

GigaDevice Semiconductor Inc.

GD32H75E

Arm[®] Cortex[®]-M7 32-bit MCU

**固件库
使用指南**

1.1 版本

(2025 年 1 月)

目录

目录.....	2
图索引	7
表索引	8
1. 介绍	43
1.1. 文档和固件库规则	43
1.1.1. 外设缩写	43
1.1.2. 命名规则	44
2. 固件库概述	45
2.1. 文件组织结构.....	45
2.1.1. Examples 文件夹	47
2.1.2. Firmware 文件夹	47
2.1.3. Template 文件夹.....	47
2.1.4. Utilities 文件夹	49
2.2. 固件库文件描述	49
3. 外设固件库	51
3.1. 外设固件库概述	51
3.2. ADC	51
3.2.1. 外设寄存器描述	51
3.2.2. 外设库函数说明	52
3.3. CAN	91
3.3.1. 外设寄存器说明	91
3.3.2. 外设库函数说明	92
3.4. CMP.....	138
3.4.1. 外设寄存器说明	138
3.4.2. 外设库函数说明	138
3.5. CPDM	153
3.5.1. 外设寄存器描述	153
3.5.2. 外设库函数说明	153
3.6. CRC.....	159
3.6.1. 外设寄存器说明	159
3.6.2. 外设库函数说明	159
3.7. CTC.....	167
3.7.1. 外设寄存器说明	167

3.7.2.	外设库函数说明	167
3.8.	DAC	181
3.8.1.	外设寄存器说明	181
3.8.2.	外设库函数说明	181
3.9.	DBG	202
3.9.1.	外设寄存器说明	202
3.9.2.	外设库函数说明	202
3.10.	DMA / DMAMUX	208
3.10.1.	外设寄存器说明	208
3.10.2.	外设库函数说明	209
3.11.	EDOUT	265
3.11.1.	外设寄存器描述	265
3.11.2.	外设库函数说明	265
3.12.	EFUSE	272
3.12.1.	外设寄存器描述	272
3.12.2.	外设库函数说明	272
3.13.	EXMC	285
3.13.1.	外设寄存器描述	285
3.13.2.	外设库函数说明	286
3.14.	EXTI	312
3.14.1.	外设寄存器说明	312
3.14.2.	外设库函数说明	312
3.15.	FAC	320
3.15.1.	外设寄存器说明	320
3.15.2.	外设库函数说明	321
3.16.	FMC	339
3.16.1.	外设寄存器说明	339
3.16.2.	外设库函数说明	340
3.17.	FWDGT	377
3.17.1.	外设寄存器说明	377
3.17.2.	外设库函数说明	377
3.18.	GPIO	382
3.18.1.	外设寄存器说明	383
3.18.2.	外设库函数说明	383
3.19.	HPDF	395
3.19.1.	外设寄存器说明	395
3.19.2.	外设库函数说明	395
3.20.	I2C	453

3.20.1.	外设寄存器说明	453
3.20.2.	外设库函数说明	453
3.21.	LPDTS.....	492
3.21.1.	外设寄存器说明	492
3.21.2.	外设库函数说明	493
3.22.	MDMA	502
3.22.1.	外设寄存器描述	502
3.22.2.	外设库函数说明	503
3.23.	OSPI	533
3.23.1.	外设寄存器描述	533
3.23.2.	外设库函数说明	533
3.24.	OSPIM.....	574
3.24.1.	外设寄存器描述	574
3.24.2.	外设库函数说明	574
3.25.	MISC	579
3.25.1.	外设寄存器说明	579
3.25.2.	外设库函数说明	580
3.26.	PMU.....	591
3.26.1.	外设寄存器说明	591
3.26.2.	外设库函数说明	591
3.27.	RAMECCUM	611
3.27.1.	外设寄存器描述	611
3.27.2.	外设库函数说明	611
3.28.	RCU.....	626
3.28.1.	外设寄存器说明	626
3.28.2.	外设库函数说明	627
3.29.	RTC	683
3.29.1.	外设寄存器描述	683
3.29.2.	外设库函数描述	684
3.30.	SPI.....	709
3.30.1.	外设寄存器说明	709
3.30.2.	外设库函数说明	710
3.31.	SYSCFG.....	755
3.31.1.	外设寄存器说明	755
3.31.2.	外设库函数说明	756
3.32.	TIMER	778
3.32.1.	外设寄存器说明	778
3.32.2.	外设库函数说明	779

3.33.	TMU.....	871
3.33.1.	外设寄存器说明	871
3.33.2.	外设库函数说明	871
3.34.	TRIGSEL.....	880
3.34.1.	外设寄存器说明	880
3.34.2.	外设库函数说明	881
3.35.	TRNG.....	889
3.35.1.	外设寄存器说明	889
3.35.2.	外设库函数说明	890
3.36.	USART.....	911
3.36.1.	外设寄存器说明	911
3.36.2.	外设库函数说明	911
3.37.	VREF	964
3.37.1.	外设寄存器说明	965
3.37.2.	外设库函数说明	965
3.38.	WWDGT	970
3.38.1.	外设寄存器说明	970
3.38.2.	外设库函数说明	970
3.39.	ESC_INTC.....	975
3.39.1.	外设寄存器描述	975
3.39.2.	外设库函数说明	975
3.40.	ESC_OSPI.....	982
3.40.1.	外设寄存器描述	982
3.40.2.	外设库函数说明	983
3.41.	ESC_PHY.....	992
3.41.1.	外设寄存器描述	992
3.41.2.	外设库函数说明	992
3.42.	ESC_PMU	995
3.42.1.	外设寄存器描述	995
3.42.2.	外设库函数说明	995
3.43.	ESC_SYSCFG.....	1002
3.43.1.	外设寄存器说明	1003
3.43.2.	外设库函数说明	1003
3.44.	ESC_TIMER.....	1005
3.44.1.	外设寄存器描述	1005
3.44.2.	外设库函数说明	1006
4.	版本历史.....	1009

图索引

图 2-1. GD32H75E 固件库文件组织结构	46
图 2-2. 选择外设例程文件	48
图 2-3. 拷贝外设例程文件	48
图 2-4. 打开工程文件.....	48
图 2-5. 配置工程文件.....	49
图 2-6. 编译调试下载.....	49

表索引

表 1-1. 外设缩写.....	43
表 2-1. 固件函数库文件描述	49
表 3-1. 外设固件库函数描述格式.....	51
表 3-2. ADC 寄存器	51
表 3-3. ADC 库函数	52
表 3-4. 函数 adc_deinit	54
表 3-5. 函数 adc_clock_config	54
表 3-6. 函数 adc_special_function_config	56
表 3-7. 函数 adc_data_alignment_config	56
表 3-8. 函数 adc_enable.....	57
表 3-9. 函数 adc_disable	58
表 3-10. 函数 adc_calibration_mode_config.....	58
表 3-11. 函数 adc_calibration_number	59
表 3-12. 函数 adc_calibration_enable	60
表 3-13. 函数 adc_resolution_config.....	60
表 3-14. 函数 adc_internal_channel_config.....	61
表 3-15. 函数 adc_dma_mode_enable	62
表 3-16. 函数 adc_dma_mode_disable	62
表 3-17. 函数 adc_dma_request_after_last_enable.....	63
表 3-18. 函数 adc_dma_request_after_last_disable.....	63
表 3-19. 函数 adc_hpdf_mode_enable	64
表 3-20. 函数 adc_hpdf_mode_disable	64
表 3-21. 函数 adc_discontinuous_mode_config.....	65
表 3-22. 函数 adc_channel_length_config	66
表 3-23. 函数 adc_regular_channel_config.....	66
表 3-24. 函数 adc_inserted_channel_config.....	67
表 3-25. 函数 adc_inserted_channel_offset_config	68
表 3-26. 函数 adc_channel_differential_mode_config.....	69
表 3-27. 函数 adc_external_trigger_config	69
表 3-28. 函数 adc_software_trigger_enable.....	70
表 3-29. 函数 adc_end_of_conversion_config	71
表 3-30. 函数 adc_regular_data_read.....	72
表 3-31. 函数 adc_inserted_data_read.....	72
表 3-32. 函数 adc_watchdog0_single_channel_enable	73
表 3-33. 函数 adc_watchdog0_group_channel_enable	74
表 3-34. 函数 adc_watchdog0_disable	74
表 3-35. 函数 adc_watchdog1_channel_config.....	75
表 3-36. 函数 adc_watchdog2_channel_config.....	76
表 3-37. 函数 adc_watchdog1_disable	77
表 3-38. 函数 adc_watchdog2_disable	77

表 3-39. 函数 adc_watchdog0_threshold_config	78
表 3-40. 函数 adc_watchdog1_threshold_config	78
表 3-41. 函数 adc_watchdog2_threshold_config	79
表 3-42. 函数 adc_oversample_mode_config	79
表 3-43. 函数 adc_oversample_mode_enable	81
表 3-44. 函数 adc_oversample_mode_disable	81
表 3-45. 函数 adc_flag_get	82
表 3-46. 函数 adc_flag_clear	83
表 3-47. 函数 adc_interrupt_enable	83
表 3-48. 函数 adc_interrupt_disable	84
表 3-49. 函数 adc_interrupt_flag_get	85
表 3-50. 函数 adc_interrupt_flag_clear	86
表 3-51. 函数 adc_sync_mode_config	86
表 3-52. 函数 adc_sync_delay_config	87
表 3-53. 函数 adc_sync_dma_config	88
表 3-54. 函数 adc_sync_dma_request_after_last_enable	89
表 3-55. 函数 adc_sync_dma_request_after_last_disable	89
表 3-56. 函数 adc_sync_master_adc_regular_data0_read	90
表 3-57. 函数 adc_sync_slave_adc_regular_data0_read	90
表 3-58. 函数 adc_sync_regular_data1_read	91
表 3-59. CAN 寄存器	91
表 3-60. CAN 库函数	92
表 3-61. 结构体 can_error_counter_struct	94
表 3-62. 结构体 can_parameter_struct	94
表 3-63. 结构体 can_mailbox_descriptor_struct	95
表 3-64. 结构体 can_rx_fifo_struct	95
表 3-65. 结构体 can_fd_parameter_struct	95
表 3-66. 结构体 can_rx_fifo_id_filter_struct	96
表 3-67. 结构体 can_fifo_parameter_struct	96
表 3-68. 结构体 can_pn_mode_filter_struct	96
表 3-69. 结构体 can_pn_mode_config_struct	97
表 3-70. 结构体 can_crc_struct	97
表 3-71. 枚举类型 can_interrupt_enum	97
表 3-72. 枚举类型 can_flag_enum	99
表 3-73. 枚举类型 can_interrupt_flag_enum	101
表 3-74. 枚举类型 can_operation_modes_enum	103
表 3-75. 枚举类型 can_struct_type_enum	103
表 3-76. 枚举类型 can_error_state_enum	104
表 3-77. 函数 can_deinit	104
表 3-78. 函数 can_software_reset	105
表 3-79. 函数 can_init	105
表 3-80. 函数 can_struct_para_init	106
表 3-81. 函数 can_private_filter_config	106

表 3-82. 函数 can_operation_mode_enter.....	107
表 3-83. 函数 can_operation_mode_get.....	108
表 3-84. 函数 can_inactive_mode_exit.....	108
表 3-85. 函数 can_pn_mode_exit.....	109
表 3-86. 函数 can_fd_config.....	109
表 3-87. 函数 can_bitrate_switch_enable.....	110
表 3-88. 函数 can_bitrate_switch_disable.....	111
表 3-89. 函数 can_tdc_get.....	111
表 3-90. 函数 can_tdc_enable.....	112
表 3-91. 函数 can_tdc_disable.....	112
表 3-92. 函数 can_rx_fifo_config.....	113
表 3-93. 函数 can_rx_fifo_filter_table_config.....	113
表 3-94. 函数 can_rx_fifo_read.....	114
表 3-95. 函数 can_rx_fifo_filter_matching_number_get.....	114
表 3-96. 函数 can_rx_fifo_clear.....	115
表 3-97. 函数 can_ram_address_get.....	115
表 3-98. 函数 can_mailbox_config.....	116
表 3-99. 函数 can_mailbox_transmit_abort.....	117
表 3-100. 函数 can_mailbox_transmit_inactive.....	117
表 3-101. 函数 can_mailbox_receive_data_read.....	118
表 3-102. 函数 can_mailbox_receive_lock.....	119
表 3-103. 函数 can_mailbox_receive_unlock.....	119
表 3-104. 函数 can_mailbox_receive_inactive.....	120
表 3-105. 函数 can_mailbox_code_get.....	120
表 3-106. 函数 can_error_counter_config.....	121
表 3-107. 函数 can_error_counter_get.....	122
表 3-108. 函数 can_error_state_get.....	122
表 3-109. 函数 can_crc_get.....	123
表 3-110. 函数 can_pn_mode_config.....	123
表 3-111. 函数 can_pn_mode_filter_config.....	124
表 3-112. 函数 can_pn_mode_num_of_match_get.....	125
表 3-113. 函数 can_pn_mode_data_read.....	125
表 3-114. 函数 can_self_reception_enable.....	126
表 3-115. 函数 can_self_reception_disable.....	127
表 3-116. 函数 can_transmit_abort_enable.....	127
表 3-117. 函数 can_transmit_abort_disable.....	128
表 3-118. 函数 can_auto_busoff_recovery_enable.....	128
表 3-119. 函数 can_auto_busoff_recovery_disable.....	129
表 3-120. 函数 can_time_sync_enable.....	129
表 3-121. 函数 can_time_sync_disable.....	130
表 3-122. 函数 can_edge_filter_mode_enable.....	130
表 3-123. 函数 can_edge_filter_mode_disable.....	131
表 3-124. 函数 can_ped_mode_enable.....	131

表 3-125. 函数 can_ped_mode_disable	132
表 3-126. 函数 can_arbitration_delay_bits_config	132
表 3-127. 函数 can_bsp_mode_config	133
表 3-128. 函数 can_flag_get	133
表 3-129. 函数 can_flag_clear	134
表 3-130. 函数 can_interrupt_enable	134
表 3-131. 函数 can_interrupt_disable	135
表 3-132. 函数 can_interrupt_flag_get	136
表 3-133. 函数 can_interrupt_flag_clear	136
表 3-134. CMP 寄存器	138
表 3-135. CMP 库函数	138
表 3-136. 枚举类型 cmp_enum	139
表 3-137. 函数 cmp_deinit	139
表 3-138. 函数 cmp_mode_init	139
表 3-139. 函数 cmp_noninverting_input_select	141
表 3-140. 函数 cmp_output_init	142
表 3-141. 函数 cmp_output_mux_config	142
表 3-142. 函数 cmp_blanking_init	143
表 3-143. 函数 cmp_enable	144
表 3-144. 函数 cmp_disable	145
表 3-145. 函数 cmp_window_enable	145
表 3-146. 函数 cmp_window_disable	146
表 3-147. 函数 cmp_lock_enable	146
表 3-148. 函数 cmp_voltage_scaler_enable	147
表 3-149. 函数 cmp_voltage_scaler_disable	147
表 3-150. 函数 cmp_scaler_bridge_enable	148
表 3-151. 函数 cmp_scaler_bridge_disable	148
表 3-152. 函数 cmp_output_level_get	149
表 3-153. 函数 cmp_flag_get	149
表 3-154. 函数 cmp_flag_clear	150
表 3-155. 函数 cmp_interrupt_enable	150
表 3-156. 函数 cmp_interrupt_disable	151
表 3-157. 函数 cmp_interrupt_flag_get	152
表 3-158. 函数 cmp_interrupt_flag_clear	152
表 3-159. CPDM 寄存器	153
表 3-160. CPDM 库函数	153
表 3-161. 枚举类型 cpdm_output_phase_enum	153
表 3-162. 函数 cpdm_enable	154
表 3-163. 函数 cpdm_disable	155
表 3-164. 函数 cpdm_delayline_sample_enable	155
表 3-165. 函数 cpdm_delayline_sample_disable	156
表 3-166. 函数 cpdm_output_clock_phase_select	156
表 3-167. 函数 cpdm_delayline_length_valid_flag_get	157

表 3-168. 函数 cpdm_delayline_length_get	157
表 3-169. 函数 cpdm_clock_output	158
表 3-170. CRC 寄存器.....	159
表 3-171. CRC 库函数.....	159
表 3-172. 函数 crc_deinit	159
表 3-173. 函数 crc_init_data_register_write.....	160
表 3-174. 函数 crc_data_register_read	160
表 3-175. 函数 crc_free_data_register_read	161
表 3-176. 函数 crc_free_data_register_write	161
表 3-177. 函数 crc_reverse_output_data_disable.....	162
表 3-178. 函数 crc_reverse_output_data_enable	162
表 3-179. 函数 crc_input_data_reverse_config	163
表 3-180. 函数 crc_data_register_reset	164
表 3-181. 函数 crc_polynomial_size_set	164
表 3-182. 函数 crc_polynomial_set.....	165
表 3-183. 函数 crc_single_data_calculate	165
表 3-184. 函数 crc_block_data_calculate	166
表 3-185. CTC 寄存器	167
表 3-186. CTC 库函数	167
表 3-187. 函数 ctc_deinit	168
表 3-188. 函数 ctc_counter_enable	168
表 3-189. 函数 ctc_counter_disable	169
表 3-190. 函数 ctc_irc48m_trim_value_config.....	169
表 3-191. 函数 ctc_software_refsource_pulse_generate	170
表 3-192. 函数 ctc_hardware_trim_mode_config.....	170
表 3-193. 函数 ctc_refsource_polarity_config	171
表 3-194. 函数 ctc_refsource_signal_select	172
表 3-195. 函数 ctc_refsource_prescaler_config.....	172
表 3-196. 函数 ctc_clock_limit_value_config.....	173
表 3-197. 函数 ctc_counter_reload_value_config.....	173
表 3-198. 函数 ctc_counter_capture_value_read	174
表 3-199. 函数 ctc_counter_direction_read	174
表 3-200. 函数 ctc_counter_reload_value_read	175
表 3-201. 函数 ctc_irc48m_trim_value_read	176
表 3-202. 函数 ctc_flag_get.....	176
表 3-203. 函数 ctc_flag_clear.....	177
表 3-204. 函数 ctc_interrupt_enable.....	178
表 3-205. 函数 ctc_interrupt_disable.....	178
表 3-206. 函数 ctc_interrupt_flag_get	179
表 3-207. 函数 ctc_interrupt_flag_clear	180
表 3-208. DAC 寄存器.....	181
表 3-209. DAC 库函数.....	181
表 3-210. 函数 dac_deinit	182

表 3-211. 函数 dac_enable.....	183
表 3-212. 函数 dac_disable	183
表 3-213. 函数 dac_dma_enable	184
表 3-214. 函数 dac_dma_disable	185
表 3-215. 函数 dac_mode_config.....	185
表 3-216. 函数 dac_trimming_value_get	186
表 3-217. 函数 dac_trimming_value_set.....	187
表 3-218. 函数 dac_trimming_enable	187
表 3-219. 函数 dac_output_value_get.....	188
表 3-220. 函数 dac_data_set.....	189
表 3-221. 函数 dac_trigger_enable	189
表 3-222. 函数 dac_trigger_disable	190
表 3-223. 函数 dac_trigger_source_config	191
表 3-224. 函数 dac_software_trigger_enable	191
表 3-225. 函数 dac_wave_mode_config	192
表 3-226. 函数 dac_lfsr_noise_config	193
表 3-227. 函数 dac_triangle_noise_config	193
表 3-228. 函数 dac_concurrent_enable	194
表 3-229. 函数 dac_concurrent_disable	195
表 3-230. 函数 dac_concurrent_software_trigger_enable.....	195
表 3-231. 函数 dac_concurrent_data_set.....	196
表 3-232. 函数 dac_sample_keep_mode_config.....	196
表 3-233. 函数 dac_flag_get.....	197
表 3-234. 函数 dac_flag_clear.....	198
表 3-235. 函数 dac_interrupt_enable.....	199
表 3-236. 函数 dac_interrupt_disable.....	199
表 3-237. 函数 dac_interrupt_flag_get	200
表 3-238. 函数 dac_interrupt_flag_clear	200
表 3-239. DBG 寄存器.....	202
表 3-240. DBG 库函数.....	202
表 3-241. 枚举类型 dbg_periph_enum.....	202
表 3-242. 函数 dbg_deinit.....	203
表 3-243. 函数 dbg_id_get.....	203
表 3-244. 函数 dbg_low_power_enable	204
表 3-245. 函数 dbg_low_power_disable	204
表 3-246. 函数 dbg_periph_enable	205
表 3-247. 函数 dbg_periph_disable	206
表 3-248. 函数 dbg_trace_pin_enable	206
表 3-249. 函数 dbg_trace_pin_disable	207
表 3-250. 函数 dbg_trace_pin_mode_set	207
表 3-251. DMA 寄存器	208
表 3-252. DMAMUX 寄存器	209
表 3-253. DMA 库函数	209

表 3-254. DMAMUX 库函数	210
表 3-255. 结构体 dma_multi_data_parameter_struct	211
表 3-256. 结构体 dma_single_data_parameter_struct	212
表 3-257. 结构体 dmamux_sync_parameter_struct	212
表 3-258. 结构体 dmamux_gen_parameter_struct	212
表 3-259. 枚举 dma_channel_enum	212
表 3-260. 枚举 dmamux_multiplexer_channel_enum	213
表 3-261. 枚举 dmamux_generator_channel_enum	213
表 3-262. 枚举 dmamux_interrupt_enum	214
表 3-263. 枚举 dmamux_flag_enum	215
表 3-264. 枚举 dmamux_interrupt_flag_enum	216
表 3-265. 函数 dma_deinit	218
表 3-266. 函数 dma_single_data_para_struct_init	218
表 3-267. 函数 dma_multi_data_para_struct_init	219
表 3-268. 函数 dma_single_data_mode_init	219
表 3-269. 函数 dma_multi_data_mode_init	220
表 3-270. 函数 dma_periph_address_config	222
表 3-271. 函数 dma_memory_address_config	222
表 3-272. 函数 dma_transfer_number_config	223
表 3-273. 函数 dma_transfer_number_get	224
表 3-274. 函数 dma_priority_config	224
表 3-275. 函数 dma_memory_burst_beats_config	225
表 3-276. 函数 dma_periph_burst_beats_config	226
表 3-277. 函数 dma_memory_width_config	227
表 3-278. 函数 dma_periph_width_config	227
表 3-279. 函数 dma_memory_address_generation_config	228
表 3-280. 函数 dma_peripheral_address_generation_config	229
表 3-281. 函数 dma_circulation_enable	230
表 3-282. 函数 dma_circulation_disable	230
表 3-283. 函数 dma_channel_enable	231
表 3-284. 函数 dma_channel_disable	231
表 3-285. 函数 dma_transfer_direction_config	232
表 3-286. 函数 dma_switch_buffer_mode_config	233
表 3-287. 函数 dma_using_memory_get	233
表 3-288. 函数 dma_switch_buffer_mode_enable	234
表 3-289. 函数 dma_switch_buffer_mode_disable	235
表 3-290. 函数 dma_fifo_status_get	235
表 3-291. 函数 dma_flag_get	236
表 3-292. 函数 dma_flag_clear	237
表 3-293. 函数 dma_interrupt_enable	238
表 3-294. 函数 dma_interrupt_disable	238
表 3-295. 函数 dma_interrupt_flag_get	239
表 3-296. 函数 dma_interrupt_flag_clear	240

表 3-297. 函数 dmamux_sync_struct_para_init	241
表 3-298. 函数 dmamux_synchronization_init	241
表 3-299. 函数 dmamux_synchronization_enable	242
表 3-300. 函数 dmamux_synchronization_disable	243
表 3-301. 函数 dmamux_event_generation_enable	243
表 3-302. 函数 dmamux_event_generation_disable	244
表 3-303. 函数 dmamux_gen_struct_para_init	244
表 3-304. 函数 dmamux_request_generator_init	245
表 3-305. 函数 dmamux_request_generator_channel_enable	246
表 3-306. 函数 dmamux_request_generator_channel_disable	246
表 3-307. 函数 dmamux_synchronization_polarity_config	247
表 3-308. 函数 dmamux_request_forward_number_config	248
表 3-309. 函数 dmamux_sync_id_config	248
表 3-310. 函数 dmamux_request_id_config	250
表 3-311. 函数 dmamux_trigger_polarity_config	258
表 3-312. 函数 dmamux_request_generate_number_config	259
表 3-313. 函数 dmamux_trigger_id_config	260
表 3-314. 函数 dmamux_flag_get	262
表 3-315. 函数 dmamux_flag_clear	262
表 3-316. 函数 dmamux_interrupt_enable	263
表 3-317. 函数 dmamux_interrupt_disable	263
表 3-318. 函数 dmamux_interrupt_flag_get	264
表 3-319. 函数 dmamux_interrupt_flag_clear	264
表 3-320. EDOUT 寄存器	265
表 3-321. EDOUT 库函数	265
表 3-322. 函数 edout_deinit	266
表 3-323. 函数 edout_init	266
表 3-324. 函数 edout_enable	267
表 3-325. 函数 edout_disable	268
表 3-326. 函数 edout_polarity_config	268
表 3-327. 函数 edout_max_location_value_config	269
表 3-328. 函数 edout_output_counter_update	269
表 3-329. 函数 edout_current_location_config	270
表 3-330. 函数 edout_current_location_get	270
表 3-331. 函数 edout_z_output_mode_config	271
表 3-332. 函数 edout_z_output_start_loc_and_width_config	271
表 3-333. EFUSE 寄存器	272
表 3-334. EFUSE 库函数	272
表 3-335. 枚举类型 efuse_system_para_size_enum	273
表 3-336. 枚举类型 efuse_system_para_index_enum	273
表 3-337. 枚举类型 efuse_state_enum	273
表 3-338. 枚举类型 efuse_interrupt_flag_enum	274
表 3-339. 函数 Function efuse_read	274

表 3-340. 函数 efuse_write	275
表 3-341. 函数 efuse_user_control_write	275
表 3-342. 函数 efuse_write	276
表 3-343. 函数 efuse_write	277
表 3-344. 函数 efuse_write	277
表 3-345. 函数 efuse_write	278
表 3-346. 函数 efuse_aes_key_crc_get.....	278
表 3-347. 函数 efuse_monitor_program_voltage_enable	279
表 3-348. 函数 efuse_monitor_program_voltage_disable	279
表 3-349. 函数 efuse_monitor_program_voltage_get	280
表 3-350. 函数 efuse_ldo_ready_get	280
表 3-351. 函数 efuse_flag_get	281
表 3-352. 函数 efuse_flag_clear	281
表 3-353. 函数 efuse_interrupt_enable	282
表 3-354. 函数 efuse_interrupt_disable	283
表 3-355. 函数 efuse_interrupt_flag_get.....	283
表 3-356. 函数 efuse_interrupt_flag_clear.....	284
表 3-357. EXMC 寄存器	285
表 3-358. EXMC 库函数	286
表 3-359. 结构体 exmc_norsram_timing_parameter_struct.....	287
表 3-360. 结构体 exmc_norsram_parameter_struct	287
表 3-361. 结构体 exmc_nand_timing_parameter_struct.....	287
表 3-362. 结构体 exmc_nand_parameter_struct	288
表 3-363. 结构体 exmc_sdram_timing_parameter_struct	288
表 3-364. 结构体 exmc_sdram_parameter_struct.....	288
表 3-365. 结构体 exmc_sdram_command_parameter_struct.....	289
表 3-366. 函数 exmc_norsram_deinit.....	289
表 3-367. 函数 exmc_norsram_struct_para_init	290
表 3-368. 函数 exmc_norsram_init	290
表 3-369. 函数 exmc_norsram_enable	292
表 3-370. 函数 exmc_norsram_disable	292
表 3-371. 函数 exmc_nand_deinit.....	293
表 3-372. 函数 exmc_nand_struct_para_init.....	293
表 3-373. 函数 exmc_nand_init	294
表 3-374. 函数 exmc_nand_enable	295
表 3-375. 函数 exmc_nand_disable	295
表 3-376. 函数 exmc_sdram_deinit.....	296
表 3-377. 函数 exmc_sdram_struct_para_init	296
表 3-378. 函数 exmc_sdram_init	297
表 3-379. 函数 exmc_norsram_sdram_remap_config.....	298
表 3-380. 函数 exmc_norsram_sdram_remap_get	299
表 3-381. 函数 exmc_norsram_consecutive_clock_config.....	300
表 3-382. 函数 exmc_norsram_page_size_config	300

表 3-383. 函数 exmc_nand_ecc_config	301
表 3-384. 函数 exmc_ecc_get	301
表 3-385. 函数 exmc_sdram_readsampler_enable	302
表 3-386. 函数 exmc_sdram_readsampler_disable	302
表 3-387. 函数 exmc_sdram_readsampler_config	303
表 3-388. 函数 exmc_sdram_command_config	304
表 3-389. 函数 exmc_sdram_refresh_count_set	304
表 3-390. 函数 exmc_sdram_autorefresh_number_set	305
表 3-391. 函数 exmc_sdram_write_protection_config	305
表 3-392. 函数 exmc_sdram_bankstatus_get	306
表 3-393. 函数 exmc_flag_get	307
表 3-394. 函数 exmc_flag_clear	307
表 3-395. 函数 exmc_interrupt_enable	308
表 3-396. 函数 exmc_interrupt_disable	309
表 3-397. 函数 exmc_interrupt_flag_get	310
表 3-398. 函数 exmc_interrupt_flag_clear	311
表 3-399. EXTI 寄存器	312
表 3-400. EXTI 库函数	312
表 3-401. 枚举类型 exti_line_enum	313
表 3-402. 枚举类型 exti_mode_enum	314
表 3-403. 枚举类型 exti_trig_type_enum	314
表 3-404. 函数 exti_deinit	314
表 3-405. 函数 exti_init	315
表 3-406. 函数 exti_interrupt_enable	315
表 3-407. 函数 exti_interrupt_disable	316
表 3-408. 函数 exti_event_enable	316
表 3-409. 函数 exti_event_disable	317
表 3-410. 函数 exti_software_interrupt_enable	317
表 3-411. 函数 exti_software_interrupt_disable	318
表 3-412. 函数 exti_flag_get	318
表 3-413. 函数 exti_flag_clear	319
表 3-414. 函数 exti_interrupt_flag_get	319
表 3-415. 函数 exti_interrupt_flag_clear	320
表 3-416. FAC 寄存器	320
表 3-417. FAC 库函数	321
表 3-418. 结构体 fac_parameter_struct	322
表 3-419. 结构体 fac_fixed_data_preload_struct	322
表 3-420. 结构体 fac_float_data_preload_struct	322
表 3-421. 函数 dac_deinit	323
表 3-422. 函数 fac_struct_para_init	323
表 3-423. 函数 fac_fixed_data_preload_init	324
表 3-424. 函数 fac_float_data_preload_init	324
表 3-425. 函数 fac_init	325

表 3-426. 函数 fac_fixed_buffer_preload	325
表 3-427. 函数 fac_float_buffer_preload	326
表 3-428. 函数 fac_fixed_data_preload	326
表 3-429. 函数 fac_float_data_preload	327
表 3-430. 函数 fac_reset	327
表 3-431. 函数 fac_clip_config	328
表 3-432. 函数 fac_float_enable	328
表 3-433. 函数 fac_float_disable	329
表 3-434. 函数 fac_dma_enable	329
表 3-435. 函数 fac_dma_disable	330
表 3-436. 函数 fac_x0_config	331
表 3-437. 函数 fac_x1_config	331
表 3-438. 函数 fac_y_config	332
表 3-439. 函数 fac_function_config	332
表 3-440. 函数 fac_start	333
表 3-441. 函数 fac_stop	333
表 3-442. 函数 fac_finish_calculate	334
表 3-443. 函数 fac_fixed_data_write	334
表 3-444. 函数 fac_fixed_data_read	335
表 3-445. 函数 fac_float_data_write	335
表 3-446. 函数 fac_float_data_read	336
表 3-447. 函数 fac_interrupt_enable	337
表 3-448. 函数 fac_interrupt_disable	337
表 3-449. 函数 fac_interrupt_flag_get	338
表 3-450. 函数 fac_flag_get	339
表 3-451. FMC 寄存器	339
表 3-452. FMC 库函数	340
表 3-453. 枚举类型 fmc_state_enum	342
表 3-454. 枚举类型 fmc_flag_enum	342
表 3-455. 枚举类型 fmc_interrupt_flag_enum	343
表 3-456. 枚举类型 fmc_interrupt_enum	343
表 3-457. 函数 fmc_unlock	343
表 3-458. 函数 fmc_lock	344
表 3-459. 函数 fmc_page_erase	344
表 3-460. 函数 fmc_mass_erase	345
表 3-461. 函数 fmc_protection_removed_mass_erase	345
表 3-462. 函数 fmc_word_program	346
表 3-463. 函数 fmc_doubleword_program	347
表 3-464. 函数 fmc_check_programming_area_enable	347
表 3-465. 函数 fmc_check_programming_area_disable	348
表 3-466. 函数 ob_unlock	348
表 3-467. 函数 ob_lock	349
表 3-468. 函数 ob_start	349

表 3-469. 函数 ob_factory_value_config	350
表 3-470. 函数 ob_secure_access_mode_enable	350
表 3-471. 函数 ob_secure_access_mode_enable	351
表 3-472. 函数 ob_security_protection_config.....	352
表 3-473. 函数 ob_bor_threshold_config	352
表 3-474. 函数 ob_low_power_config	353
表 3-475. 函数 ob_tcm_ecc_config.....	354
表 3-476. 函数 ob_iospeed_optimize_config	355
表 3-477. 函数 ob_tcm_shared_ram_config	356
表 3-478. 函数 ob_data_program	357
表 3-479. 函数 ob_boot_address_config	358
表 3-480. 函数 ob_dcrp_config.....	358
表 3-481. 函数 ob_secure_area_config.....	359
表 3-482. 函数 ob_write_protection_enable.....	360
表 3-483. 函数 ob_write_protection_disable.....	361
表 3-484. 函数 ob_secure_mode_get	362
表 3-485. 函数 ob_secure_mode_get	363
表 3-486. 函数 ob_bor_threshold_get	363
表 3-487. 函数 ob_low_power_get	364
表 3-488. 函数 ob_tcm_ecc_get	365
表 3-489. 函数 ob_secure_mode_get	366
表 3-490. 函数 ob_tcm_shared_ram_size_get.....	367
表 3-491. 函数 ob_data_get.....	368
表 3-492. 函数 ob_boot_address_get.....	368
表 3-493. 函数 ob_dcrp_area_get	369
表 3-494. 函数 ob_secure_area_get	369
表 3-495. 函数 ob_write_protection_get	370
表 3-496. 函数 fmc_no_rtdec_config.....	371
表 3-497. 函数 fmc_aes_iv_config	371
表 3-498. 函数 fmc_flash_ecc_get	372
表 3-499. 函数 fmc_no_rtdec_get	372
表 3-500. 函数 fmc_aes_iv_get.....	373
表 3-501. 函数 fmc_pid_get.....	373
表 3-502. 函数 fmc_flag_get.....	374
表 3-503. 函数 fmc_flag_clear	374
表 3-504. 函数 fmc_interrupt_enable	375
表 3-505. 函数 fmc_interrupt_disable	375
表 3-506. 函数 fmc_interrupt_flag_get.....	376
表 3-507. 函数 fmc_interrupt_flag_clear.....	376
表 3-508. FWDGT 寄存器.....	377
表 3-509. FWDGT 库函数.....	377
表 3-510. 函数 fwdgt_write_enable.....	377
表 3-511. 函数 fwdgt_write_disable	378

表 3-512. 函数 fwdgt_enable.....	378
表 3-513. 函数 fwdgt_prescaler_value_config.....	379
表 3-514. 函数 fwdgt_reload_value_config	379
表 3-515. 函数 fwdgt_window_value_config.....	380
表 3-516. 函数 fwdgt_counter_reload	381
表 3-517. 函数 fwdgt_config	381
表 3-518. 函数 fwdgt_flag_get	382
表 3-519. GPIO 寄存器.....	383
表 3-520. GPIO 库函数.....	383
表 3-521. 函数 gpio_deinit.....	384
表 3-522. 函数 gpio_mode_set	384
表 3-523. 函数 gpio_output_options_set.....	385
表 3-524. 函数 gpio_bit_set.....	386
表 3-525. 函数 gpio_bit_reset	387
表 3-526. 函数 gpio_bit_write	387
表 3-527. 函数 gpio_port_write	388
表 3-528. 函数 gpio_input_filter_set.....	389
表 3-529. 函数 gpio_input_bit_get	390
表 3-530. 函数 gpio_input_port_get	390
表 3-531. 函数 gpio_output_bit_get.....	391
表 3-532. 函数 gpio_output_port_get.....	391
表 3-533. 函数 gpio_af_set.....	392
表 3-534. 函数 gpio_pin_lock	393
表 3-535. 函数 gpio_bit_toggle.....	394
表 3-536. 函数 gpio_port_toggle	394
表 3-537. HPDF 寄存器.....	395
表 3-538. HPDF 库函数.....	396
表 3-539. 结构体 hpdf_channel_parameter_struct	398
表 3-540. 结构体 hpdf_filter_parameter_struct.....	398
表 3-541. 结构体 hpdf_rc_parameter_struct	399
表 3-542. 结构体 hpdf_ic_parameter_struct.....	399
表 3-543. 枚举类型 hpdf_channel_enum.....	399
表 3-544. 枚举类型 hpdf_filter_enum.....	399
表 3-545. 枚举类型 hpdf_flag_enum	400
表 3-546. 枚举类型 hpdf_interrput_flag_enum	401
表 3-547. 枚举类型 hpdf_interrput_enum	401
表 3-548. 函数 hpdf_deinit.....	402
表 3-549. 函数 hpdf_channel_struct_para_init.....	402
表 3-550. 函数 hpdf_filter_struct_para_init.....	403
表 3-551. 函数 hpdf_rc_struct_para_init.....	403
表 3-552. 函数 hpdf_ic_struct_para_init.....	404
表 3-553. 函数 hpdf_enable.....	404
表 3-554. 函数 hpdf_disable.....	405

表 3-555. 函数 hpdf_channel_init.....	405
表 3-556. 函数 hpdf_filter_init.....	406
表 3-557. 函数 hpdf_rc_init	407
表 3-558. 函数 hpdf_ic_init.....	408
表 3-559. 函数 hpdf_clock_output_config.....	409
表 3-560. 函数 hpdf_clock_output_source_config	409
表 3-561. 函数 hpdf_clock_output_duty_mode_disable.....	410
表 3-562. 函数 hpdf_clock_output_duty_mode_enable.....	410
表 3-563. 函数 hpdf_clock_output_divider_config	411
表 3-564. 函数 hpdf_channel_enable	411
表 3-565. 函数 hpdf_channel_disable	412
表 3-566. 函数 hpdf_spi_clock_source_config	412
表 3-567. 函数 hpdf_serial_interface_type_config	413
表 3-568. 函数 hpdf_malfunction_monitor_disable.....	414
表 3-569. 函数 hpdf_malfunction_monitor_enable.....	414
表 3-570. 函数 hpdf_clock_loss_disable	415
表 3-571. 函数 hpdf_clock_loss_enable	415
表 3-572. 函数 hpdf_channel_pin_redirection_disable.....	416
表 3-573. 函数 hpdf_channel_pin_redirection_enable	416
表 3-574. 函数 hpdf_channel_multiplexer_config.....	417
表 3-575. 函数 hpdf_data_pack_mode_config	417
表 3-576. 函数 hpdf_data_right_bit_shift_config	418
表 3-577. 函数 hpdf_calibration_offset_config.....	419
表 3-578. 函数 hpdf_malfunction_break_signal_config	419
表 3-579. 函数 hpdf_malfunction_counter_config.....	420
表 3-580. 函数 hpdf_write_parallel_data_standard_mode	421
表 3-581. 函数 hpdf_write_parallel_data_interleaved_mode	421
表 3-582. 函数 hpdf_write_parallel_data_dual_mode	422
表 3-583. 函数 hpdf_pulse_skip_update.....	422
表 3-584. 函数 hpdf_pulse_skip_read	423
表 3-585. 函数 hpdf_filter_enable	423
表 3-586. 函数 hpdf_filter_disable	424
表 3-587. 函数 hpdf_filter_config	425
表 3-588. 函数 hpdf_integrator_oversample	425
表 3-589. 函数 hpdf_threshold_monitor_filter_config	426
表 3-590. 函数 hpdf_threshold_monitor_filter_read_data	427
表 3-591. 函数 hpdf_threshold_monitor_fast_mode_disable	427
表 3-592. 函数 hpdf_threshold_monitor_fast_mode_enable	428
表 3-593. 函数 hpdf_threshold_monitor_channel.....	428
表 3-594. 函数 hpdf_threshold_monitor_high_threshold.....	429
表 3-595. 函数 hpdf_threshold_monitor_low_threshold	430
表 3-596. 函数 hpdf_high_threshold_break_signal	430
表 3-597. 函数 hpdf_low_threshold_break_signal.....	431

表 3-598. 函数 hpdf_extremes_monitor_channel	432
表 3-599. 函数 hpdf_extremes_monitor_maximum_get	432
表 3-600. 函数 hpdf_extremes_monitor_minimum_get	433
表 3-601. 函数 hpdf_conversion_time_get	433
表 3-602. 函数 hpdf_rc_continuous_disable	434
表 3-603. 函数 hpdf_rc_continuous_enable	434
表 3-604. 函数 hpdf_rc_start_by_software	435
表 3-605. 函数 hpdf_rc_syn_disable	436
表 3-606. 函数 hpdf_rc_syn_disable	436
表 3-607. 函数 hpdf_rc_dma_disable	437
表 3-608. 函数 hpdf_rc_dma_enable	437
表 3-609. 函数 hpdf_rc_channel_config	438
表 3-610. 函数 hpdf_rc_fast_mode_disable	438
表 3-611. 函数 hpdf_rc_fast_mode_enable	439
表 3-612. 函数 hpdf_rc_data_get	439
表 3-613. 函数 hpdf_rc_channel_get	440
表 3-614. 函数 hpdf_ic_start_by_software	440
表 3-615. 函数 hpdf_ic_syn_disable	441
表 3-616. 函数 hpdf_ic_syn_enable	441
表 3-617. 函数 hpdf_ic_dma_disable	442
表 3-618. 函数 hpdf_ic_dma_enable	442
表 3-619. 函数 hpdf_ic_scan_mode_disable	443
表 3-620. 函数 hpdf_ic_scan_mode_enable	443
表 3-621. 函数 hpdf_ic_trigger_signal_disable	444
表 3-622. 函数 hpdf_ic_trigger_signal_config	444
表 3-623. 函数 hpdf_ic_channel_config	445
表 3-624. 函数 hpdf_ic_data_get	446
表 3-625. 函数 hpdf_ic_channel_get	447
表 3-626. 函数 hpdf_flag_get	447
表 3-627. 函数 hpdf_flag_clear	448
表 3-628. 函数 hpdf_interrupt_enable	449
表 3-629. 函数 hpdf_interrupt_disable	450
表 3-630. 函数 hpdf_interrupt_flag_get	451
表 3-631. 函数 hpdf_interrupt_flag_clear	452
表 3-632. I2C 寄存器	453
表 3-633. I2C 库函数	453
表 3-634. 枚举类型 i2c_interrupt_flag_enum	455
表 3-635. 函数 i2c_deinit	456
表 3-636. 函数 i2c_timing_config	456
表 3-637. 函数 i2c_digital_noise_filter_config	457
表 3-638. 函数 i2c_analog_noise_filter_enable	458
表 3-639. 函数 i2c_analog_noise_filter_disable	458
表 3-640. 函数 i2c_master_clock_config	459

表 3-641. 函数 i2c_master_addressing	459
表 3-642. 函数 i2c_address10_header_enable	460
表 3-643. 函数 i2c_address10_header_disable	461
表 3-644. 函数 i2c_address10_enable	461
表 3-645. 函数 i2c_address10_disable	462
表 3-646. 函数 i2c_automatic_end_enable	462
表 3-647. 函数 i2c_automatic_end_disable	463
表 3-648. 函数 i2c_slave_response_to_gcall_enable	463
表 3-649. 函数 i2c_slave_response_to_gcall_disable	464
表 3-650. 函数 i2c_stretch_scl_low_enable	464
表 3-651. 函数 i2c_stretch_scl_low_disable	465
表 3-652. 函数 i2c_address_config	465
表 3-653. 函数 i2c_address_bit_compare_config	466
表 3-654. 函数 i2c_address_disable	467
表 3-655. 函数 i2c_second_address_config	467
表 3-656. 函数 i2c_second_address_disable	468
表 3-657. 函数 i2c_receved_address_get	469
表 3-658. 函数 i2c_slave_byte_control_enable	469
表 3-659. 函数 i2c_slave_byte_control_disable	470
表 3-660. 函数 i2c_nack_enable	470
表 3-661. 函数 i2c_nack_disable	471
表 3-662. 函数 i2c_wakeup_from_deepsleep_enable	471
表 3-663. 函数 i2c_wakeup_from_deepsleep_disable	472
表 3-664. 函数 i2c_enable	472
表 3-665. 函数 i2c_disable	473
表 3-666. 函数 i2c_start_on_bus	473
表 3-667. 函数 i2c_stop_on_bus	474
表 3-668. 函数 i2c_data_transmit	474
表 3-669. 函数 i2c_data_receive	475
表 3-670. 函数 i2c_reload_enable	475
表 3-671. 函数 i2c_reload_disable	476
表 3-672. 函数 i2c_transfer_byte_number_config	476
表 3-673. 函数 i2c_dma_enable	477
表 3-674. 函数 i2c_dma_disable	478
表 3-675. 函数 i2c_pec_transfer	478
表 3-676. 函数 i2c_pec_enable	479
表 3-677. 函数 i2c_pec_disable	479
表 3-678. 函数 i2c_pec_value_get	480
表 3-679. 函数 i2c_smbus_alert_enable	480
表 3-680. 函数 i2c_smbus_alert_disable	481
表 3-681. 函数 i2c_smbus_default_addr_enable	481
表 3-682. 函数 i2c_smbus_default_addr_disable	482
表 3-683. 函数 i2c_smbus_host_addr_enable	482

表 3-684. 函数 i2c_smbus_host_addr_disable.....	483
表 3-685. 函数 i2c_extented_clock_timeout_enable	483
表 3-686. 函数 i2c_extented_clock_timeout_disable	484
表 3-687. 函数 i2c_clock_timeout_enable	484
表 3-688. 函数 i2c_clock_timeout_disable	485
表 3-689. 函数 i2c_bus_timeout_b_config	485
表 3-690. 函数 i2c_bus_timeout_a_config	486
表 3-691. 函数 i2c_idle_clock_timeout_config	486
表 3-692. 函数 i2c_flag_get	487
表 3-693. 函数 i2c_flag_clear	488
表 3-694. 函数 i2c_interrupt_enable	489
表 3-695. 函数 i2c_interrupt_disable	489
表 3-696. 函数 i2c_interrupt_flag_get	490
表 3-697. 函数 i2c_interrupt_flag_clear	491
表 3-698. LPDTS 寄存器	492
表 3-699. LPDTS 库函数	493
表 3-700. 结构体 lpdts_parameter_struct	493
表 3-701. 函数 lpdts_deinit	494
表 3-702. 函数 lpdts_struct_para_init	494
表 3-703. 函数 lpdts_init	495
表 3-704. 函数 lpdts_enable	495
表 3-705. 函数 lpdts_disable	496
表 3-706. 函数 lpdts_soft_trigger_enable	496
表 3-707. 函数 lpdts_soft_trigger_disable	497
表 3-708. 函数 lpdts_high_threshold_set	497
表 3-709. 函数 lpdts_low_threshold_set	498
表 3-710. 函数 lpdts_ref_clock_source_config	498
表 3-711. 函数 lpdts_temperature_get	499
表 3-712. 函数 lpdts_flag_get	499
表 3-713. 函数 lpdts_interrupt_enable	500
表 3-714. 函数 lpdts_interrupt_disable	500
表 3-715. 函数 lpdts_interrupt_flag_get	501
表 3-716. 函数 lpdts_interrupt_flag_clear	502
表 3-717. MDMA 寄存器	502
表 3-718. MDMA 库函数	503
表 3-719. 结构体 mdma_parameter_struct	504
表 3-720. 结构体 mdma_multi_block_parameter_struct	505
表 3-721. 结构体 mdma_link_node_parameter_struct	505
表 3-722. 枚举 mdma_add_update_dir_enum	506
表 3-723. 枚举 mdma_channel_enum	506
表 3-724. 函数 mdma_deinit	506
表 3-725. 函数 mdma_channel_deinit	507
表 3-726. 函数 mdma_para_struct_init	507

表 3-727. 函数 mdma_multi_block_para_struct_init	508
表 3-728. 函数 mdma_link_node_para_struct_init	508
表 3-729. 函数 mdma_init	509
表 3-730. 函数 mdma_buffer_block_mode_config	510
表 3-731. 函数 mdma_multi_block_mode_config	511
表 3-732. 函数 mdma_node_create	512
表 3-733. 函数 mdma_node_add	512
表 3-734. 函数 mdma_node_delete	513
表 3-735. 函数 mdma_destination_address_config	514
表 3-736. 函数 mdma_source_address_config	514
表 3-737. 函数 mdma_destination_bus_config	515
表 3-738. 函数 mdma_source_bus_config	516
表 3-739. 函数 mdma_priority_config	516
表 3-740. 函数 mdma_endianness_config	517
表 3-741. 函数 mdma_alignment_config	518
表 3-742. 函数 mdma_source_burst_beats_config	518
表 3-743. 函数 mdma_destination_burst_beats_config	519
表 3-744. 函数 mdma_source_width_config	520
表 3-745. 函数 mdma_destination_width_config	521
表 3-746. 函数 mdma_source_increment_config	522
表 3-747. 函数 mdma_destination_increment_config	523
表 3-748. 函数 mdma_channel_bufferable_write_enable	524
表 3-749. 函数 mdma_channel_bufferable_write_disable	525
表 3-750. 函数 mdma_channel_software_request_enable	525
表 3-751. 函数 mdma_channel_enable	526
表 3-752. 函数 mdma_channel_disable	526
表 3-753. 函数 mdma_transfer_error_direction_get	527
表 3-754. 函数 mdma_transfer_error_address_get	527
表 3-755. 函数 mdma_flag_get	528
表 3-756. 函数 mdma_flag_clear	529
表 3-757. 函数 mdma_interrupt_enable	530
表 3-758. 函数 mdma_interrupt_disable	530
表 3-759. 函数 mdma_interrupt_flag_get	531
表 3-760. 函数 mdma_interrupt_flag_clear	532
表 3-761. OSPI 寄存器	533
表 3-762. OSPI 库函数	533
表 3-763. 结构体 ospi_parameter_struct	535
表 3-764. 结构体 ospi_regular_cmd_struct	535
表 3-765. 结构体 ospi_autopolling_struct	537
表 3-766. 枚举 ospi_interrupt_flag_enum	537
表 3-767. 函数 ospi_deinit	537
表 3-768. 函数 ospi_struct_init	538
表 3-769. 函数 ospi_init	538

表 3-770. 函数 <code>ospi_enable</code>	539
表 3-771. 函数 <code>ospi_disable</code>	539
表 3-772. 函数 <code>ospi_device_memory_type_config</code>	540
表 3-773. 函数 <code>ospi_device_memory_size_config</code>	541
表 3-774. 函数 <code>ospi_functional_mode_config</code>	541
表 3-775. 函数 <code>ospi_status_polling_config</code>	542
表 3-776. 函数 <code>ospi_status_mask_config</code>	543
表 3-777. 函数 <code>ospi_status_match_config</code>	543
表 3-778. 函数 <code>ospi_interval_cycle_config</code>	544
表 3-779. 函数 <code>ospi_fifo_level_config</code>	544
表 3-780. 函数 <code>ospi_chip_select_high_cycle_config</code>	545
表 3-781. 函数 <code>ospi_prescaler_config</code>	546
表 3-782. 函数 <code>ospi_dummy_cycles_config</code>	546
表 3-783. 函数 <code>ospi_delay_hold_cycle_config</code>	547
表 3-784. 函数 <code>ospi_sample_shift_config</code>	547
表 3-785. 函数 <code>ospi_data_length_config</code>	548
表 3-786. 函数 <code>ospi_instruction_config</code>	549
表 3-787. 函数 <code>ospi_address_config</code>	550
表 3-788. 函数 <code>ospi_alternate_bytes_config</code>	551
表 3-789. 函数 <code>ospi_data_config</code>	552
表 3-790. 函数 <code>ospi_data_transmit</code>	553
表 3-791. 函数 <code>ospi_data_receive</code>	554
表 3-792. 函数 <code>ospi_dma_enable</code>	554
表 3-793. 函数 <code>ospi_dma_disable</code>	555
表 3-794. 函数 <code>ospi_wrap_size_config</code>	555
表 3-795. 函数 <code>ospi_wrap_instruction_config</code>	556
表 3-796. 函数 <code>ospi_wrap_address_config</code>	557
表 3-797. 函数 <code>ospi_wrap_alternate_bytes_config</code>	558
表 3-798. 函数 <code>ospi_wrap_data_config</code>	559
表 3-799. 函数 <code>ospi_wrap_dummy_cycles_config</code>	560
表 3-800. 函数 <code>ospi_wrap_delay_hold_cycle_config</code>	561
表 3-801. 函数 <code>ospi_wrap_sample_shift_config</code>	561
表 3-802. 函数 <code>ospi_write_instruction_config</code>	562
表 3-803. 函数 <code>ospi_write_address_config</code>	563
表 3-804. 函数 <code>ospi_write_alternate_bytes_config</code>	564
表 3-805. 函数 <code>ospi_write_data_config</code>	566
表 3-806. 函数 <code>ospi_write_dummy_cycles_config</code>	567
表 3-807. 函数 <code>ospi_write_dummy_cycles_config</code>	567
表 3-808. 函数 <code>ospi_transmit</code>	568
表 3-809. 函数 <code>ospi_receive</code>	568
表 3-810. 函数 <code>ospi_autopolling_mode</code>	569
表 3-811. 函数 <code>ospi_interrupt_enable</code>	569
表 3-812. 函数 <code>ospi_interrupt_disable</code>	570

表 3-813. 函数 <code>ospi_fifo_level_get</code>	571
表 3-814. 函数 <code>ospi_flag_get</code>	571
表 3-815. 函数 <code>ospi_flag_clear</code>	572
表 3-816. 函数 <code>ospi_interrupt_flag_get</code>	572
表 3-817. 函数 <code>ospi_interrupt_flag_clear</code>	573
表 3-818. OSPIM 寄存器.....	574
表 3-819. OSPIM 库函数.....	574
表 3-820. 函数 <code>ospim_deinit</code>	575
表 3-821. 函数 <code>ospim_port_sck_config</code>	575
表 3-822. 函数 <code>ospim_port_csn_config</code>	576
表 3-823. 函数 <code>ospim_port_io3_0_config</code>	576
表 3-824. 函数 <code>ospim_port_io3_0_source_select</code>	577
表 3-825. 函数 <code>ospim_port_io7_4_config</code>	577
表 3-826. 函数 <code>ospim_port_io7_4_source_select</code>	578
表 3-827. NVIC 寄存器.....	579
表 3-828. SysTick 寄存器.....	580
表 3-829. MISC 库函数.....	580
表 3-830. 结构体 <code>mpu_region_init_struct</code>	581
表 3-831. 枚举类型 <code>IRQn_Type</code>	581
表 3-832. 函数 <code>nvic_priority_group_set</code>	585
表 3-833. 函数 <code>nvic_irq_enable</code>	585
表 3-834. 函数 <code>nvic_irq_disable</code>	586
表 3-835. 函数 <code>nvic_vector_table_set</code>	587
表 3-836. 函数 <code>system_lowpower_set</code>	587
表 3-837. 函数 <code>system_lowpower_reset</code>	588
表 3-838. 函数 <code>systick_clksource_set</code>	588
表 3-839. 函数 <code>mpu_region_struct_para_init</code>	589
表 3-840. 函数 <code>mpu_region_config</code>	590
表 3-841. 函数 <code>mpu_region_enable</code>	590
表 3-842. PMU 寄存器.....	591
表 3-843. PMU 库函数.....	591
表 3-844. 函数 <code>pmu_deinit</code>	592
表 3-845. 函数 <code>pmu_lvd_select</code>	593
表 3-846. 函数 <code>pmu_lvd_enable</code>	593
表 3-847. 函数 <code>pmu_lvd_disable</code>	594
表 3-848. 函数 <code>pmu_vavd_select</code>	594
表 3-849. 函数 <code>pmu_vavd_enable</code>	595
表 3-850. 函数 <code>pmu_vavd_disable</code>	595
表 3-851. 函数 <code>pmu_vovd_enable</code>	596
表 3-852. 函数 <code>pmu_vovd_disable</code>	596
表 3-853. 函数 <code>pmu_ldo_output_select</code>	597
表 3-854. 函数 <code>pmu_slido_output_select</code>	597
表 3-855. 函数 <code>pmu_vbat_charging_select</code>	598

表 3-856. 函数 pmu_vbat_charging_enable	599
表 3-857. 函数 pmu_vbat_charging_disable	599
表 3-858. 函数 pmu_vbat_temp_monitor_enable	600
表 3-859. 函数 pmu_vbat_temp_monitor_disable	600
表 3-860. 函数 pmu_usb_regulator_enable	601
表 3-861. 函数 pmu_usb_regulator_disable	601
表 3-862. 函数 pmu_usb_voltage_detector_enable	602
表 3-863. 函数 pmu_usb_voltage_detector_disable	602
表 3-864. 函数 pmu_smps_ldo_supply_config	603
表 3-865. 函数 pmu_to_sleepmode	603
表 3-866. 函数 pmu_to_deepsleepmode	604
表 3-867. 函数 pmu_to_standbymode	604
表 3-868. 函数 pmu_wakeup_pin_enable	605
表 3-869. 函数 pmu_wakeup_pin_disable	605
表 3-870. 函数 pmu_backup_write_enable	606
表 3-871. 函数 pmu_backup_write_disable	607
表 3-872. 函数 pmu_backup_voltage_stabilizer_enable	607
表 3-873. 函数 pmu_backup_voltage_stabilizer_disable	608
表 3-874. 函数 pmu_enter_deepsleep_wait_time_config	608
表 3-875. 函数 pmu_exit_deepsleep_wait_time_config	609
表 3-876. 函数 pmu_flag_get	609
表 3-877. 函数 pmu_flag_clear	610
表 3-878. RAMECCMU 寄存器	611
表 3-879. RAMECCMU 库函数	611
表 3-880. 枚举类型 rameccmu_monitor_enum	612
表 3-881. 函数 rameccmu_deinit	612
表 3-882. 函数 rameccmu_monitor_failing_address_get	613
表 3-883. 函数 rameccmu_monitor_failing_data_low_bits_get	614
表 3-884. 函数 rameccmu_monitor_failing_data_high_bits_get	615
表 3-885. 函数 rameccmu_monitor_failing_ecc_error_code_get	616
表 3-886. 函数 rameccmu_global_interrupt_enable	617
表 3-887. 函数 rameccmu_global_interrupt_disable	618
表 3-888. 函数 rameccmu_monitor_interrupt_enable	618
表 3-889. 函数 rameccmu_monitor_interrupt_disable	620
表 3-890. 函数 rameccmu_monitor_flag_get	621
表 3-891. 函数 rameccmu_monitor_flag_clear	622
表 3-892. 函数 rameccmu_monitor_interrupt_flag_get	623
表 3-893. 函数 rameccmu_monitor_interrupt_flag_clear	625
表 3-894. RCU 寄存器	626
表 3-895. RCU 库函数	628
表 3-896. 枚举类型 rcu_periph_enum	629
表 3-897. 枚举类型 rcu_periph_sleep_enum	632
表 3-898. 枚举类型 rcu_periph_reset_enum	634

表 3-899. 枚举类型 rcu_flag_enum	636
表 3-900. 枚举类型 rcu_int_flag_enum.....	637
表 3-901. 枚举类型 rcu_int_flag_clear_enum.....	637
表 3-902. 枚举类型 rcu_int_enum	638
表 3-903. 枚举类型 rcu_osci_type_enum.....	638
表 3-904. 枚举类型 rcu_clock_freq_enum	639
表 3-905. 枚举类型 usart_idx_enum	639
表 3-906. 枚举类型 i2c_idx_enum	639
表 3-907. 枚举类型 can_idx_enum	640
表 3-908. 枚举类型 adc_idx_enum	640
表 3-909. 枚举类型 usbhs_idx_enum	640
表 3-910. 枚举类型 pll_idx_enum	640
表 3-911. 枚举类型 spi_idx_enum	640
表 3-912. 函数 rcu_deinit.....	641
表 3-913. 函数 rcu_periph_clock_enable	641
表 3-914. 函数 rcu_periph_clock_disable	642
表 3-915. 函数 rcu_periph_clock_sleep_enable.....	642
表 3-916. 函数 rcu_periph_clock_sleep_disable.....	643
表 3-917. 函数 rcu_periph_reset_enable	643
表 3-918. 函数 rcu_periph_reset_disable	644
表 3-919. 函数 rcu_bkp_reset_enable	644
表 3-920. 函数 rcu_bkp_reset_disable	645
表 3-921. 函数 rcu_system_clock_source_config	645
表 3-922. 函数 rcu_system_clock_source_get.....	646
表 3-923. 函数 rcu_ahb_clock_config	646
表 3-924. 函数 rcu_apb1_clock_config	647
表 3-925. 函数 rcu_apb2_clock_config	647
表 3-926. 函数 rcu_apb3_clock_config	648
表 3-927. 函数 rcu_apb4_clock_config	648
表 3-928. 函数 rcu_ckout0_config	649
表 3-929. 函数 rcu_ckout1_config	650
表 3-930. 函数 rcu_pll_input_output_clock_range_config.....	651
表 3-931. 函数 rcu_pll_fractional_config	652
表 3-932. 函数 rcu_pll_fractional_latch_enable	652
表 3-933. 函数 rcu_pll_fractional_latch_disable	653
表 3-934. 函数 rcu_pll_source_config.....	653
表 3-935. 函数 rcu_pll0_config	654
表 3-936. 函数 rcu_pll1_config	655
表 3-937. 函数 rcu_pll2_config	656
表 3-938. 函数 rcu_pll_clock_output_enable.....	656
表 3-939. 函数 rcu_pll_clock_output_disable.....	657
表 3-940. 函数 rcu_pllusb0_config	658
表 3-941. 函数 rcu_pllusb1_config	659

表 3-942. 函数 rcu_rtc_clock_config	660
表 3-943. 函数 rcu_rtc_div_config	660
表 3-944. 函数 rcu_ck48m_clock_config	661
表 3-945. 函数 rcu_pll48m_clock_config	662
表 3-946. 函数 rcu_irc64mdiv_clock_config	662
表 3-947. 函数 rcu_irc64mdiv_freq_get	663
表 3-948. 函数 rcu_timer_clock_prescaler_config	663
表 3-949. 函数 rcu_spi_clock_config	664
表 3-950. 函数 rcu_sdio_clock_config	665
表 3-951. 函数 rcu_usart_clock_config	665
表 3-952. 函数 rcu_i2c_clock_config	666
表 3-953. 函数 rcu_can_clock_config	667
表 3-954. 函数 rcu_adc_clock_config	668
表 3-955. 函数 rcu_exmc_clock_config	668
表 3-956. 函数 rcu_hpdf_clock_config	669
表 3-957. 函数 rcu_per_clock_config	670
表 3-958. 函数 rcu_usbhs_pll1qpsc_config	670
表 3-959. 函数 rcu_usb48m_clock_config	671
表 3-960. 函数 rcu_usbhs_clock_config	672
表 3-961. 函数 rcu_usbhs_clock_selection_enable	672
表 3-962. 函数 rcu_usbhs_clock_selection_disable	673
表 3-963. 函数 rcu_lxtal_drive_capability_config	673
表 3-964. 函数 rcu_osci_stab_wait	674
表 3-965. 函数 rcu_osci_on	674
表 3-966. 函数 rcu_osci_off	675
表 3-967. 函数 rcu_osci_bypass_mode_enable	675
表 3-968. 函数 rcu_osci_bypass_mode_disable	676
表 3-969. 函数 rcu_irc64m_adjust_value_set	676
表 3-970. 函数 rcu_lpirc4m_adjust_value_set	677
表 3-971. 函数 rcu_hxtal_clock_monitor_enable	677
表 3-972. 函数 rcu_hxtal_clock_monitor_disable	678
表 3-973. 函数 rcu_lxtal_clock_monitor_enable	678
表 3-974. 函数 rcu_lxtal_clock_monitor_disable	679
表 3-975. 函数 rcu_clock_freq_get	679
表 3-976. 函数 rcu_flag_get	680
表 3-977. 函数 rcu_all_reset_flag_clear	680
表 3-978. 函数 rcu_interrupt_enable	681
表 3-979. 函数 rcu_interrupt_disable	681
表 3-980. 函数 rcu_interrupt_flag_get	682
表 3-981. 函数 rcu_interrupt_flag_clear	682
表 3-982. RTC 寄存器	683
表 3-983. RTC 库函数	684
表 3-984. 结构体 rtc_parameter_struct	685

表 3-985. 结构体 rtc_alarm_struct	686
表 3-986. 结构体 rtc_timestamp_struct.....	686
表 3-987. 结构体 rtc_tamper_struct.....	686
表 3-988. 函数 rtc_deinit.....	687
表 3-989. 函数 rtc_init	687
表 3-990. 函数 rtc_init_mode_enter.....	688
表 3-991. 函数 rtc_init_mode_exit	689
表 3-992. 函数 rtc_register_sync_wait.....	689
表 3-993. 函数 rtc_current_time_get	690
表 3-994. 函数 rtc_subsecond_get	690
表 3-995. 函数 rtc_alarm_config	691
表 3-996. 函数 rtc_alarm_subsecond_config	691
表 3-997. 函数 rtc_alarm_enable.....	692
表 3-998. 函数 rtc_alarm_disable.....	693
表 3-999. 函数 rtc_alarm_get	693
表 3-1000. 函数 rtc_alarm_subsecond_get.....	694
表 3-1001. 函数 rtc_timestamp_enable.....	694
表 3-1002. 函数 rtc_timestamp_disable.....	695
表 3-1003. 函数 rtc_timestamp_internalevent_config.....	695
表 3-1004. 函数 rtc_timestamp_get	696
表 3-1005. 函数 rtc_timestamp_subsecond_get	696
表 3-1006. 函数 rtc_tamper_enable	697
表 3-1007. 函数 rtc_tamper_disable	697
表 3-1008. 函数 rtc_output_pin_select	698
表 3-1009. 函数 rtc_alarm_output_config	698
表 3-1010. 函数 rtc_calibration_output_config	699
表 3-1011. 函数 rtc_hour_adjust	700
表 3-1012. 函数 rtc_second_adjust	700
表 3-1013. 函数 rtc_bypass_shadow_enable.....	701
表 3-1014. 函数 rtc_bypass_shadow_disable	701
表 3-1015. 函数 rtc_refclock_detection_enable	702
表 3-1016. 函数 rtc_refclock_detection_disable	702
表 3-1017. 函数 rtc_wakeup_enable.....	703
表 3-1018. 函数 rtc_wakeup_disable	703
表 3-1019. 函数 rtc_wakeup_clock_set.....	704
表 3-1020. 函数 rtc_wakeup_timer_set	705
表 3-1021. 函数 rtc_wakeup_timer_get.....	705
表 3-1022. 函数 rtc_smooth_calibration_config	706
表 3-1023. 函数 rtc_interrupt_enable	707
表 3-1024. 函数 rtc_interrupt_disable	707
表 3-1025. 函数 rtc_flag_get	708
表 3-1026. 函数 rtc_flag_clear	709
表 3-1027. SPI/I2S 寄存器	709

表 3-1028. SPI/I2S 库函数.....	710
表 3-1029. 结构体 spi_parameter_struct.....	712
表 3-1030. 函数 spi_i2s_deinit.....	712
表 3-1031. 函数 spi_struct_para_init.....	713
表 3-1032. 函数 spi_init.....	713
表 3-1033. 函数 spi_enable	714
表 3-1034. 函数 spi_disable	715
表 3-1035. 函数 i2s_init.....	715
表 3-1036. 函数 i2s_psc_config	716
表 3-1037. 函数 i2s_enable	718
表 3-1038. 函数 i2s_disable	718
表 3-1039. 函数 spi_io_config	719
表 3-1040. 函数 spi_nss_idleness_delay_set.....	719
表 3-1041. 函数 spi_data_frame_delay_set.....	720
表 3-1042. 函数 spi_master_receive_clock_delay_set	721
表 3-1043. 函数 spi_slave_receive_clock_delay_set	721
表 3-1044. 函数 spi_master_receive_clock_delay_clear	722
表 3-1045. 函数 spi_slave_receive_clock_delay_clear.....	722
表 3-1046. 函数 spi_nss_output_control.....	723
表 3-1047. 函数 spi_nss_polarity_set	723
表 3-1048. 函数 spi_nss_output_enable.....	724
表 3-1049. 函数 spi_nss_output_disable.....	724
表 3-1050. 函数 spi_nss_internal_high.....	725
表 3-1051. 函数 spi_nss_internal_low.....	725
表 3-1052. 函数 spi_dma_enable	726
表 3-1053. 函数 spi_dma_disable	727
表 3-1054. 函数 spi_i2s_data_frame_size_config.....	727
表 3-1055. 函数 spi_i2s_data_transmit.....	728
表 3-1056. 函数 spi_i2s_data_receive	728
表 3-1057. 函数 spi_bidirectional_transfer_config.....	729
表 3-1058. 函数 spi_master_transfer_start	730
表 3-1059. 函数 spi_current_data_num_config.....	730
表 3-1060. 函数 spi_reload_data_num_config	731
表 3-1061. 函数 spi_crc_polynomial_set.....	731
表 3-1062. 函数 spi_crc_polynomial_get.....	732
表 3-1063. 函数 spi_crc_length_config.....	732
表 3-1064. 函数 spi_crc_on.....	733
表 3-1065. 函数 spi_crc_off.....	734
表 3-1066. 函数 spi_crc_get.....	734
表 3-1067. 函数 spi_crc_full_size_enable	735
表 3-1068. 函数 spi_crc_full_size_enable	735
表 3-1069. 函数 spi_tcrc_init_pattern.....	736
表 3-1070. 函数 spi_tcrc_init_pattern.....	736

表 3-1071. 函数 spi_ti_mode_enable.....	737
表 3-1072. 函数 spi_ti_mode_disable.....	738
表 3-1073. 函数 spi_quad_enable	738
表 3-1074. 函数 spi_quad_disable	739
表 3-1075. 函数 spi_quad_write_enable	739
表 3-1076. 函数 spi_quad_read_enable	740
表 3-1077. 函数 spi_quad_io23_output_enable.....	740
表 3-1078. 函数 spi_quad_io23_output_disable	741
表 3-1079. 函数 spi_underrun_operation	741
表 3-1080. 函数 spi_underrun_config	742
表 3-1081. 函数 spi_underrun_data_config	742
表 3-1082. 函数 spi_suspend_mode_config	743
表 3-1083. 函数 spi_suspend_request.....	744
表 3-1084. 函数 spi_related_ios_af_enable.....	744
表 3-1085. 函数 spi_related_ios_af_enable.....	745
表 3-1086. 函数 spi_af_gpio_control.....	745
表 3-1087. 函数 spi_i2s_interrupt_enable	746
表 3-1088. 函数 spi_i2s_interrupt_disable	747
表 3-1089. 函数 spi_i2s_interrupt_flag_get.....	748
表 3-1090. 函数 spi_i2s_flag_get	749
表 3-1091. 函数 spi_crc_error_clear.....	750
表 3-1092. 函数 spi_i2s_rxfifo_plevel_get.....	751
表 3-1093. 函数 spi_i2s_remain_data_num_get.....	752
表 3-1094. 函数 spi_fifo_threshold_level_set.....	752
表 3-1095. 函数 spi_enable	753
表 3-1096. 函数 spi_disable	753
表 3-1097. 函数 spi_byte_access_enable.....	754
表 3-1098. 函数 spi_byte_access_disable.....	754
表 3-1099. SYSCFG 寄存器	755
表 3-1100. SYSCFG 库函数.....	756
表 3-1101. 枚举类型 timer_channel_input_enum	757
表 3-1102. 函数 syscfg_deinit.....	762
表 3-1103. 函数 syscfg_i2c_fast_mode_plus_enable.....	762
表 3-1104. 函数 syscfg_i2c_fast_mode_plus_disable.....	763
表 3-1105. 函数 syscfg_analog_switch_enable.....	764
表 3-1106. 函数 syscfg_analog_switch_disable.....	765
表 3-1107. 函数 syscfg_enet_phy_interface_config	765
表 3-1108. 函数 syscfg_exti_line_config	766
表 3-1109. 函数 syscfg_lockup_enable.....	767
表 3-1110. 函数 syscfg_lockup_disable.....	767
表 3-1111. 函数 syscfg_timer_input_source_select	768
表 3-1112. 函数 syscfg_compensation_config	769
表 3-1113. 函数 syscfg_io_low_voltage_speed_optimization_enable	769

表 3-1114. 函数 syscfg_io_low_voltage_speed_optimization_disable	770
表 3-1115. 函数 syscfg_pnmos_compensation_code_set	770
表 3-1116. 函数 syscfg_secure_sram_size_set	771
表 3-1117. 函数 syscfg_secure_sram_size_get	772
表 3-1118. 函数 syscfg_bootmode_get	772
表 3-1119. 函数 syscfg_tcm_wait_state_enable	773
表 3-1120. 函数 syscfg_tcm_wait_state_disable	773
表 3-1121. 函数 syscfg_fpu_interrupt_enable	774
表 3-1122. 函数 syscfg_fpu_interrupt_disable	775
表 3-1123. 函数 syscfg_compensation_flag_get	776
表 3-1124. 函数 syscfg_cpu_cache_status_get	776
表 3-1125. 函数 syscfg_brownout_reset_threshold_level_get	777
表 3-1126. TIMER 寄存器	778
表 3-1127. TIMER 库函数	779
表 3-1128. 结构体 timer_parameter_struct	782
表 3-1129. 结构体 timer_break_parameter_struct	783
表 3-1130. 结构体 timer_oc_parameter_struct	784
表 3-1131. 结构体 timer_omc_parameter_struct	784
表 3-1132. 结构体 timer_ic_parameter_struct	784
表 3-1133. 结构体 timer_free_complementary_parameter_struct	784
表 3-1134. 函数 timer_deinit	785
表 3-1135. 函数 timer_struct_para_init	785
表 3-1136. 函数 timer_init	786
表 3-1137. 函数 timer_enable	787
表 3-1138. 函数 timer_disable	787
表 3-1139. 函数 timer_auto_reload_shadow_enable	788
表 3-1140. 函数 timer_auto_reload_shadow_disable	788
表 3-1141. 函数 timer_update_event_enable	789
表 3-1142. 函数 timer_update_event_disable	789
表 3-1143. 函数 timer_counter_alignment	790
表 3-1144. 函数 timer_counter_up_direction	791
表 3-1145. 函数 timer_counter_down_direction	791
表 3-1146. 函数 timer_prescaler_config	792
表 3-1147. 函数 timer_repetition_value_config	793
表 3-1148. 函数 timer_runtime_repetition_value_read	793
表 3-1149. 函数 timer_autoreload_value_config	794
表 3-1150. 函数 timer_autoreload_value_read	795
表 3-1151. 函数 timer_counter_value_config	795
表 3-1152. 函数 timer_counter_read	796
表 3-1153. 函数 timer_prescaler_read	797
表 3-1154. 函数 timer_single_pulse_mode_config	797
表 3-1155. 函数 timer_delayable_single_pulse_mode_config	798
表 3-1156. 函数 timer_update_source_config	799

表 3-1157. 函数 timer_dma_enable.....	800
表 3-1158. 函数 timer_dma_disable.....	801
表 3-1159. 函数 timer_channel_dma_request_source_select	802
表 3-1160. 函数 timer_dma_transfer_config	803
表 3-1161. 函数 timer_event_software_generate	805
表 3-1162. 函数 timer_break_struct_para_init.....	807
表 3-1163. 函数 timer_break_config	807
表 3-1164. 函数 timer_break_enable	808
表 3-1165. 函数 timer_break_disable	809
表 3-1166. 函数 timer_automatic_output_enable	810
表 3-1167. 函数 timer_automatic_output_disable.....	810
表 3-1168. 函数 timer_primary_output_config	811
表 3-1169. 函数 timer_channel_control_shadow_config.....	811
表 3-1170. 函数 timer_channel_control_shadow_update_config	812
表 3-1171. 函数 timer_channel_output_struct_para_init.....	813
表 3-1172. 函数 timer_channel_output_config	813
表 3-1173. 函数 timer_channel_output_mode_config	814
表 3-1174. 函数 timer_channel_output_pulse_value_config	816
表 3-1175. 函数 timer_channel_output_shadow_config.....	817
表 3-1176. 函数 timer_channel_output_clear_config	818
表 3-1177. 函数 timer_channel_output_polarity_config	819
表 3-1178. 函数 timer_channel_complementary_output_polarity_config	820
表 3-1179. 函数 timer_channel_output_state_config	821
表 3-1180. 函数 timer_channel_complementary_output_state_config	822
表 3-1181. 函数 timer_channel_input_struct_para_init	822
表 3-1182. 函数 timer_input_capture_config	823
表 3-1183. 函数 timer_channel_input_capture_prescaler_config.....	824
表 3-1184. 函数 timer_channel_capture_value_register_read.....	825
表 3-1185. 函数 timer_input_pwm_capture_config	826
表 3-1186. 函数 timer_hall_mode_config	827
表 3-1187. 函数 timer_multi_mode_channel_output_parameter_struct_init	827
表 3-1188. 函数 timer_multi_mode_channel_output_config.....	828
表 3-1189. 函数 timer_multi_mode_channel_mode_config	829
表 3-1190. 函数 timer_input_trigger_source_select.....	830
表 3-1191. 函数 timer_master_output0_trigger_source_select.....	832
表 3-1192. 函数 timer_master_output1_trigger_source_select.....	833
表 3-1193. 函数 timer_slave_mode_select	834
表 3-1194. 函数 timer_master_slave_mode_config	835
表 3-1195. 函数 timer_external_trigger_config.....	835
表 3-1196. 函数 timer_quadrature_decoder_mode_config	836
表 3-1197. 函数 timer_non_quadrature_decoder_mode_config	838
表 3-1198. 函数 timer_internal_clock_config.....	838
表 3-1199. 函数 timer_internal_trigger_as_external_clock_config.....	839

表 3-1200. 函数 timer_external_trigger_as_external_clock_config.....	840
表 3-1201. 函数 timer_external_clock_mode0_config	841
表 3-1202. 函数 timer_external_clock_mode1_config	842
表 3-1203. 函数 timer_external_clock_mode1_disable.....	843
表 3-1204. 函数 timer_write_chxval_register_config	844
表 3-1205. 函数 timer_output_value_selection_config.....	845
表 3-1206. 函数 timer_commutation_control_shadow_register_config	845
表 3-1207. 函数 timer_output_match_pulse_select.....	846
表 3-1208. 函数 timer_channel_composite_pwm_mode_config.....	847
表 3-1209. 函数 timer_channel_composite_pwm_mode_output_pulse_value_config	848
表 3-1210. 函数 timer_channel_additional_compare_value_config	849
表 3-1211. 函数 timer_channel_additional_output_shadow_config.....	849
表 3-1212. 函数 timer_channel_additional_compare_value_read	850
表 3-1213. 函数 timer_break_external_source_config.....	851
表 3-1214. 函数 timer_break_external_polarity_config	852
表 3-1215. 函数 timer_break_lock_config	853
表 3-1216. 函数 timer_break_lock_release_config	854
表 3-1217. 函数 timer_channel_break_control_config.....	854
表 3-1218. 函数 timer_channel_dead_time_config.....	855
表 3-1219. 函数 timer_free_complementary_struct_para_init	856
表 3-1220. 函数 timer_channel_free_complementary_config	857
表 3-1221. 函数 timer_watchdog_value_config.....	857
表 3-1222. 函数 timer_watchdog_value_read	858
表 3-1223. 函数 timer_decoder_disconnection_detection_config.....	859
表 3-1224. 函数 timer_decoder_jump_detection_config	859
表 3-1225. 函数 timer_upif_backup_config	860
表 3-1226. 函数 timer_upifbu_bit_get	861
表 3-1227. 函数 timer_flag_get.....	861
表 3-1228. 函数 timer_flag_clear.....	863
表 3-1229. 函数 timer_interrupt_enable.....	865
表 3-1230. 函数 timer_interrupt_disable.....	866
表 3-1231. 函数 timer_interrupt_flag_get	867
表 3-1232. 函数 timer_interrupt_flag_clear	869
表 3-1233. TMU 寄存器.....	871
表 3-1234. TMU 库函数.....	871
表 3-1235. 结构体 tmu_parameter_struct	871
表 3-1236. 函数 tmu_deinit.....	872
表 3-1237. 函数 tmu_struct_para_init	872
表 3-1238. 函数 tmu_init.....	873
表 3-1239. 函数 tmu_read_interrupt_enable	874
表 3-1240. 函数 tmu_read_interrupt_disable	874
表 3-1241. 函数 tmu_dma_read_enable.....	875
表 3-1242. 函数 tmu_dma_read_disable.....	875

表 3-1243. 函数 tmu_dma_write_enable	876
表 3-1244. 函数 tmu_dma_write_disable	876
表 3-1245. 函数 tmu_one_q31_write	877
表 3-1246. 函数 tmu_two_q31_write	877
表 3-1247. 函数 tmu_two_q15_write	878
表 3-1248. 函数 tmu_one_q31_read	878
表 3-1249. 函数 tmu_two_q31_read	879
表 3-1250. 函数 tmu_two_q15_read	879
表 3-1251. TRIGSEL 寄存器	880
表 3-1252. TRIGSEL 库函数	882
表 3-1253. 枚举类型 trigselsource_enum	882
表 3-1254. 枚举类型 trigselperiph_enum	885
表 3-1255. 函数 trigsels_init	887
表 3-1256. 函数 trigsels_init	887
表 3-1257. 函数 trigsels_trigger_source_get	888
表 3-1258. 函数 trigsels_trigger_source_set	888
表 3-1259. 函数 trigsels_trigger_lock_get	889
表 3-1260. TRNG 寄存器	890
表 3-1261. TRNG 库函数	890
表 3-1262. 枚举 trng_inmod_enum	891
表 3-1263. 枚举 trng_outmod_enum	891
表 3-1264. 枚举 trng_modsel_enum	891
表 3-1265. 枚举 trng_flag_enum	891
表 3-1266. 枚举 trng_int_flag_enum	891
表 3-1267. 函数 trng_deinit	891
表 3-1268. 函数 trng_enable	892
表 3-1269. 函数 trng_disable	892
表 3-1270. 函数 trng_lock	893
表 3-1271. 函数 trng_mode_config	893
表 3-1272. 函数 trng_postprocessing_enable	894
表 3-1273. 函数 trng_postprocessing_disable	895
表 3-1274. 函数 trng_conditioning_enable	896
表 3-1275. 函数 trng_conditioning_disable	897
表 3-1276. 函数 trng_conditioning_input_bitwidth	897
表 3-1277. 函数 trng_conditioning_output_bitwidth	898
表 3-1278. 函数 trng_replace_test_enable	899
表 3-1279. 函数 trng_replace_test_disable	900
表 3-1280. 函数 trng_hash_init_enable	900
表 3-1281. 函数 trng_hash_init_disable	901
表 3-1282. 函数 trng_powermode_config	901
表 3-1283. 函数 trng_clockdiv_config	902
表 3-1284. 函数 trng_clockerror_detection_enable	903
表 3-1285. 函数 trng_clockerror_detection_disable	903

表 3-1286. 函数 trng_get_true_random_data.....	904
表 3-1287. 函数 trng_conditioning_reset_enable	905
表 3-1288. 函数 trng_conditioning_reset_disable	905
表 3-1289. 函数 trng_conditioning_algo_config	906
表 3-1290. 函数 trng_health_tests_config.....	907
表 3-1291. 函数 trng_flag_get.....	908
表 3-1292. 函数 trng_interrupt_enable.....	909
表 3-1293. 函数 trng_interrupt_disable.....	909
表 3-1294. 函数 trng_interrupt_flag_get	910
表 3-1295. 函数 trng_interrupt_flag_clear	910
表 3-1296. USART 寄存器.....	911
表 3-1297. USART 库函数.....	911
表 3-1298. 枚举类型 usart_flag_enum.....	914
表 3-1299. 枚举类型 usart_interrupt_flag_enum	914
表 3-1300. 枚举类型 usart_interrupt_enum	915
表 3-1301. 枚举类型 usart_invert_enum	916
表 3-1302. 函数 usart_deinit	916
表 3-1303. 函数 usart_baudrate_set.....	917
表 3-1304. 函数 usart_parity_config.....	917
表 3-1305. 函数 usart_word_length_set	918
表 3-1306. 函数 usart_stop_bit_set	919
表 3-1307. 函数 usart_enable	919
表 3-1308. 函数 usart_disable	920
表 3-1309. 函数 usart_transmit_config	920
表 3-1310. 函数 usart_receive_config	921
表 3-1311. 函数 usart_data_first_config	922
表 3-1312. 函数 usart_invert_config.....	922
表 3-1313. 函数 usart_overrun_enable	923
表 3-1314. 函数 usart_overrun_disable	923
表 3-1315. 函数 usart_oversample_config.....	924
表 3-1316. 函数 usart_sample_bit_config	925
表 3-1317. 函数 usart_receiver_timeout_enable	925
表 3-1318. 函数 usart_receiver_timeout_disable	926
表 3-1319. 函数 usart_receiver_timeout_threshold_config	926
表 3-1320. 函数 usart_data_transmit.....	927
表 3-1321. 函数 usart_data_receive.....	928
表 3-1322. 函数 usart_command_enable	928
表 3-1323. 函数 usart_address_0_match_mode_enable	929
表 3-1324. 函数 usart_address_0_match_mode_disable	929
表 3-1325. 函数 usart_address_1_match_mode_enable	930
表 3-1326. 函数 usart_address_1_match_mode_disable	930
表 3-1327. 函数 usart_address_0_config	931
表 3-1328. 函数 usart_address_1_config	932

表 3-1329. 函数 usart_address_0_detection_mode_config	932
表 3-1330. 函数 usart_address_1_detection_mode_config	933
表 3-1331. 函数 usart_mute_mode_enable	934
表 3-1332. 函数 usart_mute_mode_disable	934
表 3-1333. 函数 usart_mute_mode_wakeup_config.....	935
表 3-1334. 函数 usart_lin_mode_enable.....	935
表 3-1335. 函数 usart_lin_mode_disable.....	936
表 3-1336. 函数 usart_lin_break_dection_length_config	936
表 3-1337. 函数 usart_halfduplex_enable	937
表 3-1338. 函数 usart_halfduplex_disable	938
表 3-1339. 函数 usart_clock_enable.....	938
表 3-1340. 函数 usart_clock_disable.....	939
表 3-1341. 函数 usart_synchronous_clock_config	939
表 3-1342. 函数 usart_guard_time_config.....	940
表 3-1343. 函数 usart_smartcard_mode_enable	941
表 3-1344. 函数 usart_smartcard_mode_disable	941
表 3-1345. 函数 usart_smartcard_mode_nack_enable	942
表 3-1346. 函数 usart_smartcard_mode_nack_disable	942
表 3-1347. 函数 usart_smartcard_mode_early_nack_enable	943
表 3-1348. 函数 usart_smartcard_mode_early_nack_disable	943
表 3-1349. 函数 usart_smartcard_autoretry_config	944
表 3-1350. 函数 usart_block_length_config.....	944
表 3-1351. 函数 usart_irda_mode_enable	945
表 3-1352. 函数 usart_irda_mode_disable	945
表 3-1353. 函数 usart_prescaler_config	946
表 3-1354. 函数 usart_irda_lowpower_config.....	946
表 3-1355. 函数 usart_hardware_flow_rts_config.....	947
表 3-1356. 函数 usart_hardware_flow_cts_config.....	948
表 3-1357. 函数 usart_hardware_flow_coherence_config	948
表 3-1358. 函数 usart_rs485_driver_enable	949
表 3-1359. 函数 usart_rs485_driver_disable.....	949
表 3-1360. 函数 usart_driver_asserttime_config.....	950
表 3-1361. 函数 usart_driver_deasserttime_config	951
表 3-1362. 函数 usart_depolarity_config	951
表 3-1363. 函数 usart_dma_receive_config	952
表 3-1364. 函数 usart_dma_transmit_config	952
表 3-1365. 函数 usart_reception_error_dma_disable	953
表 3-1366. 函数 usart_reception_error_dma_enable	954
表 3-1367. 函数 usart_wakeup_enable	954
表 3-1368. 函数 usart_wakeup_disable	955
表 3-1369. 函数 usart_wakeup_mode_config.....	955
表 3-1370. 函数 usart_fifo_enable	956
表 3-1371. 函数 usart_fifo_disable	956

表 3-1372. 函数 usart_transmit_fifo_threshold_config.....	957
表 3-1373. 函数 usart_receive_fifo_threshold_config.....	958
表 3-1374. 函数 usart_receive_fifo_counter_number	959
表 3-1375. 函数 usart_flag_get.....	959
表 3-1376. 函数 usart_flag_clear.....	960
表 3-1377. 函数 usart_interrupt_enable.....	961
表 3-1378. 函数 usart_interrupt_disable.....	962
表 3-1379. 函数 usart_interrupt_flag_get	962
表 3-1380. 函数 usart_interrupt_flag_clear	963
表 3-1381. VREF 寄存器.....	965
表 3-1382. VREF 库函数.....	965
表 3-1383. 函数 vref_deinit.....	965
表 3-1384. 函数 vref_enable.....	966
表 3-1385. 函数 vref_disable.....	966
表 3-1386. 函数 vref_high_impedance_mode_enable.....	967
表 3-1387. 函数 vref_high_impedance_mode_disable.....	967
表 3-1388. 函数 vref_status_get.....	968
表 3-1389. 函数 vref_voltage_select.....	968
表 3-1390. 函数 vref_calib_value_set.....	969
表 3-1391. 函数 vref_calib_value_get.....	969
表 3-1392. WWDGT 寄存器.....	970
表 3-1393. WWDGT 库函数.....	970
表 3-1394. 函数 wwdgt_deinit.....	970
表 3-1395. 函数 wwdgt_enable	971
表 3-1396. 函数 wwdgt_counter_update.....	971
表 3-1397. 函数 wwdgt_config.....	972
表 3-1398. 函数 wwdgt_interrupt_enable	973
表 3-1399. 函数 wwdgt_flag_get	973
表 3-1400. 函数 wwdgt_flag_clear	974
表 3-1401. ESC_INTIC 寄存器	975
表 3-1402. ESC_INTIC 库函数	975
表 3-1403.枚举 intc_enable_enum.....	975
表 3-1404.枚举 intc_disable_enum.....	976
表 3-1405.枚举 intc_get_flag_enum	976
表 3-1406.枚举 intc_clear_flag_enum	976
表 3-1407. 函数 intc_deas_config.....	977
表 3-1408. 函数 intc_deas_config.....	977
表 3-1409. 函数 intc_irq_config.....	978
表 3-1410. 函数 intc_interrupt_enable	979
表 3-1411. 函数 intc_interrupt_disable	979
表 3-1412. 函数 intc_interrupt_flag_get.....	980
表 3-1413. 函数 intc_interrupt_flag_clear.....	980
表 3-1414. 函数 intc_deasstat_get.....	981

表 3-1415. 函数 intc_irqstat_get.....	981
表 3-1416. ESC_OSPI 寄存器.....	982
表 3-1417. ESC_OSPI 库函数.....	983
表 3-1418. 函数 ospi_hw_init.....	983
表 3-1419. 函数 ospi_enable_qspi_mode.....	984
表 3-1420. 函数 ospi_enable_ospi_mode.....	984
表 3-1421. 函数 ospi_reset_spi_mode	985
表 3-1422. 函数 ospi_write	986
表 3-1423. 函数 ospi_read	987
表 3-1424. 函数 OSPIWriteRegUsingCSR.....	988
表 3-1425. 函数 OSPIReadRegUsingCSR.....	989
表 3-1426. 函数 OSPIReadRegister	991
表 3-1427. 函数 OSPIWriteRegister	991
表 3-1428.ESC_PHY 控制寄存器.....	992
表 3-1429. ESC_PHY 库函数	993
表 3-1430. 函数 esc_phy_read.....	993
表 3-1431. 函数 esc_phy_write.....	993
表 3-1432. 函数 esc_mmd_read	994
表 3-1433. 函数 esc_mmd_write	995
表 3-1434. ESC_PMU 寄存器.....	995
表 3-1435. ESC_PMU 库函数.....	996
表 3-1436.枚举 pmu_esc_fun_enum	996
表 3-1437.函数 pmu_esc_power_mang_mode_config	996
表 3-1438.函数 pmu_esc_wake_up_mode_config.....	997
表 3-1439. 函数 pmu_esc_led_wm_config.....	998
表 3-1440. 函数 pmu_esc_led_inact_stat_config.....	998
表 3-1441. 函数 pmu_esc_ready_stat_get.....	999
表 3-1442. 函数 pmu_esc_edwol_stat_get.....	999
表 3-1443. 函数 pmu_esc_edwol_stat_clear.....	1000
表 3-1444. 函数 pmu_esc_fun_config	1000
表 3-1445. 函数 pmu_esc_byte_test.....	1001
表 3-1446. 函数 pmu_esc_sleep_mode_enable	1001
表 3-1447. 函数 pmu_esc_sleep_mode_disable	1002
表 3-1448. ESC_SYSCFG 寄存器	1003
表 3-1449. ESC_SYSCFG 库函数	1003
表 3-1450. 函数 syscfg_get_chip_id.....	1004
表 3-1451. 函数 syscfg_get_chip_version.....	1004
表 3-1452. 函数 syscfg_efuse_read.....	1005
表 3-1453. ESC_TIMER 寄存器	1005
表 3-1454. ESC_TIMER 库函数	1006
表 3-1455. 函数 esc_timer_enable	1006
表 3-1456. 函数 esc_timer_disable	1006
表 3-1457. 函数 esc_timer_autoreload_value_config	1007

表 3-1458. 函数 <code>esc_timer_counter_read</code>	1007
表 3-1459. 函数 <code>esc_frc_counter_read</code>	1008
表 4-1. 版本历史	1009

1. 介绍

本手册介绍了32位基于ARM微控制器GD32H75E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GD32H75E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
CAN	控制器局域网络
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
EDOUT	编码器分频输出控制器
EFUSE	熔丝

外设缩写	说明
EXMC	外部存储器控制器
EXTI	外部中断事件控制器
FAC	滤波算法加速器
FMC	闪存控制器
FWDGT	独立看门狗
GPIO	通用和备用输入
HPDF	高性能数字滤波器
I2C	内部集成电路总线接口
LPDTS	低功耗数字温度传感器
MDMA	主机直接存储器访问控制器
OSPI	八线SPI接口
OSPIIM	OSPI I/O管理器
MISC	嵌套中断向量列表控制器
PMU	电源管理单元
RAMECCMU	RAM ECC监视器单元
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
TMU	三角函数加速器
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
VREF	VREF
WWDGT	窗口看门狗
ESC	EtherCAT从站控制器

1.1.2. 命名规则

固件库遵从以下命名规则：

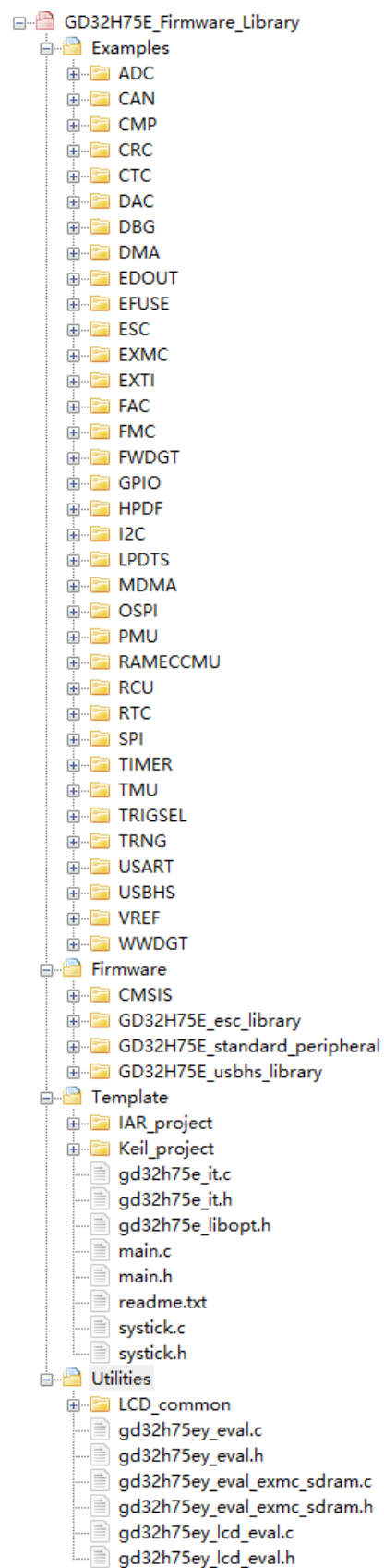
- XXX表示任一外设缩写，例如：ADC。更多缩写相关信息参阅[外设缩写](#)；
- 源文件和头文件命名都以“gd32h75e_”作为开头，例如：gd32h75e_adc.h；
- 常量仅被应用于一个文件的，定义于该文件中；被应用于多个文件的，在对应头文件中定义。所有常量都由英文字母大写书写；
- 寄存器作为常量处理。他们的命名都由英文字母大写书写。在大多数情况下，寄存器缩写规范与本用户手册一致；
- 变量名采用全部小写，有多个单词组成的，在单词之间以下划线分隔；
- 外设函数的命名以该外设的缩写加下划线为开头，有多个单词组成的，在单词之间以下划线分隔，所有外设函数都由英文字母小写书写。

2. 固件库概述

2.1. 文件组织结构

GD32H75E_Firmware_Library，文件组织结构见下图：

图 2-1. GD32H75E 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；
- **gd32h75e_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）；
- **gd32h75e_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）；
- **gd32h75e_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® M7**内核的支持文件、基于**Cortex® M7**内核处理器的启动代码和库引导文件以及基于**GD32H75E**的全局头文件和系统配置文件；
- **GD32H75E_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。

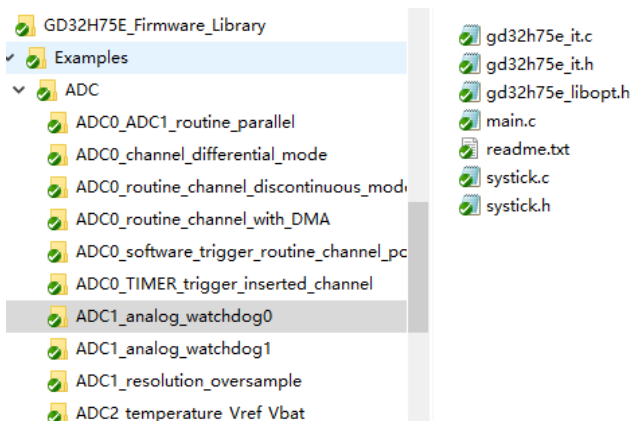
2.1.3. Template 文件夹

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）。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_watchdog0**”，如下图所示：

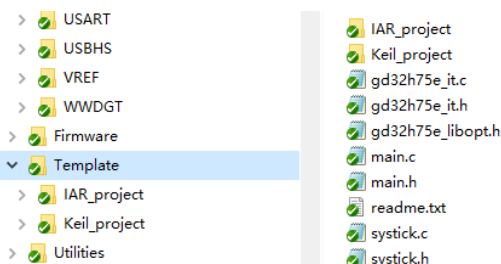
图 2-2. 选择外设例程文件



拷贝文件

打开“Template”文件夹，将“IAR_project”和“Keil_project”两个文件夹保留，其他文件都删除，然后将“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：

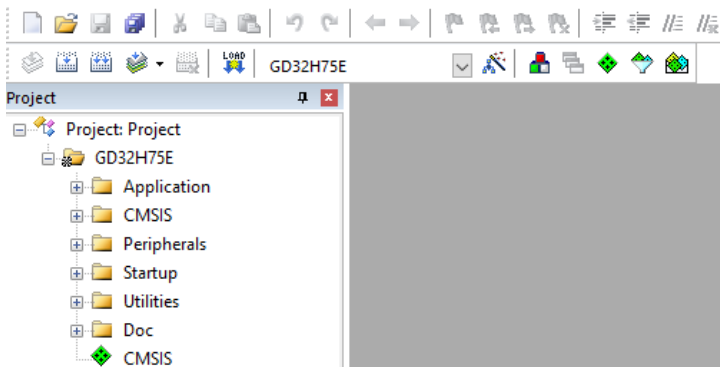
图 2-3. 拷贝外设例程文件



打开工程

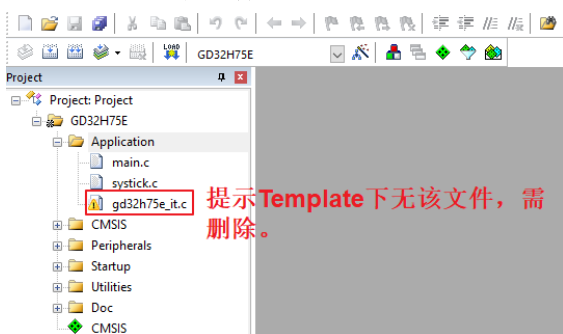
GD提供Keil和IAR两种版本的工程，根据客户所安装的软件，打开不同的project，如“Keil_project”，打开\Template\Keil_project\Project.uvprojx，如下图所示：

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：

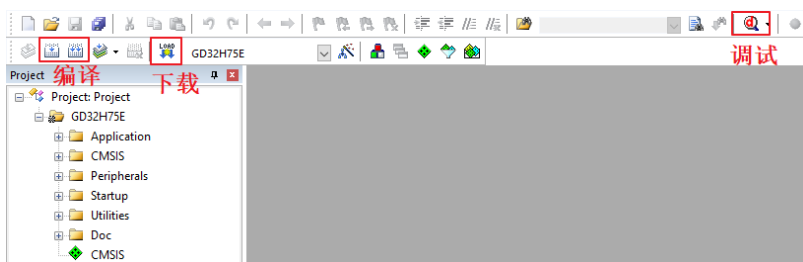
图 2-5. 配置工程文件



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：

- gd32h759i_eval.h文件是运行固件库例程所需关于评估板的头文件；
- gd32h759i_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
gd32h75e_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。
gd32h75e_it.h	头文件，包含所有中断处理函数原形。
gd32h75e_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

文件名	描述
	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。
gd32h75e_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。
gd32h75e_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

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_JOFFx	注入通道数据偏移寄存器x(x=0..3)
ADC_WDHT0	看门狗0高阈值寄存器
ADC_WDLT0	看门狗0低阈值寄存器
ADC_RSQx	规则序列寄存器x(x=0..8)
ADC_ISQx	注入序列寄存器x(x=0..2)
ADC_IDATAx (x=0..3)	注入数据寄存器x

寄存器名称	寄存器描述
ADC_RDATA	规则数据寄存器
ADC_OVSAMPCTL	过采样控制寄存器
ADC_WD1SR	看门狗1通道选择寄存器
ADC_WD2SR	看门狗2通道选择寄存器
ADC_WDHT1	看门狗1高阈值寄存器
ADC_WDLT1	看门狗1低阈值寄存器
ADC_WDHT2	看门狗2高阈值寄存器
ADC_WDLT2	看门狗2低阈值寄存器
ADC_DIFCTL	差分模式控制寄存器
ADC_SSTAT	摘要状态寄存器
ADC_SYNCCTL	同步控制寄存器
ADC_SYNCDATA0	同步规则数据寄存器0
ADC_SYNCDATA1	同步规则数据寄存器1

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_clock_config	ADC时钟配置
adc_special_function_config	使能或禁能ADC特殊功能
adc_data_alignment_config	配置ADC数据对齐方式
adc_enable	使能ADC外设
adc_disable	禁能ADC外设
adc_calibration_mode_config	ADC校准模式配置
adc_calibration_number	ADC校准次数
adc_calibration_enable	ADC校准使能
adc_resolution_config	配置ADC分辨率
adc_internal_channel_config	使能或禁能ADC内部通道
adc_dma_mode_enable	ADC DMA请求使能
adc_dma_mode_disable	ADC DMA请求禁能
adc_dma_request_after_last_enable	当DMA=1时，在每个规则通道转换结束后提出一个DMA请求
adc_dma_request_after_last_disable	DMA机制在DMA控制器的传输结束信号之后失能
adc_hpdf_mode_enable	使能HPDF功能
adc_hpdf_mode_disable	禁能HPDF功能
adc_discontinuous_mode_config	配置ADC间断模式
adc_channel_length_config	配置规则通道组或注入通道组的长度
adc_regular_channel_config	配置ADC规则通道组
adc_inserted_channel_config	配置ADC注入通道组
adc_inserted_channel_offset_config	配置ADC注入通道组数据偏移值

库函数名称	库函数描述
adc_channel_differential_mode_config	配置ADC通道差分模式
adc_external_trigger_config	配置ADC外部触发
adc_software_trigger_enable	ADC软件触发使能
adc_end_of_conversion_config	配置规则组转换结束模式
adc_regular_data_read	读ADC规则组数据寄存器
adc_inserted_data_read	读ADC注入组数据寄存器
adc_watchdog0_single_channel_enable	配置ADC模拟看门狗0单通道有效
adc_watchdog0_group_channel_enable	配置ADC模拟看门狗0在通道组有效
adc_watchdog0_disable	ADC模拟看门狗0禁能
adc_watchdog1_channel_config	配置ADC模拟看门狗1通道
adc_watchdog2_channel_config	配置ADC模拟看门狗2通道
adc_watchdog1_disable	ADC模拟看门狗1禁能
adc_watchdog2_disable	ADC模拟看门狗2禁能
adc_watchdog0_threshold_config	配置ADC模拟看门狗0阈值
adc_watchdog1_threshold_config	配置ADC模拟看门狗1阈值
adc_watchdog2_threshold_config	配置ADC模拟看门狗2阈值
adc_oversample_mode_config	配置ADC过采样模式
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
adc_sync_mode_config	ADC同步模式配置
adc_sync_delay_config	配置ADC同步模式两次采样之间的延时
adc_sync_dma_config	ADC同步DMA模式选择
adc_sync_dma_request_after_last_enable	根据SYNCDMA位来产生DMA请求
adc_sync_dma_request_after_last_disable	当检测到来自DMA控制器的传输结束信号后，DMA机制失能
adc_sync_master_adc_regular_data0_read	ADC同步模式主ADC规则数据寄存器0读取
adc_sync_slave_adc_regular_data0_read	ADC同步模式从ADC规则数据寄存器0读取
adc_sync_regular_data_read1	ADC同步模式规则数据寄存器1读取

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_clock_config

函数adc_clock_config描述见下表：

表 3-5. 函数 adc_clock_config

函数名称	adc_clock_config
函数原形	void adc_clock_config(uint32_t adc_periph, uint32_t prescaler);
功能描述	配置所有ADC时钟
先决条件	-
被调用函数	-
输入参数{in}	
adc_periph	ADC peripheral
ADCx(x=0,1,2)	ADC peripheral selection
输入参数{in}	
prescaler	预分频
ADC_CLK_SYNC_ HCLK_DIV2	HCLK时钟2分频
ADC_CLK_SYNC_ HCLK_DIV4	HCLK时钟4分频
ADC_CLK_SYNC_ HCLK_DIV6	HCLK时钟6分频

ADC_CLK_SYNC_ HCLK_DIV8	HCLK时钟8分频
ADC_CLK_SYNC_ HCLK_DIV10	HCLK时钟10分频
ADC_CLK_SYNC_ HCLK_DIV12	HCLK时钟12分频
ADC_CLK_SYNC_ HCLK_DIV14	HCLK时钟14分频
ADC_CLK_SYNC_ HCLK_DIV16	HCLK时钟16分频
ADC_CLK_ASYNC_ DIV1	HCLK时钟不分频
ADC_CLK_ASYNC_ DIV2	异步时钟2分频
ADC_CLK_ASYNC_ DIV4	异步时钟4分频
ADC_CLK_ASYNC_ DIV6	异步时钟6分频
ADC_CLK_ASYNC_ DIV8	异步时钟8分频
ADC_CLK_ASYNC_ DIV10	异步时钟10分频
ADC_CLK_ASYNC_ DIV12	异步时钟12分频
ADC_CLK_ASYNC_ DIV16	异步时钟16分频
ADC_CLK_ASYNC_ DIV32	异步时钟32分频
ADC_CLK_ASYNC_ DIV64	异步时钟64分频
ADC_CLK_ASYNC_ DIV128	异步时钟128分频
ADC_CLK_ASYNC_ DIV256	异步时钟256分频
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* configure the ADC clock: HCLK div2 */
```

```
adc_clock_config(ADC0, ADC_CLK_SYNC_HCLK_DIV2);
```

函数 adc_special_function_config

函数adc_special_function_config描述见下表：

表 3-6. 函数 adc_special_function_config

函数名称	adc_special_function_config
函数原形	void adc_special_function_config(uint32_t adc_periph, uint32_t function, ControlStatus newvalue);
功能描述	使能或禁能ADC特殊功能
先决条件	-
被调用函数	-
输入参数{in}	
adc_periph	ADC外设
ADCx(x=0,1,2)	ADC外设选择
输入参数{in}	
function	功能配置
ADC_SCAN_MODE	扫描模式选择
ADC_INSERTED_CHANNEL_AUTO	注入组自动转换
ADC_CONTINUOUS_MODE	连续模式选择
输入参数{in}	
newvalue	功能使能禁能
ENABLE	使能
DISABLE	禁能
输出参数{out}	
-	-
返回值	
-	-

例如：

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

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

函数 adc_data_alignment_config

函数adc_data_alignment_config描述见下表：

表 3-7. 函数 adc_data_alignment_config

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

被调用函数	-
输入参数{in}	
adc_periph	ADC外设
<i>ADCx(x=0,1,2)</i>	ADC外设选择
输入参数{in}	
data_alignment	数据对齐方式选择
<i>ADC_DATAALIGN_RIGHT</i>	最低有效位对齐
<i>ADC_DATAALIGN_LEFT</i>	最高有效位对齐
输出参数{out}	
-	-
返回值	
-	-

例如：

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

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

函数 **adc_enable**

函数adc_enable描述见下表：

表 3-8. 函数 **adc_enable**

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

例如：

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

```
adc_enable(ADC0);
```

函数 adc_disable

函数adc_disable描述见下表:

表 3-9. 函数 adc_disable

函数名称	adc_disable
函数原形	void adc_disable(uint32_t adc_periph);
功能描述	禁能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_calibration_mode_config

函数adc_calibration_mode_config描述见下表:

表 3-10. 函数 adc_calibration_mode_config

函数名称	adc_calibration_mode_config
函数原形	void adc_calibration_mode_config(uint32_t adc_periph, uint32_t clb_mode);
功能描述	ADC校准模式配置
先决条件	-
被调用函数	-
输入参数{in}	
adc_periph	ADC外设
ADCx(x=0,1)	ADC外设选择
输入参数{in}	
clb_mode	校准模式
ADC_CALIBRATION_OFFSET_MISMATCH	偏移、不匹配模式
ADC_CALIBRATION_OFFSET	偏移模式
返回值	

-	-
---	---

Example:

```
/* configure ADC0 calibration mode */
```

```
adc_calibration_mode_config (ADC0, ADC_CALIBRATION_OFFSET);
```

函数 `adc_calibration_number`

函数 `adc_calibration_number` 描述见下表:

表 3-11. 函数 `adc_calibration_number`

函数名称	adc_calibration_number
函数原形	void adc_calibration_number(uint32_t adc_periph, uint32_t clb_num);
功能描述	ADC校准次数
先决条件	-
被调用函数	-
输入参数{in}	
adc_periph	ADC外设
ADCx(x=0,1)	ADC外设选择
输入参数{in}	
clb_num	校准次数
ADC_CALIBRATION_NUM1	校准1次
ADC_CALIBRATION_NUM2	校准2次
ADC_CALIBRATION_NUM4	校准4次
ADC_CALIBRATION_NUM8	校准8次
ADC_CALIBRATION_NUM16	校准16次
ADC_CALIBRATION_NUM32	校准32次
输出参数{out}	
-	-
返回值	
-	-

Example:

```
/* configure ADC0 calibration number */
```

```
adc_calibration_number(ADC0, ADC_CALIBRATION_NUM1);
```

函数 adc_calibration_enable

函数adc_calibration_enable描述见下表:

表 3-12. 函数 adc_calibration_enable

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

例如:

```
/* ADC0 calibration and reset calibration */
```

```
adc_calibration_enable(ADC0);
```

函数 adc_resolution_config

函数adc_resolution_config描述见下表:

表 3-13. 函数 adc_resolution_config

函数名称	adc_resolution_config
函数原形	void adc_resolution_config(uint32_t adc_periph , uint32_t resolution);
功能描述	配置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_14B	14位分辨率（只针对ADC0/ADC1）
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（只针对ADC2）
输出参数{out}	
-	-
返回值	
-	-

Example:

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

```
adc_resolution_config(ADC0, ADC_RESOLUTION_10B);
```

函数 adc_internal_channel_config

函数adc_internal_channel_config描述见下表:

表 3-14. 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	内部通道
ADC_CHANNEL_INTERNAL_TEMPSENSOR	温度传感器通道
ADC_CHANNEL_INTERNAL_VREFINT	内部参考电压通道
ADC_CHANNEL_INTERNAL_VBAT	电池电压通道
ADC_CHANNEL_INTERNAL_HP_TEMPSENSOR	高精度温度传感器通道
输入参数{in}	
newvalue	通道使能/禁能
ENABLE	使能
DISABLE	禁能
输出参数{out}	
-	-
返回值	

-	-
---	---

Example:

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

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

函数 `adc_dma_mode_enable`

函数`adc_dma_mode_enable`描述见下表:

表 3-15. 函数 `adc_dma_mode_enable`

函数名称	<code>adc_dma_mode_enable</code>
函数原形	<code>void adc_dma_mode_enable(uint32_t adc_periph);</code>
功能描述	ADC DMA请求使能
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
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* enable ADC0 DMA request */
```

```
adc_dma_mode_enable(ADC0);
```

函数 `adc_dma_mode_disable`

函数`adc_dma_mode_disable`描述见下表:

表 3-16. 函数 `adc_dma_mode_disable`

函数名称	<code>adc_dma_mode_disable</code>
函数原形	<code>void adc_dma_mode_disable(uint32_t adc_periph);</code>
功能描述	ADC DMA请求禁能
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
输出参数{out}	
-	-

返回值	
-	-

例如：

```
/* disable ADC0 DMA request */
```

```
adc_dma_mode_disable(ADC0);
```

函数 `adc_dma_request_after_last_enable`

函数`adc_dma_request_after_last_enable`描述见下表：

表 3-17. 函数 `adc_dma_request_after_last_enable`

函数名称	<code>adc_dma_request_after_last_enable</code>
函数原形	<code>void adc_dma_request_after_last_enable(uint32_t adc_periph);</code>
功能描述	当DMA=1时，在每个规则通道转换结束后提出一个DMA请求
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
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* when DMA=1, the DMA engine issues a request at end of each regular conversion for ADC0 */
```

```
adc_dma_request_after_last_enable(ADC0);
```

函数 `adc_dma_request_after_last_disable`

函数`adc_dma_request_after_last_disable`描述见下表：

表 3-18. 函数 `adc_dma_request_after_last_disable`

函数名称	<code>adc_dma_request_after_last_disable</code>
函数原形	<code>void adc_dma_request_after_last_disable(uint32_t adc_periph);</code>
功能描述	DMA机制在DMA控制器的传输结束信号之后失能
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

输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* the DMA engine is disabled after the end of transfer signal from DMA controller is detected for ADC0 */
```

```
adc_dma_request_after_last_disable(ADC0);
```

函数 `adc_hpdf_mode_enable`

函数`adc_hpdf_mode_enable`描述见下表：

表 3-19. 函数 `adc_hpdf_mode_enable`

函数名称	<code>adc_hpdf_mode_enable</code>
函数原形	<code>void adc_hpdf_mode_enable(uint32_t adc_periph);</code>
功能描述	HPDF模式使能
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
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* enable ADC0 hpdf mode */
```

```
adc_hpdf_mode_enable(ADC0);
```

函数 `adc_hpdf_mode_disable`

函数`adc_hpdf_mode_disable`描述见下表：

表 3-20. 函数 `adc_hpdf_mode_disable`

函数名称	<code>adc_hpdf_mode_disable</code>
函数原形	<code>void adc_hpdf_mode_disable(uint32_t adc_periph);</code>
功能描述	HPDF模式禁能
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 hpdf mode */
```

```
adc_hpdf_mode_disable(ADC0);
```

函数 **adc_discontinuous_mode_config**

函数adc_discontinuous_mode_config描述见下表：

表 3-21. 函数 **adc_discontinuous_mode_config**

函数名称	adc_discontinuous_mode_config
函数原形	void adc_discontinuous_mode_config(uint32_t adc_periph, uint8_t adc_channel_group, uint32_t length);
功能描述	配置ADC间断模式
先决条件	-
被调用函数	-
输入参数{in}	
adc_periph	ADC外设
ADCx(x=0,1,2)	ADC外设选择
输入参数{in}	
adc_channel_group	通道组选择
ADC_REGULAR_CHANNEL	规则通道组
ADC_INSERTED_CHANNEL	注入通道组
ADC_CHANNEL_DISCON_DISABLE	规则通道组和注入通道组间断模式禁能
输入参数{in}	
length	间断模式下的转换数目，规则通道组取值为1..8，注入通道组取值无意义
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* configure ADC0 discontinuous mode */
```



```
adc_discontinuous_mode_config(ADC0, ADC_REGULAR_CHANNEL, 6);
```

函数 `adc_channel_length_config`

函数`adc_channel_length_config`描述见下表：

表 3-22. 函数 `adc_channel_length_config`

函数名称	<code>adc_channel_length_config</code>
函数原形	<code>void adc_channel_length_config(uint32_t adc_periph, uint8_t adc_channel_group, uint32_t length);</code>
功能描述	配置规则通道组或注入通道组的长度
先决条件	-
被调用函数	-
输入参数{in}	
<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_group</code>	通道组选择
<code>ADC_REGULAR_CHANNEL</code>	规则通道组
<code>ADC_INSERTED_CHANNEL</code>	注入通道组
输入参数{in}	
<code>length</code>	通道长度，规则通道组为1-16，注入通道组为1-4
输出参数{out}	
-	-
返回值	
-	-

例如：

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

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

函数 `adc_regular_channel_config`

函数`adc_regular_channel_config`描述见下表：

表 3-23. 函数 `adc_regular_channel_config`

函数名称	<code>adc_regular_channel_config</code>
函数原形	<code>void adc_regular_channel_config(uint32_t adc_periph, uint8_t rank, uint8_t adc_channel, uint32_t sample_time);</code>
功能描述	配置ADC规则通道组
先决条件	-

被调用函数	-
输入参数{in}	
adc_periph	ADC外设
ADCx(x=0,1,2)	ADC外设选择
输入参数{in}	
rank	规则组通道序列，取值范围为0~15
输入参数{in}	
adc_channel	ADC通道选择
ADC_CHANNEL_x	ADC 通道x (x=0..20)
输入参数{in}	
sample_time	采样时间x, ADC0/ADC1范围 (x=0..809), ADC2范围 (x=0..638). 例如.10'dx: ADC0/1为(x+3.5)周期, ADC2为(x+2.5)周期
输出参数{out}	
-	-
返回值	
-	-

例如:

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

```
adc_regular_channel_config(ADC0, 1, ADC_CHANNEL_0, 10);
```

函数 adc_inserted_channel_config

函数adc_inserted_channel_config描述见下表:

表 3-24. 函数 adc_inserted_channel_config

函数名称	adc_inserted_channel_config
函数原形	void adc_inserted_channel_config(uint32_t adc_periph, uint8_t rank, uint8_t adc_channel, uint32_t sample_time);
功能描述	配置ADC注入通道组
先决条件	-
被调用函数	-
输入参数{in}	
adc_periph	ADC外设
ADCx(x=0,1,2)	ADC外设选择
输入参数{in}	
rank	注入组通道序列，取值范围为0~3
输入参数{in}	
adc_channel	ADC通道选择
ADC_CHANNEL_x	ADC 通道x (x=0..20)
输入参数{in}	
sample_time	采样时间x, ADC0/ADC1范围 (x=0..809), ADC2范围 (x=0..638). 例如.10'dx: ADC0/1为(x+3.5)周期, ADC2为(x+2.5)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, 10);
```

