_clock_output_select	67
表 3-50. 函数 bkp_rtc_clock_calibration_direction_select	68
表 3-51. 函数 bkp_rtc_calibration_value_set	69
表 3-52. 函数 bkp_tamper_detection_enable	69
表 3-53. 函数 bkp_tamper_detection_disable	70
表 3-54. 函数 bkp_tamper_active_level_set	71
表 3-55. 函数 bkp_tamper_interrupt_enable	71
表 3-56. 函数 bkp_interrupt_disable	72
表 3-57. 函数 bkp_flag_get	72
表 3-58. 函数 bkp_flag_clear	73
表 3-59. 函数 bkp_interrupt_flag_get	74
表 3-60. 函数 bkp_interrupt_flag_clear	74
表 3-61. CAN 寄存器	75
表 3-62. CAN 库函数	76
表 3-63. 结构体 can_parameter_struct	77
表 3-64. 结构体 can_transmit_message_struct	77
表 3-65. 结构体 can_receive_message_struct	78
表 3-66. 结构体 can_filter_parameter_struct	78
表 3-67. 结构体 can_fd_tdc_struct	78
表 3-68. 结构体 can_fdframe_struct	78
表 3-69. 函数 can_deinit	79
表 3-70. 函数 can_struct_para_init	79
表 3-71. 函数 can_init	80
表 3-72. 函数 can_fd_init	81
表 3-73. 函数 can_filter_init	81
表 3-74. 函数 can_filter_mask_mode_init	82
表 3-75. 函数 can_monitor_mode_set	83
表 3-76. 函数 can_fd_function_enable	83
表 3-77. 函数 can_fd_function_disable	84
表 3-78. 函数 can1_filter_start_bank	84
表 3-79. 函数 can_debug_freeze_enable	85
表 3-80. 函数 can_debug_freeze_disable	85
表 3-81. 函数 can_time_trigger_mode_enable	86
表 3-82. 函数 can_time_trigger_mode_disable	86
表 3-83. 函数 can_message_transmit	87

表 3-84. 函数 can_transmit_states	87
表 3-85. 函数 can_transmission_stop	88
表 3-86. 函数 can_message_receive	89
表 3-87. 函数 can_fifo_release	89
表 3-88. 函数 can_receive_message_length_get	90
表 3-89. 函数 can_working_mode_set	91
表 3-90. 函数 can_wakeup	91
表 3-91. 函数 can_error_get	92
表 3-92. 函数 can_receive_error_number_get	92
表 3-93. 函数 can_transmit_error_number_get	93
表 3-94. 函数 can_flag_get	93
表 3-95. 函数 can_flag_clear	94
表 3-96. 函数 can_interrupt_enable	95
表 3-97. 函数 can_interrupt_disable	96
表 3-98. 函数 can_interrupt_flag_get	97
表 3-99. 函数 can_interrupt_flag_clear	98
表 3-100. 函数 can_fifo_overrun_flag_get	99
表 3-101. CAU 寄存器	100
表 3-102. CAU 库函数	100
表 3-103. 结构体 cau_key_parameter_struct	101
表 3-104. 结构体 cau_parameter_struct	101
表 3-105. 函数 cau_deinit	101
表 3-106. 函数 cau_struct_para_init	102
表 3-107. 函数 cau_key_struct_para_init	102
表 3-108. 函数 cau_enable	103
表 3-109. 函数 cau_disable	103
表 3-110. 函数 cau_dma_enable	104
表 3-111. 函数 cau_dma_disable	104
表 3-112. 函数 cau_init	105
表 3-113. 函数 cau_aes_keysize_config	106
表 3-114. 函数 cau_key_init	106
表 3-115. 函数 cau_fifo_flush	107
表 3-116. 函数 cau_enable_state_get	107
表 3-117. 函数 cau_data_write	108
表 3-118. 函数 cau_data_read	108
表 3-119. 函数 cau_aes_ecb	109
表 3-120. 函数 cau_flag_get	110
表 3-121. 函数 cau_interrupt_enable	110
表 3-122. 函数 cau_interrupt_disable	111
表 3-123. 函数 cau_interrupt_flag_get	111
表 3-124. CMP 寄存器	112
表 3-125. CMP 库函数	112
表 3-126. 枚举类型 cmp_enum	113
表 3-127. 函数 cmp_deinit	113

表 3-128. 函数 cmp_mode_init	114
表 3-129. 函数 cmp_output_init.....	115
表 3-130. 函数 cmp_blanking_init.....	115
表 3-131. 函数 cmp_digital_filter_init.....	116
表 3-132. 函数 cmp_enable	117
表 3-133. 函数 cmp_disable	118
表 3-134. 函数 cmp_lock_enable	118
表 3-135. 函数 cmp_voltage_scaler_enable.....	119
表 3-136. 函数 cmp_voltage_scaler_disable.....	119
表 3-137. 函数 cmp_scaler_bridge_enable	120
表 3-138. 函数 cmp_scaler_bridge_disable	120
表 3-139. 函数 cmp_output_level_get.....	121
表 3-140. 函数 cmp_flag_get.....	121
表 3-141. 函数 cmp_flag_clear	122
表 3-142. 函数 cmp_interrupt_enable	122
表 3-143. 函数 cmp_interrupt_disable	123
表 3-144. 函数 cmp_interrupt_flag_get.....	124
表 3-145. 函数 cmp_interrupt_flag_clear.....	124
表 3-146. CRC 寄存器.....	125
表 3-147. CRC 库函数.....	125
表 3-148. 函数 crc_deinit	126
表 3-149. 函数 crc_init_data_register_write.....	126
表 3-150. 函数 crc_data_register_read	126
表 3-151. 函数 crc_free_data_register_read	127
表 3-152. 函数 crc_free_data_register_write	128
表 3-153. 函数 crc_reverse_output_data_disable.....	128
表 3-154. 函数 crc_reverse_output_data_enable.....	128
表 3-155. 函数 crc_input_data_reverse_config	129
表 3-156. 函数 crc_data_register_reset	130
表 3-157. 函数 crc_polynomial_size_set	130
表 3-158. 函数 crc_polynomial_set.....	131
表 3-159. 函数 crc_single_data_calculate	131
表 3-160. 函数 crc_block_data_calculate	132
表 3-161. CTC 寄存器	133
表 3-162. CTC 库函数	133
表 3-163. 函数 ctc_deinit	134
表 3-164. 函数 ctc_counter_enable	134
表 3-165. 函数 ctc_counter_disable	135
表 3-166. 函数 ctc_irc48m_trim_value_config.....	135
表 3-167. 函数 ctc_software_refsource_pulse_generate.....	136
表 3-168. 函数 ctc_hardware_trim_mode_config.....	136
表 3-169. 函数 ctc_refsource_polarity_config.....	137
表 3-170. 函数 ctc_refsource_signal_select	137
表 3-171. 函数 ctc_refsource_prescaler_config.....	138

表 3-172. 函数 ctc_clock_limit_value_config.....	139
表 3-173. 函数 ctc_counter_reload_value_config.....	139
表 3-174. 函数 ctc_counter_capture_value_read.....	140
表 3-175. 函数 ctc_counter_direction_read.....	140
表 3-176. 函数 ctc_counter_reload_value_read.....	141
表 3-177. 函数 ctc_irc48m_trim_value_read.....	141
表 3-178. 函数 ctc_interrupt_enable.....	142
表 3-179. 函数 ctc_interrupt_disable.....	142
表 3-180. 函数 ctc_interrupt_flag_get.....	143
表 3-181. 函数 ctc_interrupt_flag_clear.....	144
表 3-182. 函数 ctc_flag_get.....	145
表 3-183. 函数 ctc_flag_clear.....	145
表 3-184. DAC 寄存器.....	146
表 3-185. DAC 库函数.....	147
表 3-186. 函数 dac_deinit.....	147
表 3-187. 函数 dac_enable.....	148
表 3-188. 函数 dac_disable.....	148
表 3-189. 函数 dac_dma_enable.....	149
表 3-190. 函数 dac_dma_disable.....	149
表 3-191. 函数 dac_dma_disable.....	150
表 3-192. 函数 dac_output_buffer_enable.....	151
表 3-193. 函数 dac_output_buffer_disable.....	151
表 3-194. 函数 dac_output_value_get.....	152
表 3-195. 函数 dac_data_set.....	152
表 3-196. 函数 dac_trigger_enable.....	153
表 3-197. 函数 dac_trigger_disable.....	154
表 3-198. 函数 dac_trigger_source_config.....	154
表 3-199. 函数 dac_software_trigger_enable.....	155
表 3-200. 函数 dac_wave_mode_config.....	156
表 3-201. 函数 dac_lfsr_noise_config.....	156
表 3-202. 函数 dac_triangle_noise_config.....	157
表 3-203. 函数 dac_output_connect_to_pin_enable.....	158
表 3-204. 函数 dac_output_connect_to_pin_enable.....	158
表 3-205. 函数 dac_flag_get.....	159
表 3-206. 函数 dac_flag_clear.....	160
表 3-207. 函数 dac_interrupt_enable.....	160
表 3-208. 函数 dac_interrupt_disable.....	161
表 3-209. 函数 dac_interrupt_flag_get.....	161
表 3-210. 函数 dac_interrupt_flag_clear.....	162
表 3-211. DBG 寄存器.....	163
表 3-212. DBG 库函数.....	163
表 3-213. 枚举类型 dbg_periph_enum.....	163
表 3-214. 函数 dbg_deinit.....	164
表 3-215. 函数 dbg_id_get.....	164

表 3-216. 函数 dbg_low_power_enable	165
表 3-217. 函数 dbg_low_power_disable	165
表 3-218. 函数 dbg_periph_enable	166
表 3-219. 函数 dbg_periph_disable	167
表 3-220. 函数 dbg_trace_pin_enable	167
表 3-221. 函数 dbg_trace_pin_disable	168
表 3-222. 函数 dbg_trace_pin_mode_set	168
表 3-223. DMA 寄存器	169
表 3-224. DMAMUX 寄存器	170
表 3-225. DMA 库函数	170
表 3-226. DMAMUX 库函数	171
表 3-227. 结构体 dma_parameter_struct	172
表 3-228. 结构体 dmamux_sync_parameter_struct	172
表 3-229. 结构体 dmamux_gen_parameter_struct	172
表 3-230. 枚举 dma_channel_enum	172
表 3-231. 枚举 dmamux_multiplexer_channel_enum	173
表 3-232. 枚举 dmamux_generator_channel_enum	173
表 3-233. 枚举 dmamux_interrupt_enum	173
表 3-234. 枚举 dmamux_flag_enum	174
表 3-235. 枚举 dmamux_interrupt_flag_enum	175
表 3-236. 函数 dma_deinit	176
表 3-237. 函数 dma_struct_para_init	177
表 3-238. 函数 dma_init	177
表 3-239. 函数 dma_circulation_enable	178
表 3-240. 函数 dma_circulation_disable	179
表 3-241. 函数 dma_memory_to_memory_enable	179
表 3-242. 函数 dma_memory_to_memory_disable	180
表 3-243. 函数 dma_channel_enable	180
表 3-244. 函数 dma_channel_disable	181
表 3-245. 函数 dma_periph_address_config	181
表 3-246. 函数 dma_memory_address_config	182
表 3-247. 函数 dma_transfer_number_config	183
表 3-248. 函数 dma_transfer_number_get	183
表 3-249. 函数 dma_priority_config	184
表 3-250. 函数 dma_memory_width_config	185
表 3-251. 函数 dma_periph_width_config	185
表 3-252. 函数 dma_memory_increase_enable	186
表 3-253. 函数 dma_memory_increase_disable	187
表 3-254. 函数 dma_periph_increase_enable	187
表 3-255. 函数 dma_periph_increase_disable	188
表 3-256. 函数 dma_transfer_direction_config	188
表 3-257. 函数 dma_flag_get	189
表 3-258. 函数 dma_flag_clear	190
表 3-259. 函数 dma_interrupt_enable	191

表 3-260. 函数 dma_interrupt_disable	191
表 3-261. 函数 dma_interrupt_flag_get.....	192
表 3-262. 函数 dma_interrupt_flag_clear.....	193
表 3-263. 函数 dmamux_sync_struct_para_init	194
表 3-264. 函数 dmamux_synchronization_init	194
表 3-265. 函数 dmamux_synchronization_enable	195
表 3-266. 函数 dmamux_synchronization_disable	195
表 3-267. 函数 dmamux_event_generation_enable.....	196
表 3-268. 函数 dmamux_event_generation_disable.....	196
表 3-269. 函数 dmamux_gen_struct_para_init.....	197
表 3-270. 函数 dmamux_request_generator_init	197
表 3-271. 函数 dmamux_request_generator_channel_enable	198
表 3-272. 函数 dmamux_request_generator_channel_disable	199
表 3-273. 函数 dmamux_synchronization_polarity_config	199
表 3-274. 函数 dmamux_request_forward_number_config	200
表 3-275. 函数 dmamux_sync_id_config.....	201
表 3-276. 函数 dmamux_request_id_config.....	202
表 3-277. 函数 dma_interrupt_disable	209
表 3-278. 函数 dmamux_request_generate_number_config	210
表 3-279. 函数 dmamux_trigger_id_config	210
表 3-280. 函数 dmamux_flag_get.....	212
表 3-281. 函数 dmamux_flag_clear.....	213
表 3-282. 函数 dmamux_interrupt_enable.....	213
表 3-283. 函数 dmamux_interrupt_disable.....	213
表 3-284. 函数 dmamux_interrupt_flag_get	214
表 3-285. 函数 dmamux_interrupt_flag_clear.....	214
表 3-286. EXMC 寄存器	215
表 3-287. EXMC 库函数	216
表 3-288. 结构体 exmc_norsram_timing_parameter_struct.....	216
表 3-289. 结构体 exmc_norsram_parameter_struct	217
表 3-290. 结构体 exmc_nand_timing_parameter_struct.....	217
表 3-291. 结构体 exmc_nand_parameter_struct	217
表 3-292. 函数 exmc_norsram_deinit.....	218
表 3-293. 函数 exmc_norsram_struct_para_init	219
表 3-294. 函数 exmc_norsram_init	219
表 3-295. 函数 exmc_norsram_enable	221
表 3-296. 函数 exmc_norsram_disable	221
表 3-297. 函数 exmc_nand_deinit.....	222
表 3-298. 函数 exmc_nand_struct_para_init.....	222
表 3-299. 函数 exmc_nand_init	223
表 3-300. 函数 exmc_nand_enable	224
表 3-301. 函数 exmc_nand_disable	225
表 3-302. 函数 exmc_norsram_page_size_config	225
表 3-303. 函数 exmc_nand_ecc_enable.....	226

表 3-304. 函数 exmc_nand_ecc_enable.....	227
表 3-305. 函数 exmc_ecc_get	228
表 3-306. 函数 exmc_flag_get.....	228
表 3-307. EXTI 寄存器.....	229
表 3-308. EXTI 库函数.....	229
表 3-309. 枚举类型 exti_line_enum	230
表 3-310. 枚举类型 exti_mode_enum	230
表 3-311. 枚举类型 exti_trig_type_enum.....	231
表 3-312. 函数 exti_deinit	231
表 3-313. 函数 exti_init.....	231
表 3-314. 函数 exti_interrupt_enable.....	232
表 3-315. 函数 exti_interrupt_disable.....	232
表 3-316. 函数 exti_event_enable	233
表 3-317. 函数 exti_event_disable	233
表 3-318. 函数 exti_software_interrupt_enable	234
表 3-319. 函数 exti_software_interrupt_disable	234
表 3-320. 函数 exti_flag_get.....	235
表 3-321. 函数 exti_flag_clear.....	235
表 3-322. 函数 exti_interrupt_flag_get	236
表 3-323. 函数 exti_interrupt_flag_clear	236
表 3-324. FMC 寄存器.....	237
表 3-325. FMC 库函数.....	237
表 3-326.枚举类型 fmc_state_enum	239
表 3-327.枚举类型 fmc_interrupt_enum.....	239
表 3-328.枚举类型 fmc_flag_enum	239
表 3-329.枚举类型 fmc_interrupt_flag_enum	239
表 3-330.函数 fmc_unlock.....	240
表 3-331.函数 fmc_bank0_unlock.....	240
表 3-332.函数 fmc_bank1_unlock.....	241
表 3-333.函数 fmc_lock	241
表 3-334.函数 fmc_bank0_lock	242
表 3-335.函数 fmc_bank1_lock	242
表 3-336.函数 fmc_page_erase	243
表 3-337.函数 fmc_mass_erase	243
表 3-338.函数 fmc_bank0_erase	244
表 3-339.函数 fmc_bank1_erase	244
表 3-340.函数 fmc_word_program.....	245
表 3-341.函数 fmc_halfword_program.....	245
表 3-342.函数 fmc_otp_word_program.....	246
表 3-343.函数 fmc_otp_half_word_program.....	246
表 3-344.函数 fmc_otp_byte_program.....	247
表 3-345.函数 fmc_nwa_enable.....	247
表 3-346.函数 fmc_nwa_disable.....	248
表 3-347.函数 otp1_read_disable.....	248

表 3-348.函数 otp2_rlock_enable.....	249
表 3-349.函数 fmc_deep_power_down_enable.....	249
表 3-350.函数 fmc_deep_power_down_disable.....	250
表 3-351.函数 ob_unlock.....	250
表 3-352.函数 ob_lock.....	251
表 3-353.函数 ob_start.....	251
表 3-354.函数 ob_erase.....	252
表 3-355.函数 ob_write_protection_enable.....	252
表 3-356.函数 ob_write_protection_disable.....	253
表 3-357.函数 ob_security_protection_config.....	253
表 3-358.函数 ob_user_write.....	254
表 3-359.函数 ob_sram_init_config.....	255
表 3-360.函数 ob_sram_ecc_config.....	255
表 3-361.函数 ob_nwa_clock_config.....	256
表 3-362.函数 ob_data_write.....	256
表 3-363.函数 ob_sram_init_get.....	257
表 3-364.函数 ob_sram_ecc_get.....	257
表 3-365.函数 ob_nwa_clock_get.....	258
表 3-366.函数 ob_data_get.....	258
表 3-367.函数 ob_write_protection_get.....	259
表 3-368.函数 ob_spc_get.....	259
表 3-369.函数 fmc_interrupt_enable.....	260
表 3-370.函数 fmc_interrupt_disable.....	260
表 3-371.函数 fmc_interrupt_flag_get.....	261
表 3-372.函数 fmc_interrupt_flag_clear.....	262
表 3-373.函数 fmc_flag_get.....	262
表 3-374.函数 fmc_flag_clear.....	263
表 3-375.函数 fmc_bank0_state_get.....	264
表 3-376.函数 fmc_bank1_state_get.....	264
表 3-377.函数 fmc_bank0_ready_wait.....	265
表 3-378.函数 fmc_bank1_ready_wait.....	265
表 3-379. FWDGT 寄存器.....	266
表 3-380. FWDGT 库函数.....	266
表 3-381. 函数 fwdgt_write_enable.....	266
表 3-382. 函数 fwdgt_write_disable.....	267
表 3-383. 函数 fwdgt_enable.....	267
表 3-384. 函数 fwdgt_prescaler_value_config.....	268
表 3-385. 函数 fwdgt_reload_value_config.....	269
表 3-386. 函数 fwdgt_counter_reload.....	269
表 3-387. 函数 fwdgt_config.....	270
表 3-388. 函数 fwdgt_flag_get.....	271
表 3-389. GPIO 寄存器.....	272
表 3-390. GPIO 库函数.....	272
表 3-391. 函数 gpio_deinit.....	273

表 3-392. 函数 gpio_afio_deinit	273
表 3-393. 函数 gpio_mode_set	274
表 3-394. 函数 gpio_output_options_set	275
表 3-395. 函数 gpio_bit_set	276
表 3-396. 函数 gpio_bit_reset	276
表 3-397. 函数 gpio_bit_write	277
表 3-398. 函数 gpio_port_write	278
表 3-399. 函数 gpio_input_bit_get	278
表 3-400. 函数 gpio_input_port_get	279
表 3-401. 函数 gpio_output_bit_get	280
表 3-402. 函数 gpio_output_port_get	280
表 3-403. 函数 gpio_af_set	281
表 3-404. 函数 gpio_exti_source_select	282
表 3-405. 函数 gpio_pin_lock	282
表 3-406. HAU 寄存器	283
表 3-407. HAU 库函数	283
表 3-408. 结构体 hau_digest_parameter_struct	284
表 3-409. 函数 hau_deinit	284
表 3-410. 函数 hau_init	285
表 3-411. 函数 hau_reset	285
表 3-412. 函数 hau_last_word_validbits_num_config	286
表 3-413. 函数 hau_data_write	286
表 3-414. 函数 hau_infifo_words_num_get	287
表 3-415. 函数 hau_digest_read	287
表 3-416. 函数 hau_digest_calculation_enable	288
表 3-417. 函数 hau_multiple_single_dma_config	288
表 3-418. 函数 hau_dma_enable	289
表 3-419. 函数 hau_dma_disable	289
表 3-420. 函数 hau_hash_sha_256	290
表 3-421. 函数 hau_flag_get	290
表 3-422. 函数 hau_flag_clear	291
表 3-423. 函数 hau_interrupt_enable	291
表 3-424. 函数 hau_interrupt_disable	292
表 3-425. 函数 hau_interrupt_flag_get	293
表 3-426. 函数 hau_interrupt_flag_clear	293
表 3-427. I2C 寄存器	294
表 3-428. I2C 库函数	294
表 3-429. 枚举类型 i2c_interrupt_flag_enum	296
表 3-430. 函数 i2c_deinit	296
表 3-431. 函数 i2c_timing_config	297
表 3-432. 函数 i2c_digital_noise_filter_config	298
表 3-433. 函数 i2c_analog_noise_filter_enable	298
表 3-434. 函数 i2c_analog_noise_filter_disable	299
表 3-435. 函数 i2c_master_clock_config	299

表 3-436. 函数 i2c_master_addressing	300
表 3-437. 函数 i2c_address10_header_enable	301
表 3-438. 函数 i2c_address10_header_disable	301
表 3-439. 函数 i2c_address10_enable	302
表 3-440. 函数 i2c_address10_disable	302
表 3-441. 函数 i2c_automatic_end_enable	303
表 3-442. 函数 i2c_automatic_end_disable	303
表 3-443. 函数 i2c_slave_response_to_gcall_enable	304
表 3-444. 函数 i2c_slave_response_to_gcall_disable	304
表 3-445. 函数 i2c_stretch_scl_low_enable	305
表 3-446. 函数 i2c_stretch_scl_low_disable	305
表 3-447. 函数 i2c_address_config	306
表 3-448. 函数 i2c_address_bit_compare_config	307
表 3-449. 函数 i2c_address_disable	307
表 3-450. 函数 i2c_second_address_config	308
表 3-451. 函数 i2c_second_address_disable	309
表 3-452. 函数 i2c_received_address_get	309
表 3-453. 函数 i2c_slave_byte_control_enable	310
表 3-454. 函数 i2c_slave_byte_control_disable	310
表 3-455. 函数 i2c_nack_enable	311
表 3-456. 函数 i2c_enable	311
表 3-457. 函数 i2c_disable	312
表 3-458. 函数 i2c_start_on_bus	312
表 3-459. 函数 i2c_stop_on_bus	313
表 3-460. 函数 i2c_data_transmit	313
表 3-461. 函数 i2c_data_receive	314
表 3-462. 函数 i2c_reload_enable	314
表 3-463. 函数 i2c_reload_disable	315
表 3-464. 函数 i2c_transfer_byte_number_config	315
表 3-465. 函数 i2c_dma_enable	316
表 3-466. 函数 i2c_dma_disable	317
表 3-467. 函数 i2c_pec_transfer	317
表 3-468. 函数 i2c_pec_enable	318
表 3-469. 函数 i2c_pec_disable	318
表 3-470. 函数 i2c_pec_value_get	319
表 3-471. 函数 i2c_smbus_alert_enable	319
表 3-472. 函数 i2c_smbus_alert_disable	320
表 3-473. 函数 i2c_smbus_default_addr_enable	320
表 3-474. 函数 i2c_smbus_default_addr_disable	321
表 3-475. 函数 i2c_smbus_host_addr_enable	321
表 3-476. 函数 i2c_smbus_host_addr_disable	322
表 3-477. 函数 i2c_extended_clock_timeout_enable	322
表 3-478. 函数 i2c_extended_clock_timeout_disable	323
表 3-479. 函数 i2c_clock_timeout_enable	323

表 3-480. 函数 i2c_clock_timeout_disable	324
表 3-481. 函数 i2c_bus_timeout_b_config	324
表 3-482. 函数 i2c_bus_timeout_a_config	325
表 3-483. 函数 i2c_idle_clock_timeout_config	325
表 3-484. 函数 i2c_flag_get	326
表 3-485. 函数 i2c_flag_clear	327
表 3-486. 函数 i2c_interrupt_enable	328
表 3-487. 函数 i2c_interrupt_disable	328
表 3-488. 函数 i2c_interrupt_flag_get	329
表 3-489. 函数 i2c_interrupt_flag_clear	330
表 3-490. NVIC 寄存器	332
表 3-491. SysTick 寄存器	333
表 3-492. 枚举类型 IRQn_Type	333
表 3-493. MISC 库函数	335
表 3-494. 函数 nvic_priority_group_set	335
表 3-495. 函数 nvic_irq_enable	336
表 3-496. 函数 nvic_irq_disable	336
表 3-497. 函数 nvic_system_reset	337
表 3-498. 函数 nvic_vector_table_set	337
表 3-499. 函数 system_lowpower_set	338
表 3-500. 函数 system_lowpower_reset	339
表 3-501. 函数 systick_clksource_set	339
表 3-502. PMU 寄存器	340
表 3-503. PMU 库函数	340
表 3-504. 函数 pmu_deinit	341
表 3-505. 函数 pmu_lvd_select	343
表 3-506. 函数 pmu_pdrvs_select	344
表 3-507. 函数 pmu_ldo_output_select	345
表 3-508. 函数 pmu_lvd_enable	346
表 3-509. 函数 pmu_lvd_disable	346
表 3-510. 函数 pmu_lowdriver_mode_enable	347
表 3-511. 函数 pmu_lowdriver_mode_disable	348
表 3-512. 函数 pmu_lowpower_driver_config	348
表 3-513. 函数 pmu_normalpower_driver_config	349
表 3-514. 函数 pmu_to_sleepmode	350
表 3-515. 函数 pmu_to_deepsleepmode	350
表 3-516. 函数 pmu_to_standbymode	351
表 3-517. 函数 pmu_wakeup_pin_enable	352
表 3-518. 函数 pmu_wakeup_pin_disable	353
表 3-519. 函数 pmu_backup_write_enable	353
表 3-520. 函数 pmu_backup_write_disable	354
表 3-521. 函数 pmu_flag_get	360
表 3-522. 函数 pmu_flag_clear	361
表 3-523. RCU 寄存器	362

表 3-524. RCU 库函数.....	362
表 3-525. 枚举类型 rcu_periph_enum.....	364
表 3-526. 枚举类型 rcu_periph_sleep_enum.....	366
表 3-527. 枚举类型 rcu_periph_reset_enum	366
表 3-528. 枚举类型 rcu_flag_enum	367
表 3-529. 枚举类型 rcu_int_flag_enum.....	368
表 3-530. 枚举类型 rcu_int_flag_clear_enum.....	369
表 3-531. 枚举类型 rcu_int_enum	369
表 3-532. 枚举类型 rcu_osci_type_enum.....	370
表 3-533. 枚举类型 rcu_clock_freq_enum	370
表 3-534. 枚举类型 rcu_ckfm_enum	370
表 3-535. 函数 rcu_deinit.....	370
表 3-536. 函数 rcu_periph_clock_enable	371
表 3-537. 函数 rcu_periph_clock_disable	372
表 3-538. 函数 rcu_periph_clock_sleep_enable.....	373
表 3-539. 函数 rcu_periph_clock_sleep_disable.....	374
表 3-540. 函数 rcu_periph_reset_enable	374
表 3-541. 函数 rcu_periph_reset_disable	375
表 3-542. 函数 rcu_bkp_reset_enable	377
表 3-543. 函数 rcu_bkp_reset_disable	377
表 3-544. 函数 rcu_system_clock_source_config	378
表 3-545. 函数 rcu_system_clock_source_get.....	378
表 3-546. 函数 rcu_ahb_clock_config	379
表 3-547. 函数 rcu_apb1_clock_config.....	379
表 3-548. 函数 rcu_apb2_clock_config	380
表 3-549. 函数 rcu_ckout_config	381
表 3-550. 函数 rcu_pll0_config	382
表 3-551. 函数 rcu_pll1_config	382
表 3-552. 函数 rcu_prediv0_config.....	383
表 3-553. 函数 rcu_prediv1_config.....	384
表 3-554. 函数 rcu_adc_clock_config	384
表 3-555. 函数 rcu_usb_clock_config	385
表 3-556. 函数 rcu_rtc_clock_config	386
表 3-557. 函数 rcu_i2s1_clock_config	387
表 3-558. 函数 rcu_i2s2_clock_config	387
表 3-559. 函数 rcu_ck48m_clock_config	388
表 3-560. 函数 rcu_fmc_clock_config	389
表 3-561. 函数 rcu_lxtal_drive_capability_config	389
表 3-562. 函数 rcu_osci_stab_wait.....	390
表 3-563. 函数 rcu_osci_on.....	390
表 3-564. 函数 rcu_osci_off	391
表 3-565. 函数 rcu_osci_bypass_mode_enable	392
表 3-566. 函数 rcu_osci_bypass_mode_disable	392
表 3-567. 函数 rcu_irc8m_adjust_value_set	393

表 3-568. 函数 rcu_hxtal_clock_monitor_enable	393
表 3-569. 函数 rcu_hxtal_clock_monitor_disable	394
表 3-570. 函数 rcu_lxtal_clock_monitor_enable	394
表 3-571. 函数 rcu_lxtal_clock_monitor_disable	395
表 3-572. 函数 rcu_clock_freq_monitor_enable	395
表 3-573. 函数 rcu_clock_freq_monitor_disable	396
表 3-574. 函数 rcu_irc8m_freq_monitor_config	396
表 3-575. 函数 rcu_hxtal_monitor_threshold_config	397
表 3-576. 函数 rcu_pll0p_monitor_threshold_config	398
表 3-577. 函数 rcu_pll1_monitor_threshold_config	398
表 3-578. 函数 rcu_deepsleep_voltage_set	399
表 3-579. 函数 rcu_deepsleep_switch_delay_set	400
表 3-580. 函数 rcu_reg_lock	400
表 3-581. 函数 rcu_reg_unlock	401
表 3-582. 函数 rcu_pll_bandwidth_config	401
表 3-583. 函数 rcu_clock_freq_get	402
表 3-584. 函数 rcu_flag_get	402
表 3-585. 函数 rcu_flag_clear	403
表 3-586. 函数 rcu_all_reset_flag_clear	404
表 3-587. 函数 rcu_interrupt_enable	405
表 3-588. 函数 rcu_interrupt_disable	405
表 3-589. 函数 rcu_interrupt_flag_get	406
表 3-590. 函数 rcu_interrupt_flag_clear	407
表 3-591. RTC 寄存器	408
表 3-592. RTC 库函数	408
表 3-593. 函数 rtc_interrupt_enable	409
表 3-594. 函数 rtc_interrupt_disable	410
表 3-595. 函数 rtc_configuration_mode_enter	410
表 3-596. 函数 rtc_configuration_mode_exit	411
表 3-597. 函数 rtc_lwoff_wait	411
表 3-598. 函数 rtc_register_sync_wait	412
表 3-599. 函数 rtc_counter_get	412
表 3-600. 函数 rtc_counter_set	413
表 3-601. 函数 rtc_prescaler_set	413
表 3-602. 函数 rtc_alarm_config	414
表 3-603. 函数 rtc_divider_get	414
表 3-604. 函数 rtc_flag_get	415
表 3-605. 函数 rtc_flag_clear	416
表 3-606. SYSCFG 寄存器	416
表 3-607. SYSCFG 库函数	418
表 3-608. 枚举类型 timer_channel_input_enum	419
表 3-609. 函数 syscfg_deinit	419
表 3-610. 函数 syscfg_bootmode_get	419
表 3-611. 函数 syscfg_i2c_fast_mode_plus_enable	420

表 3-612. 函数 syscfg_i2c_fast_mode_plus_disable	421
表 3-613. 函数 syscfg_lockup_enable	421
表 3-614. 函数 syscfg_bus_timeout_enable	422
表 3-615. 函数 syscfg_bus_timeout_disable	422
表 3-616. 函数 syscfg_timer_input_source_select.....	423
表 3-617. 函数 syscfg_sram_page_wp_enable	424
表 3-618. 函数 syscfg_sram_ecc_error_bit_get.....	426
表 3-619. 函数 syscfg_sram_ecc_error_addr_get	426
表 3-620. 函数 syscfg_fpu_interrupt_enable	427
表 3-621. 函数 syscfg_fpu_interrupt_disable	428
表 3-622. 函数 syscfg_sram_ecc_interrupt_enable.....	428
表 3-623. 函数 syscfg_sram_ecc_interrupt_disable	429
表 3-624. 函数 syscfg_bus_timeout_flag_get.....	430
表 3-625. 函数 syscfg_bus_timeout_flag_clear.....	430
表 3-626. 函数 syscfg_sram_ecc_flag_get.....	431
表 3-627. 函数 syscfg_sram_ecc_flag_clear.....	431
表 3-628. SPI/I2S 寄存器	432
表 3-629. SPI/I2S 库函数	433
表 3-630. 结构体 spi_parameter_struct.....	434
表 3-631. 枚举类型 spi_interrupt_flag_enum	434
表 3-632. 函数 spi_i2s_deinit.....	435
表 3-633. 函数 spi_struct_para_init.....	435
表 3-634. 函数 spi_init.....	436
表 3-635. 函数 spi_enable	436
表 3-636. 函数 spi_disable	437
表 3-637. 函数 i2s_init	437
表 3-638. 函数 i2s_psc_config.....	438
表 3-639. 函数 i2s_enable.....	440
表 3-640. 函数 i2s_disable.....	440
表 3-641. 函数 spi_nss_output_enable.....	441
表 3-642. 函数 spi_nss_output_disable.....	441
表 3-643. 函数 spi_nss_internal_high	442
表 3-644. 函数 spi_nss_internal_low.....	442
表 3-645. 函数 spi_dma_enable	443
表 3-646. 函数 spi_dma_disable	443
表 3-647. 函数 spi_i2s_data_frame_format_config	444
表 3-648. 函数 spi_i2s_data_transmit	445
表 3-649. 函数 spi_i2s_data_receive	445
表 3-650. 函数 spi_bidirectional_transfer_config	446
表 3-651. 函数 spi_crc_polynomial_set	446
表 3-652. 函数 spi_crc_polynomial_get.....	447
表 3-653. 函数 spi_crc_on	448
表 3-654. 函数 spi_crc_off.....	448
表 3-655. 函数 spi_crc_next.....	449

表 3-656. 函数 spi_crc_get.....	449
表 3-657. 函数 spi_crc_error_clear.....	450
表 3-658. 函数 spi_ti_mode_enable.....	450
表 3-659. 函数 spi_ti_mode_disable.....	451
表 3-660. 函数 spi_nssp_mode_enable.....	451
表 3-661. 函数 spi_nssp_mode_disable.....	452
表 3-662. 函数 spi_quad_enable.....	452
表 3-663. 函数 spi_quad_disable.....	453
表 3-664. 函数 spi_quad_write_enable.....	453
表 3-665. 函数 spi_quad_read_enable.....	454
表 3-666. 函数 spi_i2s_flag_get.....	454
表 3-667. 函数 spi_i2s_interrupt_enable.....	455
表 3-668. 函数 spi_i2s_interrupt_disable.....	456
表 3-669. 函数 spi_i2s_interrupt_flag_get.....	456
表 3-670. TIMER 寄存器.....	458
表 3-671. TIMER 库函数.....	459
表 3-672. 结构体 timer_parameter_struct.....	462
表 3-673. 结构体 timer_break_parameter_struct.....	463
表 3-674. 结构体 timer_oc_parameter_struct.....	463
表 3-675. 结构体 timer_omc_parameter_struct.....	463
表 3-676. 结构体 timer_ic_parameter_struct.....	464
表 3-677. 函数 timer_deinit.....	464
表 3-678. 函数 timer_struct_para_init.....	464
表 3-679. 函数 timer_init.....	465
表 3-680. 函数 timer_enable.....	466
表 3-681. 函数 timer_disable.....	466
表 3-682. 函数 timer_auto_reload_shadow_enable.....	467
表 3-683. 函数 timer_auto_reload_shadow_disable.....	467
表 3-684. 函数 timer_update_event_enable.....	468
表 3-685. 函数 timer_update_event_disable.....	468
表 3-686. 函数 timer_counter_alignment.....	469
表 3-687. 函数 timer_counter_up_direction.....	470
表 3-688. 函数 timer_counter_down_direction.....	470
表 3-689. 函数 timer_prescaler_config.....	471
表 3-690. 函数 timer_repetition_value_config.....	471
表 3-691. 函数 timer_runtime_repetition_value_read.....	472
表 3-692. 函数 timer_autoreload_value_config.....	472
表 3-693. 函数 timer_autoreload_value_read.....	473
表 3-694. 函数 timer_counter_value_config.....	474
表 3-695. 函数 timer_counter_read.....	474
表 3-696. 函数 timer_prescaler_read.....	475
表 3-697. 函数 timer_single_pulse_mode_config.....	475
表 3-698. 函数 timer_update_source_config.....	476
表 3-699. 函数 timer_dma_enable.....	477

表 3-700. 函数 timer_dma_disable.....	478
表 3-701. 函数 timer_channel_dma_request_source_select	479
表 3-702. 函数 timer_dma_transfer_config	479
表 3-703. 函数 timer_event_software_generate	482
表 3-704. 函数 timer_break_struct_para_init	483
表 3-705. 函数 timer_break_config	484
表 3-706. 函数 timer_break_enable	484
表 3-707. 函数 timer_break_disable	485
表 3-708. 函数 timer_automatic_output_enable.....	485
表 3-709. 函数 timer_automatic_output_disable.....	486
表 3-710. 函数 timer_primary_output_config	487
表 3-711. 函数 timer_channel_control_shadow_config	487
表 3-712. 函数 timer_channel_control_shadow_update_config	488
表 3-713. 函数 timer_channel_output_struct_para_init.....	489
表 3-714. 函数 timer_channel_output_config.....	489
表 3-715. 函数 timer_channel_output_mode_config.....	490
表 3-716. 函数 timer_channel_output_pulse_value_config	491
表 3-717. 函数 timer_channel_output_shadow_config.....	492
表 3-718. 函数 timer_channel_output_compare_fast_config	493
表 3-719. 函数 timer_channel_output_clear_config	494
表 3-720. 函数 timer_channel_output_polarity_config	495
表 3-721. 函数 timer_channel_complementary_output_polarity_config	496
表 3-722. 函数 timer_channel_output_state_config	497
表 3-723. 函数 timer_channel_complementary_output_state_config	498
表 3-724. 函数 timer_channel_composite_asymmetric_pwm_level_config	499
表 3-725. 函数 timer_channel_output_clear_invalid_selection	499
表 3-726. 函数 timer_channel_input_struct_para_init	500
表 3-727. 函数 timer_input_capture_config	501
表 3-728. 函数 timer_channel_input_capture_prescaler_config.....	502
表 3-729. 函数 timer_channel_capture_value_register_read.....	502
表 3-730. 函数 timer_input_pwm_capture_config	503
表 3-731. 函数 timer_hall_mode_config	504
表 3-732. 函数 timer_multi_mode_channel_output_parameter_struct_init	505
表 3-733. 函数 timer_multi_mode_channel_output_config.....	505
表 3-734. 函数 timer_multi_mode_channel_mode_config	506
表 3-735. 函数 timer_input_trigger_source_select.....	507
表 3-736. 函数 timer_master_output_trigger_source_select.....	508
表 3-737. 函数 timer_slave_mode_select	509
表 3-738. 函数 timer_master_slave_mode_config	510
表 3-739. 函数 timer_external_trigger_config	511
表 3-740. 函数 timer_quadrature_decoder_mode_config	512
表 3-741. 函数 timer_non_quadrature_decoder_mode_config	513
表 3-742. 函数 timer_internal_clock_config.....	514
表 3-743. 函数 timer_internal_trigger_as_external_clock_config.....	515

表 3-744. 函数 timer_external_trigger_as_external_clock_config	516
表 3-745. 函数 timer_external_clock_mode0_config	517
表 3-746. 函数 timer_external_clock_mode1_config	518
表 3-747. 函数 timer_external_clock_mode1_disable	519
表 3-748. 函数 timer_write_chxval_register_config	519
表 3-749. 函数 timer_output_value_selection_config	520
表 3-750. 函数 timer_commutation_control_shadow_register_config	521
表 3-751. 函数 timer_output_match_pulse_select	521
表 3-752. 函数 timer_channel_composite_pwm_mode_config	522
表 3-753. 函数 timer_channel_composite_pwm_mode_output_pulse_value_config	523
表 3-754. 函数 timer_channel_asymmetric_pwm_pulse_config	524
表 3-755. 函数 timer_channel_additional_compare_value_config	525
表 3-756. 函数 timer_channel_additional_output_update_select	525
表 3-757. 函数 timer_channel_additional_output_shadow_config	526
表 3-758. 函数 timer_channel_additional_compare_value_read	527
表 3-759. 函数 timer_break_external_source_config	528
表 3-760. 函数 timer_break_external_polarity_config	528
表 3-761. 函数 timer_dead_time_falling_edge_config	529
表 3-762. 函数 timer_dead_time_different_config	530
表 3-763. 函数 timer_channel_break_control_config	530
表 3-764. 函数 timer_channel_dead_time_config	531
表 3-765. 函数 timer_channel_break_config	532
表 3-766. 函数 timer_channel_break_external_status_config	533
表 3-767. 函数 timer_channel_break_external_polarity_config	534
表 3-768. 函数 timer_channel_primary_output_config	534
表 3-769. 函数 timer_channel_break_period_config	535
表 3-770. 函数 timer_watchdog_value_config	536
表 3-771. 函数 timer_watchdog_value_read	536
表 3-772. 函数 timer_decoder_disconnection_detection_config	537
表 3-773. 函数 timer_decoder_jump_detection_config	537
表 3-774. 函数 timer_counter_initial_register_config	538
表 3-775. 函数 timer_counter_initial_config	539
表 3-776. 函数 timer_synchronization_event_generate	539
表 3-777. 函数 timer_flag_get	540
表 3-778. 函数 timer_flag_clear	542
表 3-779. 函数 timer_interrupt_enable	543
表 3-780. 函数 timer_interrupt_disable	545
表 3-781. 函数 timer_interrupt_flag_get	546
表 3-782. 函数 timer_interrupt_flag_clear	548
表 3-783. TRIGSEL 寄存器	550
表 3-784. TRIGSEL 库函数	551
表 3-785. 枚举类型 trigselsource_enum	551
表 3-786. 枚举类型 trigselperiph_enum	553
表 3-787. 函数 trigseldeinit	554

表 3-788. 函数 trngsel_init.....	555
表 3-789. 函数 trngsel_trigger_source_get	555
表 3-790. 函数 trngsel_trigger_source_set	556
表 3-791. 函数 trngsel_trigger_lock_get	556
表 3-792. TRNG 寄存器	557
表 3-793. TRNG 库函数	557
表 3-794. 枚举 trng_flag_enum.....	557
表 3-795. 枚举 trng_int_flag_enum.....	557
表 3-796. 函数 trng_deinit	558
表 3-797. 函数 trng_enable	558
表 3-798. 函数 trng_disable	559
表 3-799. 函数 trng_get_true_random_data.....	559
表 3-800. 函数 trng_powermode_config.....	560
表 3-801. 函数 trng_flag_get.....	560
表 3-802. 函数 trng_interrupt_enable.....	561
表 3-803. 函数 trng_interrupt_disable.....	561
表 3-804. 函数 trng_interrupt_flag_get	562
表 3-805. 函数 trng_interrupt_flag_clear	562
表 3-806. USART 寄存器	563
表 3-807. USART 库函数.....	563
表 3-808. 枚举类型 usart_flag_enum	565
表 3-809. 枚举类型 usart_interrupt_flag_enum	565
表 3-810. 枚举类型 usart_interrupt_enum	566
表 3-811. 枚举类型 usart_invert_enum.....	566
表 3-812. 函数 usart_deinit	566
表 3-813. 函数 usart_baudrate_set.....	567
表 3-814. 函数 usart_parity_config.....	567
表 3-815. 函数 usart_word_length_set	568
表 3-816. 函数 usart_stop_bit_set	569
表 3-817. 函数 usart_enable.....	569
表 3-818. 函数 usart_disable.....	570
表 3-819. 函数 usart_transmit_config	570
表 3-820. 函数 usart_receive_config	571
表 3-821. 函数 usart_data_first_config	572
表 3-822. 函数 usart_invert_config.....	572
表 3-823. 函数 usart_oversample_config	573
表 3-824. 函数 usart_sample_bit_config	574
表 3-825. 函数 usart_receiver_timeout_enable	574
表 3-826. 函数 usart_receiver_timeout_disable	575
表 3-827. 函数 usart_receiver_timeout_threshold_config	575
表 3-828. 函数 usart_data_transmit.....	576
表 3-829. 函数 usart_data_receive.....	577
表 3-830. 函数 usart_address_config.....	577
表 3-831. 函数 usart_mute_mode_enable	578

表 3-832. 函数 usart_mute_mode_disable	578
表 3-833. 函数 usart_mute_mode_wakeup_config.....	579
表 3-834. 函数 usart_lin_mode_enable.....	579
表 3-835. 函数 usart_lin_mode_disable.....	580
表 3-836. 函数 usart_lin_break_detection_length_config	580
表 3-837. 函数 usart_send_break	581
表 3-838. 函数 usart_halfduplex_enable.....	582
表 3-839. 函数 usart_halfduplex_disable	582
表 3-840. 函数 usart_synchronous_clock_enable	583
表 3-841. 函数 usart_synchronous_clock_disable.....	583
表 3-842. 函数 usart_synchronous_clock_config.....	584
表 3-843. 函数 usart_guard_time_config.....	584
表 3-844. 函数 usart_smartcard_mode_enable	585
表 3-845. 函数 usart_smartcard_mode_disable	586
表 3-846. 函数 usart_smartcard_mode_nack_enable	586
表 3-847. 函数 usart_smartcard_mode_nack_disable	587
表 3-848. 函数 usart_smartcard_autoretry_config	587
表 3-849. 函数 usart_block_length_config.....	588
表 3-850. 函数 usart_irda_mode_enable	588
表 3-851. 函数 usart_irda_mode_disable	589
表 3-852. 函数 usart_prescaler_config	589
表 3-853. 函数 usart_irda_lowpower_config.....	590
表 3-854. 函数 usart_hardware_flow_rts_config.....	590
表 3-855. 函数 usart_hardware_flow_cts_config	591
表 3-856. 函数 usart_break_frame_coherence_config.....	592
表 3-857. 函数 usart_parity_check_coherence_config.....	592
表 3-858. 函数 usart_hardware_flow_coherence_config	593
表 3-859. 函数 usart_dma_receive_config	594
表 3-860. 函数 usart_dma_transmit_config	594
表 3-861. 函数 usart_flag_get.....	595
表 3-862. 函数 usart_flag_clear.....	596
表 3-863. 函数 usart_interrupt_enable	596
表 3-864. 函数 usart_interrupt_disable.....	597
表 3-865. 函数 usart_interrupt_flag_get	598
表 3-866. 函数 usart_interrupt_flag_clear	598
表 3-867. WWDGT 寄存器.....	599
表 3-868. WWDGT 库函数.....	599
表 3-869. 函数 wwdgt_deinit.....	600
表 3-870. 函数 wwdgt_enable	600
表 3-871. 函数 wwdgt_counter_update.....	601
表 3-872. 函数 wwdgt_config.....	601
表 3-873. 函数 wwdgt_flag_get.....	603
表 3-874. 函数 wwdgt_flag_clear	603
表 3-875. 函数 wwdgt_interrupt_enable	604

表 4-1. 版本历史	605
-------------------	-----

1. 介绍

本手册介绍了32位基于ARM微控制器GD32F50x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GD32F50x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	控制器局域网络
CAU	加密处理器
CMP	比较器
CRC	循环冗余校验计算单元
CTC	时钟校准控制器
DAC	数模转换器
DBG	调试模块
DMA	直接存储器访问控制器
DMAMUX	DMA请求多路复用器
EXMC	外部存储器控制器

外设缩写	说明
EXTI	外部中断事件控制器
FMC	闪存控制器
FWDGT	独立看门狗
GPIO	通用和备用输入
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
SPI/I2S	串行外设接口/片上音频接口
SYSCFG	系统配置
TIMER	定时器
TRIGSEL	触发选择控制器
TRNG	真随机数生成器
USART	通用同步异步收发器
USBFS	通用串行总线全速接口
WWDGT	窗口看门狗

1.1.2. 命名规则

固件库遵从以下命名规则：

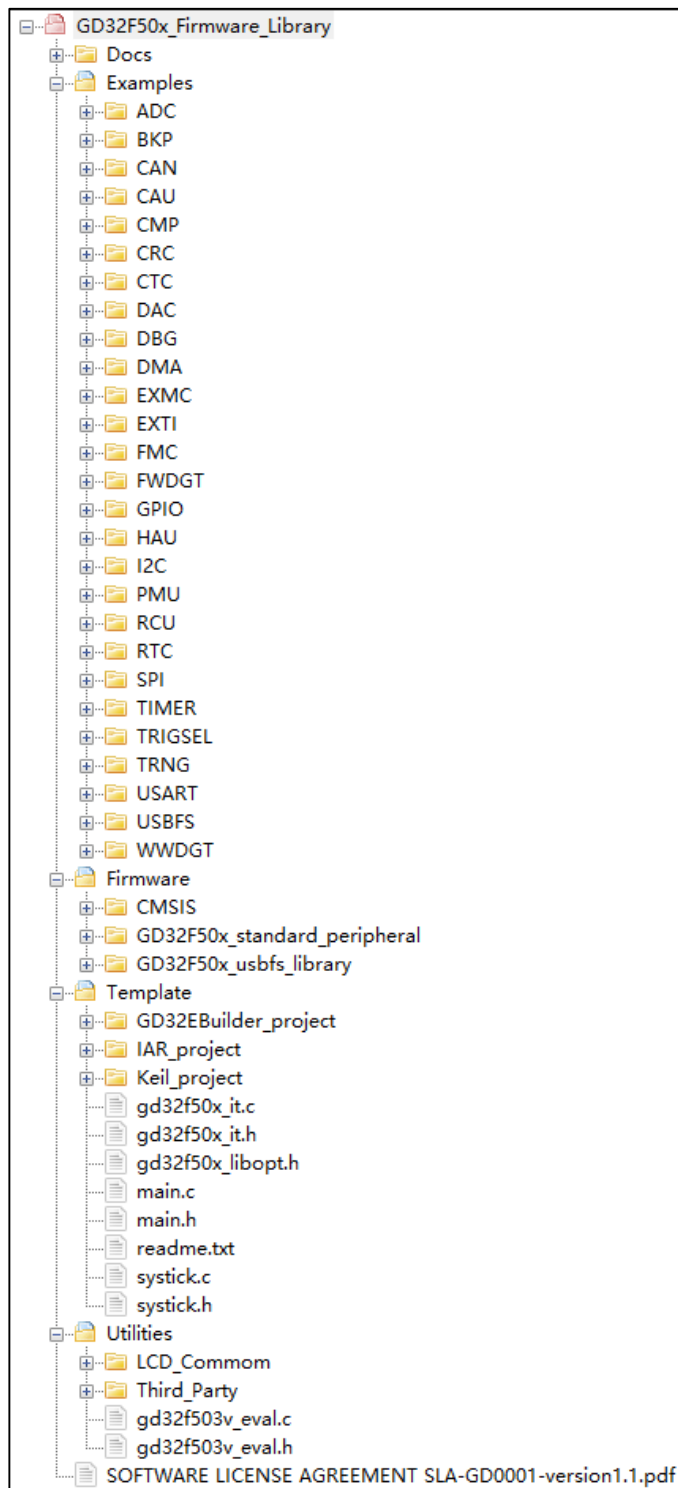
- XXX表示任一外设缩写，例如：ADC。更多缩写相关信息参阅[外设缩写](#)；
- 源文件和头文件命名都以“gd32f50x_”作为开头，例如：gd32f50x_adc.h；
- 常量仅被应用于一个文件的，定义于该文件中；被应用于多个文件的，在对应头文件中定义。所有常量都由英文字母大写书写；
- 寄存器作为常量处理。他们的命名都由英文字母大写书写。在大多数情况下，寄存器缩写规范与本用户手册一致；
- 变量名采用全部小写，有多个单词组成的，在单词之间以下划线分隔；
- 外设函数的命名以该外设的缩写加下划线为开头，有多个单词组成的，在单词之间以下划线分隔，所有外设函数都由英文字母小写书写。

2. 固件库概述

2.1. 文件组织结构

GD32F50x_Firmware_Library，文件组织结构见下图：

图 2-1. GD32F50x 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；
- **gd32f50x_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）；
- **gd32f50x_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）；
- **gd32f50x_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® M33内核的支持文件、基于Cortex® M33内核处理器的启动代码和库引导文件以及基于GD32F50x的全局头文件和系统配置文件；
- **GD32F50x_standard_peripheral**子文件夹：
 - **Include**子文件夹包含了固件函数库所需的头文件，用户无需修改该文件夹；
 - **Source**子文件夹包含了固件函数库所需的源文件，用户无需修改该文件夹；

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

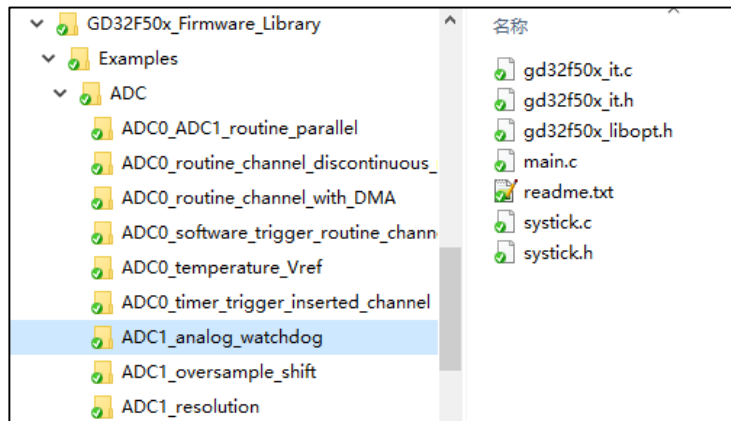
2.1.3. Template 文件夹

Template文件夹包含一个关于使用LED、USART打印、按键控制的简单例程，（**IAR_project**用于IAR编译环境，**Keil_project**用于Keil5编译环境，**GD32EBuilder_project**用于编译GD32EBuilder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ADC，打开“**ADC**”文件夹，选择ADC的一个例程，如“**ADC1_analog_watchdog**”，如下图所示：

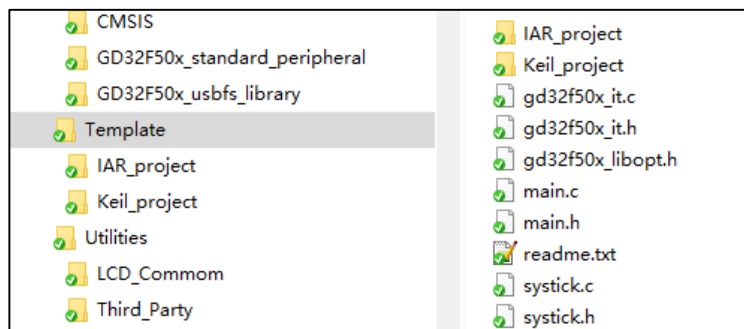
图 2-2. 选择外设例程文件



拷贝文件

打开“Template”文件夹，将“IAR_project”、“Keil_project”和“GD32EBuilder_project”三个文件夹保留，其他文件都删除，然后将“ADC1_analog_watchdog0”文件夹中的所有文件拷到“Template”文件夹子目录下，如下图所示：

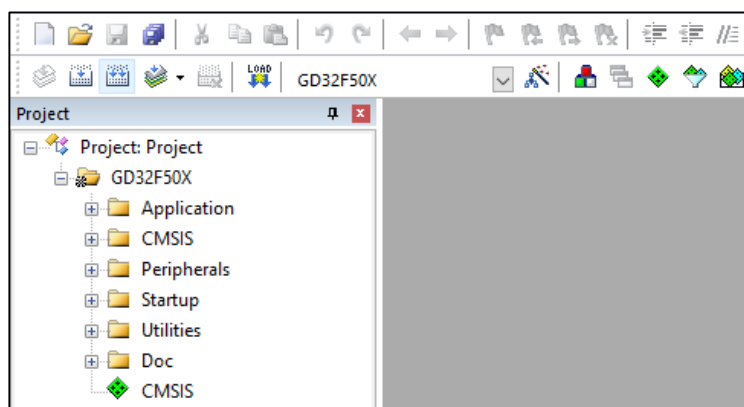
图 2-3. 拷贝外设例程文件



打开工程

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

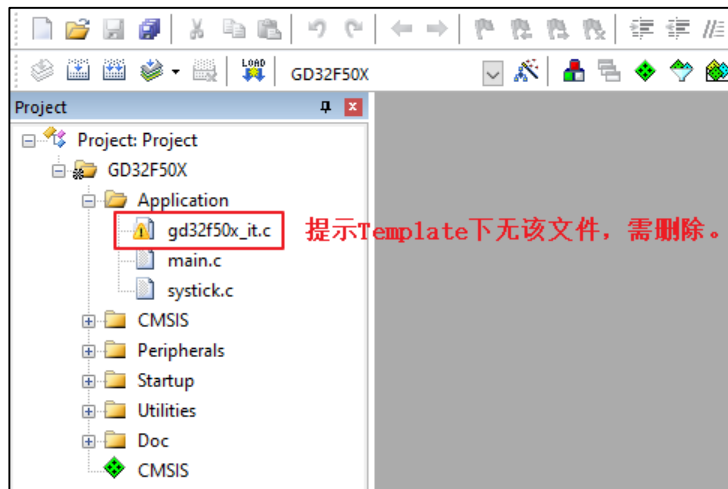
图 2-4. 打开工程文件



由于不同的模块、不同的功能，会使用到不同的文件，需要根据客户选择拷贝的文件，对工程

里的文件进行增加或删除，如下图所示：

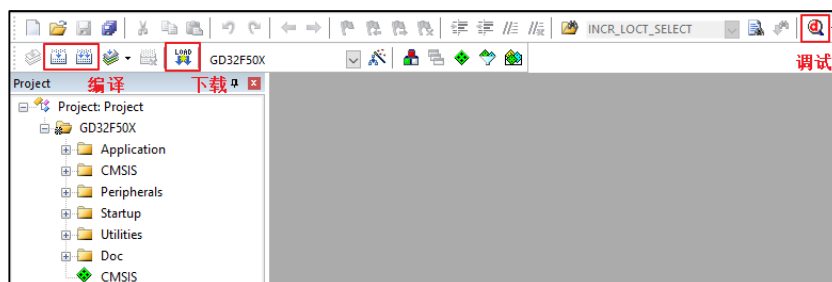
图 2-5. 配置工程文件



编译调试下载

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

图 2-6. 编译调试下载



2.1.4. Utilities 文件夹

Utilities文件夹包含运行固件库例程评估板的文件：

- gd32f503v_eval.h文件是运行固件库例程所需关于评估板的头文件；
- gd32f503v_eval.c文件是运行固件库例程所需关于评估板的源文件。

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

2.2. 固件库文件描述

下表列举和描述了固件库使用的主要文件。

表 2-1. 固件函数库文件描述

文件名	描述
-----	----

gd32f50x_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。
gd32f50x_it.h	头文件，包含所有中断处理函数原形。
gd32f50x_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。
gd32f50x_xxx.h	外设xxx的头文件。包含外设xxx函数的定义，以及这些函数使用的变量。
gd32f50x_xxx.c	由C语言编写的外设xxx的驱动源程序文件。
systick.h	systick.c的头文件。包含systick配置函数的定义，以及外部用延时函数的定义。
systick.c	systick配置与延时函数源文件。
readme.txt	固件库例程使用及配置说明文档。

3. 外设固件库

3.1. 外设固件库概述

外设固件库函数的描述格式如下表：

表 3-1. 外设固件库函数描述格式

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

3.2. ADC

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

3.2.1. 外设寄存器描述

ADC寄存器列表如下表所示：

表 3-2. ADC 寄存器

寄存器名称	寄存器描述
ADC_STAT	状态寄存器
ADC_CTL0	控制寄存器0
ADC_CTL1	控制寄存器1
ADC_SAMPT0	采样时间寄存器0
ADC_SAMPT1	采样时间寄存器1
ADC_IOFFx	注入通道数据偏移寄存器x (x=0..3)
ADC_WD0HT	看门狗0高阈值寄存器
ADC_WD0LT	看门狗0低阈值寄存器
ADC_RSQ0	常规序列寄存器0
ADC_RSQ1	常规序列寄存器1
ADC_RSQ2	常规序列寄存器2

寄存器名称	寄存器描述
ADC_ISQ	注入序列寄存器
ADC_LDATABx(x=0..3)	锁存数据寄存器x (x=0..3)
ADC_RDATA	常规数据寄存器
ADC_IDATA	注入数据寄存器
ADC_LDCTL	锁存数据控制寄存器
ADC_OVSAMPCTL	过采样控制寄存器

3.2.2. 外设库函数说明

ADC库函数列表如下表所示：

表 3-3. ADC 库函数

库函数名称	库函数描述
adc_deinit	复位ADC外设
adc_enable	使能ADC外设
adc_disable	禁能ADC外设
adc_dma_mode_enable	使能ADC DMA请求
adc_dma_mode_disable	禁能ADC DMA请求
adc_discontinuous_mode_config	配置ADC间断模式
adc_special_function_config	使能或禁能ADC特殊功能
adc_internal_channel_config	使能或禁能ADC内部通道
adc_sync_mode_config	ADC同步模式配置
adc_sync_routine_data_read	同步模式下读取ADC0和ADC1常规序列数据
adc_data_alignment_config	配置ADC数据对齐方式
adc_channel_length_config	配置常规序列或注入序列的长度
adc_routine_channel_config	配置ADC常规序列
adc_inserted_channel_config	配置ADC注入序列
adc_inserted_channel_offset_config	配置ADC注入序列数据偏移值
adc_external_trigger_config	配置ADC外部触发
adc_software_trigger_enable	ADC软件触发使能
adc_routine_data_read	读ADC常规数据寄存器
adc_inserted_data_read	读ADC注入数据寄存器
adc_latch_data_read	读ADC锁存数据寄存器
adc_latch_data_source_config	配置ADC锁存数据源
adc_watchdog0_single_channel_enable	配置ADC模拟看门狗0单通道有效
adc_watchdog0_sequence_channel_enable	配置ADC模拟看门狗0在通道序列有效
adc_watchdog0_disable	ADC模拟看门狗0禁能
adc_watchdog0_threshold_config	配置ADC模拟看门狗0阈值
adc_resolution_config	配置ADC分辨率
adc_oversample_mode_config	配置ADC过采样模式

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

函数 adc_deinit

函数adc_deinit描述见下表：

表 3-4. 函数 adc_deinit

函数名称	adc_deinit
函数原形	void adc_deinit(uint32_t adc_periph);
功能描述	复位ADC外设
先决条件	-
被调用函数	rcu_periph_reset_enable / rcu_periph_reset_disable
输入参数{in}	
adc_periph	ADC外设
ADCx(x=0,1,2)	ADC外设选择
输出参数{out}	
-	-
返回值	
-	-

例如：

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

函数 adc_enable

函数adc_enable描述见下表：

表 3-5. 函数 adc_enable

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

ADCx(x=0,1,2)	ADC外设选择
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* enable ADC0 */
```

```
adc_enable(ADC0);
```

函数 adc_disable

函数adc_disable描述见下表:

表 3-6. 函数 adc_disable

函数名称	adc_disable
函数原形	void adc_disable(uint32_t adc_periph);
功能描述	禁能ADC外设
先决条件	-
被调用函数	-
输入参数{in}	
adc_periph	ADC外设
ADCx(x=0,1,2)	ADC外设选择
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* disable ADC0 */
```

```
adc_disable(ADC0);
```

函数 adc_dma_mode_enable

函数adc_dma_mode_enable描述见下表:

表 3-7. 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, uint8_t adc_sequence,);
功能描述	使能ADC DMA请求
先决条件	-
被调用函数	-
输入参数{in}	
adc_periph	ADC外设

ADCx(x=0,1,2)	ADC外设选择
输入参数{in}	
adc_sequence	序列选择
ADC_ROUTINE_CHANNEL	常规序列
ADC_INSERTED_CHANNEL	注入序列
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* enable ADC routine sequence DMA request */
```

```
adc_dma_mode_enable(ADC0, ADC_ROUTINE_CHANNEL);
```

函数 adc_dma_mode_disable

函数adc_dma_mode_disable描述见下表：

表 3-8. 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, uint8_t adc_sequence);
功能描述	禁能ADC DMA请求
先决条件	-
被调用函数	-
输入参数{in}	
adc_periph	ADC外设
ADCx(x=0,1,2)	ADC外设选择
输入参数{in}	
adc_sequence	序列选择
ADC_ROUTINE_CHANNEL	常规序列
ADC_INSERTED_CHANNEL	注入序列
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* disable ADC routine sequence DMA request */
```

```
adc_dma_mode_disable(ADC0, ADC_ROUTINE_CHANNEL);
```

函数 `adc_discontinuous_mode_config`

函数`adc_discontinuous_mode_config`描述见下表:

表 3-9. 函数 `adc_discontinuous_mode_config`

函数名称	<code>adc_discontinuous_mode_config</code>
函数原形	<code>void adc_discontinuous_mode_config(uint32_t adc_periph, uint8_t adc_sequence, uint8_t length);</code>
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}	
<code>adc_periph</code>	ADC外设
<code>ADCx(x=0,1,2)</code>	ADC外设选择
输入参数{in}	
<code>adc_sequence</code>	序列选择
<code>ADC_ROUTINE_CHANNEL</code>	常规序列
<code>ADC_INSERTED_CHANNEL</code>	注入序列
<code>ADC_CHANNEL_DISCONT_DISABLE</code>	常规序列和注入序列间断模式禁能
输入参数{in}	
<code>length</code>	间断模式下的转换数目, 常规序列取值为1..8, 注入序列取值无意义
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* configure ADC0 discontinuous mode */
adc_discontinuous_mode_config(ADC0, ADC_ROUTINE_CHANNEL, 6);
```

函数 `adc_special_function_config`

函数`adc_special_function_config`描述见下表:

表 3-10. 函数 `adc_special_function_config`

函数名称	<code>adc_special_function_config</code>
函数原形	<code>void adc_special_function_config(uint32_t adc_periph, uint32_t function, ControlStatus newvalue);</code>
功能描述	使能或禁能ADC特殊功能
先决条件	-
被调用函数	-

输入参数{in}	
adc_periph	ADC外设
<i>ADCx(x=0,1,2)</i>	ADC外设选择
输入参数{in}	
function	功能配置
<i>ADC_SCAN_MODE</i>	扫描模式选择
<i>ADC_INSERTED_CHANNEL_AUTO</i>	注入序列自动转换
<i>ADC_CONTINUOUS_MODE</i>	连续模式选择
输入参数{in}	
newvalue	功能使能禁能
<i>ENABLE</i>	使能
<i>DISABLE</i>	禁能
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* enable ADC0 scan mode */
```

```
adc_special_function_config(ADC0, ADC_SCAN_MODE, ENABLE);
```

函数 **adc_internal_channel_config**

函数adc_internal_channel_config描述见下表：

表 3-11. 函数 **adc_internal_channel_config**

函数名称	adc_internal_channel_config
函数原形	void adc_internal_channel_config(uint32_t internal_channel, ControlStatus newvalue);
功能描述	使能或禁能ADC内部通道
先决条件	-
被调用函数	-
输入参数{in}	
internal_channel	ADC外设
<i>ADC_CHANNEL_INTERNAL_TEMPSENSOR</i>	温度传感器通道
<i>ADC_CHANNEL_INTERNAL_VREFINT</i>	内部参考电压通道
输入参数{in}	
newvalue	功能使能禁能
<i>ENABLE</i>	使能

DISABLE	禁能
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* enable ADC temperature sensor channel */
```

```
adc_internal_channel_config (ADC_CHANNEL_INTERNAL_TEMPSENSOR, ENABLE);
```

函数 adc_sync_mode_config

函数adc_sync_mode_config描述见下表:

表 3-12. 函数 adc_sync_mode_config

函数名称	adc_sync_mode_config
函数原形	void adc_sync_mode_config(uint32_t sync_mode);
功能描述	ADC同步模式配置
先决条件	-
被调用函数	-
输入参数{in}	
sync_mode	同步模式
ADC_MODE_FREE	ADC同步模式失能, 所有的ADC都独立工作
ADC_DUAL_ROUTINE_PARALLEL_INSERTED_PARALLEL	ADC0和ADC1工作在常规并行和注入并行的组合模式
ADC_DUAL_ROUTINE_PARALLEL_INSERTED_ROTATION	ADC0和ADC1工作在常规并行和注入交替触发的组合模式
ADC_DUAL_INSERTED_PARALLEL_ROUTINE_FOLLOW_UP_FAST	ADC0和ADC1工作在注入并行和快速交叉组合模式
ADC_DUAL_INSERTED_PARALLEL_ROUTINE_FOLLOW_UP_SLOW	ADC0和ADC1工作在常规并行和慢速交叉组合模式
ADC_DUAL_INSERTED_PARALLEL	ADC0和ADC1工作在注入并行模式
ADC_DUAL_ROUTINE_PARALLEL	ADC0和ADC1工作在常规并行模式
ADC_DUAL_ROUTINE_FOLLOW_UP_FAST	ADC0和ADC1工作在快速交叉组合模式

NE_FOLLOWUP_F AST	
ADC_DAUL_ROUTINE_FOLLOWUP_S LOW	ADC0和ADC1工作在慢速交叉组合模式
ADC_DAUL_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：

```
/* ADC0 and ADC1 work in combined routine parallel & inserted parallel mode */
adc_sync_mode_config(ADC_DAUL_ROUTINE_PARALLEL_INSERTED_PARALLEL);
```

函数 adc_sync_routine_data_read

函数adc_sync_routine_data_read描述见下表：

表 3-13. 函数 adc_sync_routine_data_read

函数名称	adc_sync_routine_data_read
函数原形	uint32_t adc_sync_routine_data_read(void);
功能描述	同步模式下读取ADC0和ADC1常规序列数据
先决条件	-
被调用函数	-
输入参数{in}	
-	-
输出参数{out}	
-	-
返回值	
uint32_t	ADC转换值

例如：

```
/* read the last ADC0 and ADC1 conversion result data in sync mode */
uint32_t adc_value = 0;
adc_value = adc_sync_routine_data_read();
```

函数 adc_data_alignment_config

函数adc_data_alignment_config描述见下表：

表 3-14. 函数 `adc_data_alignment_config`

函数名称	<code>adc_data_alignment_config</code>
函数原形	<code>void adc_data_alignment_config(uint32_t adc_periph, uint32_t data_alignment);</code>
功能描述	配置ADC数据对齐方式
先决条件	-
被调用函数	-
输入参数{in}	
<code>adc_periph</code>	ADC外设
<code>ADCx(x=0,1,2)</code>	ADC外设选择
输入参数{in}	
<code>data_alignment</code>	数据对齐方式选择
<code>ADC_DATAALIGN_RIGHT</code>	右对齐
<code>ADC_DATAALIGN_LEFT</code>	左对齐
输出参数{out}	
-	-
返回值	
-	-

例如：

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

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

函数 `adc_channel_length_config`

函数`adc_channel_length_config`描述见下表：

表 3-15. 函数 `adc_channel_length_config`

函数名称	<code>adc_channel_length_config</code>
函数原形	<code>adc_channel_length_config(uint32_t adc_periph, uint8_t adc_sequence, uint32_t length);</code>
功能描述	配置常规序列或注入序列的长度
先决条件	-
被调用函数	-
输入参数{in}	
<code>adc_periph</code>	ADC外设
<code>ADCx(x=0,1,2)</code>	ADC外设选择
输入参数{in}	
<code>adc_sequence</code>	序列选择
<code>ADC_ROUTINE_CHANNEL</code>	常规序列
<code>ADC_INSERTED_CHANNEL</code>	注入序列

HANNEL	
输入参数{in}	
length	通道长度，常规序列为1-16，注入序列为1-4
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* configure the length of ADC0 routine channel */
```

```
adc_channel_length_config(ADC0, ADC_ROUTINE_CHANNEL, 4);
```

函数 adc_routine_channel_config

函数adc_routine_channel_config描述见下表：

表 3-16. 函数 adc_routine_channel_config

函数名称	adc_routine_channel_config
函数原形	void adc_routine_channel_config(uint32_t adc_periph, uint8_t rank, uint8_t adc_channel, uint32_t sample_time)
功能描述	配置ADC常规序列
先决条件	-
被调用函数	-
输入参数{in}	
adc_periph	ADC外设
ADCx(x=0,1,2)	ADC外设选择
输入参数{in}	
rank	常规通道序列，取值范围为0~15
输入参数{in}	
adc_channel	ADC通道选择
ADC_CHANNEL_x	ADC_CHANNEL_x: ADC0 x=0..17, ADC1 x=0..17, ADC2 x=0..16
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

_55POINT5	
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 routine channel */
```

```
adc_routine_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 3-17. 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注入序列
先决条件	-
被调用函数	-
输入参数{in}	
adc_periph	ADC外设
ADCx(x=0,1,2)	ADC外设选择
输入参数{in}	
rank	注入通道序列，取值范围为0~3
输入参数{in}	
adc_channel	ADC通道选择
ADC_CHANNEL_x	ADC_CHANNEL_x: ADC0 x=0..17, ADC1 x=0..17, ADC2 x=0..16
输入参数{in}	
sample_time	采样时间
ADC_SAMPLETIME _1POINT5	1.5周期
ADC_SAMPLETIME _7POINT5	7.5周期
ADC_SAMPLETIME _13POINT5	13.5周期
ADC_SAMPLETIME _28POINT5	28.5周期
ADC_SAMPLETIME	41.5周期

_41POINT5	
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 3-18. 函数 adc_inserted_channel_offset_config

函数名称	adc_inserted_channel_offset_config
函数原形	void adc_inserted_channel_offset_config(uint32_t adc_periph, uint8_t inserted_channel, uint16_t offset)
功能描述	配置ADC注入序列数据偏移值
先决条件	-
被调用函数	-
输入参数{in}	
adc_periph	ADC外设
ADCx(x=0,1,2)	ADC外设选择
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_config

函数adc_external_trigger_config描述见下表:

表 3-19. 函数 adc_external_trigger_config

函数名称	adc_external_trigger_config
函数原形	void adc_external_trigger_config(uint32_t adc_periph, uint8_t adc_sequence, uint32_t trigger_mode)
功能描述	配置ADC外部触发
先决条件	-
被调用函数	-
输入参数{in}	
adc_periph	ADC外设
ADCx(x=0,1,2)	ADC外设选择
输入参数{in}	
adc_sequence	序列选择
ADC_ROUTINE_CHANNEL	常规序列
ADC_INSERTED_CHANNEL	注入序列
输入参数{in}	
trigger_mode	选择触发模式
EXTERNAL_TRIGGER_DISABLE	外部触发禁能
EXTERNAL_TRIGGER_RISING	外部触发的上升沿
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* enable ADC0 inserted sequence external trigger */
```

```
adc_external_trigger_config(ADC0, ADC_INSERTED_CHANNEL, EXTERNAL_TRIGGER_RISING);
```

函数 adc_software_trigger_enable

函数adc_software_trigger_enable描述见下表:

表 3-20. 函数 adc_software_trigger_enable

函数名称	adc_software_trigger_enable
函数原形	void adc_software_trigger_enable(uint32_t adc_periph, uint8_t adc_sequence);

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)	ADC外设选择
输入参数{in}	
adc_sequence	序列选择
ADC_ROUTINE_CHANNEL	常规序列
ADC_INSERTED_CHANNEL	注入序列
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* enable ADC routine sequence software trigger */
```

```
adc_software_trigger_enable(ADC0, ADC_ROUTINE_CHANNEL);
```

函数 adc_routine_data_read

函数adc_routine_data_read描述见下表：

表 3-21. 函数 adc_routine_data_read

函数名称	adc_routine_data_read
函数原形	uint16_t adc_routine_data_read(uint32_t adc_periph);
功能描述	读ADC常规序列数据寄存器
先决条件	-
被调用函数	-
输入参数{in}	
adc_periph	ADC外设
ADCx(x=0,1,2)	ADC外设选择
输出参数{out}	
-	-
返回值	
uint16_t	ADC转换值（0 ~ 4095）

例如：

```
/* read ADC0 sequence data register */
```

```
uint16_t adc_value = 0;
```

```
adc_value = adc_routine_data_read(ADC0);
```

函数 `adc_inserted_data_read`

函数`adc_inserted_data_read`描述见下表：

表 3-22. 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);</code>
功能描述	读ADC注入序列数据寄存器
先决条件	-
被调用函数	-
输入参数{in}	
<code>adc_periph</code>	ADC外设
<code>ADCx(x=0,1,2)</code>	ADC外设选择
输出参数{out}	
-	-
返回值	
<code>uint16_t</code>	ADC转换值（0 ~ 4095）

例如：

```
/* read ADC0 inserted sequence register */

uint16_t adc_value = 0;

adc_value = adc_inserted_data_read (ADC0);
```

函数 `adc_latch_data_read`

函数`adc_latch_data_read`描述见下表：

表 3-23. 函数 `adc_latch_data_read`

函数名称	<code>adc_latch_data_read</code>
函数原形	<code>uint16_t adc_latch_data_read(uint32_t adc_periph, uint8_t latch_data)</code>
功能描述	读ADC锁存数据寄存器
先决条件	-
被调用函数	-
输入参数{in}	
<code>adc_periph</code>	ADC外设
<code>ADCx(x=0,1,2)</code>	ADC外设选择
输入参数{in}	
<code>latch_data</code>	锁存数据选择
<code>ADC_LATCH_DATA_x</code>	锁存数据x, x=0,1,2,3
输出参数{out}	
-	-
返回值	
<code>uint16_t</code>	ADC转换值（0 ~ 4095）

例如：

```
/* read ADC0 latch data register */
```

```
uint16_t adc_value = 0;
```

```
adc_value = adc_latch_data_read(ADC0, ADC_LATCH_DATA_0);
```

函数 `adc_latch_data_source_config`

函数 `adc_latch_data_source_config` 描述见下表：

表 3-24. 函数 `adc_latch_data_source_config`

函数名称	<code>adc_latch_data_source_config</code>
函数原形	<code>void adc_latch_data_source_config(uint32_t adc_periph, uint8_t latch_data, uint8_t adc_sequence, uint8_t rank);</code>
功能描述	配置ADC锁存数据源
先决条件	-
被调用函数	-
输入参数{in}	
adc_periph	ADC外设
<code>ADCx(x=0,1,2)</code>	ADC外设选择
输入参数{in}	
latch_data	锁存数据选择
<code>ADC_LATCH_DATA_x</code>	锁存数据x, x=0,1,2,3
输入参数{in}	
adc_sequence	序列选择
<code>ADC_ROUTINE_CHANNEL</code>	常规序列
<code>ADC_INSERTED_CHANNEL</code>	注入序列
输入参数{in}	
rank	常规序列：取值范围为0~15 注入序列：取值范围为0~3
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* configure ADC latch data source */
```

```
adc_latch_data_source_config(ADC0, ADC_LATCH_DATA_0, ADC_ROUTINE_CHANNEL, 5);
```

函数 `adc_watchdog0_single_channel_enable`

函数`adc_watchdog0_single_channel_enable`描述见下表:

表 3-25. 函数 `adc_watchdog0_single_channel_enable`

函数名称	<code>adc_watchdog0_single_channel_enable</code>
函数原形	<code>void adc_watchdog0_single_channel_enable(uint32_t adc_periph, uint8_t adc_channel);</code>
功能描述	配置ADC模拟看门狗0单通道有效
先决条件	-
被调用函数	-
输入参数{in}	
<code>adc_periph</code>	ADC外设
<code>ADCx(x=0,1,2)</code>	ADC外设选择
输入参数{in}	
<code>adc_channel</code>	ADC通道选择
<code>ADC_CHANNEL_x</code>	<code>ADC_CHANNEL_x</code> : ADC0 <code>x=0..17</code> , ADC1 <code>x=0..17</code> , ADC2 <code>x=0..16</code>
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* configure ADC0 analog watchdog0 single channel */
```

```
adc_watchdog0_single_channel_enable(ADC0, ADC_CHANNEL_1);
```

函数 `adc_watchdog0_sequence_channel_enable`

函数`adc_watchdog0_sequence_channel_enable`描述见下表:

表 3-26. 函数 `adc_watchdog0_sequence_channel_enable`

函数名称	<code>adc_watchdog0_sequence_channel_enable</code>
函数原形	<code>void adc_watchdog0_sequence_channel_enable(uint32_t adc_periph, uint8_t adc_sequence);</code>
功能描述	配置ADC模拟看门狗在序列有效
先决条件	-
被调用函数	-
输入参数{in}	
<code>adc_periph</code>	ADC外设
<code>ADCx(x=0,1,2)</code>	ADC外设选择
输入参数{in}	
<code>adc_sequence</code>	序列使用模拟看门狗
<code>ADC_ROUTINE_CHANNEL_ANNEL</code>	常规序列

ADC_INSERTED_CHANNEL	注入序列
ADC_ROUTINE_INSERTED_CHANNEL	常规和注入序列
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* configure ADC analog watchdog0 sequence */
```

```
adc_watchdog0_sequence_channel_enable(ADC0, ADC_ROUTINE_CHANNEL);
```

函数 adc_watchdog0_disable

函数adc_watchdog0_disable描述见下表：

表 3-27. 函数 adc_watchdog0_disable

函数名称	adc_watchdog0_disable
函数原形	void adc_watchdog0_disable(uint32_t adc_periph);
功能描述	ADC模拟看门狗0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)	ADC外设选择
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* disable ADC0 analog watchdog0 */
```

```
adc_watchdog0_disable(ADC0);
```

函数 adc_watchdog0_threshold_config

函数adc_watchdog0_threshold_config描述见下表：

表 3-28. 函数 adc_watchdog0_threshold_config

函数名称	adc_watchdog0_threshold_config
函数原形	void adc_watchdog0_threshold_config(uint32_t adc_periph, uint32_t low_threshold, uint32_t high_threshold);

功能描述	配置ADC模拟看门狗0阈值
先决条件	-
被调用函数	-
输入参数{in}	
adc_periph	ADC外设
ADCx(x=0,1,2)	ADC外设选择
输入参数{in}	
low_threshold	模拟看门狗低阈值, $0..2^{20}-1$
输入参数{in}	
high_threshold	模拟看门狗高阈值, $0..2^{20}-1$
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* configure ADC0 analog watchdog threshold */
```

```
adc_watchdog0_threshold_config(ADC0, 0x0400, 0x0A00);
```

函数 adc_resolution_config

函数adc_resolution_config描述见下表:

表 3-29. 函数 adc_resolution_config

函数名称	adc_resolution_config
函数原形	void adc_resolution_config(uint32_t adc_periph, uint32_t resolution);
功能描述	配置ADC分辨率
先决条件	-
被调用函数	-
输入参数{in}	
adc_periph	ADC外设
ADCx(x=0,1,2)	ADC外设选择
输入参数{in}	
resolution	ADC分辨率
ADC_RESOLUTION_12B	12位分辨率
ADC_RESOLUTION_10B	10位分辨率
ADC_RESOLUTION_8B	8位分辨率
ADC_RESOLUTION_6B	6位分辨率
输出参数{out}	

-	-
返回值	
-	-

Example:

```
/* configure ADC0 resolution */
```

```
adc_resolution_config(ADC0, ADC_RESOLUTION_12B);
```

函数 `adc_oversample_mode_config`

函数 `adc_oversample_mode_config` 描述见下表:

表 3-30. 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
<code>ADCx(x=0,1,2)</code>	ADC外设选择
输入参数{in}	
mode	ADC过采样触发模式
<code>ADC_OVERSAMPLING_ALL_CONVERT</code>	在一个触发之后, 对一个通道连续进行过采样转换
<code>ADC_OVERSAMPLING_ONE_CONVERT</code>	在一个触发之后, 对一个通道只进行一次过采样转换
输入参数{in}	
shift	ADC过滤采样移位
<code>ADC_OVERSAMPLING_SHIFT_NONE</code>	不移位
<code>ADC_OVERSAMPLING_SHIFT_1B</code>	移1位
<code>ADC_OVERSAMPLING_SHIFT_2B</code>	移2位
<code>ADC_OVERSAMPLING_SHIFT_3B</code>	移3位
<code>ADC_OVERSAMPLING_SHIFT_4B</code>	移4位
<code>ADC_OVERSAMPLING_SHIFT_5B</code>	移5位

NG_SHIFT_5B	
ADC_OVERSAMPLING_SHIFT_6B	移6位
ADC_OVERSAMPLING_SHIFT_7B	移7位
ADC_OVERSAMPLING_SHIFT_8B	移8位
输入参数{in}	
ratio	ADC过采样率
ADC_OVERSAMPLING_RATIO_MUL2	过采样率为2x
ADC_OVERSAMPLING_RATIO_MUL4	过采样率为4x
ADC_OVERSAMPLING_RATIO_MUL8	过采样率为8x
ADC_OVERSAMPLING_RATIO_MUL16	过采样率为16x
ADC_OVERSAMPLING_RATIO_MUL32	过采样率为32x
ADC_OVERSAMPLING_RATIO_MUL64	过采样率为64x
ADC_OVERSAMPLING_RATIO_MUL128	过采样率为128x
ADC_OVERSAMPLING_RATIO_MUL256	过采样率为256x
输出参数{out}	
-	-
返回值	
-	-

Example:

```
/* configure ADC0 oversample mode: 16 times sample, 4 bits shift */
```

```
adc_oversample_mode_config( ADC0, ADC_OVERSAMPLING_ALL_CONVERT,
ADC_OVERSAMPLING_SHIFT_4B, ADC_OVERSAMPLING_RATIO_MUL16);
```

函数 adc_oversample_mode_enable

函数adc_oversample_mode_enable描述见下表:

表 3-31. 函数 adc_oversample_mode_enable

函数名称	adc_oversample_mode_enable
函数原形	void adc_oversample_mode_enable(uint32_t adc_periph);

功能描述	使能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
ADCx(x=0,1,2)	ADC外设选择
输入参数{in}	
-	-
输出参数{out}	
-	-
返回值	
-	-

Example:

```
/* enable ADC0 oversample mode */
adc_oversample_mode_enable (ADC0);
```

函数 adc_oversample_mode_disable

函数adc_oversample_mode_disable描述见下表:

表 3-32. 函数 adc_oversample_mode_disable

函数名称	adc_oversample_mode_disable
函数原形	void adc_oversample_mode_disable(uint32_t adc_periph);
功能描述	禁能ADC过采样
先决条件	-
被调用函数	-
输入参数{in}	
adc_periph	ADC外设
ADCx(x=0,1,2)	ADC外设选择
输出参数{out}	
-	-
返回值	
-	-

Example:

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

函数 adc_flag_get

函数adc_flag_get描述见下表:

表 3-33. 函数 `adc_flag_get`

函数名称	<code>adc_flag_get</code>
函数原形	<code>FlagStatus adc_flag_get(uint32_t adc_periph, uint32_t flag);</code>
功能描述	获取ADC标志位
先决条件	-
被调用函数	-
输入参数{in}	
<code>adc_periph</code>	ADC外设
<code>ADCx(x=0,1,2)</code>	ADC外设选择
输入参数{in}	
<code>flag</code>	ADC标志位
<code>ADC_FLAG_WD0E</code>	模拟看门狗0事件标志位
<code>ADC_FLAG_EORC</code>	常规序列转换结束标志位
<code>ADC_FLAG_EOIC</code>	注入序列转换结束标志位
<code>ADC_FLAG_STIC</code>	注入序列转换开始标志位
<code>ADC_FLAG_STRC</code>	常规序列转换开始标志位
输出参数{out}	
-	-
返回值	
<code>FlagStatus</code>	SET或RESET

例如:

```
/* get the ADC0 analog watchdog0 flag bits*/
```

```
FlagStatus flag_value;
```

```
flag_value = adc_flag_get(ADC0, ADC_FLAG_WD0E);
```

函数 `adc_flag_clear`

函数`adc_flag_clear`描述见下表:

表 3-34. 函数 `adc_flag_clear`

函数名称	<code>adc_flag_clear</code>
函数原形	<code>void adc_flag_clear(uint32_t adc_periph, uint32_t flag);</code>
功能描述	清除ADC标志位
先决条件	-
被调用函数	-
输入参数{in}	
<code>adc_periph</code>	ADC外设
<code>ADCx(x=0,1,2)</code>	ADC外设选择
输入参数{in}	
<code>flag</code>	ADC标志位
<code>ADC_FLAG_WD0E</code>	模拟看门狗0事件标志位
<code>ADC_FLAG_EORC</code>	常规序列转换结束标志位

ADC_FLAG_EOIC	注入序列转换结束标志位
ADC_FLAG_STIC	注入序列转换开始标志位
ADC_FLAG_STRC	常规序列转换开始标志位
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* clear the ADC0 analog watchdog0 flag bits*/
```

```
adc_flag_clear(ADC0, ADC_FLAG_WD0E);
```

函数 adc_interrupt_enable

函数adc_interrupt_enable描述见下表：

表 3-35. 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 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)	ADC外设选择
输入参数{in}	
interrupt	ADC中断
ADC_INT_WD0E	模拟看门狗0中断
ADC_INT_EORC	常规序列转换结束中断
ADC_INT_EOIC	注入序列转换结束中断
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* enable ADC0 analog watchdog0 interrupt */
```

```
adc_interrupt_enable(ADC0, ADC_INT_WD0E);
```

函数 adc_interrupt_disable

函数adc_interrupt_disable描述见下表：

表 3-36. 函数 `adc_interrupt_disable`

函数名称	<code>adc_interrupt_disable</code>
函数原形	<code>void adc_interrupt_disable(uint32_t adc_periph, uint32_t interrupt);</code>
功能描述	ADC中断禁能
先决条件	-
被调用函数	-
输入参数{in}	
<code>adc_periph</code>	ADC外设
<code>ADCx(x=0,1,2)</code>	ADC外设选择
输入参数{in}	
<code>interrupt</code>	ADC中断
<code>ADC_INT_WD0E</code>	模拟看门狗0中断
<code>ADC_INT_EORC</code>	常规序列转换结束中断
<code>ADC_INT_EOIC</code>	注入序列转换结束中断
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* disable ADC0 interrupt */
```

```
adc_interrupt_disable(ADC0, ADC_INT_WD0E);
```

函数 `adc_interrupt_flag_get`

函数`adc_interrupt_flag_get`描述见下表：

表 3-37. 函数 `adc_interrupt_flag_get`

函数名称	<code>adc_interrupt_flag_get</code>
函数原形	<code>FlagStatus adc_interrupt_flag_get(uint32_t adc_periph, uint32_t flag);</code>
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}	
<code>adc_periph</code>	ADC外设
<code>ADCx(x=0,1,2)</code>	ADC外设选择
输入参数{in}	
<code>flag</code>	ADC中断标志位
<code>ADC_INT_FLAG_WD0E</code>	模拟看门狗0中断标志位
<code>ADC_INT_FLAG_EORC</code>	常规序列转换结束中断标志位
<code>ADC_INT_FLAG_EOIC</code>	注入序列转换结束中断标志位

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

例如:

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

```
FlagStatus flag_value;
```

```
flag_value = adc_interrupt_flag_get(ADC0, ADC_INT_FLAG_WD0E);
```

函数 adc_interrupt_flag_clear

函数adc_interrupt_flag_clear描述见下表:

表 3-38. 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 flag);
功能描述	清除ADC中断标志位
先决条件	-
被调用函数	-
输入参数{in}	
adc_periph	ADC外设
ADCx(x=0,1,2)	ADC外设选择
输入参数{in}	
flag	ADC中断标志位
ADC_INT_FLAG_WD0E	模拟看门狗0中断标志位
ADC_INT_FLAG_EORC	常规序列转换结束中断标志位
ADC_INT_FLAG_EOIC	注入序列转换结束中断标志位
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* clear the ADC0 analog watchdog0 interrupt bits*/
```

```
adc_interrupt_flag_clear(ADC0, ADC_INT_FLAG_WD0E);
```


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 3-39. BKP 寄存器

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

3.3.2. 外设库函数说明

BKP库函数列表如下表所示：

表 3-40. BKP 库函数

库函数名称	库函数描述
bkp_deinit	复位备份数据寄存器
bkp_write_data	写备份数据寄存器
bkp_read_data	读备份数据寄存器
bkp_rtc_calibration_output_enable	RTC时钟校准输出使能
bkp_rtc_calibration_output_disable	RTC时钟校准输出失能
bkp_rtc_signal_output_enable	RTC闹钟或秒信号输出使能
bkp_rtc_signal_output_disable	RTC闹钟或秒信号输出失能
bkp_rtc_output_select	RTC输出选择，RTC输出可选择为闹钟脉冲或秒脉冲
bkp_rtc_clock_output_select	RTC时钟输出选择
bkp_rtc_clock_calibration_direction	RTC时钟校准方向选择
bkp_rtc_calibration_value_set	RTC时钟校准值

库函数名称	库函数描述
bkp_tamper_detection_enable	TAMPER引脚使能
bkp_tamper_detection_disable	TAMPER引脚失能
bkp_tamper_active_level_set	TAMPER引脚有效电平设置
bkp_tamper_interrupt_enable	TAMPER中断使能
bkp_tamper_interrupt_disable	TAMPER中断失能
bkp_flag_get	获取标志位
bkp_flag_clear	清除标志位
bkp_interrupt_flag_get	获取中断标志位
bkp_interrupt_flag_clear	清除中断标志位

函数 bkp_deinit

函数bkp_deinit描述见下表：

表 3-41. 函数 bkp_deinit

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

例如：

```

/* reset BKP registers */
bkp_deinit ();

```

函数 bkp_write_data

函数bkp_write_data描述见下表：

表 3-42. 函数 bkp_write_data

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

例如：

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

函数 bkp_read_data

函数bkp_read_data描述见下表：

表 3-43. 函数 bkp_read_data

函数名称	bkp_read_data
函数原型	uint16_t bkp_read_data(bkp_data_register_enum register_number);
功能描述	读备份数据寄存器
先决条件	-

被调用函数	-
输入参数{in}	
register_number	参考枚举bkp_data_register_enum
BKP_DATA_x(x = 0..41)	BKP数据寄存器x
输出参数{out}	
-	-
返回值	
uint16_t	0-0xffff

例如:

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

函数 bkp_rtc_calibration_output_enable

函数bkp_rtc_calibration_output_enable描述见下表:

表 3-44. 函数 bkp_rtc_calibration_output_enable

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

例如:

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

```
bkp_rtc_calibration_output_enable();
```

函数 bkp_rtc_calibration_output_disable

函数bkp_rtc_calibration_output_disable描述见下表：

表 3-45. 函数 bkp_rtc_calibration_output_disable

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

例如：

```
/* disable RTC clock calibration output */
```

```
bkp_rtc_calibration_output_disable();
```

函数 bkp_rtc_signal_output_enable

函数bkp_rtc_signal_output_enable描述见下表：

表 3-46. 函数 bkp_rtc_signal_output_enable

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

输出参数{out}	
-	-
返回值	
-	-

例如：

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

```
bkp_rtc_signal_output_enable();
```

函数 bkp_rtc_signal_output_disable

函数bkp_rtc_signal_output_disable描述见下表：

表 3-47. 函数 bkp_rtc_signal_output_disable

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

例如：

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

```
bkp_rtc_signal_output_disable();
```

函数 bkp_rtc_output_select

函数bkp_rtc_output_select描述见下表：

表 3-48. 函数 bkp_rtc_output_select

函数名称	bkp_rtc_output_select
------	-----------------------

函数原型	void bkp_rtc_output_select (uint16_t outputsel);
功能描述	RTC输出选择，RTC输出可选择为闹钟脉冲或秒脉冲
先决条件	-
被调用函数	-
输入参数{in}	
outputsel	RTC输出选择
RTC_OUTPUT_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 output alarm signal output */
```

```
bkp_rtc_output_select (RTC_OUTPUT_ALARM_PULSE);
```

函数 bkp_rtc_clock_output_select

函数bkp_rtc_clock_output_select描述见下表：

表 3-49. 函数 bkp_rtc_clock_output_select

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

<i>RTC_CLOCK_DIV_1</i>	RTC时钟输出被选择为不分频
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* select RTC clock divided 64 to output */
```

```
bkp_rtc_clock_output_select (RTC_CLOCK_DIV_64);
```

函数 **bkp_rtc_clock_calibration_direction_select**

函数**bkp_rtc_clock_calibration_direction_select**描述见下表：

表 3-50. 函数 bkp_rtc_clock_calibration_direction_select

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

例如：

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


bkp_rtc_clock_calibration_direction_select (RTC_CLOCK_SLOWED_DOWN);

函数 bkp_rtc_calibration_value_set

函数bkp_rtc_calibration_value_set描述见下表：

表 3-51. 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 3-52. 函数 bkp_tamper_detection_enable

函数名称	bkp_tamper_detection_enable
函数原型	void bkp_tamper_detection_enable (void);
功能描述	TAMPER引脚使能
先决条件	-
被调用函数	-
输入参数{in}	

-	-
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* enable tamper pin detection */
```

```
bkp_tamper_detection_enable();
```

函数 bkp_tamper_detection_disable

函数bkp_tamper_detection_disable描述见下表：

表 3-53. 函数 bkp_tamper_detection_disable

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

例如：

```
/* disable tamper pin detection */
```

```
bkp_tamper_detection_disable ();
```

函数 bkp_tamper_active_level_set

函数bkp_tamper_active_level_set描述见下表：

表 3-54. 函数 bkp_tamper_active_level_set

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

例如：

```
/* set tamper pin active level high */
```

```
bkp_tamper_active_level_set (TAMPER_PIN_ACTIVE_HIGH);
```

函数 bkp_tamper_interrupt_enable

函数bkp_tamper_interrupt_enable描述见下表：

表 3-55. 函数 bkp_tamper_interrupt_enable

函数名称	bkp_tamper_interrupt_enable
函数原型	void bkp_tamper_interrupt_enable (void);
功能描述	TAMPER中断使能
先决条件	-
被调用函数	-
输入参数{in}	
-	-

输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* enable tamper pin interrupt */
```

```
bkp_tamper_interrupt_enable ();
```

函数 bkp_tamper_interrupt_disable

函数bkp_tamper_interrupt_disable描述见下表:

表 3-56. 函数 bkp_interrupt_disable

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

例如:

```
/* disable tamper pin interrupt */
```

```
bkp_tamper_interrupt_disable ();
```

函数 bkp_flag_get

函数bkp_flag_get描述见下表:

表 3-57. 函数 bkp_flag_get

函数名称	bkp_flag_get
------	--------------

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

例如:

```
/* get BKP flag state */
```

```
FlagStatus status;
```

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

函数 bkp_flag_clear

函数bkp_flag_clear描述见下表:

表 3-58. 函数 bkp_flag_clear

函数名称	bkp_flag_clear
函数原型	void bkp_flag_clear(void);
功能描述	清除标志位
先决条件	-
被调用函数	-
输入参数{in}	
-	-
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* clear BKP flag state */
```

```
bkp_flag_clear ();
```

函数 bkp_interrupt_flag_get

函数bkp_interrupt_flag_get描述见下表：

表 3-59. 函数 bkp_interrupt_flag_get

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

例如：

```
/* get BKP interrupt flag state */
```

```
bkp_interrupt_flag_get ();
```

函数 bkp_interrupt_flag_clear

函数bkp_interrupt_flag_clear描述见下表：

表 3-60. 函数 bkp_interrupt_flag_clear

函数名称	bkp_interrupt_flag_clear
函数原型	void bkp_interrupt_flag_clear(void);
功能描述	清除中断标志位
先决条件	-
被调用函数	-

输入参数{in}	
-	-
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* clear BKP interrupt flag state */
```

```
bkp_interrupt_flag_clear ();
```

3.4. CAN

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

3.4.1. 外设寄存器说明

CAN寄存器列表如下表所示:

表 3-61. 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_FDCTL	FD控制寄存器
CAN_FDSTAT	FD状态寄存器
CAN_FDTDC	FD传输延迟补偿寄存器
CAN_DBT	数据位时序寄存器
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

寄存器名称	寄存器描述
CAN_RFIFOMPx	接收FIFO邮箱属性寄存器
CAN_RFIFOMDAT A0x	接收FIFO邮箱data0寄存器
CAN_RFIFOMDAT A1x	接收FIFO邮箱data1寄存器
CAN_FCTL	过滤器控制寄存器
CAN_FMCFG	过滤器模式配置寄存器
CAN_FSCFG	过滤器位宽配置寄存器
CAN_FAFIFO	过滤器关联FIFO寄存器
CAN_FW	过滤器激活寄存器
CAN_FxDATAy	过滤器(x)数据(y)寄存器

3.4.2. 外设库函数说明

CAN库函数列表如下表所示：

表 3-62. 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	CAN外设库使用到的各类结构体初始化
can_init	初始化外设CAN
can_fd_init	CAN FD功能初始化
can_filter_init	CAN过滤器初始化
can_filter_mask_mode_init	CAN过滤器掩码模式初始化
can_monitor_mode_set	CAN总线监听模式配置
can_fd_function_enable	FD功能使能
can_fd_function_disable	FD功能关闭
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

库函数名称	库函数描述
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
can_fifo_overnun_flag_get	获取FIFO溢出标志状态

结构体 can_parameter_struct

表 3-63. 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 3-64. 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[64]	数据值
fd_flag	FD帧标志位
fd_brs	位速率转换开关
fd_esi	错误状态指示

结构体 can_receive_message_struct

表 3-65. 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[64]	数据值
rx_fi	过滤器索引
fd_flag	FD帧标志位
fd_brs	位速率转换开关
fd_esi	错误状态指示

结构体 can_filter_parameter_struct

表 3-66. 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

结构体 can_fd_tdc_struct

表 3-67. 结构体 can_fd_tdc_struct

成员名称	功能描述
tdc_mode	传输延迟补偿工作模式
tdc_filter	传输延迟补偿过滤器
tdc_offset	传输延迟补偿偏移

结构体 can_fdframe_struct

表 3-68. 结构体 can_fdframe_struct

成员名称	功能描述
fd_frame	FD功能开关
excp_event_detect	协议异常事件检测功能
delay_compensation	传输延迟补偿

成员名称	功能描述
p_delay_compensation	传输延迟补偿配置结构体指针，详见 表3-67. 结构体can_fd_tdc_struct
edge_filter_disable	边沿滤波失能
iso_bosch	ISO/Bosch模式选择
esi_mode	错误状态指示模式
data_resync_jump_width	数据域再同步补偿宽度
data_time_segment_1	数据域位段1
data_time_segment_2	数据域位段2
data_prescaler	数据域波特率预分频器

函数 can_deinit

函数can_deinit描述见下表：

表 3-69. 函数 can_deinit

函数名称	can_deinit
函数原型	void can_deinit(uint32_t can_periph);
功能描述	复位外设CAN
先决条件	-
被调用函数	rcu_periph_reset_enable/ rcu_periph_reset_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：

```
/* CAN0 deinitialize */
can_deinit (CAN0);
```

函数 can_struct_para_init

函数can_struct_para_init描述见下表：

表 3-70. 函数 can_struct_para_init

函数名称	can_struct_para_init
函数原型	void can_struct_para_init(can_struct_type_enum type, void* p_struct)
功能描述	CAN外设库使用到的各类结构体初始化
先决条件	-

被调用函数	-
输入参数{in}	
type	需要初始化的结构体类型，仅可选择唯一参数
CAN_INIT_STRUCT	初始化结构体
CAN_FILTER_STRUCT	过滤器初始化结构体
CAN_FD_FRAME_STRUCT	FD帧初始化结构体
CAN_TX_MESSAGE_STRUCT	存储发送帧结构体
CAN_RX_MESSAGE_STRUCT	接收帧结构体
输出参数{out}	
p_struct	对应的需要初始化的结构体指针
返回值	
-	-

例如：

```
/* Initialize CAN parameter struct */
can_parameter_struct can_init;
can_struct_para_init (CAN_INIT_STRUCT, &can_init);
```

函数 can_init

函数can_init描述见下表：

表 3-71. 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	can_struct_para_init()
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}	
can_parameter_init	初始化结构体，结构体成员参考 表3-63. 结构体can_parameter_struct
输出参数{out}	
-	-
返回值	

ErrStatus	SUCCESS / ERROR
-----------	-----------------

例如:

```
/* CAN0 initialize */
can_init (CAN0);
```

函数 can_fd_init

函数can_fd_init描述见下表:

表 3-72. 函数 can_fd_init

函数名称	can_fd_init
函数原型	ErrStatus can_fd_init(uint32_t can_periph, can_fdframe_struct* can_fdframe_init);
功能描述	初始化外设CAN FD功能
先决条件	can_struct_para_init()
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}	
can_fdframe_init	初始化FD功能结构体, 结构体成员参考 表3-68. 结构体can_fdframe_struct
输出参数{out}	
-	-
返回值	
ErrStatus	SUCCESS / ERROR

例如:

```
/* CAN0 FD initialize */
can_fdframe_struct fd_init_para;
can_fd_init(CAN0, &fd_init_para);
```

函数 can_filter_init

函数can_filter_init描述见下表:

表 3-73. 函数 can_filter_init

函数名称	can_filter_init
函数原型	void can_filter_init(can_filter_parameter_struct* can_filter_parameter_init);
功能描述	CAN过滤器初始化
先决条件	can_struct_para_init()
被调用函数	-
输入参数{in}	
can_filter_paramet	过滤器初始化结构体, 结构体成员参考 表3-66. 结构体

er_init	can_filter_parameter_struct
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* initialize CAN filter */
```

```
can_filter_init(&can_filter);
```

函数 can_filter_mask_mode_init

函数can_filter_mask_mode_init描述见下表：

表 3-74. 函数 can_filter_mask_mode_init

函数名称	can_filter_mask_mode_init
函数原型	void can_filter_mask_mode_init(uint32_t id, uint32_t mask, can_format_fifo_enum format_fifo, uint16_t filter_number)
功能描述	CAN过滤器掩码模式初始化
先决条件	-
被调用函数	can_filter_init()
输入参数{in}	
id	取值范围（0x00000000 - 0xFFFFFFFF）
输入参数{in}	
mask	取值范围（0x00000000 - 0xFFFFFFFF）
输入参数{in}	
format_fifo	帧格式及FIFO选择，仅可选择唯一参数
CAN_STANDARD_FIFO0	使用标准帧格式，FIFO0存储
CAN_STANDARD_FIFO1	使用标准帧格式，FIFO1存储
CAN_EXTENDED_FIFO0	使用扩展帧格式，FIFO0存储
CAN_EXTENDED_FIFO1	使用扩展帧格式，FIFO1存储
输入参数{in}	
filter_number	使用过滤器序号，取值范围（0x00 - 0x1B）
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* CAN filter mask mode initialization */
```

```
can_filter_mask_mode_init(0x11, 0x11, CAN_STANDARD_FIFO0, 0);
```

函数 can_monitor_mode_set

函数can_monitor_mode_set描述见下表:

表 3-75. 函数 can_monitor_mode_set

函数名称	can_monitor_mode_set
函数原型	ErrStatus can_monitor_mode_set(uint32_t can_periph, uint8_t mode)
功能描述	CAN总线监听模式配置
先决条件	-
被调用函数	-
输入参数{in}	
can_periph	CAN外设
CANx(x=0,1)	CAN外设选择
输入参数{in}	
mode	监听模式, 仅可选择唯一参数
CAN_NORMAL_MODE	正常模式
CAN_LOOPBACK_MODE	回环通讯模式
CAN_SILENT_MODE	静默通讯模式
CAN_SILENT_LOOPBACK_MODE	静默回环通讯模式
输出参数{out}	
-	-
返回值	
ErrStatus	SUCCESS / ERROR

例如:

```
/* CAN communication mode configure */
```

```
can_monitor_mode_set(CAN0, CAN_NORMAL_MODE);
```

函数 can_fd_function_enable

函数can_fd_function_enable描述见下表:

表 3-76. 函数 can_fd_function_enable

函数名称	can_fd_function_enable
函数原型	void can_fd_function_enable(uint32_t can_periph)
功能描述	CAN FD功能使能
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：

```
/* CAN0 FD frame function enable */
```

```
can_fd_function_enable(CAN0);
```

函数 can_fd_function_disable

函数can_fd_function_disable描述见下表：

表 3-77. 函数 can_fd_function_disable

函数名称	can_fd_function_disable
函数原型	void can_fd_function_disable(uint32_t can_periph)
功能描述	CAN FD功能关闭
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：

```
/* CAN0 FD frame function disable */
```

```
can_fd_function_disable(CAN0);
```

函数 can1_filter_start_bank

函数can1_filter_start_bank描述见下表：

表 3-78. 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 3-79. 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 3-80. 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 3-81. 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 3-82. 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 3-83. 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_trasnmit_message_struct* transmit_message);
功能描述	CAN传输报文
先决条件	can_struct_para_init()
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3-64. 结构体 can_trasnmit_message_struct
输出参数{out}	
-	-
返回值	
uint8_t	0x00-0x03

例如:

```
/* CAN0 transmit message and return the mailbox number */
uint8_t transmit_mailbox = 0;
transmit_mailbox = can_message_transmit(CAN0, &transmit_message);
```

函数 can_transmit_states

函数can_transmit_states描述见下表:

表 3-84. 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
CAN_MAILBOXx	CAN_MAILBOXx(x=0,1,2)
输出参数{out}	
-	-
返回值	
can_transmit_state_enum	0..4

例如:

```
/* CAN0 mailbox0 transmit state */
```

```
uint8_t transmit_state = 0;
```

```
transmit_state = can_transmit_states (CAN0, CAN_MAILBOX0);
```

函数 can_transmission_stop

函数can_transmission_stop描述见下表:

表 3-85. 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
CANx(x=0,1)	CAN外设选择
输入参数{in}	
mailbox_number	邮箱标号
CAN_MAILBOXx	CAN_MAILBOXx(x=0,1,2)
输出参数{out}	
-	-
返回值	
ErrStatus	SUCCESS / 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 3-86. 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	can_struct_para_init()
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	CAN_FIFOx(x=0,1)
输入参数{in}	
receive_message	接收报文结构体，结构体成员参考 表3-65. 结构体 can_receive_message_struct
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* CAN0 FIFO0 receive message */
can_message_receive(CAN0, CAN_FIFO0, &receive_message);
```

函数 can_fifo_release

函数can_fifo_release描述见下表：

表 3-87. 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
<i>CANx(x=0,1)</i>	CAN外设选择
输入参数{in}	
fifo_number	FIFO编号
<i>CAN_FIFOx</i>	CAN_FIFOx(x=0,1)
输出参数{out}	
-	-
返回值	
-	-

例如:

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

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

函数 can_receive_message_length_get

函数can_receive_message_length_get描述见下表:

表 3-88. 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</i>	CAN_FIFOx(x=0,1)
输出参数{out}	
-	-
返回值	
uint8_t	0..3

例如:

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

```
uint8_t frame_number = 0;
```

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

函数 can_working_mode_set

函数can_working_mode_set描述见下表:

表 3-89. 函数 can_working_mode_set

函数名称	can_working_mode_set
函数原型	ErrStatus can_working_mode_set(uint32_t can_periph, uint8_t working_mode);
功能描述	CAN工作模式设置
先决条件	-
被调用函数	-
输入参数{in}	
can_periph	CAN 外设
CANx(x=0,1)	CAN外设选择
输入参数{in}	
can_working_mode	模式选择
CAN_MODE_INITIALIZE	初始化模式
CAN_MODE_NORMAL	正常模式
CAN_MODE_SLEEP	睡眠模式
输出参数{out}	
-	-
返回值	
ErrStatus	SUCCESS / ERROR

例如:

```
/* set CAN0 working at initialize mode */
```

```
can_working_mode_set (CAN0, CAN_MODE_INITIALIZE);
```

函数 can_wakeup

函数can_wakeup描述见下表:

表 3-90. 函数 can_wakeup

函数名称	can_wakeup
函数原型	ErrStatus can_wakeup(uint32_t can_periph);
功能描述	从睡眠模式中唤醒CAN
先决条件	-
被调用函数	-
输入参数{in}	
can_periph	CAN 外设
CANx(x=0,1)	CAN外设选择

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 3-91. 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	0..7

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 3-92. 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 3-93. 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 3-94. 函数 can_flag_get

函数名称	can_flag_get
函数原型	FlagStatus can_flag_get(uint32_t can_periph, can_flag_enum flag);
功能描述	获取CAN标志位状态
先决条件	-

被调用函数	-
输入参数{in}	
can_periph	CAN 外设
CANx(x=0,1)	CAN外设选择
输入参数{in}	
flag	CAN 标志位
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_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
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 3-95. 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
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_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 3-96. 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 3-97. 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	发送邮箱空中断使能
CAN_INT_RFNE0	接收FIFO0非空中断使能
CAN_INT_RFF0	接收FIFO0满中断使能
CAN_INT_RFO0	接收FIFO0溢出中断使能
CAN_INT_RFNE1	接收FIFO1非空中断使能
CAN_INT_RFF1	接收FIFO1满中断使能
CAN_INT_RFO1	接收FIFO1溢出中断使能
CAN_INT_WERR	警告错误中断使能
CAN_INT_PERR	被动错误中断使能
CAN_INT_BO	离线中断使能
CAN_INT_ERRN	错误种类中断使能
CAN_INT_ERR	错误中断使能
CAN_INT_WU	唤醒中断使能

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

例如:

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

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

函数 can_interrupt_flag_get

函数can_interrupt_flag_get描述见下表:

表 3-98. 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
CAN_INT_FLAG_SLPW	进入睡眠工作模式的状态改变中断标志
CAN_INT_FLAG_WUIF	从睡眠工作模式唤醒的状态改变中断标志
CAN_INT_FLAG_ERRIF	错误中断标志
CAN_INT_FLAG_MTF2	邮箱2发送完成中断标志
CAN_INT_FLAG_MTF1	邮箱1发送完成中断标志
CAN_INT_FLAG_MTF0	邮箱0发送完成中断标志
CAN_INT_FLAG_RF00	接收FIFO0溢出中断标志
CAN_INT_FLAG_RF0F	接收FIFO0满中断标志
CAN_INT_FLAG_RF01	接收FIFO1溢出中断标志

CAN_INT_FLAG_R FF1	接收FIFO1满中断标志
输出参数{out}	
-	-
返回值	
FlagStatus	SET / RESET

例如：

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

```
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 3-99. 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
CAN_INT_FLAG_S LPIF	进入睡眠工作模式的状态改变中断标志
CAN_INT_FLAG_W UIF	从睡眠工作模式唤醒的状态改变中断标志
CAN_INT_FLAG_E RRIF	错误中断标志
CAN_INT_FLAG_M TF2	邮箱2发送完成中断标志
CAN_INT_FLAG_M TF1	邮箱1发送完成中断标志
CAN_INT_FLAG_M TF0	邮箱0发送完成中断标志
CAN_INT_FLAG_R FO0	接收FIFO0溢出中断标志
CAN_INT_FLAG_R FF0	接收FIFO0满中断标志
CAN_INT_FLAG_R	接收FIFO1溢出中断标志

FO1	
CAN_INT_FLAG_R FF1	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);
```

函数 can_fifo_overnrun_flag_get

函数can_fifo_overnrun_flag_get描述见下表:

表 3-100. 函数 can_fifo_overnrun_flag_get

函数名称	can_fifo_overnrun_flag_get
函数原型	FlagStatus can_fifo_overnrun_flag_get(uint32_t can_periph, uint8_t fifo_number);
功能描述	获取FIFO溢出标志状态
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}	
fifo_number	FIFO编号
CAN_FIFOx	CAN_FIFOx(x=0,1)
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* get FIFO0 overrun flag state of CAN0 */
can_fifo_overnrun_flag_get(CAN0, CAN_FIFO0);
```

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 3-101. 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
CAU_KEY3H	CAU密钥3高位寄存器
CAU_KEY3L	CAU密钥3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 3-102. CAU 库函数

库函数名称	库函数描述
cau_deinit	复位CAU外设
cau_struct_para_init	初始化CAU加密解密结构体
cau_key_struct_para_init	初始化密钥结构体
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_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

库函数名称	库函数描述
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 3-103. 结构体 cau_key_parameter_struct

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

结构体 cau_parameter_struct

表 3-104. 结构体 cau_parameter_struct

成员名称	功能描述
alg_dir	算法方向
*key	密钥
key_size	密钥字节长度
*input	输入数据
in_length	输入数据字节长度

函数 cau_deinit

函数cau_deinit描述见下表：

表 3-105. 函数 cau_deinit

函数名称	cau_deinit
函数原形	void cau_deinit(void);
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_struct_para_init

函数cau_struct_para_init描述见下表：

表 3-106. 函数 cau_struct_para_init

函数名称	cau_struct_para_init
函数原形	void cau_struct_para_init(cau_parameter_struct *cau_parameter);
功能描述	初始化CAU加密解密结构体
先决条件	-
被调用函数	-
输入参数{in}	
-	-
输出参数{out}	
cau_parameter	CAU加密解密结构体，参考结构体 表3-104. 结构体cau_parameter_struct
返回值	
-	-

例如：

```
cau_parameter_struct text;

/* initialize CAU encrypt and decrypt parameter struct */
cau_struct_para_init(&text);
```

函数 cau_key_struct_para_init

The description of cau_key_struct_para_init描述见下表：

表 3-107. 函数 cau_key_struct_para_init

函数名称	cau_key_struct_para_init
函数原形	void cau_key_struct_para_init(cau_key_parameter_struct *key_initpara);
功能描述	初始化密钥结构体
先决条件	-
被调用函数	-
输入参数{in}	
-	-
输出参数{out}	
key_initpara	密钥结构体，参考结构体 表3-103. 结构体cau_key_parameter_struct
返回值	
-	-

例如：

```

/* initialize the key parameter struct */

cau_key_parameter_struct key_initpara;

cau_key_struct_para_init(&key_initpara);

```

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 alg_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}	
alg_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_AES_ECB	AES-ECB（电子密码本）
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_KEYSIZE_128BIT	128位密钥长度
CAU_KEYSIZE_192BIT	192位密钥长度
CAU_KEYSIZE_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_KEYSIZE_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	密钥参数，参考结构体 表3-103. 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;
```

```
cau_key_init(&key_initpara);
```

函数 cau_fifo_flush

函数cau_fifo_flush描述见下表：

表 3-115. 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 3-116. 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 = cau_enable_state_get();
```

函数 cau_data_write

函数cau_data_write描述见下表：

表 3-117. 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0 - 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 3-118. 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 = cau_data_read();
```


函数 cau_aes_ecb

函数cau_aes_ecb描述见下表:

表 3-119. 函数 cau_aes_ecb

函数名称	cau_aes_ecb
函数原形	ErrStatus cau_aes_ecb(cau_parameter_struct *cau_parameter, uint8_t *output);
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}	
cau_parameter	CAU加密和解密参数, 参考结构体 表3-104. 结构体cau_parameter_struct
输出参数{out}	
output	指针指向返回数组
返回值	
ErrStatus	SUCCESS或ERROR

例如:

```
cau_parameter_struct text;

uint8_t encrypt_result[TEXT_SIZE];

ErrStatus status;

.....

cau_struct_para_init(&text);

key_addr = key_select[i];

key_size = keysize[i];

text.alg_dir    = CAU_ENCRYPT;

text.key        = key_addr;

text.key_size   = key_size;

text.input      = plaintext;

text.in_length  = TEXT_SIZE;

/* encryption in ECB mode */

status = cau_aes_ecb(&text, encrypt_result);
```

函数 cau_flag_get

函数cau_flag_get描述见下表:

表 3-120. 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 3-121. 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 3-122. 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 3-123. 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 interrupt)
功能描述	获取中断标志
先决条件	-
被调用函数	-
输入参数{in}	
interrupt	CAU中断标志

CAU_INT_FLAG_INFIF O	输入FIFO中断
CAU_INT_FLAG_OUTF IFO	输出FIFO中断
输出参数{out}	
-	-
返回值	
FlagStatus	SET or RESET

例如：

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

```
FlagStatus status;
```

```
status = cau_interrupt_flag_get(CAU_INT_FLAG_INFIFO);
```

3.6. CMP

CMP通用比较器可独立工作，其输出端口可用于I/O口，也可和定时器结合使用。在一定的条件下，比较器可将模拟信号作为触发源，结合定时器的PWM输出，可以实现电流控制。章节[3.6.1](#)描述了CMP的寄存器列表，章节[3.6.2](#)对CMP库函数进行说明。

3.6.1. 外设寄存器说明

CMP寄存器列表如下表所示：

表 3-124. CMP 寄存器

寄存器名称	寄存器描述
CMP_STAT	比较器状态控制器
CMP_IFC	比较器中断标志位清除寄存器
CMP0_CS	CMP0控制状态寄存器

3.6.2. 外设库函数说明

CMP库函数列表如下表所示：

表 3-125. CMP 库函数

库函数名称	库函数描述
cmp_deinit	复位CMP
cmp_mode_init	CMP工作模式初始化
cmp_output_init	CMP输出初始化
cmp_digital_filter_init	CMP数字滤波初始化
cmp_blanking_init	CMP消隐功能初始化
cmp_enable	使能CMP
cmp_disable	禁能CMP

库函数名称	库函数描述
cmp_lock_enable	锁定CMP
cmp_voltage_scaler_enable	使能带隙标量
cmp_voltage_scaler_disable	禁能带隙标量
cmp_scaler_bridge_enable	使能标量桥接
cmp_scaler_bridge_disable	禁能标量桥接
cmp_output_level_get	获取CMP输出状态
cmp_flag_get	获取CMP标志位
cmp_flag_clear	清除CMP标志位
cmp_interrupt_enable	CMP中断使能
cmp_interrupt_disable	CMP中断禁能
cmp_interrupt_flag_get	获取CMP中断标志位
cmp_interrupt_flag_clear	清除CMP中断标志位

枚举类型 cmp_enum

表 3-126. 枚举类型 cmp_enum

成员名称	功能描述
CMP0	比较器0

函数 cmp_deinit

函数cmp_deinit描述见下表：

表 3-127. 函数 cmp_deinit

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

例如：

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

函数 cmp_mode_init

函数cmp_mode_init描述见下表：

表 3-128. 函数 cmp_mode_init

函数名称	cmp_mode_init
函数原型	void cmp_mode_init(cmp_enum cmp_periph, uint32_t inverting_input, uint32_t output_hysteresis);
功能描述	CMP工作模式初始化
先决条件	-
被调用函数	-
输入参数{in}	
cmp_periph	参考枚举 表 3-126. 枚举类型cmp_enum
输入参数{in}	
inverting_input	反相输入源
CMP_INVERTING_INPUT_1_4VREFINT	选择1/4V _{REFINT} 作为输入源
CMP_INVERTING_INPUT_1_2VREFINT	选择1/2V _{REFINT} 作为输入源
CMP_INVERTING_INPUT_3_4VREFINT	选择3/4V _{REFINT} 作为输入源
CMP_INVERTING_INPUT_VREFINT	选择V _{REFINT} 作为输入源
CMP_INVERTING_INPUT_PA4	选择PA4作为输入源
CMP_INVERTING_INPUT_PA5	选择PA5作为输入源
CMP_INVERTING_INPUT_PA2	选择PA2作为输入源
CMP_INVERTING_INPUT_DAC0_OUT0	选择DAC0_OUT0作为输入源
inverting_input	
output_hysteresis	迟滞水平
CMP_HYSTERESIS_NO	无迟滞
CMP_HYSTERESIS_LOW	低迟滞
CMP_HYSTERESIS_MIDDLE	中迟滞
CMP_HYSTERESIS_HIGH	高迟滞
输出参数{out}	
-	-
返回值	

-	-
---	---

例如:

```
/* initialize CMP0 mode */
```

```
cmp_mode_init(CMP0, CMP_INVERTING_INPUT_1_4VREFINT, CMP_HYSTERESIS_NO);
```

函数 cmp_output_init

函数cmp_output_init描述见下表:

表 3-129. 函数 cmp_output_init

函数名称	cmp_output_init
函数原型	void cmp_output_init(cmp_enum cmp_periph, uint32_t output_polarity);
功能描述	CMP输出初始化
先决条件	-
被调用函数	-
输入参数{in}	
cmp_periph	参考枚举 表 3-126. 枚举类型cmp_enum
输入参数{in}	
output_polarity	CMP输出极性
CMP_OUTPUT_POLARITY_INVERTED	输出反相
CMP_OUTPUT_POLARITY_NONINVERTED	输出正相
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* initialize CMP0 output */
```

```
cmp_output_init(CMP0, CMP_OUTPUT_POLARITY_NONINVERTED);
```

函数 cmp_blanking_init

函数cmp_blanking_init描述见下表:

表 3-130. 函数 cmp_blanking_init

函数名称	cmp_blanking_init
函数原型	void cmp_blanking_init(cmp_enum cmp_periph, uint32_t blanking_source_selection);
功能描述	CMP消隐功能初始化

先决条件	-
被调用函数	-
输入参数{in}	
cmp_periph	参考枚举 表 3-126. 枚举类型cmp_enum
输入参数{in}	
blanking_source_selection	消隐源选择
CMP_BLANKING_NONE	无消隐
CMP_BLANKING_TIMER0_OC0	选择TIMER0_CH0输出比较信号为消隐源
CMP_BLANKING_TIMER1_OC2	选择TIMER1_CH2输出比较信号为消隐源
CMP_BLANKING_TIMER2_OC2	选择TIMER2_CH2输出比较信号为消隐源
CMP_BLANKING_TIMER2_OC3	选择TIMER2_CH3输出比较信号为消隐源
CMP_BLANKING_TIMER7_OC0	选择TIMER7_CH0输出比较信号为消隐源
CMP_BLANKING_TIMER15_OC0	选择TIMER15_CH0输出比较信号为消隐源
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* initialize CMP0 blanking function */
```

```
cmp_blanking_init(CMP0, CMP_BLANKING_NONE);
```

函数 cmp_digital_filter_init

函数cmp_digital_filter_init描述见下表：

表 3-131. 函数 cmp_digital_filter_init

函数名称	cmp_digital_filter_init
函数原型	void cmp_digital_filter_init(cmp_enum cmp_periph, uint32_t sample_frequency_selection, uint32_t filter_mode_selection);
功能描述	CMP数字滤波初始化
先决条件	-
被调用函数	-
输入参数{in}	
cmp_periph	参考枚举 表 3-126. 枚举类型cmp_enum
输入参数{in}	

sample_frequency_selection	采样频率
CMP_NOISE_FILTER_NONE	不使用数字滤波
CMP_SAMPLING_FREQUENCY_PCLK_DIV8	数字滤波使用的采样频率为PCLK/8
CMP_SAMPLING_FREQUENCY_PCLK_DIV16	数字滤波使用的采样频率为PCLK/16
CMP_SAMPLING_FREQUENCY_PCLK_DIV32	数字滤波使用的采样频率为PCLK/32
输入参数{in}	
Filter_mode_selection	采样次数
CMP_FILTER_MODE_3_TIMES	数字滤波采样次数为3次
CMP_FILTER_MODE_4_TIMES	数字滤波采样次数为4次
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* initialize CMP0 digital filter */
```

```
cmp_digital_filter_init(CMP0, CMP_SAMPLING_FREQUENCY_PCLK_DIV8, CMP_FILTER_MODE_3_TIMES);
```

函数 cmp_enable

函数cmp_enable描述见下表：

表 3-132. 函数 cmp_enable

函数名称	cmp_enable
函数原型	void cmp_enable(cmp_enum cmp_periph);
功能描述	使能CMP
先决条件	-
被调用函数	-
输入参数{in}	
cmp_periph	参考枚举 表 3-126. 枚举类型cmp_enum
输出参数{out}	
-	-

返回值	
-	-

例如:

```
/* enable CMP0 */
```

```
cmp_enable(CMP0);
```

函数 cmp_disable

函数cmp_disable描述见下表:

表 3-133. 函数 cmp_disable

函数名称	cmp_disable
函数原型	void cmp_disable(cmp_enum cmp_periph);
功能描述	禁能CMP
先决条件	-
被调用函数	-
输入参数{in}	
cmp_periph	参考枚举 表 3-126. 枚举类型cmp_enum
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* disable CMP0 */
```

```
cmp_disable(CMP0);
```

函数 cmp_lock_enable

函数cmp_lock_enable描述见下表:

表 3-134. 函数 cmp_lock_enable

函数名称	cmp_lock_enable
函数原型	void cmp_lock_enable(cmp_enum cmp_periph);
功能描述	锁定CMP
先决条件	-
被调用函数	-
输入参数{in}	
cmp_periph	参考枚举 表 3-126. 枚举类型cmp_enum
输出参数{out}	
-	-
返回值	
-	-

例如：

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

函数 cmp_voltage_scaler_enable

函数cmp_voltage_scaler_enable描述见下表：

表 3-135. 函数 cmp_voltage_scaler_enable

函数名称	cmp_voltage_scaler_enable
函数原型	void cmp_voltage_scaler_enable(cmp_enum cmp_periph);
功能描述	使能CMP带隙标量
先决条件	-
被调用函数	-
输入参数{in}	
cmp_periph	参考枚举 表 3-126. 枚举类型cmp_enum
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* enable CMP0 the voltage scaler */
cmp_voltage_scaler_enable(CMP0);
```

函数 cmp_voltage_scaler_disable

函数cmp_voltage_scaler_disable描述见下表：

表 3-136. 函数 cmp_voltage_scaler_disable

函数名称	cmp_voltage_scaler_disable
函数原型	void cmp_voltage_scaler_disable(cmp_enum cmp_periph);
功能描述	禁能CMP带隙标量
先决条件	-
被调用函数	-
输入参数{in}	
cmp_periph	参考枚举 表 3-126. 枚举类型cmp_enum
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* disable CMP0 the voltage scaler */
```

```
cmp_voltage_scaler_disable(CMP0);
```

函数 cmp_scaler_bridge_enable

函数cmp_scaler_bridge_enable描述见下表：

表 3-137. 函数 cmp_scaler_bridge_enable

函数名称	cmp_scaler_bridge_enable
函数原型	void cmp_scaler_bridge_enable(cmp_enum cmp_periph);
功能描述	使能CMP标量桥接
先决条件	-
被调用函数	-
输入参数{in}	
cmp_periph	参考枚举 表 3-126. 枚举类型cmp_enum
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* enable CMP0 the scaler bridge */
```

```
cmp_scaler_bridge_enable(CMP0);
```

函数 cmp_scaler_bridge_disable

函数cmp_scaler_bridge_disable描述见下表：

表 3-138. 函数 cmp_scaler_bridge_disable

函数名称	cmp_scaler_bridge_disable
函数原型	void cmp_scaler_bridge_disable(cmp_enum cmp_periph);
功能描述	禁能CMP标量桥接
先决条件	-
被调用函数	-
输入参数{in}	
cmp_periph	参考枚举 表 3-126. 枚举类型cmp_enum
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* disable CMP0 the scaler bridge */
```

```
cmp_scaler_bridge_disable(CMP0);
```

函数 cmp_output_level_get

函数cmp_output_level_get描述见下表:

表 3-139. 函数 cmp_output_level_get

函数名称	cmp_output_level_get
函数原型	uint32_t cmp_output_level_get(uint32_t cmp_periph);
功能描述	获取CMP输出状态
先决条件	-
被调用函数	-
输入参数{in}	
cmp_periph	参考枚举 表 3-126. 枚举类型cmp_enum
输出参数{out}	
-	-
返回值	
uint32_t	输出电平
CMP_OUTPUTLEV EL_HIGH	比较器输出高电平
CMP_OUTPUTLEV EL_LOW	比较器输出低电平

例如:

```
uint32_t level;
```

```
/* get CMP0 output level */
```

```
level = cmp_output_level_get(CMP0);
```

函数 cmp_flag_get

函数cmp_flag_get描述见下表:

表 3-140. 函数 cmp_flag_get

函数名称	cmp_flag_get
函数原形	FlagStatus cmp_flag_get(cmp_enum cmp_periph, uint32_t flag);
功能描述	获取CMP标志位
先决条件	-
被调用函数	-
输入参数{in}	
cmp_periph	参考枚举 表 3-126. 枚举类型cmp_enum
输入参数{in}	
flag	CMP中断标志位
CMP_FLAG_COMP ARE	CMP0比较中断标志位
输出参数{out}	
-	-

返回值	
FlagStatus	SET或RESET

例如：

```
/* get the CMP0 interrupt flag bit */
```

```
FlagStatus flag_value;
```

```
flag_value = cmp_flag_get(CMP0, CMP_FLAG_COMPARE);
```

函数 cmp_flag_clear

函数cmp_flag_clear描述见下表：

表 3-141. 函数 cmp_flag_clear

函数名称	cmp_flag_clear
函数原形	void cmp_flag_clear(cmp_enum cmp_periph, uint32_t flag);
功能描述	清除CMP标志位
先决条件	-
被调用函数	-
输入参数{in}	
cmp_periph	参考枚举 表 3-126. 枚举类型cmp_enum
输入参数{in}	
flag	CMP中断标志位
CMP_FLAG_COMPARE	CMP0比较中断标志位
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* clear the CMP0 interrupt flag bit */
```

```
cmp_flag_clear(CMP0, CMP_FLAG_COMPARE);
```

函数 cmp_interrupt_enable

函数cmp_interrupt_enable描述见下表：

表 3-142. 函数 cmp_interrupt_enable

函数名称	cmp_interrupt_enable
函数原形	void cmp_interrupt_enable(cmp_enum cmp_periph, uint32_t interrupt);
功能描述	CMP中断使能
先决条件	-
被调用函数	-
输入参数{in}	

cmp_periph	参考枚举 表 3-126. 枚举类型cmp_enum
输入参数{in}	
interrupt	CMP中断
<i>CMP_INT_COMPARE</i>	CMP比较中断
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* enable the CMP0 interrupt */
```

```
cmp_interrupt_enable(CMP0, CMP_INT_COMPARE);
```

函数 **cmp_interrupt_disable**

函数cmp_interrupt_disable描述见下表：

表 3-143. 函数 cmp_interrupt_disable

函数名称	cmp_interrupt_disable
函数原形	void cmp_interrupt_disable(cmp_enum cmp_periph, uint32_t interrupt);
功能描述	CMP中断禁能
先决条件	-
被调用函数	-
输入参数{in}	
cmp_periph	参考枚举 表 3-126. 枚举类型cmp_enum
输入参数{in}	
interrupt	CMP中断
<i>CMP_INT_COMPARE</i>	CMP比较中断
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* disable the CMP0 interrupt */
```

```
cmp_interrupt_disable(CMP0, CMP_INT_COMPARE);
```

函数 **cmp_interrupt_flag_get**

函数cmp_interrupt_flag_get描述见下表：

表 3-144. 函数 cmp_interrupt_flag_get

函数名称	cmp_interrupt_flag_get
函数原形	FlagStatus cmp_interrupt_flag_get(cmp_enum cmp_periph, uint32_t flag);
功能描述	获取CMP中断标志位
先决条件	-
被调用函数	-
输入参数{in}	
cmp_periph	参考枚举表 3-126. 枚举类型cmp_enum
输入参数{in}	
flag	CMP中断标志位
CMP_INT_FLAG_C OMPARE	CMP比较中断标志位
输出参数{out}	
-	-
返回值	
FlagStatus	SET或RESET

例如:

```
/* get the CMP0 interrupt bit*/
```

```
FlagStatus flag_value;
```

```
flag_value = cmp_interrupt_flag_get(CMP0, CMP_INT_FLAG_COMPARE);
```

函数 cmp_interrupt_flag_clear

函数cmp_interrupt_flag_clear描述见下表:

表 3-145. 函数 cmp_interrupt_flag_clear

函数名称	cmp_interrupt_flag_clear
函数原形	void cmp_interrupt_flag_clear(cmp_enum cmp_periph, uint32_t flag);
功能描述	清除CMP中断标志位
先决条件	-
被调用函数	-
输入参数{in}	
cmp_periph	参考枚举表 3-126. 枚举类型cmp_enum
输入参数{in}	
flag	CMP中断标志位
CMP_INT_FLAG_C OMPARE	CMP比较中断标志位
输出参数{out}	
-	-
返回值	
-	-

例如:


```
/* clear the CMP0 interrupt bit*/
```

```
cmp_interrupt_flag_clear(CMP0, CMP_INT_FLAG_COMPARE);
```

3.7. 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[3.7.1](#)描述了CRC的寄存器列表，章节[3.7.2](#)对CRC库函数进行说明。

3.7.1. 外设寄存器说明

CRC寄存器列表如下表所示：

表 3-146. CRC 寄存器

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

3.7.2. 外设库函数说明

CRC库函数列表如下表所示：

表 3-147. CRC 库函数

库函数名称	库函数描述
crc_deinit	复位CRC计算单元
crc_init_data_register_write	写初值寄存器
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_reverse_output_data_disable	除能输出数据翻转功能
crc_reverse_output_data_enable	使能输出数据翻转功能
crc_input_data_reverse_config	配置输入数据翻转功能
crc_data_register_reset	根据数据寄存器的复位值复位数据寄存器
crc_polynomial_size_set	配置多项式长度
crc_polynomial_set	设置多项式寄存器数据
crc_single_data_calculate	CRC计算一个32位数据
crc_block_data_calculate	CRC计算一个32位数组

函数 crc_deinit

函数crc_deinit描述见下表：

表 3-148. 函数 `crc_deinit`

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

```
/* deinit CRC calculation unit */
crc_deinit();
```

函数 `crc_init_data_register_write`

函数`crc_init_data_register_write`描述见下表：

表 3-149. 函数 `crc_init_data_register_write`

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

例如：

```
/* write crc initializaiton data register */
crc_init_data_register_write(0x11223344);
```

函数 `crc_data_register_read`

函数`crc_data_register_read`描述见下表：

表 3-150. 函数 `crc_data_register_read`

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

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

例如：

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

函数 crc_free_data_register_read

函数crc_free_data_register_read描述见下表：

表 3-151. 函数 crc_free_data_register_read

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

例如：

```
/* read crc free data register */
uint8_t crc_value = 0;
crc_value = crc_free_data_register_read();
```

函数 crc_free_data_register_write

函数crc_free_data_register_write描述见下表：

表 3-152. 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_reverse_output_data_disable`

函数`crc_reverse_output_data_disable`描述见下表：

表 3-153. 函数 `crc_reverse_output_data_disable`

函数名称	<code>crc_reverse_output_data_disable</code>
函数原形	<code>void crc_reverse_output_data_disable(void);</code>
功能描述	除能输出数据翻转功能
先决条件	-
被调用函数	-
输入参数{in}	
-	-
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* disable crc reverse operation of output data */
crc_reverse_output_data_disable();
```

函数 `crc_reverse_output_data_enable`

函数`crc_reverse_output_data_enable`描述见下表：

表 3-154. 函数 `crc_reverse_output_data_enable`

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

函数原形	void crc_reverse_output_data_enable(void);
功能描述	使能输出数据翻转功能
先决条件	-
被调用函数	-
输入参数{in}	
-	-
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* enable CRC reverse operation of output data */
```

```
crc_reverse_output_data_enable();
```

函数 crc_input_data_reverse_config

函数crc_input_data_reverse_config描述见下表：

表 3-155. 函数 crc_input_data_reverse_config

函数名称	crc_input_data_reverse_config
函数原形	void crc_input_data_reverse_config(uint32_t data_reverse)
功能描述	配置输入数据翻转功能
先决条件	-
被调用函数	-
输入参数{in}	
data_reverse	设定的输入数据翻转功能
CRC_INPUT_DATA _NOT	输入数据不翻转
CRC_INPUT_DATA _BYTE	输入数据按字节翻转
CRC_INPUT_DATA _HALFWORD	输入数据按半字翻转
CRC_INPUT_DATA _WORD	输入数据按字翻转
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* configure the crc input data */
```

```
crc_input_data_reverse_config(CRC_INPUT_DATA_WORD);
```

函数 `crc_data_register_reset`

函数 `crc_data_register_reset` 描述见下表：

表 3-156. 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_polynomial_size_set`

函数 `crc_polynomial_size_set` 描述见下表：

表 3-157. 函数 `crc_polynomial_size_set`

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

例如：

```
/* configure the CRC polynomial size */
```

```
crc_polynomial_size_set(CRC_CTL_PS_7);
```

函数 `crc_polynomial_set`

函数 `crc_polynomial_set` 描述见下表：

表 3-158. 函数 `crc_polynomial_set`

函数名称	<code>crc_polynomial_set</code>
函数原形	<code>void crc_polynomial_set(uint32_t poly)</code>
功能描述	设置多项式寄存器值
先决条件	-
被调用函数	-
输入参数{in}	
poly	设置多项式长度寄存器值
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* configure the CRC polynomial value */
```

```
crc_polynomial_set(0x11223344);
```

函数 `crc_single_data_calculate`

函数 `crc_single_data_calculate` 描述见下表：

表 3-159. 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, uint8_t data_format);</code>
功能描述	CRC 计算一个 32 位数据
先决条件	-
被调用函数	-
输入参数{in}	
sdata	设定的 32 位数据
输入参数{in}	
data_format	数据格式
<code>INPUT_FORMAT_WORD</code>	输入数据格式为字
<code>INPUT_FORMAT_HALFWORD</code>	输入数据格式为半字
<code>INPUT_FORMAT_BYTE</code>	输入数据格式为字节
输出参数{out}	
-	-

返回值	
uint32_t	32位CRC计算结果（0-0xFFFFFFFF）

例如：

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

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

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

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

函数 **crc_block_data_calculate**

函数 **crc_block_data_calculate** 描述见下表：

表 3-160. 函数 **crc_block_data_calculate**

函数名称	crc_block_data_calculate
函数原形	<code>uint32_t crc_block_data_calculate(void *array, uint32_t size, uint8_t data_format);</code>
功能描述	CRC计算一个32位数组
先决条件	-
被调用函数	-
输入参数{in}	
array	32位数据数组
输入参数{in}	
size	数据长度
输入参数{in}	
data_format	数据格式
INPUT_FORMAT_WORD	输入数据格式为字
INPUT_FORMAT_HALFWORD	输入数据格式为半字
INPUT_FORMAT_BYTE	输入数据格式为字节
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(data_buffer, BUFFER_SIZE, INPUT_FORMAT_WORD);
```

3.8. CTC

CTC模块基于外部高精度的参考信号源来校准IRC48M的时钟频率，通过自动的或手动的调整校准值，以得到一个精准的IRC48M时钟。章节[3.8.1](#)描述了CTC的寄存器列表，章节[3.8.2](#)对CTC库函数进行说明。

3.8.1. 外设寄存器说明

CTC寄存器列表如下表所示：

表 3-161. CTC 寄存器

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

3.8.2. 外设库函数说明

CTC库函数列表如下表所示：

表 3-162. CTC 库函数

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

库函数名称	库函数描述
ctc_interrupt_enable	CTC中断使能
ctc_interrupt_disable	CTC中断禁能
ctc_interrupt_flag_get	CTC中断标志获取
ctc_interrupt_flag_clear	CTC中断标志清除
ctc_flag_get	CTC状态标志获取
ctc_flag_clear	CTC状态标志清除

函数 ctc_deinit

函数ctc_deinit描述见下表：

表 3-163. 函数 ctc_deinit

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

例如：

```
/* reset CTC */
```

```
ctc_deinit();
```

函数 ctc_counter_enable

函数ctc_counter_enable描述见下表：

表 3-164. 函数 ctc_counter_enable

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

-	-
---	---

例如:

```
/* enable CTC trim counter*/
```

```
ctc_counter_enable ();
```

函数 ctc_counter_disable

函数ctc_counter_disable描述见下表:

表 3-165. 函数 ctc_counter_disable

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

例如:

```
/* disable CTC trim counter */
```

```
ctc_counter_disable ();
```

函数 ctc_irc48m_trim_value_config

函数ctc_irc48m_trim_value_config描述见下表:

表 3-166. 函数 ctc_irc48m_trim_value_config

函数名称	ctc_irc48m_trim_value_config
函数原形	void ctc_irc48m_trim_value_config(uint32_t trim_value);
功能描述	配置IRC48M时钟校准值
先决条件	-
被调用函数	-
输入参数{in}	
trim_value	0~63
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* IRC48M trim value configuration */
```

```
ctc_irc48m_trim_value_config (0x01);
```

函数 ctc_software_refsource_pulse_generate

函数ctc_software_refsource_pulse_generate描述见下表:

表 3-167. 函数 ctc_software_refsource_pulse_generate

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

例如:

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

```
ctc_software_refsource_pulse_generate ();
```

函数 ctc_hardware_trim_mode_config

函数ctc_hardware_trim_mode_config描述见下表:

表 3-168. 函数 ctc_hardware_trim_mode_config

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

-	-
返回值	
-	-

例如：

```
/* enable CTC hardware trim */
```

```
ctc_hardware_trim_mode_config (CTC_HARDWARE_TRIM_MODE_ENABLE);
```

函数 ctc_refsource_polarity_config

函数ctc_refsource_polarity_config描述见下表：

表 3-169. 函数 ctc_refsource_polarity_config

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

例如：

```
/* set reference source polarity */
```

```
ctc_refsource_polarity_config (CTC_REFSOURCE_POLARITY_RISING);
```

函数 ctc_refsource_signal_select

函数ctc_refsource_signal_select描述见下表：

表 3-170. 函数 ctc_refsource_signal_select

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

被调用函数	-
输入参数{in}	
refs	参考信号源
CTC_REFSOURCE_GPIO	选择GPIO输入信号
CTC_REFSOURCE_LXTAL	选择LXTAL时钟
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* reference signal selection */
```

```
ctc_refsource_signal_select (CTC_REFSOURCE_LXTAL);
```

函数 ctc_refsource_prescaler_config

函数ctc_refsource_prescaler_config描述见下表：

表 3-171. 函数 ctc_refsource_prescaler_config

函数名称	ctc_refsource_prescaler_config
函数原形	void ctc_refsource_prescaler_config(uint32_t prescaler);
功能描述	参考信号源的分频设置
先决条件	-
被调用函数	-
输入参数{in}	
prescaler	分频系数
CTC_REFSOURCE_PSC_OFF	参考信号不分频
CTC_REFSOURCE_PSC_DIV2	参考信号2分频
CTC_REFSOURCE_PSC_DIV4	参考信号4分频
CTC_REFSOURCE_PSC_DIV8	参考信号8分频
CTC_REFSOURCE_PSC_DIV16	参考信号16分频
CTC_REFSOURCE_PSC_DIV32	参考信号32分频
CTC_REFSOURCE_PSC_DIV64	参考信号64分频
CTC_REFSOURCE_PSC_DIV128	参考信号128分频

输出参数{out}	
-	-
返回值	
-	-

例如：

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

函数 ctc_clock_limit_value_config

函数ctc_clock_limit_value_config描述见下表：

表 3-172. 函数 ctc_clock_limit_value_config

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

例如：

```
/* configure clock trim base limit value */
ctc_clock_limit_value_config (0x1F);
```

函数 ctc_counter_reload_value_config

函数ctc_counter_reload_value_config描述见下表：

表 3-173. 函数 ctc_counter_reload_value_config

函数名称	ctc_counter_reload_value_config
函数原形	void ctc_counter_reload_value_config(uint16_t reload_value);
功能描述	CTC计数器重载值设置
先决条件	-
被调用函数	-
输入参数{in}	
reload_value	0x0000 - 0xFFFF
输出参数{out}	
-	-

返回值	
-	-

例如：

```
/* configure CTC counter reload value */
```

```
ctc_counter_reload_value_config (0x00FF);
```

函数 ctc_counter_capture_value_read

函数ctc_counter_capture_value_read描述见下表：

表 3-174. 函数 ctc_counter_capture_value_read

函数名称	ctc_counter_capture_value_read
函数原形	uint16_t ctc_counter_capture_value_read(void);
功能描述	读取计数器捕获值
先决条件	-
被调用函数	-
输入参数{in}	
-	-
输出参数{out}	
-	-
返回值	
uint16_t	读取计数器捕获值(0x0000 - 0xFFFF)

例如：

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

```
uint16_t ctc_value = 0;
```

```
ctc_value = ctc_counter_capture_value_read ();
```

函数 ctc_counter_direction_read

函数ctc_counter_direction_read描述见下表：

表 3-175. 函数 ctc_counter_direction_read

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

FlagStatus	SET(向下计数) / RESET(向上计数)
------------	-------------------------

例如:

```
/* read ctc counter direction */

FlagStatus ctc_direction = SET;

ctc_direction = ctc_counter_direction_read ();
```

函数 ctc_counter_reload_value_read

函数ctc_counter_reload_value_read描述见下表:

表 3-176. 函数 ctc_counter_reload_value_read

函数名称	ctc_counter_reload_value_read
函数原形	uint16_t ctc_counter_reload_value_read(void);
功能描述	读取CTC计数器重载值
先决条件	-
被调用函数	-
输入参数{in}	
-	-
输出参数{out}	
-	-
返回值	
uint16_t	读取计数器重载值的16位数据 (0x0000 - 0xFFFF)

例如:

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

uint16_t ctc_reload_value = 0;

ctc_reload_value = ctc_counter_reload_value_read ();
```

函数 ctc_irc48m_trim_value_read

函数ctc_irc48m_trim_value_read描述见下表:

表 3-177. 函数 ctc_irc48m_trim_value_read

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

返回值	
uint8_t	6位IRC48M校准值 (0-63)

例如:

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

uint8_t ctc_trim_value = 0;

ctc_trim_value = ctc_irc48m_trim_value_read ();
```

函数 ctc_interrupt_enable

函数ctc_interrupt_enable描述见下表:

表 3-178. 函数 ctc_interrupt_enable

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

例如:

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

ctc_interrupt_enable (CTC_INT_CKOK);
```

函数 ctc_interrupt_disable

函数ctc_interrupt_disable描述见下表:

表 3-179. 函数 ctc_interrupt_disable

函数名称	ctc_interrupt_disable
函数原形	void ctc_interrupt_disable(uint32_t interrupt);
功能描述	禁能外设CTC中断
先决条件	-
被调用函数	-

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

例如：

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

```
ctc_interrupt_disable (CTC_INT_CKOK);
```

函数 **ctc_interrupt_flag_get**

函数ctc_interrupt_flag_get描述见下表：

表 3-180. 函数 ctc_interrupt_flag_get

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

返回值	
FlagStatus	SET或RESET

例如:

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

```
FlagStatus state = ctc_interrupt_flag_get (CTC_INT_FLAG_CKOK);
```

函数 ctc_interrupt_flag_clear

函数ctc_interrupt_flag_clear描述见下表:

表 3-181. 函数 ctc_interrupt_flag_clear

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

例如:

```
/*clear CTC interrupt flag status */
```

```
ctc_interrupt_flag_clear (CTC_INT_FLAG_CKOK);
```

函数 ctc_flag_get

函数ctc_flag_get描述见下表：

表 3-182. 函数 ctc_flag_get

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

例如：

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

```
FlagStatus state = ctc_flag_get (CTC_FLAG_CKOK);
```

函数 ctc_flag_clear

函数ctc_flag_clear描述见下表：

表 3-183. 函数 ctc_flag_clear

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

CTC_FLAG_CKOK	时钟校准完成标志位
CTC_FLAG_CKWA RN	时钟校准警告中断标志位
CTC_FLAG_ERR	错误中断标志位
CTC_FLAG_EREFS	期望参考信号中断标志位
CTC_FLAG_CKER R	时钟校准错误位
CTC_FLAG_REFMIS	参考同步脉冲信号丢失
CTC_FLAG_TRIMERR	校准值错误位
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* clear CTC flag status */
```

```
ctc_flag_clear (CTC_FLAG_CKOK);
```

3.9. 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[3.9.1](#)描述了DAC的寄存器列表，章节[3.9.2](#)对DAC库函数进行说明。

3.9.1. 外设寄存器说明

DAC寄存器列表如下表所示：

表 3-184. 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_OUT0_DO	DACx_OUT0数据输出寄存器
DAC_STAT0	DAC状态寄存器

3.9.2. 外设库函数说明

DAC库函数列表如下表所示：

表 3-185. DAC 库函数

库函数名称	库函数描述
dac_deinit	DAC外设复位
dac_enable	DAC使能
dac_disable	DAC禁能
dac_dma_enable	DAC的DMA功能使能
dac_dma_disable	DAC的DMA功能禁能
dac_pin_select	DAC输出引脚选择
dac_output_buffer_enable	DAC输出缓冲区使能
dac_output_buffer_disable	DAC输出缓冲区禁能
dac_output_value_get	DAC输出数据获取
dac_data_set	DAC输出数据设置
dac_trigger_enable	DAC触发使能
dac_trigger_disable	DAC触发禁能
dac_trigger_source_config	DAC触发源选择
dac_software_trigger_enable	DAC软件触发使能
dac_wave_mode_config	DAC噪声波模式选择
dac_lfsr_noise_config	DAC LFSR模式设置
dac_triangle_noise_config	DAC三角波模式设置
dac_output_connect_to_pin_enable	DAC连接到引脚使能
dac_output_connect_to_pin_disable	DAC连接到引脚禁能
dac_flag_get	获取DAC标志位
dac_flag_clear	清除DAC标志位
dac_interrupt_enable	DAC中断使能
dac_interrupt_disable	DAC中断禁能
dac_interrupt_flag_get	获取DAC中断标志位
dac_interrupt_flag_clear	清除DAC中断标志位

函数 dac_deinit

函数dac_deinit描述见下表：

表 3-186. 函数 dac_deinit

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

-	-
---	---

例如:

```
/* deinitialize DAC0 */
```

```
dac_deinit(DAC0);
```

函数 **dac_enable**

函数dac_enable描述见下表:

表 3-187. 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)
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 3-188. 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
<i>DAC_OUTx</i>	DAC输出通道选择 (x = 0)
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 3-189. 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
<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)
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 3-190. 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
<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)
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* disable DAC0_OUT0 DMA function */
```

```
dac_dma_disable(DAC0, DAC_OUT0);
```

函数 dac_pin_select

函数dac_pin_sel描述见下表:

表 3-191. 函数 dac_dma_disable

函数名称	dac_pin_select
函数原型	void dac_pin_select(uint32_t dac_periph, uint8_t dac_out, uint32_t pin_sel);
功能描述	DAC的输出引脚选择
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)
输入参数{in}	
pin_sel	引脚选择
<i>DAC_OUTPUT_CO NNECT_PA4</i>	DAC输出引脚连接到PA4
<i>DAC_OUTPUT_CO NNECT_PA5</i>	DAC输出引脚连接到PA5
输出参数{out}	
-	-
返回值	

-	-
---	---

例如:

```
/* DAC output pin select PA4 */
```

```
dac_pin_select (DAC0, DAC_OUT0, DAC_OUTPUT_CONNECT_PA4);
```

函数 **dac_output_buffer_enable**

函数dac_output_buffer_enable描述见下表:

表 3-192. 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)
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 3-193. 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)
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 3-194. 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)
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 3-195. 函数 dac_data_set

函数名称	dac_data_set
------	--------------

函数原型	void dac_data_set(uint32_t dac_periph, uint8_t dac_out, uint32_t dac_align, uint16_t data);
功能描述	DAC输出数据设置
先决条件	-
被调用函数	-
输入参数{in}	
dac_periph	DAC外设
DACx	DAC外设选择 (x = 0)
输入参数{in}	
dac_out	DAC输出
DAC_OUTx	DAC输出通道选择 (x = 0)
输入参数{in}	
dac_align	DAC对齐模式
DAC_ALIGN_12B_R	12位数据右对齐
DAC_ALIGN_12B_L	12位数据左对齐
DAC_ALIGN_8B_R	8位数据右对齐
输入参数{in}	
data	写入DAC_OUTx的数据 (0~4095)
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* set DAC0_OUT0 data holding register value */
```

```
dac_data_set(DAC0, DAC_OUT0, DAC_ALIGN_8B_R, 0xFF);
```

函数 dac_trigger_enable

函数dac_trigger_enable描述见下表:

表 3-196. 函数 dac_trigger_enable

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

<i>DAC_OUTx</i>	DAC输出通道选择 (x = 0)
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 3-197. 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)
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 3-198. 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触发源选择

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)
输入参数{in}	
triggersource	DAC触发源
<i>DAC_TRIGGER_EXTERNAL</i>	外部触发从TRIGSEL中选择
<i>DAC_TRIGGER_SOFTWARE</i>	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 3-199. 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
<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)
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 3-200. 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噪声波模式选择
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)
输入参数{in}	
wave_mode	噪声波模式选择
DAC_WAVE_DISABLE	噪声波模式禁能
DAC_WAVE_MODE_LFSR	LFSR噪声波模式
DAC_WAVE_MODE_TRIANGLE	三角波噪声波模式
输出参数{out}	
-	-
返回值	
-	-

例如:

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

函数 dac_lfsr_noise_config

函数dac_lfsr_noise_config描述见下表:

表 3-201. 函数 dac_lfsr_noise_config

函数名称	dac_lfsr_noise_config
------	-----------------------

函数原型	void dac_lfsr_noise_config(uint32_t dac_periph, uint8_t dac_out, uint32_t unmask_bits);
功能描述	DAC LFSR模式设置
先决条件	-
被调用函数	-
输入参数{in}	
dac_periph	DAC外设
DACx	DAC外设选择 (x = 0)
输入参数{in}	
dac_out	DAC输出
DAC_OUTx	DAC输出通道选择 (x = 0)
输入参数{in}	
unmask_bits	噪声波的非屏蔽位宽
DAC_LFSR_BIT0	LFSR模式位0非屏蔽
DAC_LFSR_BITSx_0	LFSR模式位[x:0]非屏蔽 (x = 1,2,3..11)
输出参数{out}	
-	-
返回值	
-	-

例如:

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

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

函数 dac_triangle_noise_config

函数dac_triangle_noise_config描述见下表:

表 3-202. 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三角波模式设置
先决条件	-
被调用函数	-
输入参数{in}	
dac_periph	DAC外设
DACx	DAC外设选择 (x = 0)
输入参数{in}	
dac_out	DAC输出
DAC_OUTx	DAC输出通道选择 (x = 0)
输入参数{in}	

amplitude	三角波幅值
<i>DAC_TRIANGLE_AMPLITUDE_x</i>	$x = 2^n - 1$ ($n = 1..12$)
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* configure DAC0_OUT0 triangle noise mode */
dac_triangle_noise_config(DAC0, DAC_OUT0, DAC_TRIANGLE_AMPLITUDE_1);
```

函数 dac_output_connect_to_pin_enable

函数dac_output_connect_to_pin_enable描述见下表:

表 3-203. 函数 dac_output_connect_to_pin_enable

函数名称	dac_output_connect_to_pin_enable
函数原型	void dac_output_connect_to_pin_enable(uint32_t dac_periph, uint8_t dac_out);
功能描述	DAC连接到引脚使能
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 ($x = 0$)
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* enable DAC0 output connect to pin */
dac_output_connect_to_pin_enable (DAC0, DAC_OUT0);
```

函数 dac_output_connect_to_pin_disable

函数dac_output_connect_to_pin_disable描述见下表:

表 3-204. 函数 dac_output_connect_to_pin_disable

函数名称	dac_output_connect_to_pin_disable
函数原型	void dac_output_connect_to_pin_disable(uint32_t dac_periph, uint8_t dac_out);

功能描述	DAC连接到引脚使能
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)
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* disable DAC0 output connect to pin */
```

```
dac_output_connect_to_pin_disable (DAC0, DAC_OUT0);
```

函数 dac_flag_get

函数dac_flag_get描述见下表:

表 3-205. 函数 dac_flag_get

函数名称	dac_flag_get
函数原型	FlagStatus dac_flag_get(uint32_t dac_periph, uint32_t flag);
功能描述	DAC标志位获取
先决条件	-
被调用函数	-
输入参数{in}	
dac_periph	DAC外设
DACx	DAC外设选择 (x = 0,1,2,3)
输入参数{in}	
flag	DAC状态标志位
DAC_FLAG_DDUD R0	DACx_OUT0 DMA欠载标志位
输出参数{out}	
-	-
返回值	
FlagStatus	DAC位状态 (SET或RESET)

例如:

```
/* get DAC0 flag */
```

```
FlagStatus flag;
```

```
flag = dac_flag_get(DAC0, DAC_FLAG_DDUDR0);
```

函数 `dac_flag_clear`

函数 `dac_flag_clear` 描述见下表：

表 3-206. 函数 `dac_flag_clear`

函数名称	<code>dac_flag_clear</code>
函数原型	<code>void dac_flag_clear(uint32_t dac_periph, uint32_t flag);</code>
功能描述	DAC标志位清除
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（ $x = 0, 1, 2, 3$ ）
输入参数{in}	
<code>flag</code>	DAC状态标志位
<code>DAC_FLAG_DDUDR0</code>	DACx_OUT0 DMA欠载标志位
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* clear DAC0 flag */
```

```
dac_flag_clear(DAC0, DAC_FLAG_DDUDR0);
```

函数 `dac_interrupt_enable`

函数 `dac_interrupt_enable` 描述见下表：

表 3-207. 函数 `dac_interrupt_enable`

函数名称	<code>dac_interrupt_enable</code>
函数原型	<code>void dac_interrupt_enable(uint32_t dac_periph, uint32_t interrupt);</code>
功能描述	DAC中断使能
先决条件	-
被调用函数	-
输入参数{in}	
<code>dac_periph</code>	DAC外设
<code>DACx</code>	DAC外设选择（ $x = 0, 1, 2, 3$ ）
输入参数{in}	
<code>interrupt</code>	DAC中断
<code>DAC_INT_DDUDR0</code>	DACx_OUT0 DMA欠载中断

输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* enable DAC0 interrupt */
```

```
dac_interrupt_enable (DAC0, DAC_INT_DDUDR0);
```

函数 dac_interrupt_disable

函数dac_interrupt_disable描述见下表：

表 3-208. 函数 dac_interrupt_disable

函数名称	dac_interrupt_disable
函数原型	void dac_interrupt_disable(uint32_t dac_periph, uint32_t interrupt);
功能描述	DAC中断禁能
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 = 0,1,2,3）
输入参数{in}	
interrupt	DAC中断
DAC_INT_DDUDR0	DACx_OUT0 DMA欠载中断
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* disable DAC0 interrupt */
```

```
dac_interrupt_disable (DAC0, DAC_INT_DDUDR0);
```

函数 dac_interrupt_flag_get

函数dac_interrupt_flag_get描述见下表：

表 3-209. 函数 dac_interrupt_flag_get

函数名称	dac_interrupt_flag_get
函数原型	FlagStatus dac_interrupt_flag_get(uint32_t dac_periph, uint32_t int_flag);
功能描述	DAC中断标志位获取
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 = 0,1,2,3)
输入参数{in}	
int_flag	DAC中断标志位
<i>DAC_INT_FLAG_D</i> <i>DUDR0</i>	DACx_OUT0 DMA欠载中断标志位
输出参数{out}	
-	-
返回值	
FlagStatus	DAC中断状态 (SET或RESET)

例如:

```
/* get DAC0 interrupt flag */
```

```
FlagStatus flag;
```

```
flag = dac_interrupt_flag_get (DAC0, DAC_INT_FLAG_DDUDR0);
```

函数 **dac_interrupt_flag_clear**

函数dac_interrupt_flag_clear描述见下表:

表 3-210. 函数 **dac_interrupt_flag_clear**

函数名称	dac_interrupt_flag_clear
函数原型	void dac_interrupt_flag_clear(uint32_t dac_periph, uint32_t int_flag);
功能描述	DAC中断标志位清除
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 = 0,1,2,3)
输入参数{in}	
int_flag	DAC中断标志位
<i>DAC_INT_FLAG_D</i> <i>DUDR0</i>	DACx_OUT0 DMA欠载中断标志位
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* clear DAC0 interrupt flag */
```

```
dac_interrupt_flag_clear (DAC0, DAC_INT_FLAG_DDUDR0);
```

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

3.10.1. 外设寄存器说明

DBG寄存器列表如下表所示：

表 3-211. DBG 寄存器

寄存器名称	寄存器描述
DBG_ID	DBG标识号寄存器
DBG_CTL	DBG控制寄存器

3.10.2. 外设库函数说明

DBG库函数列表如下表所示：

表 3-212. DBG 库函数

库函数名称	库函数描述
dbg_deinit	复位DBG寄存器
dbg_id_get	读DBG_ID寄存器
dbg_low_power_enable	使能低功耗模式的MCU调试保持功能
dbg_low_power_disable	禁用低功耗模式的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	禁用外设的MCU调试保持功能
dbg_trace_pin_enable	使能跟踪引脚分配
dbg_trace_pin_disable	禁用跟踪引脚分配
dbg_trace_pin_mode_set	配置跟踪引脚分配模式

枚举类型 dbg_periph_enum

表 3-213. 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，保持CAN0计数器计数值不变
DBG_I2C0_HOLD	当内核停止时，保持I2C0的SMBUS状态不变，用于调试
DBG_I2C1_HOLD	当内核停止时，保持I2C1的SMBUS状态不变，用于调试

成员名称	功能描述
DBG_TIMER7_HOLD	当内核停止时，保持TIMER7计数器计数值不变
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，保持CAN1计数器计数值不变
DBG_TIMER15_HOLD	当内核停止时，保持TIMER15计数器计数值不变
DBG_TIMER16_HOLD	当内核停止时，保持TIMER16计数器计数值不变

函数 dbg_deinit

函数dbg_deinit描述见下表：

表 3-214. 函数 dbg_deinit

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

例如：

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

函数 dbg_id_get

函数dbg_id_get描述见下表：

表 3-215. 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	
-	-

例如：

```
/* read DBG_ID code */

uint32_t id = 0;

id = dbg_id_get();
```

函数 dbg_low_power_enable

函数dbg_low_power_enable描述见下表：

表 3-216. 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_STANDBY);
```

函数 dbg_low_power_disable

函数dbg_low_power_disable描述见下表：

表 3-217. 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禁止
DBG_LOW_POWER_SLEEP	在睡眠模式下，不保持调试器连接，无法进行调试
DBG_LOW_POWER_DEEPSLEEP	在深度睡眠模式下，不保持调试器连接，无法进行调试
DBG_LOW_POWER_STANDBY	在待机模式下，不保持调试器连接，无法进行调试
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_STANDBY);
```

函数 dbg_periph_enable

函数dbg_periph_enable描述见下表：

表 3-218. 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	参考枚举变量 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，保持CANx (x=0,1)计数器计数值不变
DBG_I2Cx_HOLD	当内核停止时，保持I2Cx (x=0,1)的SMBUS状态不变，用于调试
DBG_TIMERx_HOLD	当内核停止时，保持TIMERx (x=0,1,2,3,4,5,6,7,15,16)计数器计数值不变
输出参数{out}	
-	-
返回值	
-	-

例如：

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

```
dbg_periph_enable(DBG_TIMER1_HOLD);
```

函数 dbg_periph_disable

函数dbg_periph_disable描述见下表：

表 3-219. 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	参考枚举变量 表3-213. 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，保持CANx (x=0,1)计数器计数值不变
DBG_I2Cx_HOLD	当内核停止时，保持I2Cx (x=0,1)的SMBUS状态不变，用于调试
DBG_TIMERx_HOLD	当内核停止时，保持TIMERx (x=0,1,2,3,4,5,6,7,15,16)计数器计数值不变
输出参数{out}	
-	-
返回值	
-	-

例如：

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

```
dbg_periph_disable(DBG_TIMER1_HOLD);
```

函数 dbg_trace_pin_enable

函数dbg_trace_pin_enable描述见下表：

表 3-220. 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 3-221. 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 3-222. 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

C	
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 used for async mode */
```

```
dbg_trace_pin_mode_set(TRACE_MODE_ASYNC);
```