函数 `adc_inserted_channel_offset_config`

函数`adc_inserted_channel_offset_config`描述见下表：

表 3-25. 函数 `adc_inserted_channel_offset_config`

函数名称	<code>adc_inserted_channel_offset_config</code>
函数原形	<code>void adc_inserted_channel_offset_config(uint32_t adc_periph, uint8_t inserted_channel, uint32_t offset);</code>
功能描述	配置ADC注入通道组数据偏移值
先决条件	-
被调用函数	-
输入参数{in}	
adc_periph	ADC外设
<code>ADCx(x=0,1,2)</code>	ADC外设选择
输入参数{in}	
inserted_channel	注入通道选择
<code>ADC_INSERTED_CHANNEL_x(x=0..3)</code>	注入通道，x=0,1,2,3
输入参数{in}	
offset	数据偏移值
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_channel_differential_mode_config`

函数`adc_channel_differential_mode_config`描述见下表：

表 3-26. 函数 `adc_channel_differential_mode_config`

函数名称	<code>adc_channel_differential_mode_config</code>
函数原形	<code>void adc_channel_differential_mode_config(uint32_t adc_periph, uint32_t adc_channel, ControlStatus newvalue);</code>
功能描述	配置ADC通道差分模式
先决条件	-
被调用函数	-
输入参数{in}	
adc_periph	ADC外设
<code>ADCx(x=0..2)</code>	ADC外设选择
输入参数{in}	
inserted_channel	通道使用差分模式
<code>ADC_DIFFERENTIAL_MODE_CHANNEL_x (x=0..21),</code> <code>ADC_DIFFERENTIAL_MODE_CHANNEL_ALL</code>	ADC通道差分模式
输入参数{in}	
newvalue	通道使能/禁能
<code>ENABLE</code>	使能
<code>DISABLE</code>	禁能
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* configure differential mode for ADC channel */
```

```
adc_channel_differential_mode_config(ADC0,  
ADC_DIFFERENTIAL_MODE_CHANNEL_ALL, ENABLE);
```

函数 `adc_external_trigger_config`

函数`adc_external_trigger_config`描述见下表：

表 3-27. 函数 `adc_external_trigger_config`

函数名称	<code>adc_external_trigger_config</code>
函数原形	<code>void adc_external_trigger_config(uint32_t adc_periph, uint8_t adc_channel_group, uint32_t trigger_mode);</code>
功能描述	配置ADC外部触发
先决条件	-
被调用函数	-

输入参数{in}	
adc_periph	ADC外设
<i>ADCx(x=0,1,2)</i>	ADC外设选择
输入参数{in}	
adc_channel_group	通道组选择
<i>ADC_REGULAR_CHANNEL</i>	规则通道组
<i>ADC_INSERTED_CHANNEL</i>	注入通道组
输入参数{in}	
trigger_mode	外部触发模式
<i>EXTERNAL_TRIGGER_DISABLE</i>	外部触发禁能
<i>EXTERNAL_TRIGGER_RISING</i>	上升沿触发
<i>EXTERNAL_TRIGGER_FALLING</i>	下降沿触发
<i>EXTERNAL_TRIGGER_RISING_FALLING</i>	双边沿触发
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* enable ADC0 inserted channel group external trigger */
```

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

函数 **adc_software_trigger_enable**

函数adc_software_trigger_enable描述见下表：

表 3-28. 函数 **adc_software_trigger_enable**

函数名称	adc_software_trigger_enable
函数原形	void adc_software_trigger_enable(uint32_t adc_periph, uint8_t adc_channel_group);
功能描述	ADC软件触发使能
先决条件	-
被调用函数	-
输入参数{in}	

adc_periph	ADC外设
ADCx(x=0,1,2)	ADC外设选择
输入参数{in}	
adc_channel_group	通道组选择
ADC_REGULAR_CHANNEL	规则通道组
ADC_INSERTED_CHANNEL	注入通道组
输出参数{out}	
-	-
返回值	
-	-

例如：

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

函数 adc_end_of_conversion_config

函数adc_end_of_conversion_config描述见下表：

表 3-29. 函数 adc_end_of_conversion_config

函数名称	adc_end_of_conversion_config
函数原形	void adc_end_of_conversion_config(uint32_t adc_periph , uint8_t end_selection);
功能描述	配置转换结束模式
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}	
end_selection	转换结束模式选择
ADC_EOC_SET_SEQUENCE	只有在规则转换序列转换结束时，才将EOC置1。如果不设置DMA=1，则溢出检测失能
ADC_EOC_SET_CONVERSION	在每个规则转换结束时，将EOC置1。溢出检测自动使能
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* at the end of each regular conversion, the EOC bit is set. */
```

```
adc_end_of_conversion_config (ADC0, ADC_EOC_SET_CONVERSION);
```

函数 `adc_regular_data_read`

函数`adc_inserted_regular_data_read`描述见下表：

表 3-30. 函数 `adc_regular_data_read`

函数名称	<code>adc_regular_data_read</code>
函数原形	<code>uint32_t adc_regular_data_read(uint32_t adc_periph);</code>
功能描述	读ADC规则组数据寄存器
先决条件	-
被调用函数	-
输入参数{in}	
<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>uint32_t</code>	ADC转换值(0~0xFFFF)

例如：

```
/* read ADC0 regular group data register */
```

```
uint32_t adc_value = 0;
```

```
adc_value = adc_regular_data_read(ADC0);
```

函数 `adc_inserted_data_read`

函数`adc_inserted_regular_data_read`描述见下表：

表 3-31. 函数 `adc_inserted_data_read`

函数名称	<code>adc_inserted_data_read</code>
函数原形	<code>uint32_t adc_inserted_data_read(uint32_t adc_periph, uint8_t inserted_channel);</code>
功能描述	读ADC注入组数据寄存器
先决条件	-
被调用函数	-
输入参数{in}	
<code>adc_periph</code>	ADC外设
<code>ADCx(x=0,1,2)</code>	ADC外设选择
输入参数{in}	

inserted_channel	注入通道选择
ADC_INSERTED_CHANNEL_x(x=0..3)	注入通道x, x=0,1,2,3
输出参数{out}	
-	-
返回值	
uint32_t	ADC转换值(0~0xFFFF)

例如:

```
/* read ADC0 inserted group data register */
uint32_t adc_value = 0;
adc_value = adc_inserted_data_read (ADC0, ADC_INSERTED_CHANNEL_0);
```

函数 adc_watchdog0_single_channel_enable

函数adc_watchdog0_single_channel_enable描述见下表:

表 3-32. 函数 adc_watchdog0_single_channel_enable

函数名称	adc_watchdog0_single_channel_enable
函数原形	void adc_watchdog0_single_channel_enable(uint32_t adc_periph, uint8_t adc_channel);
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}	
adc_periph	ADC外设
ADCx(x=0,1,2)	ADC外设选择
输入参数{in}	
adc_channel	选择ADC通道
ADC_CHANNEL_x(x=0..20)	ADC通道x(x=0..20)
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* configure ADC0 analog watchdog 0 single channel */
adc_watchdog0_single_channel_enable(ADC0, ADC_CHANNEL_1);
```


函数 adc_watchdog0_group_channel_enable

函数adc_watchdog0_group_channel_enable描述见下表:

表 3-33. 函数 adc_watchdog0_group_channel_enable

函数名称	adc_watchdog0_group_channel_enable
函数原形	void adc_watchdog0_group_channel_enable(uint32_t adc_periph, uint8_t adc_channel_group);
功能描述	配置ADC模拟看门狗0在通道组有效
先决条件	-
被调用函数	-
输入参数{in}	
adc_periph	ADC外设
ADCx(x=0,1,2)	ADC外设选择
输入参数{in}	
adc_channel_group	通道组使用模拟看门狗
ADC_REGULAR_CHANNEL	规则通道组
ADC_INSERTED_CHANNEL	注入通道组
ADC_REGULAR_INSERTED_CHANNEL	规则和注入通道组
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* configure ADC0 analog watchdog 0 group channel */
```

```
adc_watchdog0_group_channel_enable(ADC0, ADC_REGULAR_CHANNEL);
```

函数 adc_watchdog0_disable

函数adc_watchdog0_disable描述见下表:

表 3-34. 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
<i>ADCx(x=0,1,2)</i>	ADC外设选择
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* disable ADC0 analog watchdog 0 */
```

```
adc_watchdog0_disable(ADC0);
```

函数 **adc_watchdog1_channel_config**

函数adc_watchdog1_channel_config描述见下表:

表 3-35. 函数 **adc_watchdog1_channel_config**

函数名称	adc_watchdog1_channel_config
函数原形	void adc_watchdog1_channel_config(uint32_t adc_periph, uint32_t selection_channel, ControlStatus newvalue);
功能描述	配置ADC模拟看门狗1单通道有效
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}	
selection_channel	选择ADC通道
<i>ADC_AWD1_2_SELECTION_CHANNEL_x (x=0..20), ADC_AWD1_2_SELECTION_CHANNEL_ALL</i>	ADC通道模拟看门狗1/2选择(x=0..20)
输入参数{in}	
newvalue	使能/禁能控制
<i>ENABLE</i>	使能
<i>DISABLE</i>	禁能
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* configure ADC analog watchdog 1 channel */
```

```
adc_watchdog1_channel_config(ADC0, ADC_AWD1_2_SELECTION_CHANNEL_1,
ENABLE);
```

函数 `adc_watchdog2_channel_config`

函数`adc_watchdog2_channel_config`描述见下表：

表 3-36. 函数 `adc_watchdog2_channel_config`

函数名称	<code>adc_watchdog2_channel_config</code>
函数原形	<code>void adc_watchdog2_channel_config(uint32_t adc_periph, uint32_t selection_channel, ControlStatus newvalue);</code>
功能描述	配置ADC模拟看门狗2单通道有效
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}	
selection_channel	选择ADC通道
<code>ADC_AWD1_2_SELECTION_CHANNEL_x (x=0..20),</code> <code>ADC_AWD1_2_SELECTION_CHANNEL_ALL</code>	ADC通道模拟看门狗1/2选择(x=0..20)
输入参数{in}	
newvalue	使能/禁能控制
<code>ENABLE</code>	使能
<code>DISABLE</code>	禁能
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* configure ADC analog watchdog 2 channel */
```

```
adc_watchdog1_channel_config(ADC0, ADC_AWD1_2_SELECTION_CHANNEL_1,
ENABLE);
```

函数 `adc_watchdog1_disable`

函数`adc_watchdog1_disable`描述见下表：

表 3-37. 函数 `adc_watchdog1_disable`

函数名称	<code>adc_watchdog1_disable</code>
函数原形	<code>void adc_watchdog1_disable(uint32_t adc_periph);</code>
功能描述	ADC模拟看门狗1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
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* disable ADC0 analog watchdog 1 */
```

```
adc_watchdog1_disable(ADC0);
```

函数 `adc_watchdog2_disable`

函数`adc_watchdog2_disable`描述见下表：

表 3-38. 函数 `adc_watchdog2_disable`

函数名称	<code>adc_watchdog2_disable</code>
函数原形	<code>void adc_watchdog2_disable(uint32_t adc_periph);</code>
功能描述	ADC模拟看门狗2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
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* disable ADC0 analog watchdog 2 */
```

```
adc_watchdog2_disable(ADC0);
```

函数 `adc_watchdog0_threshold_config`

函数`adc_watchdog0_threshold_config`描述见下表：

表 3-39. 函数 `adc_watchdog0_threshold_config`

函数名称	<code>adc_watchdog0_threshold_config</code>
函数原形	<code>void adc_watchdog0_threshold_config(uint32_t adc_periph, uint32_t low_threshold, uint32_t high_threshold);</code>
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}	
<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>low_threshold</code>	模拟看门狗0低阈值, ADC0/ADC1范围为0..0xFFFFFF, ADC2范围为0..0xFFFF
输入参数{in}	
<code>high_threshold</code>	模拟看门狗0高阈值, ADC0/ADC1范围为0..0xFFFFFF, ADC2范围为0..0xFFFF
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* configure ADC2 analog watchdog 0 threshold */
```

```
adc_watchdog0_threshold_config(ADC2, 0x0400, 0xA00);
```

函数 `adc_watchdog1_threshold_config`

函数`adc_watchdog1_threshold_config`描述见下表:

表 3-40. 函数 `adc_watchdog1_threshold_config`

函数名称	<code>adc_watchdog1_threshold_config</code>
函数原形	<code>void adc_watchdog1_threshold_config(uint32_t adc_periph, uint32_t low_threshold, uint32_t high_threshold);</code>
功能描述	配置ADC模拟看门狗1阈值
先决条件	-
被调用函数	-
输入参数{in}	
<code>adc_periph</code>	ADC外设
<code>ADCx(x=0,1,2)</code>	ADC外设选择
输入参数{in}	
<code>low_threshold</code>	模拟看门狗1低阈值, ADC0/ADC1范围为0..0xFFFFFF, ADC2范围为0..0xFF
输入参数{in}	
<code>high_threshold</code>	模拟看门狗1高阈值, ADC0/ADC1范围为0..0xFFFFFF, ADC2范围为0..0xFF
输出参数{out}	
-	-

返回值	
-	-

例如:

```
/* configure ADC2 analog watchdog 1 threshold */
```

```
adc_watchdog1_threshold_config(ADC0, 0x40, 0xA0);
```

函数 adc_watchdog2_threshold_config

函数adc_watchdog2_threshold_config描述见下表:

表 3-41. 函数 adc_watchdog2_threshold_config

函数名称	adc_watchdog2_threshold_config
函数原形	void adc_watchdog2_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2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2低阈值, ADC0/ADC1范围为0..0xFFFFFF, ADC2范围为0..0xFF
输入参数{in}	
high_threshold	模拟看门狗2高阈值, ADC0/ADC1范围为0..0xFFFFFF, ADC2范围为0..0xFF
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* configure ADC2 analog watchdog 2 threshold */
```

```
adc_watchdog2_threshold_config(ADC0, 0x40, 0xA0);
```

函数 adc_oversample_mode_config

函数adc_oversample_mode_config描述见下表:

表 3-42. 函数 adc_oversample_mode_config

函数名称	adc_oversample_mode_config
函数原形	void adc_oversample_mode_config(uint32_t adc_periph, uint32_t mode, uint16_t shift, uint16_t ratio);
功能描述	配置ADC过采样模式

先决条件	-
被调用函数	-
输入参数{in}	
adc_periph	ADC外设
ADCx(x=0,1,2)	ADC外设选择
输入参数{in}	
mode	ADC过采样触发模式
ADC_OVERSAMPLING_ALL_CONVERT	在一个触发之后，对一个通道连续进行过采样转换
ADC_OVERSAMPLING_ONE_CONVERT	在一个触发之后，对一个通道只进行一次过采样转换
输入参数{in}	
shift	ADC过滤采样移位
ADC_OVERSAMPLING_SHIFT_NONE	不移位
ADC_OVERSAMPLING_SHIFT_1B	移1位
ADC_OVERSAMPLING_SHIFT_2B	移2位
ADC_OVERSAMPLING_SHIFT_3B	移3位
ADC_OVERSAMPLING_SHIFT_4B	移4位
ADC_OVERSAMPLING_SHIFT_5B	移5位
ADC_OVERSAMPLING_SHIFT_6B	移6位
ADC_OVERSAMPLING_SHIFT_7B	移7位
ADC_OVERSAMPLING_SHIFT_8B	移8位
ADC_OVERSAMPLING_SHIFT_9B	移9位（只限于ADC0/ADC1）
ADC_OVERSAMPLING_SHIFT_10B	移10位（只限于ADC0/ADC1）
ADC_OVERSAMPLING_SHIFT_11B	移11位（只限于ADC0/ADC1）
输入参数{in}	
ratio	ADC过采样率，ADC0/ADC1值0..1023对应于1x~1024x，ADC2值0..255对应于1x~256x

输出参数{out}	
-	-
返回值	
-	-

Example:

```
/* configure ADC1 oversample mode: 16 times sample, 4 bits shift */
adc_oversample_mode_config(ADC1, ADC_OVERSAMPLING_ALL_CONVERT,
ADC_OVERSAMPLING_SHIFT_4B, 15);
```

函数 adc_oversample_mode_enable

函数adc_oversample_mode_enable描述见下表:

表 3-43. 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
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:

```
/* 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 3-44. 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 3-45. 函数 **adc_flag_get**

函数名称	adc_flag_get
函数原形	FlagStatus adc_flag_get(uint32_t adc_periph , uint32_t flag);
功能描述	获取ADC标志位
先决条件	-
被调用函数	-
输入参数{in}	
adc_periph	ADC外设
ADCx(x=0,1,2)	ADC外设选择
输入参数{in}	
flag	ADC标志位
ADC_FLAG_WDE0	模拟看门狗0事件标志位
ADC_FLAG_EOC	组转换结束标志位
ADC_FLAG_EOIC	注入通道组转换结束标志位
ADC_FLAG_STIC	注入通道组转换开始标志位
ADC_FLAG_STRC	规则通道组转换开始标志位
ADC_FLAG_ROVF	规则数据寄存器溢出标志位
ADC_FLAG_WDE1	模拟看门狗1事件标志位
ADC_FLAG_WDE2	模拟看门狗2事件标志位
输出参数{out}	
-	-
返回值	
FlagStatus	SET或RESET

例如:

```
/* get the ADC0 analog watchdog 0 flag bits */
```

```
FlagStatus flag_value;
```

```
flag_value = adc_flag_get(ADC0, ADC_FLAG_WDE0);
```

函数 `adc_flag_clear`

函数`adc_flag_clear`描述见下表:

表 3-46. 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_WDE0</code>	模拟看门狗0事件标志位
<code>ADC_FLAG_EOC</code>	组转换结束标志位
<code>ADC_FLAG_EOIC</code>	注入通道组转换结束标志位
<code>ADC_FLAG_STIC</code>	注入通道组转换开始标志位
<code>ADC_FLAG_STRC</code>	规则通道组转换开始标志位
<code>ADC_FLAG_ROVF</code>	规则数据寄存器溢出标志位
<code>ADC_FLAG_WDE1</code>	模拟看门狗1事件标志位
<code>ADC_FLAG_WDE2</code>	模拟看门狗2事件标志位
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* clear the ADC0 analog watchdog 0 flag bits */
```

```
adc_flag_clear(ADC0, ADC_FLAG_WDE0);
```

函数 `adc_interrupt_enable`

函数`adc_interrupt_enable`描述见下表:

表 3-47. 函数 `adc_interrupt_enable`

函数名称	<code>adc_interrupt_enable</code>
函数原形	<code>void adc_interrupt_enable(uint32_t adc_periph, uint32_t interrupt);</code>
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
<i>ADCx(x=0,1,2)</i>	ADC外设选择
输入参数{in}	
interrupt	ADC中断标志位
<i>ADC_INT_WDE0</i>	模拟看门狗0中断标志位
<i>ADC_INT_EOC</i>	组转换结束中断标志位
<i>ADC_INT_EOIC</i>	注入通道组转换结束中断标志位
<i>ADC_INT_ROVF</i>	规则数据寄存器溢出中断标志位
<i>ADC_INT_WDE1</i>	模拟看门狗1中断标志位
<i>ADC_INT_WDE2</i>	模拟看门狗2中断标志位
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* enable ADC0 analog watchdog 0 interrupt */
adc_interrupt_enable(ADC0, ADC_INT_WDE0);
```

函数 adc_interrupt_disable

函数adc_interrupt_disable描述见下表：

表 3-48. 函数 adc_interrupt_disable

函数名称	adc_interrupt_disable
函数原形	void adc_interrupt_enable(uint32_t adc_periph, uint32_t interrupt);
功能描述	ADC中断禁能
先决条件	-
被调用函数	-
输入参数{in}	
adc_periph	ADC外设
<i>ADCx(x=0,1,2)</i>	ADC外设选择
输入参数{in}	
interrupt	ADC中断标志位
<i>ADC_INT_WDE0</i>	模拟看门狗0中断标志位
<i>ADC_INT_EOC</i>	组转换结束中断标志位
<i>ADC_INT_EOIC</i>	注入通道组转换结束中断标志位
<i>ADC_INT_ROVF</i>	规则数据寄存器溢出中断标志位
<i>ADC_INT_WDE1</i>	模拟看门狗1中断标志位
<i>ADC_INT_WDE2</i>	模拟看门狗2中断标志位
输出参数{out}	
-	-

返回值	
-	-

例如：

```
/* disable ADC0 analog watchdog 0 interrupt */
adc_interrupt_disable(ADC0, ADC_INT_WDE0);
```

函数 adc_interrupt_flag_get

函数adc_interrupt_flag_get描述见下表：

表 3-49. 函数 adc_interrupt_flag_get

函数名称	adc_interrupt_flag_get
函数原形	FlagStatus adc_interrupt_flag_get(uint32_t adc_periph, uint32_t int_flag);
功能描述	获取ADC中断标志位
先决条件	-
被调用函数	-
输入参数{in}	
adc_periph	ADC外设
ADCx(x=0,1,2)	ADC外设选择
输入参数{in}	
int_flag	ADC中断标志位
ADC_INT_FLAG_WDE0	模拟看门狗0中断标志位
ADC_INT_FLAG_EOC	组转换结束中断标志位
ADC_INT_FLAG_EOIC	注入通道组转换结束中断标志位
ADC_INT_FLAG_ROVF	规则数据寄存器溢出中断标志位
ADC_INT_FLAG_WDE1	模拟看门狗1中断标志位
ADC_INT_FLAG_WDE2	模拟看门狗2中断标志位
输出参数{out}	
-	-
返回值	
FlagStatus	SET或RESET

例如：

```
/* get the ADC0 analog watchdog 0 interrupt bits */
FlagStatus flag_value;
```

```
flag_value = adc_interrupt_flag_get(ADC0, ADC_INT_WDE0);
```

函数 `adc_interrupt_flag_clear`

函数`adc_interrupt_flag_clear`描述见下表：

表 3-50. 函数 `adc_interrupt_flag_clear`

函数名称	<code>adc_interrupt_flag_clear</code>
函数原形	<code>void adc_interrupt_flag_clear(uint32_t adc_periph, uint32_t int_flag);</code>
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}	
<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>int_flag</code>	ADC中断标志位
<code>ADC_INT_FLAG_WDE0</code>	模拟看门狗0中断标志位
<code>ADC_INT_FLAG_EOC</code>	组转换结束中断标志位
<code>ADC_INT_FLAG_EOIC</code>	注入通道组转换结束中断标志位
<code>ADC_INT_FLAG_ROVF</code>	规则数据寄存器溢出中断标志位
<code>ADC_INT_FLAG_WDE1</code>	模拟看门狗1中断标志位
<code>ADC_INT_FLAG_WDE2</code>	模拟看门狗2中断标志位
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* clear the ADC0 analog watchdog 0 interrupt bits */
adc_interrupt_flag_clear(ADC0, ADC_INT_WDE0);
```

函数 `adc_sync_mode_config`

函数`adc_sync_mode_config`描述见下表：

表 3-51. 函数 `adc_sync_mode_config`

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

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}	
adc_periph	ADC外设
ADCx(x=0..2)	ADC外设选择
输入参数{in}	
sync_mode	同步模式
ADC_SYNC_MODE_INDEPENDENT	ADC同步模式失能，所有的ADC都独立工作
ADC_DUAL_REGULAR_PARALLEL_INSERTED_PARALLEL	ADC0和ADC1工作在规则并行和注入并行的组合模式。
ADC_DUAL_REGULAR_PARALLEL_INSERTED_ROTATION	ADC0和ADC1工作在规则并行和交替触发的组合模式。
ADC_DUAL_INSERTED_PARALLEL	ADC0和ADC1工作在注入并行模式。
ADC_DUAL_REGULAR_PARALLEL	ADC0和ADC1工作在规则并行模式。
ADC_DUAL_REGULAR_FOLLOW_UP	ADC0和ADC1工作在跟随模式。
ADC_DUAL_INSERTED_TRIGGER_ROTATION	ADC0和ADC1工作在交替触发模式。
输出参数{out}	
-	-
返回值	
-	-

例如：

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

函数 adc_sync_delay_config

函数adc_sync_delay_config描述见下表：

表 3-52. 函数 adc_sync_delay_config

函数名称	adc_sync_delay_config
------	-----------------------

函数原形	void adc_sync_delay_config(uint32_t sample_delay);
功能描述	配置ADC同步模式两次采样之间的延时
先决条件	-
被调用函数	-
输入参数{in}	
sample_delay	采样阶段之间的延迟
ADC_SYNC_DELAY_5CYCLE (x=5..20)	配置两个采样阶段之间的延迟为x个时钟周期
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* configure the delay between 2 sampling phases in ADC sync modes */
```

```
adc_sync_delay_config (ADC_SYNC_DELAY_5CYCLE);
```

函数 adc_sync_dma_config

函数adc_sync_dma_config描述见下表：

表 3-53. 函数 adc_sync_dma_config

函数名称	adc_sync_dma_config
函数原形	void adc_sync_dma_config(uint32_t dma_mode);
功能描述	ADC同步DMA模式选择
先决条件	-
被调用函数	-
输入参数{in}	
dma_mode	DMA模式
ADC_SYNC_DMA_DISABLE	ADC同步DMA失能
ADC_SYNC_DMA_MODE0	ADC同步DMA模式0
ADC_SYNC_DMA_MODE1	ADC同步DMA模式1
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* configure ADC sync DMA mode selection */
```

adc_sync_dma_config (ADC_SYNC_DMA_MODE0);

函数 adc_sync_dma_request_after_last_enable

函数adc_sync_dma_request_after_last_enable描述见下表:

表 3-54. 函数 adc_sync_dma_request_after_last_enable

函数名称	adc_sync_dma_request_after_last_enable
函数原形	void adc_sync_dma_request_after_last_enable(void);
功能描述	当SYNCDMA不为00时，根据SYNCDMA位来产生DMA请求
先决条件	-
被调用函数	-
输入参数{in}	
-	-
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* when SYNCDMA is not equal to 2'b00, the DMA engine issues requests according to the
SYNCDMA bits */
```

```
adc_sync_dma_request_after_last_enable();
```

函数 adc_sync_dma_request_after_last_disable

函数adc_sync_dma_request_after_last_disable描述见下表:

表 3-55. 函数 adc_sync_dma_request_after_last_disable

函数名称	adc_sync_dma_request_after_last_disable
函数原形	void adc_sync_dma_request_after_last_disable(void);
功能描述	当检测到来自DMA控制器的传输结束信号后，DMA机制失能
先决条件	-
被调用函数	-
输入参数{in}	
-	-
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* the DMA engine is disabled after the end of transfer signal from DMA controller is detected
*/
```



```
adc_sync_dma_request_after_last_disable();
```

函数 `adc_sync_master_adc_regular_data0_read`

函数`adc_sync_master_adc_regular_data0_read`描述见下表：

表 3-56. 函数 `adc_sync_master_adc_regular_data0_read`

函数名称	<code>adc_sync_master_adc_regular_data0_read</code>
函数原形	<code>uint16_t adc_sync_master_adc_regular_data0_read(void);</code>
功能描述	ADC同步模式主ADC规则数据寄存器0读取
先决条件	-
被调用函数	-
输入参数{in}	
-	-
输出参数{out}	
-	-
返回值	
<code>uint16_t</code>	返回同步模式的ADC规则通道数据寄存器0的值

例如：

```
/* read ADC sync master adc regular data register 0*/
```

```
adc_sync_master_adc_regular_data0_read ();
```

函数 `adc_sync_slave_adc_regular_data0_read`

函数`adc_sync_slave_adc_regular_data0_read`描述见下表：

表 3-57. 函数 `adc_sync_slave_adc_regular_data0_read`

函数名称	<code>adc_sync_slave_adc_regular_data0_read</code>
函数原形	<code>uint16_t adc_sync_slave_adc_regular_data0_read(void);</code>
功能描述	ADC同步模式从ADC规则数据寄存器0读取
先决条件	-
被调用函数	-
输入参数{in}	
-	-
输出参数{out}	
-	-
返回值	
<code>uint16_t</code>	返回同步模式的ADC规则通道数据寄存器0的值

例如：

```
/* read ADC sync slave adc regular data register 0*/
```

```
adc_sync_slave_adc_regular_data0_read ();
```

函数 `adc_sync_regular_data1_read`

函数`adc_sync_regular_data1_read`描述见下表:

表 3-58. 函数 `adc_sync_regular_data1_read`

函数名称	<code>adc_sync_regular_data1_read</code>
函数原形	<code>uint32_t adc_sync_regular_data1_read(void);</code>
功能描述	ADC同步模式规则数据寄存器1读取
先决条件	-
被调用函数	-
输入参数{in}	
-	-
输出参数{out}	
-	-
返回值	
<code>uint32_t</code>	返回同步模式的ADC规则通道数据寄存器1的值

例如:

```
/* read ADC sync regular data register 1*/
```

```
adc_sync_regular_data1_read();
```

3.3. 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和设备之间相互通信的总线标准。CAN网络接口支持CAN总线协议2.0A/B、ISO11898-1:2015规范和BOSCH CAN-FD规范。章节[3.3.1](#)描述了CAN的寄存器列表，章节[3.3.2](#)对CAN库函数进行说明。

3.3.1. 外设寄存器说明

CAN寄存器列表如下表所示:

表 3-59. CAN 寄存器

寄存器名称	寄存器描述
<code>CAN_CTL0</code>	控制寄存器 0
<code>CAN_CTL1</code>	控制寄存器 1
<code>CAN_TIMER</code>	计数器寄存器
<code>CAN_RMPUBF</code>	接收邮箱公有过滤寄存器
<code>CAN_ERR0</code>	错误寄存器 0
<code>CAN_ERR1</code>	错误寄存器 1
<code>CAN_INTEN</code>	中断使能寄存器
<code>CAN_STAT</code>	状态寄存器
<code>CAN_CTL2</code>	控制寄存器 2
<code>CAN_CRCC</code>	常规帧 CRC 寄存器

寄存器名称	寄存器描述
CAN_RFIFOPUBF	接收 FIFO 共有过滤寄存器
CAN_RFIFOIFMN	接收 FIFO 标识符过滤元素匹配序号寄存器
CAN_BT	位时间寄存器
CAN_RFIFOMPFx (x = 0..31)	接收 FIFO/邮箱私有过滤 x 寄存器
CAN_PN_CTL0	虚拟联网模式控制寄存器 0
CAN_PN_TO	虚拟联网模式超时寄存器
CAN_PN_STAT	虚拟联网模式状态寄存器
CAN_PN_EID0	虚拟联网模式期望标识符 0 寄存器
CAN_PN_EDLC	虚拟联网模式期望 DLC 寄存器
CAN_PN_EDL0	虚拟联网模式期望数据低字 0 寄存器
CAN_PN_EDL1	虚拟联网模式期望数据低字 1 寄存器
CAN_PN_IFEID1	虚拟联网模式标识符过滤器 / 期望标识符 1 寄存器
CAN_PN_DF0EDH 0	虚拟联网模式数据 0 过滤器 / 期望数据高字 0 寄存器
CAN_PN_DF1EDH 1	虚拟联网模式数据 1 过滤器 / 期望数据高字 1 寄存器
CAN_PN_RWMxCS (x = 0..3)	虚拟联网模式接收唤醒邮箱 x 控制状态信息寄存器
CAN_PN_RWMxI (x = 0..3)	虚拟联网模式接收唤醒邮箱 x 标识符寄存器
CAN_PN_RWMxD0 (x = 0..3)	虚拟联网模式接收唤醒邮箱 x 数据 0 寄存器
CAN_PN_RWMxD1 (x = 0..3)	虚拟联网模式接收唤醒邮箱 x 数据 1 寄存器
CAN_FDCTL	FD 控制寄存器
CAN_FDBT	FD 位时间寄存器
CAN_CRCCFD	常规帧和 FD 帧 CRC 寄存器

3.3.2. 外设库函数说明

CAN库函数列表如下表所示：

表 3-60. CAN 库函数

库函数名称	库函数描述
can_deinit	复位 CAN
can_software_reset	复位 CAN 内部状态机和 CAN 寄存器
can_init	CAN 模块初始化
can_struct_para_init	将 CAN 结构体初始化为默认值
can_private_filter_config	配置接收 FIFO/邮箱私有过滤器
can_operation_mode_enter	进入对应的模式
can_operation_mode_get	获取操作模式

库函数名称	库函数描述
can_inactive_mode_exit	退出暂停模式
can_pn_mode_exit	退出虚拟联网模式
can_fd_config	CAN FD 初始化
can_bitrate_switch_enable	使能波特率切换功能
can_bitrate_switch_disable	禁能波特率切换功能
can_tdc_get	获取传输延迟补偿值
can_tdc_enable	使能传输延迟补偿
can_tdc_disable	禁能传输延迟补偿
can_rx_fifo_config	配置接收 FIFO
can_rx_fifo_filter_table_config	配置接收 FIFO 标识符过滤器表
can_rx_fifo_read	读取接收 FIFO 数据
can_rx_fifo_filter_matching_number_get	获取接收 FIFO 标识符过滤元素匹配序号
can_rx_fifo_clear	清接收 FIFO
can_ram_address_get	获取邮箱 RAM 地址
can_mailbox_config	配置邮箱
can_mailbox_transmit_abort	中止邮箱发送
can_mailbox_transmit_inactive	失活发送邮箱
can_mailbox_receive_data_read	读取接收邮箱数据
can_mailbox_receive_lock	锁定接收邮箱
can_mailbox_receive_unlock	解锁接收邮箱
can_mailbox_receive_inactive	失活接收邮箱
can_mailbox_code_get	获取邮箱 CODE 值
can_error_counter_config	配置错误计数
can_error_counter_get	获取错误计数
can_error_state_get	获取错误状态指示
can_crc_get	获取邮箱 CRC 值
can_pn_mode_config	配置虚拟联网模式参数
can_pn_mode_filter_config	配置虚拟联网模式过滤器参数
can_pn_mode_num_of_match_get	获取虚拟联网模式下匹配的消息计数
can_pn_mode_data_read	获取匹配的消息
can_self_reception_enable	使能自接收
can_self_reception_disable	禁能自接收
can_transmit_abort_enable	使能发送中止功能
can_transmit_abort_disable	禁能发送中止功能
can_auto_busoff_recovery_enable	使能离线自动恢复模式
can_auto_busoff_recovery_disable	禁能离线自动恢复模式
can_time_sync_enable	使能时间同步模式
can_time_sync_disable	禁能时间同步模式
can_edge_filter_mode_enable	使能边沿过滤模式
can_edge_filter_mode_disable	禁能边沿过滤模式

库函数名称	库函数描述
can_ped_mode_enable	使能协议异常检测模式
can_ped_mode_disable	禁能协议异常检测模式
can_arbitration_delay_bits_config	配置仲裁启动延迟位
can_bsp_mode_config	配置位采样模式
can_flag_get	获取 CAN 标志状态
can_flag_clear	清除 CAN 标志状态
can_interrupt_enable	使能 CAN 中断
can_interrupt_disable	禁能 CAN 中断
can_interrupt_flag_get	获取 CAN 中断标志状态
can_interrupt_flag_clear	清除 CAN 中断标志状态

结构体 can_error_counter_struct

表 3-61. 结构体 can_error_counter_struct

成员名称	功能描述
fd_data_phase_rx_errcnt	FD 帧 BRS 位为隐性位时数据阶段的接收错误计数器
fd_data_phase_tx_errcnt	FD 帧 BRS 位为隐性位时数据阶段的发送错误计数器
rx_errcnt	CAN 协议定义的接收错误计数器
tx_errcnt	CAN 协议定义的发送错误计数器

结构体 can_parameter_struct

表 3-62. 结构体 can_parameter_struct

成员名称	功能描述
internal_counter_source	内部计数器时钟源
mb_tx_order	邮箱发送顺序
mb_rx_ide_rtr_type	邮箱接收时 IDE 和 RTR 域的过滤类型
mb_remote_frame	远程请求帧存储
self_reception	使能或禁能自接收
mb_tx_abort_enable	使能或禁能发送中止
local_priority_enable	使能或禁能本地优先级
rx_private_filter_queue_enable	使能接收私有过滤使能&接收邮箱队列
edge_filter_enable	使能边沿过滤
protocol_exception_enable	使能协议异常检测
rx_filter_order	接收过滤顺序
memory_size	内存大小
mb_public_filter	邮箱公有过滤器

成员名称	功能描述
prescaler	波特率分频系数
resync_jump_width	再同步补偿宽度
prop_time_segment	传播时间段
time_segment_1	相位缓冲段 1
time_segment_2	相位缓冲段 2

结构体 can_mailbox_descriptor_struct

表 3-63. 结构体 can_mailbox_descriptor_struct

成员名称	功能描述
timestamp	来自内部计数器值产生的时间戳
dlc	数据字节长度代码
rtr	远程传输请求
ide	标识符扩展位
srr	替代远程请求
code	邮箱代码
esi	错误状态指示
brs	位速率切换
fdf	FD 格式指示
id	标识符
prio	本地优先级
data[64]	数据
data_bytes	数据字节
padding	FD 模式填充值

结构体 can_rx_fifo_struct

表 3-64. 结构体 can_rx_fifo_struct

成员名称	功能描述
timestamp	来自内部计数器值产生的时间戳
dlc	数据字节长度代码
rtr	远程传输请求
ide	标识符扩展位
srr	替代远程请求
idhit	标识符过滤元素匹配序号
id	标识符
data[2]	FIFO 数据

结构体 can_fd_parameter_struct

表 3-65. 结构体 can_fd_parameter_struct

成员名称	功能描述
------	------

iso_can_fd_enable	使能 ISO CAN FD 协议
bitrate_switch_enable	使能数据阶段波特率切换
mailbox_data_size	邮箱数据大小
tdc_enable	传输延迟补偿使能
tdc_offset	传输延迟补偿偏置
prescaler	波特率分频系数
resync_jump_width	再同步补偿宽度
prop_time_segment	传播时间段
time_segment_1	相位缓冲段 1
time_segment_2	相位缓冲段 2

结构体 can_rx_fifo_id_filter_struct

表 3-66. 结构体 can_rx_fifo_id_filter_struct

成员名称	功能描述
remote_frame	期望是否接收匹配的远程帧到 FIFO
extended_frame	期望是否接收匹配的扩展帧到 FIFO
id	期望的标识符

结构体 can_fifo_parameter_struct

表 3-67. 结构体 can_fifo_parameter_struct

成员名称	功能描述
dma_enable	使能 DMA
filter_format_and_number	FIFO 标识符过滤元素格式和数量
fifo_public_filter	FIFO 公有过滤器

结构体 can_pn_mode_filter_struct

表 3-68. 结构体 can_pn_mode_filter_struct

成员名称	功能描述
remote_frame	期望 RTR
extended_frame	期望 IDE
id	期望 ID
dlc_high_threshold	期望 DLC 上限值
dlc_low_threshold	期望 DLC 下限值
payload[2]	期望数据

结构体 `can_pn_mode_config_struct`

表 3-69. 结构体 `can_pn_mode_config_struct`

成员名称	功能描述
<code>timeout_int</code>	使能或禁能超时唤醒中断
<code>match_int</code>	使能或禁能匹配唤醒中断
<code>num_matches</code>	设置消息匹配次数
<code>match_timeout</code>	设置超时唤醒时间值
<code>frame_filter</code>	设置帧的过滤类型
<code>id_filter</code>	设置 ID 域的过滤类型
<code>data_filter</code>	设置 DATA 域的过滤类型

结构体 `can_crc_struct`

表 3-70. 结构体 `can_crc_struct`

成员名称	功能描述
<code>classical_frm_mb_number</code>	发送了 CRC 值为 CRCTC[14:0]的邮箱的编号
<code>classical_frm_transmitted_crc</code>	最新发送的常规帧的 CRC 计算值
<code>classical_fd_frm_mb_number</code>	发送常规帧或者 FD 帧时, CRC 值为 CRCTC[20:0]的邮箱的编号
<code>classical_fd_frm_transmitted_crc</code>	发送的常规帧 / FD 帧的 CRC 计算值

枚举类型 `can_interrupt_enum`

表 3-71. 枚举类型 `can_interrupt_enum`

成员名称	功能描述
<code>CAN_INT_RX_WARNING</code>	Rx 错误警告中断
<code>CAN_INT_TX_WARNING</code>	Tx 错误警告中断
<code>CAN_INT_ERR_SUMMARY</code>	错误汇总中断
<code>CAN_INT_BUSOFF</code>	离线中断
<code>CAN_INT_BUSOFF_RECOVERY</code>	离线恢复中断
<code>CAN_INT_ERR_SUMMARY_FD</code>	FD 帧数据位时间的错误汇总中断
<code>CAN_INT_MB0</code>	邮箱 0 成功发送或接收帧中断
<code>CAN_INT_MB1</code>	邮箱 1 成功发送或接收帧中断
<code>CAN_INT_MB2</code>	邮箱 2 成功发送或接收帧中断

成员名称	功能描述
CAN_INT_MB3	邮箱 3 成功发送或接收帧中断
CAN_INT_MB4	邮箱 4 成功发送或接收帧中断
CAN_INT_MB5	邮箱 5 成功发送或接收帧中断
CAN_INT_MB6	邮箱 6 成功发送或接收帧中断
CAN_INT_MB7	邮箱 7 成功发送或接收帧中断
CAN_INT_MB8	邮箱 8 成功发送或接收帧中断
CAN_INT_MB9	邮箱 9 成功发送或接收帧中断
CAN_INT_MB10	邮箱 10 成功发送或接收帧中断
CAN_INT_MB11	邮箱 11 成功发送或接收帧中断
CAN_INT_MB12	邮箱 12 成功发送或接收帧中断
CAN_INT_MB13	邮箱 13 成功发送或接收帧中断
CAN_INT_MB14	邮箱 14 成功发送或接收帧中断
CAN_INT_MB15	邮箱 15 成功发送或接收帧中断
CAN_INT_MB16	邮箱 16 成功发送或接收帧中断
CAN_INT_MB17	邮箱 17 成功发送或接收帧中断
CAN_INT_MB18	邮箱 18 成功发送或接收帧中断
CAN_INT_MB19	邮箱 19 成功发送或接收帧中断
CAN_INT_MB20	邮箱 20 成功发送或接收帧中断
CAN_INT_MB21	邮箱 21 成功发送或接收帧中断
CAN_INT_MB22	邮箱 22 成功发送或接收帧中断
CAN_INT_MB23	邮箱 23 成功发送或接收帧中断
CAN_INT_MB24	邮箱 24 成功发送或接收帧中断
CAN_INT_MB25	邮箱 25 成功发送或接收帧中断
CAN_INT_MB26	邮箱 26 成功发送或接收帧中断
CAN_INT_MB27	邮箱 27 成功发送或接收帧中断
CAN_INT_MB28	邮箱 28 成功发送或接收帧中断
CAN_INT_MB29	邮箱 29 成功发送或接收帧中断
CAN_INT_MB30	邮箱 30 成功发送或接收帧中断
CAN_INT_MB31	邮箱 31 成功发送或接收帧中断
CAN_INT_FIFO_AVAILABLE	Rx FIFO 非空中断
CAN_INT_FIFO_WARNING	Rx FIFO 警告中断
CAN_INT_FIFO_OVERFLOW	Rx FIFO 溢出中断
CAN_INT_WAKEUP_MATCH	PN 模式匹配唤醒中断
CAN_INT_WAKEUP_TIMEOUT	PN 模式超时唤醒中断

枚举类型 `can_flag_enum`

表 3-72. 枚举类型 `can_flag_enum`

成员名称	功能描述
<code>CAN_FLAG_CAN_PN</code>	虚拟联网状态
<code>CAN_FLAG_SOFT_RST</code>	软件复位状态
<code>CAN_FLAG_ERR_SUMMARY</code>	错误汇总
<code>CAN_FLAG_BUSOFF</code>	离线状态
<code>CAN_FLAG_RECEIVING</code>	接收状态
<code>CAN_FLAG_TRANSMITTING</code>	发送状态
<code>CAN_FLAG_IDLE</code>	空闲标志
<code>CAN_FLAG_RX_WARNING</code>	接收错误警告标志
<code>CAN_FLAG_TX_WARNING</code>	发送错误警告标志
<code>CAN_FLAG_STUFF_ERR</code>	填充错误状态
<code>CAN_FLAG_FORMAT_ERR</code>	格式错误状态
<code>CAN_FLAG_CRC_ERR</code>	CRC 错误状态
<code>CAN_FLAG_ACK_ERROR</code>	ACK 错误状态
<code>CAN_FLAG_BIT_DOMINANT_ERR</code>	位显性错误状态
<code>CAN_FLAG_BIT_RECESSIVE_ERR</code>	位隐性错误状态
<code>CAN_FLAG_SYNC_ERR</code>	同步标志
<code>CAN_FLAG_BUSOFF_RECOVERY</code>	离线恢复标志
<code>CAN_FLAG_ERR_SUMMARY_FD</code>	FD 帧 BRS 位为隐性位时数据阶段的错误汇总标志
<code>CAN_FLAG_ERR_OVERRUN</code>	错误溢出状态
<code>CAN_FLAG_STUFF_ERR_FD</code>	D 帧 BRS 位为隐性位时数据阶段的填充错误状态

成员名称	功能描述
CAN_FLAG_FORM_ERR_FD	FD 帧 BRS 位为隐性位时数据阶段的格式错误状态
CAN_FLAG_CRC_ERR_FD	FD 帧 BRS 位为隐性位时数据阶段的 CRC 错误状态
CAN_FLAG_BIT_DOMINANT_ERR_FD	FD 帧 BRS 位为隐性位时数据阶段的位显性错误状态
CAN_FLAG_BIT_RECESSIVE_ERR_FD	FD 帧 BRS 位为隐性位时数据阶段的位隐性错误状态
CAN_FLAG_MB0	邮箱 0 成功发送或接收帧标志
CAN_FLAG_MB1	邮箱 1 成功发送或接收帧标志
CAN_FLAG_MB2	邮箱 2 成功发送或接收帧标志
CAN_FLAG_MB3	邮箱 3 成功发送或接收帧标志
CAN_FLAG_MB4	邮箱 4 成功发送或接收帧标志
CAN_FLAG_MB5	邮箱 5 成功发送或接收帧标志
CAN_FLAG_MB6	邮箱 6 成功发送或接收帧标志
CAN_FLAG_MB7	邮箱 7 成功发送或接收帧标志
CAN_FLAG_MB8	邮箱 8 成功发送或接收帧标志
CAN_FLAG_MB9	邮箱 9 成功发送或接收帧标志
CAN_FLAG_MB10	邮箱 10 成功发送或接收帧标志
CAN_FLAG_MB11	邮箱 11 成功发送或接收帧标志
CAN_FLAG_MB12	邮箱 12 成功发送或接收帧标志
CAN_FLAG_MB13	邮箱 13 成功发送或接收帧标志
CAN_FLAG_MB14	邮箱 14 成功发送或接收帧标志
CAN_FLAG_MB15	邮箱 15 成功发送或接收帧标志
CAN_FLAG_MB16	邮箱 16 成功发送或接收帧标志
CAN_FLAG_MB17	邮箱 17 成功发送或接收帧标志
CAN_FLAG_MB18	邮箱 18 成功发送或接收帧标志
CAN_FLAG_MB19	邮箱 19 成功发送或接收帧标志
CAN_FLAG_MB20	邮箱 20 成功发送或接收帧标志
CAN_FLAG_MB21	邮箱 21 成功发送或接收帧标志
CAN_FLAG_MB22	邮箱 22 成功发送或接收帧标志
CAN_FLAG_MB23	邮箱 23 成功发送或接收帧标志
CAN_FLAG_MB24	邮箱 24 成功发送或接收帧标志
CAN_FLAG_MB25	邮箱 25 成功发送或接收帧标志
CAN_FLAG_MB26	邮箱 26 成功发送或接收帧标志
CAN_FLAG_MB27	邮箱 27 成功发送或接收帧标志
CAN_FLAG_MB28	邮箱 28 成功发送或接收帧标志
CAN_FLAG_MB29	邮箱 29 成功发送或接收帧标志
CAN_FLAG_MB30	邮箱 30 成功发送或接收帧标志
CAN_FLAG_MB31	邮箱 31 成功发送或接收帧标志

成员名称	功能描述
CAN_FLAG_FIFO_AVAILABLE	Rx FIFO 非空标志
CAN_FLAG_FIFO_WARNING	Rx FIFO 警告标志
CAN_FLAG_FIFO_OVERFLOW	Rx FIFO 溢出标志
CAN_FLAG_WAKEUP_MATCH	PN 模式匹配唤醒标志
CAN_FLAG_WAKEUP_TIMEOUT	PN 模式超时唤醒标志
CAN_FLAG_TDC_OUT_OF_RANGE	传输延迟超出补偿范围标志

枚举类型 `can_interrupt_flag_enum`

表 3-73. 枚举类型 `can_interrupt_flag_enum`

成员名称	功能描述
CAN_INT_FLAG_ERR_SUMMARY	错误汇总中断标志
CAN_INT_FLAG_BUSOFF	离线中断标志
CAN_INT_FLAG_RX_WARNING	Rx 错误警告中断标志
CAN_INT_FLAG_TX_WARNING	Tx 错误警告中断标志
CAN_INT_FLAG_BUSOFF_RECOVERY	离线恢复中断标志
CAN_INT_FLAG_ERR_SUMMARY_FD	FD 帧数据位时间的错误汇总中断标志
CAN_INT_FLAG_MB0	邮箱 0 成功发送或接收帧中断标志
CAN_INT_FLAG_MB1	邮箱 1 成功发送或接收帧中断标志
CAN_INT_FLAG_MB2	邮箱 2 成功发送或接收帧中断标志
CAN_INT_FLAG_MB3	邮箱 3 成功发送或接收帧中断标志
CAN_INT_FLAG_MB4	邮箱 4 成功发送或接收帧中断标志
CAN_INT_FLAG_MB5	邮箱 5 成功发送或接收帧中断标志

成员名称	功能描述
CAN_INT_FLAG_M B6	邮箱 6 成功发送或接收帧中断标志
CAN_INT_FLAG_M B7	邮箱 7 成功发送或接收帧中断标志
CAN_INT_FLAG_M B8	邮箱 8 成功发送或接收帧中断标志
CAN_INT_FLAG_M B9	邮箱 9 成功发送或接收帧中断标志
CAN_INT_FLAG_M B10	邮箱 10 成功发送或接收帧中断标志
CAN_INT_FLAG_M B11	邮箱 11 成功发送或接收帧中断标志
CAN_INT_FLAG_M B12	邮箱 12 成功发送或接收帧中断标志
CAN_INT_FLAG_M B13	邮箱 13 成功发送或接收帧中断标志
CAN_INT_FLAG_M B14	邮箱 14 成功发送或接收帧中断标志
CAN_INT_FLAG_M B15	邮箱 15 成功发送或接收帧中断标志
CAN_INT_FLAG_M B16	邮箱 16 成功发送或接收帧中断标志
CAN_INT_FLAG_M B17	邮箱 17 成功发送或接收帧中断标志
CAN_INT_FLAG_M B18	邮箱 18 成功发送或接收帧中断标志
CAN_INT_FLAG_M B19	邮箱 19 成功发送或接收帧中断标志
CAN_INT_FLAG_M B20	邮箱 20 成功发送或接收帧中断标志
CAN_INT_FLAG_M B21	邮箱 21 成功发送或接收帧中断标志
CAN_INT_FLAG_M B22	邮箱 22 成功发送或接收帧中断标志
CAN_INT_FLAG_M B23	邮箱 23 成功发送或接收帧中断标志
CAN_INT_FLAG_M B24	邮箱 24 成功发送或接收帧中断标志
CAN_INT_FLAG_M B25	邮箱 25 成功发送或接收帧中断标志
CAN_INT_FLAG_M B26	邮箱 26 成功发送或接收帧中断标志

成员名称	功能描述
CAN_INT_FLAG_M B27	邮箱 27 成功发送或接收帧中断标志
CAN_INT_FLAG_M B28	邮箱 28 成功发送或接收帧中断标志
CAN_INT_FLAG_M B29	邮箱 29 成功发送或接收帧中断标志
CAN_INT_FLAG_M B30	邮箱 30 成功发送或接收帧中断标志
CAN_INT_FLAG_M B31	邮箱 31 成功发送或接收帧中断标志
CAN_INT_FLAG_FI FO_AVAILABLE	Rx FIFO 非空中断标志
CAN_INT_FLAG_FI FO_WARNING	Rx FIFO 警告中断标志
CAN_INT_FLAG_FI FO_OVERFLOW	Rx FIFO 溢出中断标志
CAN_INT_FLAG_W AKEUP_MATCH	PN 模式匹配唤醒中断标志
CAN_INT_FLAG_W AKEUP_TIMEOUT	PN 模式超时唤醒中断标志

枚举类型 `can_operation_modes_enum`

表 3-74. 枚举类型 `can_operation_modes_enum`

成员名称	功能描述
CAN_NORMAL_MODE	正常模式
CAN_MONITOR_MODE	监听模式
CAN_LOOPBACK_SILENT_MODE	回环静默模式
CAN_INACTIVE_MODE	暂停模式
CAN_DISABLE_MODE	禁能模式
CAN_PN_MODE	虚拟联网模式

枚举类型 `can_struct_type_enum`

表 3-75. 枚举类型 `can_struct_type_enum`

成员名称	功能描述
CAN_INIT_STRUCT	CAN 初始化参数结构体

CAN_FD_INIT_STRUCTURE	CAN FD 参数结构体
CAN_FIFO_INIT_STRUCTURE	CAN FIFO 参数结构体
CAN_PN_MODE_INIT_STRUCTURE	虚拟联网模式参数结构体
CAN_PN_MODE_FILTER_STRUCTURE	虚拟联网模式过滤器参数结构体

枚举类型 `can_error_state_enum`

表 3-76. 枚举类型 `can_error_state_enum`

成员名称	功能描述
CAN_ERROR_STATE_ACTIVE	CAN 处于主动错误状态
CAN_ERROR_STATE_PASSIVE	CAN 处于被动错误状态
CAN_ERROR_STATE_BUS_OFF	CAN 处于离线状态

函数 `can_deinit`

函数 `can_deinit` 描述见下表：

表 3-77. 函数 `can_deinit`

函数名称	<code>can_deinit</code>
函数原型	<code>void can_deinit(uint32_t can_periph);</code>
功能描述	复位 CAN
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>can_periph</code>	CAN 外设
<code>CANx(x=0,1,2)</code>	CAN 外设选择
输出参数{out}	
-	-
返回值	
-	-

例如：

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

函数 `can_software_reset`

函数 `can_software_reset` 描述见下表：

表 3-78. 函数 `can_software_reset`

函数名称	<code>can_software_reset</code>
函数原型	<code>ErrStatus can_software_reset(uint32_t can_periph);</code>
功能描述	复位 CAN 内部状态机和 CAN 寄存器
先决条件	-
被调用函数	-
输入参数{in}	
<code>can_periph</code>	CAN 外设
<code>CANx(x=0,1,2)</code>	CAN 外设选择
输出参数{out}	
-	-
返回值	
ErrStatus	ERROR 或 SUCCESS

例如：

```
ErrStatus err;
```

```
/* reset CAN0 */
```

```
err = can_software_reset(CAN0);
```

函数 `can_init`

函数 `can_init` 描述见下表：

表 3-79. 函数 `can_init`

函数名称	<code>can_init</code>
函数原型	<code>ErrStatus can_init(uint32_t can_periph, can_parameter_struct* can_parameter_init);</code>
功能描述	CAN 模块初始化
先决条件	-
被调用函数	-
输入参数{in}	
<code>can_periph</code>	CAN 外设
<code>CANx(x=0,1,2)</code>	CAN 外设选择
输入参数{in}	
<code>can_parameter_init</code>	参见 表 3-62. 结构体 <code>can_parameter_struct</code>
输出参数{out}	
-	-
返回值	
ErrStatus	ERROR 或 SUCCESS

例如:

```
can_parameter_struct can_parameter;

ErrStatus err;

.....

/* initialize CAN */

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

函数 can_struct_para_init

函数can_struct_para_init描述见下表:

表 3-80. 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	将 CAN 结构体初始化为默认值
先决条件	-
被调用函数	-
输入参数{in}	
type	参见枚举类型 表 3-75. 枚举类型 can_struct_type_enum
输入参数{in}	
p_struct	指向特定的结构体
输出参数{out}	
-	-
返回值	
-	-

例如:

```
can_parameter_struct can_parameter;

/* initialize CAN */

can_struct_para_init(CAN_INIT_STRUCT, &can_parameter);
```

函数 can_private_filter_config

函数can_private_filter_config描述见下表:

表 3-81. 函数 can_private_filter_config

函数名称	can_private_filter_config
函数原型	void can_private_filter_config(uint32_t can_periph, uint32_t index, uint32_t filter_data);
功能描述	配置接收 FIFO/邮箱私有过滤器
先决条件	-

被调用函数	-
输入参数{in}	
can_periph	CAN 外设
CANx(x=0,1,2)	CAN 外设选择
输入参数{in}	
index	邮箱索引
0-31	邮箱索引选择
输入参数{in}	
filter_data	配置的过滤器数据
0..0xFFFFFFFF	过滤器数据
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* configure CAN0 mailbox 0 private filter */
can_private_filter_config(CAN0, 0, 0xFFFFFFFF);
```

函数 can_operation_mode_enter

函数can_operation_mode_enter描述见下表:

表 3-82. 函数 can_operation_mode_enter

函数名称	can_operation_mode_enter
函数原型	ErrStatus can_operation_mode_enter(uint32_t can_periph, can_operation_modes_enum mode);
功能描述	进入对应的模式
先决条件	-
被调用函数	-
输入参数{in}	
can_periph	CAN 外设
CANx(x=0,1,2)	CAN 外设选择
输入参数{in}	
mode	参见枚举类型 表 3-74. 枚举类型 can_operation_modes_enum
输出参数{out}	
-	-
返回值	
ErrStatus	ERROR 或 SUCCESS

例如:

```
ErrStatus err;
```

```
/* CAN0 enter normal mode */
```

```
err = can_operation_mode_enter(CAN0, CAN_NORMAL_MODE);
```

函数 can_operation_mode_get

函数can_operation_mode_get描述见下表：

表 3-83. 函数 can_operation_mode_get

函数名称	can_operation_mode_get
函数原型	can_operation_modes_enum can_operation_mode_get(uint32_t can_periph);
功能描述	获取操作模式
先决条件	-
被调用函数	-
输入参数{in}	
can_periph	CAN 外设
CANx(x=0,1,2)	CAN 外设选择
输出参数{out}	
-	-
返回值	
can_operation_modes_enum	参见枚举类型 表 3-74. 枚举类型 can operation modes enum

例如：

```
can_operation_modes_enum mode;
```

```
/* get CAN0 mode*/
```

```
mode = can_operation_mode_get(CAN0);
```

函数 can_inactive_mode_exit

函数can_inactive_mode_exit描述见下表：

表 3-84. 函数 can_inactive_mode_exit

函数名称	can_inactive_mode_exit
函数原型	ErrStatus can_inactive_mode_exit(uint32_t can_periph);
功能描述	退出暂停模式
先决条件	-
被调用函数	-
输入参数{in}	
can_periph	CAN 外设
CANx(x=0,1,2)	CAN 外设选择
输出参数{out}	
-	-

返回值	
ErrStatus	ERROR 或 SUCCESS

例如:

```
ErrStatus err;
```

```
/* CAN0 exit INACTIVE mode */
```

```
err = can_inactive_mode_exit(CAN0);
```

函数 can_pn_mode_exit

函数can_pn_mode_exit描述见下表:

表 3-85. 函数 can_pn_mode_exit

函数名称	can_pn_mode_exit
函数原型	ErrStatus can_pn_mode_exit(uint32_t can_periph);
功能描述	退出虚拟联网模式
先决条件	-
被调用函数	-
输入参数{in}	
can_periph	CAN 外设
CANx(x=0,1,2)	CAN 外设选择
输出参数{out}	
-	-
返回值	
ErrStatus	ERROR 或 SUCCESS

例如:

```
ErrStatus err;
```

```
/* CAN0 exit PN mode */
```

```
err = can_pn_mode_exit(CAN0);
```

函数 can_fd_config

函数can_fd_config描述见下表:

表 3-86. 函数 can_fd_config

函数名称	can_fd_config
函数原型	void can_fd_config(uint32_t can_periph, can_fd_parameter_struct *can_fd_para_init);
功能描述	CAN FD 初始化
先决条件	-
被调用函数	-

输入参数{in}	
can_periph	CAN 外设
<i>CANx(x=0,1,2)</i>	CAN 外设选择
输入参数{in}	
can_fd_para_init	参见结构体 表 3-65. 结构体 can_fd_parameter_struct
输出参数{out}	
-	-
返回值	
-	-

例如：

```
can_fd_parameter_struct fd_parameter;
```

```
/* FD parameter configurations */
```

```
.....
```

```
can_fd_config(CAN0, &fd_parameter);
```

函数 can_bitrate_switch_enable

函数can_bitrate_switch_enable描述见下表：

表 3-87. 函数 can_bitrate_switch_enable

函数名称	can_bitrate_switch_enable
函数原型	void can_bitrate_switch_enable(uint32_t can_periph);
功能描述	使能波特率切换功能
先决条件	-
被调用函数	-
输入参数{in}	
can_periph	CAN 外设
<i>CANx(x=0,1,2)</i>	CAN 外设选择
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* enable CAN0 bit rate switching */
```

```
can_bitrate_switch_enable(CAN0);
```

函数 can_bitrate_switch_disable

函数can_bitrate_switch_disable描述见下表：

表 3-88. 函数 can_bitrate_switch_disable

函数名称	can_bitrate_switch_disable
函数原型	void can_bitrate_switch_disable(uint32_t can_periph);
功能描述	禁能波特率切换功能
先决条件	-
被调用函数	-
输入参数{in}	
can_periph	CAN 外设
CANx(x=0,1,2)	CAN 外设选择
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* disable CAN0 bit rate switching */
```

```
can_bitrate_switch_disable(CAN0);
```

函数 can_tdc_get

函数can_tdc_get描述见下表：

表 3-89. 函数 can_tdc_get

函数名称	can_tdc_get
函数原型	uint32_t can_tdc_get(uint32_t can_periph);
功能描述	获取传输延迟补偿值
先决条件	-
被调用函数	-
输入参数{in}	
can_periph	CAN 外设
CANx(x=0,1,2)	CAN 外设选择
输出参数{out}	
-	-
返回值	
uint32_t	0 - 0x3F

例如：

```
uint32_t tdc;
```

```
/* get transmitter delay compensation value */
```

```
tdc = can_tdc_get(CAN0);
```

函数 can_tdc_enable

函数can_tdc_enable描述见下表：

表 3-90. 函数 can_tdc_enable

函数名称	can_tdc_enable
函数原型	void can_tdc_enable(uint32_t can_periph);
功能描述	使能传输延迟补偿
先决条件	-
被调用函数	-
输入参数{in}	
can_periph	CAN 外设
CANx(x=0,1,2)	CAN 外设选择
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* enable transmitter delay compensation */
```

```
can_tdc_enable(CAN0);
```

函数 can_tdc_disable

函数can_tdc_disable描述见下表：

表 3-91. 函数 can_tdc_disable

函数名称	can_tdc_disable
函数原型	void can_tdc_disable(uint32_t can_periph);
功能描述	禁能传输延迟补偿
先决条件	-
被调用函数	-
输入参数{in}	
can_periph	CAN 外设
CANx(x=0,1,2)	CAN 外设选择
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* disable transmitter delay compensation */
```

```
can_tdc_disable(CAN0);
```

函数 `can_rx_fifo_config`

函数 `can_rx_fifo_config` 描述见下表：

表 3-92. 函数 `can_rx_fifo_config`

函数名称	<code>can_rx_fifo_config</code>
函数原型	<code>void can_rx_fifo_config(uint32_t can_periph, can_fifo_parameter_struct *can_fifo_para_init);</code>
功能描述	配置接收 FIFO
先决条件	-
被调用函数	-
输入参数{in}	
<code>can_periph</code>	CAN 外设
<code>CANx(x=0,1,2)</code>	CAN 外设选择
输入参数{in}	
<code>can_fifo_para_init</code>	参见结构体 表 3-67. 结构体 <code>can_fifo_parameter_struct</code>
输出参数{out}	
-	-
返回值	
-	-

例如：

```
can_fifo_parameter_struct fifo_struct;
```

```
/* configure rx FIFO */
```

```
.....
```

```
can_rx_fifo_config(CAN0, &fifo_struct);
```

函数 `can_rx_fifo_filter_table_config`

函数 `can_rx_fifo_filter_table_config` 描述见下表：

表 3-93. 函数 `can_rx_fifo_filter_table_config`

函数名称	<code>can_rx_fifo_filter_table_config</code>
函数原型	<code>void can_rx_fifo_filter_table_config(uint32_t can_periph, can_rx_fifo_id_filter_struct id_filter_table[]);</code>
功能描述	配置接收 FIFO 标识符过滤器表
先决条件	-
被调用函数	-
输入参数{in}	
<code>can_periph</code>	CAN 外设
<code>CANx(x=0,1,2)</code>	CAN 外设选择
输入参数{in}	
<code>id_filter_table</code>	参见结构体 表 3-66. 结构体 <code>can_rx_fifo_id_filter_struct</code>

输出参数{out}	
-	-
返回值	
-	-

例如：

```
can_rx_fifo_id_filter_struct id_filter_table[104];

/* configure rx FIFO filter table */

.....

can_rx_fifo_filter_table_config(CAN0, id_filter_table);
```

函数 can_rx_fifo_read

函数can_rx_fifo_read描述见下表：

表 3-94. 函数 can_rx_fifo_read

函数名称	can_rx_fifo_read
函数原型	void can_rx_fifo_read(uint32_t can_periph, can_rx_fifo_struct *rx_fifo);
功能描述	读取接收 FIFO 数据
先决条件	-
被调用函数	-
输入参数{in}	
can_periph	CAN 外设
CANx(x=0,1,2)	CAN 外设选择
输出参数{out}	
rx_fifo	参见结构体 表 3-64. 结构体 can_rx_fifo_struct
返回值	
-	-

例如：

```
can_rx_fifo_struct rx_fifo;

/* read rx FIFO data */

can_rx_fifo_read(CAN0, &rx_fifo);
```

函数 can_rx_fifo_filter_matching_number_get

函数can_rx_fifo_filter_matching_number_get描述见下表：

表 3-95. 函数 can_rx_fifo_filter_matching_number_get

函数名称	can_rx_fifo_filter_matching_number_get
函数原型	uint32_t can_rx_fifo_filter_matching_number_get(uint32_t can_periph);
功能描述	获取接收 FIFO 标识符过滤元素匹配序号

先决条件	-
被调用函数	-
输入参数{in}	
can_periph	CAN 外设
CANx(x=0,1,2)	CAN 外设选择
输出参数{out}	
-	-
返回值	
uint32_t	0-416

例如：

```
uint32_t number;
```

```
/* get rx FIFO filter matching number */
```

```
number = can_rx_fifo_filter_matching_number_get(CAN0);
```

函数 can_rx_fifo_clear

函数can_rx_fifo_clear描述见下表：

表 3-96. 函数 can_rx_fifo_clear

函数名称	can_rx_fifo_clear
函数原型	void can_rx_fifo_clear(uint32_t can_periph);
功能描述	清接收 FIFO
先决条件	-
被调用函数	-
输入参数{in}	
can_periph	CAN 外设
CANx(x=0,1,2)	CAN 外设选择
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* clear rx FIFO */
```

```
can_rx_fifo_clear(CAN0);
```

函数 can_ram_address_get

函数can_ram_address_get描述见下表：

表 3-97. 函数 can_ram_address_get

函数名称	can_ram_address_get
------	---------------------

函数原型	uint32_t* can_ram_address_get(uint32_t can_periph, uint32_t index);
功能描述	获取邮箱 RAM 地址
先决条件	-
被调用函数	-
输入参数{in}	
can_periph	CAN 外设
CANx(x=0,1,2)	CAN 外设选择
输入参数{in}	
index	邮箱索引
0-31	邮箱索引选择
输出参数{out}	
-	-
返回值	
uint32_t	0-0xFFFFFFFF

例如:

```
uint32_t address;
```

```
/* get CAN0 mailbox 0 RAM address* /
```

```
address = can_ram_address_get(CAN0, 0);
```

函数 can_mailbox_config

函数can_mailbox_config描述见下表:

表 3-98. 函数 can_mailbox_config

函数名称	can_mailbox_config
函数原型	void can_mailbox_config(uint32_t can_periph, uint32_t index, can_mailbox_descriptor_struct *mdpara);
功能描述	配置邮箱
先决条件	-
被调用函数	-
输入参数{in}	
can_periph	CAN 外设
CANx(x=0,1,2)	CAN 外设选择
输入参数{in}	
index	邮箱索引
0-31	邮箱索引选择
输入参数{in}	
mdpara	参见结构体 表 3-63. 结构体 can_mailbox_descriptor_struct
输出参数{out}	
-	-
返回值	

-	-
---	---

例如:

```
can_mailbox_descriptor_struct transmit_message;
```

```
.....
```

```
/* transmit message */
```

```
can_mailbox_config(CAN0, 0, &transmit_message);
```

函数 can_mailbox_transmit_abort

函数can_mailbox_transmit_abort描述见下表:

表 3-99. 函数 can_mailbox_transmit_abort

函数名称	can_mailbox_transmit_abort
函数原型	void can_mailbox_transmit_abort(uint32_t can_periph, uint32_t index);
功能描述	中止邮箱发送
先决条件	-
被调用函数	-
输入参数{in}	
can_periph	CAN 外设
CANx(x=0,1,2)	CAN 外设选择
输入参数{in}	
index	邮箱索引
0-31	邮箱索引选择
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* abort mailbox transmit */
```

```
can_mailbox_transmit_abort(CAN0, 0);
```

函数 can_mailbox_transmit_inactive

函数can_mailbox_transmit_inactive描述见下表:

表 3-100. 函数 can_mailbox_transmit_inactive

函数名称	can_mailbox_transmit_inactive
函数原型	void can_mailbox_transmit_inactive(uint32_t can_periph, uint32_t index);
功能描述	失活发送邮箱
先决条件	-

被调用函数	-
输入参数{in}	
can_periph	CAN 外设
CANx(x=0,1,2)	CAN 外设选择
输入参数{in}	
index	邮箱索引
0-31	邮箱索引选择
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* inactive transmit mailbox */
```

```
can_mailbox_transmit_inactive(CAN0, 0);
```

函数 can_mailbox_receive_data_read

函数can_mailbox_receive_data_read描述见下表:

表 3-101. 函数 can_mailbox_receive_data_read

函数名称	can_mailbox_receive_data_read
函数原型	ErrStatus can_mailbox_receive_data_read(uint32_t can_periph, uint32_t index, can_mailbox_descriptor_struct *mdpara);
功能描述	读取接收邮箱数据
先决条件	-
被调用函数	-
输入参数{in}	
can_periph	CAN 外设
CANx(x=0,1,2)	CAN 外设选择
输入参数{in}	
index	邮箱索引
0-31	邮箱索引选择
输入参数{in}	
mdpara	参见结构体 表 3-63. 结构体 can_mailbox_descriptor_struct
输出参数{out}	
-	-
返回值	
ErrStatus	ERROR 或 SUCCESS

例如:

```
ErrStatus err;
```

```
can_mailbox_descriptor_struct receive_message;
```

```
.....
```

```
/* check the receive message */
```

```
err = can_mailbox_receive_data_read(CAN0, 0, &receive_message);
```

函数 can_mailbox_receive_lock

函数can_mailbox_receive_lock描述见下表：

表 3-102. 函数 can_mailbox_receive_lock

函数名称	can_mailbox_receive_lock
函数原型	void can_mailbox_receive_lock(uint32_t can_periph, uint32_t index);
功能描述	锁定接收邮箱
先决条件	-
被调用函数	-
输入参数{in}	
can_periph	CAN 外设
CANx(x=0,1,2)	CAN 外设选择
输入参数{in}	
index	邮箱索引
0-31	邮箱索引选择
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* lock the receive mailbox 0 */
```

```
can_mailbox_receive_lock(CAN0, 0);
```

函数 can_mailbox_receive_unlock

函数can_mailbox_receive_unlock描述见下表：

表 3-103. 函数 can_mailbox_receive_unlock

函数名称	can_mailbox_receive_unlock
函数原型	void can_mailbox_receive_unlock(uint32_t can_periph);
功能描述	解锁接收邮箱
先决条件	-
被调用函数	-
输入参数{in}	
can_periph	CAN 外设

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

例如：

```
/* unlock the receive mailbox */
```

```
can_mailbox_receive_unlock(CAN0);
```

函数 can_mailbox_receive_inactive

函数can_mailbox_receive_inactive描述见下表：

表 3-104. 函数 can_mailbox_receive_inactive

函数名称	can_mailbox_receive_inactive
函数原型	void can_mailbox_receive_inactive(uint32_t can_periph, uint32_t index);
功能描述	失活接收邮箱
先决条件	-
被调用函数	-
输入参数{in}	
can_periph	CAN 外设
CANx(x=0,1,2)	CAN 外设选择
输入参数{in}	
index	邮箱索引
0-31	邮箱索引选择
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* inactive the receive mailbox */
```

```
can_mailbox_receive_inactive(CAN0, 0);
```

函数 can_mailbox_code_get

函数can_mailbox_code_get描述见下表：

表 3-105. 函数 can_mailbox_code_get

函数名称	can_mailbox_code_get
函数原型	uint32_t can_mailbox_code_get(uint32_t can_periph, uint32_t index);
功能描述	获取邮箱 CODE 值

先决条件	-
被调用函数	-
输入参数{in}	
can_periph	CAN 外设
CANx(x=0,1,2)	CAN 外设选择
输入参数{in}	
index	邮箱索引
0-31	邮箱索引选择
输出参数{out}	
-	-
返回值	
uint32_t	0-0xF

例如:

```
uint32_t code;

/* get mailbox code value */

code = can_mailbox_code_get(CAN0, 0);
```

函数 can_error_counter_config

函数can_error_counter_config描述见下表:

表 3-106. 函数 can_error_counter_config

函数名称	can_error_counter_config
函数原型	void can_error_counter_config(uint32_t can_periph, can_error_counter_struct *errcnt_struct);
功能描述	配置错误计数
先决条件	-
被调用函数	-
输入参数{in}	
can_periph	CAN 外设
CANx(x=0,1,2)	CAN 外设选择
输入参数{in}	
errcnt_struct	参见结构体 表 3-61. 结构体 can_error_counter_struct
输出参数{out}	
-	-
返回值	
-	-

例如:

```
can_error_counter_struct err_struct;

.....
```



```
/* configure error counter */
```

```
can_error_counter_config(CAN0, &err_struct);
```

函数 can_error_counter_get

函数can_error_counter_get描述见下表：

表 3-107. 函数 can_error_counter_get

函数名称	can_error_counter_get
函数原型	void can_error_counter_get(uint32_t can_periph, can_error_counter_struct *errcnt_struct);
功能描述	获取错误计数
先决条件	-
被调用函数	-
输入参数{in}	
can_periph	CAN 外设
CANx(x=0,1,2)	CAN 外设选择
输入参数{in}	
errcnt_struct	参见结构体 表 3-61. 结构体 can_error_counter_struct
输出参数{out}	
-	-
返回值	
-	-

例如：

```
can_error_counter_struct err_struct;
```

```
/* get error count */
```

```
can_error_counter_get(CAN0, &err_struct);
```

函数 can_error_state_get

函数can_error_state_get描述见下表：

表 3-108. 函数 can_error_state_get

函数名称	can_error_state_get
函数原型	can_error_state_enum can_error_state_get(uint32_t can_periph);
功能描述	获取错误状态指示
先决条件	-
被调用函数	-
输入参数{in}	
can_periph	CAN 外设
CANx(x=0,1,2)	CAN 外设选择
输出参数{out}	

-	-
返回值	
can_error_state_enum	参见枚举类型 表 3-76. 枚举类型 can_error_state_enum

例如：

```
can_error_state_enum error_state;

/* get error state indicator */

error_state = can_error_state_get(CAN0);
```

函数 can_crc_get

函数can_crc_get描述见下表：

表 3-109. 函数 can_crc_get

函数名称	can_crc_get
函数原型	void can_crc_get(uint32_t can_periph, can_crc_struct *crc_struct);
功能描述	获取邮箱 CRC 值
先决条件	-
被调用函数	-
输入参数{in}	
can_periph	CAN 外设
CANx(x=0,1,2)	CAN 外设选择
返回值	
crc_struct	参见结构体 表 3-70. 结构体 can_crc_struct
输出参数{out}	
-	-
返回值	
-	-

例如：

```
can_crc_struct crc_struct;

/* get mailbox CRC value */

can_crc_get(CAN0, &crc_struct);
```

函数 can_pn_mode_config

函数can_pn_mode_config描述见下表：

表 3-110. 函数 can_pn_mode_config

函数名称	can_pn_mode_config
函数原型	void can_pn_mode_config(uint32_t can_periph, can_pn_mode_config_struct *pnmod_config);

功能描述	配置虚拟联网模式参数
先决条件	-
被调用函数	-
输入参数{in}	
can_periph	CAN 外设
CANx(x=0,1,2)	CAN 外设选择
返回值	
pnmod_config	参见结构体 表 3-69. 结构体 can_pn_mode_config_struct
输出参数{out}	
-	-
返回值	
-	-

例如：

```
can_pn_mode_config_struct pn_struct;
.....
/* configure Pretended Networking mode parameter */
can_pn_mode_config(CAN0, &pn_struct);
```

函数 can_pn_mode_filter_config

函数can_pn_mode_filter_config描述见下表：

表 3-111. 函数 can_pn_mode_filter_config

函数名称	can_pn_mode_filter_config
函数原型	void can_pn_mode_filter_config(uint32_t can_periph, can_pn_mode_filter_struct *expect, can_pn_mode_filter_struct *filter);
功能描述	配置虚拟联网模式过滤器参数
先决条件	-
被调用函数	-
输入参数{in}	
can_periph	CAN 外设
CANx(x=0,1,2)	CAN 外设选择
返回值	
expect	参见结构体 表 3-68. 结构体 can_pn_mode_filter_struct
返回值	
filter	参见结构体 表 3-68. 结构体 can_pn_mode_filter_struct
输出参数{out}	
-	-
返回值	
-	-

例如:

```
can_pn_mode_filter_struct pn_filter[2];

.....

/* configure pn mode filter */

can_pn_mode_filter_config(CAN0, &pn_filter[0], &pn_filter[1]);
```

函数 can_pn_mode_num_of_match_get

函数can_pn_mode_num_of_match_get描述见下表:

表 3-112. 函数 can_pn_mode_num_of_match_get

函数名称	can_pn_mode_num_of_match_get
函数原型	int32_t can_pn_mode_num_of_match_get(uint32_t can_periph);
功能描述	获取虚拟联网模式下匹配的消息计数
先决条件	-
被调用函数	-
输入参数{in}	
can_periph	CAN 外设
CANx(x=0,1,2)	CAN 外设选择
输出参数{out}	
-	-
返回值	
int32_t	0-255 或-1

例如:

```
int32_t counter;

/* get matching message counter of Pretended Networking mode */

counter = can_pn_mode_num_of_match_get(CAN0);
```

函数 can_pn_mode_data_read

函数can_pn_mode_data_read描述见下表:

表 3-113. 函数 can_pn_mode_data_read

函数名称	can_pn_mode_data_read
函数原型	void can_pn_mode_data_read(uint32_t can_periph, uint32_t index, can_mailbox_descriptor_struct *mdpara);
功能描述	获取匹配的消息
先决条件	-
被调用函数	-
输入参数{in}	

can_periph	CAN 外设
CANx(x=0,1,2)	CAN 外设选择
输入参数{in}	
index	邮箱索引
0-31	邮箱索引选择
输入参数{in}	
mdpara	参见结构体 表 3-63. 结构体 can_mailbox_descriptor_struct
输出参数{out}	
-	-
返回值	
-	-

例如：

```
can_mailbox_descriptor_struct mb_para;
```

```
/* get matching message */
```

```
can_pn_mode_data_read(CAN0, 0, &mb_para);
```

函数 can_self_reception_enable

函数can_self_reception_enable描述见下表：

表 3-114. 函数 can_self_reception_enable

函数名称	can_self_reception_enable
函数原型	void can_self_reception_enable(uint32_t can_periph);
功能描述	使能自接收
先决条件	-
被调用函数	-
输入参数{in}	
can_periph	CAN 外设
CANx(x=0,1,2)	CAN 外设选择
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* enable self reception */
```

```
can_self_reception_enable(CAN0);
```

函数 can_self_reception_disable

函数can_self_reception_disable描述见下表：

表 3-115. 函数 can_self_reception_disable

函数名称	can_self_reception_disable
函数原型	void can_self_reception_disable(uint32_t can_periph);
功能描述	禁能自接收
先决条件	-
被调用函数	-
输入参数{in}	
can_periph	CAN 外设
CANx(x=0,1,2)	CAN 外设选择
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* disable self reception */
```

```
can_self_reception_disable(CAN0);
```

函数 can_transmit_abort_enable

函数can_transmit_abort_enable描述见下表:

表 3-116. 函数 can_transmit_abort_enable

函数名称	can_transmit_abort_enable
函数原型	void can_transmit_abort_enable(uint32_t can_periph);
功能描述	使能发送中止功能
先决条件	-
被调用函数	-
输入参数{in}	
can_periph	CAN 外设
CANx(x=0,1,2)	CAN 外设选择
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* enable transmit abort */
```

```
can_transmit_abort_enable(CAN0);
```

函数 can_transmit_abort_disable

函数can_transmit_abort_disable描述见下表:

表 3-117. 函数 can_transmit_abort_disable

函数名称	can_transmit_abort_disable
函数原型	void can_transmit_abort_disable(uint32_t can_periph);
功能描述	禁能发送中止功能
先决条件	-
被调用函数	-
输入参数{in}	
can_periph	CAN 外设
CANx(x=0,1,2)	CAN 外设选择
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* disable transmit abort */
```

```
can_transmit_abort_disable(CAN0);
```

函数 can_auto_busoff_recovery_enable

函数can_auto_busoff_recovery_enable描述见下表:

表 3-118. 函数 can_auto_busoff_recovery_enable

函数名称	can_auto_busoff_recovery_enable
函数原型	void can_auto_busoff_recovery_enable(uint32_t can_periph);
功能描述	使能离线自动恢复模式
先决条件	-
被调用函数	-
输入参数{in}	
can_periph	CAN 外设
CANx(x=0,1,2)	CAN 外设选择
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* enable auto bus off recovery mode */
```

```
can_auto_busoff_recovery_enable(CAN0);
```

函数 can_auto_busoff_recovery_disable

函数can_auto_busoff_recovery_disable描述见下表:

表 3-119. 函数 can_auto_busoff_recovery_disable

函数名称	can_auto_busoff_recovery_disable
函数原型	void can_auto_busoff_recovery_disable(uint32_t can_periph);
功能描述	禁能离线自动恢复模式
先决条件	-
被调用函数	-
输入参数{in}	
can_periph	CAN 外设
CANx(x=0,1,2)	CAN 外设选择
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* disable auto bus off recovery mode */
can_auto_busoff_recovery_disable(CAN0);
```

函数 can_time_sync_enable

函数can_time_sync_enable描述见下表：

表 3-120. 函数 can_time_sync_enable

函数名称	can_time_sync_enable
函数原型	void can_time_sync_enable(uint32_t can_periph);
功能描述	使能时间同步模式
先决条件	-
被调用函数	-
输入参数{in}	
can_periph	CAN 外设
CANx(x=0,1,2)	CAN 外设选择
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* enable time sync mode */
can_time_sync_enable(CAN0);
```

函数 can_time_sync_disable

函数can_time_sync_disable描述见下表：

表 3-121. 函数 can_time_sync_disable

函数名称	can_time_sync_disable
函数原型	void can_time_sync_disable(uint32_t can_periph);
功能描述	禁能时间同步模式
先决条件	-
被调用函数	-
输入参数{in}	
can_periph	CAN 外设
CANx(x=0,1,2)	CAN 外设选择
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* disable time sync mode */
can_time_sync_disable(CAN0);
```

函数 can_edge_filter_mode_enable

函数can_edge_filter_mode_enable描述见下表：

表 3-122. 函数 can_edge_filter_mode_enable

函数名称	can_edge_filter_mode_enable
函数原型	void can_edge_filter_mode_enable(uint32_t can_periph);
功能描述	使能边沿过滤模式
先决条件	-
被调用函数	-
输入参数{in}	
can_periph	CAN 外设
CANx(x=0,1,2)	CAN 外设选择
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* enable edge filter mode */
can_edge_filter_mode_enable(CAN0);
```

函数 can_edge_filter_mode_disable

函数can_edge_filter_mode_disable描述见下表：

表 3-123. 函数 `can_edge_filter_mode_disable`

函数名称	<code>can_edge_filter_mode_disable</code>
函数原型	<code>void can_edge_filter_mode_disable(uint32_t can_periph);</code>
功能描述	禁能边沿过滤模式
先决条件	-
被调用函数	-
输入参数{in}	
<code>can_periph</code>	CAN 外设
<code>CANx(x=0,1,2)</code>	CAN 外设选择
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* disable edge filter mode */
can_edge_filter_mode_disable(CAN0);
```

函数 `can_ped_mode_enable`

函数`can_ped_mode_enable`描述见下表：

表 3-124. 函数 `can_ped_mode_enable`

函数名称	<code>can_ped_mode_enable</code>
函数原型	<code>void can_ped_mode_enable(uint32_t can_periph);</code>
功能描述	使能协议异常检测模式
先决条件	-
被调用函数	-
输入参数{in}	
<code>can_periph</code>	CAN 外设
<code>CANx(x=0,1,2)</code>	CAN 外设选择
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* enable protocol exception detection mode */
can_ped_mode_enable(CAN0);
```

函数 `can_ped_mode_disable`

函数`can_ped_mode_disable`描述见下表：

表 3-125. 函数 can_ped_mode_disable

函数名称	can_ped_mode_disable
函数原型	void can_ped_mode_disable(uint32_t can_periph);
功能描述	禁能协议异常检测模式
先决条件	-
被调用函数	-
输入参数{in}	
can_periph	CAN 外设
CANx(x=0,1,2)	CAN 外设选择
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* disable protocol exception detection mode */
```

```
can_ped_mode_disable(CAN0);
```

函数 can_arbitration_delay_bits_config

函数can_arbitration_delay_bits_config描述见下表:

表 3-126. 函数 can_arbitration_delay_bits_config

函数名称	can_arbitration_delay_bits_config
函数原型	void can_arbitration_delay_bits_config(uint32_t can_periph, uint32_t delay_bits);
功能描述	配置仲裁启动延迟位
先决条件	-
被调用函数	-
输入参数{in}	
can_periph	CAN 外设
CANx(x=0,1,2)	CAN 外设选择
输入参数{in}	
delay_bits	延迟位
0-31	延迟位选择
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* configure arbitration delay bits */
```

```
can_arbitration_delay_bits_config(CAN0, 2);
```

函数 can_bsp_mode_config

函数can_bsp_mode_config描述见下表:

表 3-127. 函数 can_bsp_mode_config

函数名称	can_bsp_mode_config
函数原型	void can_bsp_mode_config(uint32_t can_periph, uint32_t sampling_mode);
功能描述	配置位采样模式
先决条件	-
被调用函数	-
输入参数{in}	
can_periph	CAN 外设
CANx(x=0,1,2)	CAN 外设选择
输入参数{in}	
sampling_mode	位采样模式
CAN_BSP_MODE_ON E_SAMPLE	对接收的位只采样一次
CAN_BSP_MODE_TR HEE_SAMPLES	对接收的位采样 3 次
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* configure bit sampling mode */
```

```
can_bsp_mode_config(CAN0, CAN_BSP_MODE_ONE_SAMPLE);
```

函数 can_flag_get

函数can_flag_get描述见下表:

表 3-128. 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=0,1,2)	CAN 外设选择
输入参数{in}	

flag	参见枚举类型 表 3-72. 枚举类型 can_flag_enum
输出参数{out}	
-	-
返回值	
FlagStatus	SET 或 RESET

例如:

```
FlagStatus flag;
```

```
/* get CAN fifo available flag */
```

```
flag = can_flag_get(CAN0, CAN_FLAG_FIFO_AVAILABLE);
```

函数 can_flag_clear

函数can_flag_clear描述见下表:

表 3-129. 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=0,1,2)	CAN 外设选择
输入参数{in}	
flag	参见枚举类型 表 3-72. 枚举类型 can_flag_enum
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* clear CAN fifo available flag */
```

```
can_flag_clear(CAN0, CAN_FLAG_FIFO_AVAILABLE);
```

函数 can_interrupt_enable

函数can_interrupt_enable描述见下表:

表 3-130. 函数 can_interrupt_enable

函数名称	can_interrupt_enable
函数原型	void can_interrupt_enable(uint32_t can_periph, can_interrupt_enum interrupt);

功能描述	使能 CAN 中断
先决条件	-
被调用函数	-
输入参数{in}	
can_periph	CAN 外设
CANx(x=0,1,2)	CAN 外设选择
输入参数{in}	
interrupt	参见枚举类型 表 3-71. 枚举类型 can_interrupt_enum
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* enable CAN bus off interrupt */
can_interrupt_enable(CAN0, CAN_INT_BUSOFF);
```

函数 can_interrupt_disable

函数can_interrupt_disable描述见下表：

表 3-131. 函数 can_interrupt_disable

函数名称	can_interrupt_disable
函数原型	void can_interrupt_disable(uint32_t can_periph, can_interrupt_enum interrupt);
功能描述	禁能 CAN 中断
先决条件	-
被调用函数	-
输入参数{in}	
can_periph	CAN 外设
CANx(x=0,1,2)	CAN 外设选择
输入参数{in}	
interrupt	参见枚举类型 表 3-71. 枚举类型 can_interrupt_enum
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* disable CAN bus off interrupt */
can_interrupt_disable(CAN0, CAN_INT_BUSOFF);
```

函数 `can_interrupt_flag_get`

函数 `can_interrupt_flag_get` 描述见下表：

表 3-132. 函数 `can_interrupt_flag_get`

函数名称	<code>can_interrupt_flag_get</code>
函数原型	<code>FlagStatus can_interrupt_flag_get(uint32_t can_periph, can_interrupt_flag_enum int_flag);</code>
功能描述	获取 CAN 中断标志状态
先决条件	-
被调用函数	-
输入参数{in}	
<code>can_periph</code>	CAN 外设
<code>CANx(x=0,1,2)</code>	CAN 外设选择
输入参数{in}	
<code>int_flag</code>	参见枚举类型 表 3-73. 枚举类型 <code>can_interrupt_flag_enum</code>
输出参数{out}	
-	-
返回值	
<code>FlagStatus</code>	SET 或 RESET

例如：

```
FlagStatus int_flag;
```

```
/* get CAN fifo available interrupt flag */
```

```
int_flag = can_interrupt_flag_get(CAN0, CAN_INT_FLAG_FIFO_AVAILABLE);
```

函数 `can_interrupt_flag_clear`

函数 `can_interrupt_flag_clear` 描述见下表：

表 3-133. 函数 `can_interrupt_flag_clear`

函数名称	<code>can_interrupt_flag_clear</code>
函数原型	<code>void can_interrupt_flag_clear(uint32_t can_periph, can_interrupt_flag_enum int_flag);</code>
功能描述	清除 CAN 中断标志状态
先决条件	-
被调用函数	-
输入参数{in}	
<code>can_periph</code>	CAN 外设
<code>CANx(x=0,1,2)</code>	CAN 外设选择
输入参数{in}	
<code>int_flag</code>	参见枚举类型 表 3-73. 枚举类型 <code>can_interrupt_flag_enum</code>
输出参数{out}	

-	-
返回值	
-	-

例如：

```
/* clear CAN fifo available interrupt flag */
```

```
can_interrupt_flag_clear(CAN0, CAN_INT_FLAG_FIFO_AVAILABLE);
```


3.4. CMP

CMP通用比较器可独立工作，其输出端口可用于I/O口，也可和定时器结合使用。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[3.4.1](#)描述了CMP的寄存器列表，章节[3.4.2](#)对CMP库函数进行说明。

3.4.1. 外设寄存器说明

CMP寄存器列表如下表所示：

表 3-134. 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
CMP_SR	比较器备用选择寄存器
CMP0_CS	CMP0控制状态寄存器
CMP1_CS	CMP1控制状态寄存器

3.4.2. 外设库函数说明

CMP库函数列表如下表所示：

表 3-135. 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_noninverting_input_select	CMP正相输入选择
cmp_output_init	CMP输出初始化
cmp_output_mux_config	CMP复用输出配置
cmp_blanking_init	CMP消隐功能初始化
cmp_enable	使能CMP
cmp_disable	禁能CMP
cmp_window_enable	CMP窗口模式使能
cmp_window_disable	CMP窗口模式禁能
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

库函数名称	库函数描述
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 3-136. 枚举类型 `cmp_enum`

成员名称	功能描述
CMP0	比较器0
CMP1	比较器1

函数 `cmp_deinit`

函数`cmp_deinit`描述见下表：

表 3-137. 函数 `cmp_deinit`

函数名称	<code>cmp_deinit</code>
函数原型	<code>void cmp_deinit(cmp_enum cmp_periph);</code>
功能描述	复位CMP
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}	
cmp_periph	参考枚举 表 3-136. 枚举类型 <code>cmp_enum</code>
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* deinitialize CMP0 */
cmp_deinit(CMP0);
```

函数 `cmp_mode_init`

函数`cmp_mode_init`描述见下表：

表 3-138. 函数 `cmp_mode_init`

函数名称	<code>cmp_mode_init</code>
函数原型	<code>void cmp_mode_init(cmp_enum cmp_periph, uint32_t operating_mode, uint32_t inverting_input, uint32_t output_hysteresis);</code>
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 3-136. 枚举类型cmp_enum
输入参数{in}	
operating_mode	速度和功耗运行模式
CMP_MODE_HIGH SPEED	高速/全功耗
CMP_MODE_MIDD LESPEED	中速/中功耗
CMP_MODE_VERY LOWSPEED	超低速/超低功耗
输入参数{in}	
inverting_input	反相输入源
CMP_INVERTING_I NPUT_1_4VREFIN T	选择1/4V _{REFINT} 作为输入源
CMP_INVERTING_I NPUT_1_2VREFIN T	选择1/2V _{REFINT} 作为输入源
CMP_INVERTING_I NPUT_3_4VREFIN T	选择3/4V _{REFINT} 作为输入源
CMP_INVERTING_I NPUT_VREFINT	选择V _{REFINT} 作为输入源
CMP_INVERTING_I NPUT_DAC0_OUT0	选择PA4（DAC0_OUT0）作为输入源
CMP_INVERTING_I NPUT_DAC0_OUT1	选择PA5（DAC0_OUT1）作为输入源
CMP_INVERTING_I NPUT_PB1_PE10	PB1作为CMP0输入源或者PE10作为CMP1输入源
CMP_INVERTING_I NPUT_PC4_PE7	PC4作为CMP0输入源或者PE7作为CMP1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	高迟滞

<code>_HIGH</code>	
输出参数{out}	
-	-
返回值	
-	-

例如：

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

```
cmp_mode_init(CMP0, CMP_MODE_HIGHSPEED, CMP_1_4VREFINT, CMP_HYSTERE  
SIS_NO);
```

函数 `cmp_noninverting_input_select`

函数 `cmp_noninverting_input_select` 描述见下表：

表 3-139. 函数 `cmp_noninverting_input_select`

函数名称	<code>cmp_noninverting_input_select</code>
函数原型	<code>void cmp_noninverting_input_select(cmp_enum cmp_periph, uint32_t noninverting_input);</code>
功能描述	CMP选择正相输入源
先决条件	-
被调用函数	-
输入参数{in}	
<code>cmp_periph</code>	参考枚举 表 3-136. 枚举类型 <code>cmp_enum</code>
输入参数{in}	
<code>noninverting_input</code>	正相输入源
<code>CMP_NONINVERTING_INPUT_PB0_PE9</code>	PB0作为CMP0输入源或者PE9作为CMP1输入源
<code>CMP_NONINVERTING_INPUT_PB2_PE12</code>	PB2作为CMP0输入源或者PE12作为CMP1输入源
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* selecte the noninverting input for CMP0 */
```

```
cmp_noninverting_input_select (CMP0, CMP_NONINVERTING_INPUT_PB0_PE9);
```

函数 `cmp_output_init`

函数`cmp_output_init`描述见下表:

表 3-140. 函数 `cmp_output_init`

函数名称	<code>cmp_output_init</code>
函数原型	<code>void cmp_output_init(cmp_enum cmp_periph, uint32_t output_polarity);</code>
功能描述	CMP输出初始化
先决条件	-
被调用函数	-
输入参数{in}	
<code>cmp_periph</code>	参考枚举 表 3-136. 枚举类型<code>cmp_enum</code>
输入参数{in}	
<code>output_polarity</code>	CMP输出极性
<code>CMP_OUTPUT_POLARITY_INVERTED</code>	输出反相
<code>CMP_OUTPUT_POLARITY_NONINVERTED</code>	输出正相
输出参数{out}	
-	-
返回值	
-	-

例如:

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

```
cmp_output_init(CMP0, CMP_OUTPUT_POLARITY_NOINVERTED);
```

函数 `cmp_output_mux_config`

函数`cmp_output_mux_config`描述见下表:

表 3-141. 函数 `cmp_output_mux_config`

函数名称	<code>cmp_output_mux_config</code>
函数原型	<code>void cmp_output_mux_config(cmp_enum cmp_periph, uint32_t cmp_output_sel);</code>
功能描述	CMP输出端口配置
先决条件	-
被调用函数	-
输入参数{in}	
<code>cmp_periph</code>	参考枚举 表 3-136. 枚举类型<code>cmp_enum</code>
输入参数{in}	
<code>cmp_output_sel</code>	CMP备用输出选择
<code>CMP_AFSE_GPIO_</code>	备用输出选择PA6

<i>PA6</i>	
<i>CMP_AFSE_GPIO_</i> <i>PA8</i>	备用输出选择PA8
<i>CMP_AFSE_GPIO_</i> <i>PB12</i>	备用输出选择PB12
<i>CMP_AFSE_GPIO_</i> <i>PE6</i>	备用输出选择PE6
<i>CMP_AFSE_GPIO_</i> <i>PE15</i>	备用输出选择PE15
<i>CMP_AFSE_GPIO_</i> <i>PG2</i>	备用输出选择PG2
<i>CMP_AFSE_GPIO_</i> <i>PG3</i>	备用输出选择PG3
<i>CMP_AFSE_GPIO_</i> <i>PG4</i>	备用输出选择PG4
<i>CMP_AFSE_GPIO_</i> <i>PK0</i>	备用输出选择PK0
<i>CMP_AFSE_GPIO_</i> <i>PK1</i>	备用输出选择PK1
<i>CMP_AFSE_GPIO_</i> <i>PK2</i>	备用输出选择PK2
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* config CMP0 output port */
```

```
cmp_output_mux_config(CMP0, CMP_AFSE_PA6);
```

函数 cmp_blanking_init

函数cmp_blanking_init描述见下表：

表 3-142. 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 3-136. 枚举类型cmp_enum

输入参数{in}	
blanking_source_selection	消隐源选择
<i>CMP_BLANKING_NONE</i>	无消隐
<i>CMP_BLANKING_TIMER0_OC0</i>	选择TIMER0_CH0输出比较信号为消隐源
<i>CMP_BLANKING_TIMER1_OC2</i>	选择TIMER1_CH2输出比较信号为消隐源
<i>CMP_BLANKING_TIMER2_OC2</i>	选择TIMER2_CH2输出比较信号为消隐源
<i>CMP_BLANKING_TIMER2_OC3</i>	选择TIMER2_CH3输出比较信号为消隐源
<i>CMP_BLANKING_TIMER7_OC0</i>	选择TIMER7_CH0输出比较信号为消隐源
<i>CMP_BLANKING_TIMER14_OC0</i>	选择TIMER14_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_enable

函数cmp_enable描述见下表：

表 3-143. 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 3-136. 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 3-144. 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 3-136. 枚举类型cmp_enum
输出参数{out}	
-	-
返回值	
-	-

例如：

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

函数 cmp_window_enable

函数cmp_window_enable描述见下表：

表 3-145. 函数 cmp_window_enable

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

例如：


```
/* enable the window mode */
```

```
cmp_window_enable();
```

函数 cmp_window_disable

函数cmp_window_disable描述见下表：

表 3-146. 函数 cmp_window_disable

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

例如：

```
/* disable the window mode */
```

```
cmp_window_disable();
```

函数 cmp_lock_enable

函数cmp_lock_enable描述见下表：

表 3-147. 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 3-136. 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 3-148. 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 3-136. 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 3-149. 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 3-136. 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 3-150. 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 3-136. 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 3-151. 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 3-136. 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 3-152. 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 3-136. 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 3-153. 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 3-136. 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 3-154. 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 3-136. 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 3-155. 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 3-136. 枚举类型cmp_enum
输入参数{in}	
interrupt	CMP中断
CMP_INT_COMPARE	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 3-156. 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 3-136. 枚举类型cmp_enum
输入参数{in}	
interrupt	CMP中断
CMP_INT_COMPARE	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 3-157. 函数 `cmp_interrupt_flag_get`

函数名称	<code>cmp_interrupt_flag_get</code>
函数原形	<code>FlagStatus cmp_interrupt_flag_get(cmp_enum cmp_periph, uint32_t flag);</code>
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}	
<code>cmp_periph</code>	参考枚举 表 3-136. 枚举类型<code>cmp_enum</code>
输入参数{in}	
<code>flag</code>	CMP中断标志位
<code>CMP_INT_FLAG_C OMPARE</code>	CMP比较中断标志位
输出参数{out}	
-	-
返回值	
<code>FlagStatus</code>	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 3-158. 函数 `cmp_interrupt_flag_clear`

函数名称	<code>cmp_interrupt_flag_clear</code>
函数原形	<code>void cmp_interrupt_flag_clear(cmp_enum cmp_periph, uint32_t flag);</code>
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}	
<code>cmp_periph</code>	参考枚举 表 3-136. 枚举类型<code>cmp_enum</code>
输入参数{in}	
<code>flag</code>	CMP中断标志位
<code>CMP_INT_FLAG_C OMPARE</code>	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);
```

3.5. CPDM

时钟相位延迟模块（CPDM）模块用于将输入时钟的相位延迟后再输出时钟。章节[3.5.1](#)描述了CPDM的寄存器列表，章节[3.5.2](#)对CPDM库函数进行说明。

3.5.1. 外设寄存器描述

CPDM寄存器列表如下表所示:

表 3-159. CPDM 寄存器

寄存器名称	寄存器描述
CPDM_CTL	CPDM控制寄存器
CPDM_CFG	CPDM配置寄存器

3.5.2. 外设库函数说明

CPDM库函数列表如下表所示:

表 3-160. CPDM 库函数

库函数名称	库函数描述
cpdm_enable	使能CPDM
cpdm_disable	禁能CPDM
cpdm_delayline_sample_enable	使能CPDM延迟线模块
cpdm_delayline_sample_disable	禁能CPDM延迟线模块
cpdm_output_clock_phase_select	选择CPDM输出时钟相位
cpdm_delayline_length_valid_flag_get	获取延迟线长度有效标志
cpdm_delayline_length_get	获取延迟线长度
cpdm_clock_output	配置CPDM时钟输出

枚举类型 `cpdm_output_phase_enum`

表 3-161. 枚举类型 `cpdm_output_phase_enum`

成员名称	功能描述
CPDM_OUTPUT_PHASE_SELECTION_0	输出时钟相位 = 输入时钟

成员名称	功能描述
CPDM_OUTPUT_PHASE_SELECTION_1	输出时钟相位 = 输入时钟 + 1 * UNIT延迟
CPDM_OUTPUT_PHASE_SELECTION_2	输出时钟相位 = 输入时钟 + 2 * UNIT延迟
CPDM_OUTPUT_PHASE_SELECTION_3	输出时钟相位 = 输入时钟 + 3 * UNIT延迟
CPDM_OUTPUT_PHASE_SELECTION_4	输出时钟相位 = 输入时钟 + 4 * UNIT延迟
CPDM_OUTPUT_PHASE_SELECTION_5	输出时钟相位 = 输入时钟 + 5 * UNIT延迟
CPDM_OUTPUT_PHASE_SELECTION_6	输出时钟相位 = 输入时钟 + 6 * UNIT延迟
CPDM_OUTPUT_PHASE_SELECTION_7	输出时钟相位 = 输入时钟 + 7 * UNIT延迟
CPDM_OUTPUT_PHASE_SELECTION_8	输出时钟相位 = 输入时钟 + 8 * UNIT延迟
CPDM_OUTPUT_PHASE_SELECTION_9	输出时钟相位 = 输入时钟 + 9 * UNIT延迟
CPDM_OUTPUT_PHASE_SELECTION_10	输出时钟相位 = 输入时钟 + 10 * UNIT延迟
CPDM_OUTPUT_PHASE_SELECTION_11	输出时钟相位 = 输入时钟 + 11 * UNIT延迟
CPDM_OUTPUT_PHASE_SELECTION_12	输出时钟相位 = 输入时钟 + 12 * UNIT延迟

函数 cpdm_enable

函数cpdm_enable描述见下表:

表 3-162. 函数 cpdm_enable

函数名称	cpdm_enable
函数原形	void cpdm_enable(uint32_t cpdm_periph);
功能描述	使能CPDM
先决条件	-
被调用函数	-
输入参数{in}	
cpdm_periph	SDIO模块的时钟延迟模块
CPDM_SDIOx	x = 0,1
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* enable CPDM */  
  
cpdm_enable(CPDM_SDIO0);
```

函数 cpdm_disable

函数cpdm_disable描述见下表:

表 3-163. 函数 cpdm_disable

函数名称	cpdm_disable
函数原形	void cpdm_disable(uint32_t cpdm_periph);
功能描述	禁能CPDM
先决条件	-
被调用函数	-
输入参数{in}	
cpdm_periph	SDIO模块的时钟延迟模块
CPDM_SDIOx	x = 0,1
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* disable CPDM */  
  
cpdm_disable(CPDM_SDIO0);
```

函数 cpdm_delayline_sample_enable

函数cpdm_delayline_sample_enable描述见下表:

表 3-164. 函数 cpdm_delayline_sample_enable

函数名称	cpdm_delayline_sample_enable
函数原形	void cpdm_delayline_sample_enable(uint32_t cpdm_periph);
功能描述	使能CPDM延迟线模块
先决条件	-
被调用函数	-
输入参数{in}	
cpdm_periph	SDIO模块的时钟延迟模块
CPDM_SDIOx	x = 0,1
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* enable CPDM delay line sample module */
cpdm_delayline_sample_enable(CPDM_SDIO0);
```

函数 cpdm_delayline_sample_disable

函数cpdm_delayline_sample_disable描述见下表：

表 3-165. 函数 cpdm_delayline_sample_disable

函数名称	cpdm_delayline_sample_disable
函数原形	void cpdm_delayline_sample_disable(uint32_t cpdm_periph);
功能描述	禁能CPDM延迟线模块
先决条件	-
被调用函数	-
输入参数{in}	
cpdm_periph	SDIO模块的时钟延迟模块
CPDM_SDIOx	x = 0,1
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* disable CPDM delay line sample module */
cpdm_delayline_sample_disable(CPDM_SDIO0);
```

函数 cpdm_output_clock_phase_select

函数cpdm_output_clock_phase_select描述见下表：

表 3-166. 函数 cpdm_output_clock_phase_select

函数名称	cpdm_output_clock_phase_select
函数原形	void cpdm_output_clock_phase_select(uint32_t cpdm_periph, cpdm_output_phase_enum output_clock_phase);
功能描述	选择CPDM输出时钟相位
先决条件	-
被调用函数	-
输入参数{in}	
cpdm_periph	SDIO模块的时钟延迟模块
CPDM_SDIOx	x = 0,1
输入参数{in}	
output_clock_phase	输出时钟相位，请参考 表3-161. 枚举类型cpdm_output_phase_enum

输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* select CPDM output clock phase */
```

```
cpdm_output_clock_phase_select(CPDM_SDIO0,  
CPDM_OUTPUT_PHASE_SELECTION_0);
```

函数 cpdm_delayline_length_valid_flag_get

函数cpdm_delayline_length_valid_flag_get描述见下表：

表 3-167. 函数 cpdm_delayline_length_valid_flag_get

函数名称	cpdm_delayline_length_valid_flag_get
函数原形	FlagStatus cpdm_delayline_length_valid_flag_get(uint32_t cpdm_periph);
功能描述	获取延迟线长度有效标志
先决条件	-
被调用函数	-
输入参数{in}	
cpdm_periph	SDIO模块的时钟延迟模块
CPDM_SDIOx	x = 0,1
输出参数{out}	
-	-
返回值	
FlagStatus	SET或RESET

例如：

```
/* get delay line length valid flag */
```

```
FlagStatus flag;
```

```
flag = cpdm_delayline_length_valid_flag_get(CPDM_SDIO0);
```

函数 cpdm_delayline_length_get

函数cpdm_delayline_length_get描述见下表：

表 3-168. 函数 cpdm_delayline_length_get

函数名称	cpdm_delayline_length_get
函数原形	uint16_t cpdm_delayline_length_get(uint32_t cpdm_periph);
功能描述	获取延迟线长度
先决条件	-
被调用函数	-

输入参数{in}	
cpdm_periph	SDIO模块的时钟延迟模块
<i>CPDM_SDIOx</i>	<i>x = 0,1</i>
输出参数{out}	
-	-
返回值	
uint16_t	0x00~0xFFFF

例如:

```
/* get delay line length */
```

```
uint16_t len;
```

```
len = cpdm_delayline_length_get(CPDM_SDIO0);
```

函数 cpdm_clock_output

函数cpdm_clock_output描述见下表:

表 3-169. 函数 cpdm_clock_output

函数名称	cpdm_clock_output
函数原形	void cpdm_clock_output(uint32_t cpdm_periph, cpdm_output_phase_enum output_clock_phase);
功能描述	配置CPDM时钟输出
先决条件	-
被调用函数	-
输入参数{in}	
cpdm_periph	SDIO模块的时钟延迟模块
<i>CPDM_SDIOx</i>	<i>x = 0,1</i>
输入参数{in}	
output_clock_phase	输出时钟相位, 请参考 表3-161. 枚举类型cpdm_output_phase_enum
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* configure CPDM clock output */
```

```
cpdm_clock_output (CPDM_SDIO0, CPDM_OUTPUT_PHASE_SELECTION_1);
```

3.6. CRC

循环冗余校验码是一种用在数字网络和存储设备上的差错校验码，可以校验原始数据的偶然误差。章节[3.6.1](#)描述了CRC的寄存器列表，章节[3.6.2](#)对CRC库函数进行说明。

3.6.1. 外设寄存器说明

CRC寄存器列表如下表所示：

表 3-170. 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

3.6.2. 外设库函数说明

CRC库函数列表如下表所示：

表 3-171. 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	失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计算单个数据
crc_block_data_calculate	CRC计算一个数组

函数 crc_deinit

函数crc_deinit描述见下表：

表 3-172. 函数 crc_deinit

函数名称	crc_deinit
函数原形	void crc_deinit(void);

功能描述	复位CRC计算单元
先决条件	-
被调用函数	-
输入参数{in}	
-	-
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* reset crc */
```

```
crc_deinit();
```

函数 `crc_init_data_register_write`

函数 `crc_init_data_register_write` 描述见下表：

表 3-173. 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 3-174. 函数 `crc_data_register_read`

函数名称	<code>crc_data_register_read</code>
函数原形	<code>uint32_t crc_data_register_read(void);</code>
功能描述	读数据寄存器

先决条件	-
被调用函数	-
输入参数{in}	
-	-
输出参数{out}	
-	-
返回值	
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 3-175. 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 3-176. 函数 crc_free_data_register_write

函数名称	crc_free_data_register_write
------	------------------------------

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

例如：

```
/* write the free data register */
crc_free_data_register_write(0x11);
```

函数 crc_reverse_output_data_disable

函数crc_reverse_output_data_disable描述见下表：

表 3-177. 函数 crc_reverse_output_data_disable

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

例如：

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

函数 crc_reverse_output_data_enable

函数crc_reverse_output_data_enable描述见下表：

表 3-178. 函数 crc_reverse_output_data_enable

函数名称	crc_reverse_output_data_enable
函数原形	void crc_reverse_output_data_enable(void);

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

例如：

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

```
crc_reverse_output_data_enable();
```

函数 crc_input_data_reverse_config

函数crc_input_data_reverse_config描述见下表：

表 3-179. 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 3-180. 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 3-181. 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 3-182. 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 3-183. 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 计算单个数据
先决条件	-
被调用函数	-
输入参数{in}	
sdata	设定的输入数据
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	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 3-184. 函数 crc_block_data_calculate

函数名称	crc_block_data_calculate
函数原形	uint32_t crc_block_data_calculate(void *array, uint32_t size, uint8_t data_format);
功能描述	CRC计算一个数组
先决条件	-
被调用函数	-
输入参数{in}	
array	指向输入数组
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	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);
```

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

3.7.1. 外设寄存器说明

CTC寄存器列表如下表所示：

表 3-185. 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

3.7.2. 外设库函数说明

CTC库函数列表如下表所示：

表 3-186. 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

库函数名称	库函数描述
ctc_counter_reload_value_read	读取CTC计数器重载值
ctc_irc48m_trim_value_read	读取IRC48M校准值
ctc_flag_get	CTC状态标志获取
ctc_flag_clear	CTC状态标志清除
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

函数 ctc_deinit

函数ctc_deinit描述见下表：

表 3-187. 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 3-188. 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 3-189. 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	除能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：

```
/* 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 3-190. 函数 ctc_irc48m_trim_value_config

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

-	-
返回值	
-	-

例如：

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

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

函数 ctc_software_refsource_pulse_generate

函数ctc_software_refsource_pulse_generate描述见下表：

表 3-191. 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 3-192. 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_ENA	硬件校准开启

BLE	
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 3-193. 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 3-194. 函数 `ctc_refsource_signal_select`

函数名称	<code>ctc_refsource_signal_select</code>
函数原形	<code>void ctc_refsource_signal_select(uint32_t refs);</code>
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
<code>CTC_REFSOURCE_GPIO</code>	选择GPIO输入信号
<code>CTC_REFSOURCE_LXTAL</code>	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 3-195. 函数 `ctc_refsource_prescaler_config`

函数名称	<code>ctc_refsource_prescaler_config</code>
函数原形	<code>void ctc_refsource_prescaler_config(uint32_t prescaler);</code>
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
<code>CTC_REFSOURCE_PSC_OFF</code>	参考信号不分频
<code>CTC_REFSOURCE_PSC_DIV2</code>	参考信号2分频
<code>CTC_REFSOURCE_PSC_DIV4</code>	参考信号4分频
<code>CTC_REFSOURCE_PSC_DIV8</code>	参考信号8分频
<code>CTC_REFSOURCE_PSC_DIV16</code>	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 3-196. 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 3-197. 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 3-198. 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 3-199. 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 3-200. 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 3-201. 函数 `ctc_irc48m_trim_value_read`

函数名称	<code>ctc_irc48m_trim_value_read</code>
函数原形	<code>uint8_t ctc_irc48m_trim_value_read(void);</code>
功能描述	读取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	
<code>uint8_t</code>	8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_flag_get`

函数`ctc_flag_get`描述见下表:

表 3-202. 函数 `ctc_flag_get`

函数名称	<code>ctc_flag_get</code>
函数原形	<code>FlagStatus ctc_flag_get (uint32_t flag);</code>
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
<code>CTC_FLAG_CKOK</code>	时钟校准完成标志
<code>CTC_FLAG_CKWARN</code>	时钟校准警告标志
<code>CTC_FLAG_ERR</code>	错误标志
<code>CTC_FLAG_EREFS</code>	期望参考信号标志
<code>CTC_FLAG_CKERR</code>	时钟校准错误标志
<code>CTC_FLAG_REFMISS</code>	参考同步脉冲信号丢失标志
<code>CTC_FLAG_TRIMERR</code>	校准值错误标志
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 3-203. 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
CTC_FLAG_CKWA RN	时钟校准警告标志
CTC_FLAG_ERR	错误标志
CTC_FLAG_EREf	期望参考信号标志
CTC_FLAG_CKERR	时钟校准错误标志
CTC_FLAG_REFMIS	参考同步脉冲信号丢失标志
CTC_FLAG_TRIME RR	校准值错误标志
输出参数{out}	
-	-
返回值	
-	-

例如:

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

```
ctc_flag_clear (CTC_FLAG_CKOK);
```

函数 ctc_interrupt_enable

函数ctc_interrupt_enable描述见下表:

表 3-204. 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_CKWAR N	时钟校准警告中断
CTC_INT_ERR	错误中断
CTC_INT_EREf	期望参考信号中断
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* enable CTC clock trim OK interrupt */
ctc_interrupt_enable (CTC_INT_CKOK);
```

函数 ctc_interrupt_disable

函数ctc_interrupt_disable描述见下表:

表 3-205. 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	除能外设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_CKWAR N	时钟校准警告中断
CTC_INT_ERR	错误中断
CTC_INT_EREf	期望参考信号中断
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* disable CTC clock trim OK interrupt */
ctc_interrupt_disable (CTC_INT_CKOK);
```

函数 ctc_interrupt_flag_get

函数ctc_interrupt_flag_get描述见下表：

表 3-206. 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
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
CTC_INT_FLAG_CKWARN	时钟校准警告中断标志
CTC_INT_FLAG_ERRRR	错误中断标志
CTC_INT_FLAG_EXPECTREF	期望参考信号中断标志
CTC_INT_FLAG_KERR	时钟校准错误标志
CTC_INT_FLAG_RFMISSE	参考同步脉冲信号丢失标志
CTC_INT_FLAG_TERRR	校准值错误标志
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 3-207. 函数 `ctc_interrupt_flag_clear`

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

例如：

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

```
ctc_interrupt_flag_clear (CTC_INT_FLAG_CKOK);
```

3.8. 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[3.8.1](#)描述了DAC的寄存器列表，章节[3.8.2](#)对DAC库函数进行说明。

3.8.1. 外设寄存器说明

DAC寄存器列表如下表所示：

表 3-208. 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	DAC_OUT0 12位右对齐数据保持寄存器
DAC_OUT0_L12DH	DAC_OUT0 12位左对齐数据保持寄存器
DAC_OUT0_R8DH	DAC_OUT0 8位右对齐数据保持寄存器
DAC_OUT1_R12DH	DAC_OUT1 12位右对齐数据保持寄存器
DAC_OUT1_L12DH	DAC_OUT1 12位左对齐数据保持寄存器
DAC_OUT1_R8DH	DAC_OUT1 8位右对齐数据保持寄存器
DACC_R12DH	DAC并发模式12位右对齐数据保持寄存器
DACC_L12DH	DAC并发模式12位左对齐数据保持寄存器
DACC_R8DH	DAC并发模式8位右对齐数据保持寄存器
DAC_OUT0_DO	DAC_OUT0数据输出寄存器
DAC_OUT1_DO	DAC_OUT1数据输出寄存器
DAC_STAT0	DAC状态寄存器0
DAC_CALR	DAC校准寄存器
DAC_MDCR	DAC模式寄存器
DAC_SKSTR0	DAC采样保持模式采样时间寄存器0
DAC_SKSTR1	DAC采样保持模式采样时间寄存器1
DAC_SKKTR	DAC采样保持模式保持时间寄存器
DAC_SKRTR	DAC采样保持模式刷新时间寄存器

3.8.2. 外设库函数说明

DAC库函数列表如下表所示：

表 3-209. 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

库函数名称	库函数描述
dac_mode_config	DAC模式配置
dac_trimming_value_get	DACx偏移值获取
dac_trimming_value_set	DACx偏移值设置
dac_trimming_enable	DACx校准使能
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_concurrent_enable	并发DAC模式使能
dac_concurrent_disable	并发DAC模式禁能
dac_concurrent_software_trigger_enable	并发DAC模式软件触发使能
dac_concurrent_data_set	并发DAC模式输出数据设置
dac_sample_keep_mode_config	DAC采样保持模式配置
dac_flag_get	DAC标志位获取
dac_flag_clear	DAC标志位清除
dac_interrupt_enable	DAC中断使能
dac_interrupt_disable	DAC中断禁能
dac_interrupt_flag_get	DAC中断标志位获取
dac_interrupt_flag_clear	DAC中断标志位清除

函数 dac_deinit

函数dac_deinit描述见下表：

表 3-210. 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 3-211. 函数 dac_enable

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

例如:

```
/* enable DAC0_OUT0 */
```

```
dac_enable(DAC0, DAC_OUT0);
```

函数 **dac_disable**

函数dac_disable描述见下表:

表 3-212. 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

<i>DACx</i>	DAC外设选择 (x = 0)
输入参数{in}	
dac_out	DAC输出
<i>DAC_OUTx</i>	DAC输出通道选择 (x = 0,1)
输出参数{out}	
-	-
返回值	
-	-

例如:

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

函数 **dac_dma_enable**

函数dac_dma_enable描述见下表:

表 3-213. 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 = 0,1)
输出参数{out}	
-	-
返回值	
-	-

例如:

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

函数 **dac_dma_disable**

函数dac_dma_disable描述见下表:

表 3-214. 函数 `dac_dma_disable`

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

例如:

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

函数 `dac_mode_config`

函数 `dac_mode_config` 描述见下表:

表 3-215. 函数 `dac_mode_config`

函数名称	<code>dac_mode_config</code>
函数原型	<code>void dac_mode_config(uint32_t dac_periph, uint32_t dac_out, uint32_t mode);</code>
功能描述	DAC模式配置
先决条件	-
被调用函数	-
输入参数{in}	
<code>dac_periph</code>	DAC外设
<code>DACx</code>	DAC外设选择 ($x = 0$)
输入参数{in}	
<code>dac_out</code>	DAC输出
<code>DAC_OUTx</code>	DAC输出通道选择 ($x = 0,1$)
输入参数{in}	
<code>mode</code>	DAC工作模式
<code>NORMAL_PIN_BUFFERON</code>	普通模式下, DAC_OUTx连接到外部引脚, 缓冲区启用
<code>NORMAL_PIN_PERIPHERAL_BUFFERON</code>	普通模式下, DAC_OUTx连接到外部引脚和片上外设, 缓冲区启用

<code>NORMAL_PIN_BUFFEROFF</code>	普通模式下，DAC_OUTx连接到外部引脚，缓冲区禁用
<code>NORMAL_PIN_PERIPHERAL_BUFFEROFF</code>	普通模式下，DAC_OUTx连接到外部引脚和片上外设，缓冲区禁用
<code>SAMPLEKEEP_PIN_BUFFERON</code>	采样保持模式下，DAC_OUTx连接到外部引脚，缓冲区启用
<code>SAMPLEKEEP_PIN_PERIPHERAL_BUFFERON</code>	采样保持模式下，DAC_OUTx连接到外部引脚和片上外设，缓冲区启用
<code>SAMPLEKEEP_PIN_BUFFEROFF</code>	采样保持模式下，DAC_OUTx连接到外部引脚，缓冲区禁用
<code>SAMPLEKEEP_PIN_PERIPHERAL_BUFFEROFF</code>	采样保持模式下，DAC_OUTx连接到外部引脚和片上外设，缓冲区禁用
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* configure DAC0_OUT0 working in NORMAL_PIN_BUFFERON mode */
```

```
dac_mode_config(DAC0, DAC_OUT0, NORMAL_PIN_BUFFERON);
```

函数 `dac_trimming_value_get`

函数 `dac_trimming_value_get` 描述见下表:

表 3-216. 函数 `dac_trimming_value_get`

函数名称	<code>dac_trimming_value_get</code>
函数原型	<code>void dac_trimming_value_get(uint32_t dac_periph, uint32_t dac_out);</code>
功能描述	DACx偏移值获取
先决条件	-
被调用函数	-
输入参数{in}	
<code>dac_periph</code>	DAC外设
<code>DACx</code>	DAC外设选择 (x = 0)
输入参数{in}	
<code>dac_out</code>	DAC输出
<code>DAC_OUTx</code>	DAC输出通道选择 (x = 0,1)
输出参数{out}	
-	-
返回值	

Uint32_t	DACx偏移值
----------	---------

例如:

```
/* get the DAC0_OUT0 trimming value */
data = dac_trimming_value_get (DAC0, DAC_OUT0);
```

函数 dac_trimming_value_set

函数dac_trimming_value_set描述见下表:

表 3-217. 函数 dac_trimming_value_set

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

例如:

```
/* set the DAC0_OUT0 trimming value */
dac_trimming_value_set(DAC0, DAC_OUT0, 1);
```

函数 dac_trimming_enable

函数dac_trimming_enable描述见下表:

表 3-218. 函数 dac_trimming_enable

函数名称	dac_trimming_enable
函数原型	void dac_trimming_enable(uint32_t dac_periph, uint32_t dac_out);
功能描述	DACx校准使能
先决条件	-

被调用函数	-
输入参数{in}	
dac_periph	DAC外设
<i>DACx</i>	DAC外设选择 (x = 0)
输入参数{in}	
dac_out	DAC输出
<i>DAC_OUTx</i>	DAC输出通道选择 (x = 0,1)
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* enable the DAC0_OUT0 trimming */
dac_trimming_enable (DAC0, DAC_OUT0);
```

函数 **dac_output_value_get**

函数dac_output_value_get描述见下表:

表 3-219. 函数 **dac_output_value_get**

函数名称	dac_output_value_get
函数原型	uint16_t dac_output_value_get(uint32_t dac_periph, uint8_t dac_out);
功能描述	DAC输出数据获取
先决条件	-
被调用函数	-
输入参数{in}	
dac_periph	DAC外设
<i>DACx</i>	DAC外设选择 (x = 0)
输入参数{in}	
dac_out	DAC输出
<i>DAC_OUTx</i>	DAC输出通道选择 (x = 0,1)
输出参数{out}	
-	-
返回值	
uint16_t	外设DACx数据保持寄存器值 (0~4095)

例如:

```
/* get the DAC0_OUT0 last data output value */
uint16_t data;

data = dac_output_value_get(DAC0, DAC_OUT0);
```

函数 **dac_data_set**

函数dac_data_set描述见下表:

表 3-220. 函数 **dac_data_set**

函数名称	dac_data_set
函数原型	void dac_data_set(uint32_t dac_periph, uint8_t dac_out, uint32_t dac_align, uint16_t data);
功能描述	DAC输出数据设置
先决条件	-
被调用函数	-
输入参数{in}	
dac_periph	DAC外设
DACx	DAC外设选择 (x = 0)
输入参数{in}	
dac_out	DAC输出
DAC_OUTx	DAC输出通道选择 (x = 0,1)
输入参数{in}	
dac_align	DAC对齐模式
DAC_ALIGN_12B_R	12位数据右对齐
DAC_ALIGN_12B_L	12位数据左对齐
DAC_ALIGN_8B_R	8位数据右对齐
输入参数{in}	
data	写入DAC_OUTx的数据 (0~4095)
输出参数{out}	
-	-
返回值	
-	-

例如:

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

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

函数 **dac_trigger_enable**

函数dac_trigger_enable描述见下表:

表 3-221. 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
<i>DACx</i>	DAC外设选择 (x = 0)
输入参数{in}	
dac_out	DAC输出
<i>DAC_OUTx</i>	DAC输出通道选择 (x = 0,1)
输出参数{out}	
-	-
返回值	
-	-

例如:

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

函数 **dac_trigger_disable**

函数dac_trigger_disable描述见下表:

表 3-222. 函数 **dac_trigger_disable**

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

例如:

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

函数 dac_trigger_source_config

函数dac_trigger_source_config描述见下表:

表 3-223. 函数 dac_trigger_source_config

函数名称	dac_trigger_source_config
函数原型	void dac_trigger_source_config(uint32_t dac_periph, uint8_t dac_out, uint32_t triggersource);
功能描述	DAC触发源配置
先决条件	-
被调用函数	-
输入参数{in}	
dac_periph	DAC外设
DACx	DAC外设选择 (x = 0)
输入参数{in}	
dac_out	DAC输出
DAC_OUTx	DAC输出通道选择 (x = 0,1)
输入参数{in}	
triggersource	DAC触发源
DAC_TRIGGER_EXTERNAL	来自于TRIGSEL的外部触发
DAC_TRIGGER_SOFTWARE	软件触发
输出参数{out}	
-	-
返回值	
-	-

例如:

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

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

函数 dac_software_trigger_enable

函数dac_software_trigger_enable描述见下表:

表 3-224. 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 = 0,1)
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* enable DAC0_OUT0 software trigger */
```

```
dac_software_trigger_enable(DAC0, DAC_OUT0);
```

函数 **dac_wave_mode_config**

函数dac_wave_mode_config描述见下表:

表 3-225. 函数 **dac_wave_mode_config**

函数名称	dac_wave_mode_config
函数原型	void dac_wave_mode_config(uint32_t dac_periph, uint8_t dac_out, uint32_t wave_mode);
功能描述	DAC噪声波模式配置
先决条件	-
被调用函数	-
输入参数{in}	
dac_periph	DAC外设
<i>DACx</i>	DAC外设选择 (x = 0)
输入参数{in}	
dac_out	DAC输出
<i>DAC_OUTx</i>	DAC输出通道选择 (x = 0,1)
输入参数{in}	
wave_mode	噪声波模式选择
<i>DAC_WAVE_DISABLE</i>	噪声波模式禁能
<i>DAC_WAVE_MODE_LFSR</i>	LFSR噪声波模式
<i>DAC_WAVE_MODE_TRIANGLE</i>	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 3-226. 函数 `dac_lfsr_noise_config`

函数名称	<code>dac_lfsr_noise_config</code>
函数原型	<code>void dac_lfsr_noise_config(uint32_t dac_periph, uint8_t dac_out, uint32_t unmask_bits);</code>
功能描述	DAC LFSR 模式配置
先决条件	-
被调用函数	-
输入参数{in}	
<code>dac_periph</code>	DAC 外设
<code>DACx</code>	DAC 外设选择 ($x = 0$)
输入参数{in}	
<code>dac_out</code>	DAC 输出
<code>DAC_OUTx</code>	DAC 输出通道选择 ($x = 0, 1$)
输入参数{in}	
<code>unmask_bits</code>	噪声波的非屏蔽位宽
<code>DAC_LFSR_BIT0</code>	LFSR 模式位 0 非屏蔽
<code>DAC_LFSR_BITSx_0</code>	LFSR 模式位 $[x:0]$ 非屏蔽 ($x = 1, 2, 3 \dots 11$)
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 3-227. 函数 `dac_triangle_noise_config`

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

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

例如:

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

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

函数 dac_concurrent_enable

函数dac_concurrent_enable描述见下表:

表 3-228. 函数 dac_concurrent_enable

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

例如:

```
/* enable DAC0 concurrent mode */
```

```
dac_concurrent_enable(DAC0);
```

函数 `dac_concurrent_disable`

函数 `dac_concurrent_disable` 描述见下表：

表 3-229. 函数 `dac_concurrent_disable`

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

例如：

```
/* disable DAC0 concurrent mode */
```

```
dac_concurrent_disable(DAC0);
```

函数 `dac_concurrent_software_trigger_enable`

函数 `dac_concurrent_software_trigger_enable` 描述见下表：

表 3-230. 函数 `dac_concurrent_software_trigger_enable`

函数名称	<code>dac_concurrent_software_trigger_enable</code>
函数原型	<code>void dac_concurrent_software_trigger_enable(uint32_t dac_periph);</code>
功能描述	并发DAC模式软件触发使能
先决条件	-
被调用函数	-
输入参数{in}	
<code>dac_periph</code>	DAC外设
<code>DACx</code>	DAC外设选择（ <code>x = 0</code> ）
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* enable DAC0 concurrent software trigger */
```

dac_concurrent_software_trigger_enable(DAC0);

函数 dac_concurrent_data_set

函数dac_concurrent_data_set描述见下表:

表 3-231. 函数 dac_concurrent_data_set

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

例如:

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

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

函数 dac_sample_keep_mode_config

函数dac_sample_keep_mode_config描述见下表:

表 3-232. 函数 dac_sample_keep_mode_config

函数名称	dac_sample_keep_mode_config
函数原型	void dac_sample_keep_mode_config(uint32_t dac_periph, uint32_t dac_out, uint32_t sample_time, uint32_t keep_time, uint32_t refresh_time);
功能描述	DAC采样和保持时间值设置

先决条件	-
被调用函数	-
输入参数{in}	
dac_periph	DAC外设
<i>DACx</i>	DAC外设选择 (x = 0)
输入参数{in}	
dac_out	DAC输出
<i>DAC_OUTx</i>	DAC输出通道选择 (x = 0,1)
输入参数{in}	
sample_time	DAC采样时间
输入参数{in}	
keep_time	DAC保持时间
输入参数{in}	
refresh_time	DAC刷新时间
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* set DAC0_OUT0 sample and keep time value */
```

```
dac_sample_keep_mode_config (DAC0, DAC_OUT0, 1, 1, 1);
```

函数 **dac_flag_get**

函数dac_flag_get描述见下表:

表 3-233. 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
<i>DACx</i>	DAC外设选择 (x = 0)
输入参数{in}	
flag	DAC状态标志位
<i>DAC_FLAG_DDUD</i> <i>R0</i>	DACx_OUT0 DMA欠载标志位
<i>DAC_FLAG_CALF0</i>	DACx_OUT0 DMA校准偏移位
<i>DAC_FLAG_BWT0</i>	DACx_OUT0 采样和保持时间写入标志位
<i>DAC_FLAG_DDUD</i>	DACx_OUT1 DMA欠载标志位

<i>R1</i>	
<i>DAC_FLAG_CALF1</i>	DACx_OUT1 DMA校准偏移位
<i>DAC_FLAG_BWT1</i>	DACx_OUT1采样和保持时间写入标志位
输出参数{out}	
-	-
返回值	
FlagStatus	DAC位状态（SET或RESET）

例如:

```
/* get DAC0 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 3-234. 函数 **dac_flag_clear**

函数名称	dac_flag_clear
函数原型	void dac_flag_clear(uint32_t dac_periph, uint32_t flag);
功能描述	DAC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）
输入参数{in}	
flag	DAC状态标志位
<i>DAC_FLAG_DDUD</i> <i>R0</i>	DACx_OUT0 DMA欠载标志位
<i>DAC_FLAG_DDUD</i> <i>R1</i>	DACx_OUT1 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 3-235. 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 ($x = 0$)
输入参数{in}	
<code>interrupt</code>	DAC中断
<code>DAC_INT_DDUDR0</code>	<code>DACx_OUT0</code> DMA欠载中断
<code>DAC_INT_DDUDR1</code>	<code>DACx_OUT1</code> DMA欠载中断
输出参数{out}	
-	-
返回值	
-	-

例如:

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

```
dac_interrupt_enable (DAC0, DAC_INT_DDUDRIE0);
```

函数 `dac_interrupt_disable`

函数`dac_interrupt_disable`描述见下表:

表 3-236. 函数 `dac_interrupt_disable`

函数名称	<code>dac_interrupt_disable</code>
函数原型	<code>void dac_interrupt_disable(uint32_t dac_periph, uint32_t interrupt);</code>
功能描述	DAC中断禁能
先决条件	-
被调用函数	-
输入参数{in}	
<code>dac_periph</code>	DAC外设
<code>DACx</code>	DAC外设选择 ($x = 0$)
输入参数{in}	
<code>interrupt</code>	DAC中断
<code>DAC_INT_DDUDR0</code>	<code>DACx_OUT0</code> DMA欠载中断
<code>DAC_INT_DDUDR1</code>	<code>DACx_OUT1</code> DMA欠载中断
输出参数{out}	
-	-
返回值	
-	-

例如:

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

```
dac_interrupt_disable (DAC0, DAC_INT_DDUDRIE0);
```

函数 `dac_interrupt_flag_get`

函数 `dac_interrupt_flag_get` 描述见下表:

表 3-237. 函数 `dac_interrupt_flag_get`

函数名称	<code>dac_interrupt_flag_get</code>
函数原型	<code>FlagStatus dac_interrupt_flag_get(uint32_t dac_periph, uint32_t int_flag);</code>
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}	
<code>dac_periph</code>	DAC外设
<code>DACx</code>	DAC外设选择 ($x = 0$)
输入参数{in}	
<code>int_flag</code>	DAC中断标志位
<code>DAC_INT_FLAG_D DUDR0</code>	DACx_OUT0 DMA欠载中断标志位
<code>DAC_INT_FLAG_D DUDR1</code>	DACx_OUT1 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 3-238. 函数 `dac_interrupt_flag_clear`

函数名称	<code>dac_interrupt_flag_clear</code>
函数原型	<code>void dac_interrupt_flag_clear(uint32_t dac_periph, uint32_t int_flag);</code>
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)
输入参数{in}	
int_flag	DAC中断标志位
<i>DAC_INT_FLAG_D DUDR0</i>	DACx_OUT0 DMA欠载中断标志位
<i>DAC_INT_FLAG_D DUDR1</i>	DACx_OUT1 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);
```


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

3.9.1. 外设寄存器说明

DBG寄存器列表如下表所示：

表 3-239. DBG 寄存器

寄存器名称	寄存器描述
DBG_ID	DBG ID寄存器
DBG_CTL0	DBG控制寄存器0
DBG_CTL1	DBG控制寄存器1
DBG_CTL2	DBG控制寄存器2
DBG_CTL3	DBG控制寄存器3
DBG_CTL4	DBG控制寄存器4

3.9.2. 外设库函数说明

DBG库函数列表如下表所示：

表 3-240. DBG 库函数

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

枚举类型 dbg_periph_enum

表 3-241. 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_TIMERx_HOLD	当内核停止时，保持TIMERx (x=0,1,2,3,4,5,6,7,14,15,16,22,23,30,31,40,41,42,43,44,50,51)计

成员名称	功能描述
	计数器计数值不变
DBG_I2Cx_HOLD	当内核停止时，保持I2Cx (x=0,1,2,3)的SMBUS状态不变，用于调试
DBG_CANx_HOLD	当内核停止时，保持CANx (x=0,1,2)计数器计数值不变
DBG_RTC_HOLD	当内核停止时，保持RTC日历和唤醒计数器值不变

函数 dbg_deinit

函数dbg_deinit描述见下表：

表 3-242. 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 3-243. 函数 dbg_id_get

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

返回值	
uint32_t	DBG ID (0-0xFFFFFFFF)

例如:

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

```
uint32_t id_value = 0;
```

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

函数 dbg_low_power_enable

函数dbg_low_power_enable描述见下表:

表 3-244. 函数 dbg_low_power_enable

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

例如:

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

```
dbg_low_power_enable(DBG_LOW_POWER_SLEEP);
```

函数 dbg_low_power_disable

函数dbg_low_power_disable描述见下表:

表 3-245. 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_SLEEP);
```

函数 dbg_periph_enable

函数dbg_periph_enable描述见下表：

表 3-246. 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 表3-241. 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_TIMERx_HOLD	当内核停止时，保持TIMERx (x=0,1,2,3,4,5,6,7,14,15,16,22,23,30,31,40,41,42,43,44,50,51)计数器计数值不变
DBG_I2Cx_HOLD	当内核停止时，保持I2Cx (x=0,1,2,3)的SMBUS状态不变，用于调试
DBG_CANx_HOLD	当内核停止时，保持CANx (x=0,1,2)计数器计数值不变
DBG_RTC_HOLD	当内核停止时，保持RTC日历和唤醒计数器值不变
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 3-247. 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3-241. 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_TIMERx_HOLD	当内核停止时，保持TIMERx (x=0,1,2,3,4,5,6,7,14,15,16,22,23,30,31,40,41,42,43,44,50,51)计数器计数值 不变
DBG_I2Cx_HOLD	当内核停止时，保持I2Cx (x=0,1,2,3)的SMBUS状态不变，用于调试
DBG_CANx_HOLD	当内核停止时，保持CANx (x=0,1,2)计数器计数值不变
DBG_RTC_HOLD	当内核停止时，保持RTC日历和唤醒计数器值不变
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 3-248. 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 3-249. 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 3-250. 函数 dbg_trace_pin_mode_set

函数名称	dbg_trace_pin_mode_set
函数原形	void dbg_trace_pin_mode_set(uint32_t trace_mode);
功能描述	配置跟踪引脚分配模式

先决条件	-
被调用函数	-
输入参数{in}	
trace_mode	跟踪引脚分配模式选择
TRACE_MODE_ASYNC	跟踪引脚用于异步模式
TRACE_MODE_SYNC_DATASIZE_1	跟踪引脚用于同步模式且数据长度为1
TRACE_MODE_SYNC_DATASIZE_2	跟踪引脚用于同步模式且数据长度为2
TRACE_MODE_SYNC_DATASIZE_4	跟踪引脚用于同步模式且数据长度为4
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* trace pin used for async mode */
```

```
dbg_trace_pin_mode_set(TRACE_MODE_ASYNC);
```

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

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

3.10.1. 外设寄存器说明

DMA寄存器列表如下表所示：

表 3-251. DMA 寄存器

寄存器名称	寄存器描述
DMA_INTF0	中断标志位寄存器0
DMA_INTF1	中断标志位寄存器1
DMA_INTC0	中断标志位清除寄存器0
DMA_INTC1	中断标志位清除寄存器1

寄存器名称	寄存器描述
DMA_CHxCTL (x=0..7)	通道x控制寄存器
DMA_CHxCNT (x=0..7)	通道x计数寄存器
DMA_CHxPADDR (x=0..7)	通道x外设基地址寄存器
DMA_CHxM0ADDR (x=0..7)	通道x存储器0基地址寄存器
DMA_CHxM1ADDR (x=0..7)	通道x存储器1基地址寄存器
DMA_CHxFCTL (x=0..7)	通道x FIFO控制寄存器

DMAMUX寄存器列表如下表所示：

表 3-252. DMAMUX 寄存器

寄存器名称	寄存器描述
DMAMUX_RM_CHx CFG (x=0..15)	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..7)	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

3.10.2. 外设库函数说明

DMA库函数列表如下表所示：

表 3-253. 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_single_data_para_struct_init	将DMA单数据传输模式结构体中所有参数初始化为默认值
dma_multi_data_para_struct_init	将DMA多数据传输模式结构体中所有参数初始化为默认值
dma_single_data_mode_init	初始化外设DMA单数据传输模式
dma_multi_data_mode_init	初始化外设DMA多数据传输模式

库函数名称	库函数描述
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_burst_beats_config	配置DMA通道x存储器增量突发传输节拍数
dma_periph_burst_beats_config	配置DMA通道x外设增量突发传输节拍数
dma_memory_width_config	配置DMA通道x存储器传输数据宽度
dma_periph_width_config	配置DMA通道x外设传输数据宽度
dma_memory_address_generation_config	配置DMA通道x存储器地址算法
dma_peripheral_address_generation_config	配置DMA通道x外设地址算法
dma_circulation_enable	使能DMA循环传输模式
dma_circulation_disable	失能DMA循环传输模式
dma_channel_enable	使能DMA通道x
dma_channel_disable	禁能DMA通道x
dma_transfer_direction_config	配置DMA通道x数据传输方向
dma_switch_buffer_mode_config	配置DMA通道x存储器传输缓冲区
dma_using_memory_get	获取DMA通道x存储器传输缓冲区
dma_switch_buffer_mode_enable	使能DMA通道x存储切换模式
dma_switch_buffer_mode_disable	禁能DMA通道x存储切换模式
dma_fifo_status_get	获取DMA通道xFIFO状态
dma_flag_get	获取DMA通道x标志
dma_flag_clear	清除DMA通道x标志
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
dma_interrupt_flag_clear	清除DMA通道x中断标志

DMAMUX库函数列表如下表所示：

表 3-254. DMAMUX 库函数

Function name	Function description
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

Function name	Function description
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_multi_data_parameter_struct

表 3-255. 结构体 dma_multi_data_parameter_struct

成员名称	功能描述
request	请求路由通道输入标识
periph_addr	外设基地址
periph_width	外设数据传输宽度
periph_inc	外设地址生成算法模式
memory0_addr	存储器0基地址
memory_width	存储器0数据传输宽度
memory_inc	存储器地址生成算法模式
memory_burst_width	存储器增量突发类型
periph_burst_width	外设增量突发类型
critical_value	DMA FIFO寄存器数
circular_mode	DMA循环模式
direction	DMA通道数据传输方向
number	DMA通道数据传输数
priority	DMA通道优先级

结构体 dma_single_data_parameter_struct

表 3-256. 结构体 dma_single_data_parameter_struct

成员名称	功能描述
request	请求路由通道输入标识
periph_addr	外设基地址
periph_inc	外设地址生成算法模式
memory0_addr	存储器0基地址
memory_inc	存储器地址生成算法模式
periph_memory_width	外设数据传输宽度
circular_mode	DMA循环模式
direction	DMA通道数据传输方向
number	DMA通道数据传输数
priority	DMA通道优先级

结构体 dmamux_sync_parameter_struct

表 3-257. 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 3-258. 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 3-259. 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
DMA_CH7	DMA通道7

枚举 dmamux_multiplexer_channel_enum

表 3-260. 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
DMAMUX_MUXCH 12	DMAMUX请求路由通道12
DMAMUX_MUXCH 13	DMAMUX请求路由通道13
DMAMUX_MUXCH 14	DMAMUX请求路由通道14
DMAMUX_MUXCH 15	DMAMUX请求路由通道15

枚举 dmamux_generator_channel_enum

表 3-261. 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

成员名称	功能描述
DMAMUX_GENCH2	DMAMUX请求生成通道2
DMAMUX_GENCH3	DMAMUX请求生成通道3
DMAMUX_GENCH4	DMAMUX请求生成通道4
DMAMUX_GENCH5	DMAMUX请求生成通道5
DMAMUX_GENCH6	DMAMUX请求生成通道6
DMAMUX_GENCH7	DMAMUX请求生成通道7

枚举 dmamux_interrupt_enum

表 3-262. 枚举 dmamux_interrupt_enum

成员名称	功能描述
DMAMUX_INT_MU XCH0_SO	DMAMUX请求路由通道0同步溢出中断标志
DMAMUX_INT_MU XCH1_SO	DMAMUX请求路由通道1同步溢出中断标志
DMAMUX_INT_MU XCH2_SO	DMAMUX请求路由通道2同步溢出中断标志
DMAMUX_INT_MU XCH3_SO	DMAMUX请求路由通道3同步溢出中断标志
DMAMUX_INT_MU XCH4_SO	DMAMUX请求路由通道4同步溢出中断标志
DMAMUX_INT_MU XCH5_SO	DMAMUX请求路由通道5同步溢出中断标志
DMAMUX_INT_MU XCH6_SO	DMAMUX请求路由通道6同步溢出中断标志
DMAMUX_INT_MU XCH7_SO	DMAMUX请求路由通道7同步溢出中断标志
DMAMUX_INT_MU XCH8_SO	DMAMUX请求路由通道8同步溢出中断标志
DMAMUX_INT_MU XCH9_SO	DMAMUX请求路由通道9同步溢出中断标志
DMAMUX_INT_MU XCH10_SO	DMAMUX请求路由通道10同步溢出中断标志
DMAMUX_INT_MU XCH11_SO	DMAMUX请求路由通道11同步溢出中断标志
DMAMUX_INT_MU XCH12_SO	DMAMUX请求路由通道12同步溢出中断标志
DMAMUX_INT_MU XCH13_SO	DMAMUX请求路由通道13同步溢出中断标志
DMAMUX_INT_MU XCH14_SO	DMAMUX请求路由通道14同步溢出中断标志
DMAMUX_INT_MU	DMAMUX请求路由通道15同步溢出中断标志

成员名称	功能描述
XCH15_SO	
DMAMUX_INT_GE NCH0_TO	DMAMUX请求生成通道0触发溢出中断标志
DMAMUX_INT_GE NCH1_TO	DMAMUX请求生成通道1触发溢出中断标志
DMAMUX_INT_GE NCH2_TO	DMAMUX请求生成通道2触发溢出中断标志
DMAMUX_INT_GE NCH3_TO	DMAMUX请求生成通道3触发溢出中断标志
DMAMUX_INT_GE NCH4_TO	DMAMUX请求生成通道4触发溢出中断标志
DMAMUX_INT_GE NCH5_TO	DMAMUX请求生成通道5触发溢出中断标志
DMAMUX_INT_GE NCH6_TO	DMAMUX请求生成通道6触发溢出中断标志
DMAMUX_INT_GE NCH7_TO	DMAMUX请求生成通道7触发溢出中断标志

枚举 dmamux_flag_enum

表 3-263. 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 UXCH4_SO	DMAMUX请求路由通道4同步溢出标志
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

成员名称	功能描述
DMAMUX_FLAG_MUXCH10_SO	DMAMUX请求路由通道10同步溢出标志
DMAMUX_FLAG_MUXCH11_SO	DMAMUX请求路由通道11同步溢出标志
DMAMUX_FLAG_MUXCH12_SO	DMAMUX请求路由通道12同步溢出标志
DMAMUX_FLAG_MUXCH13_SO	DMAMUX请求路由通道13同步溢出标志
DMAMUX_FLAG_MUXCH14_SO	DMAMUX请求路由通道14同步溢出标志
DMAMUX_FLAG_MUXCH15_SO	DMAMUX请求路由通道15同步溢出标志
DMAMUX_FLAG_ENCH0_TO	DMAMUX请求生成通道0触发溢出标志
DMAMUX_FLAG_ENCH1_TO	DMAMUX请求生成通道1触发溢出标志
DMAMUX_FLAG_ENCH2_TO	DMAMUX请求生成通道2触发溢出标志
DMAMUX_FLAG_ENCH3_TO	DMAMUX请求生成通道3触发溢出标志
DMAMUX_FLAG_ENCH4_TO	DMAMUX请求生成通道4触发溢出标志
DMAMUX_FLAG_ENCH5_TO	DMAMUX请求生成通道5触发溢出标志
DMAMUX_FLAG_ENCH6_TO	DMAMUX请求生成通道6触发溢出标志
DMAMUX_FLAG_ENCH7_TO	DMAMUX请求生成通道7触发溢出标志

枚举 dmamux_interrupt_flag_enum

表 3-264. 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

成员名称	功能描述
G_MUXCH4_SO	
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_MUXCH12_SO	DMAMUX请求路由通道12同步溢出中断标志
DMAMUX_INT_FLAG_MUXCH13_SO	DMAMUX请求路由通道13同步溢出中断标志
DMAMUX_INT_FLAG_MUXCH14_SO	DMAMUX请求路由通道14同步溢出中断标志
DMAMUX_INT_FLAG_MUXCH15_SO	DMAMUX请求路由通道15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
DMAMUX_INT_FLAG_GENCH4_TO	DMAMUX请求生成通道4触发溢出中断标志
DMAMUX_INT_FLAG_GENCH5_TO	DMAMUX请求生成通道5触发溢出中断标志
DMAMUX_INT_FLAG_GENCH6_TO	DMAMUX请求生成通道6触发溢出中断标志

函数 dma_deinit

函数dma_deinit描述见下表：

表 3-265. 函数 dma_deinit

函数名称	dma_deinit
函数原型	void dma_deinit(uint32_t dma_periph, dma_channel_enum channelx);
功能描述	复位外设DMA通道x的所有寄存器
先决条件	-
被调用函数	-
输入参数{in}	
dma_periph	DMA外设
DMAx	(x=0,1)
输入参数{in}	
channelx	DMA通道，参考 表3-259. 枚举dma_channel_enum 。
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* deinitialize DMA0 channel 0 registers*/
```

```
dma_deinit(DMA0, DMA_CH0);
```

函数 dma_single_data_para_struct_init

函数dma_single_data_para_struct_init描述见下表：

表 3-266. 函数 dma_single_data_para_struct_init

函数名称	dma_single_data_para_struct_init
函数原型	void dma_single_data_para_struct_init(dma_single_data_parameter_struct *init_struct);
功能描述	将DMA单数据传输模式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}	
init_struct	初始化DMA通道所需的初始化数据，参考 表3-256. 结构体dma_single_data_parameter_struct 。
返回值	
-	-

例如：

```
/* initialize the parameters of DMA */
```

```
dma_single_data_parameter_struct init_struct;
```

```
dma_single_data_para_struct_init(&init_struct);
```

函数 dma_multi_data_para_struct_init

函数dma_multi_data_para_struct_init描述见下表：

表 3-267. 函数 dma_multi_data_para_struct_init

函数名称	dma_multi_data_para_struct_init
函数原型	void dma_multi_data_para_struct_init(dma_multi_data_parameter_struct *init_struct);
功能描述	将DMA多数据传输模式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}	
init_struct	初始化DMA通道所需的初始化数据，参考 表3-255. 结构体 dma_multi_data_parameter_struct 。
返回值	
-	-

例如：

```
/* initialize the parameters of DMA */
dma_multi_data_parameter_struct init_struct;
dma_multi_data_para_struct_init(&init_struct);
```

函数 dma_single_data_mode_init

函数dma_single_data_mode_init描述见下表：

表 3-268. 函数 dma_single_data_mode_init

函数名称	dma_single_data_mode_init
函数原型	void dma_single_data_mode_init(uint32_t dma_periph, dma_channel_enum channelx, dma_single_data_parameter_struct *init_struct);
功能描述	初始化外设DMA单数据传输模式
先决条件	-
被调用函数	-
输入参数{in}	
dma_periph	DMA外设
DMAx	(x=0,1)
输入参数{in}	
channelx	DMA通道，参考 表3-259. 枚举dma_channel_enum 。
输入参数{in}	

init_struct	用于初始化的结构体，结构体成员可以参考 表3-256. 结构体 dma_single_data_parameter_struct 。
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* initialize DMA1 channel7 */

dma_single_data_parameter_struct dma_init_struct;

dma_single_data_para_struct_init(&dma_init_struct);

dma_deinit(DMA1, DMA_CH7);

dma_init_struct.request = DMA_REQUEST_USART0_TX;

dma_init_struct.direction = DMA_MEMORY_TO_PERIPH;

dma_init_struct.memory0_addr = (uint32_t)txbuffer;

dma_init_struct.memory_inc = DMA_MEMORY_INCREASE_ENABLE;

dma_init_struct.periph_memory_width = DMA_PERIPH_WIDTH_8BIT;

dma_init_struct.number = ARRAYNUM(txbuffer);

dma_init_struct.periph_addr = USART0_TDATA_ADDRESS;

dma_init_struct.periph_inc = DMA_PERIPH_INCREASE_DISABLE;

dma_init_struct.priority = DMA_PRIORITY_ULTRA_HIGH;

dma_single_data_mode_init(DMA1, DMA_CH7, &dma_init_struct);
```

函数 dma_multi_data_mode_init

函数dma_multi_data_mode_init描述见下表：

表 3-269. 函数 dma_multi_data_mode_init

函数名称	dma_multi_data_mode_init
函数原型	void dma_multi_data_mode_init(uint32_t dma_periph, dma_channel_enum channelx, dma_multi_data_parameter_struct *init_struct);
功能描述	初始化外设DMA多数据传输模式
先决条件	-
被调用函数	-
输入参数{in}	
dma_periph	DMA外设
<i>DMAx</i>	(x=0,1)

输入参数{in}	
channelx	DMA通道，参考 表3-259. 枚举dma_channel_enum 。
输入参数{in}	
init_struct	用于初始化的结构体，结构体成员可以参考 表3-255. 结构体dma_multi_data_parameter_struct 。
输出参数{out}	
-	-
返回值	
-	-

例如：

```

/* initialize DMA1 channel 0 */

dma_multi_data_parameter_struct dma_init_parameter

dma_multi_data_para_struct_init(&dma_init_parameter);

dma_deinit(DMA1,DMA_CH0);

dma_init_parameter.periph_addr = (uint32_t)source;

dma_init_parameter.periph_width = DMA_PERIPH_WIDTH_16BIT;

dma_init_parameter.periph_inc = DMA_PERIPH_INCREASE_ENABLE;

dma_init_parameter.memory0_addr = (uint32_t)destination;

dma_init_parameter.memory_width = DMA_MEMORY_WIDTH_16BIT;

dma_init_parameter.memory_inc = DMA_MEMORY_INCREASE_ENABLE;

dma_init_parameter.memory_burst_width = DMA_MEMORY_BURST_4_BEAT;

dma_init_parameter.periph_burst_width = DMA_PERIPH_BURST_4_BEAT;

dma_init_parameter.critical_value = DMA_FIFO_2_WORD;

dma_init_parameter.circular_mode = DMA_CIRCULAR_MODE_DISABLE;

dma_init_parameter.direction = DMA_MEMORY_TO_MEMORY;

dma_init_parameter.number = BUFFER_SIZE;

dma_init_parameter.priority = DMA_PRIORITY_ULTRA_HIGH;

dma_multi_data_mode_init(DMA1,DMA_CH0,&dma_init_parameter);

```

函数 dma_periph_address_config

函数dma_periph_address_config描述见下表：

表 3-270. 函数 dma_periph_address_config

函数名称	dma_periph_address_config
函数原型	void dma_periph_address_config(uint32_t dma_periph, dma_channel_enum channelx, uint32_t address);
功能描述	配置DMA通道x传输的外设基地址
先决条件	相应通道使能位CHEN需为0
被调用函数	-
输入参数{in}	
dma_periph	DMA外设
DMAx	(x=0,1)
输入参数{in}	
channelx	DMA通道, 参考 表3-259. 枚举dma_channel_enum 。
输入参数{in}	
address	外设基地址, 0x00000000-0xFFFFFFFF
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* enable DMA0 channel 0 peripheral base address */
dma_periph_address_config(DMA0, DMA_CH0, 0x08000000);
```

函数 dma_memory_address_config

函数dma_memory_address_config描述见下表:

表 3-271. 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, uint8_t memory_flag, uint32_t address);
功能描述	配置DMA通道x传输的存储器基地址
先决条件	相应通道使能位CHEN需为0
被调用函数	-
输入参数{in}	
dma_periph	DMA外设
DMAx	(x=0,1)
输入参数{in}	
channelx	DMA通道, 参考 表3-259. 枚举dma_channel_enum 。
输入参数{in}	
memory_flag	DMA存储器选择
DMA_MEMORY_0	DMA存储器0
DMA_MEMORY_1	DMA存储器1

输入参数{in}	
address	存储器基地址, 0x00000000-0xFFFFFFFF
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* enable DMA0 channel 0 memory base address */
```

```
dma_memory_address_config(DMA0, DMA_CH0, DMA_MEMORY_0, 0x08000000);
```

函数 dma_transfer_number_config

函数dma_transfer_number_config描述见下表:

表 3-272. 函数 dma_transfer_number_config

函数名称	dma_transfer_number_config
函数原型	void dma_transfer_number_config(uint32_t dma_periph, dma_channel_enum channelx, uint32_t number);
功能描述	配置DMA通道x还有多少数据要传输
先决条件	相应通道使能位CHEN需为0
被调用函数	-
输入参数{in}	
dma_periph	DMA外设
DMAx	(x=0,1)
输入参数{in}	
channelx	DMA通道, 参考 表3-259. 枚举dma_channel_enum 。
输入参数{in}	
number	DMA待传输字节数, 0x00000000-0x0000FFFF
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* configure the number of remaining data to be transferred by the DMA0 channel 0 */
```

```
dma_transfer_number_config(DMA0, DMA_CH0, 0xFF);
```

函数 dma_transfer_number_get

函数dma_transfer_number_get描述见下表:

表 3-273. 函数 dma_transfer_number_get

函数名称	dma_transfer_number_get
函数原型	uint32_t dma_transfer_number_get(uint32_t dma_periph, dma_channel_enum channelx);
功能描述	获取DMA通道x还有多少数据要传输
先决条件	相应通道使能位CHEN需为0
被调用函数	-
输入参数{in}	
dma_periph	DMA外设
DMAx	(x=0,1)
输入参数{in}	
channelx	DMA通道, 参考表3-259. 枚举dma_channel_enum。
输出参数{out}	
-	-
返回值	
uint32_t	DMA待传输字节数, 0x00000000-0x0000FFFF

例如:

```
/* get the number of remaining data to be transferred by the DMA0 channel 0 */
```

```
uint32_t data_num;
```

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

函数 dma_priority_config

函数dma_priority_config描述见下表:

表 3-274. 函数 dma_priority_config

函数名称	dma_priority_config
函数原型	void dma_priority_config(uint32_t dma_periph, dma_channel_enum channelx, uint32_t priority);
功能描述	配置DMA通道x的传输软件优先级
先决条件	相应通道使能位CHEN需为0
被调用函数	-
输入参数{in}	
dma_periph	DMA外设
DMAx	(x=0,1)
输入参数{in}	
channelx	DMA通道, 参考表3-259. 枚举dma_channel_enum。
输入参数{in}	
priority	DMA通道软件优先级
DMA_PRIORITY_LOW	低优先级
DMA_PRIORITY_MEDI	中优先级

UM	
DMA_PRIORITY_HIGH	高优先级
DMA_PRIORITY_ULTRA_HIGH	极高优先级
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* configure priority level of DMA0 channel 0 */
```

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

函数 dma_memory_burst_beats_config

函数dma_memory_burst_beats_config描述见下表：

表 3-275. 函数 dma_memory_burst_beats_config

函数名称	dma_memory_burst_beats_config
函数原型	void dma_memory_burst_beats_config(uint32_t dma_periph, dma_channel_enum channelx, uint32_t mbeat);
功能描述	配置DMA通道x存储器增量突发传输节拍数
先决条件	相应通道使能位CHEN需为0
被调用函数	-
输入参数{in}	
dma_periph	DMA外设
DMAx	(x=0,1)
输入参数{in}	
channelx	DMA通道，参考 表3-259. 枚举dma_channel_enum 。
输入参数{in}	
mbeat	存储器增量突发类型
DMA_MEMORY_BURST_SINGLE	存储器单一传输
DMA_MEMORY_BURST_4_BEAT	存储器4拍增量突发传输
DMA_MEMORY_BURST_8_BEAT	存储器8拍增量突发传输
DMA_MEMORY_BURST_16_BEAT	存储器16拍增量突发传输
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* configure transfer burst beats of memory */
```

```
dma_memory_burst_beats_config(DMA0, DMA_CH0, DMA_MEMORY_BURST_4_BEAT);
```

函数 dma_periph_burst_beats_config

函数dma_periph_burst_beats_config描述见下表：

表 3-276. 函数 dma_periph_burst_beats_config

函数名称	dma_periph_burst_beats_config
函数原型	void dma_periph_burst_beats_config(uint32_t dma_periph, dma_channel_enum channelx, uint32_t pbeat);
功能描述	配置DMA通道x外设增量突发传输节拍数
先决条件	相应通道使能位CHEN需为0
被调用函数	-
输入参数{in}	
dma_periph	DMA外设
DMAx	(x=0,1)
输入参数{in}	
channelx	DMA通道，参考 表3-259. 枚举dma_channel_enum 。
输入参数{in}	
mbeat	外设增量突发类型
DMA_PERIPH_BURST_SINGLE	外设单一传输
DMA_PERIPH_BURST_4_BEAT	外设4拍增量突发传输
DMA_PERIPH_BURST_8_BEAT	外设8拍增量突发传输
DMA_PERIPH_BURST_16_BEAT	外设16拍增量突发传输
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* configure transfer burst beats of peripheral */
```

```
dma_periph_burst_beats_config(DMA0, DMA_CH0, DMA_PERIPH_BURST_4_BEAT);
```

函数 dma_memory_width_config

函数dma_memory_width_config描述见下表：

表 3-277. 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 msize);
功能描述	配置DMA通道x存储器传输数据宽度
先决条件	相应通道使能位CHEN需为0
被调用函数	-
输入参数{in}	
dma_periph	DMA外设
DMAx	(x=0,1)
输入参数{in}	
channelx	DMA通道, 参考 表3-259. 枚举dma_channel_enum 。
输入参数{in}	
msize	存储器传输数据宽度
DMA_MEMORY_WIDTH_H_8BIT	存储器传输数据8-bit宽度
DMA_MEMORY_WIDTH_H_16BIT	存储器传输数据16-bit宽度
DMA_MEMORY_WIDTH_H_32BIT	存储器传输数据32-bit宽度
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* configure transfer data size of memory */
```

```
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 3-278. 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 psize);
功能描述	配置DMA通道x外设传输数据宽度
先决条件	相应通道使能位CHEN需为0
被调用函数	-
输入参数{in}	
dma_periph	DMA外设
DMAx	(x=0,1)

输入参数{in}	
channelx	DMA通道, 参考 表3-259. 枚举dma_channel_enum 。
输入参数{in}	
psize	外设传输数据宽度
DMA_PERIPH_WIDTH_8BIT	外设传输数据8-bit宽度
DMA_PERIPH_WIDTH_16BIT	外设传输数据16-bit宽度
DMA_PERIPH_WIDTH_32BIT	外设传输数据32-bit宽度
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* configure transfer data size of peripheral */
```

```
dma_periph_width_config(DMA0, DMA_CH0, DMA_PERIPH_WIDTH_16BIT);
```

函数 dma_memory_address_generation_config

函数dma_memory_address_generation_config描述见下表:

表 3-279. 函数 dma_memory_address_generation_config

函数名称	dma_memory_address_generation_config
函数原型	void dma_memory_address_generation_config(uint32_t dma_periph, dma_channel_enum channelx, uint8_t generation_algorithm);
功能描述	配置DMA通道x存储器地址算法
先决条件	相应通道使能位CHEN需为0
被调用函数	-
输入参数{in}	
dma_periph	DMA外设
DMAx	(x=0,1)
输入参数{in}	
channelx	DMA通道, 参考 表3-259. 枚举dma_channel_enum 。
输入参数{in}	
generation_algorithm	存储器地址生成算法
DMA_MEMORY_INCR_EASE_ENABLE	存储器增量地址模式
DMA_MEMORY_INCR_EASE_DISABLE	存储器固定地址模式
输出参数{out}	
-	-

返回值	
-	-

例如：

```
/* configure memory address generation algorithm */
```

```
dma_memory_address_generation_config(DMA0, DMA_CH0, DMA_MEMORY_INCREMENTAL_ENABLE);
```

函数 dma_peripheral_address_generation_config

函数dma_peripheral_address_generation_config描述见下表：

表 3-280. 函数 dma_peripheral_address_generation_config

函数名称	dma_peripheral_address_generation_config
函数原型	void dma_peripheral_address_generation_config(uint32_t dma_periph, dma_channel_enum channelx, uint8_t generation_algorithm);
功能描述	配置DMA通道x外设地址算法
先决条件	相应通道使能位CHEN需为0
被调用函数	-
输入参数{in}	
dma_periph	DMA外设
DMAx	(x=0,1)
输入参数{in}	
channelx	DMA通道，参考 表3-259. 枚举dma_channel_enum 。
输入参数{in}	
generation_algorithm	外设地址生成算法
DMA_PERIPH_INCREMENTAL_ENABLE	外设增量地址模式
DMA_PERIPH_INCREMENTAL_DISABLE	外设固定地址模式
DMA_PERIPH_INCREMENTAL_FIX	外设地址增量固定为4
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* configure peripheral address generation algorithm */
```

```
dma_peripheral_address_generation_config(DMA0, DMA_CH0, DMA_PERIPH_INCREMENTAL_DISABLE);
```

函数 dma_circulation_enable

函数dma_circulation_enable描述见下表：

表 3-281. 函数 dma_circulation_enable

函数名称	dma_circulation_enable
函数原型	void dma_circulation_enable(uint32_t dma_periph, dma_channel_enum channelx);
功能描述	使能DMA循环传输模式
先决条件	相应通道使能位CHEN需为0
被调用函数	-
输入参数{in}	
dma_periph	DMA外设
DMAx	(x=0,1)
输入参数{in}	
channelx	DMA通道，参考 表3-259. 枚举dma_channel_enum 。
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* enable DMA0 channel 0 circulation mode */
dma_circulation_enable(DMA0, DMA_CH0);
```

函数 dma_circulation_disable

函数dma_circulation_disable描述见下表：

表 3-282. 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	失能DMA循环传输模式
先决条件	相应通道使能位CHEN需为0
被调用函数	-
输入参数{in}	
dma_periph	DMA外设
DMAx	(x=0,1)
输入参数{in}	
channelx	DMA通道，参考 表3-259. 枚举dma_channel_enum 。
输出参数{out}	
-	-

返回值	
-	-

例如：

```
/* disable DMA0 channel 0 circulation mode */
```

```
dma_circulation_disable(DMA0, DMA_CH0);
```

函数 dma_channel_enable

函数dma_channel_enable描述见下表：

表 3-283. 函数 dma_channel_enable

Function name	dma_channel_enable
Function prototype	void dma_channel_enable(uint32_t dma_periph, dma_channel_enum channelx);
Function descriptions	使能DMA通道x
Precondition	-
The called functions	-
输入参数{in}	
dma_periph	DMA外设
DMAx	(x=0,1)
输入参数{in}	
channelx	DMA通道，参考 表3-259. 枚举dma_channel_enum 。
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* enable DMA0 channel 0 */
```

```
dma_channel_enable(DMA0, DMA_CH0);
```

函数 dma_channel_disable

函数dma_channel_disable描述见下表：

表 3-284. 函数 dma_channel_disable

函数名称	dma_channel_disable
函数原型	void dma_channel_disable(uint32_t dma_periph, dma_channel_enum channelx);
功能描述	禁能DMA通道x
先决条件	-
被调用函数	-

输入参数{in}	
dma_periph	DMA外设
<i>DMAx</i>	(x=0,1)
输入参数{in}	
channelx	DMA通道, 参考 表3-259. 枚举dma_channel_enum 。
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* disable DMA channel 0 */
```

```
dma_channel_disable(DMA0, DMA_CH0);
```

函数 dma_transfer_direction_config

函数dma_transfer_direction_config描述见下表:

表 3-285. 函数 dma_transfer_direction_config

函数名称	dma_transfer_direction_config
函数原型	void dma_transfer_direction_config(uint32_t dma_periph, dma_channel_enum channelx, uint8_t direction);
功能描述	配置DMA通道x数据传输方向
先决条件	相应通道使能位CHEN需为0
被调用函数	-
输入参数{in}	
dma_periph	DMA外设
<i>DMAx</i>	(x=0,1)
输入参数{in}	
channelx	DMA通道, 参考 表3-259. 枚举dma_channel_enum 。
输入参数{in}	
direction	指定数据传输的方向
<i>DMA_PERIPH_TO_MEMORY</i>	从外设读取并写入存储器
<i>DMA_MEMORY_TO_PERIPH</i>	从存储器读取并写入外设
<i>DMA_MEMORY_TO_MEMORY</i>	从存储器读取并写入存储器
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* configure DMA0 channel 0 memory to memory mode */
```

```
dma_memory_to_memory_enable(DMA0, DMA_CH0, DMA_MEMORY_TO_MEMORY);
```

函数 dma_switch_buffer_mode_config

函数dma_switch_buffer_mode_config描述见下表：

表 3-286. 函数 dma_switch_buffer_mode_config

函数名称	dma_switch_buffer_mode_config
函数原型	void dma_switch_buffer_mode_config(uint32_t dma_periph, dma_channel_enum channelx, uint32_t memory1_addr, uint32_t memory_select);
功能描述	配置DMA通道x存储器传输缓冲区
先决条件	相应通道使能位CHEN需为0
被调用函数	-
输入参数{in}	
dma_periph	DMA外设
DMAx	(x=0,1)
输入参数{in}	
channelx	DMA通道，参考 表3-259. 枚举dma_channel_enum 。
输入参数{in}	
memory1_addr	存储器1基地址，0x00000000-0xFFFFFFFF
输入参数{in}	
memory_select	DMA存储器选择
DMA_MEMORY_0	DMA存储器0
DMA_MEMORY_1	DMA存储器1
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* configure DMA0 channel0 switch buffer mode */
```

```
dma_switch_buffer_mode_config(DMA0, DMA_CH0, 0x20000000, DMA_MEMORY_0);
```

函数 dma_using_memory_get

函数dma_using_memory_get描述见下表：

表 3-287. 函数 dma_using_memory_get

函数名称	dma_using_memory_get
函数原型	uint32_t dma_using_memory_get(uint32_t dma_periph,

	<code>dma_channel_enum channelx</code>);
功能描述	获取DMA通道x存储器传输缓冲区
先决条件	相应通道使能位CHEN需为0
被调用函数	-
输入参数{in}	
<code>dma_periph</code>	DMA外设
<code>DMAx</code>	(x=0,1)
输入参数{in}	
<code>channelx</code>	DMA通道, 参考 表3-259. 枚举dma_channel_enum 。
输出参数{out}	
-	-
返回值	
<code>uint32_t</code>	使用的存储器
<code>DMA_MEMORY_0</code>	DMA存储器0
<code>DMA_MEMORY_1</code>	DMA存储器1

例如:

```
/* get DMA using memory */
uint32_t memory_index;

memory_index = dma_using_memory_get(DMA0, DMA_CH0);
```

函数 dma_switch_buffer_mode_enable

函数dma_switch_buffer_mode_enable描述见下表:

表 3-288. 函数 dma_switch_buffer_mode_enable

函数名称	<code>dma_switch_buffer_mode_enable</code>
函数原型	<code>void dma_switch_buffer_mode_enable(uint32_t dma_periph, dma_channel_enum channelx);</code>
功能描述	使能DMA通道x存储切换模式
先决条件	相应通道使能位CHEN需为0
被调用函数	-
输入参数{in}	
<code>dma_periph</code>	DMA外设
<code>DMAx</code>	(x=0,1)
输入参数{in}	
<code>channelx</code>	DMA通道, 参考 表3-259. 枚举dma_channel_enum 。
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* enable DMA switch buffer mode */
```

```
dma_switch_buffer_mode_enable();
```

函数 dma_switch_buffer_mode_disable

函数dma_switch_buffer_mode_disable描述见下表：

表 3-289. 函数 dma_switch_buffer_mode_disable

函数名称	dma_switch_buffer_mode_disable
函数原型	void dma_switch_buffer_mode_disable(uint32_t dma_periph, dma_channel_enum channelx);
功能描述	禁能DMA通道x存储切换模式
先决条件	相应通道使能位CHEN需为0
被调用函数	-
输入参数{in}	
dma_periph	DMA外设
DMAx	(x=0,1)
输入参数{in}	
channelx	DMA通道，参考 表3-259. 枚举dma_channel_enum 。
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* disable DMA switch buffer mode */
```

```
dma_switch_buffer_mode_disable();
```

函数 dma_fifo_status_get

函数dma_fifo_status_get描述见下表：

表 3-290. 函数 dma_fifo_status_get

函数名称	dma_fifo_status_get
函数原型	uint32_t dma_fifo_status_get(uint32_t dma_periph, dma_channel_enum channelx);
功能描述	获取DMA通道xFIFO状态
先决条件	-
被调用函数	-
输入参数{in}	
dma_periph	DMA外设
DMAx	(x=0,1)
输入参数{in}	

channelx	DMA通道，参考 表3-259. 枚举dma_channel_enum 。
输出参数{out}	
-	-
返回值	
uint32_t	存储在FIFO中的字数
<i>DMA_FIFO_CNT_NO_DATA</i>	无数据
<i>DMA_FIFO_CNT_1_WORD</i>	1个字
<i>DMA_FIFO_CNT_2_WORD</i>	2个字
<i>DMA_FIFO_CNT_3_WORD</i>	3个字
<i>DMA_FIFO_CNT_EMPTY</i>	FIFO空
<i>DMA_FIFO_CNT_FULL</i>	FIFO满

例如：

```
/* get DMA channel 0 transfer number */
uint32_t number = 0;
number = dma_transfer_number_get(DMA0, DMA_CH0);
```

函数 dma_flag_get

函数dma_flag_get描述见下表：

表 3-291. 函数 dma_flag_get

函数名称	dma_flag_get
函数原型	FlagStatus dma_flag_get(uint32_t dma_periph, dma_channel_enum channelx, uint32_t flag);
功能描述	获取DMA通道x标志
先决条件	-
被调用函数	-
输入参数{in}	
dma_periph	DMA外设
<i>DMAx</i>	(x=0,1)
输入参数{in}	
channelx	DMA通道，参考 表3-259. 枚举dma_channel_enum 。
输入参数{in}	
flag	指定获取的标志位
<i>DMA_FLAG_FEE</i>	FIFO错误与异常标志
<i>DMA_FLAG_SDE</i>	单数据传输模式异常标志

<i>DMA_FLAG_TAE</i>	传输错误标志
<i>DMA_FLAG_HTF</i>	半传输完成标志
<i>DMA_FLAG_FTF</i>	传输完成标志
输出参数{out}	
-	-
返回值	
FlagStatus	SET或RESET

例如：

```
/* get DMA0 channel 0 flag */
```

```
FlagStatus flag = RESET;
```

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

函数 dma_flag_clear

函数dma_flag_clear描述见下表：

表 3-292. 函数 dma_flag_clear

函数名称	dma_flag_clear
函数原型	void dma_flag_clear(uint32_t dma_periph, dma_channel_enum channelx, uint32_t flag);
功能描述	清除DMA通道x标志
先决条件	-
被调用函数	-
输入参数{in}	
dma_periph	DMA外设
<i>DMAx</i>	(x=0,1)
输入参数{in}	
channelx	DMA通道，参考 表3-259. 枚举dma_channel_enum 。
输入参数{in}	
flag	指定获取的标志位
<i>DMA_FLAG_FEE</i>	FIFO错误与异常标志
<i>DMA_FLAG_SDE</i>	单数据传输模式异常标志
<i>DMA_FLAG_TAE</i>	传输错误标志
<i>DMA_FLAG_HTF</i>	半传输完成标志
<i>DMA_FLAG_FTF</i>	传输完成标志
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* clear DMA0 channel 0 flag */
```

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

函数 dma_interrupt_enable

函数dma_interrupt_enable描述见下表：

表 3-293. 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 interrupt);
功能描述	使能DMA通道x中断
先决条件	-
被调用函数	-
输入参数{in}	
dma_periph	DMA外设
DMAx	(x=0,1)
输入参数{in}	
channelx	DMA通道，参考 表3-259. 枚举dma_channel_enum 。
输入参数{in}	
interrupt	DMA中断
DMA_INT_SDE	单数据传输模式异常中断
DMA_INT_TAE	传输错误中断
DMA_INT_HTF	半传输完成中断
DMA_INT_FTF	传输完成中断
DMA_INT_FEE	FIFO异常中断
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* enable DMA0 channel 0 interrupt */
```

```
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 3-294. 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 interrupt);

功能描述	禁能DMA通道x中断
先决条件	-
被调用函数	-
输入参数{in}	
dma_periph	DMA外设
<i>DMAx</i>	(x=0,1)
输入参数{in}	
channelx	DMA通道, 参考 表3-259. 枚举dma_channel_enum 。
输入参数{in}	
interrupt	DMA中断
<i>DMA_INT_SDE</i>	单数据传输模式异常中断
<i>DMA_INT_TAE</i>	传输错误中断
<i>DMA_INT_HTF</i>	半传输完成中断
<i>DMA_INT_FTF</i>	传输完成中断
<i>DMA_INT_FEE</i>	FIFO异常中断
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* disable DMA0 channel 0 interrupt */
```

```
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 3-295. 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 int_flag);
功能描述	获取DMA通道x中断标志
先决条件	-
被调用函数	-
输入参数{in}	
dma_periph	DMA外设
<i>DMAx</i>	(x=0,1)
输入参数{in}	
channelx	DMA通道, 参考 表3-259. 枚举dma_channel_enum 。
输入参数{in}	
int_flag	指定获取的中断标志
<i>DMA_INT_FLAG_FEE</i>	FIFO错误与异常中断标志

<i>DMA_INT_FLAG_SDE</i>	单数据传输模式异常中断标志
<i>DMA_INT_FLAG_TAE</i>	传输错误中断标志
<i>DMA_INT_FLAG_HTF</i>	半传输完成中断标志
<i>DMA_INT_FLAG_FTF</i>	传输完成中断标志
输出参数{out}	
-	-
返回值	
FlagStatus	SET或RESET

例如：

```
/* get DMA0 channel 3 interrupt flag */
```

```
FlagStatus flag = RESET;
```

```
flag = dma_interrupt_flag_get(DMA0, DMA_CH3, DMA_INT_FTF);
```

函数 dma_interrupt_flag_clear

函数dma_interrupt_flag_clear描述见下表：

表 3-296. 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 int_flag);
功能描述	清除DMA通道x中断标志
先决条件	-
被调用函数	-
输入参数{in}	
dma_periph	DMA外设
<i>DMAx</i>	(x=0,1)
输入参数{in}	
channelx	DMA通道，参考 表3-259. 枚举dma_channel_enum 。
输入参数{in}	
int_flag	指定获取的中断标志
<i>DMA_INT_FLAG_FEE</i>	FIFO错误与异常中断标志
<i>DMA_INT_FLAG_SDE</i>	单数据传输模式异常中断标志
<i>DMA_INT_FLAG_TAE</i>	传输错误中断标志
<i>DMA_INT_FLAG_HTF</i>	半传输完成中断标志
<i>DMA_INT_FLAG_FTF</i>	传输完成中断标志
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* clear DMA0 channel 3 interrupt flag */
```

```
dma_interrupt_flag_clear(DMA0, DMA_CH3, DMA_INT_FTF);
```

函数 dmamux_sync_struct_para_init

函数dmamux_sync_struct_para_init描述见下表：

表 3-297. 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	-
被调用函数	-
输入参数{in}	
-	-
输出参数{out}	
init_struct	初始化DMAMUX请求多路复用器通道同步模式需要的数据，参考 表3-257. 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 3-298. 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	-
被调用函数	-
输入参数{in}	
channelx	DMAMUX请求路由通道，参考 表3-260. 枚举dmamux_multiplexer_channel_enum 。
输入参数{in}	
init_struct	初始化DMAMUX请求多路复用器通道同步模式需要的数据，参考 表3-257.

	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 3-299. 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	-
被调用函数	-
输入参数{in}	
channelx	DMAMUX请求路由通道，参考 表3-260. 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 3-300. 函数 **dmamux_synchronization_disable**

函数名称	dmamux_synchronization_disable
函数原型	void dmamux_synchronization_disable(dmamux_mux_channel_enum channelx);
功能描述	禁能DMAMUX同步模式
先决条件	-
被调用函数	-
输入参数{in}	
channelx	DMAMUX请求路由通道, 参考 表3-260. 枚举 dmamux_mux_channel_enum 。
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 3-301. 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	-
被调用函数	-
输入参数{in}	
channelx	DMAMUX请求路由通道, 参考 表3-260. 枚举 dmamux_mux_channel_enum 。
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 3-302. 函数 dmamux_event_generation_disable

函数名称	dmamux_event_generation_disable
函数原型	void dmamux_event_generation_disable(dmamux_multiplexer_channel_enum channelx);
功能描述	禁能DMAMUX事件输出
先决条件	-
被调用函数	-
输入参数{in}	
channelx	DMAMUX请求路由通道，参考 表3-260. 枚举 dmamux_multiplexer_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 3-303. 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	-
被调用函数	-
输入参数{in}	
-	-
输出参数{out}	
init_struct	初始化DMAMUX生成通道需要的数据，参考 表3-258. 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 3-304. 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	-
被调用函数	-
输入参数{in}	
channelx	DMAMUX请求生成通道，参考 表3-261. 枚举 dmamux_generator_channel_enum 。
输入参数{in}	
init_struct	初始化DMAMUX生成通道需要的数据，参考 表3-258. 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 3-305. 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	-
被调用函数	-
输入参数{in}	
channelx	DMAMUX请求生成通道，参考 表3-261. 枚举 dmamux_generator_channel_enum 。
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* enable DMAMUX request generator channel 0 */
```

```
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 3-306. 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	-
被调用函数	-
输入参数{in}	
channelx	DMAMUX请求生成通道，参考 表3-261. 枚举 dmamux_generator_channel_enum 。
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* disable DMAMUX request generator channel 0 */
```

```
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 3-307. 函数 dmamux_synchronization_polarity_config

函数名称	dmamux_synchronization_polarity_config
函数原型	void dmamux_synchronization_polarity_config(dmamux_mux_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	-
被调用函数	-
输入参数{in}	
channelx	DMAMUX请求路由通道，参考 表3-260. 枚举 dmamux_mux_channel_enum 。
输入参数{in}	
polarity	同步输入有效边沿
DMAMUX_SYNC_NO_EVENT	不检测边沿
DMAMUX_SYNC_RISING	上升沿
DMAMUX_SYNC_FALLING	下降沿
DMAMUX_SYNC_RISING_FALLING	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 3-308. 函数 `dmamux_request_forward_number_config`

函数名称	<code>dmamux_request_forward_number_config</code>
函数原型	<code>void dmamux_request_forward_number_config(dmamux_mux_channel_enum channelx, uint32_t number);</code>
功能描述	配置DMAMUX通道x要传输多少个DMA请求
先决条件	-
被调用函数	-
输入参数{in}	
channelx	DMAMUX请求路由通道，参考 表3-260. 枚举 <code>dmamux_mux_channel_enum</code> 。
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 3-309. 函数 `dmamux_sync_id_config`

函数名称	<code>dmamux_sync_id_config</code>
函数原型	<code>void dmamux_sync_id_config(dmamux_mux_channel_enum channelx, uint32_t id);</code>
功能描述	配置DMAMUX同步输入标识
先决条件	-
被调用函数	-
输入参数{in}	
channelx	DMAMUX请求路由通道，参考 表3-260. 枚举 <code>dmamux_mux_channel_enum</code> 。
输入参数{in}	
id	同步输入标识
<code>DMAMUX_SYNC_EXTI0</code>	同步输入信号为EXTI0
<code>DMAMUX_SYNC_EXTI1</code>	同步输入信号为EXTI1
<code>DMAMUX_SYNC_EXTI2</code>	同步输入信号为EXTI2

2	
DMAMUX_SYNC_EXTI	同步输入信号为EXTI3
3	
DMAMUX_SYNC_EXTI	同步输入信号为EXTI4
4	
DMAMUX_SYNC_EXTI	同步输入信号为EXTI5
5	
DMAMUX_SYNC_EXTI	同步输入信号为EXTI6
6	
DMAMUX_SYNC_EXTI	同步输入信号为EXTI7
7	
DMAMUX_SYNC_EXTI	同步输入信号为EXTI8
8	
DMAMUX_SYNC_EXTI	同步输入信号为EXTI9
9	
DMAMUX_SYNC_EXTI	同步输入信号为EXTI10
10	
DMAMUX_SYNC_EXTI	同步输入信号为EXTI11
11	
DMAMUX_SYNC_EXTI	同步输入信号为EXTI12
12	
DMAMUX_SYNC_EXTI	同步输入信号为EXTI13
13	
DMAMUX_SYNC_EXTI	同步输入信号为EXTI14
14	
DMAMUX_SYNC_EXTI	同步输入信号为EXTI15
15	
DMAMUX_SYNC_EVT X_OUT0	同步输入信号为Evt0_out
DMAMUX_SYNC_EVT X_OUT1	同步输入信号为Evt1_out
DMAMUX_SYNC_EVT X_OUT2	同步输入信号为Evt2_out
DMAMUX_SYNC_EVT X_OUT3	同步输入信号为Evt3_out
DMAMUX_SYNC_EVT X_OUT4	同步输入信号为Evt4_out
DMAMUX_SYNC_EVT X_OUT5	同步输入信号为Evt5_out
DMAMUX_SYNC_EVT X_OUT6	同步输入信号为Evt6_out
DMAMUX_SYNC_RTC _WAKEUP	同步输入信号为RTC唤醒

DMAMUX_SYNC_CMP0_OUTPUT	同步输入信号为CMP0输出
DMAMUX_SYNC_I2C0_WAKEUP	同步输入信号为I2C0唤醒
DMAMUX_SYNC_I2C1_WAKEUP	同步输入信号为I2C1唤醒
DMAMUX_SYNC_I2C2_WAKEUP	同步输入信号为I2C2唤醒
DMAMUX_SYNC_I2C3_WAKEUP	同步输入信号为I2C3唤醒
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 3-310. 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	-
被调用函数	-
输入参数{in}	
channelx	DMAMUX请求路由通道，参考 表3-260. 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

<i>ERATOR3</i>	
<i>DMA_REQUEST_GEN</i> <i>ERATOR4</i>	DMAMUX请求生成通道4请求
<i>DMA_REQUEST_GEN</i> <i>ERATOR5</i>	DMAMUX请求生成通道5请求
<i>DMA_REQUEST_GEN</i> <i>ERATOR6</i>	DMAMUX请求生成通道6请求
<i>DMA_REQUEST_GEN</i> <i>ERATOR7</i>	DMAMUX请求生成通道7请求
<i>DMA_REQUEST_ADC</i> <i>0</i>	DMAMUX ADC0请求
<i>DMA_REQUEST_ADC</i> <i>1</i>	DMAMUX ADC1请求
<i>DMA_REQUEST_TIME</i> <i>R0_CH0</i>	DMAMUX TIMER0 CH0请求
<i>DMA_REQUEST_TIME</i> <i>R0_CH1</i>	DMAMUX TIMER0 CH1请求
<i>DMA_REQUEST_TIME</i> <i>R0_CH2</i>	DMAMUX TIMER0 CH2请求
<i>DMA_REQUEST_TIME</i> <i>R0_CH3</i>	DMAMUX TIMER0 CH3请求
<i>DMA_REQUEST_TIME</i> <i>R0_MCH0</i>	DMAMUX TIMER0 MCH0请求
<i>DMA_REQUEST_TIME</i> <i>R0_MCH1</i>	DMAMUX TIMER0 MCH1请求
<i>DMA_REQUEST_TIME</i> <i>R0_MCH2</i>	DMAMUX TIMER0 MCH2请求
<i>DMA_REQUEST_TIME</i> <i>R0_MCH3</i>	DMAMUX TIMER0 MCH3请求
<i>DMA_REQUEST_TIME</i> <i>R0_UP</i>	DMAMUX TIMER0 UP请求
<i>DMA_REQUEST_TIME</i> <i>R0_TRG</i>	DMAMUX TIMER0 TRG请求
<i>DMA_REQUEST_TIME</i> <i>R0_CMT</i>	DMAMUX TIMER0 CMT请求
<i>DMA_REQUEST_TIME</i> <i>R1_CH0</i>	DMAMUX TIMER1 CH0请求
<i>DMA_REQUEST_TIME</i> <i>R1_CH1</i>	DMAMUX TIMER1 CH1请求
<i>DMA_REQUEST_TIME</i> <i>R1_CH2</i>	DMAMUX TIMER1 CH2请求
<i>DMA_REQUEST_TIME</i> <i>R1_CH3</i>	DMAMUX TIMER1 CH3请求

<i>DMA_REQUEST_TIME</i> <i>R1_UP</i>	DMAMUX TIMER1 UP请求
<i>DMA_REQUEST_TIME</i> <i>R1_TRG</i>	DMAMUX TIMER1 TRG请求
<i>DMA_REQUEST_TIME</i> <i>R2_CH0</i>	DMAMUX TIMER2 CH0请求
<i>DMA_REQUEST_TIME</i> <i>R2_CH1</i>	DMAMUX TIMER2 CH1请求
<i>DMA_REQUEST_TIME</i> <i>R2_CH2</i>	DMAMUX TIMER2 CH2请求
<i>DMA_REQUEST_TIME</i> <i>R2_CH3</i>	DMAMUX TIMER2 CH3请求
<i>DMA_REQUEST_TIME</i> <i>R2_UP</i>	DMAMUX TIMER2 UP请求
<i>DMA_REQUEST_TIME</i> <i>R2_TRG</i>	DMAMUX TIMER2 TRG请求
<i>DMA_REQUEST_TIME</i> <i>R3_CH0</i>	DMAMUX TIMER3 CH0请求
<i>DMA_REQUEST_TIME</i> <i>R3_CH1</i>	DMAMUX TIMER3 CH1请求
<i>DMA_REQUEST_TIME</i> <i>R3_CH2</i>	DMAMUX TIMER3 CH2请求
<i>DMA_REQUEST_TIME</i> <i>R3_CH3</i>	DMAMUX TIMER3 CH3请求
<i>DMA_REQUEST_TIME</i> <i>R3_CH3</i>	DMAMUX TIMER3 CH3请求
<i>DMA_REQUEST_TIME</i> <i>R3_TRG</i>	DMAMUX TIMER3 TRG请求
<i>DMA_REQUEST_TIME</i> <i>R3_UP</i>	DMAMUX TIMER3 UP请求
<i>DMA_REQUEST_I2C0</i> <i>_RX</i>	DMAMUX I2C0 RX请求
<i>DMA_REQUEST_I2C0</i> <i>_TX</i>	DMAMUX I2C0 TX请求
<i>DMA_REQUEST_I2C1</i> <i>_RX</i>	DMAMUX I2C1 RX请求
<i>DMA_REQUEST_I2C1</i> <i>_TX</i>	DMAMUX I2C1 TX请求
<i>DMA_REQUEST_SPI0</i> <i>_RX</i>	DMAMUX SPI0 RX请求
<i>DMA_REQUEST_SPI0</i> <i>_TX</i>	DMAMUX SPI0 TX请求
<i>DMA_REQUEST_SPI1</i>	DMAMUX SPI1 RX请求

<i>_RX</i>	
<i>DMA_REQUEST_SPI1_TX</i>	DMAMUX SPI1 TX请求
<i>DMA_REQUEST_USART0_RX</i>	DMAMUX USART0 RX请求
<i>DMA_REQUEST_USART0_TX</i>	DMAMUX USART0 TX请求
<i>DMA_REQUEST_USART1_RX</i>	DMAMUX USART1 RX请求
<i>DMA_REQUEST_USART1_TX</i>	DMAMUX USART1 TX请求
<i>DMA_REQUEST_USART2_RX</i>	DMAMUX USART2 RX请求
<i>DMA_REQUEST_USART2_TX</i>	DMAMUX USART2 TX请求
<i>DMA_REQUEST_TIMER7_CH0</i>	DMAMUX TIMER7 CH0请求
<i>DMA_REQUEST_TIMER7_CH1</i>	DMAMUX TIMER7 CH1请求
<i>DMA_REQUEST_TIMER7_CH2</i>	DMAMUX TIMER7 CH2请求
<i>DMA_REQUEST_TIMER7_CH3</i>	DMAMUX TIMER7 CH3请求
<i>DMA_REQUEST_TIMER7_MCH0</i>	DMAMUX TIMER7 MCH0请求
<i>DMA_REQUEST_TIMER7_MCH1</i>	DMAMUX TIMER7 MCH1请求
<i>DMA_REQUEST_TIMER7_MCH2</i>	DMAMUX TIMER7 MCH2请求
<i>DMA_REQUEST_TIMER7_MCH3</i>	DMAMUX TIMER7 MCH3请求
<i>DMA_REQUEST_TIMER7_UP</i>	DMAMUX TIMER7 UP请求
<i>DMA_REQUEST_TIMER7_TRG</i>	DMAMUX TIMER7 TRG请求
<i>DMA_REQUEST_TIMER7_CMT</i>	DMAMUX TIMER7 CMT请求
<i>DMA_REQUEST_TIMER4_CH0</i>	DMAMUX TIMER4 CH0请求
<i>DMA_REQUEST_TIMER4_CH1</i>	DMAMUX TIMER4 CH1请求
<i>DMA_REQUEST_TIMER4_CH2</i>	DMAMUX TIMER4 CH2请求

<i>DMA_REQUEST_TIME</i> <i>R4_CH3</i>	DMAMUX TIMER4 CH3请求
<i>DMA_REQUEST_TIME</i> <i>R4_UP</i>	DMAMUX TIMER4 UP请求
<i>DMA_REQUEST_TIME</i> <i>R4_TRG</i>	DMAMUX TIMER4 TRG请求
<i>DMA_REQUEST_SPI2</i> <i>_RX</i>	DMAMUX SPI2 RX请求
<i>DMA_REQUEST_SPI2</i> <i>_TX</i>	DMAMUX SPI2 TX请求
<i>DMA_REQUEST_UAR</i> <i>T3_RX</i>	DMAMUX UART3 RX请求
<i>DMA_REQUEST_UAR</i> <i>T3_TX</i>	DMAMUX UART3 TX请求
<i>DMA_REQUEST_UAR</i> <i>T4_RX</i>	DMAMUX UART4 RX请求
<i>DMA_REQUEST_UAR</i> <i>T4_TX</i>	DMAMUX UART4 TX请求
<i>DMA_REQUEST_DAC</i> <i>_CH0</i>	DMAMUX DAC CH0请求
<i>DMA_REQUEST_DAC</i> <i>_CH1</i>	DMAMUX DAC CH1请求
<i>DMA_REQUEST_TIME</i> <i>R5_UP</i>	DMAMUX TIMER5 UP请求
<i>DMA_REQUEST_TIME</i> <i>R6_UP</i>	DMAMUX TIMER6 UP请求
<i>DMA_REQUEST_USA</i> <i>RT5_RX</i>	DMAMUX USART5 RX请求
<i>DMA_REQUEST_USA</i> <i>RT5_TX</i>	DMAMUX USART5 TX请求
<i>DMA_REQUEST_I2C2</i> <i>_RX</i>	DMAMUX I2C2 RX请求
<i>DMA_REQUEST_I2C2</i> <i>_TX</i>	DMAMUX I2C2 TX请求
<i>DMA_REQUEST_UAR</i> <i>T6_RX</i>	DMAMUX UART6 RX请求
<i>DMA_REQUEST_UAR</i> <i>T6_TX</i>	DMAMUX UART6 TX请求
<i>DMA_REQUEST_UAR</i> <i>T7_RX</i>	DMAMUX UART7 RX请求
<i>DMA_REQUEST_UAR</i> <i>T7_TX</i>	DMAMUX UART7 TX请求
<i>DMA_REQUEST_SPI3</i>	DMAMUX SPI3 RX请求

<i>_RX</i>	
<i>DMA_REQUEST_SPI3_TX</i>	DMAMUX SPI3 TX请求
<i>DMA_REQUEST_SPI4_RX</i>	DMAMUX SPI4 RX请求
<i>DMA_REQUEST_SPI4_TX</i>	DMAMUX SPI4 TX请求
<i>DMA_REQUEST_HPDF_FLT0</i>	DMAMUX HPDF FLT0请求
<i>DMA_REQUEST_HPDF_FLT1</i>	DMAMUX HPDF FLT1请求
<i>DMA_REQUEST_HPDF_FLT2</i>	DMAMUX HPDF FLT2请求
<i>DMA_REQUEST_HPDF_FLT3</i>	DMAMUX HPDF FLT3请求
<i>DMA_REQUEST_TIMER14_CH0</i>	DMAMUX TIMER14 CH0请求
<i>DMA_REQUEST_TIMER14_CH1</i>	DMAMUX TIMER14 CH1请求
<i>DMA_REQUEST_TIMER14_MCH0</i>	DMAMUX TIMER14 MCH0请求
<i>DMA_REQUEST_TIMER14_UP</i>	DMAMUX TIMER14 UP请求
<i>DMA_REQUEST_TIMER14_TRG</i>	DMAMUX TIMER14 TRG请求
<i>DMA_REQUEST_TIMER14_CMT</i>	DMAMUX TIMER14 CMT请求
<i>DMA_REQUEST_TIMER15_CH0</i>	DMAMUX TIMER15 CH0请求
<i>DMA_REQUEST_TIMER15_MCH0</i>	DMAMUX TIMER15 MCH0请求
<i>DMA_REQUEST_TIMER15_UP</i>	DMAMUX TIMER15 UP请求
<i>DMA_REQUEST_TIMER16_CH0</i>	DMAMUX TIMER16 CH0请求
<i>DMA_REQUEST_TIMER16_MCH0</i>	DMAMUX TIMER16 MCH0请求
<i>DMA_REQUEST_TIMER16_UP</i>	DMAMUX TIMER16 TRG请求
<i>DMA_REQUEST_ADC2</i>	DMAMUX ADC2请求
<i>DMA_REQUEST_FAC_READ</i>	DMAMUX FAC READ请求

<i>DMA_REQUEST_FAC _WRITE</i>	DMAMUX FAC WRITE 请求
<i>DMA_REQUEST_TMU _INPUT</i>	DMAMUX TMU INPUT 请求
<i>DMA_REQUEST_TMU _OUTPUT</i>	DMAMUX TMU OUTPUT 请求
<i>DMA_REQUEST_TIME R22_CH0</i>	DMAMUX TIMER22 CH0 请求
<i>DMA_REQUEST_TIME R22_CH1</i>	DMAMUX TIMER22 CH1 请求
<i>DMA_REQUEST_TIME R22_CH2</i>	DMAMUX TIMER22 CH2 请求
<i>DMA_REQUEST_TIME R22_CH3</i>	DMAMUX TIMER22 CH3 请求
<i>DMA_REQUEST_TIME R22_UP</i>	DMAMUX TIMER22 UP 请求
<i>DMA_REQUEST_TIME R22_TRG</i>	DMAMUX TIMER22 TRG 请求
<i>DMA_REQUEST_TIME R23_CH0</i>	DMAMUX TIMER23 CH0 请求
<i>DMA_REQUEST_TIME R23_CH1</i>	DMAMUX TIMER23 CH1 请求
<i>DMA_REQUEST_TIME R23_CH2</i>	DMAMUX TIMER23 CH2 请求
<i>DMA_REQUEST_TIME R23_CH3</i>	DMAMUX TIMER23 CH3 请求
<i>DMA_REQUEST_TIME R23_UP</i>	DMAMUX TIMER23 UP 请求
<i>DMA_REQUEST_TIME R23_TRG</i>	DMAMUX TIMER23 TRG 请求
<i>DMA_REQUEST_TIME R40_CH0</i>	DMAMUX TIMER40 CH0 请求
<i>DMA_REQUEST_TIME R40_MCH0</i>	DMAMUX TIMER40 MCH0 请求
<i>DMA_REQUEST_TIME R40_CMT</i>	DMAMUX TIMER40 CMT 请求
<i>DMA_REQUEST_TIME R40_UP</i>	DMAMUX TIMER40 UP 请求
<i>DMA_REQUEST_TIME R41_CH0</i>	DMAMUX TIMER41 CH0 请求
<i>DMA_REQUEST_TIME R41_MCH0</i>	DMAMUX TIMER41 MCH0 请求
<i>DMA_REQUEST_TIME</i>	DMAMUX TIMER41 CMT 请求

<i>R41_CMT</i>	
<i>DMA_REQUEST_TIME</i> <i>R41_UP</i>	DMAMUX TIMER41 UP请求
<i>DMA_REQUEST_TIME</i> <i>R42_CH0</i>	DMAMUX TIMER42 CH0请求
<i>DMA_REQUEST_TIME</i> <i>R42_MCH0</i>	DMAMUX TIMER42 MCH0请求
<i>DMA_REQUEST_TIME</i> <i>R42_CMT</i>	DMAMUX TIMER42 CMT请求
<i>DMA_REQUEST_TIME</i> <i>R42_UP</i>	DMAMUX TIMER42 UP请求
<i>DMA_REQUEST_TIME</i> <i>R43_CH0</i>	DMAMUX TIMER43 CH0请求
<i>DMA_REQUEST_TIME</i> <i>R43_MCH0</i>	DMAMUX TIMER43 MCH0请求
<i>DMA_REQUEST_TIME</i> <i>R43_CMT</i>	DMAMUX TIMER43 CMT请求
<i>DMA_REQUEST_TIME</i> <i>R43_UP</i>	DMAMUX TIMER43 UP请求
<i>DMA_REQUEST_TIME</i> <i>R44_CH0</i>	DMAMUX TIMER44 CH0请求
<i>DMA_REQUEST_TIME</i> <i>R44_MCH0</i>	DMAMUX TIMER44 MCH0请求
<i>DMA_REQUEST_TIME</i> <i>R44_CMT</i>	DMAMUX TIMER44 CMT请求
<i>DMA_REQUEST_TIME</i> <i>R44_UP</i>	DMAMUX TIMER44 UP请求
<i>DMA_REQUEST_TIME</i> <i>R50_UP</i>	DMAMUX TIMER50 UP请求
<i>DMA_REQUEST_TIME</i> <i>R51_UP</i>	DMAMUX TIMER51 UP请求
<i>DMA_REQUEST_SPI5</i> <i>_RX</i>	DMAMUX SPI5 RX请求
<i>DMA_REQUEST_SPI5</i> <i>_TX</i>	DMAMUX SPI5 TX请求
<i>DMA_REQUEST_I2C3</i> <i>_RX</i>	DMAMUX I2C3 RX请求
<i>DMA_REQUEST_I2C3</i> <i>_TX</i>	DMAMUX I2C3 TX请求
<i>DMA_REQUEST_CAN</i> <i>0</i>	DMAMUX CAN0请求
<i>DMA_REQUEST_CAN</i> <i>1</i>	DMAMUX CAN1请求

<code>DMA_REQUEST_CAN2</code>	DMAMUX CAN2请求
<code>DMA_REQUEST_TIME_R40_CH1</code>	DMAMUX TIMER40 CH1 请求
<code>DMA_REQUEST_TIME_R40_TRG</code>	DMAMUX TIMER40 TRG 请求
<code>DMA_REQUEST_TIME_R41_CH1</code>	DMAMUX TIMER41 CH1 请求
<code>DMA_REQUEST_TIME_R41_TRG</code>	DMAMUX TIMER41 TRG 请求
<code>DMA_REQUEST_TIME_R42_CH1</code>	DMAMUX TIMER42 CH1 请求
<code>DMA_REQUEST_TIME_R42_TRG</code>	DMAMUX TIMER42 TRG 请求
<code>DMA_REQUEST_TIME_R43_CH1</code>	DMAMUX TIMER43 CH1 请求
<code>DMA_REQUEST_TIME_R43_TRG</code>	DMAMUX TIMER43 TRG 请求
<code>DMA_REQUEST_TIME_R44_CH1</code>	DMAMUX TIMER44 CH1 请求
<code>DMA_REQUEST_TIME_R44_TRG</code>	DMAMUX TIMER44 TRG 请求
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 3-311. 函数 `dmamux_trigger_polarity_config`

函数名称	<code>dmamux_trigger_polarity_config</code>
函数原型	<code>void dmamux_trigger_polarity_config(dmamux_generator_channel_enum channelx, uint32_t polarity);</code>
功能描述	配置DMAMUX触发输入的有效边沿
先决条件	-
被调用函数	-
输入参数{in}	

channelx	DMAMUX请求生成通道，参考 表3-261. 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
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 3-312. 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	-
被调用函数	-
输入参数{in}	
channelx	DMAMUX请求生成通道，参考 表3-261. 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 3-313. 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	-
被调用函数	-
输入参数{in}	
channelx	DMAMUX请求生成通道，参考 表3-261. 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

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_TRIGGER_ EVT_OUT0	触发输入为Evt0_out
DMAMUX_TRIGGER_ EVT_OUT1	触发输入为Evt1_out
DMAMUX_TRIGGER_ EVT_OUT2	触发输入为Evt2_out
DMAMUX_TRIGGER_ EVT_OUT3	触发输入为Evt3_out
DMAMUX_TRIGGER_ EVT_OUT4	触发输入为Evt4_out
DMAMUX_TRIGGER_ EVT_OUT5	触发输入为Evt5_out
DMAMUX_TRIGGER_ EVT_OUT6	触发输入为Evt6_out
DMAMUX_TRIGGER_ EVT_OUT7	触发输入为Evt7_out
DMAMUX_TRIGGER_ RTC_WAKEUP	触发输入为唤醒
DMAMUX_TRIGGER_ CMP0_OUTPUT	触发输入为CMP0输出
DMAMUX_TRIGGER_ CMP1_OUTPUT	触发输入为CMP1输出
DMAMUX_TRIGGER_I 2C0_WAKEUP	触发输入为I2C0唤醒
DMAMUX_TRIGGER_I 2C1_WAKEUP	触发输入为I2C1唤醒
DMAMUX_TRIGGER_I 2C2_WAKEUP	触发输入为I2C2唤醒
DMAMUX_TRIGGER_I 2C3_WAKEUP	触发输入为I2C3唤醒
DMAMUX_TRIGGER_I 2C0_INT_EVENT	触发输入为I2C0中断事件
DMAMUX_TRIGGER_I 2C1_INT_EVENT	触发输入为I2C1中断事件

DMAMUX_TRIGGER_I 2C2_INT_EVENT	触发输入为I2C2中断事件
DMAMUX_TRIGGER_I 2C3_INT_EVENT	t触发输入为I2C3中断事件
DMAMUX_TRIGGER_ ADC2_INT	ADC2中断
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 3-314. 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DMMUXA通道x标志位状态
先决条件	-
被调用函数	-
输入参数{in}	
flag	标志类型，参考表3-263. 枚举dmamux_flag_enum。
输出参数{out}	
-	-
返回值	
FlagStatus	SET或RESET

例如：

```
/* get DMAMUX flag */
```

```
FlagStatus flag = RESET;
```

```
flag = dmamux_flag_get(DMAMUX_FLAG_GENCH0_TO);
```

函数 dmamux_flag_clear

函数dmamux_flag_clear描述见下表：

表 3-315. 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	-
被调用函数	-
输入参数{in}	
flag	标志类型，参考 表3-263. 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 3-316. 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	-
被调用函数	-
输入参数{in}	
interrupt	指定使能哪个中断，参考 表3-262. 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 3-317. 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	-
被调用函数	-
输入参数{in}	
interrupt	指定使能哪个中断，参考 表3-262. 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 3-318. 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	-
被调用函数	-
输入参数{in}	
int_flag	标志类型，参考 表3-264. 枚举dmamux_interrupt_flag_enum 。
输出参数{out}	
-	-
返回值	
FlagStatus	SET或RESET

例如：

```
/* get DMAMUX interrupt flag */
```

```
FlagStatus flag = RESET;
```

```
flag = dmamux_interrupt_flag_get(DMAMUX_INT_FLAG_MUXCH0_SO);
```

函数 dmamux_interrupt_flag_clear

函数dmamux_interrupt_flag_clear描述见下表：

表 3-319. 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	-
被调用函数	-
输入参数{in}	
int_flag	标志类型，参考 表3-264. 枚举dmamux_interrupt_flag_enum 。
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* clear DMAMUX interrupt flag */
```

```
dmamux_interrupt_flag_clear(DMAMUX_INT_FLAG_MUXCH0_SO);
```

3.11. EDOUT

EDOUT是MCU中的编码器分频输出控制器，它把从编码器获取到的位置信息，以A相、B相和Z相脉冲的方式输出。章节[3.11.1](#)描述了EDOUT的寄存器列表，章节[3.11.2](#)对EDOUT库函数进行说明。

3.11.1. 外设寄存器描述

EDOUT寄存器列表如下表所示：

表 3-320. EDOUT 寄存器

寄存器名称	寄存器描述
EDOUT_CTL	控制寄存器
EDOUT_ENABLE	使能寄存器
EDOUT_LOC	位置寄存器
EDOUT_OCNT	输出计数器寄存器
EDOUT_LCNT	位置计数器寄存器
EDOUT_ZCR	Z相配置寄存器

3.11.2. 外设库函数说明

EDOUT库函数列表如下表所示：

表 3-321. EDOUT 库函数

库函数名称	库函数描述
edout_deinit	复位EDOUT

库函数名称	库函数描述
edout_init	初始化EDOUT
edout_enable	使能EDOUT
edout_disable	禁用EDOUT
edout_polarity_config	设置B相激活极性
edout_max_location_value_config	设置一次旋转的最大位置
edout_output_counter_update	更新输出计数器，用于设置下一个更新周期的相位差和边沿数
edout_current_location_config	设置当前位置
edout_current_location_get	获取当前位置
edout_z_output_mode_config	设置Z相输出模式
edout_z_output_start_loc_and_width_config	设置Z相输出起始位置和宽度

函数 edout_deinit

函数edout_deinit描述见下表：

表 3-322. 函数 edout_deinit

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

函数 edout_init

函数 edout_init 描述见下表：

表 3-323. 函数 edout_init

函数名称	edout_init
函数原型	void edout_init(uint32_t pol, uint32_t max_loc, uint32_t cur_loc);
功能描述	初始化EDOUT
先决条件	无
被调用函数	无

输入参数{in}	
pol	B相输出信号的极性选择
输入参数{in}	
max_loc	(max_loc+1)必须为4的倍数，取值范围15~65535。(例如max_loc = 0x000F: 最大位置值是16 (16=4*4))
输入参数{in}	
cur_loc	当前位置，0~locmax (locmax是EDOUT_LOC寄存器的LOCMAX位域值)
输出参数{out}	
-	-
返回值	
-	-

例如:

```
#define EDOUT_MAX_LOC          99
#define EDOUT_CUR_LOC          50
#define EDOUT_B_PHASE_POL      EDOUT_POL_POSITIVE

/* initialize EDOUT */
edout_init(EDOUT_B_PHASE_POL, EDOUT_MAX_LOC, EDOUT_CUR_LOC);
```

函数 edout_enable

函数edout_enable描述见下表:

表 3-324. 函数 edout_enable

函数名称	edout_enable
函数原形	void edout_enable(void);
功能描述	使能EDOUT
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 EDOUT */
edout_enable();
```

函数 edout_disable

函数edout_disable描述见下表:

表 3-325. 函数 edout_disable

函数名称	edout_disable
函数原形	void edout_disable(void);
功能描述	禁用EDOUT
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 EDOUT */
edout_disable();
```

函数 edout_polarity_config

函数edout_polarity_config描述见下表：

表 3-326. 函数 edout_polarity_config

函数名称	edout_polarity_config
函数原形	void edout_polarity_config(uint32_t pol);
功能描述	设置B相激活极性
先决条件	-
被调用函数	-
输入参数{in}	
pol	B相激活极性
EDOUT_POL_POSITIVE	激活极性为正向
EDOUT_POL_NEGATIVE	激活极性为反向
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* configure B-phase active polarity */
edout_polarity_config(EDOUT_POL_POSITIVE);
```

函数 **edout_max_location_value_config**

函数edout_max_location_value_config描述见下表:

表 3-327. 函数 **edout_max_location_value_config**

函数名称	edout_max_location_value_config
函数原形	void edout_max_location_value_config(uint32_t max_loc);
功能描述	设置一次旋转的最大位置
先决条件	-
被调用函数	-
输入参数{in}	
max_loc	(max_loc+1)必须为4的倍数，取值范围为15~65535。(例如max_loc = 0x000F: 最大位置值是16 (16=4*4))
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* set the maximum location value of EDOUT */
edout_max_location_value_config(99);
```

函数 **edout_output_counter_update**

函数edout_output_counter_update描述见下表:

表 3-328. 函数 **edout_output_counter_update**

函数名称	edout_output_counter_update
函数原形	void edout_output_counter_update(int16_t num_edges, uint16_t phase_diff);
功能描述	更新输出计数器，用于设置下一个更新周期的相位差和边沿数
先决条件	-
被调用函数	-
输入参数{in}	
num_edges	边沿计数，取值范围为-32768~32767，正值表示顺时针旋转，负值表示逆时针旋转
输入参数{in}	
phase_diff	相位差，取值范围为2~65535，单位为PCLK
输出参数{out}	
-	-
返回值	

例如:

```
/* set the edge count and the phase difference value between the phase-A and phase-B */
```

```
edout_output_counter_update(5, 0x7530);
```

函数 edout_current_location_config

函数edout_current_location_config描述见下表：

表 3-329. 函数 edout_current_location_config

函数名称	edout_current_location_config
函数原形	void edout_current_location_config(uint32_t cur_loc);
功能描述	设置当前位置
先决条件	-
被调用函数	-
输入参数{in}	
cur_loc	当前位置，0~locmax (locmax是EDOUT_LOC寄存器的LOCMAX位域值)
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* set the current location */
edout_current_location_config(90);
```

函数 edout_current_location_get

函数edout_current_location_get描述见下表：

表 3-330. 函数 edout_current_location_get

函数名称	edout_current_location_get
函数原形	uint16_t edout_current_location_get(void);
功能描述	获取当前位置
先决条件	-
被调用函数	-
输入参数{in}	
-	-
输出参数{out}	
-	-
返回值	
uint16_t	当前位置值，0~locmax (locmax是EDOUT_LOC寄存器的LOCMAX位域值)

例如：

```
uint16_t cur_loc;

/* get the current location */
cur_loc = edout_current_location_get();
```

函数 edout_z_output_mode_config

函数edout_z_output_mode_config描述见下表：

表 3-331. 函数 edout_z_output_mode_config

函数名称	edout_z_output_mode_config
函数原形	void edout_z_output_mode_config(uint32_t mode);
功能描述	设置Z相输出模式
先决条件	-
被调用函数	-
输入参数{in}	
mode	Z相输出模式
EDOUT_Z_OUTPUT T_MODE0	当前位置决定输出
EDOUT_Z_OUTPUT T_MODE1	边沿序号决定输出
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* configure Z-phase output mode */
edout_z_output_mode_config(EDOUT_Z_OUTPUT_MODE0);
```

函数 edout_z_output_start_loc_and_width_config

函数edout_z_output_start_loc_and_width_config描述见下表：

表 3-332. 函数 edout_z_output_start_loc_and_width_config

函数名称	edout_z_output_start_loc_and_width_config
函数原形	void edout_z_output_start_loc_and_width_config(uint32_t start_loc, uint32_t width);
功能描述	设置Z相输出起始位置和宽度
先决条件	-
被调用函数	-
输入参数{in}	
start_loc	Z相输出起始位置
输入参数{in}	
width	Z相输出宽度
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* configure Z-phase output start location and width */
edout_z_output_start_loc_and_width_config(15, 10);
```

3.12. EFUSE

EFUSE作为一种非易失性存储单元存储了一些必需的系统参数，其中每一个比特位只允许从0改写为1。章节[3.12.1](#)描述了EFUSE的寄存器列表，章节[3.12.2](#)对EFUSE库函数进行说明。

3.12.1. 外设寄存器描述

EFUSE寄存器列表如下表所示：

表 3-333. EFUSE 寄存器

寄存器名称	寄存器描述
EFUSE_CTL	控制寄存器
EFUSE_ADDR	地址寄存器
EFUSE_STAT	状态寄存器
EFUSE_STATC	状态清除寄存器
EFUSE_USER_CTL	用户控制寄存器
EFUSE_MCU_RSV	MCU 保留寄存器
EFUSE_DP _x (_x = 0,1)	调试密钥寄存器 _x (_x = 0,1)
EFUSE_AES_KEY _x (_x = 0...3)	AES 密钥寄存器 _x (_x = 0...3)
EFUSE_USER_DATA _x (_x = 0...3)	用户数据寄存器 _x (_x = 0...3)

3.12.2. 外设库函数说明

EFUSE库函数列表如下表所示：

表 3-334. EFUSE 库函数

库函数名称	库函数描述
efuse_read	读 EFUSE 值
efuse_write	写 EFUSE 值
efuse_user_control_write	写所有的用户控制段
efuse_mcu_reserved_write	写所有的 MCU 保留段
efuse_dp_write	写所有的调试密钥
efuse_aes_key_write	写所有的 AES 密钥
efuse_user_data_write	写所有的用户数据段
efuse_aes_key_crc_get	获取 AES 密钥的 8 位 CRC 计算结果值

库函数名称	库函数描述
efuse_monitor_program_voltage_enable	使能监控编程电压功能
efuse_monitor_program_voltage_disable	失能监控编程电压功能
efuse_monitor_program_voltage_get	获取监控编程电压功能
efuse_ldo_ready_get	获取熔丝 LDO 准备完成信号
efuse_flag_get	获取 EFUSE 标志位
efuse_flag_clear	清除 EFUSE 标志位
efuse_interrupt_enable	使能 EFUSE 中断
efuse_interrupt_disable	失能 EFUSE 中断
efuse_interrupt_flag_get	获取 EFUSE 中断标志位
efuse_interrupt_flag_clear	清除 EFUSE 中断标志位

枚举类型 efuse_system_para_size_enum

表 3-335. 枚举类型 efuse_system_para_size_enum

成员名称	功能描述
USER_CTL_MCU_RESERVED_SIZE	用户控制段大小, MCU保留段大小
DP_SIZE	调试密钥大小
AES_KEY_SIZE	AES密钥大小
USER_DATA_SIZE	用户数据段大小

枚举类型 efuse_system_para_index_enum

表 3-336. 枚举类型 efuse_system_para_index_enum

成员名称	功能描述
USER_CTL_IDX	用户控制段索引
MCU_RESERVED_IDX	MCU保留段索引
DP_IDX	调试密钥索引
AES_KEY_IDX	AES密钥索引
USER_DATA_IDX	用户数据段索引

枚举类型 efuse_state_enum

表 3-337. 枚举类型 efuse_state_enum

成员名称	功能描述
EFUSE_READY	操作完成
EFUSE_BUSY	操作进行中
EFUSE_IAERR	非法访问错误
EFUSE_TOERR	超时错误

枚举类型 `efuse_interrupt_flag_enum`表 3-338. 枚举类型 `efuse_interrupt_flag_enum`

成员名称	功能描述
EFUSE_INT_FLAG_ILLEGAL_ACCESS_ERR	非法访问错误中断标志
EFUSE_INT_FLAG_PROGRAM_COMPLETE	写操作完成中断标志
EFUSE_INT_FLAG_READ_COMPLETE	读操作完成中断标志
EFUSE_INT_FLAG_PROGRAM_VOLTAGE_ERROR	编程电压设置错误中断标志

函数 `efuse_read`

函数 `efuse_read` 描述见下表：

表 3-339. 函数 `Function efuse_read`

函数名称	<code>efuse_read</code>
函数原型	<code>ErrStatus efuse_read(uint32_t ef_addr, uint32_t size, uint32_t buf[]);</code>
功能描述	读 EFUSE 值
先决条件	-
被调函数	-
输入参数{in}	
ef_addr	要读取的系统参数的起始地址
<code>USER_CTL_EFADDR</code>	用户控制段起始地址
<code>MCU_RESERVED_EFADDR</code>	MCU 保留段起始地址
<code>DP_EFADDR</code>	调试密钥起始地址
<code>USER_DATA_EFADDR</code>	用户数据段起始地址
输入参数{in}	
size	要读取的系统参数的字段大小（字节）
<code>USER_CTL_SIZE</code>	用户控制段大小
<code>MCU_RESERVED_SIZE</code>	MCU 保留段大小
<code>DP_SIZE</code>	调试密钥大小
<code>USER_DATA_SIZE</code>	用户数据段大小
输入参数{in}	
buf	存储从 EFUSE 字段读取出数据的数组
输出参数{out}	
-	-
返回值	
ErrStatus	ERROR 或 SUCCESS

例如：

```
/* read user data from EFUSE macro to registers */
```

```
ErrStatus error_status = efuse_read(USER_DATA_EFADDR, USER_DATA_SIZE, (uint32_t *)&rd_data);;
```

函数 efuse_write

函数efuse_write描述见下表：

表 3-340. 函数 efuse_write

函数名称	efuse_write
函数原型	ErrStatus efuse_write(uint32_t ef_addr, uint32_t size, uint8_t *buf);
功能描述	写 EFUSE
先决条件	-
被调用函数	-
输入参数{in}	
pgm_addr	要写的 EFUSE 地址，地址不能超过 384，且需是 8 的整数倍
输入参数{in}	
size	要写入的 EFUSE 字段大小（字节）
输入参数{in}	
buf	存储写入 EFUSE 字段数据的数组
输出参数{out}	
-	-
返回值	
ErrStatus	ERROR 或 SUCCESS

例如：

```
/* write EFUSE USER DATA*/
```

```
uint8_t buffer[8] = { 0x11, 0x22, 0x33, 0x44, 0x55, 0x66, 0x77, 0x88 };
```

```
ErrStatus flag = ERROR;
```

```
flag = efuse_write (0x100, 8, buffer);
```

函数 efuse_user_control_write

函数efuse_user_control_write描述见下表：

表 3-341. 函数 efuse_user_control_write

函数名称	efuse_user_control_write
函数原型	ErrStatus efuse_user_control_write(uint8_t *buf);
功能描述	写所有的用户控制段
先决条件	-
被调用函数	efuse_write

输入参数{in}	
buf	存储写入 EFUSE 字段数据的数组
输出参数{out}	
-	-
返回值	
ErrStatus	ERROR 或 SUCCESS

例如：

```
/* write EFUSE user control parameter */
uint8_t buffer[4] = { 0x11, 0x22, 0x33, 0x44 };
ErrStatus flag = ERROR;
flag = efuse_user_control_write(buffer);
```

函数 efuse_mcu_reserved_write

函数efuse_mcu_reserved_write描述见下表：

表 3-342. 函数 efuse_write

函数名称	efuse_mcu_reserved_write
函数原型	ErrStatus efuse_mcu_reserved_write(uint8_t *buf);
功能描述	写所有的 MCU 保留段
先决条件	-
被调用函数	efuse_write
输入参数{in}	
buf	存储写入 EFUSE 字段数据的数组
输出参数{out}	
-	-
返回值	
ErrStatus	ERROR 或 SUCCESS

例如：

```
/* write EFUSE MCU reserved parameter */
uint8_t buffer[4] = { 0x11, 0x22, 0x33, 0x44 };
ErrStatus flag = ERROR;
flag = efuse_mcu_reserved_write(buffer);
```

函数 efuse_dp_write

函数efuse_dp_write描述见下表：

表 3-343. 函数 efuse_write

函数名称	efuse_dp_write
函数原型	ErrStatus efuse_dp_write(uint8_t *buf);
功能描述	写所有的调试密钥
先决条件	-
被调用函数	efuse_write
输入参数{in}	
buf	存储写入 EFUSE 字段数据的数组
输出参数{out}	
-	-
返回值	
ErrStatus	ERROR 或 SUCCESS

例如：

```
/* write EFUSE debug password parameter */
```

```
uint8_t buffer[8] = { 0x11, 0x22, 0x33, 0x44, 0x55, 0x66, 0x77, 0x88 };
```

```
ErrStatus flag = ERROR;
```

```
flag = efuse_dp_write(buffer);
```

函数 efuse_aes_key_write

函数 efuse_aes_key_write 描述见下表：

表 3-344. 函数 efuse_write

函数名称	efuse_aes_key_write
函数原型	ErrStatus efuse_aes_key_write(uint8_t *buf);
功能描述	写所有的 AES 密钥
先决条件	-
被调用函数	efuse_write
输入参数{in}	
buf	存储写入 EFUSE 字段数据的数组
输出参数{out}	
-	-
返回值	
ErrStatus	ERROR 或 SUCCESS

例如：

```
/* write EFUSE AES key parameter */
```

```
uint8_t buffer[16] = { 0x11, 0x22, 0x33, 0x44, 0x55, 0x66, 0x77, 0x88, 0x99, 0xAA, 0xBB, 0xCC, 0xDD, 0xEE, 0xFF, 0x11 };
```

```
ErrStatus flag = ERROR;
```

```
flag = efuse_aes_key_write(buffer);
```

函数 efuse_user_data_write

函数efuse_user_data_write描述见下表：

表 3-345. 函数 efuse_write

函数名称	efuse_user_data_write
函数原型	ErrStatus efuse_user_data_write(uint8_t *buf);
功能描述	写所有的用户数据段
先决条件	-
被调用函数	efuse_write
输入参数{in}	
buf	存储写入 EFUSE 字段数据的数组
输出参数{out}	
-	-
返回值	
ErrStatus	ERROR 或 SUCCESS

例如：

```
/* write EFUSE user data parameter */
```

```
uint8_t buffer[16] = { 0x11, 0x22, 0x33, 0x44, 0x55, 0x66, 0x77, 0x88, 0x99, 0xAA, 0xBB, 0xCC, 0xDD, 0xEE, 0xFF, 0x11 };
```

```
ErrStatus flag = ERROR;
```

```
flag = efuse_user_data_write(buffer);
```

函数 efuse_aes_key_crc_get

函数efuse_aes_key_crc_get描述见下表：

表 3-346. 函数 efuse_aes_key_crc_get

函数名称	efuse_aes_key_crc_get
函数原型	uint8_t efuse_aes_key_crc_get(void);
功能描述	获取 AES 密钥的 8 位 CRC 计算结果值
先决条件	-
被调用函数	-
输入参数{in}	
-	-
输出参数{out}	
-	-
返回值	
uint8_t	AES 密钥的 8 位 CRC 计算结果值

例如:

```
/* get 8-bits CRC calculation result value of AES key */
```

```
uint8_t crc_value = 0;
```

```
crc_value = efuse_aes_key_crc_get();
```

函数 efuse_monitor_program_voltage_enable

函数efuse_monitor_program_voltage_enable描述见下表

表 3-347. 函数 efuse_monitor_program_voltage_enable

函数名称	efuse_monitor_program_voltage_enable
函数原型	void efuse_monitor_program_voltage_enable(void);
功能描述	使能监控编程电压功能
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 monitor program voltage function */
```

```
efuse_monitor_program_voltage_enable();
```

函数 efuse_monitor_program_voltage_disable

函数efuse_monitor_program_voltage_disable描述见下表

表 3-348. 函数 efuse_monitor_program_voltage_disable

函数名称	efuse_monitor_program_voltage_disable
函数原型	void efuse_monitor_program_voltage_disable(void);
功能描述	失能监控编程电压功能
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 monitor program voltage function */
```

```
efuse_monitor_program_voltage_disable();
```

函数 efuse_monitor_program_voltage_get

函数efuse_monitor_program_voltage_get描述见下表

表 3-349. 函数 efuse_monitor_program_voltage_get

函数名称	efuse_monitor_program_voltage_get
函数原型	void efuse_monitor_program_voltage_get(void);
功能描述	获取监控编程电压功能
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 monitor program voltage function */
```

```
FlagStatus flag_sts = efuse_monitor_program_voltage_get();
```

函数 efuse_ldo_ready_get

函数efuse_ldo_ready_get描述见下表

表 3-350. 函数 efuse_ldo_ready_get

函数名称	efuse_ldo_ready_get
函数原型	FlagStatus efuse_ldo_ready_get(void);
功能描述	获取熔丝 LDO 准备完成信号
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 ldo ready signal */
```

```
FlagStatus flag_sts = efuse_ldo_ready_get();
```

函数 efuse_flag_get

函数efuse_flag_get描述见下表：

表 3-351. 函数 efuse_flag_get

函数名称	efuse_flag_get
函数原型	FlagStatus efuse_flag_get(uint32_t efuse_flag);
功能描述	获取EFUSE标志位
先决条件	-
被调用函数	-
输入参数{in}	
efuse_flag	EFUSE状态标志
EFUSE_FLAG_ILLEGAL_ACCESS_ERR	非法访问错误标志位
EFUSE_FLAG_PROG_RAM_COMPLETE	写操作完成标志位
EFUSE_FLAG_READ_COMPLETE	读操作完成标志位
EFUSE_FLAG_PROG_RAM_VOLTAGE_ERR	编程电压设置错误标志位
输出参数{out}	
-	-
返回值	
FlagStatus	SET或RESET

例如：

```
/* get the EFUSE illegal access error flag */
```

```
FlagStatus flag_value;
```

```
flag_value = efuse_flag_get(EFUSE_FLAG_ILLEGAL_ACCESS_ERR);
```

函数 efuse_flag_clear

函数efuse_flag_clear描述见下表：

表 3-352. 函数 efuse_flag_clear

函数名称	efuse_flag_clear
函数原型	void efuse_flag_clear(uint32_t efuse_cflag);
功能描述	清除EFUSE标志位
先决条件	-
被调用函数	-

输入参数{in}	
efuse_periph	EFUSE状态标志
<i>EFUSE_FLAG_ILLEGAL_ACCESS_ERR_CLR</i>	非法访问错误标志位
<i>EFUSE_FLAG_PROGRAM_COMPLETE_CLR</i>	清除写操作完成标志位
<i>EFUSE_FLAG_READ_COMPLETE_CLR</i>	清除读操作完成标志位
<i>EFUSE_FLAG_PROGRAM_VOLTAGE_ERR_CLR</i>	清除编程电压设置错误标志位
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* clear the EFUSE illegal access error flag */
```

```
efuse_flag_clear(EFUSE_FLAG_ILLEGAL_ACCESS_ERR_CLR);
```

函数 efuse_interrupt_enable

函数efuse_interrupt_enable描述见下表：

表 3-353. 函数 efuse_interrupt_enable

函数名称	efuse_interrupt_enable
函数原型	void efuse_interrupt_enable(uint32_t source);
功能描述	使能EFUSE中断
先决条件	-
被调用函数	-
输入参数{in}	
source	EFUSE状态标志
<i>EFUSE_INT_ILLEGAL_ACCESS_ERR</i>	使能非法访问错误中断
<i>EFUSE_INT_PROGRAM_COMPLETE</i>	使能写操作完成中断
<i>EFUSE_INT_READ_COMPLETE</i>	使能读操作完成中断
<i>EFUSE_INT_PROGRAM_VOLTAGE_ERR</i>	使能编程电压设置错误中断
输出参数{out}	
-	-

返回值	
-	-

例如：

```
/* enable EFUSE illegal access error interrupt */
```

```
efuse_interrupt_enable(EFUSE_INT_ILLEGAL_ACCESS_ERR);
```

函数 efuse_interrupt_disable

函数efuse_interrupt_disable描述见下表：

表 3-354. 函数 efuse_interrupt_disable

函数名称	efuse_interrupt_disable
函数原型	void efuse_interrupt_disable(uint32_t source);
功能描述	失能EFUSE中断
先决条件	-
被调用函数	-
输入参数{in}	
source	specifies an interrupt to disable
EFUSE_INT_ILLEGAL_ACCESS_ERR	失能非法访问错误中断
EFUSE_INT_PROGRAM_COMPLETE	失能写操作完成中断
EFUSE_INT_READ_COMPLETE	失能读操作完成中断
EFUSE_INT_PROGRAM_VOLTAGE_ERR	失能编程电压设置错误中断
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* disable EFUSE illegal access error interrupt */
```

```
efuse_interrupt_disable(EFUSE_INT_ILLEGAL_ACCESS_ERR);
```

函数 efuse_interrupt_flag_get

函数efuse_interrupt_flag_get描述见下表：

表 3-355. 函数 efuse_interrupt_flag_get

函数名称	efuse_interrupt_flag_get
函数原型	FlagStatus efuse_interrupt_flag_get(uint32_t int_flag);

功能描述	获取EFUSE中断标志位
先决条件	-
被调用函数	-
输入参数{in}	
int_flag	中断标志位
EFUSE_INT_FLAG_ILLEGAL_ACCESS_ERR	非法访问错误中断标志
EFUSE_INT_FLAG_PROGRAM_COMPLETE	写操作完成中断标志
EFUSE_INT_FLAG_READ_COMPLETE	读操作完成中断标志
EFUSE_INT_FLAG_PROGRAM_VOLTAGE_ERROR	编程电压设置错误中断标志
输出参数{out}	
-	-
返回值	
FlagStatus	SET或RESET

例如：

```
/* get the EFUSE illegal access error interrupt bits*/
```

```
FlagStatus flag_value;
```

```
flag_value = efuse_interrupt_flag_get(EFUSE_INT_FLAG_ILLEGAL_ACCESS_ERR);
```

函数 efuse_interrupt_flag_clear

函数efuse_interrupt_flag_clear描述见下表：

表 3-356. 函数 efuse_interrupt_flag_clear

函数名称	efuse_interrupt_flag_clear
函数原型	void efuse_interrupt_flag_clear(uint32_t efuse_periph, uint32_t int_flag);
功能描述	清除EFUSE中断标志位
先决条件	-
被调用函数	-
输入参数{in}	
efuse_flag	中断标志清除位
EFUSE_INT_FLAG_ILLEGAL_ACCESS_ERR_CLR	清除非法访问错误中断标志
EFUSE_INT_FLAG_PROGRAM_COMPLETE_CLR	清除写操作完成中断标志

EFUSE_INT_FLAG_RE AD_COMPLETE_CLR	清除读操作完成中断标志
EFUSE_INT_FLAG_PR OGRAM_VOLTAGE_E RR_CLR	清除编程电压设置错误中断标志
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* clear the EFUSE illegal access error interrupt bits*/
```

```
efuse_interrupt_flag_clear(EFUSE_INT_FLAG_ILLEGAL_ACCESS_ERR_CLR);
```

3.13. EXMC

外部存储器控制器EXMC，用来访问各种片外存储器。章节[3.13.1](#)描述了EXMC的寄存器列表，章节[3.13.2](#)对EXMC库函数进行说明。

3.13.1. 外设寄存器描述

EXMC寄存器列表如下表所示：

表 3-357. EXMC 寄存器

寄存器名称	寄存器描述
EXMC_SNCTL	SRAM/NOR flash控制寄存器
EXMC_SNTCFG	SRAM/NOR flash时序寄存器
EXMC_SNWTCFG	SRAM/NOR flash写时序寄存器
EXMC_NCTL	NAND flash控制寄存器
EXMC_NINTEN	NAND flash中断使能寄存器
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
EXMC_SDCTL	SDRAM控制寄存器
EXMC_SDTCFG	SDRAM时序寄存器
EXMC_SDCMD	SDRAM命令寄存器
EXMC_SDARI	SDRAM自动刷新闻隔寄存器
EXMC_SDSTAT	SDRAM状态寄存器
EXMC_SDRSCTL	SDRAM读采样控制寄存器

3.13.2. 外设库函数说明

EXMC库函数列表如下表所示：

表 3-358. EXMC 库函数

库函数名称	库函数描述
exmc_norsram_deinit	复位EXMC NOR/SRAM regionx
exmc_norsram_struct_para_init	初始化结构体exmc_norsram_parameter_struct
exmc_norsram_init	初始化EXMC NOR/SRAM 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 bank
exmc_nand_struct_para_init	初始化结构体exmc_norsram_parameter_struct
exmc_nand_init	初始化EXMC NAND bank
exmc_nand_enable	使能EXMC NAND bank
exmc_nand_disable	禁用EXMC NAND bank
exmc_sdram_deinit	复位EXMC SDRAM device
exmc_sdram_struct_para_init	初始化结构体exmc_sdram_parameter_struct
exmc_sdram_init	初始化EXMC SDRAM device
exmc_norsram_sdram_remap_config	配置EXMC NOR/SRAM SDRAM重映射功能
exmc_norsram_sdram_remap_get	获取EXMC NOR/SRAM SDRAM重映射状态
exmc_norsram_consecutive_clock_config	配置连续时钟产生条件
exmc_norsram_page_size_config	配置CRAM页大小
exmc_nand_ecc_config	配置EXMC NAND ECC功能
exmc_ecc_get	获取EXMC ECC值
exmc_sdram_readsample_enable	使能读采样功能
exmc_sdram_readsample_disable	禁用读采样功能
exmc_sdram_readsample_config	配置读采样的延迟单元及周期
exmc_sdram_command_config	配置SDRAM存储器命令及相关参数
exmc_sdram_refresh_count_set	设置自动刷新闻隔
exmc_sdram_autorefresh_number_set	设置连续自动刷新的个数
exmc_sdram_write_protection_config	设置写保护功能
exmc_sdram_bankstatus_get	获取SDRAM device0或device1状态
exmc_flag_get	获取EXMC状态
exmc_flag_clear	清除EXMC状态
exmc_interrupt_enable	使能EXMC中断
exmc_interrupt_disable	禁用EXMC中断
exmc_interrupt_flag_get	获取EXMC中断状态
exmc_interrupt_flag_clear	清除EXMC中断状态

结构体 `exmc_norsram_timing_parameter_struct`

表 3-359. 结构体 `exmc_norsram_timing_parameter_struct`

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

结构体 `exmc_norsram_parameter_struct`

表 3-360. 结构体 `exmc_norsram_parameter_struct`

成员名称	功能描述
<code>norsram_region</code>	选择EXMC NOR/SRAM Region
<code>write_mode</code>	写模式（同步模式或者异步模式）
<code>extended_mode</code>	使能或者禁用扩展模式
<code>asyn_wait</code>	使能或者禁用异步等待功能
<code>nwait_signal</code>	在同步突发模式中，使能或者禁用NWAIT信号
<code>memory_write</code>	使能或者禁用写操作
<code>nwait_config</code>	配置NWAIT信号
<code>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>cram_page_size</code>	指定CRAM页大小
<code>read_write_timing</code>	未用扩展模式时，读时序参数和写时序参数；或采用扩展模式时，读时序参数
<code>write_timing</code>	未用扩展模式时，写时序参数

结构体 `exmc_nand_timing_parameter_struct`

表 3-361. 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>setup_time</code>	地址信号的建立时间

结构体 `exmc_nand_parameter_struct`

表 3-362. 结构体 `exmc_nand_parameter_struct`

成员名称	功能描述
<code>ecc_size</code>	ECC块大小
<code>atr_latency</code>	ALE至RE的延迟
<code>ctr_latency</code>	CLE至RE的延迟
<code>ecc_logic</code>	配置ECC使能或禁用
<code>databus_width</code>	NAND flash数据宽度
<code>wait_feature</code>	配置NWAIT信号使能或禁用
<code>common_space_timing</code>	NAND flash通用空间时序配置
<code>attribute_space_timing</code>	NAND flash属性空间时序配置