3.11. DMA / DMAMUX

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

DMAMUX是DMA请求的传输调度器。可编程的DMA请求多路复用器DMAMUX，可在外设和DMA控制器之间路由DMA请求线路，或者DMAMUX也可以将可编程事件连入到输入触发信号上，作为一个DMAMUX请求发生器，再由DMAMUX请求路由器在DMAMUX请求生成器产生的DMA请求和DMA控制器之间路由DMA请求线路。章节[3.11.1](#)描述了DMAMUX的寄存器列表，章节[3.11.2](#)对DMAMUX库函数进行说明。

3.11.1. 外设寄存器说明

DMA寄存器列表如下表所示：

表 3-223. DMA 寄存器

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

寄存器名称	寄存器描述
(x=0..6)	

DMAMUX寄存器列表如下表所示:

表 3-224. DMAMUX 寄存器

寄存器名称	寄存器描述
DMAMUX_RM_CHx CFG (x=0..11)	请求路由通道x配置寄存器
DMAMUX_RM_INT F	请求路由通道中断标志位寄存器
DMAMUX_RM_INT C	请求路由通道中断标志位清除寄存器
DMAMUX_RG_CHx CFG (x=0..3)	请求生成通道x配置寄存器
DMAMUX_RG_INT F	请求生成通道中断标志位寄存器
DMAMUX_RG_INT C	请求生成通道中断标志位清除寄存器

3.11.2. 外设库函数说明

DMA库函数列表如下表所示:

表 3-225. DMA 库函数

库函数名称	库函数描述
dma_deinit	复位外设DMA通道x的所有寄存器
dma_struct_para_init	将DMA结构体中所有参数初始化为默认值
dma_init	初始化外设DMA的通道x
dma_circulation_enable	使能DMA循环模式
dma_circulation_disable	禁能DMA循环模式
dma_memory_to_memory_enable	使能存储器到存储器DMA传输
dma_memory_to_memory_disable	禁能存储器到存储器DMA传输
dma_channel_enable	使能DMA通道x传输
dma_channel_disable	禁能DMA通道x传输
dma_periph_address_config	配置DMA通道x传输的外设基地址
dma_memory_address_config	配置DMA通道x传输的存储器基地址
dma_transfer_number_config	配置DMA通道x还有多少数据要传输
dma_transfer_number_get	获取DMA通道x还有多少数据要传输
dma_priority_config	配置DMA通道x的传输软件优先级
dma_memory_width_config	配置DMA通道x传输的存储器数据宽度
dma_periph_width_config	配置DMA通道x传输的外设数据宽度
dma_memory_increase_enable	使能DMA通道x传输的存储器地址生成算法增量模式
dma_memory_increase_disable	禁能DMA通道x传输的存储器地址生成算法增量模式
dma_periph_increase_enable	使能DMA通道x传输的外设地址生成算法增量模式

库函数名称	库函数描述
dma_periph_increase_disable	禁能DMA通道x传输的外设地址生成算法增量模式
dma_transfer_direction_config	配置DMA通道x的传输方向
dma_flag_get	获取DMA通道x标志位状态
dma_flag_clear	清除DMA通道x标志位状态
dma_interrupt_enable	使能DMA通道x中断
dma_interrupt_disable	禁能DMA通道x中断
dma_interrupt_flag_get	获取DMA通道x中断标志位状态
dma_interrupt_flag_clear	清除DMA通道x中断标志位状态

DMAMUX库函数列表如下表所示：

表 3-226. DMAMUX 库函数

库函数名称	库函数描述
dmamux_sync_struct_para_init	将DMAMUX同步结构体中所有参数初始化为默认值
dmamux_synchronization_init	初始化DMAMUX同步结构体通道x
dmamux_synchronization_enable	使能DMAMUX同步模式
dmamux_synchronization_disable	禁能DMAMUX同步模式
dmamux_event_generation_enable	使能DMAMUX事件输出
dmamux_event_generation_disable	禁能DMAMUX事件输出
dmamux_gen_struct_para_init	将DMAMUX请求生成结构体中所有参数初始化为默认值
dmamux_request_generator_init	初始化DMAMUX请求生成结构体通道x
dmamux_request_generator_channel_enable	使能DMAMUX请求生成通道x
dmamux_request_generator_channel_disable	禁能DMAMUX请求生成通道x
dmamux_synchronization_polarity_config	配置DMAMUX同步输入的有效边沿
dmamux_request_forward_number_config	配置DMAMUX通道x要传输多少个DMA请求
dmamux_sync_id_config	配置DMAMUX同步输入标识
dmamux_request_id_config	配置DMAMUX请求路由通道输入标识
dmamux_trigger_polarity_config	配置DMAMUX触发输入的有效边沿
dmamux_request_generate_number_config	配置DMAMUX请求生成器生成请求的数量
dmamux_trigger_id_config	配置DMAMUX触发输入标识
dmamux_flag_get	获取DMAMUX通道x标志位状态
dmamux_flag_clear	清除DMAMUX通道x标志位状态
dmamux_interrupt_enable	使能DMAMUX通道x中断
dmamux_interrupt_disable	禁能DMAMUX通道x中断
dmamux_interrupt_flag_get	获取DMAMUX通道x中断标志位状态
dmamux_interrupt_flag_clear	清除DMAMUX通道x中断标志位状态

结构体 dma_parameter_struct

表 3-227. 结构体 dma_parameter_struct

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

结构体 dmamux_sync_parameter_struct

表 3-228. 结构体 dmamux_sync_parameter_struct

成员名称	功能描述
sync_id	同步输入标识
sync_polarity	同步输入信号有效边沿
request_number	要传输的DMA请求数量

结构体 dmamux_gen_parameter_struct

表 3-229. 结构体 dmamux_gen_parameter_struct

成员名称	功能描述
trigger_id	触发输入标识
trigger_polarity	DMAMUX请求生成器触发输入信号有效边沿
request_number	要生成的DMA请求数量

枚举 dma_channel_enum

表 3-230. 枚举 dma_channel_enum

成员名称	功能描述
DMA_CH0	DMA通道0
DMA_CH1	DMA通道1
DMA_CH2	DMA通道2
DMA_CH3	DMA通道3
DMA_CH4	DMA通道4
DMA_CH5	DMA通道5
DMA_CH6	DMA通道6

枚举 dmamux_multiplexer_channel_enum

表 3-231. 枚举 dmamux_multiplexer_channel_enum

成员名称	功能描述
DMAMUX_MUXCH 0	DMAMUX请求路由通道0
DMAMUX_MUXCH 1	DMAMUX请求路由通道1
DMAMUX_MUXCH 2	DMAMUX请求路由通道2
DMAMUX_MUXCH 3	DMAMUX请求路由通道3
DMAMUX_MUXCH 4	DMAMUX请求路由通道4
DMAMUX_MUXCH 5	DMAMUX请求路由通道5
DMAMUX_MUXCH 6	DMAMUX请求路由通道6
DMAMUX_MUXCH 7	DMAMUX请求路由通道7
DMAMUX_MUXCH 8	DMAMUX请求路由通道8
DMAMUX_MUXCH 9	DMAMUX请求路由通道9
DMAMUX_MUXCH 10	DMAMUX请求路由通道10
DMAMUX_MUXCH 11	DMAMUX请求路由通道11

枚举 dmamux_generator_channel_enum

表 3-232. 枚举 dmamux_generator_channel_enum

成员名称	功能描述
DMAMUX_GENCH0	DMAMUX请求生成通道0
DMAMUX_GENCH1	DMAMUX请求生成通道1
DMAMUX_GENCH2	DMAMUX请求生成通道2
DMAMUX_GENCH3	DMAMUX请求生成通道3

枚举 dmamux_interrupt_enum

表 3-233. 枚举 dmamux_interrupt_enum

成员名称	功能描述
DMAMUX_INT_MUXCH0_SO	DMAMUX请求路由通道0同步溢出中断

DMAMUX_INT_MU XCH1_SO	DMAMUX请求路由通道1同步溢出中断
DMAMUX_INT_MU XCH2_SO	DMAMUX请求路由通道2同步溢出中断
DMAMUX_INT_MU XCH3_SO	DMAMUX请求路由通道3同步溢出中断
DMAMUX_INT_MU XCH4_SO	DMAMUX请求路由通道4同步溢出中断
DMAMUX_INT_MU XCH5_SO	DMAMUX请求路由通道5同步溢出中断
DMAMUX_INT_MU XCH6_SO	DMAMUX请求路由通道6同步溢出中断
DMAMUX_INT_MU XCH7_SO	DMAMUX请求路由通道7同步溢出中断
DMAMUX_INT_MU XCH8_SO	DMAMUX请求路由通道8同步溢出中断
DMAMUX_INT_MU XCH9_SO	DMAMUX请求路由通道9同步溢出中断
DMAMUX_INT_MU XCH10_SO	DMAMUX请求路由通道10同步溢出中断
DMAMUX_INT_MU XCH11_SO	DMAMUX请求路由通道11同步溢出中断
DMAMUX_INT_GE NCH0_TO	DMAMUX请求生成通道0触发溢出中断
DMAMUX_INT_GE NCH1_TO	DMAMUX请求生成通道1触发溢出中断
DMAMUX_INT_GE NCH2_TO	DMAMUX请求生成通道2触发溢出中断
DMAMUX_INT_GE NCH3_TO	DMAMUX请求生成通道3触发溢出中断

枚举 dmamux_flag_enum

表 3-234. 枚举 dmamux_flag_enum

成员名称	功能描述
DMAMUX_FLAG_M UXCH0_SO	DMAMUX请求路由通道0同步溢出标志
DMAMUX_FLAG_M UXCH1_SO	DMAMUX请求路由通道1同步溢出标志
DMAMUX_FLAG_M UXCH2_SO	DMAMUX请求路由通道2同步溢出标志
DMAMUX_FLAG_M UXCH3_SO	DMAMUX请求路由通道3同步溢出标志
DMAMUX_FLAG_M	DMAMUX请求路由通道4同步溢出标志

UXCH4_SO	
DMAMUX_FLAG_M UXCH5_SO	DMAMUX请求路由通道5同步溢出标志
DMAMUX_FLAG_M UXCH6_SO	DMAMUX请求路由通道6同步溢出标志
DMAMUX_FLAG_M UXCH7_SO	DMAMUX请求路由通道7同步溢出标志
DMAMUX_FLAG_M UXCH8_SO	DMAMUX请求路由通道8同步溢出标志
DMAMUX_FLAG_M UXCH9_SO	DMAMUX请求路由通道9同步溢出标志
DMAMUX_FLAG_M UXCH10_SO	DMAMUX请求路由通道10同步溢出标志
DMAMUX_FLAG_M UXCH11_SO	DMAMUX请求路由通道11同步溢出标志
DMAMUX_FLAG_G ENCH0_TO	DMAMUX请求生成通道0触发溢出标志
DMAMUX_FLAG_G ENCH1_TO	DMAMUX请求生成通道1触发溢出标志
DMAMUX_FLAG_G ENCH2_TO	DMAMUX请求生成通道2触发溢出标志
DMAMUX_FLAG_G ENCH3_TO	DMAMUX请求生成通道3触发溢出标志

枚举 dmamux_interrupt_flag_enum

表 3-235. 枚举 dmamux_interrupt_flag_enum

成员名称	功能描述
DMAMUX_INT_FLAG_MUXCH0_SO	DMAMUX请求路由通道0同步溢出中断标志
DMAMUX_INT_FLAG_MUXCH1_SO	DMAMUX请求路由通道1同步溢出中断标志
DMAMUX_INT_FLAG_MUXCH2_SO	DMAMUX请求路由通道2同步溢出中断标志
DMAMUX_INT_FLAG_MUXCH3_SO	DMAMUX请求路由通道3同步溢出中断标志
DMAMUX_INT_FLAG_MUXCH4_SO	DMAMUX请求路由通道4同步溢出中断标志
DMAMUX_INT_FLAG_MUXCH5_SO	DMAMUX请求路由通道5同步溢出中断标志
DMAMUX_INT_FLAG_MUXCH6_SO	DMAMUX请求路由通道6同步溢出中断标志
DMAMUX_INT_FLAG_MUXCH7_SO	DMAMUX请求路由通道7同步溢出中断标志

DMAMUX_INT_FLAG_MUXCH8_SO	DMAMUX请求路由通道8同步溢出中断标志
DMAMUX_INT_FLAG_MUXCH9_SO	DMAMUX请求路由通道9同步溢出中断标志
DMAMUX_INT_FLAG_MUXCH10_SO	DMAMUX请求路由通道10同步溢出中断标志
DMAMUX_INT_FLAG_MUXCH11_SO	DMAMUX请求路由通道11同步溢出中断标志
DMAMUX_INT_FLAG_GENCH0_TO	DMAMUX请求生成通道0触发溢出中断标志
DMAMUX_INT_FLAG_GENCH1_TO	DMAMUX请求生成通道1触发溢出中断标志
DMAMUX_INT_FLAG_GENCH2_TO	DMAMUX请求生成通道2触发溢出中断标志
DMAMUX_INT_FLAG_GENCH3_TO	DMAMUX请求生成通道3触发溢出中断标志

函数 dma_deinit

函数dma_deinit描述见下表：

表 3-236. 函数 dma_deinit

函数名称	dma_deinit
函数原型	void dma_deinit(uint32_t dma_periph, dma_channel_enum channelx);
功能描述	复位外设 DMAx 的通道 y 的所有寄存器
先决条件	无
被调用函数	无
输入参数{in}	
dma_periph	DMA 外设
DMAx(x=0,1)	DMA 外设选择
输入参数{in}	
channelx	DMA 通道
DMA_CHx	DMA 通道选择，参考 表 3-230 。枚举 dma_channel_enum
输出参数{out}	
-	-
返回值	
-	-

例如：

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

函数 dma_struct_para_init

函数dma_struct_para_init描述见下表：

表 3-237. 函数 dma_struct_para_init

函数名称	dma_struct_para_init
函数原型	void dma_struct_para_init(dma_parameter_struct* init_struct);
功能描述	将 DMA 结构体中所有参数初始化为默认值
先决条件	无
被调用函数	无
输入参数{in}	
*init_struct	一个已经定义的 dma_parameter_struct 结构体变量地址，参考 表 3-227. 结构体 dma_parameter_struct
输出参数{out}	
-	-
返回值	
-	-

例如：

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

函数 dma_init

函数dma_init描述见下表：

表 3-238. 函数 dma_init

函数名称	dma_init
函数原型	void dma_init(uint32_t dma_periph, dma_channel_enum channelx, dma_parameter_struct* init_struct);
功能描述	初始化外设 DMAx 的通道 y
先决条件	无
被调用函数	无
输入参数{in}	
dma_periph	DMA 外设
DMAx	DMA 外设选择
输入参数{in}	
channelx	DMA 通道
DMA_CHx	DMA 通道选择，参考 表 3-230. 枚举 dma_channel_enum
输入参数{in}	
init_struct	初始化结构体，结构体成员参考 表 3-227. 结构体 dma_parameter_struct
输出参数{out}	
-	-
返回值	
-	-

例如：

```

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

dma_init_struct.direction = DMA_PERIPHERAL_TO_MEMORY;
dma_init_struct.memory_addr = (uint32_t)g_destbuf;
dma_init_struct.memory_inc = DMA_MEMORY_INCREASE_ENABLE;
dma_init_struct.memory_width = DMA_MEMORY_WIDTH_32BIT;
dma_init_struct.number = TRANSFER_NUM/4;
dma_init_struct.periph_addr = (uint32_t) FLASH_WRITE_START_ADDR;
dma_init_struct.periph_inc = DMA_PERIPH_INCREASE_ENABLE;
dma_init_struct.periph_width = DMA_PERIPHERAL_WIDTH_32BIT;
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 3-239. 函数 dma_circulation_enable

函数名称	dma_circulation_enable
函数原型	void dma_circulation_enable(uint32_t dma_periph, dma_channel_enum channelx);
功能描述	DMA 循环模式使能
先决条件	无
被调用函数	无
输入参数{in}	
dma_periph	DMA 外设
<i>DMAx</i>	DMA 外设选择
输入参数{in}	
channelx	DMA 通道
<i>DMA_CHx</i>	DMA 通道选择，参考 表 3-230. 枚举 dma_channel_enum
输出参数{out}	
-	-
返回值	
-	-

例如：

```

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

```

函数 dma_circulation_disable

函数dma_circulation_disable描述见下表：

表 3-240. 函数 dma_circulation_disable

函数名称	dma_circulation_disable
函数原型	void dma_circulation_disable(uint32_t dma_periph, dma_channel_enum channelx);
功能描述	DMA 循环模式禁能
先决条件	无
被调用函数	无
输入参数{in}	
dma_periph	DMA 外设
DMAx	DMA 外设选择
输入参数{in}	
channelx	DMA 通道
DMA_CHx	DMA 通道选择, 参考 表 3-230. 枚举 dma_channel_enum
输出参数{out}	
-	-
返回值	
-	-

例如:

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

函数 dma_memory_to_memory_enable

函数dma_memory_to_memory_enable描述见下表:

表 3-241. 函数 dma_memory_to_memory_enable

函数名称	dma_memory_to_memory_enable
函数原型	void dma_memory_to_memory_enable(uint32_t dma_periph, dma_channel_enum channelx);
功能描述	存储器到存储器 DMA 传输使能
先决条件	无
被调用函数	无
输入参数{in}	
dma_periph	DMA 外设
DMAx	DMA 外设选择
输入参数{in}	
channelx	DMA 通道
DMA_CHx	DMA 通道选择, 参考 表 3-230. 枚举 dma_channel_enum
输出参数{out}	
-	-
返回值	
-	-

例如:

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

函数 dma_memory_to_memory_disable

函数dma_memory_to_memory_disable描述见下表:

表 3-242. 函数 dma_memory_to_memory_disable

函数名称	dma_memory_to_memory_disable
函数原形	void dma_memory_to_memory_disable(uint32_t dma_periph, dma_channel_enum channelx);
功能描述	存储器到存储器 DMA 传输禁能
先决条件	无
被调用函数	无
输入参数{in}	
dma_periph	DMA 外设
DMAx	DMA 外设选择
输入参数{in}	
channelx	DMA 通道
DMA_CHx	DMA 通道选择, 参考 表 3-230. 枚举 dma_channel_enum
输出参数{out}	
-	-
返回值	
-	-

例如:

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

函数 dma_channel_enable

函数dma_channel_enable描述见下表:

表 3-243. 函数 dma_channel_enable

函数名称	dma_channel_enable
函数原型	void dma_channel_enable(uint32_t dma_periph, dma_channel_enum channelx);
功能描述	外设 DMAx 的通道 y 传输使能
先决条件	无
被调用函数	无
输入参数{in}	
dma_periph	DMA 外设
DMAx	DMA 外设选择
输入参数{in}	
channelx	DMA 通道

<i>DMA_CHx</i>	DMA 通道选择, 参考 表 3-230. 枚举 dma_channel_enum
输出参数{out}	
-	-
返回值	
-	-

例如:

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

函数 dma_channel_disable

函数dma_channel_disable描述见下表:

表 3-244. 函数 dma_channel_disable

函数名称	dma_channel_disable
函数原型	void dma_channel_disable(uint32_t dma_periph, dma_channel_enum channelx);
功能描述	外设 DMAx 的通道 y 传输禁能
先决条件	无
被调用函数	无
输入参数{in}	
dma_periph	DMA 外设
<i>DMAx</i>	DMA 外设选择
输入参数{in}	
channelx	DMA 通道
<i>DMA_CHx</i>	DMA 通道选择, 参考 表 3-230. 枚举 dma_channel_enum
输出参数{out}	
-	-
返回值	
-	-

例如:

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

函数 dma_periph_address_config

函数dma_periph_address_config描述见下表:

表 3-245. 函数 dma_periph_address_config

函数名称	dma_periph_address_config
函数原型	void dma_periph_address_config(uint32_t dma_periph, dma_channel_enum channelx, uint32_t address);
功能描述	DMAx 通道 y 传输的外设基地址配置

先决条件	无
被调用函数	无
输入参数{in}	
dma_periph	DMA 外设
<i>DMAx</i>	DMA 外设选择
输入参数{in}	
channelx	DMA 通道
<i>DMA_CHx</i>	DMA 通道选择, 参考 表 3-230. 枚举 dma_channel_enum
输入参数{in}	
address	外设基地址
输出参数{out}	
-	-
返回值	
-	-

例如:

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

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

函数 dma_memory_address_config

函数 dma_memory_address_config 描述见下表:

表 3-246. 函数 dma_memory_address_config

函数名称	dma_memory_address_config
函数原型	void dma_memory_address_config(uint32_t dma_periph, dma_channel_enum channelx, uint32_t address);
功能描述	DMAx 通道 y 传输的存储器基地址配置
先决条件	无
被调用函数	无
输入参数{in}	
dma_periph	DMA 外设
<i>DMAx</i>	DMA 外设选择
输入参数{in}	
channelx	DMA 通道
<i>DMA_CHx</i>	DMA 通道选择, 参考 表 3-230. 枚举 dma_channel_enum
输入参数{in}	
address	存储器基地址
输出参数{out}	
-	-
返回值	
-	-

例如:

```
uint8_t g_destbuf[TRANSFER_NUM];
```

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

函数 dma_transfer_number_config

函数dma_transfer_number_config描述见下表：

表 3-247. 函数 dma_transfer_number_config

函数名称	dma_transfer_number_config
函数原型	void dma_transfer_number_config(uint32_t dma_periph, dma_channel_enum channelx, uint32_t number);
功能描述	配置 DMAx 通道 y 还有多少数据要传输
先决条件	无
被调用函数	无
输入参数{in}	
dma_periph	DMA 外设
DMAx	DMA 外设选择
输入参数{in}	
channelx	DMA 通道
DMA_CHx	DMA 通道选择，参考 表 3-230. 枚举 dma_channel_enum
输入参数{in}	
number	数据传输数量（0x00000000 – 0x0000FFFF）
输出参数{out}	
-	-
返回值	
-	-

例如：

```
#define TRANSFER_NUM                0x400
```

```
dma_transfer_number_config(DMA0, DMA_CH0, TRANSFER_NUM);
```

函数 dma_transfer_number_get

函数dma_transfer_number_get描述见下表：

表 3-248. 函数 dma_transfer_number_get

函数名称	dma_transfer_number_get
函数原型	uint32_t dma_transfer_number_get(uint32_t dma_periph, dma_channel_enum channelx);
功能描述	获取 DMAx 通道 y 还有多少数据要传输
先决条件	无
被调用函数	无
输入参数{in}	
dma_periph	DMA 外设

<i>DMAx</i>	DMA 外设选择
输入参数{in}	
channelx	DMA 通道
<i>DMA_CHx</i>	DMA 通道选择, 参考 表 3-230. 枚举 dma_channel_enum
输出参数{out}	
-	-
返回值	
uint32_t	DMA 数据传输剩余数量 (0x00000000 – 0x0000FFFF)

例如:

```
uint32_t number = 0;
```

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

函数 dma_priority_config

函数dma_priority_config描述见下表:

表 3-249. 函数 dma_priority_config

函数名称	dma_priority_config
函数原型	void dma_priority_config(uint32_t dma_periph, dma_channel_enum channelx, uint32_t priority);
功能描述	DMAx 通道 y 的传输软件优先级配置
先决条件	无
被调用函数	无
输入参数{in}	
dma_periph	DMA 外设
<i>DMAx</i>	DMA 外设选择
输入参数{in}	
channelx	DMA 通道
<i>DMA_CHx</i>	DMA 通道选择, 参考 表 3-230. 枚举 dma_channel_enum
输入参数{in}	
priority	DMA 通道软件优先级
<i>DMA_PRIORITY_LOW</i>	低优先级
<i>DMA_PRIORITY_MEDIUM</i>	中优先级
<i>DMA_PRIORITY_HIGH</i>	高优先级
<i>DMA_PRIORITY_ULTRA_HIGH</i>	极高优先级
输出参数{out}	
-	-
返回值	
-	-

例如：

```
dma_priority_config(DMA0, DMA_CH0, DMA_PRIORITY_ULTRA_HIGH);
```

函数 dma_memory_width_config

函数dma_memory_width_config描述见下表：

表 3-250. 函数 dma_memory_width_config

函数名称	dma_memory_width_config
函数原型	void dma_memory_width_config (uint32_t dma_periph, dma_channel_enum channelx, uint32_t mwidth);
功能描述	DMAx 通道 y 传输的存储器数据宽度配置
先决条件	无
被调用函数	无
输入参数{in}	
dma_periph	DMA 外设
DMAx	DMA 外设选择
输入参数{in}	
channelx	DMA 通道
DMA_CHx	DMA 通道选择，参考 表 3-230. 枚举 dma_channel_enum
输入参数{in}	
mwidth	存储器数据传输宽度
DMA_MEMORY_WIDTH_8BIT	8 位数据传输宽度
DMA_MEMORY_WIDTH_16BIT	16 位数据传输宽度
DMA_MEMORY_WIDTH_32BIT	32 位数据传输宽度
输出参数{out}	
-	-
返回值	
-	-

例如：

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

函数 dma_periph_width_config

函数dma_periph_width_config描述见下表：

表 3-251. 函数 dma_periph_width_config

函数名称	dma_periph_width_config
函数原型	void dma_periph_width_config (uint32_t dma_periph, dma_channel_enum channelx, uint32_t pwidth);
功能描述	DMAx 通道 y 传输的外设数据宽度配置

先决条件	无
被调用函数	无
输入参数{in}	
dma_periph	DMA 外设
<i>DMAx</i>	DMA 外设选择
输入参数{in}	
channelx	DMA 通道
<i>DMA_CHx</i>	DMA 通道选择, 参考 表 3-230. 枚举 dma_channel_enum
输入参数{in}	
pwidth	外设数据传输宽度
<i>DMA_PERIPHERAL_WIDTH_8BIT</i>	8 位数据传输宽度
<i>DMA_PERIPHERAL_WIDTH_16BIT</i>	16 位数据传输宽度
<i>DMA_PERIPHERAL_WIDTH_32BIT</i>	32 位数据传输宽度
输出参数{out}	
-	-
返回值	
-	-

例如:

```
dma_periph_width_config(DMA0, DMA_CH0, DMA_PERIPHERAL_WIDTH_8BIT);
```

函数 dma_memory_increase_enable

函数 dma_memory_increase_enable 描述见下表:

表 3-252. 函数 dma_memory_increase_enable

函数名称	dma_memory_increase_enable
函数原型	void dma_memory_increase_enable(uint32_t dma_periph, dma_channel_enum channelx);
功能描述	DMAx 通道 y 传输的存储器地址生成算法增量模式使能
先决条件	无
被调用函数	无
输入参数{in}	
dma_periph	DMA 外设
<i>DMAx</i>	DMA 外设选择
输入参数{in}	
channelx	DMA 通道
<i>DMA_CHx</i>	DMA 通道选择, 参考 表 3-230. 枚举 dma_channel_enum
输出参数{out}	
-	-
返回值	

-	-
---	---

例如：

```
dma_memory_increase_enable(DMA0, DMA_CH0);
```

函数 dma_memory_increase_disable

函数dma_memory_increase_disable描述见下表：

表 3-253. 函数 dma_memory_increase_disable

函数名称	dma_memory_increase_disable
函数原型	void dma_memory_increase_disable(uint32_t dma_periph, dma_channel_enum channelx);
功能描述	DMAx 通道 y 传输的存储器地址生成算法增量模式禁能
先决条件	无
被调用函数	无
输入参数{in}	
dma_periph	DMA 外设
DMAx	DMA 外设选择
输入参数{in}	
channelx	DMA 通道
DMA_CHx	DMA 通道选择，参考 表 3-230. 枚举 dma_channel_enum
输出参数{out}	
-	-
返回值	
-	-

例如：

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

函数 dma_periph_increase_enable

函数dma_periph_increase_enable描述见下表：

表 3-254. 函数 dma_periph_increase_enable

函数名称	dma_periph_increase_enable
函数原型	void dma_periph_increase_enable(uint32_t dma_periph, dma_channel_enum channelx);
功能描述	DMAx 通道 y 传输的外设地址生成算法增量模式使能
先决条件	无
被调用函数	无
输入参数{in}	
dma_periph	DMA 外设
DMAx	DMA 外设选择
输入参数{in}	

channelx	DMA 通道
<i>DMA_CHx</i>	DMA 通道选择, 参考 表 3-230. 枚举 dma_channel_enum
输出参数{out}	
-	-
返回值	
-	-

例如:

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

函数 dma_periph_increase_disable

函数dma_periph_increase_disable描述见下表:

表 3-255. 函数 dma_periph_increase_disable

函数名称	dma_periph_increase_disable
函数原型	void dma_periph_increase_disable(uint32_t dma_periph, dma_channel_enum channelx);
功能描述	DMAx 通道 y 传输的外设地址生成算法增量模式禁能
先决条件	无
被调用函数	无
输入参数{in}	
dma_periph	DMA 外设
<i>DMAx</i>	DMA 外设选择
输入参数{in}	
channelx	DMA 通道
<i>DMA_CHx</i>	DMA 通道选择, 参考 表 3-230. 枚举 dma_channel_enum
输出参数{out}	
-	-
返回值	
-	-

例如:

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

函数 dma_transfer_direction_config

函数dma_transfer_direction_config描述见下表:

表 3-256. 函数 dma_transfer_direction_config

函数名称	dma_transfer_direction_config
函数原型	void dma_transfer_direction_config(uint32_t dma_periph, dma_channel_enum channelx, uint32_t direction);
功能描述	DMAx 通道 y 的传输方向配置
先决条件	无

被调用函数	无
输入参数{in}	
dma_periph	DMA 外设
<i>DMAx</i>	DMA 外设选择
输入参数{in}	
channelx	DMA 通道
<i>DMA_CHx</i>	DMA 通道选择, 参考 表 3-230. 枚举 dma_channel_enum
输入参数{in}	
direction	数据传输方向
<i>DMA_PERIPHERAL_TO_MEMORY</i>	读取外设中数据, 写入存储器
<i>DMA_MEMORY_TO_PERIPHERAL</i>	读取存储器中数据, 写入外设
输出参数{out}	
-	-
返回值	
-	-

例如:

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

函数 dma_flag_get

函数dma_flag_get描述见下表:

表 3-257. 函数 dma_flag_get

函数名称	dma_flag_get
函数原型	FlagStatus dma_flag_get(uint32_t dma_periph, dma_channel_enum channelx, uint32_t flag);
功能描述	获取 DMAx 通道 y 标志位状态
先决条件	无
被调用函数	无
输入参数{in}	
dma_periph	DMA 外设
<i>DMAx</i>	DMA 外设选择
输入参数{in}	
channelx	DMA 通道
<i>DMA_CHx</i>	DMA 通道选择, 参考 表 3-230. 枚举 dma_channel_enum
输入参数{in}	
flag	DMA 标志
<i>DMA_FLAG_G</i>	DMA 通道全局中断标志
<i>DMA_FLAG_FTF</i>	DMA 通道传输完成标志
<i>DMA_FLAG_HTF</i>	DMA 通道半传输完成标志
<i>DMA_FLAG_ERR</i>	DMA 通道错误标志

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

例如：

```
FlagStatus flag = RESET;
```

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

函数 dma_flag_clear

函数dma_flag_clear描述见下表：

表 3-258. 函数 dma_flag_clear

函数名称	dma_flag_clear
函数原型	void dma_flag_clear(uint32_t dma_periph, dma_channel_enum channelx, uint32_t flag);
功能描述	清除 DMAx 通道 y 标志位状态
先决条件	无
被调用函数	无
输入参数{in}	
dma_periph	DMA 外设
DMAx	DMA 外设选择
输入参数{in}	
channelx	DMA 通道
DMA_CHx	DMA 通道选择，参考 表 3-230. 枚举 dma_channel_enum
输入参数{in}	
flag	DMA 标志
DMA_FLAG_G	DMA 通道全局中断标志
DMA_FLAG_FTF	DMA 通道传输完成标志
DMA_FLAG_HTF	DMA 通道半传输完成标志
DMA_FLAG_ERR	DMA 通道错误标志
输出参数{out}	
-	-
返回值	
-	-

例如：

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

函数 dma_interrupt_enable

函数dma_interrupt_enable描述见下表：

表 3-259. 函数 dma_interrupt_enable

函数名称	dma_interrupt_enable
函数原型	void dma_interrupt_enable(uint32_t dma_periph, dma_channel_enum channelx, uint32_t source);
功能描述	DMAx 通道 y 中断使能
先决条件	无
被调用函数	无
输入参数{in}	
dma_periph	DMA 外设
DMAx	DMA 外设选择
输入参数{in}	
channelx	DMA 通道
DMA_CHx	DMA 通道选择, 参考 表 3-230. 枚举 dma_channel_enum
输入参数{in}	
source	DMA 中断源
DMA_INT_FTF	DMA 通道传输完成中断
DMA_INT_HTF	DMA 通道半传输完成中断
DMA_INT_ERR	DMA 通道错误中断
输出参数{out}	
-	-
返回值	
-	-

例如:

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

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

函数 dma_interrupt_disable

函数dma_interrupt_disable描述见下表:

表 3-260. 函数 dma_interrupt_disable

函数名称	dma_interrupt_disable
函数原型	void dma_interrupt_disable(uint32_t dma_periph, dma_channel_enum channelx, uint32_t source);
功能描述	DMAx 通道 y 中断禁能
先决条件	无
被调用函数	无
输入参数{in}	
dma_periph	DMA 外设
DMAx	DMA 外设选择
输入参数{in}	
channelx	DMA 通道

<i>DMA_CHx</i>	DMA 通道选择, 参考 表 3-230. 枚举 dma_channel_enum
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_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 3-261. 函数 dma_interrupt_flag_get

函数名称	dma_interrupt_flag_get
函数原型	FlagStatus dma_interrupt_flag_get(uint32_t dma_periph, dma_channel_enum channelx, uint32_t flag);
功能描述	获取 DMAx 通道 y 中断标志位状态
先决条件	无
被调用函数	无
输入参数{in}	
dma_periph	DMA 外设
<i>DMAx</i>	DMA 外设选择
输入参数{in}	
channelx	DMA 通道
<i>DMA_CHx</i>	DMA 通道选择, 参考 表 3-230. 枚举 dma_channel_enum
输入参数{in}	
flag	DMA 标志
<i>DMA_INT_FLAG_FTF</i>	DMA 通道传输完成中断标志
<i>DMA_INT_FLAG_HTF</i>	DMA 通道半传输完成中断标志
<i>DMA_INT_FLAG_ERR</i>	DMA 通道错误中断标志
输出参数{out}	
-	-
返回值	

FlagStatus	SET 或 RESET
------------	-------------

例如:

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

函数 dma_interrupt_flag_clear

函数dma_interrupt_flag_clear描述见下表:

表 3-262. 函数 dma_interrupt_flag_clear

函数名称	dma_interrupt_flag_clear
函数原型	void dma_interrupt_flag_clear(uint32_t dma_periph, dma_channel_enum channelx, uint32_t flag);
功能描述	清除 DMAx 通道 y 中断标志位状态
先决条件	无
被调用函数	无
输入参数{in}	
dma_periph	DMA 外设
DMAx	DMA 外设选择
输入参数{in}	
channelx	DMA 通道
DMA_CHx	DMA 通道选择, 参考 表 3-230. 枚举 dma_channel_enum
输入参数{in}	
flag	DMA 标志
DMA_INT_FLAG_G	DMA 通道全局中断标志
DMA_INT_FLAG_FTF	DMA 通道传输完成中断标志
DMA_INT_FLAG_HTF	DMA 通道半传输完成中断标志
DMA_INT_FLAG_ERR	DMA 通道错误中断标志
输出参数{out}	
-	-
返回值	
-	-

例如:

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

函数 dmamux_sync_struct_para_init

函数 dmamux_sync_struct_para_init 描述见下表:

表 3-263. 函数 dmamux_sync_struct_para_init

函数名称	dmamux_sync_struct_para_init
函数原型	void dmamux_sync_struct_para_init(dmamux_sync_parameter_struct *init_struct);
功能描述	将DMAMUX同步结构体中所有参数初始化为默认值
先决条件	无
被调用函数	无
输入参数{in}	
-	-
输出参数{out}	
init_struct	一个已经定义的dmamux_sync_parameter_struct结构体变量地址，参考 表 3-228. 结构体dmamux_sync_parameter_struct
返回值	
-	-

例如：

```
/* initialize DMAMUX synchronization mode structure */
dmamux_sync_parameter_struct dmamux_sync_init_struct;
dmamux_sync_struct_para_init(&dmamux_sync_init_struct);
```

函数 dmamux_synchronization_init

函数 dmamux_synchronization_init 描述见下表：

表 3-264. 函数 dmamux_synchronization_init

函数名称	dmamux_synchronization_init
函数原型	void dmamux_synchronization_init(dmamux_multiplexer_channel_enum channelx, dmamux_sync_parameter_struct *init_struct);
功能描述	初始化DMAMUX同步结构体通道x
先决条件	无
被调用函数	无
输入参数{in}	
channelx	指定要初始化的DMAMUX请求路由通道
DMAMUX_MUXCH x	DMAMUX通道选择，参考 表3-231. 枚举 dmamux_multiplexer_channel_enum
输入参数{in}	
init_struct	一个已经定义的dmamux_sync_parameter_struct结构体变量地址，参考 表 3-228. 结构体dmamux_sync_parameter_struct
输出参数{out}	
-	-
返回值	
-	-

例如：

```

/* initialize DMAMUX synchronization mode structure */
dmamux_sync_parameter_struct dmamux_sync_init_struct;
dmamux_sync_struct_para_init(&dmamux_sync_init_struct);
/* initialize DMA request multiplexer channel 0 with synchronization mode */
dmamux_sync_init_struct.sync_id      = DMAMUX_SYNC_EXTI0;
dmamux_sync_init_struct.sync_polarity = DMAMUX_SYNC_RISING;
dmamux_sync_init_struct.request_number = 4;
dmamux_synchronization_init(DMAMUX_MUXCH0, &dmamux_sync_init_struct);

```

函数 dmamux_synchronization_enable

函数 dmamux_synchronization_enable 描述见下表:

表 3-265. 函数 dmamux_synchronization_enable

函数名称	dmamux_synchronization_enable
函数原型	void dmamux_synchronization_enable(dmamux_multiplexer_channel_enum channelx);
功能描述	使能DMAMUX同步模式
先决条件	无
被调用函数	无
输入参数{in}	
channelx	指定要初始化的DMAMUX请求路由通道
DMAMUX_MUXCHx	DMAMUX通道选择, 参考 表3-231. 枚举 dmamux_multiplexer_channel_enum
输出参数{out}	
-	-
返回值	
-	-

例如:

```

/* enable synchronization mode */
dmamux_synchronization_enable(DMAMUX_MUXCH0);

```

函数 dmamux_synchronization_disable

函数 dmamux_synchronization_disable 描述见下表:

表 3-266. 函数 dmamux_synchronization_disable

函数名称	dmamux_synchronization_disable
函数原型	void dmamux_synchronization_disable(dmamux_multiplexer_channel_enum channelx);
功能描述	禁能DMAMUX同步模式
先决条件	无
被调用函数	无
输入参数{in}	

channelx	指定要初始化的DMAMUX请求路由通道
<i>DMAMUX_MUXCH</i> <i>x</i>	DMAMUX通道选择, 参考 表3-231. 枚举 <i>dmamux_mux_channel_enum</i>
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* disable synchronization mode */
dmamux_synchronization_disable(DMAMUX_MUXCH0);
```

函数 dmamux_event_generation_enable

函数 dmamux_event_generation_enable 描述见下表:

表 3-267. 函数 dmamux_event_generation_enable

函数名称	dmamux_event_generation_enable
函数原型	void dmamux_event_generation_enable(dmamux_mux_channel_enum channelx);
功能描述	使能DMAMUX事件输出
先决条件	无
被调用函数	无
输入参数{in}	
channelx	指定要初始化的DMAMUX请求路由通道
<i>DMAMUX_MUXCH</i> <i>x</i>	DMAMUX通道选择, 参考 表3-231. 枚举 <i>dmamux_mux_channel_enum</i>
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* enable event generation */
dmamux_event_generation_enable(DMAMUX_MUXCH0);
```

函数 dmamux_event_generation_disable

函数 dmamux_event_generation_disable 描述见下表:

表 3-268. 函数 dmamux_event_generation_disable

函数名称	dmamux_event_generation_disable
函数原型	void dmamux_event_generation_disable(dmamux_mux_channel_enum channelx);
功能描述	禁能DMAMUX事件输出

先决条件	无
被调用函数	无
输入参数{in}	
channelx	指定要初始化的DMAMUX请求路由通道
DMAMUX_MUXCH x	DMAMUX通道选择, 参考 表3-231. 枚举 dmamux_mux_channel_enum
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* disable event generation */
dmamux_event_generation_disable(DMAMUX_MUXCH0);
```

函数 dmamux_gen_struct_para_init

函数 dmamux_gen_struct_para_init 描述见下表:

表 3-269. 函数 dmamux_gen_struct_para_init

函数名称	dmamux_gen_struct_para_init
函数原型	void dmamux_gen_struct_para_init(dmamux_gen_parameter_struct *init_struct);
功能描述	将DMAMUX请求生成结构体中所有参数初始化为默认值
先决条件	无
被调用函数	无
输入参数{in}	
-	-
输出参数{out}	
init_struct	一个已经定义的dmamux_gen_parameter_struct结构体变量地址, 参考 表3-229. 结构体 dmamux_gen_parameter_struct
返回值	
-	-

例如:

```
/* initialize DMA request generator structure */
dmamux_gen_parameter_struct dmamux_gen_init_struct;
dmamux_gen_struct_para_init(&dmamux_gen_init_struct);
```

函数 dmamux_request_generator_init

函数 dmamux_request_generator_init 描述见下表:

表 3-270. 函数 dmamux_request_generator_init

函数名称	dmamux_request_generator_init
------	-------------------------------

函数原型	void dmamux_request_generator_init(dmamux_generator_channel_enum channelx, dmamux_gen_parameter_struct *init_struct);
功能描述	初始化DMAMUX请求生成结构体通道x
先决条件	无
被调用函数	无
输入参数{in}	
channelx	指定要初始化的DMAMUX请求生成通道
DMAMUX_GENCHx	DMAMUX请求生成通道选择, 参考 表3-232. 枚举 dmamux_generator_channel_enum
输入参数{in}	
init_struct	一个已经定义的dmamux_gen_parameter_struct结构体变量地址, 参考 表3-229. 结构体dmamux_gen_parameter_struct
输出参数{out}	
-	-
返回值	
-	-

例如:

```

/* initialize DMA request generator channel 0 */
dmamux_gen_parameter_struct    dmamux_gen_init_struct;
dmamux_gen_struct_para_init(&dmamux_gen_init_struct);
dmamux_gen_init_struct.trigger_id      = DMAMUX_TRIGGER_EXTI13;
dmamux_gen_init_struct.trigger_polarity = DMAMUX_GEN_RISING;
dmamux_gen_init_struct.request_number = 1;
dmamux_request_generator_init(DMAMUX_GENCH0, &dmamux_gen_init_struct);

```

函数 dmamux_request_generator_channel_enable

函数 dmamux_request_generator_channel_enable 描述见下表:

表 3-271. 函数 dmamux_request_generator_channel_enable

函数名称	dmamux_request_generator_channel_enable
函数原型	void dmamux_request_generator_channel_enable(dmamux_generator_channel_enum channelx);
功能描述	使能DMAMUX请求生成通道x
先决条件	无
被调用函数	无
输入参数{in}	
channelx	指定要初始化的DMAMUX请求生成通道
DMAMUX_GENCHx	DMAMUX请求生成通道选择, 参考 表3-232. 枚举 dmamux_generator_channel_enum
输出参数{out}	
-	-

返回值	
-	-

例如：

```
/* enable DMAMUX request generator channel */
dmamux_request_generator_channel_enable(DMAMUX_GENCH0);
```

函数 dmamux_request_generator_channel_disable

函数 dmamux_request_generator_channel_disable 描述见下表：

表 3-272. 函数 dmamux_request_generator_channel_disable

函数名称	dmamux_request_generator_channel_disable
函数原型	void dmamux_request_generator_channel_disable(dmamux_generator_channel_enum channelx);
功能描述	禁能DMAMUX请求生成通道x
先决条件	无
被调用函数	无
输入参数{in}	
channelx	指定要初始化的DMAMUX请求生成通道
DMAMUX_GENCHx	DMAMUX请求生成通道选择，参考 表3-232. 枚举 dmamux_generator_channel_enum
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* disable DMAMUX request generator channel */
dmamux_request_generator_channel_disable(DMAMUX_GENCH0);
```

函数 dmamux_synchronization_polarity_config

函数 dmamux_synchronization_polarity_config 描述见下表：

表 3-273. 函数 dmamux_synchronization_polarity_config

函数名称	dmamux_synchronization_polarity_config
函数原型	void dmamux_synchronization_polarity_config(dmamux_multiplexer_channel_enum channelx, uint32_t polarity);
功能描述	配置DMAMUX同步输入的有效边沿
先决条件	无
被调用函数	无
输入参数{in}	

channelx	指定要初始化的DMAMUX请求路由通道
<i>DMAMUX_MUXCH</i> <i>x(x=0..6)</i>	DMAMUX通道选择, 参考 表3-231. 枚举 <i>dmamux_mux_channel_enum</i>
输入参数{in}	
polarity	同步输入有效边沿
<i>DMAMUX_SYNC_NO_EVENT</i>	不检测边沿
<i>DMAMUX_SYNC_RISING</i>	上升沿
<i>DMAMUX_SYNC_FALLING</i>	下降沿
<i>DMAMUX_SYNC_RISING_FALLING</i>	上升和下降沿
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* configure synchronization input polarity */
dmamux_synchronization_polarity_config(DMAMUX_MUXCH0, DMAMUX_SYNC_RISING);
```

函数 dmamux_request_forward_number_config

函数 dmamux_request_forward_number_config 描述见下表:

表 3-274. 函数 dmamux_request_forward_number_config

函数名称	dmamux_request_forward_number_config
函数原型	void dmamux_request_forward_number_config(dmamux_mux_channel_enum channelx, uint32_t number);
功能描述	配置DMAMUX通道x要传输多少个DMA请求
先决条件	无
被调用函数	无
输入参数{in}	
channelx	指定要初始化的DMAMUX请求路由通道
<i>DMAMUX_MUXCH</i> <i>x</i>	DMAMUX通道选择, 参考 表3-231. 枚举 <i>dmamux_mux_channel_enum</i>
输入参数{in}	
number	要传输的DMA请求数量 (1 - 32)
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* configure number of DMA requests to forward */
dmamux_request_forward_number_config(DMAMUX_MUXCH0, 4);
```

函数 dmamux_sync_id_config

函数 dmamux_sync_id_config 描述见下表:

表 3-275. 函数 dmamux_sync_id_config

函数名称	dmamux_sync_id_config
函数原型	void dmamux_sync_id_config(dmamux_multiplexer_channel_enum channelx, uint32_t id);
功能描述	配置DMAMUX同步输入标识
先决条件	无
被调用函数	无
输入参数{in}	
channelx	指定要初始化的DMAMUX请求路由通道
DMAMUX_MUXCHx	DMAMUX通道选择, 参考 表3-231. 枚举 dmamux_multiplexer_channel_enum
输入参数{in}	
id	同步输入标识
DMAMUX_SYNC_EXTI0	同步输入信号为EXTI0
DMAMUX_SYNC_EXTI1	同步输入信号为EXTI1
DMAMUX_SYNC_EXTI2	同步输入信号为EXTI2
DMAMUX_SYNC_EXTI3	同步输入信号为EXTI3
DMAMUX_SYNC_EXTI4	同步输入信号为EXTI4
DMAMUX_SYNC_EXTI5	同步输入信号为EXTI5
DMAMUX_SYNC_EXTI6	同步输入信号为EXTI6
DMAMUX_SYNC_EXTI7	同步输入信号为EXTI7
DMAMUX_SYNC_EXTI8	同步输入信号为EXTI8
DMAMUX_SYNC_EXTI9	同步输入信号为EXTI9
DMAMUX_SYNC_EXTI10	同步输入信号为EXTI10
DMAMUX_SYNC_EXTI11	同步输入信号为EXTI11

XTI11	
DMAMUX_SYNC_E XTI12	同步输入信号为EXTI12
DMAMUX_SYNC_E XTI13	同步输入信号为EXTI13
DMAMUX_SYNC_E XTI14	同步输入信号为EXTI14
DMAMUX_SYNC_E XTI15	同步输入信号为EXTI15
DMAMUX_SYNC_E VTx_OUT0	同步输入信号为Evt_out0
DMAMUX_SYNC_E VTx_OUT1	同步输入信号为Evt_out1
DMAMUX_SYNC_E VTx_OUT2	同步输入信号为Evt_out2
DMAMUX_SYNC_E VTx_OUT3	同步输入信号为Evt_out3
DMAMUX_SYNC_L PTIMER_OUT	同步输入信号为LPTIMER_OUT
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* configure synchronization input identification */
dmamux_sync_id_config(DMAMUX_MUXCH0, DMAMUX_SYNC_EXTI0);
```

函数 dmamux_request_id_config

函数 dmamux_request_id_config 描述见下表：

表 3-276. 函数 dmamux_request_id_config

函数名称	dmamux_request_id_config
函数原型	void dmamux_request_id_config(dmamux_multiplexer_channel_enum channelx, uint32_t id);
功能描述	配置DMAMUX请求路由通道输入标识
先决条件	无
被调用函数	无
输入参数{in}	
channelx	指定要初始化的DMAMUX请求路由通道
DMAMUX_MUXCH x	DMAMUX通道选择，参考 表3-231. 枚举 dmamux_multiplexer_channel_enum
输入参数{in}	

id	DMA请求输入标识
DMA_REQUEST_M2M	内存到内存传输
DMA_REQUEST_GENERATOR0	DMAMUX请求生成通道0请求
DMA_REQUEST_GENERATOR1	DMAMUX请求生成通道1请求
DMA_REQUEST_GENERATOR2	DMAMUX请求生成通道2请求
DMA_REQUEST_GENERATOR3	DMAMUX请求生成通道3请求
DMA_REQUEST_ADC0	请求DMAMUX ADC0
DMA_REQUEST_DAC0_CH0	请求DMAMUX DAC0 CH0
DMA_REQUEST_DAC0_CH1	请求DMAMUX DAC0 CH1
DMA_REQUEST_TIMER5_UP	请求DMAMUX TIMER5_UP
DMA_REQUEST_TIMER6_UP	请求DMAMUX TIMER6_UP
DMA_REQUEST_SPI0_RX	请求DMAMUX SPI0_RX
DMA_REQUEST_SPI0_TX	请求DMAMUX SPI0_TX
DMA_REQUEST_SPI1_RX	请求DMAMUX SPI1_RX
DMA_REQUEST_SPI1_TX	请求DMAMUX SPI1_TX
DMA_REQUEST_SPI2_RX	请求DMAMUX SPI2_RX
DMA_REQUEST_SPI2_TX	请求DMAMUX SPI2_TX
DMA_REQUEST_I2C0_RX	请求DMAMUX I2C0_RX
DMA_REQUEST_I2C0_TX	请求DMAMUX I2C0_TX
DMA_REQUEST_I2C1_RX	请求DMAMUX I2C1_RX
DMA_REQUEST_I2C1_TX	请求DMAMUX I2C1_TX
DMA_REQUEST_I2C2_RX	请求DMAMUX I2C2_RX

<i>DMA_REQUEST_I2C2_TX</i>	请求DMAMUX I2C2_TX
<i>DMA_REQUEST_I2C3_RX</i>	请求DMAMUX I2C3_RX
<i>DMA_REQUEST_I2C3_TX</i>	请求DMAMUX I2C3_TX
<i>DMA_REQUEST_USART0_RX</i>	请求DMAMUX USART0_RX
<i>DMA_REQUEST_USART0_TX</i>	请求DMAMUX USART0_TX
<i>DMA_REQUEST_USART1_RX</i>	请求DMAMUX USART1_RX
<i>DMA_REQUEST_USART1_TX</i>	请求DMAMUX USART1_TX
<i>DMA_REQUEST_USART2_RX</i>	请求DMAMUX USART2_RX
<i>DMA_REQUEST_USART2_TX</i>	请求DMAMUX USART2_TX
<i>DMA_REQUEST_UART3_RX</i>	请求DMAMUX UART3_RX
<i>DMA_REQUEST_UART3_TX</i>	请求DMAMUX UART3_TX
<i>DMA_REQUEST_UART4_RX</i>	请求DMAMUX UART4_RX
<i>DMA_REQUEST_UART4_TX</i>	请求DMAMUX UART4_TX
<i>DMA_REQUEST_ADC1</i>	请求DMAMUX ADC1
<i>DMA_REQUEST_ADC2</i>	请求DMAMUX ADC2
<i>DMA_REQUEST_ADC3</i>	请求DMAMUX ADC3
<i>DMA_REQUEST_QSPI</i>	请求DMAMUX QSPI
<i>DMA_REQUEST_DAC1_CH0</i>	请求DMAMUX DAC1_CH0
<i>DMA_REQUEST_DAC1_CH1</i>	请求DMAMUX DAC1_CH1
<i>DMA_REQUEST_TIMER0_CH0</i>	请求DMAMUX TIMER0_CH0
<i>DMA_REQUEST_TIMER0_CH1</i>	请求DMAMUX TIMER0_CH1
<i>DMA_REQUEST_TIMER0_CH2</i>	请求DMAMUX TIMER0_CH2

MER0_CH2	
DMA_REQUEST_TIMER0_CH3	请求DMAMUX TIMER0_CH3
DMA_REQUEST_TIMER0_CH0N	请求DMAMUX TIMER0_CH0N
DMA_REQUEST_TIMER0_CH1N	请求DMAMUX TIMER0_CH1N
DMA_REQUEST_TIMER0_CH2N	请求DMAMUX TIMER0_CH2N
DMA_REQUEST_TIMER0_CH3N	请求DMAMUX TIMER0_CH3N
DMA_REQUEST_TIMER0_UP	请求DMAMUX TIMER0_UP
DMA_REQUEST_TIMER0_TRIG	请求DMAMUX TIMER0_TRIG
DMA_REQUEST_TIMER0_COM	请求DMAMUX TIMER0_COM
DMA_REQUEST_TIMER7_CH0	请求DMAMUX TIMER7_CH0
DMA_REQUEST_TIMER7_CH1	请求DMAMUX TIMER7_CH1
DMA_REQUEST_TIMER7_CH2	请求DMAMUX TIMER7_CH2
DMA_REQUEST_TIMER7_CH3	请求DMAMUX TIMER7_CH3
DMA_REQUEST_TIMER7_CH0N	请求DMAMUX TIMER7_CH0N
DMA_REQUEST_TIMER7_CH1N	请求DMAMUX TIMER7_CH1N
DMA_REQUEST_TIMER7_CH2N	请求DMAMUX TIMER7_CH2N
DMA_REQUEST_TIMER7_CH3N	请求DMAMUX TIMER7_CH3N
DMA_REQUEST_TIMER7_UP	请求DMAMUX TIMER7_UP
DMA_REQUEST_TIMER7_TRIG	请求DMAMUX TIMER7_TRIG
DMA_REQUEST_TIMER7_COM	请求DMAMUX TIMER7_COM
DMA_REQUEST_TIMER1_CH0	请求DMAMUX TIMER1_CH0
DMA_REQUEST_TIMER1_CH1	请求DMAMUX TIMER1_CH1

DMA_REQUEST_TIMER1_CH2	请求DMAMUX TIMER1_CH2
DMA_REQUEST_TIMER1_CH3	请求DMAMUX TIMER1_CH3
DMA_REQUEST_TIMER1_UP	请求DMAMUX TIMER1_UP
DMA_REQUEST_TIMER1_TRIG	请求DMAMUX TIMER1_TRIG
DMA_REQUEST_TIMER2_CH0	请求DMAMUX TIMER2_CH0
DMA_REQUEST_TIMER2_CH1	请求DMAMUX TIMER2_CH1
DMA_REQUEST_TIMER2_CH2	请求DMAMUX TIMER2_CH2
DMA_REQUEST_TIMER2_CH3	请求DMAMUX TIMER2_CH3
DMA_REQUEST_TIMER2_UP	请求DMAMUX TIMER2_UP
DMA_REQUEST_TIMER2_TRIG	请求DMAMUX TIMER2_TRIG
DMA_REQUEST_TIMER3_CH0	请求DMAMUX TIMER3_CH0
DMA_REQUEST_TIMER3_CH1	请求DMAMUX TIMER3_CH1
DMA_REQUEST_TIMER3_CH2	请求DMAMUX TIMER3_CH2
DMA_REQUEST_TIMER3_CH3	请求DMAMUX TIMER3_CH3
DMA_REQUEST_TIMER3_UP	请求DMAMUX TIMER3_UP
DMA_REQUEST_TIMER3_TRIG	请求DMAMUX TIMER3_TRIG
DMA_REQUEST_TIMER4_CH0	请求DMAMUX TIMER4_CH0
DMA_REQUEST_TIMER4_CH1	请求DMAMUX TIMER4_CH1
DMA_REQUEST_TIMER4_CH2	请求DMAMUX TIMER4_CH2
DMA_REQUEST_TIMER4_CH3	请求DMAMUX TIMER4_CH3
DMA_REQUEST_TIMER4_UP	请求DMAMUX TIMER4_UP
DMA_REQUEST_TIMER4_TRIG	请求DMAMUX TIMER4_TRIG

MER4_TRIG	
DMA_REQUEST_TIMER14_CH0	请求DMAMUX TIMER14_CH0
DMA_REQUEST_TIMER14_CH1	请求DMAMUX DMAMUXTIMER14_CH1
DMA_REQUEST_TIMER14_CH0N	请求DMAMUX TIMER14_CH0N
DMA_REQUEST_TIMER14_UP	请求DMAMUX TIMER14_UP
DMA_REQUEST_TIMER14_TRIG	请求DMAMUX TIMER14_TRIG
DMA_REQUEST_TIMER14_COM	请求DMAMUX TIMER14_COM
DMA_REQUEST_TIMER15_CH0	请求DMAMUX TIMER15_CH0
DMA_REQUEST_TIMER15_CH0N	请求DMAMUX TIMER15_CH0N
DMA_REQUEST_TIMER15_UP	请求DMAMUX TIMER15_UP
DMA_REQUEST_TIMER16_CH0	请求DMAMUX TIMER16_CH0
DMA_REQUEST_TIMER16_CH0N	请求DMAMUX TIMER16_CH0N
DMA_REQUEST_TIMER16_UP	请求DMAMUX TIMER16_UP
DMA_REQUEST_TIMER19_CH0	请求DMAMUX TIMER19_CH0
DMA_REQUEST_TIMER19_CH1	请求DMAMUX TIMER19_CH1
DMA_REQUEST_TIMER19_CH2	请求DMAMUX TIMER19_CH2
DMA_REQUEST_TIMER19_CH3	请求DMAMUX TIMER19_CH3
DMA_REQUEST_TIMER19_CH0N	请求DMAMUX TIMER19_CH0N
DMA_REQUEST_TIMER19_CH1N	请求DMAMUX TIMER19_CH1N
DMA_REQUEST_TIMER19_CH2N	请求DMAMUX TIMER19_CH2N
DMA_REQUEST_TIMER19_CH3N	请求DMAMUX TIMER19_CH3N
DMA_REQUEST_TIMER19_UP	请求DMAMUX TIMER19_UP

DMA_REQUEST_TIMER19_TRIG	请求DMAMUX TIMER19_TRIG
DMA_REQUEST_TIMER19_COM	请求DMAMUX TIMER19_COM
DMA_REQUEST_CAU_IN	请求DMAMUX CAU_IN
DMA_REQUEST_CAU_OUT	请求DMAMUX CAU_OUT
DMA_REQUEST_HRTIMER_MASTER	请求DMAMUX HRTIMER_MASTER
DMA_REQUEST_HRTIMER_TIMER0	请求DMAMUX HRTIMER_TIMER0
DMA_REQUEST_HRTIMER_TIMER1	请求DMAMUX HRTIMER_TIMER1
DMA_REQUEST_HRTIMER_TIMER2	请求DMAMUX HRTIMER_TIMER2
DMA_REQUEST_HRTIMER_TIMER3	请求DMAMUX HRTIMER_TIMER3
DMA_REQUEST_HRTIMER_TIMER4	请求DMAMUX HRTIMER_TIMER4
DMA_REQUEST_HRTIMER_TIMER5	请求DMAMUX HRTIMER_TIMER5
DMA_REQUEST_HRTIMER_TIMER6	请求DMAMUX HRTIMER_TIMER6
DMA_REQUEST_HRTIMER_TIMER7	请求DMAMUX HRTIMER_TIMER7
DMA_REQUEST_DAC2_CH0	请求DMAMUX DAC2_CH0
DMA_REQUEST_DAC2_CH1	请求DMAMUX DAC2_CH1
DMA_REQUEST_DAC3_CH0	请求DMAMUX DAC3_CH0
DMA_REQUEST_DAC3_CH1	请求DMAMUX DAC3_CH1
DMA_REQUEST_HPDI_FLT0	请求DMAMUX HPDI_FLT0
DMA_REQUEST_HPDI_FLT1	请求DMAMUX HPDI_FLT1
DMA_REQUEST_HPDI_FLT2	请求DMAMUX HPDI_FLT2
DMA_REQUEST_HPDI_FLT3	请求DMAMUX HPDI_FLT3
DMA_REQUEST_FAC_READ	请求DMAMUX FAC_READ

AC_READ	
DMA_REQUEST_F AC_WRITE	请求DMAMUX FAC_WRITE
DMA_REQUEST_T MU_INPUT	请求DMAMUX TMU_INPUT
DMA_REQUEST_T MU_OUTPUT	请求DMAMUX TMU_OUTPUT
DMA_REQUEST_C AN0	请求DMAMUX CAN0
DMA_REQUEST_C AN1	请求DMAMUX CAN1
DMA_REQUEST_C AN2	请求DMAMUX CAN2
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* configure multiplexer input identification */
dmamux_request_id_config(DMAMUX_MUXCH0, DMA_REQUEST_GENERATOR0);
```

函数 dmamux_trigger_polarity_config

函数 dmamux_trigger_polarity_config 描述见下表：

表 3-277. 函数 dma_interrupt_disable

函数名称	dmamux_trigger_polarity_config
函数原型	void dmamux_trigger_polarity_config(dmamux_generator_channel_enum channelx, uint32_t polarity);
功能描述	配置DMAMUX触发输入的有效边沿
先决条件	无
被调用函数	无
输入参数{in}	
channelx	指定要初始化的DMAMUX请求生成通道
DMAMUX_GENCHx	DMAMUX请求生成通道选择，参考 表3-232. 枚举 dmamux_generator_channel_enum
输入参数{in}	
polarity	触发输入信号有效边沿
DMAMUX_GEN_NO_EVENT	不检测边沿
DMAMUX_GEN_RISING	上升沿
DMAMUX_GEN_FALLING	下降沿

LLING	
DMAMUX_GEN_RISING_FALLING	上升和下降沿
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* configure trigger input polarity */
dmamux_trigger_polarity_config(DMAMUX_GENCH0, DMAMUX_GEN_RISING);
```

函数 dmamux_request_generate_number_config

函数 dmamux_request_generate_number_config 描述见下表:

表 3-278. 函数 dmamux_request_generate_number_config

函数名称	dmamux_request_generate_number_config
函数原型	void dmamux_request_generate_number_config(dmamux_generator_channel_enum channelx, uint32_t number);
功能描述	配置DMAMUX请求生成器生成请求的数量
先决条件	无
被调用函数	无
输入参数{in}	
channelx	指定要初始化的DMAMUX请求生成通道
DMAMUX_GENCHx	DMAMUX请求生成通道选择, 参考 表3-232. 枚举 dmamux_generator_channel_enum
输入参数{in}	
number	要生成的DMA请求数量 (1 - 32)
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* configure number of DMA requests to be generated */
dmamux_request_generate_number_config(DMAMUX_GENCH0, 1);
```

函数 dmamux_trigger_id_config

函数 dmamux_trigger_id_config 描述见下表:

表 3-279. 函数 dmamux_trigger_id_config

函数名称	dmamux_trigger_id_config
------	--------------------------

函数原型	void dmamux_trigger_id_config(dmamux_generator_channel_enum channelx, uint32_t id);
功能描述	配置DMAMUX触发输入标识
先决条件	无
被调用函数	无
输入参数{in}	
channelx	指定要初始化的DMAMUX请求生成通道
DMAMUX_GENCHx	DMAMUX请求生成通道选择, 参考 表3-232. 枚举 dmamux_generator_channel_enum
输入参数{in}	
id	触发输入标识
DMAMUX_TRIGGER_EXTI0	触发输入为EXTI0
DMAMUX_TRIGGER_EXTI1	触发输入为EXTI1
DMAMUX_TRIGGER_EXTI2	触发输入为EXTI2
DMAMUX_TRIGGER_EXTI3	触发输入为EXTI3
DMAMUX_TRIGGER_EXTI4	触发输入为EXTI4
DMAMUX_TRIGGER_EXTI5	触发输入为EXTI5
DMAMUX_TRIGGER_EXTI6	触发输入为EXTI6
DMAMUX_TRIGGER_EXTI7	触发输入为EXTI7
DMAMUX_TRIGGER_EXTI8	触发输入为EXTI8
DMAMUX_TRIGGER_EXTI9	触发输入为EXTI9
DMAMUX_TRIGGER_EXTI10	触发输入为EXTI10
DMAMUX_TRIGGER_EXTI11	触发输入为EXTI11
DMAMUX_TRIGGER_EXTI12	触发输入为EXTI12
DMAMUX_TRIGGER_EXTI13	触发输入为EXTI13
DMAMUX_TRIGGER_EXTI14	触发输入为EXTI14
DMAMUX_TRIGGER_EXTI15	触发输入为EXTI15

DMAMUX_TRIGGE R_EVTX_OUT0	触发输入为Evt_out0
DMAMUX_TRIGGE R_EVTX_OUT1	触发输入为Evt_out1
DMAMUX_TRIGGE R_EVTX_OUT2	触发输入为Evt_out2
DMAMUX_TRIGGE R_EVTX_OUT3	触发输入为Evt_out3
DMAMUX_TRIGGE R_LPTIMER_OUT	触发输入为LPTIMER_OUT
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* configure trigger input identification */
dmamux_trigger_id_config(DMAMUX_GENCH0, DMAMUX_TRIGGER_EXTI13);
```

函数 dmamux_flag_get

函数 dmamux_flag_get 描述见下表：

表 3-280. 函数 dmamux_flag_get

函数名称	dmamux_flag_get
函数原型	FlagStatus dmamux_flag_get(dmamux_flag_enum flag);
功能描述	获取DMAMUXA通道x标志位状态
先决条件	无
被调用函数	无
输入参数{in}	
flag	标志类型，参考 表3-234. 枚举dmamux_flag_enum
输出参数{out}	
-	-
返回值	
FlagStatus	SET或RESET

例如：

```
FlagStatus flag = RESET;
/* get DMAMUX flag */
flag = dmamux_flag_get(DMAMUX_FLAG_GENCH0_TO);
```

函数 dmamux_flag_clear

函数 dmamux_flag_clear 描述见下表：

表 3-281. 函数 dmamux_flag_clear

函数名称	dmamux_flag_clear
函数原型	void dmamux_flag_clear(dmamux_flag_enum flag);
功能描述	清除DMAMUX通道x标志位状态
先决条件	无
被调用函数	无
输入参数{in}	
flag	标志类型，参考 表3-234. 枚举dmamux_flag_enum
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* clear DMAMUX flag */
dmamux_flag_clear(DMAMUX_FLAG_GENCH0_TO);
```

函数 dmamux_interrupt_enable

函数 dmamux_interrupt_enable 描述见下表：

表 3-282. 函数 dmamux_interrupt_enable

函数名称	dmamux_interrupt_enable
函数原型	void dmamux_interrupt_enable(dmamux_interrupt_enum interrupt);
功能描述	使能DMAMUX通道x中断
先决条件	无
被调用函数	无
输入参数{in}	
interrupt	中断类型，参考 表3-233. 枚举dmamux_interrupt_enum
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* enable DMAMUX interrupt */
dmamux_interrupt_enable(DMAMUX_INT_MUXCH0_SO);
```

函数 dmamux_interrupt_disable

函数 dmamux_interrupt_disable 描述见下表：

表 3-283. 函数 dmamux_interrupt_disable

函数名称	dmamux_interrupt_disable
函数原型	void dmamux_interrupt_disable(dmamux_interrupt_enum interrupt);

功能描述	禁能DMAMUX通道x中断
先决条件	无
被调用函数	无
输入参数{in}	
interrupt	中断类型，参考 表3-233. 枚举dmamux_interrupt_enum
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* disable DMAMUX interrupt */
dmamux_interrupt_disable(DMAMUX_INT_MUXCH0_SO);
```

函数 dmamux_interrupt_flag_get

函数 dmamux_interrupt_flag_get 描述见下表：

表 3-284. 函数 dmamux_interrupt_flag_get

函数名称	dmamux_interrupt_flag_get
函数原型	FlagStatus dmamux_interrupt_flag_get(dmamux_interrupt_flag_enum int_flag);
功能描述	获取DMAMUX通道x中断标志位状态
先决条件	无
被调用函数	无
输入参数{in}	
int_flag	标志类型，参考 表3-235. 枚举dmamux_interrupt_flag_enum
输出参数{out}	
-	-
返回值	
FlagStatus	SET或RESET

例如：

```
/* check DMAMUX interrupt flag */
if(dmamux_interrupt_flag_get(DMAMUX_INT_FLAG_GENCH0_TO)) {
    dmamux_interrupt_flag_clear(DMAMUX_INT_FLAG_GENCH0_TO);
}
```

函数 dmamux_interrupt_flag_clear

函数 dmamux_interrupt_flag_clear 描述见下表：

表 3-285. 函数 dmamux_interrupt_flag_clear

函数名称	dmamux_interrupt_flag_clear
函数原型	FlagStatus dmamux_interrupt_flag_get(dmamux_interrupt_flag_enum int_flag);
功能描述	清除DMAMUX通道x中断标志位状态

先决条件	无
被调用函数	无
输入参数{in}	
int_flag	标志类型，参考 表3-235. 枚举dmamux_interrupt_flag_enum
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* check DMAMUX interrupt flag */
if(dmamux_interrupt_flag_get(DMAMUX_INT_FLAG_GENCH0_TO)) {
    dmamux_interrupt_flag_clear(DMAMUX_INT_FLAG_GENCH0_TO);
}
```

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 3-286. EXMC 寄存器

寄存器名称	寄存器描述
EXMC_PNCTL	PSRAM/NOR Flash控制寄存器
EXMC_PNTCFG	PSRAM/NOR Flash时序寄存器
EXMC_NCTL	NAND flash控制寄存器
EXMC_NSTAT	NAND flash状态寄存器
EXMC_NCTCFG	NAND flash通用空间时序寄存器
EXMC_NATCFG	NAND flash属性空间时序寄存器
EXMC_NECC	NAND flash ECC结果寄存器

3.12.2. 外设库函数说明

EXMC库函数列表如下表所示：

表 3-287. EXMC 库函数

库函数名称	库函数描述
exmc_norsram_deinit	复位EXMC NOR/PSRAM regionx
exmc_norsram_struct_para_init	初始化结构体exmc_norsram_parameter_struct
exmc_norsram_init	初始化EXMC NOR/PSRAM regionx
exmc_norsram_enable	使能EXMC NOR/PSRAM regionx
exmc_norsram_disable	禁用EXMC NOR/PSRAM regionx
exmc_nand_deinit	复位EXMC NAND bankx
exmc_nand_struct_para_init	初始化结构体exmc_nand_struct_para_init
exmc_nand_init	初始化EXMC NAND bankx
exmc_nand_enable	使能EXMC NAND bankx
exmc_nand_disable	禁用EXMC NAND bankx
exmc_norsram_page_size_config	配置CRAM页大小
exmc_nand_ecc_enable	使能EXMC NAND ECC功能
exmc_nand_ecc_disable	禁用EXMC NAND ECC功能
exmc_ecc_get	获取EXMC ECC值
exmc_flag_get	获取EXMC状态

结构体 exmc_norsram_timing_parameter_struct

表 3-288. 结构体 exmc_norsram_timing_parameter_struct

成员名称	功能描述
syn_data_latency	数据延迟
syn_clk_division	同步时钟分频比
bus_latency	总线延迟
asyn_data_setup_time	数据建立时间
asyn_address_hold_time	地址保持时间
asyn_address_setup_time	地址建立时间

结构体 `exmc_norsram_parameter_struct`

表 3-289. 结构体 `exmc_norsram_parameter_struct`

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

结构体 `exmc_nand_timing_parameter_struct`

表 3-290. 结构体 `exmc_nand_timing_parameter_struct`

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

结构体 `exmc_nand_parameter_struct`

表 3-291. 结构体 `exmc_nand_parameter_struct`

成员名称	功能描述
<code>nand_bank</code>	选择EXMC NAND Bank
<code>ecc_size</code>	ECC块大小
<code>atr_latency</code>	ALE至RE的延迟

成员名称	功能描述
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
attribute_space_timing	NAND flash属性空间时序配置

函数 exmc_norsram_deinit

函数exmc_norsram_deinit描述见下表：

表 3-292. 函数 exmc_norsram_deinit

函数名称	exmc_norsram_deinit
函数原型	void exmc_norsram_deinit(void);
功能描述	复位NOR/PSRAM region
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 EXMC NOR/PSRAM region0 of bank0 */
exmc_norsram_deinit();
```

函数 exmc_norsram_struct_para_init

函数exmc_norsram_struct_para_init描述见下表：

表 3-293. 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}	
<code>exmc_norsram_init_struct</code>	初始化结构体，结构体成员参考 表 3-289. 结构体 <code>exmc_norsram_parameter_struct</code>
输出参数{out}	
-	-
返回值	
-	-

例如：

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

函数 `exmc_norsram_init`

函数`exmc_norsram_init`描述见下表：

表 3-294. 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NOR/PSRAM 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 3-289. 结构体 <code>exmc_norsram_parameter_struct</code>

输出参数{out}	
-	-
返回值	
-	-

例如:

```

/* initialize EXMC NOR/PSRAM bank */

exmc_norsram_parameter_struct lcd_init_struct;

exmc_norsram_timing_parameter_struct lcd_timing_init_struct;

/* configure timing parameter */

lcd_timing_init_struct.syn_data_latency = EXMC_DATA_LAT_2_CLK;

lcd_timing_init_struct.syn_clk_division = EXMC_SYN_CLOCK_RATIO_DISABLE;

lcd_timing_init_struct.bus_latency = 1;

lcd_timing_init_struct.asyn_data_setup_time = 5;

lcd_timing_init_struct.asyn_address_hold_time = 2;

lcd_timing_init_struct.asyn_address_setup_time = 2;

/* configure EXMC bus parameters */

lcd_init_struct.write_mode = EXMC_ASYNC_WRITE;

lcd_init_struct.asyn_wait = DISABLE;

lcd_init_struct.nwait_signal = DISABLE;

lcd_init_struct.memory_write = ENABLE;

lcd_init_struct.nwait_config = EXMC_NWAIT_CONFIG_BEFORE;

lcd_init_struct.wrap_burst_mode = DISABLE;

lcd_init_struct.nwait_polarity = EXMC_NWAIT_POLARITY_LOW;

lcd_init_struct.burst_mode = DISABLE;

lcd_init_struct.databus_width = EXMC_NOR_DATABUS_WIDTH_16B;

lcd_init_struct.memory_type = EXMC_MEMORY_TYPE_NOR;

lcd_init_struct.address_data_mux = DISABLE;

lcd_init_struct.read_write_timing = &lcd_timing_init_struct;

exmc_norsram_init(&lcd_init_struct);

```


函数 `exmc_norsram_enable`

函数 `exmc_norsram_enable` 描述见下表：

表 3-295. 函数 `exmc_norsram_enable`

函数名称	<code>exmc_norsram_enable</code>
函数原型	<code>void exmc_norsram_enable(void);</code>
功能描述	使能EXMC NOR/PSRAM region
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 EXMC NOR/PSRAM region0 of bank0 */
```

```
exmc_norsram_enable();
```

函数 `exmc_norsram_disable`

函数 `exmc_norsram_disable` 描述见下表：

表 3-296. 函数 `exmc_norsram_disable`

函数名称	<code>exmc_norsram_disable</code>
函数原型	<code>void exmc_norsram_disable(void);</code>
功能描述	禁用EXMC NOR/PSRAM region
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 EXMC NOR/PSRAM region0 of bank0 */
```

```
exmc_norsram_disable();
```

函数 exmc_nand_deinit

函数exmc_nand_deinit描述见下表：

表 3-297. 函数 exmc_nand_deinit

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

例如：

```
/* deinitialize EXMC NOR/PSRAM bank1 */
```

```
exmc_norsram_deinit();
```

函数 exmc_nand_struct_para_init

函数exmc_nand_struct_para_init描述见下表：

表 3-298. 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}	
exmc_nand_init_struct	初始化结构体，结构体成员参考 表 3-291. 结构体 exmc_nand_parameter_struct
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* initialize the struct 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 3-299. 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 3-291. 结构体 exmc_nand_parameter_struct
输出参数{out}	
-	-
返回值	

-	-
---	---

例如:

```

exmc_nand_parameter_struct nand_init_struct;

exmc_nand_pccard_timing_parameter_struct nand_timing_init_struct;

/* EXMC configuration */

nand_timing_init_struct.setup_time = 5;

nand_timing_init_struct.wait_time = 4;

nand_timing_init_struct.hold_time = 2;

nand_timing_init_struct.databus_hiz_time = 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 3-300. 函数 exmc_nand_enable

函数名称	exmc_nand_enable
函数原型	void exmc_nand_enable(void);
功能描述	使能 EXMC NAND bank
先决条件	-
被调用函数	-
输入参数 {in}	
-	-

输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* enable EXMC NAND bank1 */
```

```
exmc_nand_enable();
```

函数 exmc_nand_disable

函数exmc_nand_disable描述见下表:

表 3-301. 函数 exmc_nand_disable

函数名称	exmc_nand_disable
函数原型	exmc_nand_disable(void);
功能描述	禁用EXMC NAND bank
先决条件	-
被调用函数	-
输入参数{in}	
-	-
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* disable EXMC NAND bank1 */
```

```
exmc_nand_disable();
```

函数 exmc_norsram_page_size_config

函数exmc_norsram_page_size_config描述见下表:

表 3-302. 函数 exmc_norsram_page_size_config

函数名称	exmc_norsram_page_size_config
------	-------------------------------

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

例如:

```
/* configure CRAM page size */
```

```
exmc_norsram_page_size_config (EXMC_CRAM_PAGE_SIZE_128_BYTES);
```

函数 exmc_nand_ecc_enable

函数exmc_nand_ecc_enable描述见下表:

表 3-303. 函数 exmc_nand_ecc_enable

函数名称	exmc_nand_ecc_enable
------	----------------------

函数原型	void exmc_nand_ecc_enable(void);
功能描述	使能EXMC NAND ECC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 the EXMC NAND ECC function */
```

```
exmc_nand_ecc_enable();
```

函数 exmc_nand_ecc_disable

函数exmc_nand_ecc_disable描述见下表：

表 3-304. 函数 exmc_nand_ecc_enable

函数名称	exmc_nand_ecc_disable
函数原型	void exmc_nand_ecc_disable(void);
功能描述	禁用EXMC NAND ECC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 the EXMC NAND ECC function */
```

```
exmc_nand_ecc_disable();
```

函数 exmc_ecc_get

函数exmc_ecc_get描述见下表:

表 3-305. 函数 exmc_ecc_get

函数名称	exmc_ecc_get
函数原型	uint32_t exmc_ecc_get(void);
功能描述	获取EXMC NAND ECC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	ECC计算的值

例如:

```
/* get the EXMC ECC value */
```

```
uint32_t ecc_value;
```

```
ecc_value = exmc_ecc_get(void);
```

函数 exmc_flag_get

函数exmc_flag_get描述见下表:

表 3-306. 函数 exmc_flag_get

函数名称	exmc_flag_get
函数原型	FlagStatus exmc_flag_get(uint32_t flag);
功能描述	获取EXMC状态
先决条件	-
被调用函数	-

输入参数{in}	
flag	EXMC标志状态
EXMC_NAND_FLAG_FIFOE	FIFO空状态
输出参数{out}	
-	-
返回值	
FlagStatus	SET或RESET

例如:

```
/* check FIFO status is set or not*/
if(RESET != exmc_flag_get (EXMC_NAND_FLAG_FIFOE));
```

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 3-307. 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 3-308. 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

库函数名称	库函数描述
exti_interrupt_enable	EXTI线x中断使能
exti_interrupt_disable	EXTI线x中断禁能
exti_event_enable	EXTI线x事件使能
exti_event_disable	EXTI线x事件禁能
exti_software_interrupt_enable	EXTI线x软件中断事件使能
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 3-309. 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 3-310. 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 3-311. 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 3-312. 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 3-313. 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, 参考 表3-309. 枚举类型<code>exti_line_enum</code>
输入参数{in}	
mode	EXTI模式, 参考 表3-310. 枚举类型<code>exti_mode_enum</code>

输入参数{in}	
trig_type	触发类型, 参考 表3-311. 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 3-314. 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3-309. 枚举类型exti_line_enum
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* enable the interrupts from EXTI line 0 */
```

```
exti_interrupt_enable(EXTI_0);
```

函数 exti_interrupt_disable

函数exti_interrupt_disable描述见下表:

表 3-315. 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3-309. 枚举类型exti_line_enum

输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* disable the interrupts from EXTI line 0 */
```

```
exti_interrupt_disable(EXTI_0);
```

函数 exti_event_enable

函数exti_event_enable描述见下表：

表 3-316. 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3-309. 枚举类型exti_line_enum
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* enable the events from EXTI line 0 */
```

```
exti_event_enable(EXTI_0);
```

函数 exti_event_disable

函数exti_event_disable描述见下表：

表 3-317. 函数 exti_event_disable

函数名称	exti_event_disable
函数原形	void exti_event_disable(exti_line_enum linex);
功能描述	EXTI线x事件禁能
先决条件	-
被调用函数	-
输入参数{in}	
linex	EXTI线x，参考 表3-309. 枚举类型exti_line_enum
输出参数{out}	
-	-

返回值	
-	-

例如：

```
/* disable the events from EXTI line 0 */
```

```
exti_event_disable(EXTI_0);
```

函数 exti_software_interrupt_enable

函数exti_software_interrupt_enable描述见下表：

表 3-318. 函数 exti_software_interrupt_enable

函数名称	exti_software_interrupt_enable
函数原形	void exti_software_interrupt_enable(exti_line_enum linex);
功能描述	EXTI线x软件中断事件使能
先决条件	-
被调用函数	-
输入参数{in}	
linex	EXTI线x，参考 表3-309. 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 3-319. 函数 exti_software_interrupt_disable

函数名称	exti_software_interrupt_disable
函数原形	void exti_software_interrupt_disable(exti_line_enum linex);
功能描述	EXTI线x软件中断事件禁能
先决条件	-
被调用函数	-
输入参数{in}	
linex	EXTI线x，参考 表3-309. 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 3-320. 函数 exti_flag_get

函数名称	exti_flag_get
函数原形	FlagStatus exti_flag_get(exti_line_enum linex);
功能描述	获取EXTI线x中断标志位
先决条件	-
被调用函数	-
输入参数{in}	
linex	EXTI线x, 参考 表3-309. 枚举类型exti_line_enum
输出参数{out}	
-	-
返回值	
FlagStatus	SET或RESET

例如：

```
/* get EXTI line 0 flag status */
FlagStatus state = exti_flag_get(EXTI_0);
```

函数 exti_flag_clear

函数exti_flag_clear描述见下表：

表 3-321. 函数 exti_flag_clear

函数名称	exti_flag_clear
函数原形	void exti_flag_clear(exti_line_enum linex);
功能描述	清除EXTI线x中断标志位
先决条件	-
被调用函数	-
输入参数{in}	
linex	EXTI线x, 参考 表3-309. 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 3-322. 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3-309. 枚举类型exti_line_enum
输出参数{out}	
-	-
返回值	
FlagStatus	SET或RESET

例如：

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