结构体 `exmc_sdram_timing_parameter_struct`

表 3-363. 结构体 `exmc_sdram_timing_parameter_struct`

成员名称	功能描述
<code>row_to_column_delay</code>	行至列的延迟
<code>row_precharge_delay</code>	行预充电延迟
<code>write_recovery_delay</code>	写恢复延迟
<code>auto_refresh_delay</code>	自动刷新延迟
<code>row_address_select_delay</code>	行地址选择延迟
<code>exit_selfrefresh_delay</code>	退出自刷新的延迟
<code>load_mode_register_delay</code>	加载模式寄存器延迟

结构体 `exmc_sdram_parameter_struct`

表 3-364. 结构体 `exmc_sdram_parameter_struct`

成员名称	功能描述
<code>sdram_device</code>	选择EXMC SDRAM device x
<code>pipeline_read_delay</code>	流水线读数据延迟
<code>burst_read_switch</code>	突发读开关
<code>sdclock_config</code>	SDRAM时钟配置
<code>write_protection</code>	配置写保护功能
<code>cas_latency</code>	配置CAS延迟

成员名称	功能描述
internal_bank_number	内部Bank的个数
data_width	SDRAM数据总线宽度
row_address_width	行地址位宽
column_address_width	列地址位宽
timing	读写时序参数

结构体 `exmc_sdram_command_parameter_struct`

表 3-365. 结构体 `exmc_sdram_command_parameter_struct`

成员名称	功能描述
mode_register_content	指定SDRAM模式寄存器内容
auto_refresh_number	连续的自动刷新个数
bank_select	选择SDRAM device
command	指定发送到SDRAM上的命令

函数 `exmc_norsram_deinit`

函数`exmc_norsram_deinit`描述见下表：

表 3-366. 函数 `exmc_norsram_deinit`

函数名称	<code>exmc_norsram_deinit</code>
函数原型	<code>void exmc_norsram_deinit(uint32_t exmc_norsram_region);</code>
功能描述	复位NOR/SRAM region
先决条件	-
被调用函数	-
输入参数{in}	
<code>exmc_norsram_region</code>	EXMC NOR/SRAM region
<code>EXMC_BANK0_NORSRAM_REGIONx</code>	x=0,1,2,3
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* deinitialize the EXMC NOR/SRAM region1 of bank0 */
exmc_norsram_deinit(EXMC_BANK0_NORSRAM_REGION1);
```


函数 `exmc_norsram_struct_para_init`

函数 `exmc_norsram_struct_para_init` 描述见下表：

表 3-367. 函数 `exmc_norsram_struct_para_init`

函数名称	<code>exmc_norsram_struct_para_init</code>
函数原型	<code>void exmc_norsram_struct_para_init(exmc_norsram_parameter_struct* exmc_norsram_init_struct);</code>
功能描述	初始化结构体 <code>exmc_norsram_parameter_struct</code>
先决条件	-
被调用函数	-
输入参数{in}	
-	-
输出参数{out}	
<code>exmc_norsram_init_struct</code>	初始化结构体，结构体成员参考 表3-360. 结构体 <code>exmc_norsram_parameter_struct</code>
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 3-368. 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/SRAM region
先决条件	-
被调用函数	-
输入参数{in}	
<code>exmc_norsram_init_struct</code>	初始化结构体，结构体成员参考 表3-360. 结构体 <code>exmc_norsram_parameter_struct</code>
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* initialize EXMC NOR/SRAM bank */

exmc_norsram_parameter_struct lcd_init_struct;

exmc_norsram_timing_parameter_struct lcd_timing_init_struct;

exmc_norsram_struct_para_init (&lcd_init_struct);

/* configure timing parameter */

lcd_timing_init_struct.asyn_access_mode = EXMC_ACCESS_MODE_A;

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.norsram_region = EXMC_BANK0_NORSRAM_REGION1;

lcd_init_struct.write_mode = EXMC_ASYN_WRITE;

lcd_init_struct.extended_mode = DISABLE;

lcd_init_struct.asyn_wait = DISABLE;

lcd_init_struct.nwait_signal = DISABLE;

lcd_init_struct.memory_write = ENABLE;

lcd_init_struct.nwait_config = EXMC_NWAIT_CONFIG_BEFORE;

lcd_init_struct.nwait_polarity = EXMC_NWAIT_POLARITY_LOW;

lcd_init_struct.burst_mode = DISABLE;

lcd_init_struct.databus_width = EXMC_NOR_DATABUS_WIDTH_16B;

lcd_init_struct.memory_type = EXMC_MEMORY_TYPE_SRAM;

lcd_init_struct.address_data_mux = DISABLE;

lcd_init_struct.cram_page_size = EXMC_CRAM_AUTO_SPLIT;

lcd_init_struct.read_write_timing = &lcd_timing_init_struct;

lcd_init_struct.write_timing = &lcd_timing_init_struct;

exmc_norsram_init(&lcd_init_struct);
```

函数 exmc_norsram_enable

函数exmc_norsram_enable描述见下表：

表 3-369. 函数 exmc_norsram_enable

函数名称	exmc_norsram_enable
函数原型	void exmc_norsram_enable(uint32_t exmc_norsram_region);
功能描述	使能EXMC NOR/SRAM region
先决条件	-
被调用函数	-
输入参数{in}	
exmc_norsram_region	EXMC NOR/SRAM region
EXMC_BANK0_NORSRAM_REGIONx	x=0,1,2,3
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* enable the EXMC NOR/SRAM region1 of bank0 */
```

```
exmc_norsram_enable(EXMC_BANK0_NORSRAM_REGION1);
```

函数 exmc_norsram_disable

函数exmc_norsram_disable描述见下表：

表 3-370. 函数 exmc_norsram_disable

函数名称	exmc_norsram_disable
函数原型	void exmc_norsram_disable(uint32_t exmc_norsram_region);
功能描述	禁用EXMC NOR/SRAM region
先决条件	-
被调用函数	-
输入参数{in}	
exmc_norsram_region	EXMC NOR/SRAM region
EXMC_BANK0_NORSRAM_REGIONx	x=0,1,2,3
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* disable the EXMC NOR/SRAM region1 of bank0 */
exmc_norsram_disable(EXMC_BANK0_NORSRAM_REGION1);
```

函数 **exmc_nand_deinit**

函数exmc_nand_deinit描述见下表：

表 3-371. 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 NAND bank */
exmc_nand_deinit();
```

函数 **exmc_nand_struct_para_init**

函数exmc_nand_struct_para_init描述见下表：

表 3-372. 函数 exmc_nand_struct_para_init

函数名称	exmc_nand_struct_para_init
函数原型	void exmc_nand_struct_para_init(exmc_nand_parameter_struct* exmc_nand_init_struct);
功能描述	初始化结构体exmc_nand_parameter_struct
先决条件	-
被调用函数	-
输入参数{in}	
-	-
输出参数{out}	
exmc_nand_init_struct	初始化结构体，结构体成员参考 表3-362. 结构体 exmc_nand_parameter_struct
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 3-373. 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3-362. 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_nand_struct_para_init(&nand_init_struct);

/* EXMC configuration */

nand_timing_init_struct.setuptime = 5;

nand_timing_init_struct.waittime = 4;

nand_timing_init_struct.holdtime = 2;

nand_timing_init_struct.databus_hiztime = 2;

nand_init_struct.ecc_size = EXMC_ECC_SIZE_2048BYTES;

nand_init_struct.atr_latency = EXMC_ALE_RE_DELAY_1_CK_EXMC;

nand_init_struct.ctr_latency = EXMC_CLE_RE_DELAY_1_CK_EXMC;

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 3-374. 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 bank2 */  
  
exmc_nand_enable();
```

函数 exmc_nand_disable

函数exmc_nand_disable描述见下表:

表 3-375. 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 bank2 */
```

```
exmc_nand_disable(void);
```

函数 **exmc_sdram_deinit**

函数exmc_sdram_deinit描述见下表:

表 3-376. 函数 exmc_sdram_deinit

函数名称	exmc_sdram_deinit
函数原型	void exmc_sdram_deinit(uint32_t exmc_sdram_device);
功能描述	复位EXMC SDRAM device
先决条件	-
被调用函数	-
输入参数{in}	
exmc_sdram_device	EXMC SDRAM device
EXMC_SDRAM_DEVICE1	x=0,1
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* deinitialize EXMC SDRAM device1 */
```

```
exmc_sdram_deinit(EXMC_SDRAM_DEVICE1);
```

函数 **exmc_sdram_struct_para_init**

函数exmc_sdram_struct_para_init描述见下表:

表 3-377. 函数 exmc_sdram_struct_para_init

函数名称	exmc_sdram_struct_para_init
函数原型	exmc_sdram_struct_para_init(exmc_sdram_parameter_struct* exmc_sdram_init_struct);
功能描述	初始化结构体exmc_sdram_parameter_struct
先决条件	-
被调用函数	-
输入参数{in}	
-	-

输出参数{out}	
exmc_sdram_init_struct	初始化结构体，结构体成员参考 表3-364. 结构体 exmc_sdram_parameter_struct
返回值	
-	-

例如：

```
/* initialize the struct sdram_init_struct */
exmc_sdram_parameter_struct sdram_init_struct;
exmc_sdram_struct_para_init (&sdram_init_struct);
```

函数 exmc_sdram_init

函数exmc_sdram_init描述见下表：

表 3-378. 函数 exmc_sdram_init

函数名称	exmc_sdram_init
函数原型	void exmc_sdram_init(exmc_sdram_parameter_struct* exmc_sdram_init_struct);
功能描述	初始化EXMC SDRAM device
先决条件	-
被调用函数	-
输入参数{in}	
exmc_sdram_init_struct	初始化结构体，结构体成员参考 表3-364. 结构体 exmc_sdram_parameter_struct
输出参数{out}	
-	-
返回值	
-	-

例如：

```
exmc_sdram_parameter_struct sdram_init_struct;
exmc_sdram_timing_parameter_struct sdram_timing_init_struct;
exmc_sdram_struct_para_init (&sdram_init_struct);

/* EXMC configuration */
sdram_timing_init_struct.load_mode_register_delay = 2;

/* XSRD: min = 67ns */
sdram_timing_init_struct.exit_selfrefresh_delay = 7;

/* RASD: min=42ns , max=120k (ns) */
```



```

sdram_timing_init_struct.row_address_select_delay = 5;

/* ARFD: min=60ns */

sdram_timing_init_struct.auto_refresh_delay = 6;

/* WRD: min=1 Clock cycles +6ns */

sdram_timing_init_struct.write_recovery_delay = 2;

/* RPD: min=18ns */

sdram_timing_init_struct.row_precharge_delay = 2;

/* RCD: min=18ns */

sdram_timing_init_struct.row_to_column_delay = 2;

sdram_init_struct.sdram_device = sdram_device;

sdram_init_struct.column_address_width = EXMC_SDRAM_COW_ADDRESS_9;

sdram_init_struct.row_address_width = EXMC_SDRAM_ROW_ADDRESS_13;

sdram_init_struct.data_width = EXMC_SDRAM_DATABUS_WIDTH_16B;

sdram_init_struct.internal_bank_number = EXMC_SDRAM_4_INTER_BANK;

sdram_init_struct.cas_latency = EXMC_CAS_LATENCY_3_SDCLK;

sdram_init_struct.write_protection = DISABLE;

sdram_init_struct.sdclk_config = EXMC_SDCLK_PERIODS_2_HCLK;

sdram_init_struct.burst_read_switch = ENABLE;

sdram_init_struct.pipeline_read_delay = EXMC_PIPELINE_DELAY_1_HCLK;

sdram_init_struct.timing = &sdram_timing_init_struct;

/* EXMC SDRAM bank initialization */

exmc_sdram_init(&sdram_init_struct);

```

函数 exmc_norsram_sdram_remap_config

函数exmc_norsram_sdram_remap_config描述见下表：

表 3-379. 函数 exmc_norsram_sdram_remap_config

函数名称	exmc_norsram_sdram_remap_config
函数原型	void exmc_norsram_sdram_remap_config(uint32_t bank_remap);
功能描述	配置EXMC NOR/SRAM SDRAM重映射功能
先决条件	-
被调用函数	-

输入参数{in}	
bank_remap	NOR/PSRAM和SDRAM重映射
<i>EXMC_BANK_REMAP_DEFAULT</i>	默认映射
<i>EXMC_BANK_NORPSRAM_SDRAM_SWAP</i>	NOR/PSRAM bank和SDRAM device 0地址交换
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* configure NOR/PSRAM and SDRAM remap */
exmc_norsram_sdram_remap_config(EXMC_BANK_NORPSRAM_SDRAM_SWAP);
```

函数 **exmc_norsram_sdram_remap_get**

函数exmc_norsram_sdram_remap_get描述见下表：

表 3-380. 函数 exmc_norsram_sdram_remap_get

函数名称	exmc_norsram_sdram_remap_get
函数原型	uint32_t exmc_norsram_sdram_remap_get(void);
功能描述	获取EXMC NOR/PSRAM SDRAM重映射状态
先决条件	-
被调用函数	-
输入参数{in}	
-	-
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* get NOR/PSRAM and SDRAM remap configuration */
uint32_t remap_state;
remap_state = exmc_norsram_sdram_remap_get();
```

函数 **exmc_norsram_consecutive_clock_config**

函数exmc_norsram_consecutive_clock_config描述见下表：

表 3-381. 函数 `exmc_norsram_consecutive_clock_config`

函数名称	<code>exmc_norsram_consecutive_clock_config</code>
函数原型	<code>void exmc_norsram_consecutive_clock_config(uint32_t clock_mode);</code>
功能描述	配置连续时钟
先决条件	-
被调用函数	-
输入参数{in}	
<code>clock_mode</code>	连续时钟模式
<code>EXMC_CLOCK_SYN_MODE</code>	EXMC_CLK只在同步模式产生
<code>EXMC_CLOCK_UNCONDITIONALLY</code>	EXMC_CLK无条件产生
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* configure consecutive clock */
```

```
exmc_norsram_consecutive_clock_config(EXMC_CLOCK_SYN_MODE);
```

函数 `exmc_norsram_page_size_config`

函数`exmc_norsram_page_size_config`描述见下表：

表 3-382. 函数 `exmc_norsram_page_size_config`

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

S	
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_config

函数exmc_nand_ecc_config描述见下表：

表 3-383. 函数 exmc_nand_ecc_config

函数名称	exmc_nand_ecc_config
函数原型	void exmc_nand_ecc_config(ControlStatus newvalue);
功能描述	使能或禁用EXMC NAND ECC功能
先决条件	-
被调用函数	-
输入参数{in}	
newvalue	ENABLE或DISABLE
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* enable the EXMC NAND ECC function */
```

```
exmc_nand_ecc_config(ENABLE);
```

函数 exmc_ecc_get

函数exmc_ecc_get描述见下表：

表 3-384. 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();
```

函数 exmc_sdram_readsample_enable

函数exmc_sdram_readsample_enable描述见下表：

表 3-385. 函数 exmc_sdram_readsample_enable

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

函数 exmc_sdram_readsample_disable

函数exmc_sdram_readsample_disable描述见下表：

表 3-386. 函数 exmc_sdram_readsample_disable

函数名称	exmc_sdram_readsample_disable
函数原型	void exmc_sdram_readsample_disable(void);
功能描述	禁用读采样功能
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 read sample */
exmc_sdram_readsample_enable();
```

函数 exmc_sdram_readsample_config

函数exmc_sdram_readsample_config描述见下表：

表 3-387. 函数 exmc_sdram_readsample_config

函数名称	exmc_sdram_readsample_config
函数原型	void exmc_sdram_readsample_config(uint32_t delay_cell, uint32_t extra_clk);
功能描述	配置读数据的采样时钟的延迟单元及采样周期
先决条件	-
被调用函数	-
输入参数{in}	
delay_cell	读数据的采样时钟的延迟单元
EXMC_SDRAM_x_ DELAY_CELL	x=0...15
输入参数{in}	
extra_hclk	读数据的采样周期
EXMC_SDRAM_RE ADSAMPLE_x_EXT RACK	x=0,1
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* configure the delayed sample clock and sample cycle of read data */
exmc_sdram_readsample_config(EXMC_SDRAM_1_DELAY_CELL,
EXMC_SDRAM_READSAMPLE_1_EXTRACK);
```

函数 exmc_sdram_command_config

函数exmc_sdram_command_config描述见下表:

表 3-388. 函数 exmc_sdram_command_config

函数名称	exmc_sdram_command_config
函数原型	void exmc_sdram_command_config(exmc_sdram_command_parameter_struct* exmc_sdram_command_init_struct);
功能描述	配置SDRAM存储器命令参数
先决条件	-
被调用函数	-
输入参数{in}	
exmc_sdram_command_init_struct	初始化结构体，结构体成员参考 表3-365. 结构体 exmc_sdram_command_parameter_struct
输出参数{out}	
-	-
返回值	
-	-

例如:

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

exmc_sdram_command_parameter_struct    sdram_command_init_struct;

sdram_command_init_struct.command = EXMC_SDRAM_CLOCK_ENABLE;

sdram_command_init_struct.bank_select = bank_select;

sdram_command_init_struct.auto_refresh_number = EXMC_SDRAM_AUTO_REFRESH_1_SDCLK;

sdram_command_init_struct.mode_register_content = 0;

exmc_sdram_command_config(&sdram_command_init_struct);
```

函数 exmc_sdram_refresh_count_set

函数exmc_sdram_refresh_count_set描述见下表:

表 3-389. 函数 exmc_sdram_refresh_count_set

函数名称	exmc_sdram_refresh_count_set
函数原型	void exmc_sdram_refresh_count_set(uint32_t exmc_count);
功能描述	配置自刷新闻隔
先决条件	-
被调用函数	-
输入参数{in}	

exmc_count	两个连续的自动刷新命令之间间隔的存储器时钟周期单元 0x0000~0x1FFF
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* set the SDRAM auto-refresh rate counter */
```

```
exmc_sdram_refresh_count_set(761);
```

函数 exmc_sdram_autorefresh_number_set

函数exmc_sdram_autorefresh_number_set描述见下表：

表 3-390. 函数 exmc_sdram_autorefresh_number_set

函数名称	exmc_sdram_autorefresh_number_set
函数原型	void exmc_sdram_autorefresh_number_set(uint32_t exmc_number);
功能描述	配置连续自刷新个数
先决条件	-
被调用函数	-
输入参数{in}	
exmc_number	连续的自动刷新个数
输出参数{out}	
-	-
返回值	
-	-

例如：

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

```
exmc_sdram_autorefresh_number_set(10);
```

函数 exmc_sdram_write_protection_config

函数exmc_sdram_write_protection_config描述见下表：

表 3-391. 函数 exmc_sdram_write_protection_config

函数名称	exmc_sdram_write_protection_config
函数原型	void exmc_sdram_write_protection_config(uint32_t exmc_sdram_device, ControlStatus newvalue);
功能描述	使能或禁用写保护功能
先决条件	-
被调用函数	-

输入参数{in}	
exmc_sdram_device	EXMC SDRAM device
<i>EXMC_SDRAM_DEVICE0</i> <i>VICE0</i>	x=0,1
输入参数{in}	
newvalue	ENABLE或DISABLE
输出参数{out}	
-	-
返回值	
-	-

例如:

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

```
exmc_sdram_write_protection_config(EXMC_SDRAM_DEVICE1, ENABLE);
```

函数 **exmc_sdram_bankstatus_get**

函数exmc_sdram_bankstatus_get描述见下表:

表 3-392. 函数 exmc_sdram_bankstatus_get

函数名称	exmc_sdram_bankstatus_get
函数原型	uint32_t exmc_sdram_bankstatus_get(uint32_t exmc_sdram_device);
功能描述	获取SDRAM device0或device1状态
先决条件	-
被调用函数	-
输入参数{in}	
exmc_sdram_device	EXMC SDRAM device
<i>EXMC_SDRAM_DEVICE0</i> <i>VICE0</i>	x=0,1
输出参数{out}	
-	-
返回值	
uint32_t	SDRAM devicex状态

例如:

```
/* get the status of SDRAM device1 */
```

```
uint32_t status;
```

```
status = exmc_sdram_bankstatus_get (EXMC_SDRAM_DEVICE1);
```

函数 **exmc_flag_get**

函数exmc_flag_get描述见下表:

表 3-393. 函数 **exmc_flag_get**

函数名称	exmc_flag_get
函数原型	FlagStatus exmc_flag_get(uint32_t exmc_bank,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}	
exmc_bank	EXMC外设
<i>EXMC_BANK2_NAND</i>	NAND bank2
<i>EXMC_SDRAM_DEVICE0</i>	SDRAM device0
<i>EXMC_SDRAM_DEVICE1</i>	SDRAM device1
输入参数{in}	
flag	EXMC标志状态
<i>EXMC_NAND_FLAG_LEVEL</i>	高电平状态
<i>EXMC_NAND_FLAG_RISE</i>	上升沿状态
<i>EXMC_NAND_FLAG_FALL</i>	下降沿状态
<i>EXMC_SDRAM_FLAG_REFRESH</i>	刷新错误状态
输出参数{out}	
-	-
返回值	
FlagStatus	SET或RESET

例如:

```
/* check EXMC_NAND_FLAG_RISE is set or not */
```

```
if(RESET != exmc_flag_get (EXMC_BANK2_NAND, EXMC_NAND_FLAG_RISE));
```

函数 **exmc_flag_clear**

函数exmc_flag_clear描述见下表:

表 3-394. 函数 **exmc_flag_clear**

函数名称	exmc_flag_clear
函数原型	FlagStatus exmc_flag_clear (uint32_t exmc_bank,uint32_t flag);

功能描述	清除EXMC状态
先决条件	-
被调用函数	-
输入参数{in}	
exmc_bank	EXMC外设
<i>EXMC_BANK2_NAND</i>	NAND bank2
<i>EXMC_SDRAM_DEVICE0</i>	SDRAM device0
<i>EXMC_SDRAM_DEVICE1</i>	SDRAM device1
输入参数{in}	
flag	EXMC标志状态
<i>EXMC_NAND_FLAG_LEVEL</i>	高电平状态
<i>EXMC_NAND_FLAG_RISE</i>	上升沿状态
<i>EXMC_NAND_FLAG_FALL</i>	下降沿状态
<i>EXMC_SDRAM_FLAG_REFRESH</i>	刷新错误状态
输出参数{out}	
-	-
返回值	
-	-

例如：

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

```
exmc_flag_clear(EXMC_BANK2_NAND, EXMC_NAND_FLAG_RISE);
```

函数 **exmc_interrupt_enable**

函数exmc_interrupt_enable描述见下表：

表 3-395. 函数 exmc_interrupt_enable

函数名称	exmc_interrupt_enable
函数原型	void exmc_interrupt_enable(uint32_t exmc_bank, uint32_t interrupt);
功能描述	使能EXMC中断
先决条件	-
被调用函数	-
输入参数{in}	
exmc_bank	EXMC外设
<i>EXMC_BANK2_NAND</i>	NAND bank2

ND	
EXMC_SDRAM_DE VICE0	SDRAM device0
EXMC_SDRAM_DE VICE1	SDRAM device1
输入参数{in}	
interrupt	EXMC中断
EXMC_NAND_INT_ FLAG_LEVEL	高电平中断
EXMC_NAND_INT_ FLAG_RISE	上升沿中断
EXMC_NAND_INT_ FLAG_FALL	下降沿中断
EXMC_SDRAM_IN T_FLAG_REFRESH	刷新错误中断
输出参数{out}	
-	-
返回值	
-	-

例如：

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

```
exmc_interrupt_enable(EXMC_BANK2_NAND, EXMC_NAND_INT_FLAG_RISE);
```

函数 exmc_interrupt_disable

函数exmc_interrupt_disable描述见下表：

表 3-396. 函数 exmc_interrupt_disable

函数名称	exmc_interrupt_disable
函数原型	void exmc_interrupt_disable(uint32_t exmc_bank, uint32_t interrupt);
功能描述	禁用EXMC中断
先决条件	-
被调用函数	-
输入参数{in}	
exmc_bank	EXMC外设
EXMC_BANK2_NA ND	NAND bank2
EXMC_SDRAM_DE VICE0	SDRAM device0
EXMC_SDRAM_DE VICE1	SDRAM device1
输入参数{in}	

interrupt	EXMC中断
<i>EXMC_NAND_INT_FLAG_LEVEL</i>	高电平中断
<i>EXMC_NAND_INT_FLAG_RISE</i>	上升沿中断
<i>EXMC_NAND_INT_FLAG_FALL</i>	下降沿中断
<i>EXMC_SDRAM_INT_FLAG_REFRESH</i>	刷新错误中断
输出参数{out}	
-	-
返回值	
-	-

例如：

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

```
exmc_interrupt_disable(EXMC_BANK2_NAND, EXMC_NAND_INT_FLAG_RISE);
```

函数 **exmc_interrupt_flag_get**

函数exmc_interrupt_flag_get描述见下表：

表 3-397. 函数 exmc_interrupt_flag_get

函数名称	exmc_interrupt_flag_get
函数原型	FlagStatus exmc_interrupt_flag_get(uint32_t exmc_bank, uint32_t interrupt);
功能描述	获取EXMC中断状态
先决条件	-
被调用函数	-
输入参数{in}	
exmc_bank	EXMC外设
<i>EXMC_BANK2_NAND</i>	NAND bank2
<i>EXMC_SDRAM_DEVICE0</i>	SDRAM device0
<i>EXMC_SDRAM_DEVICE1</i>	SDRAM device1
输入参数{in}	
interrupt	EXMC中断状态
<i>EXMC_NAND_FLAG_LEVEL</i>	高电平中断
<i>EXMC_NAND_FLAG_RISE</i>	上升沿中断
<i>EXMC_NAND_FLAG_FALL</i>	下降沿中断

<i>G_FALL</i>	
<i>EXMC_SDRAM_FL</i> <i>AG_REFRESH</i>	刷新错误中断
输出参数{out}	
-	-
返回值	
FlagStatus	SET或RESET

例如：

```
/* check EXMC_NAND_INT_FLAG_RISE is set or not*/
```

```
if(RESET != exmc_interrupt_flag_get (EXMC_BANK2_NAND,
EXMC_NAND_INT_FLAG_RISE));
```

函数 **exmc_interrupt_flag_clear**

函数exmc_interrupt_flag_clear描述见下表：

表 3-398. 函数 exmc_interrupt_flag_clear

函数名称	exmc_interrupt_flag_clear
函数原型	void exmc_interrupt_flag_clear(uint32_t exmc_bank,uint32_t interrupt);
功能描述	清除EXMC中断状态
先决条件	-
被调用函数	-
输入参数{in}	
exmc_bank	EXMC外设
<i>EXMC_BANK2_NAND</i>	NAND bank2
<i>EXMC_SDRAM_DEVICE0</i>	SDRAM device0
<i>EXMC_SDRAM_DEVICE1</i>	SDRAM device1
输入参数{in}	
interrupt	EXMC中断状态
<i>EXMC_NAND_FLAG_LEVEL</i>	高电平中断
<i>EXMC_NAND_FLAG_RISE</i>	上升沿中断
<i>EXMC_NAND_FLAG_FALL</i>	下降沿中断
<i>EXMC_SDRAM_FLAG_REFRESH</i>	刷新错误中断
输出参数{out}	
-	-

返回值	
-	-

例如:

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

```
exmc_interrupt_flag_clear(EXMC_BANK2_NAND, EXMC_NAND_INT_FLAG_RISE);
```

3.14. EXTI

EXTI是MCU中的中断/事件控制器，包括38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[3.14.1](#)描述了EXTI的寄存器列表，章节[3.14.2](#)对EXTI库函数进行说明。

3.14.1. 外设寄存器说明

EXTI寄存器列表如下表所示:

表 3-399. EXTI 寄存器

寄存器名称	寄存器描述
EXTI_INTEN0	中断使能寄存器0
EXTI_EVEN0	事件使能寄存器0
EXTI_RTEN0	上升沿触发使能寄存器0
EXTI_FTEN0	下降沿触发使能寄存器0
EXTI_SWIEV0	软件中断事件寄存器0
EXTI_PD0	挂起寄存器0
EXTI_INTEN1	中断使能寄存器1
EXTI_EVEN1	事件使能寄存器1
EXTI_RTEN1	上升沿触发使能寄存器1
EXTI_FTEN1	下降沿触发使能寄存器1
EXTI_SWIEV1	软件中断事件寄存器1
EXTI_PD1	挂起寄存器1

3.14.2. 外设库函数说明

EXTI库函数列表如下表所示:

表 3-400. EXTI 库函数

库函数名称	库函数描述
exti_deinit	复位EXTI
exti_init	初始化EXTI线x
exti_interrupt_enable	EXTI线x中断使能
exti_interrupt_disable	EXTI线x中断禁能

库函数名称	库函数描述
exti_event_enable	EXTI线x事件使能
exti_event_disable	EXTI线x事件禁能
exti_software_interrupt_enable	EXTI线x软件中断使能
exti_software_interrupt_disable	EXTI线x软件中断禁能
exti_flag_get	获取EXTI线x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 3-401. 枚举类型 `exti_line_enum`

枚举名称	枚举描述
EXTI_0	EXTI线0
EXTI_1	EXTI线1
EXTI_2	EXTI线2
EXTI_3	EXTI线3
EXTI_4	EXTI线4
EXTI_5	EXTI线5
EXTI_6	EXTI线6
EXTI_7	EXTI线7
EXTI_8	EXTI线8
EXTI_9	EXTI线9
EXTI_10	EXTI线10
EXTI_11	EXTI线11
EXTI_12	EXTI线12
EXTI_13	EXTI线13
EXTI_14	EXTI线14
EXTI_15	EXTI线15
EXTI_16	EXTI线16
EXTI_17	EXTI线17
EXTI_18	EXTI线18
EXTI_19	EXTI线19
EXTI_20	EXTI线20
EXTI_21	EXTI线21
EXTI_22	EXTI线22
EXTI_23	EXTI线23
EXTI_24	EXTI线24
EXTI_25	EXTI线25
EXTI_26	EXTI线26
EXTI_27	EXTI线27

枚举名称	枚举描述
EXTI_28	EXTI线28
EXTI_29	EXTI线29
EXTI_30	EXTI线30
EXTI_31	EXTI线31
EXTI_32	EXTI线32
EXTI_33	EXTI线33
EXTI_34	EXTI线34
EXTI_35	EXTI线35
EXTI_36	EXTI线36
EXTI_37	EXTI线37

枚举类型 `exti_mode_enum`

表 3-402. 枚举类型 `exti_mode_enum`

枚举名称	枚举描述
EXTI_INTERRUPT	EXTI中断模式
EXTI_EVENT	EXTI事件模式

枚举类型 `exti_trig_type_enum`

表 3-403. 枚举类型 `exti_trig_type_enum`

枚举名称	枚举描述
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 3-404. 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 3-405. 函数 exti_init

函数名称	exti_init
函数原型	void exti_init(exti_line_enum linex, exti_mode_enum mode, exti_trig_type_enum trig_type);
功能描述	初始化EXTI
先决条件	-
被调用函数	-
输入参数{in}	
linex	EXTI线x, 参考 表3-401. 枚举类型exti_line_enum
输入参数{in}	
mode	EXTI模式, 参考 表3-402. 枚举类型exti_mode_enum
输入参数{in}	
trig_type	触发类型, 参考 表3-403. 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 3-406. 函数 exti_interrupt_enable

函数名称	exti_interrupt_enable
函数原型	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3-401. 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 3-407. 函数 exti_interrupt_disable

函数名称	exti_interrupt_disable
函数原型	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3-401. 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 3-408. 函数 exti_event_enable

函数名称	exti_event_enable
函数原型	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3-401. 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 3-409. 函数 exti_event_disable

函数名称	exti_event_disable
函数原型	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3-401. 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 3-410. 函数 exti_software_interrupt_enable

函数名称	exti_software_interrupt_enable
函数原型	void exti_software_interrupt_enable(exti_line_enum linex);
功能描述	使能EXTI线x软件中断
先决条件	-
被调用函数	-
输入参数{in}	
linex	EXTI线x，参考 表3-401. 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 3-411. 函数 exti_software_interrupt_disable

函数名称	exti_software_interrupt_disable
函数原型	void exti_software_interrupt_disable(exti_line_enum linex);
功能描述	禁能EXTI线x软件中断
先决条件	-
被调用函数	-
输入参数{in}	
linex	EXTI线x，参考 表3-401. 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 3-412. 函数 exti_flag_get

函数名称	exti_flag_get
函数原型	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3-401. 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 3-413. 函数 exti_flag_clear

函数名称	exti_flag_clear
函数原型	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3-401. 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 3-414. 函数 exti_interrupt_flag_get

函数名称	exti_interrupt_flag_get
函数原型	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3-401. 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 3-415. 函数 exti_interrupt_flag_clear

函数名称	exti_interrupt_flag_clear
函数原型	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3-401. 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);
```

3.15. FAC

滤波器算法加速器（FAC）可以实现包含有限冲激响应（FIR）滤波器和无限冲激响应（IIR）滤波器的数字滤波器。章节[3.15.1](#)描述了FAC的寄存器列表，章节[3.15.2](#)对FAC库函数进行说明。

3.15.1. 外设寄存器说明

FAC寄存器列表如下表所示：

表 3-416. FAC 寄存器

寄存器名称	寄存器描述
FAC_X0BCFG	FAC X0缓冲区配置寄存器
FAC_X1BCFG	FAC X1缓冲区配置寄存器
FAC_YBCFG	FAC Y缓冲区配置寄存器
FAC_PARACFG	FAC参数配置寄存器
FAC_CTL	FAC控制寄存器
FAC_STAT	FAC状态寄存器

寄存器名称	寄存器描述
FAC_WDATA	FAC写数据寄存器
FAC_RDATA	FAC读数据寄存器

3.15.2. 外设库函数说明

FAC库函数列表如下表所示：

表 3-417. FAC 库函数

库函数名称	库函数描述
fac_deinit	FAC外设复位
fac_struct_para_init	将FAC初始化结构体中所有参数初始化为默认值
fac_fixed_data_preload_init	将FAC定点预装载结构体中所有参数初始化为默认值
fac_float_data_preload_init	将FAC浮点预装载结构体中所有参数初始化为默认值
fac_init	初始化外设FAC
fac_fixed_buffer_preload	FAC预装载X0 X1 Y定点缓冲区
fac_float_buffer_preload	FAC预装载X0 X1 Y浮点缓冲区
fac_fixed_data_preload	FAC预装载定点数据
fac_float_data_preload	FAC预装载浮点数据
fac_reset	复位FAC
fac_clip_config	FAC限幅配置
fac_float_enable	使能FAC浮点数据类型
fac_float_disable	禁能FAC浮点数据类型
fac_dma_enable	使能FAC DMA功能
fac_dma_disable	禁能FAC DMA功能
fac_x0_config	配置FAC输入缓冲区
fac_x1_config	配置FAC滤波参数缓冲区
fac_y_config	配置FAC输出缓冲区
fac_function_config	配置FAC执行函数
fac_start	启动FAC
fac_stop	停止FAC
fac_finish_calculate	FAC完成滤波计算
fac_interrupt_enable	使能FAC中断
fac_interrupt_disable	禁能FAC中断
fac_interrupt_flag_get	获取FAC中断标志
fac_flag_get	获取FAC的状态标志
fac_fixed_data_write	FAC写定点数据
fac_fixed_data_read	FAC读定点数据
fac_float_data_write	FAC写浮点数据
fac_float_data_read	FAC读浮点数据

结构体 fac_parameter_struct

表 3-418. 结构体 fac_parameter_struct

成员名称	功能描述
input_addr	输入缓冲区（X0）基地址
input_size	输入缓冲区长度
coeff_addr	滤波参数缓冲区（X1）基地址
coeff_size	滤波参数缓冲区长度
output_addr	输出缓冲区（Y）基地址
output_size	输出缓冲区长度
ipp	IPP矢量长度
ipq	IPQ矢量长度
ipr	IPR矢量长度
input_threshold	输入缓冲区已满阈值
output_threshold	输出缓冲区已空阈值
clip	使能或禁能限幅操作
func	FAC功能选择

结构体 fac_fixed_data_preload_struct

表 3-419. 结构体 fac_fixed_data_preload_struct

成员名称	功能描述
coeffa_size	滤波参数向量A的大小
*coeffa_ctx	滤波参数向量A的内容（int16_t类型）
coeffb_size	滤波参数向量B的大小
*coeffb_ctx	滤波参数向量B的内容（int16_t类型）
input_size	输入数据向量长度
*input_ctx	输入数据向量的内容（int16_t类型）
output_size	输出数据向量长度
*output_ctx	输出数据向量的内容（int16_t类型）

结构体 fac_float_data_preload_struct

表 3-420. 结构体 fac_float_data_preload_struct

成员名称	功能描述
coeffa_size	滤波参数向量A的大小
*coeffa_ctx	滤波参数向量A的内容（float类型）
coeffb_size	滤波参数向量B的大小
*coeffb_ctx	滤波参数向量B的内容（float类型）
input_size	输入数据向量长度
*input_ctx	输入数据向量的内容（float类型）
output_size	输出数据向量长度
*output_ctx	输出数据向量的内容（float类型）

函数 fac_deinit

函数fac_deinit描述见下表:

表 3-421. 函数 dac_deinit

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

例如:

```
/* deinitialize FAC */
```

```
fac_deinit();
```

函数 fac_struct_para_init

函数fac_struct_para_init描述见下表:

表 3-422. 函数 fac_struct_para_init

函数名称	fac_struct_para_init
函数原型	void fac_struct_para_init(fac_parameter_struct* fac_parameter);
功能描述	FAC外设复位
先决条件	-
被调用函数	-
输入参数{in}	
fac_parameter	FAC初始化参数结构体, 结构体成员参考 结构体fac_parameter_struct 。
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* initialize FAC init parameter struct with a default value */
```

```
fac_parameter_struct facconfig;
```

```
fac_struct_para_init(&facconfig);
```

函数 fac_fixed_data_preload_init

函数fac_fixed_data_preload_init描述见下表：

表 3-423. 函数 fac_fixed_data_preload_init

函数名称	fac_fixed_data_preload_init
函数原型	void fac_fixed_data_preload_init(fac_fixed_data_preload_struct *init_struct);
功能描述	FAC定点预装载数据初始化
先决条件	-
被调用函数	-
输入参数{in}	
init_struct	FAC定点预装载参数结构体，结构体成员参考 结构体 fac fixed data preload struct。
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* initialize the FAC fixed data preload parameter struct with the default values */
```

```
fac_fixed_data_preload_struct init_struct;
```

```
fac_fixed_data_preload_init(&init_struct);
```

函数 fac_float_data_preload_init

函数fac_float_data_preload_init描述见下表：

表 3-424. 函数 fac_float_data_preload_init

函数名称	fac_float_data_preload_init
函数原型	void fac_float_data_preload_init(fac_float_data_preload_struct *init_struct);
功能描述	FAC浮点预装载数据初始化
先决条件	-
被调用函数	-
输入参数{in}	
init_struct	FAC浮点预装载参数结构体，结构体成员参考 结构体 fac float data preload struct。
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* initialize the FAC float data preload parameter struct with the default values */
```

```
fac_float_data_preload_struct init_struct;
```

```
fac_float_data_preload_init(&init_struct);
```

函数 fac_init

函数fac_init描述见下表：

表 3-425. 函数 fac_init

函数名称	fac_init
函数原型	void fac_init(fac_parameter_struct* fac_parameter);
功能描述	FAC外设参数设置
先决条件	-
被调用函数	-
输入参数{in}	
fac_parameter	FAC初始化参数结构体，结构体成员参考 结构体fac_parameter_struct 。
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* Initialize the FAC peripheral */
```

```
fac_parameter_struct facconfig;
```

```
fac_init(&facconfig);
```

函数 fac_fixed_buffer_preload

函数fac_fixed_buffer_preload描述见下表：

表 3-426. 函数 fac_fixed_buffer_preload

函数名称	fac_fixed_buffer_preload
函数原型	void fac_fixed_buffer_preload(fac_fixed_data_preload_struct* init_struct);
功能描述	FAC预装载X0 X1 Y定点缓冲区
先决条件	-
被调用函数	-
输入参数{in}	
init_struct	FAC初始化参数结构体，结构体成员参考 结构体fac_fixed_data_preload_struct 。
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* FAC preload X0 X1 Y fixed buffer */
fac_fixed_data_preload_struct faccoeff;
fac_fixed_buffer_preload(&faccoeff);
```

函数 `fac_float_buffer_preload`

函数`fac_float_buffer_preload`描述见下表:

表 3-427. 函数 `fac_float_buffer_preload`

函数名称	<code>fac_float_buffer_preload</code>
函数原型	<code>void fac_float_buffer_preload(fac_float_data_preload_struct* init_struct);</code>
功能描述	FAC预装载X0 X1 Y浮点缓冲区
先决条件	-
被调用函数	-
输入参数{in}	
<code>init_struct</code>	FAC初始化参数结构体，结构体成员参考 结构体 <code>fac_float_data_preload_struct</code> 。
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* FAC preload X0 X1 Y float buffer */
fac_float_data_preload_struct faccoeff;
fac_float_buffer_preload(&faccoeff);
```

函数 `fac_fixed_data_preload`

函数`fac_fixed_data_preload`描述见下表:

表 3-428. 函数 `fac_fixed_data_preload`

函数名称	<code>fac_fixed_data_preload</code>
函数原型	<code>void fac_fixed_data_preload(uint8_t size, int16_t *data);</code>
功能描述	FAC预装载定点数据
先决条件	-
被调用函数	-
输入参数{in}	
<code>size</code>	数据长度
输入参数{in}	
<code>*data</code>	16位数据格式

输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* FAC preload context of the coefficient vector B */
```

```
fac_fixed_data_preload_struct init_struct;
```

```
fac_fixed_data_preload (init_struct->coeffB_size, (init_struct->coeffB_ctx));
```

函数 fac_float_data_preload

函数fac_float_data_preload描述见下表:

表 3-429. 函数 fac_float_data_preload

函数名称	fac_float_data_preload
函数原型	void fac_float_data_preload(uint8_t size, float *data);
功能描述	FAC预装载浮点数据
先决条件	-
被调用函数	-
输入参数{in}	
size	数据长度
输入参数{in}	
*data	32位数据格式
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* FAC preload context of the coefficient vector B */
```

```
fac_float_data_preload_struct init_struct;
```

```
fac_float_data_preload (init_struct->coeffB_size, (init_struct->coeffB_ctx));
```

函数 fac_reset

函数fac_reset描述见下表:

表 3-430. 函数 fac_reset

函数名称	fac_reset
函数原型	void fac_reset(void);
功能描述	FAC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:

```
/* FAC reset write and read pointers */
```

```
fac_reset();
```

函数 fac_clip_config

函数fac_clip_config描述见下表:

表 3-431. 函数 fac_clip_config

函数名称	fac_clip_config
函数原型	void fac_clip_config(uint8_t cpmode);
功能描述	FAC限幅使能或禁能
先决条件	-
被调用函数	-
输入参数{in}	
cpmode	限幅标志
FAC_CP_ENABLE	使能限幅
FAC_CP_DISABLE	禁能限幅
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* config the FAC clip enable */
```

```
fac_clip_config(FAC_CP_ENABLE);
```

函数 fac_float_enable

函数fac_float_enable描述见下表:

表 3-432. 函数 fac_float_enable

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

功能描述	浮点数据格式使能
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 FAC float point format */
```

```
fac_float_enable ();
```

函数 fac_float_disable

函数fac_float_disable描述见下表:

表 3-433. 函数 fac_float_disable

函数名称	fac_float_disable
函数原型	void fac_float_disable(void);
功能描述	浮点数据格式禁能
先决条件	-
被调用函数	-
输入参数{in}	
-	-
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* disable FAC float point format */
```

```
fac_float_disable ();
```

函数 fac_dma_enable

函数fac_dma_enable描述见下表:

表 3-434. 函数 fac_dma_enable

函数名称	fac_dma_enable
函数原型	void fac_dma_enable(uint32_t dma_req);
功能描述	使能FAC DMA功能

先决条件	-
被调用函数	-
输入参数{in}	
dma_req	DMA转换类型
FAC_DMA_READ	DMA读数据使能
FAC_DMA_WRITE	DMA写数据使能
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* enable the FAC read buffer by DMA */
```

```
fac_dma_enable(FAC_DMA_READ);
```

函数 fac_dma_disable

函数fac_dma_disable描述见下表:

表 3-435. 函数 fac_dma_disable

函数名称	fac_dma_disable
函数原型	void fac_dma_disable(uint32_t dma_req);
功能描述	禁能FAC DMA功能
先决条件	-
被调用函数	-
输入参数{in}	
dma_req	DMA转换类型
FAC_DMA_READ	DMA读数据禁能
FAC_DMA_WRITE	DMA写数据禁能
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* disable the FAC read buffer by DMA */
```

```
fac_dma_disable(FAC_DMA_READ);
```

函数 fac_x0_config

函数fac_x0_config描述见下表:

表 3-436. 函数 `fac_x0_config`

函数名称	<code>fac_x0_config</code>
函数原型	<code>void fac_x0_config(uint32_t watermark, uint8_t baseaddr, uint8_t bufsize);</code>
功能描述	FAC配置输入缓冲区
先决条件	-
被调用函数	-
输入参数{in}	
watermark	输入缓冲区阈值
<code>FAC_THRESHOLD</code> <code>_x</code>	X0缓冲区满阈值 (x =1, 2, 4, 8)
输入参数{in}	
baseaddr	输入缓冲区基地址, 0..255
输入参数{in}	
bufsize	输入缓冲区长度
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* config input buffer threshold of watermark, base address, buffer size */
```

```
fac_x0_config(FAC_THRESHOLD_1,20,10);
```

函数 `fac_x1_config`

函数`fac_x1_config`描述见下表:

表 3-437. 函数 `fac_x1_config`

函数名称	<code>fac_x1_config</code>
函数原型	<code>void fac_x1_config(uint8_t baseaddr, uint8_t bufsize);</code>
功能描述	FAC配置滤波参数缓冲区
先决条件	-
被调用函数	-
输入参数{in}	
baseaddr	滤波参数缓冲区基地址, 0..255
输入参数{in}	
bufsize	滤波参数缓冲区长度
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* config coefficient buffer base address and buffer size */
```

```
fac_x1_config(10,10);
```

函数 fac_y_config

函数fac_y_config描述见下表：

表 3-438. 函数 fac_y_config

函数名称	fac_y_config
函数原型	void fac_y_config(uint32_t watermark, uint8_t baseaddr, uint8_t bufsize);
功能描述	FAC配置输出缓冲区
先决条件	-
被调用函数	-
输入参数{in}	
watermark	输出缓冲区阈值
FAC_THRESHOLD_x	Y缓冲区空阈值 (x =1, 2, 4, 8)
输入参数{in}	
baseaddr	输出缓冲区基地址, 0..255
输入参数{in}	
bufsize	输出缓冲区长度
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* config output buffer threshold of watermark, base address, buffer size */
```

```
fac_y_config(FAC_THRESHOLD_1,30,20);
```

函数 fac_function_config

函数fac_function_config描述见下表：

表 3-439. 函数 fac_function_config

函数名称	fac_function_config
函数原型	void fac_function_config(fac_parameter_struct* fac_parameter);
功能描述	配置FAC执行函数
先决条件	-
被调用函数	-
输入参数{in}	
fac_parameter	FAC初始化参数结构体，结构体成员参考 结构体fac_parameter_struct
输出参数{out}	

-	-
返回值	
-	-

例如:

Example:

```
/* config FAC work in FIR mode*/  
  
fac_parameter_struct facconfig;  
  
facconfig.func = FUNC_CONVO_FIR;  
  
facconfig.ipp = fir_coeffb_size;  
  
facconfig.ipq = 0;  
  
facconfig.ipr = fir_gain;  
  
fac_function_config(&facconfig);
```

函数 fac_start

函数fac_start描述见下表:

表 3-440. 函数 fac_start

函数名称	fac_start
函数原型	void fac_start(void);
功能描述	启动FAC
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 FAC*/  
  
fac_start();
```

函数 fac_stop

函数fac_stop描述见下表:

表 3-441. 函数 fac_stop

函数名称	fac_stop
------	----------

函数原型	void fac_start(void);
功能描述	停止FAC
先决条件	-
被调用函数	-
输入参数{in}	
-	-
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* stop the FAC*/
```

```
fac_stop();
```

函数 fac_finish_calculate

函数fac_finish_calculate描述见下表:

表 3-442. 函数 fac_finish_calculate

函数名称	fac_finish_calculate
函数原型	void fac_finish_calculate(void);
功能描述	FAC完成滤波计算
先决条件	-
被调用函数	-
输入参数{in}	
-	-
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* finish the filter calculate*/
```

```
fac_finish_calculate ();
```

函数 fac_fixed_data_write

函数fac_fixed_data_write描述见下表:

表 3-443. 函数 fac_fixed_data_write

函数名称	fac_fixed_data_write
函数原型	void fac_fixed_data_write(int16_t data);

功能描述	FAC写定点数据
先决条件	-
被调用函数	-
输入参数{in}	
data	16位数据
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* FAC fixed data write */
```

```
fac_fixed_data_write(300);
```

函数 fac_fixed_data_read

函数fac_fixed_data_read描述见下表:

表 3-444. 函数 fac_fixed_data_read

函数名称	fac_fixed_data_read
函数原型	int16_t fac_fixed_data_read(void);
功能描述	FAC读定点数据
先决条件	-
被调用函数	-
输入参数{in}	
-	-
输出参数{out}	
-	-
返回值	
int16_t	16位定点数据

例如:

```
/* FAC fixed data read */
```

```
int16_t data;
```

```
data = fac_fixed_data_read();
```

函数 fac_float_data_write

函数fac_float_data_write描述见下表:

表 3-445. 函数 fac_float_data_write

函数名称	fac_float_data_write
函数原型	void fac_float_data_write(float data);

功能描述	FAC写浮点数据
先决条件	-
被调用函数	-
输入参数{in}	
data	32位数据
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* FAC fixed data read */

/* FAC float data write */

fac_float_data_write(200.0f);
```

函数 fac_float_data_read

函数fac_float_data_read描述见下表:

表 3-446. 函数 fac_float_data_read

函数名称	fac_float_data_read
函数原型	float fac_float_data_read(void);
功能描述	FAC读浮点数据
先决条件	-
被调用函数	-
输入参数{in}	
-	-
输出参数{out}	
-	-
返回值	
float	32位浮点数据

例如:

```
* FAC float data read */

float data;

data = fac_float_data_read();
```

函数 fac_interrupt_enable

函数fac_interrupt_enable描述见下表:

表 3-447. 函数 `fac_interrupt_enable`

函数名称	<code>fac_interrupt_enable</code>
函数原型	<code>void fac_interrupt_enable(uint32_t interrupt);</code>
功能描述	FAC中断使能
先决条件	-
被调用函数	-
输入参数{in}	
interrupt	FAC中断
<code>FAC_CTL_RIE</code>	FAC读缓冲区中断
<code>FAC_CTL_WIE</code>	FAC写缓冲区中断
<code>FAC_CTL_OFEIE</code>	FAC上溢错误中断
<code>FAC_CTL_UFEIE</code>	FAC下溢错误中断
<code>FAC_CTL_STEIE</code>	FAC饱和错误中断
<code>FAC_CTL_GSTEIE</code>	FAC增益饱和错误中断
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* enable the FAC read buffer interrupt*/
```

```
fac_interrupt_enable(FAC_CTL_RIE);
```

函数 `fac_interrupt_disable`

函数`fac_interrupt_disable`描述见下表:

表 3-448. 函数 `fac_interrupt_disable`

函数名称	<code>fac_interrupt_disable</code>
函数原型	<code>void fac_interrupt_disable(uint32_t interrupt);</code>
功能描述	FAC中断禁能
先决条件	-
被调用函数	-
输入参数{in}	
interrupt	FAC中断
<code>FAC_CTL_RIE</code>	FAC读缓冲区中断
<code>FAC_CTL_WIE</code>	FAC写缓冲区中断
<code>FAC_CTL_OFEIE</code>	FAC上溢错误中断
<code>FAC_CTL_UFEIE</code>	FAC下溢错误中断
<code>FAC_CTL_STEIE</code>	FAC饱和错误中断
<code>FAC_CTL_GSTEIE</code>	FAC增益饱和错误中断
输出参数{out}	
-	-

返回值	
-	-

例如:

```
/* disable the FAC read buffer interrupt*/
```

```
fac_interrupt_disable(FAC_CTL_RIE);
```

函数 **fac_interrupt_flag_get**

函数fac_interrupt_flag_get描述见下表:

表 3-449. 函数 fac_interrupt_flag_get

函数名称	fac_interrupt_disable
函数原型	FlagStatus fac_interrupt_flag_get(uint8_t interrupt);
功能描述	获取FAC中断标志
先决条件	-
被调用函数	-
输入参数{in}	
interrupt	FAC中断
FAC_INT_FLAG_YBEF	Y缓冲区空中断标志
FAC_INT_FLAG_X0BFF	X0缓冲区满中断标志
FAC_INT_FLAG_OFEF	上溢错误中断标志
FAC_INT_FLAG_UFEF	下溢错误中断标志
FAC_INT_FLAG_STEF	饱和错误中断标志
FAC_INT_FLAG_GSTEF	增益饱和和错误中断标志
输出参数{out}	
-	-
返回值	
FlagStatus	SET/RESET

例如:

```
/* get Y buffer empty flag status */
```

```
fac_interrupt_flag_get(FAC_INT_FLAG_YBEF);
```

函数 **fac_flag_get**

函数fac_flag_get描述见下表:

表 3-450. 函数 fac_flag_get

函数名称	fac_flag_get
函数原型	FlagStatus fac_flag_get(uint32_t flag);
功能描述	获取FAC状态标志
先决条件	-
被调用函数	-
输入参数{in}	
interrupt	FAC中断
FAC_FLAG_YBEF	Y缓冲区空标志
FAC_FLAG_X0BFF	X0缓冲区满标志
FAC_FLAG_OFEF	上溢错误标志
FAC_FLAG_UFEF	下溢错误标志
FAC_FLAG_STEF	饱和错误标志
FAC_FLAG_GSTEF	增益饱和错误标志
输出参数{out}	
-	-
返回值	
FlagStatus	SET/RESET

例如:

```
/* get Y buffer empty flag status */
fac_flag_get(FAC_FLAG_YBEF);
```

3.16. 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[3.16.1](#)描述了FMC的寄存器列表，章节[3.16.2](#)对FMC库函数进行说明。

3.16.1. 外设寄存器说明

FMC寄存器列表如下表所示：

表 3-451. FMC 寄存器

寄存器名称	寄存器描述
FMC_KEY	解锁寄存器
FMC_OBKEY	选项字节操作解锁寄存器
FMC_CTL	控制寄存器
FMC_STAT	状态寄存器
FMC_ADDR	地址寄存器
FMC_OBCTL	选项字节控制寄存器
FMC_OBSTAT0_EFT	选项字节状态生效寄存器0

寄存器名称	寄存器描述
FMC_OBSTAT0_MDF	选项字节状态修改寄存器0
FMC_DCRPADDR_EFT	DCRP地址生效寄存器
FMC_DCRPADDR_MDF	DCRP地址修改寄存器
FMC_SCRADDR_EFT	SCR地址生效寄存器
FMC_SCRADDR_MDF	SCR地址修改寄存器
FMC_WP_EFT	擦除/编程保护生效寄存器
FMC_WP_MDF	擦除/编程保护修改寄存器
FMC_BTADDR_EFT	引导装载地址生效寄存器
FMC_BTADDR_MDF	引导装载地址修改寄存器
FMC_OBSTAT1_EFT	选项字节状态生效寄存器1
FMC_OBSTAT1_MDF	选项字节状态修改寄存器1
FMC_NODEC	NO-RTDEC区域寄存器
FMC_ECCADDR	ECC错误地址寄存器
FMC_AESIVx_EFT(x = 0...2)	AES初始向量生效寄存器x (x = 0...2)
FMC_AESIVx_MDF(x = 0...2)	AES初始向量修改寄存器x (x = 0...2)
EFUSE_PIDx(x = 0,1)	产品 ID 寄存器 x (x = 0,1)

3.16.2. 外设库函数说明

FMC库函数列表如下表所示：

表 3-452. FMC 库函数

库函数名称	库函数描述
fmc_unlock	解锁FMC_CTL寄存器
fmc_lock	锁定FMC_CTL寄存器
fmc_sector_erase	擦除扇区
fmc_typical_mass_erase	标准整片擦除
fmc_protection_removed_mass_erase	带清除保护的整片擦除
fmc_word_program	在主编程块相应地址全字编程
fmc_doubleword_program	在主编程块相应地址双字编程

库函数名称	库函数描述
fmc_check_programming_area_enable	使能编程区域检查
fmc_check_programming_area_disable	失能编程区域检查
ob_unlock	解锁选项字节操作
ob_lock	锁定选项字节操作
ob_start	发送选项字节修改启动命令
ob_factory_value_config	修改选项字节值为出厂值
ob_secure_access_mode_enable	使能安全访问模式
ob_secure_access_mode_disable	禁能安全访问模式
ob_security_protection_config	配置安全保护等级选项字节
ob_bor_threshold_config	配置BOR阈值选项字节
ob_low_power_config	配置低功耗相关选项字节
ob_tcm_ecc_config	配置TCM ECC选项字节
ob_iospeed_optimize_config	配置I/O速度优化选项字节
ob_tcm_shared_ram_config	配置共享RAM中的TCM大小
ob_data_program	编程数据选项字节
ob_boot_address_config	配置引导装载地址
ob_dcrp_config	配置DCRP区域
ob_secure_area_config	配置安全访问区域
ob_write_protection_enable	使能擦除/编程保护
ob_write_protection_disable	禁能擦除/编程保护
ob_secure_mode_get	获取安全访问模式
ob_security_protection_flag_get	获取安全保护等级
ob_bor_threshold_get	获取BOR阈值
ob_low_power_get	获取低功耗相关选项字节值
ob_tcm_ecc_get	获取TCM ECC配置
ob_iospeed_optimize_get	获取IO速度优化配置
ob_tcm_shared_ram_size_get	获取共享RAM中的TCM大小
ob_data_get	获取用户数据
ob_boot_address_get	获取引导装载地址
ob_dcrp_area_get	获取DCRP区域配置
ob_secure_area_get	获取安全访问区域配置
ob_write_protection_get	获取擦除/编程保护配置
fmc_no_rtdec_config	配置NO-RTDEC区域
fmc_aes_iv_config	配置AES初始向量的高96位
fmc_flash_ecc_get	获取闪存ECC功能使能标志
fmc_no_rtdec_get	获取NO-RTDEC区域
fmc_aes_iv_get	获取AES初始向量的高96位
fmc_pid_get	获取产品ID
fmc_flag_get	获取FMC标志状态

库函数名称	库函数描述
fmc_flag_clear	清除 FMC 标志
fmc_interrupt_enable	使能 FMC 中断
fmc_interrupt_disable	禁能 FMC 中断
fmc_interrupt_flag_get	获取 FMC 中断标志状态
fmc_interrupt_flag_clear	清除 FMC 中断标志

枚举类型 **fmc_state_enum**

表 3-453. 枚举类型 **fmc_state_enum**

枚举名称	枚举描述
FMC_READY	操作完成
FMC_BUSY	操作进行中
FMC_WPERR	擦除/编程保护错误
FMC_PGSERR	编程顺序错误
FMC_RPERR	读保护错误
FMC_RSERR	读安全错误
FMC_ECCCOR	检测并纠正单个位错误标志
FMC_ECCDET	检测到双位错误标志
FMC_OBMERR	选项字节修改错误
FMC_TOERR	超时错误

枚举类型 **fmc_flag_enum**

表 3-454. 枚举类型 **fmc_flag_enum**

枚举名称	枚举描述
FMC_FLAG_BUSY	闪存忙标志
FMC_FLAG_END	闪存操作结束标志
FMC_FLAG_WPER R	闪存擦除/编程保护错误标志
FMC_FLAG_PGSE RR	闪存编程顺序错误标志
FMC_FLAG_RPER R	闪存读保护错误标志
FMC_FLAG_RSER R	闪存读安全错误错误
FMC_FLAG_ECCC OR	检测并纠正单个位错误标志
FMC_FLAG_ECCD ET	检测到双位错误标志
FMC_FLAG_OBME RR	选项字节修改错误标志
FMC_FLAG_FECC	闪存ECC功能标志

枚举类型 `fmc_interrupt_flag_enum`表 3-455. 枚举类型 `fmc_interrupt_flag_enum`

枚举名称	枚举描述
FMC_INT_FLAG_END	闪存操作结束中断标志
FMC_INT_FLAG_WPERR	闪存擦除/编程保护错误中断标志
FMC_INT_FLAG_PGERR	闪存编程顺序错误中断标志
FMC_INT_FLAG_RPERR	闪存读保护错误中断标志
FMC_INT_FLAG_RSERR	闪存读安全错误中断错误
FMC_INT_FLAG_ECCOR	检测并纠正单个位错误中断标志
FMC_INT_FLAG_ECCDET	检测到双位错误中断标志
FMC_INT_FLAG_OBERR	选项字节修改错误中断标志

枚举类型 `fmc_interrupt_enum`表 3-456. 枚举类型 `fmc_interrupt_enum`

枚举名称	枚举描述
FMC_INT_END	FMC操作结束中断
FMC_INT_WPERR	FMC擦除/编程保护错误中断
FMC_INT_PGERR	FMC编程顺序错误中断
FMC_INT_RPERR	FMC读保护错误中断
FMC_INT_RSERR	FMC读安全错误中断
FMC_INT_ECCOR	FMC检测并纠正单个位错误中断
FMC_INT_ECCDET	FMC检测到双位错误中断
FMC_INT_OBERR	FMC选项字节修改错误中断

函数 `fmc_unlock`

函数 `fmc_unlock` 描述见下表：

表 3-457. 函数 `fmc_unlock`

函数名称	<code>fmc_unlock</code>
函数原型	<code>void fmc_unlock(void);</code>
功能描述	解锁FMC_CTL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 FMC_CTL register */
```

```
fmc_unlock();
```

函数 fmc_lock

函数fmc_lock描述见下表：

表 3-458. 函数 fmc_lock

函数名称	fmc_lock
函数原型	void fmc_lock(void);
功能描述	锁定FMC_CTL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 FMC_CTL register */
```

```
fmc_lock();
```

函数 fmc_sector_erase

函数fmc_sector_erase描述见下表

表 3-459. 函数 fmc_page_erase

函数名称	fmc_sector_erase
函数原型	fmc_state_enum fmc_sector_erase(uint32_t address);
功能描述	FMC扇区擦除
先决条件	fmc_unlock

被调用函数	-
输入参数{in}	
address	擦除地址
输出参数{out}	
-	-
返回值	
fmc_state_enum	FMC状态，参考 表3-453. 枚举类型fmc_state_enum

例如：

```
fmc_unlock();
```

```
/* erase sector */
```

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

函数 fmc_typical_mass_erase

函数fmc_typical_mass_erase描述见下表

表 3-460. 函数 fmc_mass_erase

函数名称	fmc_typical_mass_erase
函数原型	fmc_state_enum fmc_typical_mass_erase(void);
功能描述	FMC标准整片擦除
先决条件	fmc_unlock
被调用函数	-
输入参数{in}	
-	-
输出参数{out}	
-	-
返回值	
fmc_state_enum	FMC状态，参考 表3-453. 枚举类型fmc_state_enum

例如：

```
fmc_unlock();
```

```
/* FMC typical mass erase */
```

```
fmc_state_enum state = fmc_typical_mass_erase();
```

函数 fmc_protection_removed_mass_erase

函数fmc_protection_removed_mass_erase描述见下表

表 3-461. 函数 fmc_protection_removed_mass_erase

函数名称	fmc_protection_removed_mass_erase
函数原型	fmc_state_enum fmc_protection_removed_mass_erase(void);

功能描述	带清除保护的整片擦除
先决条件	fmc_unlock
被调用函数	-
输入参数{in}	
-	-
输出参数{out}	
-	-
返回值	
fmc_state_enum	FMC状态，参考 表3-453. 枚举类型fmc_state_enum

例如：

```
fmc_unlock();
```

```
/* FMC protection-removed mass erase */
```

```
fmc_state_enum state = fmc_typical_mass_erase();
```

函数 fmc_word_program

函数fmc_word_program描述见下表

表 3-462. 函数 fmc_word_program

函数名称	fmc_word_program
函数原型	fmc_state_enum fmc_word_program(uint32_t address, uint32_t data);
功能描述	在相应地址全字编程
先决条件	fmc_unlock
被调用函数	-
输入参数{in}	
address	编程地址
输入参数{in}	
data	待编程数据
输出参数{out}	
-	-
返回值	
fmc_state_enum	FMC状态，参考 表3-453. 枚举类型fmc_state_enum

例如：

```
fmc_unlock();
```

```
fmc_page_erase(0x08004000);
```

```
/* program a word at the corresponding address in main flash */
```

```
fmc_state_enum fmc_state = fmc_doubleword_program(0x08004000, 0x11223344);
```

函数 fmc_doubleword_program

函数fmc_doubleword_program描述见下表

表 3-463. 函数 fmc_doubleword_program

函数名称	fmc_doubleword_program
函数原型	fmc_state_enum fmc_doubleword_program(uint32_t address, uint64_t data);
功能描述	在相应地址双字编程
先决条件	fmc_unlock
被调用函数	-
输入参数{in}	
address	编程地址
输入参数{in}	
data	待编程数据
输出参数{out}	
-	-
返回值	
fmc_state_enum	FMC状态，参考 表3-453. 枚举类型fmc_state_enum

例如：

```
fmc_unlock();
```

```
fmc_page_erase(0x08004000);
```

```
/* program a double word at the corresponding address in main flash */
```

```
fmc_state_enum fmc_state = fmc_doubleword_program(0x08004000, 0x11223344aabbccdd);
```

函数 fmc_check_programming_area_enable

函数fmc_check_programming_area_enable描述见下表：

表 3-464. 函数 fmc_check_programming_area_enable

函数名称	fmc_check_programming_area_enable
函数原型	fmc_state_enum fmc_check_programming_area_enable(void);
功能描述	使能编程区域检查
先决条件	fmc_unlock
被调用函数	-
输入参数{in}	
-	-
输出参数{out}	
-	-
返回值	
fmc_state_enum	FMC状态，参考 表3-453. 枚举类型fmc_state_enum

例如：

```
fmc_unlock();

/* enable check programming area before program operation */

fmc_state_enum fmc_state = fmc_check_programming_area_enable();
```

函数 fmc_check_programming_area_disable

函数fmc_check_programming_area_disable描述见下表：

表 3-465. 函数 fmc_check_programming_area_disable

函数名称	fmc_check_programming_area_disable
函数原型	fmc_state_enum fmc_check_programming_area_disable(void);
功能描述	失能编程区域检查
先决条件	fmc_unlock
被调用函数	-
输入参数{in}	
-	-
输出参数{out}	
-	-
返回值	
fmc_state_enum	FMC状态，参考 表3-453. 枚举类型fmc_state_enum

例如：

```
fmc_unlock();

/* disable check programming area before program operation */

fmc_state_enum fmc_state = fmc_check_programming_area_disable();
```

函数 ob_unlock

函数ob_unlock描述见下表

表 3-466. 函数 ob_unlock

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

-	-
---	---

例如:

```
fmc_unlock();
```

```
/* unlock the option bytes operation */
```

```
ob_unlock();
```

函数 ob_lock

函数ob_lock描述见下表

表 3-467. 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 bytes operation */
```

```
ob_lock();
```

函数 ob_start

函数ob_start描述见下表

表 3-468. 函数 ob_start

函数名称	ob_start
函数原型	fmc_state_enum ob_start(void);
功能描述	发送更新选项字节指令
先决条件	ob_unlock
被调用函数	-
输入参数{in}	
-	-
输出参数{out}	
-	-
返回值	

fmc_state_enum	FMC状态，参考 表3-453. 枚举类型fmc_state_enum
-----------------------	---

例如：

```
fmc_state_enum fmc_state;

fmc_unlock();

ob_unlock();

/* start modify WP option bytes */

fmc_state = ob_write_protection_disable(OB_WP_1);

ob_start();
```

函数 ob_factory_value_config

函数ob_factory_value_config描述见下表

表 3-469. 函数 ob_factory_value_config

函数名称	ob_factory_value_config
函数原型	fmc_state_enum ob_factory_value_config(void);
功能描述	修改选项字节值为出厂值
先决条件	ob_unlock
被调用函数	ob_start
输入参数{in}	
-	-
输出参数{out}	
-	-
返回值	
fmc_state_enum	FMC状态，参考 表3-453. 枚举类型fmc_state_enum

例如：

```
fmc_state_enum fmc_state;

fmc_unlock();

ob_unlock();

/* modify option byte to factory value */

fmc_state = ob_factory_value_config ();
```

函数 ob_secure_access_mode_enable

函数ob_secure_access_mode_enable描述见下表

表 3-470. 函数 ob_secure_access_mode_enable

函数名称	ob_secure_access_mode_enable
------	------------------------------

函数原型	fmc_state_enum ob_secure_access_mode_enable(void);
功能描述	使能安全访问模式
先决条件	ob_unlock
被调用函数	-
输入参数{in}	
-	-
输出参数{out}	
-	-
返回值	
fmc_state_enum	FMC状态，参考 表3-453. 枚举类型fmc_state_enum

例如：

```
fmc_state_enum fmc_state;

fmc_unlock();

ob_unlock();

/* enable secure access mode */

fmc_state = ob_secure_access_mode_enable();
```

函数 ob_secure_access_mode_disable

函数ob_secure_access_mode_disable描述见下表

表 3-471. 函数 ob_secure_access_mode_enable

函数名称	ob_secure_access_mode_disable
函数原型	fmc_state_enum ob_secure_access_mode_disable(void);
功能描述	失能安全访问模式
先决条件	ob_unlock
被调用函数	-
输入参数{in}	
-	-
输出参数{out}	
-	-
返回值	
fmc_state_enum	FMC状态，参考 表3-453. 枚举类型fmc_state_enum

例如：

```
fmc_state_enum fmc_state;

fmc_unlock();

ob_unlock();

/* disable security protection */
```

```
fmc_state = ob_secure_access_mode_disable();
```

函数 ob_security_protection_config

函数ob_security_protection_config描述见下表

表 3-472. 函数 ob_security_protection_config

函数名称	ob_security_protection_config
函数原型	fmc_state_enum ob_security_protection_config(uint8_t ob_spc)
功能描述	配置安全保护等级选项字节
先决条件	ob_unlock
被调用函数	-
输入参数{in}	
ob_spc	安全保护等级选项
FMC_NSPC	无保护状态
FMC_LSPC	保护等级低
FMC_HSPC	保护等级高
输出参数{out}	
-	-
返回值	
fmc_state_enum	FMC状态，参考 表3-453. 枚举类型fmc_state_enum

例如：

```
fmc_state_enum fmc_state;

fmc_unlock();

ob_unlock();

/* enable security protection */

fmc_state = ob_security_protection_config(FMC_LSPC);
```

函数 ob_bor_threshold_config

函数ob_bor_threshold_config描述见下表

表 3-473. 函数 ob_bor_threshold_config

函数名称	ob_bor_threshold_config
函数原型	fmc_state_enum ob_bor_threshold_config(uint32_t ob_bor_th);
功能描述	配置BOR阈值选项字节
先决条件	ob_unlock
被调用函数	-
输入参数{in}	
ob_bor_th	BOR阈值选项
OB_BOR_TH_VALUE3	BOR阈值3

OB_BOR_TH_VALUE2	BOR阈值2
OB_BOR_TH_VALUE1	BOR阈值 1
OB_BOR_TH_OFF	无BOR功能
输出参数{out}	
-	-
返回值	
fmc_state_enum	FMC状态，参考 表3-453. 枚举类型fmc_state_enum

例如：

```
fmc_state_enum fmc_state;

fmc_unlock();

ob_unlock();

/* set BOR threshold value 3 */

fmc_state = ob_bor_threshold_config(OB_BOR_TH_VALUE3);
```

函数 ob_low_power_config

函数ob_low_power_config描述见下表

表 3-474. 函数 ob_low_power_config

函数名称	ob_low_power_config
函数原型	fmc_state_enum ob_low_power_config(uint32_t ob_fwdgt, uint32_t ob_deepsleep, uint32_t ob_stdby, uint32_t ob_fwdg_suspend_deepsleep, uint32_t ob_fwdg_suspend_standby);
功能描述	配置低功耗相关选项字节
先决条件	ob_unlock
被调用函数	-
输入参数{in}	
ob_fwdgt	看门狗选项
OB_FWDGT_SW	软件控制开门狗
OB_FWDGT_HW	硬件控制开门狗
输入参数{in}	
ob_deepsleep	深度睡眠模式复位选项
OB_DEEPSLEEP_NRS T	进入深度睡眠模式时不产生复位
OB_DEEPSLEEP_RST	进入深度睡眠模式时产生复位
输入参数{in}	
ob_stdby	待机模式复位选项
OB_STDBY_NRST	进入待机模式时不产生复位
OB_STDBY_RST	进入待机模式时产生复位
输入参数{in}	

ob_fwdg_suspend_deepsleep	深度睡眠模式下独立看门狗暂停选项
OB_DPSLP_FWDGT_SUSPEND	在深度睡眠状态下暂停独立看门狗
OB_DPSLP_FWDGT_RUN	在深度睡眠状态下运行独立看门狗
输入参数{in}	
ob_fwdg_suspend_standby	待机模式下独立看门狗暂停选项配置位
OB_STDBY_FWDGT_SUSPEND	在待机状态下暂停独立看门狗
OB_STDBY_FWDGT_RUN	在待机状态下运行独立看门狗
输出参数{out}	
-	-
返回值	
fmc_state_enum	FMC状态，参考 表3-453. 枚举类型fmc_state_enum

例如：

```
fmc_state_enum fmc_state;
```

```
fmc_unlock();
```

```
ob_unlock();
```

```
/* configure low power related option byte */
```

```
fmc_state = fmc_state_enum ob_low_power_config(OB_FWDGT_SW, OB_DEEPSLEEP_
_NRST, OB_STDBY_NRST, OB_DPSLP_FWDGT_SUSPEND, OB_STDBY_FWDGT_SU
SPEND);
```

函数 ob_tcm_ecc_config

函数ob_tcm_ecc_config描述见下表

表 3-475. 函数 ob_tcm_ecc_config

函数名称	ob_tcm_ecc_config
函数原型	fmc_state_enum ob_tcm_ecc_config(uint32_t ob_itcmecce, uint32_t ob_dtcmecc, uint32_t ob_dtcmecc1);
功能描述	配置TCM ECC选项字节
先决条件	ob_unlock
被调用函数	-
输入参数{in}	
ob_itcmecce	ITCM的ECC功能使能选项
OB_ITCMECCEN_DISABLE	失能ITCM的ECC功能

OB_ITCMECCEN_ENABLE	使能ITCM的ECC功能
输入参数{in}	
ob_dtc0ecc	DTCM0的ECC功能使能选项
OB_DTCM0ECCEN_DISABLE	失能DTCM0的ECC功能
OB_DTCME0CCEN_ENABLE	使能DTCM0的ECC功能
输入参数{in}	
ob_dtc1ecc	DTCM1的ECC功能使能选项
OB_DTCM1ECCEN_DISABLE	失能DTCM1的ECC功能
OB_DTCM1ECCEN_ENABLE	使能DTCM1的ECC功能
输出参数{out}	
-	-
返回值	
fmc_state_enum	FMC状态, 参考 表3-453. 枚举类型fmc_state_enum

例如:

```
fmc_state_enum fmc_state;
```

```
fmc_unlock();
```

```
ob_unlock();
```

```
/* configure TCM ECC option byte */
```

```
fmc_state = fmc_state_enum ob_tcm_ecc_config(OB_ITCMECCEN_ENABLE, OB_DTCM0ECCEN_DISABLE, OB_DTCM1ECCEN_DISABLE);
```

函数 ob_iospeed_optimize_config

函数ob_iospeed_optimize_config描述见下表

表 3-476. 函数 ob_iospeed_optimize_config

函数名称	ob_iospeed_optimize_config
函数原型	fmc_state_enum ob_iospeed_optimize_config(uint32_t ob_iospeed_op);
功能描述	配置I/O速度优化选项字节
先决条件	ob_unlock
被调用函数	-
输入参数{in}	
ob_iospeed_op	I/O速度优化, 低压条件下I/O高速使能选项
OB_IOSPDOPEN_DISABLE	失能I/O速度优化

OB_IOSPDOPEN_ENABLE	使能I/O速度优化
输出参数{out}	
-	-
返回值	
fmc_state_enum	FMC状态，参考 表3-453. 枚举类型fmc_state_enum

例如：

```
fmc_state_enum fmc_state;
```

```
fmc_unlock();
```

```
ob_unlock();
```

```
/* disabled I/O speed optimization */
```

```
fmc_state = fmc_state_enum ob_iospeed_optimize_config(OB_IOSPDOPEN_DISABLE);
```

函数 ob_tcm_shared_ram_config

函数ob_tcm_shared_ram_config描述见下表

表 3-477. 函数 ob_tcm_shared_ram_config

函数名称	ob_tcm_shared_ram_config
函数原型	fmc_state_enum ob_tcm_shared_ram_config(uint32_t itcm_shared_ram_size, uint32_t dtcm_shared_ram_size);
功能描述	配置共享RAM中TCM大小
先决条件	ob_unlock
被调用函数	-
输入参数{in}	
itcm_shared_ram_size	共享RAM中的ITCM大小
OB_ITCM_SHARED_RAM_0KB	0-KB ITCM
OB_ITCM_SHARED_RAM_64KB	64-KB ITCM
OB_ITCM_SHARED_RAM_128KB	128-KB ITCM
OB_ITCM_SHARED_RAM_256KB	256-KB ITCM
OB_ITCM_SHARED_RAM_512KB	512-KB ITCM
输入参数{in}	
dtcm_shared_ram_size	共享RAM中的DTCM大小
OB_DTCM_SHARED_0KB	0-KB DTCM

RAM_0KB	
OB_DTCM_SHARED_ RAM_64KB	64-KB DTCM
OB_DTCM_SHARED_ RAM_128KB	128-KB DTCM
OB_DTCM_SHARED_ RAM_256KB	256-KB DTCM
OB_DTCM_SHARED_ RAM_512KB	512-KB DTCM
输出参数{out}	
-	-
返回值	
fmc_state_enum	FMC状态，参考 表3-453. 枚举类型fmc_state_enum

例如：

```
fmc_state_enum fmc_state;
```

```
fmc_unlock();
```

```
ob_unlock();
```

```
/* configure option byte TCM shared RAM size */
```

```
fmc_state = fmc_state_enum ob_tcm_shared_ram_config(OB_ITCM_SHARED_RAM_64KB,  
OB_DTCM_SHARED_RAM_64KB);
```

函数 ob_data_program

函数ob_data_program描述见下表

表 3-478. 函数 ob_data_program

函数名称	ob_data_program
函数原型	fmc_state_enum ob_data_program(uint16_t ob_data);
功能描述	修改DATA选项字节
先决条件	ob_unlock
被调用函数	-
输入参数{in}	
ob_data	用户自定义数据
输出参数{out}	
-	-
返回值	
fmc_state_enum	FMC状态，参考 表3-453. 枚举类型fmc_state_enum

例如：

```
fmc_state_enum fmc_state;
```

```
fmc_unlock();

ob_unlock();

/* modify option byte DATA */

fmc_state = ob_data_program(0x1234);
```

函数 ob_boot_address_config

函数ob_boot_address_config描述见下表

表 3-479. 函数 ob_boot_address_config

函数名称	ob_iospeed_optimize_config
函数原型	fmc_state_enum ob_boot_address_config(uint8_t boot_pin, uint16_t boot_address);
功能描述	配置引导装载地址
先决条件	ob_unlock
被调用函数	-
输入参数{in}	
boot_pin	boot管脚状态
BOOT_PIN_0	boot管脚拉低
BOOT_PIN_1	boot管脚拉高
输入参数{in}	
boot_address	引导装载地址的高16位
输出参数{out}	
-	-
返回值	
fmc_state_enum	FMC状态，参考 表3-453. 枚举类型fmc_state_enum

例如：

```
fmc_state_enum fmc_state;

fmc_unlock();

ob_unlock();

/* configure boot address */

fmc_state = fmc_state_enum ob_boot_address_config(BOOT_PIN_1, 0x1FF0)
```

函数 ob_dcrp_config

函数ob_dcrp_config描述见下表

表 3-480. 函数 ob_dcrp_config

函数名称	ob_dcrp_config
函数原型	fmc_state_enum ob_dcrp_config(uint32_t dcrp_eren, uint32_t dcrp_start,

	uint32_t dcrp_end);
功能描述	配置DCRP区域
先决条件	ob_unlock
被调用函数	-
输入参数{in}	
dcrp_eren	DCRP区域擦除选项
OB_DCRP_AREA_ERASE_DISABLE	DCRP区域擦除失能
OB_DCRP_AREA_ERASE_ENABLE	DCRP区域擦除使能
输入参数{in}	
dcrp_start	DCRP区域起始地址(0 - 0x3BF)
输入参数{in}	
dcrp_end	DCRP区域结束地址(0 - 0x3BF)
输出参数{out}	
-	-
返回值	
fmc_state_enum	FMC状态，参考 表3-453. 枚举类型fmc_state_enum

例如：

```
fmc_state_enum fmc_state;
```

```
fmc_unlock();
```

```
ob_unlock();
```

```
/* configure DCRP area */
```

```
fmc_state = fmc_state_enum ob_dcrp_config(OB_DCRP_AREA_ERASE_ENABLE, 0x10, 0x1F)
```

函数 ob_secure_area_config

函数ob_secure_area_config描述见下表

表 3-481. 函数 ob_secure_area_config

函数名称	ob_secure_area_config
函数原型	fmc_state_enum ob_secure_area_config(uint32_t scr_eren, uint32_t scr_start, uint32_t scr_end);
功能描述	配置安全访问区域
先决条件	ob_unlock
被调用函数	-
输入参数{in}	
scr_eren	安全访问区域擦除选项
OB_SCR_AREA_ERASE_DISABLE	安全访问区域擦除失能

<i>E_DISABLE</i>	
<i>OB_SCR_AREA_ERAS</i> <i>E_ENABLE</i>	安全访问区域擦除使能
输入参数{in}	
scr_start	安全访问区域起始地址(0 - 0x3BF)
输入参数{in}	
dcrp_end	安全访问区域结束地址(0 - 0x3BF)
输出参数{out}	
-	-
返回值	
fmc_state_enum	FMC状态, 参考 表3-453. 枚举类型fmc_state_enum

例如:

```
fmc_state_enum fmc_state;
```

```
fmc_unlock();
```

```
ob_unlock();
```

```
/* configure secure-access area */
```

```
fmc_state = fmc_state_enum ob_secure_area_config(OB_SCR_AREA_ERASE_ENABLE,  
0x10, 0x1F)
```

函数 ob_write_protection_enable

函数ob_write_protection_enable描述见下表

表 3-482. 函数 ob_write_protection_enable

函数名称	ob_write_protection_enable
函数原型	fmc_state_enum ob_write_protection_enable(uint32_t ob_wp);
功能描述	使能扇区编程/擦除保护
先决条件	ob_unlock
被调用函数	-
输入参数{in}	
ob_wp	使能编程/擦除保护的扇区
<i>OB_WP_0</i>	编程/擦除保护扇区0 ~ 扇区15
<i>OB_WP_1</i>	编程/擦除保护扇区16 ~ 扇区31
<i>OB_WP_2</i>	编程/擦除保护扇区32 ~ 扇区47
<i>OB_WP_3</i>	编程/擦除保护扇区48 ~ 扇区63
<i>OB_WP_4</i>	编程/擦除保护扇区64 ~ 扇区79
<i>OB_WP_5</i>	编程/擦除保护扇区80 ~ 扇区95
<i>OB_WP_6</i>	编程/擦除保护扇区96 ~ 扇区111
<i>OB_WP_7</i>	编程/擦除保护扇区112 ~ 扇区127
<i>OB_WP_8</i>	编程/擦除保护扇区128 ~ 扇区143

OB_WP_9	编程/擦除保护扇区144 ~ 扇区159
OB_WP_10	编程/擦除保护扇区160 ~ 扇区175
OB_WP_11	编程/擦除保护扇区176 ~ 扇区191
OB_WP_12	编程/擦除保护扇区192 ~ 扇区207
OB_WP_13	编程/擦除保护扇区208 ~ 扇区223
OB_WP_14	编程/擦除保护扇区224 ~ 扇区239
OB_WP_15	编程/擦除保护扇区240 ~ 扇区255
OB_WP_16	编程/擦除保护扇区256 ~ 扇区383
OB_WP_17	编程/擦除保护扇区384 ~ 扇区511
OB_WP_18	编程/擦除保护扇区512 ~ 扇区639
OB_WP_19	编程/擦除保护扇区640 ~ 扇区767
OB_WP_20	编程/擦除保护扇区768 ~ 扇区895
OB_WP_21	编程/擦除保护扇区896 ~ 扇区959
OB_WP_ALL	all sectors
输出参数{out}	
-	-
返回值	
fmc_state_enum	FMC状态, 参考 表3-453. 枚举类型fmc_state_enum

例如:

```
fmc_unlock();
```

```
ob_unlock();
```

```
/* enable sector 80 ~ sector 95 erase/program protection */
```

```
fmc_state_enum fmc_state = ob_write_protection_enable(OB_WP_5);
```

函数 ob_write_protection_disable

函数ob_write_protection_disable描述见下表

表 3-483. 函数 ob_write_protection_disable

函数名称	ob_write_protection_disable
函数原型	fmc_state_enum ob_write_protection_disable(uint32_t ob_wp);
功能描述	失能扇区编程/擦除保护
先决条件	ob_unlock
被调用函数	-
输入参数{in}	
ob_wp	失能编程/擦除保护的扇区
OB_WP_0	编程/擦除保护扇区0 ~ 扇区15
OB_WP_1	编程/擦除保护扇区16 ~ 扇区31
OB_WP_2	编程/擦除保护扇区32 ~ 扇区47
OB_WP_3	编程/擦除保护扇区48 ~ 扇区63
OB_WP_4	编程/擦除保护扇区64 ~ 扇区79

OB_WP_5	编程/擦除保护扇区80 ~ 扇区95
OB_WP_6	编程/擦除保护扇区96 ~ 扇区111
OB_WP_7	编程/擦除保护扇区112 ~ 扇区127
OB_WP_8	编程/擦除保护扇区128 ~ 扇区143
OB_WP_9	编程/擦除保护扇区144 ~ 扇区159
OB_WP_10	编程/擦除保护扇区160 ~ 扇区175
OB_WP_11	编程/擦除保护扇区176 ~ 扇区191
OB_WP_12	编程/擦除保护扇区192 ~ 扇区207
OB_WP_13	编程/擦除保护扇区208 ~ 扇区223
OB_WP_14	编程/擦除保护扇区224 ~ 扇区239
OB_WP_15	编程/擦除保护扇区240 ~ 扇区255
OB_WP_16	编程/擦除保护扇区256 ~ 扇区383
OB_WP_17	编程/擦除保护扇区384 ~ 扇区511
OB_WP_18	编程/擦除保护扇区512 ~ 扇区639
OB_WP_19	编程/擦除保护扇区640 ~ 扇区767
OB_WP_20	编程/擦除保护扇区768 ~ 扇区895
OB_WP_21	编程/擦除保护扇区896 ~ 扇区959
OB_WP_ALL	all sectors
输出参数{out}	
-	-
返回值	
fmc_state_enum	FMC状态，参考 表3-453. 枚举类型fmc_state_enum

例如：

```
fmc_unlock();
```

```
ob_unlock();
```

```
/* disable sector 80 ~ sector 95 erase/program protection */
```

```
fmc_state_enum fmc_state = ob_write_protection_enable(OB_WP_5);
```

函数 ob_secure_mode_get

函数ob_secure_mode_get描述见下表

表 3-484. 函数 ob_secure_mode_get

函数名称	ob_secure_mode_get
函数原型	FlagStatus ob_secure_mode_get(void);
功能描述	获取安全访问模式
先决条件	-
被调用函数	-
输入参数{in}	
-	-
输出参数{out}	

-	-
返回值	
FlagStatus	SET或RESET

例如：

```
/* get the option byte secure access mode */
```

```
FlagStatus flag = ob_secure_mode_get( );
```

函数 ob_security_protection_flag_get

函数ob_security_protection_flag_get描述见下表

表 3-485. 函数 ob_secure_mode_get

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

例如：

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

```
FlagStatus flag = ob_security_protection_flag_get();
```

函数 ob_bor_threshold_get

函数ob_bor_threshold_get描述见下表

表 3-486. 函数 ob_bor_threshold_get

函数名称	ob_bor_threshold_get
函数原型	uint32_t ob_bor_threshold_get(void);
功能描述	获取BOR阈值配置
先决条件	-
被调用函数	-
输入参数{in}	
-	-
输出参数{out}	
-	-

返回值	
uint32_t	BOR阈值
<i>OB_BOR_TH_VALUE3</i>	BOR阈值3
<i>OB_BOR_TH_VALUE2</i>	BOR阈值2
<i>OB_BOR_TH_VALUE1</i>	BOR阈值 1
<i>OB_BOR_TH_OFF</i>	无BOR功能

例如：

```
/* get the BOR threshold value */
```

```
uint32_t user = ob_bor_threshold_get();
```

函数 **ob_low_power_get**

函数ob_low_power_get描述见下表

表 3-487. 函数 ob_low_power_get

函数名称	ob_low_power_get
函数原型	void ob_low_power_get(uint32_t *fwdgt, uint32_t *deepsleep, uint32_t *standby, uint32_t *fwdg_suspend_deepsleep, uint32_t *fwdg_suspend_standby);
功能描述	获取低功耗相关配置
先决条件	-
被调用函数	-
输入参数{in}	
-	-
输出参数{out}	
fwdgt	看门狗选项
<i>OB_FWDGT_SW</i>	软件控制开门狗
<i>OB_FWDGT_HW</i>	硬件控制开门狗
输出参数{out}	
deepsleep	进入深度睡眠模式复位选项
<i>OB_DEEPSLEEP_NRS</i> <i>T</i>	进入深度睡眠模式时不产生复位
<i>OB_DEEPSLEEP_RST</i>	进入深度睡眠模式时产生复位
输出参数{out}	
standby	进入待机模式复位选项
<i>OB_STDBY_NRST</i>	进入待机模式时不产生复位
<i>OB_STDBY_RST</i>	进入待机模式时产生复位
输出参数{out}	
fwdg_suspend_deepsleep	深度睡眠模式下独立看门狗暂停选项
<i>OB_DPSLP_FWDGT_SUSPEND</i>	在深度睡眠状态下暂停独立看门狗

<i>OB_DPSLP_FWDGT_RUN</i>	在深度睡眠状态下运行独立看门狗
输出参数{out}	
<i>fwdg_suspend_standby</i>	待机模式下独立看门狗暂停选项
<i>OB_STDBY_FWDGT_SUSPEND</i>	在待机状态下暂停独立看门狗
<i>OB_STDBY_FWDGT_RUN</i>	在待机状态下运行独立看门狗
返回值	
-	-

例如：

```
/* get low power related option byte */
```

```
uint32_t fwdgt_value, deepsleep_value, standby_value, fwgd_suspend_deepsleep_value,
fwgd_suspend_standby_value;
```

```
ob_low_power_get(&fwdgt_value, &deepsleep_value, &standby_value, &fwgd_suspend_deepsleep_value,
&fwgd_suspend_standby_value);
```

函数 **ob_tcm_ecc_get**

函数ob_tcm_ecc_get描述见下表

表 3-488. 函数 **ob_tcm_ecc_get**

函数名称	ob_tcm_ecc_get
函数原型	void ob_tcm_ecc_get(uint32_t *itcm_ecc_option, uint32_t *dtcm0ecc_option, uint32_t *dtcm1ecc_option);
功能描述	获取TCM ECC配置
先决条件	-
被调用函数	-
输入参数{in}	
-	-
输出参数{out}	
itcm_ecc_option	CPU ITCM ECC功能选项
<i>OB_ITCMECCEN_DISABLE</i>	失能ITCM的ECC功能
<i>OB_ITCMECCEN_ENABLE</i>	使能ITCM的ECC功能
输出参数{out}	
dtcm0ecc_option	CPU DTCM0 ECC功能选项
<i>OB_DTCM0ECCEN_DISABLE</i>	失能DTCM0的ECC功能

<i>OB_DTCME0CCEN_ENABLE</i>	使能DTCM0的ECC功能
输出参数{out}	
dtcm1ecc_option	CPU DTCM1 ECC功能选项
<i>OB_DTCM1ECCEN_DISABLE</i>	失能DTCM1的ECC功能
<i>OB_DTCM1ECCEN_ENABLE</i>	使能DTCM1的ECC功能
返回值	
-	-

例如：

```
/* get TCM ECC configuration */
```

```
uint32_t itcmecc_option_value, dtcm0ecc_option_value, dtcm1ecc_option_value;
```

```
ob_tcm_ecc_get(&itcmecc_option_value, &dtcm0ecc_option_value, &dtcm1ecc_option_value);
```

函数 **ob_iospeed_optimize_get**

函数ob_iospeed_optimize_get描述见下表

表 3-489. 函数 ob_secure_mode_get

函数名称	ob_iospeed_optimize_get
函数原型	FlagStatus ob_iospeed_optimize_get(void);
功能描述	获取I/O速度优化配置
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 IO speed optimize configuration */
```

```
FlagStatus flag = ob_iospeed_optimize_get();
```

函数 **ob_tcm_shared_ram_size_get**

函数ob_tcm_shared_ram_size_get描述见下表

表 3-490. 函数 `ob_tcm_shared_ram_size_get`

函数名称	<code>ob_tcm_shared_ram_size_get</code>
函数原型	<code>void ob_tcm_shared_ram_size_get(uint32_t *itcm_shared_ram_kb_size, uint32_t *dtcm_shared_ram_kb_size);</code>
功能描述	获取共享RAM中TCM大小
先决条件	-
被调用函数	-
输入参数{in}	
-	-
输出参数{out}	
<code>itcm_shared_ram_kb_size</code>	ITCM shared RAM size in KB unit
<code>OB_ITCM_SHARED_RAM_0KB</code>	0-KB ITCM
<code>OB_ITCM_SHARED_RAM_64KB</code>	64-KB ITCM
<code>OB_ITCM_SHARED_RAM_128KB</code>	128-KB ITCM
<code>OB_ITCM_SHARED_RAM_256KB</code>	256-KB ITCM
<code>OB_ITCM_SHARED_RAM_512KB</code>	512-KB ITCM
输出参数{out}	
<code>dtcm_shared_ram_kb_size</code>	DTCM shared RAM size in KB unit
<code>OB_DTCM_SHARED_RAM_0KB</code>	0-KB DTCM
<code>OB_DTCM_SHARED_RAM_64KB</code>	64-KB DTCM
<code>OB_DTCM_SHARED_RAM_128KB</code>	128-KB DTCM
<code>OB_DTCM_SHARED_RAM_256KB</code>	256-KB DTCM
<code>OB_DTCM_SHARED_RAM_512KB</code>	512-KB DTCM
返回值	
-	-

例如:

```
/* get TCM shared RAM size */
```

```
uint32_t itcm_shared_ram_kb_size_value, dtcm_shared_ram_kb_size_value;
```

```
ob_tcm_shared_ram_size_get(&itcm_shared_ram_kb_size_value, &dtcm_shared_ram_kb_
```

size_value);

函数 ob_data_get

函数ob_data_get描述见下表

表 3-491. 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	获取用户自定义数据选项字节
先决条件	-
被调用函数	-
输入参数{in}	
-	-
输出参数{out}	
-	-
返回值	
uint16_t	用户自定义数据

例如:

```
/* get the value of FMC option bytes DATA in FMC_OBSTAT1 register */
```

```
uint16_t data = ob_data_get();
```

函数 ob_boot_address_get

函数ob_boot_address_get描述见下表

表 3-492. 函数 ob_boot_address_get

函数名称	ob_boot_address_get
函数原型	uint32_t ob_boot_address_get(uint8_t boot_pin);
功能描述	获取引导装载地址
先决条件	-
被调用函数	-
输入参数{in}	
boot_pin	boot管脚状态
BOOT_PIN_0	boot管脚拉低
BOOT_PIN_1	boot管脚拉高
输出参数{out}	
-	-
返回值	
uint32_t	引导装载地址

例如:

```
/* get boot address */
```

```
uint32_t bootaddr = ob_boot_address_get(BOOT_PIN_0);
```

函数 ob_dcrp_area_get

函数ob_dcrp_area_get描述见下表

表 3-493. 函数 ob_dcrp_area_get

函数名称	ob_dcrp_area_get
函数原型	uint8_t ob_dcrp_area_get(uint32_t *dcrp_erase_option, uint32_t *dcrp_area_start_addr, uint32_t *dcrp_area_end_addr);
功能描述	获取DCRP区域配置
先决条件	-
被调用函数	-
输入参数{in}	
-	-
输出参数{out}	
dcrp_erase_option	DCRP区域擦除选项
OB_DCRP_AREA_ERASE_DISABLE	DCRP区域擦除失能
OB_DCRP_AREA_ERASE_ENABLE	DCRP区域擦除使能
输出参数{out}	
dcrp_start_addr	DCRP区域起始地址(0 - 0x3BF)
输出参数{out}	
dcrp_end_addr	DCRP区域结束地址(0 - 0x3BF)
返回值	
uint8_t	INVLD_AREA_ADDRESS或VLD_AREA_ADDRESS

例如:

```
/* get DCRP area configuration */
```

```
uint32_t dcrp_erase_option, dcrp_startaddr, dcrp_endaddr;
```

```
uint8_t flag = ob_dcrp_area_get(&dcrp_erase_option, &dcrp_startaddr, &dcrp_endaddr);
```

函数 ob_secure_area_get

函数ob_secure_area_get描述见下表

表 3-494. 函数 ob_secure_area_get

函数名称	ob_secure_area_get
函数原型	uint8_t ob_secure_area_get(uint32_t *secure_area_option, uint32_t *scr_area_start_addr, uint32_t *scr_area_end_addr);
功能描述	获取安全访问区域配置
先决条件	-

被调用函数	-
输入参数{in}	
-	-
输出参数{out}	
secure_erase_option	安全访问区域擦除选项
OB_SCR_AREA_ERASE_DISABLE	安全访问区域擦除失能
OB_SCR_AREA_ERASE_ENABLE	安全访问区域擦除使能
输出参数{out}	
scr_area_start_addr	安全访问区域起始地址(0 - 0x3BF)
输出参数{out}	
scr_area_end_addr	安全访问区域结束地址(0 - 0x3BF)
返回值	
uint8_t	INVLD_AREA_ADDRESS或VLD_AREA_ADDRESS

例如:

```
/* get secure-access area configuration */
```

```
uint32_t scr_erase_option, scr_startaddr, scr_endaddr;
```

```
uint8_t flag = ob_dcrp_area_get(&scr_erase_option, &scr_startaddr, &scr_endaddr);
```

函数 ob_write_protection_get

函数ob_write_protection_get描述见下表

表 3-495. 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	获取扇区擦除/编程保护选项
先决条件	-
被调用函数	-
输入参数{in}	
-	-
输出参数{out}	
-	-
返回值	
uint32_t	扇区擦除/编程保护选项(0 - 0xFFFFFFFF)

例如:

```
/* get the option byte erase/program protection */
```

```
uint32_t wp = ob_write_protection_get();
```

函数 `fmc_no_rtdec_config`

函数 `fmc_no_rtdec_config` 描述见下表

表 3-496. 函数 `fmc_no_rtdec_config`

函数名称	<code>fmc_no_rtdec_config</code>
函数原型	<code>fmc_state_enum fmc_no_rtdec_config(uint32_t nodec_area_start, uint32_t nodec_area_end);</code>
功能描述	配置NO-RTDEC区域
先决条件	<code>fmc_unlock</code>
被调用函数	-
输入参数{in}	
<code>nodec_area_start</code>	NO-RTDEC区域起始地址
输入参数{in}	
<code>nodec_area_end</code>	NO-RTDEC区域结束地址
输出参数{out}	
-	-
返回值	
<code>fmc_state_enum</code>	FMC状态，参考 表3-453. 枚举类型 <code>fmc_state_enum</code>

例如：

```
/* configure NO-RTDEC area */
```

```
fmc_state_enum fmc_state = fmc_no_rtdec_config (2U, 4U);
```

函数 `fmc_aes_iv_config`

函数 `fmc_aes_iv_config` 描述见下表

表 3-497. 函数 `fmc_aes_iv_config`

函数名称	<code>fmc_aes_iv_config</code>
函数原型	<code>fmc_state_enum fmc_aes_iv_config(uint32_t *aes_iv);</code>
功能描述	配置AES初始向量的高96位
先决条件	<code>fmc_unlock</code>
被调用函数	-
输入参数{in}	
<code>aes_iv</code>	AES初始向量的高96位
输出参数{out}	
-	-
返回值	
<code>fmc_state_enum</code>	FMC状态，参考 表3-453. 枚举类型 <code>fmc_state_enum</code>

例如：

```
/* configure NO-RTDEC area */
```

```
uint32_t aes_high96[3] = {0x11111111U, 0x22222222U, 0x33333333U};
```

```
fmc_state_enum fmc_state = fmc_aes_iv_config(2U, 4U);
```

函数 fmc_flash_ecc_get

函数fmc_flash_ecc_get描述见下表

表 3-498. 函数 fmc_flash_ecc_get

函数名称	fmc_flash_ecc_get
函数原型	FlagStatus fmc_flash_ecc_get(void);
功能描述	获取闪存ECC功能使能标志
先决条件	-
被调用函数	-
输入参数{in}	
-	-
输出参数{out}	
-	-
返回值	
FlagStatus	SET或RESET

例如：

```
/* get the value of flash ECC enable bit */
```

```
fmc_flash_ecc_get();
```

函数 fmc_no_rtdec_get

函数fmc_no_rtdec_get描述见下表

表 3-499. 函数 fmc_no_rtdec_get

函数名称	fmc_no_rtdec_get
函数原型	void fmc_no_rtdec_get(uint32_t *nodec_area_start, uint32_t *nodec_area_end);
功能描述	获取NO-RTDEC区域
先决条件	-
被调用函数	-
输入参数{in}	
-	-
输出参数{out}	
nodec_area_start	NO-RTDEC区域起始地址
输出参数{out}	
nodec_area_end	NO-RTDEC区域结束地址
返回值	
-	-

例如:

```
/* get NO-RTDEC area */

uint32_t nodec_startaddr, nodec_endaddr;

fmc_no_rtdec_get(&nodec_startaddr, &nodec_endaddr);
```

函数 fmc_aes_iv_get

函数fmc_aes_iv_get描述见下表

表 3-500. 函数 fmc_aes_iv_get

函数名称	fmc_aes_iv_get
函数原型	void fmc_aes_iv_get(uint32_t *aes_iv);
功能描述	获取AES初始向量的高96位
先决条件	-
被调用函数	-
输入参数{in}	
-	-
输出参数{out}	
aes_iv	AES初始向量的高96位
返回值	
-	-

例如:

```
/* get AES initialization vector */

uint32_t aes_iv[3];

fmc_aes_iv_get(&aes_iv);
```

函数 fmc_pid_get

函数fmc_pid_get描述见下表

表 3-501. 函数 fmc_pid_get

函数名称	fmc_pid_get
函数原型	void fmc_pid_get(uint32_t *pid);
功能描述	get product ID
先决条件	-
被调用函数	-
输入参数{in}	
-	-
输出参数{out}	
pid	产品ID
返回值	

例如：

```
/* get product ID */
```

```
uint32_t pid[2];
```

```
fmc_pid_get(&pid);
```

函数 fmc_flag_get

函数fmc_flag_get描述见下表

表 3-502. 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	获取FMC标志状态
先决条件	-
被调用函数	-
输入参数{in}	
flag	受检查的FMC标志，参考 表3-454. 枚举类型fmc_flag_enum
输出参数{out}	
-	-
返回值	
FlagStatus	SET或RESET

例如：

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

```
FlagStatus flag = fmc_flag_get(FMC_FLAG_WPERR);
```

函数 fmc_flag_clear

函数fmc_flag_clear描述见下表

表 3-503. 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}	
fmc_flag	待清除的FMC标志，参考 表3-454. 枚举类型fmc_flag_enum
输出参数{out}	
-	-
返回值	

-	-
---	---

例如:

```
/* clear RPERR flag */
```

```
fmc_flag_clear(FMC_FLAG_RPERR);
```

函数 fmc_interrupt_enable

函数fmc_interrupt_enable描述见下表

表 3-504. 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中断使能选项，参考 表3-456. 枚举类型fmc_interrupt_enum
输出参数{out}	
-	-
返回值	
-	-

例如:

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

```
fmc_interrupt_enable(FMC_INT_END);
```

函数 fmc_interrupt_disable

函数fmc_interrupt_disable描述见下表

表 3-505. 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	禁能FMC中断
先决条件	-
被调用函数	-
输入参数{in}	
interrupt	FMC中断失能选项，参考 表3-456. 枚举类型fmc_interrupt_enum
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* disable FMC end interrupt */  
  
fmc_interrupt_disable(FMC_INT_END);
```

函数 fmc_interrupt_flag_get

函数fmc_interrupt_flag_get描述见下表

表 3-506. 函数 fmc_interrupt_flag_get

函数名称	fmc_interrupt_flag_get
函数原型	FlagStatus fmc_interrupt_flag_get(fmc_interrupt_flag_enum int_flag);
功能描述	获取FMC中断标志状态
先决条件	-
被调用函数	-
输入参数{in}	
int_flag	受检查的FMC中断标志，参考 表3-455. 枚举类型fmc_interrupt_flag_enum
输出参数{out}	
-	-
返回值	
FlagStatus	SET或RESET

例如：

```
/* check FMC program sequence error flag is set or not */  
  
FlagStatus flag = fmc_interrupt_flag_get(FMC_INT_FLAG_PGSERR);
```

函数 fmc_interrupt_flag_clear

函数fmc_interrupt_flag_clear描述见下表

表 3-507. 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 int_flag);
功能描述	清除FMC中断标志
先决条件	-
被调用函数	-
输入参数{in}	
int_flag	受检查的FMC中断标志，参考 表3-455. 枚举类型fmc_interrupt_flag_enum
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* clear FMC program sequence error flag */
```

```
fmc_interrupt_flag_get(FMC_INT_FLAG_PGSERR);
```

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

3.17.1. 外设寄存器说明

FWDGT寄存器列表如下表所示：

表 3-508. 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
FWDGT_WND	窗口寄存器

3.17.2. 外设库函数说明

FWDGT库函数列表如下表所示：

表 3-509. FWDGT 库函数

库函数名称	库函数描述
fdwgt_write_enable	使能对寄存器FWDGT_PSC, FWDGT_RLD和FWDGT_WND的写操作
fdwgt_write_disable	除能对寄存器FWDGT_PSC, FWDGT_RLD和FWDGT_WND的写操作
fdwgt_enable	使能FWDGT
fdwgt_prescaler_value_config	配置独立看门狗定时器时钟预分频数
fdwgt_reload_value_config	配置独立看门狗定时器计数器重装载值
fdwgt_window_value_config	配置独立看门狗定时器计数窗口值
fdwgt_counter_reload	按照FWDGT_RLD寄存器的值重装载FWDGT计数器
fdwgt_config	设置FWDGT重装载值、预分频值
fdwgt_flag_get	获取FWDGT标志位状态

函数 fdwgt_write_enable

函数fdwgt_write_enable描述见下表：

表 3-510. 函数 fdwgt_write_enable

函数名称	fdwgt_write_enable
------	--------------------

函数原型	void fwdgt_write_enable(void);
功能描述	使能对寄存器FWDGT_PSC, FWDGT_RLD和FWDGT_WND的写操作
先决条件	-
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、FWDGT_RLD and FWDGT_WND */
```

```
fwdgt_write_enable();
```

函数 fwdgt_write_disable

函数fwdgt_write_disable描述见下表：

表 3-511. 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	除能对寄存器FWDGT_PSC, FWDGT_RLD和FWDGT_WND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、FWDGT_RLD and FWDGT_WND */
```

```
fwdgt_write_disable();
```

函数 fwdgt_enable

函数fwdgt_enable描述见下表：

表 3-512. 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 FWDGT counter */
```

```
fwdgt_enable();
```

函数 fwdgt_prescaler_value_config

函数fwdgt_prescaler_value_config描述见下表：

表 3-513. 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时钟预分频数
先决条件	-
被调用函数	-
输入参数{in}	
prescaler_value	预分频值
FWDGT_PSC_DIVx	FWDGT预分频值设为x（x=4,8,16,32,64,128,256）
输出参数{out}	
-	-
返回值	
ErrStatus	ERROR / SUCCESS

例如：

```
/* set FWDGT prescaler to 4 */
```

```
ErrStatus flag;
```

```
flag = fwdgt_prescaler_value_config(FWDGT_PSC_DIV4);
```

函数 fwdgt_reload_value_config

函数fwdgt_reload_value_config描述见下表：

表 3-514. 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计数器重载值
先决条件	-
被调用函数	-
输入参数{in}	
reload_value	重载值，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 0xFFFF */
```

```
ErrStatus flag;
```

```
flag = fwdgt_reload_value_config(0xFFFF);
```

函数 fwdgt_window_value_reload

函数fwdgt_window_value_config描述见下表：

表 3-515. 函数 fwdgt_window_value_config

函数名称	fwdgt_window_value_config
函数原型	ErrStatus fwdgt_window_value_config(uint16_t window_value);
功能描述	配置独立看门狗定时器计数器窗口值
先决条件	-
被调用函数	-
输入参数{in}	
window_value	窗口值,数值范围为0x0000 – 0x0FFF
输出参数{out}	
-	-
返回值	
ErrStatus	ERROR / SUCCESS

例如：

```
/* set FWDGT window value to 0xFFFF */
```

```
ErrStatus flag;
```

```
flag = fwdgt_window_value_config(0xFFFF);
```

函数 fwdgt_counter_reload

函数fwdgt_counter_reload描述见下表：

表 3-516. 函数 fwdgt_counter_reload

函数名称	fwdgt_counter_reload
函数原型	void fwdgt_counter_reload(void);
功能描述	按照FWDGT_RLD寄存器的值重装载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 3-517. 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重装载值和预分频值
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
6	
FWDGT_PSC_DIV32	FWDGT预分频值设为32
2	
FWDGT_PSC_DIV64	FWDGT预分频值设为64
4	
FWDGT_PSC_DIV128	FWDGT预分频值设为128
28	
FWDGT_PSC_DIV256	FWDGT预分频值设为256
56	

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

例如:

```
/* configure FWDGT counter clock: 32KHz(IRC32K) / 64 = 0.5 KHz */
fwdgt_config(2*500, FWDGT_PSC_DIV64);
```

函数 fwdgt_flag_get

函数fwdgt_flag_get描述见下表:

表 3-518. 函数 fwdgt_flag_get

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

例如:

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

3.18. GPIO

GPIO用来实现各片上设备的逻辑输入/输出功能。章节[3.18.1](#)描述了GPIO的寄存器列表，章节[3.18.2](#)对GPIO库函数进行说明。

3.18.1. 外设寄存器说明

GPIO寄存器列表如下表所示：

表 3-519. 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_OSPD	端口输出速度寄存器
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
GPIOx_TG	端口位翻转寄存器
GPIO_IFL	输入滤波寄存器
GPIO_IFTP	输入滤波类型寄存器

3.18.2. 外设库函数说明

GPIO库函数列表如下表所示：

表 3-520. GPIO 库函数

库函数名称	库函数描述
gpio_deinit	复位外设GPIOx
gpio_mode_set	设置GPIO模式
gpio_output_options_set	设置GPIO输出模式和速度
gpio_bit_set	置位引脚值
gpio_bit_reset	复位引脚值
gpio_bit_write	将特定的值写入引脚
gpio_port_write	将特定的值写入一组端口
gpio_input_filter_set	设置GPIO输入过滤器
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_pin_lock	相应的引脚配置被锁定
gpio_bit_toggle	翻转GPIO引脚状态
gpio_port_toggle	翻转一组GPIO状态

函数 **gpio_deinit**

函数gpio_deinit描述见下表:

表 3-521. 函数 **gpio_deinit**

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

例如:

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

```
gpio_deinit(GPIOA);
```

函数 **gpio_mode_set**

函数gpio_mode_set描述见下表:

表 3-522. 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	-
输入参数{in}	
gpio_periph	GPIO端口
GPIOx	端口选择 (x = A,B,C,D,E,F,G,H,J,K)
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_ANA	模拟模式

LOG	
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	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 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 3-523. 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,F,G,H,J,K)
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_12	最大输出速度为12MHz

<i>MHZ</i>	
<i>GPIO_OSPEED_60MHZ</i>	最大输出速度为60MHz
<i>GPIO_OSPEED_85MHZ</i>	最大输出速度为85MHz
<i>GPIO_OSPEED_100_220MHZ</i>	最大输出速度为100/220MHz
输入参数{in}	
pin	GPIO引脚
<i>GPIO_PIN_x</i>	引脚选择 (x = 0..15)
<i>GPIO_PIN_ALL</i>	所有引脚
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* 配置PA0工作于推挽输出模式 */
```

```
gpio_output_options_set(GPIOA, GPIO_OTYPE_PP, GPIO_OSPEED_12MHZ,
GPIO_PIN_0);
```

函数 gpio_bit_set

函数gpio_bit_set描述见下表:

表 3-524. 函数 gpio_bit_set

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

例如：

```
/* set PA0 */

gpio_bit_set(GPIOA, GPIO_PIN_0);
```

函数 gpio_bit_reset

函数gpio_bit_reset描述见下表：

表 3-525. 函数 gpio_bit_reset

函数名称	gpio_bit_reset
函数原型	void gpio_bit_reset(uint32_t gpio_periph, uint32_t pin);
功能描述	复位引脚值
先决条件	-
被调用函数	-
输入参数{in}	
gpio_periph	GPIO端口
GPIOx	端口选择 (x = A,B,C,D,E,F,G,H,J,K)
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 3-526. 函数 gpio_bit_write

函数名称	gpio_bit_write
函数原型	void gpio_bit_write(uint32_t gpio_periph, uint32_t pin, bit_status bit_value);
功能描述	将特定的值写入引脚
先决条件	-
被调用函数	-
输入参数{in}	
gpio_periph	GPIO端口
GPIOx	端口选择 (x = A,B,C,D,E,F,G,H,J,K)

输入参数{in}	
pin	GPIO引脚
<i>GPIO_PIN_x</i>	引脚选择 (x = 0..15)
<i>GPIO_PIN_ALL</i>	所有引脚
输入参数{in}	
bit_value	设置或清除
<i>RESET</i>	清除引脚值
<i>SET</i>	设置引脚值
输出参数{out}	
-	-
返回值	
-	-

例如:

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

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

函数 gpio_port_write

函数gpio_port_write描述见下表:

表 3-527. 函数 gpio_port_write

函数名称	gpio_port_write
函数原型	void gpio_port_write(uint32_t gpio_periph, uint16_t data);
功能描述	将特定的值写入端口
先决条件	-
被调用函数	-
输入参数{in}	
gpio_periph	GPIO端口
<i>GPIOx</i>	端口选择 (x = A,B,C,D,E,F,G,H,J,K)
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_filter_set**

函数gpio_input_filter_set描述见下表：

表 3-528. 函数 **gpio_input_filter_set**

函数名称	gpio_input_filter_set
函数原型	void gpio_input_filter_set(uint32_t gpio_periph, uint8_t speriod, uint32_t iftype, uint32_t pin);
功能描述	设置GPIO的输入过滤
先决条件	-
被调用函数	-
输入参数{in}	
gpio_periph	GPIO端口
GPIOx	端口选择 (x = A,B,C,D,E,F,G,H,J,K)
输入参数{in}	
speriod	过滤采样周期
GPIO_ISPERIOD(x)	周期值 (x = 0 ~ 255)
输入参数{in}	
iftype	过滤输入类型
GPIO_IFTYPE_SYNC	同步类型
GPIO_IFTYPE_3_SAMPLES	过滤 (3个采样点)
GPIO_IFTYPE_6_SAMPLES	过滤 (6个采样点)
GPIO_IFTYPE_ASYNC	异步类型
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 GPIO input filter */
```

```
gpio_input_filter_set(GPIOA, GPIO_ISPERIOD(100), GPIO_IFTYPE_SYNC);
```

函数 **gpio_input_bit_get**

函数gpio_input_bit_get描述见下表：

表 3-529. 函数 `gpio_input_bit_get`

函数名称	<code>gpio_input_bit_get</code>
函数原型	<code>FlagStatus gpio_input_bit_get(uint32_t gpio_periph, uint32_t pin);</code>
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}	
<code>gpio_periph</code>	GPIO端口
<code>GPIOx</code>	端口选择 (x = A,B,C,D,E,F,G,H,J,K)
输入参数{in}	
<code>pin</code>	GPIO引脚
<code>GPIO_PIN_x</code>	引脚选择 (x = 0..15)
<code>GPIO_PIN_ALL</code>	所有引脚
输出参数{out}	
-	-
返回值	
<code>FlagStatus</code>	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 3-530. 函数 `gpio_input_port_get`

函数名称	<code>gpio_input_port_get</code>
函数原型	<code>uint16_t gpio_input_port_get(uint32_t gpio_periph);</code>
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}	
<code>gpio_periph</code>	GPIO端口
<code>GPIOx</code>	端口选择 (x = A,B,C,D,E,F,G,H,J,K)
输出参数{out}	
-	-
返回值	
<code>uint16_t</code>	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 3-531. 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	获取端口所有引脚的输出值
先决条件	-
被调用函数	-
输入参数{in}	
gpio_periph	GPIO端口
GPIOx	端口选择 (x = A,B,C,D,E,F,G,H,J,K)
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 3-532. 函数 gpio_output_port_get

函数名称	gpio_output_port_get
函数原型	uint16_t gpio_output_port_get(uint32_t gpio_periph);
功能描述	获取引脚的输出值
先决条件	-
被调用函数	-
输入参数{in}	

gpio_periph	GPIO端口
<i>GPIOx</i>	端口选择 (x = A,B,C,D,E,F,G,H,J,K)
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 3-533. 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
<i>GPIOx</i>	端口选择 (x = A,B,C,D,E,F,G,H,J,K)
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
<i>GPIO_AF_0</i>	SYSTEM, TIMER40, TIMER41, TIMER42, TIMER43, TIMER44
<i>GPIO_AF_1</i>	TIMER0, TIMER1, TIMER15, TIMER16, EXMC
<i>GPIO_AF_2</i>	TIMER2, TIMER3, TIMER4, TIMER7, TIMER14, CAN2, EXMC
<i>GPIO_AF_3</i>	TIMER7, TIMER9, EDOUT, EXMC, HPDF, OSPIM
<i>GPIO_AF_4</i>	TIMER14, I2C0, I2C1, I2C2, I2C3, USART0, HPDF, OSPIM
<i>GPIO_AF_5</i>	SPI0, SPI1, SPI2, SPI3, SPI4, SPI5, CAN2
<i>GPIO_AF_6</i>	UART3, SPI2, I2C3, HPDF, EDOUT, OSPIM
<i>GPIO_AF_7</i>	USART0, USART1, USART2, USART5, UART6, TIMER40, TIMER41, TIMER42, TIMER43, SPI1, SPI2, SPI5, USBHS1
<i>GPIO_AF_8</i>	UART3, UART4, UART7, SPI5, TIMER44, USBHS1
<i>GPIO_AF_9</i>	TRGSEL, CAN0, CAN1, OPSIM, EXMC
<i>GPIO_AF_10</i>	CMP, USBHS0, OPSIM, EXMC
<i>GPIO_AF_11</i>	CMP, UART6, EXMC, HPDF, I2C3, OPSIM
<i>GPIO_AF_12</i>	TIMER0, EXMC, OPSIM, CMP, USBHS1
<i>GPIO_AF_13</i>	TRGSEL, COMP0, CMP, TIMER22

<i>GPIO_AF_14</i>	<i>UART4, TIMER23</i>
<i>GPIO_AF_15</i>	<i>EVENTOUT</i>
输入参数{in}	
pin	GPIO引脚
<i>GPIO_PIN_x</i>	引脚选择 (x = 0..15)
<i>GPIO_PIN_ALL</i>	所有引脚
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/*set PA0 alternate function 0*/
```

```
gpio_af_set(GPIOA, GPIO_AF_0, GPIO_PIN_0);
```

函数 gpio_pin_lock

函数gpio_pin_lock描述见下表:

表 3-534. 函数 gpio_pin_lock

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

例如:

```
/* lock PA0 */
```

```
gpio_pin_lock(GPIOA, GPIO_PIN_0);
```


函数 **gpio_bit_toggle**

函数gpio_bit_toggle描述见下表:

表 3-535. 函数 **gpio_bit_toggle**

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

例如:

```
/* toggle PA0 */
```

```
gpio_bit_toggle(GPIOA, GPIO_PIN_0);
```

函数 **gpio_port_toggle**

函数gpio_port_toggle描述见下表:

表 3-536. 函数 **gpio_port_toggle**

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

例如:

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

3.19. HPDF

GD32H75E 内部集成了一种专门用于外部 $\Sigma-\Delta$ 调制器的高性能数字滤波器模块 (HPDF)。章节 [3.19.1](#) 描述了 HPDF 的寄存器列表, 章节 [3.19.2](#) 对 HPDF 库函数进行说明。

3.19.1. 外设寄存器说明

HPDF 寄存器列表如下表所示:

表 3-537. HPDF 寄存器

寄存器名称	寄存器描述
HPDF_CHxCTL	通道x控制寄存器
HPDF_CHxCFG0	通道x配置寄存器
HPDF_CHxCFG1	通道x配置寄存器1
HPDF_CHxTMFDT	通道x阈值监视器滤波器数据寄存器
HPDF_CHxPDI	通道x并行数据输入寄存器
HPDF_CHxPS	通道x跳频寄存器
HPDF_FLTyCTL0	滤波器y控制寄存器0
HPDF_FLTyCTL1	滤波器y控制寄存器1
HPDF_FLTySTAT	滤波器y状态寄存器
HPDF_FLTyINTC	滤波器y中断标志清除寄存器
HPDF_FLTyICGS	滤波器y注入组通道选择寄存器
HPDF_FLTySFCFG	滤波器y SINC滤波器配置寄存器
HPDF_FLTyIDATA	滤波器y注入组转换数据寄存器
HPDF_FLTyRDATA	滤波器y规则通道转换数据寄存器
HPDF_FLTyTMHT	滤波器y阈值监视器上限阈值寄存器
HPDF_FLTyTMLT	滤波器y阈值监视器下限阈值寄存器
HPDF_FLTyTMSTAT	滤波器y阈值监视器状态寄存器
HPDF_FLTyTMFC	滤波器y阈值监视器标志清除寄存器
HPDF_FLTyEMMAX	滤波器y极值监视器最大值寄存器
HPDF_FLTyEMMIN	滤波器y极值监视器最小值寄存器
HPDF_FLTyCT	滤波器y转换定时器寄存器

3.19.2. 外设库函数说明

HPDF库函数列表如下表所示:

表 3-538. HPDF 库函数

库函数名称	库函数描述
hpdf_deinit	复位HPDF外设
hpdf_channel_struct_para_init	初始化HPDF通道结构体参数
hpdf_filter_struct_para_init	初始化HPDF滤波器结构体参数
hpdf_rc_struct_para_init	初始化规则转换结构体参数
hpdf_ic_struct_para_init	初始化注入转换结构体参数
hpdf_enable	使能HPDF模块
hpdf_disable	禁止HPDF模块
hpdf_channel_init	初始化HPDF通道
hpdf_filter_init	初始化HPDF滤波器
hpdf_rc_init	初始化规则转换
hpdf_ic_init	初始化注入转换
hpdf_clock_output_config	配置串行输出时钟
hpdf_clock_output_source_config	配置串行输出时钟源
hpdf_clock_output_duty_mode_disable	禁止串行输出时钟占空比模式
hpdf_clock_output_duty_mode_enable	使能串行输出时钟占空比模式
hpdf_clock_output_divider_config	配置串行输出时钟分频
hpdf_channel_enable	使能HPDF通道
hpdf_channel_disable	禁止HPDF通道
hpdf_spi_clock_source_config	配置SPI接口时钟源
hpdf_serial_interface_type_config	配置串行接口类型
hpdf_malfunction_monitor_disable	禁止故障监视器
hpdf_malfunction_monitor_enable	使能故障监视器
hpdf_clock_loss_disable	禁止时钟丢失检测
hpdf_clock_loss_enable	使能时钟丢失检测
hpdf_channel_pin_redirection_disable	禁止通道输入引脚重定向
hpdf_channel_pin_redirection_enable	使能通道输入引脚重定向
hpdf_channel_multiplexer_config	配置复用通道输入数据源
hpdf_data_pack_mode_config	配置数据封装模式
hpdf_data_right_bit_shift_config	配置数据右移位数
hpdf_calibration_offset_config	配置数据校准偏移
hpdf_malfunction_break_signal_config	配置故障监视器断路信号
hpdf_malfunction_counter_config	配置故障监视器计数器阈值
hpdf_write_parallel_data_standard_mode	写入数据封装标准模式下的并行数据
hpdf_write_parallel_data_interleaved_mode	写入数据封装交错模式下的并行数据
hpdf_write_parallel_data_dual_mode	写入数据封装双通道模式下的并行数据
hpdf_pulse_skip_update	更新跳频脉冲数量
hpdf_pulse_skip_read	读取跳频脉冲数量
hpdf_filter_enable	使能滤波器
hpdf_filter_disable	禁止滤波器
hpdf_filter_config	配置滤波器阶数和过采样率

库函数名称	库函数描述
hpdf_integrator_oversample	配置积分器过采样率
hpdf_threshold_monitor_filter_config	配置阈值监视器的滤波器
hpdf_threshold_monitor_filter_read_data	读取阈值监视器滤波器的数据
hpdf_threshold_monitor_fast_mode_disable	禁止阈值监视器快速模式
hpdf_threshold_monitor_fast_mode_enable	使能阈值监视器快速模式
hpdf_threshold_monitor_channel	配置阈值监视器通道
hpdf_threshold_monitor_high_threshold	配置阈值监视器上限阈值
hpdf_threshold_monitor_low_threshold	配置阈值监视器下限阈值
hpdf_high_threshold_break_signal	配置阈值监视器上限阈值事件断路信号
hpdf_low_threshold_break_signal	配置阈值监视器下限阈值事件断路信号
hpdf_extremes_monitor_channel	配置极值监视器通道
hpdf_extremes_monitor_maximum_get	获取极值监视器最大极值
hpdf_extremes_monitor_minimum_get	获取极值监视器最小极值
hpdf_conversion_time_get	获取转换计时器值
hpdf_rc_continuous_disable	禁止规则转换连续模式
hpdf_rc_continuous_enable	使能规则转换连续模式
hpdf_rc_start_by_software	软件启动规则转换
hpdf_rc_syn_disable	禁止规则转换同步
hpdf_rc_syn_enable	使能规则转换同步
hpdf_rc_dma_disable	禁止规则转换DMA
hpdf_rc_dma_enable	使能规则转换DMA
hpdf_rc_channel_config	配置规则转换通道
hpdf_rc_fast_mode_disable	禁止规则转换快速模式
hpdf_rc_fast_mode_enable	使能规则转换快速模式
hpdf_rc_data_get	获取规则转换数据
hpdf_rc_channel_get	获取最近一次规则转换的通道
hpdf_ic_start_by_software	软件启动注入转换
hpdf_ic_syn_disable	禁止注入转换同步
hpdf_ic_syn_enable	使能注入转换同步
hpdf_ic_dma_disable	禁止注入转换DMA
hpdf_ic_dma_enable	使能注入转换DMA
hpdf_ic_scan_mode_disable	禁止注入转换扫描模式
hpdf_ic_scan_mode_enable	使能注入转换扫描模式
hpdf_ic_trigger_signal_disable	禁止注入转换触发信号
hpdf_ic_trigger_signal_config	配置注入转换触发信号和边沿
hpdf_ic_channel_config	配置注入组转换通道
hpdf_ic_data_get	获取注入转换数据
hpdf_ic_channel_get	获取最近一次注入转换的通道
hpdf_flag_get	获取HPDF标志位
hpdf_flag_clear	清除HPDF标志位
hpdf_interrupt_enable	使能HPDF中断

库函数名称	库函数描述
hpdf_interrupt_disable	禁止HPDF中断
hpdf_interrupt_flag_get	获取HPDF中断标志位
hpdf_interrupt_flag_clear	清楚HPDF中断标志位

结构体 hpdf_channel_parameter_struct

表 3-539. 结构体 hpdf_channel_parameter_struct

成员名称	功能描述
data_packing_mode	并行数据寄存器的数据封装模式
channel_muxlexer	复用通道输入数据源
channel_pin_select	通道输入引脚选择
ck_loss_detector	时钟丢失检测
malfunction_monitor	故障监视器
spi_ck_source	SPI接口时钟源
serial_interface	串行接口类型
calibration_offset	24位校准偏移
right_bit_shift	右移位数
tm_filter	阈值监视器滤波器阶数选择
tm_filter_oversample	阈值监视器滤波器过采样率
mm_break_signal	分配故障监视器断路信号
mm_counter_threshold	故障监视器计数阈值
plsk_value	跳频脉冲数

结构体 hpdf_filter_parameter_struct

表 3-540. 结构体 hpdf_filter_parameter_struct

成员名称	功能描述
tm_fast_mode	阈值监视器快速模式
tm_channel	阈值监视器通道
tm_high_threshold	阈值监视器上限阈值
tm_low_threshold	阈值监视器下限阈值
extreme_monitor_channel	极值监视器通道
sinc_filter	Sinc滤波器阶数
sinc_oversample	Sinc滤波器过采样率
integrator_oversample	积分器过采样率
ht_break_signal	分配上限阈值断路信号
lt_break_signal	分配下限阈值断路信号

结构体 `hpdf_rc_parameter_struct`

表 3-541. 结构体 `hpdf_rc_parameter_struct`

成员名称	功能描述
<code>fast_mode</code>	规则转换快速转换模式
<code>rsc_channel</code>	规则转换通道
<code>rcdmaen</code>	使能读取规则转换数据的DMA通道
<code>rcsyn</code>	规则转换同步
<code>continuous_mode</code>	规则转换连续模式

结构体 `hpdf_ic_parameter_struct`

表 3-542. 结构体 `hpdf_ic_parameter_struct`

成员名称	功能描述
<code>trigger_edge</code>	注入转换触发边沿
<code>trigger_signal</code>	注入转换触发信号
<code>icdmaen</code>	使能读取注入转换数据的DMA通道
<code>scmod</code>	注入转换扫描模式
<code>icsyn</code>	注入转换同步
<code>ic_channel_group</code>	选择注入转换通道组

枚举类型 `hpdf_channel_enum`

表 3-543. 枚举类型 `hpdf_channel_enum`

成员名称	功能描述
<code>CHANNEL0</code>	HPDF通道0
<code>CHANNEL1</code>	HPDF通道1
<code>CHANNEL2</code>	HPDF通道2
<code>CHANNEL3</code>	HPDF通道3
<code>CHANNEL4</code>	HPDF通道4
<code>CHANNEL5</code>	HPDF通道5
<code>CHANNEL6</code>	HPDF通道6
<code>CHANNEL7</code>	HPDF通道7

枚举类型 `hpdf_filter_enum`

表 3-544. 枚举类型 `hpdf_filter_enum`

成员名称	功能描述
<code>FLT0</code>	HPDF滤波器0
<code>FLT1</code>	HPDF滤波器1
<code>FLT2</code>	HPDF滤波器2
<code>FLT3</code>	HPDF滤波器3

枚举类型 `hpdf_flag_enum`

表3-545. 枚举类型 `hpdf_flag_enum`

成员名称	功能描述
<code>HPDF_FLAG_FLTy_ICEF</code>	注入转换结束标志
<code>HPDF_FLAG_FLTy_RCEF</code>	规则转换结束标志
<code>HPDF_FLAG_FLTy_ICDOF</code>	注入转换溢出标志
<code>HPDF_FLAG_FLTy_RCDOF</code>	规则转换溢出标志
<code>HPDF_FLAG_FLTy_TMEOF</code>	阈值监视器事件标志
<code>HPDF_FLAG_FLTy_ICPF</code>	注入转换正在进行标志
<code>HPDF_FLAG_FLTy_RCPF</code>	规则转换正在进行标志
<code>HPDF_FLAG_FLT0_CKLF0</code>	通道0时钟丢失标志
<code>HPDF_FLAG_FLT0_CKLF1</code>	通道1时钟丢失标志
<code>HPDF_FLAG_FLT0_CKLF2</code>	通道2时钟丢失标志
<code>HPDF_FLAG_FLT0_CKLF3</code>	通道3时钟丢失标志
<code>HPDF_FLAG_FLT0_CKLF4</code>	通道4时钟丢失标志
<code>HPDF_FLAG_FLT0_CKLF5</code>	通道5时钟丢失标志
<code>HPDF_FLAG_FLT0_CKLF6</code>	通道6时钟丢失标志
<code>HPDF_FLAG_FLT0_CKLF7</code>	通道7时钟丢失标志
<code>HPDF_FLAG_FLT0_MMF0</code>	通道0故障事件标志
<code>HPDF_FLAG_FLT0_MMF1</code>	通道1故障事件标志
<code>HPDF_FLAG_FLT0_MMF2</code>	通道2故障事件标志
<code>HPDF_FLAG_FLT0_MMF3</code>	通道3故障事件标志
<code>HPDF_FLAG_FLT0_MMF4</code>	通道4故障事件标志
<code>HPDF_FLAG_FLT0_MMF5</code>	通道5故障事件标志
<code>HPDF_FLAG_FLT0_MMF6</code>	通道6故障事件标志
<code>HPDF_FLAG_FLT0_MMF7</code>	通道7故障事件标志
<code>HPDF_FLAG_FLTy_RCHPDT</code>	规则通道等待处理数据
<code>HPDF_FLAG_FLTy_LTF0</code>	通道0阈值监视器下限阈值标志
<code>HPDF_FLAG_FLTy_LTF1</code>	通道1阈值监视器下限阈值标志
<code>HPDF_FLAG_FLTy_LTF2</code>	通道2阈值监视器下限阈值标志
<code>HPDF_FLAG_FLTy_LTF3</code>	通道3阈值监视器下限阈值标志
<code>HPDF_FLAG_FLTy_LTF4</code>	通道4阈值监视器下限阈值标志
<code>HPDF_FLAG_FLTy_LTF5</code>	通道5阈值监视器下限阈值标志
<code>HPDF_FLAG_FLTy_LTF6</code>	通道6阈值监视器下限阈值标志
<code>HPDF_FLAG_FLTy_LTF7</code>	通道7阈值监视器下限阈值标志
<code>HPDF_FLAG_FLTy_HTF0</code>	通道0阈值监视器上限阈值标志
<code>HPDF_FLAG_FLTy_HTF1</code>	通道1阈值监视器上限阈值标志
<code>HPDF_FLAG_FLTy_HTF2</code>	通道2阈值监视器上限阈值标志
<code>HPDF_FLAG_FLTy_HTF3</code>	通道3阈值监视器上限阈值标志
<code>HPDF_FLAG_FLTy_HTF4</code>	通道4阈值监视器上限阈值标志
<code>HPDF_FLAG_FLTy_HTF5</code>	通道5阈值监视器上限阈值标志

成员名称	功能描述
HPDF_FLAG_FLTY_HTF6	通道6阈值监视器上限阈值标志
HPDF_FLAG_FLTY_HTF7	通道7阈值监视器上限阈值标志

枚举类型 `hpdf_interrupt_flag_enum`

表3-546. 枚举类型 `hpdf_interrupt_flag_enum`

成员名称	功能描述
HPDF_INT_FLAG_FLTY_ICEF	注入转换结束中断标志
HPDF_INT_FLAG_FLTY_RCEF	规则转换结束中断标志
HPDF_INT_FLAG_FLTY_ICDOF	注入转换溢出中断标志
HPDF_INT_FLAG_FLTY_RCDOF	规则转换溢出中断标志
HPDF_INT_FLAG_FLTY_TMEOF	阈值监视器事件中断标志
HPDF_INT_FLAG_FLT0_CKLF0	通道0时钟丢失中断标志
HPDF_INT_FLAG_FLT0_CKLF1	通道1时钟丢失中断标志
HPDF_INT_FLAG_FLT0_CKLF2	通道2时钟丢失中断标志
HPDF_INT_FLAG_FLT0_CKLF3	通道3时钟丢失中断标志
HPDF_INT_FLAG_FLT0_CKLF4	通道4时钟丢失中断标志
HPDF_INT_FLAG_FLT0_CKLF5	通道5时钟丢失中断标志
HPDF_INT_FLAG_FLT0_CKLF6	通道6时钟丢失中断标志
HPDF_INT_FLAG_FLT0_CKLF7	通道7时钟丢失中断标志
HPDF_INT_FLAG_FLT0_MMF0	通道0故障事件中断标志
HPDF_INT_FLAG_FLT0_MMF1	通道1故障事件中断标志
HPDF_INT_FLAG_FLT0_MMF2	通道2故障事件中断标志
HPDF_INT_FLAG_FLT0_MMF3	通道3故障事件中断标志
HPDF_INT_FLAG_FLT0_MMF4	通道4故障事件中断标志
HPDF_INT_FLAG_FLT0_MMF5	通道5故障事件中断标志
HPDF_INT_FLAG_FLT0_MMF6	通道6故障事件中断标志
HPDF_INT_FLAG_FLT0_MMF7	通道7故障事件中断标志

枚举类型 `hpdf_interrupt_enum`

表3-547. 枚举类型 `hpdf_interrupt_enum`

成员名称	功能描述
HPDF_INT_FLTY_ICEIE	使能注入转换结束中断
HPDF_INT_FLTY_RCEIE	使能规则转换结束中断
HPDF_INT_FLTY_ICDOIE	使能注入转换溢出中断
HPDF_INT_FLTY_RCDOIE	使能规则转换溢出中断
HPDF_INT_FLTY_TMIE	使能阈值监视器事件中断
HPDF_INT_FLT0_MMIE	使能故障监视中断
HPDF_INT_FLT0_CKLIE	使能时钟丢失中断

函数 hpdf_deinit

函数hpdf_deinit描述见下表：

表 3-548. 函数 hpdf_deinit

函数名称	hpdf_deinit
函数原型	void hpdf_deinit(void);
功能描述	复位外设HPDF
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 HPDF */
```

```
hpdf_deinit();
```

函数 hpdf_channel_struct_para_init

函数hpdf_channel_struct_para_init描述见下表：

表 3-549. 函数 hpdf_channel_struct_para_init

函数名称	hpdf_channel_struct_para_init
函数原型	void hpdf_channel_struct_para_init(hpdf_channel_parameter_struct* init_struct);
功能描述	初始化HPDF通道结构体参数
先决条件	-
被调用函数	-
输入参数{in}	
*init_struct	HPDF通道初始化结构体，参考 表3-543. 枚举类型hpdf_channel_enum
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* initialize the parameters of HPDF channel */
```

```
hpdf_channel_parameter_struct hpdf_channel_init_struct;
```

```
hpdf_channel_struct_para_init(&hpdf_channel_init_struct);
```

函数 `hpdf_filter_struct_para_init`

函数 `hpdf_filter_struct_para_init` 描述见下表：

表 3-550. 函数 `hpdf_filter_struct_para_init`

函数名称	<code>hpdf_filter_struct_para_init</code>
函数原型	<code>void hpdf_filter_struct_para_init(hpdf_filter_parameter_struct* init_struct);</code>
功能描述	初始化HPDF滤波器结构体参数
先决条件	-
被调用函数	-
输入参数{in}	
<code>*init_struct</code>	HPDF滤波器初始化结构体，参考 表3-540. 结构体 <code>hpdf_filter_parameter_struct</code>
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* initialize the parameters of HPDF filter */
```

```
hpdf_filter_parameter_struct hpdf_filter_init_struct;
```

```
hpdf_filter_struct_para_init(&hpdf_filter_init_struct);
```

函数 `hpdf_rc_struct_para_init`

函数 `hpdf_rc_struct_para_init` 描述见下表：

表 3-551. 函数 `hpdf_rc_struct_para_init`

函数名称	<code>hpdf_rc_struct_para_init</code>
函数原型	<code>void hpdf_rc_struct_para_init(hpdf_rc_parameter_struct* init_struct);</code>
功能描述	初始化规则转换结构体参数
先决条件	-
被调用函数	-
输入参数{in}	
<code>*init_struct</code>	HPDF规则转换初始化结构体，参考 表3-541. 结构体 <code>hpdf_rc_parameter_struct</code>
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* initialize the parameters of regular conversion */
```

```

hpdf_rc_parameter_struct hpdf_rc_init_struct;

hpdf_rc_struct_para_init(&hpdf_rc_init_struct);

```

函数 hpdf_ic_struct_para_init

函数hpdf_ic_struct_para_init描述见下表：

表 3-552. 函数 hpdf_ic_struct_para_init

函数名称	hpdf_ic_struct_para_init
函数原型	void hpdf_ic_struct_para_init(hpdf_ic_parameter_struct* init_struct);
功能描述	初始化注入转换结构体参数
先决条件	-
被调用函数	-
输入参数{in}	
*init_struct	HPDF注入转换初始化结构体，参考 表3-542. 结构体 hpdf_ic_parameter_struct
输出参数{out}	
-	-
返回值	
-	-

例如：

```

/* initialize the parameters of inserted conversion */

hpdf_ic_parameter_struct hpdf_ic_init_struct;

hpdf_ic_struct_para_init(&hpdf_ic_init_struct);

```

函数 hpdf_enable

函数hpdf_enable描述见下表：

表 3-553. 函数 hpdf_enable

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

例如：

```
/* enable the HPDF module */
```

```
hpdf_enable();
```

函数 hpdf_disable

函数hpdf_disable描述见下表：

表 3-554. 函数 hpdf_disable

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

例如：

```
/* disable the HPDF module */
```

```
hpdf_disable();
```

函数 hpdf_channel_init

函数hpdf_channel_init描述见下表：

表 3-555. 函数 hpdf_channel_init

函数名称	hpdf_channel_init
函数原型	void hpdf_channel_init(hpdf_channel_enum channelx, hpdf_channel_parameter_struct* init_struct);
功能描述	初始化HPDF通道
先决条件	-
被调用函数	-
输入参数{in}	
channelx	HPDF模块通道
CHANNELx(x=0..7)	HPDF通道选择，参考 表3-543. 枚举类型hpdf_channel_enum
输入参数{in}	
*init_struct	初始化HPDF通道参数，结构体成员参考 表3-539. 结构体hpdf_channel_parameter_struct
输出参数{out}	

-	-
返回值	
-	-

例如：

```

/* initialize the HPDF channel0 */

hpdf_channel_parameter_struct hpdf_channel_init_struct;

/* initialize HPDF channel0 */

hpdf_channel_init_struct.data_packing_mode = DPM_STANDARD_MODE;

hpdf_channel_init_struct.malfunction_monitor = MM_ENABLE;

hpdf_channel_init_struct.spi_ck_source = EXTERNAL_CKIN;

hpdf_channel_init_struct.channel_muxlexer = SERIAL_INPUT;

hpdf_channel_init_struct.serial_interface = SPI_RISING_EDGE;

hpdf_channel_init_struct.calibration_offset = 0;

hpdf_channel_init_struct.right_bit_shift = 0;

hpdf_channel_init_struct.mm_counter_threshold = 110;

hpdf_channel_init_struct.plsk_value = 0;

hpdf_channel_init(CHANNEL0, &hpdf_channel_init_struct);

```

函数 hpdf_filter_init

函数hpdf_filter_init描述见下表：

表 3-556. 函数 hpdf_filter_init

函数名称	hpdf_filter_init
函数原型	void hpdf_filter_init(hpdf_filter_enum filtery, hpdf_filter_parameter_struct* init_struct);
功能描述	初始化HPDF滤波器
先决条件	-
被调用函数	-
输入参数{in}	
filtery	HPDF模块滤波器
FLTy(y=0..3)	HPDF滤波器选择，参考 表3-544. 枚举类型hpdf_filter_enum
输入参数{in}	
*init_struct	初始化HPDF滤波器参数，结构体成员参考 表3-540. 结构体 hpdf_filter_parameter_struct
输出参数{out}	
-	-

返回值	
-	-

例如：

```
/* initialize the HPDF filter0 */

hpdf_filter_parameter_struct hpdf_filter_init_struct;

hpdf_filter_init_struct.tm_fast_mode = TMFM_DISABLE;

hpdf_filter_init_struct.tm_channel = TMCHEN_CHANNEL0;

hpdf_filter_init_struct.tm_high_threshold = tm_high_val;

hpdf_filter_init_struct.tm_low_threshold = tm_low_val;

hpdf_filter_init_struct.extreme_monitor_channel = EM_CHANNEL0;

hpdf_filter_init_struct.sinc_filter = FLT_SINC3;

hpdf_filter_init_struct.sinc_oversample = FLT_OVER_SAMPLE_32;

hpdf_filter_init_struct.integrator_oversample = INTEGRATOR_BYPASS;

hpdf_filter_init(FLT0, &hpdf_filter_init_struct);
```

函数 hpdf_rc_init

函数hpdf_rc_init描述见下表：

表 3-557. 函数 hpdf_rc_init

函数名称	hpdf_rc_init
函数原型	void hpdf_rc_init(hpdf_filter_enum filtery, hpdf_rc_parameter_struct* init_struct);
功能描述	初始化规则转换
先决条件	-
被调用函数	-
输入参数{in}	
filtery	HPDF模块滤波器
<i>FLTy(y=0..3)</i>	HPDF滤波器选择，参考 表3-544. 枚举类型hpdf_filter_enum
输入参数{in}	
*init_struct	初始化规则转换参数，结构体成员参考 表3-541. 结构体 hpdf_rc_parameter_struct
输出参数{out}	
-	-
返回值	
-	-

例如：

```

/* initialize regular conversion of the HPDF filter0 */

hpdf_rc_parameter_struct hpdf_rc_init_struct;

hpdf_rc_init_struct.fast_mode = FAST_DISABLE;

hpdf_rc_init_struct.rcs_channel = RCS_CHANNEL0;

hpdf_rc_init_struct.rcdmaen = RCDMAEN_ENABLE;

hpdf_rc_init_struct.continuous_mode = RCCM_ENABLE;

hpdf_rc_init(FLT0, &hpdf_rc_init_struct);

```

函数 hpdf_ic_init

函数hpdf_ic_init描述见下表：

表 3-558. 函数 hpdf_ic_init

函数名称	hpdf_ic_init
函数原型	void hpdf_ic_init(hpdf_filter_enum filtery, hpdf_ic_parameter_struct* init_struct);
功能描述	初始化注入转换
先决条件	-
被调用函数	-
输入参数{in}	
filtery	HPDF模块滤波器
FLTy(y=0..3)	HPDF滤波器选择，参考 枚举类型hpdf_filter_enum
输入参数{in}	
*init_struct	初始化注入转换参数，结构体成员参考 表3-542. 结构体 hpdf_ic_parameter_struct
输出参数{out}	
-	-
返回值	
-	-

例如：

```

/* initialize inserted conversion of the HPDF filter0 */

hpdf_ic_parameter_struct hpdf_ic_init_struct;

hpdf_ic_init_struct.ic_channel_group = IGCSEL_CHANNEL0;

hpdf_ic_init_struct.scmmod = SCMOD_ENABLE;

hpdf_ic_init_struct.icdmaen = ICDMAEN_ENABLE;

hpdf_ic_init_struct.icsyn = ICSYN_DISABLE;

hpdf_ic_init(FLT0, &hpdf_ic_init_struct);

```

函数 **hpdf_clock_output_config**

函数hpdf_clock_output_config描述见下表：

表 3-559. 函数 **hpdf_clock_output_config**

函数名称	hpdf_clock_output_config
函数原型	void hpdf_clock_output_config(uint32_t source, uint8_t divider, uint32_t mode);
功能描述	配置串行输出时钟
先决条件	禁止HPDF
被调用函数	-
输入参数{in}	
source	串行输出时钟源
SERIAL_SYSTEM_CLK	串行输出时钟源为系统时钟
SERIAL_SYSTEM_CLK	串行输出时钟源为音频时钟
输入参数{in}	
divider	串行输出时钟分频系数（0-255）
输入参数{in}	
mode	串行输出时钟占空比模式
CKOUTDM_DISABLE	禁止串行输出时钟占空比模式
CKOUTDM_ENABLE	使能串行输出时钟占空比模式
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* configure serial clock output */
```

```
hpdf_clock_output_config(SERIAL_SYSTEM_CLK, 175, CKOUTDM_ENABLE);
```

函数 **hpdf_clock_output_source_config**

函数hpdf_clock_output_source_config描述见下表：

表 3-560. 函数 **hpdf_clock_output_source_config**

函数名称	hpdf_clock_output_source_config
函数原型	void hpdf_clock_output_source_config(uint32_t source);
功能描述	配置串行输出时钟源
先决条件	禁止HPDF
被调用函数	-
输入参数{in}	
source	串行输出时钟源
SERIAL_SYSTEM_CLK	串行输出时钟源为系统时钟

SERIAL_SYSTEM_CLK	串行输出时钟源为音频时钟
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* configure serial clock output source */
```

```
hpdf_clock_output_config(SERIAL_SYSTEM_CLK);
```

函数 hpdf_clock_output_duty_mode_disable

函数hpdf_clock_output_duty_mode_disable描述见下表：

表 3-561. 函数 hpdf_clock_output_duty_mode_disable

函数名称	hpdf_clock_output_duty_mode_disable
函数原型	void hpdf_clock_output_duty_mode_disable(void);
功能描述	禁止串行输出时钟占空比模式
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 serial clock output duty mode*/
```

```
hpdf_clock_output_duty_mode_disable();
```

函数 hpdf_clock_output_duty_mode_enable

函数hpdf_clock_output_duty_mode_enable描述见下表：

表 3-562. 函数 hpdf_clock_output_duty_mode_enable

函数名称	hpdf_clock_output_duty_mode_enable
函数原型	void hpdf_clock_output_duty_mode_enable(void);
功能描述	使能串行输出时钟占空比模式
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 serial clock output duty mode*/
```

```
hpdf_clock_output_duty_mode_enable();
```

函数 hpdf_clock_output_divider_config

函数hpdf_clock_output_divider_config描述见下表：

表 3-563. 函数 hpdf_clock_output_divider_config

函数名称	hpdf_clock_output_divider_config
函数原型	void hpdf_clock_output_divider_config(uint8_t divider);
功能描述	配置串行输出时钟分频系数
先决条件	禁止HPDF
被调用函数	-
输入参数{in}	
divider	串行输出时钟分频系数（0-255）
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* configure serial clock output divider */
```

```
hpdf_clock_output_divider_config (255);
```

函数 hpdf_channel_enable

函数hpdf_channel_enable描述见下表：

表 3-564. 函数 hpdf_channel_enable

函数名称	hpdf_channel_enable
函数原型	void hpdf_channel_enable(hpdf_channel_enum channelx);
功能描述	使能通道
先决条件	-
被调用函数	-
输入参数{in}	
channelx	HPDF模块通道
CHANNELx(x=0..7)	HPDF通道选择，参考 表3-543. 枚举类型hpdf_channel_enum

输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* enable channel0 */
```

```
hpdf_channel_enable(CHANNEL0);
```

函数 hpdf_channel_disable

函数hpdf_channel_disable描述见下表：

表 3-565. 函数 hpdf_channel_disable

函数名称	hpdf_channel_disable
函数原型	void hpdf_channel_disable(hpdf_channel_enum channelx);
功能描述	禁止通道
先决条件	-
被调用函数	-
输入参数{in}	
channelx	HPDF模块通道
CHANNELx(x=0..7)	HPDF通道选择，参考 表3-543. 枚举类型hpdf_channel_enum
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* disable channel0 */
```

```
hpdf_channel_disable(CHANNEL0);
```

函数 hpdf_spi_clock_source_config

函数hpdf_spi_clock_source_config描述见下表：

表 3-566. 函数 hpdf_spi_clock_source_config

函数名称	hpdf_spi_clock_source_config
函数原型	void hpdf_spi_clock_source_config(hpdf_channel_enum channelx, uint32_t clock_source);
功能描述	配置SPI接口时钟源
先决条件	-
被调用函数	-
输入参数{in}	

channelx	HPDF模块通道
<i>CHANNELx(x=0..7)</i>	HPDF通道选择, 参考 表3-543. 枚举类型hpdf_channel_enum
输入参数{in}	
clock_source	SPI接口时钟源
<i>EXTERNAL_CKIN</i>	外部输入时钟
<i>INTERNAL_CKOUT</i>	内部CKOUT时钟
<i>HALF_CKOUT_FALLIN G_EDGE</i>	内部每第二个CKOUT时钟的下降沿
<i>HALF_CKOUT_RISING _EDGE</i>	内部每第二个CKOUT时钟的上升沿
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* configure external input clock as SPI clock source */
```

```
hpdf_spi_clock_source_config(CHANNEL0, EXTERNAL_CKIN);
```

函数 hpdf_serial_interface_type_config

函数hpdf_serial_interface_type_config描述见下表:

表 3-567. 函数 hpdf_serial_interface_type_config

函数名称	hpdf_serial_interface_type_config
函数原型	void hpdf_serial_interface_type_config(hpdf_channel_enum channelx, uint32_t type);
功能描述	配置串行接口类型
先决条件	-
被调用函数	-
输入参数{in}	
channelx	HPDF模块通道
<i>CHANNELx(x=0..7)</i>	HPDF通道选择, 参考 表3-543. 枚举类型hpdf_channel_enum
输入参数{in}	
type	串行接口类型
<i>SPI_RISING_EDGE</i>	SPI接口上升沿采样
<i>SPI_FALLING_EDGE</i>	SPI接口下降沿采样
<i>MANCHESTER_CODE 0</i>	曼切斯特编码输入: 上升沿=逻辑0, 下降沿=逻辑1
<i>MANCHESTER_CODE 1</i>	曼切斯特编码输入: 上升沿=逻辑1, 下降沿=逻辑0
输出参数{out}	
-	-

返回值	
-	-

例如：

```
/*configure external input clock as SPI clock source */
```

```
hpdf_spi_clock_source_config(CHANNEL0, EXTERNAL_CKIN);
```

函数 hpdf_malfunction_monitor_disable

函数hpdf_malfunction_monitor_disable描述见下表：

表 3-568. 函数 hpdf_malfunction_monitor_disable

函数名称	hpdf_malfunction_monitor_disable
函数原型	void hpdf_malfunction_monitor_disable(hpdf_channel_enum channelx);
功能描述	禁止故障检测器
先决条件	-
被调用函数	-
输入参数{in}	
channelx	HPDF模块通道
CHANNELx(x=0..7)	HPDF通道选择，参考 表3-543. 枚举类型hpdf_channel_enum
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* disable malfunction monitor */
```

```
hpdf_malfunction_monitor_disable(CHANNEL0);
```

函数 hpdf_malfunction_monitor_enable

函数hpdf_malfunction_monitor_enable描述见下表：

表 3-569. 函数 hpdf_malfunction_monitor_enable

函数名称	hpdf_malfunction_monitor_enable
函数原型	void hpdf_malfunction_monitor_enable(hpdf_channel_enum channelx);
功能描述	使能故障检测器
先决条件	-
被调用函数	-
输入参数{in}	
channelx	HPDF模块通道
CHANNELx(x=0..7)	HPDF通道选择，参考 表3-543. 枚举类型hpdf_channel_enum
输出参数{out}	

-	-
返回值	
-	-

例如：

```
/* enable malfunction monitor */
```

```
hpdf_malfunction_monitor_enable(CHANNEL1);
```

函数 hpdf_clock_loss_disable

函数hpdf_clock_loss_disable描述见下表：

表 3-570. 函数 hpdf_clock_loss_disable

函数名称	hpdf_clock_loss_disable
函数原型	void hpdf_clock_loss_disable(hpdf_channel_enum channelx);
功能描述	禁止时钟检测
先决条件	CHANNELx(x=0...7)禁止
被调用函数	-
输入参数{in}	
channelx	HPDF模块通道
CHANNELx(x=0..7)	HPDF通道选择，参考 表3-543. 枚举类型hpdf_channel_enum
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* disable clock loss detector */
```

```
hpdf_clock_loss_disable(CHANNEL0);
```

函数 hpdf_clock_loss_enable

函数hpdf_clock_loss_enable描述见下表：

表 3-571. 函数 hpdf_clock_loss_enable

函数名称	hpdf_clock_loss_enable
函数原型	void hpdf_clock_loss_enable(hpdf_channel_enum channelx);
功能描述	使能时钟检测
先决条件	CHANNELx(x=0...7)禁止
被调用函数	-
输入参数{in}	
channelx	HPDF模块通道
CHANNELx(x=0..7)	HPDF通道选择，参考 表3-543. 枚举类型hpdf_channel_enum

输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* enable clock loss detector */
```

```
hpdf_clock_loss_enable(CHANNEL0);
```

函数 hpdf_channel_pin_redirection_disable

函数hpdf_channel_pin_redirection_disable描述见下表：

表 3-572. 函数 hpdf_channel_pin_redirection_disable

函数名称	hpdf_channel_pin_redirection_disable
函数原型	void hpdf_channel_pin_redirection_disable(hpdf_channel_enum channelx);
功能描述	禁止通道引脚重定向
先决条件	CHANNELx(x=0...7)禁止
被调用函数	-
输入参数{in}	
channelx	HPDF模块通道
CHANNELx(x=0..7)	HPDF通道选择，参考 表3-543. 枚举类型hpdf_channel_enum
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* disable channel0 inputs pins redirection */
```

```
hpdf_channel_pin_redirection_disable(CHANNEL0);
```

函数 hpdf_channel_pin_redirection_enable

函数hpdf_channel_pin_redirection_enable描述见下表：

表 3-573. 函数 hpdf_channel_pin_redirection_enable

函数名称	hpdf_channel_pin_redirection_enable
函数原型	void hpdf_channel_pin_redirection_enable(hpdf_channel_enum channelx);
功能描述	使能通道引脚重定向
先决条件	CHANNELx(x=0...7)禁止
被调用函数	-
输入参数{in}	
channelx	HPDF模块通道

CHANNELx(x=0..7)	HPDF通道选择, 参考 表3-543. 枚举类型hpdf_channel_enum
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* enable channel0 inputs pins redirection */
hpdf_channel_pin_redirection_enable(CHANNEL0);
```

函数 hpdf_channel_mux_config

函数hpdf_channel_mux_config描述见下表:

表 3-574. 函数 hpdf_channel_mux_config

函数名称	hpdf_channel_mux_config
函数原型	void hpdf_channel_mux_config(hpdf_channel_enum channelx, uint32_t data_source);
功能描述	配置复用通道输入数据源
先决条件	CHANNELx(x=0...7)禁止
被调用函数	-
输入参数{in}	
channelx	HPDF模块通道
CHANNELx(x=0..7)	HPDF通道选择, 参考 表3-543. 枚举类型hpdf_channel_enum
输入参数{in}	
data_source	输入数据源
SERIAL_INPUT	输入数据源为串行输入数据
INTERNAL_INPUT	输入数据源为内部并行数据寄存器数据
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* configure channel mux select input data source */
hpdf_channel_mux_config(CHANNEL0, SERIAL_INPUT);
```

函数 hpdf_data_pack_mode_config

函数hpdf_data_pack_mode_config描述见下表:

表 3-575. 函数 hpdf_data_pack_mode_config

函数名称	hpdf_data_pack_mode_config
------	----------------------------

函数原型	void hpdf_data_pack_mode_config(hpdf_channel_enum channelx, uint32_t mode);
功能描述	配置数据封装模式
先决条件	CHANNELx(x=0...7)禁止
被调用函数	-
输入参数{in}	
channelx	HPDF模块通道
CHANNELx(x=0..7)	HPDF通道选择, 参考 表3-543. 枚举类型hpdf_channel_enum
输入参数{in}	
mode	数据封装模式
DPM_STANDARD_MODE	标准模式
DPM_INTERLEAVED_MODE	交错模式
DPM_DUAL_MODE	双通道模式
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* configure data packing mode */
```

```
hpdf_data_pack_mode_config(CHANNEL0, DPM_STANDARD_MODE);
```

函数 hpdf_data_right_bit_shift_config

函数hpdf_data_right_bit_shift_config描述见下表:

表 3-576. 函数 hpdf_data_right_bit_shift_config

函数名称	hpdf_data_right_bit_shift_config
函数原型	void hpdf_data_right_bit_shift_config(hpdf_channel_enum channelx, uint8_t right_shift);
功能描述	配置数据封装模式
先决条件	-
被调用函数	-
输入参数{in}	
channelx	HPDF模块通道
CHANNELx(x=0..7)	HPDF通道选择, 参考 表3-543. 枚举类型hpdf_channel_enum
输入参数{in}	
right_shift	数据右移的位数 (0-31)
输出参数{out}	
-	-
返回值	

-	-
---	---

例如：

```
/* configure data right bit-shift */
```

```
hpdf_data_right_bit_shift_config(CHANNEL0, 5);
```

函数 hpdf_calibration_offset_config

函数hpdf_calibration_offset_config描述见下表：

表 3-577. 函数 hpdf_calibration_offset_config

函数名称	hpdf_calibration_offset_config
函数原型	void hpdf_calibration_offset_config(hpdf_channel_enum channelx, int32_t offset);
功能描述	配置偏移校正
先决条件	-
被调用函数	-
输入参数{in}	
channelx	HPDF模块通道
CHANNELx(x=0..7)	HPDF通道选择，参考 表3-543. 枚举类型hpdf_channel_enum
输入参数{in}	
offset	24位偏移校正（-8388608~8388607）
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* configure calibration offset */
```

```
hpdf_calibration_offset_config (CHANNEL0, -255);
```

函数 hpdf_malfunction_break_signal_config

函数hpdf_malfunction_break_signal_config描述见下表：

表 3-578. 函数 hpdf_malfunction_break_signal_config

函数名称	hpdf_malfunction_break_signal_config
函数原型	void hpdf_malfunction_break_signal_config(hpdf_channel_enum channelx, uint32_t break_signal);
功能描述	配置故障检测器断路信号
先决条件	-
被调用函数	-
输入参数{in}	

channelx	HPDF模块通道
<i>CHANNELx(x=0..7)</i>	HPDF通道选择, 参考 表3-543. 枚举类型hpdf_channel_enum
输入参数{in}	
break_signal	数据封装模式
<i>NO_MM_BREAK</i>	无断路信号被分配
<i>MM_BREAK0</i>	断路信号0被分配至所监视的通道
<i>MM_BREAK1</i>	断路信号1被分配至所监视的通道
<i>MM_BREAK2</i>	断路信号2被分配至所监视的通道
<i>MM_BREAK3</i>	断路信号3被分配至所监视的通道
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* configure break signal is distributed to malfunction monitor on channel0 */
```

```
hpdf_data_pack_mode_config(CHANNEL0, MM_BREAK0_1);
```

函数 hpdf_malfunction_counter_config

函数hpdf_malfunction_counter_config描述见下表:

表 3-579. 函数 hpdf_malfunction_counter_config

函数名称	hpdf_malfunction_counter_config
函数原型	void hpdf_malfunction_counter_config(hpdf_channel_enum channelx, uint8_t threshold);
功能描述	配置故障监视器计数器阈值
先决条件	-
被调用函数	-
输入参数{in}	
channelx	HPDF模块通道
<i>CHANNELx(x=0..7)</i>	HPDF通道选择, 参考 表3-543. 枚举类型hpdf_channel_enum
输入参数{in}	
threshold	故障监视器计数器阈值 (0-255)
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* configure malfunction monitor counter threshold */
```

```
hpdf_malfunction_counter_config(CHANNEL0, 255);
```

函数 `hpdf_write_parallel_data_standard_mode`

函数 `hpdf_write_parallel_data_standard_mode` 描述见下表:

表 3-580. 函数 `hpdf_write_parallel_data_standard_mode`

函数名称	<code>hpdf_write_parallel_data_standard_mode</code>
函数原型	<code>void hpdf_write_parallel_data_standard_mode(hpdf_channel_enum channelx, int16_t data);</code>
功能描述	写入数据封装标准模式下的并行数据
先决条件	-
被调用函数	-
输入参数{in}	
channelx	HPDF模块通道
<code>CHANNELx(x=0..7)</code>	HPDF通道选择, 参考 表3-543. 枚举类型hpdf_channel_enum
输入参数{in}	
data	并行数据
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* write the parallel data on standard mode of data packing */
```

```
write the parallel data on standard mode of data packing(CHANNEL0, 0xEFFF);
```

函数 `hpdf_write_parallel_data_interleaved_mode`

函数 `hpdf_write_parallel_data_interleaved_mode` 描述见下表:

表 3-581. 函数 `hpdf_write_parallel_data_interleaved_mode`

函数名称	<code>hpdf_write_parallel_data_interleaved_mode</code>
函数原型	<code>void hpdf_write_parallel_data_interleaved_mode(hpdf_channel_enum channelx, int32_t data);</code>
功能描述	写入数据封装交错模式下的并行数据
先决条件	-
被调用函数	-
输入参数{in}	
channelx	HPDF模块通道
<code>CHANNELx(x=0..7)</code>	HPDF通道选择, 参考 表3-543. 枚举类型hpdf_channel_enum
输入参数{in}	
data	并行数据
输出参数{out}	
-	-

返回值	
-	-

例如：

```
/* write the parallel data on interleaved mode of data packing */
```

```
hpdf_write_parallel_data_interleaved_mode(CHANNEL0, 0xEFFFFFFF);
```

函数 hpdf_write_parallel_data_dual_mode

函数hpdf_write_parallel_data_dual_mode描述见下表：

表 3-582. 函数 hpdf_write_parallel_data_dual_mode

函数名称	hpdf_write_parallel_data_dual_mode
函数原型	void hpdf_write_parallel_data_dual_mode(hpdf_channel_enum channelx, int32_t data);
功能描述	写入数据封装双通道模式下的并行数据
先决条件	-
被调用函数	-
输入参数{in}	
channelx	HPDF模块通道
CHANNELx(x=0..7)	HPDF通道选择，参考 表3-543. 枚举类型hpdf_channel_enum
输入参数{in}	
data	并行数据
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* write the parallel data on dual mode of data packing */
```

```
hpdf_write_parallel_data_dual_mode(CHANNEL0, 0xEFFFFFFF);
```

函数 hpdf_pulse_skip_update

函数hpdf_pulse_skip_update描述见下表：

表 3-583. 函数 hpdf_pulse_skip_update

函数名称	hpdf_pulse_skip_update
函数原型	void hpdf_pulse_skip_update(hpdf_channel_enum channelx, uint8_t number);
功能描述	更新跳频脉冲数量
先决条件	-
被调用函数	-

输入参数{in}	
channelx	HPDF模块通道
<i>CHANNELx(x=0..7)</i>	HPDF通道选择, 参考 表3-543. 枚举类型hpdf_channel_enum
输入参数{in}	
number	将要被跳过的串行输入样本数
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* update the number of pulses to skip */
pdf_pulse_skip_update(CHANNEL0, 63);
```

函数 hpdf_pulse_skip_read

函数hpdf_pulse_skip_read描述见下表:

表 3-584. 函数 hpdf_pulse_skip_read

函数名称	hpdf_pulse_skip_read
函数原型	uint8_t hpdf_pulse_skip_read(hpdf_channel_enum channelx);
功能描述	读取跳频脉冲数量
先决条件	-
被调用函数	-
输入参数{in}	
channelx	HPDF模块通道
<i>CHANNELx(x=0..7)</i>	HPDF通道选择, 参考 表3-543. 枚举类型hpdf_channel_enum
输出参数{out}	
-	-
返回值	
uint8_t	跳频脉冲数量

例如:

```
/* read the number of pulses to skip */
uint8_t value;
value = hpdf_pulse_skip_read(CHANNEL0);
```

函数 hpdf_filter_enable

函数hpdf_filter_enable描述见下表:

表 3-585. 函数 hpdf_filter_enable

函数名称	hpdf_filter_enable
-------------	--------------------

函数原型	void hpdf_filter_enable(hpdf_filter_enum filtery);
功能描述	使能滤波器
先决条件	-
被调用函数	-
输入参数{in}	
filtery	HPDF模块滤波器
FLTy(y=0..3)	HPDF滤波器选择, 参考 表3-544. 枚举类型hpdf_filter_enum
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* enable filter0 */
hpdf_filter_enable(FLT0);
```

函数 hpdf_filter_disable

函数hpdf_filter_disable描述见下表:

表 3-586. 函数 hpdf_filter_disable

函数名称	hpdf_filter_disable
函数原型	void hpdf_filter_disable(hpdf_filter_enum filtery);
功能描述	禁止滤波器
先决条件	-
被调用函数	-
输入参数{in}	
filtery	HPDF模块滤波器
FLTy(y=0..3)	HPDF滤波器选择, 参考 表3-544. 枚举类型hpdf_filter_enum
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* disable filter0 */
hpdf_filter_disable(FLT0);
```

函数 hpdf_filter_config

函数hpdf_filter_config描述见下表:

表 3-587. 函数 `hpdf_filter_config`

函数名称	<code>hpdf_filter_config</code>
函数原型	<code>void hpdf_filter_config(hpdf_filter_enum filtery, uint32_t order, uint16_t oversample);</code>
功能描述	配置滤波器阶数和过采样率
先决条件	-
被调用函数	-
输入参数{in}	
filtery	HPDF模块滤波器
<i>FLTy(y=0..3)</i>	HPDF滤波器选择, 参考 表3-544. 枚举类型hpdf_filter_enum
输入参数{in}	
order	滤波器阶数
<i>FLT_FASTSINC</i>	FastSinc型滤波器
<i>FLT_SINC1</i>	Sinc1型滤波器
<i>FLT_SINC2</i>	Sinc2型滤波器
<i>FLT_SINC3</i>	Sinc3型滤波器
<i>FLT_SINC4</i>	Sinc4型滤波器
<i>FLT_SINC5</i>	Sinc5型滤波器
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* configure sinc filter order and oversample */
```

```
hpdf_filter_config(FLT0, FLT_SINC2, 64);
```

函数 `hpdf_integrator_oversample`

函数`hpdf_integrator_oversample`描述见下表:

表 3-588. 函数 `hpdf_integrator_oversample`

函数名称	<code>hpdf_integrator_oversample</code>
函数原型	<code>void hpdf_integrator_oversample(hpdf_filter_enum filtery, uint8_t oversample);</code>
功能描述	配置积分器过采样率
先决条件	-
被调用函数	-
输入参数{in}	
filtery	HPDF模块滤波器
<i>FLTy(y=0..3)</i>	HPDF滤波器选择, 参考 表3-544. 枚举类型hpdf_filter_enum

输入参数{in}	
oversample	积分器过采样率（1-256）
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* configure integrator oversampling rate */
```

```
hpdf_integrator_oversample(FLT0, 256);
```

函数 hpdf_threshold_monitor_filter_config

函数hpdf_threshold_monitor_filter_config描述见下表：

表 3-589. 函数 hpdf_threshold_monitor_filter_config

函数名称	hpdf_threshold_monitor_filter_config
函数原型	void hpdf_threshold_monitor_filter_config(hpdf_channel_enum channelx, uint32_t order, uint8_t oversample);
功能描述	配置阈值监视器的滤波器
先决条件	
被调用函数	-
输入参数{in}	
channelx	HPDF模块通道
CHANNELx(x=0..7)	HPDF通道选择，参考 表3-543. 枚举类型hpdf_channel_enum
输入参数{in}	
order	滤波器阶数
TM_FASTSINC	FastSinc型滤波器
TM_SINC1	Sinc1型滤波器
TM_SINC2	Sinc2型滤波器
TM_SINC3	Sinc3型滤波器
输入参数{in}	
oversample	滤波器过采样率（1-32）
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* configure threshold monitor filter order and oversample */
```

```
hpdf_threshold_monitor_filter_config (FLT0, TM_SINC3);
```

函数 `hpdf_threshold_monitor_filter_read_data`

函数 `hpdf_threshold_monitor_filter_read_data` 描述见下表：

表 3-590. 函数 `hpdf_threshold_monitor_filter_read_data`

函数名称	<code>hpdf_threshold_monitor_filter_read_data</code>
函数原型	<code>int16_t hpdf_threshold_monitor_filter_read_data(hpdf_channel_enum channelx);</code>
功能描述	读取阈值监视器滤波器的数据
先决条件	
被调用函数	-
输入参数{in}	
channelx	HPDF模块通道
<code>CHANNELx(x=0..7)</code>	HPDF通道选择，参考 表3-543. 枚举类型hpdf_channel_enum
输出参数{out}	
-	-
返回值	
int16_t	阈值监视器滤波器数据

例如：

```
/* read the threshold monitor filter data */
```

```
int16_t data;
```

```
data = hpdf_threshold_monitor_filter_read_data(CHANNEL0);
```

函数 `hpdf_threshold_monitor_fast_mode_disable`

函数 `hpdf_threshold_monitor_fast_mode_disable` 描述见下表：

表 3-591. 函数 `hpdf_threshold_monitor_fast_mode_disable`

函数名称	<code>hpdf_threshold_monitor_fast_mode_disable</code>
函数原型	<code>void hpdf_threshold_monitor_fast_mode_disable(hpdf_filter_enum filtery);</code>
功能描述	禁止阈值监视器快速模式
先决条件	
被调用函数	-
输入参数{in}	
filtery	HPDF模块滤波器
<code>FLTy(y=0..3)</code>	HPDF滤波器选择，参考 表3-544. 枚举类型hpdf_filter_enum
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* disable threshold monitor fast mode */
```

```
hpdf_threshold_monitor_fast_mode_disable (FLT0);
```

函数 hpdf_threshold_monitor_fast_mode_enable

函数hpdf_threshold_monitor_fast_mode_enable描述见下表：

表 3-592. 函数 hpdf_threshold_monitor_fast_mode_enable

函数名称	hpdf_threshold_monitor_fast_mode_enable
函数原型	void hpdf_threshold_monitor_fast_mode_enable(hpdf_filter_enum filtery);
功能描述	使能阈值监视器快速模式
先决条件	-
被调用函数	-
输入参数{in}	
filtery	HPDF模块滤波器
FLTy(y=0..3)	HPDF滤波器选择，参考 表3-544. 枚举类型hpdf_filter_enum
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* enable threshold monitor fast mode */
```

```
hpdf_threshold_monitor_fast_mode_enable(FLT0);
```

函数 hpdf_threshold_monitor_channel

函数hpdf_threshold_monitor_channel描述见下表：

表 3-593. 函数 hpdf_threshold_monitor_channel

函数名称	hpdf_threshold_monitor_channel
函数原型	void hpdf_threshold_monitor_channel(hpdf_filter_enum filtery, uint32_t channel);
功能描述	配置阈值监视器通道
先决条件	-
被调用函数	-
输入参数{in}	
filtery	HPDF模块滤波器
FLTy(y=0..3)	HPDF滤波器选择，参考 表3-544. 枚举类型hpdf_filter_enum
输入参数{in}	
channel	阈值监视器所监视的通道
TMCHEN_DISABLE	禁止所有阈值监视器监视通道
TMCHEN_CHANNEL0	阈值监视器监视通道0

TMCHEN_CHANNEL1	阈值监视器监视通道1
TMCHEN_CHANNEL2	阈值监视器监视通道2
TMCHEN_CHANNEL3	阈值监视器监视通道3
TMCHEN_CHANNEL4	阈值监视器监视通道4
TMCHEN_CHANNEL5	阈值监视器监视通道5
TMCHEN_CHANNEL6	阈值监视器监视通道6
TMCHEN_CHANNEL7	阈值监视器监视通道7
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* configure threshold monitor channel */
```

```
hpdf_threshold_monitor_channel(FLT0, TMCHEN_CHANNEL1);
```

函数 hpdf_threshold_monitor_high_threshold

函数hpdf_threshold_monitor_high_threshold描述见下表：

表 3-594. 函数 hpdf_threshold_monitor_high_threshold

函数名称	hpdf_threshold_monitor_high_threshold
函数原型	void hpdf_threshold_monitor_high_threshold(hpdf_filter_enum filtery, int32_t value);
功能描述	配置阈值监视器上限阈值
先决条件	-
被调用函数	-
输入参数{in}	
filtery	HPDF模块滤波器
FLTy(y=0..3)	HPDF滤波器选择，参考 表3-544. 枚举类型hpdf_filter_enum
输入参数{in}	
value	上限阈值(-8388608~8388607)
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* configure threshold monitor high threshold value */
```

```
hpdf_threshold_monitor_high_threshold(FLT0, 32767);
```

函数 `hpdf_threshold_monitor_low_threshold`

函数 `hpdf_threshold_monitor_low_threshold` 描述见下表：

表 3-595. 函数 `hpdf_threshold_monitor_low_threshold`

函数名称	<code>hpdf_threshold_monitor_low_threshold</code>
函数原型	<code>void hpdf_threshold_monitor_low_threshold(hpdf_filter_enum filtery, int32_t value);</code>
功能描述	配置阈值监视器下限阈值
先决条件	-
被调用函数	-
输入参数{in}	
filtery	HPDF模块滤波器
<i>FLTy(y=0..3)</i>	HPDF滤波器选择，参考 表3-544. 枚举类型hpdf_filter_enum
输入参数{in}	
value	下限阈值(-8388608~8388607)
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* configure threshold monitor low threshold value */
```

```
hpdf_threshold_monitor_low_threshold(FLT0, -32768);
```

函数 `hpdf_high_threshold_break_signal`

函数 `hpdf_high_threshold_break_signal` 描述见下表：

表 3-596. 函数 `hpdf_high_threshold_break_signal`

函数名称	<code>hpdf_high_threshold_break_signal</code>
函数原型	<code>void hpdf_high_threshold_break_signal(hpdf_filter_enum filtery, uint32_t break_signal);</code>
功能描述	配置阈值监视器上限阈值事件断路信号
先决条件	-
被调用函数	-
输入参数{in}	
filtery	HPDF模块滤波器
<i>FLTy(y=0..3)</i>	HPDF滤波器选择，参考 表3-544. 枚举类型hpdf_filter_enum
输入参数{in}	
break_signal	HPDF断路信号
<i>NO_TM_HT_BREAK</i>	禁止断路信号分配给阈值监视器的上限阈值事件
<i>TM_HT_BREAK0</i>	断路信号0分配给阈值监视器的上限阈值事件

<i>TM_HT_BREAK1</i>	断路信号1分配给阈值监视器的上限阈值事件
<i>TM_HT_BREAK2</i>	断路信号2分配给阈值监视器的上限阈值事件
<i>TM_HT_BREAK3</i>	断路信号3分配给阈值监视器的上限阈值事件
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* configure threshold monitor high threshold event break signal */
```

```
hpdf_high_threshold_break_signal(FTL0, TM_HT_BREAK1);
```

函数 `hpdf_low_threshold_break_signal`

函数 `hpdf_low_threshold_break_signal` 描述见下表：

表 3-597. 函数 `hpdf_low_threshold_break_signal`

函数名称	<code>hpdf_low_threshold_break_signal</code>
函数原型	<code>void hpdf_low_threshold_break_signal(hpdf_filter_enum filtery, uint32_t break_signal);</code>
功能描述	配置阈值监视器下限阈值事件断路信号
先决条件	-
被调用函数	-
输入参数{in}	
filtery	HPDF模块滤波器
<i>FLTy</i> (y=0..3)	HPDF滤波器选择，参考 表3-544. 枚举类型 <code>hpdf_filter_enum</code>
输入参数{in}	
break_signal	HPDF断路信号
<i>NO_TM_LT_BREAK</i>	禁止断路信号分配给阈值监视器的下限阈值事件
<i>TM_LT_BREAK0</i>	断路信号0分配给阈值监视器的下限阈值事件
<i>TM_LT_BREAK1</i>	断路信号1分配给阈值监视器的下限阈值事件
<i>TM_LT_BREAK2</i>	断路信号2分配给阈值监视器的下限阈值事件
<i>TM_LT_BREAK3</i>	断路信号3分配给阈值监视器的下限阈值事件
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* configure threshold monitor low threshold event break signal */
```

```
hpdf_low_threshold_break_signal(FTL0, TM_LT_BREAK1);
```

函数 `hpdf_extremes_monitor_channel`

函数 `hpdf_extremes_monitor_channel` 描述见下表：

表 3-598. 函数 `hpdf_extremes_monitor_channel`

函数名称	<code>hpdf_extremes_monitor_channel</code>
函数原型	<code>void hpdf_extremes_monitor_channel(hpdf_filter_enum filtery, uint32_t channel);</code>
功能描述	配置极值监视器通道
先决条件	-
被调用函数	-
输入参数{in}	
filtery	HPDF模块滤波器
<i>FLTy(y=0..3)</i>	HPDF滤波器选择，参考 表3-544. 枚举类型hpdf_filter_enum
输入参数{in}	
channel	极值监视器所监视的通道
<i>EM_CHANNEL_DISABLE</i>	禁止所有极值监视y接收通道的数据
<i>EM_CHANNEL0</i>	极值监视y接收通道0的数据
<i>EM_CHANNEL1</i>	极值监视y接收通道1的数据
<i>EM_CHANNEL2</i>	极值监视y接收通道2的数据
<i>EM_CHANNEL3</i>	极值监视y接收通道3的数据
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* configure extremes monitor channel */
```

```
hpdf_extremes_monitor_channel(FTL0, EM_CHANNEL0_1);
```

函数 `hpdf_extremes_monitor_maximum_get`

函数 `hpdf_extremes_monitor_maximum_get` 描述见下表：

表 3-599. 函数 `hpdf_extremes_monitor_maximum_get`

函数名称	<code>hpdf_extremes_monitor_maximum_get</code>
函数原型	<code>int32_t hpdf_extremes_monitor_maximum_get(hpdf_filter_enum filtery);</code>
功能描述	获取极值监视器最大极值
先决条件	-
被调用函数	-
输入参数{in}	
filtery	HPDF模块滤波器

<i>FLTy</i> (y=0..3)	HPDF滤波器选择, 参考 表3-544. 枚举类型hpdf_filter_enum
输出参数{out}	
-	-
返回值	
int32_t	最大极值

例如:

```
/* get the extremes monitor maximum value */
int32_t vlaue;
vlaue = hpdf_extremes_monitor_maximum_get(FTL0);
```

函数 hpdf_extremes_monitor_minimum_get

函数hpdf_extremes_monitor_minimum_get描述见下表:

表 3-600. 函数 hpdf_extremes_monitor_minimum_get

函数名称	hpdf_extremes_monitor_minimum_get
函数原型	int32_t hpdf_extremes_monitor_minimum_get(hpdf_filter_enum filtery);
功能描述	获取极值监视器最小极值
先决条件	-
被调用函数	-
输入参数{in}	
filtery	HPDF模块滤波器
<i>FLTy</i> (y=0..3)	HPDF滤波器选择, 参考 表3-544. 枚举类型hpdf_filter_enum
输出参数{out}	
-	-
返回值	
int32_t	最小极值

例如:

```
/* get the extremes monitor minimum value */
int32_t vlaue;
vlaue = hpdf_extremes_monitor_minimum_get(FTL0,);
```

函数 hpdf_conversion_time_get

函数hpdf_conversion_time_get描述见下表:

表 3-601. 函数 hpdf_conversion_time_get

函数名称	hpdf_conversion_time_get
函数原型	uint32_t hpdf_conversion_time_get(hpdf_filter_enum filtery);
功能描述	获取转换计时器值

先决条件	-
被调用函数	-
输入参数{in}	
filtery	HPDF模块滤波器
FLTy(y=0..3)	HPDF滤波器选择, 参考 表3-544. 枚举类型hpdf_filter_enum
输出参数{out}	
-	-
返回值	
int32_t	最小极值

例如:

```
/* get the conversion timer value */
int32_t vlaue;

vlaue = hpdf_conversion_time_get(FTL0);
```

函数 hpdf_rc_continuous_disable

函数hpdf_rc_continuous_disable描述见下表:

表 3-602. 函数 hpdf_rc_continuous_disable

函数名称	hpdf_rc_continuous_disable
函数原型	void hpdf_rc_continuous_disable(hpdf_filter_enum filtery);
功能描述	禁止规则转换连续模式
先决条件	-
被调用函数	-
输入参数{in}	
filtery	HPDF模块滤波器
FLTy(y=0..3)	HPDF滤波器选择, 参考 表3-544. 枚举类型hpdf_filter_enum
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* disable regular conversions continuous mode */
hpdf_rc_continuous_disable(FTL0);
```

函数 hpdf_rc_continuous_enable

函数hpdf_rc_continuous_enable描述见下表:

表 3-603. 函数 hpdf_rc_continuous_enable

函数名称	hpdf_rc_continuous_enable
------	---------------------------

函数原型	void hpdf_rc_continuous_enable(hpdf_filter_enum filtery);
功能描述	使能规则转换连续模式
先决条件	-
被调用函数	-
输入参数{in}	
filtery	HPDF模块滤波器
FLTy(y=0..3)	HPDF滤波器选择, 参考 表3-544. 枚举类型hpdf_filter_enum
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* enable regular conversions continuous mode */
```

```
hpdf_rc_continuous_enable(FTL0);
```

函数 hpdf_rc_start_by_software

函数hpdf_rc_start_by_software描述见下表:

表 3-604. 函数 hpdf_rc_start_by_software

函数名称	hpdf_rc_start_by_software
函数原型	void hpdf_rc_start_by_software(hpdf_filter_enum filtery);
功能描述	软件启动规则转换
先决条件	-
被调用函数	-
输入参数{in}	
filtery	HPDF模块滤波器
FLTy(y=0..3)	HPDF滤波器选择, 参考 表3-544. 枚举类型hpdf_filter_enum
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* start regular channel conversion by software */
```

```
hpdf_rc_start_by_software(FTL0);
```

函数 hpdf_rc_syn_disable

函数hpdf_rc_syn_disable描述见下表:

表 3-605. 函数 `hpdf_rc_syn_disable`

函数名称	<code>hpdf_rc_syn_disable</code>
函数原型	<code>void hpdf_rc_syn_disable(hpdf_filter_enum filtery);</code>
功能描述	禁止规则转换同步
先决条件	禁止FLTy(y=0..3)
被调用函数	-
输入参数{in}	
filtery	HPDF模块滤波器
<i>FLTy(y=0..3)</i>	HPDF滤波器选择, 参考 表3-544. 枚举类型hpdf_filter_enum
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* disable regular conversion synchronously */
```

```
hpdf_rc_syn_disable(FTL0);
```

函数 `hpdf_rc_syn_enable`

函数`hpdf_rc_syn_enable`描述见下表:

表 3-606. 函数 `hpdf_rc_syn_disable`

函数名称	<code>hpdf_rc_syn_enable</code>
函数原型	<code>void hpdf_rc_syn_enable(hpdf_filter_enum filtery);</code>
功能描述	使能规则转换同步
先决条件	禁止FLTy(y=0..3)
被调用函数	-
输入参数{in}	
filtery	HPDF模块滤波器
<i>FLTy(y=0..3)</i>	HPDF滤波器选择, 参考 表3-544. 枚举类型hpdf_filter_enum
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* enable regular conversion synchronously */
```

```
hpdf_rc_syn_enable(FTL0);
```

函数 `hpdf_rc_dma_disable`

函数`hpdf_rc_dma_disable`描述见下表:

表 3-607. 函数 `hpdf_rc_dma_disable`

函数名称	<code>hpdf_rc_dma_disable</code>
函数原型	<code>void hpdf_rc_dma_disable(hpdf_filter_enum filtery);</code>
功能描述	禁止规则转换DMA
先决条件	禁止FLTy(y=0..3)
被调用函数	-
输入参数{in}	
filtery	HPDF模块滤波器
<i>FLTy(y=0..3)</i>	HPDF滤波器选择, 参考 表3-544. 枚举类型hpdf_filter_enum
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* disable regular conversion DMA channel */
```

```
hpdf_rc_dma_disable(FTL0);
```

函数 `hpdf_rc_dma_enable`

函数`hpdf_rc_dma_enable`描述见下表:

表 3-608. 函数 `hpdf_rc_dma_enable`

函数名称	<code>hpdf_rc_dma_enable</code>
函数原型	<code>void hpdf_rc_dma_enable(hpdf_filter_enum filtery);</code>
功能描述	使能规则转换DMA
先决条件	禁止FLTy(y=0..3)
被调用函数	-
输入参数{in}	
filtery	HPDF模块滤波器
<i>FLTy(y=0..3)</i>	HPDF滤波器选择, 参考 表3-544. 枚举类型hpdf_filter_enum
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* enable regular conversion DMA channel */
```

```
hpdf_rc_dma_enable(FTL0);
```

函数 `hpdf_rc_channel_config`

函数`hpdf_rc_channel_config`描述见下表:

表 3-609. 函数 `hpdf_rc_channel_config`

函数名称	<code>hpdf_rc_channel_config</code>
函数原型	<code>void hpdf_rc_channel_config(hpdf_filter_enum filtery, hpdf_channel_enum channelx);</code>
功能描述	配置规则转换通道
先决条件	-
被调用函数	-
输入参数{in}	
filtery	HPDF模块滤波器
<i>FLTy(y=0..3)</i>	HPDF滤波器选择, 参考 表3-544. 枚举类型hpdf_filter_enum
输入参数{in}	
channelx	HPDF模块通道
<i>CHANNELx(x=0..7)</i>	HPDF通道选择, 参考 表3-543. 枚举类型hpdf_channel_enum
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* configure regular conversion channel */
hpdf_rc_channel_config(FTL0, CHANNEL1);
```

函数 `hpdf_rc_fast_mode_disable`

函数`hpdf_rc_fast_mode_disable`描述见下表:

表 3-610. 函数 `hpdf_rc_fast_mode_disable`

函数名称	<code>hpdf_rc_fast_mode_disable</code>
函数原型	<code>void hpdf_rc_fast_mode_disable(hpdf_filter_enum filtery);</code>
功能描述	禁止规则转换快速模式
先决条件	禁止 <code>FLTy(y=0..3)</code>
被调用函数	-
输入参数{in}	
filtery	HPDF模块滤波器
<i>FLTy(y=0..3)</i>	HPDF滤波器选择, 参考 表3-544. 枚举类型hpdf_filter_enum
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* disable regular conversion fast conversion mode */
```

hpdf_rc_fast_mode_disable(FTL0);

函数 hpdf_rc_fast_mode_enable

函数hpdf_rc_fast_mode_enable描述见下表：

表 3-611. 函数 hpdf_rc_fast_mode_enable

函数名称	hpdf_rc_fast_mode_enable
函数原型	void hpdf_rc_fast_mode_enable(hpdf_filter_enum filtery);
功能描述	使能规则转换快速模式
先决条件	禁止FLTy(y=0..3)
被调用函数	-
输入参数{in}	
filtery	HPDF模块滤波器
FLTy(y=0..3)	HPDF滤波器选择，参考 表3-544. 枚举类型hpdf_filter_enum
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* enable regular conversion fast conversion mode */
```

```
hpdf_rc_fast_mode_enable(FTL0);
```

函数 hpdf_rc_data_get

函数hpdf_rc_data_get描述见下表：

表 3-612. 函数 hpdf_rc_data_get

函数名称	hpdf_rc_data_get
函数原型	int32_t hpdf_rc_data_get(hpdf_filter_enum filtery);
功能描述	获取规则转换数据
先决条件	-
被调用函数	-
输入参数{in}	
filtery	HPDF模块滤波器
FLTy(y=0..3)	HPDF滤波器选择，参考 表3-544. 枚举类型hpdf_filter_enum
输出参数{out}	
-	-
返回值	
int32_t	规则转换数据

例如：

```
/* get the regular conversion data */
```

```
int32_t vlaue;
```

```
vlaue = hpdf_rc_data_get(FTL0);
```

函数 hpdf_rc_channel_get

函数hpdf_rc_channel_get描述见下表：

表 3-613. 函数 hpdf_rc_channel_get

函数名称	hpdf_rc_channel_get
函数原型	uint8_t hpdf_rc_channel_get(hpdf_filter_enum filtery);
功能描述	获取最近一次规则转换的通道
先决条件	-
被调用函数	-
输入参数{in}	
filtery	HPDF模块滤波器
FLTy(y=0..3)	HPDF滤波器选择，参考 表3-544. 枚举类型hpdf_filter_enum
输出参数{out}	
-	-
返回值	
uint8_t	通道

例如：

```
/* get the channel of regular channel most recently converted */
```

```
uint8_t channel;
```

```
channel = hpdf_rc_channel_get(FTL0);
```

函数 hpdf_ic_start_by_software

函数hpdf_ic_start_by_software描述见下表：

表 3-614. 函数 hpdf_ic_start_by_software

函数名称	hpdf_ic_start_by_software
函数原型	void hpdf_ic_start_by_software(hpdf_filter_enum filtery);
功能描述	软件启动注入转换
先决条件	-
被调用函数	-
输入参数{in}	
filtery	HPDF模块滤波器
FLTy(y=0..3)	HPDF滤波器选择，参考 表3-544. 枚举类型hpdf_filter_enum
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* start inserted channel conversion by software */
```

```
hpdf_ic_start_by_software(FTL0);
```

函数 hpdf_ic_syn_disable

函数hpdf_ic_syn_disable描述见下表：

表 3-615. 函数 hpdf_ic_syn_disable

函数名称	hpdf_ic_syn_disable
函数原型	void hpdf_ic_syn_disable(hpdf_filter_enum filtery);
功能描述	禁止注入转换同步
先决条件	禁止FLTy(y=0..3)
被调用函数	-
输入参数{in}	
filtery	HPDF模块滤波器
FLTy(y=0..3)	HPDF滤波器选择，参考 表3-544. 枚举类型hpdf_filter_enum
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* disable inserted conversion synchronously */
```

```
hpdf_ic_syn_disable(FTL0);
```

函数 hpdf_ic_syn_enable

函数hpdf_ic_syn_enable描述见下表：

表 3-616. 函数 hpdf_ic_syn_enable

函数名称	hpdf_ic_syn_enable
函数原型	void hpdf_ic_syn_enable(hpdf_filter_enum filtery);
功能描述	使能注入转换同步
先决条件	禁止FLTy(y=0..3)
被调用函数	-
输入参数{in}	
filtery	HPDF模块滤波器
FLTy(y=0..3)	HPDF滤波器选择，参考 表3-544. 枚举类型hpdf_filter_enum
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* enable inserted conversion synchronously */
```

```
hpdf_ic_syn_enable(FTL0);
```

函数 hpdf_ic_dma_disable

函数hpdf_ic_dma_disable描述见下表：

表 3-617. 函数 hpdf_ic_dma_disable

函数名称	hpdf_ic_dma_disable
函数原型	void hpdf_ic_dma_disable(hpdf_filter_enum filtery);
功能描述	禁止注入转换DMA
先决条件	禁止FLTy(y=0..3)
被调用函数	-
输入参数{in}	
filtery	HPDF模块滤波器
FLTy(y=0..3)	HPDF滤波器选择，参考 表3-544. 枚举类型hpdf_filter_enum
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* disable inserted conversion DMA channel */
```

```
hpdf_ic_dma_disable(FTL0);
```

函数 hpdf_ic_dma_enable

函数hpdf_ic_dma_enable描述见下表：

表 3-618. 函数 hpdf_ic_dma_enable

函数名称	hpdf_ic_dma_enable
函数原型	void hpdf_ic_dma_enable(hpdf_filter_enum filtery);
功能描述	使能注入转换DMA
先决条件	禁止FLTy(y=0..3)
被调用函数	-
输入参数{in}	
filtery	HPDF模块滤波器
FLTy(y=0..3)	HPDF滤波器选择，参考 表3-544. 枚举类型hpdf_filter_enum
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* enable inserted conversion DMA channel */
```

```
hpdf_ic_dma_enable(FTL0);
```

函数 hpdf_ic_scan_mode_disable

函数hpdf_ic_scan_mode_disable描述见下表:

表 3-619. 函数 hpdf_ic_scan_mode_disable

函数名称	hpdf_ic_scan_mode_disable
函数原型	void hpdf_ic_scan_mode_disable(hpdf_filter_enum filtery);
功能描述	禁止注入转换扫描模式
先决条件	禁止FLTy(y=0..3)
被调用函数	-
输入参数{in}	
filtery	HPDF模块滤波器
FLTy(y=0..3)	HPDF滤波器选择, 参考 表3-544. 枚举类型hpdf_filter_enum
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* disable scan conversion mode */
```

```
hpdf_ic_scan_mode_disable(FTL0);
```

函数 hpdf_ic_scan_mode_enable

函数hpdf_ic_scan_mode_enable描述见下表:

表 3-620. 函数 hpdf_ic_scan_mode_enable

函数名称	hpdf_ic_scan_mode_enable
函数原型	void hpdf_ic_scan_mode_enable(hpdf_filter_enum filtery);
功能描述	使能注入转换扫描模式
先决条件	禁止FLTy(y=0..3)
被调用函数	-
输入参数{in}	
filtery	HPDF模块滤波器
FLTy(y=0..3)	HPDF滤波器选择, 参考 表3-544. 枚举类型hpdf_filter_enum
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* enable scan conversion mode */
hpdf_ic_scan_mode_enable(FTL0);
```

函数 hpdf_ic_trigger_signal_disable

函数hpdf_ic_trigger_signal_disable描述见下表：

表 3-621. 函数 hpdf_ic_trigger_signal_disable

函数名称	hpdf_ic_trigger_signal_disable
函数原型	void hpdf_ic_trigger_signal_disable(hpdf_filter_enum filtery);
功能描述	禁止注入转换触发信号
先决条件	禁止FLTy(y=0..3)
被调用函数	-
输入参数{in}	
filtery	HPDF模块滤波器
FLTy(y=0..3)	HPDF滤波器选择，参考 表3-544. 枚举类型hpdf_filter_enum
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* disable inserted conversions trigger signal */
hpdf_ic_trigger_signal_disable(FTL0);
```

函数 hpdf_ic_trigger_signal_config

函数hpdf_ic_trigger_signal_config描述见下表：

表 3-622. 函数 hpdf_ic_trigger_signal_config

函数名称	hpdf_ic_trigger_signal_config
函数原型	void hpdf_ic_trigger_signal_config(hpdf_filter_enum filtery, uint32_t trigger, uint32_t trigger_edge);
功能描述	配置注入转换触发信号和边沿
先决条件	禁止FLTy(y=0..3)
被调用函数	-
输入参数{in}	
filtery	HPDF模块滤波器
FLTy(y=0..3)	HPDF滤波器选择，参考 表3-544. 枚举类型hpdf_filter_enum
输入参数{in}	
trigger	注入转换触发信号
HPDF_ITRG0	注入转换触发信号为TIMER0_TRGO0

HPDF_ITRG1	注入转换触发信号为TIMER0_TRGO1
HPDF_ITRG2	注入转换触发信号为TIMER7_TRGO0
HPDF_ITRG3	注入转换触发信号为TIMER7_TRGO1
HPDF_ITRG4	注入转换触发信号为TIMER2_TRGO0
HPDF_ITRG5	注入转换触发信号为TIMER3_TRGO0
HPDF_ITRG6	注入转换触发信号为TIMER15_CH1
HPDF_ITRG7	注入转换触发信号为TIMER5_TRGO0
HPDF_ITRG8	注入转换触发信号为TIMER6_TRGO0
HPDF_ITRG11	注入转换触发信号为TIMER22_TRGO0
HPDF_ITRG12	注入转换触发信号为TIMER23_TRGO0
HPDF_ITRG24	注入转换触发信号为EXTI11
HPDF_ITRG25	注入转换触发信号为EXTI15
HPDF_ITRG31	注入转换触发信号为HPDF_ITRG
输入参数{in}	
trigger_edge	注入转换触发边沿
TRG_DISABLE	禁止触发信号
RISING_EDGE_TRG	触发信号上升沿
FALLING_EDGE_TRG	触发信号下降沿
EDGE_TRG	触发信号双边沿
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* configure inserted conversions trigger signal and trigger edge */
```

```
hpdf_ic_trigger_signal_config(FTL0, HPDF_ITRG3, FALLING_EDGE_TRG);
```

函数 hpdf_ic_channel_config

函数hpdf_ic_channel_config描述见下表:

表 3-623. 函数 hpdf_ic_channel_config

函数名称	hpdf_ic_channel_config
函数原型	void hpdf_ic_channel_config(hpdf_filter_enum filtery, uint32_t channel);
功能描述	配置注入组转换通道
先决条件	-
被调用函数	-
输入参数{in}	
filtery	HPDF模块滤波器
FLTy(y=0..3)	HPDF滤波器选择, 参考 表3-544. 枚举类型hpdf_filter_enum
输入参数{in}	
channel	注入组通道

IGCSEL_CHANNEL0	通道0属于注入组
IGCSEL_CHANNEL1	通道1属于注入组
IGCSEL_CHANNEL2	通道2属于注入组
IGCSEL_CHANNEL3	通道3属于注入组
IGCSEL_CHANNEL4	通道4属于注入组
IGCSEL_CHANNEL5	通道5属于注入组
IGCSEL_CHANNEL6	通道6属于注入组
IGCSEL_CHANNEL7	通道7属于注入组
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* configure inserted group conversions channel */
```

```
hpdf_ic_channel_config(FTL0, IGCSEL_CHANNEL2);
```

函数 hpdf_ic_data_get

函数hpdf_ic_data_get描述见下表：

表 3-624. 函数 hpdf_ic_data_get

函数名称	hpdf_ic_data_get
函数原型	int32_t hpdf_ic_data_get(hpdf_filter_enum filtery);
功能描述	获取注入转换数据
先决条件	-
被调用函数	-
输入参数{in}	
filtery	HPDF模块滤波器
FLTy(y=0..3)	HPDF滤波器选择，参考 表3-544. 枚举类型hpdf_filter_enum
输出参数{out}	
-	-
返回值	
int32_t	注入转换数据

例如：

```
/* get the inserted conversions data */
```

```
int32_t vlaue;
```

```
vlaue = hpdf_ic_data_get(FTL0);
```

函数 `hpdf_ic_channel_get`

函数 `hpdf_ic_channel_get` 描述见下表：

表 3-625. 函数 `hpdf_ic_channel_get`

函数名称	<code>hpdf_ic_channel_get</code>
函数原型	<code>uint8_t hpdf_ic_channel_get(hpdf_filter_enum filtery);</code>
功能描述	获取最近一次注入转换的通道
先决条件	-
被调用函数	-
输入参数{in}	
filtery	HPDF模块滤波器
<i>FLTy(y=0..3)</i>	HPDF滤波器选择，参考 表3-544. 枚举类型hpdf_filter_enum
输出参数{out}	
-	-
返回值	
uint8_t	通道

例如：

```
/* get the channel of inserted group channel most recently converted */
```

```
uint8_t channel;
```

```
channel = hpdf_ic_channel_get(FTL0);
```

函数 `hpdf_flag_get`

函数 `hpdf_flag_get` 描述见下表：

表 3-626. 函数 `hpdf_flag_get`

函数名称	<code>hpdf_flag_get</code>
函数原型	<code>FlagStatus hpdf_flag_get(hpdf_filter_enum filtery, hpdf_flag_enum flag);</code>
功能描述	获取HPDF标志位
先决条件	-
被调用函数	-
输入参数{in}	
filtery	HPDF模块滤波器
<i>FLTy(y=0..3)</i>	HPDF滤波器选择，参考 表3-544. 枚举类型hpdf_filter_enum
输入参数{in}	
flag	HPDF模块标志位
<i>HPDF_FLAG_FLTy_IC EF</i>	注入转换结束标志
<i>HPDF_FLAG_FLTy_RC EF</i>	规则转换结束标志
<i>HPDF_FLAG_FLTy_IC</i>	注入转换溢出标志

<i>DOF</i>	
<i>HPDF_FLAG_FLTy_RC</i> <i>DOF</i>	规则转换溢出标志
<i>HPDF_FLAG_FLTy_TM</i> <i>EOF</i>	阈值监视器事件标志
<i>HPDF_FLAG_FLTy_IC</i> <i>PF</i>	注入转换正在进行标志
<i>HPDF_FLAG_FLTy_RC</i> <i>PF</i>	规则转换正在进行标志
<i>HPDF_FLAG_FLT0_CK</i> <i>LFx</i>	通道(x = 0..7)时钟丢失标志
<i>HPDF_FLAG_FLT0_M</i> <i>MFx</i>	通道(x = 0..7)故障事件标志
<i>HPDF_FLAG_FLTy_RC</i> <i>HPDT</i>	规则通道等待处理数据
<i>HPDF_FLAG_FLTy_LT</i> <i>Fx</i>	通道(x = 0..7)阈值监视器下限阈值标志
<i>HPDF_FLAG_FLTy_HT</i> <i>Fx</i>	通道(x = 0..7)阈值监视器上限阈值标志
输出参数{out}	
-	-
返回值	
FlagStatus	SET或RESET

例如:

```
/* get the FLT0 threshold monitor event occurred flag */
```

```
hpdf_flag_get(FTL0, HPDF_FLAG_FLTy_TMEOF);
```

函数 hpdf_flag_clear

函数hpdf_flag_clear描述见下表:

表 3-627. 函数 hpdf_flag_clear

函数名称	hpdf_flag_clear
函数原型	void hpdf_flag_clear(hpdf_filter_enum filtery, hpdf_flag_enum flag);
功能描述	清楚HPDF标志位
先决条件	-
被调用函数	-
输入参数{in}	
filtery	HPDF模块滤波器
<i>FLTy</i> (y=0..3)	HPDF滤波器选择, 参考 表3-544. 枚举类型hpdf_filter_enum
输入参数{in}	
flag	HPDF模块标志位

HPDF_FLAG_FLTy_IC EF	注入转换结束标志
HPDF_FLAG_FLTy_RC EF	规则转换结束标志
HPDF_FLAG_FLTy_IC DOF	注入转换溢出标志
HPDF_FLAG_FLTy_RC DOF	规则转换溢出标志
HPDF_FLAG_FLTy_TM EOF	阈值监视器事件标志
HPDF_FLAG_FLT0_CK LFx	通道(x = 0..7)时钟丢失标志
HPDF_FLAG_FLT0_M MFx	通道(x = 0..7)故障事件标志
HPDF_FLAG_FLTy_LT Fx	通道(x = 0..7)阈值监视器下限阈值标志
HPDF_FLAG_FLTy_HT Fx	通道(x = 0..7)阈值监视器上限阈值标志
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* clear the FLT0 threshold monitor event occurred flag */
```

```
hpdf_flag_clear(FTL0, HPDF_FLAG_FLTy_TMEOF);
```

函数 hpdf_interrupt_enable

函数hpdf_interrupt_enable描述见下表：

表 3-628. 函数 hpdf_interrupt_enable

函数名称	hpdf_interrupt_enable
函数原型	void hpdf_interrupt_enable(hpdf_filter_enum filtery, hpdf_interrput_enum interrupt);
功能描述	使能HPDF中断
先决条件	-
被调用函数	-
输入参数{in}	
filtery	HPDF模块滤波器
FLTy(y=0..3)	HPDF滤波器选择，参考 表3-544. 枚举类型hpdf_filter_enum
输入参数{in}	
interrupt	HPDF模块中断

HPDF_INT_FLTy_ICEI E	使能注入转换结束中断
HPDF_INT_FLTy_RCEI E	使能规则转换结束中断
HPDF_INT_FLTy_ICD OIE	使能注入转换溢出中断
HPDF_INT_FLTy_RCD OIE	使能规则转换溢出中断
HPDF_INT_FLTy_TMIE	使能阈值监视器事件中断
HPDF_INT_FLT0_MMI E	使能故障监视中断
HPDF_INT_FLT0_CKLI E	使能时钟丢失中断
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* enable FLT0 threshold monitor interrupt */
```

```
hpdf_interrupt_enable(FTL0, HPDF_INT_FLTy_TMEIOIE);
```

函数 hpdf_interrupt_disable

函数hpdf_interrupt_disable描述见下表：

表 3-629. 函数 hpdf_interrupt_disable

函数名称	hpdf_interrupt_disable
函数原型	void hpdf_interrupt_disable(hpdf_filter_enum filtery, hpdf_interrput_enum interrupt);
功能描述	禁止HPDF中断
先决条件	-
被调用函数	-
输入参数{in}	
filtery	HPDF模块滤波器
FLTy(y=0..3)	HPDF滤波器选择，参考 表3-544. 枚举类型hpdf_filter_enum
输入参数{in}	
interrupt	HPDF模块中断
HPDF_INT_FLTy_ICEI E	使能注入转换结束中断
HPDF_INT_FLTy_RCEI E	使能规则转换结束中断
HPDF_INT_FLTy_ICD	使能注入转换溢出中断

OIE	
HPDF_INT_FLTy_RCD OIE	使能规则转换溢出中断
HPDF_INT_FLTy_TMIE	使能阈值监视器事件中断
HPDF_INT_FLT0_MMIE	使能故障监视中断
HPDF_INT_FLT0_CKLI E	使能时钟丢失中断
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* disable FLT0 threshold monitor interrupt */
```

```
hpdf_interrupt_disable(FLT0, HPDF_INT_FLT0_CKLI);
```

函数 hpdf_interrupt_flag_get

函数hpdf_interrupt_flag_get描述见下表：

表 3-630. 函数 hpdf_interrupt_flag_get

函数名称	hpdf_interrupt_flag_get
函数原型	FlagStatus hpdf_interrupt_flag_get(hpdf_filter_enum filtery, hpdf_interrput_flag_enum int_flag);
功能描述	获取HPDF中断标志位
先决条件	-
被调用函数	-
输入参数{in}	
filtery	HPDF模块滤波器
FLTy(y=0..3)	HPDF滤波器选择，参考 表3-544. 枚举类型hpdf_filter_enum
输入参数{in}	
int_flag	HPDF模块中断标志位
HPDF_INT_FLAG_FLTy_ICEF	注入转换结束中断标志
HPDF_INT_FLAG_FLTy_RCEF	规则转换结束中断标志
HPDF_INT_FLAG_FLTy_ICDOF	注入转换溢出中断标志
HPDF_INT_FLAG_FLTy_RCDOF	规则转换溢出中断标志
HPDF_INT_FLAG_FLTy_TMEOF	阈值监视器事件中断标志

HPDF_INT_FLAG_FLT0_CKLFx	通道(x = 0..7)时钟丢失中断标志
HPDF_INT_FLAG_FLT0_MMFx	通道(x = 0..7)故障事件中断标志
输出参数{out}	
-	-
返回值	
FlagStatus	SET或RESET

例如:

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

```
hpdf_interrupt_flag_get (FTL0, HPDF_INT_FLAG_FLT0_CKLF0);
```

函数 hpdf_interrupt_flag_clear

函数hpdf_interrupt_flag_clear描述见下表:

表 3-631. 函数 hpdf_interrupt_flag_clear

函数名称	hpdf_interrupt_flag_clear
函数原型	void hpdf_interrupt_flag_clear(hpdf_filter_enum filtery, hpdf_interrput_flag_enum int_flag);
功能描述	清楚HPDF中断标志位
先决条件	-
被调用函数	-
输入参数{in}	
filtery	HPDF模块滤波器
FLTy(y=0..3)	HPDF滤波器选择, 参考 表3-544. 枚举类型hpdf_filter_enum
输入参数{in}	
int_flag	HPDF模块中断标志位
HPDF_INT_FLAG_FLTy_ICEF	注入转换结束中断标志
HPDF_INT_FLAG_FLTy_RCEF	规则转换结束中断标志
HPDF_INT_FLAG_FLTy_ICDOF	注入转换溢出中断标志
HPDF_INT_FLAG_FLTy_RCDOF	规则转换溢出中断标志
HPDF_INT_FLAG_FLTy_TMEOF	阈值监视器事件中断标志
HPDF_INT_FLAG_FLT0_CKLFx	通道(x = 0..7)时钟丢失中断标志
HPDF_INT_FLAG_FLT0_MMFx	通道(x = 0..7)故障事件中断标志

输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* clear the clock loss interrupt flag */
```

```
hpdf_interrupt_flag_get(FTL0, HPDF_INT_FLAG_FLT0_CKLF0);
```

3.20. 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[3.20.1](#)描述了I2C的寄存器列表，章节[3.20.2](#)对I2C库函数进行说明。

3.20.1. 外设寄存器说明

I2C寄存器列表如下表所示：

表 3-632. 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

3.20.2. 外设库函数说明

I2C库函数列表如下表所示：

表 3-633. 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

库函数名称	库函数描述
i2c_digital_noise_filter_config	配置数字噪声过滤器
i2c_analog_noise_filter_enable	使能数字噪声过滤器
i2c_analog_noise_filter_disable	禁能数字噪声过滤器
i2c_master_clock_config	配置主机模式下 SCL 高低电平时钟周期
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_receved_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_nack_disable	从机模式下产生 ACK
i2c_wakeup_from_deepsleep_enable	使能从 Deep-sleep 模式中唤醒
i2c_wakeup_from_deepsleep_disable	禁止从 Deep-sleep 模式中唤醒
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

库函数名称	库函数描述
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_extented_clock_timeout_enable	使能时钟信号延展超时检测
i2c_extented_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 3-634. 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 3-635. 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=0,1,2,3)
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 3-636. 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=0,1,2,3)
输入参数{in}	
psc	0-0xf, 时序分频
输入参数{in}	
scl_dely	0-0xf, 数据建立时间
输入参数{in}	
sda_dely	0-0xf, 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 3-637. 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=0,1,2,3)
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 3-638. 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=0,1,2,3)
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 3-639. 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=0,1,2,3)
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 3-640. 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=0,1,2,3)
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 3-641. 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

<i>I2Cx</i>	(x=0,1,2,3)
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
<i>I2C_MASTER_TRANS MIT</i>	主机发送
<i>I2C_MASTER_RECEIV E</i>	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 3-642. 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
<i>I2Cx</i>	(x=0,1,2,3)
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 3-643. 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=0,1,2,3)
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 3-644. 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=0,1,2,3)
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 3-645. 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=0,1,2,3)
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 3-646. 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=0,1,2,3)
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 3-647. 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=0,1,2,3)
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 3-648. 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=0,1,2,3)
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 3-649. 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=0,1,2,3)
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 3-650. 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=0,1,2,3)
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 3-651. 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=0,1,2,3)
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 3-652. 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=0,1,2,3)
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 3-653. 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=0,1,2,3)
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 3-654. 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=0,1,2,3)
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 3-655. 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=0,1,2,3)
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
ADDRESS2_MASK_ALL	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 3-656. 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=0,1,2,3)
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_receved_address_get

函数i2c_receved_address_get描述见下表：

表 3-657. 函数 i2c_receved_address_get

函数名称	i2c_receved_address_get
函数原型	uint32_t i2c_receved_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
I2Cx	(x=0,1,2,3)
输出参数{out}	
-	-
返回值	
uint32_t	0x00..0x7F

例如：

```
/* get received match address in slave mode */
```

```
uint32_t address;
```

```
address = i2c_receved_address_get(I2C0);
```

函数 i2c_slave_byte_control_enable

函数i2c_slave_byte_control_enable描述见下表：

表 3-658. 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
I2Cx	(x=0,1,2,3)
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 3-659. 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=0,1,2,3)
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 3-660. 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=0,1,2,3)
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* generate a NACK in slave mode */
```

```
i2c_nack_enable(I2C0);
```

函数 i2c_nack_disable

函数i2c_nack_disable描述见下表：

表 3-661. 函数 i2c_nack_disable

函数名称	i2c_nack_disable
函数原型	void i2c_nack_disable(uint32_t i2c_periph);
功能描述	从机模式下产生 ACK
先决条件	-
被调用函数	-
输入参数{in}	
i2c_periph	I2C 外设
I2Cx	(x=0,1,2,3)
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* generate a ACK in slave mode */
```

```
i2c_nack_disable(I2C0);
```

函数 i2c_wakeup_from_deepsleep_enable

函数i2c_wakeup_from_deepsleep_enable描述见下表：

表 3-662. 函数 i2c_wakeup_from_deepsleep_enable

函数名称	i2c_wakeup_from_deepsleep_enable
函数原型	void i2c_wakeup_from_deepsleep_enable(uint32_t i2c_periph);
功能描述	使能从Deep-sleep模式中唤醒
先决条件	-
被调用函数	-
输入参数{in}	
i2c_periph	I2C外设
I2Cx	(x=0,1,2,3)
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* enable wakeup from Deep-sleep mode */
```

```
i2c_wakeup_from_deepsleep_enable(I2C0);
```

函数 i2c_wakeup_from_deepsleep_disable

函数i2c_wakeup_from_deepsleep_disable描述见下表：

表 3-663. 函数 i2c_wakeup_from_deepsleep_disable

函数名称	i2c_wakeup_from_deepsleep_disable
函数原型	void i2c_wakeup_from_deepsleep_disable(uint32_t i2c_periph);
功能描述	禁止从Deep-sleep模式中唤醒
先决条件	-
被调用函数	-
输入参数{in}	
i2c_periph	I2C外设
I2Cx	(x=0,1,2,3)
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* disable wakeup from Deep-sleep mode */
```

```
i2c_wakeup_from_deepsleep_disable(I2C0);
```

函数 i2c_enable

函数i2c_enable描述见下表：

表 3-664. 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=0,1,2,3)
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 3-665. 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=0,1,2,3)
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 3-666. 函数 i2c_start_on_bus

函数名称	i2c_start_on_bus
函数原型	void i2c_start_on_bus(uint32_t i2c_periph);
功能描述	在 I2C 总线上生成起始位
先决条件	-
被调用函数	-
输入参数{in}	
i2c_periph	I2C 外设
I2Cx	(x=0,1,2,3)
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 3-667. 函数 i2c_stop_on_bus

函数名称	i2c_stop_on_bus
函数原型	void i2c_stop_on_bus(uint32_t i2c_periph);
功能描述	在 I2C 总线上生成停止位
先决条件	-
被调用函数	-
输入参数{in}	
i2c_periph	I2C 外设
I2Cx	(x=0,1,2,3)
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 3-668. 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=0,1,2,3)
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 3-669. 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=0,1,2,3)
输出参数{out}	
-	-
返回值	
uint32_t	0x0000..0x00FF

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 3-670. 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=0,1,2,3)
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 3-671. 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	禁用 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=0,1,2,3)
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 3-672. 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

<i>I2Cx</i>	(x=0,1,2,3)
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 3-673. 函数 i2c_dma_enable

函数名称	i2c_dma_enable
函数原型	void i2c_dma_enable(uint32_t i2c_periph, uint8_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
<i>I2Cx</i>	(x=0,1,2,3)
输入参数{in}	
dma	I2C DMA
<i>I2C_DMA_TRANSMIT</i>	采用 DMA 方式发送数据
<i>I2C_DMA_RECEIVE</i>	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 3-674. 函数 i2c_dma_disable

函数名称	i2c_dma_disable
函数原型	void i2c_dma_disable(uint32_t i2c_periph, uint8_t dma);
功能描述	禁能发送/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=0,1,2,3)
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 3-675. 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=0,1,2,3)
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 3-676. 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=0,1,2,3)
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 3-677. 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=0,1,2,3)
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 3-678. 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=0,1,2,3)
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 3-679. 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=0,1,2,3)
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 3-680. 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=0,1,2,3)
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 3-681. 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=0,1,2,3)
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 3-682. 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=0,1,2,3)
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 3-683. 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=0,1,2,3)
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 3-684. 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=0,1,2,3)
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* disable SMBus Host address */
```

```
i2c_smbus_host_addr_disable(I2C0);
```

函数 i2c_extented_clock_timeout_enable

函数i2c_extented_clock_timeout_enable描述见下表：

表 3-685. 函数 i2c_extented_clock_timeout_enable

函数名称	i2c_extented_clock_timeout_enable
函数原型	void i2c_extented_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=0,1,2,3)
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* enable extended clock timeout detection */
```

```
i2c_extented_clock_timeout_enable(I2C0);
```

函数 i2c_extented_clock_timeout_disable

函数i2c_extented_clock_timeout_disable描述见下表：

表 3-686. 函数 i2c_extented_clock_timeout_disable

函数名称	i2c_extented_clock_timeout_disable
函数原型	void i2c_extented_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=0,1,2,3)
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* disable extended clock timeout detection */
```

```
i2c_extented_clock_timeout_disable(I2C0);
```

函数 i2c_clock_timeout_enable

函数i2c_clock_timeout_enable描述见下表：

表 3-687. 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=0,1,2,3)
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 3-688. 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=0,1,2,3)
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 3-689. 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=0,1,2,3)
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 3-690. 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=0,1,2,3)
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 3-691. 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
I2Cx	(x=0,1,2,3)

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 3-692. 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=0,1,2,3)
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 3-693. 函数 i2c_flag_clear

函数名称	i2c_flag_clear
函数原型	void i2c_flag_clear(uint32_t i2c_periph, uint32_t flag);
功能描述	清除 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=0,1,2,3)
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 3-694. 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
I2Cx	(x=0,1,2,3)
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：

```
/* 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 3-695. 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=0,1,2,3)
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 3-696. 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=0,1,2,3)
输入参数{in}	
int_flag	I2C 中断标志位，参考 表 3-634. 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

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_SMBAL	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 3-697. 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=0,1,2,3)
输入参数{in}	
int_flag	I2C 中断标志位，参考 表 3-634. 枚举类型 i2c_interrupt_flag_enum 。
I2C_INT_FLAG_ADDS	从机模式下，接收到的地址与自身地址匹配中断标志

<i>END</i>	
<i>I2C_INT_FLAG_NACK</i>	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_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	
-	-

例如：

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

```
i2c_interrupt_flag_clear(I2C0, I2C_INT_FLAG_BERR);
```

3.21. LPDTS

低功耗数字温度传感器（LPDTS），提供了将温度转换为频率与绝对温度（CLK_PTAT）成正比的方波。章节[3.21.1](#)描述了LPDTS的寄存器列表，章节[3.21.2](#)对LPDTS库函数进行说明。

3.21.1. 外设寄存器说明

LPDTS寄存器列表如下表所示：

表 3-698. LPDTS 寄存器

寄存器名称	寄存器描述
LPDTS_CFG	配置寄存器
LPDTS_SDATA	传感器数据寄存器
LPDTS_RDATA	斜率数据寄存器
LPDTS_IT	中断阈值寄存器
LPDTS_DATA	温度值寄存器
LPDTS_STAT	温度传感器状态寄存器
LPDTS_INTEN	中断使能寄存器

寄存器名称	寄存器描述
LPDTS_INTC	中断标志清除寄存器
LPDTS_OP	选择寄存器

3.21.2. 外设库函数说明

LPDTS库函数列表如下表所示：

表 3-699. LPDTS 库函数

库函数名称	库函数描述
lpdts_deinit	复位外设LPDTS
lpdts_struct_para_init	初始化LPDTS结构体中所有参数为默认值
lpdts_init	初始化外设LPDTS
lpdts_enable	使能外设LPDTS
lpdts_disable	失能外设LPDTS
lpdts_soft_trigger_enable	使能LPDTS软件触发
lpdts_soft_trigger_disable	失能LPDTS软件触发
lpdts_high_threshold_set	设置LPDTS中断高阈值
lpdts_low_threshold_set	设置LPDTS中断低阈值
lpdts_ref_clock_source_config	设置LPDTS时钟源
lpdts_temperature_get	获取外设LPDTS采集温度值
lpdts_flag_get	获取外设LPDTS状态
lpdts_interrupt_enable	使能LPDTS中断
lpdts_interrupt_disable	禁能LPDTS中断
lpdts_interrupt_flag_get	获取外设LPDTS中断状态
lpdts_interrupt_flag_clear	清除外设LPDTS中断状态

结构体 lpdts_parameter_struct

表 3-700. 结构体 lpdts_parameter_struct

成员名称	功能描述
ref_clock	参考时钟选择 (REF_PCLK, REF_LXTAL)
trigger_input	输入触发源选择 (NO_HARDWARE_TRIGGER, LPDTS_TRG)
sampling_time	采样时间设置 (SPT_CLOCK_1, SPT_CLOCK_2, SPT_CLOCK_3, SPT_CLOCK_4, SPT_CLOCK_5, SPT_CLOCK_6, SPT_CLOCK_7, SPT_CLOCK_8, SPT_CLOCK_9, SPT_CLOCK_10, SPT_CLOCK_11, SPT_CLOCK_12, SPT_CLOCK_13, SPT_CLOCK_14, SPT_CLOCK_15)

函数 lpdts_deinit

函数lpdts_deinit描述见下表：

表 3-701. 函数 lpdts_deinit

函数名称	lpdts_deinit
函数原形	void lpdts_deinit(void);
功能描述	复位外设LPDTS
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 LPDTS registers */
```

```
lpdts_deinit();
```

函数 lpdts_struct_para_init

函数lpdts_struct_para_init描述见下表：

表 3-702. 函数 lpdts_struct_para_init

函数名称	lpdts_struct_para_init
函数原形	void lpdts_struct_para_init(lpdts_parameter_struct* init_struct);
功能描述	初始化LPDTS结构体中所有参数为默认值
先决条件	-
被调用函数	-
输入参数{in}	
init_struct	LPDTS初始化结构体，结构体成员参考 表3-700. 结构体 lpdts_parameter_struct
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* initialize the parameters of LPDTS */
```

```
lpdts_parameter_struct lpdts_init_struct;
```

```
lpdts_struct_para_init(&lpdts_init_struct);
```

函数 lpdts_init

函数lpdts_init描述见下表：

表 3-703. 函数 lpdts_init

函数名称	lpdts_init
函数原形	void lpdts_init(lpdts_parameter_struct* init_struct);
功能描述	初始化外设LPDTS
先决条件	-
被调用函数	-
输入参数{in}	
init_struct	LPDTS初始化结构体，结构体成员参考 表3-700. 结构体 lpdts_parameter_struct
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* initialize the LPDTS */

lpdts_parameter_struct lpdts_init_struct;

lpdts_init(&lpdts_init_struct);
```

函数 lpdts_enable

函数lpdts_enable描述见下表：

表 3-704. 函数 lpdts_enable

函数名称	lpdts_enable
函数原形	void lpdts_enable(void);
功能描述	使能外设LPDTS
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 LPDTS temperature sensor */

lpdts_enable();
```

函数 lpdts_disable

函数lpdts_disable描述见下表：

表 3-705. 函数 lpdts_disable

函数名称	lpdts_disable
函数原形	void lpdts_disable(void);
功能描述	失能外设LPDTS
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 LPDTS temperature sensor */
```

```
lpdts_disable();
```

函数 lpdts_soft_trigger_enable

函数lpdts_soft_trigger_enable描述见下表：

表 3-706. 函数 lpdts_soft_trigger_enable

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

例如：

```
/* software trigger start */
```

```
lpdts_soft_trigger_enable();
```

函数 lpdts_soft_trigger_disable

函数lpdts_soft_trigger_disable描述见下表:

表 3-707. 函数 lpdts_soft_trigger_disable

函数名称	lpdts_soft_trigger_disable
函数原形	void lpdts_soft_trigger_disable(void);
功能描述	失能LPDTS软件触发
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 LPDTS software trigger */
```

```
lpdts_soft_trigger_disable();
```

函数 lpdts_high_threshold_set

函数lpdts_high_threshold_set描述见下表:

表 3-708. 函数 lpdts_high_threshold_set

函数名称	lpdts_high_threshold_set
函数原形	void lpdts_high_threshold_set(uint16_t value);
功能描述	设置LPDTS中断高阈值
先决条件	-
被调用函数	-
输入参数{in}	
value	中断高阈值
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* set threshold value */
```

```
lpdts_high_threshold_set(0U);
```


函数 lpdts_low_threshold_set

函数lpdts_low_threshold_set描述见下表：

表 3-709. 函数 lpdts_low_threshold_set

函数名称	lpdts_low_threshold_set
函数原形	void lpdts_low_threshold_set(uint16_t value);
功能描述	设置LPDTS中断低阈值
先决条件	-
被调用函数	-
输入参数{in}	
value	中断低阈值
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* set threshold value */
```

```
lpdts_low_threshold_set(0U);
```

函数 lpdts_ref_clock_source_config

函数lpdts_ref_clock_source_config描述见下表：

表 3-710. 函数 lpdts_ref_clock_source_config

函数名称	lpdts_ref_clock_source_config
函数原形	void lpdts_ref_clock_source_config(uint32_t source);
功能描述	设置LPDTS时钟源
先决条件	-
被调用函数	-
输入参数{in}	
source	时钟源
REF_PCLK	高速参考时钟
REF_LXTAL	低速参考时钟
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* select reference clock */
```

```
lpdts_ref_clock_source_config(REF_PCLK);
```

函数 `lpdts_temperature_get`

函数 `lpdts_temperature_get` 描述见下表：

表 3-711. 函数 `lpdts_temperature_get`

函数名称	<code>lpdts_temperature_get</code>
函数原形	<code>int32_t lpdts_temperature_get(void);</code>
功能描述	获取外设LPDTS采集温度值
先决条件	-
被调用函数	-
输入参数{in}	
-	-
输出参数{out}	
-	-
返回值	
<code>int32_t</code>	最终获得的温度值

例如：

```
int32_t temperature;
```

```
/* wait till TSRF flag is set */
```

```
while(RESET == lpdts_flag_get(LPPTS_FLAG_TSRF)){
}
```

```
temperature = lpdts_temperature_get();
```

函数 `lpdts_flag_get`

函数 `lpdts_flag_get` 描述见下表：

表 3-712. 函数 `lpdts_flag_get`

函数名称	<code>lpdts_flag_get</code>
函数原形	<code>FlagStatus lpdts_flag_get(uint32_t flag);</code>
功能描述	获取外设LPDTS状态
先决条件	-
被调用函数	-
输入参数{in}	
<code>flag</code>	标志位
<code>LPPTS_FLAG_TSR</code>	温度传感器准备标志位
输出参数{out}	
-	-
返回值	
<code>FlagStatus</code>	SET或RESET

例如：

```
/* wait till TSRF flag is set */  
  
while(RESET == lpdts_flag_get(LPPTS_FLAG_TSR)) {  
  
}
```

函数 lpdts_interrupt_enable

函数lpdts_interrupt_enable描述见下表：

表 3-713. 函数 lpdts_interrupt_enable

函数名称	lpdts_interrupt_enable
函数原形	void lpdts_interrupt_enable(uint32_t interrupt);
功能描述	使能LPDTS中断
先决条件	-
被调用函数	-
输入参数{in}	
interrupt	中断种类
LPDTS_INT_EM	测量完成中断（与PCLK同步）
LPDTS_INT_LT	低阈值中断（与PCLK同步）
LPDTS_INT_HT	高阈值中断（与PCLK同步）
LPDTS_INT_EMA	测量完成异步中断
LPDTS_INT_LTA	低阈值异步中断
LPDTS_INT_HTA	高阈值异步中断
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* enable end of measurement interrupt */  
  
lpdts_interrupt_enable(LPPTS_INT_EM);
```

函数 lpdts_interrupt_disable

函数lpdts_interrupt_disable描述见下表：

表 3-714. 函数 lpdts_interrupt_disable

函数名称	lpdts_interrupt_disable
函数原形	void lpdts_interrupt_disable(uint32_t interrupt);
功能描述	失能LPDTS中断
先决条件	-
被调用函数	-
输入参数{in}	
interrupt	中断种类

LPDTS_INT_EM	测量完成中断（与PCLK同步）
LPDTS_INT_LT	低阈值中断（与PCLK同步）
LPDTS_INT_HT	高阈值中断（与PCLK同步）
LPDTS_INT_EMA	测量完成异步中断
LPDTS_INT_LTA	低阈值异步中断
LPDTS_INT-HTA	高阈值异步中断
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* enable end of measurement interrupt */
```

```
lpdts_interrupt_disable(LPOTS_INT_EM);
```

函数 lpdts_interrupt_flag_get

函数lpdts_interrupt_flag_get描述见下表：

表 3-715. 函数 lpdts_interrupt_flag_get

函数名称	lpdts_interrupt_flag_get
函数原形	FlagStatus lpdts_interrupt_flag_get(uint32_t flag);
功能描述	获取外设LPDTS中断状态
先决条件	-
被调用函数	-
输入参数{in}	
flag	中断标志位
LPDTS_INT_FLAG_EM	测量完成中断标志位
LPDTS_INT_FLAG_LT	低阈值中断标志位
LPDTS_INT_FLAG-HT	高阈值中断标志位
LPDTS_INT_FLAG_EMA	测量完成异步中断标志位
LPDTS_INT_FLAG_LTA	低阈值异步中断标志位
LPDTS_INT_FLAG-HTA	高阈值异步中断标志位
输出参数{out}	
-	-
返回值	
FlagStatus	SET或RESET

例如：

```
/* wait till HTA flag is set */
```

```
while(RESET == lpdts_interrupt_flag_get(LPOTS_INT_FLAG-HTA)) {
}
```

函数 `lpdts_interrupt_flag_clear`

函数 `lpdts_interrupt_flag_clear` 描述见下表：

表 3-716. 函数 `lpdts_interrupt_flag_clear`

函数名称	<code>lpdts_interrupt_flag_clear</code>
函数原形	<code>void lpdts_flag_interrupt_clear(uint32_t flag);</code>
功能描述	清除外设LPDTS中断状态
先决条件	-
被调用函数	-
输入参数{in}	
flag	中断标志位
<code>LPDTS_INT_FLAG_EM</code>	测量完成中断标志位
<code>LPDTS_INT_FLAG_LT</code>	低阈值中断标志位
<code>LPDTS_INT_FLAG_HT</code>	高阈值中断标志位
<code>LPDTS_INT_FLAG_EMA</code>	测量完成异步中断标志位
<code>LPDTS_INT_FLAG_LTA</code>	低阈值异步中断标志位
<code>LPDTS_INT_FLAG-HTA</code>	高阈值异步中断标志位
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* clear HTA flag */
lpdts_flag_clear(LPDTTS_INT_FLAG-HTA);
```

3.22. MDMA

MDMA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[3.22.1](#)描述了MDMA的寄存器列表，章节[3.22.2](#)对MDMA库函数进行说明。

3.22.1. 外设寄存器描述

MDMA寄存器列表如下表所示：

表 3-717. MDMA 寄存器

寄存器名称	寄存器描述
<code>MDMA_GINTF</code>	全局中断标志寄存器
<code>MDMA_CHxSTAT0</code> (x=0..15)	通道x状态寄存器0
<code>MDMA_CHxSTATC</code>	通道x状态清除寄存器

寄存器名称	寄存器描述
(x=0..15)	
MDMA_CHxSTAT1 (x=0..15)	通道x状态寄存器1
MDMA_CHxCTL0 (x=0..15)	通道x控制寄存器0
MDMA_CHxCFG (x=0..15)	通道x配置寄存器
MDMA_CHxBTCFG (x=0..15)	通道x块传输配置寄存器
MDMA_CHxSADDR (x=0..15)	通道x源地址寄存器
MDMA_CHxDADDR (x=0..15)	通道x目的地址寄存器
MDMA_CHxMBAD DRU (x=0..15)	通道x多块地址更新寄存器
MDMA_CHxLADDR (x=0..15)	通道x链路地址寄存器
MDMA_CHxCTL1 (x=0..15)	通道x控制寄存器1
MDMA_CHxMADDR R (x=0..15)	通道x掩码地址寄存器
MDMA_CHxMDATA (x=0..15)	通道x掩码数据寄存器

3.22.2. 外设库函数说明

MDMA库函数列表如下表所示：

表 3-718. MDMA 库函数

库函数名称	库函数描述
mdma_deinit	复位外设 MDMA
mdma_channel_deinit	复位外设 MDMA 通道 x 的所有寄存器
mdma_para_struct_init	将 MDMA 结构体中所有参数初始化为默认值
mdma_multi_block_para_struct_init	将 MDMA 多块传输模式结构体中所有参数初始化为默认值
mdma_link_node_para_struct_init	将 MDMA 链表节点结构体中所有参数初始化为默认值
mdma_init	初始化外设 MDMA 的通道 x
mdma_buffer_block_mode_config	配置 MDMA 缓冲区/块传输模式
mdma_multi_block_mode_config	配置 MDMA 缓冲多块传输模式
mdma_node_create	创建 MDMA 链路列表节点
mdma_node_add	MDMA 添加节点到链路列表

库函数名称	库函数描述
mdma_node_delete	MDMA 断开链路列表节点
mdma_destination_address_config	配置 MDMA 目标基地址
mdma_source_address_config	配置 MDMA 源基地址
mdma_destination_bus_config	配置 MDMA 目标总线
mdma_source_bus_config	配置 MDMA 源总线
mdma_priority_config	配置 MDMA 通道 x 优先级
mdma_endianness_config	配置 MDMA 通道 x 数据大小端
mdma_alignment_config	配置 MDMA 通道 x 数据对齐方式
mdma_source_burst_beats_config	配置源突发传输类型
mdma_destination_burst_beats_config	配置目标突发传输类型
mdma_source_width_config	配置源数据长度
mdma_destination_width_config	配置目的数据长度
mdma_source_increment_config	配置源地址增长模式
mdma_destination_increment_config	配置目的地址增长模式
mdma_channel_bufferable_write_enable	使能 MDMA 通道 x 可缓冲写模式
mdma_channel_bufferable_write_disable	禁能 MDMA 通道 x 可缓冲写模式
mdma_channel_software_request_enable	使能 MDMA 通道 x 软件请求
mdma_channel_enable	使能 MDMA 通道 x
mdma_channel_disable	禁能 MDMA 通道 x
mdma_transfer_error_direction_get	获取 MDMA 传输错误方向
mdma_transfer_error_address_get	获取 MDMA 传输错误地址
mdma_flag_get	获取 MDMA 标志
mdma_flag_clear	清除 MDMA 标志
mdma_interrupt_enable	使能 MDMA 中断
mdma_interrupt_disable	禁能 MDMA 中断
mdma_interrupt_flag_get	获取 MDMA 中断标志
mdma_interrupt_flag_clear	清除 MDMA 中断标志

结构体 mdma_parameter_struct

表 3-719. 结构体 mdma_parameter_struct

成员名称	功能描述
request	指定 MDMA 请求
trans_trig_mode	指定触发传输方式
priority	指定 MDMA 通道 x 的软件优先级
endianness	指定 MDMA 传输是否保留小的字节顺序
source_inc	指定源增量方式
dest_inc	指定目标增量方式

成员名称	功能描述
source_data_size	指定源数据大小
dest_data_size	指定目标数据大小
data_alignment	指定源到目标内存数据封装/填充模式
buff_trans_len	指定缓冲区传输长度（字节数）
source_burst	指定源存储器传输的突发传输配置
dest_burst	指定目标存储器传输的突发传输配置
mask_addr	掩码地址
mask_data	掩码数据
source_addr	指定源地址
dest_addr	指定目标地址
tbytes_num_in_block	指定缓冲区或块传输中的传输字节数
source_bus	指定源总线
dest_bus	指定目标总线
bufferable_write_mode	指定可缓冲写模式

结构体 mdma_multi_block_parameter_struct

表 3-720. 结构体 mdma_multi_block_parameter_struct

成员名称	功能描述
block_num	多块的块数量
saddr_update_val	源地址更新值
dstaddr_update_val	目标地址更新值
saddr_update_dir	源地址更新方向
dstaddr_update_dir	目的地址更新方向

结构体 mdma_link_node_parameter_struct

表 3-721. 结构体 mdma_link_node_parameter_struct

成员名称	功能描述
chxcfg_reg	通道x配置寄存器
chxbtcfg_reg	通道x块传输配置寄存器
chxsaddr_reg	通道x源地址寄存器
chxdaddr_reg	通道x目的地址寄存器
chxmbaddru_reg	通道x多块地址更新寄存器
chxladdr_reg	通道x链路地址寄存器
chxctl1_reg	通道x控制寄存器1
reserved	通道x保留寄存器
chxmaddr_reg	通道x掩码地址寄存器
chxmdata_reg	通道x掩码数据寄存器

枚举 mdma_add_update_dir_enum

表 3-722. 枚举 mdma_add_update_dir_enum

成员名称	功能描述
UPDATE_DIR_INC REASE	MDMA地址向上更新
UPDATE_DIR_DEC REASE	MDMA地址向下更新

枚举 mdma_channel_enum

表 3-723. 枚举 mdma_channel_enum

成员名称	功能描述
MDMA_CH0	MDMA通道0
MDMA_CH1	MDMA通道1
MDMA_CH2	MDMA通道2
MDMA_CH3	MDMA通道3
MDMA_CH4	MDMA通道4
MDMA_CH5	MDMA通道5
MDMA_CH6	MDMA通道6
MDMA_CH7	MDMA通道7
MDMA_CH8	MDMA通道8
MDMA_CH9	MDMA通道9
MDMA_CH10	MDMA通道10
MDMA_CH11	MDMA通道11
MDMA_CH12	MDMA通道12
MDMA_CH13	MDMA通道13
MDMA_CH14	MDMA通道14
MDMA_CH15	MDMA通道15

函数 mdma_deinit

函数mdma_deinit描述见下表:

表 3-724. 函数 mdma_deinit

函数名称	mdma_deinit
函数原型	void mdma_deinit(void);
功能描述	复位外设MDMA
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 MDMA */
```

```
mdma_deinit();
```