```
FlagStatus state = exti_interrupt_flag_get(EXTI_0);
```

函数 exti_interrupt_flag_clear

函数exti_interrupt_flag_clear描述见下表：

表 3-323. 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3-309. 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. 外设寄存器描述

FMC寄存器列表如下表所示：

表 3-324. FMC 寄存器

寄存器名称	寄存器描述
FMC_KEY0	FMC 解锁密钥寄存器 0
FMC_OBKEY	FMC 选项字节解锁密钥寄存器
FMC_STAT0	FMC 状态寄存器 0
FMC_CTL0	FMC 控制寄存器 0
FMC_ADDR0	FMC 地址寄存器 0
FMC_OBCTL0	FMC 选项字节控制寄存器 0
FMC_OBCTL1	FMC 选项字节控制寄存器 1
FMC_OBCTL2	FMC 选项字节控制寄存器 2
FMC_OTP1CFG	FMCOTP1 配置寄存器
FMC_OBSTAT	FMC 选项字节状态寄存器
FMC_KEY1	FMC 解锁密钥寄存器 1
FMC_STAT1	FMC 状态寄存器 1
FMC_CTL1	FMC 控制寄存器 1
FMC_ADDR1	FMC 地址寄存器 1
FMC_OTP3_STAT	FMCOTP3 状态寄存器
FMC_PID	FMC 产品 ID 寄存器
OB_SPC	选项字节安全保护值
OB_USER	选项字节用户值
OB_DATA1	选项字节数据 1 值
OB_DATA2	选项字节数据 2 值
OB_WP0	选项字节写保护 0
OB_WP1	选项字节写保护 1
OB_WP2	选项字节写保护 2
OB_WP3	选项字节写保护 3

3.14.2. 外设库函数说明

FMC库函数列表如下表所示：

表 3-325. FMC 库函数

库函数名称	库函数描述
fmc_unlock	解锁主 FMC 操作

库函数名称	库函数描述
fmc_bank0_unlock	解锁 FMC bank0 操作
fmc_bank1_unlock	解锁 FMC bank1 操作
fmc_lock	锁定主 FMC 操作
fmc_bank0_lock	锁定 FMC bank0 操作
fmc_bank1_lock	锁定 FMC Bank1 操作
fmc_page_erase	页擦除
fmc_mass_erase	擦除整个芯片
fmc_bank0_erase	擦除 bank0
fmc_bank1_erase	擦除 bank1
fmc_word_program	在对应地址编程一个字。
fmc_halfword_program	在对应地址编程半字
fmc_otp_word_program	在对应地址编程一个字。
fmc_otp_half_word_program	在对应地址编程半字
fmc_otp_byte_program	在对应地址编程一个字节
fmc_nwa_enable	使能系统复位后无等待时间区载入
fmc_nwa_disable	系统复位后失能无等待时间区域加载
otp1_read_disable	设置 OTP1 数据块不可读取
otp2_rlock_enable	使能 OTP2 的读锁块保护
fmc_deep_power_down_enable	当无操作时使能深度掉电模式
fmc_deep_power_down_disable	当无操作时失能深度掉电模式
ob_unlock	解锁 Option Byte 操作
ob_lock	锁定选项字节操作
ob_start	发送选项字节更改命令
ob_erase	擦除选项字节
ob_write_protection_enable	使能写保护
ob_write_protection_disable	失能写保护
ob_security_protection_config	配置安全保护等级
ob_user_write	编程 FMC 用户选项字节
ob_sram_init_config	配置 Option Byte SRAMC 初始化值
ob_sram_ecc_config	配置选项字节 SRAM ECC 功能值
ob_nwa_clock_config	配置 Option Byte 时钟以加载无等待时间区域
ob_data_write	编程选项字节数据
ob_sram_init_get	获取选项字节 SRAM 初始化值
ob_sram_ecc_get	获取选项字节 SRAMC ECC 使能值
ob_nwa_clock_get	获取选项字节 加载时钟 无等待时间区域
ob_data_get	获取 FMC 数据选项字节
ob_write_protection_get	获取 FMC 选项字节写保护
ob_spc_get	获取选项字节安全保护码值
fmc_interrupt_enable	使能 FMC 中断
fmc_interrupt_disable	失能 FMC 中断
fmc_flag_get	检查标志是否置位
fmc_flag_clear	清除 FMC 标志

库函数名称	库函数描述
fmc_interrupt_flag_get	获取 FMC 中断标志状态
fmc_interrupt_flag_clear	清除 FMC 中断标志状态
fmc_bank0_state_get	获取 FMC bank0 状态
fmc_bank1_state_get	获取 FMC bank1 状态
fmc_bank0_ready_wait	检查 FMC bank0 是否就绪
fmc_bank1_ready_wait	检查 FMC bank1 是否就绪

枚举类型 fmc_state_enum

表 3-326.枚举类型 fmc_state_enum

成员名称	功能描述
FMC_READY	操作已完成
FMC_BUSY	操作正在进行中
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 3-327.枚举类型 fmc_interrupt_enum

成员名称	功能描述
FMC_INT_BANK0_END	FMC Bank0 编程结束中断
FMC_INT_BANK0_ERR	FMC bank0 错误中断
FMC_INT_BANK1_END	FMC bank1 编程结束中断
FMC_INT_BANK1_ERR	FMC bank1 错误中断

枚举类型 fmc_flag_enum

表 3-328.枚举类型 fmc_flag_enum

成员名称	功能描述
FMC_FLAG_BANK0_BUSY	FMC bank0 忙标志
FMC_FLAG_BANK0_PGERR	FMC bank0 操作错误标志位
FMC_FLAG_BANK0_WPERR	FMC bank0 擦除/编程保护错误标志位
FMC_FLAG_BANK0_END	FMC bank0 操作结束标志位
FMC_FLAG_OBERR	FMC 选项字节读取错误标志
FMC_FLAG_BANK1_BUSY	FMC bank1 忙标志
FMC_FLAG_BANK1_PGERR	FMC Bank1 操作错误标志位
FMC_FLAG_BANK1_WPERR	FMC bank1 擦除/程序保护错误标志位
FMC_FLAG_BANK1_END	FMC bank1 操作结束标志位

枚举类型 fmc_interrupt_flag_enum

表 3-329.枚举类型 fmc_interrupt_flag_enum

成员名称	功能描述
------	------

FMC_INT_FLAG_BANK0_PGERR	FMC bank0 操作错误中断标志位
FMC_INT_FLAG_BANK0_WPERR	FMC bank0 擦写保护错误中断标志位
FMC_INT_FLAG_BANK0_END	FMC bank0 操作结束中断标志位
FMC_INT_FLAG_BANK1_PGERR	FMC Bank1 操作错误中断标志位
FMC_INT_FLAG_BANK1_WPERR	FMC Bank1 擦除/编程保护错误中断标志位
FMC_INT_FLAG_BANK1_END	FMC bank1 操作结束中断标志位

函数 fmc_unlock

函数fmc_unlock描述见下表：

表 3-330.函数 fmc_unlock

函数名称	fmc_unlock
函数原型	void fmc_unlock(void)
功能描述	解锁主 FMC 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 fmc */
```

```
fmc_unlock();
```

函数 fmc_bank0_unlock

函数fmc_bank0_unlock描述见下表：

表 3-331.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 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 fmc bank0 */
fmc_bank0_unlock();
```

函数 fmc_bank1_unlock

函数fmc_bank1_unlock描述见下表：

表 3-332.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 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 fmc bank1 */
fmc_bank1_unlock();
```

函数 fmc_lock

函数fmc_lock描述见下表：

表 3-333.函数 fmc_lock

函数名称	fmc_lock
函数原型	void fmc_lock(void)
功能描述	锁定主 FMC 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 fmc */
```

fmc_lock();

函数 fmc_bank0_lock

函数fmc_bank0_lock描述见下表:

表 3-334.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 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:

```
/* lock fmc bank0 */
```

```
fmc_bank0_lock();
```

函数 fmc_bank1_lock

函数fmc_bank1_lock描述见下表:

表 3-335.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 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 fmc bank1 */
```

```
fmc_bank1_lock();
```

函数 fmc_page_erase

函数fmc_page_erase描述见下表:

表 3-336.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	-
被调用函数	-
输入参数 {in}	
page_address	要擦除的页地址。
输出参数 {out}	
-	-
返回值	
fmc_state_enum	state of FMC, refer to 枚举类型 fmc_state_enum

例如:

```
/* erase page of 0x08000000 */
fmc_state_enum state;
state = fmc_page_erase(0x08000000);
```

函数 fmc_mass_erase

函数fmc_mass_erase描述见下表:

表 3-337.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	擦除整个芯片
先决条件	-
被调用函数	-
输入参数 {in}	
-	-
输出参数 {out}	
-	-
返回值	
fmc_state_enum	state of FMC, refer to 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 3-338.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	擦除 bank0
先决条件	-
被调用函数	-
输入参数 {in}	
-	-
输出参数 {out}	
-	-
返回值	
fmc_state_enum	state of FMC, refer to 枚举类型 fmc_state_enum

例如：

```
/* erase bank0 */

fmc_state_enum state;

state = fmc_bank0_erase();
```

函数 fmc_bank1_erase

函数fmc_bank1_erase描述见下表：

表 3-339.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	擦除 Bank1
先决条件	-
被调用函数	-
输入参数 {in}	
-	-
输出参数 {out}	
-	-
返回值	
fmc_state_enum	state of FMC, refer to 枚举类型 fmc_state_enum

例如：

```
/* erase bank1 */

fmc_state_enum state;

state = fmc_bank1_erase();
```


函数 fmc_word_program

函数fmc_word_program描述见下表：

表 3-340.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	在对应地址编程一个字。
先决条件	-
被调用函数	-
输入参数 {in}	
address	用于编程的地址
输入参数 {in}	
data	待编程字
输出参数 {out}	
-	-
返回值	
fmc_state_enum	state of FMC, refer to 枚举类型 fmc_state_enum

例如：

```
/* Write 0x12345678 to the address 0x08000000. */
```

```
fmc_state_enum state;
```

```
state = fmc_word_program(0x08000000, 0x12345678);
```

函数 fmc_halfword_program

函数fmc_halfword_program描述见下表：

表 3-341.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	在对应地址编程半字
先决条件	-
被调用函数	-
输入参数 {in}	
address	待编程的地址
输入参数 {in}	
data	半字待编程
输出参数 {out}	
-	-
返回值	
fmc_state_enum	state of FMC, refer to 枚举类型 fmc_state_enum

例如：

```
/* Write 0x1234 to the address 0x08000000. */
```

```
fmc_state_enum state;
```

```
state = fmc_halfword_program(0x08000000, 0x1234);
```

函数 fmc_otp_word_program

函数fmc_otp_word_program描述见下表：

表 3-342.函数 fmc_otp_word_program

函数名称	fmc_otp_word_program
函数原型	fmc_state_enum fmc_otp_word_program(uint32_t address, uint32_t data)
功能描述	在对应地址编程一个字。
先决条件	-
被调用函数	-
输入参数 {in}	
address	OTP 地址用于编程
输入参数 {in}	
data	编程字
输出参数 {out}	
-	-
返回值	
fmc_state_enum	state of FMC, refer to 枚举类型 fmc_state_enum

例如：

```
/* Write 0x12345678 to the address 0x1FF00000 */
```

```
fmc_state_enum state;
```

```
state = fmc_otp_word_program(0x1FF00000, 0x12345678);
```

函数 fmc_otp_half_word_program

函数fmc_otp_half_word_program描述见下表：

表 3-343.函数 fmc_otp_half_word_program

函数名称	fmc_otp_half_word_program
函数原型	fmc_state_enum fmc_otp_half_word_program(uint32_t address, uint16_t data)
功能描述	在对应地址编程半字
先决条件	-
被调用函数	-
输入参数 {in}	
address	OTP 地址用于编程
输入参数 {in}	
data	半字编程

输出参数 {out}	
-	-
返回值	
fmc_state_enum	state of FMC, refer to 枚举类型 fmc_state_enum

例如:

```
/* Write 0x1234 to the address 0x1FF00000 */
```

```
fmc_state_enum state;
```

```
state = fmc_otp_half_word_program(0x1FF00000, 0x1234);
```

函数 fmc_otp_byte_program

函数fmc_otp_byte_program描述见下表:

表 3-344.函数 fmc_otp_byte_program

函数名称	fmc_otp_byte_program
函数原型	fmc_state_enum fmc_otp_byte_program(uint32_t address, uint8_t data)
功能描述	在对应地址编程一个字节
先决条件	-
被调用函数	-
输入参数 {in}	
address	用于编程的 OTP 地址
输入参数 {in}	
data	待编程字节 (0x00 - 0xFF)
输出参数 {out}	
-	-
返回值	
fmc_state_enum	state of FMC, refer to 枚举类型 fmc_state_enum

例如:

```
/* Write 0x12 to the address 0x1FF00000 */
```

```
fmc_state_enum state;
```

```
state = fmc_otp_byte_program(0x1FF00000, 0x12);
```

函数 fmc_nwa_enable

函数fmc_nwa_enable描述见下表:

表 3-345.函数 fmc_nwa_enable

函数名称	fmc_nwa_enable
函数原型	void fmc_nwa_enable(void)
功能描述	使能系统复位后无等待时间区载入
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 no waiting time area load after system reset */
```

```
fmc_nwa_enable();
```

函数 fmc_nwa_disable

函数fmc_nwa_disable描述见下表:

表 3-346.函数 fmc_nwa_disable

函数名称	fmc_nwa_disable
函数原型	void fmc_nwa_disable(void)
功能描述	系统复位后失能无等待时间区域加载
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 no waiting time area load after system reset */
```

```
fmc_nwa_disable();
```

函数 otp1_read_disable

函数otp1_read_disable描述见下表:

表 3-347.函数 otp1_read_disable

函数名称	otp1_read_disable
函数原型	void otp1_read_disable(uint32_t block)
功能描述	设置 OTP1 数据块不可读取
先决条件	-
被调用函数	-
输入参数 {in}	

block	指定 OTP1 数据块 x 不可读
<i>OTP1_DATA_BLOCK_x</i>	数据块 x (x = 0,1,2...15)
<i>OTP1_DATA_BLOCK_ALL</i>	所有数据块
输出参数 {out}	
-	-
返回值	
-	-

例如:

```
/* set all OTP1 data block not be read */
```

```
otp1_read_disable(OTP1_DATA_BLOCK_ALL);
```

函数 otp2_rlock_enable

函数otp2_rlock_enable描述见下表:

表 3-348.函数 otp2_rlock_enable

函数名称	otp2_rlock_enable
函数原型	void otp2_rlock_enable(void)
功能描述	使能 OTP2 的读锁块保护
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 lock block protection for OTP2 */
```

```
otp2_rlock_enable();
```

函数 fmc_deep_power_down_enable

函数fmc_deep_power_down_enable描述见下表:

表 3-349.函数 fmc_deep_power_down_enable

函数名称	fmc_deep_power_down_enable
函数原型	void fmc_deep_power_down_enable(void)
功能描述	当无操作时使能深度掉电模式
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 deep power down mode when no operation */
```

```
fmc_deep_power_down_enable();
```

函数 fmc_deep_power_down_disable

函数fmc_deep_power_down_disable描述见下表:

表 3-350.函数 fmc_deep_power_down_disable

函数名称	fmc_deep_power_down_disable
函数原型	void fmc_deep_power_down_disable(void)
功能描述	当无操作时失能深度掉电模式
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 deep power down mode when no operation */
```

```
fmc_deep_power_down_disable();
```

函数 ob_unlock

函数ob_unlock描述见下表:

表 3-351.函数 ob_unlock

函数名称	ob_unlock
函数原型	void ob_unlock(void)
功能描述	解锁 Option Byte 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 option byte operation */
```

```
ob_unlock();
```

函数 ob_lock

函数ob_lock描述见下表:

表 3-352.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	-
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_start

函数ob_start描述见下表:

表 3-353.函数 ob_start

函数名称	ob_start
函数原型	void ob_start(void)
功能描述	发送选项字节更改命令
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 option byte change command */
```

```
ob_start();
```

函数 ob_erase

函数ob_erase描述见下表:

表 3-354.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	-
被调用函数	-
输入参数 {in}	
-	-
输出参数 {out}	
-	-
返回值	
-	-

例如:

```
/* erase option byte */
```

```
ob_erase();
```

函数 ob_write_protection_enable

函数ob_write_protection_enable描述见下表:

表 3-355.函数 ob_write_protection_enable

函数名称	ob_write_protection_enable
函数原型	ErrStatus ob_write_protection_enable(uint32_t ob_wp)
功能描述	使能写保护
先决条件	-
被调用函数	-
输入参数 {in}	
ob_wp	指定需写保护的页
OB_WP_x(x=0..31)	扇区 x (x = 0,1,2...31)
OB_WP_ALL	所有页
输出参数 {out}	
-	-
返回值	

ErrStatus	SUCCESS or ERROR
------------------	------------------

例如:

```
/* enable page0/1 write protection */
ob_write_protection_enable(OB_WP_0);
```

函数 ob_write_protection_disable

函数ob_write_protection_disable描述见下表:

表 3-356.函数 ob_write_protection_disable

函数名称	ob_write_protection_disable
函数原型	ErrStatus ob_write_protection_disable(uint32_t ob_wp)
功能描述	失能写保护
先决条件	-
被调用函数	-
输入参数 {in}	
ob_wp	指定需写保护的扇区
<i>OB_WP_x</i> (<i>x</i> =0..31)	扇区 <i>x</i> (<i>x</i> = 0,1,2...31)
<i>OB_WP_ALL</i>	所有 sector
输出参数 {out}	
-	-
返回值	
ErrStatus	SUCCESS or ERROR

例如:

```
/* disable page0/1 write protection */
ob_write_protection_disable(OB_WP_0);
```

函数 ob_security_protection_config

函数ob_security_protection_config描述见下表:

表 3-357.函数 ob_security_protection_config

函数名称	ob_security_protection_config
函数原型	void ob_security_protection_config(uint32_t ob_spc)
功能描述	配置安全保护等级
先决条件	-
被调用函数	-
输入参数 {in}	
ob_spc	指定安全保护等级
<i>FMC_NSPC</i>	无安全保护
<i>FMC_LSPC</i>	低安全性保护
<i>FMC_HSPC</i>	高安全保护

输出参数 {out}	
-	-
返回值	
-	-

例如:

```
/* configure low security protection level */
```

```
ob_security_protection_config(FMC_LSPC);
```

函数 ob_user_write

函数ob_user_write描述见下表:

表 3-358.函数 ob_user_write

函数名称	ob_user_write
函数原型	void ob_user_write(uint32_t ob_fwdgt, uint32_t ob_deepsleep, uint32_t ob_stdby)
功能描述	编程 FMC 用户选项字节
先决条件	-
被调用函数	-
输入参数 {in}	
ob_fwdgt	选项字节看门狗值
OB_FWDGT_SW	软件自由看门狗
OB_FWDGT_HW	硬件独立看门狗
输入参数 {in}	
ob_deepsleep	选项字节深度休眠复位值
OB_DEEPSLEEP_N_RST	进入 DeepSleep 模式时无复位
OB_DEEPSLEEP_RST	生成复位, 而非进入深度睡眠模式
输入参数 {in}	
ob_stdby	选项字节待机复位值
OB_STDBY_N_RST	进入 Standby 模式时不复位
OB_STDBY_RST	生成复位, 而不是进入待机模式
输出参数 {out}	
-	-
返回值	
-	-

例如:

```
/* Configure the option bytes to enable the hardware watchdog */
```

```
ob_user_write(OB_FWDGT_HW, OB_DEEPSLEEP_N_RST, OB_STDBY_N_RST);
```

函数 ob_sram_init_config

函数ob_sram_init_config描述见下表:

表 3-359.函数 ob_sram_init_config

函数名称	ob_sram_init_config
函数原型	void ob_sram_init_config(uint32_t init_mode)
功能描述	配置 Option Byte SRAM 初始化值
先决条件	-
被调用函数	-
输入参数 {in}	
init_mode	指定 Option Byte 中 SRAM 初始化值
OB_SRAM_INI_ENABLE	SRAM 在上电复位后初始化
OB_SRAM_INI_DISABLE	SRAM 在上电复位后未初始化
输出参数 {out}	
-	-
返回值	
-	-

例如:

```
/* Enable SRAM power-on initialization */
```

```
ob_sram_init_config(OB_SRAMCINI_ENABLE);
```

函数 ob_sram_ecc_config

函数ob_sram_ecc_config描述见下表:

表 3-360.函数 ob_sram_ecc_config

函数名称	ob_sram_ecc_config
函数原型	void ob_sram_ecc_config(uint32_t ecc_mode)
功能描述	配置选项字节 SRAM ECC 功能值
先决条件	-
被调用函数	-
输入参数 {in}	
ecc_mode	指定选项字节 SRAMC ECC 功能值
OB_SRAMC_ECC_DISABLE	SRAM ECC 失能
OB_SRAMC_ECC_ENABLE	SRAM ECC 使能
输出参数 {out}	
-	-
返回值	
-	-

例如:

```
/* Enable SRAM ECC verification */
```

```
ob_sram_ecc_config(OB_SRAMC_ECC_ENABLE);
```

函数 ob_nwa_clock_config

函数ob_nwa_clock_config描述见下表：

表 3-361.函数 ob_nwa_clock_config

函数名称	ob_nwa_clock_config
函数原型	void ob_nwa_clock_config(uint32_t clock_mode)
功能描述	配置 Option Byte 时钟以加载零等待时间区域
先决条件	-
被调用函数	-
输入参数 {in}	
clock_mode	指定电源复位后读取 flash 到零等待区时的时钟。
<i>PLL_CLK_200M</i>	来自 PLL 的 200M
<i>PLL_CLK_160M</i>	来自 PLL 的 160M
<i>PLL_CLK_120M</i>	来自 PLL 的 120M
<i>IRC8M_CLK_8M</i>	来自 IRC8M 的 8M
输出参数 {out}	
-	-
返回值	
-	-

例如：

```
/* select PLL 200MHz */
ob_nwa_clock_config(PLL_CLK_200M);
```

函数 ob_data_write

函数ob_data_write描述见下表：

表 3-362.函数 ob_data_write

函数名称	ob_data_write
函数原型	void ob_data_write(uint16_t data)
功能描述	编程选项字节数据
先决条件	-
被调用函数	-
输入参数 {in}	
data	待编程数据
输出参数 {out}	
-	-
返回值	
-	-

例如：

```
/* Program the option byte DATA field to 0x1234 */
```

```
ob_data_write(0x1234);
```

函数 ob_sram_init_get

函数ob_sram_init_get描述见下表:

表 3-363.函数 ob_sram_init_get

函数名称	ob_sram_init_get
函数原型	uint32_t ob_sram_init_get(void)
功能描述	获取选项字节 SRAM 初始化值
先决条件	-
被调用函数	-
输入参数 {in}	
-	-
输出参数 {out}	
-	-
返回值	
uint32_t	SRAM 初始化状态
OB_SRAMCINI_DISABLE	失能
OB_SRAMCINI_ENABLE	使能

例如:

```
/* Obtain the SRAM initialization configuration status. */
```

```
uint32_t sram_init;
```

```
sram_init = ob_sram_init_get();
```

函数 ob_sram_ecc_get

函数ob_sram_ecc_get描述见下表:

表 3-364.函数 ob_sram_ecc_get

函数名称	ob_sram_ecc_get
函数原型	uint32_t ob_sram_ecc_get(void)
功能描述	获取选项字节 SRAMC ECC 使能值
先决条件	-
被调用函数	-
输入参数 {in}	
-	-
输出参数 {out}	
-	-
返回值	
uint32_t	SRAM ECC 功能值
OB_SRAMC_ECC_ENABLE	使能 SRAM ECC
OB_SRAMC_ECC_DISABLE	失能 SRAM ECC

例如:

```
/* get the option byte sramc ecc enable value */
uint32_t sram_ecc;

sram_ecc = ob_sram_ecc_get();
```

函数 ob_nwa_clock_get

函数ob_nwa_clock_get描述见下表:

表 3-365.函数 ob_nwa_clock_get

函数名称	ob_nwa_clock_get
函数原型	uint32_t ob_nwa_clock_get(void)
功能描述	获取选项字节加载时钟无等待时间区域
先决条件	-
被调用函数	-
输入参数 {in}	
-	-
输出参数 {out}	
-	-
返回值	
uint32_t	电源复位后读取 flash 到内存时的时钟
PLL_CLK_200M	来自 PLL 的 200M
PLL_CLK_160M	来自 PLL 的 160M
PLL_CLK_120M	来自 PLL 的 120M
HSI_CLK_8M	来自 HSI 的 8M

例如:

```
/* get the option byte the clock for loading no waiting time area */
uint32_t nwa_clock;

nwa_clock = ob_nwa_clock_get();
```

函数 ob_data_get

函数ob_data_get描述见下表:

表 3-366.函数 ob_data_get

函数名称	ob_data_get
函数原型	uint16_t ob_data_get(void)
功能描述	获取 FMC 数据选项字节
先决条件	-
被调用函数	-
输入参数 {in}	
-	-

输出参数 {out}	
-	-
返回值	
uint16_t	选项字节 DATA 字段值

例如:

```
/* check whether FMC bank1 is ready or not */
```

```
uint16_t ob_data;
```

```
ob_data = ob_data_get();
```

函数 ob_write_protection_get

函数ob_write_protection_get描述见下表:

表 3-367.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 FMC 选项字节写保护
先决条件	-
被调用函数	-
输入参数 {in}	
-	-
输出参数 {out}	
-	-
返回值	
uint32_t	选项字节写保护值

例如:

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

```
uint32_t wp_value;
```

```
wp_value = ob_write_protection_get();
```

函数 ob_spc_get

函数ob_spc_get描述见下表:

表 3-368.函数 ob_spc_get

函数名称	ob_spc_get
函数原型	uint32_t ob_spc_get(void)
功能描述	获取选项字节安全保护码值
先决条件	-
被调用函数	-
输入参数 {in}	

-	-
输出参数 {out}	
-	-
返回值	
uint32_t	FMC_NSPC, FMC_LSPC, FMC_HSPC

例如:

```
/* get option byte security protection code value */
```

```
uint32_t ob_spc;
```

```
ob_spc = ob_spc_get();
```

函数 fmc_interrupt_enable

函数fmc_interrupt_enable描述见下表:

表 3-369.函数 fmc_interrupt_enable

函数名称	fmc_interrupt_enable
函数原型	void fmc_interrupt_enable(fmc_interrupt_enum interrupt)
功能描述	使能 FMC 中断
先决条件	-
被调用函数	-
输入参数 {in}	
interrupt	FMC 中断源
FMC_INT_BANK0_END	bank0 程序结束中断
FMC_INT_BANK0_ERR	bank0 错误中断
FMC_INT_BANK1_END	bank1 程序结束中断
FMC_INT_BANK1_ERR	bank1 错误中断
输出参数 {out}	
-	-
返回值	
-	-

例如:

```
/* Enable BANK0 complete interrupt. */
```

```
fmc_interrupt_enable(FMC_INT_BANK0_END);
```

函数 fmc_interrupt_disable

函数fmc_interrupt_disable描述见下表:

表 3-370.函数 fmc_interrupt_disable

函数名称	fmc_interrupt_disable
函数原型	void fmc_interrupt_disable(fmc_interrupt_enum interrupt)
功能描述	失能 FMC 中断

先决条件	-
被调用函数	-
输入参数 {in}	
interrupt	bank0 程序结束中断
<i>FMC_INT_BANK0_END</i>	bank0 错误中断
<i>FMC_INT_BANK0_ERR</i>	bank1 程序结束中断
<i>FMC_INT_BANK1_END</i>	bank1 错误中断
<i>FMC_INT_BANK1_ERR</i>	bank0 程序结束中断
输出参数 {out}	
-	-
返回值	
-	-

例如:

```
/* disable BANK0 complete 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 3-371.函数 fmc_interrupt_flag_get

函数名称	fmc_interrupt_flag_get
函数原型	FlagStatus fmc_interrupt_flag_get(fmc_interrupt_flag_enum flag)
功能描述	获取 FMC 中断标志状态
先决条件	-
被调用函数	-
输入参数 {in}	
flag	FMC 中断标志, 参见 fmc_interrupt_flag_enum
<i>FMC_INT_FLAG_BANK0_PGERR</i>	FMC bank0 操作错误中断标志位
<i>FMC_INT_FLAG_BANK0_WPERR</i>	FMC bank0 擦除/编程保护错误中断标志位
<i>FMC_INT_FLAG_BANK0_END</i>	FMC Bank0 操作结束中断标志位
<i>FMC_INT_FLAG_BANK1_PGERR</i>	FMC bank1 操作错误中断标志位
<i>FMC_INT_FLAG_BANK1_WPERR</i>	FMC bank1 擦除/编程保护错误中断标志位
<i>FMC_INT_FLAG_BANK1_END</i>	FMC bank1 操作结束中断标志位
输出参数 {out}	
-	-
返回值	
FlagStatus	SET 或 RESET

例如:

```
/* Get the BANK0 operation complete flag */
```

FlagStatus flag;

```
flag = fmc_interrupt_flag_get(FMC_INT_FLAG_BANK0_END);
```

函数 fmc_interrupt_flag_clear

函数fmc_interrupt_flag_clear描述见下表:

表 3-372.函数 fmc_interrupt_flag_clear

函数名称	fmc_interrupt_flag_clear
函数原型	void fmc_interrupt_flag_clear(fmc_interrupt_flag_enum flag)
功能描述	清除 FMC 中断标志状态
先决条件	-
被调用函数	-
输入参数 {in}	
flag	FMC 中断标志, 请参考 fmc_interrupt_flag_enum
FMC_INT_FLAG_BANK0_PGERR	FMC Bank0 操作错误中断标志位
FMC_INT_FLAG_BANK0_WPERR	FMC bank0 擦除/编程保护错误中断标志位
FMC_INT_FLAG_BANK0_END	FMC bank0 操作结束中断标志位
FMC_INT_FLAG_BANK1_PGERR	FMC bank1 操作错误中断标志位
FMC_INT_FLAG_BANK1_WPERR	FMC bank1 擦除/编程保护错误中断标志位
FMC_INT_FLAG_BANK1_END	FMC Bank1 操作结束中断标志位
输出参数 {out}	
-	-
返回值	
-	-

例如:

```
/* clear the BANK0 operation complete flag */
```

```
fmc_interrupt_flag_clear(FMC_INT_FLAG_BANK0_END);
```

函数 fmc_flag_get

函数fmc_flag_get描述见下表:

表 3-373.函数 fmc_flag_get

函数名称	fmc_flag_get
函数原型	FlagStatus fmc_flag_get(fmc_flag_enum flag)
功能描述	检查标志是否置位
先决条件	-
被调用函数	-
输入参数 {in}	
flag	检查 FMC 标志
FMC_FLAG_BANK0_BUSY	FMC bank0 忙标志位
FMC_FLAG_BANK0_PGERR	FMC bank0 操作错误标志位

<i>FMC_FLAG_BANK0_WPERR</i>	FMC Bank0 擦除/编程保护错误标志位
<i>FMC_FLAG_BANK0_END</i>	FMC bank0 操作结束标志位
<i>FMC_FLAG_OBERR</i>	FMC 选项字节读取错误标志位
<i>FMC_FLAG_BANK1_BUSY</i>	FMC bank1 忙标志位
<i>FMC_FLAG_BANK1_PGERR</i>	FMC bank1 操作错误标志位
<i>FMC_FLAG_BANK1_WPERR</i>	FMC bank1 擦除/编程保护错误标志位
<i>FMC_FLAG_BANK1_END</i>	FMC bank1 操作结束标志位
输出参数 {out}	
-	-
返回值	
FlagStatus	SET 或 RESET

例如:

```
/* Get the BANK0 complete interrupt 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 3-374.函数 fmc_flag_clear

函数名称	fmc_flag_clear
函数原型	void fmc_flag_clear(fmc_flag_enum flag)
功能描述	清除 FMC 标志
先决条件	-
被调用函数	-
输入参数 {in}	
flag	清除 FMC 标志
<i>FMC_FLAG_BANK0_PGERR</i>	FMC bank0 操作错误标志位
<i>FMC_FLAG_BANK0_WPERR</i>	FMC Bank0 擦除/编程保护错误标志位
<i>FMC_FLAG_BANK0_END</i>	FMC bank0 操作结束标志位
<i>FMC_FLAG_BANK1_PGERR</i>	FMC Bank1 操作错误标志位
<i>FMC_FLAG_BANK1_WPERR</i>	FMC Bank1 擦除/编程保护错误标志位
<i>FMC_FLAG_BANK1_END</i>	FMC Bank1 操作结束标志位
输出参数 {out}	
-	-
返回值	
-	-

例如:

```
/* clear the BANK0 complete interrupt flag */
```

```
fmc_flag_clear(FMC_FLAG_BANK0_END);
```

函数 fmc_bank0_state_get

函数fmc_bank0_state_get描述见下表：

表 3-375.函数 fmc_bank0_state_get

函数名称	fmc_bank0_state_get
函数原型	fmc_state_enum fmc_bank0_state_get(void)
功能描述	获取 FMC bank0 状态
先决条件	-
被调用函数	-
输入参数 {in}	
-	-
输出参数 {out}	
-	-
返回值	
fmc_state_enum	state of FMC, refer to 枚举类型 fmc_state_enum

例如：

```
/* Get the BANK0 status. */
fmc_state_enum state;
state = fmc_bank0_state_get();
```

函数 fmc_bank1_state_get

函数fmc_bank1_state_get描述见下表：

表 3-376.函数 fmc_bank1_state_get

函数名称	fmc_bank1_state_get
函数原型	fmc_state_enum fmc_bank1_state_get(void)
功能描述	获取 FMC bank1 状态
先决条件	-
被调用函数	-
输入参数 {in}	
-	-
输出参数 {out}	
-	-
返回值	
fmc_state_enum	state of FMC, refer to 枚举类型 fmc_state_enum

例如：

```
/* Get the BANK1 status. */
fmc_state_enum state;
state = fmc_bank1_state_get();
```

函数 fmc_bank0_ready_wait

函数fmc_bank0_ready_wait描述见下表：

表 3-377.函数 fmc_bank0_ready_wait

函数名称	fmc_bank0_ready_wait
函数原型	fmc_state_enum fmc_bank0_ready_wait(uint32_t timeout)
功能描述	检查 FMC bank0 是否就绪
先决条件	-
被调用函数	-
输入参数 {in}	
timeout	循环计数
输出参数 {out}	
-	-
返回值	
fmc_state_enum	state of FMC, refer to 枚举类型 fmc_state_enum

例如：

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

函数 fmc_bank1_ready_wait

函数fmc_bank1_ready_wait描述见下表：

表 3-378.函数 fmc_bank1_ready_wait

函数名称	fmc_bank1_ready_wait
函数原型	fmc_state_enum fmc_bank1_ready_wait(uint32_t timeout)
功能描述	检查 FMC bank1 是否就绪。
先决条件	-
被调用函数	-
输入参数 {in}	
timeout	循环计数
输出参数 {out}	
-	-
返回值	
fmc_state_enum	state of FMC, refer to 枚举类型 fmc_state_enum

例如：

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

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 3-379. 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 3-380. FWDGT 库函数

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

函数 fwdgt_write_enable

函数fwdgt_write_enable描述见下表：

表 3-381. 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 3-382. 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 3-383. 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 3-384. 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
FWDGT_PSC_DIV512	FWDGT预分频值设为512
FWDGT_PSC_DIV1024	FWDGT预分频值设为1024
FWDGT_PSC_DIV2048	FWDGT预分频值设为2048
FWDGT_PSC_DIV4096	FWDGT预分频值设为4096
FWDGT_PSC_DIV8192	FWDGT预分频值设为8192
FWDGT_PSC_DIV16384	FWDGT预分频值设为16384
FWDGT_PSC_DIV32768	FWDGT预分频值设为32768

2768	
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 3-385. 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 3-386. 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	重装载FWDG计数器
先决条件	-
被调用函数	-
输入参数{in}	

-	-
输出参数{out}	
-	-
返回值	
-	-

例如:

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

```
fwdgt_counter_reload( );
```

函数 fwdgt_config

函数fwdgt_config描述见下表:

表 3-387. 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预分频值设为128
FWDGT_PSC_DIV256	FWDGT预分频值设为256
FWDGT_PSC_DIV512	FWDGT预分频值设为512
FWDGT_PSC_DIV1024	FWDGT预分频值设为1024
FWDGT_PSC_DIV2048	FWDGT预分频值设为2048
FWDGT_PSC_DIV4096	FWDGT预分频值设为4096

<i>FWDGT_PSC_DIV8</i> 192	FWDGT预分频值设为8192
<i>FWDGT_PSC_DIV1</i> 6384	FWDGT预分频值设为16384
<i>FWDGT_PSC_DIV3</i> 2768	FWDGT预分频值设为32768
输出参数{out}	
-	-
返回值	
ErrStatus	ERROR / SUCCESS

例如:

```
/* configure 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 3-388. 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
<i>FWDGT_FLAG_PUD</i>	预分频值更新进行中
<i>FWDGT_FLAG_RUD</i>	重装载值更新进行中
输出参数{out}	
-	-
返回值	
FlagStatus	SET / RESET

例如:

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

```
FlagStatus status;
```

```
status = fwdgt_flag_get (FWDGT_FLAG_PUD);
```

```
if(status == RESET)
```

```
{
```

```

...
}else
{
...
}

```

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 3-389. GPIO 寄存器

寄存器名称	寄存器描述
GPIOx_CTL	端口控制寄存器
GPIOx_OMODE	端口输出模式寄存器
GPIOx_OSPD0	端口输出速度寄存器0
GPIOx_PUD	端口上拉/下拉寄存器
GPIOx_ISTAT	端口输入状态寄存器
GPIOx_OCTL	端口输出控制寄存器
GPIOx_BOP	端口位操作寄存器
GPIOx_LOCK	端口配置锁定寄存器
GPIOx_AFSEL0	备用功能选择寄存器0
GPIOx_AFSEL1	备用功能选择寄存器1
GPIOx_BC	位清除寄存器
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

3.16.2. 外设库函数说明

GPIO库函数列表如下表所示：

表 3-390. GPIO 库函数

库函数名称	库函数描述
gpio_deinit	复位外设GPIOx
gpio_afio_deinit	复位AFIO
gpio_mode_set	设置GPIO模式

库函数名称	库函数描述
gpio_output_options_set	设置GPIO输出模式和速度
gpio_bit_set	置位引脚值
gpio_bit_reset	复位引脚值
gpio_bit_write	将特定的值写入引脚
gpio_port_write	将特定的值写入一组端口
gpio_input_bit_get	获取引脚的输入值
gpio_input_port_get	获取一组端口的输入值
gpio_output_bit_get	获取引脚的输出值
gpio_output_port_get	获取一组端口的输出值
gpio_af_set	设置GPIO复用功能
gpio_exti_source_select	选择哪个引脚作为EXTI源
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 3-391. 函数 gpio_deinit

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

例如：

```
/* reset GPIOA */
```

```
gpio_deinit (GPIOA);
```

函数 gpio_afio_deinit

函数gpio_afio_deinit描述见下表：

表 3-392. 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_mode_set

函数gpio_mode_set描述见下表:

表 3-393. 函数 gpio_mode_set

函数名称	gpio_mode_set
函数原型	void gpio_mode_set(uint32_t gpio_periph, uint32_t mode, uint32_t pull_up_down, uint32_t pin);
功能描述	设置GPIO模式
先决条件	-
被调用函数	rcu_periph_reset_enable / rcu_periph_reset_disable
输入参数{in}	
gpio_periph	GPIO端口
GPIOx	端口选择(x = A,B,C,D,E)
输入参数{in}	
mode	GPIO引脚模式
GPIO_MODE_INPUT	输入模式
GPIO_MODE_OUTPUT	输出模式
GPIO_MODE_AF	备用功能模式
GPIO_MODE_ANALOG	模拟模式
输入参数{in}	
pull_up_down	GPIO引脚上拉下拉电阻设置
GPIO_PUPD_NONE	悬空模式, 无上拉和下拉
GPIO_PUPD_PULLUP	带上拉电阻
GPIO_PUPD_PULLDOWN	带下拉电阻
输入参数{in}	
pin	GPIO pin

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

例如:

```
/* config PA0 as input mode with pullup */
```

```
gpio_mode_set (GPIOA, GPIO_MODE_INPUT, GPIO_PUPD_PULLUP, GPIO_PIN_0);
```

函数 gpio_output_options_set

函数gpio_output_options_set描述见下表:

表 3-394. 函数 gpio_output_options_set

函数名称	gpio_output_options_set
函数原型	void gpio_output_options_set(uint32_t gpio_periph, uint8_t otype, uint32_t speed, uint32_t pin);
功能描述	设置GPIO输出模式和速度
先决条件	-
被调用函数	-
输入参数{in}	
gpio_periph	GPIO端口
GPIOx	端口选择(x = A,B,C,D,E)
输入参数{in}	
otype	GPIO引脚输出模式
GPIO_OTYPE_PP	推挽输出模式
GPIO_OTYPE_OD	开漏输出模式
输入参数{in}	
speed	GPIO引脚输出最大速度
GPIO_OSPEED_LE VEL0	最大输出速度为速度等级0
GPIO_OSPEED_LE VEL1	最大输出速度为速度等级1
GPIO_OSPEED_LE VEL2	最大输出速度为速度等级2
GPIO_OSPEED_LE VEL3	最大输出速度为速度等级3
输入参数{in}	
pin	GPIO引脚
GPIO_PIN_x	引脚选择 (x=0..15)
GPIO_PIN_ALL	所有引脚

输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* config PA0 as push pull mode */
```

```
gpio_output_options_set (GPIOA, GPIO_OTYPE_PP, GPIO_OSPEED_2MHZ,
GPIO_PIN_0);
```

函数 gpio_bit_set

函数gpio_bit_set描述见下表:

表 3-395. 函数 gpio_bit_set

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

例如:

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

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

函数 gpio_bit_reset

函数gpio_bit_reset描述见下表:

表 3-396. 函数 gpio_bit_reset

函数名称	gpio_bit_reset
函数原型	void gpio_bit_reset(uint32_t gpio_periph,uint32_t pin);

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

例如:

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

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

函数 gpio_bit_write

函数gpio_bit_write描述见下表:

表 3-397. 函数 gpio_bit_write

函数名称	gpio_bit_write
函数原型	void gpio_bit_write(uint32_t gpio_periph,uint32_t pin,bit_status bit_value);
功能描述	将特定的值写入引脚
先决条件	-
被调用函数	-
输入参数{in}	
gpio_periph	GPIO端口
GPIOx	端口选择(x = A,B,C,D,E)
输入参数{in}	
pin	GPIO引脚
GPIO_PIN_x	引脚选择(x=0..15)
GPIO_PIN_ALL	所有引脚
输入参数{in}	
bit_value	设置或清除
RESET	清除引脚值
SET	设置引脚值
输出参数{out}	
-	-
返回值	

-	-
---	---

例如:

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

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

函数 gpio_port_write

函数gpio_port_write描述见下表:

表 3-398. 函数 gpio_port_write

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

例如:

```
/* write 1010 0101 1010 0101 to Port A */
```

```
gpio_port_write (GPIOA, 0xA5A5);
```

函数 gpio_input_bit_get

函数gpio_input_bit_get描述见下表:

表 3-399. 函数 gpio_input_bit_get

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

输入参数{in}	
pin	GPIO引脚
<i>GPIO_PIN_x</i>	引脚选择 (x=0..15)
<i>GPIO_PIN_ALL</i>	所有引脚
输出参数{out}	
-	-
返回值	
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 3-400. 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
<i>GPIOx</i>	端口选择(x = A,B,C,D,E)
输出参数{out}	
-	-
返回值	
uint16_t	0x0000-0xFFFF

例如:

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

```
uint16_t port_state;
```

```
port_state = gpio_input_port_get (GPIOA);
```

函数 gpio_output_bit_get

函数gpio_output_bit_get描述见下表:

表 3-401. 函数 gpio_output_bit_get

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

例如:

```
/* get output status of PA0 */
```

```
FlagStatus bit_state;
```

```
bit_state = gpio_output_bit_get (GPIOA, GPIO_PIN_0);
```

函数 gpio_output_port_get

函数gpio_output_port_get描述见下表:

表 3-402. 函数 gpio_output_port_get

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

例如:

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

```
uint16_t port_state;

port_state = gpio_output_port_get (GPIOA);
```

函数 gpio_af_set

函数gpio_af_set描述见下表:

表 3-403. 函数 gpio_af_set

函数名称	gpio_af_set
函数原型	void gpio_af_set(uint32_t gpio_periph, uint32_t alt_func_num, uint32_t pin);
功能描述	设置GPIO的备用功能
先决条件	-
被调用函数	-
输入参数{in}	
gpio_periph	GPIO 端口
GPIOx	端口选择(x = A,B,C,D,E)
输入参数{in}	
alt_func_num	GPIO引脚备用功能, 请参见特定设备的数据手册
GPIO_AF_0	备用功能0
GPIO_AF_1	备用功能1
GPIO_AF_2	备用功能2
GPIO_AF_3	备用功能3
GPIO_AF_4	备用功能4
GPIO_AF_5	备用功能5
GPIO_AF_6	备用功能6
GPIO_AF_7	备用功能7
GPIO_AF_8	备用功能8
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 alternate function 0*/

gpio_af_set(GPIOA, GPIO_AF_0, GPIO_PIN_0);
```

函数 gpio_exti_source_select

函数gpio_exti_source_select描述见下表:

表 3-404. 函数 gpio_exti_source_select

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

例如:

```
/* config PA0 as EXTI source */
```

```
gpio_exti_source_select(GPIO_PORT_SOURCE_GPIOA, GPIO_PIN_SOURCE_0);
```

函数 gpio_pin_lock

函数gpio_pin_lock描述见下表:

表 3-405. 函数 gpio_pin_lock

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

-	-
---	---

例如:

```
/* lock PA0 */
```

```
gpio_pin_lock (GPIOA, GPIO_PIN_0);
```

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-256）。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 3-406. 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 3-407. 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_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

库函数名称	库函数描述
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	配置使用多DMA或单DMA，从未确定是否在DMA传输结束后计算摘要
hau_dma_enable	使能HAU DMA接口
hau_dma_disable	除能HAU DMA接口
hau_hash_sha_256	在HASH模式下使用SHA256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_digest_parameter_struct

表 3-408. 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 3-409. 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	rcu_periph_reset_enable / rcu_periph_reset_disable
输入参数{in}	
-	-
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* reset the HAU peripheral */
```

```
hau_deinit();
```

函数 hau_init

函数hau_init描述见下表：

表 3-410. 函数 hau_init

函数名称	hau_init
函数原形	void hau_init(hau_init_parameter_struct* initpara);
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}	
datatype	数据交换模式
HAU_SWAPPING_32BIT	不交换
HAU_SWAPPING_16BIT	半字交换
HAU_SWAPPING_8BIT	字节交换
HAU_SWAPPING_1BIT	位交换
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* initialize the HAU peripheral parameters */
```

```
hau_init(HAU_SWAPPING_8BIT);
```

函数 hau_reset

函数hau_reset描述见下表：

表 3-411. 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 3-412. 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 3-413. 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 3-414. 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 3-415. 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	消息摘要结果，参考结构体 表3-408. 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 3-416. 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 3-417. 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，从未确定是否在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 3-418. 函数 `hau_dma_enable`

函数名称	<code>hau_dma_enable</code>
函数原形	<code>void hau_dma_enable(void);</code>
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 3-419. 函数 `hau_dma_disable`

函数名称	<code>hau_dma_disable</code>
函数原形	<code>void hau_dma_disable(void);</code>
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_256

函数hau_hash_sha_256描述见下表：

表 3-420. 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;
```

```
status = hau_hash_sha_256 (&input, 0x10, output[0]);
```

函数 hau_flag_get

函数hau_flag_get描述见下表：

表 3-421. 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_INPUT	输入FIFO有足够空间
HAU_FLAG_CALCULATION_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	输入FIFO非空

<i>O_EMPTY</i>	
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 3-422. 函数 `hau_flag_clear`

函数名称	<code>hau_flag_clear</code>
函数原形	<code>void hau_flag_clear(uint32_t flag);</code>
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
<code>HAU_FLAG_DATA_INP UT</code>	输入FIFO有足够空间
<code>HAU_FLAG_CALCULA TION_COMPLETE</code>	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 3-423. 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}	
interrupt	HAU标志状态
<i>HAU_INT_DATA_INPUT_T</i>	新数据块进入IN缓存区
<i>HAU_INT_CALCULATION_COMPLETE</i>	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 3-424. 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}	
interrupt	HAU标志状态
<i>HAU_INT_DATA_INPUT_T</i>	新数据块进入IN缓存区
<i>HAU_INT_CALCULATION_COMPLETE</i>	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 3-425. 函数 hau_interrupt_flag_get

函数名称	hau_interrupt_flag_get
函数原形	FlagStatus hau_interrupt_flag_get(uint32_t int_flag)
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}	
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	
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 3-426. 函数 hau_interrupt_flag_clear

函数名称	hau_interrupt_flag_clear
函数原形	void hau_interrupt_flag_clear(uint32_t int_flag)
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	
-	-

例如：

```
/* 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 3-427. 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_TIMING	时序寄存器
I2C_TIMEOUT	超时寄存器
I2C_STAT	状态寄存器
I2C_STATC	状态清除寄存器
I2C_PEC	PEC 寄存器
I2C_RDATA	接收数据寄存器
I2C_TDATA	发送数据寄存器
I2C_CTL2	控制寄存器 2

3.18.2. 外设库函数说明

I2C库函数列表如下表所示：

表 3-428. I2C 库函数

库函数名称	库函数描述
i2c_deinit	复位外设 I2C
i2c_timing_config	配置时序参数
i2c_digital_noise_filter_config	配置数字噪声过滤器
i2c_analog_noise_filter_enable	使能模拟噪声过滤器
i2c_analog_noise_filter_disable	禁能模拟噪声过滤器
i2c_master_clock_config	配置主机模式下 SCL 高低电平时钟周期

库函数名称	库函数描述
i2c_master_addressing	配置 I2C 从机地址以及数据传输方向
i2c_address10_header_enable	主机接收模式下，10 位地址头只执行读操作
i2c_address10_header_disable	主机接收模式下，10 位地址头执行完整的读操作序列
i2c_address10_enable	使能主机模式下 10 位地址寻址模式
i2c_address10_disable	禁能主机模式下 10 位地址寻址模式
i2c_automatic_end_enable	使能主机模式下 I2C 自动结束模式
i2c_automatic_end_disable	禁能主机模式下 I2C 自动结束模式
i2c_slave_response_to_gcall_enable	使能从机响应广播呼叫
i2c_slave_response_to_gcall_disable	禁能从机响应广播呼叫
i2c_stretch_scl_low_enable	当从机数据没有准备好时拉低 SCL
i2c_stretch_scl_low_disable	当从机数据没有准备好时不拉低 SCL
i2c_address_config	配置 I2C 从机地址
i2c_address_bit_compare_config	定义 ADDRESS[7:1]的哪些位和接收到的地址进行比较
i2c_address_disable	禁能从机模式下 I2C 地址
i2c_second_address_config	配置 I2C 从机第二个地址
i2c_second_address_disable	禁能 I2C 从机第二个地址
i2c_received_address_get	获取从机模式下匹配成功的地址
i2c_slave_byte_control_enable	使能从机字节控制
i2c_slave_byte_control_disable	禁能从机字节控制
i2c_nack_enable	从机模式下产生 NACK
i2c_enable	使能 I2C
i2c_disable	禁能 I2C
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_reload_enable	使能 I2C 重载模式
i2c_reload_disable	禁能 I2C 重载模式
i2c_transfer_byte_number_config	配置待发送字节数
i2c_dma_enable	使能发送/接收模式下 DMA
i2c_dma_disable	禁能发送/接收模式下 DMA
i2c_pec_transfer	I2C 传输 PEC 值
i2c_pec_enable	使能报文错误校验
i2c_pec_disable	禁能报文错误校验
i2c_pec_value_get	获取报文错误校验值
i2c_smbus_alert_enable	使能 SMBus 报警
i2c_smbus_alert_disable	禁能 SMBus 报警
i2c_smbus_default_addr_enable	使能 SMBus 设备默认地址
i2c_smbus_default_addr_disable	禁能 SMBus 设备默认地址
i2c_smbus_host_addr_enable	使能 SMBus 主机地址
i2c_smbus_host_addr_disable	禁能 SMBus 主机地址
i2c_extended_clock_timeout_enable	使能时钟信号延展超时检测

库函数名称	库函数描述
i2c_extended_clock_timeout_disable	禁能时钟信号延展超时检测
i2c_clock_timeout_enable	使能时钟超时检测
i2c_clock_timeout_disable	禁能时钟超时检测
i2c_bus_timeout_b_config	配置总线超时 B
i2c_bus_timeout_a_config	配置总线超时 A
i2c_idle_clock_timeout_config	配置空闲时钟超时检测
i2c_flag_get	获取 I2C 标志位
i2c_flag_clear	清除 I2C 标志位
i2c_interrupt_enable	中断使能
i2c_interrupt_disable	中断除能
i2c_interrupt_flag_get	获取中断标志位
i2c_interrupt_flag_clear	清除中断标志位

枚举类型 i2c_interrupt_flag_enum

表 3-429. 枚举类型 i2c_interrupt_flag_enum

枚举名称	枚举描述
I2C_INT_FLAG_TI	发送中断标志
I2C_INT_FLAG_RBNE	接收期间I2C_RDATA非空中断标志
I2C_INT_FLAG_ADDSEND	从机模式下，接收到的地址与自身地址匹配中断标志
I2C_INT_FLAG_NACK	NACK中断标志
I2C_INT_FLAG_STPDET	从机模式下检测到STOP信号中断标志
I2C_INT_FLAG_TC	主机模式下传输完成中断标志
I2C_INT_FLAG_TCR	传输完成重载中断标志
I2C_INT_FLAG_BERR	总线错误中断标志
I2C_INT_FLAG_LOSTARB	仲裁丢失中断标志
I2C_INT_FLAG_OUERR	从机模式下，过载/欠载错误中断标志
I2C_INT_FLAG_PECERR	PEC错误中断标志
I2C_INT_FLAG_TIMEOUT	超时中断标志
I2C_INT_FLAG_SMBALT	SMBus报警中断标志

函数 i2c_deinit

函数i2c_deinit描述见下表：

表 3-430. 函数 i2c_deinit

函数名称	i2c_deinit
函数原型	void i2c_deinit(uint32_t i2c_periph);
功能描述	复位外设 I2C
先决条件	-
被调用函数	rcu_periph_reset_enable / rcu_periph_reset_disable
输入参数{in}	
i2c_periph	I2C 外设

I2Cx	(x=0,1)
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* reset I2C0 */
i2c_deinit(I2C0);
```

函数 i2c_timing_config

函数i2c_timing_config描述见下表：

表 3-431. 函数 i2c_timing_config

函数名称	i2c_timing_config
函数原型	void i2c_timing_config(uint32_t i2c_periph, uint32_t psc, uint32_t scl_dely, uint32_t sda_dely);
功能描述	配置时序参数
先决条件	-
被调用函数	-
输入参数{in}	
i2c_periph	I2C 外设
I2Cx	(x=0,1)
输入参数{in}	
psc	0-0x0000000F, 时序分频
输入参数{in}	
scl_dely	0-0x0000000F, 数据建立时间
输入参数{in}	
sda_dely	0-0x0000000F, 数据保持时间
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* configure the timing parameters */
i2c_timing_config(I2C0, 0x1, 0x2, 0x1);
```

函数 i2c_digital_noise_filter_config

函数i2c_digital_noise_filter_config描述见下表：

表 3-432. 函数 i2c_digital_noise_filter_config

函数名称	i2c_digital_noise_filter_config
函数原型	void i2c_digital_noise_filter_config(uint32_t i2c_periph, uint32_t filter_length);
功能描述	配置数字噪声过滤器
先决条件	-
被调用函数	-
输入参数{in}	
i2c_periph	I2C 外设
I2Cx	(x=0,1)
输入参数{in}	
filter_length	过滤长度
FILTER_DISABLE	数字噪声过滤器禁能
FILTER_LENGTH_1	数字噪声滤波使能并且可以滤除脉宽宽度不大于 1 t _{I2CCLK} 的尖峰
FILTER_LENGTH_2	数字噪声滤波使能并且可以滤除脉宽宽度不大于 2 t _{I2CCLK} 的尖峰
FILTER_LENGTH_3	数字噪声滤波使能并且可以滤除脉宽宽度不大于 3 t _{I2CCLK} 的尖峰
FILTER_LENGTH_4	数字噪声滤波使能并且可以滤除脉宽宽度不大于 4 t _{I2CCLK} 的尖峰
FILTER_LENGTH_5	数字噪声滤波使能并且可以滤除脉宽宽度不大于 5 t _{I2CCLK} 的尖峰
FILTER_LENGTH_6	数字噪声滤波使能并且可以滤除脉宽宽度不大于 6 t _{I2CCLK} 的尖峰
FILTER_LENGTH_7	数字噪声滤波使能并且可以滤除脉宽宽度不大于 7 t _{I2CCLK} 的尖峰
FILTER_LENGTH_8	数字噪声滤波使能并且可以滤除脉宽宽度不大于 8 t _{I2CCLK} 的尖峰
FILTER_LENGTH_9	数字噪声滤波使能并且可以滤除脉宽宽度不大于 9 t _{I2CCLK} 的尖峰
FILTER_LENGTH_10	数字噪声滤波使能并且可以滤除脉宽宽度不大于 10 t _{I2CCLK} 的尖峰
FILTER_LENGTH_11	数字噪声滤波使能并且可以滤除脉宽宽度不大于 11 t _{I2CCLK} 的尖峰
FILTER_LENGTH_12	数字噪声滤波使能并且可以滤除脉宽宽度不大于 12 t _{I2CCLK} 的尖峰
FILTER_LENGTH_13	数字噪声滤波使能并且可以滤除脉宽宽度不大于 13 t _{I2CCLK} 的尖峰
FILTER_LENGTH_14	数字噪声滤波使能并且可以滤除脉宽宽度不大于 14 t _{I2CCLK} 的尖峰
FILTER_LENGTH_15	数字噪声滤波使能并且可以滤除脉宽宽度不大于 15 t _{I2CCLK} 的尖峰
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* I2C0 digital filter filters spikes with a length of up to 1 tI2CCLK */
```

```
i2c_digital_noise_filter_config(I2C0, FILTER_LENGTH_1);
```

函数 i2c_analog_noise_filter_enable

函数 i2c_analog_noise_filter_enable 描述见下表：

表 3-433. 函数 i2c_analog_noise_filter_enable

函数名称	i2c_analog_noise_filter_enable
------	--------------------------------

函数原型	void i2c_analog_noise_filter_enable(uint32_t i2c_periph);
功能描述	使能模拟噪声滤波器
先决条件	-
被调用函数	-
输入参数{in}	
i2c_periph	I2C外设
I2Cx	(x=0,1)
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* enable analog noise filter */
```

```
i2c_analog_noise_filter_enable(I2C0);
```

函数 i2c_analog_noise_filter_disable

函数i2c_analog_noise_filter_disable描述见下表:

表 3-434. 函数 i2c_analog_noise_filter_disable

函数名称	i2c_analog_noise_filter_disable
函数原型	void i2c_analog_noise_filter_disable(uint32_t i2c_periph);
功能描述	禁能模拟噪声滤波器
先决条件	-
被调用函数	-
输入参数{in}	
i2c_periph	I2C外设
I2Cx	(x=0,1)
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* disable analog noise filter */
```

```
i2c_analog_noise_filter_disable(I2C0);
```

函数 i2c_master_clock_config

函数i2c_master_clock_config描述见下表:

表 3-435. 函数 i2c_master_clock_config

函数名称	i2c_master_clock_config
------	-------------------------

函数原型	void i2c_master_clock_config(uint32_t i2c_periph, uint32_t sclh, uint32_t scll);
功能描述	配置主机模式下 SCL 高低电平周期
先决条件	-
被调用函数	-
输入参数{in}	
i2c_periph	I2C 外设
I2Cx	(x=0,1)
输入参数{in}	
sclh	0-0xff, SCL 高电平周期
输入参数{in}	
scll	0-0xff, SCL 低电平周期
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* configure the SCL and SDA period of clock in master mode */
```

```
i2c_master_clock_config(I2C0, 0x0f, 0x0f);
```

函数 i2c_master_addressing

函数i2c_master_addressing描述见下表：

表 3-436. 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 address, uint32_t trans_direction);
功能描述	配置 I2C 从机地址以及数据传输方向
先决条件	-
被调用函数	-
输入参数{in}	
i2c_periph	I2C 外设
I2Cx	(x=0,1)
输入参数{in}	
address	除保留地址外的地址，0-0x3FF，由主机发送给从机的地址
输入参数{in}	
trans_direction	主机模式下，I2C 传输方向
I2C_MASTER_TRANSMIT	主机发送
I2C_MASTER_RECEIVE	主机接收

输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* send slave address to I2C bus */
```

```
i2c_master_addressing(I2C0, 0x82, I2C_MASTER_TRANSMIT);
```

函数 i2c_address10_header_enable

函数i2c_address10_header_enable描述见下表：

表 3-437. 函数 i2c_address10_header_enable

函数名称	i2c_address10_header_enable
函数原型	void i2c_address10_header_enable(uint32_t i2c_periph);
功能描述	主机接收模式下，10 位地址头只执行读操作
先决条件	-
被调用函数	-
输入参数{in}	
i2c_periph	I2C 外设
I2Cx	(x=0,1)
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* 10-bit address header executes read direction only in master receive mode */
```

```
i2c_address10_header_enable(I2C0);
```

函数 i2c_address10_header_disable

函数i2c_address10_header_disable描述见下表：

表 3-438. 函数 i2c_address10_header_disable

函数名称	i2c_address10_header_disable
函数原型	void i2c_address10_header_disable(uint32_t i2c_periph);
功能描述	主机接收模式下，10 位地址头执行完整的读操作序列
先决条件	-
被调用函数	-
输入参数{in}	
i2c_periph	I2C 外设
I2Cx	(x=0,1)

输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* 10-bit address header executes complete sequence in master receive mode */
```

```
i2c_address10_header_disable(I2C0);
```

函数 i2c_address10_enable

函数i2c_address10_enable描述见下表：

表 3-439. 函数 i2c_address10_enable

函数名称	i2c_address10_enable
函数原型	void i2c_address10_enable(uint32_t i2c_periph);
功能描述	使能主机模式下 10 位地址寻址模式
先决条件	-
被调用函数	-
输入参数{in}	
i2c_periph	I2C 外设
I2Cx	(x=0,1)
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* enable 10-bit addressing mode in master mode */
```

```
i2c_address10_enable(I2C0);
```

函数 i2c_address10_disable

函数i2c_address10_disable描述见下表：

表 3-440. 函数 i2c_address10_disable

函数名称	i2c_address10_disable
函数原型	void i2c_address10_disable(uint32_t i2c_periph);
功能描述	禁能主机模式下 10 位地址寻址模式
先决条件	-
被调用函数	-
输入参数{in}	
i2c_periph	I2C 外设
I2Cx	(x=0,1)

输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* disable 10-bit addressing mode in master mode */
```

```
i2c_address10_disable(I2C0);
```

函数 i2c_automatic_end_enable

函数i2c_automatic_end_enable描述见下表:

表 3-441. 函数 i2c_automatic_end_enable

函数名称	i2c_automatic_end_enable
函数原型	void i2c_automatic_end_enable(uint32_t i2c_periph);
功能描述	使能主机模式下 I2C 自动结束模式
先决条件	-
被调用函数	-
输入参数{in}	
i2c_periph	I2C 外设
I2Cx	(x=0,1)
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* enable I2C automatic end mode in master mode */
```

```
i2c_automatic_end_enable(I2C0);
```

函数 i2c_automatic_end_disable

函数i2c_automatic_end_disable描述见下表:

表 3-442. 函数 i2c_automatic_end_disable

函数名称	i2c_automatic_end_disable
函数原型	void i2c_automatic_end_disable(uint32_t i2c_periph);
功能描述	禁能主机模式下 I2C 自动结束模式
先决条件	-
被调用函数	-
输入参数{in}	
i2c_periph	I2C 外设
I2Cx	(x=0,1)

输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* disable I2C automatic end mode in master mode */
```

```
i2c_automatic_end_disable(I2C0);
```

函数 i2c_slave_response_to_gcall_enable

函数i2c_slave_response_to_gcall_enable描述见下表:

表 3-443. 函数 i2c_slave_response_to_gcall_enable

函数名称	i2c_slave_response_to_gcall_enable
函数原型	void i2c_slave_response_to_gcall_enable(uint32_t i2c_periph);
功能描述	使能从机响应广播呼叫
先决条件	-
被调用函数	-
输入参数{in}	
i2c_periph	I2C 外设
I2Cx	(x=0,1)
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* enable the response to a general call */
```

```
i2c_slave_response_to_gcall_enable(I2C0);
```

函数 i2c_slave_response_to_gcall_disable

函数i2c_slave_response_to_gcall_disable描述见下表:

表 3-444. 函数 i2c_slave_response_to_gcall_disable

函数名称	i2c_slave_response_to_gcall_disable
函数原型	void i2c_slave_response_to_gcall_disable(uint32_t i2c_periph);
功能描述	禁能从机响应广播呼叫
先决条件	-
被调用函数	-
输入参数{in}	
i2c_periph	I2C 外设
I2Cx	(x=0,1)

输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* disable the response to a general call */
```

```
i2c_slave_response_to_gcall_disable(I2C0);
```

函数 i2c_stretch_scl_low_enable

函数i2c_stretch_scl_low_enable描述见下表：

表 3-445. 函数 i2c_stretch_scl_low_enable

函数名称	i2c_stretch_scl_low_enable
函数原型	void i2c_stretch_scl_low_enable(uint32_t i2c_periph);
功能描述	当从机数据没有准备好时拉低 SCL
先决条件	-
被调用函数	-
输入参数{in}	
i2c_periph	I2C 外设
I2Cx	(x=0,1)
输出参数{out}	
-	-
返回值	
-	-

例如：

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

```
i2c_stretch_scl_low_enable(I2C0);
```

函数 i2c_stretch_scl_low_disable

函数i2c_stretch_scl_low_disable描述见下表：

表 3-446. 函数 i2c_stretch_scl_low_disable

函数名称	i2c_stretch_scl_low_disable
函数原型	void i2c_stretch_scl_low_disable(uint32_t i2c_periph);
功能描述	当从机数据没有准备好时不拉低 SCL
先决条件	-
被调用函数	-
输入参数{in}	
i2c_periph	I2C 外设
I2Cx	(x=0,1)

输出参数{out}	
-	-
返回值	
-	-

例如:

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

```
i2c_stretch_scl_low_disable(I2C0);
```

函数 i2c_address_config

函数i2c_address_config描述见下表:

表 3-447. 函数 i2c_address_config

函数名称	i2c_address_config
函数原型	void i2c_address_config(uint32_t i2c_periph, uint32_t address, uint32_t addr_format);
功能描述	配置 I2C 从机地址
先决条件	-
被调用函数	-
输入参数{in}	
i2c_periph	I2C 外设
I2Cx	(x=0,1)
输入参数{in}	
address	I2C 地址
输入参数{in}	
addr_format	7 位地址或 10 位地址
I2C_ADDFORMAT_7BITS	7 位地址
I2C_ADDFORMAT_10BITS	10 位地址
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* configure i2c slave address */
```

```
i2c_address_config(I2C0, 0x82, I2C_ADDFORMAT_7BITS);
```

函数 i2c_address_bit_compare_config

函数i2c_address_bit_compare_config描述见下表:

表 3-448. 函数 i2c_address_bit_compare_config

函数名称	i2c_address_bit_compare_config
函数原型	void i2c_address_bit_compare_config(uint32_t i2c_periph, uint32_t compare_bits);
功能描述	定义 ADDRESS[7:1]的哪些位和接收到的地址进行比较
先决条件	-
被调用函数	-
输入参数{in}	
i2c_periph	I2C peripheral
I2Cx	(x=0,1)
输入参数{in}	
compare_bits	需要进行比较的位
ADDRESS_BIT1_COMPARE	地址的第 1 位需要进行比较
ADDRESS_BIT2_COMPARE	地址的第 2 位需要进行比较
ADDRESS_BIT3_COMPARE	地址的第 3 位需要进行比较
ADDRESS_BIT4_COMPARE	地址的第 4 位需要进行比较
ADDRESS_BIT5_COMPARE	地址的第 5 位需要进行比较
ADDRESS_BIT6_COMPARE	地址的第 6 位需要进行比较
ADDRESS_BIT7_COMPARE	地址的第 7 位需要进行比较
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* define which bits of ADDRESS[7:1] need to compare with the incoming address byte */
```

```
i2c_address_bit_compare_config(I2C0, ADDRESS_BIT1_COMPARE);
```

函数 i2c_address_disable

函数i2c_address_disable描述见下表:

表 3-449. 函数 i2c_address_disable

函数名称	i2c_address_disable
函数原型	void i2c_address_disable(uint32_t i2c_periph);
功能描述	禁能从机模式下 I2C 地址
先决条件	-

被调用函数	-
输入参数{in}	
i2c_periph	I2C 外设
I2Cx	(x=0,1)
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* disable i2c address in slave mode */
```

```
i2c_address_disable(I2C0);
```

函数 i2c_second_address_config

函数i2c_second_address_config描述见下表:

表 3-450. 函数 i2c_second_address_config

函数名称	i2c_second_address_config
函数原型	void i2c_second_address_config(uint32_t i2c_periph, uint32_t address, uint32_t addr_mask);
功能描述	配置 I2C 从机第二个地址
先决条件	-
被调用函数	-
输入参数{in}	
i2c_periph	I2C 外设
I2Cx	(x=0,1)
输入参数{in}	
address	I2C 地址
输入参数{in}	
addr_mask	不需要进行比较的地址
ADDRESS2_NO_MASK	无屏蔽, 全部都需要进行比较
ADDRESS2_MASK_BIT1	ADDRESS2[1]屏蔽, ADDRESS2[7:2]进行比较
ADDRESS2_MASK_BIT1_2	ADDRESS2[2:1]屏蔽, ADDRESS2[7:3]进行比较
ADDRESS2_MASK_BIT1_3	ADDRESS2[3:1]屏蔽, ADDRESS2[7:4]进行比较
ADDRESS2_MASK_BIT1_4	ADDRESS2[4:1]屏蔽, ADDRESS2[7:5]进行比较
ADDRESS2_MASK_BIT1_5	ADDRESS2[5:1]屏蔽, ADDRESS2[7:6]进行比较
ADDRESS2_MASK_BIT1_6	ADDRESS2[6:1]屏蔽, ADDRESS2[7]进行比较

T1_6	
ADDRESS2_MASK_AL L	ADDRESS2[7:1]屏蔽
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* configure i2c second slave address */
i2c_second_address_config(I2C0, 0x82, ADDRESS2_MASK_BIT1_2);
```

函数 i2c_second_address_disable

函数i2c_second_address_disable描述见下表:

表 3-451. 函数 i2c_second_address_disable

函数名称	i2c_second_address_disable
函数原型	void i2c_second_address_disable(uint32_t i2c_periph);
功能描述	禁能 I2C 从机第二个地址
先决条件	-
被调用函数	-
输入参数{in}	
i2c_periph	I2C 外设
I2Cx	(x=0,1)
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* disable i2c second address in slave mode */
i2c_second_address_disable(I2C0);
```

函数 i2c_received_address_get

函数i2c_received_address_get描述见下表:

表 3-452. 函数 i2c_received_address_get

函数名称	i2c_received_address_get
函数原型	uint32_t i2c_received_address_get(uint32_t i2c_periph);
功能描述	获取从机模式下匹配成功的地址
先决条件	-
被调用函数	-

输入参数{in}	
i2c_periph	I2C 外设
<i>I2Cx</i>	(x=0,1)
输出参数{out}	
-	-
返回值	
uint32_t	0x00000000..0x0000007F

例如:

```
/* get received match address in slave mode */
```

```
uint32_t address;
```

```
address = i2c_received_address_get(I2C0);
```

函数 i2c_slave_byte_control_enable

函数i2c_slave_byte_control_enable描述见下表:

表 3-453. 函数 i2c_slave_byte_control_enable

函数名称	i2c_slave_byte_control_enable
函数原型	void i2c_slave_byte_control_enable(uint32_t i2c_periph);
功能描述	使能从机字节控制
先决条件	-
被调用函数	-
输入参数{in}	
i2c_periph	I2C 外设
<i>I2Cx</i>	(x=0,1)
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* enable slave byte control */
```

```
i2c_slave_byte_control_enable(I2C0);
```

函数 i2c_slave_byte_control_disable

函数i2c_slave_byte_control_disable描述见下表:

表 3-454. 函数 i2c_slave_byte_control_disable

函数名称	i2c_slave_byte_control_disable
函数原型	void i2c_slave_byte_control_disable(uint32_t i2c_periph);
功能描述	禁能从机字节控制
先决条件	-

被调用函数	-
输入参数{in}	
i2c_periph	I2C 外设
I2Cx	(x=0,1)
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* disable slave byte control */
```

```
i2c_slave_byte_control_disable(I2C0);
```

函数 i2c_nack_enable

函数i2c_nack_enable描述见下表：

表 3-455. 函数 i2c_nack_enable

函数名称	i2c_nack_enable
函数原型	void i2c_nack_enable(uint32_t i2c_periph);
功能描述	从机模式下产生 NACK
先决条件	-
被调用函数	-
输入参数{in}	
i2c_periph	I2C 外设
I2Cx	(x=0,1)
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* generate a NACK in slave mode */
```

```
i2c_nack_enable(I2C0);
```

函数 i2c_enable

函数i2c_enable描述见下表：

表 3-456. 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
先决条件	-

被调用函数	-
输入参数{in}	
i2c_periph	I2C 外设
I2Cx	(x=0,1)
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* enable I2C0 */
i2c_enable(I2C0);
```

函数 i2c_disable

函数i2c_disable描述见下表：

表 3-457. 函数 i2c_disable

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

例如：

```
/* disable I2C0 */
i2c_disable(I2C0);
```

函数 i2c_start_on_bus

函数i2c_start_on_bus描述见下表：

表 3-458. 函数 i2c_start_on_bus

函数名称	i2c_start_on_bus
函数原型	void i2c_start_on_bus(uint32_t i2c_periph);
功能描述	在 I2C 总线上生成起始位
先决条件	-

被调用函数	-
输入参数{in}	
i2c_periph	I2C 外设
I2Cx	(x=0,1)
输出参数{out}	
-	-
返回值	
-	-

例如：

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

```
i2c_start_on_bus(I2C0);
```

函数 i2c_stop_on_bus

函数i2c_stop_on_bus描述见下表：

表 3-459. 函数 i2c_stop_on_bus

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

例如：

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

```
i2c_stop_on_bus(I2C0);
```

函数 i2c_data_transmit

函数i2c_data_transmit描述见下表：

表 3-460. 函数 i2c_data_transmit

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

被调用函数	-
输入参数{in}	
i2c_periph	I2C 外设
I2Cx	(x=0,1)
输入参数{in}	
data	transmit data
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* I2C0 transmit data */
```

```
i2c_data_transmit(I2C0, 0x80);
```

函数 i2c_data_receive

函数i2c_data_receive描述见下表:

表 3-461. 函数 i2c_data_receive

函数名称	i2c_data_receive
函数原型	uint32_t i2c_data_receive(uint32_t i2c_periph);
功能描述	接收数据
先决条件	-
被调用函数	-
输入参数{in}	
i2c_periph	I2C 外设
I2Cx	(x=0,1)
输出参数{out}	
-	-
返回值	
uint32_t	0x00000000..0x000000FF

例如:

```
/* I2C0 receive data */
```

```
uint32_t i2c_receiver;
```

```
i2c_receiver = i2c_data_receive(I2C0);
```

函数 i2c_reload_enable

函数i2c_reload_enable描述见下表:

表 3-462. 函数 i2c_reload_enable

函数名称	i2c_reload_enable
------	-------------------

函数原型	void i2c_reload_enable(uint32_t i2c_periph);
功能描述	使能 I2C 重载模式
先决条件	-
被调用函数	-
输入参数{in}	
i2c_periph	I2C 外设
I2Cx	(x=0,1)
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* enable I2C reload mode */
```

```
i2c_reload_enable(I2C0);
```

函数 i2c_reload_disable

函数i2c_reload_disable描述见下表：

表 3-463. 函数 i2c_reload_disable

函数名称	i2c_reload_disable
函数原型	void i2c_reload_disable(uint32_t i2c_periph);
功能描述	禁能 I2C 重载模式
先决条件	-
被调用函数	-
输入参数{in}	
i2c_periph	I2C 外设
I2Cx	(x=0,1)
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* disable I2C reload mode */
```

```
i2c_reload_disable(I2C0);
```

函数 i2c_transfer_byte_number_config

函数i2c_transfer_byte_number_config描述见下表：

表 3-464. 函数 i2c_transfer_byte_number_config

函数名称	i2c_transfer_byte_number_config
------	---------------------------------

函数原型	void i2c_transfer_byte_number_config(uint32_t i2c_periph, uint32_t byte_number);
功能描述	配置待发送字节数
先决条件	-
被调用函数	-
输入参数{in}	
i2c_periph	I2C 外设
I2Cx	(x=0,1)
输入参数{in}	
byte_number	0x0-0xFF, 待传输的字节数
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* configure number of bytes to be transferred */
```

```
i2c_transfer_byte_number_config(I2C0, 0xFF);
```

函数 i2c_dma_enable

函数i2c_dma_enable描述见下表:

表 3-465. 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 dma);
功能描述	使能发送/接收模式下 DMA
先决条件	-
被调用函数	-
输入参数{in}	
i2c_periph	I2C 外设
I2Cx	(x=0,1)
输入参数{in}	
dma	I2C DMA
I2C_DMA_TRANSMIT	采用 DMA 方式发送数据
I2C_DMA_RECEIVE	采用 DMA 方式接收数据
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* enable I2C DMA for transmission or reception */
```



```
i2c_dma_enable(I2C0, I2C_DMA_RECEIVE);
```

函数 i2c_dma_disable

函数i2c_dma_disable描述见下表:

表 3-466. 函数 i2c_dma_disable

函数名称	i2c_dma_disable
函数原型	void i2c_dma_disable(uint32_t i2c_periph, uint32_t dma);
功能描述	禁用发送/接收模式下 DMA
先决条件	-
被调用函数	-
输入参数{in}	
i2c_periph	I2C 外设
I2Cx	(x=0,1)
输入参数{in}	
dma	I2C DMA
I2C_DMA_TRANSMIT	采用 DMA 方式发送数据
I2C_DMA_RECEIVE	采用 DMA 方式接收数据
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* disable I2C DMA for transmission or reception */
```

```
i2c_dma_disable(I2C0, I2C_DMA_RECEIVE);
```

函数 i2c_pec_transfer

函数i2c_pec_transfer描述见下表:

表 3-467. 函数 i2c_pec_transfer

函数名称	i2c_pec_transfer
函数原型	void i2c_pec_transfer(uint32_t i2c_periph);
功能描述	I2C 传输 PEC 值
先决条件	-
被调用函数	-
输入参数{in}	
i2c_periph	I2C 外设
I2Cx	(x=0,1)
输出参数{out}	
-	-
返回值	

-	-
---	---

例如:

```
/* I2C transfers PEC value */
```

```
i2c_pec_transfer(I2C0);
```

函数 i2c_pec_enable

函数i2c_pec_enable描述见下表:

表 3-468. 函数 i2c_pec_enable

函数名称	i2c_pec_enable
函数原型	void i2c_pec_enable(uint32_t i2c_periph);
功能描述	使能报文错误校验
先决条件	-
被调用函数	-
输入参数{in}	
i2c_periph	I2C 外设
I2Cx	(x=0,1)
输出参数{out}	
-	-
返回值	
-	-

例如:

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

```
i2c_pec_enable(I2C0);
```

函数 i2c_pec_disable

函数i2c_pec_disable描述见下表:

表 3-469. 函数 i2c_pec_disable

函数名称	i2c_pec_disable
函数原型	void i2c_pec_disable(uint32_t i2c_periph);
功能描述	禁用报文错误校验
先决条件	-
被调用函数	-
输入参数{in}	
i2c_periph	I2C 外设
I2Cx	(x=0,1)
输出参数{out}	
-	-
返回值	

-	-
---	---

例如:

```
/* disable I2C PEC calculation */
```

```
i2c_pec_disable(I2C0);
```

函数 i2c_pec_value_get

函数i2c_pec_value_get描述见下表:

表 3-470. 函数 i2c_pec_value_get

函数名称	i2c_pec_value_get
函数原型	uint32_t i2c_pec_value_get(uint32_t i2c_periph);
功能描述	获取报文错误校验值
先决条件	-
被调用函数	-
输入参数{in}	
i2c_periph	I2C 外设
I2Cx	(x=0,1)
输出参数{out}	
-	-
返回值	
uint32_t	PEC 值

例如:

```
/* I2C0 get packet error checking value */
```

```
uint32_t pec_value;
```

```
pec_value = i2c_pec_value_get(I2C0);
```

函数 i2c_smbus_alert_enable

函数i2c_smbus_alert_enable描述见下表:

表 3-471. 函数 i2c_smbus_alert_enable

函数名称	i2c_smbus_alert_enable
函数原型	void i2c_smbus_alert_enable(uint32_t i2c_periph);
功能描述	使能 SMBus 报警
先决条件	-
被调用函数	-
输入参数{in}	
i2c_periph	I2C 外设
I2Cx	(x=0,1)
输出参数{out}	
-	-

返回值	
-	-

例如：

```
/* enable SMBus Alert */
```

```
i2c_smbus_alert_enable(I2C0);
```

函数 i2c_smbus_alert_disable

函数i2c_smbus_alert_disable描述见下表：

表 3-472. 函数 i2c_smbus_alert_disable

函数名称	i2c_smbus_alert_disable
函数原型	void i2c_smbus_alert_disable(uint32_t i2c_periph);
功能描述	禁能 SMBus 报警
先决条件	-
被调用函数	-
输入参数{in}	
i2c_periph	I2C 外设
I2Cx	(x=0,1)
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* disable SMBus Alert */
```

```
i2c_smbus_alert_disable(I2C0);
```

函数 i2c_smbus_default_addr_enable

函数i2c_smbus_default_addr_enable描述见下表：

表 3-473. 函数 i2c_smbus_default_addr_enable

函数名称	i2c_smbus_default_addr_enable
函数原型	void i2c_smbus_default_addr_enable(uint32_t i2c_periph);
功能描述	使能 SMBus 设备默认地址
先决条件	-
被调用函数	-
输入参数{in}	
i2c_periph	I2C 外设
I2Cx	(x=0,1)
输出参数{out}	
-	-

返回值	
-	-

例如：

```
/* enable SMBus device default address */
```

```
i2c_smbus_default_addr_enable(I2C0);
```

函数 i2c_smbus_default_addr_disable

函数i2c_smbus_default_addr_disable描述见下表：

表 3-474. 函数 i2c_smbus_default_addr_disable

函数名称	i2c_smbus_default_addr_disable
函数原型	void i2c_smbus_default_addr_disable(uint32_t i2c_periph);
功能描述	禁能 SMBus 设备默认地址
先决条件	-
被调用函数	-
输入参数{in}	
i2c_periph	I2C 外设
I2Cx	(x=0,1)
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* disable SMBus device default address */
```

```
i2c_smbus_default_addr_disable(I2C0);
```

函数 i2c_smbus_host_addr_enable

函数i2c_smbus_host_addr_enable描述见下表：

表 3-475. 函数 i2c_smbus_host_addr_enable

函数名称	i2c_smbus_host_addr_enable
函数原型	void i2c_smbus_host_addr_enable(uint32_t i2c_periph);
功能描述	使能 SMBus 主机地址
先决条件	-
被调用函数	-
输入参数{in}	
i2c_periph	I2C 外设
I2Cx	(x=0,1)
输出参数{out}	
-	-