函数 mdma_channel_deinit

函数mdma_channel_deinit描述见下表：

表 3-725. 函数 mdma_channel_deinit

函数名称	mdma_channel_deinit
函数原型	void mdma_channel_deinit(mdma_channel_enum channelx);
功能描述	复位外设MDMA通道x的所有寄存器
先决条件	-
被调用函数	-
输入参数{in}	
channelx	MDMA通道
MDMA_CHx(x=0..15)	MDMA通道选择，参考 表3-723. 枚举mdma_channel_enum 。
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* deinitialize MDMA channel 0 */
```

```
mdma_channel_deinit(MDMA_CH0);
```

函数 mdma_para_struct_init

函数mdma_para_struct_init描述见下表：

表 3-726. 函数 mdma_para_struct_init

函数名称	mdma_para_struct_init
函数原型	void mdma_para_struct_init(mdma_parameter_struct *init_struct);
功能描述	将MDMA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}	
init_struct	初始化MDMA通道所需的初始化数据，参考 表3-719. 结构体

	<u>mdma_parameter_struct</u> 。
返回值	
-	-

例如：

```
/* initialize the MDMA single data mode parameters struct with the default values */
mdma_parameter_struct mdma_init_struct;
mdma_para_struct_init(&mdma_init_struct);
```

函数 mdma_multi_block_para_struct_init

函数mdma_multi_block_para_struct_init描述见下表：

表 3-727. 函数 mdma_multi_block_para_struct_init

函数名称	mdma_multi_block_para_struct_init
函数原型	void mdma_multi_block_para_struct_init(mdma_multi_block_parameter_struct *block_init_struct);
功能描述	将MDMA多块传输模式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}	
block_init_struct	初始化MDMA通道所需的初始化数据，参考 <u>表3-720. 结构体 mdma_multi_block_parameter_struct</u> 。
返回值	
-	-

例如：

```
/* initialize the MDMA multi block transfer mode parameters struct with the default values */
mdma_multi_block_parameter_struct mdma_multi_block_init_struct;
mdma_multi_block_para_struct_init(&mdma_multi_block_init_struct);
```

函数 mdma_link_node_para_struct_init

函数mdma_link_node_para_struct_init描述见下表：

表 3-728. 函数 mdma_link_node_para_struct_init

函数名称	mdma_link_node_para_struct_init
函数原型	void mdma_link_node_para_struct_init(mdma_link_node_parameter_struct *node);

功能描述	将MDMA链表节点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}	
node	将MDMA链表节点结构体中所有参数初始化为默认值，参考 表3-721. 结构体 <i>mdma_link_node_parameter_struct</i> 。
返回值	
-	-

例如：

```
/* initialize the MDMA link node configuration struct with the default values */
```

```
mdma_link_node_parameter_struct node;
```

```
mdma_link_node_para_struct_init (&node);
```

函数 mdma_init

函数mdma_init描述见下表：

表 3-729. 函数 mdma_init

函数名称	mdma_init
函数原型	void mdma_init(mdma_channel_enum channelx, mdma_parameter_struct *init_struct);
功能描述	初始化外设MDMA的通道x
先决条件	-
被调用函数	-
输入参数{in}	
channelx	MDMA通道
MDMA_CHx(x=0..15)	MDMA通道选择，参考 表3-723. 枚举mdma_channel_enum 。
输入参数{in}	
init_struct	初始化MDMA通道所需的初始化数据，参考 表3-719. 结构体 <i>mdma_parameter_struct</i> 。
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* initialize MDMA channel 0 */
```

```
mdma_para_struct_init(&mdma_init_struct);
```

```
mdma_init_struct.request = MDMA_REQUEST_DMA1_CHANNEL2_FTFIF;
```

```
mdma_init_struct.transfer_trigger_mode = MDMA_BUFFER_TRANSFER;

mdma_init_struct.priority = MDMA_PRIORITY_HIGH;

mdma_init_struct.endianness = MDMA_LITTLE_ENDIANNESS;

mdma_init_struct.source_addr = (uint32_t)&usart_rxbuffer;

mdma_init_struct.dest_addr = mdma_rxbuffer_addr;

mdma_init_struct.source_inc = MDMA_SOURCE_INCREASE_8BIT;

mdma_init_struct.dest_inc = MDMA_DESTINATION_INCREASE_8BIT;

mdma_init_struct.source_data_size = MDMA_SOURCE_DATASIZE_8BIT;

mdma_init_struct.dest_data_size = MDMA_DESTINATION_DATASIZE_8BIT;

mdma_init_struct.source_burst = MDMA_SOURCE_BURST_SINGLE;

mdma_init_struct.dest_burst = MDMA_DESTINATION_BURST_SINGLE;

mdma_init_struct.source_bus = MDMA_SOURCE_AXI;

mdma_init_struct.dest_bus = MDMA_DESTINATION_AHB_TCM;

mdma_init_struct.data_alignment = MDMA_DATAALIGN_PKEN;

mdma_init_struct.buffertransfer_length = 10U;

mdma_init_struct.tbytes_num_in_block = 10U;

mdma_init_struct.mask_addr = DMA1_INTC_ADDRESS;

mdma_init_struct.mask_data = DMA1_INTC_FTFIFC2;

mdma_init(MDMA_CH0, &mdma_init_struct);
```

函数 mdma_buffer_block_mode_config

函数mdma_buffer_block_mode_config描述见下表：

表 3-730. 函数 mdma_buffer_block_mode_config

函数名称	mdma_buffer_block_mode_config
函数原型	void mdma_buffer_block_mode_config(mdma_channel_enum channelx, uint32_t saddr, uint32_t daddr, uint32_t tbnun);
功能描述	配置MDMA缓冲区/块传输模式
先决条件	-
被调用函数	-
输入参数{in}	
channelx	MDMA通道
MDMA_CHx(x=0..15)	MDMA通道选择，参考 表3-723. 枚举mdma_channel_enum 。
输入参数{in}	

saddr	源地址, 0x00000000-0xFFFFFFFF
输入参数{in}	
daddr	目标地址, 0x00000000-0xFFFFFFFF
输入参数{in}	
tbnun	待传输字节数, 0x00000000-0x00010000
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* configure MDMA buffer/block transfer mode */
```

```
mdma_buffer_block_mode_config(0x08000000, 0x24100000, 256);
```

函数 mdma_multi_block_mode_config

函数mdma_multi_block_mode_config描述见下表:

表 3-731. 函数 mdma_multi_block_mode_config

函数名称	mdma_multi_block_mode_config
函数原型	void mdma_multi_block_mode_config(mdma_channel_enum channelx, uint32_t tbnun, mdma_multi_block_parameter_struct *block_init_struct);
功能描述	配置MDMA缓冲多块传输模式
先决条件	-
被调用函数	-
输入参数{in}	
channelx	MDMA通道
<i>MDMA_CHx(x=0..15)</i>	MDMA通道选择, 参考 表3-723. 枚举mdma_channel_enum 。
输入参数{in}	
tbnun	块中要传输的字节数, 0x00000000-0x00000FFF
输入参数{in}	
block_init_struct	初始化MDMA通道所需的初始化数据, 参考 表3-720. 结构体mdma_multi_block_parameter_struct 。
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* configure MDMA multi block transfer mode */
```

```
mdma_multi_block_parameter_struct block_init_struct;
```

```
mdma_multi_block_mode_config(MDMA_CH0, 0xFF, & block_init_struct);
```

函数 `mdma_node_create`

函数`mdma_node_create`描述见下表：

表 3-732. 函数 `mdma_node_create`

函数名称	<code>mdma_node_create</code>
函数原型	<code>void mdma_node_create(mdma_link_node_parameter_struct *node, mdma_multi_block_parameter_struct *block_init_struct, mdma_parameter_struct *init_struct);</code>
功能描述	创建MDMA链路列表节点
先决条件	-
被调用函数	-
输入参数{in}	
<code>block_init_struct</code>	初始化MDMA通道所需的初始化数据，参考 表3-720. 结构体 <code>mdma_multi_block_parameter_struct</code> 。
输入参数{in}	
<code>init_struct</code>	初始化MDMA通道所需的初始化数据，参考 表3-719. 结构体 <code>mdma_parameter_struct</code> 。
输出参数{out}	
<code>node</code>	链路节点结构体，结构体成员可参考 表3-721. 结构体 <code>mdma_link_node_parameter_struct</code> 。
返回值	
-	-

例如：

```
/* create MDMA link list node */
mdma_link_node_parameter_struct node;
mdma_multi_block_parameter_struct block_init_struct;
mdma_parameter_struct init_struct;
mdma_node_create(&node, &block_init_struct, &init_struct);
```

函数 `mdma_node_add`

函数`mdma_node_add`描述见下表：

表 3-733. 函数 `mdma_node_add`

函数名称	<code>mdma_node_add</code>
函数原型	<code>void mdma_node_add(mdma_link_node_parameter_struct *pre_node, mdma_link_node_parameter_struct *new_node);</code>
功能描述	MDMA添加节点到链路列表
先决条件	-
被调用函数	-

输入参数{in}	
pre_node	前一个节点的结构体，结构体成员可参考 表3-721. 结构体 mdma_link_node_parameter_struct 。
输入参数{in}	
new_node	新节点的结构体，结构体成员可参考 表3-721. 结构体 mdma_link_node_parameter_struct 。
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* MDMA add node to link list */

mdma_link_node_parameter_struct pre_node;

mdma_link_node_parameter_struct new_node;

mdma_node_add(&pre_node, &new_node);
```

函数 mdma_node_delete

函数mdma_node_delete描述见下表：

表 3-734. 函数 mdma_node_delete

函数名称	mdma_node_delete
函数原型	ErrStatus mdma_node_delete(mdma_link_node_parameter_struct *pre_node, mdma_link_node_parameter_struct *unused_node);
功能描述	MDMA断开链路列表节点
先决条件	-
被调用函数	-
输入参数{in}	
pre_node	前一个节点的结构体，结构体成员可参考 表3-721. 结构体 mdma_link_node_parameter_struct 。
输入参数{in}	
new_node	新节点的结构体，结构体成员可参考 表3-721. 结构体 mdma_link_node_parameter_struct 。
输出参数{out}	
-	-
返回值	
ErrStatus	SUCCESS或ERROR

例如：

```
/* MDMA disconnect link list node */

mdma_link_node_parameter_struct pre_node;
```



```
mdma_link_node_parameter_struct unused_node;
```

```
mdma_node_delete(&pre_node, &unused_node);
```

函数 mdma_destination_address_config

函数mdma_destination_address_config描述见下表：

表 3-735. 函数 mdma_destination_address_config

函数名称	mdma_destination_address_config
函数原型	void mdma_destination_address_config(mdma_channel_enum channelx, uint32_t address);
功能描述	配置MDMA目标基地址
先决条件	-
被调用函数	-
输入参数{in}	
channelx	MDMA通道
MDMA_CHx(x=0..15)	MDMA通道选择，参考 表3-723. 枚举mdma_channel_enum 。
输入参数{in}	
address	目标基地址，0x00000000-0xFFFFFFFF
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* configure MDMA destination base address */
```

```
mdma_destination_address_config(MDMA_CH0, 0x08000000);
```

函数 mdma_source_address_config

函数mdma_source_address_config描述见下表：

表 3-736. 函数 mdma_source_address_config

函数名称	mdma_source_address_config
函数原型	void mdma_source_address_config(mdma_channel_enum channelx, uint32_t address);
功能描述	配置MDMA源基地址
先决条件	-
被调用函数	-
输入参数{in}	
channelx	MDMA通道
MDMA_CHx(x=0..15)	MDMA通道选择，参考 表3-723. 枚举mdma_channel_enum 。
输入参数{in}	

address	源基地址, 0x00000000-0xFFFFFFFF
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* configure MDMA source base address */
mdma_source_address_config(MDMA_CH0, 0x20000000);
```

函数 mdma_destination_bus_config

函数mdma_destination_bus_config描述见下表:

表 3-737. 函数 mdma_destination_bus_config

函数名称	mdma_destination_bus_config
函数原型	void mdma_destination_bus_config(mdma_channel_enum channelx, uint32_t bus);
功能描述	配置MDMA目标总线
先决条件	-
被调用函数	-
输入参数{in}	
channelx	MDMA通道
MDMA_CHx(x=0..15)	MDMA通道选择, 参考 表3-723. 枚举mdma_channel_enum 。
输入参数{in}	
bus	目标总线
MDMA_DESTINATION_AXI	目标总线是系统总线或AXI总线
MDMA_DESTINATION_AHB_TCM	目标总线是AHB总线或TCM
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* configure MDMA destination bus */
mdma_destination_bus_config(MDMA_CH0, MDMA_DESTINATION_AHB_TCM);
```

函数 mdma_source_bus_config

函数mdma_source_bus_config描述见下表:

表 3-738. 函数 mdma_source_bus_config

函数名称	mdma_source_bus_config
函数原型	void mdma_source_bus_config(mdma_channel_enum channelx, uint32_t bus);
功能描述	配置MDMA源总线
先决条件	-
被调用函数	-
输入参数{in}	
channelx	MDMA通道
MDMA_CHx(x=0..15)	MDMA通道选择, 参考 表3-723. 枚举mdma_channel_enum 。
输入参数{in}	
bus	源总线
MDMA_SOURCE_AXI	源总线是系统总线或AXI总线
MDMA_SOURCE_AHB_TCM	源总线是AHB总线或TCM
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* configure MDMA source bus */
```

```
mdma_source_bus_config(MDMA_CH0, MDMA_SOURCE_AHB_TCM);
```

函数 mdma_priority_config

函数mdma_priority_config描述见下表:

表 3-739. 函数 mdma_priority_config

函数名称	mdma_priority_config
函数原型	void mdma_priority_config(mdma_channel_enum channelx, uint32_t priority);
功能描述	配置MDMA通道x优先级
先决条件	-
被调用函数	-
输入参数{in}	
channelx	MDMA通道
MDMA_CHx(x=0..15)	MDMA通道选择, 参考 表3-723. 枚举mdma_channel_enum 。
输入参数{in}	
priority	该通道的优先级
MDMA_PRIORITY_LOW	低优先级
MDMA_PRIORITY_MEDIUM	中优先级

<i>DIUM</i>	
<i>MDMA_PRIORITY_HIGH</i>	高优先级
<i>MDMA_PRIORITY_ULTRA_HIGH</i>	极高优先级
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* configure priority level of MDMA channel */
```

```
mdma_priority_config(MDMA_CH0, MDMA_PRIORITY_ULTRA_HIGH);
```

函数 mdma_endianness_config

函数mdma_endianness_config描述见下表:

表 3-740. 函数 mdma_endianness_config

函数名称	mdma_endianness_config
函数原型	void mdma_endianness_config(mdma_channel_enum channelx, uint32_t endianness);
功能描述	配置MDMA通道x数据大小端
先决条件	-
被调用函数	-
输入参数{in}	
channelx	MDMA通道
<i>MDMA_CHx(x=0..15)</i>	MDMA通道选择, 参考 表3-723. 枚举mdma_channel_enum 。
输入参数{in}	
endianness	MDMA字节顺序
<i>MDMA_LITTLE_ENDIANNESS</i>	小端
<i>MDMA_BYTE_ENDIANNESS_EXCHANGE</i>	交换半字中字节顺序
<i>MDMA_HALFWORD_ENDIANNESS_EXCHANGE</i>	交换字中半字顺序
<i>MDMA_WORD_ENDIANNESS_EXCHANGE</i>	交换双字中字顺序
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* configure endianness of MDMA channel */
```

```
mdma_endianness_config(MDMA_CH0, MDMA_BYTE_ENDIANNESS_EXCHANGE);
```

函数 mdma_alignment_config

函数mdma_alignment_config描述见下表：

表 3-741. 函数 mdma_alignment_config

函数名称	mdma_alignment_config
函数原型	void mdma_endianness_config(mdma_channel_enum channelx, uint32_t endianness);
功能描述	配置MDMA通道x数据对齐方式
先决条件	-
被调用函数	-
输入参数{in}	
channelx	MDMA通道
MDMA_CHx(x=0..15)	MDMA通道选择，参考 表3-723. 枚举mdma_channel_enum 。
输入参数{in}	
alignment	MDMA数对齐方式
MDMA_DATAALIGN_P KEN	打包/解包源数据以匹配目标数据大小
MDMA_DATAALIGN_R IGHT	右对齐，填充0（默认）
MDMA_DATAALIGN_R IGHT_SIGNED	右对齐与符号扩展，注意，仅当源数据大小小于目标数据大小时，才允许使用模式
MDMA_DATAALIGN_L EFT	左对齐，当源数据大小小于目标数据大小时，在低字节位置用0填充，当源数据大小大于目标数据大小时，只写入源的高字节
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* configure data alignment of MDMA channel */
```

```
mdma_alignment_config(MDMA_CH0, MDMA_DATAALIGN_RIGHT);
```

函数 mdma_source_burst_beats_config

函数mdma_source_burst_beats_config描述见下表：

表 3-742. 函数 mdma_source_burst_beats_config

函数名称	mdma_source_burst_beats_config
------	--------------------------------

函数原型	void mdma_source_burst_beats_config(mdma_channel_enum channelx, uint32_t mbeat);
功能描述	配置源突发传输类型
先决条件	-
被调用函数	-
输入参数{in}	
channelx	MDMA通道
MDMA_CHx(x=0..15)	MDMA通道选择, 参考 表3-723. 枚举mdma_channel_enum 。
输入参数{in}	
sbeat	源传输突发类型
MDMA_SOURCE_BURST_SINGLE	源单一传输
MDMA_SOURCE_BURST_2BEATS	源2拍增量突发传输
MDMA_SOURCE_BURST_4BEATS	源4拍增量突发传输
MDMA_SOURCE_BURST_8BEATS	源8拍增量突发传输
MDMA_SOURCE_BURST_16BEATS	源16拍增量突发传输
MDMA_SOURCE_BURST_32BEATS	源32拍增量突发传输
MDMA_SOURCE_BURST_64BEATS	源64拍增量突发传输
MDMA_SOURCE_BURST_128BEATS	源128拍增量突发传输
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* configure transfer burst beats of source */
```

```
mdma_source_burst_beats_config(MDMA_CH0, MDMA_SOURCE_BURST_4BEATS);
```

函数 mdma_destination_burst_beats_config

函数mdma_destination_burst_beats_config描述见下表:

表 3-743. 函数 mdma_destination_burst_beats_config

函数名称	mdma_destination_burst_beats_config
函数原型	void mdma_destination_burst_beats_config(mdma_channel_enum channelx, uint32_t pbeat);

功能描述	配置目标突发传输类型
先决条件	-
被调用函数	-
输入参数{in}	
channelx	MDMA通道
MDMA_CHx(x=0..15)	MDMA通道选择, 参考 表3-723. 枚举mdma_channel_enum 。
输入参数{in}	
dbeat	目标传输突发类型
MDMA_DESTINATION_BURST_SINGLE	目标单一传输
MDMA_DESTINATION_BURST_2BEATS	目标2拍增量突发传输
MDMA_DESTINATION_BURST_4BEATS	目标4拍增量突发传输
MDMA_DESTINATION_BURST_8BEATS	目标8拍增量突发传输
MDMA_DESTINATION_BURST_16BEATS	目标16拍增量突发传输
MDMA_DESTINATION_BURST_32BEATS	目标32拍增量突发传输
MDMA_DESTINATION_BURST_64BEATS	目标64拍增量突发传输
MDMA_DESTINATION_BURST_128BEATS	目标128拍增量突发传输
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* configure transfer burst beats of destination */
```

```
mdma_destination_burst_beats_config(MDMA_CH0,  
MDMA_DESTINATION_BURST_4BEATS);
```

函数 mdma_source_width_config

函数mdma_source_width_config描述见下表:

表 3-744. 函数 mdma_source_width_config

函数名称	mdma_source_width_config
函数原型	void mdma_source_width_config(mdma_channel_enum channelx, uint32_t swidth);
功能描述	配置源数据长度

先决条件	-
被调用函数	-
输入参数{in}	
channelx	MDMA通道
MDMA_CHx(x=0..15)	MDMA通道选择, 参考 表3-723. 枚举mdma_channel_enum 。
输入参数{in}	
swidth	源地址数据长度
MDMA_SOURCE_DAT ASIZE_8BIT	源地址数据长度为8位
MDMA_SOURCE_DAT ASIZE_16BIT	源地址数据长度为16位
MDMA_SOURCE_DAT ASIZE_32BIT	源地址数据长度为32位
MDMA_SOURCE_DAT ASIZE_64BIT	源地址数据长度为64位
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* configure data size of source */
```

```
mdma_source_width_config (MDMA_CH0, MDMA_SOURCE_DATASIZE_8BIT);
```

函数 mdma_destination_width_config

函数mdma_destination_width_config描述见下表:

表 3-745. 函数 mdma_destination_width_config

函数名称	mdma_destination_width_config
函数原型	void mdma_destination_width_config(mdma_channel_enum channelx, uint32_t dwidth);
功能描述	配置目的数据长度
先决条件	-
被调用函数	-
输入参数{in}	
channelx	MDMA通道
MDMA_CHx(x=0..15)	MDMA通道选择, 参考 表3-723. 枚举mdma_channel_enum 。
输入参数{in}	
dwidth	目的地址数据长度
MDMA_DESTINATION _DATASIZE_8BIT	目的地址数据长度为8位
MDMA_DESTINATION	目的地址数据长度为16位

<code>_DATASIZE_16BIT</code>	
<code>MDMA_DESTINATION</code> <code>_DATASIZE_32BIT</code>	目的地址数据长度为32位
<code>MDMA_DESTINATION</code> <code>_DATASIZE_64BIT</code>	目的地址数据长度为64位
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* configure data size of destination */
```

```
mdma_destination_width_config (MDMA_CH0, MDMA_DESTINATION_DATASIZE_16BIT);
```

函数 mdma_source_increment_config

函数mdma_source_increment_config描述见下表：

表 3-746. 函数 mdma_source_increment_config

函数名称	mdma_source_increment_config
函数原型	void mdma_source_increment_config(mdma_channel_enum channelx, uint32_t sinc);
功能描述	配置源地址增长模式
先决条件	-
被调用函数	-
输入参数{in}	
channelx	MDMA通道
MDMA_CHx(x=0..15)	MDMA通道选择，参考 表3-723. 枚举mdma_channel_enum 。
输入参数{in}	
sinc	源地址增长模式
MDMA_SOURCE_INC REASE_DISABLE	无增长
MDMA_SOURCE_INC REASE_8BIT	源地址按8位增长
MDMA_SOURCE_INC REASE_16BIT	源地址按16位增长
MDMA_SOURCE_INC REASE_32BIT	源地址按32位增长
MDMA_SOURCE_INC REASE_64BIT	源地址按64位增长
MDMA_SOURCE_DEC REASE_8BIT	源地址按8位减少
MDMA_SOURCE_DEC	源地址按16位减少

REASE_16BIT	
MDMA_SOURCE_DEC REASE_32BIT	源地址按32位减少
MDMA_SOURCE_DEC REASE_64BIT	源地址按64位减少
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* configure source adress increment mode */
```

```
mdma_source_increment_config (MDMA_CH0, MDMA_SOURCE_INCREASE_16BIT);
```

函数 mdma_destination_increment_config

函数mdma_destination_increment_config描述见下表：

表 3-747. 函数 mdma_destination_increment_config

函数名称	mdma_destination_increment_config
函数原型	void mdma_destination_increment_config(mdma_channel_enum channelx, uint32_t dinc);
功能描述	配置目的地址增长模式
先决条件	-
被调用函数	-
输入参数{in}	
channelx	MDMA通道
MDMA_CHx(x=0..15)	MDMA通道选择，参考 表3-723. 枚举mdma_channel_enum 。
输入参数{in}	
dinc	目的地址增长模式
MDMA_DESTINATION_INCREASE_DISABLE	无增长
MDMA_DESTINATION_INCREASE_8BIT	目的地址按8位增长
MDMA_DESTINATION_INCREASE_16BIT	目的地址按16位增长
MDMA_DESTINATION_INCREASE_32BIT	目的地址按32位增长
MDMA_DESTINATION_INCREASE_64BIT	目的地址按64位增长
MDMA_DESTINATION_INCREASE_64BIT	目的地址按8位减少
MDMA_DESTINATION	目的地址按16位减少

<code>_DECREASE_16BIT</code>	
<code>MDMA_DESTINATION_DECREASE_32BIT</code>	目的地址按32位减少
<code>MDMA_DESTINATION_DECREASE_64BIT</code>	目的地址按64位减少
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* configure source adress increment mode */
```

```
mdma_destination_increment_config (MDMA_CH0, MDMA_DESTINATION_INCREASE_16BIT);
```

函数 mdma_channel_bufferable_write_enable

函数mdma_destination_burst_beats_config描述见下表：

表 3-748. 函数 mdma_channel_bufferable_write_enable

函数名称	mdma_channel_bufferable_write_enable
函数原型	void mdma_channel_bufferable_write_enable(mdma_channel_enum channelx);
功能描述	使能MDMA通道x可缓冲写模式
先决条件	-
被调用函数	-
输入参数{in}	
channelx	MDMA通道
MDMA_CHx(x=0..15)	MDMA通道选择，参考 表3-723. 枚举mdma_channel_enum 。
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* enable MDMA channel bufferable write mode */
```

```
mdma_channel_bufferable_write_enable(MDMA_CH0);
```

函数 mdma_channel_bufferable_write_disable

函数mdma_destination_burst_beats_config描述见下表：

表 3-749. 函数 mdma_channel_bufferable_write_disable

函数名称	mdma_channel_bufferable_write_disable
函数原型	void mdma_channel_bufferable_write_disable(mdma_channel_enum channelx);
功能描述	禁能MDMA通道x可缓冲写模式
先决条件	-
被调用函数	-
输入参数{in}	
channelx	MDMA通道
MDMA_CHx(x=0..15)	MDMA通道选择, 参考 表3-723. 枚举mdma_channel_enum 。
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* disable MDMA channel bufferable write mode */
mdma_channel_bufferable_write_disable(MDMA_CH0);
```

函数 mdma_channel_software_request_enable

函数mdma_channel_software_request_enable描述见下表:

表 3-750. 函数 mdma_channel_software_request_enable

函数名称	mdma_channel_software_request_enable
函数原型	void mdma_channel_software_request_enable(mdma_channel_enum channelx);
功能描述	使能MDMA通道x软件请求
先决条件	-
被调用函数	-
输入参数{in}	
channelx	MDMA通道
MDMA_CHx(x=0..15)	MDMA通道选择, 参考 表3-723. 枚举mdma_channel_enum 。
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* enable MDMA channel software request */
mdma_channel_software_request_enable(MDMA_CH0);
```

函数 mdma_channel_enable

函数mdma_channel_enable描述见下表：

表 3-751. 函数 mdma_channel_enable

函数名称	mdma_channel_enable
函数原型	void mdma_channel_enable(mdma_channel_enum channelx);
功能描述	使能MDMA通道x
先决条件	-
被调用函数	-
输入参数{in}	
channelx	MDMA通道
MDMA_CHx(x=0..15)	MDMA通道选择，参考 表3-723. 枚举mdma_channel_enum 。
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* enable MDMA channel */  
  
mdma_channel_enable(MDMA_CH0);
```

函数 mdma_channel_disable

函数mdma_channel_disable描述见下表：

表 3-752. 函数 mdma_channel_disable

函数名称	mdma_channel_disable
函数原型	void mdma_channel_disable(mdma_channel_enum channelx);
功能描述	禁能MDMA通道x
先决条件	-
被调用函数	-
输入参数{in}	
channelx	MDMA通道
MDMA_CHx(x=0..15)	MDMA通道选择，参考 表3-723. 枚举mdma_channel_enum 。
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* disable MDMA channel */  
  
mdma_channel_disable(MDMA_CH0);
```

函数 mdma_transfer_error_direction_get

函数mdma_transfer_error_direction_get描述见下表:

表 3-753. 函数 mdma_transfer_error_direction_get

Function name	mdma_transfer_error_direction_get
Function prototype	uint32_t mdma_transfer_error_direction_get(mdma_channel_enum channelx);
Function descriptions	获取MDMA传输错误方向
Precondition	-
The called functions	-
输入参数{in}	
channelx	MDMA通道
MDMA_CHx(x=0..15)	MDMA通道选择, 参考 表3-723. 枚举mdma_channel_enum 。
输出参数{out}	
-	-
返回值	
uint32_t	传输错误方向
MDMA_READ_ERROR	读访问错误
MDMA_WRITE_ERROR	写访问错误

例如:

```
/* get MDMA transfer error direction */
```

```
uint32_t dir;
```

```
dir = mdma_transfer_error_direction_get(MDMA_CH0);
```

函数 mdma_transfer_error_address_get

函数mdma_transfer_error_address_get描述见下表:

表 3-754. 函数 mdma_transfer_error_address_get

函数名称	mdma_transfer_error_address_get
函数原型	uint32_t mdma_transfer_error_address_get(mdma_channel_enum channelx);
功能描述	获取MDMA传输错误地址
先决条件	-
被调用函数	-
输入参数{in}	
channelx	MDMA通道
MDMA_CHx(x=0..15)	MDMA通道选择, 参考 表3-723. 枚举mdma_channel_enum 。
输出参数{out}	
-	-

返回值	
uint32_t	传输错误地址的低7位, 0x00000000-0x0000007F

例如:

```
/* get MDMA transfer error address */
```

```
uint32_t err_addr;
```

```
err_addr = mdma_transfer_error_address_get(MDMA_CH0);
```

函数 mdma_flag_get

函数mdma_flag_get描述见下表:

表 3-755. 函数 mdma_flag_get

函数名称	mdma_flag_get
函数原型	FlagStatus mdma_flag_get(mdma_channel_enum channelx, uint32_t flag);
功能描述	获取MDMA标志
先决条件	-
被调用函数	-
输入参数{in}	
channelx	MDMA通道
MDMA_CHx(x=0..15)	MDMA通道选择, 参考 表3-723. 枚举mdma_channel_enum 。
输入参数{in}	
flag	MDMA标志
MDMA_FLAG_ERR	传输错误标志
MDMA_FLAG_CHTCF	通道传输完成标志
MDMA_FLAG_MBTCTF	多块传输完成标志
MDMA_FLAG_BTCTF	块传输完成标志
MDMA_FLAG_TCTF	缓冲区传输完成标志
MDMA_FLAG_REQAF	请求激活标志
MDMA_FLAG_LDTER R	通道最后一次传输中的链路数据传输错误标志
MDMA_FLAG_MDTER R	掩码数据错误标志
MDMA_FLAG_ASERR	地址和大小错误标志
MDMA_FLAG_BZERR	块大小错误标志
输出参数{out}	
-	-
返回值	
FlagStatus	SET或RESET

例如:

```
/* get MDMA flag */
```

FlagStatus status;

```
status = mdma_flag_get(MDMA_CH0, MDMA_FLAG_TCF);
```

函数 mdma_flag_clear

函数mdma_flag_clear描述见下表：

表 3-756. 函数 mdma_flag_clear

函数名称	mdma_flag_clear
函数原型	void mdma_flag_clear(mdma_channel_enum channelx, uint32_t flag);
功能描述	清除MDMA标志
先决条件	-
被调用函数	-
输入参数{in}	
channelx	MDMA通道
MDMA_CHx(x=0..15)	MDMA通道选择，参考 表3-723. 枚举mdma_channel_enum 。
输入参数{in}	
flag	MDMA标志
MDMA_FLAG_ERR	传输错误标志
MDMA_FLAG_CHTCF	通道传输完成标志
MDMA_FLAG_MBTCF	多块传输完成标志
MDMA_FLAG_BTCF	块传输完成标志
MDMA_FLAG_TCF	缓冲区传输完成标志
MDMA_FLAG_LDTER R	通道最后一次传输中的链路数据传输错误标志
MDMA_FLAG_MDTER R	掩码数据错误标志
MDMA_FLAG_ASERR	地址和大小错误标志
MDMA_FLAG_BZERR	块大小错误标志
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* clear MDMA flag */
```

```
mdma_flag_clear(MDMA_CH0, MDMA_FLAG_TCF);
```

函数 mdma_interrupt_enable

函数mdma_interrupt_enable描述见下表：

表 3-757. 函数 mdma_interrupt_enable

函数名称	mdma_interrupt_enable
函数原型	void mdma_interrupt_enable(mdma_channel_enum channelx, uint32_t interrupt);
功能描述	使能MDMA中断
先决条件	-
被调用函数	-
输入参数{in}	
channelx	MDMA通道
MDMA_CHx(x=0..15)	MDMA通道选择, 参考 表3-723. 枚举mdma_channel_enum 。
输出参数{out}	
interrupt	MDMA中断
MDMA_INT_ERR	传输错误中断
MDMA_INT_ERR	通道传输完成中断
MDMA_INT_MBTC	多块传输完成中断
MDMA_INT_BTC	块传输完成中断
MDMA_INT_TC	缓冲区传输完成中断
返回值	
-	-

例如:

```
/* enable MDMA interrupt */
mdma_interrupt_enable(MDMA_CH0, MDMA_INT_TC);
```

函数 mdma_interrupt_disable

函数mdma_interrupt_disable描述见下表:

表 3-758. 函数 mdma_interrupt_disable

函数名称	mdma_interrupt_disable
函数原型	void mdma_interrupt_disable(mdma_channel_enum channelx, uint32_t interrupt);
功能描述	禁能MDMA中断
先决条件	-
被调用函数	-
输入参数{in}	
channelx	MDMA通道
MDMA_CHx(x=0..15)	MDMA通道选择, 参考 表3-723. 枚举mdma_channel_enum 。
输入参数{in}	
interrupt	MDMA中断
MDMA_INT_ERR	传输错误中断
MDMA_INT_ERR	通道传输完成中断
MDMA_INT_MBTC	多块传输完成中断

<i>MDMA_INT_BT</i>	块传输完成中断
<i>MDMA_INT_TC</i>	缓冲区传输完成中断
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* disable MDMA interrupt */
```

```
mdma_interrupt_disable(MDMA_CH0, MDMA_INT_TC);
```

函数 mdma_interrupt_flag_get

函数mdma_interrupt_flag_get描述见下表：

表 3-759. 函数 mdma_interrupt_flag_get

函数名称	mdma_interrupt_flag_get
函数原型	FlagStatus mdma_interrupt_flag_get(mdma_channel_enum channelx, uint32_t int_flag);
功能描述	获取MDMA中断标志
先决条件	-
被调用函数	-
输入参数{in}	
channelx	MDMA通道
<i>MDMA_CHx</i> (x=0..15)	MDMA通道选择，参考 表3-723. 枚举mdma_channel_enum 。
输入参数{in}	
int_flag	MDMA中断标志
<i>MDMA_INT_FLAG_ER</i> <i>R</i>	传输错误中断标志
<i>MDMA_INT_FLAG_CH</i> <i>TCF</i>	通道传输完成中断标志
<i>MDMA_INT_FLAG_MB</i> <i>TCF</i>	多块传输完成中断标志
<i>MDMA_INT_FLAG_BT</i> <i>CF</i>	块传输完成中断标志
<i>MDMA_INT_FLAG_TC</i> <i>F</i>	缓冲区传输完成中断标志
输出参数{out}	
-	-
返回值	
FlagStatus	SET或RESET

例如：

```
/* get MDMA interrupt flag */
```

```
FlagStatus status;
```

```
status = mdma_interrupt_flag_get(MDMA_CH0, MDMA_INT_FLAG_TCF);
```

函数 mdma_interrupt_flag_clear

函数mdma_interrupt_flag_get描述见下表：

表 3-760. 函数 mdma_interrupt_flag_clear

函数名称	mdma_interrupt_flag_clear
函数原型	void mdma_interrupt_flag_clear(mdma_channel_enum channelx, uint32_t int_flag);
功能描述	清除MDMA中断标志
先决条件	-
被调用函数	-
输入参数{in}	
channelx	MDMA通道
MDMA_CHx(x=0..15)	MDMA通道选择，参考 表3-723. 枚举mdma_channel_enum 。
输入参数{in}	
int_flag	MDMA中断标志
MDMA_INT_FLAG_ER R	传输错误中断标志
MDMA_INT_FLAG_CH TCF	通道传输完成中断标志
MDMA_INT_FLAG_MB TCF	多块传输完成中断标志
MDMA_INT_FLAG_BT CF	块传输完成中断标志
MDMA_INT_FLAG_TC F	缓冲区传输完成中断标志
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* clear MDMA interrupt flag */
```

```
mdma_interrupt_flag_clear(MDMA_CH0, MDMA_INT_FLAG_TCF);
```

3.23. OSPI

OSPI是一种专用于和外部存储器通信的接口，支持单线，双线，四线和八线SPI存储器（PSRAMS, NAND, NOR flash等）。章节[3.23.1](#)描述了OSPI的寄存器列表，章节[3.23.2](#)对OSPI库函数进行说明。

3.23.1. 外设寄存器描述

OSPI寄存器列表如下表所示：

表 3-761. OSPI 寄存器

寄存器名称	寄存器描述
OSPI_CTL	OSPI控制寄存器
OSPI_DCFG0	OSPI设备配置寄存器0
OSPI_DCFG1	OSPI设备配置寄存器1
OSPI_STAT	OSPI状态寄存器
OSPI_STATC	OSPI状态清除寄存器
OSPI_DTLEN	OSPI数据长度寄存器
OSPI_ADDR	OSPI地址寄存器
OSPI_DATA	OSPI数据寄存器
OSPI_STATMK	OSPI状态屏蔽寄存器
OSPI_STATMATCH	OSPI状态匹配寄存器
OSPI_INTERVAL	OSPI间隔寄存器
OSPI_TCFG	OSPI传输配置寄存器
OSPI_TIMCFG	OSPI时需配置寄存器
OSPI_INS	OSPI指令寄存器
OSPI_ALTE	OSPI交替字节寄存器
OSPI_WPTCFG	OSPI回卷传输配置寄存器
OSPI_WPTIMCFG	OSPI回卷时序配置寄存器
OSPI_WPINS	OSPI回卷指令寄存器
OSPI_WPALTE	OSPI回卷交替字节寄存器
OSPI_WTCFG	OSPI写入传输配置寄存器
OSPI_WTIMCFG	OSPI写入时序配置寄存器
OSPI_WINS	OSPI写入指令寄存器
OSPI_WALTE	OSPI写入交替字节寄存器

3.23.2. 外设库函数说明

OSPI库函数列表如下表所示：

表 3-762. OSPI 库函数

库函数名称	库函数描述
ospi_deinit	复位OSPI

库函数名称	库函数描述
ospi_struct_init	初始化OSPI结构体参数为默认值
ospi_init	初始化OSPI参数
ospi_enable	使能OSPI
ospi_disable	失能OSPI
ospi_device_memory_type_config	配置设备内存类型
ospi_device_memory_size_config	配置设备内存大小
ospi_functional_mode_config	选择功能模式
ospi_status_polling_config	配置状态轮询模式
ospi_status_mask_config	配置状态屏蔽
ospi_status_match_config	配置状态匹配
ospi_interval_cycle_config	配置间隔周期
ospi_fifo_level_config	配置FIFO阈值
ospi_chip_select_high_cycle_config	配置片选高电平周期
ospi_prescaler_config	配置OSPI预分频
ospi_dummy_cycles_config	配置空指令周期
ospi_delay_hold_cycle_config	配置延迟保持1/4周期
ospi_sample_shift_config	配置采样移位
ospi_data_length_config	配置数据长度
ospi_instruction_config	配置指令模式
ospi_address_config	配置地址模式
ospi_alternate_bytes_config	配置交替字节模式
ospi_data_config	配置数据模式
ospi_data_transmit	OSPI发送数据
ospi_data_receive	OSPI接收数据
ospi_dma_enable	使能OSPI DMA
ospi_dma_disable	失能OSPI DMA
ospi_wrap_size_config	配置回卷大小
ospi_wrap_instruction_config	配置回卷指令模式
ospi_wrap_address_config	配置回卷地址模式
ospi_wrap_alternate_bytes_config	配置回卷交替字节模式
ospi_wrap_data_config	配置回卷数据模式
ospi_wrap_dummy_cycles_config	配置回卷空指令周期
ospi_wrap_delay_hold_cycle_config	配置回卷延迟保持1/4周期
ospi_wrap_sample_shift_config	配置回卷采样移位
ospi_write_instruction_config	配置写入指令模式
ospi_write_address_config	配置写入地址模式
ospi_write_alternate_bytes_config	配置写入交替字节模式
ospi_write_data_config	配置写入数据模式
ospi_write_dummy_cycles_config	配置写入空指令周期
ospi_command_config	配置OSPI常规命令参数
ospi_transmit	发送数据

库函数名称	库函数描述
ospi_receive	接收数据
ospi_autopolling_mode	配置OSPI状态轮询模式
ospi_interrupt_enable	使能OSPI中断
ospi_interrupt_disable	失能OSPI中断
ospi_fifo_level_get	获取OSPI FIFO阈值
ospi_flag_get	获取OSPI状态标志
ospi_flag_clear	清除OSPI状态标志
ospi_interrupt_flag_get	获取OSPI中断状态标志
ospi_interrupt_flag_clear	清除OSPI中断状态标志

结构体 `ospi_parameter_struct`

表 3-763. 结构体 `ospi_parameter_struct`

成员名称	功能描述
prescaler	从内核时钟分频产生OSPI时钟的分频因子 (0-255)
fifo_threshold	FIFO中的阈值字节数 (0-31)
sample_shift	指定采样移位 (OSPI_SAMPLE_SHIFTING_NONE, OSPI_SAMPLE_SHIFTING_HALF_CYCLE)
device_size	指定设备大小 (OSPI_MESZ_x_BYTES(x = 2, 4, 8, ..., 512, 1024), OSPI_MESZ_x_KBS(x = 2, 4, 8, ..., 512, 1024), OSPI_MESZ_x_MBS(x = 2, 4, 8, ..., 2048, 4096))
cs_hightime	片选高电平时间 (OSPI_CS_HIGH_TIME_x_CYCLE (x = 1, 2, ..., 63, 64))
memory_type	外部设备类型 (OSPI_MICRON_MODE, OSPI_MACRONIX_MODE, OSPI_STANDARD_MODE, OSPI_MACRONIX_RAM_MODE)
wrap_size	指定外部设备回卷大小 (OSPI_DIRECT, OSPI_WRAP_16BYTES, OSPI_WRAP_32BYTES, OSPI_WRAP_64BYTES, OSPI_WRAP_128BYTES)
delay_hold_cycle	延迟保持1/4周期 (OSPI_DELAY_HOLD_NONE, OSPI_DELAY_HOLD_QUARTER_CYCLE)

结构体 `ospi_regular_cmd_struct`

表 3-764. 结构体 `ospi_regular_cmd_struct`

成员名称	功能描述
operation_type	指定配置是应用于常规寄存器还是写入操作的寄存器 (OSPI_OPTYPE_COMMON_CFG, OSPI_OPTYPE_READ_CFG, OSPI_OPTYPE_WRITE_CFG, OSPI_OPTYPE_WRAP_CFG)

instruction	要发送的指令 (0-0xFFFFFFFF)
ins_mode	指定指令模式 (OSPI_INSTRUCTION_NONE, OSPI_INSTRUCTION_1_LINE, OSPI_INSTRUCTION_2_LINES, OSPI_INSTRUCTION_4_LINES, OSPI_INSTRUCTION_8_LINES)
ins_size	指定指令大小 (OSPI_INSTRUCTION_8_BITS, OSPI_INSTRUCTION_16_BITS, OSPI_INSTRUCTION_24_BITS, OSPI_INSTRUCTION_32_BITS)
address	要发送的地址 (0-0xFFFFFFFF)
addr_mode	地址模式 (OSPI_ADDRESS_NONE, OSPI_ADDRESS_1_LINE, OSPI_ADDRESS_2_LINES, OSPI_ADDRESS_4_LINES, OSPI_ADDRESS_8_LINES)
addr_size	地址大小 (OSPI_ADDRESS_8_BITS, OSPI_ADDRESS_16_BITS, OSPI_ADDRESS_24_BITS, OSPI_ADDRESS_32_BITS)
addr_dtr_mode	地址阶段是否使能DTR模式 (OSPI_ADDRDTR_MODE_DISABLE, OSPI_ADDTR_MODE_ENABLE)
alter_bytes	要发送的交替字节 (0-0xFFFFFFFF)
alter_bytes_mode	交替字节模式 (OSPI_ALTERNATE_BYTES_NONE, OSPI_ALTERNATE_BYTES_1_LINE, OSPI_ALTERNATE_BYTES_2_LINES, OSPI_ALTERNATE_BYTES_4_LINES, OSPI_ALTERNATE_BYTES_8_LINES)
alter_bytes_size	交替字节大小 (OSPI_ALTERNATE_BYTES_8_BITS, OSPI_ALTERNATE_BYTES_16_BITS, OSPI_ALTERNATE_BYTES_24_BITS, OSPI_ALTERNATE_BYTES_32_BITS)
alter_bytes_dtr_mode	交替字节阶段是否使能DTR模式 (OSPI_ABDTR_MODE_DISABLE, OSPI_ABDTR_MODE_ENABLE)
data_mode	数据模式 (OSPI_DATA_NONE, OSPI_DATA_1_LINE, OSPI_DATA_2_LINES, OSPI_DATA_4_LINES, OSPI_DATA_8_LINES)
nbddata	传输的数据量 (1-0xFFFFFFFF)
data_dtr_mode	数据阶段是否使能DTR模式 (OSPI_DADTR_MODE_DISABLE, OSPI_DADTR_MODE_ENABLE)
dummy_cycles	数据阶段之前插入的空指令周期数 (OSPI_DUMYC_CYCLES_x (x = 0, 1, 2, ..., 30, 31))

结构体 `ospi_autopolling_struct`表 3-765. 结构体 `ospi_autopolling_struct`

成员名称	功能描述
match	与屏蔽状态寄存器进行比较以获得匹配的值 (0- 0xFFFFFFFF)
mask	用来屏蔽接收的状态字节 (0- 0xFFFFFFFF)
interval	状态轮询模式下两次读命令之间的SCK周期数 (0- 0xFFFF)
match_mode	匹配模式 (OSPI_MATCH_MODE_AND, OSPI_MATCH_MODE_OR)
automatic_stop	表明在产生匹配后停止状态轮询模式 (OSPI_AUTOMATIC_STOP_ABORT, OSPI_AUTOMATIC_STOP_MATCH)

枚举 `ospi_interrupt_flag_enum`表 3-766. 枚举 `ospi_interrupt_flag_enum`

成员名称	功能描述
OSPI_INT_FLAG_TERR	发送错误中断标志
OSPI_INT_FLAG_TC	发送完成中断标志
OSPI_INT_FLAG_FT	Fifo阈值中断标志
OSPI_INT_FLAG_SM	状态匹配中断标志

函数 `ospi_deinit`

函数`ospi_deinit`描述见下表：

表 3-767. 函数 `ospi_deinit`

函数名称	<code>ospi_deinit</code>
函数原形	<code>void ospi_deinit(uint32_t ospi_periph);</code>
功能描述	复位OSPI
先决条件	-
被调用函数	<code>rcu_periph_reset_enable / rcu_periph_reset_disable</code>
输入参数{in}	
-	-
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* reset the OSPI peripheral */
```

```
ospi_deinit();
```


函数 ospi_struct_init

函数ospi_struct_init描述见下表:

表 3-768. 函数 ospi_struct_init

函数名称	ospi_struct_init
函数原形	void ospi_struct_init(ospi_parameter_struct *ospi_struct);
功能描述	初始化OSPI结构体参数为默认值
先决条件	-
被调用函数	-
输入参数{in}	
ospi_struct	OSPI参数结构体, 结构体成员参考 表3-763. 结构体ospi_parameter_struct
输出参数{out}	
-	-
返回值	
-	-

例如:

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

```
ospi_parameter_struct ospi_struct;
```

```
ospi_struct_init(&ospi_struct);
```

函数 ospi_init

函数ospi_init描述见下表:

表 3-769. 函数 ospi_init

函数名称	ospi_init
函数原形	void ospi_init(uint32_t ospi_periph, ospi_parameter_struct *ospi_struct);
功能描述	初始化OSPI参数
先决条件	-
被调用函数	-
输入参数{in}	
ospi_periph	OSPIx(x=0,1)
输入参数{in}	
ospi_struct	OSPI参数结构体, 结构体成员参考 表3-763. 结构体ospi_parameter_struct
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* initialize OSPI parameter */
```

```

ospi_parameter_struct ospi_struct;

ospi_struct.prescaler = 8;

ospi_struct.fifo_threshold = OSPI_FIFO_THRESHOLD_10;

ospi_struct.sample_shift = OSPI_SAMPLE_SHIFTING_NONE;

ospi_struct.device_size = OSPI_MESZ_512_MBS;

ospi_struct.cs_hightime = OSPI_CS_HIGH_TIME_10_CYCLE;

ospi_struct.memory_type = OSPI_STANDARD_MODE;

ospi_struct.wrap_size = OSPI_DIRECT;

ospi_struct.delay_hold_cycle = OSPI_DELAY_HOLD_QUARTER_CYCLE;

osp_init(OSPI0, &ospi_struct);

```

函数 **ospi_enable**

函数ospi_enable描述见下表：

表 3-770. 函数 ospi_enable

函数名称	ospi_enable
函数原形	void ospi_enable(uint32_t ospi_periph);
功能描述	使能OSPI
先决条件	-
被调用函数	-
输入参数{in}	
ospi_periph	OSPIx(x=0,1)
输出参数{out}	
-	-
返回值	
-	-

例如：

```

/* enable OSPI0 */

ospi_enable(OSPI0);

```

函数 **ospi_disable**

函数ospi_disable描述见下表：

表 3-771. 函数 ospi_disable

函数名称	ospi_disable
函数原形	void ospi_disable(uint32_t ospi_periph);
功能描述	失能OSPI

先决条件	-
被调用函数	-
输入参数{in}	
ospi_periph	OSPIx(x=0,1)
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* disable OSPI0 */
```

```
ospi_disable(OSPI0);
```

函数 ospi_device_memory_type_config

函数ospi_device_memory_type_config描述见下表：

表 3-772. 函数 ospi_device_memory_type_config

函数名称	ospi_device_memory_type_config
函数原形	void ospi_device_memory_type_config(uint32_t ospi_periph, uint32_t dtysel);
功能描述	配置设备内存类型
先决条件	-
被调用函数	-
输入参数{in}	
ospi_periph	OSPIx(x=0,1)
输入参数{in}	
dtysel	OSPI设备类型选择
OSPI_MICRON_MODE	Micron模式
OSPI_MACRONIX_MODE	Micronix模式
OSPI_STANDARD_MODE	标准模式
OSPI_MACRONIX_RAM_MODE	Micronix RAM模式
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* configure device memory type */
```

```
ospi_device_memory_type_config(OSPI0, OSPI_STANDARD_MODE);
```

函数 `ospi_device_memory_size_config`

函数 `ospi_device_memory_size_config` 描述见下表：

表 3-773. 函数 `ospi_device_memory_size_config`

函数名称	<code>ospi_device_memory_size_config</code>
函数原形	<code>void ospi_device_memory_size_config(uint32_t ospi_periph, uint32_t mesz);</code>
功能描述	配置设备内存大小
先决条件	-
被调用函数	-
输入参数{in}	
<code>ospi_periph</code>	OSPIx (x=0,1)
输入参数{in}	
<code>mesz</code>	设备内存大小
<code>OSPI_MESE_x_BYTES</code>	x = 2, 4, 8, ..., 512, 1024
<code>OSPI_MESE_x_KBS</code>	x = 2, 4, 8, ..., 512, 1024
<code>OSPI_MESE_x_MBS</code>	x = 2, 4, 8, ..., 2048, 4096
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* configure device memory size */
```

```
ospi_device_memory_size_config(OSPI0, OSPI_MESE_1024_MBS);
```

函数 `ospi_functional_mode_config`

函数 `ospi_functional_mode_config` 描述见下表：

表 3-774. 函数 `ospi_functional_mode_config`

函数名称	<code>ospi_functional_mode_config</code>
函数原形	<code>void ospi_functional_mode_config(uint32_t ospi_periph, uint32_t fmod);</code>
功能描述	选择功能模式
先决条件	-
被调用函数	-
输入参数{in}	
<code>ospi_periph</code>	OSPIx (x=0,1)
输入参数{in}	

fmod	OSPI功能模式
<i>OSPI_INDIRECT_WRITE</i>	OSPI间接写入模式
<i>OSPI_INDIRECT_READ</i>	OSPI间接读取模式
<i>OSPI_STATUS_POLLING</i>	OSPI状态轮询模式
<i>OSPI_MEMORY_MAPPED</i>	OSPI内存映射模式
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* select functional mode */
```

```
ospi_functional_mode_config(OSPI0, OSPI_INDIRECT_WRITE);
```

函数 **ospi_status_polling_config**

函数ospi_status_polling_config描述见下表：

表 3-775. 函数 ospi_status_polling_config

函数名称	ospi_status_polling_config
函数原形	void ospi_status_polling_config(uint32_t ospi_periph, uint32_t stop, uint32_t mode);
功能描述	配置状态轮询模式
先决条件	-
被调用函数	-
输入参数{in}	
ospi_periph	OSPIx (x=0,1)
输入参数{in}	
stop	OSPI自动停止
<i>OSPI_AUTOMATIC_STOP_MATCH</i>	状态轮询模式在匹配时停止
输入参数{in}	
mode	OSPI匹配模式
<i>OSPI_MATCH_MODE_AND</i>	状态轮询匹配与模式
<i>OSPI_MATCH_MODE_OR</i>	状态轮询匹配或模式
输出参数{out}	
-	-

返回值	
-	-

例如：

```
/* configure status polling mode */
```

```
ospi_status_polling_config(OSPI0, OSPI_AUTOMATIC_STOP_ABORT,
OSPI_MATCH_MODE_OR);
```

函数 `ospi_status_mask_config`

函数 `ospi_status_mask_config` 描述见下表：

表 3-776. 函数 `ospi_status_mask_config`

函数名称	<code>ospi_status_mask_config</code>
函数原形	<code>void ospi_status_mask_config(uint32_t ospi_periph, uint32_t mask);</code>
功能描述	配置状态屏蔽
先决条件	-
被调用函数	-
输入参数{in}	
<code>ospi_periph</code>	OSPIx (x=0,1)
输入参数{in}	
<code>mask</code>	状态屏蔽 (0-0xFFFFFFFF)
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* configure status mask */
```

```
ospi_status_mask_config (OSPI0, 0x55555555);
```

函数 `ospi_status_match_config`

函数 `ospi_status_match_config` 描述见下表：

表 3-777. 函数 `ospi_status_match_config`

函数名称	<code>ospi_status_match_config</code>
函数原形	<code>void ospi_status_match_config(uint32_t ospi_periph, uint32_t match);</code>
功能描述	配置状态匹配
先决条件	-
被调用函数	-
输入参数{in}	
<code>ospi_periph</code>	OSPIx (x=0,1)

输入参数{in}	
match	状态匹配(0-0xFFFFFFFF)
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* configure status match */
```

```
ospi_status_match_config(OSPI0, 0x55555555);
```

函数 **ospi_interval_cycle_config**

函数ospi_interval_cycle_config描述见下表:

表 3-778. 函数 ospi_interval_cycle_config

函数名称	ospi_interval_cycle_config
函数原形	void ospi_interval_cycle_config(uint32_t ospi_periph, uint16_t interval);
功能描述	配置间隔周期
先决条件	-
被调用函数	-
输入参数{in}	
ospi_periph	OSPIx (x=0,1)
输入参数{in}	
interval	间隔周期 (0-0xFFFF)
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* configure interval cycle */
```

```
ospi_interval_cycle_config(OSPI0, 0x5555);
```

函数 **ospi_fifo_level_config**

函数ospi_fifo_level_config描述见下表:

表 3-779. 函数 ospi_fifo_level_config

函数名称	ospi_fifo_level_config
函数原形	void ospi_fifo_level_config(uint32_t ospi_periph, uint32_t fttl)
功能描述	配置FIFO阈值
先决条件	-

被调用函数	-
输入参数{in}	
ospi_periph	OSPIx (x=0,1)
输入参数{in}	
ftl	FIFO阈值
OSPI_FIFO_THRES HOLD_x	阈值设置为x (x = 1, 2, ..., 31, 32)
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* ospi_fifo_level_config */
```

```
ospi_fifo_level_config(OSPI0, OSPI_FIFO_THRESHOLD_10);
```

函数 ospi_chip_select_high_cycle_config

函数ospi_chip_select_high_cycle_config描述见下表:

表 3-780. 函数 ospi_chip_select_high_cycle_config

函数名称	ospi_chip_select_high_cycle_config
函数原形	void ospi_chip_select_high_cycle_config(uint32_t ospi_periph, uint32_t cshc);
功能描述	配置片选高电平周期
先决条件	-
被调用函数	-
输入参数{in}	
ospi_periph	OSPIx (x=0,1)
输入参数{in}	
cshc	OSPI片选高电平周期
OSPI_CS_HIGH_TIME_x_CYCLE	片选高点周期设置为x (x = 1, 2, ..., 63, 64)
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* configure chip select high cycle */
```

```
ospi_chip_select_high_cycle_config(OSPI0, OSPI_CS_HIGH_TIME_3_CYCLE);
```


函数 ospi_prescaler_config

函数ospi_prescaler_config描述见下表：

表 3-781. 函数 ospi_prescaler_config

函数名称	ospi_prescaler_config
函数原形	void ospi_prescaler_config(uint32_t ospi_periph, uint32_t psc);
功能描述	配置OSPI预分频
先决条件	-
被调用函数	-
输入参数{in}	
ospi_periph	OSPIx (x=0,1)
输入参数{in}	
psc	从内核时钟分频产生OSPI时钟的分频因子 (0-0xFF)
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* configure OSPI prescaler */
```

```
ospi_prescaler_config(OSPI0, 0x08);
```

函数 ospi_dummy_cycles_config

函数ospi_dummy_cycles_config描述见下表：

表 3-782. 函数 ospi_dummy_cycles_config

函数名称	ospi_dummy_cycles_config
函数原形	void ospi_dummy_cycles_config(uint32_t ospi_periph, uint32_t dumyc);
功能描述	配置空指令周期数
先决条件	-
被调用函数	-
输入参数{in}	
ospi_periph	OSPIx (x=0,1)
输入参数{in}	
dumyc	空指令周期数
OSPI_DUMYC_CYCLE_x	空指令周期数设置x (x = 0, 1, 2, ..., 30, 31)
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* configure dummy cycles number */
```

```
ospi_dummy_cycles_config(OSPI0, OSPI_DUMYC_CYCLES_5);
```

函数 `ospi_delay_hold_cycle_config`

函数 `ospi_delay_hold_cycle_config` 描述见下表：

表 3-783. 函数 `ospi_delay_hold_cycle_config`

函数名称	<code>ospi_delay_hold_cycle_config</code>
函数原形	<code>void ospi_delay_hold_cycle_config(uint32_t ospi_periph, uint32_t dehqc);</code>
功能描述	配置延迟保持1/4周期
先决条件	-
被调用函数	-
输入参数{in}	
<code>ospi_periph</code>	OSPIx (x=0,1)
输入参数{in}	
<code>dehqc</code>	OSPI延迟保持1/4周期
<code>OSPI_DELAY_HOLD_NONE</code>	OSPI无延迟保持周期
<code>OSPI_DELAY_HOLD_QUARTER_CYCLE</code>	OSPI延迟保持1/4周期
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* delay hold 1/4 cycle */
```

```
ospi_delay_hold_cycle_config(OSPI0, OSPI_DELAY_HOLD_QUARTER_CYCLE);
```

函数 `ospi_sample_shift_config`

函数 `ospi_sample_shift_config` 描述见下表：

表 3-784. 函数 `ospi_sample_shift_config`

函数名称	<code>ospi_sample_shift_config</code>
函数原形	<code>void ospi_sample_shift_config(uint32_t ospi_periph, uint32_t ssample);</code>
功能描述	配置采样移位
先决条件	-
被调用函数	-
输入参数{in}	

ospi_periph	OSPIx (x=0,1)
输入参数{in}	
ssample	OSPI采样移位
OSPI_SAMPLE_SHIFTING_NONE	OSPI无采样移位
OSPI_SAMPLE_SHIFTING_HALF_CYCLE	OSPI采样移位1/2周期
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* configure sample shift */
```

```
ospi_sample_shift_config(OSPI0, OSPI_SAMPLE_SHIFTING_NONE);
```

函数 **ospi_data_length_config**

函数ospi_data_length_config描述见下表:

表 3-785. 函数 ospi_data_length_config

函数名称	ospi_data_length_config
函数原形	void ospi_data_length_config(uint32_t ospi_periph, uint32_t dtlen);
功能描述	配置数据长度
先决条件	-
被调用函数	-
输入参数{in}	
ospi_periph	OSPIx (x=0,1)
输入参数{in}	
dtlen	数据长度 (0-0xFFFFFFFF)
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* configure data length */
```

```
ospi_data_length_config(OSPI0, 0x00005555);
```

函数 **ospi_instruction_config**

函数ospi_instruction_config描述见下表:

表 3-786. 函数 `ospi_instruction_config`

函数名称	<code>ospi_instruction_config</code>
函数原形	<code>void ospi_instruction_config(uint32_t ospi_periph, uint32_t imod, uint32_t inssz, uint32_t instruction);</code>
功能描述	配置指令模式
先决条件	-
被调用函数	-
输入参数{in}	
ospi_periph	OSPIx (x=0,1)
输入参数{in}	
imod	OSPI指令模式
<code>OSPI_INSTRUCTION_NONE</code>	无指令模式
<code>OSPI_INSTRUCTION_1_LINE</code>	单线指令模式
<code>OSPI_INSTRUCTION_2_LINES</code>	双线指令模式
<code>OSPI_INSTRUCTION_4_LINES</code>	四线指令模式
<code>OSPI_INSTRUCTION_8_LINES</code>	八线指令模式
输入参数{in}	
输入参数{in}	
inssz	OSPI指令大小
<code>OSPI_INSTRUCTION_8_BITS</code>	8位指令
<code>OSPI_INSTRUCTION_16_BITS</code>	16位指令
<code>OSPI_INSTRUCTION_24_BITS</code>	24位指令
<code>OSPI_INSTRUCTION_32_BITS</code>	32位指令
输入参数{in}	
instruction	要发送的指令 (0-0xFFFFFFFF)
输出参数{out}	
-	-
返回值	
-	-

例如:

/* configure instruction mode */

`ospi_instruction_config(OSPI0,``OSPI_INSTRUCTION_8_LINES,`

```
OSPI_INSTRUCTION_8_BITS, 0x00005555);
```

函数 `ospi_address_config`

函数 `ospi_address_config` 描述见下表：

表 3-787. 函数 `ospi_address_config`

函数名称	<code>ospi_address_config</code>
函数原形	<code>void ospi_address_config(uint32_t ospi_periph, uint32_t addrmod, uint32_t addrdrtr, uint32_t addrsh, uint32_t addr);</code>
功能描述	配置地址模式
先决条件	-
被调用函数	-
输入参数{in}	
ospi_periph	OSPIx (x=0,1)
输入参数{in}	
addrmod	OSPI地址模式
<code>OSPI_ADDRESS_NONE</code>	无地址模式
<code>OSPI_ADDRESS_1_LINE</code>	单线地址模式
<code>OSPI_ADDRESS_2_LINES</code>	双线地址模式
<code>OSPI_ADDRESS_4_LINES</code>	四线地址模式
<code>OSPI_ADDRESS_8_LINES</code>	八线地址模式
输入参数{in}	
addrdrtr	OSPI地址DTR模式
<code>OSPI_ADDRDTR_MODE_DISABLE</code>	地址阶段禁用DTR
<code>OSPI_ADDRDTR_MODE_ENABLE</code>	地址阶段使能DTR
输入参数{in}	
addrsh	OSPI地址大小
<code>OSPI_ADDRESS_8_BITS</code>	8位地址
<code>OSPI_ADDRESS_16_BITS</code>	16位地址
<code>OSPI_ADDRESS_24_BITS</code>	24位地址
<code>OSPI_ADDRESS_32_BITS</code>	32位地址

输入参数{in}	
addr	要发送的地址（0-0xFFFFFFFF）
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* configure address mode */
```

```
ospi_address_config(OSPI0, OSPI_ADDRESS_8_LINES,
OSPI_ADDRDTR_MODE_DISABLE, OSPI_ADDRESS_32_BITS, 0xFFFFFFFF);
```

函数 **ospi_alterate_bytes_config**

函数ospi_alterate_bytes_config描述见下表：

表 3-788. 函数 ospi_alterate_bytes_config

函数名称	ospi_alterate_bytes_config
函数原形	void ospi_alterate_bytes_config(uint32_t ospi_periph, uint32_t atlemod, uint32_t abdtr, uint32_t altesz, uint32_t alte)
功能描述	配置交替字节模式
先决条件	-
被调用函数	-
输入参数{in}	
ospi_periph	OSPIx (x=0,1)
输入参数{in}	
atlemod	OSPI交替字节模式
OSPI_ALTERNATE_BYTES_NONE	无交替字节模式
OSPI_ALTERNATE_BYTES_1_LINE	单线交替字节模式
OSPI_ALTERNATE_BYTES_2_LINES	双线交替字节模式
OSPI_ALTERNATE_BYTES_4_LINES	四线交替字节模式
OSPI_ALTERNATE_BYTES_8_LINES	八线交替字节模式
输入参数{in}	
abdtr	OSPI交替字节DTR模式
OSPI_ABDTR_MODE_DISABLE	交替字节阶段禁用DTR
OSPI_ABDTR_MODE_ENABLE	交替字节阶段使能DTR

输入参数{in}	
altesz	OSPI交替字节大小
<i>OSPI_ALTERNATE_BYTES_8_BITS</i>	8位交替字节
<i>OSPI_ALTERNATE_BYTES_16_BITS</i>	16位交替字节
<i>OSPI_ALTERNATE_BYTES_24_BITS</i>	24位交替字节
<i>OSPI_ALTERNATE_BYTES_32_BITS</i>	32位交替字节
输入参数{in}	
alte	要发送的交替字节（0-0xFFFFFFFF）
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* configure alternate bytes mode */
```

```
ospi_alternate_bytes_config(OSPI0, OSPI_ALTERNATE_BYTES_8_LINES,
OSPI_ABDTR_MODE_DISABLE, OSPI_ALTERNATE_BYTES_32_BITS, 0xFFFFFFFF);
```

函数 **ospi_data_config**

函数ospi_data_config描述见下表：

表 3-789. 函数 ospi_data_config

函数名称	ospi_data_config
函数原形	void ospi_data_config(uint32_t ospi_periph, uint32_t datamod, uint32_t dadtr);
功能描述	配置数据模式
先决条件	-
被调用函数	-
输入参数{in}	
ospi_periph	OSPIx (x=0,1)
输入参数{in}	
datamod	OSPI数据模式
<i>OSPI_DATA_NONE</i>	无数据模式
<i>OSPI_DATA_1_LINE</i>	单线数据模式
<i>OSPI_DATA_2_LINES</i>	双线数据模式
<i>OSPI_DATA_4_LINES</i>	四线数据模式

OSPI_DATA_8_LINES	八线数据模式
输入参数{in}	
dadtr	OSPI数据DTR模式
OSPI_DADTR_MODE_DISABLE	数据阶段禁用DTR
OSPI_DADTR_MODE_ENABLE	数据阶段使能DTR
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* configure data mode */
```

```
ospi_data_config(OSPI0, OSPI_DATA_8_LINES, OSPI_DADTR_MODE_ENABLE);
```

函数 ospi_data_transmit

函数描述ospi_data_transmit见下表:

表 3-790. 函数 ospi_data_transmit

函数名称	ospi_data_transmit
函数原形	void ospi_data_transmit(uint32_t ospi_periph, uint32_t data);
功能描述	OSPI发送数据
先决条件	-
被调用函数	-
输入参数{in}	
ospi_periph	OSPIx (x=0,1)
输入参数{in}	
data	transmit data (0-0xFFFFFFFF)
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* OSPI transmit data */
```

```
ospi_data_transmit(OSPI0, 0x00FF0000);
```

函数 ospi_data_receive

函数ospi_data_receive描述见下表:

表 3-791. 函数 **ospi_data_receive**

函数名称	ospi_data_receive
函数原形	uint32_t ospi_data_receive(uint32_t ospi_periph);
功能描述	OSPI接收数据
先决条件	-
被调用函数	-
输入参数{in}	
ospi_periph	OSPIx (x=0,1)
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* OSPI receive data */  
ospi_data_receive(OSPI0);
```

函数 **ospi_dma_enable**

函数ospi_dma_enable描述见下表:

表 3-792. 函数 **ospi_dma_enable**

函数名称	ospi_dma_enable
函数原形	void ospi_dma_enable(uint32_t ospi_periph);
功能描述	使能OSPI DMA
先决条件	-
被调用函数	-
输入参数{in}	
ospi_periph	OSPIx (x=0,1)
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* enable OSPI DMA */  
ospi_dma_enable(OSPI0);
```

函数 **ospi_dma_disable**

函数ospi_dma_disable描述见下表:

表 3-793. 函数 `ospi_dma_disable`

函数名称	<code>ospi_dma_disable</code>
函数原形	<code>void ospi_dma_disable(uint32_t ospi_periph);</code>
功能描述	失能OSPI DMA
先决条件	-
被调用函数	-
输入参数{in}	
<code>ospi_periph</code>	OSPIx (x=0,1)
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* disable OSPI DMA */
```

```
ospi_dma_disable(OSPI0);
```

函数 `ospi_wrap_size_config`

函数`ospi_wrap_size_config`描述见下表:

表 3-794. 函数 `ospi_wrap_size_config`

函数名称	<code>ospi_wrap_size_config</code>
函数原形	<code>void ospi_wrap_size_config(uint32_t ospi_periph, uint32_t wpsz);</code>
功能描述	配置回卷大小
先决条件	-
被调用函数	-
输入参数{in}	
<code>ospi_periph</code>	OSPIx (x=0,1)
输入参数{in}	
<code>wpsz</code>	OSPI回卷大小设置
<code>OSPI_DIRECT</code>	外部存储器设备不支持回卷读取
<code>OSPI_WRAP_16BYTES</code>	外部存储器设备支持16字节回卷大小
<code>OSPI_WRAP_32BYTES</code>	外部存储器设备支持32字节回卷大小
<code>OSPI_WRAP_64BYTES</code>	外部存储器设备支持64字节回卷大小
<code>OSPI_WRAP_128BYTES</code>	外部存储器设备支持128字节回卷大小
输出参数{out}	
-	-
返回值	

-	-
---	---

例如:

```
/* configure wrap size */
```

```
ospi_wrap_size_config(OSPI0, OSPI_WRAP_32BYTES);
```

函数 `ospi_wrap_instruction_config`

函数 `ospi_wrap_instruction_config` 描述见下表:

表 3-795. 函数 `ospi_wrap_instruction_config`

函数名称	<code>ospi_wrap_instruction_config</code>
函数原形	<code>void ospi_wrap_instruction_config(uint32_t ospi_periph, uint32_t imod, uint32_t inssz, uint32_t instruction);</code>
功能描述	配置回卷指令模式
先决条件	-
被调用函数	-
输入参数{in}	
ospi_periph	OSPIx (x=0,1)
输入参数{in}	
imod	OSPI指令模式
<code>OSPI_INSTRUCTION_NONE</code>	无指令模式
<code>OSPI_INSTRUCTION_1_LINE</code>	单线指令模式
<code>OSPI_INSTRUCTION_2_LINES</code>	双线指令模式
<code>OSPI_INSTRUCTION_4_LINES</code>	四线指令模式
<code>OSPI_INSTRUCTION_8_LINES</code>	八线指令模式
输入参数{in}	
inssz	OSPI指令大小
<code>OSPI_INSTRUCTION_8_BITS</code>	8位指令
<code>OSPI_INSTRUCTION_16_BITS</code>	16位指令
<code>OSPI_INSTRUCTION_24_BITS</code>	24位指令
<code>OSPI_INSTRUCTION_32_BITS</code>	32位指令
输入参数{in}	
instruction	要发送的指令 (0-0xFFFFFFFF)

输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* configure wrap instruction mode */
```

```
ospi_wrap_instruction_config(OSPI0, OSPI_INSTRUCTION_8_LINES,
OSPI_INSTRUCTION_8_BITS, 0x00005555);
```

函数 `ospi_wrap_address_config`

函数 `ospi_wrap_address_config` 描述见下表:

表 3-796. 函数 `ospi_wrap_address_config`

函数名称	<code>ospi_wrap_address_config</code>
函数原形	<code>void ospi_wrap_address_config (uint32_t ospi_periph, uint32_t addrmod, uint32_t addrdtr, uint32_t addrsz, uint32_t addr);</code>
功能描述	Configure wrap address mode
先决条件	-
被调用函数	-
输入参数{in}	
ospi_periph	OSPIx (x=0,1)
输入参数{in}	
addrmod	OSPI地址模式
<code>OSPI_ADDRESS_NONE</code>	无地址模式
<code>OSPI_ADDRESS_1_LINE</code>	单线地址模式
<code>OSPI_ADDRESS_2_LINES</code>	双线地址模式
<code>OSPI_ADDRESS_4_LINES</code>	四线地址模式
<code>OSPI_ADDRESS_8_LINES</code>	八线地址模式
输入参数{in}	
addrdtr	OSPI地址DTR模式
<code>OSPI_ADDRDTR_MODE_DISABLE</code>	地址阶段禁用DTR
<code>OSPI_ADDTR_MODE_ENABLE</code>	地址阶段使能DTR
输入参数{in}	
addrsz	OSPI地址大小

OSPI_ADDRESS_8_BITS	8位地址
OSPI_ADDRESS_16_BITS	16位地址
OSPI_ADDRESS_24_BITS	24位地址
OSPI_ADDRESS_32_BITS	32位地址
输入参数{in}	
addr	要发送的地址（0-0xFFFFFFFF）
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* configure wrap address mode */
```

```
ospi_wrap_address_config(OSPI0, OSPI_ADDRESS_8_LINES,
OSPI_ADDRDTR_MODE_DISABLE, OSPI_ADDRESS_32_BITS, 0xFFFFFFFF);
```

函数 ospi_wrap_alternate_bytes_config

函数ospi_wrap_alternate_bytes_config描述见下表：

表 3-797. 函数 ospi_wrap_alternate_bytes_config

函数名称	ospi_wrap_alternate_bytes_config
函数原形	void ospi_wrap_alternate_bytes_config(uint32_t ospi_periph, uint32_t atlemod, uint32_t abdtr, uint32_t altesz, uint32_t alte)
功能描述	配置回卷交替字节模式
先决条件	-
被调用函数	-
输入参数{in}	
ospi_periph	OSPIx (x=0,1)
输入参数{in}	
atlemod	OSPI交替字节模式
OSPI_ALTERNATE_BYTES_NONE	无交替字节模式
OSPI_ALTERNATE_BYTES_1_LINE	单线交替字节模式
OSPI_ALTERNATE_BYTES_2_LINES	双线交替字节模式
OSPI_ALTERNATE_BYTES_4_LINES	四线交替字节模式

OSPI_ALTERNATE_BYTES_8_LINES	八线交替字节模式
输入参数{in}	
abdtr	OSPI交替字节DTR模式
OSPI_ABDTR_MODE_DISABLE	交替字节阶段禁用DTR
OSPI_ABDTR_MODE_ENABLE	交替字节阶段使能DTR
输入参数{in}	
altesz	OSPI交替字节大小
OSPI_ALTERNATE_BYTES_8_BITS	8位交替字节
OSPI_ALTERNATE_BYTES_16_BITS	16位交替字节
OSPI_ALTERNATE_BYTES_24_BITS	24位交替字节
OSPI_ALTERNATE_BYTES_32_BITS	32位交替字节
输入参数{in}	
alte	要发送的交替字节（0-0xFFFFFFFF）
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* configure wrap alternate bytes mode */
```

```
ospi_wrap_alternate_bytes_config(OSPI0, OSPI_ALTERNATE_BYTES_8_LINES,
OSPI_ABDTR_MODE_DISABLE, OSPI_ALTERNATE_BYTES_32_BITS, 0xFFFFFFFF);
```

函数 ospi_wrap_data_config

函数ospi_wrap_data_config描述见下表：

表 3-798. 函数 ospi_wrap_data_config

函数名称	ospi_wrap_data_config
函数原形	void ospi_wrap_data_config(uint32_t ospi_periph, uint32_t datamod, uint32_t dadtr);
功能描述	配置回卷数据模式
先决条件	-
被调用函数	-
输入参数{in}	
ospi_periph	OSPIx (x=0,1)

输入参数{in}	
datamod	OSPI数据模式
<i>OSPI_DATA_NONE</i>	无数据模式
<i>OSPI_DATA_1_LINES</i>	单线数据模式
<i>OSPI_DATA_2_LINES</i>	双线数据模式
<i>OSPI_DATA_4_LINES</i>	四线数据模式
<i>OSPI_DATA_8_LINES</i>	八线数据模式
输入参数{in}	
dadtr	OSPI数据DTR模式
<i>OSPI_DADTR_MODE_DISABLE</i>	数据阶段禁用DTR
<i>OSPI_DADTR_MODE_ENABLE</i>	数据阶段使能DTR
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* configure wrap data mode */
```

```
ospi_wrap_data_config(OSPI0, OSPI_DATA_8_LINES, OSPI_DADTR_MODE_ENABLE);
```

函数 **ospi_wrap_dummy_cycles_config**

函数ospi_wrap_dummy_cycles_config描述见下表:

表 3-799. 函数 ospi_wrap_dummy_cycles_config

函数名称	ospi_wrap_dummy_cycles_config
函数原形	void ospi_wrap_dummy_cycles_config(uint32_t ospi_periph, uint32_t dumyc);
功能描述	配置回卷空指令周期数
先决条件	-
被调用函数	-
输入参数{in}	
ospi_periph	OSPIx (x=0,1)
输入参数{in}	
dumyc	空指令周期数(0-0x1F)
输出参数{out}	
-	-
返回值	

-	-
---	---

例如:

```
/* configure wrap dummy cycles number */
ospi_wrap_dummy_cycles_config(OSPI0, 0x0F);
```

函数 **ospi_wrap_delay_hold_cycle_config**

函数ospi_wrap_delay_hold_cycle_config描述见下表:

表 3-800. 函数 ospi_wrap_delay_hold_cycle_config

函数名称	ospi_wrap_delay_hold_cycle_config
函数原形	void ospi_wrap_delay_hold_cycle_config(uint32_t ospi_periph, uint32_t dehqc);
功能描述	配置回卷延迟保持1/4周期
先决条件	-
被调用函数	-
输入参数{in}	
ospi_periph	OSPIx (x=0,1)
输入参数{in}	
dehqc	OSPI延迟保持1/4周期
OSPI_DELAY_HOLD_NONE	OSPI无延迟保持
OSPI_DELAY_HOLD_QUARTER_CYCLE	OSPI延迟保持1/4周期
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* delay hold 1/4 cycle in wrap */
ospi_wrap_delay_hold_cycle_config(OSPI0, OSPI_DELAY_HOLD_QUARTER_CYCLE);
```

函数 **ospi_wrap_sample_shift_config**

函数ospi_wrap_sample_shift_config描述见下表:

表 3-801. 函数 ospi_wrap_sample_shift_config

函数名称	ospi_wrap_sample_shift_config
函数原形	void ospi_wrap_sample_shift_config(uint32_t ospi_periph, uint32_t ssample);
功能描述	配置回卷采样移位
先决条件	-

被调用函数	-
输入参数{in}	
ospi_periph	OSPIx (x=0,1)
输入参数{in}	
ssample	OSPI采样移位
OSPI_SAMPLE_SHIFTING_NONE	OSPI无采样移位
OSPI_SAMPLE_SHIFTING_HALF_CYCLE	OSPI采样移位1/2周期
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* configure sample shift in wrap */
```

```
ospi_wrap_sample_shift_config(OSPI0, OSPI_SAMPLE_SHIFTING_HALF_CYCLE);
```

函数 ospi_write_instruction_config

函数ospi_write_instruction_config描述见下表:

表 3-802. 函数 ospi_write_instruction_config

函数名称	ospi_write_instruction_config
函数原形	void ospi_write_instruction_config(uint32_t ospi_periph, uint32_t imod, uint32_t inssz, uint32_t instruction);
功能描述	配置写入指令模式
先决条件	-
被调用函数	-
输入参数{in}	
ospi_periph	OSPIx (x=0,1)
输入参数{in}	
imod	OSPI指令模式
OSPI_INSTRUCTION_NONE	无指令模式
OSPI_INSTRUCTION_1_LINE	单线指令模式
OSPI_INSTRUCTION_2_LINES	双线指令模式
OSPI_INSTRUCTION_4_LINES	四线指令模式
OSPI_INSTRUCTION_8_LINES	八线指令模式

<i>N_8_LINES</i>	
输入参数{in}	
inssz	OSPI指令大小
<i>OSPI_INSTRUCTION_8_BITS</i>	8位指令
<i>OSPI_INSTRUCTION_16_BITS</i>	16位指令
<i>OSPI_INSTRUCTION_24_BITS</i>	24位指令
<i>OSPI_INSTRUCTION_32_BITS</i>	32位指令
输入参数{in}	
instruction	要发送的指令（0-0xFFFFFFFF）
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* configure wrap instruction mode */
```

```
ospi_wrtie_instruction_config(OSPI0, OSPI_INSTRUCTION_8_LINES, OSPI_INSTRUCTION_8_BITS, 0x00005555);
```

函数 **ospi_write_address_config**

函数ospi_write_address_config描述见下表：

表 3-803. 函数 ospi_write_address_config

函数名称	ospi_write_address_config
函数原形	void ospi_write_address_config(uint32_t ospi_periph, uint32_t addrmod, uint32_t addrdtr, uint32_t addrsz, uint32_t addr);
功能描述	配置写入地址模式
先决条件	-
被调用函数	-
输入参数{in}	
ospi_periph	OSPIx (x=0,1)
输入参数{in}	
addrmod	OSPI地址模式
<i>OSPI_ADDRESS_NONE</i>	无地址模式
<i>OSPI_ADDRESS_1_LINE</i>	单线地址模式
<i>OSPI_ADDRESS_2</i>	双线地址模式

<code>_LINES</code>	
<code>OSPI_ADDRESS_4_LINES</code>	四线地址模式
<code>OSPI_ADDRESS_8_LINES</code>	八线地址模式
输入参数{in}	
<code>addrdtr</code>	OSPI地址DTR模式
<code>OSPI_ADDRDTR_MODE_DISABLE</code>	地址阶段禁用DTR
<code>OSPI_ADDTR_MODE_ENABLE</code>	地址阶段使能DTR
输入参数{in}	
<code>addrsz</code>	OSPI地址大小
<code>OSPI_ADDRESS_8_BITS</code>	8位地址
<code>OSPI_ADDRESS_16_BITS</code>	16位地址
<code>OSPI_ADDRESS_24_BITS</code>	24位地址
<code>OSPI_ADDRESS_32_BITS</code>	32位地址
输入参数{in}	
<code>addr</code>	要发送的地址（0-0xFFFFFFFF）
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* configure write address mode */
```

```
ospi_write_address_config(OSPI0, OSPI_ADDRESS_8_LINES,
OSPI_ADDRDTR_MODE_DISABLE, OSPI_ADDRESS_32_BITS, 0xFFFFFFFF);
```

函数 `ospi_write_alternate_bytes_config`

函数 `ospi_write_alternate_bytes_config` 描述见下表：

表 3-804. 函数 `ospi_write_alternate_bytes_config`

函数名称	<code>ospi_write_alternate_bytes_config</code>
函数原形	<code>void ospi_write_alternate_bytes_config(uint32_t ospi_periph, uint32_t atlemod, uint32_t abdtr, uint32_t altesz, uint32_t alte);</code>
功能描述	配置写入交替字节模式
先决条件	-

被调用函数	-
输入参数{in}	
ospi_periph	OSPIx (x=0,1)
输入参数{in}	
atlemod	OSPI交替字节模式
OSPI_ALTERNATE_BYTES_NONE	无交替字节模式
OSPI_ALTERNATE_BYTES_1_LINE	单线交替字节模式
OSPI_ALTERNATE_BYTES_2_LINES	双线交替字节模式
OSPI_ALTERNATE_BYTES_4_LINES	四线交替字节模式
OSPI_ALTERNATE_BYTES_8_LINES	八线交替字节模式
输入参数{in}	
abdtr	OSPI交替字节DTR模式
OSPI_ABDTR_MODE_DISABLE	交替字节阶段禁用DTR
OSPI_ABDTR_MODE_ENABLE	交替字节阶段使能DTR
输入参数{in}	
altesz	OSPI交替字节大小
OSPI_ALTERNATE_BYTES_8_BITS	8位交替字节
OSPI_ALTERNATE_BYTES_16_BITS	16位交替字节
OSPI_ALTERNATE_BYTES_24_BITS	24位交替字节
OSPI_ALTERNATE_BYTES_32_BITS	32位交替字节
输入参数{in}	
alte	要发送的交替字节 (0-0xFFFFFFFF)
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* configure write alternate bytes mode */
```

```
ospi_write_alternate_bytes_config(OSPI0, OSPI_ALTERNATE_BYTES_8_LINES,
OSPI_ABDTR_MODE_DISABLE, OSPI_ALTERNATE_BYTES_32_BITS, 0xFFFFFFFF);
```

函数 `ospi_write_data_config`

函数 `ospi_write_data_config` 描述见下表：

表 3-805. 函数 `ospi_write_data_config`

函数名称	<code>ospi_write_data_config</code>
函数原形	<code>void ospi_write_data_config(uint32_t ospi_periph, uint32_t datamod, uint32_t dadtr);</code>
功能描述	配置写入数据模式
先决条件	-
被调用函数	-
输入参数{in}	
ospi_periph	OSPIx (x=0,1)
输入参数{in}	
datamod	OSPI数据模式
<code>OSPI_DATA_NONE</code>	无数据模式
<code>OSPI_DATA_1_LINES</code>	单线数据模式
<code>OSPI_DATA_2_LINES</code>	双线数据模式
<code>OSPI_DATA_4_LINES</code>	四线数据模式
<code>OSPI_DATA_8_LINES</code>	八线数据模式
输入参数{in}	
dadtr	OSPI数据DTR模式
<code>OSPI_DADTR_MODE_DISABLE</code>	数据阶段禁用DTR
<code>OSPI_DADTR_MODE_ENABLE</code>	数据阶段使能DTR
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* configure write data mode */
```

```
ospi_write_data_config(OSPI0, OSPI_DATA_8_LINES, OSPI_DADTR_MODE_ENABLE);
```

函数 `ospi_write_dummy_cycles_config`

函数 `ospi_write_dummy_cycles_config` 描述见下表：

表 3-806. 函数 `ospi_write_dummy_cycles_config`

函数名称	<code>ospi_write_dummy_cycles_config</code>
函数原形	<code>void ospi_write_dummy_cycles_config(uint32_t ospi_periph, uint32_t dumyc);</code>
功能描述	配置写入空指令周期数
先决条件	-
被调用函数	-
输入参数{in}	
<code>ospi_periph</code>	OSPIx (x=0,1)
输入参数{in}	
<code>dumyc</code>	空指令周期数
<code>OSPI_DUMYC_CYCLES_x</code>	空指令周期数设置为x (x=0,1,2,...,30,31)
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* configure write dummy cycles number */
```

```
ospi_write_dummy_cycles_config(OSPI0, OSPI_DUMYC_CYCLES_10);
```

函数 `ospi_command_config`

函数`ospi_command_config`描述见下表:

表 3-807. 函数 `ospi_write_dummy_cycles_config`

函数名称	<code>ospi_write_dummy_cycles_config</code>
函数原形	<code>void ospi_command_config(uint32_t ospi_periph, ospi_parameter_struct *ospi_struct, ospi_regular_cmd_struct *cmd_struct);</code>
功能描述	配置OSPI常规命令参数
先决条件	-
被调用函数	-
输入参数{in}	
<code>ospi_periph</code>	OSPIx(x=0,1)
输入参数{in}	
<code>ospi_struct</code>	OSPI 参数结构体, 结构体成员参考 表 3-763. 结构体 <code>ospi_parameter_struct</code>
输入参数{in}	
<code>cmd_struct</code>	OSPI 命令, 结构体成员参考 表 3-764. 结构体 <code>ospi_regular_cmd_struct</code>
输出参数{out}	
-	-
返回值	
-	-

Example:

```
/* configure OSPI regular command parameter */
```

```
ospi_command_config(OSPI0, ospi_struct, cmd_struct);
```

函数 **ospi_transmit**

函数ospi_transmit描述见下表:

表 3-808. 函数 **ospi_transmit**

函数名称	ospi_transmit
函数原形	void ospi_transmit(uint32_t ospi_periph, uint8_t *pdata);
功能描述	发送数据
先决条件	-
被调用函数	-
输入参数{in}	
ospi_periph	OSPIx(x=0,1)
输入参数{in}	
pdata	数据缓冲区指针
输出参数{out}	
-	-
返回值	
-	-

Example:

```
/* transmit data */
```

```
ospi_transmit(OSPI0, pdata);
```

函数 **ospi_receive**

函数ospi_receive描述见下表:

表 3-809. 函数 **ospi_receive**

函数名称	ospi_receive
函数原形	void ospi_receive(uint32_t ospi_periph, uint8_t *pdata);
功能描述	接收数据
先决条件	-
被调用函数	-
输入参数{in}	
ospi_periph	OSPIx(x=0,1)
输入参数{in}	
pdata	数据缓冲区指针
输出参数{out}	
-	-

返回值	
-	-

Example:

```
/* receive data */
```

```
ospi_receive(OSPI0, pdata);
```

函数 **ospi_autopolling_mode**

函数 **ospi_autopolling_mode**描述见下表:

表 3-810. 函数 **ospi_autopolling_mode**

函数名称	ospi_autopolling_mode
函数原形	void ospi_autopolling_mode(uint32_t ospi_periph, ospi_parameter_struct *ospi_struct, ospi_autopolling_struct *autopl_cfg_struct);
功能描述	配置OSPI状态轮询模式
先决条件	-
被调用函数	-
输入参数{in}	
ospi_periph	OSPIx(x=0,1)
输入参数{in}	
ospi_struct	OSPI 参数结构体, 结构体成员参考 表 3-763. 结构体 ospi_parameter_struct
输入参数{in}	
ospi_autopolling_struct	OSPI 状态轮询结构体, 结构体成员参考 表 3-765. 结构体 ospi_autopolling_struct
输出参数{out}	
-	-
返回值	
-	-

Example:

```
/* configure the OSPI automatic polling mode */
```

```
ospi_autopolling_mode(OSPI0, ospi_struct, autopl_cfg_struct);
```

函数 **ospi_interrupt_enable**

函数**ospi_interrupt_enable**描述见下表:

表 3-811. 函数 **ospi_interrupt_enable**

函数名称	ospi_interrupt_enable
函数原形	void ospi_interrupt_enable(uint32_t ospi_periph, uint8_t interrupt);
功能描述	使能OSPI中断

先决条件	-
被调用函数	-
输入参数{in}	
ospi_periph	OSPIx (x=0,1)
输入参数{in}	
interrupt	OSPI中断
OSPI_INT_TERR	传输错误中断
OSPI_INT_TC	传输完成中断
OSPI_INT_FT	FIFO阈值中断
OSPI_INT_SM	状态匹配中断
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* enable OSPI interrupt */
```

```
ospi_interrupt_enable(OSPI0, OSPI_INT_TERR);
```

函数 ospi_interrupt_disable

函数ospi_interrupt_disable描述见下表：

表 3-812. 函数 ospi_interrupt_disable

函数名称	ospi_interrupt_disable
函数原形	void ospi_interrupt_disable(uint32_t ospi_periph, uint8_t interrupt);
功能描述	失能OSPI中断
先决条件	-
被调用函数	-
输入参数{in}	
ospi_periph	OSPIx (x=0,1)
输入参数{in}	
interrupt	OSPI中断
OSPI_INT_TERR	传输错误中断
OSPI_INT_TC	传输完成中断
OSPI_INT_FT	FIFO阈值中断
OSPI_INT_SM	状态匹配中断
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* disable OSPI interrupt */
```

```
ospi_interrupt_disable(OSPI0, OSPI_INT_TERR);
```

函数 `ospi_fifo_level_get`

函数 `ospi_fifo_level_get` 描述见下表：

表 3-813. 函数 `ospi_fifo_level_get`

函数名称	<code>ospi_fifo_level_get</code>
函数原形	<code>uint32_t ospi_fifo_level_get(uint32_t ospi_periph);</code>
功能描述	get OSPI fifo level
先决条件	-
被调用函数	-
输入参数{in}	
<code>ospi_periph</code>	OSPIx (x=0,1)
输出参数{out}	
-	-
返回值	
<code>uint32_t</code>	0-0x3F

例如：

```
/* get OSPI fifo level */
```

```
ospi_fifo_level_get(OSPI0);
```

函数 `ospi_flag_get`

函数 `ospi_flag_get` 描述见下表：

表 3-814. 函数 `ospi_flag_get`

函数名称	<code>ospi_flag_get</code>
函数原形	<code>FlagStatus ospi_flag_get(uint32_t ospi_periph, uint32_t flag);</code>
功能描述	获取OSPI状态标志
先决条件	-
被调用函数	-
输入参数{in}	
<code>ospi_periph</code>	OSPIx (x=0,1)
输入参数{in}	
<code>flag</code>	OSPI状态标志
<code>OSPI_FLAG_TERR</code>	传输错误标志
<code>OSPI_FLAG_TC</code>	传输完成标志
<code>OSPI_FLAG_FT</code>	FIFO阈值标志
<code>OSPI_FLAG_SM</code>	状态匹配
<code>OSPI_FLAG_BUSY</code>	忙标志

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

例如：

```
/* get OSPI flag status */
ospi_flag_get(OSPI0, OSPI_FLAG_BUSY);
```

函数 ospi_flag_clear

函数ospi_flag_clear描述见下表：

表 3-815. 函数 ospi_flag_clear

函数名称	ospi_flag_clear
函数原形	void ospi_flag_clear(uint32_t ospi_periph, uint32_t flag);
功能描述	清除OSPI状态标志
先决条件	-
被调用函数	-
输入参数{in}	
ospi_periph	OSPIx (x=0,1)
输入参数{in}	
flag	OSPI清除状态标志
OSPI_FLAG_TERR	传输错误标志
OSPI_FLAG_TC	传输完成标志
OSPI_FLAG_SM	状态匹配
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* clear OSPI flag status */
ospi_flag_clear(OSPI0, OSPI_STATC_TERRC);
```

函数 ospi_interrupt_flag_get

函数ospi_interrupt_flag_get描述见下表：

表 3-816. 函数 ospi_interrupt_flag_get

函数名称	ospi_interrupt_flag_get
函数原形	FlagStatus ospi_interrupt_flag_get(uint32_t ospi_periph, uint8_t int_flag);
功能描述	获取OSPI中断状态标志

先决条件	-
被调用函数	-
输入参数{in}	
ospi_periph	OSPIx (x=0,1)
输入参数{in}	
int_flag	OSPI中断状态标志
OSPI_INT_FLAG_TERR	传输错误中断标志
OSPI_INT_FLAG_TC	传输完成中断标志
OSPI_INT_FLAG_FT	FIFO阈值中断标志
OSPI_INT_FLAG_SM	状态匹配中断标志
输出参数{out}	
FlagStatus	SET或RESET
返回值	
-	-

例如:

Example:

```
/* get OSPI interrupt flag status */
```

```
ospi_interrupt_flag_get(OSPI0, OSPI_INT_FLAG_TERR);
```

函数 ospi_interrupt_flag_clear

函数ospi_interrupt_flag_clear描述见下表:

表 3-817. 函数 ospi_interrupt_flag_clear

函数名称	ospi_interrupt_flag_clear
函数原形	void ospi_interrupt_flag_clear(uint32_t ospi_periph, uint32_t int_flag);
功能描述	清除OSPI中断状态标志
先决条件	-
被调用函数	-
输入参数{in}	
ospi_periph	OSPIx (x=0,1)
输入参数{in}	
int_flag	清除OSPI中断状态标志
OSPI_INT_FLAG_TERR	传输错误中断标志
OSPI_INT_FLAG_TC	传输完成中断标志

OSPI_INT_FLAG_S M	状态匹配中断标志
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* clear OSPI interrupt flag status */
```

```
ospi_interrupt_flag_clear(OSPI0, OSPI_STATC_TERRC);
```

3.24. OSPIM

OSPI支持在复用功能映射之前, 对OSPI全IO矩阵引脚进行分配。章节[3.24.1](#)描述了OSPI的寄存器列表, 章节[3.24.2](#)对OSPI库函数进行说明。

3.24.1. 外设寄存器描述

OSPI寄存器列表如下表所示:

表 3-818. OSPIM 寄存器

寄存器名称	寄存器描述
OSPIM_PCFG0	OSPIM端口配置寄存器0

3.24.2. 外设库函数说明

OSPI库函数列表如下表所示:

表 3-819. OSPIM 库函数

库函数名称	库函数描述
ospim_deinit	复位OSPI
ospim_port_sck_config	配置端口的SCK
ospim_port_csn_config	配置端口的CSN
ospim_port_io3_0_config	配置端口的IO[3:0]
ospim_port_io3_0_source_select	选择端口的IO[3:0]源
ospim_port_io7_4_config	配置端口的IO[7:4]
ospim_port_io7_4_source_select	选择端口的IO[7:4]源

函数 ospim_deinit

函数ospim_deinit描述见下表:

表 3-820. 函数 `ospim_deinit`

函数名称	<code>ospim_deinit</code>
函数原形	<code>void ospim_deinit();</code>
功能描述	复位OSPIM
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}	
-	-
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* reset the OSPIM peripheral */
```

```
ospim_deinit();
```

函数 `ospim_port_sck_config`

函数`ospim_port_sck_config`描述见下表：

表 3-821. 函数 `ospim_port_sck_config`

函数名称	<code>ospim_port_sck_config</code>
函数原形	<code>void ospim_port_sck_config(uint32_t sckconfig);</code>
功能描述	配置端口的SCK
先决条件	-
被调用函数	-
输入参数{in}	
sckconfig	使能/失能SCK
<code>OSPIM_PORT_SCK_DISABLE</code>	失能SCK
<code>OSPIM_PORT_SCK_ENABLE</code>	使能SCK
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* configurate SCK for port */
```

```
ospim_port_sck_config(OSPIM_PORT_SCK_ENABLE);
```

函数 ospim_port_csn_config

函数ospim_port_csn_config描述见下表:

表 3-822. 函数 ospim_port_csn_config

函数名称	ospim_port_csn_config
函数原形	void ospim_port_csn_config(uint32_t csnconfig);
功能描述	configure CSN for port
先决条件	-
被调用函数	-
输入参数{in}	
csnconfig	使能/失能CSN
OSPIM_PORT_CS N_DISABLE	失能CSN
OSPIM_PORT_CS N_ENABLE	使能CSN
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* configure CSN for port */
```

```
ospim_port_csn_config(OSPIM_PORT0, OSPIM_PORT_CSN_ENABLE);
```

函数 ospim_port_io3_0_config

函数ospim_port_io3_0_config描述见下表:

表 3-823. 函数 ospim_port_io3_0_config

函数名称	ospim_port_io3_0_config
函数原形	void ospim_port_io3_0_config(uint32_t ioconfig);
功能描述	配置端口的IO[3:0]
先决条件	-
被调用函数	-
输入参数{in}	
ioconfig	使能/失能IO[3:0]
OSPIM_IO_LOW_D ISABLE	失能IO[3:0]
OSPIM_IO_LOW_E NABLE	使能IO[3:0]
输出参数{out}	
-	-
返回值	

例如:

```
/* configurate IO[3:0] for port */
```

```
ospim_port_io3_0_config(OSPIM_IO_LOW_ENABLE);
```

函数 ospim_port_io3_0_source_select

函数ospim_port_io3_0_source_select描述见下表:

表 3-824. 函数 ospim_port_io3_0_source_select

函数名称	ospim_port_io3_0_source_select
函数原形	void ospim_port_io3_0_source_select(uint32_t io_source);
功能描述	选择端口IO[3:0]的源
先决条件	-
被调用函数	-
输入参数{in}	
csn_source	IO[3:0]源
OSPIM_SRCPLIO_ OSPI0_IO_LOW	选择OSPI0_IO[3:0]
OSPIM_SRCPLIO_ OSPI0_IO_HIGH	选择OSPI0_IO[7:4]
OSPIM_SRCPLIO_ OSPI1_IO_LOW	选择OSPI1_IO[3:0]
OSPIM_SRCPLIO_ OSPI1_IO_HIGH	选择OSPI1_IO[7:4]
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* configurate IO[3:0] for port */
```

```
ospim_port_io3_0_config(OSPIM_IO_LOW_ENABLE);
```

函数 ospim_port_io7_4_config

函数ospim_port_io7_4_config描述见下表:

表 3-825. 函数 ospim_port_io7_4_config

函数名称	ospim_port_io7_4_config
函数原形	void ospim_port_io7_4_config(uint32_t ioconfig);
功能描述	配置端口的IO[7:4]

先决条件	-
被调用函数	-
输入参数{in}	
ioconfig	使能/失能IO[7:4]
OSPIM_IO_HIGH_DISABLE	失能IO[7:4]
OSPIM_IO_HIGH_ENABLE	使能IO[7:4]
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* configurate IO[7:4] for port */
```

```
ospim_port_io7_4_config(OSPIM_IO_HIGH_ENABLE);
```

函数 ospim_port_io7_4_source_select

函数ospim_port_io7_4_source_select描述见下表:

表 3-826. 函数 ospim_port_io7_4_source_select

函数名称	ospim_port_io7_4_source_select
函数原形	void ospim_port_io7_4_source_select(uint32_t io_source);
功能描述	select source of IO[7:4] for port
先决条件	-
被调用函数	-
输入参数{in}	
csn_source	source of IO[3:0]
OSPIM_SRCPHIO_OSPIO_IO_LOW	选择OSPI0_IO[3:0]
OSPIM_SRCPHIO_OSPIO_IO_HIGH	选择OSPI0_IO[7:4]
OSPIM_SRCPHIO_OSPI1_IO_LOW	选择OSPI1_IO[3:0]
OSPIM_SRCPHIO_OSPI1_IO_HIGH	选择OSPI1_IO[7:4]
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* select source of IO[7:4] for port */
```

```
ospim_port_io7_4_source_select(OSPIM_SRCPHIO_OSPI0_IO_LOW);
```

3.25. 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 [3.25.1](#) 描述了 NVIC 和 SysTick 的寄存器列表，章节 [3.25.2](#) 对 MISC 库函数进行说明。

3.25.1. 外设寄存器说明

表 3-827. 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
SHPR ⁽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⁾	可配置故障状态寄存器
HFSR ⁽²⁾	硬件故障状态寄存器
DFSR ⁽²⁾	调试故障状态寄存器
MMFAR ⁽²⁾	内存管理故障地址寄存器
BFAR ⁽²⁾	总线故障地址寄存器
AFSR ⁽²⁾	辅助故障状态寄存器
ID_PFR ⁽²⁾	处理器功能寄存器
ID_DFR ⁽²⁾	调试功能寄存器
ID_AFR ⁽²⁾	辅助功能寄存器
ID_MFR ⁽²⁾	内存模型特征寄存器
ID_ISAR ⁽²⁾	指令设置属性寄存器
CLIDR ⁽²⁾	缓存级别ID寄存器
CTR ⁽²⁾	缓存类型寄存器
CCSIDR ⁽²⁾	缓存大小ID寄存器

寄存器名称	寄存器描述
CSSELR ⁽²⁾	缓存大小选择寄存器
CPACR ⁽²⁾	协处理器访问控制寄存器
STIR ⁽²⁾	软件触发中断寄存器
MVFR0 ⁽²⁾	媒介和VFP特性寄存器0
MVFR1 ⁽²⁾	媒介和VFP特性寄存器1
MVFR2 ⁽²⁾	媒介和VFP特性寄存器2
ICIALLU ⁽²⁾	无效化指令缓存所有到PoU
ICIMVAU ⁽²⁾	通过虚拟地址无效化指令缓存到PoU
DCIMVAC ⁽²⁾	通过虚拟地址无效化数据缓存到PoC
DCISW ⁽²⁾	通过路/组无效化数据缓存
DCCMVAU ⁽²⁾	通过虚拟地址清除数据缓存到PoU
DCCMVAC ⁽²⁾	通过虚拟地址清除数据缓存到PoC
DCCSW ⁽²⁾	通过路/组清除数据缓存
DCCIMVAC ⁽²⁾	通过虚拟地址清除和无效化数据缓存到PoC
DCCISW ⁽²⁾	通过路/组清除和无效化数据缓存
ITCMCR ⁽²⁾	ITCM控制寄存器
DTCMCR ⁽²⁾	DTCM控制寄存器
AHBPCR ⁽²⁾	AHBP控制寄存器
CACR ⁽²⁾	L1级缓存控制寄存器
AHBSCR ⁽²⁾	AHB从控制寄存器
ABFSR ⁽²⁾	辅助总线故障状态寄存器

1. 参考 core_cm7.h 文件中定义的结构体类型 NVIC_Type
2. 参考 core_cm7.h 文件中定义的结构体类型 SCB_Type

表 3-828. 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_cm7.h 文件中定义的结构体类型 SysTick_Type

3.25.2. 外设库函数说明

MISC库函数列表如下表所示：

表 3-829. MISC 库函数

库函数名称	库函数描述
nvic_priority_group_set	配置中断优先级组
nvic_irq_enable	使能NVIC的中断
nvic_irq_disable	禁能NVIC的中断
nvic_vector_table_set	设置向量表基地址

库函数名称	库函数描述
system_lowpower_set	设置系统低功耗模式状态
system_lowpower_reset	复位系统低功耗模式状态
systick_clksource_set	设置Systick时钟源
mpu_region_struct_para_init	将MPU结构体中所有参数初始化为默认值
mpu_config_region	配置MPU区域
mpu_region_enable	使能MPU区域

结构体 mpu_region_init_struct

表 3-830. 结构体 mpu_region_init_struct

成员名称	功能描述
region_base_address	区域基地址
region_number	区域编号
region_size	区域大小
subregion_disable	禁用子区域
tex_type	TEX类型
access_permission	访问权限(AP)字段
access_shareable	访问可共享
access_cacheable	访问可共享
access_bufferable	访问可缓冲
instruction_exec	指令执行

枚举类型 IRQn_Type

表 3-831. 枚举类型 IRQn_Type

成员名称	功能描述
WWDGT_IRQn	窗口看门狗中断
AVD_PVD_IRQn	连接到EXTI线的AVD/LVD/OVD中断
TAMPER_STAMP_LSE_IRQn	连接到EXTI线的RTC侵入和时间戳中断 LXTAL时钟阻塞中断
RTC_WKUP_IRQn	连接到EXTI线的RTC唤醒中断
FMC_IRQn	FMC全局中断
RCU_IRQn	RCU全局中断
EXTI0_IRQn	EXTI线0中断
EXTI1_IRQn	EXTI线1中断
EXTI2_IRQn	EXTI线2中断
EXTI3_IRQn	EXTI线3中断
EXTI4_IRQn	EXTI线4中断
DMA0_Channel0_IRQn	DMA0通道0全局中断
DMA0_Channel1_IRQn	DMA0通道1全局中断
DMA0_Channel2_IRQn	DMA0通道2全局中断
DMA0_Channel3_IRQn	DMA0通道3全局中断

成员名称	功能描述
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触发和换相中断
TIMER0_Channel_IRQn	TIMER0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全局和唤醒中断
USART1_IRQn	USART1全局和唤醒中断
USART2_IRQn	USART2全局和唤醒中断
EXTI10_15_IRQn	EXTI线[15:10]中断
RTC_Alarm_IRQn	连接到EXTI线的RTC闹钟中断
TIMER7_BRK_IRQn	TIMER7中止中断
TIMER7_UP_IRQn	TIMER7更新中断
TIMER7_TRG_CMT_IRQn	TIMER7触发和换相中断
TIMER7_Channel_IRQn	TIMER7捕获比较中断
DMA0_Channel7_IRQn	DMA0通道7全局中断
EXMC_IRQn	EXMC全局中断
TIMER4_IRQn	TIMER4全局中断
SPI2_IRQn	SPI2全局中断
UART3_IRQn	UART3全局中断
UART4_IRQn	UART4全局中断
TIMER5_DACOVN_IRQn	TIMER5全局中断 DAC0和DAC1下溢错误中断
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

成员名称	功能描述
DMA1_Channel5_IRQn	DMA1通道5全局中断
DMA1_Channel6_IRQn	DMA1通道6全局中断
DMA1_Channel7_IRQn	DMA1通道7全局中断
USART5_IRQn	USART5全局和唤醒中断
I2C2_EV_IRQn	I2C2事件中断
I2C2_ER_IRQn	I2C2错误中断
USBHS0_EP1_Out_IRQn	USBHS0端点1输出中断
USBHS0_EP1_In_IRQn	USBHS0端点1输入中断
USBHS0_WKUP_IRQn	连接到EXTI线的USBHS0唤醒中断
USBHS0_IRQn	USBHS0全局中断
TRNG_IRQn	TRNG全局中断
FPU_IRQn	FPU全局中断
UART6_IRQn	UART6全局中断
UART7_IRQn	UART7全局中断
SPI3_IRQn	SPI3全局中断
SPI4_IRQn	SPI4全局中断
SPI5_IRQn	SPI5全局中断
QSPI0_IRQn	OSPI0全局中断
I2C3_EV_IRQn	I2C3事件中断
I2C3_ER_IRQn	I2C3错误中断
DMAMUX0_OVR_IRQn	DMAMUX上溢中断
HPDF0_IRQn	HPDF全局中断0
HPDF1_IRQn	HPDF全局中断1
HPDF2_IRQn	HPDF全局中断2
HPDF3_IRQn	HPDF全局中断3
TIMER14_IRQn	TIMER14全局中断
TIMER15_IRQn	TIMER15全局中断
TIMER16_IRQn	TIMER16全局中断
MDMA_IRQn	MDMA全局中断
ADC2_IRQn	ADC2全局中断
CMP0_1_IRQn	CMP0和CMP1全局中断 连接到EXTI线的CMP0和CMP1中断
WWDGT_RST_IRQn	WWDGT复位中断
CTC_IRQn	CTC中断
ECC_IRQn	RAMECCMU全局中断
OSPI1_IRQn	OSPI1全局中断
FAC_IRQn	FAC全局中断
TMU_IRQn	TMU全局中断
TIMER22_IRQn	TIMER22全局中断
TIMER23_IRQn	TIMER23全局中断
TIMER40_IRQn	TIMER40全局中断

成员名称	功能描述
TIMER41_IRQn	TIMER41全局中断
TIMER42_IRQn	TIMER42全局中断
TIMER43_IRQn	TIMER43全局中断
TIMER44_IRQn	TIMER44全局中断
TIMER50_IRQn	TIMER50全局中断
TIMER51_IRQn	TIMER51全局中断
USBHS1_EP1_Out_IRQn	USBHS1端点1输出中断
USBHS1_EP1_In_IRQn	USBHS1端点1输入中断
USBHS1_WKUP_IRQn	连接到EXTI线的USBHS1唤醒中断
USBHS1_IRQn	USBHS1全局中断
CAN0_WKUP_IRQn	连接到EXTI线的CAN0唤醒中断
CAN0_MSGBUFFER_IRQn	CAN0消息缓冲区中断
CAN0_BUSOFF_IRQn	CAN0总线关闭/总线关闭完成中断
CAN0_ER_IRQn	CAN0错误中断
CAN0_ER_FAST_IRQn	CAN0快速传输错误中断
CAN0_TX_WARNING_IRQn	CAN0发送警告中断
CAN0_RX_WARNING_IRQn	CAN0接收警告中断
CAN1_WKUP_IRQn	连接到EXTI线的CAN1唤醒中断
CAN1_MSGBUFFER_IRQn	CAN1消息缓冲区中断
CAN1_BUSOFF_IRQn	CAN1总线关闭/总线关闭完成中断
CAN1_ER_IRQn	CAN1错误中断
CAN1_ER_FAST_IRQn	CAN1快速传输错误中断
CAN1_TX_WARNING_IRQn	CAN1发送警告中断
CAN1_RX_WARNING_IRQn	CAN1接收警告中断
CAN2_WKUP_IRQn	连接到EXTI线的CAN2唤醒中断
CAN2_MSGBUFFER_IRQn	CAN2消息缓冲区中断
CAN2_BUSOFF_IRQn	CAN2总线关闭/总线关闭完成中断
CAN2_ER_IRQn	CAN2错误中断
CAN2_ER_FAST_IRQn	CAN2快速传输错误中断
CAN2_TX_WARNING_IRQn	CAN2发送警告中断
CAN2_RX_WARNING_IRQn	CAN2接收警告中断
EFUSE_IRQn	EFUSE全局中断
I2C0_WKUP_IRQn	连接到EXTI线的I2C0唤醒中断
I2C1_WKUP_IRQn	连接到EXTI线的I2C1唤醒中断
I2C2_WKUP_IRQn	连接到EXTI线的I2C2唤醒中断
I2C3_WKUP_IRQn	连接到EXTI线的I2C3唤醒中断
LPDTS_IRQn	LPDTS中断
LPDTS_WKUP_IRQn	连接到EXTI线的LPDTS唤醒中断
TIMER0_DEC_IRQn	TIMER0译码器检测中断
TIMER7_DEC_IRQn	TIMER7译码器检测中断
TIMER1_DEC_IRQn	TIMER1译码器检测中断

成员名称	功能描述
TIMER2_DEC_IRQn	TIMER2译码器检测中断
TIMER3_DEC_IRQn	TIMER3译码器检测中断
TIMER4_DEC_IRQn	TIMER4译码器检测中断
TIMER22_DEC_IRQn	TIMER22译码器检测中断
TIMER23_DEC_IRQn	TIMER23译码器检测中断

函数 nvic_priority_group_set

函数nvic_priority_group_set描述见下表:

表 3-832. 函数 nvic_priority_group_set

函数名称	nvic_priority_group_set
函数原型	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位用于次优先级
NVIC_PRIGROUP_PRE1_SUB3	1位用于抢占优先级, 3位用于次优先级
NVIC_PRIGROUP_PRE2_SUB2	2位用于抢占优先级, 2位用于次优先级
NVIC_PRIGROUP_PRE3_SUB1	3位用于抢占优先级, 1位用于次优先级
NVIC_PRIGROUP_PRE4_SUB0	4位用于抢占优先级, 0位用于次优先级
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 3-833. 函数 nvic_irq_enable

函数名称	nvic_irq_enable
------	-----------------

函数原型	void nvic_irq_enable(IRQn_Type nvic_irq, uint8_t nvic_irq_pre_priority, uint8_t nvic_irq_sub_priority);
功能描述	使能NVIC中断
先决条件	-
被调用函数	nvic_priority_group_set / __NVIC_SetPriority
输入参数{in}	
nvic_irq	NVIC中断, 参考枚举类型 表3-831. 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	次优先级
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* enable window watchDog timer interrupt, pre-priority is 1, sub-priority 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 3-834. 函数 nvic_irq_disable

函数名称	nvic_irq_disable
函数原型	void nvic_irq_disable(IRQn_Type nvic_irq);
功能描述	禁能NVIC中断
先决条件	-
被调用函数	NVIC_DisableIRQ
输入参数{in}	
nvic_irq	NVIC中断, 参考枚举类型 表3-831. 枚举类型IRQn_Type
输出参数{out}	
-	-
返回值	
-	-

例如:

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

函数 `nvic_vector_table_set`

函数 `nvic_vector_table_set` 描述见下表：

表 3-835. 函数 `nvic_vector_table_set`

函数名称	<code>nvic_vector_table_set</code>
函数原型	<code>void nvic_vector_table_set(uint32_t nvic_vect_tab, uint32_t offset);</code>
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}	
<code>nvic_vect_tab</code>	RAM或者FLASH基地址
<code>NVIC_VECTTAB_RAM</code>	RAM基地址
<code>NVIC_VECTTAB_FLASH</code>	FLASH基地址
输入参数{in}	
<code>offset</code>	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 3-836. 函数 `system_lowpower_set`

函数名称	<code>system_lowpower_set</code>
函数原型	<code>void system_lowpower_set(uint8_t lowpower_mode);</code>
功能描述	设置系统低功耗模式状态
先决条件	-
被调用函数	-
输入参数{in}	
<code>lowpower_mode</code>	系统低功耗模式的状态
<code>SCB_LPM_SLEEP_EXIT_ISR</code>	该位为1时，退出ISR时一直处于低功耗模式
<code>SCB_LPM_DEEPSLEEP</code>	该位为1时，系统处于deep sleep模式
<code>SCB_LPM_WAKEUP</code>	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）

BY_ALL_INT	
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 3-837. 函数 system_lowpower_reset

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

例如：

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

```
system_lowpower_reset(SCB_LPM_SLEEP_EXIT_ISR);
```

函数 systick_clksource_set

函数systick_clksource_set描述见下表：

表 3-838. 函数 systick_clksource_set

函数名称	systick_clksource_set
------	-----------------------

函数原型	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
<i>SYSTICK_CLKSOURCE_CKSYS</i>	systick时钟源为CK_SYS时钟
<i>SYSTICK_CLKSOURCE_CKSYS_DIV8</i>	systick时钟源为CK_SYS时钟的8分频
输出参数{out}	
-	-
返回值	
-	-

例如：

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

```
systick_clksource_set(SYSTICK_CLKSOURCE_CKSYS_DIV8);
```

函数 mpu_region_struct_para_init

函数mpu_region_struct_para_init描述见下表：

表 3-839. 函数 mpu_region_struct_para_init

函数名称	mpu_region_struct_para_init
函数原型	void mpu_region_struct_para_init(mpu_region_init_struct *mpu_init_struct);
功能描述	将MPU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}	
mpu_init_struct	MPU初始化结构体，参考 表3-830. 结构体mpu_region_init_struct 。
输出参数{out}	
-	-
返回值	
-	-

例如：

```
mpu_region_init_struct mpu_init_para;
```

```
/* initialize mpu_region_init_struct with the default values */
```

```
mpu_region_struct_para_init(&mpu_init_para);
```

函数 mpu_region_config

函数mpu_region_config描述见下表:

表 3-840. 函数 mpu_region_config

函数名称	mpu_region_config
函数原型	void mpu_region_config(mpu_region_init_struct *mpu_init_struct);
功能描述	配置MPU区域
先决条件	-
被调用函数	-
输入参数{in}	
mpu_init_struct	MPU初始化结构体, 参考 表3-830. 结构体mpu_region_init_struct 。
输出参数{out}	
-	-
返回值	
-	-

例如:

```
mpu_region_init_struct mpu_init_para;

mpu_region_struct_para_init(&mpu_init_para);

mpu_init_para.region_base_address = 0x24004000;

mpu_init_para.region_number = MPU_REGION_3;

mpu_init_para.access_permission = MPU_AP_PRIV_RW;

mpu_init_para.tex_type = MPU_TEX_TYPE1;

/* configure the MPU region */

mpu_region_config(&mpu_init_para);
```

函数 mpu_region_enable

函数mpu_region_enable描述见下表:

表 3-841. 函数 mpu_region_enable

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

返回值	
-	-

例如：

```
/* enable MPU region */
mpu_region_enable();
```

3.26. 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 [3.26.1](#) 描述了 PMU 的寄存器列表，章节 [3.26.2](#) 对 PMU 库函数进行说明。

3.26.1. 外设寄存器说明

PMU 寄存器列表如下表所示：

表 3-842. PMU 寄存器

寄存器名称	寄存器描述
PMU_CTL0	PMU控制寄存器0
PMU_CS	PMU电源控制和状态寄存器
PMU_CTL1	PMU控制寄存器1
PMU_CTL2	PMU控制寄存器2
PMU_CTL3	PMU控制寄存器3
PMU_PAR	PMU参数寄存器