返回值	
-	-

例如：

```
/* enable SMBus Host address */
```

```
i2c_smbus_host_addr_enable(I2C0);
```

函数 i2c_smbus_host_addr_disable

函数i2c_smbus_host_addr_disable描述见下表：

表 3-476. 函数 i2c_smbus_host_addr_disable

函数名称	i2c_smbus_host_addr_disable
函数原型	void i2c_smbus_host_addr_disable(uint32_t i2c_periph);
功能描述	禁能 SMBus 主机地址
先决条件	-
被调用函数	-
输入参数{in}	
i2c_periph	I2C 外设
I2Cx	(x=0,1)
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* disable SMBus Host address */
```

```
i2c_smbus_host_addr_disable(I2C0);
```

函数 i2c_extended_clock_timeout_enable

函数i2c_extended_clock_timeout_enable描述见下表：

表 3-477. 函数 i2c_extended_clock_timeout_enable

函数名称	i2c_extended_clock_timeout_enable
函数原型	void i2c_extended_clock_timeout_enable(uint32_t i2c_periph);
功能描述	使能时钟信号延展超时检测
先决条件	-
被调用函数	-
输入参数{in}	
i2c_periph	I2C 外设
I2Cx	(x=0,1)
输出参数{out}	
-	-

返回值	
-	-

例如：

```
/* enable extended clock timeout detection */
```

```
i2c_extended_clock_timeout_enable(I2C0);
```

函数 i2c_extended_clock_timeout_disable

函数i2c_extended_clock_timeout_disable描述见下表：

表 3-478. 函数 i2c_extended_clock_timeout_disable

函数名称	i2c_extended_clock_timeout_disable
函数原型	void i2c_extended_clock_timeout_disable(uint32_t i2c_periph);
功能描述	禁能扩展时钟超时检测
先决条件	-
被调用函数	-
输入参数{in}	
i2c_periph	I2C 外设
I2Cx	(x=0,1)
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* disable extended clock timeout detection */
```

```
i2c_extended_clock_timeout_disable(I2C0);
```

函数 i2c_clock_timeout_enable

函数i2c_clock_timeout_enable描述见下表：

表 3-479. 函数 i2c_clock_timeout_enable

函数名称	i2c_clock_timeout_enable
函数原型	void i2c_clock_timeout_enable(uint32_t i2c_periph);
功能描述	禁能时钟信号延展超时检测
先决条件	-
被调用函数	-
输入参数{in}	
i2c_periph	I2C 外设
I2Cx	(x=0,1)
输出参数{out}	
-	-

返回值	
-	-

例如：

```
/* enable clock timeout detection */
```

```
i2c_clock_timeout_enable(I2C0);
```

函数 i2c_clock_timeout_disable

函数i2c_clock_timeout_disable描述见下表：

表 3-480. 函数 i2c_clock_timeout_disable

函数名称	i2c_clock_timeout_disable
函数原型	void i2c_clock_timeout_disable(uint32_t i2c_periph);
功能描述	禁用时钟超时检测
先决条件	-
被调用函数	-
输入参数{in}	
i2c_periph	I2C 外设
I2Cx	(x=0,1)
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* disable clock timeout detection */
```

```
i2c_clock_timeout_disable(I2C0);
```

函数 i2c_bus_timeout_b_config

函数i2c_bus_timeout_b_config描述见下表：

表 3-481. 函数 i2c_bus_timeout_b_config

函数名称	i2c_bus_timeout_b_config
函数原型	void i2c_bus_timeout_b_config(uint32_t i2c_periph, uint32_t timeout);
功能描述	配置总线超时 B
先决条件	-
被调用函数	-
输入参数{in}	
i2c_periph	I2C 外设
I2Cx	(x=0,1)
输入参数{in}	
timeout	0-0xffff, 总线超时 B

输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* configure bus timeout B */
i2c_bus_timeout_b_config(I2C0, 0xff);
```

函数 i2c_bus_timeout_a_config

函数i2c_bus_timeout_a_config描述见下表：

表 3-482. 函数 i2c_bus_timeout_a_config

函数名称	i2c_bus_timeout_a_config
函数原型	void i2c_bus_timeout_a_config(uint32_t i2c_periph, uint32_t timeout);
功能描述	配置总线超时 A
先决条件	-
被调用函数	-
输入参数{in}	
i2c_periph	I2C 外设
I2Cx	(x=0,1)
输入参数{in}	
timeout	0-0xffff, 总线超时 A
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* configure bus timeout A */
i2c_bus_timeout_a_config(I2C0, 0xff);
```

函数 i2c_idle_clock_timeout_config

函数i2c_idle_clock_timeout_config描述见下表：

表 3-483. 函数 i2c_idle_clock_timeout_config

函数名称	i2c_idle_clock_timeout_config
函数原型	void i2c_idle_clock_timeout_config(uint32_t i2c_periph, uint32_t timeout);
功能描述	配置空闲时钟超时检测
先决条件	-
被调用函数	-
输入参数{in}	

i2c_periph	I2C 外设
<i>I2Cx</i>	(x=0,1)
输入参数{in}	
timeout	总线超时 A
<i>BUSTOA_DETECT_SCL_LOW</i>	BUSTOA 用于检测 SCL 低电平超时
<i>BUSTOA_DETECT_IDLE</i>	BUSTOA 用于检测总线空闲情况下 SCL 和 SDA 高电平超时
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* configure idle clock timeout detection */
```

```
i2c_idle_clock_timeout_config(I2C0, BUSTOA_DETECT_SCL_LOW);
```

函数 i2c_flag_get

函数i2c_flag_get描述见下表：

表 3-484. 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	获取 I2C 标志位
先决条件	-
被调用函数	-
输入参数{in}	
i2c_periph	I2C 外设
<i>I2Cx</i>	(x=0,1)
输入参数{in}	
flag	I2C 标志位
<i>I2C_FLAG_TBE</i>	发送期间 I2C_TDATA 寄存器空标志
<i>I2C_FLAG_TI</i>	发送中断标志
<i>I2C_FLAG_RBNE</i>	接收期间 I2C_RDATA 非空标志
<i>I2C_FLAG_ADDSEND</i>	从机模式下，接收到的地址与自身地址匹配
<i>I2C_FLAG_NACK</i>	NACK 标志
<i>I2C_FLAG_STPDET</i>	从机模式下检测到 STOP 信号
<i>I2C_FLAG_TC</i>	主机模式下传输完成标志
<i>I2C_FLAG_TCR</i>	传输完成重载标志
<i>I2C_FLAG_BERR</i>	总线错误标志
<i>I2C_FLAG_LOSTARB</i>	仲裁丢失标志
<i>I2C_FLAG_OUERR</i>	从机模式下，过载/欠载错误标志

<i>I2C_FLAG_PECERR</i>	PEC 错误标志
<i>I2C_FLAG_TIMEOUT</i>	超时标志
<i>I2C_FLAG_SMBALT</i>	SMBus 报警标志
<i>I2C_FLAG_I2CBSY</i>	忙标志
<i>I2C_FLAG_TR</i>	从机模式下，I2C 作为发送器还是接收器标志
输出参数{out}	
-	-
返回值	
FlagStatus	SET / RESET

例如：

```
/* get I2C flag status */
```

```
FlagStatus flag_state = RESET;
```

```
flag_state = i2c_flag_get(I2C0, I2C_FLAG_TBE);
```

函数 **i2c_flag_clear**

函数*i2c_flag_clear*描述见下表：

表 3-485. 函数 *i2c_flag_clear*

函数名称	<i>i2c_flag_clear</i>
函数原型	<code>void i2c_flag_clear(uint32_t i2c_periph, uint32_t flag);</code>
功能描述	清除 I2C 标志位
先决条件	-
被调用函数	-
输入参数{in}	
i2c_periph	I2C 外设
<i>I2Cx</i>	(x=0,1)
输入参数{in}	
flag	I2C 标志位
<i>I2C_FLAG_ADDSEND</i>	从机模式下，接收到的地址与自身地址匹配
<i>I2C_FLAG_NACK</i>	NACK 标志
<i>I2C_FLAG_STPDET</i>	从机模式下检测到 STOP 信号
<i>I2C_FLAG_BERR</i>	总线错误标志
<i>I2C_FLAG_LOSTARB</i>	仲裁丢失标志
<i>I2C_FLAG_OUERR</i>	从机模式下，过载/欠载错误标志
<i>I2C_FLAG_PECERR</i>	PEC 错误标志
<i>I2C_FLAG_TIMEOUT</i>	超时标志
<i>I2C_FLAG_SMBALT</i>	SMBus 报警标志
输出参数{out}	
-	-
返回值	
-	-

例如：

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

函数 i2c_interrupt_enable

函数i2c_interrupt_enable描述见下表：

表 3-486. 函数 i2c_interrupt_enable

函数名称	i2c_interrupt_enable
函数原型	void i2c_interrupt_enable(uint32_t i2c_periph, uint32_t interrupt);
功能描述	中断使能
先决条件	-
被调用函数	-
输入参数{in}	
i2c_periph	I2C 外设
<i>I2Cx</i>	(x=0,1)
输入参数{in}	
interrupt	I2C 中断
<i>I2C_INT_ERR</i>	错误中断
<i>I2C_INT_TC</i>	发送完成中断
<i>I2C_INT_STPDET</i>	检测到 STOP 中断
<i>I2C_INT_NACK</i>	接收到 NACK 中断
<i>I2C_INT_ADDM</i>	地址匹配中断
<i>I2C_INT_RBNE</i>	接收中断
<i>I2C_INT_TI</i>	发送中断
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* enable I2C0 transmit interrupt */
i2c_interrupt_enable(I2C0, I2C_INT_TI);
```

函数 i2c_interrupt_disable

函数i2c_interrupt_disable描述见下表：

表 3-487. 函数 i2c_interrupt_disable

函数名称	i2c_interrupt_disable
函数原型	void i2c_interrupt_disable(uint32_t i2c_periph, uint32_t interrupt);
功能描述	中断除能
先决条件	-

被调用函数	-
输入参数{in}	
i2c_periph	I2C 外设
I2Cx	(x=0,1)
输入参数{in}	
interrupt	I2C 中断
I2C_INT_ERR	错误中断
I2C_INT_TC	发送完成中断
I2C_INT_STPDET	检测到 STOP 中断
I2C_INT_NACK	接收到 NACK 中断
I2C_INT_ADDM	地址匹配中断
I2C_INT_RBNE	接收中断
I2C_INT_TI	发送中断
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* disable I2C0 transmit interrupt */
```

```
i2c_interrupt_disable(I2C0, I2C_INT_TI);
```

函数 i2c_interrupt_flag_get

函数 i2c_interrupt_flag_get 描述见下表：

表 3-488. 函数 i2c_interrupt_flag_get

函数名称	i2c_interrupt_flag_get
函数原型	FlagStatus i2c_interrupt_flag_get(uint32_t i2c_periph, i2c_interrupt_flag_enum int_flag);
功能描述	获取中断标志位
先决条件	-
被调用函数	-
输入参数{in}	
i2c_periph	I2C 外设
I2Cx	(x=0,1)
输入参数{in}	
int_flag	I2C 中断标志位，参考 表 3-429. 枚举类型 i2c_interrupt_flag_enum 。
I2C_INT_FLAG_TI	发送中断标志
I2C_INT_FLAG_RBNE	接收期间 I2C_RDATA 非空中断标志
I2C_INT_FLAG_ADDS END	从机模式下，接收到的地址与自身地址匹配中断标志
I2C_INT_FLAG_NACK	NACK 中断标志

<i>I2C_INT_FLAG_STPD ET</i>	从机模式下检测到 STOP 信号中断标志
<i>I2C_INT_FLAG_TC</i>	主机模式下传输完成中断标志
<i>I2C_INT_FLAG_TCR</i>	传输完成重载中断标志
<i>I2C_INT_FLAG_BERR</i>	总线错误中断标志
<i>I2C_INT_FLAG_LOSTA RB</i>	仲裁丢失中断标志
<i>I2C_INT_FLAG_OUER R</i>	从机模式下，过载/欠载错误中断标志
<i>I2C_INT_FLAG_PEC RR</i>	PEC 错误中断标志
<i>I2C_INT_FLAG_TIMEO UT</i>	超时中断标志
<i>I2C_INT_FLAG_SMBA LT</i>	SMBus 报警中断标志
输出参数{out}	
-	-
返回值	
FlagStatus	SET / RESET

例如：

```
/* get I2C interrupt flag status */
```

```
FlagStatus flag_state = RESET;
```

```
flag_state = i2c_interrupt_flag_get(I2C0, I2C_INT_FLAG_TI);
```

函数 i2c_interrupt_flag_clear

函数 i2c_interrupt_flag_clear 描述见下表：

表 3-489. 函数 i2c_interrupt_flag_clear

函数名称	i2c_interrupt_flag_clear
函数原型	void i2c_interrupt_flag_clear(uint32_t i2c_periph, i2c_interrupt_flag_enum int_flag);
功能描述	清除中断标志位
先决条件	-
被调用函数	-
输入参数{in}	
i2c_periph	I2C 外设
I2Cx	(x=0,1)
输入参数{in}	
int_flag	I2C 中断标志位，参考 表 3-429. 枚举类型 i2c_interrupt_flag_enum 。
<i>I2C_INT_FLAG_ADDS END</i>	从机模式下，接收到的地址与自身地址匹配中断标志

<i>I2C_INT_FLAG_NACK</i>	NACK 中断标志
<i>I2C_INT_FLAG_STPDET</i>	从机模式下检测到 STOP 信号中断标志
<i>I2C_INT_FLAG_BERR</i>	总线错误中断标志
<i>I2C_INT_FLAG_LOSTARB</i>	仲裁丢失中断标志
<i>I2C_INT_FLAG_OVERR</i>	从机模式下，过载/欠载错误中断标志
<i>I2C_INT_FLAG_PECERR</i>	PEC 错误中断标志
<i>I2C_INT_FLAG_TIMEOUT</i>	超时中断标志
<i>I2C_INT_FLAG_SMBAL</i>	SMBus 报警中断标志
输出参数{out}	
-	-
返回值	
-	-

例如：

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

```
i2c_interrupt_flag_clear(I2C0, I2C_INT_FLAG_BERR);
```

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 3-490. NVIC 寄存器

寄存器名称	寄存器描述
ISER ⁽¹⁾	中断使能寄存器
ICER ⁽¹⁾	中断禁能寄存器
ISPR ⁽¹⁾	中断挂起寄存器
ICPR ⁽¹⁾	中断清除寄存器
IABR ⁽¹⁾	中断活动状态寄存器
IP ⁽¹⁾	中断优先级寄存器
STIR ⁽¹⁾	软触发中断寄存器
CPUID ⁽²⁾	CPUID 寄存器
ICSR ⁽²⁾	中断控制及状态寄存器
VTOR ⁽²⁾	向量表偏移量寄存器
AIRCR ⁽²⁾	应用程序中断及复位控制寄存器
SCR ⁽²⁾	系统控制寄存器
CCR ⁽²⁾	配置与控制寄存器
SHP ⁽²⁾	系统异常优先级寄存器
SHCSR ⁽²⁾	系统异常控制及状态寄存器
CFSR ⁽²⁾	配置错误状态寄存器
HFSR ⁽²⁾	硬错误状态寄存器
DFSR ⁽²⁾	调试错误状态寄存器
MMFAR ⁽²⁾	存储管理错误地址寄存器
BFAR ⁽²⁾	总线错误地址寄存器
AFSR ⁽²⁾	辅助错误地址寄存器
PFR ⁽²⁾	处理器特性寄存器
DFR ⁽²⁾	调试特性寄存器
ADR ⁽²⁾	辅助特性寄存器
MMFR ⁽²⁾	存储模型特性寄存器
ISAR ⁽²⁾	指令设置属性寄存器
CPACR ⁽²⁾	协处理器访问控制寄存器

1. 参考 core_cm33.h 文件中定义的结构体类型 NVIC_Type

2. 参考 core_cm33.h 文件中定义的结构体类型 SCB_Type

表 3-491. SysTick 寄存器

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

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

3.19.2. 外设库函数说明

枚举类型 IRQn_Type

表 3-492. 枚举类型 IRQn_Type

成员名称	功能描述
WWDGT_IRQn	窗口看门狗中断
LVD_IRQn	连接到 EXTI 线的 LVD 中断
TAMPER_IRQn	侵入检测中断
RTC_IRQn	RTC 全局中断
FMC_IRQn	FMC 全局中断
RCU_CTC_IRQn	RCU 和 CTC 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_IRQn	TIMER0 中止中断
TIMER0_UP_IRQn	TIMER0 更新中断
TIMER0_TRG_CMT_IRQn	TIMER0 触发与通道换相中断
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	USBFS 唤醒中断
TIMER7_BRK_IRQn	TIMER7 中止中断
TIMER7_UP_IRQn	TIMER7 更新中断
TIMER7_TRG_CMT_IRQn	TIMER7 触发与通道换相中断
TIMER7_Channel_IRQn	TIMER7 通道捕获比较中断
ADC2_IRQn	ADC2 全局中断
RCU_CKFM_IRQn	RCU 时钟时钟频率监视器中断
CMP_WAKEUP_IRQn	连接 EXTI 线的 CMP 唤醒中断
SPI2_IRQn	SPI2 全局中断
UART3_IRQn	UART3 全局中断
UART4_IRQn	UART4 全局中断
TIMER5_IRQn	TIMER5 全局中断
TIMER6_IRQn	TIMER6 全局中断
DMA1_Channel0_IRQn	DMA1 通道 0 全局中断
DMA1_Channel1_IRQn	DMA1 通道 1 全局中断
DMA1_Channel2_IRQn	DMA1 通道 2 全局中断
DMA1_Channel3_IRQn	DMA1 通道 3 全局中断
DMA1_Channel4_IRQn	DMA1 通道 4 全局中断
DAC_IRQn	DAC 全局中断
PMU_UVD_OVD_IRQn	VUVD/VOVD 中断
CAN1_TX_IRQn	CAN1 发送中断
CAN1_RX0_IRQn	CAN1 接收 0 中断
CAN1_RX1_IRQn	CAN1 接收 1 中断
CAN1_EWMC_IRQn	CAN1 EWMC 中断
SYSTEM_IRQn	系统中断
FPU_IRQn	FPU 中断
CMP_IRQn	CMP 中断
DMAMUX_IRQn	DMAMUX 中断
CAU_IRQn	CAU 中断
HAU_IRQn	HAU 中断

TRNG_IRQn	TRNG 中断
USBFS_IRQn	USBFS 中断
TIMER4_IRQn	TIMER4 中断
TIMER15_IRQn	TIMER15 中断
TIMER16_IRQn	TIMER16 中断
TIMER0_BRK_Channel_IRQn	TIMER4 中止中断
TIMER7_BRK_Channel_IRQn	TIMER4 中止中断

MISC库函数列表如下表所示：

表 3-493. 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	禁能NVIC的中断
nvic_system_reset	请求系统复位
nvic_vector_table_set	设置向量表地址
system_lowpower_set	设置系统低功耗模式状态
system_lowpower_reset	复位系统低功耗模式状态
systick_clksource_set	设置系统定时器时钟源

函数 nvic_priority_group_set

函数nvic_priority_group_set描述见下表：

表 3-494. 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 3-495. 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 3-492. 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 3-496. 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 3-492. 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_system_reset

函数nvic_system_reset描述见下表：

表 3-497. 函数 nvic_system_reset

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

例如：

```
/* request system reset */
```

```
nvic_system_reset ();
```

函数 nvic_vector_table_set

函数nvic_vector_table_set描述见下表：

表 3-498. 函数 nvic_vector_table_set

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

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

例如：

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

```
nvic_vector_table_set (NVIC_VECTTAB_FLASH, 0x200);
```

函数 **system_lowpower_set**

函数system_lowpower_set描述见下表：

表 3-499. 函数 system_lowpower_set

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

例如：

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

```
system_lowpower_set (SCB_LPM_SLEEP_EXIT_ISR);
```

函数 system_lowpower_reset

函数system_lowpower_reset描述见下表：

表 3-500. 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	系统低功耗模式状态的管理
先决条件	-
被调用函数	-
输入参数{in}	
lowpower_mode	系统低功耗模式的状态
SCB_LPM_SLEEP_EXIT_ISR	该位为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_WAKEUP_BY_ALL_INT	该位为0时，系统只能被使能的中断唤醒
输出参数{out}	
-	-
返回值	
-	-

例如：

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

```
system_lowpower_reset (SCB_LPM_SLEEP_EXIT_ISR);
```

函数 systick_clksource_set

函数systick_clksource_set描述见下表：

表 3-501. 函数 systick_clksource_set

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

-	-
返回值	
-	-

例如:

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

```
systick_clksource_set (SYSTICK_CLKSOURCE_HCLK_DIV8);
```

3.20. 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 3-502. PMU 寄存器

寄存器名称	寄存器描述
PMU_CTL0	控制寄存器0
PMU_CS	电源控制和状态寄存器
PMU_CTL1	控制寄存器1

3.20.2. 外设库函数说明

PMU 库函数列表如下表所示:

表 3-503. PMU 库函数

库函数名称	库函数描述
pmu_deinit	复位外设PMU
Pmu_lock	锁定寄存器写
Pmu_unlock	解锁寄存器写
pmu_lvd_select	选择低压检测阈值
Pmu_pdrvs_select	复位电压阈值检测
pmu_ldo_output_select	LDO输出电压选择
pmu_lvd_disable	关闭低压检测器
pmu_lowdriver_mode_enable	深度睡眠模式下使能低驱动模式

库函数名称	库函数描述
pmu_lowdriver_mode_disable	深度睡眠模式下失能低驱动模式
pmu_lowpower_driver_config	使用低功耗LDO时驱动模式选择
pmu_normalpower_driver_config	使用正常LDO时驱动模式选择
pmu_to_sleepmode	进入睡眠模式
pmu_to_deepsleepmode	进入深度睡眠模式
pmu_to_standbymode	进入待机模式
pmu_wakeup_pin_enable	WKUP引脚唤醒使能
pmu_wakeup_pin_disable	WKUP引脚唤醒失能
pmu_backup_write_enable	备份域写使能
pmu_backup_write_disable	备份域写失能
pmu_vavd_disable	Vavd禁能
pmu_vavd_enable	Vavd使能
pmu_vovd_disable	Vovd禁能
pmu_vovd_enable	Vovd使能
pmu_vuvd_disable	Vuvd禁能
pmu_vuvd_enable	Vuvd使能
pmu_vavd_select	Vavd阈值选择
pmu_vovd_select	Vode阈值选择
pmu_vuvd_select	Vuvd阈值选择
pmu_vuvdo_dnf_select	VUVD 模拟输出数字噪声滤波器峰值长度选择
pmu_vovdo_dnf_select	VOVD模拟输出数字噪声滤波器峰值长度选择
pmu_flag_get	获取标志位
pmu_flag_clear	清除标志位

函数 pmu_deinit

函数pmu_deinit描述见下表：

表 3-504. 函数 pmu_deinit

函数名称	pmu_deinit
------	------------

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

例如:

```
/* reset PMU */
```

```
pmu_deinit ();
```

函数 pmu_lock

The description of pmu_highdriver_mode_enable is shown as below:

Table 3-1. Function pmu_lock

函数名称	pmu_lock
函数原型	void pmu_lock(void)
功能描述	lock the pmu register
先决条件	-
被调用函数	-
Input parameter{in}	
-	-
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* lock the pmu register */
```

```
pmu_lock ();
```

函数 pmu_unlock

The description of pmu_highdriver_mode_enable is shown as below:

Table 3-2. Function pmu_unlock

函数名称	pmu_unlock
函数原型	void pmu_unlock(void)
功能描述	unlock the pmu register
先决条件	-
被调用函数	-
Input parameter{in}	
-	-
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* unlock the pmu register */
```

```
pmu_unlock ();
```

函数 pmu_lvd_select

函数pmu_lvd_select描述见下表:

表 3-505. 函数 pmu_lvd_select

函数名称	pmu_lvd_select
函数原型	void pmu_lvd_select(uint32_t lvdtn);
功能描述	选择低压检测阈值
先决条件	-
被调用函数	-
输入参数{in}	

lvdt_n	电压阈值
<i>PMU_LVDT_0</i>	电压阈值为2.2V
<i>PMU_LVDT_1</i>	电压阈值为2.3V
<i>PMU_LVDT_2</i>	电压阈值为2.4V
<i>PMU_LVDT_3</i>	电压阈值为2.5V
<i>PMU_LVDT_4</i>	电压阈值为2.6V
<i>PMU_LVDT_5</i>	电压阈值为2.7V
<i>PMU_LVDT_6</i>	电压阈值为2.8V
<i>PMU_LVDT_7</i>	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_pdrvs_select

函数pmu_pdrvs_select描述见下表:

表 3-506. 函数 pmu_pdrvs_select

函数名称	pmu_pdrvs_select
函数原型	void pmu_pdrvs_select(uint32_t lvdt_n);
功能描述	选择复位压检测阈值
先决条件	-
被调用函数	-
输入参数{in}	
lvdt_n	电压阈值
<i>PMU_PDRVS_LEV_ELO</i>	电压阈值为2.35V
<i>PMU_PDRVS_LEV</i>	电压阈值为1.8V

EL1	
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* select low voltage detector threshold as 2.35V */
```

```
pmu_pdrvs_select (PMU_PDRVS_LEVEL0);
```

函数 pmu_ldo_output_select

函数pmu_ldo_output_select描述见下表：

表 3-507. 函数 pmu_ldo_output_select

函数名称	pmu_ldo_output_select
函数原型	void pmu_ldo_output_select(uint32_t ldo_output);
功能描述	内部电压调节器（LDO）输出电压选择
先决条件	-
被调用函数	-
输入参数{in}	
ldo_output	LDO输出电压等级
PMU_LDOVS_1	LDO输出电压0.9V
PMU_LDOVS_2	LDO输出电压0.95V
PMU_LDOVS_3	LDO输出电压1.0V
PMU_LDOVS_4	LDO输出电压1.05V
PMU_LDOVS_5	LDO输出电压1.1V
PMU_LDOVS_6	LDO输出电压1.15V
PMU_LDOVS_7	LDO输出电压1.2V
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* select output low voltage mode */
pmu_ldo_output_select (PMU_LDOVS_LOW);
```

函数 pmu_lvd_enable

函数pmu_lvd_disable描述见下表：

表 3-508. 函数 pmu_lvd_enable

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

例如：

```
/* enable PMU lvd */
pmu_lvd_enable ();
```

函数 pmu_lvd_disable

函数pmu_lvd_disable描述见下表：

表 3-509. 函数 pmu_lvd_disable

函数名称	pmu_lvd_disable
函数原型	void pmu_lvd_disable (void);
功能描述	关闭低压检测器
先决条件	-
被调用函数	-

输入参数{in}	
-	-
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* disable PMU lvd */
```

```
pmu_lvd_disable ();
```

函数 pmu_lowdriver_mode_enable

函数pmu_lowdriver_mode_enable描述见下表:

表 3-510. 函数 pmu_lowdriver_mode_enable

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

例如:

```
/* enable low-driver mode */
```

```
pmu_lowdriver_mode_enable ();
```

函数 pmu_lowdriver_mode_disable

函数pmu_lowdriver_mode_disable描述见下表:

表 3-511. 函数 pmu_lowdriver_mode_disable

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

例如：

```
/* disable low-driver mode */
pmu_lowdriver_mode_disable ();
```

函数 pmu_lowpower_driver_config

函数pmu_lowpower_driver_config描述见下表：

表 3-512. 函数 pmu_lowpower_driver_config

函数名称	pmu_lowpower_driver_config
函数原型	void pmu_lowpower_driver_config (void);
功能描述	低功耗LDO下驱动模式配置
先决条件	-
被调用函数	-
输入参数{in}	
mode	驱动模式
PMU_NORMALDR_ LOWPWR	正常驱动模式
PMU_LOWDR_LO WPWR	低驱动模式

输出参数{out}	
-	-
返回值	
-	-

例如:

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

```
pmu_lowpower_driver_config (PMU_NORMALDR_LOWPWR);
```

函数 pmu_normalpower_driver_config

函数pmu_normalpower_driver_config描述见下表:

表 3-513. 函数 pmu_normalpower_driver_config

函数名称	pmu_normalpower_driver_config
函数原型	void pmu_normalpower_driver_config (void);
功能描述	正常功耗LDO下驱动模式配置
先决条件	-
被调用函数	-
输入参数{in}	
mode	驱动模式
PMU_NORMALDR_LOWPWR	正常驱动模式
PMU_LOWDR_NORMALPWR	低驱动模式
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* normal driver when use low power LDO */
```

```
pmu_normalpower_driver_config (PMU_LOWDR_NORMALPWR);
```

函数 pmu_to_sleepmode

函数pmu_to_sleepmode描述见下表:

表 3-514. 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 3-515. 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

<code>PMU_LDO_NORMAL</code>	当系统进入深度睡眠模式时，LDO仍正常工作
<code>PMU_LDO_LOWPOWER</code>	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}	
<code>PMU_LOWDRIVER_ENABLE</code>	在深度睡眠模式使能低驱模式
<code>PMU_LOWDRIVER_DISABLE</code>	在深度睡眠模式关闭低驱模式
输入参数{in}	
<code>deepsleepmodecmd</code>	进入深度睡眠模式命令
<code>WFI_CMD</code>	WFI命令
<code>WFE_CMD</code>	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 3-516. 函数 `pmu_to_standbymode`

函数名称	<code>pmu_to_standbymode</code>
函数原型	<code>void pmu_to_standbymode(void);</code>
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 3-517. 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 3-518. 函数 pmu_wakeup_pin_disable

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

例如:

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

函数 pmu_backup_write_enable

函数pmu_backup_write_enable描述见下表:

表 3-519. 函数 pmu_backup_write_enable

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

例如:

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

函数 pmu_backup_write_disable

函数pmu_backup_write_disable描述见下表:

表 3-520. 函数 pmu_backup_write_disable

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

例如:

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

函数 pmu_vavd_enable

The description of pmu_vavd_enable is shown as below:

Table 3-3. Function pmu_vavd_enable

函数名称	pmu_vavd_enable
函数原型	void pmu_vavd_enable(void);
功能描述	使能PMU模拟电压检测
先决条件	-
被调用函数	-
Input parameter{in}	
-	-
Output parameter{out}	

-	-
Return value	
-	-

Example:

```
/* enable PMU analog voltage detector */
pmu_vavd_enable();
```

函数 pmu_vavd_disable

The description of pmu_vavd_disable is shown as below:

Table 3-4. Function pmu_vavd_disable

函数名称	pmu_vavd_disable
函数原型	void pmu_vavd_disable(void);
功能描述	disable PMU模拟电压检测
先决条件	-
被调用函数	-
Input parameter{in}	
-	-
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* disable PMU analog voltage detector */
pmu_vavd_disable();
```

函数 pmu_vovd_enable

The description of pmu_vovd_enable is shown as below:

Table 3-5. Function pmu_vovd_enable

函数名称	pmu_vovd_enable
函数原型	void pmu_vovd_enable(void);
功能描述	使能PMU过压检测
先决条件	-
被调用函数	-
Input parameter{in}	
-	-
Output parameter{out}	
-	-
Return value	

-	-
---	---

Example:

```
/* enable PMU voltage detector */
pmu_vovd_enable();
```

函数 pmu_vovd_disable

The description of pmu_vovd_disable is shown as below:

Table 3-6. Function pmu_vovd_disable

函数名称	pmu_vovd_disable
函数原型	void pmu_vovd_disable(void);
功能描述	失能PMU过压检测
先决条件	-
被调用函数	-
Input parameter{in}	
-	-
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* disable PMU voltage detector */
pmu_vovd_disable();
```

函数 pmu_vuvd_enable

The description of pmu_vuvd_enable is shown as below:

Table 3-7. Function pmu_vuvd_enable

函数名称	pmu_vuvd_enable
函数原型	void pmu_vuvd_enable (void);
功能描述	使能PMU低压检测
先决条件	-
被调用函数	-
Input parameter{in}	
-	-
Output parameter{out}	
-	-
Return value	
-	-

Example:


```
/* enable PMU core under voltage detector */
```

```
pmu_vuvd_enable();
```

函数 pmu_vuvd_disable

The description of pmu_vuvd_disable is shown as below:

Table 3-8. Function pmu_vuvd_disable

函数名称	pmu_vuvd_disable
函数原型	void pmu_vuvd_disable (void);
功能描述	失能PMU低压检测
先决条件	-
被调用函数	-
Input parameter{in}	
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* disable PMU core under voltage detector */
```

```
pmu_vuvd_disable();
```

函数 pmu_vavd_select

The description of pmu_vavd_select is shown as below:

Table 3-9. Function pmu_vavd_select

函数名称	pmu_vavd_select
函数原型	void pmu_vavd_select(uint32_t avdt_n);
功能描述	select analog voltage detector threshold
先决条件	-
被调用函数	-
Input parameter{in}	
avdt_n	select analog voltage detector threshold
PMU_VAVDVC_T_0	voltage threshold of analog voltage detector is 2.3V
PMU_VAVDVC_1	voltage threshold of analog voltage detector is 2.5V
PMU_VAVDVC_2	voltage threshold of analog voltage detector is 2.7V
PMU_VAVDVC_3	voltage threshold of analog voltage detector is 2.9V
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* select analog voltage detector threshold 2.9V */
pmu_vavd_select(PMU_VAVDVC_3);
```

函数 pmu_vovd_select

The description of pmu_vovd_select is shown as below:

Table 3-10. Function pmu_vovd_enable

函数名称	pmu_vovd_select
函数原型	void pmu_vovd_select (uint32_t ovdn);
功能描述	enable PMU voltage detector
先决条件	-
被调用函数	-
Input parameter{in}	
ovdn	voltage threshold value
PMU_VOVDVC_0	oltage threshold is 1.25V
PMU_VOVDVC_1	voltage threshold is 1.30V
PMU_VOVDVC_2	voltage threshold is 1.35V
PMU_VOVDVC_3	voltage threshold is 1.40V
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* select PMU core over voltage detector threshold 1.25V */
pmu_vovd_select (PMU_VOVDVC_0);
```

函数 pmu_vuvd_select

The description of pmu_vuvd_select is shown as below:

Table 3-11. Function pmu_vuvd_select

函数名称	pmu_vuvd_select
函数原型	void pmu_vuvd_select (uint32_t ldo_n);
功能描述	select PMU core under voltage detector threshold
先决条件	-
被调用函数	-
Input parameter{in}	
ldo_n	Select LDO output voltage
PMU_VUVDVC_0	LDO output voltage 1.05V mode
PMU_VUVDVC_1	LDO output voltage 0.95V mode
PMU_VUVDVC_2	LDO output voltage 0.85V mode

Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* select LDO output voltage 0.85V */
pmu_vuvd_select(PMU_VUVDVC_2);
```

函数 pmu_vovdo_dnf_select

The description of pmu_vovdo_dnf_select is shown as below:

Table 3-12. Function pmu_vovdo_dnf_enable

函数名称	pmu_vovdo_dnf_select
函数原型	void pmu_vovdo_dnf_select (uint32_t vovdo_value);
功能描述	选择过压数字滤波器滤波峰值长度
先决条件	-
被调用函数	-
Input parameter{in}	
vovdo_value	0x00-0xff
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* Filtering peak length*/
pmu_vovdo_dnf_select (0x55);
```

函数 pmu_vuvdo_dnf_select

The description of pmu_vuvdo_dnf_select is shown as below:

Table 3-13. Function pmu_vuvdo_dnf_select

函数名称	pmu_vuvdo_dnf_select
函数原型	void pmu_vuvdo_dnf_select (uint32_t vuvdo_value);
功能描述	选择低压数字滤波器滤波峰值长度
先决条件	-
被调用函数	-
Input parameter{in}	
vuvdo_value	0x00-0xff
Output parameter{out}	
-	-

Return value	
-	-

Example:

```
/* Filtering peak length */
```

```
pmu_vuvdo_dnf_select(0x55);
```

函数 pmu_flag_get

函数pmu_flag_get描述见下表:

表 3-521. 函数 pmu_flag_get

函数名称	pmu_flag_get
函数原型	FlagStatus pmu_flag_get(uint32_t flag);
功能描述	获取标志位
先决条件	-
被调用函数	-
输入参数{in}	
flag	标志位
PMU_FLAG_WAKEUP	唤醒标志
PMU_FLAG_STANDBY	待机标志
PMU_FLAG_LVD	低电压状态标志
PMU_FLAG_VAVDF	模拟电压检测标志
PMU_FLAG_VUVD F0	VCORE低压检测标志0
PMU_FLAG_VOVD F	VCORE过压检测标志
PMU_FLAG_VUVD F1	VCORE低压检测标志1
PMU_FLAG_LDOVSRF	电压选择准备标志
PMU_FLAG_LDRF	低电压准备标志

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

例如:

```
/* get flag state */
```

```
FlagStatus status;
```

```
status = pmu_flag_get (PMU_FLAG_WAKEUP);
```

函数 pmu_flag_clear

函数pmu_flag_clear描述见下表:

表 3-522. 函数 pmu_flag_clear

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

例如:

```
/* clear flag bit */
```

```
pmu_flag_clear (PMU_FLAG_RESET_WAKEUP);
```

3.21. 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. 外设寄存器描述

RCU寄存器列表如下表所示：

表 3-523. RCU 寄存器

寄存器名称	寄存器描述
RCU_CTL	控制寄存器
RCU_CFG0	时钟配置寄存器0
RCU_INT	时钟中断寄存器
RCU_APB2RST	APB2复位寄存器
RCU_APB1RST	APB1复位寄存器
RCU_AHBEN	AHB1使能寄存器
RCU_APB2EN	APB2使能寄存器
RCU_APB1EN	APB1使能寄存器
RCU_BDCTL	备份域控制寄存器
RCU_RSTSCK	复位源/时钟寄存器
RCU_AHBRST	AHB复位寄存器
RCU_CFG1	时钟配置寄存器1
RCU_PLLBWCFG	带宽配置寄存器
RCU_DSV	深度睡眠模式电压寄存器
RCU_CKFMCFG0	时钟频率监测配置寄存器0
RCU_CKFMCFG1	时钟频率监测配置寄存器1
RCU_CKFMCFG2	时钟频率监测配置寄存器2
RCU_CKFMCFG3	时钟频率监测配置寄存器3
RCU_ADDCTL	附加时钟控制寄存器
RCU_ADDINT	附加时钟中断寄存器
RCU_ADDAPB1RST	附加复位寄存器
RCU_ADDAPB1EN	附加使能寄存器
RCU_LOCK	锁定寄存器

3.21.2. 外设库函数说明

RCU库函数列表如下表所示：

表 3-524. RCU 库函数

库函数名称	库函数描述
rcu_deinit	复位RCU

rcu_periph_clock_enable	使能外设时钟
rcu_periph_clock_disable	禁能外设时钟
rcu_periph_clock_sleep_enable	使能睡眠模式下外设时钟
rcu_periph_clock_sleep_disable	禁能睡眠模式下外设时钟
rcu_periph_reset_enable	复位外设
rcu_periph_reset_disable	取消外设复位
rcu_bkp_reset_enable	复位备份域
rcu_bkp_reset_disable	取消备份域复位
rcu_system_clock_source_config	配置系统时钟源
rcu_system_clock_source_get	获取系统时钟源
rcu_ahb_clock_config	配置AHB时钟分频选择
rcu_apb1_clock_config	配置APB1时钟分频选择
rcu_apb2_clock_config	配置APB2时钟分频选择
rcu_ckout_config	配置CK_OUT时钟源
rcu_pll0_config	配置PLL0时钟
rcu_pll1_config	配置PLL1时钟
rcu_prediv0_config	配置PREDIV0分频因子
rcu_prediv1_config	配置PREDIV1分频因子
rcu_adc_clock_config	配置ADC预分频因子
rcu_usb_clock_config	配置USBFS预分频因子
rcu_rtc_clock_config	配置RTC时钟源选择
rcu_i2s1_clock_config	配置I2S1时钟源选择
rcu_i2s2_clock_config	配置I2S2时钟源选择
rcu_ck48m_clock_config	配置CK48M时钟源选择
rcu_fmc_clock_config	配置FMC时钟源选择
rcu_lxtal_drive_capability_config	配置LXTAL驱动能力
rcu_osc_i_stab_wait	等待振荡器稳定标志置位或振荡器启动超时
rcu_osc_i_on	打开振荡器
rcu_osc_i_off	关闭振荡器
rcu_osc_i_bypass_mode_enable	使能振荡器旁路模式，需先复位HXTALEN或LXTALEN
rcu_osc_i_bypass_mode_disable	禁能振荡器旁路模式，需先复位HXTALEN或LXTALEN

rcu_irc8m_adjust_value_set	设置IRC8M调节值
rcu_hxtal_clock_monitor_enable	使能HXTAL时钟监测
rcu_hxtal_clock_monitor_disable	禁能HXTAL时钟监测
rcu_lxtal_clock_monitor_enable	使能LXTAL时钟监测
rcu_lxtal_clock_monitor_disable	禁能LXTAL时钟监测
rcu_clock_freq_monitor_enable	使能时钟频率监测
rcu_clock_freq_monitor_disable	禁能时钟频率监测
rcu_irc8m_freq_monitor_config	配置IRC8M时钟频率监测范围
rcu_hxtal_monitor_thres_hold_config	配置HXTAL时钟频率监测阈值
rcu_pll0p_monitor_thres_hold_config	配置PLL0P时钟频率监测阈值
rcu_pll1_monitor_thresh_old_config	配置PLL1时钟频率监测阈值
rcu_deepsleep_voltage_set	设置深度睡眠模式电压
rcu_deepsleep_switch_delay_set	切换到IRC8M时钟并进入深度睡眠模式前的延时
rcu_reg_lock	锁定RCU寄存器
rcu_reg_unlock	解锁RCU寄存器
rcu_pll_bandwidth_config	配置PLL0/PLL1带宽
rcu_clock_freq_get	获取系统、总线和外设时钟频率
rcu_flag_get	获取时钟稳定、外设复位和时钟频率失效标志
rcu_flag_clear	清除时钟频率失效标志
rcu_all_reset_flag_clear	清除所有复位标志
rcu_interrupt_enable	使能时钟中断
rcu_interrupt_disable	禁能时钟中断
rcu_interrupt_flag_get	获取时钟中断标志
rcu_interrupt_flag_clear	清除时钟中断标志

枚举类型 rcu_periph_enum

表 3-525. 枚举类型 rcu_periph_enum

库函数名称	库函数描述
RCU_DMA0	DMA0时钟

RCU_DMA1	DMA1时钟
RCU_CRC	CRC时钟
RCU_EXMC	EXMC时钟
RCU_USBFS	USBFS时钟
RCU_CAU	CAU时钟
RCU_HAU	HAU时钟
RCU_TRNG	TRNG时钟
RCU_DMAMUX	DMAMUX时钟
RCU_TIMER1	TIMER1时钟
RCU_TIMER2	TIMER2时钟
RCU_TIMER3	TIMER3时钟
RCU_TIMER4	TIMER4时钟
RCU_TIMER5	TIMER5时钟
RCU_TIMER6	TIMER6时钟
RCU_TIMER16	TIMER16时钟
RCU_WWDGT	WWDGT时钟
RCU_SPI1	SPI1时钟
RCU_SPI2	SPI2时钟
RCU_USART1	USART1时钟
RCU_USART2	USART2时钟
RCU_UART3	UART3时钟
RCU_UART4	UART4时钟
RCU_I2C0	I2C0时钟
RCU_I2C1	I2C1时钟
RCU_CMP	CMP时钟
RCU_BKPI	BKPI时钟
RCU_PMU	PMU时钟
RCU_DAC	DAC时钟
RCU_RTC	RTC时钟
RCU_CTC	CTC时钟
RCU_AF	复用功能时钟
RCU_GPIOA	GPIOA时钟
RCU_GPIOB	GPIOB时钟
RCU_GPIOC	GPIOC时钟
RCU_GPIOD	GPIOD时钟
RCU_GPIOE	GPIOE时钟
RCU_ADC0	ADC0时钟
RCU_ADC1	ADC1时钟
RCU_TIMER0	TIMER0时钟
RCU_SPI0	SPI0时钟
RCU_TIMER7	TIMER7时钟
RCU_USART0	USART0时钟
RCU_ADC2	ADC2时钟

RCU_TIMER15	TIMER16时钟
RCU_TRIGSEL	TRIGSEL时钟
RCU_SYSCFG	SYSCFG时钟
RCU_CAN0	CAN0时钟
RCU_CAN1	CAN1时钟

枚举类型 `rcu_periph_sleep_enum`

表 3-526. 枚举类型 `rcu_periph_sleep_enum`

库函数名称	库函数描述
RCU_SRAM_SLP	睡眠模式下使能SRAM接口时钟
RCU_FMC_SLP	睡眠模式下使能FMC时钟

枚举类型 `rcu_periph_reset_enum`

表 3-527. 枚举类型 `rcu_periph_reset_enum`

库函数名称	库函数描述
RCU_USBFSRST	USBFS时钟复位
RCU_CAURST	CAU时钟复位
RCU_HAURST	HAU时钟复位
RCU_TRNGRST	TRNG时钟复位
RCU_DMAMUXRST	DMAMUX时钟复位
RCU_DMA0RST	DMA0X时钟复位
RCU_DMA1RST	DMA1时钟复位
RCU_TIMER1RST	TIMER1时钟复位
RCU_TIMER2RST	TIMER2时钟复位
RCU_TIMER3RST	TIMER3时钟复位
RCU_TIMER4RST	TIMER4时钟复位
RCU_TIMER5RST	TIMER5时钟复位
RCU_TIMER6RST	TIMER6时钟复位
RCU_TIMER16RST	TIMER16时钟复位
RCU_WWDGTRST	WWDGT时钟复位
RCU_SPI1RST	SPI1时钟复位
RCU_SPI2RST	SPI2时钟复位
RCU_USART1RST	USART1时钟复位
RCU_USART2RST	USART2时钟复位
RCU_UART3RST	UART3时钟复位
RCU_UART4RST	UART4时钟复位
RCU_I2C0RST	I2C0时钟复位
RCU_I2C1RST	I2C1时钟复位
RCU_CMPRST	CMP时钟复位
RCU_BKPIRST	BKPI时钟复位
RCU_PMURST	PMU时钟复位
RCU_DACRST	DAC时钟复位

RCU_CTCRST	CTC时钟复位
RCU_AFRST	复用功能时钟复位
RCU_GPIOARST	GPIOA时钟复位
RCU_GPIOBRST	GPIOB时钟复位
RCU_GPIOCRST	GPIOC时钟复位
RCU_GPIODRST	GPIOD时钟复位
RCU_GPIOERST	GPIOE时钟复位
RCU_ADC0RST	ADC0时钟复位
RCU_ADC1RST	ADC1时钟复位
RCU_TIMER0RST	TIMER0时钟复位
RCU_SPI0RST	SPI0时钟复位
RCU_TIMER7RST	TIMER7时钟复位
RCU_USART0RST	USART0时钟复位
RCU_ADC2RST	ADC2时钟复位
RCU_TIMER15RST	TIMER16时钟复位
RCU_TRIGSELRST	TRIGSEL时钟复位
RCU_SYSCFGRST	SYSCFG时钟复位
RCU_CAN0RST	CAN0时钟复位
RCU_CAN1RST	CAN1时钟复位

枚举类型 `rcu_flag_enum`

表 3-528. 枚举类型 `rcu_flag_enum`

库函数名称	库函数描述
RCU_FLAG_IRC8MST B	IRC8M稳定标志
RCU_FLAG_HXTALST B	HXTAL稳定标志
RCU_FLAG_PLL0STB	PLL0稳定标志
RCU_FLAG_PLL1STB	PLL1稳定标志
RCU_FLAG_LXTALST B	LXTAL稳定标志
RCU_FLAG_IRC40KST B	IRC40K稳定标志
RCU_FLAG_IRC48MS TB	IRC48M稳定标志
RCU_FLAG_EPRST	外部引脚复位标志
RCU_FLAG_PORRST	电源复位标志
RCU_FLAG_SWRST	软件复位标志
RCU_FLAG_FWDGTR ST	FWDGT复位标志
RCU_FLAG_WWDGTR ST	WWDGT复位标志

RCU_FLAG_LPRST	低功耗复位标志
RCU_FLAG_IRC8MCK FF	IRC8M时钟频率失效标志
RCU_FLAG_HXTALCK FF	HXTAL时钟频率失效标志
RCU_FLAG_PLL0PCK FF	PLL0P时钟频率失效标志
RCU_FLAG_PLL1CKF F	PLL1时钟频率失效标志

枚举类型 `rcu_int_flag_enum`

表 3-529. 枚举类型 `rcu_int_flag_enum`

库函数名称	库函数描述
RCU_INT_FLAG_IRC4 0KSTB	IRC40K稳定中断标志
RCU_INT_FLAG_LXTA LSTB	LXTAL稳定中断标志
RCU_INT_FLAG_IRC8 MSTB	IRC8M稳定中断标志
RCU_INT_FLAG_HXTA LSTB	HXTAL稳定中断标志
RCU_INT_FLAG_PLL0 STB	PLL0稳定中断标志
RCU_INT_FLAG_PLL1 STB	PLL1稳定中断标志
RCU_INT_FLAG_HCK M	HXTAL时钟卡死中断标志
RCU_INT_FLAG_LCK M	LXTAL时钟卡死中断标志
RCU_INT_FLAG_IRC4 8MSTB	IRC48M稳定中断标志

枚举类型 `rcu_flag_clear_enum`

枚举类型 `rcu_flag_clear_enum`

库函数名称	库函数描述
RCU_FLAG_IRC8MCK FF_CLR	清除IRC8M时钟频率失效中断标志
RCU_FLAG_HXTALCK FF_CLR	清除HXTAL时钟频率失效中断标志
RCU_FLAG_PLL0PCK FF_CLR	清除PLL0P时钟频率失效中断标志

RCU_FLAG_PLL1CKF F_CLR	清除PLL1时钟频率失效中断标志
---------------------------	------------------

枚举类型 `rcu_int_flag_clear_enum`

表 3-530. 枚举类型 `rcu_int_flag_clear_enum`

库函数名称	库函数描述
RCU_INT_FLAG_IRC4 0KSTB_CLR	清除IRC40K稳定中断标志
RCU_INT_FLAG_LXTA LSTB_CLR	清除LXTAL稳定中断标志
RCU_INT_FLAG_IRC8 MSTB_CLR	清除IRC8M稳定中断标志
RCU_INT_FLAG_HXTA LSTB_CLR	清除HXTAL稳定中断标志
RCU_INT_FLAG_PLLS TB_CLR	清除PLL稳定中断标志
RCU_INT_FLAG_PLL1 STB_CLR	清除PLL1稳定中断标志
RCU_INT_FLAG_HCK M_CLR	清除HXTAL时钟卡死中断标志
RCU_INT_FLAG_LCK M_CLR	清除LXTAL时钟卡死中断标志
RCU_INT_FLAG_IRC4 8MSTB_CLR	清除内部48MHz RC振荡器稳定中断标志

枚举类型 `rcu_int_enum`

表 3-531. 枚举类型 `rcu_int_enum`

库函数名称	库函数描述
RCU_INT_IRC40KSTB	IRC40K稳定中断
RCU_INT_LXTALSTB	LXTAL稳定中断
RCU_INT_IRC8MSTB	IRC8M稳定中断
RCU_INT_HXTALSTB	HXTAL稳定中断
RCU_INT_PLLSTB	PLL稳定中断
RCU_INT_PLL1STB	PLL1稳定中断
RCU_INT_LXTALCSS	LXTAL CSS中断
RCU_INT_IRC48MSTB	内部48MHz RC振荡器稳定中断
RCU_INT_IRC8MCKFF	IRC8M时钟频率失效中断
RCU_INT_HXTALCKFF	HXTAL时钟频率失效中断
RCU_INT_PLL0PCKFF	PLL0P时钟频率失效中断
RCU_INT_PLL1CKFF	PLL1时钟频率失效中断

枚举类型 `rcu_osc_type_enum`

表 3-532. 枚举类型 `rcu_osc_type_enum`

库函数名称	库函数描述
RCU_HXTAL	HXTAL
RCU_LXTAL	LXTAL
RCU_IRC8M	IRC8M
RCU_IRC48M	IRC48M
RCU_IRC40K	IRC40K
RCU_PLL0_CK	PLL0
RCU_PLL1_CK	PLL1

枚举类型 `rcu_clock_freq_enum`

表 3-533. 枚举类型 `rcu_clock_freq_enum`

库函数名称	库函数描述
CK_SYS	系统时钟
CK_AHB	AHB时钟
CK_APB1	APB1时钟
CK_APB2	APB2时钟
CK_PLL0	PLL0时钟

枚举类型 `rcu_ckfm_enum`

表 3-534. 枚举类型 `rcu_ckfm_enum`

库函数名称	库函数描述
RCU_IRC8MCKFM	IRC8M时钟频率监测
RCU_HXTALCKFM	HXTAL时钟频率监测
RCU_PLL0PCKFM	PLL0P时钟频率监测
RCU_PLL1CKFM	PLL1时钟频率监测

函数 `rcu_deinit`

函数`rcu_deinit`描述见下表：

表 3-535. 函数 `rcu_deinit`

函数名称	<code>rcu_deinit</code>
函数原型	<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 3-536. 函数 rcu_periph_clock_enable

函数名称	rcu_periph_clock_enable
函数原型	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, 请参考 rcu_periph_enum
RCU_DMAx (x = 0,1)	DMA 时钟
RCU_CRC	CRC 时钟
RCU_EXMC	EXMC 时钟
RCU_USBFS	USBFS 时钟
RCU_CAU	CAU 时钟
RCU_HAU	HAU 时钟
RCU_TRNG	TRNG 时钟
RCU_DMAMUX	DMAMUX 时钟
RCU_TIMERx (x = 0,1,2,3,4,5,6,7,16)	TIMER 时钟
RCU_WWDGT	WWDGT 时钟
RCU_SPIx (x = 0,1,2)	SPI 时钟
RCU_USARTx (x = 0,1,2)	USART 时钟
RCU_UARTx (x = 3,4)	UART 时钟
RCU_I2Cx (x = 0,1)	I2C 时钟
RCU_CMP	CMP 时钟
RCU_BKPI	BKP 接口时钟
RCU_PMU	PMU 时钟
RCU_DAC	DAC 时钟
RCU_RTC	RTC 时钟
RCU_CTC	CTC 时钟
RCU_AF	复用功能时钟

<i>RCU_GPIOx</i> ($x = A, B, C, D, E$)	GPIO 端口时钟
<i>RCU_ADCx</i> ($x = 0, 1, 2$)	ADC 时钟
<i>RCU_TRIGSEL</i>	TRIGSEL 时钟
<i>RCU_SYSCFG</i>	SYSCFG 时钟
<i>RCU_CANx</i> ($x = 0, 1$)	CAN 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 3-537. 函数 `rcu_periph_clock_disable`

函数名称	<code>rcu_periph_clock_disable</code>
函数原型	<code>void rcu_periph_clock_disable(rcu_periph_enum periph)</code>
功能描述	禁能外设时钟
先决条件	-
被调用函数	-
输入参数{in}	
periph	RCU 外设, 请参考 <code>rcu_periph_enum</code>
<i>RCU_DMAx</i> ($x = 0, 1$)	DMA 时钟
<i>RCU_CRC</i>	CRC 时钟
<i>RCU_EXMC</i>	EXMC 时钟
<i>RCU_USBFS</i>	USBFS 时钟
<i>RCU_CAU</i>	CAU 时钟
<i>RCU_HAU</i>	HAU 时钟
<i>RCU_TRNG</i>	TRNG 时钟
<i>RCU_DMAMUX</i>	DMAMUX 时钟
<i>RCU_TIMERx</i> ($x = 0, 1, 2, 3, 4, 5, 6, 7, 16$)	TIMER 时钟
<i>RCU_WWDGT</i>	WWDGT 时钟
<i>RCU_SPIx</i> ($x = 0, 1, 2$)	SPI 时钟
<i>RCU_USARTx</i> ($x = 0, 1, 2$)	USART 时钟
<i>RCU_UARTx</i> ($x = 3, 4$)	UART 时钟
<i>RCU_I2Cx</i> ($x = 0, 1$)	I2C 时钟

<i>RCU_CMP</i>	CMP 时钟
<i>RCU_BKPI</i>	BKP 接口时钟
<i>RCU_PMU</i>	PMU 时钟
<i>RCU_DAC</i>	DAC 时钟
<i>RCU_RTC</i>	RTC 时钟
<i>RCU_CTC</i>	CTC 时钟
<i>RCU_AF</i>	复用功能时钟
<i>RCU_GPIOx</i> (<i>x</i> = <i>A,B,C,D,E</i>)	GPIO 端口时钟
<i>RCU_ADCx</i> (<i>x</i> = 0, 1, 2)	ADC 时钟
<i>RCU_TRIGSEL</i>	TRIGSEL 时钟
<i>RCU_SYSCFG</i>	SYSCFG 时钟
<i>RCU_CANx</i> (<i>x</i> = 0, 1)	CAN 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 3-538. 函数 rcu_periph_clock_sleep_enable

函数名称	rcu_periph_clock_sleep_enable
函数原型	void rcu_periph_clock_sleep_enable(rcu_periph_sleep_enum periph)
功能描述	使能睡眠模式下外设时钟
先决条件	-
被调用函数	-
输入参数{in}	
periph	RCU 外设, 参考 rcu_periph_sleep_enum
<i>RCU_SRAM_SLP</i>	SRAM 时钟
<i>RCU_FMC_SLP</i>	FMC 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 3-539. 函数 rcu_periph_clock_sleep_disable

函数名称	rcu_periph_clock_sleep_disable
函数原型	void rcu_periph_clock_sleep_disable(rcu_periph_sleep_enum periph)
功能描述	禁能睡眠模式下外设时钟
先决条件	-
被调用函数	-
输入参数{in}	
periph	RCU 外设，参考 rcu_periph_sleep_enum
RCU_SRAM_SLP	SRAM 时钟
RCU_FMC_SLP	FMC 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 3-540. 函数 rcu_periph_reset_enable

函数名称	rcu_periph_reset_enable
函数原型	void rcu_periph_reset_enable(rcu_periph_reset_enum periph_reset)
功能描述	复位外设
先决条件	-
被调用函数	-
输入参数{in}	
periph_reset	RCU 外设复位，请参阅 rcu_periph_reset_enum
RCU_USBFSRST	复位 USBFS
RCU_CAURST	复位 CAU
RCU_HAURST	复位 HAU
RCU_TRNGRST	复位 TRNG
RCU_DMAMUXRST	复位 DMAMUX
RCU_DMAxRST (x = 0, 1)	DMA 时钟

<i>RCU_TIMERxRST</i> ($x = 0, 1, 2, 3, 4, 5, 6, 7, 16$)	复位 TIMER
<i>RCU_WWDGTRST</i>	复位 WWDGT
<i>RCU_SPIxRST</i> ($x = 0, 1, 2$)	复位 SPI
<i>RCU_USARTxRST</i> ($x = 0, 1, 2$)	复位 USART
<i>RCU_UARTxRST</i> ($x = 3, 4$)	复位 UART
<i>RCU_I2CxRST</i> ($x = 0, 1$)	复位 I2C
<i>RCU_CMPRST</i>	复位 CMP
<i>RCU_BKPIRST</i>	复位 BKPI
<i>RCU_PMURST</i>	复位 PMU
<i>RCU_DACRST</i>	复位 DAC
<i>RCU_CTCRST</i>	复位 CTC
<i>RCU_AFRST</i>	复位复用功能时钟
<i>RCU_ADCRST</i> ($x = 0, 1, 2$)	复位 ADC
<i>RCU_GPIOxRST</i> ($x = A, B, C, D, E$)	复位 GPIO 端口
<i>RCU_TRIGSELRST</i>	复位 TRIGSEL
<i>RCU_SYSCFGRST</i>	复位 SYSCFG
<i>RCU_CANxRST</i> ($x = 0, 1$)	复位 CAN
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 3-541. 函数 `rcu_periph_reset_disable`

函数名称	<code>rcu_periph_reset_disable</code>
函数原型	<code>void rcu_periph_reset_disable(rcu_periph_reset_enum periph_reset)</code>
功能描述	取消外设复位
先决条件	-
被调用函数	-
输入参数{in}	

periph_reset	RCU 外设复位, 请参阅 rcu_periph_reset_enum
<i>RCU_USBFSRST</i>	复位 USBFS
<i>RCU_CAURST</i>	复位 CAU
<i>RCU_HAURST</i>	复位 HAU
<i>RCU_TRNGRST</i>	复位 TRNG
<i>RCU_DMAMUXRST</i>	复位 DMAMUX
<i>RCU_DMAxRST</i> ($x = 0, 1$)	DMA 时钟
<i>RCU_TIMERxRST</i> ($x = 0, 1, 2, 3, 4, 5, 6, 7, 16$)	复位 TIMER
<i>RCU_WWDGTRST</i>	复位 WWDGT
<i>RCU_SPIxRST</i> ($x = 0, 1, 2$)	复位 SPI
<i>RCU_USARTxRST</i> ($x = 0, 1, 2$)	复位 USART
<i>RCU_UARTxRST</i> ($x = 3, 4$)	复位 UART
<i>RCU_I2CxRST</i> ($x = 0, 1$)	复位 I2C
<i>RCU_CMPRST</i>	复位 CMP
<i>RCU_BKPIRST</i>	复位 BKPI
<i>RCU_PMURST</i>	复位 PMU
<i>RCU_DACRST</i>	复位 DAC
<i>RCU_CTCRST</i>	复位 CTC
<i>RCU_AFRST</i>	复位复用功能时钟
<i>RCU_ADCRST</i> ($x = 0, 1, 2$)	复位 ADC
<i>RCU_GPIOxRST</i> ($x = A, B, C, D, E$)	复位 GPIO 端口
<i>RCU_TRIGSELRST</i>	复位 TRIGSEL
<i>RCU_SYSCFGRST</i>	复位 SYSCFG
<i>RCU_CANxRST</i> ($x = 0, 1$)	复位 CAN
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 3-542. 函数 rcu_bkp_reset_enable

函数名称	rcu_bkp_reset_enable
函数原型	void rcu_bkp_reset_enable(void)
功能描述	复位备份域
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 3-543. 函数 rcu_bkp_reset_disable

函数名称	rcu_bkp_reset_disable
函数原型	void rcu_bkp_reset_disable(void)
功能描述	取消备份域复位
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 3-544. 函数 rcu_system_clock_source_config

函数名称	rcu_system_clock_source_config
函数原型	void rcu_system_clock_source_config(uint32_t ck_sys)
功能描述	配置系统时钟源
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_IRC8M</i>	选择 CK_IRC8M 作为 CK_SYS 源
<i>RCU_CKSYSSRC_HXTAL</i>	选择 CK_HXTAL 作为 CK_SYS 源
<i>RCU_CKSYSSRC_PLL0P</i>	选择 CK_PLL0P 作为 CK_SYS 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 3-545. 函数 rcu_system_clock_source_get

函数名称	rcu_system_clock_source_get
函数原型	uint32_t rcu_system_clock_source_get(void)
功能描述	获取系统时钟源
先决条件	-
被调用函数	-
输入参数{in}	
-	-
输出参数{out}	
-	-
返回值	
uint32_t	选择哪个时钟作为 CK_SYS 源
<i>RCU_SCSS_IRC8M</i>	CK_IRC8M 被选作 CK_SYS 源

<i>RCU_SCSS_HXTAL</i>	CK_HXTAL 被选作 CK_SYS 源
<i>RCU_SCSS_PLL0P</i>	CK_PLL0P 被选作 CK_SYS 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 3-546. 函数 **rcu_ahb_clock_config**

函数名称	rcu_ahb_clock_config
函数原型	void rcu_ahb_clock_config(uint32_t ck_ahb)
功能描述	configure the AHB clock prescaler selection
先决条件	-
被调用函数	-
输入参数{in}	
ck_ahb	AHB 时钟分频器选择
<i>RCU_AHB_CKSYS_DIV1</i> <i>Vx(x = 1, 2, 4, 8, 16, 64, 128, 256, 512)</i>	选择 CK_SYS / x 作为 CK_AHB
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 3-547. 函数 **rcu_apb1_clock_config**

函数名称	rcu_apb1_clock_config
函数原型	void rcu_apb1_clock_config(uint32_t ck_apb1)
功能描述	configure the APB1 clock prescaler selection
先决条件	-
被调用函数	-
输入参数{in}	
ck_apb1	APB1 时钟分频器选择

<i>RCU_APB1_CKAHB_D</i> IV1	选择 CK_AHB 作为 CK_APB1
<i>RCU_APB1_CKAHB_D</i> IV2	选择 CK_AHB/2 作为 CK_APB1
<i>RCU_APB1_CKAHB_D</i> IV4	选择 CK_AHB/4 作为 CK_APB1
<i>RCU_APB1_CKAHB_D</i> IV8	选择 CK_AHB/8 作为 CK_APB1
<i>RCU_APB1_CKAHB_D</i> IV16	选择 CK_AHB/16 作为 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 3-548. 函数 rcu_apb2_clock_config

函数名称	rcu_apb2_clock_config
函数原型	void rcu_apb2_clock_config(uint32_t ck_apb2)
功能描述	配置 APB2 时钟分频选择
先决条件	-
被调用函数	-
输入参数{in}	
ck_apb2	APB2 时钟预分频器选择
<i>RCU_APB2_CKAHB_D</i> IV1	选择 CK_AHB 作为 CK_APB2
<i>RCU_APB2_CKAHB_D</i> IV2	选择 CK_AHB/2 作为 CK_APB2
<i>RCU_APB2_CKAHB_D</i> IV4	选择 CK_AHB/4 作为 CK_APB2
<i>RCU_APB2_CKAHB_D</i> IV8	选择 CK_AHB/8 作为 CK_APB2
<i>RCU_APB2_CKAHB_D</i> IV16	选择 CK_AHB/16 作为 CK_APB2
输出参数{out}	
-	-
返回值	

-	-
---	---

例如:

```
/* configure CK_AHB/8 as CK_APB2 */
rcu_apb2_clock_config(RCU_APB2_CK_AHB_DIV8);
```

函数 rcu_ckout_config

函数rcu_ckout_config描述见下表:

表 3-549. 函数 rcu_ckout_config

函数名称	rcu_ckout_config
函数原型	void rcu_ckout_config(uint32_t ckout_src)
功能描述	配置 CK_OUT 时钟源
先决条件	-
被调用函数	-
输入参数{in}	
ckout_src	CK_OUT 时钟源选择
RCU_CKOUT0SRC_CKSYS	系统时钟已选择
RCU_CKOUT0SRC_IRC8M	高速 8M 内部振荡器时钟已选择
RCU_CKOUT0SRC_HXTAL	HXTAL 已选择
RCU_CKOUT0SRC_CKPLL0_DIV2	CK_PLL0 / 2 已选择
RCU_CKOUT0SRC_CKPLL1_DIV2	CK_PLL1 / 2 已选择
RCU_CKOUT0SRC_LXTAL	LXTAL 时钟已选择
RCU_CKOUT0SRC_IRC48M	高速 48M 内部振荡器时钟已选择
RCU_CKOUT0SRC_IRC40K	IRC40K 时钟已选择
输出参数{out}	
-	-
返回值	
-	-

例如:

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

函数 rcu_pll0_config

函数rcu_pll0_config描述见下表:

表 3-550. 函数 rcu_pll0_config

函数名称	rcu_pll0_config
函数原型	void rcu_pll0_config(uint32_t pll0_src, uint32_t pll0_mul, uint32_t pll0_div)
功能描述	配置 PLL0 时钟
先决条件	-
被调用函数	-
输入参数{in}	
pll0_src	PLL0 时钟源选择
RCU_PLL0SRC_IRC8M	选择 IRC8M 时钟作为 PLL0 的源时钟
RCU_PLL0SRC_IRC48M	选择 IRC48M 作为 PLL0 的源时钟
RCU_PLL0SRC_HXTAL	选择 HXTAL 作为 PLL0 的源时钟
RCU_PLL0SRC_CKPL1	选择 CKPLL1 作为 PLL0 的源时钟
输入参数{in}	
pll0_mul	PLL0 时钟倍频因子
RCU_PLL0_MULx (x = 4,5,6..63)	PLL0 时钟倍频因子
输入参数{in}	
pll0_div	PLL0 时钟分频因子
RCU_PLL0_DIVx (x = 1..16)	PLL0 时钟分频因子
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* configure the PLL0 clock */
```

```
rcu_pll0_config(RCU_PLL0SRC_IRC48M, RCU_PLL0_MUL4, RCU_PLL0_DIV2);
```

函数 rcu_pll1_config

函数rcu_pll1_config描述见下表:

表 3-551. 函数 rcu_pll1_config

函数名称	rcu_pll1_config
函数原型	void rcu_pll1_config(uint32_t pll1_src, uint32_t pll1_mul)

功能描述	配置 PLL1 时钟
先决条件	-
被调用函数	-
输入参数{in}	
pll1_src	PLL1 时钟源选择
RCU_PLL1SRC_HXTAL	选择 HXTAL 作为 PLL1 的源时钟
RCU_PPLL1SRC_IRC48M	选择 IRC48M 作为 PLL1 的源时钟
输入参数{in}	
pll11_mul	PLL1 时钟倍频因数
RCU_PLL1_MULx (x = 4,5,6..63)	PLL1 时钟倍频因子
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* configure the PLL1 clock */
```

```
rcu_pll1_config(RCU_PLL1SRC_HXTAL, RCU_PLL1_MUL4);
```

函数 rcu_prediv0_config

函数rcu_prediv0_config描述见下表：

表 3-552. 函数 rcu_prediv0_config

函数名称	rcu_prediv0_config
函数原型	void rcu_prediv0_config(uint32_t prediv0_div)
功能描述	配置 PREDIV0 分频因子
先决条件	-
被调用函数	-
输入参数{in}	
prediv0_div	PREDIV0 分频因子
RCU_PREDIV0_DIVx (x = 1..16)	PREDIV0 输入源时钟被除以 x
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* configure the PREDIV0 division factor */
```

```
rcu_prediv0_config(RCU_PREDIV0_DIV2);
```

函数 rcu_prediv1_config

函数rcu_prediv1_config描述见下表：

表 3-553. 函数 rcu_prediv1_config

函数名称	rcu_prediv1_config
函数原型	void rcu_prediv1_config(uint32_t prediv1_div)
功能描述	配置 PREDIV1 分频因子
先决条件	-
被调用函数	-
输入参数{in}	
prediv1_div	PREDIV1 分频因子
RCU_PREDIV1_DIVx (x = 1..16)	PREDIV1 输入源时钟被除以 x
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* configure the PREDIV1 division factor */
```

```
rcu_prediv1_config(RCU_PREDIV1_DIV2);
```

函数 rcu_adc_clock_config

函数rcu_adc_clock_config描述见下表：

表 3-554. 函数 rcu_adc_clock_config

函数名称	rcu_adc_clock_config
函数原型	void rcu_adc_clock_config(uint32_t adc_psc)
功能描述	配置 ADC 预分频因子
先决条件	-
被调用函数	-
输入参数{in}	
adc_psc	ADC 预分频因子
RCU_CKADC_CKAPB2_DIV2	ADC 预分频选择 CK_APB2 / 2
RCU_CKADC_CKAPB2_DIV4	ADC 预分频选择 CK_APB2 / 4
RCU_CKADC_CKAPB2_DIV6	ADC 预分频选择 CK_APB2 / 6
RCU_CKADC_CKAPB2_DIV8	ADC 预分频选择 CK_APB2 / 8

<i>RCU_CKADC_CKAPB2</i> <i>_DIV12</i>	ADC 预分频选择 CK_APB2 / 12
<i>RCU_CKADC_CKAPB2</i> <i>_DIV16</i>	ADC 预分频选择 CK_APB2 / 16
<i>RCU_CKADC_CKAHB</i> <i>_DIV3</i>	ADC 预分频选择 CK_AHB / 3
<i>RCU_CKADC_CKAHB</i> <i>_DIV5</i>	ADC 预分频选择 CK_AHB / 5
<i>RCU_CKADC_CKAHB</i> <i>_DIV6</i>	ADC 预分频选择 CK_AHB / 6
<i>RCU_CKADC_CKAHB</i> <i>_DIV10</i>	ADC 预分频选择 CK_AHB / 10
<i>RCU_CKADC_CKAHB</i> <i>_DIV20</i>	ADC 预分频选择 CK_AHB / 20
输出参数{out}	
-	-
返回值	
-	-

例如:

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

```
rcu_adc_clock_config(RCU_CKADC_CKAPB2_DIV2);
```

函数 rcu_usb_clock_config

函数rcu_usb_clock_config描述见下表:

表 3-555. 函数 rcu_usb_clock_config

函数名称	rcu_usb_clock_config
函数原型	void rcu_usb_clock_config(uint32_t usb_psc)
功能描述	配置 USBFS 预分频因子
先决条件	-
被调用函数	-
输入参数{in}	
usb_psc	USB 预分频因子
<i>RCU_CKUSB_CKPLL</i> <i>_DIV2</i>	USBFS 预分频选择 CK_PLL0 / 2
<i>RCU_CKUSB_CKPLL</i> <i>_DIV3</i>	USBFS 预分频选择 CK_PLL0 / 3
<i>RCU_CKUSB_CKPLL</i> <i>_DIV4</i>	USBFS 预分频选择 CK_PLL0 / 4
<i>RCU_CKUSB_CKPLL</i> <i>_DIV5</i>	USBFS 预分频选择 CK_PLL0 / 5

<i>RCU_CKUSB_CKPLL_</i> <i>DIV6</i>	USBFS 预分频选择 CK_PLL0 / 6
<i>RCU_CKUSB_CKPLL_</i> <i>DIV7</i>	USBFS 预分频选择 CK_PLL0 / 7
<i>RCU_CKUSB_CKPLL_</i> <i>DIV8</i>	USBFS 预分频选择 CK_PLL0 / 8
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* configure the USBFS prescaler factor */
rcu_usb_clock_config(RCU_CKUSB_CKPLL_DIV2);
```

函数 rcu_rtc_clock_config

函数rcu_rtc_clock_config描述见下表:

表 3-556. 函数 rcu_rtc_clock_config

函数名称	rcu_rtc_clock_config
函数原型	void rcu_rtc_clock_config(uint32_t rtc_clock_source)
功能描述	配置 RTC 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_NONE</i>	未选择时钟
<i>RCU_RTCSRC_LXTAL</i>	选择 CK_LXTAL 作为 RTC 源时钟
<i>RCU_RTCSRC_IRC40</i> <i>K</i>	选择 CK_IRC40K 作为 RTC 源时钟
<i>RCU_RTCSRC_HXTAL</i> <i>_DIV_128</i>	选择 CK_HXTAL / 128 作为 RTC 源时钟
输出参数{out}	
-	-
返回值	
-	-

例如:

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

函数 rcu_i2s1_clock_config

函数rcu_i2s1_clock_config描述见下表:

表 3-557. 函数 rcu_i2s1_clock_config

函数名称	rcu_i2s1_clock_config
函数原型	void rcu_i2s1_clock_config(uint32_t i2s_clock_source, uint32_t i2s_clock_div)
功能描述	配置 I2S1 时钟源选择
先决条件	-
被调用函数	-
输入参数{in}	
i2s_clock_source	I2S1 时钟源选择
RCU_I2S1SRC_CKSYS	系统时钟被选作 I2S1 源时钟
RCU_I2S1SRC_CKPLL1	CK_PLL1 被选作 I2S1 源时钟
输入参数{in}	
i2s_clock_div	I2S 时钟分频因子
RCU_I2S1_DIVx	(x = 1...32)
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* configure the I2S1 clock source selection */
```

```
rcu_i2s1_clock_config(RCU_I2S1SRC_CKSYS, RCU_I2S1_DIV2);
```

函数 rcu_i2s2_clock_config

函数rcu_i2s2_clock_config描述见下表:

表 3-558. 函数 rcu_i2s2_clock_config

函数名称	rcu_i2s2_clock_config
函数原型	void rcu_i2s2_clock_config(uint32_t i2s_clock_source, uint32_t i2s_clock_div)
功能描述	配置 I2S2 时钟源选择
先决条件	-
被调用函数	-
输入参数{in}	
i2s_clock_source	I2S2 时钟源选择
RCU_I2S2SRC_CKSYS	系统时钟被选为 I2S2 源时钟

<i>RCU_I2S2SRC_CKPLL</i> 1	CK_PLL1 被选为 I2S1 源时钟
输入参数{in}	
i2s_clock_div	I2S 时钟分频因子
<i>RCU_I2S2_DIVx</i>	(x = 1...32)
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* configure the I2S2 clock source selection */
```

```
rcu_i2s2_clock_config(RCU_I2S2SRC_CKSYS, RCU_I2S2_DIV2 );
```

函数 rcu_ck48m_clock_config

函数rcu_ck48m_clock_config描述见下表:

表 3-559. 函数 rcu_ck48m_clock_config

函数名称	rcu_ck48m_clock_config
函数原型	void rcu_ck48m_clock_config(uint32_t ck48m_clock_source)
功能描述	配置 CK48M 时钟源选择
先决条件	-
被调用函数	-
输入参数{in}	
ck48m_clock_source	CK48M 时钟源选择
<i>RCU_CK48MSRC_CKPLL</i>	选择 CK_PLL0 作为 CK48M 源时钟
<i>RCU_CK48MSRC_IRC48M</i>	选择 CK_IRC48M 作为 CK48M 源时钟
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* configure the CK48M clock source selection */
```

```
rcu_ck48m_clock_config(RCU_CK48MSRC_CKPLL0);
```

函数 rcu_fmc_clock_config

函数rcu_fmc_clock_config描述见下表:

表 3-560. 函数 rcu_fmc_clock_config

函数名称	rcu_fmc_clock_config
函数原型	void rcu_fmc_clock_config(uint32_t fmc_clock_source, uint32_t fmc_clock_div)
功能描述	配置 FMC 时钟源选择
先决条件	-
被调用函数	-
输入参数{in}	
fmc_clock_source	FMC 时钟源选择
RCU_FMC_CK_AHB	选择 CK_AHB 时钟作为 FMC 源时钟
RCU_FMC_CK_SYS	选择系统时钟作为 FMC 源时钟
RCU_FMC_CK_PLL0	选择 CK_PLL0 时钟作为 FMC 源时钟
RCU_FMC_CK_PLL1	选择 CK_PLL1 时钟作为 FMC 源时钟
输入参数{in}	
fmc_clock_div	FMC clock division factor
RCU_FMC_DIVx	(x = 1...16)
输出参数{out}	
-	-
返回值	
-	-

例如:

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

```
rcu_fmc_clock_config(RCU_FMC_CK_AHB, RCU_FMC_DIV2);
```

函数 rcu_lxtal_drive_capability_config

函数rcu_lxtal_drive_capability_config描述见下表:

表 3-561. 函数 rcu_lxtal_drive_capability_config

函数名称	rcu_lxtal_drive_capability_config
函数原型	void rcu_lxtal_drive_capability_config(uint32_t lxtal_dricap)
功能描述	configure the LXTAL drive capability
先决条件	-
被调用函数	-
输入参数{in}	
lxtal_dricap	LXTAL 驱动能力
RCU_LXTAL_LOWDRI	较低驱动能力
RCU_LXTAL_MED_LO WDRI	中低驱动能力
RCU_LXTAL_MED_HI GHDRI	中高驱动能力
RCU_LXTAL_HIGHDRI	较高驱动能力

输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* configure the LXTAL drive capability */
```

```
rcu_lxtal_drive_capability_config(RCU_LXTAL_LOWDRI);
```

函数 rcu_osci_stab_wait

函数rcu_osci_stab_wait描述见下表：

表 3-562. 函数 rcu_osci_stab_wait

函数名称	rcu_osci_stab_wait
函数原型	ErrStatus rcu_osci_stab_wait(rcu_osci_type_enum osci)
功能描述	等待振荡器稳定标志置位或振荡器启动超时
先决条件	-
被调用函数	-
输入参数{in}	
osci	振荡器类型，参见 rcu_osci_type_enum
RCU_HXTAL	高速晶体振荡器 (HXTAL)
RCU_LXTAL	低速晶体振荡器 (LXTAL)
RCU_IRC8M	内部 8M RC 振荡器 (IRC8M)
RCU_IRC48M	内部 48M RC 振荡器 (IRC48M)
RCU_IRC40K	内部 40K RC 振荡器 (IRC40K)
RCU_PLL0_CK	锁相环 0 (PLL0)
RCU_PLL1_CK	锁相环 1 (PLL1)
输出参数{out}	
-	-
返回值	
ErrStatus	SUCCESS or 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 3-563. 函数 rcu_osci_on

函数名称	rcu_osci_on
------	-------------

函数原型	void rcu_osc_on(rcu_osc_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 rcu_osc_type_enum
<i>RCU_HXTAL</i>	高速晶体振荡器 (HXTAL)
<i>RCU_LXTAL</i>	低速晶体振荡器 (LXTAL)
<i>RCU_IRC8M</i>	内部 8M RC 振荡器 (IRC8M)
<i>RCU_IRC48M</i>	内部 48M RC 振荡器 (IRC48M)
<i>RCU_IRC40K</i>	内部 40K RC 振荡器 (IRC40K)
<i>RCU_PLL0_CK</i>	锁相环 0 (PLL0)
<i>RCU_PLL1_CK</i>	锁相环 1 (PLL1)
输出参数{out}	
-	-
返回值	
-	-

例如:

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

```
rcu_osc_on(RCU_HXTAL);
```

函数 rcu_osc_off

函数rcu_osc_off描述见下表:

表 3-564. 函数 rcu_osc_off

函数名称	rcu_osc_off
函数原型	void rcu_osc_off(rcu_osc_type_enum osci)
功能描述	关闭振荡器
先决条件	-
被调用函数	-
输入参数{in}	
osci	振荡器类型, 请参考 rcu_osc_type_enum
<i>RCU_HXTAL</i>	高速晶体振荡器 (HXTAL)
<i>RCU_LXTAL</i>	低速晶体振荡器 (LXTAL)
<i>RCU_IRC16M</i>	内部 8M RC 振荡器 (IRC8M)
<i>RCU_IRC48M</i>	内部 48M RC 振荡器 (IRC48M)
<i>RCU_IRC40K</i>	内部 40K RC 振荡器 (IRC40K)
<i>RCU_PLL0_CK</i>	锁相环 0 (PLL0)
<i>RCU_PLL1_CK</i>	锁相环 1 (PLL1)
输出参数{out}	
-	-

返回值	
-	-

例如：

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

```
rcu_osc_off(RCU_HXTAL);
```

函数 rcu_osc_bypass_mode_enable

函数rcu_osc_bypass_mode_enable描述见下表：

表 3-565. 函数 rcu_osc_bypass_mode_enable

函数名称	rcu_osc_bypass_mode_enable
函数原型	void rcu_osc_bypass_mode_enable(rcu_osc_type_enum osci)
功能描述	使能振荡器旁路模式，需先复位 HXTALEN 或 LXTALEN
先决条件	-
被调用函数	-
输入参数{in}	
osci	振荡器类型，参阅 rcu_osc_type_enum
RCU_HXTAL	高速晶体振荡器 (HXTAL)
RCU_LXTAL	低速晶体振荡器 (LXTAL)
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_osc_bypass_mode_disable

函数rcu_osc_bypass_mode_disable描述见下表：

表 3-566. 函数 rcu_osc_bypass_mode_disable

函数名称	rcu_osc_bypass_mode_disable
函数原型	void rcu_osc_bypass_mode_disable(rcu_osc_type_enum osci)
功能描述	禁能振荡器旁路模式，需先复位 HXTALEN 或 LXTALEN
先决条件	-
被调用函数	-
输入参数{in}	
osci	振荡器类型，参阅 rcu_osc_type_enum
RCU_HXTAL	高速晶体振荡器 (HXTAL)
RCU_LXTAL	低速晶体振荡器 (LXTAL)

输出参数{out}	
-	-
返回值	
-	-

例如：

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

```
rcu_osci_bypass_mode_disable(RCU_HXTAL);
```