3.26.2. 外设库函数说明

PMU 库函数列表如下表所示：

表 3-843. PMU 库函数

库函数名称	库函数描述
pmu_deinit	复位外设PMU
pmu_lvd_select	选择低压检测阈值
pmu_lvd_enable	打开低电压检测器
pmu_lvd_disable	关闭低压检测器
pmu_vavd_select	选择模拟电压检测器阈值
pmu_vavd_enable	打开模拟电压检测器
pmu_vavd_disable	关闭模拟电压检测器
pmu_vovd_enable	打开V0.9V内核电压检测器
pmu_vovd_disable	关闭V0.9V内核电压检测器
pmu_ldo_output_select	选择LDO V0.9V供电电压
pmu_sldo_output_select	选择LDO深度模式V0.9V供电电压
pmu_vbat_charging_select	选择VBAT电池充电电阻

库函数名称	库函数描述
pmu_vbat_charging_enable	使能V _{BAT} 电池充电
pmu_vbat_charging_disable	失能V _{BAT} 电池充电
pmu_vbat_temp_monitor_enable	使能V _{BAT} 和温度检测器
pmu_vbat_temp_monitor_disable	失能V _{BAT} 和温度检测器
pmu_usb_regulator_disable	使能USB电压稳压器
pmu_usb_regulator_disable	失能USB电压稳压器
pmu_usb_voltage_detector_enable	使能V _{DD33USB} 电压检测器
pmu_usb_voltage_detector_disable	失能V _{DD33USB} 电压检测器
pmu_smps_ldo_supply_config	供电模式配置
pmu_to_sleepmode	进入睡眠模式
pmu_to_deepsleepmode	进入深度睡眠模式
pmu_to_standbymode	进入待机模式
pmu_wakeup_pin_enable	WKUP引脚唤醒使能
pmu_wakeup_pin_disable	WKUP引脚唤醒失能
pmu_backup_write_enable	备份域写使能
pmu_backup_write_disable	备份域写失能
pmu_backup_voltage_stabilizer_enable	使能备份域电压稳压器
pmu_backup_voltage_stabilizer_disable	失能备份域电压稳压器
pmu_enter_deepsleep_wait_time_config	配置进入Deep-sleep模式时IRC计数值
pmu_exit_deepsleep_wait_time_config	配置退出Deep-sleep模式时IRC计数值
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 3-844. 函数 pmu_deinit

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

例如：

```
/* reset PMU */
```

```
pmu_deinit();
```

函数 pmu_lvd_select

函数pmu_lvd_select描述见下表:

表 3-845. 函数 pmu_lvd_select

函数名称	pmu_lvd_select
函数原型	void pmu_lvd_select(uint32_t lvd_t_n);
功能描述	选择低压检测阈值
先决条件	-
被调用函数	-
输入参数{in}	
lvd_t_n	电压阈值
PMU_LVDT_0	voltage threshold is 2.1V
PMU_LVDT_1	voltage threshold is 2.3V
PMU_LVDT_2	voltage threshold is 2.4V
PMU_LVDT_3	voltage threshold is 2.6V
PMU_LVDT_4	voltage threshold is 2.7V
PMU_LVDT_5	voltage threshold is 2.9V
PMU_LVDT_6	voltage threshold is 3.0V
PMU_LVDT_7	input analog voltage on PB7 (compared with 0.8V)
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* select low voltage detector threshold as 3.0V */
```

```
pmu_lvd_select (PMU_LVDT_6);
```

函数 pmu_lvd_enable

函数pmu_lvd_enable描述见下表:

表 3-846. 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	打开低压检测器
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 3-847. 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_vavd_select

函数pmu_vavd_select描述见下表：

表 3-848. 函数 pmu_vavd_select

函数名称	pmu_vavd_select
函数原型	void pmu_vavd_select(uint32_t vavdt_n);
功能描述	选择模拟电压检测器阈值
先决条件	-
被调用函数	-
输入参数{in}	
avdt_n	选择模拟电压检测器阈值
PMU_VAVDVC_0	选择模拟电压检测器阈值为1.7V
PMU_VAVDVC_1	选择模拟电压检测器阈值为2.1V

PMU_VAVDVC_2	选择模拟电压检测器阈值为2.5V
PMU_VAVDVC_3	选择模拟电压检测器阈值为2.8V
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* select analog voltage detector threshold 2.8V */
```

```
pmu_vavd_select(PMU_VAVDVC_3);
```

函数 pmu_vavd_enable

函数pmu_vavd_enable描述见下表：

表 3-849. 函数 pmu_vavd_enable

函数名称	pmu_vavd_enable
函数原型	void pmu_vavd_enable(void);
功能描述	打开模拟电压检测器
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 analog voltage detector */
```

```
pmu_vavd_enable();
```

函数 pmu_vavd_disable

函数pmu_vavd_disable描述见下表：

表 3-850. 函数 pmu_vavd_disable

函数名称	pmu_vavd_disable
函数原型	void pmu_vavd_disable(void);
功能描述	关闭模拟电压检测器
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 analog voltage detector */
```

```
pmu_vavd_disable();
```

函数 pmu_vovd_enable

函数pmu_vovd_enable描述见下表：

表 3-851. 函数 pmu_vovd_enable

函数名称	pmu_vovd_enable
函数原型	void pmu_vovd_enable(void);
功能描述	打开V _{0.9V} 内核电压检测器
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 V0.9V voltage detector */
```

```
pmu_vovd_enable();
```

函数 pmu_vovd_disable

函数pmu_vovd_disable描述见下表：

表 3-852. 函数 pmu_vovd_disable

函数名称	pmu_vovd_disable
函数原型	void pmu_vovd_disable(void);
功能描述	关闭V _{0.9V} 内核电压检测器
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 analog voltage detector */
```

```
pmu_vovd_disable();
```

函数 pmu_ldo_output_select

函数pmu_ldo_output_select描述见下表：

表 3-853. 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_n);
功能描述	选择LDO V0.9V供电电压
先决条件	-
被调用函数	-
输入参数{in}	
ldo_n	选择LDO输出电压
PMU_LDOVS_0	选择LDO输出电压为0.8V
PMU_LDOVS_1	选择LDO输出电压为0.85V
PMU_LDOVS_2	选择LDO输出电压为0.9V
PMU_LDOVS_3	选择LDO输出电压为0.95V
PMU_LDOVS_4	选择LDO输出电压为0.975V
PMU_LDOVS_5	选择LDO输出电压为1V
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* Select LDO output voltage 1V */
```

```
pmu_ldo_output_select(PMU_LDOVS_5);
```

函数 pmu_slido_output_select

函数pmu_slido_output_select描述见下表：

表 3-854. 函数 pmu_slido_output_select

函数名称	pmu_slido_output_select
函数原型	void pmu_slido_output_select(uint32_t slido_n);

功能描述	选择LDO深度睡眠模式V _{0.9V} 内核供电电压
先决条件	-
被调用函数	-
输入参数{in}	
sldo_n	选择LDO Stop模式V _{0.9V} 供电电压
PMU_SLDOVS_0	SLDOVS输出电压为0.6V
PMU_SLDOVS_1	SLDOVS输出电压为0.7V
PMU_SLDOVS_2	SLDOVS输出电压为0.8V
PMU_SLDOVS_3	SLDOVS输出电压为0.9V
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* select Deep-sleep mode voltage 0.9V */
```

```
pmu_sldo_output_select(PMU_SLDOVS_3);
```

函数 pmu_vbat_charging_select

函数pmu_vbat_charging_select描述见下表：

表 3-855. 函数 pmu_vbat_charging_select

函数名称	pmu_vbat_charging_select
函数原型	void pmu_vbat_charging_select(uint32_t resistor);
功能描述	选择V _{BAT} 电池充电电阻
先决条件	-
被调用函数	-
输入参数{in}	
resistor	选择V _{BAT} 电池充电电阻
PMU_VCRSEL_5K	选择V _{BAT} 电池充电电阻为5 kOhms
PMU_VCRSEL_1P5K	选择V _{BAT} 电池充电电阻1.5 kOhms
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* select PMU VBAT battery charging resistor to 1.5 kOhms */
```

```
pmu_vbat_charging_select(PMU_VCRSEL_1P5K);
```

函数 pmu_vbat_charging_enable

函数pmu_vbat_charging_enable描述见下表:

表 3-856. 函数 pmu_vbat_charging_enable

函数名称	pmu_vbat_charging_enable
函数原型	void pmu_vbat_charging_enable(void);
功能描述	使能V _{BAT} 电池充电
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 VBAT battery charging */
```

```
pmu_vbat_charging_enable();
```

函数 pmu_vbat_charging_disable

函数pmu_vbat_charging_disable描述见下表:

表 3-857. 函数 pmu_vbat_charging_disable

函数名称	pmu_vbat_charging_disable
函数原型	void pmu_vbat_charging_disable(void);
功能描述	失能V _{BAT} 电池充电
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 VBAT battery charging */
```

```
pmu_vbat_charging_disable();
```

函数 pmu_vbat_temp_monitor_enable

函数pmu_vbat_temp_monitor_enable描述见下表:

表 3-858. 函数 pmu_vbat_temp_monitor_enable

函数名称	pmu_vbat_temp_monitor_enable
函数原型	void pmu_vbat_temp_monitor_enable(void);
功能描述	使能V _{BAT} 和温度检测器
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 VBAT and temperature monitoring */  
pmu_vbat_temp_monitor_enable();
```

函数 pmu_vbat_temp_monitor_disable

函数pmu_vbat_temp_monitor_disable描述见下表:

表 3-859. 函数 pmu_vbat_temp_monitor_disable

函数名称	pmu_vbat_temp_monitor_disable
函数原型	void pmu_vbat_temp_monitor_disable(void);
功能描述	失能V _{BAT} 和温度检测器
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 VBAT and temperature monitoring */  
pmu_vbat_temp_monitor_disable();
```

函数 pmu_usb_regulator_enable

函数pmu_usb_regulator_enable描述见下表：

表 3-860. 函数 pmu_usb_regulator_enable

函数名称	pmu_usb_regulator_enable
函数原型	void pmu_usb_regulator_enable(void);
功能描述	使能USB电压稳压器
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 USB regulator */
pmu_usb_regulator_enable();
```

函数 pmu_usb_regulator_disable

函数pmu_usb_regulator_disable描述见下表：

表 3-861. 函数 pmu_usb_regulator_disable

函数名称	pmu_usb_regulator_disable
函数原型	void pmu_usb_regulator_disable(void);
功能描述	失能USB电压稳压器
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 USB regulator */
pmu_usb_regulator_disable();
```


函数 pmu_usb_voltage_detector_enable

函数pmu_usb_voltage_detector_enable描述见下表：

表 3-862. 函数 pmu_usb_voltage_detector_enable

函数名称	pmu_usb_voltage_detector_enable
函数原型	void pmu_usb_voltage_detector_enable(void);
功能描述	使能V _{DD33USB} 电压检测器
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 VDD33USB voltage level detector */
```

```
pmu_usb_voltage_detector_enable();
```

函数 pmu_usb_voltage_detector_disable

函数pmu_usb_voltage_detector_disable描述见下表：

表 3-863. 函数 pmu_usb_voltage_detector_disable

函数名称	pmu_usb_voltage_detector_disable
函数原型	void pmu_usb_voltage_detector_disable(void);
功能描述	失能V _{DD33USB} 电压检测器
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 VDD33USB voltage level detector */
```

```
pmu_usb_voltage_detector_disable();
```

函数 **pmu_smpps_ldo_supply_config**

函数pmu_smpps_ldo_supply_config描述见下表:

表 3-864. 函数 **pmu_smpps_ldo_supply_config**

函数名称	pmu_smpps_ldo_supply_config
函数原型	void pmu_smpps_ldo_supply_config(uint32_t smppsmode);
功能描述	供电模式配置
先决条件	-
被调用函数	-
输入参数{in}	
smppsmode	供电模式
PMU_LDO_SUPPL Y	V _{0.9V} 电源域由LDO供电
PMU_DIRECT_SM PS_SUPPLY	V _{0.9V} 电源域由SMPS供电
PMU_BYPASS	SMPS使能、LDO旁路，外部电源供电V _{0.9V} 电源域
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* configure V0.9V domain Bypass */
```

```
pmu_smpps_ldo_supply_config(PMU_BYPASS);
```

函数 **pmu_to_sleepmode**

函数pmu_to_sleepmode描述见下表:

表 3-865. 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 3-866. 函数 pmu_to_deepsleepmode

函数名称	pmu_to_deepsleepmode
函数原型	void pmu_to_deepsleepmode(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}	
deepsleepmodecmd	进入深度睡眠模式命令
WFI_CMD	WFI命令
WFE_CMD	WFE命令
输出参数{out}	
-	-
返回值	
-	-

例如：

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

```
pmu_to_deepsleepmode(WFI_CMD);
```

函数 pmu_to_standbymode

函数pmu_to_standbymode描述见下表：

表 3-867. 函数 pmu_to_standbymode

函数名称	pmu_to_standbymode
函数原型	void pmu_to_standbymode(void);
功能描述	进入待机模式
先决条件	-
被调用函数	-
输入参数{in}	
-	-
输出参数{out}	

-	-
返回值	
-	-

例如：

```
/* PMU work at standby mode */
```

```
pmu_to_standby();
```

函数 pmu_wakeup_pin_enable

函数pmu_wakeup_pin_enable描述见下表：

表 3-868. 函数 pmu_wakeup_pin_enable

函数名称	pmu_wakeup_pin_enable
函数原型	void pmu_wakeup_pin_enable(uint32_t wakeup_pin);
功能描述	WKUP引脚唤醒使能
先决条件	-
被调用函数	-
输入参数{in}	
wakeup_pin	唤醒引脚
PMU_WAKEUP_PIN0	唤醒引脚0（PA0）使能
PMU_WAKEUP_PIN1	唤醒引脚1（PA2）使能
PMU_WAKEUP_PIN3	唤醒引脚3（PC13）使能
PMU_WAKEUP_PIN5	唤醒引脚5（PC1）使能
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* enable wakeup pin */
```

```
pmu_wakeup_pin_enable (PMU_WAKEUP_PIN0);
```

函数 pmu_wakeup_pin_disable

函数pmu_wakeup_pin_disable描述见下表：

表 3-869. 函数 pmu_wakeup_pin_disable

函数名称	pmu_wakeup_pin_disable
------	------------------------

函数原型	void pmu_wakeup_pin_disable(uint32_t wakeup_pin);
功能描述	WKUP引脚唤醒失能
先决条件	-
被调用函数	-
输入参数{in}	
wakeup_pin	唤醒引脚
PMU_WAKEUP_PIN0	唤醒引脚0（PA0）使能
PMU_WAKEUP_PIN1	唤醒引脚1（PA2）使能
PMU_WAKEUP_PIN3	唤醒引脚3（PC13）使能
PMU_WAKEUP_PIN5	唤醒引脚5（PC1）使能
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* disable wakeup pin */
```

```
pmu_wakeup_pin_disable (PMU_WAKEUP_PIN0);
```

函数 pmu_backup_write_enable

函数pmu_backup_write_enable描述见下表：

表 3-870. 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 3-871. 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_backup_voltage_stabilizer_enable

函数pmu_backup_voltage_stabilizer_enable描述见下表：

表 3-872. 函数 pmu_backup_voltage_stabilizer_enable

函数名称	pmu_backup_voltage_stabilizer_enable
函数原型	void pmu_backup_voltage_stabilizer_enable (void);
功能描述	使能备份域电压稳压器
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 backup voltage stabilizer */
```

```
pmu_backup_voltage_stabilizer_enable();
```

函数 pmu_backup_voltage_stabilizer_disable

函数pmu_backup_voltage_stabilizer_disable描述见下表:

表 3-873. 函数 pmu_backup_voltage_stabilizer_disable

函数名称	pmu_backup_voltage_stabilizer_disable
函数原型	void pmu_backup_voltage_stabilizer_disable (void);
功能描述	使能备份域电压稳压器
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 backup voltage stabilizer */
```

```
pmu_backup_voltage_stabilizer_disable();
```

函数 pmu_enter_deepsleep_wait_time_config

函数pmu_enter_deepsleep_wait_time_config描述见下表:

表 3-874. 函数 pmu_enter_deepsleep_wait_time_config

函数名称	pmu_enter_deepsleep_wait_time_config
函数原型	void pmu_enter_deepsleep_wait_time_config(int32_t wait_time);;
功能描述	配置进入Deep-sleep模式时IRC计数值
先决条件	-
被调用函数	-
输入参数{in}	
wait_time	进入Deep-sleep模式前等待的IRC计数值（0x00~0x1F）
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* configure IRC counter before enter Deep-sleep mode to 0x10 */
```

```
pmu_enter_deepsleep_wait_time_config(0x10);
```

函数 `pmu_exit_deepsleep_wait_time_config`

函数 `pmu_exit_deepsleep_wait_time_config` 描述见下表：

表 3-875. 函数 `pmu_exit_deepsleep_wait_time_config`

函数名称	<code>pmu_exit_deepsleep_wait_time_config</code>
函数原型	<code>void pmu_exit_deepsleep_wait_time_config(int32_t wait_time);</code>
功能描述	配置退出Deep-sleep模式时IRC计数值
先决条件	-
被调用函数	-
输入参数{in}	
<code>wait_time</code>	退出Deep-sleep模式前等待的IRC计数值（0x00~0xFFFF）
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* configure IRC counter before enter Deep-sleep mode to 0x10 */
```

```
pmu_enter_deepsleep_wait_time_config(0x10);
```

函数 `pmu_flag_get`

函数 `pmu_flag_get` 描述见下表：

表 3-876. 函数 `pmu_flag_get`

函数名称	<code>pmu_flag_get</code>
函数原型	<code>FlagStatus pmu_flag_get(uint32_t flag);</code>
功能描述	获取标志位
先决条件	-
被调用函数	-
输入参数{in}	
<code>flag</code>	标志位
<code>PMU_FLAG_WAKEUP</code>	唤醒标志
<code>PMU_FLAG_STANDBY</code>	待机标志
<code>PMU_FLAG_LVDF</code>	低电压标志
<code>PMU_FLAG_VAVDF</code>	V _{DDA} 模拟电压检测器标志
<code>PMU_FLAG_VOVD</code>	V _{DDA} 外设电压检测器标志
<code>PMU_FLAG_VBATL</code>	V _{BAT} 电压检测器低电压标志

<i>PMU_FLAG_VBAT</i> <i>HF</i>	V _{BAT} 电压检测器高电压标志
<i>PMU_FLAG_TEMP</i> <i>LF</i>	温度检测器低电压标志
<i>PMU_FLAG_TEMP</i> <i>HF</i>	温度检测器高电压标志
<i>PMU_FLAG_DVSR</i> <i>F</i>	降压稳压器就绪标志
<i>PMU_FLAG_USB33</i> <i>RF</i>	USB供电就绪标志
<i>PMU_FLAG_PWRR</i> <i>F</i>	电源供电就绪标志位
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 3-877. 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
<i>PMU_FLAG_WAKEUP</i>	清除唤醒标志
<i>PMU_FLAG_STANDBY</i>	清除待机标志
输出参数{out}	
-	-
返回值	
-	-

例如:

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

3.27. RAMECCMU

RAMECCMU 是 RAM ECC 监视器单元，它提供了一种方法来验证应用程序的 ECC 状态，并在发生错误时执行错误处理。时钟控制单元提供了一系列频率的时钟功能。章节 [3.27.1](#) 描述了 RAMECCMU 的寄存器列表，章节 [3.27.2](#) 对 RAMECCMU 库函数进行说明。

3.27.1. 外设寄存器描述

RAMECCMU 寄存器列表如下表所示:

表 3-878. RAMECCMU 寄存器

寄存器名称	寄存器描述
RAMECCMU_INT	RAMECCMU全局中断寄存器
RAMECCMU_MxCTL	RAMECCMU监视器x控制寄存器
RAMECCMU_MxSTATUS	RAMECCMU监视器x状态寄存器
RAMECCMU_MxFAADDR	RAMECCMU监视器x故障地址寄存器
RAMECCMU_MxFDL	RAMECCMU监视器x故障数据低位寄存器
RAMECCMU_MxFDH	RAMECCMU监视器x故障数据高位寄存器
RAMECCMU_MxFECODE	RAMECCMU监视器x故障ECC错误代码寄存器

3.27.2. 外设库函数说明

RAMECCMU 库函数列表如下表所示:

表 3-879. RAMECCMU 库函数

库函数名称	库函数描述
rameccmu_deinit	复位RAMECCMU单元
rameccmu_monitor_failing_address_get	获取RAMECCMU监视器ECC故障地址
rameccmu_monitor_failing_data_low_bits_get	获取RAMECCMU监视器ECC故障数据低32位
rameccmu_monitor_failing_data_high_bits_get	获取RAMECCMU监视器ECC故障数据高32位

库函数名称	库函数描述
rameccmu_monitor_failing_ecc_error_code_get	获取RAMECCMU监视器ECC故障错误代码
rameccmu_global_interrupt_enable	使能RAMECCMU全局ECC中断
rameccmu_global_interrupt_disable	禁能RAMECCMU全局ECC中断
rameccmu_monitor_interrupt_enable	使能RAMECCMU监视器ECC错误中断
rameccmu_monitor_interrupt_disable	禁能RAMECCMU监视器ECC错误中断
rameccmu_monitor_flag_get	获取RAMECCMU监视器ECC错误标志
rameccmu_monitor_flag_clear	清除RAMECCMU监视器ECC错误标志
rameccmu_monitor_interrupt_flag_get	获取RAMECCMU监视器ECC中断错误标志
rameccmu_monitor_interrupt_flag_clear	清除RAMECCMU监视器ECC中断错误标志

枚举类型 `rameccmu_monitor_enum`

表 3-880. 枚举类型 `rameccmu_monitor_enum`

枚举名称	枚举描述
RAMECCMU0_MONITOR0	RAMECCMU0监视器0
RAMECCMU0_MONITOR1	RAMECCMU0监视器1
RAMECCMU0_MONITOR2	RAMECCMU0监视器2
RAMECCMU0_MONITOR3	RAMECCMU0监视器3
RAMECCMU0_MONITOR4	RAMECCMU0监视器4
RAMECCMU1_MONITOR0	RAMECCMU1监视器0
RAMECCMU1_MONITOR1	RAMECCMU1监视器1
RAMECCMU1_MONITOR2	RAMECCMU1监视器2

函数 `rameccmu_deinit`

函数`rameccmu_deinit`描述见下表:

表 3-881. 函数 `rameccmu_deinit`

函数名称	<code>rameccmu_deinit</code>
函数原型	<code>void rameccmu_deinit(uint32_t rameccmu_periph);</code>
功能描述	复位RAMECCMU单元
先决条件	-
被调用函数	-
输入参数{in}	
<code>rameccmu_periph</code>	<code>rameccmu</code>
<code>RAMECCMU0</code>	Region 0的RAMECC监视器单元
<code>RAMECCMU1</code>	Region 1的RAMECC监视器单元
输出参数{out}	
-	-
返回值	

-	-
---	---

例如:

```
/* deinit RAMECCMU0 unit */
rameccmu_deinit(RAMECCMU0);
```

函数 `rameccmu_monitor_failing_address_get`

函数 `rameccmu_monitor_failing_address_get` 描述见下表:

表 3-882. 函数 `rameccmu_monitor_failing_address_get`

函数名称	<code>rameccmu_monitor_failing_address_get</code>
函数原型	<code>uint32_t rameccmu_monitor_failing_address_get(rameccmu_monitor_enum rameccmu_monitor);</code>
功能描述	获取RAMECCMU监视器ECC故障地址
先决条件	-
被调用函数	-
输入参数{in}	
<code>rameccmu_monitor</code>	RAMECCMU监视器
<code>RAMECCMU0_MONITOR0</code>	RAMECCMU0监视器0
<code>RAMECCMU0_MONITOR1</code>	RAMECCMU0监视器1
<code>RAMECCMU0_MONITOR2</code>	RAMECCMU0监视器2
<code>RAMECCMU0_MONITOR3</code>	RAMECCMU0监视器3
<code>RAMECCMU0_MONITOR4</code>	RAMECCMU0监视器4
<code>RAMECCMU1_MONITOR0</code>	RAMECCMU1监视器0
<code>RAMECCMU1_MONITOR1</code>	RAMECCMU1监视器1
<code>RAMECCMU1_MONITOR2</code>	RAMECCMU1监视器2
输出参数{out}	
-	-
返回值	
<code>uint32_t</code>	ECC故障地址

例如:

```
/* get RAMECCMU monitor ECC failing address */
```

```
uint32_t val;
```

```
val = rameccmu_monitor_failing_address_get(RAMECCMU0_MONITOR0);
```

函数 rameccmu_monitor_failing_data_low_bits_get

函数rameccmu_monitor_failing_data_low_bits_get描述见下表：

表 3-883. 函数 rameccmu_monitor_failing_data_low_bits_get

函数名称	rameccmu_monitor_failing_data_low_bits_get
函数原型	uint32_t rameccmu_monitor_failing_data_low_bits_get(rameccmu_monitor_enum rameccmu_monitor);
功能描述	获取RAMECCMU监视器ECC故障数据低32位
先决条件	-
被调用函数	-
输入参数{in}	
rameccmu_monitor	RAMECCMU监视器
RAMECCMU0_MONITOR0	RAMECCMU0监视器0
RAMECCMU0_MONITOR1	RAMECCMU0监视器1
RAMECCMU0_MONITOR2	RAMECCMU0监视器2
RAMECCMU0_MONITOR3	RAMECCMU0监视器3
RAMECCMU0_MONITOR4	RAMECCMU0监视器4
RAMECCMU1_MONITOR0	RAMECCMU1监视器0
RAMECCMU1_MONITOR1	RAMECCMU1监视器1
RAMECCMU1_MONITOR2	RAMECCMU1监视器2
输出参数{out}	
-	-
返回值	
uint32_t	ECC故障数据低32位

例如：

```
/* get RAMECCMU monitor ECC failing data low 32 bits */
```

```
uint32_t val_low;
```

```
val_low = rameccmu_monitor_failing_data_low_bits_get(RAMECCMU0_MONITOR0);
```

函数 rameccmu_monitor_failing_data_high_bits_get

函数rameccmu_monitor_failing_data_high_bits_get描述见下表：

表 3-884. 函数 rameccmu_monitor_failing_data_high_bits_get

函数名称	rameccmu_monitor_failing_data_high_bits_get
函数原型	uint32_t rameccmu_monitor_failing_data_high_bits_get(rameccmu_monitor_enum rameccmu_monitor);
功能描述	获取RAMECCMU监视器ECC故障数据高32位
先决条件	-
被调用函数	-
输入参数{in}	
rameccmu_monito r	RAMECCMU监视器
RAMECCMU0_MO NITOR0	RAMECCMU0监视器0
RAMECCMU0_MO NITOR1	RAMECCMU0监视器1
RAMECCMU0_MO NITOR2	RAMECCMU0监视器2
RAMECCMU0_MO NITOR3	RAMECCMU0监视器3
RAMECCMU0_MO NITOR4	RAMECCMU0监视器4
RAMECCMU1_MO NITOR0	RAMECCMU1监视器0
RAMECCMU1_MO NITOR1	RAMECCMU1监视器1
RAMECCMU1_MO NITOR2	RAMECCMU1监视器2
输出参数{out}	
-	-
返回值	
uint32_t	ECC故障数据高32位

例如：

```
/* get RAMECCMU monitor ECC failing data high 32 bits */
```

```
uint32_t val_high;
```

```
val_high = rameccmu_monitor_failing_data_high_bits_get(RAMECCMU0_MONITOR0);
```

函数 `rameccmu_monitor_failing_ecc_error_code_get`

函数 `rameccmu_monitor_failing_ecc_error_code_get` 描述见下表：

表 3-885. 函数 `rameccmu_monitor_failing_ecc_error_code_get`

函数名称	<code>rameccmu_monitor_failing_ecc_error_code_get</code>
函数原型	<pre>uint32_t rameccmu_monitor_failing_ecc_error_code_get(rameccmu_monitor_enum rameccmu_monitor);</pre>
功能描述	获取RAMECCMU监视器ECC故障错误代码
先决条件	-
被调用函数	-
输入参数{in}	
<code>rameccmu_monitor</code>	RAMECCMU监视器
<code>RAMECCMU0_MONITOR0</code>	RAMECCMU0监视器0
<code>RAMECCMU0_MONITOR1</code>	RAMECCMU0监视器1
<code>RAMECCMU0_MONITOR2</code>	RAMECCMU0监视器2
<code>RAMECCMU0_MONITOR3</code>	RAMECCMU0监视器3
<code>RAMECCMU0_MONITOR4</code>	RAMECCMU0监视器4
<code>RAMECCMU1_MONITOR0</code>	RAMECCMU1监视器0
<code>RAMECCMU1_MONITOR1</code>	RAMECCMU1监视器1
<code>RAMECCMU1_MONITOR2</code>	RAMECCMU1监视器2
输出参数{out}	
-	-
返回值	
<code>uint32_t</code>	ECC故障错误代码

例如：

```
/* get RAMECCMU monitor failing ECC error code */
uint32_t val;
val = rameccmu_monitor_failing_ecc_error_code_get(RAMECCMU0_MONITOR0);
```

函数 rameccmu_global_interrupt_enable

函数rameccmu_global_interrupt_enable描述见下表:

表 3-886. 函数 rameccmu_global_interrupt_enable

函数名称	rameccmu_global_interrupt_enable
函数原型	void rameccmu_global_interrupt_enable(uint32_t rameccmu_periph, uint32_t interrupt);
功能描述	使能RAMECCMU全局ECC中断
先决条件	-
被调用函数	-
输入参数{in}	
rameccmu_periph	rameccmu
RAMECCMU0	Region 0的RAMECC监视器单元
RAMECCMU1	Region 1的RAMECC监视器单元
输入参数{in}	
interrupt	全局ECC中断
RAMECCMU_INT_ECC_GLOBAL_ERROR	ECC全局错误中断
RAMECCMU_INT_ECC_SINGLE_ERROR	ECC单差错中断
RAMECCMU_INT_ECC_DOUBLE_ERROR	ECC双差错中断
RAMECCMU_INT_ECC_DOUBLE_ERROR_BYTE_WRITE	ECC双差错字节写中断
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* enable ECC global error interrupt */
rameccmu_global_interrupt_enable(RAMECCMU0,
RAMECCMU_INT_ECC_GLOBAL_ERROR);
```

函数 rameccmu_global_interrupt_disable

函数rameccmu_global_interrupt_disable描述见下表:

表 3-887. 函数 `rameccmu_global_interrupt_disable`

函数名称	<code>rameccmu_global_interrupt_disable</code>
函数原型	<code>void rameccmu_global_interrupt_disable(uint32_t rameccmu_periph, uint32_t interrupt);</code>
功能描述	禁能RAMECCMU全局ECC中断
先决条件	-
被调用函数	-
输入参数{in}	
<code>rameccmu_periph</code>	<code>rameccmu</code>
<code>RAMECCMU0</code>	Region 0的RAMECC监视器单元
<code>RAMECCMU1</code>	Region 1的RAMECC监视器单元
输入参数{in}	
<code>interrupt</code>	全局ECC中断
<code>RAMECCMU_INT_ECC_GLOBAL_ERROR</code>	ECC全局错误中断
<code>RAMECCMU_INT_ECC_SINGLE_ERROR</code>	ECC单差错中断
<code>RAMECCMU_INT_ECC_DOUBLE_ERROR</code>	ECC双差错中断
<code>RAMECCMU_INT_ECC_DOUBLE_ERROR_BYTE_WRITE</code>	ECC双差错字节写中断
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* disable ECC global error interrupt */
```

```
rameccmu_global_interrupt_disable(RAMECCMU0,  
RAMECCMU_INT_ECC_GLOBAL_ERROR);
```

函数 `rameccmu_monitor_interrupt_enable`

函数`rameccmu_monitor_interrupt_enable`描述见下表：

表 3-888. 函数 `rameccmu_monitor_interrupt_enable`

函数名称	<code>rameccmu_monitor_interrupt_enable</code>
函数原型	<code>void rameccmu_monitor_interrupt_enable(rameccmu_monitor_enum</code>

	rameccmu_monitor, uint32_t monitor_interrupt);
功能描述	使能RAMECCMU监视器ECC错误中断
先决条件	-
被调用函数	-
输入参数{in}	
rameccmu_monito r	RAMECCMU监视器
RAMECCMU0_MO NITOR0	RAMECCMU0监视器0
RAMECCMU0_MO NITOR1	RAMECCMU0监视器1
RAMECCMU0_MO NITOR2	RAMECCMU0监视器2
RAMECCMU0_MO NITOR3	RAMECCMU0监视器3
RAMECCMU0_MO NITOR4	RAMECCMU0监视器4
RAMECCMU1_MO NITOR0	RAMECCMU1监视器0
RAMECCMU1_MO NITOR1	RAMECCMU1监视器1
RAMECCMU1_MO NITOR2	RAMECCMU1监视器2
输入参数{in}	
monitor_interrupt	监视器中断
RAMECCMU_INT_ ECC_SINGLE_ERR OR	ECC单差错中断
RAMECCMU_INT_ ECC_DOUBLE_ER ROR	ECC双差错中断
RAMECCMU_INT_ ECC_DOUBLE_ER ROR_BYTE_WRIT E	ECC双差错字节写中断
RAMECCMU_INT_ ECC_ERROR_LAT CHING	ECC错误锁存
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* enable RAMECCMU monitor ECC error interrupt */  
  
rameccmu_monitor_interrupt_enable(RAMECCMU0_MONITOR0,  
RAMECCMU_INT_ECC_SINGLE_ERROR);
```

函数 `rameccmu_monitor_interrupt_disable`

函数 `rameccmu_monitor_interrupt_disable` 描述见下表:

表 3-889. 函数 `rameccmu_monitor_interrupt_disable`

函数名称	<code>rameccmu_monitor_interrupt_disable</code>
函数原型	<code>void rameccmu_monitor_interrupt_disable(rameccmu_monitor_enum rameccmu_monitor, uint32_t monitor_interrupt);</code>
功能描述	禁能RAMECCMU监视器ECC错误中断
先决条件	-
被调用函数	-
输入参数{in}	
<code>rameccmu_monitor</code>	RAMECCMU监视器
<code>RAMECCMU0_MONITOR0</code>	RAMECCMU0监视器0
<code>RAMECCMU0_MONITOR1</code>	RAMECCMU0监视器1
<code>RAMECCMU0_MONITOR2</code>	RAMECCMU0监视器2
<code>RAMECCMU0_MONITOR3</code>	RAMECCMU0监视器3
<code>RAMECCMU0_MONITOR4</code>	RAMECCMU0监视器4
<code>RAMECCMU1_MONITOR0</code>	RAMECCMU1监视器0
<code>RAMECCMU1_MONITOR1</code>	RAMECCMU1监视器1
<code>RAMECCMU1_MONITOR2</code>	RAMECCMU1监视器2
输入参数{in}	
<code>monitor_interrupt</code>	监视器中断
<code>RAMECCMU_INT_ECC_SINGLE_ERROR</code>	ECC单差错中断
<code>RAMECCMU_INT_ECC_DOUBLE_ERROR</code>	ECC双差错中断

RAMECCMU_INT_ECC_DOUBLE_ERROR_BYTE_WRITE	ECC双差错字节写中断
RAMECCMU_INT_ECC_ERROR_LATCHING	ECC错误锁存
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* disable RAMECCMU monitor ECC error interrupt */
```

```
rameccmu_monitor_interrupt_disable(RAMECCMU0_MONITOR0,
RAMECCMU_INT_ECC_SINGLE_ERROR);
```

函数 rameccmu_monitor_flag_get

函数rameccmu_monitor_flag_get描述见下表：

表 3-890. 函数 rameccmu_monitor_flag_get

函数名称	rameccmu_monitor_flag_get
函数原型	FlagStatus rameccmu_monitor_flag_get(rameccmu_monitor_enum rameccmu_monitor, uint32_t flag);
功能描述	获取RAMECCMU监视器ECC错误标志
先决条件	-
被调用函数	-
输入参数{in}	
rameccmu_monitor	RAMECCMU监视器
RAMECCMU0_MONITOR0	RAMECCMU0监视器0
RAMECCMU0_MONITOR1	RAMECCMU0监视器1
RAMECCMU0_MONITOR2	RAMECCMU0监视器2
RAMECCMU0_MONITOR3	RAMECCMU0监视器3
RAMECCMU0_MONITOR4	RAMECCMU0监视器4
RAMECCMU1_MONITOR0	RAMECCMU1监视器0

RAMECCMU1_MONITOR1	RAMECCMU1监视器1
RAMECCMU1_MONITOR2	RAMECCMU1监视器2
输入参数{in}	
flag	RAMECCMU监视器标志
RAMECCMU_FLAG_ECC_SINGLE_ERROR	ECC单差错检测和纠正标志
RAMECCMU_FLAG_ECC_DOUBLE_ERROR	ECC双差错检测标志
RAMECCMU_FLAG_ECC_DOUBLE_ERROR_BYTE_WRITE	字节写入时ECC双差错检测标志
输出参数{out}	
-	-
返回值	
FlagStatus	RESET或SET

例如:

```
/* get RAMECCMU monitor ECC error flag */
```

```
FlagStatus flag;
```

```
flag = rameccmu_monitor_flag_get(RAMECCMU0_MONITOR0,
RAMECCMU_FLAG_ECC_SINGLE_ERROR);
```

函数 rameccmu_monitor_flag_clear

函数rameccmu_monitor_flag_clear描述见下表:

表 3-891. 函数 rameccmu_monitor_flag_clear

函数名称	rameccmu_monitor_flag_clear
函数原型	void rameccmu_monitor_flag_clear(rameccmu_monitor_enum rameccmu_monitor, uint32_t flag);
功能描述	清除RAMECCMU监视器ECC错误标志
先决条件	-
被调用函数	-
输入参数{in}	
rameccmu_monitor	RAMECCMU监视器
RAMECCMU0_MONITOR0	RAMECCMU0监视器0

RAMECCMU0_MON ITOR1	RAMECCMU0监视器1
RAMECCMU0_MON ITOR2	RAMECCMU0监视器2
RAMECCMU0_MON ITOR3	RAMECCMU0监视器3
RAMECCMU0_MON ITOR4	RAMECCMU0监视器4
RAMECCMU1_MON ITOR0	RAMECCMU1监视器0
RAMECCMU1_MON ITOR1	RAMECCMU1监视器1
RAMECCMU1_MON ITOR2	RAMECCMU1监视器2
输入参数{in}	
flag	RAMECCMU监视器标志
RAMECCMU_FLAG _ECC_SINGLE_ER ROR	ECC单差错检测和纠正标志
RAMECCMU_FLAG _ECC_DOUBLE_E RROR	ECC双差错检测标志
RAMECCMU_FLAG _ECC_DOUBLE_E RROR_BYTE_WRI TE	字节写入时ECC双差错检测标志
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* clear RAMECCMU monitor ECC error flag */
```

```
rameccmu_monitor_flag_clear(RAMECCMU0_MONITOR0,  
RAMECCMU_FLAG_ECC_SINGLE_ERROR);
```

函数 rameccmu_monitor_interrupt_flag_get

函数rameccmu_monitor_interrupt_flag_get描述见下表：

表 3-892. 函数 rameccmu_monitor_interrupt_flag_get

函数名称	rameccmu_monitor_interrupt_flag_get
函数原型	FlagStatus rameccmu_monitor_interrupt_flag_get(rameccmu_monitor_enum

	rameccmu_monitor, uint32_t flag);
功能描述	获取RAMECCMU监视器ECC中断错误标志
先决条件	-
被调用函数	-
输入参数{in}	
rameccmu_monitor	RAMECCMU监视器
RAMECCMU0_MONITOR0	RAMECCMU0监视器0
RAMECCMU0_MONITOR1	RAMECCMU0监视器1
RAMECCMU0_MONITOR2	RAMECCMU0监视器2
RAMECCMU0_MONITOR3	RAMECCMU0监视器3
RAMECCMU0_MONITOR4	RAMECCMU0监视器4
RAMECCMU1_MONITOR0	RAMECCMU1监视器0
RAMECCMU1_MONITOR1	RAMECCMU1监视器1
RAMECCMU1_MONITOR2	RAMECCMU1监视器2
输入参数{in}	
flag	RAMECCMU监视器标志
RAMECCMU_INTERRUPT_FLAG_ECC_SINGLE_ERROR	ECC单差错检测和纠正标志
RAMECCMU_INTERRUPT_FLAG_ECC_DOUBLE_ERROR	ECC双差错检测标志
RAMECCMU_INTERRUPT_FLAG_ECC_DOUBLE_ERROR_BYTE_WRITE	字节写入时ECC双差错检测标志
输出参数{out}	
-	-
返回值	
FlagStatus	RESET或SET

例如：

```
/* get RAMECCMU monitor ECC error flag */
```

FlagStatus flag;

```
flag = rameccmu_monitor_interrupt_flag_get(RAMECCMU0_MONITOR0,
RAMECCMU_INT_FLAG_ECC_SINGLE_ERROR);
```

函数 rameccmu_monitor_interrupt_flag_clear

函数rameccmu_monitor_interrupt_flag_clear描述见下表:

表 3-893. 函数 rameccmu_monitor_interrupt_flag_clear

函数名称	rameccmu_monitor_interrupt_flag_clear
函数原型	void rameccmu_monitor_interrupt_flag_clear(rameccmu_monitor_enum rameccmu_monitor, uint32_t flag);
功能描述	清除RAMECCMU监视器ECC中断错误标志
先决条件	-
被调用函数	-
输入参数{in}	
rameccmu_monitor	RAMECCMU监视器
RAMECCMU0_MONITOR0	RAMECCMU0监视器0
RAMECCMU0_MONITOR1	RAMECCMU0监视器1
RAMECCMU0_MONITOR2	RAMECCMU0监视器2
RAMECCMU0_MONITOR3	RAMECCMU0监视器3
RAMECCMU0_MONITOR4	RAMECCMU0监视器4
RAMECCMU1_MONITOR0	RAMECCMU1监视器0
RAMECCMU1_MONITOR1	RAMECCMU1监视器1
RAMECCMU1_MONITOR2	RAMECCMU1监视器2
输入参数{in}	
flag	RAMECCMU监视器标志
RAMECCMU_INT_FLAG_ECC_SINGLE_ERROR	ECC单差错检测和纠正标志
RAMECCMU_INT_FLAG_ECC_DOUBLE_ERROR	ECC双差错检测标志
RAMECCMU_INT_FLAG_ECC_SINGLE_ERROR	字节写入时ECC双差错检测标志

FLAG_ECC_DOUBLE_ERROR_BYTE_WRITE	
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* clear RAMECCMU monitor ECC error flag */
rameccmu_monitor_flag_clear(RAMECCMU0_MONITOR0,
RAMECCMU_INT_FLAG_ECC_SINGLE_ERROR);
```

3.28. 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 [3.28.1](#) 描述了 RCU 的寄存器列表，章节 [3.28.2](#) 对 RCU 库函数进行说明。

3.28.1. 外设寄存器说明

RCU寄存器列表如下表所示：

表 3-894. RCU 寄存器

寄存器名称	寄存器描述
RCU_CTL	控制寄存器
RCU_PLL0	PLL0寄存器
RCU_CFG0	配置寄存器0
RCU_INT	中断寄存器
RCU_AHB1RST	AHB1复位寄存器
RCU_AHB2RST	AHB2复位寄存器
RCU_AHB3RST	AHB3复位寄存器
RCU_AHB4RST	AHB4复位寄存器
RCU_APB1RST	APB1复位寄存器
RCU_APB2RST	APB2复位寄存器
RCU_APB3RST	APB3复位寄存器
RCU_APB4RST	APB4复位寄存器
RCU_AHB1EN	AHB1使能寄存器
RCU_AHB2EN	AHB2使能寄存器
RCU_AHB3EN	AHB3使能寄存器
RCU_AHB4EN	AHB4使能寄存器
RCU_APB1EN	APB1使能寄存器

寄存器名称	寄存器描述
RCU_APB2EN	APB2使能寄存器
RCU_APB3EN	APB3使能寄存器
RCU_APB4EN	APB4使能寄存器
RCU_AHB1SPEN	AHB1睡眠模式使能寄存器
RCU_AHB2SPEN	AHB2睡眠模式使能寄存器
RCU_AHB3SPEN	AHB3睡眠模式使能寄存器
RCU_AHB4SPEN	AHB4睡眠模式使能寄存器
RCU_APB1SPEN	APB1睡眠模式使能寄存器
RCU_APB2SPEN	APB2睡眠模式使能寄存器
RCU_APB3SPEN	APB3睡眠模式使能寄存器
RCU_APB4SPEN	APB4睡眠模式使能寄存器
RCU_BDCTL	备份域控制寄存器
RCU_RSTSCK	复位源/时钟寄存器
RCU_PLLADDCTL	PLL时钟附加控制寄存器
RCU_PLL1	PLL1寄存器
RCU_PLL2	PLL2寄存器
RCU_CFG1	配置寄存器1
RCU_CFG2	配置寄存器2
RCU_CFG3	配置寄存器3
RCU_PLLALL	PLL控制寄存器
RCU_PLL0FRA	PLL0小数配置寄存器
RCU_PLL1FRA	PLL1小数配置寄存器
RCU_PLL2FRA	PLL2小数配置寄存器
RCU_ADDCTL0	附加时钟控制寄存器0
RCU_ADDCTL1	附加时钟控制寄存器1
RCU_ADDINT	附加时钟中断寄存器
RCU_CFG4	时钟配置寄存器4
RCU_USBCLKCTL	USB时钟控制寄存器
RCU_PLLUSBCFG	PLLUSB时钟配置寄存器
RCU_ADDAPB2RS T	APB2附加复位寄存器
RCU_ADDAPB2EN	APB2附加使能寄存器
RCU_ADDAPB2SP EN	APB2附加睡眠模式使能寄存器
RCU_CFG5	时钟配置寄存器5

3.28.2. 外设库函数说明

RCU库函数列表如下表所示：

表 3-895. RCU 库函数

库函数名称	库函数描述
rcu_deinit	复位RCU，将RCU寄存器复位为初始值
rcu_periph_clock_enable	使能外设时钟
rcu_periph_clock_disable	禁能外设时钟
rcu_periph_clock_sleep_enable	在睡眠模式下，使能外设时钟
rcu_periph_clock_sleep_disable	在睡眠模式下，禁能外设时钟
rcu_periph_reset_enable	外设时钟复位使能
rcu_periph_reset_disable	外设时钟复位禁能
rcu_bkp_reset_enable	备份域时钟复位使能
rcu_bkp_reset_disable	备份域时钟复位禁能
rcu_system_clock_source_config	配置选择系统时钟源
rcu_system_clock_source_get	获取系统时钟源选择状态
rcu_ahb_clock_config	配置AHB时钟预分频选择
rcu_apb1_clock_config	配置APB1时钟预分频选择
rcu_apb2_clock_config	配置APB2时钟预分频选择
rcu_apb3_clock_config	配置APB3时钟预分频选择
rcu_apb4_clock_config	配置APB4时钟预分频选择
rcu_ckout0_config	配置CKOUT0时钟源选择及分频系数
rcu_ckout1_config	配置CKOUT1时钟源选择及分频系数
rcu_pll_input_output_clock_range_config	配置PLL输入/输出时钟范围
rcu_pll_fractional_config	配置PLL VCO倍频因子小数部分
rcu_pll_fractional_latch_enable	使能PLL小数锁存功能
rcu_pll_fractional_latch_disable	禁能PLL小数锁存功能
rcu_pll_source_config	配置主PLL时钟源选择
rcu_pll0_config	配置PLL0时钟
rcu_pll1_config	配置PLL1时钟
rcu_pll2_config	配置PLL2时钟
rcu_pll_clock_output_enable	使能PLL P/Q/R时钟输出
rcu_pll_clock_output_disable	使能PLL P/Q/R时钟输出
rcu_pllusb0_config	配置PLLUSBHS0时钟
rcu_pllusb1_config	配置PLLUSBHS1时钟
rcu_rtc_clock_config	配置RTC时钟
rcu_rtc_div_config	当选择HXTAL作为RTC时钟源时，配置RTC时钟分频
rcu_ck48m_clock_config	配置CK48M时钟源选择
rcu_pll48m_clock_config	配置PLL48M时钟源选择
rcu_irc64mdiv_clock_config	配置IRC64M时钟分频选择
rcu_irc64mdiv_freq_get	获取IRC64MDIV时钟
rcu_timer_clock_prescaler_config	配置TIMER时钟预分频
rcu_spi_clock_config	配置SPI时钟源时钟
rcu_deepsleep_wakeup_sys_clock_c	配置深度睡眠模式下系统唤醒时钟源选择

库函数名称	库函数描述
onfig	
rcu_usart_clock_config	配置USART时钟源时钟
rcu_i2c_clock_config	配置I2C时钟源时钟
rcu_can_clock_config	配置CAN时钟源时钟
rcu_adc_clock_config	配置ADC时钟源时钟
rcu_exmc_clock_config	配置EXMC时钟源时钟
rcu_hpdc_clock_config	配置HPDF时钟源时钟
rcu_per_clock_config	配置外设时钟源时钟
rcu_usbhs_pll1qpsc_config	配置PLL1Q预分频时钟源时钟
rcu_usb48m_clock_config	配置USBHS48M时钟源选择
rcu_usbhs_clock_config	配置USBHS时钟源选择
rcu_usbhs_clock_selection_enable	使能USBHS时钟源选择
rcu_usbhs_clock_selection_disable	禁能USBHS时钟源选择
rcu_lxtal_drive_capability_config	配置LXTAL的驱动力
rcu_osci_stab_wait	等待振荡器稳定标志位置位或振荡器起振超时
rcu_osci_on	打开振荡器
rcu_osci_off	关闭振荡器
rcu_osci_bypass_mode_enable	使能时钟旁路模式
rcu_osci_bypass_mode_disable	禁能时钟旁路模式
rcu_irc64m_adjust_value_set	设置内部64MHz RC振荡器时钟调整值
rcu_lpirc4m_adjust_value_set	设置内部低功耗4MHz RC振荡器时钟调整值
rcu_hxtal_clock_monitor_enable	HXTAL时钟监视器使能
rcu_hxtal_clock_monitor_disable	HXTAL时钟监视器禁能
rcu_lxtal_clock_monitor_enable	LXTAL时钟监视器使能
rcu_lxtal_clock_monitor_disable	LXTAL时钟监视器禁能
rcu_clock_freq_get	获取系统、总线或外设时钟频率
rcu_flag_get	获取时钟稳定状态和外设复位标志
rcu_all_reset_flag_clear	清除复位标志
rcu_interrupt_enable	时钟稳定中断使能
rcu_interrupt_disable	时钟稳定中断禁能
rcu_interrupt_flag_get	获取时钟稳定中断和CKM中断标志
rcu_interrupt_flag_clear	清除中断标志

枚举类型 rcu_periph_enum

表 3-896. 枚举类型 rcu_periph_enum

成员名称	功能描述
RCU_USBHS0	USBHS0时钟
RCU_USBHS0ULPI	USBHS0 ULPI时钟
RCU_DMA0	DMA0时钟
RCU_DMA1	DMA1时钟

成员名称	功能描述
RCU_DMAMUX	IPA时钟
RCU_USBHS1	USBHS1时钟
RCU_USBHS1ULPI	USBHS1 ULPI时钟
RCU_FAC	FAC时钟
RCU_TRNG	TRNG时钟
RCU_TMU	TMU时钟
RCU_RAMECCMU1	RAMECCMU1时钟
RCU_EXMC	EXMC时钟
RCU_MDMA	MDMMA时钟
RCU_OSPIM	OSPIM时钟
RCU_OSPI0	OSPI0时钟
RCU_OSPI1	OSPI1时钟
RCU_RTDEC0	RTDEC0时钟
RCU_RTDEC1	RTDEC1时钟
RCU_RAMECCMU0	RAMECCMU0时钟
RCU_CPU	CPU时钟
RCU_GPIOA	GPIOA时钟
RCU_GPIOB	GPIOB时钟
RCU_GPIOC	GPIOC时钟
RCU_GPIOD	GPIOD时钟
RCU_GPIOE	GPIOE时钟
RCU_GPIOF	GPIOF时钟
RCU_GPIOG	GPIOG时钟
RCU_GPIOH	GPIOH时钟
RCU_GPIOJ	GPIOJ时钟
RCU_GPIOK	GPIOK时钟
RCU_BKPSRAM	BKPSRAM时钟
RCU_CRC	CRC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_TIMER22	TIMER22时钟
RCU_TIMER23	TIMER23时钟
RCU_TIMER50	TIMER50时钟
RCU_TIMER51	TIMER51时钟
RCU_RSPDIF	RSPDIF时钟
RCU_SPI1	SPI1时钟
RCU_SPI2	SPI2时钟

成员名称	功能描述
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_I2C2	I2C2时钟
RCU_I2C3	I2C3时钟
RCU_CTC	CTC时钟
RCU_DACHOLD	DACHOLD时钟
RCU_DAC	DAC时钟
RCU_UART6	UART6时钟
RCU_UART7	UART7时钟
RCU_TIMER0	TIMER0时钟
RCU_TIMER7	TIMER7时钟
RCU_USART0	USART0时钟
RCU_USART5	USART5时钟
RCU_ADC0	ADC0时钟
RCU_ADC1	ADC1时钟
RCU_ADC2	ADC2时钟
RCU_SPI0	SPI0时钟
RCU_SPI3	SPI3时钟
RCU_TIMER14	TIMER14时钟
RCU_TIMER15	TIMER15时钟
RCU_TIMER16	TIMER16时钟
RCU_SPI4	SPI4时钟
RCU_SPI5	SPI5时钟
RCU_TIMER40	TIMER40时钟
RCU_TIMER41	TIMER41时钟
RCU_TIMER42	TIMER42时钟
RCU_TIMER43	TIMER43时钟
RCU_TIMER44	TIMER44时钟
RCU_EDOUT	EDOUT时钟
RCU_TRIGSEL	TRIGSEL时钟
RCU_WWDGT	WWDGT时钟
RCU_SYSCFG	SYSCFG时钟
RCU_CMP	CMP时钟
RCU_VREF	VREF时钟
RCU_LPDTs	LPDTs时钟
RCU_PMU	PMU时钟
RCU_RTC	RTC时钟

成员名称	功能描述
RCU_CAN0	CAN0时钟
RCU_CAN1	CAN1时钟
RCU_CAN2	CAN2时钟

枚举类型 `rcu_periph_sleep_enum`

表 3-897. 枚举类型 `rcu_periph_sleep_enum`

成员名称	功能描述
RCU_USBHS0_SLP	USBHS0时钟
RCU_USBHS0ULPI_SLP	USBHS0ULPI时钟
RCU_SRAM0_SLP	SRAM0时钟
RCU_SRAM1_SLP	SRAM1时钟
RCU_DMA0_SLP	DMA0时钟
RCU_DMA1_SLP	DMA1时钟
RCU_DMAMUX_SLP	DMAMUX时钟
RCU_USBHS1_SLP	USBHS1时钟
RCU_USBHS1ULPI_SLP	USBHS1ULPI时钟
RCU_FAC_SLP	FAC时钟
RCU_TRNG_SLP	TRNG时钟
RCU_TMU_SLP	TMU时钟
RCU_RAMECCMU1_SLP	RAMECCMU1时钟
RCU_EXMC_SLP	EXMC时钟
RCU_MDMA_SLP	MDMMA时钟
RCU_OSPIM_SLP	OSPIM时钟
RCU_OSPI0_SLP	OSPI0时钟
RCU_OSPI1_SLP	OSPI1时钟
RCU_RTDEC0_SLP	RTDEC0时钟
RCU_RTDEC1_SLP	RTDEC1时钟
RCU_RAMECCMU0_SLP	RAMECCMU0时钟
RCU_AXISRAM_SLP	AXISRAM时钟
RCU_FMC_SLP	FMC时钟
RCU_GPIOA_SLP	GPIOA时钟
RCU_GPIOB_SLP	GPIOB时钟
RCU_GPIOC_SLP	GPIOC时钟
RCU_GPIOD_SLP	GPIOD时钟
RCU_GPIOE_SLP	GPIOE时钟
RCU_GPIOF_SLP	GPIOF时钟
RCU_GPIOG_SLP	GPIOG时钟
RCU_GPIOH_SLP	GPIOH时钟
RCU_GPIOJ_SLP	GPIOJ时钟
RCU_GPIOK_SLP	GPIOK时钟

成员名称	功能描述
RCU_BKPSRAM_SLP	BKPSRAM时钟
RCU_CRC_SLP	CRC时钟
RCU_TIMER1_SLP	TIMER1时钟
RCU_TIMER2_SLP	TIMER2时钟
RCU_TIMER3_SLP	TIMER3时钟
RCU_TIMER4_SLP	TIMER4时钟
RCU_TIMER5_SLP	TIMER5时钟
RCU_TIMER6_SLP	TIMER6时钟
RCU_TIMER22_SLP	TIMER22时钟
RCU_TIMER23_SLP	TIMER23时钟
RCU_TIMER50_SLP	TIMER50时钟
RCU_TIMER51_SLP	TIMER51时钟
RCU_SPI1_SLP	SPI1时钟
RCU_SPI2_SLP	SPI2时钟
RCU_USART1_SLP	USART1时钟
RCU_USART2_SLP	USART2时钟
RCU_UART3_SLP	UART3时钟
RCU_UART4_SLP	UART4时钟
RCU_I2C0_SLP	I2C0时钟
RCU_I2C1_SLP	I2C1时钟
RCU_I2C2_SLP	I2C2时钟
RCU_I2C3_SLP	I2C3时钟
RCU_CTC_SLP	CTC时钟
RCU_DACHOLD_SLP	DACHOLD时钟
RCU_DAC_SLP	DAC时钟
RCU_UART6_SLP	UART6时钟
RCU_UART7_SLP	UART7时钟
RCU_TIMER0_SLP	TIMER0时钟
RCU_TIMER7_SLP	TIMER7时钟
RCU_USART0_SLP	USART0时钟
RCU_USART5_SLP	USART5时钟
RCU_ADC0_SLP	ADC0时钟
RCU_ADC1_SLP	ADC1时钟
RCU_ADC2_SLP	ADC2时钟
RCU_SPI0_SLP	SPI0时钟
RCU_SPI3_SLP	SPI3时钟
RCU_TIMER14_SLP	TIMER14时钟
RCU_TIMER15_SLP	TIMER15时钟
RCU_TIMER16_SLP	TIMER16时钟
RCU_SPI4_SLP	SPI4时钟
RCU_SPI5_SLP	SPI5时钟

成员名称	功能描述
RCU_SAI2_SLP	SAI2时钟
RCU_TIMER40_SLP	TIMER40时钟
RCU_TIMER41_SLP	TIMER41时钟
RCU_TIMER42_SLP	TIMER42时钟
RCU_TIMER43_SLP	TIMER43时钟
RCU_TIMER44_SLP	TIMER44时钟
RCU_EDOUT_SLP	EDOUT时钟
RCU_TRIGSEL_SLP	TRIGSEL时钟
RCU_WWDGT_SLP	WWDGT时钟
RCU_SYSCFG_SLP	SYSCFG时钟
RCU_CMP_SLP	CMP时钟
RCU_VREF_SLP	VREF时钟
RCU_LPDTS_SLP	LPDTS时钟
RCU_PMU_SLP	PMU时钟
RCU_CAN0_SLP	CAN0时钟
RCU_CAN1_SLP	CAN1时钟
RCU_CAN2_SLP	CAN2时钟

枚举类型 `rcu_periph_reset_enum`

表 3-898. 枚举类型 `rcu_periph_reset_enum`

成员名称	功能描述
RCU_USBHS0RST	复位USBHS0时钟
RCU_DMA0RST	复位DMA0时钟
RCU_DMA1RST	复位DMA1时钟
RCU_DMAMUXRST	复位DMAMUX时钟
RCU_USBHS1RST	复位USBHS1HS时钟
RCU_DCIRST	复位DCI时钟
RCU_FACRST	复位FAC时钟
RCU_TRNGRST	复位TRNG时钟
RCU_TMURST	复位TMU时钟
RCU_EXMCRST	复位EXMC时钟
RCU_MDMARST	复位MDMMA时钟
RCU_OSPIMRST	复位OSPIM时钟
RCU_OSPI0RST	复位OSPI0时钟
RCU_OSPI1RST	复位OSPI1时钟
RCU_RTDEC0RST	复位RTDEC0时钟
RCU_RTDEC1RST	复位RTDEC1时钟
RCU_GPIOARST	复位GPIOA时钟
RCU_GPIOBRST	复位GPIOB时钟
RCU_GPIOCRST	复位GPIOC时钟

成员名称	功能描述
RCU_GPIODRST	复位GPIOD时钟
RCU_GPIOERST	复位GPIOE时钟
RCU_GPIOFRST	复位GPIOF时钟
RCU_GPIOGRST	复位GPIOG时钟
RCU_GPIOHRST	复位GPIOH时钟
RCU_GPIOJRST	复位GPIOJ时钟
RCU_GPIOKRST	复位GPIOK时钟
RCU_CRCRST	复位CRC时钟
RCU_TIMER1RST	复位TIMER1时钟
RCU_TIMER2RST	复位TIMER2时钟
RCU_TIMER3RST	复位TIMER3时钟
RCU_TIMER4RST	复位TIMER4时钟
RCU_TIMER5RST	复位TIMER5时钟
RCU_TIMER6RST	复位TIMER6时钟
RCU_TIMER22RST	复位TIMER22时钟
RCU_TIMER23RST	复位TIMER23时钟
RCU_TIMER50RST	复位TIMER50时钟
RCU_TIMER51RST	复位TIMER51时钟
RCU_SPI1RST	复位SPI1时钟
RCU_SPI2RST	复位SPI2时钟
RCU_USART1RST	复位USART1时钟
RCU_USART2RST	复位USART2时钟
RCU_UART3RST	复位UART3时钟
RCU_UART4RST	复位UART4时钟
RCU_I2C0RST	复位I2C0时钟
RCU_I2C1RST	复位I2C1时钟
RCU_I2C2RST	复位I2C2时钟
RCU_I2C3RST	复位I2C3时钟
RCU_CTCRST	复位CTC时钟
RCU_DACHOLDRST	复位DACHOLD时钟
RCU_DACRST	复位DAC时钟
RCU_UART6RST	复位UART6时钟
RCU_UART7RST	复位UART7时钟
RCU_TIMER0RST	复位TIMER0时钟
RCU_TIMER7RST	复位TIMER7时钟
RCU_USART0RST	复位USART0时钟
RCU_USART5RST	复位USART5时钟
RCU_ADC0RST	复位ADC0时钟
RCU_ADC1RST	复位ADC1时钟
RCU_ADC2RST	复位ADC2时钟
RCU_SPI0RST	复位SPI0时钟

成员名称	功能描述
RCU_SPI3RST	复位SPI3时钟
RCU_TIMER14RST	复位TIMER14时钟
RCU_TIMER15RST	复位TIMER15时钟
RCU_TIMER16RST	复位TIMER16时钟
RCU_HPDRST	复位HPDF时钟
RCU_SPI4RST	复位SPI4时钟
RCU_SPI5RST	复位SPI5时钟
RCU_TIMER40RST	复位TIMER40时钟
RCU_TIMER41RST	复位TIMER41时钟
RCU_TIMER42RST	复位TIMER42时钟
RCU_TIMER43RST	复位TIMER43时钟
RCU_TIMER44RST	复位TIMER44时钟
RCU_EDOUTRST	复位EDOUT时钟
RCU_TRIGSELRST	复位TRIGSEL时钟
RCU_WWDGTRST	复位WWDGT时钟
RCU_SYSCFGRST	复位SYSCFG时钟
RCU_CMPRST	复位CMP时钟
RCU_VREFRST	复位VREF时钟
RCU_LPDTSRST	复位LPDTS时钟
RCU_PMURST	复位PMU时钟
RCU_CAN0RST	复位CAN0时钟
RCU_CAN1RST	复位CAN1时钟
RCU_CAN2RST	复位CAN2时钟

枚举类型 `rcu_flag_enum`

表 3-899. 枚举类型 `rcu_flag_enum`

成员名称	功能描述
RCU_FLAG_IRC64MSTB	IRC64M稳定标志
RCU_FLAG_HXTALSTB	HXTAL稳定标志
RCU_FLAG_PLL0STB	PLL0稳定标志
RCU_FLAG_PLL1STB	PLL1稳定标志
RCU_FLAG_PLL2STB	PLL2稳定标志
RCU_FLAG_LXTALSTB	LXTAL稳定标志
RCU_FLAG_IRC32KSTB	IRC32K稳定标志
RCU_FLAG_IRC48MSTB	IRC48M稳定标志
RCU_FLAG_LPIRC4MSTB	LPIRC4M稳定标志
RCU_FLAG_PLLUSBHS0STB	PLLUSBHS0稳定标志
RCU_FLAG_PLLUSBHS1STB	PLLUSBHS1稳定标志
RCU_FLAG_LCKMD	LXTAL时钟故障检测标志
RCU_FLAG_BORRST	欠压复位标志

成员名称	功能描述
RCU_FLAG_EPRST	外部引脚复位标志
RCU_FLAG_PORRST	电源复位标志
RCU_FLAG_SWRST	软件复位标志
RCU_FLAG_FWDGTRST	独立看门狗复位标志
RCU_FLAG_WWDGTRST	窗口看门狗复位标志
RCU_FLAG_LPRST	低功耗复位标志

枚举类型 rcu_int_flag_enum

表 3-900. 枚举类型 rcu_int_flag_enum

成员名称	功能描述
RCU_INT_FLAG_IRC32KSTB	IRC32K时钟稳定中断标志
RCU_INT_FLAG_LXTALSTB	LXTAL时钟稳定中断标志
RCU_INT_FLAG_IRC64MSTB	IRC64M时钟稳定中断标志
RCU_INT_FLAG_HXTALSTB	HXTAL时钟稳定中断标志
RCU_INT_FLAG_PLL0STB	PLL0时钟稳定中断标志
RCU_INT_FLAG_PLL1STB	PLL1时钟稳定中断标志
RCU_INT_FLAG_PLL2STB	PLL2时钟稳定中断标志
RCU_INT_FLAG_CKM	外部高速晶振时钟监视器中断标志
RCU_INT_FLAG_LCKM	外部低速晶振时钟监视器中断标志
RCU_INT_FLAG_LPIRC4MSTB	LPIRC4M时钟稳定中断标志
RCU_INT_FLAG_IRC48MSTB	IRC48M时钟稳定中断标志
RCU_INT_FLAG_PLLUSBHS0STB	PLLUSBHS0时钟稳定中断标志
RCU_INT_FLAG_PLLUSBHS1STB	PLLUSBHS1时钟稳定中断标志

枚举类型 rcu_int_flag_clear_enum

表 3-901. 枚举类型 rcu_int_flag_clear_enum

成员名称	功能描述
RCU_INT_FLAG_IRC32KSTB_CLR	IRC32K时钟稳定中断清除标志
RCU_INT_FLAG_LXTALSTB_CLR	外部低速晶振时钟稳定中断清除标志
RCU_INT_FLAG_IRC64MSTB_CLR	IRC64M时钟稳定中断清除标志
RCU_INT_FLAG_HXTALSTB_CLR	外部高速晶振时钟稳定中断清除标志
RCU_INT_FLAG_PLL0STB_CLR	PLL0时钟稳定中断清除标志
RCU_INT_FLAG_PLL1STB_CLR	PLL1时钟稳定中断清除标志
RCU_INT_FLAG_PLL2STB_CLR	PLL2时钟稳定中断清除标志

成员名称	功能描述
RCU_INT_FLAG_CKM_CLR	外部高速晶振时钟监视器中断清除标志
RCU_INT_FLAG_LCKM_CLR	外部低速晶振时钟监视器中断清除标志
RCU_INT_FLAG_LPIRC4MSTB_CLR	LPIRC4M 时钟稳定中断清除标志
RCU_INT_FLAG_IRC48MSTB_CLR	IRC48M时钟稳定中断清除标志
RCU_INT_FLAG_PLLUSBHS0STB_CLR	PLLUSBHS0时钟稳定中断清除标志
RCU_INT_FLAG_PLLUSBHS1STB_CLR	PLLUSBHS1时钟稳定中断清除标志

枚举类型 rcu_int_enum

表 3-902. 枚举类型 rcu_int_enum

成员名称	功能描述
RCU_INT_IRC32KSTB	IRC32K时钟稳定中断
RCU_INT_LXTALSTB	外部低速晶振时钟稳定中断
RCU_INT_IRC64MSTB	IRC64M时钟稳定中断
RCU_INT_HXTALSTB	外部高速晶振时钟稳定中断
RCU_INT_PLL0STB	PLL0时钟稳定中断
RCU_INT_PLL1STB	PLL1时钟稳定中断
RCU_INT_PLL2STB	PLL2时钟稳定中断
RCU_INT_IRC48MSTB	IRC48M时钟稳定中断
RCU_INT_LPIRC4MSTB	LPIRC4M 时钟稳定中断
RCU_INT_PLLUSBHS0STB	PLLUSBHS0时钟稳定中断
RCU_INT_PLLUSBHS1STB	PLLUSBHS1时钟稳定中断

枚举类型 rcu_osci_type_enum

表 3-903. 枚举类型 rcu_osci_type_enum

成员名称	功能描述
RCU_HXTAL	外部高速振荡器
RCU_LXTAL	外部低速振荡器
RCU_IRC64M	IRC64M振荡器
RCU_IRC48M	IRC48M振荡器
RCU_IRC32K	IRC32K振荡器
RCU_LPIRC4M	LPIRC4M振荡器
RCU_PLL0_CK	锁相环PLL0时钟
RCU_PLL1_CK	锁相环PLL1时钟
RCU_PLL2_CK	锁相环PLL2时钟
RCU_PLLUSBHS0_CK	锁相环PLLUSBHS0时钟
RCU_PLLUSBHS1_CK	锁相环PLLUSBHS1时钟

枚举类型 `rcu_clock_freq_enum`

表 3-904. 枚举类型 `rcu_clock_freq_enum`

成员名称	功能描述
CK_SYS	系统时钟
CK_AHB	AHB时钟
CK_APB1	APB1时钟
CK_APB2	APB2时钟
CK_APB3	APB3时钟
CK_APB4	APB4时钟
CK_PLL0P	PLL0P时钟
CK_PLL0Q	PLL0Q时钟
CK_PLL0R	PLL0R时钟
CK_PLL1P	PLL1P时钟
CK_PLL1Q	PLL1Q时钟
CK_PLL1R	PLL1R时钟
CK_PLL2P	PLL2P时钟
CK_PLL2Q	PLL2Q时钟
CK_PLL2R	PLL2R时钟
CK_PER	PER时钟
CK_USART0	USART0时钟
CK_USART1	USART1时钟
CK_USART2	USART2时钟
CK_USART5	USART5时钟
CK_IRC64MDIV	IRC64MDIV时钟
CK_HXTAL	HXTAL时钟
CK_LPIRC4M	LPIRC4M时钟

枚举类型 `usart_idx_enum`

表 3-905. 枚举类型 `usart_idx_enum`

成员名称	功能描述
IDX_USART0	USART0索引
IDX_USART1	USART1索引
IDX_USART2	USART2索引
IDX_USART5	USART5索引

枚举类型 `i2c_idx_enum`

表 3-906. 枚举类型 `i2c_idx_enum`

成员名称	功能描述
IDX_I2C0	I2C0索引
IDX_I2C1	I2C1索引

成员名称	功能描述
IDX_I2C2	I2C2索引
IDX_I2C3	I2C3索引

枚举类型 `can_idx_enum`

表 3-907. 枚举类型 `can_idx_enum`

成员名称	功能描述
IDX_CAN0	CAN0索引
IDX_CAN1	CAN1索引
IDX_CAN2	CAN2索引

枚举类型 `adc_idx_enum`

表 3-908. 枚举类型 `adc_idx_enum`

成员名称	功能描述
IDX_ADC0	ADC0索引
IDX_ADC1	ADC1索引
IDX_ADC2	ADC2索引

枚举类型 `usbhs_idx_enum`

表 3-909. 枚举类型 `usbhs_idx_enum`

成员名称	功能描述
IDX_USBHS0	USBHS0索引
IDX_USBHS1	USBHS1索引

枚举类型 `pll_idx_enum`

表 3-910. 枚举类型 `pll_idx_enum`

成员名称	功能描述
IDX_PLL0	PLL0索引
IDX_PLL1	PLL1索引
IDX_PLL2	PLL2索引

枚举类型 `spi_idx_enum`

表 3-911. 枚举类型 `spi_idx_enum`

成员名称	功能描述
IDX_SPI0	SPI0索引
IDX_SPI1	SPI1索引
IDX_SPI2	SPI2索引
IDX_SPI3	SPI3索引
IDX_SPI4	SPI4索引

成员名称	功能描述
IDX_SPI5	SPI5索引

函数 rcu_deinit

函数rcu_deinit描述见下表：

表 3-912. 函数 rcu_deinit

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

例如：

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

```
rcu_deinit();
```

函数 rcu_periph_clock_enable

函数rcu_periph_clock_enable描述见下表：

表 3-913. 函数 rcu_periph_clock_enable

函数名称	rcu_periph_clock_enable
函数原形	void rcu_periph_clock_enable(rcu_periph_enum periph);
功能描述	使能外设时钟
先决条件	-
被调用函数	-
输入参数{in}	
periph	RCU外设，具体参考 表3-896. 枚举类型rcu_periph_enum
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* enable the USART0 clock */
```



```
rcu_periph_clock_enable(RCU_USART0);
```

函数 rcu_periph_clock_disable

函数rcu_periph_clock_disable描述见下表:

表 3-914. 函数 rcu_periph_clock_disable

函数名称	rcu_periph_clock_disable
函数原形	void rcu_periph_clock_disable(rcu_periph_enum periph);
功能描述	禁能外设时钟
先决条件	-
被调用函数	-
输入参数{in}	
periph	RCU外设, 具体参考 表3-896. 枚举类型rcu_periph_enum
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* disable the USART0 clock */
```

```
rcu_periph_clock_disable(RCU_USART0);
```

函数 rcu_periph_clock_sleep_enable

函数rcu_periph_clock_sleep_enable描述见下表:

表 3-915. 函数 rcu_periph_clock_sleep_enable

函数名称	rcu_periph_clock_sleep_enable
函数原形	void rcu_periph_clock_sleep_enable(rcu_periph_sleep_enum periph);
功能描述	在睡眠模式下, 使能外设时钟
先决条件	-
被调用函数	-
输入参数{in}	
periph	RCU外设, 参考 表3-897. 枚举类型rcu_periph_sleep_enum
输出参数{out}	
-	-
返回值	
-	-

例如:

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

```
rcu_periph_clock_sleep_enable(RCU_FMC_SLP);
```

函数 rcu_periph_clock_sleep_disable

函数rcu_periph_clock_sleep_disable描述见下表：

表 3-916. 函数 rcu_periph_clock_sleep_disable

函数名称	rcu_periph_clock_sleep_disable
函数原形	void rcu_periph_clock_sleep_disable(rcu_periph_sleep_enum periph);
功能描述	在睡眠模式下，禁能外设时钟
先决条件	-
被调用函数	-
输入参数{in}	
periph	RCU外设，参考 表3-897. 枚举类型rcu_periph_sleep_enum
输出参数{out}	
-	-
返回值	
-	-

例如：

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

```
rcu_periph_clock_sleep_disable(RCU_FMC_SLP);
```

函数 rcu_periph_reset_enable

函数rcu_periph_reset_enable描述见下表：

表 3-917. 函数 rcu_periph_reset_enable

函数名称	rcu_periph_reset_enable
函数原形	void rcu_periph_reset_enable(rcu_periph_reset_enum periph_reset);
功能描述	使能外设复位
先决条件	-
被调用函数	-
输入参数{in}	
periph_reset	RCU外设复位，参考 表3-917. 函数rcu_periph_reset_enable
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 3-918. 函数 rcu_periph_reset_disable

函数名称	rcu_periph_reset_disable
函数原形	void rcu_periph_reset_disable(rcu_periph_reset_enum periph_reset);
功能描述	禁能外设复位
先决条件	-
被调用函数	-
输入参数{in}	
periph_reset	RCU外设复位, 参考 表3-917. 函数rcu_periph_reset_enable
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 3-919. 函数 rcu_bkp_reset_enable

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

例如:

```
/* reset the BKP domain */
```

```
rcu_bkp_reset_enable();
```

函数 rcu_bkp_reset_disable

函数rcu_bkp_reset_disable描述见下表:

表 3-920. 函数 rcu_bkp_reset_disable

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

例如:

```
/* disable the BKP domain reset */
```

```
rcu_bkp_reset_disable();
```

函数 rcu_system_clock_source_config

函数rcu_system_clock_source_config描述见下表:

表 3-921. 函数 rcu_system_clock_source_config

函数名称	rcu_system_clock_source_config
函数原形	void rcu_system_clock_source_config(uint32_t ck_sys);
功能描述	配置选择系统时钟源
先决条件	-
被调用函数	-
输入参数{in}	
ck_sys	系统时钟源选择
<i>RCU_CKSYSSRC_IRC64MDIV</i>	选择CK_IRC64MDIV时钟作为CK_SYS时钟源
<i>RCU_CKSYSSRC_HXTAL</i>	选择CK_HXTAL时钟作为CK_SYS时钟源
<i>RCU_CKSYSSRC_LPIRC4M</i>	选择CK_LPIRC4M时钟作为CK_SYS时钟源
<i>RCU_CKSYSSRC_PLL0P</i>	选择CK_PLL0P时钟作为CK_SYS时钟源
输出参数{out}	
-	-
返回值	

-	-
---	---

例如：

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

函数 rcu_system_clock_source_get

函数rcu_system_clock_source_get描述见下表：

表 3-922. 函数 rcu_system_clock_source_get

函数名称	rcu_system_clock_source_get
函数原形	uint32_t rcu_system_clock_source_get(void);
功能描述	获取系统时钟源选择状态
先决条件	-
被调用函数	-
输入参数{in}	
-	-
输出参数{out}	
-	-
返回值	
uint32_t	RCU_SCSS_IRC64MDIV / RCU_SCSS_HXTAL / RCU_SCSS_LPIRC4M / RCU_SCSS_PLLP

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 3-923. 函数 rcu_ahb_clock_config

函数名称	rcu_ahb_clock_config
函数原形	void rcu_ahb_clock_config(uint32_t ck_ahb);
功能描述	配置AHB时钟预分频选择
先决条件	-
被调用函数	-
输入参数{in}	
ck_ahb	AHB预分频选择
RCU_AHB_CKSYS _DIVx	选择CK_SYS时钟x分频（x = 1, 2, 4, 8, 16, 64, 128, 256, 512）

输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* configure CK_SYS/128 */
```

```
rcu_ahb_clock_config(RCU_AHB_CKSYS_DIV128);
```

函数 rcu_apb1_clock_config

函数rcu_apb1_clock_config描述见下表：

表 3-924. 函数 rcu_apb1_clock_config

函数名称	rcu_apb1_clock_config
函数原形	void rcu_apb1_clock_config(uint32_t ck_apb1);
功能描述	配置APB1时钟预分频选择
先决条件	-
被调用函数	-
输入参数{in}	
ck_apb1	APB1预分频选择
<i>RCU_APB1_CK_AHB</i> <i>B_DIVx</i>	选择CK_AHB时钟x分频作为CK_APB1时钟（x = 1, 2, 4, 8, 16）
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* configure CK_AHB/16 as CK_APB1 */
```

```
rcu_apb1_clock_config(RCU_APB1_CK_AHB_DIV16);
```

函数 rcu_apb2_clock_config

函数rcu_apb2_clock_config描述见下表：

表 3-925. 函数 rcu_apb2_clock_config

函数名称	rcu_apb2_clock_config
函数原形	void rcu_apb2_clock_config(uint32_t ck_apb2);
功能描述	配置APB2时钟预分频选择
先决条件	-
被调用函数	-
输入参数{in}	

ck_apb2	APB2预分频选择
<i>RCU_APB2_CK_AHB_DIVx</i>	选择CK_AHB时钟x分频作为CK_APB2时钟（x = 1, 2, 4, 8, 16）
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* configure CK_AHB/8 as CK_APB2 */
rcu_apb2_clock_config(RCU_APB2_CK_AHB_DIV8);
```

函数 rcu_apb3_clock_config

函数rcu_apb3_clock_config描述见下表：

表 3-926. 函数 rcu_apb3_clock_config

函数名称	rcu_apb3_clock_config
函数原形	void rcu_apb3_clock_config(uint32_t ck_apb3);
功能描述	配置APB3时钟预分频选择
先决条件	-
被调用函数	-
输入参数{in}	
ck_apb3	APB3预分频选择
<i>RCU_APB3_CK_AHB_DIVx</i>	选择CK_AHB时钟x分频作为CK_APB2时钟（x = 1, 2, 4, 8, 16）
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* configure CK_AHB/8 as CK_APB3 */
rcu_apb3_clock_config(RCU_APB3_CK_AHB_DIV8);
```

函数 rcu_apb4_clock_config

函数rcu_apb4_clock_config描述见下表：

表 3-927. 函数 rcu_apb4_clock_config

函数名称	rcu_apb4_clock_config
函数原形	void rcu_apb4_clock_config(uint32_t ck_apb4);
功能描述	配置APB4时钟预分频选择

先决条件	-
被调用函数	-
输入参数{in}	
ck_apb4	APB4预分频选择
RCU_APB4_CKAHB_DIVx	选择CK_AHB时钟x分频作为CK_APB4时钟 (x = 1, 2, 4, 8, 16)
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* configure CK_AHB/8 as CK_APB4 */
```

```
rcu_apb4_clock_config(RCU_APB4_CKAHB_DIV8);
```

函数 rcu_ckout0_config

函数rcu_ckout0_config描述见下表:

表 3-928. 函数 rcu_ckout0_config

函数名称	rcu_ckout0_config
函数原形	void rcu_ckout0_config(uint32_t ckout0_src, uint32_t ckout0_div);
功能描述	配置CKOUT0时钟源选择及分频系数
先决条件	-
被调用函数	-
输入参数{in}	
ckout0_src	CKOUT0时钟源选择
RCU_CKOUT0SRC_IRC64MDIV	选择内部IRC64MDIV时钟
RCU_CKOUT0SRC_LXTAL	选择外部低速晶体振荡器时钟 (LXTAL)
RCU_CKOUT0SRC_HXTAL	选择外部高速晶体振荡器时钟 (HXTAL)
RCU_CKOUT0SRC_PLL0P	选择CK_PLL0P时钟
RCU_CKOUT0SRC_IRC48M	选择内部48M RC振荡器时钟
RCU_CKOUT0SRC_PER	选择外设时钟
RCU_CKOUT0SRC_USBHS060M	选择USBHS0 60M时钟
RCU_CKOUT0SRC_USBHS160M	选择USBHS1 60M时钟

输入参数{in}	
ckout1_div	CKOUT1分频系数
<i>RCU_CKOUT0_DIV</i> x	将CKOUT所选时钟x分频 (x = 1, 2, 3, 4...15)
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* configure the HXTAL as CK_OUT0 clock source */
```

```
rcu_ckout0_config(RCU_CKOUT0SRC_HXTAL, RCU_CKOUT0_DIV1);
```

函数 rcu_ckout1_config

函数rcu_ckout1_config描述见下表:

表 3-929. 函数 rcu_ckout1_config

函数名称	rcu_ckout1_config
函数原形	void rcu_ckout1_config(uint32_t ckout1_src, uint32_t ckout1_div);
功能描述	配置CKOUT1时钟源选择及分频系数
先决条件	-
被调用函数	-
输入参数{in}	
ckout1_src	CKOUT1时钟源选择
<i>RCU_CKOUT1SRC_SYSTEMCLOCK</i>	选择系统时钟CK_SYS
<i>RCU_CKOUT1SRC_PLL1R</i>	选择CK_PLL1R时钟
<i>RCU_CKOUT1SRC_HXTAL</i>	选择外部高速晶体振荡器时钟 (HXTAL)
<i>RCU_CKOUT1SRC_PLL0P</i>	选择CK_PLL0P时钟
<i>RCU_CKOUT1SRC_LPIRC4M</i>	选择内部低功耗4M RC振荡器时钟
<i>RCU_CKOUT1SRC_IRC32K</i>	选择内部低速32K振荡器时钟 (IRC32K)
<i>RCU_CKOUT1SRC_PLL2R</i>	选择CK_PLL2R时钟
输入参数{in}	
ckout1_div	CKOUT1分频系数
<i>RCU_CKOUT1_DIV</i> x	将CKOUT1所选时钟x分频 (x = 1,2,3,...15)

输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* configure the HXTAL as CK_OUT1 clock source */
```

```
rcu_ckout1_config(RCU_CKOUT1SRC_HXTAL, RCU_CKOUT1_DIV1);
```

函数 rcu_pll_input_output_clock_range_config

函数rcu_pll_input_output_clock_range_config描述见下表:

表 3-930. 函数 rcu_pll_input_output_clock_range_config

函数名称	rcu_pll_input_output_clock_range_config
函数原形	void rcu_pll_input_output_clock_range_config(pll_idx_enum pll_idx, uint32_t ck_input, uint32_t ck_output);
功能描述	配置PLL输入/输出时钟范围
先决条件	-
被调用函数	-
输入参数{in}	
pll_idx	pll索引, 参考 表3-910. 枚举类型pll_idx_enum
输入参数{in}	
ck_input	输入时钟范围
RCU_PLLRNG_1M_2M	输入时钟频率: 1-2MHz
RCU_PLLRNG_2M_4M	输入时钟频率: 2-4MHz
RCU_PLLRNG_4M_8M	输入时钟频率: 4-8MHz
RCU_PLLRNG_8M_16M	输入时钟频率: 8-16MHz
输入参数{in}	
RCU_PLLVCO_192M_836M	选择宽输出时钟范围: 192-836MHz
RCU_PLLVCO_150M_420M	选择窄输出时钟范围: 150-420MHz
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* configure the pll0 input and output clock range */
```

```
rcu_pll_input_output_clock_range_config(IDX_PLL0, RCU_PLLRNG_4M_8M,  
RCU_PLLVCO_192M_836M);
```

函数 rcu_pll_fractional_config

函数rcu_pll_fractional_config描述见下表：

表 3-931. 函数 rcu_pll_fractional_config

函数名称	rcu_pll_fractional_config
函数原形	void rcu_pll_fractional_config(pll_idx_enum pll_idx, uint32_t pll_fracn);
功能描述	配置PLL VCO倍频因子小数部分
先决条件	-
被调用函数	-
输入参数{in}	
pll_idx	pll索引，参考 表3 1121. 枚举类型pll_idx_enum
输入参数{in}	
pll_fracn	倍频因子小数部分
uint32_t	0x00-0x00001fff
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* configure fractional part of the multiplication factor for PLL0 VCO */
```

```
rcu_pll_fractional_config(IDX_PLL0, 1);
```

函数 rcu_pll_fractional_latch_enable

函数rcu_pll_fractional_latch_enable描述见下表：

表 3-932. 函数 rcu_pll_fractional_latch_enable

函数名称	rcu_pll_fractional_latch_enable
函数原形	void rcu_pll_fractional_latch_enable(pll_idx_enum pll_idx);
功能描述	使能PLL小数锁存功能
先决条件	-
被调用函数	-
输入参数{in}	
pll_idx	pll索引，参考 表3 1121. 枚举类型pll_idx_enum
输出参数{out}	
-	-
返回值	

-	-
---	---

例如：

```
/* enable PLL0 fractional latch */
```

```
rcu_pll_fractional_latch_enable(IDX_PLL0);
```

函数 rcu_pll_fractional_latch_disable

函数rcu_pll_fractional_latch_disable描述见下表：

表 3-933. 函数 rcu_pll_fractional_latch_disable

函数名称	rcu_pll_fractional_latch_disable
函数原形	void rcu_pll_fractional_latch_disable pll_idx_enum pll_idx);
功能描述	禁能PLL小数锁存功能
先决条件	-
被调用函数	-
输入参数{in}	
pll_idx	pll索引，参考 表3 1121. 枚举类型pll_idx_enum
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* disable PLL0 fractional latch */
```

```
rcu_pll_fractional_latch_disable(IDX_PLL0);
```

函数 rcu_pll_source_config

函数rcu_pll_source_config描述见下表：

表 3-934. 函数 rcu_pll_source_config

函数名称	rcu_pll_source_config
函数原形	void rcu_pll_source_config(uint32_t pll_src);
功能描述	配置主PLL时钟源选择
先决条件	-
被调用函数	-
输入参数{in}	
pll_src	PLL时钟源选择
RCU_PLLSRC_IRC64MDIV	IRC64MDIV时钟被选择为PLL时钟的时钟源
RCU_PLLSRC_LPIRC4M	LPIRC4M时钟被选择为PLL时钟的时钟源

RCU_PLLSRC_HXTAL	HXTAL时钟被选择为PLL时钟的时钟源
AL	
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* select RCU_PLLSRC_HXTAL as the PLL clock source */
```

```
rcu_pll_source_config(RCU_PLLSRC_HXTAL);
```

函数 rcu_pll0_config

函数rcu_pll0_config描述见下表:

表 3-935. 函数 rcu_pll0_config

函数名称	rcu_pll0_config
函数原形	ErrStatus rcu_pll0_config(uint32_t pll0_psc, uint32_t pll0_n, uint32_t pll0_p, uint32_t pll0_q, uint32_t pll0_r);
功能描述	配置PLL0时钟
先决条件	-
被调用函数	-
输入参数{in}	
pll0_psc	PLL0 VCO源时钟分频
uint32_t	1-63
输入参数{in}	
pll0_n	PLL0时钟倍频因子
uint32_t	9-512
输入参数{in}	
pll0_p	PLL0P时钟分频因子
uint32_t	1-128
输入参数{in}	
pll0_q	PLL0Q时钟分频因子
uint32_t	1-128
输入参数{in}	
pll0_r	PLL0R时钟分频因子
uint32_t	1-128
输出参数{out}	
-	-
返回值	
ErrStatus	SUCCESS or ERROR-

例如:

```
/* configure the PLL0 */
```

```
rcu_pll0_config(1, 200, 1, 2, 2);
```

函数 rcu_pll1_config

函数rcu_pll1_config描述见下表：

表 3-936. 函数 rcu_pll1_config

函数名称	rcu_pll1_config
函数原形	ErrStatus rcu_pll1_config(uint32_t pll1_psc, uint32_t pll1_n, uint32_t pll1_p, uint32_t pll1_q, uint32_t pll1_r);
功能描述	配置PLL1时钟
先决条件	-
被调用函数	-
输入参数{in}	
pll1_psc	PLL1 VCO源时钟分频
uint32_t	1-63
输入参数{in}	
pll1_n	PLL1时钟倍频因子
uint32_t	9-512
输入参数{in}	
pll1_p	PLL1P时钟分频因子
uint32_t	1-128
输入参数{in}	
pll1_q	PLL1Q时钟分频因子
uint32_t	1-128
输入参数{in}	
pll1_r	PLL1R时钟分频因子
uint32_t	1-128
输出参数{out}	
-	-
返回值	
ErrStatus	SUCCESS or ERROR-

例如：

```
/* configure the PLL1 */
```

```
rcu_pll1_config(1, 200, 1, 2, 2);
```

函数 rcu_pll2_config

函数rcu_pll2_config描述见下表：

表 3-937. 函数 rcu_pll2_config

函数名称	rcu_pll2_config
函数原形	ErrStatus rcu_pll2_config(uint32_t pll2_psc, uint32_t pll2_n, uint32_t pll2_p, uint32_t pll2_q, uint32_t pll2_r);
功能描述	配置PLL2时钟
先决条件	-
被调用函数	-
输入参数{in}	
pll2_psc	PLL2 VCO源时钟分频
uint32_t	1-63
输入参数{in}	
pll2_n	PLL2时钟倍频因子
uint32_t	9-512
输入参数{in}	
pll2_p	PLL2P时钟分频因子
uint32_t	1-128
输入参数{in}	
pll2_q	PLL2Q时钟分频因子
uint32_t	1-128
输入参数{in}	
pll2_r	PLL2R时钟分频因子
uint32_t	1-128
输出参数{out}	
-	-
返回值	
ErrStatus	SUCCESS or ERROR-

例如:

```
/* configure the PLL2 */
```

```
rcu_pll2_config(1, 200, 1, 2, 2);
```

函数 rcu_pll_clock_output_enable

函数rcu_pll_clock_output_enable描述见下表:

表 3-938. 函数 rcu_pll_clock_output_enable

函数名称	rcu_pll_clock_output_enable
函数原形	void rcu_pll_clock_output_enable(uint32_t pllxy);
功能描述	使能PLL P/Q/R时钟输出
先决条件	-
被调用函数	-
输入参数{in}	
pllxy	选择PLL P/Q/R分频输出使能

<i>RCU_PLL0P</i>	选择PLL0P分频输出使能
<i>RCU_PLL0Q</i>	选择PLL0Q分频输出使能
<i>RCU_PLL0R</i>	选择PLL0R分频输出使能
<i>RCU_PLL1P</i>	选择PLL1P分频输出使能
<i>RCU_PLL1Q</i>	选择PLL1Q分频输出使能
<i>RCU_PLL1R</i>	选择PLL1R分频输出使能
<i>RCU_PLL2P</i>	选择PLL2P分频输出使能
<i>RCU_PLL2Q</i>	选择PLL2Q分频输出使能
<i>RCU_PLL2R</i>	选择PLL2R分频输出使能
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* enable PLL0P divider output */
```

```
rcu_pll_clock_output_enable(RCU_PLL0P);
```

函数 `rcu_pll_clock_output_disable`

函数`rcu_pll_clock_output_disable`描述见下表：

表 3-939. 函数 `rcu_pll_clock_output_disable`

函数名称	<code>rcu_pll_clock_output_disable</code>
函数原形	<code>void rcu_pll_clock_output_disable(uint32_t pllxy);</code>
功能描述	禁能PLL P/Q/R时钟输出
先决条件	-
被调用函数	-
输入参数{in}	
pllxy	选择PLL P/Q/R分频输出使能
<i>RCU_PLL0P</i>	选择PLL0P分频输出使能
<i>RCU_PLL0Q</i>	选择PLL0Q分频输出使能
<i>RCU_PLL0R</i>	选择PLL0R分频输出使能
<i>RCU_PLL1P</i>	选择PLL1P分频输出使能
<i>RCU_PLL1Q</i>	选择PLL1Q分频输出使能
<i>RCU_PLL1R</i>	选择PLL1R分频输出使能
<i>RCU_PLL2P</i>	选择PLL2P分频输出使能
<i>RCU_PLL2Q</i>	选择PLL2Q分频输出使能
<i>RCU_PLL2R</i>	选择PLL2R分频输出使能
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* disable PLL0P divider output */
rcu_pll_clock_output_disable(RCU_PLL0P);
```

函数 rcu_pllusb0_config

函数rcu_pllusb0_config描述见下表:

表 3-940. 函数 rcu_pllusb0_config

函数名称	rcu_pllusb0_config
函数原形	void rcu_pllusb0_config(uint32_t pllusb_presel, uint32_t pllusb_predv, uint32_t pllusb_mf, uint32_t usbhsv);
功能描述	配置PLLUSBHSP0时钟
先决条件	-
被调用函数	-
输入参数{in}	
pllusb_presel	PLLUSBHSPRE时钟选择
RCU_PLLUSBHSP RE_HXTAL	PLLUSBHSPRE选择HXTAL作为时钟
RCU_PLLUSBHSP RE_IRC48M	PLLUSBHSPRE选择IRC48M作为时钟
输入参数{in}	
pllusb_predv	PLLUSBHS时钟分频因子
RCU_PLLUSBHSP RE_DIVx (x= 1...15)	选择RCU_PLLUSBHSPRE_DIVx作为PLLUSBHS时钟分频因子
输入参数{in}	
pllusb_mf	PLLUSBHS时钟倍频因子
RCU_PLLUSBHS_ MULx (x = 16,17...127)	选择RCU_PLLUSBHS_MULx作为PLLUSBHS时钟分频因子
输入参数{in}	
usbhsv	USBHSDV时钟分频因子
RCU_USBHS_DIVx (x = 2,4...16)	选择 RCU_USBHS_DIVx作为USBHSDV时钟分频因子
输出参数{out}	
-	-
返回值	
ErrStatus	SUCCESS or ERROR-

例如:

```
/* configure the PLLUSBHSP0 */
rcu_pllusb0_config(RCU_PLLUSBHSPRE_IRC48M, RCU_PLLUSBHSPRE_DIV1,
```

RCU_PLLUSBHS_MUL1, RCU_USBHS_DIV1);

函数 rcu_pllusb1_config

函数rcu_pllusb1_config描述见下表:

表 3-941. 函数 rcu_pllusb1_config

函数名称	rcu_pllusb1_config
函数原形	void rcu_pllusb1_config(uint32_t pllusb_presel, uint32_t pllusb_predv, uint32_t pllusb_mf, uint32_t usbhsv);
功能描述	配置PLLUSBHS1时钟
先决条件	-
被调用函数	-
输入参数{in}	
pllusb_presel	PLLUSBHSPRE时钟选择
RCU_PLLUSBHSPRE_HXTAL	PLLUSBHSPRE选择HXTAL作为时钟
RCU_PLLUSBHSPRE_IRC48M	PLLUSBHSPRE选择IRC48M作为时钟
输入参数{in}	
pllusb_predv	PLLUSBHS时钟分频因子
RCU_PLLUSBHSPRE_DIVx (x = 1...15)	选择RCU_PLLUSBHSPRE_DIVx作为PLLUSBHS时钟分频因子
输入参数{in}	
pllusb_mf	PLLUSBHS时钟倍频因子
RCU_PLLUSBHSMULx (x = 16,17...127)	选择RCU_PLLUSBHSMULx作为PLLUSBHS时钟分频因子
输入参数{in}	
usbhsv	USBHSDV时钟分频因子
RCU_USBHS_DIVx (x = 2,4...16)	选择 RCU_USBHS_DIVx作为USBHSDV时钟分频因子
输出参数{out}	
-	-
返回值	
ErrStatus	SUCCESS or ERROR-

例如:

```
/* configure the PLLUSBHS1 */
```

```
rcu_pllusb1_config(RCU_PLLUSBHSPRE_IRC48M, RCU_PLLUSBHSPRE_DIV1,
RCU_PLLUSBHSMUL1, RCU_USBHS_DIV1);
```

函数 rcu_rtc_clock_config

函数rcu_rtc_clock_config描述见下表：

表 3-942. 函数 rcu_rtc_clock_config

函数名称	rcu_rtc_clock_config
函数原形	void rcu_rtc_clock_config(uint32_t rtc_clock_source);
功能描述	配置RTC时钟
先决条件	-
被调用函数	-
输入参数{in}	
rtc_clock_source	RTC时钟源选择
RCU_RTCSRC_NO NE	未选择时钟
RCU_RTCSRC_LX TAL	选择CK_LXTAL作为RTC时钟源
RCU_RTCSRC_IRC 32K	选择内部32K RC振荡器时钟作为RTC时钟源
RCU_RTCSRC_HX TAL_DIV_RTCDIV	选择外部高速晶振的x分频作为RTC时钟源
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* configure the RTC clock source selection */
rcu_rtc_clock_config(RCU_RTCSRC_IRC32K);
```

函数 rcu_rtc_div_config

函数rcu_rtc_div_config描述见下表：

表 3-943. 函数 rcu_rtc_div_config

函数名称	rcu_rtc_div_config
函数原形	void rcu_rtc_div_config(uint32_t rtc_div);
功能描述	当选择HXTAL作为RTC时钟源时，配置RTC时钟分频
先决条件	-
被调用函数	-
输入参数{in}	
rtc_div	RTC时钟源选择
RCU_RTC_HXTAL_	未选择时钟

NONE	
RCU_RTC_HXTAL_ DIVx (x = 2,3...63)	选择CK_HXTAL / x作为RTC时钟源
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* select CK_HXTAL / 25 as the RTC clock source */
```

```
rcu_rtc_div_config(RCU_RTC_HXTAL_DIV25);
```

函数 rcu_ck48m_clock_config

函数rcu_ck48m_clock_config描述见下表:

表 3-944. 函数 rcu_ck48m_clock_config

函数名称	rcu_ck48m_clock_config
函数原形	void rcu_ck48m_clock_config(uint32_t ck48m_clock_source);
功能描述	配置CK48M的时钟源选择
先决条件	-
被调用函数	
输入参数{in}	
ck48m_clock_source	CK48M时钟源选择
RCU_CK48MSRC_ PLL48M	PLL48M被选择为CK48M时钟的时钟源
RCU_CK48MSRC_ IRC48M	IRC48M被选择为CK48M时钟的时钟源
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* configure the CK48M clock source selection */
```

```
rcu_ck48m_clock_config(RCU_CK48MSRC_IRC48M);
```

函数 rcu_pll48m_clock_config

函数rcu_pll48m_clock_config描述见下表:

表 3-945. 函数 rcu_pll48m_clock_config

函数名称	rcu_pll48m_clock_config
函数原形	void rcu_pll48m_clock_config(uint32_t pll48m_clock_source);
功能描述	配置PLL48M的时钟源选择
先决条件	-
被调用函数	
输入参数{in}	
pll48m_clock_source	PLL48M时钟源选择
RCU_PLL48MSRC_PLL0Q	PLL0Q被选择为PLL48M时钟的时钟源
RCU_PLL48MSRC_PLL2P	PLL2P被选择为PLL48M时钟的时钟源
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* configure the PLL48M clock selection */
```

```
rcu_pll48m_clock_config(RCU_PLL48MSRC_PLL0Q);
```

函数 rcu_irc64mdiv_clock_config

函数rcu_irc64mdiv_clock_config描述见下表：

表 3-946. 函数 rcu_irc64mdiv_clock_config

函数名称	rcu_irc64mdiv_clock_config
函数原形	void rcu_irc64mdiv_clock_config(uint32_t ck_irc64mdiv);
功能描述	配置IRC64M时钟分频选择
先决条件	-
被调用函数	
输入参数{in}	
ck_irc64mdiv	IRC64M时钟分频选择
RCU_IRC64M_DIV1	选择CK_IRC64M/1
RCU_IRC64M_DIV2	选择CK_IRC64M/2
RCU_IRC64M_DIV4	选择CK_IRC64M/4
RCU_IRC64M_DIV8	选择CK_IRC64M/8
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* configure the IRC64M clock divider selection */
rcu_irc64mdiv_clock_config(RCU_IRC64M_DIV1);
```

函数 rcu_irc64mdiv_freq_get

函数rcu_irc64mdiv_freq_get描述见下表：

表 3-947. 函数 rcu_irc64mdiv_freq_get

函数名称	rcu_irc64mdiv_freq_get
函数原形	uint32_t rcu_irc64mdiv_freq_get(void);
功能描述	获取IRC64MDIV时钟
先决条件	-
被调用函数	
输入参数{in}	
-	-
输出参数{out}	
-	-
返回值	
uint32_t	0x0-0xFFFFFFFF

例如：

```
/* get the irc64mdiv clock */
rcu_irc64mdiv_freq_get();
```

函数 rcu_timer_clock_prescaler_config

函数rcu_timer_clock_prescaler_config描述见下表：

表 3-948. 函数 rcu_timer_clock_prescaler_config

函数名称	rcu_timer_clock_prescaler_config
函数原形	void rcu_timer_clock_prescaler_config(uint32_t timer_clock_prescaler);
功能描述	配置TIMER时钟源
先决条件	-
被调用函数	
输入参数{in}	
timer_clock_prescaler	TIMER时钟源选择
RCU_TIMER_PSC_MUL2	如果CK_APBx = CK_AHB 或者 CK_APBx = CK_AHB / 2, CK_TIMERx = CK_AHB, 否则CK_TIMERx = 2 x CK_APBx
RCU_TIMER_PSC_MUL4	如果CK_APBx = CK_AHB 或者 CK_APBx = CK_AHB / 2 或者CK_APBx = CK_AHB / 4, CK_TIMERx = CK_AHB, 否则CK_TIMERx = 4 x CK_APBx
输出参数{out}	

-	-
返回值	
-	-

例如：

```
/* configure TIMER clock source */
```

```
rcu_timer_clock_prescaler_config(RCU_TIMER_PSC_MUL4);
```

函数 rcu_spi_clock_config

函数rcu_spi_clock_config描述见下表：

表 3-949. 函数 rcu_spi_clock_config

函数名称	rcu_spi_clock_config
函数原形	void rcu_spi_clock_config(spi_idx_enum spi_idx, uint32_t ck_spi);
功能描述	配置SPI时钟
先决条件	-
被调用函数	-
输入参数{in}	
spi_idx	SPI索引，参考 表3-911. 枚举类型spi_idx_enum
输入参数{in}	
ck_spi	USARTx时钟源
RCU_SPISRC_PLL0Q	选择CK_PLL0Q时钟作为SPI时钟，适用于SPI0 / SPI1 / SPI2
RCU_SPISRC_PLL1P	选择CK_PLL1P时钟作为SPI时钟，适用于SPI0/SPI1/SPI2
RCU_SPISRC_PLL2P	选择CK_PLL2P时钟作为SPI时钟，适用于SPI0 / SPI1 / SPI2
RCU_SPISRC_I2S_CKIN	选择I2S_CKIN时钟作为SPI时钟，适用于SPI0 / SPI1 / SPI2
RCU_SPISRC_PER	选择CK_PER时钟作为SPI时钟，适用于SPI0 / SPI1 / SPI2
RCU_SPISRC_APB2	选择CK_APB2时钟作为SPI时钟，适用于SPI3 / SPI4 / SPI5
RCU_SPISRC_PLL1Q	选择CK_PLL1Q时钟作为SPI时钟，适用于SPI3 / SPI4 / SPI5
RCU_SPISRC_PLL2Q	选择CK_PLL2Q时钟作为SPI时钟，适用于SPI3 / SPI4 / SPI5
RCU_SPISRC_IRC64MDIV	选择CK_IRC64MDIV时钟作为SPI时钟，适用于SPI3 / SPI4 / SPI5
RCU_SPISRC_LPIRC4M	选择CK_LPIRC4M时钟作为SPI时钟，适用于SPI3 / SPI4 / SPI5
RCU_SPISRC_HXTAL	选择CK_HXTAL时钟作为SPI时钟，适用于SPI3 / SPI4 / SPI5

<code>RCU_SPI5SRC_I2S</code> <code>_CKIN</code>	选择I2S_CKIN时钟作为SPI时钟，适用于SPI5
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* configure the PLLQ as SPI0 clock */
```

```
rcu_spi_clock_config(IDX_SPI0, RCU_SPI5SRC_PLLQ);
```

函数 `rcu_deepsleep_wakeup_sys_clock_config`

函数`rcu_deepsleep_wakeup_sys_clock_config`描述见下表：

表 3-950. 函数 `rcu_sdio_clock_config`

函数名称	<code>rcu_deepsleep_wakeup_sys_clock_config</code>
函数原形	<code>void rcu_deepsleep_wakeup_sys_clock_config(uint32_t ck_dspwussel);</code>
功能描述	配置深度睡眠模式下系统唤醒时钟源选择
先决条件	-
被调用函数	-
输入参数{in}	
<code>ck_dspwussel</code>	深度睡眠模式下系统唤醒时钟源
<code>RCU_DSPWUSSEL</code> <code>_IRC64MDIV</code>	选择CK_IRC64MDIV作为深度睡眠模式下系统唤醒时钟源
<code>RCU_DSPWUSSEL</code> <code>_LPIRC4M</code>	选择CK_LPIRC4M作为深度睡眠模式下系统唤醒时钟源
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* configure the CK_LPIRC4M as Deep-sleep wakeup system clock source */
```

```
rcu_deepsleep_wakeup_sys_clock_config(RCU_DSPWUSSEL_LPIRC4M);
```

函数 `rcu_usart_clock_config`

函数`rcu_usart_clock_config`描述见下表：

表 3-951. 函数 `rcu_usart_clock_config`

函数名称	<code>rcu_usart_clock_config</code>
函数原形	<code>void rcu_usart_clock_config(usart_idx_enum usart_idx, uint32_t ck_usart);</code>

功能描述	配置串口时钟
先决条件	-
被调用函数	-
输入参数{in}	
usart_idx	USART索引, 参考 表3-905. 枚举类型usart_idx_enum
输入参数{in}	
ck_usart	USARTx时钟源
RCU_USARTSRC_APB	选择CK_APB时钟作为USART时钟
RCU_USARTSRC_AHB	选择CK_AHB时钟作为USART时钟
RCU_USARTSRC_LXTAL	选择CK_LXTAL时钟作为USART时钟
RCU_USARTSRC_IRC64MDIV	选择CK_IRC64MDIV时钟作为USART时钟
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* configure the LXTAL as USART0 clock */
```

```
rcu_usart_clock_config(IDX_USART0, RCU_USARTSRC_LXTAL);
```

函数 rcu_i2c_clock_config

函数rcu_i2c_clock_config描述见下表:

表 3-952. 函数 rcu_i2c_clock_config

函数名称	rcu_i2c_clock_config
函数原形	void rcu_i2c_clock_config(i2c_idx_enum i2c_idx, uint32_t ck_i2c);
功能描述	配置I2C时钟
先决条件	-
被调用函数	-
输入参数{in}	
i2c_idx	I2C索引, 参考 表3-906. 枚举类型i2c_idx_enum
输入参数{in}	
ck_i2c	I2Cx时钟源
RCU_I2CSRC_APB1	选择CK_APB1时钟作为I2C时钟
RCU_I2CSRC_PLL2R	选择CK_PLL2R时钟作为I2C时钟
RCU_I2CSRC_IRC	选择CK_IRC64MDIV时钟作为I2C时钟

64MDIV	
RCU_I2CSRC_LPIRC4M	选择CK_LPIRC4M时钟作为I2C时钟
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* configure the CK_APB1 as I2C0 clock */
```

```
rcu_i2c_clock_config(IDX_I2C0, RCU_I2CSRC_APB1);
```

函数 rcu_can_clock_config

函数rcu_can_clock_config描述见下表:

表 3-953. 函数 rcu_can_clock_config

函数名称	rcu_can_clock_config
函数原形	void rcu_can_clock_config(can_idx_enum can_idx, uint32_t ck_can);
功能描述	配置CAN时钟
先决条件	-
被调用函数	-
输入参数{in}	
can_idx_enum	CAN索引, 参考 表3-907. 枚举类型can_idx_enum
输入参数{in}	
ck_can	CANx时钟源
RCU_CANSRC_HXTAL	选择HXTAL时钟作为CAN时钟
RCU_CANSRC_APB2	选择CK_APB2时钟作为CAN时钟
RCU_CANSRC_APB2_DIV2	选择CK_APB2/2时钟作为CAN时钟
RCU_CANSRC_IRC64MDIV	选择CK_IRC64MDIV时钟作为CAN时钟
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* configure the CK_IRC64MDIV as the CAN0 clock source */
```

```
rcu_can_clock_config(IDX_CAN0, RCU_CANSRC_IRC64MDIV);
```

函数 **rcu_adc_clock_config**

函数rcu_adc_clock_config描述见下表：

表 3-954. 函数 **rcu_adc_clock_config**

函数名称	rcu_adc_clock_config
函数原形	void rcu_adc_clock_config(adc_idx_enum adc_idx, uint32_t ck_adc);
功能描述	配置adc时钟
先决条件	-
被调用函数	-
输入参数{in}	
adc_idx	ADC索引，参考 表3-908. 枚举类型adc_idx_enum
输入参数{in}	
ck_adc	ADCx时钟源
RCU_ADCSRC_PL L1P	选择CK_PLL1P时钟作为ADC时钟
RCU_ADCSRC_PL L2R	选择CK_PLL2R时钟作为ADC时钟
RCU_ADCSRC_PE R	选择CK_PER时钟作为ADC时钟
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* configure the CK_PLL1P as the ADC0 clock source */
```

```
rcu_adc_clock_config(IDX_ADC0, CK_PLL1P);
```

函数 **rcu_exmc_clock_config**

函数rcu_exmc_clock_config描述见下表：

表 3-955. 函数 **rcu_exmc_clock_config**

函数名称	rcu_exmc_clock_config
函数原形	void rcu_exmc_clock_config(uint32_t ck_exmc);
功能描述	配置exmc时钟源时钟
先决条件	-
被调用函数	-
输入参数{in}	
ck_exmc	EXMC时钟源
RCU_EXMCSRC_A HB	选择CK_AHB时钟作为EXMC时钟
RCU_EXMCSRC_P	选择CK_PLLQ时钟作为EXMC时钟

LLQ	
RCU_EXMCSRC_P LL1R	选择CK_PLL1R时钟作为EXMC时钟
RCU_EXMCSRC_P ER	选择CK_PER时钟作为EXMC时钟
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* configure the CK_AHB as the EXMC clock source */
```

```
rcu_exmc_clock_config(RCU_EXMCSRC_AHB);
```

函数 rcu_hpdf_clock_config

函数rcu_hpdf_clock_config描述见下表：

表 3-956. 函数 rcu_hpdf_clock_config

函数名称	rcu_hpdf_clock_config
函数原形	void rcu_hpdf_clock_config(uint32_t ck_hppdf);
功能描述	配置HPDF时钟源时钟
先决条件	-
被调用函数	-
输入参数{in}	
ck_hppdf	HPDF时钟源
RCU_HPDFSRC_A PB2	选择CK_APB2时钟作为HPDF时钟
RCU_HPDFSRC_A HB	选择CK_AHB时钟作为HPDF时钟
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* configure the CK_AHB as the HPDF clock source */
```

```
rcu_hpdf_clock_config(CK_AHB);
```

函数 rcu_per_clock_config

函数rcu_per_clock_config描述见下表：

表 3-957. 函数 rcu_per_clock_config

函数名称	rcu_per_clock_config
函数原形	void rcu_per_clock_config(uint32_t ck_per)
功能描述	配置PER时钟源时钟
先决条件	-
被调用函数	-
输入参数{in}	
ck_per	PER时钟源
RCU_PERSRC_IRC64MDIV	选择CK_IRC64MDIV时钟作为PER时钟
RCU_PERSRC_LPIRC4M	选择CK_LPIRC4M时钟作为PER时钟
RCU_PERSRC_HXTAL	选择CK_HXTAL时钟作为PER时钟
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* configure the CK_IRC64MDIV as the PER clock source */
```

```
rcu_per_clock_config(RCU_PERSRC_IRC64MDIV);
```

函数 rcu_usbhs_pll1qpsc_config

函数rcu_usbhs_pll1qpsc_config描述见下表：

表 3-958. 函数 rcu_usbhs_pll1qpsc_config

函数名称	rcu_usbhs_pll1qpsc_config
函数原形	void rcu_usbhs_pll1qpsc_config(usbhs_idx_enum usbhs_idx,uint32_t ck_usbhspsc);
功能描述	配置PLL1Q预分频时钟源时钟
先决条件	-
被调用函数	-
输入参数{in}	
usbhs_idx	USBHS索引，参考 表3-909. 枚举类型usbhs_idx_enum
输入参数{in}	
ck_usbhspsc	PLL1Q时钟分频
RCU_USBHSPSC_DIV(x=1,2...8)	CK_PLL1Q / X
输出参数{out}	
-	-
返回值	

-	-
---	---

例如：

```
/* configure the RCU_USBHSPSC_DIV as RCU_USBHSPSC_DIV1 */
rcu_usbhs_pll1qpsc_config(IDX_USBHS0, RCU_USBHSPSC_DIV1);
```

函数 rcu_usb48m_clock_config

函数rcu_usb48m_clock_config描述见下表：

表 3-959. 函数 rcu_usb48m_clock_config

函数名称	rcu_usb48m_clock_config
函数原形	void rcu_usb48m_clock_config(usbhs_idx_enum usbhs_idx,uint32_t ck_usb48m);
功能描述	配置USBHS48M时钟源选择
先决条件	-
被调用函数	-
输入参数{in}	
usbhs_idx	USBHS索引，参考 表3 1120. 枚举类型usbhs_idx_enum
输入参数{in}	
ck_usb48m	USBHS48M时钟源选择
RCU_USB48MSRC_PLL0R	选择CK_PLL0R作为USBHS48M时钟源
RCU_USB48MSRC_PLLUSBHS	选择CK_PLLUSBHS作为USBHS48M时钟源
RCU_USB48MSRC_PLL1Q	选择CK_PLL1Q作为USBHS48M时钟源
RCU_USB48MSRC_IRC48M	选择CK_IRC48M作为USBHS48M时钟源
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* configure the CK_IRC48M as USBHS48M clock */
rcu_usb48m_clock_config(IDX_USBHS0, RCU_USB48MSRC_IRC48M);
```

函数 rcu_usbhs_clock_config

函数rcu_usbhs_clock_config描述见下表：

表 3-960. 函数 rcu_usbhs_clock_config

函数名称	rcu_usbhs_clock_config
函数原形	void rcu_usbhs_clock_config(usbhs_idx_enum usbhs_idx,uint32_t ck_usbhs);
功能描述	配置USBHS时钟源选择
先决条件	-
被调用函数	-
输入参数{in}	
usbhs_idx	USBHS索引, 参考 表3 1120. 枚举类型usbhs_idx_enum
输入参数{in}	
ck_usbhs	USBHS时钟源选择
RCU_USBHSSEL_4 8M	选择48M作为USBHS时钟源
RCU_USBHSSEL_6 0M	选择60M作为USBHS时钟源
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* configure the 48M as USBHS clock */
```

```
rcu_usbhs_clock_config(IDX_USBHS0, RCU_USBHSSEL_48M);
```

函数 rcu_usbhs_clock_selection_enable

函数rcu_usbhs_clock_selection_enable描述见下表:

表 3-961. 函数 rcu_usbhs_clock_selection_enable

函数名称	rcu_usbhs_clock_selection_enable
函数原形	void rcu_usbhs_clock_selection_enable(usbhs_idx_enum usbhs_idx);
功能描述	使能 USBHS 时钟源选择
先决条件	-
被调用函数	-
输入参数{in}	
usbhs_idx	USBHS索引, 参考 表3 1120. 枚举类型usbhs_idx_enum
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* enable the USBHS0 clock source selection */
```

```
rcu_usbhs_clock_selection_enable(IDX_USBHS0);
```

函数 rcu_usbhs_clock_selection_disable

函数rcu_usbhs_clock_selection_disable描述见下表：

表 3-962. 函数 rcu_usbhs_clock_selection_disable

函数名称	rcu_usbhs_clock_selection_disable
函数原形	void rcu_usbhs_clock_selection_disable(usbhs_idx_enum usbhs_idx);
功能描述	禁能 USBHS 时钟源选择
先决条件	-
被调用函数	-
输入参数{in}	
usbhs_idx	USBHS索引，参考 表3 1120. 枚举类型usbhs_idx_enum
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* disable the USBHS0 clock source selection */
```

```
rcu_usbhs_clock_selection_disable(IDX_USBHS0);
```

函数 rcu_lxtal_drive_capability_config

函数rcu_lxtal_drive_capability_config描述见下表：

表 3-963. 函数 rcu_lxtal_drive_capability_config

函数名称	rcu_lxtal_drive_capability_config
函数原形	void rcu_lxtal_drive_capability_config(uint32_t lxtal_dricap);
功能描述	配置LXTAL驱动能力
先决条件	-
被调用函数	-
输入参数{in}	
lxtal_dricap	LXTAL驱动能力
RCU_LXTAL_LOW_DRI	低驱动力
RCU_LXTAL_MED_LOWDRI	中低驱动力
RCU_LXTAL_MED_HIGHDRI	中高驱动力
RCU_LXTAL_HIGH_DRI	高驱动力
输出参数{out}	

-	-
返回值	
-	-

例如：

```
/* configure the LXTAL drive capability */
```

```
rcu_lxtal_drive_capability_config(RCU_LAXTAL_LOWDRI);
```

函数 rcu_osci_stab_wait

函数rcu_osci_stab_wait描述见下表：

表 3-964. 函数 rcu_osci_stab_wait

函数名称	rcu_osci_stab_wait
函数原形	ErrStatus rcu_osci_stab_wait(rcu_osci_type_enum osci);
功能描述	等待振荡器稳定标志位置位或振荡器起振超时
先决条件	-
被调用函数	rcu_flag_get
输入参数{in}	
osci	振荡器类型，参考 表3-903. 枚举类型rcu_osci_type_enum
输出参数{out}	
-	-
返回值	
ErrStatus	SUCCESS 或 ERROR

例如：

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

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

函数 rcu_osci_on

函数rcu_osci_on描述见下表：

表 3-965. 函数 rcu_osci_on

函数名称	rcu_osci_on
函数原形	void rcu_osci_on(rcu_osci_type_enum osci);
功能描述	打开振荡器
先决条件	-
被调用函数	-
输入参数{in}	
osci	振荡器类型，参考 表3-903. 枚举类型rcu_osci_type_enum
输出参数{out}	

-	-
返回值	
-	-

例如：

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