函数 rcu_irc8m_adjust_value_set

函数rcu_irc8m_adjust_value_set描述见下表：

表 3-567. 函数 rcu_irc8m_adjust_value_set

函数名称	rcu_irc8m_adjust_value_set
函数原型	void rcu_irc8m_adjust_value_set(uint32_t irc8m_adjval)
功能描述	设置 IRC8M 调节值
先决条件	-
被调用函数	-
输入参数{in}	
irc8m_adjval	IRC8M 校准值必须介于 0 到 0x1F 之间
0x00 - 0x1F	IRC8M 校准值
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* set the IRC8M adjust value */
```

```
rcu_irc8m_adjust_value_set(0x10);
```

函数 rcu_hxtal_clock_monitor_enable

函数rcu_hxtal_clock_monitor_enable描述见下表：

表 3-568. 函数 rcu_hxtal_clock_monitor_enable

函数名称	rcu_hxtal_clock_monitor_enable
函数原型	void rcu_hxtal_clock_monitor_enable(void)
功能描述	使能 HXTAL 时钟监测
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 3-569. 函数 rcu_hxtal_clock_monitor_disable

函数名称	rcu_hxtal_clock_monitor_disable
函数原型	void rcu_hxtal_clock_monitor_disable(void)
功能描述	禁能 HXTAL 时钟监测
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_lxtal_clock_monitor_enable

函数rcu_lxtal_clock_monitor_enable描述见下表：

表 3-570. 函数 rcu_lxtal_clock_monitor_enable

函数名称	rcu_lxtal_clock_monitor_enable
函数原型	void rcu_lxtal_clock_monitor_enable(void)
功能描述	使能 LXTAL 时钟监测
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 LXTAL clock monitor */
```

```
rcu_lxtal_clock_monitor_enable();
```

函数 rcu_lxtal_clock_monitor_disable

函数rcu_lxtal_clock_monitor_disable描述见下表：

表 3-571. 函数 rcu_lxtal_clock_monitor_disable

函数名称	rcu_lxtal_clock_monitor_disable
函数原型	void rcu_lxtal_clock_monitor_disable(void)
功能描述	禁能 LXTAL 时钟监测
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 LXTAL clock monitor */
```

```
rcu_lxtal_clock_monitor_disable();
```

函数 rcu_clock_freq_monitor_enable

函数rcu_clock_freq_monitor_enable描述见下表：

表 3-572. 函数 rcu_clock_freq_monitor_enable

函数名称	rcu_clock_freq_monitor_enable
函数原型	void rcu_clock_freq_monitor_enable(rcu_ckfm_enum ckfm)
功能描述	使能时钟频率监测
先决条件	-
被调用函数	-
输入参数{in}	
ckfm	时钟频率监测使能类型，参考 rcu_ckfm_enum
RCU_IRC8MCKFM	IRC8M 时钟频率监测
RCU_HXTALCKFM	HXTAL 时钟频率监测
RCU_PLL0PCKFM	PLL0P 时钟频率监测
RCU_PLL1CKFM	PLL1 时钟频率监测
输出参数{out}	

-	-
返回值	
-	-

例如：

```
/* enable the IRC8M clock frequency monitor */
```

```
rcu_clock_freq_monitor_enable(RCU_IRC8MCKFM);
```

函数 rcu_clock_freq_monitor_disable

函数rcu_clock_freq_monitor_disable描述见下表：

表 3-573. 函数 rcu_clock_freq_monitor_disable

函数名称	rcu_clock_freq_monitor_disable
函数原型	void rcu_clock_freq_monitor_disable(rcu_ckfm_enum ckfm)
功能描述	禁能时钟频率监测
先决条件	-
被调用函数	-
输入参数{in}	
ckfm	时钟频率监测使能类型，参考 rcu_ckfm_enum
<i>RCU_IRC8MCKFM</i>	IRC8M 时钟频率监测
<i>RCU_HXTALCKFM</i>	HXTAL 时钟频率监测
<i>RCU_PLL0PCKFM</i>	PLL0P 时钟频率监测
<i>RCU_PLL1CKFM</i>	PLL1 时钟频率监测
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* disable the IRC8M clock frequency monitor */
```

```
rcu_clock_freq_monitor_disable(RCU_IRC8MCKFM);
```

函数 rcu_irc8m_freq_monitor_config

函数rcu_irc8m_freq_monitor_config描述见下表：

表 3-574. 函数 rcu_irc8m_freq_monitor_config

函数名称	rcu_irc8m_freq_monitor_config
函数原型	void rcu_irc8m_freq_monitor_config(uint32_t range)
功能描述	配置 IRC8M 时钟频率监测范围
先决条件	-
被调用函数	-
输入参数{in}	

range	IRC8M 时钟频率监测配置
<code>RCU_IRC8MCKFMC_5_PERCENT</code>	IRC8M 时钟频率监测范围为±5%
<code>RCU_IRC8MCKFMC_10_PERCENT</code>	IRC8M 时钟频率监测范围为±10%
<code>RCU_IRC8MCKFMC_15_PERCENT</code>	IRC8M 时钟频率监测范围为±15%
<code>RCU_IRC8MCKFMC_20_PERCENT</code>	IRC8M 时钟频率监测范围为±20%
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* IRC8M clock frequency monitor range configuration */
```

```
rcu_irc8m_freq_monitor_config(RCU_IRC8MCKFMC_5_PERCENT);
```

函数 `rcu_hxtal_monitor_threshold_config`

函数 `rcu_hxtal_monitor_threshold_config` 描述见下表：

表 3-575. 函数 `rcu_hxtal_monitor_threshold_config`

函数名称	<code>rcu_hxtal_monitor_threshold_config</code>
函数原型	<code>void rcu_hxtal_monitor_threshold_config(uint32_t lthreshold, uint32_t hthreshold)</code>
功能描述	配置 HXTAL 时钟频率监测阈值
先决条件	-
被调用函数	-
输入参数{in}	
lthreshold	低频阈值配置
<code>0x00 - 0xffff</code>	低频阈值
输入参数{in}	
hthreshold	高频阈值配置
<code>0x00 - 0xffff</code>	高频阈值
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* HXTAL clock frequency monitor threshold configuration */
```

```
rcu_hxtal_monitor_threshold_config(0x07f, 07ff);
```

函数 rcu_pll0p_monitor_threshold_config

函数rcu_pll0p_monitor_threshold_config描述见下表:

表 3-576. 函数 rcu_pll0p_monitor_threshold_config

函数名称	rcu_pll0p_monitor_threshold_config
函数原型	void rcu_pll0p_monitor_threshold_config(uint32_t lthreshold, uint32_t hthreshold)
功能描述	配置 PLL0P 时钟频率监测阈值
先决条件	-
被调用函数	-
输入参数{in}	
lthreshold	低频阈值配置
0x00 - 0x3ff	低频阈值
输入参数{in}	
hthreshold	高频阈值配置
0x00 - 0x3ff	高频阈值
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* PLL0P clock frequency monitor threshold configuration */
```

```
rcu_pll0p_monitor_threshold_config(0x07f, 0x300);
```

函数 rcu_pll1_monitor_threshold_config

函数rcu_pll1_monitor_threshold_config描述见下表:

表 3-577. 函数 rcu_pll1_monitor_threshold_config

函数名称	rcu_pll1_monitor_threshold_config
函数原型	void rcu_pll1_monitor_threshold_config(uint32_t lthreshold, uint32_t hthreshold)
功能描述	配置 PLL1 时钟频率监测阈值
先决条件	-
被调用函数	-
输入参数{in}	
lthreshold	低频阈值配置
0x00 - 0x3ff	低频阈值
输入参数{in}	
hthreshold	高频阈值配置
0x00 - 0x3ff	高频阈值
输出参数{out}	

-	-
返回值	
-	-

例如：

```
/* PLL1 clock frequency monitor threshold configuration */
```

```
rcu_pll1_monitor_threshold_config(0x07f, 0x300);
```

函数 rcu_deepsleep_voltage_set

函数rcu_deepsleep_voltage_set描述见下表：

表 3-578. 函数 rcu_deepsleep_voltage_set

函数名称	rcu_deepsleep_voltage_set
函数原型	void rcu_deepsleep_voltage_set(uint32_t dsvol)
功能描述	设置深度睡眠模式电压
先决条件	-
被调用函数	-
输入参数{in}	
dsvol	深度睡眠模式电压
<i>RCU_DEEPSLEEP_V_0</i>	内核电压为默认值
<i>RCU_DEEPSLEEP_V_1</i>	内核电压为（默认值-0.05）V
<i>RCU_DEEPSLEEP_V_2</i>	内核电压为（默认值-0.1）V
<i>RCU_DEEPSLEEP_V_3</i>	内核电压为（默认值-0.15）V
<i>RCU_DEEPSLEEP_V_4</i>	内核电压为（默认值-0.2）V
<i>RCU_DEEPSLEEP_V_5</i>	内核电压为（默认值-0.25）V
<i>RCU_DEEPSLEEP_V_6</i>	内核电压为（默认值-0.3）V（不建议客户使用）
<i>RCU_DEEPSLEEP_V_7</i>	内核电压为（默认值-0.35）V（不建议客户使用）
输出参数{out}	
-	-
返回值	
-	-

例如：

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

```
rcu_deepsleep_voltage_set(RCU_DEEPSLEEP_V_0);
```

函数 rcu_deepsleep_switch_delay_set

函数rcu_deepsleep_switch_delay_set描述见下表：

表 3-579. 函数 rcu_deepsleep_switch_delay_set

函数名称	rcu_deepsleep_switch_delay_set
函数原型	void rcu_deepsleep_switch_delay_set(uint32_t irc8m_cnt)
功能描述	切换到 IRC8M 时钟并进入深度睡眠模式前的延时
先决条件	-
被调用函数	-
输入参数{in}	
hsi_cnt	1 ~ 31
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* delay before switch to IRC8M clock and enter deep-sleep mode */
```

```
rcu_deepsleep_switch_delay_set(5);
```

函数 rcu_reg_lock

函数rcu_reg_lock描述见下表：

表 3-580. 函数 rcu_reg_lock

函数名称	rcu_reg_lock
函数原型	void rcu_reg_lock(void)
功能描述	锁定 RCU 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：

```
/* lock rcu register */
```

```
rcu_reg_lock();
```

函数 rcu_reg_unlock

函数rcu_reg_unlock描述见下表:

表 3-581. 函数 rcu_reg_unlock

函数名称	rcu_reg_unlock
函数原型	void rcu_reg_unlock(void)
功能描述	解锁 RCU 寄存器
先决条件	-
被调用函数	-
输入参数{in}	
-	-
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* unlock rcu register */
rcu_reg_unlock();
```

函数 rcu_pll_bandwidth_config

函数rcu_pll_bandwidth_config描述见下表:

表 3-582. 函数 rcu_pll_bandwidth_config

函数名称	rcu_pll_bandwidth_config
函数原型	void rcu_pll_bandwidth_config(uint32_t pll0_bw, uint32_t pll1_bw)
功能描述	配置 PLL0/PLL1 带宽
先决条件	-
被调用函数	-
输入参数{in}	
pll0_bw	PLL0 带宽配置, 必须在 0 到 0xF 之间
0 ~ 15	PLL0 带宽配置
输入参数{in}	
pll1_bw	PLL1 带宽配置, 必须在 0 到 0xF 之间
0 ~ 15	PLL1 带宽配置
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* config PLL0/PLL1 bandwidth */
```

```
rcu_pll_bandwidth_config(0, 0);
```

函数 rcu_clock_freq_get

函数rcu_clock_freq_get描述见下表:

表 3-583. 函数 rcu_clock_freq_get

函数名称	rcu_clock_freq_get
函数原型	uint32_t rcu_clock_freq_get(rcu_clock_freq_enum clock)
功能描述	获取系统、总线和外设时钟频率
先决条件	-
被调用函数	-
输入参数{in}	
clock	要获取的时钟频率
CK_SYS	系统时钟频率
CK_AHB	AHB 时钟频率
CK_APB1	APB1 时钟频率
CK_APB2	APB2 时钟频率
CK_PLL0	PLL0 时钟频率
输出参数{out}	
-	-
返回值	
uint32_t	系统时钟频率, AHB, APB1, APB2, PLL0

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 3-584. 函数 rcu_flag_get

函数名称	rcu_flag_get
函数原型	FlagStatus rcu_flag_get(rcu_flag_enum flag)
功能描述	获取时钟稳定、外设复位和时钟频率失效标志
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, 请参考 rcu_flag_enum
RCU_FLAG_IRC8MST B	IRC8M 稳定标志

<i>RCU_FLAG_HXTALST</i> <i>B</i>	HXTAL 稳定标志
<i>RCU_FLAG_PLL0STB</i>	PLL0 稳定标志
<i>RCU_FLAG_PLL1STB</i>	PLL1 稳定标志（仅适用于 CL 系列）
<i>RCU_FLAG_LXTALST</i> <i>B</i>	LXTAL 稳定标志
<i>RCU_FLAG_IRC40KST</i> <i>B</i>	IRC40K 稳定标志
<i>RCU_FLAG_IRC48MS</i> <i>TB</i>	IRC48M 稳定标志
<i>RCU_FLAG_EPRST</i>	外部 PIN 复位标志
<i>RCU_FLAG_PORRST</i>	电源复位标志
<i>RCU_FLAG_SWRST</i>	软件复位标志
<i>RCU_FLAG_FWDGTR</i> <i>ST</i>	自由看门狗定时器复位标志
<i>RCU_FLAG_WWDGTR</i> <i>ST</i>	窗口看门狗定时器复位标志
<i>RCU_FLAG_LPRST</i>	低功耗复位标志
<i>RCU_FLAG_IRC8MCK</i> <i>FF</i>	IRC8M 时钟频率故障标志
<i>RCU_FLAG_HXTALCK</i> <i>FF</i>	HXTAL 时钟频率故障标志
<i>RCU_FLAG_PLL0PCK</i> <i>FF</i>	PLL0P 时钟频率故障标志
<i>RCU_FLAG_PLL1CKF</i> <i>F</i>	PLL1 时钟频率故障标志
输出参数{out}	
-	-
返回值	
FlagStatus	status of flag (RESET or SET)

例如：

```

/* get the clock stabilization, peripheral reset and clock frequency failure flags */

/* get the clock stabilization interrupt flag */

if(SET == rcu_interrupt_flag_get(RCU_INT_FLAG_HXTALSTB)){

};

```

函数 rcu_flag_clear

函数rcu_flag_clear描述见下表：

表 3-585. 函数 rcu_flag_clear

函数名称	rcu_flag_clear
------	----------------

函数原型	void rcu_flag_clear(rcu_flag_clear_enum flag)
功能描述	清除时钟频率失效标志
先决条件	-
被调用函数	-
输入参数{in}	
flag	时钟频率故障标志，参见 rcu_flag_clear_enum
RCU_FLAG_IRC8MCK FF_CLR	清除 IRC8M 时钟频率故障中断标志
RCU_FLAG_HXTALCK FF_CLR	清除 HXTAL 时钟频率故障中断标志
RCU_FLAG_PLL0PCK FF_CLR	清除 PLL0P 时钟频率故障中断标志
RCU_FLAG_PLL1CKF F_CLR	清除 PLL1 时钟频率故障中断标志
输出参数{out}	
-	-
返回值	
-	-

Example:

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

函数 rcu_all_reset_flag_clear

函数rcu_all_reset_flag_clear描述见下表:

表 3-586. 函数 rcu_all_reset_flag_clear

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

例如:

```
/* clear 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 3-587. 函数 rcu_interrupt_enable

函数名称	rcu_interrupt_enable
函数原型	void rcu_interrupt_enable(rcu_int_enum interrupt)
功能描述	使能时钟中断
先决条件	-
被调用函数	-
输入参数{in}	
interrupt	中断时钟稳定中断，参考 rcu_int_enum
RCU_INT_IRC40KSTB	IRC40K 稳定中断使能
RCU_INT_LXTALSTB	LXTAL 稳定中断使能
RCU_INT_IRC8MSTB	IRC8M 稳定中断使能
RCU_INT_HXTALSTB	HXTAL 稳定中断使能
RCU_INT_PLL0STB	PLL0 稳定中断使能
RCU_INT_PLL1STB	PLL1 稳定中断使能
RCU_INT_IRC48MSTB	IRC48M 稳定中断使能
RCU_INT_IRC8MCKFF	IRC8M 时钟频率故障中断使能
RCU_INT_HXTALCKFF	HXTAL 时钟频率故障中断使能
RCU_INT_PLL0PCKFF	PLL0P 时钟频率故障中断使能
RCU_INT_PLL1CKFF	PLL1 时钟频率故障中断使能
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* enable the HXTAL stabilization interrupt */
rcu_interrupt_enable(RCU_INT_HXTALSTB);
```

函数 rcu_interrupt_disable

函数rcu_interrupt_disable描述见下表：

表 3-588. 函数 rcu_interrupt_disable

函数名称	rcu_interrupt_disable
函数原型	void rcu_interrupt_disable(rcu_int_enum interrupt)
功能描述	禁能时钟中断
先决条件	-
被调用函数	-
输入参数{in}	
interrupt	中断时钟稳定中断，参考 rcu_int_enum

<i>RCU_INT_IRC40KSTB</i>	IRC40K 稳定中断使能
<i>RCU_INT_LXTALSTB</i>	LXTAL 稳定中断使能
<i>RCU_INT_IRC8MSTB</i>	IRC8M 稳定中断使能
<i>RCU_INT_HXTALSTB</i>	HXTAL 稳定中断使能
<i>RCU_INT_PLL0STB</i>	PLL0 稳定中断使能
<i>RCU_INT_PLL1STB</i>	PLL1 稳定中断使能
<i>RCU_INT_IRC48MSTB</i>	IRC48M 稳定中断使能
<i>RCU_INT_IRC8MCKFF</i>	IRC8M 时钟频率故障中断使能
<i>RCU_INT_HXTALCKFF</i>	HXTAL 时钟频率故障中断使能
<i>RCU_INT_PLL0CKFF</i>	PLL0P 时钟频率故障中断使能
<i>RCU_INT_PLL1CKFF</i>	PLL1 时钟频率故障中断使能
输出参数{out}	
-	-
返回值	
-	-

例如:

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

```
rcu_interrupt_disable(RCU_INT_HXTALSTB);
```

函数 rcu_interrupt_flag_get

函数rcu_interrupt_flag_get描述见下表:

表 3-589. 函数 rcu_interrupt_flag_get

函数名称	rcu_interrupt_flag_get
函数原型	FlagStatus rcu_interrupt_flag_get(rcu_int_flag_enum int_flag)
功能描述	获取时钟中断标志
先决条件	-
被调用函数	-
输入参数{in}	
int_flag	中断和时钟卡住标志, 参考 rcu_int_flag_enum
<i>RCU_INT_FLAG_IRC40KSTB</i>	IRC40K 稳定中断标志
<i>RCU_INT_FLAG_LXTALSTB</i>	LXTAL 稳定中断标志
<i>RCU_INT_FLAG_IRC8MSTB</i>	IRC8M 稳定中断标志
<i>RCU_INT_FLAG_HXTALSTB</i>	HXTAL 稳定中断标志
<i>RCU_INT_FLAG_PLL0STB</i>	PLL0 稳定中断标志
<i>RCU_INT_FLAG_PLL1STB</i>	PLL1 稳定中断标志

<i>RCU_INT_FLAG_HCK</i> <i>M</i>	HXTAL 时钟卡住中断标志
<i>RCU_INT_FLAG_LCK</i> <i>M</i>	LXTAL 时钟卡住中断标志
<i>RCU_INT_FLAG_IRC4</i> <i>8MSTB</i>	IRC48M 稳定中断标志
输出参数{out}	
-	-
返回值	
FlagStatus	SET or 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 3-590. 函数 rcu_interrupt_flag_clear

函数名称	rcu_interrupt_flag_clear
函数原型	void rcu_interrupt_flag_clear(rcu_int_flag_clear_enum int_flag)
功能描述	清除时钟中断标志
先决条件	-
被调用函数	-
输入参数{in}	
int_flag	清除时钟稳定和卡死中断标志, 详见 rcu_int_flag_clear_enum
<i>RCU_INT_FLAG_IRC4</i> <i>0KSTB_CLR</i>	清除 IRC40K 稳定中断标志
<i>RCU_INT_FLAG_LXTA</i> <i>LSTB_CLR</i>	清除 LXTAL 稳定中断标志
<i>RCU_INT_FLAG_IRC8</i> <i>MSTB_CLR</i>	清除 IRC8M 稳定中断标志
<i>RCU_INT_FLAG_HXTA</i> <i>LSTB_CLR</i>	清除 HXTAL 稳定中断标志
<i>RCU_INT_FLAG_PLL0</i> <i>STB_CLR</i>	清除 PLL0 稳定中断标志
<i>RCU_INT_FLAG_PLL1</i> <i>STB_CLR</i>	清除 PLL1 稳定中断标志
<i>RCU_INT_FLAG_HCK</i> <i>M_CLR</i>	清除 HXTAL 时钟卡死中断标志

<code>RCU_INT_FLAG_LCK M_CLR</code>	清除 LXTAL 时钟卡死中断标志
<code>RCU_INT_FLAG_IRC4 8MSTB_CLR</code>	清除 IRC48M 稳定中断标志
输出参数{out}	
-	-
返回值	
-	-

例如：

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

```
rcu_interrupt_flag_clear(RCU_INT_FLAG_HXTALSTB_CLR);
```

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RTC库函数进行说明。

3.22.1. 外设寄存器描述

RTC寄存器列表如下表所示：

表 3-591. 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. 外设库函数描述

RTC库函数列表如下表所示：

表 3-592. RTC 库函数

库函数名称	库函数描述
<code>rtc_configuration_mode_enter</code>	进入RTC配置模式

库函数名称	库函数描述
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_interrupt_enable	使能RTC中断
rtc_interrupt_disable	失能RTC中断
rtc_flag_get	获取RTC标志位状态
rtc_flag_clear	清除RTC标志位状态

函数 rtc_interrupt_enable

函数rtc_interrupt_enable描述见下表：

表 3-593. 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 3-594. 函数 rtc_interrupt_disable

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

例如：

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

```
rtc_lwoff_wait();
```

```
/* disable the RTC second interrupt */
```

```
rtc_interrupt_disable(RTC_INT_SECOND);
```

函数 rtc_configuration_mode_enter

函数rtc_configuration_mode_enter描述见下表：

表 3-595. 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 3-596. 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 3-597. 函数 rtc_lwoff_wait

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

例如：

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

```
rtc_lwoff_wait();

/* enable the RTC second interrupt */

rtc_interrupt_enable(RTC_INT_SECOND);
```

函数 rtc_register_sync_wait

函数rtc_register_sync_wait描述见下表：

表 3-598. 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 3-599. 函数 rtc_counter_get

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

例如：


```

/* get the counter value */

uint32_t rtc_counter_value;

rtc_counter_value = rtc_counter_get();

```

函数 rtc_counter_set

函数rtc_counter_set描述见下表：

表 3-600. 函数 rtc_counter_set

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

例如：

```

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

rtc_lwoff_wait();

/* set counter value to 0xFFFF */

rtc_counter_set(0xFFFF);

```

函数 rtc_prescaler_set

函数rtc_prescaler_set描述见下表：

表 3-601. 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 3-602. 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 3-603. 函数 rtc_divider_get

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

-	-
返回值	
RTC divider value	RTC分频值

例如：

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

函数 rtc_flag_get

函数rtc_flag_get描述见下表：

表 3-604. 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
RTC_FLAG_OVERF LOW	溢出中断标志位
RTC_FLAG_RSYN	寄存器同步标志位
RTC_FLAG_LWOF	最近一次对RTC寄存器的写操作完成标志位
输出参数{out}	
-	-
返回值	
FlagStatus	SET 或 RESET

例如：

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

函数 rtc_flag_clear

函数rtc_flag_clear描述见下表：

表 3-605. 函数 rtc_flag_clear

函数名称	rtc_flag_clear
函数原型	void rtc_flag_clear(uint32_t flag);
功能描述	清除RTC标志位状态
先决条件	-
被调用函数	-
输入参数{in}	
flag	待清除的RTC标志位
RTC_FLAG_SECON D	秒中断标志位
RTC_FLAG_ALARM	闹钟中断标志位
RTC_FLAG_OVERF LOW	溢出中断标志位
RTC_FLAG_RSYN	寄存器同步标志位
输出参数{out}	
-	-
返回值	
-	-

例如：

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

3.23. SYSCFG

章节 [3.23.1](#) 描述了SYSCFG的寄存器列表，章节 [3.23.2](#) 对SYSCFG库函数进行说明。

3.23.1. 外设寄存器说明

SYSCFG寄存器列表如下表所示：

表 3-606. SYSCFG 寄存器

寄存器名称	寄存器描述
SYSCFG_CFG0	SYSCFG配置寄存器0
SYSCFG_CFG1	SYSCFG配置寄存器1
SYSCFG_LKCTL	SYSCFG死锁控制寄存器
SYSCFG_BUSTO	SYSCFG总线超时寄存器
SYSCFG_TIMERCI SEL	SYSCFG定时器输入选择寄存器
SYSCFG_FPUINTE N	SYSCFGFPU中断使能寄存器
SYSCFG_SRAMW P	SYSCFGSRAM写保护寄存器

寄存器名称	寄存器描述
SYSCFG_SRAMC CSTAT	SYSCFGSRAMECC状态寄存器
SYSCFG_SRAMC CCS	SYSCFGSRAMECC控制和状态寄存器
SYSCFG_BUSTOS TAT	SYSCFG总线超时状态寄存器
SYSCFG_TIMER0T RGCFG0	TIMER0 TRG配置寄存器0
SYSCFG_TIMER0T RGCFG1	TIMER0 TRG配置寄存器1
SYSCFG_TIMER0T RGCFG2	TIMER0 TRG配置寄存器2
SYSCFG_TIMER1T RGCFG0	TIMER1 TRG配置寄存器0
SYSCFG_TIMER1T RGCFG1	TIMER1 TRG配置寄存器1
SYSCFG_TIMER1T RGCFG2	TIMER1 TRG配置寄存器2
SYSCFG_TIMER2T RGCFG0	TIMER2 TRG配置寄存器0
SYSCFG_TIMER2T RGCFG1	TIMER2 TRG配置寄存器1
SYSCFG_TIMER2T RGCFG2	TIMER2 TRG配置寄存器2
SYSCFG_TIMER3T RGCFG0	TIMER3 TRG配置寄存器0
SYSCFG_TIMER3T RGCFG1	TIMER3 TRG配置寄存器1
SYSCFG_TIMER3T RGCFG2	TIMER3 TRG配置寄存器2
SYSCFG_TIMER4T RGCFG0	TIMER4 TRG配置寄存器0
SYSCFG_TIMER4T RGCFG1	TIMER4 TRG配置寄存器1
SYSCFG_TIMER4T RGCFG2	TIMER4 TRG配置寄存器2
SYSCFG_TIMER7T RGCFG0	TIMER7 TRG配置寄存器0
SYSCFG_TIMER7T RGCFG1	TIMER7 TRG配置寄存器1
SYSCFG_TIMER7T RGCFG2	TIMER7 TRG配置寄存器2

寄存器名称	寄存器描述
SYSCFG_TIMER15 TRGCFG0	TIMER15 TRG配置寄存器0
SYSCFG_TIMER15 TRGCFG1	TIMER15 TRG配置寄存器1
SYSCFG_TIMER15 TRGCFG2	TIMER15 TRG配置寄存器2
SYSCFG_TIMER16 TRGCFG0	TIMER16 TRG配置寄存器0
SYSCFG_TIMER16 TRGCFG1	TIMER16 TRG配置寄存器1
SYSCFG_TIMER16 TRGCFG2	TIMER16 TRG配置寄存器2

3.23.2. 外设库函数说明

SYSCFG库函数列表如下表所示：

表 3-607. SYSCFG 库函数

库函数名称	库函数描述
syscfg_deinit	复位SYSCFG寄存器
syscfg_bootmode_get	获取当前启动模式
syscfg_i2c_fast_mode_plus_enable	使能 I2Cx Fast Mode Plus
syscfg_i2c_fast_mode_plus_disable	失能 I2Cx Fast Mode Plus
syscfg_lockup_enable	使能模块锁定功能
syscfg_bus_timeout_enable	使能总线超时
syscfg_bus_timeout_disable	失能总线超时
syscfg_timer_input_source_select	选择TIMER通道输入源
syscfg_sram_page_wp_enable	使能SRAM页面写保护
syscfg_sram_ecc_error_bit_get	获取SRAM ECC单比特错误位位置
syscfg_sram_ecc_error_addr_get	获取SRAM ECC错误地址
syscfg_fpu_interrupt_enable	使能FPU中断
syscfg_fpu_interrupt_disable	失能 FPU 中断
syscfg_sram_ecc_interrupt_enable	使能SRAM ECC错误中断
syscfg_sram_ecc_interrupt_disable	失能SRAM ECC错误中断
syscfg_bus_timeout_flag_get	获取总线超时标志
syscfg_bus_timeout_flag_clear	清除总线超时标志
syscfg_sram_ecc_flag_get	获取SRAM ECC错误标志
syscfg_sram_ecc_flag_clear	清除SRAM ECC错误标志

枚举类型 `timer_channel_input_enum`表 3-608. 枚举类型 `timer_channel_input_enum`

枚举名称	枚举描述
<code>TIMER15_CIO_INPUT_TIMER15_CH0</code>	选择GPIO作为TIMER15 CIO输入
<code>TIMER15_CIO_INPUT_IRC40K</code>	选择IRC40K作为TIMER15 CIO输入
<code>TIMER15_CIO_INPUT_LXTAL</code>	选择LXTAL作为TIMER15 CIO输入
<code>TIMER16_CIO_INPUT_TIMER16_CH0</code>	选择GPIO作为TIMER16 CIO输入
<code>TIMER16_CIO_INPUT_CKOUT</code>	选择CKOUT作为TIMER16 CIO输入

函数 `syscfg_deinit`

函数`syscfg_deinit`描述见下表:

表 3-609. 函数 `syscfg_deinit`

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

例如:

```
/* reset SYSCFG registers */
```

```
syscfg_deinit();
```

函数 `syscfg_bootmode_get`

函数`syscfg_bootmode_get`描述见下表:

表 3-610. 函数 `syscfg_bootmode_get`

函数名称	<code>syscfg_bootmode_get</code>
函数原型	<code>uint32_t syscfg_bootmode_get(void)</code>
功能描述	获取当前启动模式
先决条件	-
被调用函数	-
输入参数{in}	
-	-

输出参数{out}	
-	-
返回值	
uint32_t	当前启动模式
SYSCFG_BOOTMODE_FLASH	从主 Flash 启动
SYSCFG_BOOTMODE_BOOTLOADER	从引导程序启动
SYSCFG_BOOTMODE_SRAM	从嵌入式 SRAM 启动
SYSCFG_BOOTMODE_OTP1	从嵌入式 OTP1 启动

Example:

```
/* select boot from main flash */
```

```
syscfg_bootmode_select(SYSCFG_BOOTMODE_FLASH);
```

函数 syscfg_i2c_fast_mode_plus_enable

函数syscfg_i2c_fast_mode_plus_enable描述见下表:

表 3-611. 函数 syscfg_i2c_fast_mode_plus_enable

函数名称	syscfg_i2c_fast_mode_plus_enable
函数原型	void syscfg_i2c_fast_mode_plus_enable(uint32_t i2c_fmp)
功能描述	使能 I2Cx Fm+模式
先决条件	-
被调用函数	-
输入参数{in}	
i2c_fmp	I2C Fm+模式
SYSCFG_I2C0_FMP	I2C0 Fm+模式
SYSCFG_I2C1_FMP	I2C1 Fm+模式
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* enable I2C0 fust mode plus function */
```

```
syscfg_i2c_fast_mode_plus_enable(SYSCFG_I2C0_FMP);
```


函数 syscfg_i2c_fast_mode_plus_disable

函数syscfg_i2c_fast_mode_plus_disable描述见下表:

表 3-612. 函数 syscfg_i2c_fast_mode_plus_disable

函数名称	syscfg_i2c_fast_mode_plus_disable
函数原型	void syscfg_i2c_fast_mode_plus_disable(uint32_t i2c_fmp);
功能描述	禁能I2C Fm+模式
先决条件	-
被调用函数	-
输入参数{in}	
i2c_fmp	I2C Fm+模式
SYSCFG_I2C0_FMP P	I2C0 Fm+模式
SYSCFG_I2C1_FMP P	I2C1 Fm+模式
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* disable I2C0 fust mode plus function */
```

```
syscfg_i2c_fast_mode_plus_disable(SYSCFG_I2C0_FMP);
```

函数 syscfg_lockup_enable

函数syscfg_lockup_enable描述见下表:

表 3-613. 函数 syscfg_lockup_enable

函数名称	syscfg_lockup_enable
函数原型	void syscfg_lockup_enable(uint32_t lockup)
功能描述	使能模块锁定功能
先决条件	-
被调用函数	-
输入参数{in}	
lockup	请提供具体内容以便翻译。
SYSCFG_CPU_LOCKUP P	CPU 锁定信号
SYSCFG_SRAM_LOCKUP KUP	SRAM ECC 双错误信号
SYSCFG_LVD_LOCKUP P	LVD 信号已连接
输出参数{out}	

-	-
返回值	
-	-

Example:

```
/* enable module lockup function */
```

```
syscfg_lockup_enable(SYSCFG_CPU_LOCKUP);
```

函数 syscfg_bus_timeout_enable

函数syscfg_bus_timeout_enable描述见下表:

表 3-614. 函数 syscfg_bus_timeout_enable

函数名称	syscfg_bus_timeout_enable
函数原型	void syscfg_bus_timeout_enable(uint32_t busto)
功能描述	使能总线超时
先决条件	-
被调用函数	-
输入参数 {in}	
busto	总线超时
SYSCFG_CPUCBUS_T IMEROUT	CPU Cbus 超时使能位
SYSCFG_CPUSBUS_T IMEROUT	CPU Sbus 超时使能位
SYSCFG_DMA0BUS_T IMEROUT	DMA0 总线超时使能位
SYSCFG_DMA1BUS_T IMEROUT	DMA1 总线超时使能位
输出参数 {out}	
-	-
返回值	
-	-

Example:

```
/* enable CPU Cbus timeout */
```

```
syscfg_bus_timeout_enable(SYSCFG_BUSTO_CPUCBUSTO);
```

函数 syscfg_bus_timeout_disable

函数syscfg_bus_timeout_disable描述见下表:

表 3-615. 函数 syscfg_bus_timeout_disable

函数名称	syscfg_bus_timeout_disable
函数原型	void syscfg_bus_timeout_disable(uint32_t busto)

功能描述	失能总线超时
先决条件	-
被调用函数	-
输入参数 {in}	
busto	总线超时
<i>SYSCFG_CPUCBUS_T IMEROUT</i>	CPU Cbus 超时使能位
<i>SYSCFG_CPUSBUS_T IMEROUT</i>	CPU Sbus 超时使能位
<i>SYSCFG_DMA0BUS_T IMEROUT</i>	DMA0 总线超时使能位
<i>SYSCFG_DMA1BUS_T IMEROUT</i>	DMA1 总线超时使能位
输出参数 {out}	
-	-
返回值	
-	-

Example:

```
/* disable CPU Cbus timeout */
```

```
syscfg_bus_timeout_disable(SYSCFG_BUSTO_CPUCBUSTO);
```

函数 **syscfg_timer_input_source_select**

函数 **syscfg_timer_input_source_select** 描述见下表:

表 3-616. 函数 **syscfg_timer_input_source_select**

函数名称	syscfg_timer_input_source_select
函数原型	void syscfg_timer_input_source_select(timer_channel_input_enum timer_input)
功能描述	选择 TIMER 通道输入源
先决条件	-
被调用函数	-
输入参数 {in}	
timer_input	TIMER 通道输入选择, 参考 timer_channel_input_enum
<i>TIMER15_CIO_INPUT_ TIMER15_CH0</i>	TIMER15_CH0 输入
<i>TIMER15_CIO_INPUT_ RC40K</i>	IRC40K
<i>TIMER15_CIO_INPUT_ LXTAL</i>	LXTAL
<i>TIMER16_CIO_INPUT_ TIMER16_CH0</i>	TIMER16_CH0 输入
<i>TIMER16_CIO_INPUT_ CKOUT</i>	CKOUT

CKOUT	
输出参数 {out}	
-	-
返回值	
-	-

Example:

```
/* select timer channel input source */
syscfg_timer_input_source_select(TIMER15_CIO_INPUT_IRC40K);
```

函数 syscfg_sram_page_wp_enable

函数syscfg_sram_page_wp_enable描述见下表:

表 3-617. 函数 syscfg_sram_page_wp_enable

函数名称	syscfg_sram_page_wp_enable
函数原型	void syscfg_sram_page_wp_enable(uint32_t page)
功能描述	使能 SRAM 页面写保护
先决条件	-
被调用函数	-
输入参数 {in}	
page	SRAM 页
SYSCFG_SRAM_WP_PAGE0	SRAM 第 0 页面写保护
SYSCFG_SRAM_WP_PAGE1	SRAM 第 1 页面写保护
SYSCFG_SRAM_WP_PAGE2	SRAM 第 2 页面写保护
SYSCFG_SRAM_WP_PAGE3	SRAM 第 3 页面写保护
SYSCFG_SRAM_WP_PAGE4	SRAM 第 4 页面写保护
SYSCFG_SRAM_WP_PAGE5	SRAM 第 5 页面写保护
SYSCFG_SRAM_WP_PAGE6	SRAM 第 6 页面写保护
SYSCFG_SRAM_WP_PAGE7	SRAM 第 7 页面写保护
SYSCFG_SRAM_WP_PAGE8	SRAM 第 8 页面写保护
SYSCFG_SRAM_WP_PAGE9	SRAM 第 9 页面写保护
SYSCFG_SRAM_WP_PAGE10	SRAM 第 10 页面写保护

SYSCFG_SRAM_WP_ PAGE11	SRAM 第 11 页面写保护
SYSCFG_SRAM_WP_ PAGE12	SRAM 第 12 页面写保护
SYSCFG_SRAM_WP_ PAGE13	SRAM 第 13 页面写保护
SYSCFG_SRAM_WP_ PAGE14	SRAM 第 14 页写保护
SYSCFG_SRAM_WP_ PAGE15	SRAM 第 15 页写保护
SYSCFG_SRAM_WP_ PAGE16	SRAM 第 16 页写保护
SYSCFG_SRAM_WP_ PAGE17	SRAM 第 17 页写保护
SYSCFG_SRAM_WP_ PAGE18	SRAM 第 18 页写保护
SYSCFG_SRAM_WP_ PAGE19	SRAM 第 19 页写保护
SYSCFG_SRAM_WP_ PAGE20	SRAM 第 20 页写保护
SYSCFG_SRAM_WP_ PAGE21	SRAM 第 21 页写保护
SYSCFG_SRAM_WP_ PAGE22	SRAM 第 22 页写保护
SYSCFG_SRAM_WP_ PAGE23	SRAM 第 23 页写保护
SYSCFG_SRAM_WP_ PAGE24	SRAM 第 24 页写保护
SYSCFG_SRAM_WP_ PAGE25	SRAM 第 25 页写保护
SYSCFG_SRAM_WP_ PAGE26	SRAM 第 26 页写保护
SYSCFG_SRAM_WP_ PAGE27	SRAM 第 27 页写保护
SYSCFG_SRAM_WP_ PAGE28	SRAM 第 28 页写保护
SYSCFG_SRAM_WP_ PAGE29	SRAM 第 29 页写保护
SYSCFG_SRAM_WP_ PAGE30	SRAM 第 30 页写保护
SYSCFG_SRAM_WP_ PAGE31	SRAM 第 31 页写保护
输出参数 {out}	

-	-
返回值	
-	-

Example:

```
/* enable SRAM page 0 write protection */
syscfg_sram_page_wp_enable(SYSCFG_SRAMWP_SRAM_P0WP);
syscfg_sram_page_wp_enable();
```

函数 syscfg_sram_ecc_error_bit_get

函数syscfg_sram_ecc_error_bit_get描述见下表:

表 3-618. 函数 syscfg_sram_ecc_error_bit_get

函数名称	syscfg_sram_ecc_error_bit_get
函数原型	uint32_t syscfg_sram_ecc_error_bit_get(void)
功能描述	获取 SRAM ECC 单比特错误位位置
先决条件	-
被调用函数	-
输入参数 {in}	
-	-
输出参数 {out}	
-	-
返回值	
uint32_t	错误位位置 (0-38, 0 表示无错误)

Example:

```
/* get SRAM ECC single-bit error bit position (API_ID(0x000AU)) */
uint32_t ecc_bit;
ecc_bit = syscfg_sram_ecc_error_bit_get();
```

函数 syscfg_sram_ecc_error_addr_get

函数syscfg_sram_ecc_error_addr_get描述见下表:

表 3-619. 函数 syscfg_sram_ecc_error_addr_get

函数名称	syscfg_sram_ecc_error_addr_get
函数原型	uint32_t syscfg_sram_ecc_error_addr_get(void)
功能描述	获取 SRAM ECC 错误地址
先决条件	-
被调用函数	-
输入参数 {in}	
-	-
输出参数 {out}	

-	-
返回值	
uint32_t	发生 SRAM 上最近一次 ECC 事件的故障系统地址

Example:

```
/* get SRAM ECC error address */
uint32_t addr;
addr = syscfg_sram_ecc_error_addr_get();
```

函数 syscfg_fpu_interrupt_enable

函数syscfg_fpu_interrupt_enable描述见下表:

表 3-620. 函数 syscfg_fpu_interrupt_enable

函数名称	syscfg_fpu_interrupt_enable
函数原型	void syscfg_fpu_interrupt_enable(uint32_t interrupt)
功能描述	使能 FPU 中断
先决条件	-
被调用函数	-
输入参数 {in}	
interrupt	FPU 中断
SYSCFG_FPUINT_INV ALID_OPERATION	无效操作中断使能位
SYSCFG_FPUINT_DIV 0	除以 0 中断使能位
SYSCFG_FPUINT_UN DERFLOW	下溢中断使能位
SYSCFG_FPUINT_OV ERFLOW	上溢中断使能位
SYSCFG_FPUINT_INP UT_ABNORMAL	输入异常中断使能位
SYSCFG_FPUINT_INE XACT	不精确中断使能位
输出参数 {out}	
-	-
返回值	
-	-

Example:

```
/* enable FPU inexact interrupt */
syscfg_fpu_interrupt_enable(SYSCFG_FPUINT_INEXACT);
```

函数 syscfg_fpu_interrupt_disable

函数syscfg_fpu_interrupt_disable描述见下表：

表 3-621. 函数 syscfg_fpu_interrupt_disable

函数名称	syscfg_fpu_interrupt_disable
函数原型	void syscfg_fpu_interrupt_disable(uint32_t interrupt)
功能描述	失能 FPU 中断
先决条件	-
被调用函数	-
输入参数 {in}	
interrupt	FPU 中断
SYSCFG_FPUINT_INV ALID_OPERATION	无效操作中断使能位
SYSCFG_FPUINT_DIV 0	除以 0 中断使能位
SYSCFG_FPUINT_UN DERFLOW	下溢中断使能位
SYSCFG_FPUINT_OV ERFLOW	上溢中断使能位
SYSCFG_FPUINT_INP UT_ABNORMAL	输入异常中断使能位
SYSCFG_FPUINT_INE XACT	不精确中断使能位
输出参数 {out}	
-	-
返回值	
-	-

Example:

```
/* disable FPU inexact interrupt */
syscfg_fpu_interrupt_disable(SYSCFG_FPUINT_INEXACT);
```

函数 syscfg_sram_ecc_interrupt_enable

函数syscfg_sram_ecc_interrupt_enable描述见下表：

表 3-622. 函数 syscfg_sram_ecc_interrupt_enable

函数名称	syscfg_sram_ecc_interrupt_enable
函数原型	void syscfg_sram_ecc_interrupt_enable(uint32_t sram_ecc)
功能描述	使能 SRAM ECC 错误中断
先决条件	-
被调用函数	-
输入参数 {in}	

sram_ecc	SRAM ECC 错误
SYSCFG_SRAM_ECCC S_SRAM_ECCMEIE	使能 SRAM 多比特不可纠正中断
SYSCFG_SRAM_ECCC S_SRAM_ECCSEIE	使能 SRAM 单比特纠正中断
输出参数 {out}	
-	-
返回值	
-	-

Example:

```
/* enable SRAM ECC error interrupt */
```

```
syscfg_sram_ecc_interrupt_enable(SYSCFG_SRAM_ECCCS_SRAM_ECCMEIE);
```

函数 syscfg_sram_ecc_interrupt_disable

函数syscfg_sram_ecc_interrupt_disable描述见下表:

表 3-623. 函数 syscfg_sram_ecc_interrupt_disable

函数名称	syscfg_sram_ecc_interrupt_disable
函数原型	void syscfg_sram_ecc_interrupt_disable(uint32_t sram_ecc)
功能描述	失能 SRAM ECC 错误中断
先决条件	-
被调用函数	-
输入参数 {in}	
sram_ecc	SRAM ECC 错误
SYSCFG_SRAM_ECCC S_SRAM_ECCMEIE	使能 SRAM 多比特非纠正中断
SYSCFG_SRAM_ECCC S_SRAM_ECCSEIE	使能 SRAM 单比特纠正中断
输出参数 {out}	
-	-
返回值	
-	-

Example:

```
/* disable SRAM ECC error interrupt */
```

```
syscfg_sram_ecc_interrupt_disable(SYSCFG_SRAM_ECCCS_SRAM_ECCMEIE);
```

函数 syscfg_bus_timeout_flag_get

函数syscfg_bus_timeout_flag_get描述见下表:

表 3-624. 函数 syscfg_bus_timeout_flag_get

函数名称	syscfg_bus_timeout_flag_get
函数原型	FlagStatus syscfg_bus_timeout_flag_get(uint32_t bustoflag)
功能描述	获取总线超时标志
先决条件	-
被调用函数	-
输入参数 {in}	
bustoflag	总线超时标志
SYSCFG_BUSTOSTAT_CPUCBUSTOF	CPU Cbus 超时标志
SYSCFG_BUSTOSTAT_CPUSBUSTOF	CPU Sbus 超时标志
SYSCFG_BUSTOSTAT_DMA0BUSTOF	DMA0 总线超时标志
SYSCFG_BUSTOSTAT_DMA1BUSTOF	DMA1 总线超时标志
输出参数 {out}	
-	-
返回值	
FlagStatus	置位或复位

Example:

```
/* get bus timeout flag */
```

```
syscfg_bus_timeout_flag_get(CPU Cbus timeout flag);
```

函数 syscfg_bus_timeout_flag_clear

函数syscfg_bus_timeout_flag_clear描述见下表:

表 3-625. 函数 syscfg_bus_timeout_flag_clear

函数名称	syscfg_bus_timeout_flag_clear
函数原型	void syscfg_bus_timeout_flag_clear(uint32_t bustoflag)
功能描述	清除总线超时标志
先决条件	-
被调用函数	-
输入参数 {in}	
bustoflag	总线超时标志
SYSCFG_BUSTOSTAT_CPUCBUSTOF	CPU Cbus 超时标志
SYSCFG_BUSTOSTAT_CPUSBUSTOF	CPU Sbus 超时标志
SYSCFG_BUSTOSTAT_DMA0BUSTOF	DMA0 总线超时标志
SYSCFG_BUSTOSTAT_DMA1BUSTOF	DMA1 总线超时标志

<code>_DMA1BUSTOF</code>	
输出参数 {out}	
-	-
返回值	
-	-

Example:

```
/* clear bus timeout flag */
```

```
syscfg_bus_timeout_flag_clear(SYSCFG_BUSTOFLAG_CPUCBUSTOF);
```

函数 `syscfg_sram_ecc_flag_get`

函数 `syscfg_sram_ecc_flag_get` 描述见下表:

表 3-626. 函数 `syscfg_sram_ecc_flag_get`

函数名称	<code>syscfg_sram_ecc_flag_get</code>
函数原型	<code>FlagStatus syscfg_sram_ecc_flag_get(uint32_t ecc_flag)</code>
功能描述	获取 SRAM ECC 错误标志
先决条件	-
被调用函数	-
输入参数 {in}	
<code>ecc_flag</code>	SRAM ECC 错误标志
<code>SYSCFG_SRAM_ECC_STAT_SRAMECCMEIF</code>	检测到 SRAM 非校正事件
<code>SYSCFG_SRAM_ECC_STAT_SRAMECCSEIF</code>	检测到 SRAM 单比特校正事件
输出参数 {out}	
-	-
返回值	
<code>FlagStatus</code>	置位或复位

Example:

```
/* get SRAM ECC error flag */
```

```
syscfg_sram_ecc_flag_get(SYSCFG_SRAM_ECC_STAT_SRAMECCMEIF);
```

函数 `syscfg_sram_ecc_flag_clear`

函数 `syscfg_sram_ecc_flag_clear` 描述见下表:

表 3-627. 函数 `syscfg_sram_ecc_flag_clear`

函数名称	<code>syscfg_sram_ecc_flag_clear</code>
函数原型	<code>void syscfg_sram_ecc_flag_clear(uint32_t ecc_flag)</code>
功能描述	清除 SRAM ECC 错误标志
先决条件	-

被调用函数	-
输入参数 {in}	
ecc_flag	SRAM ECC 错误标志
SYSCFG_SRAM_ECCS TAT_SRAM_ECCMEIF	检测到 SRAM 非校正事件
SYSCFG_SRAM_ECCS TAT_SRAM_ECCSEIF	检测到 SRAM 单比特校正事件
输出参数 {out}	
-	-
返回值	
-	-

Example:

```
/* clear SRAM ECC error flag */
```

```
syscfg_sram_ecc_flag_clear(SYSCFG_SRAM_ECCSTAT_SRAM_ECCMEIF);
```

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 3-628. SPI/I2S 寄存器

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

3.24.2. 外设库函数说明

SPI/I2S库函数列表如下表所示：

表 3-629. SPI/I2S 库函数

库函数名称	库函数描述
spi_i2s_deinit	复位外设SPIx/I2Sx
spi_struct_para_init	将SPI结构体中所有参数初始化为默认值
spi_init	初始化外设SPIx
spi_enable	使能外设SPIx
spi_disable	失能外设SPIx
i2s_init	初始化外设I2Sx
i2s_psc_config	配置I2Sx预分频器
i2s_enable	使能外设I2Sx
i2s_disable	失能外设I2Sx
spi_nss_output_enable	使能外设SPIx NSS输出
spi_nss_output_disable	失能外设SPIx NSS输出
spi_nss_internal_high	NSS软件模式下NSS引脚拉高
spi_nss_internal_low	NSS软件模式下NSS引脚拉低
spi_dma_enable	使能外设SPIx的DMA功能
spi_dma_disable	失能外设SPIx的DMA功能
spi_i2s_data_frame_format_config	配置外设SPIx/I2Sx数据帧格式
spi_i2s_data_transmit	发送数据
spi_i2s_data_receive	接收数据
spi_bidirectional_transfer_config	配置外设SPIx的数据传输方向
spi_i2s_format_error_clear	清除SPI/I2S帧格式错误标志
spi_crc_polynomial_set	设置外设SPIx的CRC多项式值
spi_crc_polynomial_get	获取外设SPIx的CRC多项式值
spi_crc_on	打开外设SPIx的CRC功能
spi_crc_off	关闭外设SPIx的CRC功能
spi_crc_next	设置外设SPIx下一次传输数据为CRC值
spi_crc_get	外设SPIx获取CRC值
spi_crc_error_clear	清除SPIx CRC错误标志状态
spi_ti_mode_enable	使能SPI TI模式
spi_ti_mode_disable	禁能SPI TI模式
spi_nssp_mode_enable	使能SPI NSS脉冲模式
spi_nssp_mode_disable	禁能SPI NSS脉冲模式
spi_quad_enable	使能四线SPI模式
spi_quad_disable	禁能四线SPI模式
spi_quad_write_enable	使能四线SPI写
spi_quad_read_enable	使能四线SPI读
spi_i2s_flag_get	获取外设SPIx/I2Sx标志状态
spi_i2s_interrupt_enable	使能外设SPIx/I2Sx中断
spi_i2s_interrupt_disable	失能外设SPIx/I2Sx中断
spi_i2s_interrupt_flag_get	获取外设SPIx/I2Sx中断状态

结构体 spi_parameter_struct

表 3-630. 结构体 spi_parameter_struct

成员名称	功能描述
device_mode	配置SPI为主机或从机模式 (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	配置NSS由软件或硬件控制 (SPI_NSS_SOFT, SPI_NSS_HARD)
endian	大端或小端模式配置 (SPI_ENDIAN_MSB, SPI_ENDIAN_LSB)
clock_polarity_phase	相位和极性配置 (SPI_CLOCK_POLARITY_LOW_PHASE_1EDGE, SPI_CLOCK_POLARITY_HIGH_PHASE_1EDGE, SPI_CLOCK_POLARITY_LOW_PHASE_2EDGE, SPI_CLOCK_POLARITY_HIGH_PHASE_2EDGE)
prescale	预分频器配置 (SPI_PSC_n (n=2,4,8,16,32,64,128,256))

枚举类型 spi_interrupt_flag_enum

表 3-631. 枚举类型 spi_interrupt_flag_enum

成员名称	功能描述
SPI_I2S_INT_FLAG_TBE	发送缓冲区空中断标志
SPI_I2S_INT_FLAG_RBNE	接收缓冲区非空中断标志
SPI_I2S_INT_FLAG_RXORERR	接收溢出错误中断标志
SPI_INT_FLAG_CONFERR	配置错误中断标志
SPI_INT_FLAG_CRCERR	CRC错误中断标志
I2S_INT_FLAG_TXURERR	发送欠载错误中断标志
SPI_I2S_INT_FLAG_FERR	帧格式错误中断标志

函数 spi_i2s_deinit

函数spi_i2s_deinit描述见下表:

表 3-632. 函数 spi_i2s_deinit

函数名称	spi_i2s_deinit
函数原形	void spi_i2s_deinit(uint32_t spi_periph);
功能描述	复位外设SPIx/I2Sx
先决条件	-
被调用函数	rcu_periph_reset_enable / rcu_periph_reset_disable
输入参数{in}	
spi_periph	外设SPIx
SPIx	x=0,1,2
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* reset SPI0 */
spi_i2s_deinit(SPI0);
```

函数 spi_struct_para_init

函数spi_struct_para_init描述见下表：

表 3-633. 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结构体中所有参数初始化为默认值
先决条件	-
被调用函数	-
输入参数{in}	
-	-
输出参数{out}	
spi_struct	SPI初始化结构体，结构体成员参考 表 3-630. 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 3-634. 函数 spi_init

函数名称	spi_init
函数原形	void spi_init(uint32_t spi_periph, spi_parameter_struct* spi_struct);
功能描述	初始化外设SPIx
先决条件	-
被调用函数	-
输入参数{in}	
spi_periph	外设SPIx
SPIx	x=0,1,2
输入参数{in}	
spi_struct	初始化结构体，结构体成员参考 表 3-630. 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_PRESCALE_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 3-635. 函数 spi_enable

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

<i>SPIx</i>	<i>x</i> =0,1,2
输出参数{out}	
-	-
返回值	
-	-

例如:

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

```
spi_enable(SPI0);
```

函数 **spi_disable**

函数spi_disable描述见下表:

表 3-636. 函数 spi_disable

函数名称	spi_disable
函数原形	void spi_disable(uint32_t spi_periph);
功能描述	禁能外设SPIx
先决条件	-
被调用函数	-
输入参数{in}	
spi_periph	外设SPIx
<i>SPIx</i>	<i>x</i> =0,1,2
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* disable SPI0 */
```

```
spi_disable(SPI0);
```

函数 **i2s_init**

函数i2s_init描述见下表:

表 3-637. 函数 i2s_init

函数名称	i2s_init
函数原形	void i2s_init(uint32_t spi_periph, uint32_t i2s_mode, uint32_t i2s_standard, uint32_t i2s_ckpl);
功能描述	初始化外设I2Sx
先决条件	-
被调用函数	-
输入参数{in}	

spl_periph	外设SPIx
<i>SPIx</i>	x=1,2
输入参数{in}	
i2s_mode	I2S运行模式
<i>I2S_MODE_SLAVE_TX</i>	I2S从机发送模式
<i>I2S_MODE_SLAVE_RX</i>	I2S从机接收模式
<i>I2S_MODE_MASTERTX</i>	I2S主机发送模式
<i>I2S_MODE_MASTERRX</i>	I2S主机接收模式
输入参数{in}	
i2s_standard	I2S标准选择
<i>I2S_STD_PHILIPS</i>	I2S飞利浦标准
<i>I2S_STD_MSB</i>	I2S MSB对齐标准
<i>I2S_STD_LSB</i>	I2S LSB对齐标准
<i>I2S_STD_PCMSHORT</i>	I2S PCM短帧标准
<i>I2S_STD_PCMLONG</i>	I2S PCM长帧标准
输入参数{in}	
i2s_ckpl	I2S空闲状态时钟极性
<i>I2S_CKPL_LOW</i>	I2S_CK空闲状态为低电平
<i>I2S_CKPL_HIGH</i>	I2S_CK空闲状态为高电平
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* initialize I2S1 */
```

```
i2s_init(SPI1, I2S_MODE_MASTERTX, I2S_STD_PHILIPS, I2S_CKPL_LOW);
```

函数 i2s_psc_config

函数i2s_psc_config描述见下表：

表 3-638. 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 i2s_audiosample, uint32_t i2s_frameformat, uint32_t i2s_mckout);
功能描述	配置I2Sx预分频器
先决条件	-

被调用函数	rcu_clock_freq_get
输入参数{in}	
spi_periph	外设SPIx
<i>SPIx</i>	x=1,2
输入参数{in}	
i2s_audiosample	I2S音频采样频率
<i>I2S_AUDIOSAMPL E_8K</i>	音频采样频率为8KHz
<i>I2S_AUDIOSAMPL E_11K</i>	音频采样频率为11KHz
<i>I2S_AUDIOSAMPL E_16K</i>	音频采样频率为16KHz
<i>I2S_AUDIOSAMPL E_22K</i>	音频采样频率为22KHz
<i>I2S_AUDIOSAMPL E_32K</i>	音频采样频率为32KHz
<i>I2S_AUDIOSAMPL E_44K</i>	音频采样频率为44KHz
<i>I2S_AUDIOSAMPL E_48K</i>	音频采样频率为48KHz
<i>I2S_AUDIOSAMPL E_96K</i>	音频采样频率为96KHz
<i>I2S_AUDIOSAMPL E_192K</i>	音频采样频率为192KHz
输入参数{in}	
i2s_frameformat	I2S数据长度和通道长度
<i>I2S_FRAMEFORMA T_DT16B_CH16B</i>	I2S数据长度为16位，通道长度为16位
<i>I2S_FRAMEFORMA T_DT16B_CH32B</i>	I2S数据长度为16位，通道长度为32位
<i>I2S_FRAMEFORMA T_DT24B_CH32B</i>	I2S数据长度为24位，通道长度为32位
<i>I2S_FRAMEFORMA T_DT32B_CH32B</i>	I2S数据长度为32位，通道长度为32位
输入参数{in}	
i2s_mckout	2S_MCK输出使能
<i>I2S_MCKOUT_ENA BLE</i>	I2S_MCK输出使能
<i>I2S_MCKOUT_DIS ABLE</i>	I2S_MCK输出禁止
输出参数{out}	
-	-
返回值	

-	-
---	---

例如:

```
/* configure I2S1 prescaler */
```

```
i2s_psc_config(SPI1, I2S_AUDIOSAMPLE_44K, I2S_FRAMEFORMAT_DT16B_CH16B,  
I2S_MCKOUT_DISABLE);
```

函数 i2s_enable

函数i2s_enable描述见下表:

表 3-639. 函数 i2s_enable

函数名称	i2s_enable
函数原形	void i2s_enable(uint32_t spi_periph);
功能描述	使能外设I2Sx
先决条件	-
被调用函数	-
输入参数{in}	
spi_periph	外设SPIx
SPIx	x=1,2
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* enable I2S1 */
```

```
i2s_enable(SPI1);
```

函数 i2s_disable

函数i2s_disable描述见下表:

表 3-640. 函数 i2s_disable

函数名称	i2s_disable
函数原形	void i2s_disable(uint32_t spi_periph);
功能描述	禁能外设I2Sx
先决条件	-
被调用函数	-
输入参数{in}	
spi_periph	外设SPIxLLL
SPIx	x=1,2
输出参数{out}	
-	-

返回值	
-	-

例如：

```
/* disable I2S1 */
```

```
i2s_disable(SPI1);
```

函数 spi_nss_output_enable

函数spi_nss_output_enable描述见下表：

表 3-641. 函数 spi_nss_output_enable

函数名称	spi_nss_output_enable
函数原形	void spi_nss_output_enable(uint32_t spi_periph);
功能描述	使能外设SPIx NSS输出
先决条件	-
被调用函数	-
输入参数{in}	
spi_periph	外设SPIx
SPIx	x=0,1,2
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* enable SPI0 NSS output */
```

```
spi_nss_output_enable(SPI0);
```

函数 spi_nss_output_disable

函数spi_nss_output_disable描述见下表：

表 3-642. 函数 spi_nss_output_disable

函数名称	spi_nss_output_disable
函数原形	void spi_nss_output_disable(uint32_t spi_periph);
功能描述	禁能外设SPIx NSS输出
先决条件	-
被调用函数	-
输入参数{in}	
spi_periph	外设SPIx
SPIx	x=0,1,2
输出参数{out}	
-	-

返回值	
-	-

例如：

```
/* disable SPI0 NSS output */
spi_nss_output_disable(SPI0);
```

函数 spi_nss_internal_high

函数spi_nss_internal_high描述见下表：

表 3-643. 函数 spi_nss_internal_high

函数名称	spi_nss_internal_high
函数原形	void spi_nss_internal_high(uint32_t spi_periph);
功能描述	NSS软件模式下NSS引脚拉高
先决条件	-
被调用函数	-
输入参数{in}	
spi_periph	外设SPIx
SPIx	x=0,1,2
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* SPI0 NSS pin is pulled high level in software mode */
spi_nss_internal_high(SPI0);
```

函数 spi_nss_internal_low

函数spi_nss_internal_low描述见下表：

表 3-644. 函数 spi_nss_internal_low

函数名称	spi_nss_internal_low
函数原形	void spi_nss_internal_low(uint32_t spi_periph);
功能描述	NSS软件模式下NSS引脚拉低
先决条件	-
被调用函数	-
输入参数{in}	
spi_periph	外设SPIx
SPIx	x=0,1,2
输出参数{out}	
-	-

返回值	
-	-

例如：

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

```
spi_nss_internal_low(SPI0);
```

函数 spi_dma_enable

函数spi_dma_enable描述见下表：

表 3-645. 函数 spi_dma_enable

函数名称	spi_dma_enable
函数原形	void spi_dma_enable(uint32_t spi_periph, uint8_t dma);
功能描述	使能外设SPIx的DMA功能
先决条件	-
被调用函数	-
输入参数{in}	
spi_periph	外设SPIx
SPIx	x=0,1,2
输入参数{in}	
dma	SPI DMA模式
SPI_DMA_TRANSMIT	SPI发送缓冲区DMA使能
SPI_DMA_RECEIVE	SPI接收缓冲区DMA使能
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* enable SPI0 transmit data DMA function */
```

```
spi_dma_enable(SPI0, SPI_DMA_TRANSMIT);
```

函数 spi_dma_disable

函数spi_dma_disable描述见下表：

表 3-646. 函数 spi_dma_disable

函数名称	spi_dma_disable
函数原形	void spi_dma_disable(uint32_t spi_periph, uint8_t dma);
功能描述	禁能外设SPIx的DMA功能
先决条件	-

被调用函数	-
输入参数{in}	
spi_periph	外设SPIx
<i>SPIx</i>	x=0,1,2
输入参数{in}	
dma	SPI DMA模式
<i>SPI_DMA_TRANSMIT</i>	SPI发送缓冲区DMA使能
<i>SPI_DMA_RECEIVE</i>	SPI接收缓冲区DMA使能
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* disable SPI0 transmit data 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 3-647. 函数 spi_i2s_data_frame_format_config

函数名称	spi_i2s_data_frame_format_config
函数原形	void spi_i2s_data_frame_format_config(uint32_t spi_periph, uint16_t frame_format);
功能描述	配置外设SPIx/I2Sx数据帧格式
先决条件	-
被调用函数	-
输入参数{in}	
spi_periph	外设SPIx
<i>SPIx</i>	x=0,1,2
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_i2s_data_transmit

函数spi_i2s_data_transmit描述见下表：

表 3-648. 函数 spi_i2s_data_transmit

函数名称	spi_i2s_data_transmit
函数原形	void spi_i2s_data_transmit(uint32_t spi_periph, uint16_t data);
功能描述	SPI发送数据
先决条件	-
被调用函数	-
输入参数{in}	
spi_periph	外设SPIx
SPIx	x=0,1,2
输入参数{in}	
data	16位数据
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* SPI0 transmit data */
```

```
uint16_t spi0_send_array[] = {0x5050, 0xA0A0};
```

```
spi_i2s_data_transmit(SPI0, spi0_send_array[0]);
```

函数 spi_i2s_data_receive

函数spi_i2s_data_receive描述见下表：

表 3-649. 函数 spi_i2s_data_receive

函数名称	spi_i2s_data_receive
函数原形	uint16_t spi_i2s_data_receive(uint32_t spi_periph);
功能描述	SPI接收数据
先决条件	-
被调用函数	-
输入参数{in}	
spi_periph	外设SPIx
SPIx	x=0,1,2
输出参数{out}	

-	-
返回值	
-	16位数据

例如：

```
/* SPI0 receive data */
```

```
uint16_t spi0_receive_data;
```

```
spi0_receive_data = spi_i2s_data_receive(SPI0);
```

函数 spi_bidirectional_transfer_config

函数spi_bidirectional_transfer_config描述见下表：

表 3-650. 函数 spi_bidirectional_transfer_config

函数名称	spi_bidirectional_transfer_config
函数原形	void spi_bidirectional_transfer_config(uint32_t spi_periph, uint32_t transfer_direction);
功能描述	配置外设SPIx的数据传输方向
先决条件	-
被调用函数	-
输入参数{in}	
spi_periph	外设SPIx
SPIx	x=0,1,2
输入参数{in}	
transfer_direction	SPI双向传输输出使能
SPI_BIDIRECTIONAL_TRANSMIT	SPI工作在只发送模式
SPI_BIDIRECTIONAL_RECEIVE	SPI工作在只接收模式
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* SPI0 works in transmit-only mode */
```

```
spi_bidirectional_transfer_config(SPI0, SPI_BIDIRECTIONAL_TRANSMIT);
```

函数 spi_crc_polynomial_set

函数spi_crc_polynomial_set描述见下表：

表 3-651. 函数 spi_crc_polynomial_set

函数名称	spi_crc_polynomial_set
------	------------------------

函数原形	void spi_crc_polynomial_set(uint32_t spi_periph, uint16_t crc_poly);
功能描述	设置外设SPIx的CRC多项式值
先决条件	-
被调用函数	-
输入参数{in}	
spi_periph	外设SPIx
SPIx	x=0,1,2
输入参数{in}	
crc_poly	CRC多项式值
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* set SPI0 CRC polynomial */
uint16_t CRC_VALUE = 0x5050;
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 3-652. 函数 spi_crc_polynomial_get

函数名称	spi_crc_polynomial_get
函数原形	uint16_t spi_crc_polynomial_get(uint32_t spi_periph);
功能描述	获取外设SPIx的CRC多项式值
先决条件	-
被调用函数	-
输入参数{in}	
spi_periph	外设SPIx
SPIx	x=0,1,2
输出参数{out}	
-	-
返回值	
uint16_t	16位CRC多项式值（0-0xFFFF）

例如：

```
/* get SPI0 CRC polynomial */
uint16_t crc_val;
crc_val = spi_crc_polynomial_get(SPI0);
```

函数 spi_crc_on

函数spi_crc_on描述见下表：

表 3-653. 函数 spi_crc_on

函数名称	spi_crc_on
函数原形	void spi_crc_on(uint32_t spi_periph);
功能描述	打开外设SPIx的CRC功能
先决条件	-
被调用函数	-
输入参数{in}	
spi_periph	外设SPIx
SPIx	x=0,1,2
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* turn on SPI0 CRC function */
```

```
spi_crc_on(SPI0);
```

函数 spi_crc_off

函数spi_crc_off描述见下表：

表 3-654. 函数 spi_crc_off

函数名称	spi_crc_off
函数原形	void spi_crc_off(uint32_t spi_periph);
功能描述	关闭外设SPIx的CRC功能
先决条件	-
被调用函数	-
输入参数{in}	
spi_periph	外设SPIx
SPIx	x=0,1,2
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* turn off SPI0 CRC function */
```

```
spi_crc_off(SPI0);
```

函数 spi_crc_next

函数spi_crc_next描述见下表:

表 3-655. 函数 spi_crc_next

函数名称	spi_crc_next
函数原形	void spi_crc_next(uint32_t spi_periph);
功能描述	设置外设SPIx下一次传输数据为CRC值
先决条件	-
被调用函数	-
输入参数{in}	
spi_periph	外设SPIx
SPIx	x=0,1,2
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* SPI0 next data is CRC value */
```

```
spi_crc_next(SPI0);
```

函数 spi_crc_get

函数spi_crc_get描述见下表:

表 3-656. 函数 spi_crc_get

函数名称	spi_crc_get
函数原形	uint16_t spi_crc_get(uint32_t spi_periph, uint8_t crc);
功能描述	外设SPIx获取CRC值
先决条件	-
被调用函数	-
输入参数{in}	
spi_periph	外设SPIx
SPIx	x=0,1,2
输入参数{in}	
crc	SPI crc值
SPI_CRC_TX	获取发送CRC寄存器值
SPI_CRC_RX	获取接收CRC寄存器值
输出参数{out}	
-	-
返回值	
uint16_t	16位CRC值 (0-0xFFFF)

例如:

```

/* get SPI0 CRC send value */

uint16_t crc_val;

crc_val = spi_crc_get(SPI0, SPI_CRC_TX);

```

函数 spi_crc_error_clear

函数spi_crc_error_clear描述见下表：

表 3-657. 函数 spi_crc_error_clear

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

例如：

```

/* clear SPI0 CRC error flag status */

spi_crc_error_clear(SPI0);

```

函数 spi_ti_mode_enable

函数spi_ti_mode_enable描述见下表：

表 3-658. 函数 spi_ti_mode_enable

函数名称	spi_ti_mode_enable
函数原形	void spi_ti_mode_enable(uint32_t spi_periph);
功能描述	使能SPI TI模式
先决条件	-
被调用函数	-
输入参数{in}	
spi_periph	外设SPIx
SPIx	x=0,1,2
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* enable SPI0 TI mode */
spi_ti_mode_enable(SPI0);
```

函数 spi_ti_mode_disable

函数spi_ti_mode_disable描述见下表：

表 3-659. 函数 spi_ti_mode_disable

函数名称	spi_ti_mode_disable
函数原形	void spi_ti_mode_disable(uint32_t spi_periph);
功能描述	禁能SPI TI模式
先决条件	-
被调用函数	-
输入参数{in}	
spi_periph	外设SPIx
SPIx	x=0,1,2
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* disable SPI0 TI mode */
spi_ti_mode_disable(SPI0);
```

函数 spi_nssp_mode_enable

函数spi_nssp_mode_enable描述见下表：

表 3-660. 函数 spi_nssp_mode_enable

函数名称	spi_nssp_mode_enable
函数原形	void spi_nssp_mode_enable(uint32_t spi_periph);
功能描述	使能SPI NSS脉冲模式
先决条件	-
被调用函数	-
输入参数{in}	
spi_periph	外设SPIx
SPIx	x=0,1,2
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* enable SPI0 NSS pulse mode */
spi_nssp_mode_enable(SPI0);
```

函数 spi_nssp_mode_disable

函数spi_nssp_mode_disable描述见下表:

表 3-661. 函数 spi_nssp_mode_disable

函数名称	spi_nssp_mode_disable
函数原形	void spi_nssp_mode_disable(uint32_t spi_periph);
功能描述	禁能SPI NSS脉冲模式
先决条件	-
被调用函数	-
输入参数{in}	
spi_periph	外设SPIx
SPIx	x=0,1,2
输出参数{out}	
-	-
返回值	
-	-

例如:

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

函数 spi_quad_enable

函数spi_quad_enable描述见下表:

表 3-662. 函数 spi_quad_enable

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

例如:

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

函数 spi_quad_disable

函数spi_quad_disable描述见下表:

表 3-663. 函数 spi_quad_disable

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

例如:

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

函数 spi_quad_write_enable

函数spi_quad_write_enable描述见下表:

表 3-664. 函数 spi_quad_write_enable

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

例如:

```
/* enable SPI0 quad wire write */
spi_quad_write_enable(SPI0);
```

函数 spi_quad_read_enable

函数spi_quad_read_enable描述见下表:

表 3-665. 函数 spi_quad_read_enable

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

例如:

```
/* enable SPI0 quad wire read */
spi_quad_read_enable(SPI0);
```

函数 spi_i2s_flag_get

函数spi_i2s_flag_get描述见下表:

表 3-666. 函数 spi_i2s_flag_get

函数名称	spi_i2s_flag_get
函数原形	FlagStatus spi_i2s_flag_get(uint32_t spi_periph, uint32_t flag);
功能描述	获取外设SPIx/I2Sx标志状态
先决条件	-
被调用函数	-
输入参数{in}	
spi_periph	外设SPIx
SPIx	x=0,1,2
输入参数{in}	
flag	SPI/I2S标志状态
SPI_FLAG_TBE	发送缓冲区空标志
SPI_FLAG_RBNE	接收缓冲区非空标志
SPI_FLAG_TRANS	通信进行中标志

<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>SPI_FLAG_FERR</code>	帧错误标志
<code>I2S_FLAG_TBE</code>	发送缓冲区空标志
<code>I2S_FLAG_RBNE</code>	接收缓冲区非空标志
<code>I2S_FLAG_TRANS</code>	通信进行中标志
<code>I2S_FLAG_RXORERR</code>	接收过载错误标志
<code>I2S_FLAG_TXURERR</code>	发送欠载错误标志
<code>I2S_FLAG_CH</code>	通道标志
<code>I2S_FLAG_FERR</code>	帧错误标志
输出参数{out}	
-	-
返回值	
FlagStatus	SET 或 RESET

例如:

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

函数 spi_i2s_interrupt_enable

函数spi_i2s_interrupt_enable描述见下表:

表 3-667. 函数 spi_i2s_interrupt_enable

函数名称	spi_i2s_interrupt_enable
函数原形	void spi_i2s_interrupt_enable(uint32_t spi_periph, uint8_t interrupt);
功能描述	使能外设SPIx/I2Sx中断
先决条件	-
被调用函数	-
输入参数{in}	
spi_periph	外设SPIx
SPIx	x=0,1,2
输入参数{in}	
interrupt	SPI/I2S中断
<code>SPI_I2S_INT_TBE</code>	发送缓冲区空中断使能
<code>SPI_I2S_INT_RBNE</code>	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 3-668. 函数 spi_i2s_interrupt_disable

函数名称	spi_i2s_interrupt_disable
函数原形	void spi_i2s_interrupt_disable(uint32_t spi_periph, uint8_t interrupt);
功能描述	禁能外设SPIx/I2Sx中断
先决条件	-
被调用函数	-
输入参数{in}	
spi_periph	外设SPIx
<i>SPIx</i>	x=0,1,2
输入参数{in}	
interrupt	SPI/I2S中断
<i>SPI_I2S_INT_TBE</i>	发送缓冲区空中断使能
<i>SPI_I2S_INT_RBNE</i>	接收缓冲区非空中断使能
<i>SPI_I2S_INT_ERR</i>	错误中断使能
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* 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 3-669. 函数 spi_i2s_interrupt_flag_get

函数名称	spi_i2s_interrupt_flag_get
函数原形	FlagStatus spi_i2s_interrupt_flag_get(uint32_t spi_periph, uint8_t interrupt);

功能描述	获取外设SPIx/I2Sx中断状态
先决条件	-
被调用函数	-
输入参数{in}	
spi_periph	外设SPIx
SPIx	x=0,1,2
输入参数{in}	
interrupt	SPI/I2S中断状态
SPI_I2S_INT_FLAG_TBE	发送缓冲区空中断
SPI_I2S_INT_FLAG_RBNE	接收缓冲区非空中断
SPI_I2S_INT_FLAG_RXORERR	接收过载错误中断
SPI_INT_FLAG_COFERR	配置错误中断
SPI_INT_FLAG_CRCERR	CRC错误中断
I2S_INT_FLAG_TXURERR	发送欠载错误中断
SPI_I2S_INT_FLAG_FERR	帧错误中断
输出参数{out}	
-	-
返回值	
FlagStatus	SET 或 RESET

例如：

```

/* get SPI0 transmit buffer empty interrupt status */
if(RESET != spi_i2s_interrupt_flag_get(SPI0, SPI_I2S_INT_FLAG_TBE)){
    while(RESET == spi_i2s_flag_get(SPI0, SPI_FLAG_TBE));
    spi_i2s_data_transmit(SPI0, spi0_send_array[send_n++]);
}

```

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，分为四种类型：高级定时器（TIMERx, x=0, 7），通用定时器L0（TIMERx, x=1~4），通用定时器L3（TIMERx, x=15, 16），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 3-670. 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_CREP0	重复计数寄存器0
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_CCHP0	互补通道保护寄存器0
TIMER_MCHCTL0	TIMER多模式通道控制寄存器0
TIMER_MCHCTL2	TIMER多模式通道控制寄存器2
TIMER_MCH0CV	TIMER多模式通道0比较/捕获寄存器
TIMER_MCH2CV	TIMER多模式通道2比较/捕获寄存器
TIMER_MCH3CV	TIMER多模式通道3比较/捕获寄存器
TIMER_CH0COMV_ADD	TIMER通道0附加比较寄存器
TIMER_CH1COMV_ADD	TIMER通道1附加比较寄存器
TIMER_CH2COMV_ADD	TIMER通道2附加比较寄存器
TIMER_CH3COMV_ADD	TIMER通道3附加比较寄存器
TIMER_CTL2	TIMER控制寄存器2
TIMER_AFCTL0	备用功能控制寄存器0
TIMER_CCHP1	互补通道保护寄存器1
TIMER_WDGER	看门狗计数器周期寄存器
TIMER_CINITCTL	TIMER计数器初始控制寄存器
TIMER_CINITV	TIMER计数器初始值寄存器
TIMER_CHBRKCTL	TIMER通道中止控制寄存器
TIMER_CHBRKPER	TIMER通道中止周期寄存器
TIMER_CHBRKINTF	TIMER通道中止中断标志寄存器
TIMER_DMACFG	DMA配置寄存器

寄存器名称	寄存器描述
TIMER_DMATB	DMA发送缓冲区寄存器
TIMER_CFG	配置寄存器

3.25.2. 外设库函数说明

TIMER库函数列表如下表所示：

表 3-671. 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	将TIMER初始化结构体中所有参数初始化为默认值
timer_init	初始化外设TIMER
timer_enable	使能外设TIMER
timer_disable	禁能外设TIMER
timer_auto_reload_shadow_enable	TIMER自动重载影子使能
timer_auto_reload_shadow_disable	TIMER自动重载影子禁能
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_runtime_repetition_value_read	配置外设TIMER实时重复计数器值
timer_autoreload_value_config	配置外设TIMER的自动重载寄存器
timer_autoreload_value_read	读取TIMER自动重载寄存器计数器值
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	将TIMER中止功能参数结构体中所有参数初始化为默认值
timer_break_config	配置中止功能
timer_break_enable	使能TIMER的中止功能
timer_break_disable	禁能TIMER的中止功能
timer_automatic_output_enable	自动输出使能

库函数名称	库函数描述
timer_automatic_output_disable	自动输出禁能
timer_primary_output_config	所有的通道输出使能
timer_channel_control_shadow_config	配置通道控制影子寄存器
timer_channel_control_shadow_update_config	配置TIMER通道控制影子寄存器更新控制
timer_channel_output_struct_para_init	将TIMER通道输出参数结构体中所有参数初始化为默认值
timer_channel_output_config	外设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
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_compare_fast_config	配置TIMER的通道输出比较快速输出功能
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_composite_asymmetric_pwm_level_config	复合PWM模式或者非对称PWM模式下配置TIMER通道周期点匹配时OxCPRE电平
timer_channel_output_clear_invalid_selection	输出清除无效事件选择
timer_channel_input_struct_para_init	将TIMER通道输入参数结构体中所有参数初始化为默认值
timer_input_capture_config	配置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	读取通道输入捕获值
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_multi_mode_channel_output_parameter_struct_init	将TIMER多模式通道输出参数结构体中所有参数初始化为默认值
timer_multi_mode_channel_output_config	外设TIMER的多模式通道输出配置
timer_multi_mode_channel_mode_config	外设TIMER多模式通道模式选择

库函数名称	库函数描述
nfig	
timer_input_trigger_source_select	TIMER的输入触发源选择
timer_master_output_trigger_source_select	选择TIMER主模式输出触发源
timer_slave_mode_select	TIMER从模式配置
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_non_quadrature_decoder_mode_config	TIMER配置为非正交译码器模式
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，ETI作为时钟源
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_write_chxval_register_config	配置TIMER写CHxVAL选择位
timer_output_value_selection_config	配置TIMER输出值选择位
timer_commutation_control_shadow_register_config	配置换相控制影子寄存器更新选择
timer_output_match_pulse_select	配置TIMER输出匹配脉冲选择
timer_channel_composite_pwm_mode_config	配置TIMER的复合PWM模式
timer_channel_composite_pwm_mode_output_pulse_value_config	配置TIMER的复合PWM模式输出脉冲值
timer_channel_asymmetric_pwm_pulse_config	配置外设TIMERx的非对称PWM模式输出比较值
timer_channel_additional_compare_value_config	配置TIMER通道附加比较寄存器值
timer_channel_additional_output_shadow_config	配置TIMER通道附加输出比较影子寄存器功能
timer_channel_additional_output_update_select	选择TIMER通道附加输出比较寄存器更新源
timer_channel_additional_compare_value_read	读取TIMER通道附加输出比较寄存器值
timer_break_external_source_config	配置TIMER中止功能外部输入源
timer_break_external_polarity_config	配置TIMER中止功能输入极性

库函数名称	库函数描述
timer_dead_time_falling_edge_config	配置TIMER下降沿死区时间
timer_dead_time_different_config	配置TIMER的不同死区时间功能
timer_channel_break_control_config	配置TIMER通道的中止功能
timer_channel_dead_time_config	配置TIMER通道的死区功能
timer_channel_break_config	配置TIMER通道中止
timer_channel_break_external_status_config	配置TIMER通道中止外部输入使能
timer_channel_break_external_polarity_config	配置TIMER通道中止外部输入极性
timer_channel_primary_output_config	配置TIMER通道主输出功能
timer_channel_break_period_config	配置TIMER通道中止周期值
timer_watchdog_value_config	正交编码器信号断线检测看门狗计数器值配置
timer_watchdog_value_read	读取正交编码器信号断线检测看门狗计数器值
timer_decoder_disconnection_detection_config	正交编码器信号断线检测功能配置
timer_decoder_jump_detection_config	正交编码器信号信号跳变检测功能配置
timer_counter_initial_register_config	配置计数器初始值寄存器
timer_counter_initial_config	配置计数器的初始值和计数方向
timer_synchronization_event_generate	产生软件同步事件
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 3-672. 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 (0~0xFFFF(TIMERx(x=0,2~7,15,16)), 0~0xFFFFFFFF(TIMERx(x=1)))
clockdivision	时钟分频因子 (TIMER_CKDIV_DIV1, TIMER_CKDIV_DIV2, TIMER_CKDIV_DIV4)
repetitioncounter	重复计数器值 (0~0xFF)

结构体 timer_break_parameter_struct

表 3-673. 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）
outputautostate	自动输出使能（TIMER_OUTAUTO_ENABLE, TIMER_OUTAUTO_DISABLE）
protectmode	互补寄存器保护控制（TIMER_CCHP0_PROT_OFF, TIMER_CCHP0_PROT_0, TIMER_CCHP0_PROT_1, TIMER_CCHP0_PROT_2）
break0state	BREAK0输入使能（TIMER_BREAK0_ENABLE, TIMER_BREAK0_DISABLE）
break0filter	BREAK0输入滤波（0~15）
break0polarity	BREAK0输入极性（TIMER_BREAK0_POLARITY_LOW, TIMER_BREAK0_POLARITY_HIGH）

结构体 timer_oc_parameter_struct

表 3-674. 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_omc_parameter_struct

表 3-675. 结构体 timer_omc_parameter_struct

成员名称	功能描述
outputmode	多模式通道输出模式选择（TIMER_MCH_MODE_INDEPENDENTLY, TIMER_MCH_MODE_COMPLEMENTARY）
outputstate	多模式通道输出状态（TIMER_MCCX_ENABLE, TIMER_MCCX_DISABLE）
ocpolarity	多模式通道输出极性（TIMER_OMC_POLARITY_HIGH, TIMER_OMC_POLARITY_LOW）

结构体 `timer_ic_parameter_struct`表 3-676. 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, TIMER_IC_SELECTION_PAIR)
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 3-677. 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=0~7, 15, 16)	TIMER外设选择
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* reset TIMER0 */
```

```
timer_deinit (TIMER0);
```

函数 `timer_struct_para_init`

函数 `timer_struct_para_init` 描述见下表:

表 3-678. 函数 `timer_struct_para_init`

函数名称	timer_struct_para_init
函数原型	void timer_struct_para_init(timer_parameter_struct* initpara);
功能描述	将TIMER初始化参数结构体中所有参数初始化为默认值
先决条件	-

被调用函数	-
输入参数{in}	
initpara	TIMER初始化结构体，结构体成员参考 结构体timer_parameter_struct 。
输出参数{out}	
-	-
返回值	
-	-

例如：

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

```
timer_parameter_struct timer_initpara;
```

```
timer_struct_para_init(&timer_initpara);
```

函数 timer_init

函数timer_init描述见下表：

表 3-679. 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=0~7, 15, 16)	TIMER外设选择
输入参数{in}	
initpara	TIMER初始化结构体，结构体成员参考 结构体timer_parameter_struct 。
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* initialize TIMER0 */
```

```
timer_parameter_struct timer_initpara;
```

```
timer_initpara.prescaler = 107;
```

```
timer_initpara.alignedmode = TIMER_COUNTER_EDGE;
```

```
timer_initpara.counterdirection = TIMER_COUNTER_UP;
```

```
timer_initpara.period = 999;
```

```
timer_initpara.clockdivision    = TIMER_CKDIV_DIV1;
```

```
timer_initpara.repetitioncounter = 1;
```

```
timer_init(TIMER0, &timer_initpara);
```

函数 timer_enable

函数timer_enable描述见下表:

表 3-680. 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=0~7, 15, 16)	TIMER外设选择
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* enable TIMER0 */
```

```
timer_enable(TIMER0);
```

函数 timer_disable

函数timer_disable描述见下表:

表 3-681. 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=0~7, 15, 16)	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 3-682. 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=0~7, 15, 16)	TIMER外设选择
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* enable the TIMER0 auto reload shadow function */
```

```
timer_auto_reload_shadow_enable(TIMER0);
```

函数 timer_auto_reload_shadow_disable

函数timer_auto_reload_shadow_disable描述见下表:

表 3-683. 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=0~7, 15, 16)	TIMER外设选择
输出参数{out}	

-	-
返回值	
-	-

例如：

```
/* disable the TIMER0 auto reload shadow function */
```

```
timer_auto_reload_shadow_disable(TIMER0);
```

函数 timer_update_event_enable

函数timer_update_event_enable描述见下表：

表 3-684. 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=0~7, 15, 16)	TIMER外设选择
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* enable TIMER0 the update event */
```

```
timer_update_event_enable(TIMER0);
```

函数 timer_update_event_disable

函数timer_update_event_disable描述见下表：

表 3-685. 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=0~7, 15, 16)	TIMER外设选择

6)	
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 3-686. 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=0~4,7,15,16)	TIMER外设选择
输入参数{in}	
aligned	对齐模式
TIMER_COUNTER_EDGE	边沿对齐模式
TIMER_COUNTER_CENTER_DOWN	中央对齐向下计数模式
TIMER_COUNTER_CENTER_UP	中央对齐向上计数模式
TIMER_COUNTER_CENTER_BOTH	中央对齐上下计数模式
输出参数{out}	
-	-
返回值	
-	-

例如：

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

```
timer_counter_alignment(TIMER0, TIMER_COUNTER_CENTER_UP);
```

函数 timer_counter_up_direction

函数timer_counter_up_direction描述见下表：

表 3-687. 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=0~4,7,15,16)	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 3-688. 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=0~4,7,15,16)	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 3-689. 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, uint32_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=0~7, 15, 16)	TIMER外设选择
输入参数{in}	
prescaler	预分频值, 0~0xFFFF
输入参数{in}	
pscreload	预分频值加载模式
TIMER_PSC_RELOAD_NOW	预分频值立即加载
TIMER_PSC_RELOAD_UPDATE	预分频值在下次更新事件发生时加载
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* configure TIMER0 prescaler */
```

```
timer_prescaler_config(TIMER0, 3000, TIMER_PSC_RELOAD_NOW);
```

函数 timer_repetition_value_config

函数timer_repetition_value_config描述见下表：

表 3-690. 函数 timer_repetition_value_config

函数名称	timer_repetition_value_config
函数原型	void timer_repetition_value_config(uint32_t timer_periph, uint16_t repetition)
功能描述	配置外设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
<i>TIMERx</i> (<i>x</i> =0~7, 15, 16)	TIMER外设选择
输入参数{in}	
repetition	重复计数器值，取值范围（0~0xFF）
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* configure TIMER0 repetition register 0 value */
```

```
timer_repetition_value_config(TIMER0, 98);
```

函数 timer_runtime_repetition_value_read

函数timer_runtime_repetition_value_read描述见下表：

表 3-691. 函数 timer_runtime_repetition_value_read

函数名称	timer_runtime_repetition_value_read
函数原型	uint32_t timer_runtime_repetition_value_read(uint32_t timer_periph);
功能描述	读取外设TIMER的实时重复计数器值
先决条件	-
被调用函数	-
输入参数{in}	
timer_periph	TIMER外设
<i>TIMERx</i> (<i>x</i> =0, 7, 15, 16)	TIMER外设选择
输出参数{out}	
-	-
返回值	
uint32_t	TIMER_CREP0寄存器的计数重复值（0~0xFF）

例如：

```
/* read TIMER0 runtime repetition register value */
```

```
uint32_t i = 0;
```

```
i = timer_runtime_repetition_value_read(TIMER0);
```

函数 timer_autoreload_value_config

函数timer_autoreload_value_config描述见下表：

表 3-692. 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, uint64_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=0~7, 15, 16)	TIMER外设选择
输入参数{in}	
autoreload	计数器自动重载值 0~0xFFFF, TIMERx(x=0,2,3,4,7,15,16) 0~0xFFFFFFFF, TIMERx(x=1)
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* configure TIMER autoreload register value */
timer_autoreload_value_config(TIMER0, 3000);
```

函数 timer_autoreload_value_read

函数timer_autoreload_value_read描述见下表:

表 3-693. 函数 timer_autoreload_value_read

函数名称	timer_autoreload_value_read
函数原型	uint32_t timer_autoreload_value_read(uint32_t timer_periph);
功能描述	读取外设TIMER自动重载寄存器值
先决条件	-
被调用函数	-
输入参数{in}	
timer_periph	TIMER外设
TIMERx(x=0~7, 15, 16)	TIMER外设选择
输出参数{out}	
-	-
返回值	
uint32_t	计数器自动重载值 0~0xFFFF, TIMERx(x=0,2,3,4,7,15,16) 0~0xFFFFFFFF, TIMERx(x=1)

例如:

```
/* get TIMER autoreload register value */
```

```
uint32_t i = 0;
```

```
i=(uint32_t) timer_autoreload_value_read (TIMER0);
```

函数 timer_counter_value_config

函数timer_counter_value_config描述见下表:

表 3-694. 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, uint64_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=0~7, 15, 16)	TIMER外设选择
输入参数{in}	
counter	计数器值 0~0xFFFF, TIMERx(x=0,2,3,4,7,15,16) 0~0xFFFFFFFF, TIMERx(x=1)
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* configure TIMER0 counter register value */
```

```
timer_counter_value_config(TIMER0, 999);
```

函数 timer_counter_read

函数timer_counter_read描述见下表:

表 3-695. 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=0~7, 15, 16)	TIMER外设选择

输出参数{out}	
-	-
返回值	
uint32_t	外设TIMER的计数器值 0~0xFFFF, TIMERx(x=0,2,3,4,7,15,16) 0~0xFFFFFFFF, TIMERx(x=1)

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 3-696. 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
TIMERx(x=0~7, 15, 16)	TIMER外设选择
输出参数{out}	
-	-
返回值	
uint16_t	外设TIMER的预分频器值 (0~0xFFFF)

例如:

```
/* read TIMER0 prescaler value */
uint16_t i = 0;
i = timer_prescaler_read(TIMER0);
```

函数 timer_single_pulse_mode_config

函数timer_single_pulse_mode_config描述见下表:

表 3-697. 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, uint32_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
TIMERx(x=0~7, 15, 16)	TIMER外设选择
输入参数{in}	
spmode	脉冲模式
TIMER_SP_MODE_SINGLE	单脉冲模式计数
TIMER_SP_MODE_REPETITIVE	重复模式计数
输出参数{out}	
-	-
返回值	
-	-

例如：

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

```
timer_single_pulse_mode_config(TIMER0, TIMER_SP_MODE_SINGLE);
```

函数 timer_update_source_config

函数timer_update_source_config描述见下表：

表 3-698. 函数 timer_update_source_config

函数名称	timer_update_source_config
函数原型	void timer_update_source_config(uint32_t timer_periph, uint32_t update);
功能描述	配置外设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=0~7, 15, 16)	TIMER外设选择
输入参数{in}	
update	更新源
TIMER_UPDATE_SRC_GLOBAL	下述任一事件产生更新中断或DMA请求： – UPG位被置1 – 计数器溢出/下溢 – 从模式控制器产生的更新
TIMER_UPDATE_SRC_REGULAR	只有计数器溢出/ 下溢才产生更新中断或DMA请求

输出参数{out}	
-	-
返回值	
-	-

例如:

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

```
timer_update_source_config(TIMER0, TIMER_UPDATE_SRC_REGULAR);
```

函数 timer_dma_enable

函数timer_dma_enable描述见下表:

表 3-699. 函数 timer_dma_enable

函数名称	timer_dma_enable
函数原型	void timer_dma_enable(uint32_t timer_periph, uint32_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(x=0~7, 15, 16)	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, 15, 16)
TIMER_DMA_CH0D	通道0比较/捕获DMA请求, TIMERx (x=0~4, 7, 15, 16)
TIMER_DMA_CH1D	通道1比较/捕获DMA请求, TIMERx(x=0~4, 7, 15, 16)
TIMER_DMA_CH2D	通道2比较/捕获DMA请求, TIMERx (x=0~4, 7)
TIMER_DMA_CH3D	通道3比较/捕获DMA请求, TIMERx (x=0~4, 7)
TIMER_DMA_CMTD	换相DMA更新请求, TIMERx (x=0, 7, 15, 16)
TIMER_DMA_TRGD	触发DMA请求, TIMERx (x=0, 7, 15, 16)
TIMER_DMA_MCH0D	多模式通道0比较/捕获DMA请求, TIMERx (x=15, 16)
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 3-700. 函数 timer_dma_disable

函数名称	timer_dma_disable
函数原型	void timer_dma_disable (uint32_t timer_periph, uint32_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(x=0~7, 15, 16)	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, 15, 16)
TIMER_DMA_CH0D	通道0比较/捕获DMA请求，TIMERx (x=0~4, 7, 15, 16)
TIMER_DMA_CH1D	通道1比较/捕获DMA请求，TIMERx(x=0~4, 7, 15, 16)
TIMER_DMA_CH2D	通道2比较/捕获DMA请求，TIMERx (x=0~4, 7)
TIMER_DMA_CH3D	通道3比较/捕获DMA请求，TIMERx (x=0~4, 7)
TIMER_DMA_CMTD	换相DMA更新请求，TIMERx (x=0, 7, 15, 16)
TIMER_DMA_TRGD	触发DMA请求，TIMERx (x=0, 7, 15, 16)
TIMER_DMA_MCH0D	多模式通道0比较/捕获DMA请求，TIMERx (x=15, 16)
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 3-701. 函数 timer_channel_dma_request_source_select

函数名称	timer_channel_dma_request_source_select
函数原型	void timer_channel_dma_request_source_select(uint32_t timer_periph, uint32_t dma_request);
功能描述	外设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
TIMERx(x=0~4, 7, 15, 16)	TIMER外设选择
输入参数{in}	
dma_request	通道的DMA请求源选择
TIMER_DMAREQUEST_CHANNELEVENT	当通道捕获/比较事件发生时，发送通道n的DMA请求
TIMER_DMAREQUEST_UPDATEEVENT	当更新事件发生，发送通道n的DMA请求
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* TIMER0 channel DMA request of channel n is sent when channel 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 3-702. 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
<i>TIMERx</i> (<i>x</i> =0~4,7,15,16)	参考具体参数
输入参数{in}	
dma_baseaddr	DMA传输起始地址
<i>TIMER_DMACFG_DMATA_CTL0</i>	DMA传输起始地址: <i>TIMER_CTL0</i> , <i>TIMERx</i> (<i>x</i> =0,7,1,2,3,4,15,16)
<i>TIMER_DMACFG_DMATA_CTL1</i>	DMA传输起始地址: <i>TIMER_CTL1</i> , <i>TIMERx</i> (<i>x</i> =0,7,1,2,3,4,15,16)
<i>TIMER_DMACFG_DMATA_SMCFG</i>	DMA传输起始地址: <i>TIMER_SMCFG</i> , <i>TIMERx</i> (<i>x</i> =0,7,1,2,3,4,15,16)
<i>TIMER_DMACFG_DMATA_DMAINTEN</i>	DMA传输起始地址: <i>TIMER_DMAINTEN</i> , <i>TIMERx</i> (<i>x</i> =0,7,1,2,3,4,15,16)
<i>TIMER_DMACFG_DMATA_INTF</i>	DMA传输起始地址: <i>TIMER_INTF</i> , <i>TIMERx</i> (<i>x</i> =0,7,1,2,3,4,15,16)
<i>TIMER_DMACFG_DMATA_SWEVG</i>	DMA传输起始地址: <i>TIMER_SWEVG</i> , <i>TIMERx</i> (<i>x</i> =0,7,1,2,3,4,15,16)
<i>TIMER_DMACFG_DMATA_CHCTL0</i>	DMA传输起始地址: <i>TIMER_CHCTL0</i> , <i>TIMERx</i> (<i>x</i> =0,7,1,2,3,4,15,16)
<i>TIMER_DMACFG_DMATA_CHCTL1</i>	DMA传输起始地址: <i>TIMER_CHCTL1</i> , <i>TIMERx</i> (<i>x</i> =0,7,1,2,3,4)
<i>TIMER_DMACFG_DMATA_CHCTL2</i>	DMA传输起始地址: <i>TIMER_CHCTL2</i> , <i>TIMERx</i> (<i>x</i> =0,7,1,2,3,4,15,16)
<i>TIMER_DMACFG_DMATA_CNT</i>	DMA传输起始地址: <i>TIMER_CNT</i> , <i>TIMERx</i> (<i>x</i> =0,7,1,2,3,4,15,16)
<i>TIMER_DMACFG_DMATA_PSC</i>	DMA传输起始地址: <i>TIMER_PSC</i> , <i>TIMERx</i> (<i>x</i> =0,7,1,2,3,4,15,16)
<i>TIMER_DMACFG_DMATA_CAR</i>	DMA传输起始地址: <i>TIMER_CAR</i> , <i>TIMERx</i> (<i>x</i> =0,7,1,2,3,4,15,16)
<i>TIMER_DMACFG_DMATA_CREP0</i>	DMA传输起始地址: <i>TIMER_CREP0</i> , <i>TIMERx</i> (<i>x</i> =0,7,15,16)
<i>TIMER_DMACFG_DMATA_CH0CV</i>	DMA传输起始地址: <i>TIMER_CH0CV</i> , <i>TIMERx</i> (<i>x</i> =0,7,1,2,3,4,15,16)
<i>TIMER_DMACFG_DMATA_CH1CV</i>	DMA传输起始地址: <i>TIMER_CH1CV</i> , <i>TIMERx</i> (<i>x</i> =0,7,1,2,3,4,15,16)
<i>TIMER_DMACFG_DMATA_CH2CV</i>	DMA传输起始地址: <i>TIMER_CH2CV</i> , <i>TIMERx</i> (<i>x</i> =0,7,1,2,3,4)
<i>TIMER_DMACFG_DMATA_CH3CV</i>	DMA传输起始地址: <i>TIMER_CH3CV</i> , <i>TIMERx</i> (<i>x</i> =0,7,1,2,3,4)
<i>TIMER_DMACFG_DMATA_CCHP0</i>	DMA传输起始地址: <i>TIMER_CCHP0</i> , <i>TIMERx</i> (<i>x</i> =0,7,15,16)

<code>TIMER_DMACFG_DMATA_MCHCTL0</code>	DMA传输起始地址: <code>TIMER_MCHCTL0</code> , <code>TIMERx(x=15,16)</code>
<code>TIMER_DMACFG_DMATA_MCHCTL2</code>	DMA传输起始地址: <code>TIMER_MCHCTL2</code> , <code>TIMERx(x=15,16)</code>
<code>TIMER_DMACFG_DMATA_MCH0CV</code>	DMA传输起始地址: <code>TIMER_MCH0CV</code> , <code>TIMERx(x=15,16)</code>
<code>TIMER_DMACFG_DMATA_CH0COMV_ADD</code>	DMA传输起始地址: <code>TIMER_CH0COMV_ADD</code> , <code>TIMERx(x=0,7,15,16)</code>
<code>TIMER_DMACFG_DMATA_CH1COMV_ADD</code>	DMA传输起始地址: <code>TIMER_CH1COMV_ADD</code> , <code>TIMERx(x=0,7,15,16)</code>
<code>TIMER_DMACFG_DMATA_CH2COMV_ADD</code>	DMA传输起始地址: <code>TIMER_CH2COMV_ADD</code> , <code>TIMERx(x=0,7)</code>
<code>TIMER_DMACFG_DMATA_CH3COMV_ADD</code>	DMA传输起始地址: <code>TIMER_CH3COMV_ADD</code> , <code>TIMERx(x=0,7)</code>
<code>TIMER_DMACFG_DMATA_CTL2</code>	DMA传输起始地址: <code>TIMER_CTL2</code> , <code>TIMERx(x=0,7,1,2,3,4,15,16)</code>
<code>TIMER_DMACFG_DMATA_AFCTL0</code>	DMA传输起始地址: <code>TIMER_AFCTL0</code> , <code>TIMERx(x=0,7,15,16)</code>
<code>TIMER_DMACFG_DMATA_CCHP1</code>	DMA传输起始地址: <code>TIMER_CCHP1</code> , <code>TIMERx(x=0,7)</code>
<code>TIMER_DMACFG_DMATA_WDGCNT</code>	DMA传输起始地址: <code>TIMER_WDGCNT</code> , <code>TIMERx(x=0~4,7,22,23,30,31)</code>
<code>TIMER_DMACFG_DMATA_CINITCTL</code>	DMA传输起始地址: <code>TIMER_CINITCTL</code> , <code>TIMERx(x=0,7,15,16)</code>
<code>TIMER_DMACFG_DMATA_CINITV</code>	DMA传输起始地址: <code>TIMER_CINITV</code> , <code>TIMERx(x=0,7,15,16)</code>
<code>TIMER_DMACFG_DMATA_CHBRKCTL</code>	DMA传输起始地址: <code>TIMER_CHBRKCTL</code> , <code>TIMERx(x=0,7)</code>
<code>TIMER_DMACFG_DMATA_CHBRKPER</code>	DMA传输起始地址: <code>TIMER_CHBRKPER</code> , <code>TIMERx(x=0,7)</code>
<code>TIMER_DMACFG_DMATA_CHBRKINTF</code>	DMA传输起始地址: <code>TIMER_CHBRKINTF</code> , <code>TIMERx(x=0,7)</code>
输入参数{in}	
<code>dma_lenth</code>	DMA传输长度
<code>TIMER_DMACFG_DMATC_xTRANSF</code>	(<code>x=1~38</code>), DMA传输 <code>x</code> 次

ER	
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 3-703. 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, uint32_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(x=0~7, 15, 16)	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=0~7, 15, 16)
TIMER_EVENT_SRC_CH0G	通道0捕获或比较事件发生, TIMERx(x=0, 7, 1, 2, 3, 4, 15, 16)
TIMER_EVENT_SRC_CH1G	通道1捕获或比较事件发生, TIMERx(x=0, 7, 1, 2, 3, 4, 15, 16)
TIMER_EVENT_SRC_CH2G	通道2捕获或比较事件发生, TIMERx(x=0, 7, 1, 2, 3, 4)
TIMER_EVENT_SRC_CH3G	通道3捕获或比较事件发生, TIMERx(x=0, 7, 1, 2, 3, 4)
TIMER_EVENT_SRC_CMTG	通道换相更新事件发生, TIMERx(x=0, 7, 14~16, 40~44)
TIMER_EVENT_SRC_TRGG	触发事件产生, TIMERx(x=0, 7, 15, 16)
TIMER_EVENT_SRC_BRK0G	产生BREAK0事件, TIMERx(x=0, 7, 1, 2, 3, 4, 15, 16)
TIMER_EVENT_SRC	多模式通道0捕获或比较事件发生, TIMERx(x=15, 16)

C_MCH0G	
TIMER_EVENT_SR C_CH0COMADDG	通道0附加比较事件发生, TIMERx(x=0,7,15,16)
TIMER_EVENT_SR C_CH1COMADDG	通道1附加比较事件发生, TIMERx(x=0,7,15,16)
TIMER_EVENT_SR C_CH2COMADDG	通道2附加比较事件发生, TIMERx(x=0,7)
TIMER_EVENT_SR C_CH3COMADDG	通道3附加比较事件发生, TIMERx(x=0,7)
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* software generate update event*/
```

```
timer_event_software_generate(TIMER0, TIMER_EVENT_SRC_UPG);
```

函数 timer_break_struct_para_init

函数timer_break_struct_para_init描述见下表:

表 3-704. 函数 timer_break_struct_para_init

函数名称	timer_break_struct_para_init
函数原型	void timer_break_struct_para_init(timer_break_parameter_struct* breakpara);
功能描述	将TIMER中止功能参数结构体中所有参数初始化为默认值
先决条件	-
被调用函数	-
输入参数{in}	
breakpara	中止功能配置结构体, 详见 结构体timer_break_parameter_struct 。
输出参数{out}	
-	-
返回值	
-	-

例如:

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

```
timer_break_parameter_struct timer_breakpara;
```

```
timer_break_struct_para_init(&timer_breakpara);
```

函数 timer_break_config

函数timer_break_config描述见下表:

表 3-705. 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=0,7,15,16)	TIMER外设选择
输入参数{in}	
breakpara	中止功能配置结构体，详见 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         = 0U;
timer_breakpara.outputautostate  = TIMER_OUTAUTO_ENABLE;
timer_breakpara.protectmode      = TIMER_CCHP0_PROT_OFF;
timer_breakpara.break0state      = TIMER_BREAK0_ENABLE;
timer_breakpara.break0filter     = 0U;
timer_breakpara.break0polarity   = TIMER_BREAK0_POLARITY_LOW;

timer_break_config(TIMER0, &timer_breakpara);

```

函数 timer_break_enable

函数timer_break_enable描述见下表：

表 3-706. 函数 timer_break_enable

函数名称	timer_break_enable
函数原型	void timer_break_enable(uint32_t timer_periph, uint16_t break_num);
功能描述	使能TIMER的中止功能

先决条件	只有在TIMERx_CCHP0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=0,7,15,16)	TIMER外设选择
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* enable TIMER0 BREAK0 function*/
timer_break_enable(TIMER0);
```

函数 timer_break_disable

函数timer_break_disable描述见下表：

表 3-707. 函数 timer_break_disable

函数名称	timer_break_disable
函数原型	void timer_break_disable(uint32_t timer_periph, uint16_t break_num);
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=0,7,15,16)	TIMER外设选择
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* disable TIMER0 BREAK0 function*/
timer_break_disable(TIMER0);
```

函数 timer_automatic_output_enable

函数timer_automatic_output_enable描述见下表：

表 3-708. 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_CCHP0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=0,7,15,16)	TIMER外设选择
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* enable TIMER0 output automatic function */
```

```
timer_automatic_output_enable(TIMER0);
```

函数 timer_automatic_output_disable

函数timer_automatic_output_disable描述见下表：

表 3-709. 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_CCHP0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=0,7,15,16)	TIMER外设选择
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* disable TIMER0 output automatic function */
```

```
timer_automatic_output_disable(TIMER0);
```

函数 timer_primary_output_config

函数timer_primary_output_config描述见下表：

表 3-710. 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=0,7,15,16)	TIMER外设选择
输入参数{in}	
newvalue	控制状态
ENABLE	使能
DISABLE	禁能
输出参数{out}	
-	-
返回值	
-	-

例如:

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

函数 timer_channel_control_shadow_config

函数timer_channel_control_shadow_config描述见下表:

表 3-711. 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	配置通道控制影子寄存器
先决条件	-
被调用函数	-
输入参数{in}	
timer_periph	TIMER外设
TIMERx(x=0,7,15,16)	TIMER外设选择
输入参数{in}	
newvalue	控制状态
ENABLE	使能
DISABLE	禁能
输出参数{out}	

-	-
返回值	
-	-

例如：

```
/* channel commutation 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 3-712. 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, uint32_t ccuctl);
功能描述	配置TIMER通道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=0,7,15,16)	TIMER外设选择
输入参数{in}	
ccuctl	通道换相控制影子寄存器更新控制
TIMER_UPDATECT L_CCUCU	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
TIMER_UPDATECT L_CCUOVER	当计数器上溢事件发生时，影子寄存器更新
TIMER_UPDATECT L_CCUUNDER	当计数器下溢事件发生时，影子寄存器更新
TIMER_UPDATECT L_CCUOVERUNDE R	当计数器上溢/ 下溢事件发生时，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 3-713. 函数 timer_channel_output_struct_para_init

函数名称	timer_channel_output_struct_para_init
函数原型	void timer_channel_output_struct_para_init(timer_oc_parameter_struct* ocpara);
功能描述	将TIMER通道输出参数结构体中所有参数初始化为默认值
先决条件	-
被调用函数	-
输入参数{in}	
ocpara	输出通道结构体，详见 结构体timer oc parameter struct 。
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* initialize TIMER channel output parameter struct with a default value */
```

```
timer_oc_parameter_struct timer_ocinitpara;
```

```
timer_channel_output_struct_para_init(&timer_ocinitpara);
```

函数 timer_channel_output_config

函数timer_channel_output_config描述见下表：

表 3-714. 函数 timer_channel_output_config

函数名称	timer_channel_output_config
函数原型	void timer_channel_output_config(uint32_t timer_periph, uint16_t channel, timer_oc_parameter_struct* ocpara);
功能描述	外设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(x=0~4,7,15,16)	参考具体参数
输入参数{in}	
channel	TIMER通道
TIMER_CH_0	通道0, (TIMERx(x=0,7,1,2,3,4,15,16))
TIMER_CH_1	通道1, (TIMERx(x=0,7,1,2,3,4,15,16))

<i>TIMER_CH_2</i>	通道2, (TIMERx(x=0,7,1,2,3,4))
<i>TIMER_CH_3</i>	通道3, (TIMERx(x=0,7,1,2,3,4))
输入参数{in}	
ocpara	输出通道结构体, 详见 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_ocinitpara;

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_ocinitpara);

```

函数 timer_channel_output_mode_config

函数timer_channel_output_mode_config描述见下表:

表 3-715. 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, uint32_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(x=0~4,7,15,16)</i>	参考具体参数
输入参数{in}	
channel	TIMER通道
<i>TIMER_CH_0</i>	通道0, (TIMERx(x=0~4,7,15,16))
<i>TIMER_CH_1</i>	通道1, (TIMERx(x=0~4,7,15,16))
<i>TIMER_CH_2</i>	通道2, (TIMERx(x=0~4,7))

<code>TIMER_CH_3</code>	通道3, (TIMERx(x=0~4,7))
<code>TIMER_MCH_0</code>	多模式通道0, (TIMERx, x=15,16)
输入参数{in}	
<code>ocmode</code>	通道输出比较模式
<code>TIMER_OC_MODE_TIMING</code>	时基模式
<code>TIMER_OC_MODE_ACTIVE</code>	匹配时设置为高
<code>TIMER_OC_MODE_INACTIVE</code>	匹配时设置为低
<code>TIMER_OC_MODE_TOGGLE</code>	匹配时翻转
<code>TIMER_OC_MODE_LOW</code>	强制为低
<code>TIMER_OC_MODE_HIGH</code>	强制为高
<code>TIMER_OC_MODE_PWM0</code>	PWM模式0
<code>TIMER_OC_MODE_PWM1</code>	PWM模式1
<code>TIMER_OC_MODE_ASYMMETRIC_PWM0</code>	非对称PWM模式0 (TIMERx, x=0,7)
<code>TIMER_OC_MODE_ASYMMETRIC_PWM1</code>	非对称PWM模式1 (TIMERx, x=0,7)
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 3-716. 函数 timer_channel_output_pulse_value_config

函数名称	timer_channel_output_pulse_value_config
函数原型	void timer_channel_output_pulse_value_config(uint32_t timer_periph, uint16_t channel, uint32_t pulse);
功能描述	配置外设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
TIMERx(x=0~4,7,15,16)	参考具体参数
输入参数{in}	
channel	TIMER通道
TIMER_CH_0	通道0, TIMERx(x=0~4,7,15,16)
TIMER_CH_1	通道1, TIMERx(x=0~4,7,15,16)
TIMER_CH_2	通道2, TIMERx(x=0~4,7)
TIMER_CH_3	通道3, TIMERx(x=0~4,7)
TIMER_MCH_0	多模式通道0, TIMERx, (x=15,16)
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 3-717. 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
TIMERx(x=0~4,7,15,16)	参考具体参数
输入参数{in}	
channel	TIMER通道
TIMER_CH_0	通道0, TIMERx(x=0~4,7,15,16)
TIMER_CH_1	通道1, TIMERx(x=0~4,7,15,16)

<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)
<i>TIMER_MCH_0</i>	多模式通道0, <i>TIMERx</i> , (<i>x</i> =15,16)
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>	通道输出比较影子寄存器使能
<i>TIMER_OC_SHADOW_DISABLE</i>	通道输出比较影子寄存器禁能
<i>TIMER_OMC_SHADOW_ENABLE</i>	多模式通道输出比较影子寄存器使能
<i>TIMER_OMC_SHADOW_DISABLE</i>	多模式通道输出比较影子寄存器禁能
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_channel_output_shadow_config(TIMER0,                                TIMER_OC_SHADOW_ENABLE);
```

函数 timer_channel_output_compare_fast_config

函数timer_channel_output_compare_fast_config描述见下表:

表 3-718. 函数 timer_channel_output_compare_fast_config

函数名称	timer_channel_output_compare_fast_config
函数原型	void timer_channel_output_compare_fast_config(uint32_t timer_periph, uint16_t channel, uint16_t ocfast);
功能描述	配置TIMER的通道输出比较快速输出功能
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> =0~4,7,15,16)	参考具体参数
输入参数{in}	
channel	TIMER通道
<i>TIMER_CH_0</i>	通道0, <i>TIMERx</i> (<i>x</i> =0~4,7,15,16)
<i>TIMER_CH_1</i>	通道1, <i>TIMERx</i> (<i>x</i> =0~4,7,15,16)
<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)

<i>TIMER_MCH_0</i>	多模式通道0, <i>TIMERx</i> , (<i>x</i> =15,16)
输入参数{in}	
occlear	通道比较快速输出功能状态
<i>TIMER_OC_FAST_ENABLE</i>	通道比较快速输出功能使能
<i>TIMER_OC_FAST_DISABLE</i>	通道比较快速输出功能禁能
<i>TIMER_OMC_FAST_ENABLE</i>	多模式通道比较快速输出功能使能
<i>TIMER_OMC_FAST_DISABLE</i>	多模式通道比较快速输出功能禁能
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* configure TIMER0 channel0 output compare fast function */
```

```
timer_channel_output_compare_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 3-719. 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 oclear);
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
<i>TIMERx</i> (<i>x</i> =0~4,7)	参考具体参数
输入参数{in}	
channel	TIMER通道
<i>TIMER_CH_0</i>	通道0, <i>TIMERx</i> (<i>x</i> =0~4,7)
<i>TIMER_CH_1</i>	通道1, <i>TIMERx</i> (<i>x</i> =0~4,7)
<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}	
occlear	通道比较输出清0功能状态
<i>TIMER_OC_CLEAR</i>	通道比较输出清0功能使能

<code>_ENABLE</code>	
<code>TIMER_OC_CLEAR_DISABLE</code>	通道比较输出清0功能禁能
<code>TIMER_OMC_CLEAR_ENABLE</code>	多模式通道比较输出清0功能使能
<code>TIMER_OMC_CLEAR_DISABLE</code>	多模式通道比较输出清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 3-720. 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(x=0~4,7,15,16)	参考具体参数
输入参数{in}	
channel	TIMER通道
TIMER_CH_0	通道0, TIMERx(x=0~4,7,15,16)
TIMER_CH_1	通道1, TIMERx(x=0~4,7,15,16)
TIMER_CH_2	通道2, TIMERx(x=0~4,7)
TIMER_CH_3	通道3, TIMERx(x=0~4,7)
TIMER_MCH_0	多模式通道0, TIMERx, (x=15,16)
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

ITY_LOW	
TIMER_OMC_POL ARITY_HIGH	多模式通道输出极性高电平有效
TIMER_OMC_POL ARITY_LOW	多模式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 3-721. 函数 timer_channel_complementary_output_polarity_config

函数名称	timer_channel_complementary_output_polarity_config
函数原型	void timer_channel_complementary_output_polarity_config(uint32_t timer_periph, uint16_t channel, uint16_t ocnpolarity);
功能描述	互补通道输出极性配置
先决条件	-
被调用函数	-
输入参数{in}	
timer_periph	TIMER外设
TIMERx(x=0~4,7,15 ,16)	TIMER外设选择
输入参数{in}	
channel	TIMER通道
TIMER_CH_0	通道0, TIMERx(x=0~4,7,15,16)
TIMER_CH_1	通道1, TIMERx(x=0~4,7,15,16)
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}	
ocnpolarity	互补通道输出极性
TIMER_OCN_POLA RITY_HIGH	互补通道输出极性高电平有效
TIMER_OCN_POLA RITY_LOW	互补通道输出极性低电平有效
输出参数{out}	
-	-

返回值	
-	-

例如:

```
/* configure TIMER0 channel 0 complementary output polarity */
```

```
timer_channel_complementary_output_polarity_config(TIMER0, TIMER_CH_0,  
TIMER_OCN_POLARITY_HIGH);
```

函数 timer_channel_output_state_config

函数timer_channel_output_state_config描述见下表:

表 3-722. 函数 timer_channel_output_state_config

函数名称	timer_channel_output_state_config
函数原型	void timer_channel_output_state_config(uint32_t timer_periph, uint16_t channel, uint32_t state);
功能描述	配置通道状态
先决条件	-
被调用函数	-
输入参数{in}	
timer_periph	TIMER外设
TIMERx(x=0~4,7,15,16)	参考具体参数
输入参数{in}	
channel	TIMER通道
TIMER_CH_0	通道0, TIMERx(x=0~4,7,15,16)
TIMER_CH_1	通道1, TIMERx(x=0~4,7,15,16)
TIMER_CH_2	通道2, TIMERx(x=0~4,7)
TIMER_CH_3	通道3, TIMERx(x=0~4,7)
TIMER_MCH_0	多模式通道0, TIMERx, (x=15,16)
输入参数{in}	
state	通道状态
TIMER_CCX_ENABLE	通道使能
TIMER_CCX_DISABLE	通道禁能
TIMER_MCCX_ENABLE	多模式通道使能
TIMER_MCCX_DISABLE	多模式通道禁能
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 3-723. 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=0,7,14~16,40~44)	TIMER外设选择
输入参数{in}	
channel	TIMER通道
TIMER_CH_0	通道0, TIMERx(x=0~4,7,15,16)
TIMER_CH_1	通道1, TIMERx(x=0~4,7)
TIMER_CH_2	通道2, TIMERx(x=0~4,7)
TIMER_CH_3	通道3, TIMERx(x=0~4,7)
输入参数{in}	
state	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_composite_asymmetric_pwm_level_config

函数timer_channel_composite_asymmetric_pwm_level_config描述见下表:

表 3-724. 函数 timer_channel_composite_asymmetric_pwm_level_config

函数名称	timer_channel_composite_asymmetric_pwm_level_config
函数原型	void timer_channel_composite_asymmetric_pwm_level_config(uint32_t timer_periph, uint16_t channel, ControlStatus newvalue);
功能描述	复合PWM模式或者非对称PWM模式下配置TIMER通道周期点匹配时OxCPRE电平
先决条件	-
被调用函数	-
输入参数{in}	
timer_periph	TIMER外设
TIMERx(x=0,7,15,16)	TIMER外设选择
输入参数{in}	
channel	TIMER通道
TIMER_CH_0	TIMER通道0(TIMERx, x=0,7,15,16)
TIMER_CH_1	TIMER通道1(TIMERx, x=0,7,15,16)
TIMER_CH_2	TIMER通道2(TIMERx, x=0,7)
TIMER_CH_3	TIMER通道3(TIMERx, x=0,7)
输入参数{in}	
newvalue	控制值
ENABLE	使能
DISABLE	禁能
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* enable TIMER0 channel 0 period point match moment OxCPRE level in the composite
PWM or asymmetric PWM mode */
```

```
timer_channel_composite_asymmetric_pwm_level_config(TIMER0,          TIMER_CH_0,
ENABLE);
```

函数 timer_channel_output_clear_invalid_selection

函数timer_channel_output_clear_invalid_selection描述见下表:

表 3-725. 函数 timer_channel_output_clear_invalid_selection

函数名称	timer_channel_output_clear_invalid_selection
函数原型	oid timer_channel_output_clear_invalid_selection(uint32_t timer_periph, uint32_t event);

功能描述	输出清除无效事件选择
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}	
event	TIMER事件
TIMER_OCREFCLR_IN VALID_FLOW	输出清除无效在下一个上溢或者下溢事件
TIMER_OCREFCLR_IN VALID_UPDATE	输出清除无效在下一个更新事件
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* select update event as TIMER0 output clear invalid event*/
```

```
timer_channel_output_clear_invalid_selection(TIMER0,  
TIMER_OCREFCLR_INVALID_UPDATE);
```

函数 timer_channel_input_struct_para_init

函数timer_channel_input_struct_para_init描述见下表：

表 3-726. 函数 timer_channel_input_struct_para_init

函数名称	timer_channel_input_struct_para_init
函数原型	void timer_channel_input_struct_para_init(timer_ic_parameter_struct* icpara);
功能描述	将TIMER通道输入参数结构体中所有参数初始化为默认值
先决条件	-
被调用函数	-
输入参数{in}	
icpara	通道输入结构体，详见 结构体timer_ic_parameter_struct 。
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* initialize TIMER channel input parameter struct with a default value */
```

```
timer_ic_parameter_struct timer_icinitpara;
```

```
timer_channel_input_struct_para_init(&timer_icinitpara);
```


函数 timer_input_capture_config

函数timer_input_capture_config描述见下表：

表 3-727. 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(x=0~4, 7, 15, 16)	参考具体参数
输入参数{in}	
channel	TIMER通道
TIMER_CH_0	通道0(TIMERx, x=0~4, 7, 15, 16)
TIMER_CH_1	通道1(TIMERx, x=0~4, 7, 15, 16)
TIMER_CH_2	通道2(TIMERx, x=0~4, 7)
TIMER_CH_3	通道3(TIMERx, x=0~4, 7)
TIMER_MCH_0	多模式通道0(TIMERx, x=15, 16)
输入参数{in}	
icpara	通道输入结构体，详见 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.icselecion = TIMER_IC_SELECTION_DIRECTTTI;

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 3-728. 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
TIMERx(x=0~4,7,15,16)	参考具体参数
输入参数{in}	
channel	TIMER通道
TIMER_CH_0	通道0(TIMERx, x=0~4,7,15,16)
TIMER_CH_1	通道1(TIMERx, x=0~4,7,15,16)
TIMER_CH_2	通道2(TIMERx, x=0~4,7)
TIMER_CH_3	通道3(TIMERx, x=0~4,7)
TIMER_MCH_0	多模式通道0(TIMERx, x=15,16)
输入参数{in}	
prescaler	通道输入捕获预分频值
TIMER_IC_PSC_DIV1	不分频
TIMER_IC_PSC_DIV2	2分频
TIMER_IC_PSC_DIV4	4分频
TIMER_IC_PSC_DIV8	8分频
输出参数{out}	
-	-
返回值	
-	-

例如:

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

```
timer_channel_input_capture_prescaler_config(TIMER0, TIMER_CH_0,
TIMER_IC_PSC_DIV2);
```

函数 timer_channel_capture_value_register_read

函数timer_channel_capture_value_register_read描述见下表:

表 3-729. 函数 timer_channel_capture_value_register_read

函数名称	timer_channel_capture_value_register_read
------	---

函数原型	uint32_t timer_channel_capture_value_register_read(uint32_t timer_periph, uint16_t channel);
功能描述	读取通道捕获值
先决条件	-
被调用函数	-
输入参数{in}	
timer_periph	TIMER外设
TIMERx(x=0~4, 7, 15, 16)	参考具体参数
输入参数{in}	
channel	TIMER通道
TIMER_CH_0	通道0(TIMERx, x=0~4, 7, 15, 16)
TIMER_CH_1	通道1(TIMERx, x=0~4, 7, 15, 16)
TIMER_CH_2	通道2(TIMERx, x=0~4, 7)
TIMER_CH_3	通道3(TIMERx, x=0~4, 7)
TIMER_MCH_0	多模式通道0(TIMERx, x=15, 16)
输出参数{out}	
-	-
返回值	
uint32_t	通道输入捕获值 (0~0xFFFFFFFF)

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 3-730. 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)	TIMER外设选择
输入参数{in}	
channel	TIMER通道
TIMER_CH_0	通道0

<i>TIMER_CH_1</i>	通道1
输入参数{in}	
icpwm	输入PWM捕获结构体，详见 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_ic_initpara;

timer_ic_initpara.icpolarity = TIMER_IC_POLARITY_RISING;

timer_ic_initpara.icselection = TIMER_IC_SELECTION_DIRECTTTI;

timer_ic_initpara.icprescaler = TIMER_IC_PSC_DIV1;

timer_ic_initpara.icfilter = 0x0;

timer_input_pwm_capture_config(TIMER0, TIMER_CH_0, &timer_ic_initpara);

```

函数 timer_hall_mode_config

函数timer_hall_mode_config描述见下表：

表 3-731. 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, uint32_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
<i>TIMERx(x=0~4,7,15,16)</i>	TIMER外设选择
输入参数{in}	
hallmode	HALL接口功能状态
<i>TIMER_HALLINTE RFACE_ENABLE</i>	HALL接口使能
<i>TIMER_HALLINTE RFACE_DISABLE</i>	HALL接口禁能
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* configure TIMER0 hall sensor mode */
timer_hall_mode_config(TIMER0, TIMER_HALLINTERFACE_ENABLE);
```

函数 timer_multi_mode_channel_output_parameter_struct_init

函数timer_multi_mode_channel_output_parameter_struct_init描述见下表：

表 3-732. 函数 timer_multi_mode_channel_output_parameter_struct_init

函数名称	timer_multi_mode_channel_output_parameter_struct_init
函数原型	void timer_multi_mode_channel_output_parameter_struct_init(timer_omc_parameter_struct *omcpara);
功能描述	将TIMER多模式通道输出参数结构体中所有参数初始化为默认值
先决条件	-
被调用函数	-
输入参数{in}	
omcpara	多模式通道输出参数结构体，详见 结构体timer_omc_parameter_struct 。
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* initialize TIMER multi mode channel output parameter struct with a default value */
timer_omc_parameter_struct timer_omcinitpara;
timer_multi_mode_channel_output_parameter_struct_init(&timer_omcinitpara);
```

函数 timer_multi_mode_channel_output_config

函数timer_multi_mode_channel_output_config描述见下表：

表 3-733. 函数 timer_multi_mode_channel_output_config

函数名称	timer_multi_mode_channel_output_config
函数原型	void timer_multi_mode_channel_output_config(uint32_t timer_periph, uint16_t channel, timer_omc_parameter_struct *omcpara);
功能描述	外设TIMER的多模式通道输出配置
先决条件	-
被调用函数	-
输入参数{in}	
timer_periph	TIMER外设
TIMERx(x=15, 16)	参考具体参数
输入参数{in}	

channel	TIMER通道
<i>TIMER_MCH_0</i>	多模式通道0, TIMERx (x=15,16)
输入参数{in}	
omcpara	多模式通道输出参数结构体, 详见 结构体timer_omc_parameter_struct 。
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* configure TIMER15 multi mode channel 0 output function */
```

```
timer_omc_parameter_struct timer_omcinitpara;
```

```
omcpara->outputmode = TIMER_MCH_MODE_INDEPENDENTLY;
```

```
omcpara->outputstate = TIMER_MCCX_ENABLE;
```

```
omcpara->ocpolarity = TIMER_OMC_POLARITY_HIGH;
```

```
timer_multi_mode_channel_output_parameter_struct_init(TIMER15,      TIMER_MCH_0,
&timer_omcinitpara);
```

函数 timer_multi_mode_channel_mode_config

函数timer_multi_mode_channel_mode_config描述见下表:

表 3-734. 函数 timer_multi_mode_channel_mode_config

函数名称	timer_multi_mode_channel_mode_config
函数原型	void timer_multi_mode_channel_mode_config(uint32_t timer_periph, uint32_t channel, uint32_t multi_mode_sel);
功能描述	外设TIMER多模式通道模式选择
先决条件	-
被调用函数	-
输入参数{in}	
timer_periph	TIMER外设
<i>TIMERx(x=15,16)</i>	参考具体参数
输入参数{in}	
channel	TIMER通道
<i>TIMER_MCH_0</i>	多模式通道0, TIMERx (x=15,16)
输入参数{in}	
ocmode	通道输出比较模式
<i>TIMER_MCH_MODE_INDEPENDENTLY</i>	多模式通道为独立模式
<i>TIMER_MCH_MODE_COMPLEMENTARY</i>	多模式通道为互补模式

<i>RY</i>	
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* select TIMER15 multi mode channel 0 mode */
```

```
timer_multi_mode_channel_mode_config      (TIMER15,          TIMER_MCH_0,
TIMER_MCH_MODE_INDEPENDENTLY);
```

函数 timer_input_trigger_source_select

函数timer_input_trigger_source_select描述见下表:

表 3-735. 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(x=0~4,7,15,16)	TIMER外设选择
输入参数{in}	
intrigger	输入触发源
TIMER_SMCFG_T RGSEL_ITI0	内部触发输入0(ITI0, TIMERx(x=0..4,7,15,16))
TIMER_SMCFG_T RGSEL_ITI1	内部触发输入1(ITI0, TIMERx(x=0..4,7,15,16))
TIMER_SMCFG_T RGSEL_ITI2	内部触发输入2(ITI0, TIMERx(x=0..4,7,15,16))
TIMER_SMCFG_T RGSEL_ITI3	内部触发输入3(ITI0, TIMERx(x=0..4,7,15,16))
TIMER_SMCFG_T RGSEL_CIOF_ED	TIO的边沿检测(ITI0, TIMERx(x=0..4,7,15,16))
TIMER_SMCFG_T RGSEL_CIOFE0	滤波后的通道0输入(CIOFE0, TIMERx(x=0..4,7,15,16))
TIMER_SMCFG_T RGSEL_CI1FE1	滤波后的通道1输入(CI1FE1, TIMERx(x=0..4,7,15,16))
TIMER_SMCFG_T RGSEL_ETIFP	滤波后的外部触发输入(ETIFP, TIMERx(x=0~4,7))

<code>TIMER_SMCFG_TRGSEL_CI2FE2</code>	滤波后的通道2输入(CI2FE2, TIMERx(x=0~4,7))
<code>TIMER_SMCFG_TRGSEL_CI3FE3</code>	滤波后的通道3输入(CI3FE3, TIMERx(x=0~4,7))
<code>TIMER_SMCFG_TRGSEL_MCI0FEM0</code>	滤波后的多模式通道0输入(MCI0FEM0, TIMERx(x=15,16))
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 3-736. 函数 timer_master_output_trigger_source_select

函数名称	timer_master_output0_trigger_source_select
函数原型	void timer_master_output0_trigger_source_select(uint32_t timer_periph, uint32_t outrigger);
功能描述	选择TIMER主模式输出0触发源
先决条件	-
被调用函数	-
输入参数{in}	
timer_periph	TIMER外设
TIMERx(x=0~7,15,16)	TIMER外设选择
输入参数{in}	
outrigger	输出触发源
TIMER_TRI_OUT0_SRC_RESET	UPG位作为TRGO0 (TIMERx(x=0~7,15,16))
TIMER_TRI_OUT0_SRC_ENABLE	TIMER使能信号作为TRGO0(TIMERx(x=0~7,15,16))
TIMER_TRI_OUT0_SRC_UPDATE	更新事件作为TRGO0 (TIMERx(x=0~7,15,16))
TIMER_TRI_OUT0_SRC_CH0	通道0捕获/比较事件作为TRGO0 (TIMERx(x=0~4,7,15,16))
TIMER_TRI_OUT0_SRC_O0CPRE	O0CPRE作为触发输出TRGO0(TIMERx(x=0~4,7,15,16))
TIMER_TRI_OUT0_SRC_O1CPRE	O1CPRE作为触发输出TRGO0(TIMERx(x=0~4,7,15,16))

<code>TIMER_TRI_OUT0_SRC_O2CPRE</code>	O2CPRE作为触发输出TRGO0(TIMERx(x=0~4,7))
<code>TIMER_TRI_OUT0_SRC_O3CPRE</code>	O3CPRE作为触发输出TRGO0(TIMERx(x=0~4,7))
<code>TIMER_TRI_OUT0_SRC_DECODER_CLOCK</code>	译码器时钟作为触发输出TRGO0(TIMERx(x=0,7,1,2,3,4))
<code>TIMER_TRI_OUT0_SRC_SYNC</code>	同步事件作为触发输出TRGO0(TIMERx(x=0,7,15,16))
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_OUT0_SRC_RESET);
```

函数 timer_slave_mode_select

函数timer_slave_mode_select描述见下表:

表 3-737. 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=0~4,7,15,16)	TIMER外设选择
输入参数{in}	
slavemode	从模式
TIMER_SLAVE_MODE_DISABLE	关闭从模式, TIMERx(x=0~4,7,15,16)
TIMER_QUAD_ENCODER_MODE0	正交编码器模式0, TIMERx(x=0~4,7)
TIMER_QUAD_ENCODER_MODE1	正交编码器模式1, TIMERx(x=0~4,7)
TIMER_QUAD_ENCODER_MODE2	正交编码器模式2, TIMERx(x=0~4,7)
TIMER_SLAVE_MODE_RESTART	复位模式, TIMERx(x=0~4,7,15,16)

TIMER_SLAVE_MODE_PAUSE	暂停模式, TIMERx(x=0~4,7,15,16)
TIMER_SLAVE_MODE_EVENT	事件模式, TIMERx(x=0~4,7,15,16)
TIMER_SLAVE_MODE_EXTERNAL0	外部时钟模式0, TIMERx(x=0~4,7,15,16)
TIMER_NONQUAD_DECODER_MODE0	非正交编码器模式0(TIMERx(x=1~4))
TIMER_NONQUAD_DECODER_MODE1	非正交编码器模式1(TIMERx(x=1~4))
TIMER_NONQUAD_DECODER_MODE2	非正交编码器模式2(TIMERx(x=1~4))
TIMER_NONQUAD_DECODER_MODE3	非正交编码器模式3(TIMERx(x=1~4))
TIMER_QUAD_DECODER_MODE3	正交编码器模式3(TIMERx(x=1~4))
TIMER_QUAD_DECODER_MODE4	正交编码器模式4(TIMERx(TIMERx(x=1~4))
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 3-738. 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, uint32_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=0~4,7,15,16)	TIMER外设选择
输入参数{in}	
masterslave	主从模式使能状态
TIMER_MASTER_SLAVE_MODE_ENABLE	主从模式使能
TIMER_MASTER_SLAVE_MODE_DISABLE	主从模式禁能
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* configure TIMER0 master slave mode */
```

```
timer_master_slave_mode_config(TIMER0, TIMER_MASTER_SLAVE_MODE_ENABLE);
```

函数 timer_external_trigger_config

函数timer_external_trigger_config描述见下表:

表 3-739. 函数 timer_external_trigger_config

函数名称	timer_external_trigger_config
函数原型	void timer_external_trigger_config(uint32_t timer_periph, uint32_t extprescaler, uint32_t expolarity, uint32_t extfilter);
功能描述	配置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
<i>TIMER_ETP_FALLING</i>	低电平或者下降沿有效
<i>TIMER_ETP_RISING</i>	高电平或者上升沿有效
输入参数{in}	
extfilter	外部触发滤波控制（0~15）
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* configure TIMER0 external trigger input */
```

```
timer_external_trigger_config(TIMER0, TIMER_EXT_TRI_PSC_DIV2,  
TIMER_ETP_FALLING, 10);
```

函数 timer_quadrature_decoder_mode_config

函数timer_quadrature_decoder_mode_config描述见下表：

表 3-740. 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
<i>TIMERx(x=0~4,7)</i>	TIMER外设选择
输入参数{in}	
decomode	编码器模式
<i>TIMER_QUAD_DECODER_MODE0</i>	根据CI0FE0的电平，计数器在CI1FE1的边沿向上/下计数(TIMERx(x=0..4,7))
<i>TIMER_QUAD_DECODER_MODE1</i>	根据CI1FE1的电平，计数器在CI0FE0的边沿向上/下计数(TIMERx(x=0..4,7))
<i>TIMER_QUAD_DECODER_MODE2</i>	根据另一个信号的输入电平，计数器在CI0FE0和CI1FE1的边沿向上/下计数(TIMERx(x=0..4,7))
<i>TIMER_QUAD_DECODER_MODE3</i>	根据CI0FE0的电平，计数器在CI0FE0的边沿向上/下计数(TIMERx(x=1..4))
<i>TIMER_QUAD_DECODER_MODE4</i>	根据CI1FE1的电平，计数器在CI1FE1的边沿向上/下计数(TIMERx(x=1..4))
输入参数{in}	

ic0polarity	IC0输入极性
<i>TIMER_IC_POLARITY_RISING</i>	捕获上升边沿
<i>TIMER_IC_POLARITY_FALLING</i>	捕获下降边沿
<i>TIMER_IC_POLARITY_BOTH_EDGE</i>	双边沿有效
输入参数{in}	
ic1polarity	IC1输入极性
<i>TIMER_IC_POLARITY_RISING</i>	捕获上升边沿
<i>TIMER_IC_POLARITY_FALLING</i>	捕获下降边沿
<i>TIMER_IC_POLARITY_BOTH_EDGE</i>	双边沿有效
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* configure TIMER0 quadrature decoder mode */
```

```
timer_quadrature_decoder_mode_config(TIMER0, TIMER_ENCODER_MODE0,
TIMER_IC_POLARITY_RISING, TIMER_IC_POLARITY_RISING);
```

函数 timer_non_quadrature_decoder_mode_config

函数timer_non_quadrature_decoder_mode_config述见下表：

表 3-741. 函数 timer_non_quadrature_decoder_mode_config

函数名称	timer_non_quadrature_decoder_mode_config
函数原型	void timer_non_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
<i>TIMERx(x=1~4)</i>	TIMER外设选择
输入参数{in}	
decomode	编码器模式
<i>TIMER_NONQUAD_DECODER_MODE_0</i>	非正交编码器模式0

<code>TIMER_NONQUAD_DECODER_MODE1</code>	非正交编码器模式1
<code>TIMER_NONQUAD_DECODER_MODE2</code>	非正交编码器模式2
<code>TIMER_NONQUAD_DECODER_MODE3</code>	非正交编码器模式3
输入参数{in}	
<code>IC0polarity</code>	CI0输入极性
<code>TIMER_IC_POLARITY_RISING</code>	捕获上升边沿
<code>TIMER_IC_POLARITY_FALLING</code>	捕获下降边沿
<code>TIMER_IC_POLARITY_BOTH_EDGE</code>	捕获双边沿
输入参数{in}	
<code>IC1polarity</code>	CI1输入极性
<code>TIMER_IC_POLARITY_RISING</code>	捕获上升边沿
<code>TIMER_IC_POLARITY_FALLING</code>	捕获下降边沿
<code>TIMER_IC_POLARITY_BOTH_EDGE</code>	捕获双边沿
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* configure TIMER0 non-quadrature decoder mode */

timer_non_quadrature_decoder_mode_config (TIMER0,
TIMER_NONQUAD_DECODER_MODE3, 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 3-742. 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
TIMERx(x=0~4,7,15,16)	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 3-743. 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=0~4,7,15,16)	TIMER外设选择
输入参数{in}	
intrigger	被选择的内部触发源
TIMER_SMCFG_T RGSEL_ITI0	内部触发输入0(ITI0, TIMERx(x=0~4,7,15,16))
TIMER_SMCFG_T RGSEL_ITI1	内部触发输入1(ITI1, TIMERx(x=0~4,7,15,16))
TIMER_SMCFG_T RGSEL_ITI2	内部触发输入2(ITI2, TIMERx(x=0~4,7,15,16))
TIMER_SMCFG_T RGSEL_ITI3	内部触发输入3(ITI3, TIMERx(x=0~4,7,15,16))
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* configure TIMER0 the internal trigger ITIO as external clock input */
```

```
timer_internal_trigger_as_external_clock_config(TIMER0, TIMER_SMCFG_TRGSEL_ITIO);
```

函数 timer_external_trigger_as_external_clock_config

函数timer_external_trigger_as_external_clock_config描述见下表:

表 3-744. 函数 timer_external_trigger_as_external_clock_config

函数名称	timer_external_trigger_as_external_clock_config
函数原型	void timer_external_trigger_as_external_clock_config(uint32_t timer_periph, uint32_t extrigger, uint16_t expolarity, uint32_t extfilter);
功能描述	配置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=0~4,7,15,16)	TIMER外设选择
输入参数{in}	
extrigger	外部触发源
TIMER_SMCFG_TRGSEL_CIOF_ED	TI0的边沿检测(CIOF_ED, TIMERx(x=0~4,7,15,16))
TIMER_SMCFG_TRGSEL_CIOFE0	滤波后的通道0输入(CIOFE0, TIMERx(x=0~4,7,15,16))
TIMER_SMCFG_TRGSEL_C11FE1	滤波后的通道1输入(C11FE1, TIMERx(x=0~4,7,15,16))
TIMER_SMCFG_TRGSEL_C12FE2	滤波后的通道2输入(x=0~4,7,15,16))
TIMER_SMCFG_TRGSEL_C13FE3	滤波后的通道3输入(x=0~4,7,15,16))
TIMER_SMCFG_TRGSEL_MCIOFEM0	滤波后的多模式通道0输入(TIMERx(x=15,16))
输入参数{in}	
expolarity	外部触发源极性
TIMER_IC_POLARITY_RISING	外部触发源高电平或者上升沿有效
TIMER_IC_POLARITY_FALLING	外部触发源低电平或者下降沿有效
TIMER_IC_POLARITY_BOTH_EDGE	外部触发双边沿有效
输入参数{in}	
extfilter	滤波参数 (0~15)

输出参数{out}	
-	-
返回值	
-	-

例如:

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

函数 timer_external_clock_mode0_config

函数timer_external_clock_mode0_config描述见下表:

表 3-745. 函数 timer_external_clock_mode0_config

函数名称	timer_external_clock_mode0_config
函数原型	void timer_external_clock_mode0_config(uint32_t timer_periph, uint32_t extprescaler, uint32_t expolarity, uint32_t extfilter);
功能描述	配置TIMER外部时钟模式0, ETI作为时钟源
先决条件	-
被调用函数	timer_external_trigger_config
输入参数{in}	
timer_periph	TIMER外设
TIMERx(x=0~4,7)	TIMER外设选择
输入参数{in}	
extprescaler	ETI触发源预分频值
TIMER_EXT_TRI_P SC_OFF	不分频
TIMER_EXT_TRI_P SC_DIV2	2分频
TIMER_EXT_TRI_P SC_DIV4	4分频
TIMER_EXT_TRI_P SC_DIV8	8分频
输入参数{in}	
expolarity	ETI触发源极性
TIMER_ETP_FALLI NG	下降沿或者低电平有效
TIMER_ETP_RISIN G	上升沿或者高电平有效
输入参数{in}	
extfilter	ETI触发源滤波参数 (0~15)
输出参数{out}	
-	-

返回值	
-	-

例如：

```
/* configure TIMER0 the external clock 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 3-746. 函数 timer_external_clock_mode1_config

函数名称	timer_external_clock_mode1_config
函数原型	void timer_external_clock_mode1_config(uint32_t timer_periph, uint32_t extprescaler, uint32_t expolarity, uint32_t extfilter);
功能描述	配置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_PSC_OFF	不分频
TIMER_EXT_TRI_PSC_DIV2	2分频
TIMER_EXT_TRI_PSC_DIV4	4分频
TIMER_EXT_TRI_PSC_DIV8	8分频
输入参数{in}	
expolarity	ETI触发源极性
TIMER_ETP_FALLING	下降沿或者低电平有效
TIMER_ETP_RISING	上升沿或者高电平有效
输入参数{in}	
extfilter	ETI触发源滤波参数（0~15）
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* configure TIMER0 the external clock mode1 */

timer_external_clock_mode1_config(TIMER0, TIMER_EXT_TRI_PSC_DIV2,
TIMER_ETP_FALLING, 0);
```

函数 timer_external_clock_mode1_disable

函数timer_external_clock_mode1_disable描述见下表：

表 3-747. 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
TIMERx(x=0~4,7)	TIMER外设选择
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* disable TIMER0 the external clock mode1 */

timer_external_clock_mode1_disable(TIMER0);
```

函数 timer_write_chxval_register_config

函数timer_write_chxval_register_config描述见下表：

表 3-748. 函数 timer_write_chxval_register_config

函数名称	timer_write_chxval_register_config
函数原型	void timer_write_chxval_register_config(uint32_t timer_periph, uint16_t ccsel);
功能描述	配置TIMER写CHxVAL选择位
先决条件	-
被调用函数	-
输入参数{in}	
timer_periph	TIMER外设
TIMERx(x=0~4,7,15,16)	TIMER外设选择
输入参数{in}	
ccsel	写CHxVAL寄存器选择位
TIMER_CHVSEL_D	无影响

<i>ISABLE</i>	
<i>TIMER_CHVSEL_ENABLE</i>	当写入捕获比较寄存器的值与寄存器当前值相等时，写入操作无效。
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* configure TIMER0 write CHxVAL register selection */
```

```
timer_write_chxval_register_config(TIMER0, TIMER_CHVSEL_ENABLE);
```

函数 timer_output_value_selection_config

函数timer_output_value_selection_config描述见下表：

表 3-749. 函数 timer_output_value_selection_config

函数名称	timer_output_value_selection_config
函数原型	void timer_output_value_selection_config(uint32_t timer_periph, uint16_t outsel);
功能描述	配置TIMER输出值选择位
先决条件	-
被调用函数	-
输入参数{in}	
timer_periph	TIMER外设
TIMERx(x=0,7,15,16)	TIMER外设选择
输入参数{in}	
ccsel	输出值选择位
TIMER_OUTSEL_DISABLE	无影响
TIMER_OUTSEL_ENABLE	如果POEN位与IOS位均为0，则输出无效。
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* configure TIMER output value selection */
```

```
timer_output_value_selection_config(TIMER0, TIMER_OUTSEL_ENABLE);
```

函数 timer_commutation_control_shadow_register_config

函数timer_commutation_control_shadow_register_config描述见下表:

表 3-750. 函数 timer_commutation_control_shadow_register_config

函数名称	timer_commutation_control_shadow_register_config
函数原型	void timer_commutation_control_shadow_register_config(uint32_t timer_periph, uint16_t ccssel);
功能描述	配置换相控制影子寄存器更新选择
先决条件	-
被调用函数	-
输入参数{in}	
timer_periph	TIMER外设
TIMERx(x=0,7,15,16)	TIMER外设选择
输入参数{in}	
ccsel	换相控制影子寄存器选择
TIMER_CCUSEL_ENABLE	当计数器产生一个上溢/下溢事件时，影子寄存器才更新
TIMER_CCUSEL_DISABLE	当重复计数器值为0，且计数器产生一个上溢/下溢事件时，影子寄存器才更新
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* configure commutation control shadow register update selection */
```

```
timer_commutation_control_shadow_register_config (TIMER0, TIMER_CCUSEL_ENABLE);
```

函数 timer_output_match_pulse_select

函数timer_output_match_pulse_select描述见下表:

表 3-751. 函数 timer_output_match_pulse_select

函数名称	timer_output_match_pulse_select
函数原型	void timer_output_match_pulse_select(uint32_t timer_periph, uint32_t channel, uint16_t pulsesel);
功能描述	通道TIMER输出匹配脉冲选择
先决条件	-
被调用函数	-
输入参数{in}	
timer_periph	TIMER外设
TIMERx(x=0,7,15,16)	参考具体参数

6)	
输入参数{in}	
channel	TIMER通道
<i>TIMER_CH_0</i>	通道0, TIMERx(x=0,7,15,16)
<i>TIMER_CH_1</i>	通道1, TIMERx(x=0,7,15,16)
<i>TIMER_CH_2</i>	通道2, TIMERx(x=0,7)
<i>TIMER_CH_3</i>	通道3, TIMERx(x=0,7)
输入参数{in}	
pulsesel	输出匹配脉冲选择
<i>TIMER_PULSE_OUTPUT_NORMAL</i>	通道输出正常
<i>TIMER_PULSE_OUTPUT_CNT_UP</i>	仅在向上计数时, 通道输出脉冲
<i>TIMER_PULSE_OUTPUT_CNT_DOWN</i>	仅在向下计数时, 通道输出脉冲
<i>TIMER_PULSE_OUTPUT_CNT_BOTH</i>	向上/向下计数时, 通道输出脉冲
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* configure TIMER0 channel 0 match pulse selection */
```

```
timer_output_match_pulse_select(TIMER0, TIMER_CH_0,
TIMER_PULSE_OUTPUT_CNT_UP);
```

函数 timer_channel_composite_pwm_mode_config

函数timer_channel_composite_pwm_mode_config描述见下表:

表 3-752. 函数 timer_channel_composite_pwm_mode_config

函数名称	timer_channel_composite_pwm_mode_config
函数原型	void timer_channel_composite_pwm_mode_config(uint32_t timer_periph, uint32_t channel, ControlStatus newvalue);
功能描述	配置TIMER的复合PWM模式
先决条件	-
被调用函数	-
输入参数{in}	
timer_periph	TIMER外设
<i>TIMERx(x=0,7,15,16)</i>	参考具体参数
6)	
输入参数{in}	
channel	TIMER通道

<i>TIMER_CH_0</i>	通道0, <i>TIMERx</i> (<i>x</i> =0,7,15,16)
<i>TIMER_CH_1</i>	通道1, <i>TIMERx</i> (<i>x</i> =0,7,15,16)
<i>TIMER_CH_2</i>	通道2, <i>TIMERx</i> (<i>x</i> =0,7)
<i>TIMER_CH_3</i>	通道3, <i>TIMERx</i> (<i>x</i> =0,7)
输入参数{in}	
newvalue	控制状态
<i>ENABLE</i>	使能
<i>DISABLE</i>	禁能
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* configure the TIMER composite PWM mode */
```

```
timer_channel_composite_pwm_mode_config(TIMER0, TIMER_CH_0, ENABLE);
```

函数 `timer_channel_composite_pwm_mode_output_pulse_value_config`

函数 `timer_channel_composite_pwm_mode_output_pulse_value_config` 描述见下表:

表 3-753. 函数 `timer_channel_composite_pwm_mode_output_pulse_value_config`

函数名称	<code>timer_channel_composite_pwm_mode_output_pulse_value_config</code>
函数原型	void <code>timer_channel_composite_pwm_mode_output_pulse_value_config(uint32_t timer_periph, uint32_t channel, uint32_t pulse, uint32_t add_pulse);</code>
功能描述	配置TIMER的复合PWM模式输出脉冲值
先决条件	-
被调用函数	-
输入参数{in}	
timer_periph	TIMER外设
<i>TIMERx</i> (<i>x</i> =0,7,15,16)	参考具体参数
输入参数{in}	
channel	TIMER通道
<i>TIMER_CH_0</i>	通道0, <i>TIMERx</i> (<i>x</i> =0,7,15,16)
<i>TIMER_CH_1</i>	通道1, <i>TIMERx</i> (<i>x</i> =0,7,15,16)
<i>TIMER_CH_2</i>	通道2, <i>TIMERx</i> (<i>x</i> =0,7)
<i>TIMER_CH_3</i>	通道3, <i>TIMERx</i> (<i>x</i> =0,7)
输入参数{in}	
pulse	通道比较值 (0~0xFFFFFFFF)
输入参数{in}	
add_pulse	通道附加比较值 (0~0xFFFFFFFF)

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, 3999);
```

函数 timer_channel_asymmetric_pwm_pulse_config

函数timer_channel_asymmetric_pwm_pulse_config描述见下表：

表 3-754. 函数 timer_channel_asymmetric_pwm_pulse_config

函数名称	timer_channel_asymmetric_pwm_pulse_config
函数原型	void timer_channel_asymmetric_pwm_pulse_config (uint32_t timer_periph, uint32_t channel, uint32_t pulse, uint32_t add_pulse);
功能描述	配置TIMER的非对称PWM模式输出脉冲值
先决条件	-
被调用函数	-
输入参数{in}	
timer_periph	TIMER外设
TIMERx(x=0,7)	参考具体参数
输入参数{in}	
channel	TIMER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}	
pulse	通道比较值（0~0xFFFF）
输入参数{in}	
add_pulse	通道附加比较值（0~0xFFFF）
输出参数{out}	
-	-
返回值	
-	-

例如：

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

```
timer_channel_asymmetric_pwm_pulse_config (TIMER0, TIMER_CH_0, 399, 3999);
```


函数 timer_channel_additional_compare_value_config

函数timer_channel_additional_compare_value_config描述见下表:

表 3-755. 函数 timer_channel_additional_compare_value_config

函数名称	timer_channel_additional_compare_value_config
函数原型	void timer_channel_additional_compare_value_config(uint32_t timer_periph, uint16_t channel, uint32_t value)
功能描述	配置TIMER通道附加比较寄存器
先决条件	-
被调用函数	-
输入参数{in}	
timer_periph	TIMER外设
TIMERx(x=0,7,15,16)	参考具体参数
输入参数{in}	
channel	TIMER通道
TIMER_CH_0	通道0, TIMERx(x=0,7,15,16)
TIMER_CH_1	通道1, TIMERx(x=0,7,15,16)
TIMER_CH_2	通道2, TIMERx(x=0,7)
TIMER_CH_3	通道3, TIMERx(x=0,7)
输入参数{in}	
value	通道附加比较值 (0~0xFFFFFFFF)
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* configure TIMER0 channel 0 additional compare value */
```

```
timer_channel_additional_compare_value_config (TIMER0, TIMER_CH_0, 399);
```

函数 timer_channel_additional_output_update_select

函数timer_channel_additional_output_update_select描述见下表:

表 3-756. 函数 timer_channel_additional_output_update_select

函数名称	timer_channel_additional_output_update_select
函数原型	void timer_channel_additional_output_update_select(uint32_t timer_periph, uint16_t channel, uint16_t update);
功能描述	选择TIMER通道附加输出比较寄存器更新源
先决条件	-
被调用函数	-
输入参数{in}	

timer_periph	TIMER外设
<i>TIMERx(x=0,7,15,16)</i>	参考具体参数
输入参数{in}	
channel	TIMER通道
<i>TIMER_CH_0</i>	通道0, <i>TIMERx(x=0,7,15,16)</i>
<i>TIMER_CH_1</i>	通道1, <i>TIMERx(x=0,7,15,16)</i>
<i>TIMER_CH_2</i>	通道2, <i>TIMERx(x=0,7)</i>
<i>TIMER_CH_3</i>	通道3, <i>TIMERx(x=0,7)</i>
输入参数{in}	
update	channel additional output register update
<i>TIMER_ADD_UPDATE_BY_UPDATE</i>	channel additional compare value register update by update event
<i>TIMER_ADD_UPDATE_BY_COMPARE_MATCH</i>	channel additional compare value register update by compare value match event
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* select TIMER channel additional output register update source */
```

```
timer_channel_additional_output_update_select (TIMER0, TIMER_CH_0,
TIMER_ADD_UPDATE_BY_UPDATE);
```

函数 timer_channel_additional_output_shadow_config

函数timer_channel_additional_output_shadow_config描述见下表:

表 3-757. 函数 timer_channel_additional_output_shadow_config

函数名称	timer_channel_additional_output_shadow_config
函数原型	void timer_channel_additional_output_shadow_config(uint32_t timer_periph, uint16_t channel, uint16_t aocshadow);
功能描述	配置TIMER通道附加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(x=0,7,15,16)</i>	参考具体参数
输入参数{in}	
channel	TIMER通道
<i>TIMER_CH_0</i>	通道0, <i>TIMERx(x=0,7,15,16)</i>

<i>TIMER_CH_1</i>	通道1, <i>TIMERx</i> (<i>x</i> =0,7,15,16)
<i>TIMER_CH_2</i>	通道2, <i>TIMERx</i> (<i>x</i> =0,7)
<i>TIMER_CH_3</i>	通道3, <i>TIMERx</i> (<i>x</i> =0,7)
输入参数{in}	
aocshadow	通道附加输出比较影子寄存器状态
<i>TIMER_ADD_SHADOW_ENABLE</i>	通道附加输出比较影子寄存器使能
<i>TIMER_ADD_SHADOW_DISABLE</i>	通道附加输出比较影子寄存器禁能
输出参数{out}	
-	-
返回值	
-	-

例如:

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

```
timer_channel_additional_output_shadow_config (TIMER0,          TIMER_CH_0,
TIMER_OC_SHADOW_ENABLE);
```

函数 timer_channel_additional_compare_value_read

函数timer_channel_additional_compare_value_read描述见下表:

表 3-758. 函数 timer_channel_additional_compare_value_read

函数名称	timer_channel_additional_compare_value_read
函数原型	uint32_t timer_channel_additional_compare_value_read(uint32_t timer_periph, uint16_t channel);
功能描述	读取TIMER通道附加输出比较寄存器值
先决条件	-
被调用函数	-
输入参数{in}	
timer_periph	TIMER外设
<i>TIMERx</i> (<i>x</i> =0,7,15,16)	TIMER外设选择
输入参数{in}	
channel	TIMER通道
<i>TIMER_CH_0</i>	通道0, <i>TIMERx</i> (<i>x</i> =0,7,15,16)
<i>TIMER_CH_1</i>	通道1, <i>TIMERx</i> (<i>x</i> =0,7,15,16)
<i>TIMER_CH_2</i>	通道2, <i>TIMERx</i> (<i>x</i> =0,7)
<i>TIMER_CH_3</i>	通道3, <i>TIMERx</i> (<i>x</i> =0,7)
输出参数{out}	
-	-
返回值	
uint32_t	附加输出比较寄存器值, 0~0xFFFFFFFF

例如：

```
/* get TIMER autoreload register value */
uint32_t i = 0;

i = timer_channel_additional_compare_value_read(TIMER0, TIMER_CH_0);
```

函数 timer_break_external_source_config

函数timer_break_external_source_config描述见下表：

表 3-759. 函数 timer_break_external_source_config

函数名称	timer_break_external_source_config
函数原型	void timer_break_external_source_config(uint32_t timer_periph, uint32_t break_src, ControlStatus newvalue);
功能描述	配置TIMER中止功能外部输入源
先决条件	-
被调用函数	-
输入参数{in}	
timer_periph	TIMER外设
TIMERx(x=0,7,15,16)	参考具体参数
输入参数{in}	
break_src	break source
TIMER_BRKIN0	BRKIN0备用功能输入使能，TIMERx(x=0,7,15,16)
TIMER_BRKCMPO	CMPO输入使能，TIMERx(x=0,7,15,16)
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：

```
/* configure the TIMER break source */

timer_break_external_source_config(TIMER0, TIMER_BRKIN0, ENABLE);
```

函数 timer_break_external_polarity_config

函数timer_break_external_polarity_config描述见下表：

表 3-760. 函数 timer_break_external_polarity_config

函数名称	timer_break_external_polarity_config
------	--------------------------------------

函数原型	void timer_break_external_polarity_config(uint32_t timer_periph, uint16_t break_num, uint32_t break_src, uint16_t bkinpolarity);
功能描述	配置TIMER中止功能输入极性
先决条件	-
被调用函数	-
输入参数{in}	
timer_periph	TIMER外设
TIMERx(x=0,7,15,16)	参考具体参数
输入参数{in}	
break_src	break source
TIMER_BRKIN0	BRKIN0备用功能输入使能, TIMERx(x=0,7,15,16)
TIMER_BRKCMPO	CMP0输入使能, TIMERx(x=0,7,15,16)
输入参数{in}	
bkinpolarity	中止极性
TIMER_BRKIN_POLARITY_LOW	低电平有效
TIMER_BRKIN_POLARITY_HIGH	高电平有效
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* configure the TIMER break polarity */
```

```
timer_break_external_polarity_config(TIMER0,                                TIMER_BRKIN0,
timer_break_external_polarity_config(TIMER0,                                TIMER_BRKIN_POLARITY_HIGH);
```

函数 timer_dead_time_falling_edge_config

函数timer_dead_time_falling_edge_config描述见下表:

表 3-761. 函数 timer_dead_time_falling_edge_config

函数名称	timer_dead_time_falling_edge_config
函数原型	void timer_dead_time_falling_edge_config(uint32_t timer_periph, uint32_t deadtime);
功能描述	配置TIMER下降沿死区时间
先决条件	-
被调用函数	-
输入参数{in}	
timer_periph	TIMER外设
TIMERx(x=0,7)	参考具体参数
输入参数{in}	

deadtime	死区时间，0~255
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* configure the TIMER falling edge dead time */
```

```
timer_dead_time_falling_edge_config (TIMER0, 255);
```

函数 timer_dead_time_different_config

函数timer_dead_time_different_config描述见下表：

表 3-762. 函数 timer_dead_time_different_config

函数名称	timer_dead_time_different_config
函数原型	void timer_dead_time_different_config(uint32_t timer_periph, ControlStatus newvalue);
功能描述	配置TIMER的不同死区时间功能
先决条件	-
被调用函数	-
输入参数{in}	
timer_periph	TIMER外设
TIMERx(x=0,7)	参考具体参数
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：

```
/* configure the TIMER dead time different function */
```

```
timer_dead_time_different_config (TIMER0, ENABLE);
```

函数 timer_channel_break_control_config

函数timer_channel_break_control_config描述见下表：

表 3-763. 函数 timer_channel_break_control_config

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

	channel, ControlStatus newvalue);
功能描述	配置TIMER通道的中止功能
先决条件	-
被调用函数	-
输入参数{in}	
timer_periph	TIMER外设
TIMERx(x=0,7)	参考具体参数
输入参数{in}	
channel	TIMER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}	
newvalue	控制状态
ENABLE	使能
DISABLE	禁能
输出参数{out}	
-	-
返回值	
-	-

例如：

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

```
timer_channel_break_control_config (TIMER0, TIMER_CH_0, ENABLE);
```

函数 timer_channel_dead_time_config

函数timer_channel_dead_time_config描述见下表：

表 3-764. 函数 timer_channel_dead_time_config

函数名称	timer_channel_dead_time_config
函数原型	void timer_channel_dead_time_config(uint32_t timer_periph, uint32_t channel, ControlStatus newvalue);
功能描述	配置TIMER通道的死区功能
先决条件	-
被调用函数	-
输入参数{in}	
timer_periph	TIMER外设
TIMERx(x=0,7)	参考具体参数
输入参数{in}	
channel	TIMER通道
TIMER_CH_0	通道0

<i>TIMER_CH_1</i>	通道1
<i>TIMER_CH_2</i>	通道2
<i>TIMER_CH_3</i>	通道3
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:

```
/* configure TIMER0 channel 0 free dead time function */
```

```
timer_channel_dead_time_config (TIMER0, TIMER_CH_0, ENABLE);
```

函数 timer_channel_break_config

函数timer_channel_break_config描述见下表:

表 3-765. 函数 timer_channel_break_config

函数名称	timer_channel_break_config
函数原型	void timer_channel_break_config(uint32_t timer_periph, uint32_t broken, uint32_t breakpolarity, uint32_t break0filter);
功能描述	配置TIMER通道中止
先决条件	-
被调用函数	-
输入参数{in}	
timer_periph	TIMER外设
<i>TIMERx(x=0,7)</i>	参考具体参数
输入参数{in}	
broken	TIMER通道中止使能
<i>TIMER_CH_BREAK_ENABLE</i>	通道中止输入信号使能
<i>TIMER_CH_BREAK_DISABLE</i>	通道中止输入信号禁能
输入参数{in}	
breakpolarity	TIMER通道中止输入信号极性
<i>TIMER_CH_BREAK_POLARITY_HIGH</i>	通道中止输入高有效
<i>TIMER_CH_BREAK_POLARITY_LOW</i>	通道中止输入低有效
输入参数{in}	
break0filter	通道中止输入滤波(0~15)

输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* configure TIMER0 channel break */
```

```
timer_channel_break_config (TIMER0, TIMER_CH_0, TIMER_CH_BREAK_ENABLE ,  
TIMER_CH_BREAK_POLARITY_LOW, 15);
```

函数 timer_channel_break_external_status_config

函数timer_channel_break_external_status_config描述见下表：

表 3-766. 函数 timer_channel_break_external_status_config

函数名称	timer_channel_break_external_status_config
函数原型	void timer_channel_break_external_status_config (uint32_t timer_periph, uint32_t channel, ControlStatus newvalue);
功能描述	配置TIMER通道中止外部输入使能
先决条件	-
被调用函数	-
输入参数{in}	
timer_periph	TIMER外设
<i>TIMERx(x=0,7)</i>	参考具体参数
输入参数{in}	
channel	TIMER通道
<i>TIMER_CH_0</i>	TIMER通道0
<i>TIMER_CH_1</i>	TIMER通道1
<i>TIMER_CH_2</i>	TIMER通道2
输入参数{in}	
newvalue	控制值
<i>ENABLE</i>	使能
<i>DISABLE</i>	禁能
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* configure TIMER0 channel0 break external input enable */
```

```
timer_channel_break_external_status_config(TIMER0, TIMER_CH_0, ENABLE);
```

函数 timer_channel_break_external_polarity_config

函数timer_channel_break_external_polarity_config描述见下表：

表 3-767. 函数 timer_channel_break_external_polarity_config

函数名称	timer_channel_break_external_polarity_config
函数原型	void timer_channel_break_external_polarity_config(uint32_t timer_periph, uint32_t channel, uint16_t bkinpolarity);
功能描述	配置TIMER通道中止外部输入极性
先决条件	-
被调用函数	-
输入参数{in}	
timer_periph	TIMER外设
TIMERx(x=0,7)	参考具体参数
输入参数{in}	
channel	TIMER通道
TIMER_CH_0	TIMER通道0
TIMER_CH_1	TIMER通道1
TIMER_CH_2	TIMER通道2
输入参数{in}	
bkinpolarity	通道x中止输入信号极性
TIMER_CHx_BREAK_IN_INV	通道x中止输入信号反相
TIMER_CHx_BREAK_IN_NOT_INV	通道x中止输入信号不反相
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* configure TIMER0 channel0 break external input polarity */
```

```
timer_channel_break_external_polarity_config(TIMER0, TIMER_CH_0,
TIMER_CHx_BREAK_IN_NOT_INV);
```

函数 timer_channel_primary_output_config

函数timer_channel_primary_output_config描述见下表：

表 3-768. 函数 timer_channel_primary_output_config

函数名称	timer_channel_primary_output_config
函数原型	void timer_channel_primary_output_config(uint32_t timer_periph, uint32_t channel, ControlStatus newvalue);
功能描述	配置TIMER通道主输出功能

先决条件	-
被调用函数	-
输入参数{in}	
timer_periph	TIMER外设
<i>TIMERx(x=0,7)</i>	参考具体参数
输入参数{in}	
channel	TIMER通道
<i>TIMER_CH_0</i>	TIMER通道0
<i>TIMER_CH_1</i>	TIMER通道1
<i>TIMER_CH_2</i>	TIMER通道2
输入参数{in}	
newvalue	控制值
<i>ENABLE</i>	使能
<i>DISABLE</i>	禁能
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* configure TIMER0 channel0 primary output function */
```

```
timer_channel_primary_output_config (TIMER0, TIMER_CH_0, ENABLE);
```

函数 timer_channel_break_period_config

函数timer_channel_break_period_config描述见下表：

表 3-769. 函数 timer_channel_break_period_config

函数名称	timer_channel_break_period_config
函数原型	void timer_channel_break_period_config(uint32_t timer_periph, uint16_t value);
功能描述	配置TIMER通道中止周期值
先决条件	-
被调用函数	-
输入参数{in}	
timer_periph	TIMER外设
<i>TIMERx(x=0,7)</i>	参考具体参数
输入参数{in}	
value	通道中止周期值，0~0xFFFF
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* configure TIMER0 channel break period value */
timer_channel_break_period_config(TIMER0, 0xFFFF);
```

函数 timer_watchdog_value_config

函数timer_watchdog_value_config描述见下表：

表 3-770. 函数 timer_watchdog_value_config

函数名称	timer_watchdog_value_config
函数原型	void timer_watchdog_value_config(uint32_t timer_periph, uint32_t value);
功能描述	正交译码器信号断线检测看门狗计数器值配置
先决条件	-
被调用函数	-
输入参数{in}	
timer_periph	TIMER外设
TIMERx(x=1~4)	TIMER外设选择
输入参数{in}	
value	看门狗计数器周期值，0~0xFFFFFFFF
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* configure quadrature decoder signal disconnection detection watchdog value */
timer_watchdog_value_config(TIMER0, 3000);
```

函数 timer_watchdog_value_read

函数timer_watchdog_value_read描述见下表：

表 3-771. 函数 timer_watchdog_value_read

函数名称	timer_watchdog_value_read
函数原型	uint32_t timer_watchdog_value_read(uint32_t timer_periph);
功能描述	读取正交译码器信号断线检测看门狗计数器值
先决条件	-
被调用函数	-
输入参数{in}	
timer_periph	TIMER外设
TIMERx(x=1~4)	TIMER外设选择
输出参数{out}	
-	-
返回值	

uint32_t	看门狗计数器周期值，0~0xFFFFFFFF
-----------------	------------------------

例如：

```
/* read quadrature decoder signal disconnection detection watchdog value */
```

```
uint32_t i = 0;
```

```
i = timer_watchdog_value_read(TIMER0);
```

函数 timer_decoder_disconnection_detection_config

函数timer_decoder_disconnection_detection_config描述见下表：

表 3-772. 函数 timer_decoder_disconnection_detection_config

函数名称	timer_decoder_disconnection_detection_config
函数原型	void timer_decoder_disconnection_detection_config(uint32_t timer_periph, ControlStatus newvalue);
功能描述	正交编码器信号断线检测功能配置
先决条件	-
被调用函数	-
输入参数{in}	
timer_periph	TIMER外设
TIMERx(x=1~4)	参考具体参数
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：

```
/* configure quadrature decoder signal disconnection detection function */
```

```
timer_decoder_disconnection_detection_config(TIMER0, ENABLE);
```

函数 timer_decoder_jump_detection_config

函数timer_decoder_jump_detection_config描述见下表：

表 3-773. 函数 timer_decoder_jump_detection_config

函数名称	timer_decoder_jump_detection_config
函数原型	void timer_decoder_jump_detection_config(uint32_t timer_periph, ControlStatus newvalue);
功能描述	正交编码器信号跳变检测功能配置
先决条件	-

被调用函数	-
输入参数{in}	
timer_periph	TIMER外设
TIMERx(x=1~4)	参考具体参数
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：

```
/* configure quadrature decoder signal jump detection function */
```

```
timer_decoder_jump_detection_config(TIMER0, ENABLE);
```

函数 timer_counter_initial_register_config

函数timer_counter_initial_register_config描述见下表：

表 3-774. 函数 timer_counter_initial_register_config

函数名称	timer_counter_initial_register_config
函数原型	void timer_counter_initial_register_config(uint32_t timer_periph, ControlStatus newvalue);
功能描述	配置计数器初始值寄存器
先决条件	-
被调用函数	-
输入参数{in}	
timer_periph	TIMER外设
TIMERx(x=0,7,15,16)	TIMER外设选择
输入参数{in}	
newvalue	控制状态
ENABLE	使能
DISABLE	禁能
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* configure counter initial value register */
```

timer_counter_initial_register_config (TIMER0, ENABLE);

函数 timer_counter_initial_config

函数timer_counter_initial_config描述见下表:

表 3-775. 函数 timer_counter_initial_config

函数名称	timer_counter_initial_config
函数原型	void timer_counter_initial_config(uint32_t timer_periph, uint32_t value, uint32_t direction);
功能描述	配置计数器初始值和计数方向
先决条件	-
被调用函数	-
输入参数{in}	
timer_periph	TIMER外设
TIMERx(x=0,7,15,16)	TIMER外设选择
输入参数{in}	
value	计数器初始值, 0~0xFFFF
输入参数{in}	
direction	计数器初始计数方向
TIMER_INITIAL_DIRECTION_DOWN	当同步事件发生时, 计数器向下计数
TIMER_INITIAL_DIRECTION_UP	当同步事件发生时, 计数器向上计数
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* configure counter initial value and direction */
```

```
timer_counter_initial_config (TIMER0, TIMER_INITIAL_DIRECTION_UP);
```

函数 timer_synchronization_event_generate

函数timer_synchronization_event_generate描述见下表:

表 3-776. 函数 timer_synchronization_event_generate

函数名称	timer_synchronization_event_generate
函数原型	void timer_synchronization_event_generate(uint32_t timer_periph);
功能描述	产生软件同步事件
先决条件	-
被调用函数	-
输入参数{in}	

timer_periph	TIMER外设
<i>TIMERx</i> (<i>x</i> =0, 7, 15, 16)	TIMER外设选择
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* generate soft synchronization event */
```

```
timer_synchronization_event_generate (TIMER0);
```

函数 timer_flag_get

函数timer_flag_get描述见下表:

表 3-777. 函数 timer_flag_get

函数名称	timer_flag_get
函数原型	FlagStatus timer_flag_get(uint32_t timer_periph, uint32_t flag);
功能描述	获取外设TIMERx的状态标志
先决条件	-
被调用函数	-
输入参数{in}	
timer_periph	TIMER外设
<i>TIMERx</i> (<i>x</i> =0~7, 15, 16)	参考具体参数
输入参数{in}	
flag	状态标志
<i>TIMER_FLAG_UP</i>	更新标志, <i>TIMERx</i> (<i>x</i> =0~7, 15, 16)
<i>TIMER_FLAG_CH0</i>	通道0比较/捕获标志, <i>TIMERx</i> (<i>x</i> =0~4, 7, 15, 16)
<i>TIMER_FLAG_CH1</i>	通道1比较/捕获标志, <i>TIMERx</i> (<i>x</i> =0~4, 7, 15, 16)
<i>TIMER_FLAG_CH2</i>	通道2比较/捕获标志, <i>TIMERx</i> (<i>x</i> =0~4, 7)
<i>TIMER_FLAG_CH3</i>	通道3比较/捕获标志, <i>TIMERx</i> (<i>x</i> =0~4, 7)
<i>TIMER_FLAG_CMT</i>	通道换相更新标志, <i>TIMERx</i> (<i>x</i> =0, 7, 15, 16)
<i>TIMER_FLAG_TRG</i>	触发标志, <i>TIMERx</i> (<i>x</i> =0~4, 7, 15, 16)
<i>TIMER_FLAG_BRK0</i>	BREAK标志, <i>TIMERx</i> (<i>x</i> =0, 7, 15, 16)
<i>TIMER_FLAG_CH0O</i>	通道0捕获溢出标志, <i>TIMERx</i> (<i>x</i> =0~4, 7, 15, 16)
<i>TIMER_FLAG_CH1O</i>	通道1捕获溢出标志, <i>TIMERx</i> (<i>x</i> =0~4, 7, 15, 16)
<i>TIMER_FLAG_CH2O</i>	通道2捕获溢出标志, <i>TIMERx</i> (<i>x</i> =0~4, 7)
<i>TIMER_FLAG_CH3O</i>	通道3捕获溢出标志, <i>TIMERx</i> (<i>x</i> =0~4, 7)

O	
TIMER_FLAG_SYSB	系统源中止标志, TIMERx(x=0,7)
TIMER_FLAG_DECJ	正交编码器跳变标志, TIMERx(x=1~4)
TIMER_FLAG_DECDIS	正交编码器断线标志, TIMERx(x=1~4)
TIMER_FLAG_MCH0	多模式通道0比较/捕获标志, TIMERx(x=15,16)
TIMER_FLAG_MCH00	多模式通道0捕获溢出标志, TIMERx(x=15,16)
TIMER_FLAG_CH0COMADD	通道0附加比较标志, TIMERx(x=0,7,15,16)
TIMER_FLAG_CH1COMADD	通道1附加比较标志, TIMERx(x=0,7,15,16)
TIMER_FLAG_CH2COMADD	通道2附加比较标志, TIMERx(x=0,7)
TIMER_FLAG_CH3COMADD	通道3附加比较标志, TIMERx(x=0,7)
TIMER_FLAG_CH0BRKM	通道0中止多周期中断标志, TIMERx(x=0,7)
TIMER_FLAG_CH1BRKM	通道1中止多周期中断标志, TIMERx(x=0,7)
TIMER_FLAG_CH2BRKM	通道2中止多周期中断标志, TIMERx(x=0,7)
TIMER_FLAG_CH0BRK	通道0中止中断标志, TIMERx(x=0,7)
TIMER_FLAG_CH1BRK	通道1中止中断标志, TIMERx(x=0,7)
TIMER_FLAG_CH2BRK	通道2中止中断标志, TIMERx(x=0,7)
输出参数{out}	
-	-
返回值	
FlagStatus	SET或者RESET

例如:

```
/* get TIMER0 update flags */
```

```
FlagStatus Flag_status = RESET;
```

```
Flag_status = timer_flag_get(TIMER0, TIMER_FLAG_UP);
```

函数 timer_flag_clear

函数timer_flag_clear描述见下表:

表 3-778. 函数 timer_flag_clear

函数名称	timer_flag_clear
函数原型	void timer_flag_clear(uint32_t timer_periph, uint32_t flag);
功能描述	清除外设TIMERx状态标志
先决条件	-
被调用函数	-
输入参数{in}	
timer_periph	TIMER外设
TIMERx(x=0~7,15,16)	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=0~7,15,16)
TIMER_FLAG_CH0	通道0比较/捕获标志, TIMERx(x=0~4,7,15,16)
TIMER_FLAG_CH1	通道1比较/捕获标志, TIMERx(x=0~4,7,15,16)
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=0,7,15,16)
TIMER_FLAG_TRG	触发标志, TIMERx(x=0~4,7,15,16)
TIMER_FLAG_BRK0	BREAK标志, TIMERx(x=0,7,15,16)
TIMER_FLAG_CH0O	通道0捕获溢出标志, TIMERx(x=0~4,7,15,16)
TIMER_FLAG_CH1O	通道1捕获溢出标志, TIMERx(x=0~4,7,15,16)
TIMER_FLAG_CH2O	通道2捕获溢出标志, TIMERx(x=0~4,7)
TIMER_FLAG_CH3O	通道3捕获溢出标志, TIMERx(x=0~4,7)
TIMER_FLAG_SYSB	系统源中止标志, TIMERx(x=0,7)
TIMER_FLAG_DECJ	正交编码器跳变标志, TIMERx(x=1~4)
TIMER_FLAG_DECDIS	正交编码器断线标志, TIMERx(x=1~4)
TIMER_FLAG_MCH0	多模式通道0比较/捕获标志, TIMERx(x=15,16)
TIMER_FLAG_MCH0O	多模式通道0捕获溢出标志, TIMERx(x=15,16)
TIMER_FLAG_CH0	通道0附加比较标志, TIMERx(x=0,7,15,16)

COMADD	
TIMER_FLAG_CH1 COMADD	通道1附加比较标志, TIMERx(x=0,7,15,16)
TIMER_FLAG_CH2 COMADD	通道2附加比较标志, TIMERx(x=0,7)
TIMER_FLAG_CH3 COMADD	通道3附加比较标志, TIMERx(x=0,7)
TIMER_FLAG_CH0 BRKM	通道0中止多周期中断标志, TIMERx(x=0,7)
TIMER_FLAG_CH1 BRKM	通道1中止多周期中断标志, TIMERx(x=0,7)
TIMER_FLAG_CH2 BRKM	通道2中止多周期中断标志, TIMERx(x=0,7)
TIMER_FLAG_CH0 BRK	通道0中止中断标志, TIMERx(x=0,7)
TIMER_FLAG_CH1 BRK	通道1中止中断标志, TIMERx(x=0,7)
TIMER_FLAG_CH2 BRK	通道2中止中断标志, TIMERx(x=0,7)
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* clear TIMER0 update flags */
```

```
timer_flag_clear(TIMER0, TIMER_FLAG_UP);
```

函数 timer_interrupt_enable

函数timer_interrupt_enable描述见下表:

表 3-779. 函数 timer_interrupt_enable

函数名称	timer_interrupt_enable
函数原型	void timer_interrupt_enable(uint32_t timer_periph, uint32_t interrupt);
功能描述	外设TIMERx中断使能
先决条件	-
被调用函数	-
输入参数{in}	
timer_periph	TIMER外设
TIMERx(x=0~7, 15, 16)	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> (<i>x</i> =0~7,15,16)
<i>TIMER_INT_CH0</i>	通道0比较/捕获中断, <i>TIMERx</i> (<i>x</i> =0~4,7,15,16)
<i>TIMER_INT_CH1</i>	通道1比较/捕获中断, <i>TIMERx</i> (<i>x</i> =0~4,7,15,16)
<i>TIMER_INT_CH2</i>	通道2比较/捕获中断, <i>TIMERx</i> (<i>x</i> =0~4,7)
<i>TIMER_INT_CH3</i>	通道3比较/捕获中断, <i>TIMERx</i> (<i>x</i> =0~4,7)
<i>TIMER_INT_CMT</i>	换相更新中断, <i>TIMERx</i> (<i>x</i> =0,7,15,16)
<i>TIMER_INT_TRG</i>	触发中断, <i>TIMERx</i> (<i>x</i> =0~4,7,15,16)
<i>TIMER_INT_BRK</i>	中止中断, <i>TIMERx</i> (<i>x</i> =0,7,15,16)
<i>TIMER_INT_DECJ</i>	正交编码器跳变中断, <i>TIMERx</i> (<i>x</i> =1~4)
<i>TIMER_INT_DECDIS</i>	正交编码器断线中断, <i>TIMERx</i> (<i>x</i> =1~4)
<i>TIMER_INT_MCH0</i>	多模式通道0比较/捕获中断, <i>TIMERx</i> (<i>x</i> =15,16)
<i>TIMER_INT_CH0COMADD</i>	通道0附加比较中断, <i>TIMERx</i> (<i>x</i> =0,7,15,16)
<i>TIMER_INT_CH1COMADD</i>	通道1附加比较中断, <i>TIMERx</i> (<i>x</i> =0,7,15,16)
<i>TIMER_INT_CH2COMADD</i>	通道2附加比较中断, <i>TIMERx</i> (<i>x</i> =0,7)
<i>TIMER_INT_CH3COMADD</i>	通道3附加比较中断, <i>TIMERx</i> (<i>x</i> =0,7)
<i>TIMER_INT_CH0BRKM</i>	通道0中止多周期中断, <i>TIMERx</i> (<i>x</i> =0,7)
<i>TIMER_INT_CH1BRKM</i>	通道1中止多周期中断, <i>TIMERx</i> (<i>x</i> =0,7)
<i>TIMER_INT_CH2BRKM</i>	通道2中止多周期中断, <i>TIMERx</i> (<i>x</i> =0,7)
<i>TIMER_INT_CH0BRK</i>	通道0中止中断, <i>TIMERx</i> (<i>x</i> =0,7)
<i>TIMER_INT_CH1BRK</i>	通道1中止中断, <i>TIMERx</i> (<i>x</i> =0,7)
<i>TIMER_INT_CH2BRK</i>	通道2中止中断, <i>TIMERx</i> (<i>x</i> =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 3-780. 函数 timer_interrupt_disable

函数名称	timer_interrupt_disable
函数原型	void timer_interrupt_disable (uint32_t timer_periph, uint32_t interrupt);
功能描述	外设TIMERx中断禁能
先决条件	-
被调用函数	-
输入参数{in}	
timer_periph	TIMER外设
TIMERx(x=0~7,15,16)	参考具体参数
输入参数{in}	
interrupt	中断源
TIMER_INT_UP	更新中断, TIMERx(x=0~7,15,16)
TIMER_INT_CH0	通道0比较/捕获中断, TIMERx(x=0~4,7,15,16)
TIMER_INT_CH1	通道1比较/捕获中断, TIMERx(x=0~4,7,15,16)
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=0,7,15,16)
TIMER_INT_TRG	触发中断, TIMERx(x=0~4,7,15,16)
TIMER_INT_BRK	中止中断, TIMERx(x=0,7,15,16)
TIMER_INT_DECJ	正交编码器跳变中断, TIMERx(x=1~4)
TIMER_INT_DECDIS	正交编码器断线中断, TIMERx(x=1~4)
TIMER_INT_MCH0	多模式通道0比较/捕获中断, TIMERx(x=15,16)
TIMER_INT_CH0C0MADD	通道0附加比较中断, TIMERx(x=0,7,15,16)
TIMER_INT_CH0C1MADD	通道1附加比较中断, TIMERx(x=0,7,15,16)
TIMER_INT_CH0C2MADD	通道2附加比较中断, TIMERx(x=0,7)
TIMER_INT_CH0C3MADD	通道3附加比较中断, TIMERx(x=0,7)
TIMER_INT_CH0BRKM	通道0中止多周期中断, TIMERx(x=0,7)
TIMER_INT_CH1BRKM	通道1中止多周期中断, TIMERx(x=0,7)
TIMER_INT_CH2BRKM	通道2中止多周期中断, TIMERx(x=0,7)
TIMER_INT_CH0BRK	通道0中止中断, TIMERx(x=0,7)

<i>TIMER_INT_CH1B</i> <i>RK</i>	通道1中止中断, <i>TIMERx</i> (<i>x</i> =0,7)
<i>TIMER_INT_CH2B</i> <i>RK</i>	通道2中止中断, <i>TIMERx</i> (<i>x</i> =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 3-781. 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 int_flag);
功能描述	获取外设 <i>TIMERx</i> 中断标志
先决条件	-
被调用函数	-
输入参数{in}	
timer_periph	<i>TIMER</i> 外设
<i>TIMERx</i> (<i>x</i> =0~7,15,16)	参考具体参数
输入参数{in}	
int_flag	中断源
<i>TIMER_INT_FLAG_UP</i>	更新中断标志, <i>TIMERx</i> (<i>x</i> =0~7,15,16)
<i>TIMER_INT_FLAG_CH0</i>	通道0比较/捕获中断标志, <i>TIMERx</i> (<i>x</i> =0~4,7,15,16)
<i>TIMER_INT_FLAG_CH1</i>	通道1比较/捕获中断标志, <i>TIMERx</i> (<i>x</i> =0~4,7,15,16)
<i>TIMER_INT_FLAG_CH2</i>	通道2比较/捕获中断标志, <i>TIMERx</i> (<i>x</i> =0~4,7)
<i>TIMER_INT_FLAG_CH3</i>	通道3比较/捕获中断标志, <i>TIMERx</i> (<i>x</i> =0~4,7)
<i>TIMER_INT_FLAG_CMT</i>	换相更新中断标志, <i>TIMERx</i> (<i>x</i> =0,7,15,16)
<i>TIMER_INT_FLAG_TRG</i>	触发中断标志, <i>TIMERx</i> (<i>x</i> =0~4,7,15,16)
<i>TIMER_INT_FLAG_</i>	<i>BREAK0</i> 中断标志, <i>TIMERx</i> (<i>x</i> =0,7,15,16)

<i>BRK0</i>	
<i>TIMER_INT_FLAG_SYSB</i>	系统源中止中断标志，TIMERx(x=0,7)
<i>TIMER_INT_FLAG_DECJ</i>	正交编码器跳变中断标志，TIMERx(x=1~4)
<i>TIMER_INT_FLAG_DECDIS</i>	正交编码器断线中断标志，TIMERx(x=1~4)
<i>TIMER_INT_FLAG_MCH0</i>	多模式通道0比较/捕获中断标志，TIMERx(x=15,16)
<i>TIMER_INT_FLAG_CH0COMADD</i>	通道0附加比较中断标志，TIMERx(x=0,7,15,16)
<i>TIMER_INT_FLAG_CH1COMADD</i>	通道1附加比较中断标志，TIMERx(x=0,7,15,16)
<i>TIMER_INT_FLAG_CH2COMADD</i>	通道2附加比较中断标志，TIMERx(x=0,7)
<i>TIMER_INT_FLAG_CH3COMADD</i>	通道3附加比较中断标志，TIMERx(x=0,7)
<i>TIMER_INT_FLAG_CH0BRKM</i>	通道0中止多周期中断标志位，TIMERx(x=0,7)
<i>TIMER_INT_FLAG_CH1BRKM</i>	通道1中止多周期中断标志位，TIMERx(x=0,7)
<i>TIMER_INT_FLAG_CH2BRKM</i>	通道2中止多周期中断标志位，TIMERx(x=0,7)
<i>TIMER_INT_FLAG_CH0BRK</i>	通道0中止中断标志位，TIMERx(x=0,7)
<i>TIMER_INT_FLAG_CH1BRK</i>	通道1中止中断标志位，TIMERx(x=0,7)
<i>TIMER_INT_FLAG_CH2BRK</i>	通道2中止中断标志位，TIMERx(x=0,7)
输出参数{out}	
-	-
返回值	
FlagStatus	SET或者RESET

例如：

```
/* get TIMER0 update interrupt flag */
```

```
FlagStatus Flag_interrupt = RESET;
```

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

函数 timer_interrupt_flag_clear

函数timer_interrupt_flag_clear描述见下表：

表 3-782. 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 int_flag);
功能描述	清除外设TIMERx的中断标志
先决条件	-
被调用函数	-
输入参数{in}	
timer_periph	TIMER外设
TIMERx(x=0~7, 15, 16)	参考具体参数
输入参数{in}	
int_flag	中断源
TIMER_INT_FLAG_UP	update interrupt flag, TIMERx(x=0~7, 15, 16)
TIMER_INT_FLAG_CH0	channel 0 capture or compare interrupt flag, TIMERx(x=0~4, 7, 15, 16)
TIMER_INT_FLAG_CH1	channel 1 capture or compare interrupt flag, TIMERx(x=0~4, 7, 15, 16)
TIMER_INT_FLAG_CH2	channel 2 capture or compare interrupt flag, TIMERx(x=0~4, 7)
TIMER_INT_FLAG_CH3	channel 3 capture or compare interrupt flag, TIMERx(x=0~4, 7)
TIMER_INT_FLAG_CMT	channel commutation interrupt flag, TIMERx(x=0, 7, 15, 16)
TIMER_INT_FLAG_TRG	trigger interrupt flag, TIMERx(x=0~4, 7, 15, 16)
TIMER_INT_FLAG_BRK0	BREAK0 interrupt flag, TIMERx(x=0, 7, 15, 16)
TIMER_INT_FLAG_SYSB	system source break interrupt flag, TIMERx(x=0, 7)
TIMER_INT_FLAG_DECJ	quadrature decoder signal jump interrupt flag, TIMERx(x=1~4)
TIMER_INT_FLAG_DECDIS	quadrature decoder signal disconnection interrupt flag, TIMERx(x=1~4)
TIMER_INT_FLAG_MCH0	multi mode channel 0 capture or compare interrupt flag, TIMERx(x=15, 16)
TIMER_INT_FLAG_CH0COMADD	channel 0 additional compare interrupt flag, TIMERx(x=0, 7, 15, 16)
TIMER_INT_FLAG_CH1COMADD	channel 1 additional compare interrupt flag, TIMERx(x=0, 7, 15, 16)
TIMER_INT_FLAG_CH2COMADD	channel 2 additional compare interrupt flag, TIMERx(x=0, 7)
TIMER_INT_FLAG_CH3COMADD	channel 3 additional compare interrupt flag, TIMERx(x=0, 7)

<i>CH3COMADD</i>	
<i>TIMER_INT_FLAG_</i> <i>CH0BRKM</i>	channel 0 BREAK multi-period interrupt flag, <i>TIMERx</i> (<i>x</i> =0,7)
<i>TIMER_INT_FLAG_</i> <i>CH1BRKM</i>	channel 1 BREAK multi-period interrupt flag, <i>TIMERx</i> (<i>x</i> =0,7)
<i>TIMER_INT_FLAG_</i> <i>CH2BRKM</i>	channel 2 BREAK multi-period interrupt flag, <i>TIMERx</i> (<i>x</i> =0,7)
<i>TIMER_INT_FLAG_</i> <i>CH0BRK</i>	channel 0 BREAK interrupt flag, <i>TIMERx</i> (<i>x</i> =0,7)
<i>TIMER_INT_FLAG_</i> <i>CH1BRK</i>	channel 1 BREAK interrupt flag, <i>TIMERx</i> (<i>x</i> =0,7)
<i>TIMER_INT_FLAG_</i> <i>CH2BRK</i>	channel 2 BREAK interrupt flag, <i>TIMERx</i> (<i>x</i> =0,7)
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* clear TIMER0 update interrupt flag */
```

```
timer_interrupt_flag_clear(TIMER0, TIMER_INT_FLAG_UP);
```

3.26. TRIGSEL

TRIGSEL 是 MCU 中的触发选择控制器。可通过软件配置的方式，为各种外设选择触发输入信号。章节 [3.26.1](#) 描述了 TRIGSEL 的寄存器列表，章节 [3.26.2](#) 对 TRIGSEL 库函数进行说明。

3.26.1. 外设寄存器说明

TRIGSEL 寄存器列表如下表所示：

表 3-783. TRIGSEL 寄存器

寄存器名称	寄存器描述
TRIGSEL_EXTOUT0	EXTOUT 触发选择寄存器 0
TRIGSEL_EXTOUT1	EXTOUT 触发选择寄存器 1
TRIGSEL_EXTOUT2	EXTOUT 触发选择寄存器 2
TRIGSEL_EXTOUT3	EXTOUT 触发选择寄存器 3
TRIGSEL_TIMER0ITI	TIMER0_ITI 触发选择寄存器
TRIGSEL_TIMER1ITI	TIMER1_ITI 触发选择寄存器
TRIGSEL_TIMER2ITI	TIMER2_ITI 触发选择寄存器
TRIGSEL_TIMER3ITI	TIMER3_ITI 触发选择寄存器
TRIGSEL_TIMER4ITI	TIMER4_ITI 触发选择寄存器
TRIGSEL_TIMER7ITI	TIMER7_ITI 触发选择寄存器
TRIGSEL_TIMER15ITI	TIMER15_ITI 触发选择寄存器
TRIGSEL_TIMER16ITI	TIMER16_ITI 触发选择寄存器
TRIGSEL_DAC	DAC 触发选择寄存器
TRIGSEL_ADC0_ROUTRG	ADC0 规则通道触发选择寄存器
TRIGSEL_ADC0_INSTRG	ADC0 插入通道触发选择寄存器
TRIGSEL_ADC1_ROUTRG	ADC1 规则通道触发选择寄存器
TRIGSEL_ADC1_INSTRG	ADC1 插入通道触发选择寄存器
TRIGSEL_ADC2_ROUTRG	ADC2 规则通道触发选择寄存器
TRIGSEL_ADC2_INSTRG	ADC2 插入通道触发选择寄存器
TRIGSEL_TIMER0BRKIN	TIMER0_BRKIN 触发选择寄存器
TRIGSEL_TIMER0CHBRKIN	TIMER0 通道刹车触发选择寄存器
TRIGSEL_TIMER7BRKIN	TIMER7_BRKIN 触发选择寄存器
TRIGSEL_TIMER7CHBRKIN	TIMER7 通道刹车触发选择寄存器
TRIGSEL_TIMER15BRKIN	TIMER15_BRKIN 触发选择寄存器
TRIGSEL_TIMER16BRKIN	TIMER16_BRKIN 触发选择寄存器
TRIGSEL_EXTOUT0	EXTOUT 触发选择寄存器 0
TRIGSEL_EXTOUT1	EXTOUT 触发选择寄存器 1
TRIGSEL_EXTOUT2	EXTOUT 触发选择寄存器 2
TRIGSEL_EXTOUT3	EXTOUT 触发选择寄存器 3
TRIGSEL_TIMER0ITI	TIMER0_ITI 触发选择寄存器
TRIGSEL_TIMER1ITI	TIMER1_ITI 触发选择寄存器

寄存器名称	寄存器描述
TRIGSEL_TIMER2ITI	TIMER2_ITI 触发选择寄存器
TRIGSEL_TIMER3ITI	TIMER3_ITI 触发选择寄存器
TRIGSEL_TIMER4ITI	TIMER4_ITI 触发选择寄存器
TRIGSEL_TIMER7ITI	TIMER7_ITI 触发选择寄存器
TRIGSEL_TIMER15ITI	TIMER15_ITI 触发选择寄存器
TRIGSEL_TIMER16ITI	TIMER16_ITI 触发选择寄存器
TRIGSEL_DAC	DAC 触发选择寄存器
TRIGSEL_ADC0_ROUTRG	ADC0 规则通道触发选择寄存器
TRIGSEL_ADC0_INSTRG	ADC0 插入通道触发选择寄存器
TRIGSEL_ADC1_ROUTRG	ADC1 规则通道触发选择寄存器
TRIGSEL_ADC1_INSTRG	ADC1 插入通道触发选择寄存器
TRIGSEL_ADC2_ROUTRG	ADC2 规则通道触发选择寄存器
TRIGSEL_ADC2_INSTRG	ADC2 插入通道触发选择寄存器
TRIGSEL_TIMER0BRKIN	TIMER0_BRKIN 触发选择寄存器
TRIGSEL_TIMER0CHBRKIN	TIMER0 通道刹车触发选择寄存器

3.26.2. 外设库函数说明

TRIGSEL库函数列表如下表所示：

表 3-784. TRIGSEL 库函数

函数名称	功能描述
trigsel_deinit	复位 TRIGSEL
trigsel_init	为外设选择触发输入源
trigsel_trigger_source_get	获取外设的触发输入源
trigsel_register_lock_set	锁定触发寄存器
trigsel_register_lock_get	获取触发寄存器锁定状态

枚举类型 trigsel_source_enum

表 3-785. 枚举类型 trigsel_source_enum

成员名称	功能描述
TRIGSEL_INPUT_VSS	触发输入源 VSS
TRIGSEL_INPUT_VDD	触发输入源 VDD
TRIGSEL_INPUT_TRIGSEL_IN0	触发输入源 TRIGSEL_IN0 引脚
TRIGSEL_INPUT_TRIGSEL_IN1	触发输入源 TRIGSEL_IN1 引脚
TRIGSEL_INPUT_TRIGSEL_IN2	触发输入源 TRIGSEL_IN2 引脚
TRIGSEL_INPUT_TRIGSEL_IN3	触发输入源 TRIGSEL_IN3 引脚
TRIGSEL_INPUT_TRIGSEL_IN4	触发输入源 TRIGSEL_IN4 引脚
TRIGSEL_INPUT_TRIGSEL_IN5	触发输入源 TRIGSEL_IN5 引脚
TRIGSEL_INPUT_TRIGSEL_IN6	触发输入源 TRIGSEL_IN6 引脚
TRIGSEL_INPUT_TRIGSEL_IN7	触发输入源 TRIGSEL_IN7 引脚
TRIGSEL_INPUT_TIMER0_TRGO	触发输入源 TIMER0_TRGO

成员名称	功能描述
TRIGSEL_INPUT_TIMER0_CH0	触发输入源 TIMER0_CH0
TRIGSEL_INPUT_TIMER0_CH1	触发输入源 TIMER0_CH1
TRIGSEL_INPUT_TIMER0_CH2	触发输入源 TIMER0_CH2
TRIGSEL_INPUT_TIMER0_CH3	触发输入源 TIMER0_CH3
TRIGSEL_INPUT_TIMER7_TRGO	触发输入源 TIMER7_TRGO
TRIGSEL_INPUT_TIMER7_CH0	触发输入源 TIMER7_CH0
TRIGSEL_INPUT_TIMER7_CH1	触发输入源 TIMER7_CH1
TRIGSEL_INPUT_TIMER7_CH2	触发输入源 TIMER7_CH2
TRIGSEL_INPUT_TIMER7_CH3	触发输入源 TIMER7_CH3
TRIGSEL_INPUT_TIMER5_TRGO	触发输入源 TIMER5_TRGO
TRIGSEL_INPUT_TIMER6_TRGO	触发输入源 TIMER6_TRGO
TRIGSEL_INPUT_TIMER1_TRGO	触发输入源 TIMER1_TRGO
TRIGSEL_INPUT_TIMER1_CH0	触发输入源 TIMER1_CH0
TRIGSEL_INPUT_TIMER1_CH1	触发输入源 TIMER1_CH1
TRIGSEL_INPUT_TIMER1_CH2	触发输入源 TIMER1_CH2
TRIGSEL_INPUT_TIMER1_CH3	触发输入源 TIMER1_CH3
TRIGSEL_INPUT_TIMER2_TRGO	触发输入源 TIMER2_TRGO
TRIGSEL_INPUT_TIMER2_CH0	触发输入源 TIMER2_CH0
TRIGSEL_INPUT_TIMER2_CH1	触发输入源 TIMER2_CH1
TRIGSEL_INPUT_TIMER2_CH2	触发输入源 TIMER2_CH2
TRIGSEL_INPUT_TIMER2_CH3	触发输入源 TIMER2_CH3
TRIGSEL_INPUT_TIMER3_TRGO	触发输入源 TIMER3_TRGO
TRIGSEL_INPUT_TIMER3_CH0	触发输入源 TIMER3_CH0
TRIGSEL_INPUT_TIMER3_CH1	触发输入源 TIMER3_CH1
TRIGSEL_INPUT_TIMER3_CH2	触发输入源 TIMER3_CH2
TRIGSEL_INPUT_TIMER3_CH3	触发输入源 TIMER3_CH3
TRIGSEL_INPUT_TIMER4_TRGO	触发输入源 TIMER4_TRGO
TRIGSEL_INPUT_TIMER4_CH0	触发输入源 TIMER4_CH0
TRIGSEL_INPUT_TIMER4_CH1	触发输入源 TIMER4_CH1
TRIGSEL_INPUT_TIMER4_CH2	触发输入源 TIMER4_CH2
TRIGSEL_INPUT_TIMER4_CH3	触发输入源 TIMER4_CH3
TRIGSEL_INPUT_TIMER15_TRGO	触发输入源 TIMER15_TRGO
TRIGSEL_INPUT_TIMER15_CH0	触发输入源 TIMER15_CH0
TRIGSEL_INPUT_TIMER15_CH1	触发输入源 TIMER15_CH1
TRIGSEL_INPUT_TIMER15_MCH0	触发输入源 TIMER15_MCH0
TRIGSEL_INPUT_TIMER16_TRGO	触发输入源 TIMER16_TRGO
TRIGSEL_INPUT_TIMER16_CH0	触发输入源 TIMER16_CH0
TRIGSEL_INPUT_TIMER16_CH1	触发输入源 TIMER16_CH1
TRIGSEL_INPUT_TIMER16_MCH0	触发输入源 TIMER16_MCH0
TRIGSEL_INPUT_ADC0_WD0_OUT	触发输入源 ADC0_WD0_OUT
TRIGSEL_INPUT_ADC0_WD1_OUT	触发输入源 ADC0_WD1_OUT
TRIGSEL_INPUT_ADC0_WD2_OUT	触发输入源 ADC0_WD2_OUT

成员名称	功能描述
TRIGSEL_INPUT_ADC1_WD0_OUT	触发输入源 ADC1_WD0_OUT
TRIGSEL_INPUT_ADC1_WD1_OUT	触发输入源 ADC1_WD1_OUT
TRIGSEL_INPUT_ADC1_WD2_OUT	触发输入源 ADC1_WD2_OUT
TRIGSEL_INPUT_ADC2_WD0_OUT	触发输入源 ADC2_WD0_OUT
TRIGSEL_INPUT_ADC2_WD1_OUT	触发输入源 ADC2_WD1_OUT
TRIGSEL_INPUT_ADC2_WD2_OUT	触发输入源 ADC2_WD2_OUT
TRIGSEL_INPUT_CMP0_OUT	触发输入源 CMP0_OUT
TRIGSEL_INPUT_CK_OUT	触发输入源 CK_OUT
TRIGSEL_INPUT_TIMER0_BKIN	触发输入源 TIMER0_BKIN
TRIGSEL_INPUT_TIMER0_CH0BKIN	触发输入源 TIMER0_CH0BKIN
TRIGSEL_INPUT_TIMER0_CH1BKIN	触发输入源 TIMER0_CH1BKIN
TRIGSEL_INPUT_TIMER0_CH2BKIN	触发输入源 TIMER0_CH2BKIN
TRIGSEL_INPUT_TIMER7_BKIN	触发输入源 TIMER7_BKIN
TRIGSEL_INPUT_TIMER7_CH0BKIN	触发输入源 TIMER7_CH0BKIN
TRIGSEL_INPUT_TIMER7_CH1BKIN	触发输入源 TIMER7_CH1BKIN
TRIGSEL_INPUT_TIMER7_CH2BKIN	触发输入源 TIMER7_CH2BKIN
TRIGSEL_INPUT_TIMER15_BKIN	触发输入源 TIMER15_BKIN
TRIGSEL_INPUT_TIMER16_BKIN	触发输入源 TIMER16_BKIN
TRIGSEL_INPUT_EXTI9	触发输入源 EXTI9
TRIGSEL_INPUT_EXTI11	触发输入源 EXTI11
TRIGSEL_INPUT_EXTI15	触发输入源 EXTI15

枚举类型 `trigsel_periph_enum`

表 3-786. 枚举类型 `trigsel_periph_enum`

成员名称	功能描述
TRIGSEL_OUTPUT_TRIGSEL_OUT0	输出目标外设 TRIGSEL_OUT0 引脚
TRIGSEL_OUTPUT_TRIGSEL_OUT1	输出目标外设 TRIGSEL_OUT1 引脚
TRIGSEL_OUTPUT_TRIGSEL_OUT2	输出目标外设 TRIGSEL_OUT2 引脚
TRIGSEL_OUTPUT_TRIGSEL_OUT3	输出目标外设 TRIGSEL_OUT3 引脚
TRIGSEL_OUTPUT_TRIGSEL_OUT4	输出目标外设 TRIGSEL_OUT4 引脚
TRIGSEL_OUTPUT_TRIGSEL_OUT5	输出目标外设 TRIGSEL_OUT5 引脚
TRIGSEL_OUTPUT_TRIGSEL_OUT6	输出目标外设 TRIGSEL_OUT6 引脚
TRIGSEL_OUTPUT_TRIGSEL_OUT7	输出目标外设 TRIGSEL_OUT7 引脚
TRIGSEL_OUTPUT_TIMER0_ITI	输出目标外设 TIMER0_ITI
TRIGSEL_OUTPUT_TIMER1_ITI	输出目标外设 TIMER1_ITI
TRIGSEL_OUTPUT_TIMER2_ITI	输出目标外设 TIMER2_ITI
TRIGSEL_OUTPUT_TIMER3_ITI	输出目标外设 TIMER3_ITI
TRIGSEL_OUTPUT_TIMER4_ITI	输出目标外设 TIMER4_ITI
TRIGSEL_OUTPUT_TIMER7_ITI	输出目标外设 TIMER7_ITI
TRIGSEL_OUTPUT_TIMER15_ITI	输出目标外设 TIMER15_ITI
TRIGSEL_OUTPUT_TIMER16_ITI	输出目标外设 TIMER16_ITI

成员名称	功能描述
TRIGSEL_OUTPUT_DAC_OUT_EXTRG	输出目标外设 DAC_OUT_EXTRG
TRIGSEL_OUTPUT_ADC0_ROUTRG	输出目标外设 ADC0_ROUTRG
TRIGSEL_OUTPUT_ADC0_INSTRG	输出目标外设 ADC0_INSTRG
TRIGSEL_OUTPUT_ADC1_ROUTRG	输出目标外设 ADC1_ROUTRG
TRIGSEL_OUTPUT_ADC1_INSTRG	输出目标外设 ADC1_INSTRG
TRIGSEL_OUTPUT_ADC2_ROUTRG	输出目标外设 ADC2_ROUTRG
TRIGSEL_OUTPUT_ADC2_INSTRG	输出目标外设 ADC2_INSTRG
TRIGSEL_OUTPUT_TIMER0_BRKIN	输出目标外设 TIMER0_BRKIN
TRIGSEL_OUTPUT_TIMER0_CH0BRKIN	输出目标外设 TIMER0_CH0BRKIN
TRIGSEL_OUTPUT_TIMER0_CH1BRKIN	输出目标外设 TIMER0_CH1BRKIN
TRIGSEL_OUTPUT_TIMER0_CH2BRKIN	输出目标外设 TIMER0_CH2BRKIN
TRIGSEL_OUTPUT_TIMER7_BRKIN	输出目标外设 TIMER7_BRKIN
TRIGSEL_OUTPUT_TIMER7_CH0BRKIN	输出目标外设 TIMER7_CH0BRKIN
TRIGSEL_OUTPUT_TIMER7_CH1BRKIN	输出目标外设 TIMER7_CH1BRKIN
TRIGSEL_OUTPUT_TIMER7_CH2BRKIN	输出目标外设 TIMER7_CH2BRKIN
TRIGSEL_OUTPUT_TIMER15_BRKIN	输出目标外设 TIMER15_BRKIN
TRIGSEL_OUTPUT_TIMER16_BRKIN	输出目标外设 TIMER16_BRKIN

函数 trigsel_deinit

函数trigsel_deinit描述见下表：

表 3-787. 函数 trigsel_deinit

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

例如：

```
/* deinitialize TRIGSEL */
```

```
trigsel_deinit();
```

函数 trigsel_init

函数trigsel_init描述见下表：

表 3-788. 函数 trigsels_init

函数名称	trigsels_init
函数原型	void trigsels_init(trigsels_periph_enum target_periph, trigsels_source_enum trigger_source);
功能描述	为外设选择触发输入源
先决条件	-
被调用函数	-
输入参数{in}	
target_periph	目标外设，参考 表 3-786. 枚举类型 trigsels_periph_enum
输入参数{in}	
trigger_source	触发源，参考 表 3-785. 枚举类型 trigsels_source_enum
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* select TIMER0_CH2 to trigger ADC0 */
trigsels_init(TRIGSELS_OUTPUT_ADC0_ROUTRG, TRIGSELS_INPUT_TIMER0_CH2);
```

函数 trigsels_trigger_source_get

函数trigsels_trigger_source_get描述见下表：

表 3-789. 函数 trigsels_trigger_source_get

函数名称	trigsels_trigger_source_get
函数原型	trigsels_source_enum trigsels_trigger_source_get(trigsels_periph_enum target_periph);
功能描述	获取外设的触发输入源
先决条件	-
被调用函数	-
输入参数{in}	
target_periph	目标外设，参考 表 3-786. 枚举类型 trigsels_periph_enum
输出参数{out}	
-	-
返回值	
trigger_source	触发源，参考 表 3-785. 枚举类型 trigsels_source_enum

例如：

```
/* get the trigger input signal for ADC0 */
trigsels_source_enum input_signal;
input_signal = trigsels_trigger_source_get(TRIGSELS_OUTPUT_ADC0_ROUTRG);
```

函数 trigsels_register_lock_set

函数trigsels_register_lock_set描述见下表：

表 3-790. 函数 trigsels_trigger_source_set

函数名称	trigsels_register_lock_set
函数原型	void trigsels_register_lock_set(trigsels_periph_enum target_periph);
功能描述	锁定触发寄存器
先决条件	-
被调用函数	-
输入参数{in}	
target_periph	目标外设，参考 表 3-786. 枚举类型 trigsels_periph_enum
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* lock the trigger register for ADC0 */
trigsels_register_lock_set(TRIGSELS_OUTPUT_ADC0_ROUTRG);
```

函数 trigsels_register_lock_get

函数trigsels_register_lock_get描述见下表：

表 3-791. 函数 trigsels_trigger_lock_get

函数名称	trigsels_register_lock_get
函数原型	FlagStatus trigsels_register_lock_get(trigsels_periph_enum target_periph);
功能描述	获取触发寄存器锁定状态
先决条件	-
被调用函数	-
输入参数{in}	
target_periph	目标外设，参考 表 3-786. 枚举类型 trigsels_periph_enum
输出参数{out}	
-	-
返回值	
FlagStatus	SET 或 RESET

例如：

```
/* get the trigger register lock status of ADC0 */
FlagStatus status;

status = trigsels_register_lock_get(TRIGSELS_OUTPUT_ADC0_ROUTRG);
```


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 3-792. TRNG 寄存器

寄存器名称	寄存器描述
TRNG_CTL	TRNG控制寄存器
TRNG_STAT	TRNG状态寄存器
TRNG_DATA	TRNG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 3-793. TRNG 库函数

库函数名称	库函数描述
trng_deinit	复位TRNG
trng_enable	使能TRNG接口
trng_disable	禁能TRNG接口
trng_get_true_random_data	获取真随机值
trng_powermode_config	配置TRNG模拟电源模式
trng_flag_get	获取TRNG状态标志
trng_interrupt_enable	使能TRNG中断
trng_interrupt_disable	禁能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 trng_flag_enum

表 3-794. 枚举 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 trng_int_flag_enum

表 3-795. 枚举 trng_int_flag_enum

成员名称	功能描述
------	------

TRNG_INT_FLAG_CEIF	时钟错误中断标志
TRNG_INT_FLAG_SEIF	种子错误中断标志

函数 trng_deinit

函数trng_deinit描述见下表:

表 3-796. 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	rcu_periph_reset_enable / rcu_periph_reset_disable
输入参数{in}	
-	-
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* reset TRNG */
```

```
trng_deinit();
```

函数 trng_enable

函数trng_enable描述见下表:

表 3-797. 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 TRNG */
```

```
trng_enable();
```

函数 trng_disable

函数trng_disable描述见下表：

表 3-798. 函数 trng_disable

函数名称	trng_disable
函数原形	void trng_disable(void);
功能描述	禁能TRNG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 TRNG */
trng_disable();
```

函数 trng_get_true_random_data

函数trng_get_true_random_data描述见下表：

表 3-799. 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	生成的随机数据，0x0 – 0xFFFFFFFF

例如：

```
/* get true random data */
uint32_t data;
data = trng_get_true_random_data();
```

函数 `trng_powermode_config`

函数 `trng_powermode_config` 描述见下表：

表 3-800. 函数 `trng_powermode_config`

函数名称	<code>trng_powermode_config</code>
函数原形	<code>void trng_powermode_config(uint32_t powermode);</code>
功能描述	配置TRNG模拟电源模式
先决条件	-
被调用函数	-
输入参数{in}	
powermode	模拟电源模式序
<code>TRNG_NR_ULTRALOW</code>	TRNG模拟电源超低模式
<code>TRNG_NR_LOW</code>	TRNG模拟电源低模式
<code>TRNG_NR_MEDIUM</code>	TRNG模拟电源中等模式
<code>TRNG_NR_HIGH</code>	TRNG模拟电源高模式
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* configure TRNG analog power mode */
```

```
trng_powermode_config (TRNG_FLAG_CECS);
```

函数 `trng_flag_get`

函数 `trng_flag_get` 描述见下表：

表 3-801. 函数 `trng_flag_get`

函数名称	<code>trng_flag_get</code>
函数原形	<code>FlagStatus trng_flag_get(trng_flag_enum flag);</code>
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3-794. 枚举trng_flag_enum
输出参数{out}	
-	-
返回值	
FlagStatus	SET或RESET

例如：

```

/* get TRNG clock error current flag status */

FlagStatus flag_status = RESET;

flag_status == trng_flag_get(TRNG_FLAG_CECS);

```

函数 trng_interrupt_enable

函数trng_interrupt_enable描述见下表：

表 3-802. 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 TRNG interrupt */

trng_interrupt_enable();

```

函数 trng_interrupt_disable

函数trng_interrupt_disable描述见下表：

表 3-803. 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	禁能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 TRNG interrupt */

```

```
trng_interrupt_disable();
```

函数 trng_interrupt_flag_get

函数trng_interrupt_flag_get描述见下表：

表 3-804. 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3-795. 枚举trng_int_flag_enum
输出参数{out}	
-	-
返回值	
FlagStatus	SET或RESET

例如：

```
/* get TRNG clock error interrupt flag */
FlagStatus interrupt_flag = RESET;
interrupt_flag = trng_interrupt_flag_get(TRNG_INT_FLAG_CEIF);
```

函数 trng_interrupt_flag_clear

函数trng_interrupt_flag_clear描述见下表：

表 3-805. 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3-795. 枚举trng_int_flag_enum
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* clear TRNG clock error interrupt flag */
trng_interrupt_flag_clear(TRNG_INT_FLAG_CEIF);
```

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 3-806. USART 寄存器

寄存器名称	寄存器描述
USART_STAT0	状态寄存器0
USART_DATA	数据寄存器
USART_BAUD	波特率寄存器
USART_CTL0	控制寄存器0
USART_CTL1	控制寄存器1
USART_CTL2	控制寄存器2
USART_GP	保护时间和预分频器寄存器
USART_CTL3	控制寄存器3
USART_RT	接收超时寄存器
USART_STAT1	状态寄存器1
USART_CHC	兼容性控制寄存器

3.28.2. 外设库函数说明

USART库函数列表如下表所示：

表 3-807. USART 库函数

库函数名称	库函数描述
usart_deinit	复位外设USART
usart_baudrate_set	配置USART波特率
usart_parity_config	配置USART奇偶校验
usart_word_length_set	配置USART字长
usart_stop_bit_set	配置USART停止位
usart_enable	使能USART
usart_disable	失能USART
usart_transmit_config	USART发送配置
usart_receive_config	USART接收配置
usart_data_first_config	配置数据传输时低位在前或高位在前
usart_invert_config	配置USART反转功能
usart_oversample_config	配置USART过采样模式
usart_sample_bit_config	配置USART采样位方法
usart_receiver_timeout_enable	使能USART接收超时

库函数名称	库函数描述
usart_receiver_timeout_disable	失能USART接收超时
usart_receiver_timeout_threshold_config	设置USART接收超时阈值
usart_data_transmit	USART发送数据功能
usart_data_receive	USART接收数据功能
usart_address_config	在地址掩码唤醒模式下配置USART地址
usart_mute_mode_enable	使能USART静默模式
usart_mute_mode_disable	失能USART静默模式
usart_mute_mode_wakeup_config	配置USART静默模式唤醒方式
usart_lin_mode_enable	使能USART LIN模式
usart_lin_mode_disable	失能USART LIN模式
usart_lin_break_detection_length_config	配置USART LIN模式中中断帧长度
usart_send_break	配置USART发送断开帧
usart_halfduplex_enable	使能USART半双工模式
usart_halfduplex_disable	失能USART半双工模式
usart_synchronous_clock_enable	在USART同步通讯模式下使能CK引脚
usart_synchronous_clock_disable	在USART同步通讯模式下失能CK引脚
usart_synchronous_clock_config	配置USART同步通讯模式参数
usart_guard_time_config	在USART智能卡模式下配置保护时间值
usart_smartcard_mode_enable	使能USART智能卡模式
usart_smartcard_mode_disable	失能USART智能卡模式
usart_smartcard_mode_nack_enable	在USART智能卡模式下使能NACK
usart_smartcard_mode_nack_disable	在USART智能卡模式下失能NACK
usart_smartcard_autoretry_config	配置智能卡自动重试次数
usart_block_length_config	配置智能卡T=1的接收时块的长度
usart_irda_mode_enable	使能USART串行红外编解码功能模块
usart_irda_mode_disable	失能USART串行红外编解码功能模块
usart_prescaler_config	在USART IrDA低功耗模式下配置外设时钟分频系数
usart_irda_lowpower_config	配置USART IrDA低功耗模式
usart_hardware_flow_rts_config	配置USART RTS硬件控制流
usart_hardware_flow_cts_config	配置USART CTS硬件控制流
usart_break_frame_coherence_config	配置USART断开帧兼容模式
usart_parity_check_coherence_config	配置USART校验兼容模式
usart_hardware_flow_coherence_config	配置USART硬件流控制兼容模式
usart_dma_receive_config	配置USART DMA接收功能
usart_dma_transmit_config	配置USART DMA发送功能
usart_flag_get	获取USART状态寄存器标志位
usart_flag_clear	清除USART状态寄存器标志位
usart_interrupt_enable	使能USART中断
usart_interrupt_disable	失能USART中断

库函数名称	库函数描述
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 3-808. 枚举类型 `usart_flag_enum`

成员名称	功能描述
USART_FLAG_CTS	CTS电平
USART_FLAG_LBD	LIN断开检测标志
USART_FLAG_TBE	发送数据寄存器空
USART_FLAG_RBNE	读数据缓冲区非空标志
USART_FLAG_TC	发送完成标志
USART_FLAG_IDLE	空闲线检测标志
USART_FLAG_ORERR	溢出错误标志
USART_FLAG_NERR	噪声错误标志
USART_FLAG_FERR	帧错误标志
USART_FLAG_PERR	校验错误标志
USART_FLAG_BSY	忙标志
USART_FLAG_EB	块结束标志
USART_FLAG_RT	接收超时标志
USART_FLAG_EPERR	提前校验错误标志

枚举类型 `usart_interrupt_flag_enum`

表 3-809. 枚举类型 `usart_interrupt_flag_enum`

成员名称	功能描述
USART_INT_FLAG_PERR	校验错误中断标志
USART_INT_FLAG_TBE	发送数据寄存器空中断标志
USART_INT_FLAG_TC	发送完成中断标志
USART_INT_FLAG_RBNE	读数据缓冲区非空中断标志
USART_INT_FLAG_RBNE_ORE RR	溢出错误中断标志
USART_INT_FLAG_IDLE	空闲线检测中断标志
USART_INT_FLAG_LBD	LIN断开检测中断标志
USART_INT_FLAG_CTS	CTS中断中断标志
USART_INT_FLAG_ERR_ORER R	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 3-810. 枚举类型 `usart_interrupt_enum`

成员名称	功能描述
USART_INT_PERR	校验错误中断使能
USART_INT_TBE	发送寄存器空中断使能
USART_INT_TC	发送完成中断使能
USART_INT_RBNE	读数据缓冲区非空中断和过载错误中断使能
USART_INT_IDLE	IDLE线检测中断使能
USART_INT_LBD	LIN断开信号检测中断使能
USART_INT_CTS	CTS中断使能
USART_INT_ERR	错误中断使能
USART_INT_EB	块尾中断使能
USART_INT_RT	接收超时中断使能

枚举类型 `usart_invert_enum`

表 3-811. 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 3-812. 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USARTx/UARTx
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
<code>UARTx</code>	x=3,4
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* reset USART0 */
usart_deinit (USART0);
```

函数 usart_baudrate_set

函数usart_baudrate_set描述见下表：

表 3-813. 函数 usart_baudrate_set

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

例如：

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

函数 usart_parity_config

函数usart_parity_config描述见下表：

表 3-814. 函数 usart_parity_config

函数名称	usart_parity_config
函数原型	void usart_parity_config(uint32_t usart_periph, uint32_t paritycfg);
功能描述	配置USART奇偶校验
先决条件	-
被调用函数	-
输入参数{in}	
usart_periph	外设USARTx/UARTx
USARTx	x=0,1,2
UARTx	x=3,4
输入参数{in}	

paritycfg	配置USART奇偶校验
<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 3-815. 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
<i>UARTx</i>	x=3,4
输入参数{in}	
wlen	配置USART字长
<i>USART_WL_8BIT</i>	8 bits
<i>USART_WL_9BIT</i>	9 bits
输出参数{out}	
-	-
返回值	
-	-

例如:

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

函数 usart_stop_bit_set

函数usart_stop_bit_set描述见下表:

表 3-816. 函数 usart_stop_bit_set

函数名称	usart_stop_bit_set
函数原型	void usart_stop_bit_set(uint32_t usart_periph, uint32_t stblen);
功能描述	配置USART停止位
先决条件	-
被调用函数	-
输入参数{in}	
usart_periph	外设USARTx/UARTx
USARTx	x=0,1,2
UARTx	x=3,4
输入参数{in}	
stblen	配置USART停止位
USART_STB_1BIT	1 bit
USART_STB_0_5BIT	0.5 bit,该位对UARTx(x=3,4)无效
USART_STB_2BIT	2 bits
USART_STB_1_5BIT	1.5 bits,该位对UARTx(x=3,4)无效
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* configure USART0 stop bit length */
usart_stop_bit_set(USART0, USART_STB_1_5BIT);
```

函数 usart_enable

函数usart_enable描述见下表:

表 3-817. 函数 usart_enable

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

返回值	
-	-

例如:

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

函数 usart_disable

函数usart_disable描述见下表:

表 3-818. 函数 usart_disable

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

例如:

```
/* disable USART0 */
usart_disable(USART0);
```

函数 usart_transmit_config

函数usart_transmit_config描述见下表:

表 3-819. 函数 usart_transmit_config

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

输入参数{in}	
txconfig	使能/失能USART发送器
<i>USART_TRANSMIT_ENABLE</i>	使能USART发送
<i>USART_TRANSMIT_DISABLE</i>	失能USART发送
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* configure USART0 transmitter */
```

```
usart_transmit_config(USART0,USART_TRANSMIT_ENABLE);
```

函数 usart_receive_config

函数usart_receive_config描述见下表：

表 3-820. 函数 usart_receive_config

函数名称	usart_receive_config
函数原型	void usart_receive_config(uint32_t usart_periph, uint32_t rxconfig);
功能描述	USART接收器配置
先决条件	-
被调用函数	-
输入参数{in}	
usart_periph	外设USARTx/UARTx
<i>USARTx</i>	x=0,1,2
<i>UARTx</i>	x=3,4
输入参数{in}	
rxconfig	使能/失能USART接收器
<i>USART_RECEIVE_ENABLE</i>	使能USART接收
<i>USART_RECEIVE_DISABLE</i>	失能USART接收
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* configure USART0 receiver */
```

```
usart_receive_config(USART0, USART_RECEIVE_ENABLE);
```

函数 usart_data_first_config

函数usart_data_first_config描述见下表：

表 3-821. 函数 usart_data_first_config

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

例如：

```
/* configure LSB of data first */
```

```
usart_data_first_config(USART0, USART_MSBF_LSB);
```

函数 usart_invert_config

函数usart_invert_config描述见下表：

表 3-822. 函数 usart_invert_config

函数名称	usart_invert_config
函数原型	void usart_invert_config(uint32_t usart_periph, usart_invert_enum invertpara);
功能描述	配置USART反转功能
先决条件	-
被调用函数	-
输入参数{in}	
usart_periph	外设USARTx
USARTx	x=0,1,2
输入参数{in}	
invertpara	参考枚举usart_invert_enum
USART_DINV_ENA	数据位电平反转

<i>BLE</i>	
<i>USART_DINV_DISABLE</i>	数据位电平不反转
<i>USART_TXPIN_ENABLE</i>	TX引脚电平反转
<i>USART_TXPIN_DISABLE</i>	TX引脚电平不反转
<i>USART_RXPIN_ENABLE</i>	RX引脚电平反转
<i>USART_RXPIN_DISABLE</i>	RX引脚电平不反转
输出参数{out}	
-	-
返回值	
-	-

例如：

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

```
usart_invert_config(USART0, USART_DINV_ENABLE);
```

函数 usart_oversample_config

函数usart_oversample_config描述见下表：

表 3-823. 函数 usart_oversample_config

函数名称	usart_oversample_config
函数原型	void usart_oversample_config(uint32_t usart_periph, uint32_t oversamp);
功能描述	配置USART过采样模式
先决条件	-
被调用函数	-
输入参数{in}	
usart_periph	外设USARTx/UARTx
<i>USARTx</i>	x=0,1,2
<i>UARTx</i>	x=3,4
输入参数{in}	
oversamp	过采样值
<i>USART_OVSMOD_8</i>	8位
<i>USART_OVSMOD_16</i>	16位
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* configure USART0 oversample mode */
usart_oversample_config(USART0, USART_OVSMOD_8);
```

函数 usart_sample_bit_config

函数usart_sample_bit_config描述见下表：

表 3-824. 函数 usart_sample_bit_config

函数名称	usart_sample_bit_config
函数原型	void usart_sample_bit_config(uint32_t usart_periph, uint32_t obsm);
功能描述	配置USART采样位方法
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}	
obsbm	采样位
USART_OSB_1bit	1位
USART_OSB_3bit	3位
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* configure USART0 sample bit */
usart_sample_bit_config(USART0, USART_OSB_1bit);
```

函数 usart_receiver_timeout_enable

函数usart_receiver_timeout_enable描述见下表：

表 3-825. 函数 usart_receiver_timeout_enable

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

输出参数{out}	
-	-
返回值	
-	-

例如：

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

函数 usart_receiver_timeout_disable

函数usart_receiver_timeout_disable描述见下表：

表 3-826. 函数 usart_receiver_timeout_disable

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

例如：

```
/* disable receiver timeout of USART */
usart_receiver_timeout_disable(USART0);
```

函数 usart_receiver_timeout_threshold_config

函数usart_receiver_timeout_threshold_config描述见下表：

表 3-827. 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 timeout);
功能描述	设置USART接收超时阈值
先决条件	-
被调用函数	-
输入参数{in}	
usart_periph	外设USARTx

USARTx	x=0,1,2
输入参数{in}	
rtimeout	超时时间
0-0xFFFFF	超时时间值
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* set the receiver timeout threshold of USART0 */
```

```
usart_receiver_timeout_threshold_config(USART0, 115200*3);
```

函数 usart_data_transmit

函数usart_data_transmit描述见下表：

表 3-828. 函数 usart_data_transmit

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

例如：

```
/* USART0 transmit data */
```

```
usart_data_transmit(USART0, 0xAA);
```

函数 usart_data_receive

函数usart_data_receive描述见下表：

表 3-829. 函数 usart_data_receive

函数名称	usart_data_receive
函数原型	uint16_t usart_data_receive(uint32_t usart_periph);
功能描述	USART接收数据功能
先决条件	-
被调用函数	-
输入参数{in}	
usart_periph	外设USARTx/UARTx
USARTx	x=0,1,2
UARTx	x=3,4
输出参数{out}	
-	-
返回值	
uint32_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 3-830. 函数 usart_address_config

函数名称	usart_address_config
函数原型	void usart_address_config(uint32_t usart_periph, uint8_t addr);
功能描述	在地址掩码唤醒模式下配置USART地址
先决条件	-
被调用函数	-
输入参数{in}	
usart_periph	外设USARTx/UARTx
USARTx	x=0,1,2
UARTx	x=3,4
输入参数{in}	
addr	USART/UART地址
0-0xFF	USART/UART地址
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* configure address of the USART0 */
```

```
usart_address_config(USART0, 0x00);
```

函数 usart_mute_mode_enable

函数usart_mute_mode_enable描述见下表：

表 3-831. 函数 usart_mute_mode_enable

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

例如：

```
/* enable USART0 receiver in mute mode */
```

```
usart_mute_mode_enable(USART0);
```

函数 usart_mute_mode_disable

函数usart_mute_mode_disable描述见下表：

表 3-832. 函数 usart_mute_mode_disable

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

例如:

```
/* disable USART0 receiver in mute mode */
usart_mute_mode_disable(USART0);
```

函数 usart_mute_mode_wakeup_config

函数usart_mute_mode_wakeup_config描述见下表:

表 3-833. 函数 usart_mute_mode_wakeup_config

函数名称	usart_mute_mode_wakeup_config
函数原型	void usart_mute_mode_wakeup_config(uint32_t usart_periph, uint32_t wmethod);
功能描述	配置USART静默模式唤醒方式
先决条件	-
被调用函数	-
输入参数{in}	
usart_periph	外设USARTx/UARTx
USARTx	x=0,1,2
UARTx	x=3,4
输入参数{in}	
wmethod	两种方法用于进入或退出静默模式
USART_WM_IDLE	空闲线唤醒
USART_WM_ADDR	地址掩码唤醒
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* configure USART0 wakeup method in mute mode */
usart_mute_mode_wakeup_config(USART0, USART_WM_IDLE);
```

函数 usart_lin_mode_enable

函数usart_lin_mode_enable描述见下表:

表 3-834. 函数 usart_lin_mode_enable

函数名称	usart_lin_mode_enable
函数原型	void usart_lin_mode_enable(uint32_t usart_periph);
功能描述	使能USART LIN模式
先决条件	-
被调用函数	-
输入参数{in}	
usart_periph	外设USARTx/UARTx

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

例如：

```
/* USART0 LIN mode enable */
```

```
usart_lin_mode_enable(USART0);
```

函数 usart_lin_mode_disable

函数usart_lin_mode_disable描述见下表：

表 3-835. 函数 usart_lin_mode_disable

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

例如：

```
/* USART0 LIN mode disable */
```

```
usart_lin_mode_disable(USART0);
```

函数 usart_lin_break_detection_length_config

函数usart_lin_break_detection_length_config描述见下表：

表 3-836. 函数 usart_lin_break_detection_length_config

函数名称	usart_lin_break_detection_length_config
函数原型	void usart_lin_break_detection_length_config(uint32_t usart_periph, uint32_t lflen);
功能描述	配置USART LIN模式中断帧长度
先决条件	-

被调用函数	-
输入参数{in}	
usart_periph	外设USARTx/UARTx
USARTx	x=0,1,2
UARTx	x=3,4
输入参数{in}	
lblen	LIN模式中断帧长度
USART_LBLEN_10 B	断开帧长度为10 bits
USART_LBLEN_11 B	断开帧长度为11 bits
输出参数{out}	
-	-
返回值	
-	-

例如：

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

```
usart_lin_break_detection_length_config(USART0, USART_LBLEN_10B);
```

函数 usart_send_break

函数usart_send_break描述见下表：

表 3-837. 函数 usart_send_break

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

例如：

```
/* USART0 send break frame */
```

```
usart_send_break(USART0);
```

函数 usart_halfduplex_enable

函数usart_halfduplex_enable描述见下表：

表 3-838. 函数 usart_halfduplex_enable

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

例如：

```
/* enable USART0 half duplex mode*/
```

```
usart_halfduplex_enable(USART0);
```

函数 usart_halfduplex_disable

函数usart_halfduplex_disable描述见下表：

表 3-839. 函数 usart_halfduplex_disable

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

例如：

```
/* disable USART0 half duplex mode*/
```

```
usart_halfduplex_disable(USART0);
```

函数 usart_synchronous_clock_enable

函数usart_synchronous_clock_enable描述见下表：

表 3-840. 函数 usart_synchronous_clock_enable

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

例如：

```
/* enable USART0 CK pin in synchronous mode */
```

```
usart_synchronous_clock_enable(USART0);
```

函数 usart_synchronous_clock_disable

函数usart_synchronous_clock_disable描述见下表：

表 3-841. 函数 usart_synchronous_clock_disable

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

例如：

```
/* disable USART0 CK pin in synchronous mode */
```

```
usart_synchronous_clock_disable(USART0);
```

函数 usart_synchronous_clock_config

函数usart_synchronous_clock_config描述见下表：

表 3-842. 函数 usart_synchronous_clock_config

函数名称	usart_synchronous_clock_config
函数原型	void usart_synchronous_clock_config(uint32_t usart_periph, uint32_t clen, uint32_t cph, uint32_t cpl);
功能描述	配置USART同步通讯模式参数
先决条件	-
被调用函数	-
输入参数{in}	
usart_periph	外设USARTx
USARTx	x=0,1,2
输入参数{in}	
clen	CK信号长度
USART_CLEN_NO NE	8位数据帧中有7个CK脉冲，9位数据帧中有8个CK脉冲
USART_CLEN_EN	8位数据帧中有8个CK脉冲，9位数据帧中有9个CK脉冲
输入参数{in}	
cph	时钟相位
USART_CPH_1CK	在首个时钟边沿采样第一个数据
USART_CPH_2CK	在第二个时钟边沿采样第一个数据
输入参数{in}	
cpl	时钟极性
USART_CPL_LOW	CK引脚不对外发送时保持为低电平
USART_CPL_HIGH	CK引脚不对外发送时保持为高电平
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* configure USART0 synchronous mode parameters */
```

```
usart_synchronous_clock_config(USART0,USART_CLEN_EN,USART_CPH_2CK,  
USART_CPL_HIGH);
```

函数 usart_guard_time_config

函数usart_guard_time_config描述见下表：

表 3-843. 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 gaut);
功能描述	在USART智能卡模式下配置保护时间值
先决条件	-
被调用函数	-
输入参数{in}	
usart_periph	外设USARTx
USARTx	x=0,1,2
输入参数{in}	
gaut	保护时间值
0-0x000000FF	保护时间值
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* configure USART0 guard time value in smartcard mode */
```

```
usart_guard_time_config(USART0, 0x0000 0055);
```

函数 usart_smartcard_mode_enable

函数usart_smartcard_mode_enable描述见下表：

表 3-844. 函数 usart_smartcard_mode_enable

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

例如：

```
/* USART0 smartcard mode enable */
```

```
usart_smartcard_mode_enable(USART0);
```

函数 usart_smartcard_mode_disable

函数usart_smartcard_mode_disable描述见下表：

表 3-845. 函数 usart_smartcard_mode_disable

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

例如:

```
/* USART0 smartcard mode disable */
usart_smartcard_mode_disable(USART0);
```

函数 usart_smartcard_mode_nack_enable

函数usart_smartcard_mode_nack_enable描述见下表:

表 3-846. 函数 usart_smartcard_mode_nack_enable

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

例如:

```
/* enable USART0 NACK in smartcard mode */
usart_smartcard_mode_nack_enable(USART0);
```

函数 usart_smartcard_mode_nack_disable

函数usart_smartcard_mode_nack_disable描述见下表:

表 3-847. 函数 usart_smartcard_mode_nack_disable

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

例如:

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

函数 usart_smartcard_autoretry_config

函数usart_smartcard_autoretry_config描述见下表:

表 3-848. 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
USARTx	x=0,1,2
输入参数{in}	
scrtnum	智能卡自动重试次数
0-0xFFFFFFFF	自动重试次数
输出参数{out}	
-	-
返回值	
-	-

例如:

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

函数 usart_block_length_config

函数usart_block_length_config描述见下表:

表 3-849. 函数 usart_block_length_config

函数名称	usart_block_length_config
函数原型	void usart_block_length_config(uint32_t usart_periph, uint32_t bl);
功能描述	配置智能卡T=1的接收时块的长度
先决条件	-
被调用函数	-
输入参数{in}	
usart_periph	外设USARTx
USARTx	x=0,1,2
输入参数{in}	
bl	块长度
0-0xFFFFFFFF	块长度
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* configure block length in Smartcard T=1 reception */
```

```
usart_block_length_config(USART0, 0xFFFFFFFF);
```

函数 usart_irda_mode_enable

函数usart_irda_mode_enable描述见下表:

表 3-850. 函数 usart_irda_mode_enable

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

例如:


```
/* enable USART0 IrDA mode */
```

```
usart_irda_mode_enable(USART0);
```

函数 usart_irda_mode_disable

函数usart_irda_mode_disable描述见下表：

表 3-851. 函数 usart_irda_mode_disable

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

例如：

```
/* disable USART0 IrDA mode */
```

```
usart_irda_mode_disable(USART0);
```

函数 usart_prescaler_config

函数usart_prescaler_config描述见下表：

表 3-852. 函数 usart_prescaler_config

函数名称	usart_prescaler_config
函数原型	void usart_prescaler_config(uint32_t usart_periph, uint8_t psc);
功能描述	在USART IrDA低功耗模式下配置外设时钟分频系数
先决条件	-
被调用函数	-
输入参数{in}	
usart_periph	外设USARTx/UARTx
USARTx	x=0,1,2
UARTx	x=3,4
输入参数{in}	
psc	时钟分频系数
0-0xFF	时钟分频系数
输出参数{out}	

-	-
返回值	
-	-

例如：

```
/* configure the USART0 peripheral clock prescaler in USART IrDA low-power mode */
```

```
usart_prescaler_config(USART0, 0x00);
```

函数 usart_irda_lowpower_config

函数usart_irda_lowpower_config描述见下表：

表 3-853. 函数 usart_irda_lowpower_config

函数名称	usart_irda_lowpower_config
函数原型	void usart_irda_lowpower_config(uint32_t usart_periph, uint32_t irlp);
功能描述	配置USART IrDA低功耗模式
先决条件	-
被调用函数	-
输入参数{in}	
usart_periph	外设USARTx/UARTx
USARTx	x=0,1,2
UARTx	x=3,4
输入参数{in}	
irlp	IrDA低功耗模式或正常模式
USART_IRLP_LOW	低功耗模式
USART_IRLP_NORMAL	正常模式
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* configure USART0 IrDA low-power */
```

```
usart_irda_lowpower_config(USART0, USART_IRLP_LOW);
```

函数 usart_hardware_flow_rts_config

函数usart_hardware_flow_rts_config描述见下表：

表 3-854. 函数 usart_hardware_flow_rts_config

函数名称	usart_hardware_flow_rts_config
函数原型	void usart_hardware_flow_rts_config(uint32_t usart_periph, uint32_t rtsconfig);
功能描述	配置USART RTS硬件控制流

先决条件	-
被调用函数	-
输入参数{in}	
usart_periph	外设USARTx
USARTx	x=0,1,2
输入参数{in}	
rtsconfig	使能/失能RTS
USART_RTS_ENA BLE	使能RTS
USART_RTS_DISA BLE	失能RTS
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* configure USART0 hardware flow control RTS */
```

```
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 3-855. 函数 usart_hardware_flow_cts_config

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

例如：

```
/* configure USART0 hardware flow control CTS */
```

```
usart_hardware_flow_cts_config(USART0, USART_CTS_ENABLE);
```

函数 usart_break_frame_coherence_config

函数usart_break_frame_coherence_config描述见下表：

表 3-856. 函数 usart_break_frame_coherence_config

函数名称	usart_break_frame_coherence_config
函数原型	void usart_break_frame_coherence_config(uint32_t usart_periph, uint32_t bcm);
功能描述	配置USART断开帧兼容模式
先决条件	-
被调用函数	-
输入参数{in}	
usart_periph	外设USARTx
USARTx	x=0,1,2
输入参数{in}	
bcm	
USART_BCM_NONE	没有检测到校验错误
USART_BCM_EN	检测到校验错误
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* configure USART0 break frame coherence mode */
```

```
usart_break_frame_coherence_config(USART0, USART_BCM_NONE);
```

函数 usart_parity_check_coherence_config

函数usart_parity_check_coherence_config描述见下表：

表 3-857. 函数 usart_parity_check_coherence_config

函数名称	usart_parity_check_coherence_config
函数原型	void usart_parity_check_coherence_config(uint32_t usart_periph, uint32_t pcm);
功能描述	配置USART校验兼容模式
先决条件	-
被调用函数	-
输入参数{in}	

usart_periph	外设USARTx/UARTx
USARTx	x=0,1,2
UARTx	x=3,4
输入参数{in}	
pcm	
USART_PCM_NONE	不检查奇偶校验
USART_PCM_EN	检查奇偶校验
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* configure USART0 parity check coherence mode */
```

```
usart_parity_check_coherence_config(USART0, USART_PCM_NONE);
```

函数 usart_hardware_flow_coherence_config

函数usart_hardware_flow_coherence_config描述见下表:

表 3-858. 函数 usart_hardware_flow_coherence_config

函数名称	usart_hardware_flow_coherence_config
函数原型	void usart_hardware_flow_coherence_config(uint32_t usart_periph, uint32_t hcm);
功能描述	配置USART硬件流控制兼容模式
先决条件	-
被调用函数	-
输入参数{in}	
usart_periph	外设USARTx
USARTx	x=0,1,2
输入参数{in}	
hcm	
USART_HCM_NONE	nRTS信号与USART_STAT0寄存器中RBNE位相同
USART_HCM_EN	nRTS信号在最后一个数据位被采样后被置位
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* configure USART0 hardware flow control coherence mode */
```

```
usart_hardware_flow_coherence_config(USART0, USART_HCM_NONE);
```

函数 usart_dma_receive_config

函数usart_dma_receive_config描述见下表:

表 3-859. 函数 usart_dma_receive_config

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

例如:

```
/* USART0 DMA enable for reception */
```

```
usart_dma_receive_config(USART0, USART_DENR_ENABLE);
```

函数 usart_dma_transmit_config

函数usart_dma_transmit_config描述见下表:

表 3-860. 函数 usart_dma_transmit_config

函数名称	usart_dma_transmit_config
函数原型	void usart_dma_transmit_config(uint32_t usart_periph, uint32_t dmacmd);
功能描述	配置USART DMA发送功能
先决条件	-
被调用函数	-
输入参数{in}	
usart_periph	外设USARTx/UARTx
USARTx	x=0,1,2
UARTx	x=3,4

输入参数{in}	
dmacmd	使能/失能DMA发送功能
<i>USART_DENT_ENABLE</i>	使能DMA发送功能
<i>USART_DENT_DISABLE</i>	失能DMA发送功能
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* USART0 DMA enable for transmission */
```

```
usart_dma_transmit_config(USART0, USART_DENT_ENABLE);
```

函数 usart_flag_get

函数usart_flag_get描述见下表:

表 3-861. 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
<i>USARTx</i>	x=0,1,2
<i>UARTx</i>	x=3,4
输入参数{in}	
flag	USART标志位, 参考 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 3-862. 函数 usart_flag_clear

函数名称	usart_flag_clear
函数原型	void usart_flag_clear(uint32_t usart_periph, usart_flag_enum flag);
功能描述	清除USART状态寄存器标志位
先决条件	-
被调用函数	-
输入参数{in}	
usart_periph	外设USARTx/UARTx
USARTx	x=0,1,2
UARTx	x=3,4
输入参数{in}	
flag	USART标志位, 参考 枚举类型usart_flag_enum
USART_FLAG_CTS	CTS变化标志
USART_FLAG_LBD	LIN断开检测标志
USART_FLAG_TC	发送完成
USART_FLAG_RBNE	读数据缓冲区非空
USART_FLAG_EB	块结束标志
USART_FLAG_RT	接收超时标志
USART_FLAG_EPE RR	校验错误超前检测标志
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* clear USART0 flag */
```

```
usart_flag_clear(USART0,USART_FLAG_TC);
```

函数 usart_interrupt_enable

函数usart_interrupt_enable描述见下表:

表 3-863. 函数 usart_interrupt_enable

函数名称	usart_interrupt_enable
函数原型	void usart_interrupt_enable(uint32_t usart_periph, usart_interrupt_enum interrupt);
功能描述	使能USART中断
先决条件	-

被调用函数	-
输入参数{in}	
usart_periph	外设USARTx/UARTx
USARTx	x=0,1,2
UARTx	x=3,4
输入参数{in}	
interrupt	USART中断, 参考 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 3-864. 函数 usart_interrupt_disable

函数名称	usart_interrupt_disable
函数原型	void usart_interrupt_disable(uint32_t usart_periph, usart_interrupt_enum interrupt);
功能描述	失能USART中断
先决条件	-
被调用函数	-
输入参数{in}	
usart_periph	外设USARTx/UARTx
USARTx	x=0,1,2
UARTx	x=3,4
输入参数{in}	
interrupt	USART中断, 参考 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 3-865. 函数 usart_interrupt_flag_get

函数名称	usart_interrupt_flag_get
函数原型	FlagStatus usart_interrupt_flag_get(uint32_t usart_periph, usart_interrupt_flag_enum int_flag);
功能描述	获取USART中断标志位状态
先决条件	-
被调用函数	-
输入参数{in}	
usart_periph	外设USARTx/UARTx
USARTx	x=0,1,2
UARTx	x=3,4
输入参数{in}	
int_flag	USART中断标志，参考 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 3-866. 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 枚举类型usart_interrupt_flag_enum

USART_INT_FLAG_CTS	CTS变化中断标志
USART_INT_FLAG_LBD	LIN断开检测中断标志
USART_INT_FLAG_TC	发送完成中断标志
USART_INT_FLAG_RBNE	读数据缓冲区非空中断标志
USART_INT_FLAG_EB	块结束事件中断标志
USART_INT_FLAG_RT	超时事件中断标志
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* clear the USART0 interrupt flag */
usart_interrupt_flag_clear(USART0, USART_INT_FLAG_RBNE);
```

3.29. 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 3-867. 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 3-868. 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

库函数名称	库函数说明
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

函数 wwdgt_deinit

函数wwdgt_deinit描述见下表:

表 3-869. 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 3-870. 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 3-871. 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 3-872. 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	WWDGT计数器时钟为 (PCLK / 4096) / 1

_DIV1	
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
WWDGT_CFG_PSC _DIV16	WWDGT计数器时钟为 (PCLK / 4096) / 16
WWDGT_CFG_PSC _DIV32	WWDGT计数器时钟为 (PCLK / 4096) / 32
WWDGT_CFG_PSC _DIV64	WWDGT计数器时钟为 (PCLK / 4096) / 64
WWDGT_CFG_PSC _DIV128	WWDGT计数器时钟为 (PCLK / 4096) / 128
WWDGT_CFG_PSC _DIV256	WWDGT计数器时钟为 (PCLK / 4096) / 256
WWDGT_CFG_PSC _DIV512	WWDGT计数器时钟为 (PCLK / 4096) / 512
WWDGT_CFG_PSC _DIV1024	WWDGT计数器时钟为 (PCLK / 4096) / 1024
WWDGT_CFG_PSC _DIV2048	WWDGT计数器时钟为 (PCLK / 4096) / 2048
WWDGT_CFG_PSC _DIV4096	WWDGT计数器时钟为 (PCLK / 4096) / 4096
WWDGT_CFG_PSC _DIV8192	WWDGT计数器时钟为 (PCLK / 4096) / 8192
WWDGT_CFG_PSC _DIV16384	WWDGT计数器时钟为 (PCLK / 4096) / 16384
WWDGT_CFG_PSC _DIV32768	WWDGT计数器时钟为 (PCLK / 4096) / 32768
WWDGT_CFG_PSC _DIV65536	WWDGT计数器时钟为 (PCLK / 4096) / 65536
WWDGT_CFG_PSC _DIV131072	WWDGT计数器时钟为 (PCLK / 4096) / 131072
WWDGT_CFG_PSC _DIV262144	WWDGT计数器时钟为 (PCLK / 4096) / 262144
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 3-873. 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 / RESET

例如：

```
/* test if the counter value update has reached the 0x40 */
```

```
FlagStatus status;
```

```
status = wwdgt_flag_get ( );
```

```
if(status == RESET)
```

```
{
```

```
...
```

```
}else
```

```
{
```

```
...
```

```
}
```

函数 wwdgt_flag_clear

函数 wwdgt_flag_clear 描述见下表：

表 3-874. 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 3-875. 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 ( );
```


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	2025 年 11 月 3 日
1.1	1. 入参修改: dac_flag_get dac_flag_clear dac_interrupt_enable dac_interrupt_disable dac_interrupt_flag_get dac_interrupt_flag_clear 2. 添加 can_transmission_stop 函数返回值描述	2026 年 2 月 25 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.