```
rcu_osci_on(RCU_HXTAL);
```

函数 rcu_osci_off

函数rcu_osci_off描述见下表：

表 3-966. 函数 rcu_osci_off

函数名称	rcu_osci_off
函数原形	void rcu_osci_off(rcu_osci_type_enum osci);
功能描述	关闭振荡器
先决条件	-
被调用函数	-
输入参数{in}	
osci	振荡器类型，参考 表3-903. 枚举类型rcu_osci_type_enum
输出参数{out}	
-	-
返回值	
-	-

例如：

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

```
rcu_osci_off(RCU_HXTAL);
```

函数 rcu_osci_bypass_mode_enable

函数rcu_osci_bypass_mode_enable描述见下表：

表 3-967. 函数 rcu_osci_bypass_mode_enable

函数名称	rcu_osci_bypass_mode_enable
函数原形	void rcu_osci_bypass_mode_enable(rcu_osci_type_enum osci);
功能描述	使能振荡器时钟旁路模式
先决条件	HXTALEN或LXTALEN应在使能振荡器时钟旁路模式前先复位
被调用函数	-
输入参数{in}	
osci	振荡器类型，参考 表3-903. 枚举类型rcu_osci_type_enum
输出参数{out}	
-	-

返回值	
-	-

例如：

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

```
rcu_osc_bypass_mode_enable(RCU_HXTAL);
```

函数 rcu_osc_bypass_mode_disable

函数rcu_osc_bypass_mode_disable描述见下表：

表 3-968. 函数 rcu_osc_bypass_mode_disable

函数名称	rcu_osc_bypass_mode_disable
函数原形	void rcu_osc_bypass_mode_disable(rcu_osc_type_enum osci);
功能描述	禁能振荡器时钟旁路模式
先决条件	HXTALEN或LXTALEN应在使能振荡器时钟旁路模式前先复位
被调用函数	-
输入参数{in}	
osci	振荡器类型，参考 表3-903. 枚举类型rcu_osc_type_enum
输出参数{out}	
-	-
返回值	
-	-

例如：

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

```
rcu_hxtal_clock_monitor_disable();
```

函数 rcu_irc64m_adjust_value_set

函数rcu_irc64m_adjust_value_set描述见下表：

表 3-969. 函数 rcu_irc64m_adjust_value_set

函数名称	rcu_irc64m_adjust_value_set
函数原形	void rcu_irc64m_adjust_value_set(uint32_t irc64m_adjval);
功能描述	设置内部64MHz RC振荡器时钟调整值
先决条件	-
被调用函数	-
输入参数{in}	
irc64m_adjval	IRC64M调整值（0到0x7F之间）
输出参数{out}	
-	-
返回值	

-	-
---	---

例如：

```
/* set the IRC6M adjust value */
```

```
rcu_irc64m_adjust_value_set(0x20);
```

函数 rcu_lpirc4m_adjust_value_set

函数rcu_lpirc4m_adjust_value_set描述见下表：

表 3-970. 函数 rcu_lpirc4m_adjust_value_set

函数名称	rcu_lpirc4m_adjust_value_set
函数原形	void rcu_lpirc4m_adjust_value_set(uint32_t lpirc4m_adjval);
功能描述	设置内部低功耗4MHz RC振荡器时钟调整值
先决条件	-
被调用函数	-
输入参数{in}	
lpirc4m_adjval	LPIRC4M调整值（0到0x3F之间）
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* set the LPIRC4M adjust value */
```

```
rcu_lpirc4m_adjust_value_set(0x10);
```

函数 rcu_hxtal_clock_monitor_enable

函数rcu_hxtal_clock_monitor_enable描述见下表：

表 3-971. 函数 rcu_hxtal_clock_monitor_enable

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

例如:

```
/* enable the HXTAL clock monitor */  
  
rcu_hxtal_clock_monitor_enable();
```

函数 rcu_hxtal_clock_monitor_disable

函数rcu_hxtal_clock_monitor_disable描述见下表:

表 3-972. 函数 rcu_hxtal_clock_monitor_disable

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

例如:

```
/* disable the HXTAL clock monitor */  
  
rcu_hxtal_clock_monitor_disable();
```

函数 rcu_lxtal_clock_monitor_enable

函数rcu_lxtal_clock_monitor_enable描述见下表:

表 3-973. 函数 rcu_lxtal_clock_monitor_enable

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

例如:

```
/* enable the 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 3-974. 函数 rcu_lxtal_clock_monitor_disable

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

例如：

```
/* disable the LXTAL clock monitor */
```

```
rcu_lxtal_clock_monitor_disable();
```

函数 rcu_clock_freq_get

函数rcu_clock_freq_get描述见下表：

表 3-975. 函数 rcu_clock_freq_get

函数名称	rcu_clock_freq_get
函数原形	uint32_t rcu_clock_freq_get(rcu_clock_freq_enum clock);
功能描述	获取系统、总线以及外设时钟频率
先决条件	-
被调用函数	-
输入参数{in}	
clock	要获取的时钟频率，具体参考 表3-904. 枚举类型rcu_clock_freq_enum
输出参数{out}	
-	-
返回值	
uint32_t	系统时钟/AHB时钟/APB1时钟/APB2时钟/APB3时钟/APB4时钟/PLL时钟 /USART时钟频率

例如：

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

函数 rcu_flag_get

函数rcu_flag_get描述见下表:

表 3-976. 函数 rcu_flag_get

函数名称	rcu_flag_get
函数原形	FlagStatus rcu_flag_get(rcu_flag_enum flag);
功能描述	获取时钟稳定和外设复位标志
先决条件	-
被调用函数	-
输入参数{in}	
flag	时钟稳定和外设复位标志, 参考 表3-899. 枚举类型rcu_flag_enum
输出参数{out}	
-	-
返回值	
-	SET或RESET

例如:

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

函数 rcu_all_reset_flag_clear

函数rcu_all_reset_flag_clear描述见下表:

表 3-977. 函数 rcu_all_reset_flag_clear

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

例如：

```
/* clear all the reset flag */  
  
rcu_all_reset_flag_clear();
```

函数 rcu_interrupt_enable

函数rcu_interrupt_enable描述见下表：

表 3-978. 函数 rcu_interrupt_enable

函数名称	rcu_interrupt_enable
函数原形	void rcu_interrupt_enable(rcu_int_enum interrupt);
功能描述	使能时钟稳定中断
先决条件	-
被调用函数	-
输入参数{in}	
interrupt	时钟稳定中断，具体参考 表3-902. 枚举类型rcu_int_enum
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* enable the HXTAL stabilization interrupt */  
  
rcu_interrupt_enable(RCU_INT_HXTALSTB);
```

函数 rcu_interrupt_disable

函数rcu_interrupt_disable描述见下表：

表 3-979. 函数 rcu_interrupt_disable

函数名称	rcu_interrupt_disable
函数原形	void rcu_interrupt_disable(rcu_int_enum interrupt);
功能描述	禁能时钟稳定中断
先决条件	-
被调用函数	-
输入参数{in}	
interrupt	时钟稳定中断，具体参考 表3-902. 枚举类型rcu_int_enum
输出参数{out}	
-	-
返回值	
-	-

例如：


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

```
rcu_interrupt_disable(RCU_INT_HXTALSTB);
```

函数 rcu_interrupt_flag_get

函数rcu_interrupt_flag_get描述见下表：

表 3-980. 函数 rcu_interrupt_flag_get

函数名称	rcu_interrupt_flag_get
函数原形	FlagStatus rcu_interrupt_flag_get(rcu_int_flag_enum int_flag);
功能描述	获取时钟稳定中断和时钟阻塞中断标志
先决条件	-
被调用函数	-
输入参数{in}	
int_flag	中断以及CKM标志，参考 表3-902. 枚举类型rcu_int_enum
输出参数{out}	
-	-
返回值	
FlagStatus	SET或RESET

例如：

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

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

函数 rcu_interrupt_flag_clear

函数rcu_interrupt_flag_clear描述见下表：

表 3-981. 函数 rcu_interrupt_flag_clear

函数名称	rcu_interrupt_flag_clear
函数原形	void rcu_interrupt_flag_clear(rcu_int_flag_clear_enum int_flag_clear);
功能描述	清除中断标志和时钟阻塞中断标志
先决条件	-
被调用函数	-
输入参数{in}	
int_flag_clear	时钟稳定和阻塞中断标志清除，参考 表3-901. 枚举类型rcu_int_flag_clear_enum
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* clear the interrupt HXTAL stabilization interrupt flag */
rcu_interrupt_flag_clear(RCU_INT_FLAG_HXTALSTB_CLR);
```

3.29. 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[3.29.1](#)描述了RTC的寄存器列表，章节[3.29.2](#)对RTC库函数进行说明。

3.29.1. 外设寄存器描述

RTC寄存器列表如下表所示：

表 3-982. RTC 寄存器

寄存器名称	寄存器描述
RTC_TIME	RTC时间寄存器
RTC_DATE	RTC日期寄存器
RTC_CTL	RTC控制寄存器
RTC_STAT	RTC状态寄存器
RTC_PSC	RTC预分频寄存器
RTC_WUT	RTC唤醒定时器寄存器
RTC_ALRM0TD	RTC闹钟0时间日期寄存器
RTC_ALRM1TD	RTC闹钟1时间日期寄存器
RTC_WPK	RTC写保护钥匙寄存器
RTC_SS	RTC亚秒寄存器
RTC_SHIFTCTL	RTC移位控制寄存器
RTC_TTS	RTC时间戳时间寄存器
RTC_DTS	RTC时间戳日期寄存器
RTC_SSTS	RTC时间戳亚秒寄存器
RTC_HRFC	RTC高精度频率补偿寄存器
RTC_TAMP	RTC侵入寄存器
RTC_ALRM0SS	RTC闹钟0亚秒寄存器
RTC_ALRM1SS	RTC闹钟1亚秒寄存器
RTC_CFG	RTC配置寄存器
RTC_BKP0	RTC备份域寄存器0
RTC_BKP1	RTC备份域寄存器1
RTC_BKP2	RTC备份域寄存器2
RTC_BKP3	RTC备份域寄存器3
RTC_BKP4	RTC备份寄存器4
RTC_BKP5	RTC备份寄存器5

寄存器名称	寄存器描述
RTC_BKP6	RTC备份寄存器6
RTC_BKP7	RTC备份寄存器7
RTC_BKP8	RTC备份寄存器8
RTC_BKP9	RTC备份寄存器9
RTC_BKP10	RTC备份寄存器10
RTC_BKP11	RTC备份寄存器11
RTC_BKP12	RTC备份寄存器12
RTC_BKP13	RTC备份寄存器13
RTC_BKP14	RTC备份寄存器14
RTC_BKP15	RTC备份寄存器15
RTC_BKP16	RTC备份寄存器16
RTC_BKP17	RTC备份寄存器17
RTC_BKP18	RTC备份寄存器18
RTC_BKP19	RTC备份寄存器19
RTC_BKP20	RTC备份寄存器20
RTC_BKP21	RTC备份寄存器21
RTC_BKP22	RTC备份寄存器22
RTC_BKP23	RTC备份寄存器23
RTC_BKP24	RTC备份寄存器24
RTC_BKP25	RTC备份寄存器25
RTC_BKP26	RTC备份寄存器26
RTC_BKP27	RTC备份寄存器27
RTC_BKP28	RTC备份寄存器28
RTC_BKP29	RTC备份寄存器29
RTC_BKP30	RTC备份寄存器30
RTC_BKP31	RTC备份寄存器31

3.29.2. 外设库函数描述

RTC库函数列表如下表所示：

表 3-983. RTC 库函数

库函数名称	库函数描述
rtc_deinit	复位大多数RTC寄存器
rtc_init	初始化RTC寄存器
rtc_init_mode_enter	进入RTC初始化模式
rtc_init_mode_exit	退出RTC初始化模式
rtc_register_sync_wait	等待直到RTC_TIME和RTC_DATE寄存器与APB时钟同步，并且阴影寄存器被更新
rtc_current_time_get	获取当前的时间和日期
rtc_subsecond_get	获取当前的亚秒值
rtc_alarm_config	配置RTC闹钟

库函数名称	库函数描述
rtc_alarm_subsecond_config	配置RTC闹钟的亚秒值
rtc_alarm_get	获取RTC闹钟
rtc_alarm_subsecond_get	获取RTC闹钟亚秒值
rtc_alarm_enable	使能RTC 闹钟
rtc_alarm_disable	失能RTC 闹钟
rtc_timestamp_enable	使能RTC 时间戳
rtc_timestamp_disable	失能RTC时间戳
rtc_timestamp_get	获取RTC时间戳时间和日期
rtc_timestamp_internalevent_config	RTC时间戳内部事件配置
rtc_timestamp_subsecond_get	获取RTC时间戳亚秒值
rtc_tamper_enable	使能RTC侵入检测
rtc_tamper_disable	失能RTC侵入检测
rtc_output_pin_select	配置RTC输出引脚
rtc_alarm_output_config	配置RTC闹钟输出
rtc_calibration_output_config	配置RTC校准输出
rtc_hour_adjust	通过在当前时间上增加或者减少一个小时来适应夏令时和冬令时
rtc_second_adjust	调整RTC当前时间的秒或亚秒值
rtc_bypass_shadow_enable	使能RTC影子寄存器
rtc_bypass_shadow_disable	失能RTC影子寄存器
rtc_refclock_detection_enable	使能RTC参考时钟检测功能
rtc_refclock_detection_disable	失能RTC参考时钟检测功能
rtc_wakeup_enable	使能RTC自动唤醒功能
rtc_wakeup_disable	失能RTC自动唤醒功能
rtc_wakeup_clock_set	设置RTC自动唤醒定时器时钟
rtc_wakeup_timer_set	设置自动唤醒定时器值
rtc_wakeup_timer_get	获取自动唤醒定时器值
rtc_smooth_calibration_config	配置RTC平滑校准
rtc_interrupt_enable	使能RTC指定的中断
rtc_interrupt_disable	失能RTC指定中断
rtc_flag_get	获取指定中断标志位
rtc_flag_clear	清除指定中断标志位

结构体 rtc_parameter_struct

表 3-984. 结构体 rtc_parameter_struct

成员名称	功能描述
year	RTC年份值：0x0 - 0x99（BCD格式）
month	RTC月份值（BCD格式）
date	RTC日期值：0x1 - 0x31（BCD格式）
day_of_week	RTC星期值（BCD格式）

hour	RTC 小时值: 0x1 - 0x12 (BCD格式) or 0x0 - 0x23 (BCD格式)
minute	RTC分钟值: 0x0 - 0x59 (BCD格式)
second	RTC秒值: 0x0 - 0x59 (BCD格式)
factor_asyn	RTC一步分频值: 0x0 - 0x7F
factor_syn	RTC同步分频值: 0x0 - 0x7FFF
am_pm	RTC AM/PM值
display_format	RTC时间格式

结构体 `rtc_alarm_struct`

表 3-985. 结构体 `rtc_alarm_struct`

成员名称	功能描述
alarm_mask	RTC闹钟屏蔽
weekday_or_date	指定RTC闹钟是日期还是星期几
alarm_day	RTC闹钟日期或者星期几的值 (BCD格式)
alarm_hour	RTC闹钟小时值: 0x1 - 0x12 (BCD格式) 或 0x0 - 0x23 (BCD格式)
alarm_minute	RTC闹钟分钟值: 0x0 - 0x59 (BCD格式)
alarm_second	RTC闹钟秒数值: 0x0 - 0x59 (BCD格式)
am_pm	RTC闹钟AM/PM数值

结构体 `rtc_timestamp_struct`

表 3-986. 结构体 `rtc_timestamp_struct`

成员名称	功能描述
timestamp_month	RTC时间戳月份值
timestamp_date	RTC 时间戳日期值: 0x1 - 0x31 (BCD格式)
timestamp_day	RTC时间戳星期值 (BCD格式)
timestamp_hour	RTC 时间戳小时值: 0x1 - 0x12 (BCD格式) 或 0x0 - 0x23 (BCD格式)
timestamp_minute	RTC时间戳分钟值: 0x0 - 0x59 (BCD格式)
timestamp_second	RTC时间戳秒数值: 0x0 - 0x59 (BCD格式)
am_pm	RTC时间戳AM/PM数值

结构体 `rtc_tamper_struct`

表 3-987. 结构体 `rtc_tamper_struct`

成员名称	功能描述
tamper_source	RTC侵入检测源
tamper_trigger	RTC侵入事件检测触发沿
tamper_filter	RTC 侵入事件检测在电平检测期间需要的连续采样次数
tamper_sample_frequency	RTC侵入事件电平模式检测的采样频率
tamper_precharge_enable	RTC在电压电平检测期间的预充电功能

tamper_precharge_time	RTC侵入事件电平检测采样预充电时间，如果预充电功能使能
tamper_with_timestamp	RTC侵入事件触发时间戳

函数 rtc_deinit

函数rtc_deinit描述见下表：

表 3-988. 函数 rtc_deinit

函数名称	rtc_deinit
函数原型	ErrStatus rtc_deinit(void);
功能描述	复位大多数RTC寄存器
先决条件	-
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	
ErrStatus	ERROR或SUCCESS

例如：

```
/* reset most of the RTC registers*/
```

```
ErrStatus error_status = rtc_deinit();
```

函数 rtc_init

函数rtc_init描述见下表：

表 3-989. 函数 rtc_init

函数名称	rtc_init
函数原型	ErrStatus rtc_init(rtc_parameter_struct* rtc_initpara_struct);
功能描述	初始化RTC寄存器
先决条件	-
被调用函数	-
输入参数{in}	
rtc_initpara_struct	初始化结构体，结构体成员参考 表3-984. 结构体rtc_parameter_struct
输出参数{out}	
-	-
返回值	
ErrStatus	ERROR或SUCCESS

例如：

```

/* initialize RTC registers */

rtc_parameter_struct rtc_initpara;

rtc_interrupt_disable(RTC_INT_SECOND);

rtc_initpara.factor_asyn = prescaler_a;

rtc_initpara.factor_syn = prescaler_s;

rtc_initpara.year = 0x16;

rtc_initpara.day_of_week = RTC_SATURDAY;

rtc_initpara.month = RTC_APR;

rtc_initpara.date = 0x30;

rtc_initpara.display_format = RTC_24HOUR;

rtc_initpara.am_pm = RTC_AM;

rtc_init(&rtc_initpara);

```

函数 rtc_init_mode_enter

函数rtc_init_mode_enter描述见下表：

表 3-990. 函数 rtc_init_mode_enter

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

例如：

```

/*enter RTC init mode*/

ErrStatus error_status = rtc_init_mode_enter();

```

函数 rtc_init_mode_exit

函数rtc_init_mode_exit描述见下表：

表 3-991. 函数 `rtc_init_mode_exit`

函数名称	<code>rtc_init_mode_exit</code>
函数原型	<code>void rtc_init_mode_exit(void);</code>
功能描述	退出RTC初始化模式
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 init mode*/
```

```
rtc_init_mode_exit();
```

函数 `rtc_register_sync_wait`

函数`rtc_register_sync_wait`描述见下表：

表 3-992. 函数 `rtc_register_sync_wait`

函数名称	<code>rtc_register_sync_wait</code>
函数原型	<code>ErrStatus rtc_register_sync_wait(void);</code>
功能描述	等待直到RTC_TIME和RTC_DATE寄存器与APB时钟同步，并且阴影寄存器被更新
先决条件	-
被调用函数	-
输入参数{in}	
-	-
输入参数{in}	
-	-
返回值	
ErrStatus	ERROR或SUCCESS

例如：

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

```
ErrStatus error_status = rtc_register_sync_wait();
```

函数 `rtc_current_time_get`

函数`rtc_current_time_get`描述见下表：

表 3-993. 函数 `rtc_current_time_get`

函数名称	<code>rtc_current_time_get</code>
函数原型	<code>void rtc_current_time_get(rtc_parameter_struct* rtc_initpara_struct);</code>
功能描述	获取当前的时间和日期
先决条件	-
被调用函数	-
输入参数{in}	
-	-
输出参数{out}	
<code>rtc_initpara_struct</code>	初始化结构体，结构体成员参考 表3-984. 结构体 <code>rtc_parameter_struct</code>
返回值	
-	-

例如：

```
/*get current time and date*/

rtc_parameter_struct rtc_initpara_struct;

rtc_current_time_get(&rtc_initpara_struct);
```

函数 `rtc_subsecond_get`

函数`rtc_subsecond_get`描述见下表：

表 3-994. 函数 `rtc_subsecond_get`

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

例如：

```
/*get current subsecond value*/

uint32_t sub_second = rtc_subsecond_get();
```

函数 `rtc_alarm_config`

函数`rtc_alarm_config`描述见下表：

表 3-995. 函数 `rtc_alarm_config`

函数名称	<code>rtc_alarm_config</code>
函数原型	<code>void rtc_alarm_config(uint8_t rtc_alarm, rtc_alarm_struct *rtc_alarm_time);</code>
功能描述	配置RTC闹钟
先决条件	-
被调用函数	-
输入参数{in}	
<code>rtc_alarm</code>	RTC_ALARM0或者RTC_ALARM1
输入参数{in}	
<code>rtc_alarm_time</code>	闹钟结构体，结构体成员参考 表3-985. 结构体 <code>rtc_alarm_struct</code>
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/*rtc_alarm_config*/
```

```
rtc_alarm_struct rtc_alarm_time;
```

```
rtc_alarm_config(RTC_ALARM0,&rtc_alarm_time);
```

函数 `rtc_alarm_subsecond_config`

函数`rtc_alarm_subsecond_config`描述见下表：

表 3-996. 函数 `rtc_alarm_subsecond_config`

函数名称	<code>rtc_alarm_subsecond_config</code>
函数原型	<code>void rtc_alarm_subsecond_config(uint8_t rtc_alarm, uint32_t mask_subsecond, uint32_t subsecond)</code>
功能描述	配置RTC闹钟的亚秒值
先决条件	-
被调用函数	-
输入参数{in}	
<code>rtc_alarm</code>	RTC_ALARM0或者RTC_ALARM1
输入参数{in}	
<code>mask_subsecond</code>	闹钟亚秒屏蔽位
<code>RTC_MASKSSC_0_14</code>	屏蔽闹钟亚秒设置
<code>RTC_MASKSSC_1_14</code>	屏蔽RTC_ALRM0SS_SSC[14:1]，SSC[0]位用于时间匹配
<code>RTC_MASKSSC_2_14</code>	屏蔽RTC_ALRM0SS_SSC[14:2]，SSC[1:0]位用于时间匹配
<code>RTC_MASKSSC_3_14</code>	屏蔽RTC_ALRM0SS_SSC[14:3]，SSC[2:0]位用于时间匹配
<code>RTC_MASKSSC_4_14</code>	屏蔽RTC_ALRM0SS_SSC[14:4]，SSC[3:0]位用于时间匹配
<code>RTC_MASKSSC_5_14</code>	屏蔽RTC_ALRM0SS_SSC[14:5]，SSC[4:0]位用于时间匹配
<code>RTC_MASKSSC_6_14</code>	屏蔽RTC_ALRM0SS_SSC[14:6]，SSC[5:0]位用于时间匹配

<i>RTC_MASKSSC_7_14</i>	屏蔽RTC_ALARM0SS_SSC[14:7], SSC[6:0]位用于时间匹配
<i>RTC_MASKSSC_8_14</i>	屏蔽RTC_ALARM0SS_SSC[14:8], SSC[7:0]位用于时间匹配
<i>RTC_MASKSSC_9_14</i>	屏蔽RTC_ALARM0SS_SSC[14:9], SSC[8:0]位用于时间匹配
<i>RTC_MASKSSC_10_14</i>	屏蔽RTC_ALARM0SS_SSC[14:10], SSC[9:0]位用于时间匹配
<i>RTC_MASKSSC_11_14</i>	屏蔽RTC_ALARM0SS_SSC[14:11], SSC[10:0]位用于时间匹配
<i>RTC_MASKSSC_12_14</i>	屏蔽RTC_ALARM0SS_SSC[14:12], SSC[11:0]位用于时间匹配
<i>RTC_MASKSSC_13_14</i>	屏蔽RTC_ALARM0SS_SSC[14:13], SSC[12:0]位用于时间匹配
<i>RTC_MASKSSC_14</i>	屏蔽RTC_ALARM0SS_SSC[14], SSC[13:0]位用于时间匹配
<i>RTC_MASKSSC_NONE</i>	无屏蔽, SSC[14:0]位用于时间匹配
输入参数{in}	
subsecond	闹钟亚秒值(0x000 - 0x7FFF)
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/*configure subsecond of RTC alarm*/
```

```
rtc_subsecond_config(RTC_ALARM0,RTC_MASKSSC_9_14, 0x7FFF);
```

函数 rtc_alarm_enable

函数rtc_alarm_enable描述见下表:

表 3-997. 函数 rtc_alarm_enable

函数名称	rtc_alarm_enable
函数原型	void rtc_alarm_enable(uint8_t rtc_alarm);
功能描述	使能RTC闹钟
先决条件	-
被调用函数	-
输入参数{in}	
rtc_alarm	RTC_ALARM0或者RTC_ALARM1
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/*enable RTC alarm*/
```

```
rtc_alarm_enable(RTC_ALARM0);
```

函数 rtc_alarm_disable

函数rtc_alarm_disable描述见下表：

表 3-998. 函数 rtc_alarm_disable

函数名称	rtc_alarm_disable
函数原型	ErrStatus rtc_alarm_disable(uint8_t rtc_alarm);
功能描述	失能RTC闹钟
先决条件	-
被调用函数	-
输入参数{in}	
rtc_alarm	RTC_ALARM0或者RTC_ALARM1
输出参数{out}	
-	-
返回值	
ErrStatus	ERROR或SUCCESS

例如：

```
/*disable RTC alarm*/
```

```
ErrStatus error_status = rtc_alarm_disable(RTC_ALARM0);
```

函数 rtc_alarm_get

函数rtc_alarm_get描述见下表：

表 3-999. 函数 rtc_alarm_get

函数名称	rtc_alarm_get
函数原型	void rtc_alarm_get(rtc_alarm_struct* rtc_alarm_time);
功能描述	获取RTC闹钟
先决条件	-
被调用函数	-
输入参数{in}	
rtc_alarm	RTC_ALARM0或者RTC_ALARM1
输出参数{out}	
rtc_alarm_time	闹钟结构体，结构体成员参考 表3-985. 结构体rtc_alarm_struct
返回值	
-	-

例如：

```
/* get RTC alarm */
```

```
rtc_alarm_struct rtc_alarm_time;
```

```
rtc_alarm_get(RTC_ALARM0,&rtc_alarm_time);
```

函数 rtc_alarm_subsecond_get

函数rtc_alarm_subsecond_get描述见下表:

表 3-1000. 函数 rtc_alarm_subsecond_get

函数名称	rtc_alarm_subsecond_get
函数原型	uint32_t rtc_alarm_subsecond_get(uint8_t rtc_alarm);
功能描述	获取RTC闹钟亚秒值
先决条件	-
被调用函数	-
输入参数{in}	
rtc_alarm	RTC_ALARM0 or RTC_ALARM1
输出参数{out}	
-	-
返回值	
uint32_t	RTC 闹钟亚秒值(0x0-0x3FFF)

例如:

```
/*get RTC alarm subsecond*/
```

```
uint32_t subsecond = rtc_alarm_subsecond_get(RTC_ALARM0);
```

函数 rtc_timestamp_enable

函数rtc_timestamp_enable描述见下表:

表 3-1001. 函数 rtc_timestamp_enable

函数名称	rtc_timestamp_enable
函数原型	void rtc_timestamp_enable(uint32_t edge);
功能描述	使能RTC时间戳
先决条件	-
被调用函数	-
输入参数{in}	
edge	选定哪种边沿触发时间戳检测
RTC_TIMESTAMP_RISING_EDGE	上升沿是时间戳事件有效检测沿
RTC_TIMESTAMP_FALLING_EDGE	下降沿是时间戳事件有效检测沿
输出参数{out}	
-	-
返回值	

例如：

```
/*enable RTC time-stamp*/
```

```
rtc_timestamp_enable(RTC_TIMESTAMP_RISING_EDGE);
```

函数 `rtc_timestamp_disable`

函数 `rtc_timestamp_disable` 描述见下表：

表 3-1002. 函数 `rtc_timestamp_disable`

函数名称	<code>rtc_timestamp_disable</code>
函数原型	<code>void rtc_timestamp_disable(void);</code>
功能描述	失能RTC时间戳
先决条件	-
被调用函数	-
输入参数{in}	
-	-
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/*disable RTC time-stamp*/
```

```
rtc_timestamp_disable();
```

函数 `rtc_timestamp_internalevent_config`

函数 `rtc_timestamp_internalevent_config` 描述见下表：

表 3-1003. 函数 `rtc_timestamp_internalevent_config`

函数名称	<code>rtc_timestamp_internalevent_config</code>
函数原型	<code>void rtc_timestamp_internalevent_config(uint32_t mode)</code>
功能描述	配置RTC时间戳内部事件
先决条件	-
被调用函数	-
输入参数{in}	
mode	配置内部事件还是外部事件被检测
<code>RTC_ITSEN_DISABLE</code>	失能RTC时间戳内部事件
<code>RTC_ITSEN_ENABLE</code>	使能RTC时间戳内部事件
输出参数{out}	
-	-

返回值	
-	-

例如：

```
/* disable RTC time-stamp internal event */
```

```
rtc_timestamp_internalevent_config(RTC_ITSEN_DISABLE);
```

函数 rtc_timestamp_get

函数rtc_timestamp_get描述见下表：

表 3-1004. 函数 rtc_timestamp_get

函数名称	rtc_timestamp_get
函数原型	void rtc_timestamp_get(rtc_timestamp_struct* rtc_timestamp);
功能描述	获取RTC时间戳时间和日期
先决条件	-
被调用函数	-
输入参数{in}	
-	-
输出参数{out}	
rtc_timestamp	时间戳结构体，结构体成员参考 表3-986. 结构体rtc_timestamp_struct
返回值	
-	-

例如：

```
/* get RTC timestamp time and date */
```

```
rtc_timestamp_struct rtc_timestamp;
```

```
rtc_timestamp_get(& rtc_timestamp);
```

函数 rtc_timestamp_subsecond_get

函数rtc_timestamp_subsecond_get描述见下表：

表 3-1005. 函数 rtc_timestamp_subsecond_get

函数名称	rtc_timestamp_subsecond_get
函数原型	uint32_t rtc_timestamp_subsecond_get(void);
功能描述	获取RTC时间戳亚秒值
先决条件	-
被调用函数	-
输入参数{in}	
-	-
输出参数{out}	
-	-

返回值	
uint32_t	RTC时间戳亚秒值

例如：

```
/* get RTC time-stamp subsecond */
```

```
uint32_t subsecond = rtc_timestamp_subsecond_get();
```

函数 rtc_tamper_enable

函数rtc_tamper_enable描述见下表：

表 3-1006. 函数 rtc_tamper_enable

函数名称	rtc_tamper_enable
函数原型	void rtc_tamper_enable(rtc_tamper_struct* rtc_tamper);
功能描述	使能RTC侵入检测
先决条件	-
被调用函数	-
输入参数{in}	
rtc_tamper	tamper化结构体，结构体成员参考 表3-987. 结构体rtc_tamper_struct
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* enable RTC tamper */
```

```
rtc_tamper_struct rtc_tamper
```

```
rtc_tamper_enable(& rtc_tamper);
```

函数 rtc_tamper_disable

函数rtc_tamper_disable描述见下表：

表 3-1007. 函数 rtc_tamper_disable

函数名称	rtc_tamper_disable
函数原型	void rtc_tamper_disable(uint32_t source);
功能描述	失能RTC侵入检测
先决条件	-
被调用函数	-
输入参数{in}	
source	选定被失能的侵入检测来源
RTC_TAMPER0	RTC 侵入检测0
RTC_TAMPER1	RTC 侵入检测1

输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* disable RTC tamper0 */
```

```
rtc_tamper_disable(RTC_TAMPER0);
```

函数 rtc_output_pin_select

函数rtc_output_pin_select描述见下表：

表 3-1008. 函数 rtc_output_pin_select

函数名称	rtc_output_pin_select
函数原型	void rtc_output_pin_select(uint32_t outputpin);
功能描述	选择RTC输出引脚
先决条件	-
被调用函数	-
输入参数{in}	
outputpin	选择RTC输出引脚
RTC_OUT_PC13	RTC输出引脚为PC13
RTC_OUT_PB2	RTC输出引脚为PB2
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* specify the rtc output pin is PC13 */
```

```
rtc_output_pin_select(RTC_OUT_PC13);
```

函数 rtc_alarm_output_config

函数rtc_alarm_output_config描述见下表：

表 3-1009. 函数 rtc_alarm_output_config

函数名称	rtc_alarm_output_config
函数原型	void rtc_alarm_output_config(uint32_t source, uint32_t mode);
功能描述	配置RTC闹钟输出源
先决条件	-
被调用函数	-
输入参数{in}	

source	specify signal to output
<i>RTC_ALARM0_HIGH</i>	当alarm0标志置位，输出引脚为高电平
<i>RTC_ALARM0_LOW</i>	当alarm0标志置位，输出引脚为低电平
<i>RTC_ALARM1_HIGH</i>	当alarm1标志置位，输出引脚为高电平
<i>RTC_ALARM1_LOW</i>	当alarm1标志置位，输出引脚为低电平
<i>RTC_WAKEUP_HIGH</i>	当唤醒标志置位，输出引脚为高电平
<i>RTC_WAKEUP_LOW</i>	当唤醒标志置位，输出引脚为低电平
输入参数{in}	
mode	当输出闹钟信号或者唤醒信号时指定输出引脚的模式
<i>RTC_ALARM_OUTPUT_T_OD</i>	开漏输出
<i>RTC_ALARM_OUTPUT_T_PP</i>	推挽输出
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* configure rtc alarm0 output source */
```

```
rtc_alarm_output_config(RTC_ALARM0_LOW, RTC_ALARM_OUTPUT_PP);
```

函数 rtc_calibration_output_config

函数rtc_calibration_output_config描述见下表：

表 3-1010. 函数 rtc_calibration_output_config

函数名称	rtc_calibration_output_config
函数原型	void rtc_calibration_output_config(uint32_t source);
功能描述	RTC校准输出配置
先决条件	-
被调用函数	-
输入参数{in}	
source	配置输出信号
<i>RTC_CALIBRATION_512HZ</i>	当LSE频率为32768Hz并且RTC_PSC为默认值，输出512Hz信号
<i>RTC_CALIBRATION_1HZ</i>	当LSE频率为32768Hz并且RTC_PSC为默认值，输出1Hz信号
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* when the LSE frequency is 32768Hz and the RTC_PSC  
is the default value, output 1Hz signal */  
rtc_calibration_output_config(RTC_CALIBRATION_1HZ);
```

函数 `rtc_hour_adjust`

函数`rtc_hour_adjust`描述见下表:

表 3-1011. 函数 `rtc_hour_adjust`

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

例如:

```
/* adjust the daylight saving time by adding one hour from the current time */  
rtc_hour_adjust(RTC_CTL_A1H);
```

函数 `rtc_second_adjust`

函数`rtc_second_adjust`描述见下表:

表 3-1012. 函数 `rtc_second_adjust`

函数名称	<code>rtc_second_adjust</code>
函数原型	<code>ErrStatus rtc_second_adjust(uint32_t add, uint32_t minus);</code>
功能描述	调整RTC当前时间的秒或亚秒值
先决条件	-
被调用函数	-
输入参数{in}	
add	在当前时间上增加1S或者不增加
<code>RTC_SHIFT_ADD1S_R ESET</code>	无影响

RTC_SHIFT_ADD1S_SET	在当前时间增加1秒
输入参数{in}	
minus	在当前是时间上减少的亚秒值(0x0 - 0x7FFF)
输出参数{out}	
-	-
返回值	
ErrStatus	ERROR或 SUCCESS

例如:

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

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

函数 rtc_bypass_shadow_enable

函数rtc_bypass_shadow_enable描述见下表:

表 3-1013. 函数 rtc_bypass_shadow_enable

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

例如:

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

```
rtc_bypass_shadow_enable();
```

函数 rtc_bypass_shadow_disable

函数rtc_bypass_shadow_disable描述见下表:

表 3-1014. 函数 rtc_bypass_shadow_disable

函数名称	rtc_bypass_shadow_disable
函数原型	void rtc_bypass_shadow_disable(void);
功能描述	失能RTC影子寄存器
先决条件	-

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

例如：

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

```
rtc_bypass_shadow_disable();
```

函数 rtc_refclock_detection_enable

函数rtc_refclock_detection_enable描述见下表：

表 3-1015. 函数 rtc_refclock_detection_enable

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

例如：

```
/* enable RTC reference clock detection function*/
```

```
ErrStatus error_status = rtc_refclock_detection_enable();
```

函数 rtc_refclock_detection_disable

函数rtc_refclock_detection_disable描述见下表：

表 3-1016. 函数 rtc_refclock_detection_disable

函数名称	rtc_refclock_detection_disable
函数原型	ErrStatus rtc_refclock_detection_disable(void);
功能描述	失能RTC参考时钟检测功能
先决条件	-
被调用函数	rtc_init_mode_enter/rtc_init_mode_exit

输入参数{in}	
-	-
输出参数{out}	
-	-
返回值	
ErrStatus	ERROR or SUCCESS

例如：

```
/* disableRTC reference clock detection function*/
```

```
ErrStatus error_status = rtc_refclock_detection_disable();
```

函数 rtc_wakeup_enable

函数rtc_wakeup_enable描述见下表：

表 3-1017. 函数 rtc_wakeup_enable

函数名称	rtc_wakeup_enable
函数原型	void rtc_wakeup_enable(void);
功能描述	使能RTC自动唤醒功能
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 RTC auto wakeup function*/
```

```
rtc_wakeup_enable();
```

函数 rtc_wakeup_disable

函数rtc_wakeup_disable描述见下表：

表 3-1018. 函数 rtc_wakeup_disable

函数名称	rtc_wakeup_disable
函数原型	ErrStatus rtc_wakeup_disable(void);
功能描述	失能RTC自动唤醒功能
先决条件	-
被调用函数	-
输入参数{in}	

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

例如:

```
/* disable RTC auto wakeup function*/
```

```
ErrStatus error_status = rtc_wakeup_disable();
```

函数 rtc_wakeup_clock_set

函数rtc_wakeup_clock_set描述见下表:

表 3-1019. 函数 rtc_wakeup_clock_set

函数名称	rtc_wakeup_clock_set
函数原型	ErrStatus rtc_wakeup_clock_set(uint8_t wakeup_clock);
功能描述	设置RTC自动唤醒定时器时钟
先决条件	-
被调用函数	-
输入参数{in}	
wakeup_clock	时钟选择
WAKEUP_RTCK_DIV 16	RTC时钟的16分频
WAKEUP_RTCK_DIV 8	RTC时钟的8分频
WAKEUP_RTCK_DIV 4	RTC时钟的4分频
WAKEUP_RTCK_DIV 2	RTC时钟的2分频
WAKEUP_CKSPRE	ck_spre(默认1Hz)时钟
WAKEUP_CKSPRE_2 EXP16	ck_spre(默认1Hz)时钟并且将唤醒计数器值增加 2^{16}
输出参数{out}	
-	-
返回值	
ErrStatus	ERROR 或 SUCCESS

例如:

```
/* RTC auto wakeup timer clock is ckspre */
```

```
ErrStatus error_status = rtc_wakeup_clock_set(WAKEUP_CKSPRE);
```

函数 rtc_wakeup_timer_set

函数rtc_wakeup_timer_set描述见下表:

表 3-1020. 函数 rtc_wakeup_timer_set

函数名称	rtc_wakeup_timer_set
函数原型	ErrStatus rtc_wakeup_timer_set(uint16_t wakeup_timer);
功能描述	设置RTC自动唤醒定时器值
先决条件	-
被调用函数	-
输入参数{in}	
wakeup_timer	定时器选择
uint16_t	0x0000-0xffff
输出参数{out}	
-	-
返回值	
ErrStatus	ERROR 或 SUCCESS

例如:

```
/* set wakeup timer value */
```

```
ErrStatus error_status = rtc_wakeup_timer_set(0XFFEE);
```

函数 rtc_wakeup_timer_get

函数rtc_wakeup_timer_get描述见下表:

表 3-1021. 函数 rtc_wakeup_timer_get

函数名称	rtc_wakeup_timer_get
函数原型	uint16_t rtc_wakeup_timer_get(void);
功能描述	获取唤醒定时器值
先决条件	-
被调用函数	-
输入参数{in}	
-	-
输出参数{out}	
-	-
返回值	
uint16_t	0-0xFFFF

例如:

```
/* get wakeup timer value*/
```

```
uint16_t wakeuptimer_value;
```

```
wakeuptimer_value = rtc_wakeup_timer_get();
```


函数 `rtc_smooth_calibration_config`

函数 `rtc_smooth_calibration_config` 描述见下表：

表 3-1022. 函数 `rtc_smooth_calibration_config`

函数名称	<code>rtc_smooth_calibration_config</code>
函数原型	<code>ErrStatus rtc_smooth_calibration_config(uint32_t window, uint32_t plus, uint32_t minus);</code>
功能描述	配置RTC平滑校准
先决条件	-
被调用函数	-
输入参数{in}	
window	校准窗口选择
<code>RTC_CALIBRATION_WINDOW_32S</code>	采用32S校准周期
<code>RTC_CALIBRATION_WINDOW_16S</code>	采用16S校准周期
<code>RTC_CALIBRATION_WINDOW_8S</code>	采用8S校准周期
输入参数{in}	
plus	增加脉冲
<code>RTC_CALIBRATION_PLUS_SET</code>	每2048个脉冲增加一个RTCCLK脉冲
<code>RTC_CALIBRATION_PLUS_RESET</code>	无影响
输入参数{in}	
minus	校准窗口校准周期RTCCLK脉冲屏蔽数（0x0-0x1FF）
输出参数{out}	
-	-
返回值	
ErrStatus	ERROR 或 SUCCESS

例如：

```
/* configure RTC smooth calibration*/
```

```
ErrStatus error_status;
```

```
error_status = rtc_smooth_calibration_config(RTC_CALIBRATION_WINDOW_32S,
RTC_CALIBRATION_PLUS_SET, 0x10);
```

函数 `rtc_interrupt_enable`

函数 `rtc_interrupt_enable` 描述见下表：

表 3-1023. 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指定的中断
先决条件	-
被调用函数	-
输入参数{in}	
interrupt	选定被使能的中断源
RTC_INT_TIMESTAMP	时间戳中断
RTC_INT_ALARM0	闹钟0中断
RTC_INT_ALARM1	闹钟1中断
RTC_INT_TAMP_ALL	所有侵入检测中断
RTC_INT_WAKEUP	唤醒定时器中断
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* enable specified RTC interrupt*/
rtc_interrupt_enable(RTC_INT_TAMP0);
```

函数 rtc_interrupt_disable

函数rtc_interrupt_disable描述见下表：

表 3-1024. 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指定中断
先决条件	-
被调用函数	-
输入参数{in}	
interrupt	选定被失能的RTC中断
RTC_INT_TIMESTAMP	时间戳中断
RTC_INT_ALARM0	闹钟0中断
RTC_INT_ALARM1	闹钟1中断
RTC_INT_TAMP_ALL	所有侵入检测中断
RTC_INT_WAKEUP	唤醒定时器中断
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* disable RTC ALARM interrupt */

rtc_interrupt_disable(RTC_INT_TAMP0);
```

函数 rtc_flag_get

函数rtc_flag_get描述见下表:

表 3-1025. 函数 rtc_flag_get

函数名称	rtc_flag_get
函数原型	FlagStatus rtc_flag_get(uint32_t flag);
功能描述	获取指定中断标志位
先决条件	-
被调用函数	-
输入参数{in}	
flag	选定被获取的中断标志
RTC_FLAG_SCP	平滑校准挂起标志
RTC_FLAG_TP2	tamper 2事件标志
RTC_FLAG_TP1	tamper 1事件标志
RTC_FLAG_TP0	tamper 0事件标志
RTC_FLAG_TSOVR	时间戳事件溢出标志
RTC_FLAG_TS	时间戳事件标志
RTC_FLAG_ALARM0	Alarm0发生标志
RTC_FLAG_ALARM1	Alarm1发生标志
RTC_FLAG_WT	唤醒事件标志
RTC_FLAG_INIT	进入初始化模式
RTC_FLAG_RSYN	寄存器同步标志
RTC_FLAG_YCM	年份配置标志
RTC_FLAG_SOP	移位功能操作挂起标志
RTC_FLAG_ALARM0 W	Alarm0配置可写标志
RTC_FLAG_ALARM1 W	Alarm1配置可写标志
RTC_FLAG_WTW	唤醒计数器可写标志
输出参数{out}	
-	-
返回值	
FlagStatus	SET 或 RESET

例如:

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

FlagStatus = rtc_flag_get(RTC_FLAG_TS);
```

函数 rtc_flag_clear

函数rtc_flag_clear描述见下表:

表 3-1026. 函数 rtc_flag_clear

函数名称	rtc_flag_clear
函数原型	void rtc_flag_clear(uint32_t flag);
功能描述	清除指定中断标志位
先决条件	-
被调用函数	-
输入参数{in}	
flag	要清除的中断标志位
RTC_FLAG_TP1	tamper 1事件标志
RTC_FLAG_TP0	tamper 0事件标志
RTC_FLAG_TSOVR	时间戳事件溢出标志
RTC_FLAG_TS	时间戳事件标志
RTC_FLAG_WT	唤醒标志
RTC_FLAG_ALARM0	Alarm0发生标志
RTC_FLAG_ALARM1	Alarm1发生标志
RTC_FLAG_RSYN	寄存器同步标志
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* cleartime-stamp event flag */
rtc_flag_clear(RTC_FLAG_TS);
```

3.30. 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[3.30.1](#)描述了SPI/I2S的寄存器列表，章节[3.30.2](#)对SPI/I2S库函数进行说明。

3.30.1. 外设寄存器说明

SPI/I2S寄存器列表如下表所示:

表 3-1027. 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_CFG0	配置寄存器0

寄存器名称	寄存器描述
SPI_CFG1	配置寄存器1
SPI_INT	中断寄存器
SPI_STAT	状态寄存器
SPI_STATC	中断/状态标志清除寄存器
SPI_TDATA	数据发送寄存器
SPI_RDATA	数据接收寄存器
SPI_CRCPOLY	CRC多项式寄存器
SPI_TCRC	发送CRC寄存器
SPI_RCRC	接收CRC寄存器
SPI_URDATA	下溢数据寄存器
SPI_I2SCTL	I2S控制寄存器
SPI_QCTL	四线SPI控制寄存器
SPI_RXDLYCK	接收时钟延迟寄存器

3.30.2. 外设库函数说明

SPI/I2S库函数列表如下表所示：

表 3-1028. SPI/I2S 库函数

库函数名称	库函数描述
spi_i2s_deinit	复位外设SPI/I2S
spi_struct_para_init	初始化SPI结构体中所有参数为默认值
spi_init	初始化外设SPI
spi_enable	使能外设SPI
spi_disable	禁能外设SPI
i2s_init	初始化外设I2S
i2s_psc_config	配置I2S预分频器
i2s_enable	使能外设I2S
i2s_disable	禁能外设I2S
spi_io_config	SPI MOSI脚与MISO脚替换
spi_nss_idleness_delay_set	设置SPI主机模式时NSS有效沿与数据开始传输或接收之间的延时
spi_data_frame_delay_set	设置SPI主机模式时数据帧之间延时
spi_master_receive_clock_delay_set	设置SPI主机接收时钟延迟
spi_slave_receive_clock_delay_set	设置SPI从机接收时钟延迟
spi_master_receive_clock_delay_clear	清除主机接收时钟延迟
spi_slave_receive_clock_delay_clear	清除从机接收时钟延迟
spi_nss_output_control	SPI主机模式时NSS引脚输出控制模式
spi_nss_polarity_set	设置SPI NSS输入/输出有效极性
spi_nss_output_enable	使能外设SPI NSS输出
spi_nss_output_disable	禁能外设SPI NSS输出

库函数名称	库函数描述
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SPI的DMA功能
spi_dma_disable	禁能外设SPI的DMA功能
spi_i2s_data_frame_size_config	配置外设SPI/I2S数据帧范围
spi_i2s_data_transmit	发送数据
spi_i2s_data_receive	接收数据
spi_bidirectional_transfer_config	配置外设SPI的数据传输方向
spi_master_transfer_start	主机启动传输
spi_current_data_num_config	配置当前传输数据量
spi_reload_data_num_config	配置重载数据量
spi_crc_polynomial_set	设置外设SPI的CRC多项式值
spi_crc_polynomial_get	获取外设SPI的CRC多项式值
spi_crc_length_config	配置SPI的CRC长度
spi_crc_on	打开外设SPI的CRC功能
spi_crc_off	关闭外设SPI的CRC功能
spi_crc_get	外设SPI获取CRC值
spi_crc_full_size_enable	使能全尺寸CRC多项式
spi_crc_full_size_disable	禁能全尺寸CRC多项式
spi_tcr_init_pattern	配置发送器CRC初始值
spi_rcrc_init_pattern	配置接收器CRC初始值
spi_ti_mode_enable	使能SPI TI模式
spi_ti_mode_disable	禁能SPI TI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_quad_io23_output_enable	使能SPI_IO2和SPI_IO3输出
spi_quad_io23_output_disable	禁能SPI_IO2和SPI_IO3输出
spi_underrun_operation	从机发送时检测到下溢后的处理
spi_underrun_config	配置从机发送时检测下溢
spi_underrun_data_config	配置从机模式传输下溢数据
spi_suspend_mode_config	配置主机在接收模式时被自动挂起
spi_suspend_request	SPI主机模式挂起请求
spi_related_ios_af_enable	使能SPI相关IO的AF配置功能
spi_related_ios_af_disable	禁能SPI相关IO的AF配置功能
spi_af_gpio_control	SPI相关IO的AF控制
spi_i2s_interrupt_enable	使能外设SPI/I2S中断
spi_i2s_interrupt_disable	禁能外设SPI/I2S中断
spi_i2s_interrupt_flag_get	获取外设SPI/I2S中断状态
spi_i2s_flag_get	获取外设SPI/I2S标志状态

库函数名称	库函数描述
spi_i2s_flag_clear	清除外设SPI/I2S标志状态
spi_i2s_rxfifo_plevel_get	获取外设SPI/I2S RxFIFO数据包级别
spi_i2s_remain_data_num_get	获取外设SPI/I2S TXSIZE区域中剩余的数据帧数
spi_fifo_threshold_level_set	设置SPI FIFO阈值
spi_word_access_enable	使能SPI字访问模式
spi_word_access_disable	禁能SPI字访问模式
spi_byte_access_enable	使能SPI字节访问模式
spi_byte_access_disable	禁能SPI字节访问模式

结构体 spi_parameter_struct

表 3-1029. 结构体 spi_parameter_struct

成员名称	功能描述
device_mode	主机或设备模式配置 (SPI_MASTER, SPI_SLAVE)
trans_mode	传输模式 (SPI_TRANSMODE_FULLDUPLEX, SPI_TRANSMODE_RECEIVEONLY, SPI_TRANSMODE_BDRECEIVE, SPI_TRANSMODE_BDTRANSMIT)
frame_size	数据帧格式配置 (SPI_DATASIZE_xBIT, x=4,5,...32)
nss	NSS由软件或硬件控制配置 (SPI_NSS_SOFT, SPI_NSS_HARD)
endian	大端或小端模式配置 (SPI_ENDIAN_MSB, SPI_ENDIAN_LSB)
clock_polarity_phase	相位和极性配置 (SPI_CK_PL_LOW_PH_1EDGE, SPI_CK_PL_HIGH_PH_1EDGE, SPI_CK_PL_LOW_PH_2EDGE, SPI_CK_PL_HIGH_PH_2EDGE)
prescale	预分频器配置 (SPI_PSC_n (n=2,4,8,16,32,64,128,256))

函数 spi_i2s_deinit

函数spi_i2s_deinit描述见下表：

表 3-1030. 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SPI/I2S
先决条件	-
被调用函数	rcu_periph_reset_enable / rcu_periph_reset_disable
输入参数{in}	
spi_periph	外设SPIx
SPIx	x=0,1...5

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 3-1031. 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	初始化SPI结构体中所有参数为默认值
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3-1029. 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 3-1032. 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SPI
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...5
输入参数{in}	
spi_struct	初始化结构体，结构体成员参考 表3-1029. 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 3-1033. 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SPI
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...5
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 3-1034. 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SPI
先决条件	-
被调用函数	-
输入参数{in}	
spi_periph	外设SPIx
SPIx	x=0,1...5
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 3-1035. 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I2S
先决条件	-
被调用函数	-
输入参数{in}	
spi_periph	外设I2Sx
SPIx	x=0,1,2,5
输入参数{in}	
i2s_mode	I2S运行模式
I2S_MODE_SLAVE TX	I2S从机发送模式

<i>I2S_MODE_SLAVE</i> <i>RX</i>	I2S从机接收模式
<i>I2S_MODE_MASTE</i> <i>RTX</i>	I2S主机发送模式
<i>I2S_MODE_MASTE</i> <i>RRX</i>	I2S主机接收模式
输入参数{in}	
i2s_standard	I2S标准选择
<i>I2S_STD_PHILLIPS</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_PCMSHO</i> <i>RT</i>	I2S PCM短帧标准
<i>I2S_STD_PCMLON</i> <i>G</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 I2S0 */
```

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

函数 i2s_psc_config

函数i2s_psc_config描述见下表：

表 3-1036. 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I2S预分频器
先决条件	-
被调用函数	rcu_clock_freq_get
输入参数{in}	
spi_periph	外设I2Sx
<i>SPIx</i>	x=0,1,2,5
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	I2S_MCK输出
<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 I2S0 prescaler */
```

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

函数 i2s_enable

函数i2s_enable描述见下表：

表 3-1037. 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I2S
先决条件	-
被调用函数	-
输入参数{in}	
spi_periph	外设I2Sx
SPIx	x=0,1,2,5
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* enable I2S0 */
i2s_enable(SPI0);
```

函数 i2s_disable

函数i2s_disable描述见下表：

表 3-1038. 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I2S
先决条件	-
被调用函数	-
输入参数{in}	
spi_periph	外设I2Sx
SPIx	x=0,1,2,5
输出参数{out}	
-	-
返回值	
-	-

例如：

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

```
i2s_disable(SPI0);
```

函数 spi_io_config

函数spi_io_config描述见下表：

表 3-1039. 函数 spi_io_config

函数名称	spi_io_config
函数原形	void spi_io_config(uint32_t spi_periph, uint32_t io_cfg);
功能描述	SPI MOSI脚与MISO脚替换
先决条件	-
被调用函数	-
输入参数{in}	
spi_periph	外设SPIx
SPIx	x=0,1...5
输入参数{in}	
io_cfg	MISO与MOSI替换配置
SPI_IO_SWAP	SPI MISO与MOSI脚功能发生替换
SPI_IO_NORMAL	SPI MISO与MOSI脚功能不发生替换
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* SPI0 MOSI and MISO pin swap */
```

```
spi_io_config(SPI0, SPI_IO_SWAP);
```

函数 spi_nss_idleness_delay_set

函数spi_nss_idleness_delay_set描述见下表：

表 3-1040. 函数 spi_nss_idleness_delay_set

函数名称	spi_nss_idleness_delay_set
函数原形	void spi_nss_idleness_delay_set(uint32_t spi_periph, uint32_t delay_cycle);
功能描述	设置SPI主机模式时NSS有效沿与数据开始传输或接收之间的延时
先决条件	-
被调用函数	-
输入参数{in}	
spi_periph	外设SPIx
SPIx	x=0,1...5
输入参数{in}	

delay_cycle	时钟延时
<i>SPI_NSS_IDLENESS_</i> <i>S_xCYCLE</i>	SPI主机模式时NSS有效沿与数据开始传输或接收之间存在x个时钟延时（x = 00,01,02,03,04,05,06,07,08,09,10,11,12,13,14,15）
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* set SPI0 1 cycle nss idleness delay */
spi_nss_idleness_delay_set(SPI0, SPI_NSS_IDLENESS_01CYCLE);
```

函数 spi_data_frame_delay_set

函数spi_data_frame_delay_set描述见下表：

表 3-1041. 函数 spi_data_frame_delay_set

函数名称	spi_data_frame_delay_set
函数原形	void spi_data_frame_delay_set(uint32_t spi_periph, uint32_t delay_cycle);
功能描述	设置SPI主机模式时数据帧之间延时
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...5
输入参数{in}	
delay_cycle	时钟延时
<i>SPI_DATA_IDLENESS_</i> <i>SS_xCYCLE</i>	SPI主机模式时数据帧之间存在x个时钟延时（x = 00,01,02,03,04,05,06,07,08,09,10,11,12,13,14,15）
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* set SPI0 1 cycle data frame delay */
spi_data_frame_delay_set(SPI0, SPI_DATA_IDLENESS_01CYCLE);
```

函数 spi_master_receive_clock_delay_set

函数spi_master_receive_clock_delay_set描述见下表：

表 3-1042. 函数 `spi_master_receive_clock_delay_set`

函数名称	<code>spi_master_receive_clock_delay_set</code>
函数原形	<code>void spi_master_receive_clock_delay_set(uint32_t spi_periph, uint32_t delay_unit);</code>
功能描述	设置SPI主机接收时钟延时
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...5
输入参数{in}	
delay_unit	时钟延时单元（0-0x1F）
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* set SPI0 master mode rx clock delay 16 units*/
```

```
spi_master_receive_clock_delay_set (SPI0, 0x0F);
```

函数 `spi_slave_receive_clock_delay_set`

函数`spi_slave_receive_clock_delay_set`描述见下表：

表 3-1043. 函数 `spi_slave_receive_clock_delay_set`

函数名称	<code>spi_slave_receive_clock_delay_set</code>
函数原形	<code>spi_slave_receive_clock_delay_set (uint32_t spi_periph, uint32_t delay_unit);</code>
功能描述	设置SPI从机接收时钟延时
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...5
输入参数{in}	
delay_unit	时钟延时单元（0-0x1F）
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* set SPI0 slave mode rx clock delay 16 units*/
```


spi_slave_receive_clock_delay_set (SPI0, 0x0F);

函数 spi_master_receive_clock_delay_clear

函数spi_master_receive_clock_delay_clear描述见下表:

表 3-1044. 函数 spi_master_receive_clock_delay_clear

函数名称	spi_master_receive_clock_delay_clear
函数原形	void spi_master_receive_clock_delay_clear (uint32_t spi_periph);
功能描述	清除主机接收时钟延迟
先决条件	-
被调用函数	-
输入参数{in}	
spi_periph	外设SPIx
SPIx	x=0,1...5
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* clear SPI0 master mode rx clock delay */
```

```
spi_master_receive_clock_delay_clear (SPI0);
```

函数 spi_slave_receive_clock_delay_clear

函数spi_slave_receive_clock_delay_clear描述见下表:

表 3-1045. 函数 spi_slave_receive_clock_delay_clear

函数名称	spi_slave_receive_clock_delay_clear
函数原形	void spi_slave_receive_clock_delay_clear (uint32_t spi_periph);
功能描述	清除从机接收时钟延迟
先决条件	-
被调用函数	-
输入参数{in}	
spi_periph	外设SPIx
SPIx	x=0,1...5
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* clear SPI0 slave mode rx clock delay */
```

spi_slave_receive_clock_delay_clear (SPI0);

函数 spi_nss_output_control

函数spi_nss_output_control描述见下表:

表 3-1046. 函数 spi_nss_output_control

函数名称	spi_nss_output_control
函数原形	void spi_nss_output_control(uint32_t spi_periph, uint32_t nss_ctl);
功能描述	设置SPI主机模式时NSS引脚输出控制模式
先决条件	-
被调用函数	-
输入参数{in}	
spi_periph	外设SPIx
SPIx	x=0,1...5
输入参数{in}	
nss_ctl	NSS引脚控制模式
SPI_NSS_HOLD_UNTIL_TRANS_END	SPI NSS保持有效电平直到数据传输完成
SPI_NSS_INVALID_PULSE	在SPI NSS每个数据帧之间插入无效脉冲（当MDFD[3:0] > 1）
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* set SPI0 NSS hold until trans end */
```

```
spi_nss_output_control(SPI0, SPI_NSS_HOLD_UNTIL_TRANS_END);
```

函数 spi_nss_polarity_set

函数spi_nss_polarity_set描述见下表:

表 3-1047. 函数 spi_nss_polarity_set

函数名称	spi_nss_polarity_set
函数原形	void spi_nss_polarity_set(uint32_t spi_periph, uint32_t polarity);
功能描述	设置SPI NSS输入/输出有效极性
先决条件	-
被调用函数	-
输入参数{in}	
spi_periph	外设SPIx
SPIx	x=0,1...5
输入参数{in}	

polarity	NSS引脚有效极性
<i>SPI_NSS_POLARITY_HIGH</i>	SPI NSS高电平有效
<i>SPI_NSS_POLARITY_LOW</i>	SPI NSS低电平有效
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* set SPI0 NSS high level is active */
```

```
spi_nss_polarity_set(SPI0, SPI_NSS_POLARITY_HIGH);
```

函数 spi_nss_output_enable

函数spi_nss_output_enable描述见下表：

表 3-1048. 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SPI NSS输出
先决条件	-
被调用函数	-
输入参数{in}	
spi_periph	外设SPIx
<i>SPIx</i>	x=0,1...5
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 3-1049. 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SPI NSS输出
先决条件	-
被调用函数	-
输入参数{in}	
spi_periph	外设SPIx
SPIx	x=0,1...5
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 3-1050. 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...5
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 3-1051. 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...5
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 3-1052. 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 spi_dma);
功能描述	使能外设SPI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...5
输入参数{in}	
spi_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 3-1053. 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 spi_dma);
功能描述	失能外设SPI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...5
输入参数{in}	
spi_dma	SPI DMA模式
SPI_DMA_TRANSMIT	SPI发送缓冲区DMA失能
SPI_DMA_RECEIVE	SPI接收缓冲区DMA失能
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_size_config

函数spi_i2s_data_frame_size_config描述见下表：

表 3-1054. 函数 spi_i2s_data_frame_size_config

函数名称	spi_i2s_data_frame_size_config
函数原形	void spi_i2s_data_frame_size_config(uint32_t spi_periph, uint32_t frame_size);
功能描述	配置外设SPI/I2S数据帧范围
先决条件	-
被调用函数	-
输入参数{in}	
spi_periph	外设SPIx
SPIx	x=0,1...5
输入参数{in}	

frame_size	SPI帧范围
<i>SPI_DATASIZE_xBIT</i> <i>T</i>	SPI x位数据帧范围 (x=4,5,...32)
输出参数{out}	
-	-
返回值	

例如:

```
/* configure SPI0/I2S0 data frame size is 16 bits */
```

```
spi_i2s_data_frame_size_config(SPI0, SPI_DATASIZE_16BIT);
```

函数 spi_i2s_data_transmit

函数spi_i2s_data_transmit描述见下表:

表 3-1055. 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, uint32_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
<i>SPIx</i>	x=0,1...5
输入参数{in}	
data	32位数据
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* SPI0 transmit data */
```

```
spi_i2s_data_transmit(SPI0, spi0_send_array[send_n]);
```

函数 spi_i2s_data_receive

函数spi_i2s_data_receive描述见下表:

表 3-1056. 函数 spi_i2s_data_receive

函数名称	spi_i2s_data_receive
函数原形	uint32_t spi_i2s_data_receive(uint32_t spi_periph);

功能描述	SPI接收数据
先决条件	-
被调用函数	-
输入参数{in}	
spi_periph	外设SPIx
<i>SPIx</i>	x=0,1...5
输出参数{out}	
-	-
返回值	
uint32_t	32位数据

例如:

```
/* SPI0 receive data */
```

```
spi0_receive_array[receive_n] = spi_i2s_data_receive(SPI0);
```

函数 **spi_bidirectional_transfer_config**

函数spi_bidirectional_transfer_config描述见下表:

表 3-1057. 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SPI的数据传输方向
先决条件	-
被调用函数	-
输入参数{in}	
spi_periph	外设SPIx
<i>SPIx</i>	x=0,1...5
输入参数{in}	
transfer_direction	SPI双向传输输出使能
<i>SPI_BIDIRECTIONAL_TRANSMIT</i>	SPI工作在只发送模式
<i>SPI_BIDIRECTIONAL_RECEIVE</i>	SPI工作在只接收模式
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* SPI0 works in transmit-only mode */
```


`spi_bidirectional_transfer_config(SPI0, SPI_BIDIRECTIONAL_TRANSMIT);`

函数 `spi_master_transfer_start`

函数 `spi_master_transfer_start` 描述见下表：

表 3-1058. 函数 `spi_master_transfer_start`

函数名称	<code>spi_master_transfer_start</code>
函数原形	<code>void spi_master_transfer_start(uint32_t spi_periph, uint32_t transfer_start);</code>
功能描述	主机启动传输
先决条件	-
被调用函数	-
输入参数{in}	
<code>spi_periph</code>	外设SPIx
<code>SPIx</code>	x=0,1...5
输入参数{in}	
<code>transfer_start</code>	传输启动
<code>SPI_TRANS_START</code>	主机开始传输，或者被自动挂起功能临时挂起
<code>SPI_TRANS_IDLE</code>	主机传输处于空闲状态
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* SPI0 master transfer start */
```

```
spi_master_transfer_start(SPI0, SPI_TRANS_START);
```

函数 `spi_current_data_num_config`

函数 `spi_current_data_num_config` 描述见下表：

表 3-1059. 函数 `spi_current_data_num_config`

函数名称	<code>spi_current_data_num_config</code>
函数原形	<code>void spi_current_data_num_config(uint32_t spi_periph, uint32_t current_num);</code>
功能描述	配置当前传输的数据量
先决条件	-
被调用函数	-
输入参数{in}	
<code>spi_periph</code>	外设SPIx
<code>SPIx</code>	x=0,1...5
输入参数{in}	
<code>current_num</code>	当前数据量（0-0xFFFF）

输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* SPI0 transfer current data number */
```

```
spi_current_data_num_config(SPI0, spi0_current_array[current_n]);
```

函数 spi_reload_data_num_config

函数spi_reload_data_num_config描述见下表：

表 3-1060. 函数 spi_reload_data_num_config

函数名称	spi_reload_data_num_config
函数原形	void spi_reload_data_num_config (uint32_t spi_periph, uint32_t reload_num);
功能描述	配置重载数据量
先决条件	-
被调用函数	-
输入参数{in}	
spi_periph	外设SPIx
SPIx	x=0,1...5
输入参数{in}	
reload_num	重载数据量（0-0xFFFF）
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* SPI0 transfer reload data number */
```

```
spi_reload_data_num_config(SPI0, spi0_reload_array[reload_n]);
```

函数 spi_crc_polynomial_set

函数spi_crc_polynomial_set描述见下表：

表 3-1061. 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, uint32_t crc_poly);
功能描述	设置外设SPI的CRC多项式值
先决条件	-
被调用函数	-

输入参数{in}	
spi_periph	外设SPIx
<i>SPIx</i>	x=0,1...5
输入参数{in}	
crc_poly	CRC多项式值
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* set SPI0 CRC polynomial */
spi_crc_polynomial_set(SPI0,CRC_VALUE);
```

函数 spi_crc_polynomial_get

函数spi_crc_polynomial_get描述见下表：

表 3-1062. 函数 spi_crc_polynomial_get

函数名称	spi_crc_polynomial_get
函数原形	uint32_t spi_crc_polynomial_get(uint32_t spi_periph);
功能描述	获取外设SPI的CRC多项式值
先决条件	-
被调用函数	-
输入参数{in}	
spi_periph	外设SPIx
<i>SPIx</i>	x=0,1...5
输出参数{out}	
-	-
返回值	
uint32_t	32位CRC多项式值（0-0xFFFFFFFF）

例如：

```
/* get SPI0 CRC polynomial */
uint32_t crc_val;
crc_val = spi_crc_polynomial_get(SPI0);
```

函数 spi_crc_length_config

函数spi_crc_length_config描述见下表：

表 3-1063. 函数 spi_crc_length_config

函数名称	spi_crc_length_config
------	-----------------------

函数原形	void spi_crc_length_config(uint32_t spi_periph, uint32_t crc_size);
功能描述	配置外设SPI的CRC长度
先决条件	-
被调用函数	-
输入参数{in}	
spi_periph	外设SPIx
SPIx	x=0,1...5
输入参数{in}	
crc_size	CRC长度
SPI_CRCSIZE_xBIT	SPI x位CRC长度 (x = 4,5,6...32)
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* config SPI0 CRC 16-bit length */
```

```
spi_crc_length_config(SPI0, SPI_CRCSIZE_16BIT);
```

函数 spi_crc_on

函数spi_crc_on描述见下表:

表 3-1064. 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SPI的CRC功能
先决条件	-
被调用函数	-
输入参数{in}	
spi_periph	外设SPIx
SPIx	x=0,1...5
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 3-1065. 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SPI的CRC功能
先决条件	-
被调用函数	-
输入参数{in}	
spi_periph	外设SPIx
SPIx	x=0,1...5
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* turn off SPI0 CRC function */
```

```
spi_crc_off(SPI0);
```

函数 **spi_crc_get**

函数spi_crc_get描述见下表:

表 3-1066. 函数 **spi_crc_get**

函数名称	spi_crc_get
函数原形	uint32_t spi_crc_get(uint32_t spi_periph, uint8_t crc);
功能描述	外设SPI获取CRC值
先决条件	-
被调用函数	-
输入参数{in}	
spi_periph	外设SPIx
SPIx	x=0,1...5
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	
uint32_t	32位CRC值 (0-0xFFFFFFFF)

例如：

```
/* get SPI0 CRC send value */

uint32_t crc_val;

crc_val = spi_crc_get(SPI0, SPI_CRC_TX);
```

函数 spi_crc_full_size_enable

函数spi_crc_full_size_enable描述见下表：

表 3-1067. 函数 spi_crc_full_size_enable

函数名称	spi_crc_full_size_enable
函数原形	void spi_crc_full_size_enable(uint32_t spi_periph);
功能描述	使能全尺寸CRC多项式
先决条件	-
被调用函数	-
输入参数{in}	
spi_periph	外设SPIx
SPIx	x=0,1...5
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* enable SPI0 crc full size */

spi_crc_full_size_enable(SPI0);
```

函数 spi_crc_full_size_disable

函数spi_crc_full_size_disable描述见下表：

表 3-1068. 函数 spi_crc_full_size_disable

函数名称	spi_crc_full_size_disable
函数原形	void spi_crc_full_size_disable (uint32_t spi_periph);
功能描述	禁能全尺寸CRC多项式
先决条件	-
被调用函数	-
输入参数{in}	
spi_periph	外设SPIx
SPIx	x=0,1...5
输出参数{out}	
-	-

返回值	
-	-

例如：

```
/* disable SPI0 crc full size */
```

```
spi_crc_full_size_disable (SPI0);
```

函数 spi_tcrc_init_pattern

函数spi_tcrc_init_pattern描述见下表：

表 3-1069. 函数 spi_tcrc_init_pattern

函数名称	spi_tcrc_init_pattern
函数原形	void spi_tcrc_init_pattern(uint32_t spi_periph, uint32_t init_pattern);
功能描述	配置发送器CRC初始值
先决条件	-
被调用函数	-
输入参数{in}	
spi_periph	外设SPIx
SPIx	x=0,1...5
输入参数{in}	
init_pattern	初始模式
SPI_TCRC_INIT_1	全1模式
SPI_TCRC_INIT_0	全0模式
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* config SPI0 TCRC all 1 initial pattern */
```

```
spi_tcrc_init_pattern(SPI0, SPI_TCRC_INIT_1);
```

函数 spi_rcrc_init_pattern

函数spi_rcrc_init_pattern描述见下表：

表 3-1070. 函数 spi_tcrc_init_pattern

函数名称	spi_tcrc_init_pattern
函数原形	void spi_rcrc_init_pattern(uint32_t spi_periph, uint32_t init_pattern);
功能描述	配置接收器CRC初始值
先决条件	-
被调用函数	-

输入参数{in}	
spi_periph	外设SPIx
<i>SPIx</i>	x=0,1...5
输入参数{in}	
init_pattern	初始模式
<i>SPI_RCRC_INIT_1</i>	全1模式
<i>SPI_RCRC_INIT_0</i>	全0模式
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* config SPI0 RCRC all 1 initial pattern */
spi_rcrc_init_pattern(SPI0, SPI_RCRC_INIT_1);
```

函数 spi_ti_mode_enable

函数spi_ti_mode_enable描述见下表：

表 3-1071. 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
<i>SPIx</i>	x=0,1...5
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 3-1072. 函数 `spi_ti_mode_disable`

函数名称	<code>spi_ti_mode_disable</code>
函数原形	<code>void spi_ti_mode_disable(uint32_t spi_periph);</code>
功能描述	禁用SPI TI模式
先决条件	-
被调用函数	-
输入参数{in}	
<code>spi_periph</code>	外设SPIx
<code>SPIx</code>	x=0,1...5
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* disable SPI0 TI mode */
spi_ti_mode_disable(SPI0);
```

函数 `spi_quad_enable`

函数`spi_quad_enable`描述见下表:

表 3-1073. 函数 `spi_quad_enable`

函数名称	<code>spi_quad_enable</code>
函数原形	<code>void spi_quad_enable(uint32_t spi_periph);</code>
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}	
<code>spi_periph</code>	外设SPIx
<code>SPIx</code>	x=3,4
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* enable SPI3 quad wire mode */
spi_quad_enable(SPI3);
```

函数 `spi_quad_disable`

函数`spi_quad_disable`描述见下表:

表 3-1074. 函数 spi_quad_disable

函数名称	spi_quad_disable
函数原形	void spi_quad_disable(uint32_t spi_periph);
功能描述	禁能四线SPI模式
先决条件	-
被调用函数	-
输入参数{in}	
spi_periph	外设SPIx
SPIx	x=3,4
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* disable SPI3 quad wire mode */
```

```
spi_quad_disable(SPI3);
```

函数 spi_quad_write_enable

函数spi_quad_write_enable描述见下表：

表 3-1075. 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=3,4
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* enable SPI3 quad wire write */
```

```
spi_quad_write_enable(SPI3);
```

函数 spi_quad_read_enable

函数spi_quad_read_enable描述见下表：

表 3-1076. 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=3,4
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* enable SPI3 quad wire read */
```

```
spi_quad_read_enable(SPI3);
```

函数 spi_quad_io23_output_enable

函数spi_quad_io23_output_enable描述见下表：

表 3-1077. 函数 spi_quad_io23_output_enable

函数名称	spi_quad_io23_output_enable
函数原形	void spi_quad_io23_output_enable(uint32_t spi_periph);
功能描述	使能SPI_IO2和SPI_IO3输出
先决条件	-
被调用函数	-
输入参数{in}	
spi_periph	外设SPIx
SPIx	x=3,4
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* enable SPI3 SPI_IO2 and SPI_IO3 pin output */
```

```
spi_quad_io23_output_enable(SPI3);
```

函数 spi_quad_io23_output_disable

函数spi_quad_io23_output_disable描述见下表：

表 3-1078. 函数 spi_quad_io23_output_disable

函数名称	spi_quad_io23_output_disable
函数原形	void spi_quad_io23_output_disable(uint32_t spi_periph);
功能描述	禁能SPI_IO2和SPI_IO3输出
先决条件	-
被调用函数	-
输入参数{in}	
spi_periph	外设SPIx
SPIx	x=3,4
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* disable SPI3 SPI_IO2 and SPI_IO3 pin output */
```

```
spi_quad_io23_output_disable(SPI3);
```

函数 spi_underrun_operation

函数spi_underrun_operation描述见下表：

表 3-1079. 函数 spi_underrun_operation

函数名称	spi_underrun_operation
函数原形	void spi_underrun_operation(uint32_t spi_periph, uint32_t ur_ope);
功能描述	从机发送时检测到下溢后的处理
先决条件	-
被调用函数	-
输入参数{in}	
spi_periph	外设SPIx
SPIx	x=0,1...5
输入参数{in}	
ur_ope	下溢操作
SPI_CONFIG_REGI STER_PATTERN	从机发送定义在SPI_URDATA寄存器中的常数
SPI_CONFIG_LAST _RECEIVED	从机发送从主机获取的最后一帧数据
SPI_CONFIG_LAST _TRANSMITTED	从机发送最后一次发送的数据帧（该数据帧存储在TxFIFO中）
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* slave underrun detected send a constant value defined by the SPI_URDATA register */
```

```
spi_underrun_operation(SPI0, SPI_CONFIG_REGISTER_PATTERN);
```

函数 spi_underrun_config

函数spi_underrun_config描述见下表：

表 3-1080. 函数 spi_underrun_config

函数名称	spi_underrun_config
函数原形	void spi_underrun_config(uint32_t spi_periph, uint32_t ur_cfg);
功能描述	配置从机发送时检测下溢
先决条件	-
被调用函数	-
输入参数{in}	
spi_periph	外设SPIx
SPIx	x=0,1...5
输入参数{in}	
ur_cfg	下溢配置
SPI_DETECT_BEGIN_DATA_FRAME	在数据帧开始时检测到下溢（无第一位保护）
SPI_DETECT_END_DATA_FRAME	在最后一个数据帧结束时检测到下溢
SPI_DETECT_BEGIN_ACTIVE_NSS	在NSS信号开始时检测到下溢
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* slave underrun detected at start of data frame (no bit 1 protection) */
```

```
spi_underrun_config(SPI0, SPI_DETECT_BEGIN_DATA_FRAME);
```

函数 spi_underrun_data_config

函数spi_underrun_data_config描述见下表：

表 3-1081. 函数 spi_underrun_data_config

函数名称	spi_underrun_data_config
函数原形	void spi_underrun_data_config(uint32_t spi_periph, uint32_t udata);
功能描述	配置从机模式传输下溢数据
先决条件	-

被调用函数	-
输入参数{in}	
spi_periph	外设SPIx
<i>SPIx</i>	x=0,1...5
输入参数{in}	
udata	SPI_URDATA下溢数据（0-0xFFFFFFFF）
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* config SPI0 underrun data at slave mode */
```

```
spi_underrun_data_config(SPI0, SPI0_URDATA);
```

函数 spi_suspend_mode_config

函数spi_suspend_mode_config描述见下表：

表 3-1082. 函数 spi_suspend_mode_config

函数名称	spi_suspend_mode_config
函数原形	void spi_suspend_mode_config(uint32_t spi_periph, uint32_t sus_mode);
功能描述	配置主机在接收模式时被自动挂起
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...5
输入参数{in}	
sus_mode	挂起模式
<i>SPI_AUTO_SUSPEND</i>	当上溢出现之前，当RxFIFO已满时，SPI数据流被挂起
<i>SPI_CONTINUOUS</i>	不论上溢是否发生，SPI的数据流和时钟都持续
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* config SPI0 auto suspend */
```

```
spi_suspend_mode_config(SPI0, SPI_AUTO_SUSPEND);
```

函数 spi_suspend_request

函数spi_suspend_request描述见下表：

表 3-1083. 函数 spi_suspend_request

函数名称	spi_suspend_request
函数原形	void spi_suspend_request(uint32_t spi_periph);
功能描述	SPI主机模式挂起请求
先决条件	-
被调用函数	-
输入参数{in}	
spi_periph	外设SPIx
SPIx	x=0,1...5
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* SPI0 master request suspend */
```

```
spi_suspend_request(SPI0);
```

函数 spi_related_ios_af_enable

函数spi_related_ios_af_enable描述见下表：

表 3-1084. 函数 spi_related_ios_af_enable

函数名称	spi_related_ios_af_enable
函数原形	void spi_related_ios_af_enable(uint32_t spi_periph);
功能描述	使能SPI相关IO的AF配置功能
先决条件	-
被调用函数	-
输入参数{in}	
spi_periph	外设SPIx
SPIx	x=0,1...5
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* enable spi0 related IOs AF */
```

```
spi_related_ios_af_enable(SPI0);
```

函数 **spi_related_ios_af_disable**

函数spi_related_ios_af_disable描述见下表:

表 3-1085. 函数 **spi_related_ios_af_enable**

函数名称	spi_related_ios_af_disable
函数原形	void spi_related_ios_af_disable(uint32_t spi_periph);
功能描述	禁能SPI相关IO的AF配置功能
先决条件	-
被调用函数	-
输入参数{in}	
spi_periph	外设SPIx
SPIx	x=0,1...5
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* disable spi0 related IOs AF */
spi_related_ios_af_disable(SPI0);
```

函数 **spi_af_gpio_control**

函数spi_af_gpio_control描述见下表:

表 3-1086. 函数 **spi_af_gpio_control**

函数名称	spi_af_gpio_control
函数原形	void spi_af_gpio_control(uint32_t spi_periph, uint32_t ctl);
功能描述	SPI相关IO的AF控制
先决条件	-
被调用函数	-
输入参数{in}	
spi_periph	外设SPIx
SPIx	x=0,1...5
输入参数{in}	
ctl	控制模式
SPI_GPIO_CONTR OL	外设总是控制相关的GPIOs
SPI_GPIO_FREE	外设禁止时不能控制GPIOs
输出参数{out}	
-	-
返回值	

-	-
---	---

例如:

```
/* SPI0 do not control GPIO when disabled */
```

```
spi_af_gpio_control(SPI0, SPI_GPIO_FREE);
```

函数 spi_i2s_interrupt_enable

函数spi_i2s_interrupt_enable描述见下表:

表 3-1087. 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SPI/I2S中断
先决条件	-
被调用函数	-
输入参数{in}	
spi_periph	外设SPIx
SPIx	x=0,1...5
输入参数{in}	
interrupt	SPI/I2S中断
SPI_I2S_INT_RP	RP中断
SPI_I2S_INT_TP	TP中断
SPI_I2S_INT_DP	DP中断
SPI_I2S_INT_ESTC	传输结束、挂起、TxFIFO清空中断
SPI_I2S_INT_TXF	传输已填充中断
SPI_I2S_INT_TXUR E	下溢错误中断
SPI_I2S_INT_RXO RE	上溢错误中断
SPI_I2S_INT_CRC ER	CRC错误中断
SPI_INT_FE	TI帧错误中断
SPI_I2S_INT_CON FE	SPI配置错误中断
SPI_I2S_INT_TXSE RF	TXSER重载中断
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* enable SPI0 crc error interrupt */
```

```
spi_i2s_interrupt_enable(SPI0, SPI_I2S_INT_CRCER);
```

函数 spi_i2s_interrupt_disable

函数spi_i2s_interrupt_disable描述见下表：

表 3-1088. 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	禁用外设SPI/I2S中断
先决条件	-
被调用函数	-
输入参数{in}	
spi_periph	外设SPIx
SPIx	x=0,1...5
输入参数{in}	
interrupt	SPI/I2S中断
SPI_I2S_INT_RP	RP中断
SPI_I2S_INT_TP	TP中断
SPI_I2S_INT_DP	DP中断
SPI_I2S_INT_ESTC	传输结束、挂起、TxFIFO清空中断
SPI_I2S_INT_TXF	传输已填充中断
SPI_I2S_INT_TXUR E	下溢错误中断
SPI_I2S_INT_RXO RE	上溢错误中断
SPI_I2S_INT_CRC ER	CRC错误中断
SPI_INT_FE	TI帧错误中断
SPI_I2S_INT_CON FE	SPI配置错误中断
SPI_I2S_INT_TXSE RF	TXSER重载中断
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* disable SPI0 crc error interrupt */
```

```
spi_i2s_interrupt_disable(SPI0, SPI_I2S_INT_CRCER);
```

函数 **spi_i2s_interrupt_flag_get**

函数 `spi_i2s_interrupt_flag_get` 描述见下表：

表 3-1089. 函数 **spi_i2s_interrupt_flag_get**

函数名称	<code>spi_i2s_interrupt_flag_get</code>
函数原形	<code>FlagStatus spi_i2s_interrupt_flag_get(uint32_t spi_periph, uint8_t interrupt);</code>
功能描述	获取外设SPI/I2S中断状态
先决条件	-
被调用函数	-
输入参数{in}	
spi_periph	外设SPIx
<i>SPIx</i>	x=0,1...5
输入参数{in}	
interrupt	SPI/I2S中断状态
<code>SPI_I2S_INT_FLAG</code> <code>_RP</code>	RP中断标志
<code>SPI_I2S_INT_FLAG</code> <code>_TP</code>	TP中断标志
<code>SPI_I2S_INT_FLAG</code> <code>_DP</code>	DP中断标志
<code>SPI_I2S_INT_FLAG</code> <code>_ET</code>	发送或接收结束中断标志
<code>SPI_I2S_INT_FLAG</code> <code>_TXF</code>	传输已填充中断标志
<code>SPI_I2S_INT_FLAG</code> <code>_TXURERR</code>	下溢错误中断标志
<code>SPI_I2S_INT_FLAG</code> <code>_RXORERR</code>	上溢错误中断标志
<code>SPI_I2S_INT_FLAG</code> <code>_CRCERR</code>	CRC错误中断标志
<code>SPI_I2S_INT_FLAG</code> <code>_FERR</code>	TI帧错误中断标志
<code>SPI_I2S_INT_FLAG</code> <code>_CONFERR</code>	SPI配置错误中断标志
<code>SPI_I2S_INT_FLAG</code> <code>_TXSERF</code>	TXSER重载中断标志
<code>SPI_I2S_INT_FLAG</code> <code>_SPD</code>	挂起中断标志
<code>SPI_I2S_INT_FLAG</code> <code>_TC</code>	TxFIFO清空中断标志
输出参数{out}	
-	-

返回值	
FlagStatus	SET 或 RESET

例如:

```
/* get SPI0 RP interrupt status */

if((spi_i2s_flag_get(SPI0, SPI_FLAG_RWNE) | (RESET != spi_i2s_interrupt_flag_get(SPI0,
SPI_I2S_INT_FLAG_RP))){

    RxBuffer1[RxCounter1++] = spi_i2s_data_receive(SPI0);

}
```

函数 spi_i2s_flag_get

函数spi_i2s_flag_get描述见下表:

表 3-1090. 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SPI/I2S标志状态
先决条件	-
被调用函数	-
输入参数{in}	
spi_periph	外设SPIx
SPIx	x=0,1...5
输入参数{in}	
flag	SPI/I2S标志状态
SPI_FLAG_RP	SPI RP标志
SPI_FLAG_TP	SPI TP标志
SPI_FLAG_DP	SPI DP标志
SPI_FLAG_ET	SPI发送或接收结束标志
SPI_FLAG_TXF	SPI TxFIFO已填充标志
SPI_FLAG_TXURERR	SPI传输下溢错误标志
SPI_FLAG_RXORERR	SPI接收上溢错误标志
SPI_FLAG_CRCERR	SPI CRC错误标志
SPI_FLAG_FERR	SPI TI帧格式错误标志
SPI_FLAG_CONFERR	SPI配置错误标志
SPI_FLAG_TXSERF	SPI TXSER已重载标志
SPI_FLAG_SPD	SPI挂起标志

<i>SPI_FLAG_TC</i>	SPI TxFIFO清空标志
<i>SPI_FLAG_RWNE</i>	SPI RxFIFO中字长非空标志
<i>I2S_FLAG_RP</i>	I2S RP标志
<i>I2S_FLAG_TP</i>	I2S TP标志
<i>I2S_FLAG_DP</i>	I2S DP标志
<i>I2S_FLAG_ET</i>	I2S发送或接收结束标志
<i>I2S_FLAG_TXF</i>	I2S TxFIFO已填充标志
<i>I2S_FLAG_TXURERRR</i>	I2S传输下溢错误标志
<i>I2S_FLAG_RXORERRR</i>	I2S接收上溢错误标志
输出参数{out}	
-	-
返回值	
FlagStatus	SET 或 RESET

例如：

```

/* get SPI0 RWNE flag status */

if((spi_i2s_flag_get(SPI0, SPI_FLAG_RWNE) | (RESET != spi_i2s_interrupt_flag_get(SPI0,
SPI_I2S_INT_FLAG_RP))){

    RxBuffer1[RxCounter1++] = spi_i2s_data_receive(SPI0);

}

```

函数 spi_i2s_flag_clear

函数spi_i2s_flag_clear描述见下表：

表 3-1091. 函数 spi_crc_error_clear

函数名称	spi_i2s_flag_clear
函数原形	void spi_i2s_flag_clear(uint32_t spi_periph, uint32_t flag);
功能描述	清除外设SPI/I2S标志状态
先决条件	-
被调用函数	-
输入参数{in}	
spi_periph	外设SPIx
SPIx	x=0,1...5
输入参数{in}	
flag	SPI/I2S标志状态
<i>SPI_STATC_ETC</i>	清除传输/接收结束标志
<i>SPI_STATC_TXFC</i>	清除TxFIFO传输填充标志
<i>SPI_STATC_TXURERRC</i>	清除传输下溢错误标志

<code>SPI_STATC_RXORERRC</code>	清除接收上溢错误标志
<code>SPI_STATC_CRCE</code> <code>RRC</code>	清除CRC错误标志
<code>SPI_STATC_FERR</code> <code>C</code>	清除SPI TI格式错误标志
<code>SPI_STATC_CONF</code> <code>ERRC</code>	清除配置错误标志
<code>SPI_STATC_TXSERF</code> <code>RFC</code>	清除TXSERF标志
<code>SPI_STATC_SPDC</code>	清除挂起标志
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* clear SPI0 CRC error flag status */
```

```
spi_i2s_flag_clear(SPI0, SPI_STATC_CRCERRC);
```

函数 `spi_i2s_rxfifo_plevel_get`

函数 `spi_i2s_rxfifo_plevel_get` 描述见下表：

表 3-1092. 函数 `spi_i2s_rxfifo_plevel_get`

函数名称	<code>spi_i2s_rxfifo_plevel_get</code>
函数原形	<code>uint32_t spi_i2s_rxfifo_plevel_get(uint32_t spi_periph);</code>
功能描述	获取外设SPI/I2S RxFIFO数据包级别
先决条件	-
被调用函数	-
输入参数{in}	
<code>spi_periph</code>	外设SPIx
<code>SPIx</code>	x=0,1...5
输出参数{out}	
-	-
返回值	
<code>uint32_t</code>	2位存储在RxFIFO中的数据帧数

例如：

```
/* get SPI0 RxFIFO packing data frame number */
```

```
uint32_t rxfifo_val;
```

```
rxfifo_val = spi_i2s_rxfifo_plevel_get (SPI0);
```

函数 spi_i2s_remain_data_num_get

函数spi_i2s_remain_data_num_get描述见下表:

表 3-1093. 函数 spi_i2s_remain_data_num_get

函数名称	spi_i2s_remain_data_num_get
函数原形	uint32_t spi_i2s_remain_data_num_get(uint32_t spi_periph);
功能描述	获取外设SPI/I2S TXSIZE区域中剩余的数据帧数
先决条件	-
被调用函数	-
输入参数{in}	
spi_periph	外设SPIx
SPIx	x=0,1...5
输出参数{out}	
-	-
返回值	
uint32_t	32位剩余数据帧数（0-0xFFFF）

例如:

```
/* get SPI0 TXSIZE value */
```

```
uint32_t txsize_val;
```

```
txsize_val = spi_i2s_remain_data_num_get(SPI0);
```

函数 spi_fifo_threshold_level_set

函数spi_fifo_threshold_level_set描述见下表:

表 3-1094. 函数 spi_fifo_threshold_level_set

函数名称	spi_fifo_threshold_level_set
函数原形	void spi_fifo_threshold_level_set(uint32_t spi_periph, uint32_t fifo_thl);
功能描述	设置SPI FIFO阈值
先决条件	-
被调用函数	-
输入参数{in}	
spi_periph	外设SPIx
SPIx	x=0,1...5
输入参数{in}	
fifo_thl	fifo阈值
SPI_FIFO_TH_xDATA	单个数据包中包含的数据帧数x（x = 01,02,03,04,05,06,07,08,09,10,11,12,13,14,15,16）
TA	
输出参数{out}	
-	-
返回值	

-	-
---	---

例如：

```
/* set SPI0 4-byte fifo threshold */
```

```
spi_fifo_threshold_level_set(SPI0, SPI_FIFO_TH_04DATA);
```

函数 **spi_word_access_enable**

函数 **spi_word_access_enable** 描述见下表：

表 3-1095. 函数 **spi_enable**

函数名称	spi_word_access_enable
函数原形	void spi_word_access_enable(uint32_t spi_periph);
功能描述	使能外设SPI字访问模式
先决条件	-
被调用函数	-
输入参数{in}	
spi_periph	外设SPIx
SPIx	x=0,1...5
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* enable SPI0 word access */
```

```
spi_word_access_enable(SPI0);
```

函数 **spi_word_access_disable**

函数 **spi_disable** 描述见下表：

表 3-1096. 函数 **spi_disable**

函数名称	spi_word_access_disable
函数原形	void spi_word_access_disable(uint32_t spi_periph);
功能描述	禁能外设SPI字访问模式
先决条件	-
被调用函数	-
输入参数{in}	
spi_periph	外设SPIx
SPIx	x=0,1...5
输出参数{out}	
-	-

返回值	
-	-

例如：

```
/* disable SPI0 word access */
```

```
spi_word_access_disable(SPI0);
```

函数 spi_byte_access_enable

函数spi_byte_access_enable描述见下表：

表 3-1097. 函数 spi_byte_access_enable

函数名称	spi_byte_access_enable
函数原形	void spi_byte_access_enable(uint32_t spi_periph);
功能描述	使能外设SPI字节访问模式
先决条件	-
被调用函数	-
输入参数{in}	
spi_periph	外设SPIx
SPIx	x=0,1...5
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* enable SPI0 byte access */
```

```
spi_byte_access_enable(SPI0);
```

函数 spi_byte_access_disable

函数spi_byte_access_disable描述见下表：

表 3-1098. 函数 spi_byte_access_disable

函数名称	spi_byte_access_disable
函数原形	void spi_byte_access_disable(uint32_t spi_periph);
功能描述	禁能外设SPI字节访问模式
先决条件	-
被调用函数	-
输入参数{in}	
spi_periph	外设SPIx
SPIx	x=0,1...5
输出参数{out}	

-	-
返回值	
-	-

例如:

```
/* disable SPI0 byte access */
```

```
spi_byte_access_disable(SPI0);
```

3.31. SYSCFG

章节[3.31.1](#)描述了SYSCFG的寄存器列表，章节[3.31.2](#)对SYSCFG库函数进行说明。

3.31.1. 外设寄存器说明

SYSCFG寄存器列表如下表所示:

表 3-1099. SYSCFG 寄存器

寄存器名称	寄存器描述
SYSCFG_PMCFG	外设模式配置寄存器
SYSCFG_EXTISS0	EXTI源选择寄存器0
SYSCFG_EXTISS1	EXTI源选择寄存器1
SYSCFG_EXTISS2	EXTI源选择寄存器2
SYSCFG_EXTISS3	EXTI源选择寄存器3
SYSCFG_LKCTL	锁定控制寄存器
SYSCFG_CPSCCTL	I/O补偿控制寄存器
SYSCFG_CPSCCFG	I/O补偿单元代码配置寄存器
SYSCFG_TIMERCI SEL0	TIMER输入选择寄存器0
SYSCFG_TIMERCI SEL1	TIMER输入选择寄存器1
SYSCFG_TIMERCI SEL2	TIMER输入选择寄存器2
SYSCFG_TIMERCI SEL3	TIMER输入选择寄存器3
SYSCFG_TIMERCI SEL4	TIMER输入选择寄存器4
SYSCFG_TIMERCI SEL5	TIMER输入选择寄存器5
SYSCFG_TIMERCI SEL6	TIMER输入选择寄存器6
SYSCFG_CPUICA	CPU ICACHE错误状态寄存器

寄存器名称	寄存器描述
C	
SYSCFG_CPUDCA C	CPU DCACHE错误状态寄存器
SYSCFG_FPUINTE N	FPU中断使能寄存器
SYSCFG_SRAMCF G0	SYSCFG SRAM配置寄存器0
SYSCFG_SRAMCF G1	SYSCFG SRAM配置寄存器1
SYSCFG_USERCF G	用户配置寄存器

3.31.2. 外设库函数说明

SYSCFG库函数列表如下表所示：

表 3-1100. SYSCFG 库函数

库函数名称	库函数描述
syscfg_deinit	复位SYSCFG寄存器
syscfg_i2c_fast_mode_plus_enable	使能I2Cx(x=0,1,2,3)或PBx(x=6,7,8,9)Fm+模式
syscfg_i2c_fast_mode_plus_disable	禁能I2Cx(x=0,1,2,3)或PBx(x=6,7,8,9)Fm+模式
syscfg_analog_switch_enable	模拟开关打开
syscfg_analog_switch_disable	模拟开关关闭
syscfg_enet_phy_interface_config	选择以太网PHY接口
syscfg_exti_line_config	配置GPIO引脚作为EXTI
syscfg_lockup_enable	使能SYSCFG锁定功能
syscfg_timer_input_source_select	选择TIMER输入源
syscfg_compensation_config	配置I/O补偿单元
syscfg_io_low_voltage_speed_optimization_enable	使能I/O速度优化，低电压下高速功能
syscfg_io_low_voltage_speed_optimization_disable	禁能I/O速度优化，低电压下高速功能
syscfg_pnmos_compensation_code_set	设置PMOS/NMOS补偿单元代码
syscfg_secure_sram_size_set	设置安全SRAM的大小
syscfg_secure_sram_size_get	获取安全SRAM的大小
syscfg_bootmode_get	获取启动方式
syscfg_tcm_wait_state_enable	使能TCM等待功能
syscfg_tcm_wait_state_disable	禁能TCM等待功能
syscfg_fpu_interrupt_enable	使能FPU中断
syscfg_fpu_interrupt_disable	禁能FPU中断
syscfg_compensation_flag_get	获取SYSCFG补偿单元标志

库函数名称	库函数描述
syscfg_cpu_cache_status_get	获取SYSCFG CPU CACHE状态
syscfg_brownout_reset_threshold_level_get	获取BOR掉电复位阈值电平

枚举类型 timer_channel_input_enum

表 3-1101. 枚举类型 timer_channel_input_enum

枚举名称	枚举描述
TIMER7_CIO_INPUT_TIMER7_CH0	选择CMP1输出作为TIMER7 CIO输入
TIMER7_CIO_INPUT_CMP1_OUT	选择TIMER7 CH0作为TIMER7 CIO输入
TIMER7_CI1_INPUT_TIMER7_CH1	选择TIMER7 CH1作为TIMER7 CI1输入
TIMER7_CI2_INPUT_TIMER7_CH2	选择TIMER7 CH2作为TIMER7 CI2输入
TIMER7_CI3_INPUT_TIMER7_CH3	选择TIMER7 CH3作为TIMER7 CI3输入
TIMER0_CIO_INPUT_TIMER0_CH0	选择CMP0输出作为TIMER0 CIO输入
TIMER0_CIO_INPUT_CMP0_OUT	选择TIMER0 CH0作为TIMER0 CIO输入
TIMER0_CI1_INPUT_TIMER0_CH1	选择TIMER0 CH1作为TIMER0 CI1输入
TIMER0_CI2_INPUT_TIMER0_CH2	选择TIMER0 CH2作为TIMER0 CI2输入
TIMER0_CI3_INPUT_TIMER0_CH3	选择TIMER0 CH3作为TIMER0 CI3输入
TIMER2_CIO_INPUT_TIMER2_CH0	选择TIMER2 CH0作为TIMER2 CIO输入
TIMER2_CIO_INPUT_CMP0_OUT	选择CMP0作为TIMER2 CIO输入
TIMER2_CIO_INPUT_CMP1_OUT	选择CMP1作为TIMER2 CIO输入
TIMER2_CIO_INPUT_CMP0_OUT_CMP1_OUT	选择CMP0 or CMP1作为TIMER2 CIO输入
TIMER2_CI1_INPUT_TIMER2_CH1	选择TIMER2 CH1作为TIMER2 CI1输入
TIMER2_CI2_INPUT_TIMER2_CH2	选择TIMER2 CH2作为TIMER2 CI2输入
TIMER2_CI3_INPUT_TIMER2_CH3	选择TIMER2 CH3作为TIMER2 CI3输入

枚举名称	枚举描述
TIMER1_CIO_INPUT_TIMER1_CH0	选择TIMER1 CH0作为TIMER1 CIO输入
TIMER1_CI1_INPUT_TIMER1_CH1	选择TIMER1 CH1作为TIMER1 CI1输入
TIMER1_CI2_INPUT_TIMER1_CH2	选择TIMER1 CH2作为TIMER1 CI2输入
TIMER1_CI3_INPUT_TIMER1_CH3	选择TIMER1 CH3作为TIMER1 CI3输入
TIMER1_CI3_INPUT_CMP0_OUT	选择CMP0输出作为TIMER1 CI3输入
TIMER1_CI3_INPUT_CMP1_OUT	选择CMP1输出作为TIMER1 CI3输入
TIMER1_CI3_INPUT_CMP0_OUT_CMP1_OUT	选择CMP0或CMP1输出作为TIMER1 CI3输入
TIMER4_CIO_INPUT_TIMER4_CH0	选择TIMER4 CH0作为TIMER4 CIO输入
TIMER4_CI1_INPUT_TIMER4_CH1	选择TIMER4 CH1作为TIMER4 CI1输入
TIMER4_CI2_INPUT_TIMER4_CH2	选择TIMER4 CH2作为TIMER4 CI2输入
TIMER4_CI3_INPUT_TIMER4_CH3	选择TIMER4 CH3作为TIMER4 CI3输入
TIMER3_CIO_INPUT_TIMER3_CH0	选择TIMER3 CH0作为TIMER3 CIO输入
TIMER3_CI1_INPUT_TIMER3_CH1	选择TIMER3 CH1作为TIMER3 CI1输入
TIMER3_CI2_INPUT_TIMER3_CH2	选择TIMER3 CH2作为TIMER3 CI2输入
TIMER3_CI3_INPUT_TIMER3_CH3	选择TIMER3 CH3作为TIMER3 CI3输入
TIMER23_CIO_INPUT_TIMER23_CH0	选择TIMER23 CH0作为TIMER23 CIO输入
TIMER23_CI1_INPUT_TIMER23_CH1	选择TIMER23 CH1作为TIMER23 CI1输入
TIMER23_CI2_INPUT_TIMER23_CH2	选择TIMER23 CH2作为TIMER23 CI2输入
TIMER23_CI3_INPUT_TIMER23_CH3	选择TIMER23 CH3作为TIMER23 CI3输入
TIMER22_CIO_INPUT_TIMER22_CH0	选择TIMER22 CH0作为TIMER22 CIO输入
TIMER22_CI1_INPUT_TIMER22_CH1	选择TIMER22 CH1作为TIMER22 CI1输入

枚举名称	枚举描述
TIMER22_CI2_INPUT_TIMER22_CH2	选择TIMER22 CH2作为TIMER22 CI2输入
TIMER22_CI3_INPUT_TIMER22_CH3	选择TIMER22 CH3作为TIMER22 CI3输入
TIMER22_CI3_INPUT_CMP0_OUT	选择CMP0输出作为TIMER22 CI3输入
TIMER22_CI3_INPUT_CMP1_OUT	选择CMP1输出作为TIMER22 CI3输入
TIMER22_CI3_INPUT_CMP0_OUT_CMP1_OUT	选择CMP0或CMP1输出作为TIMER22 CI3输入
TIMER14_CI0_INPUT_TIMER14_CH0	选择TIMER14 CH0作为TIMER14 CI0输入
TIMER14_CI0_INPUT_TIMER1_CH0	选择TIMER1 CH0作为TIMER14 CI0输入
TIMER14_CI0_INPUT_TIMER2_CH0	选择TIMER2 CH0作为TIMER14 CI0输入
TIMER14_CI0_INPUT_TIMER3_CH0	选择TIMER3 CH0作为TIMER14 CI0输入
TIMER14_CI0_INPUT_LXTAL	选择LXTAL作为TIMER14 CI0输入
TIMER14_CI0_INPUT_LPIRC4M	选择LPIRC4M作为TIMER14 CI0输入
TIMER14_CI0_INPUT_CKOUT1	选择CKOUT1作为TIMER14 CI0输入
TIMER14_CI1_INPUT_TIMER14_CH1	选择TIMER14 CH1作为TIMER14 CI1输入
TIMER14_CI1_INPUT_TIMER1_CH1	选择TIMER1 CH1作为TIMER14 CI1输入
TIMER14_CI1_INPUT_TIMER2_CH1	选择TIMER2 CH1作为TIMER14 CI1输入
TIMER14_CI1_INPUT_TIMER3_CH1	选择TIMER3 CH1作为TIMER14 CI1输入
TIMER40_CI0_INPUT_TIMER40_CH0	选择TIMER40 CH0作为TIMER40 CI0输入
TIMER40_CI0_INPUT_TIMER2_CH0	选择TIMER2 CH0作为TIMER40 CI0输入
TIMER40_CI0_INPUT_TIMER3_CH0	选择TIMER3 CH0作为TIMER40 CI0输入
TIMER40_CI0_INPUT_TIMER4_CH0	选择TIMER4 CH0作为TIMER40 CI0输入
TIMER40_CI0_INPUT_LXTAL	选择LXTAL作为TIMER40 CI0输入
TIMER40_CI0_INPUT_LPIRC4M	选择LPIRC4M作为TIMER40 CI0输入
TIMER40_CI0_INPUT_CKOUT1	选择CKOUT1作为TIMER40 CI0输入

枚举名称	枚举描述
TIMER40_C11_INPUT_TIMER40_CH1	选择TIMER40 CH1作为TIMER40 CI0输入
TIMER40_C11_INPUT_TIMER2_CH1	选择TIMER2 CH1作为TIMER40 CI0输入
TIMER40_C11_INPUT_TIMER3_CH1	选择TIMER3 CH1作为TIMER40 CI0输入
TIMER40_C11_INPUT_TIMER4_CH1	选择TIMER4 CH1作为TIMER40 CI0输入
TIMER41_C10_INPUT_TIMER41_CH0	选择TIMER41 CH0作为TIMER41 CI0输入
TIMER41_C10_INPUT_TIMER3_CH0	选择TIMER3 CH0作为TIMER41 CI0输入
TIMER41_C10_INPUT_TIMER4_CH0	选择TIMER4 CH0作为TIMER41 CI0输入
TIMER41_C10_INPUT_TIMER22_CH0	选择TIMER22 CH0作为TIMER41 CI0输入
TIMER41_C10_INPUT_LXTAL	选择LXTAL作为TIMER41 CI0输入
TIMER41_C10_INPUT_LPIRC4M	选择LPIRC4M作为TIMER41 CI0输入
TIMER41_C10_INPUT_CKOUT1	选择CKOUT1作为TIMER41 CI0输入
TIMER41_C11_INPUT_TIMER41_CH1	选择TIMER41 CH1作为TIMER41 CI1输入
TIMER41_C11_INPUT_TIMER3_CH1	选择TIMER3 CH1作为TIMER41 CI1输入
TIMER41_C11_INPUT_TIMER4_CH1	选择TIMER4 CH1作为TIMER41 CI1输入
TIMER41_C11_INPUT_TIMER22_CH1	选择TIMER22 CH1作为TIMER41 CI1输入
TIMER42_C10_INPUT_TIMER42_CH0	选择TIMER42 CH0作为TIMER42 CI0输入
TIMER42_C10_INPUT_TIMER4_CH0	选择TIMER4 CH0作为TIMER42 CI0输入
TIMER42_C10_INPUT_TIMER22_CH0	选择TIMER22 CH0作为TIMER42 CI0输入
TIMER42_C10_INPUT_TIMER23_CH0	选择TIMER23 CH0作为TIMER42 CI0输入
TIMER42_C10_INPUT_LXTAL	选择LXTAL作为TIMER42 CI0输入
TIMER42_C10_INPUT_LPIRC4M	选择LPIRC4M作为TIMER42 CI0输入
TIMER42_C10_INPUT_CKOUT1	选择CKOUT1作为TIMER42 CI0输入
TIMER42_C11_INPUT_TIMER42_CH1	选择TIMER42 CH1作为TIMER42 CI1输入

枚举名称	枚举描述
TIMER42_CI1_INPUT_TIMER4_CH1	选择TIMER4 CH1作为TIMER42 CI1输入
TIMER42_CI1_INPUT_TIMER22_CH1	选择TIMER22 CH1作为TIMER42 CI1输入
TIMER42_CI1_INPUT_TIMER23_CH1	选择TIMER23 CH1作为TIMER42 CI1输入
TIMER15_CIO_INPUT_TIMER15_CH0	选择TIMER15 CH0作为TIMER15 CIO输入
TIMER15_CIO_INPUT_IRC32K	选择IRC32K作为TIMER15 CIO输入
TIMER15_CIO_INPUT_LXTAL	选择LXTAL作为TIMER15 CIO输入
TIMER15_CIO_INPUT_WKUP_IT	选择WKUP IT作为TIMER15 CIO输入
TIMER16_CIO_INPUT_TIMER16_CH0	选择TIMER16 CH0作为TIMER16 CIO输入
TIMER16_CIO_INPUT_HXTAL_RTCDIV	选择HXTAL/RTCDIV 1M作为TIMER16 CIO输入
TIMER16_CIO_INPUT_CKOUT0	选择CKOUT0作为TIMER16 CIO输入
TIMER43_CIO_INPUT_TIMER43_CH0	选择TIMER43 CH0作为TIMER43 CIO输入
TIMER43_CIO_INPUT_TIMER22_CH0	选择TIMER22 CH0作为TIMER43 CIO输入
TIMER43_CIO_INPUT_TIMER23_CH0	选择TIMER23 CH0作为TIMER43 CIO输入
TIMER43_CIO_INPUT_LXTAL	选择LXTAL作为TIMER43 CIO输入
TIMER43_CIO_INPUT_LPIRC4M	选择LPIRC4M作为TIMER43 CIO输入
TIMER43_CIO_INPUT_CKOUT1	选择CKOUT1作为TIMER43 CIO输入
TIMER43_CI1_INPUT_TIMER43_CH1	选择TIMER43 CH1作为TIMER43 CI1输入
TIMER43_CI1_INPUT_TIMER22_CH1	选择TIMER22 CH1作为TIMER43 CI1输入
TIMER43_CI1_INPUT_TIMER23_CH1	选择TIMER23 CH1作为TIMER43 CI1输入
TIMER44_CIO_INPUT_TIMER44_CH0	选择TIMER44 CH0作为TIMER44 CIO输入
TIMER44_CIO_INPUT_TIMER23_CH0	选择TIMER23 CH0作为TIMER44 CIO输入
TIMER44_CIO_INPUT_LXTAL	选择LXTAL作为TIMER44 CIO输入
TIMER44_CIO_INPUT_LPIRC4M	选择LPIRC4M作为TIMER44 CIO输入
TIMER44_CIO_INPUT_CKOUT1	选择CKOUT1作为TIMER44 CIO输入

枚举名称	枚举描述
TIMER44_CI1_INPUT_TIMER44_CH1	选择TIMER44 CH1作为TIMER44 CI1输入
TIMER44_CI1_INPUT_TIMER23_CH1	选择TIMER23 CH1作为TIMER44 CI1输入

函数 syscfg_deinit

函数syscfg_deinit描述见下表:

表 3-1102. 函数 syscfg_deinit

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

例如:

```
/* reset SYSCFG registers */
```

```
syscfg_deinit();
```

函数 syscfg_i2c_fast_mode_plus_enable

函数syscfg_i2c_fast_mode_plus_enable描述见下表:

表 3-1103. 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(x=0,1,2,3)或PBx(x=6,7,8,9)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
SYSCFG_I2C2_FMP	I2C2 Fm+模式

<i>P</i>	
SYSCFG_I2C3_FM <i>P</i>	I2C3 Fm+模式
SYSCFG_I2C_FMP _PB6	PB6引脚Fm+模式
SYSCFG_I2C_FMP _PB7	PB7引脚Fm+模式
SYSCFG_I2C_FMP _PB8	PB8引脚Fm+模式
SYSCFG_I2C_FMP _PB9	PB9引脚Fm+模式
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* enable I2C0 fast 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 3-1104. 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I2Cx(x=0,1,2,3)或PBx(x=6,7,8,9)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_FM <i>P</i>	I2C0 Fm+模式
SYSCFG_I2C1_FM <i>P</i>	I2C1 Fm+模式
SYSCFG_I2C2_FM <i>P</i>	I2C2 Fm+模式
SYSCFG_I2C3_FM <i>P</i>	I2C3 Fm+模式
SYSCFG_I2C_FMP _PB6	PB6引脚Fm+模式
SYSCFG_I2C_FMP	PB7引脚Fm+模式

<code>_PB7</code>	
<code>SYSCFG_I2C_FMP</code> <code>_PB8</code>	PB8引脚Fm+模式
<code>SYSCFG_I2C_FMP</code> <code>_PB9</code>	PB9引脚Fm+模式
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* disable I2C0 fast mode plus function */
```

```
syscfg_i2c_fast_mode_plus_disable(SYSCFG_I2C0_FMP);
```

函数 `syscfg_analog_switch_enable`

函数`syscfg_analog_switch_enable`描述见下表：

表 3-1105. 函数 `syscfg_analog_switch_enable`

函数名称	<code>syscfg_analog_switch_enable</code>
函数原型	<code>void syscfg_analog_switch_enable(uint32_t gpio_answ);</code>
功能描述	模拟开关打开
先决条件	-
被调用函数	-
输入参数{in}	
<code>gpio_answ</code>	GPIO模拟开关
<code>SYSCFG_PA0_AN</code> <code>ALOG_SWITCH</code>	PA0模拟开关
<code>SYSCFG_PA1_AN</code> <code>ALOG_SWITCH</code>	PA1模拟开关
<code>SYSCFG_PC2_AN</code> <code>ALOG_SWITCH</code>	PC2模拟开关
<code>SYSCFG_PC3_AN</code> <code>ALOG_SWITCH</code>	PC3模拟开关
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* open PA0 analog switch function */
```

```
syscfg_analog_switch_enable(SYSCFG_PA0_ANALOG_SWITCH);
```

函数 **syscfg_analog_switch_disable**

函数syscfg_analog_switch_disable描述见下表：

表 3-1106. 函数 **syscfg_analog_switch_disable**

函数名称	syscfg_analog_switch_disable
函数原型	void syscfg_analog_switch_disable(uint32_t gpio_answ);
功能描述	模拟开关关闭
先决条件	-
被调用函数	-
输入参数{in}	
gpio_answ	GPIO模拟开关
SYSCFG_PA0_ANALOG_SWITCH	PA0模拟开关
SYSCFG_PA1_ANALOG_SWITCH	PA1模拟开关
SYSCFG_PC2_ANALOG_SWITCH	PC2模拟开关
SYSCFG_PC3_ANALOG_SWITCH	PC3模拟开关
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* close PA0 analog switch function */
```

```
syscfg_analog_switch_disable(SYSCFG_PA0_ANALOG_SWITCH);
```

函数 **syscfg_enet_phy_interface_config**

函数syscfg_enet_phy_interface_config描述见下表：

表 3-1107. 函数 **syscfg_enet_phy_interface_config**

函数名称	syscfg_enet_phy_interface_config
函数原型	void syscfg_enet_phy_interface_config(uint32_t ethernet, uint32_t phy_interface)
功能描述	选择以太网PHY接口
先决条件	-
被调用函数	-
输入参数{in}	
ethernet	以太网
ENET0	Ethernet 0
ENET1	Ethernet 1

输入参数{in}	
phy_interface	指定以太网接口模式
<i>SYSCFG_ENET_PHY_MII</i>	选择MII模式
<i>SYSCFG_ENET_PHY_RMII</i>	选择RMII模式
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* configure the PHY interface for the Ethernet 0 MAC */
syscfg_enet_phy_interface_config(ENET0, SYSCFG_ENET_PHY_MII);
```

函数 syscfg_exti_line_config

函数syscfg_exti_line_config描述见下表:

表 3-1108. 函数 syscfg_exti_line_config

函数名称	syscfg_exti_line_config
函数原形	void syscfg_exti_line_config(uint8_t exti_port, uint8_t exti_pin);
功能描述	配置GPIO引脚作为EXTI
先决条件	-
被调用函数	-
输入参数{in}	
exti_port	指定EXTI使用的GPIO端口
<i>EXTI_SOURCE_GPIOx</i>	x = A,B,C,D,E,F,G,H,J,K
输入参数{in}	
exti_pin	EXTI引脚
<i>EXTI_SOURCE_PINx</i>	GPIOA x = 0..15,GPIOB x = 0..15,GPIOC x = 0..15,GPIOD x = 0..15,GPIOE x = 0..15, GPIOF x = 0..15,GPIOG x = 0..15,GPIOH x = 0..15,GPIOI x = 0..15,GPIOJ x = 8,9,10,11, GPIOK x = 0,1,2,4,5,6
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* configure the PA0 pin as EXTI Line */
syscfg_exti_line_config(EXTI_SOURCE_GPIOA, EXTI_SOURCE_PIN0);
```

函数 syscfg_lockup_enable

函数syscfg_lockup_enable描述见下表：

表 3-1109. 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SYSCFG锁定功能
先决条件	-
被调用函数	-
输入参数{in}	
lockup	锁定功能
SYSCFG_LVD_LO CKUP	LVD锁定
SYSCFG_CPU_LO CKUP	CPU锁定
SYSCFG_BKPRAM _LOCKUP	Region 2 备份SRAM ECC双差错锁定
SYSCFG_SRAM1_ LOCKUP	Region 1 SRAM1 ECC双差错锁定
SYSCFG_SRAM0_ LOCKUP	Region 1 SRAM0 ECC双差错锁定
SYSCFG_DTCM_L OCKUP	Region 0 DTCM ECC双差错锁定
SYSCFG_ITCM_LO CKUP	Region 0 ITCM-RAM ECC双差错锁定
SYSCFG_AXIRAM_ LOCKUP	Region 0 AXI-SRAM ECC双差错锁定
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* enable module lockup function */
```

```
syscfg_lockup_enable(SYSCFG_CPU_LOCKUP);
```

函数 syscfg_lockup_disable

函数syscfg_lockup_disable描述见下表：

表 3-1110. 函数 syscfg_lockup_disable

函数名称	syscfg_lockup_disable
函数原型	void syscfg_lockup_disable(uint32_t lockup);

功能描述	禁能SYSCFG锁定功能
先决条件	-
被调用函数	-
输入参数{in}	
lockup	锁定功能
SYSCFG_LVD_LO CKUP	LVD锁定
SYSCFG_CPU_LO CKUP	CPU锁定
SYSCFG_BKPRAM _LOCKUP	Region 2备份SRAM ECC双差错锁定
SYSCFG_SRAM1_ LOCKUP	Region 1 SRAM1 ECC双差错锁定
SYSCFG_SRAM0_ LOCKUP	Region 1 SRAM0 ECC双差错锁定
SYSCFG_DTCM_L OCKUP	Region 0 DTCM ECC双差错锁定
SYSCFG_ITCM_LO CKUP	Region 0 ITCM-RAM ECC双差错锁定
SYSCFG_AXIRAM_ LOCKUP	Region 0 AXI-SRAM ECC双差错锁定
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* disable module lockup function */
```

```
syscfg_lockup_disable(SYSCFG_CPU_LOCKUP);
```

函数 syscfg_timer_input_source_select

函数syscfg_timer_input_source_select描述见下表：

表 3-1111. 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 表3-1101. 枚举类型timer_channel_input_enum

输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* select timer channel input source */
```

```
syscfg_timer_input_source_select(TIMER7_CIO_INPUT_TIMER7_CH0);
```

函数 syscfg_compensation_config

函数syscfg_compensation_config描述见下表：

表 3-1112. 函数 syscfg_compensation_config

函数名称	syscfg_compensation_config
函数原型	void syscfg_compensation_config(uint32_t syscfg_cps);
功能描述	配置I/O补偿单元
先决条件	-
被调用函数	-
输入参数{in}	
syscfg_compensation	SYSCFG补偿单元
SYSCFG_COMPENSATION_ENABLE	I/O补偿单元使能
SYSCFG_COMPENSATION_DISABLE	I/O补偿单元禁能
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* configure the I/O compensation cell */
```

```
syscfg_compensation_config(SYSCFG_COMPENSATION_ENABLE);
```

函数 syscfg_io_low_voltage_speed_optimization_enable

函数syscfg_io_low_voltage_speed_optimization_enable描述见下表：

表 3-1113. 函数 syscfg_io_low_voltage_speed_optimization_enable

函数名称	syscfg_io_low_voltage_speed_optimization_enable
函数原型	void syscfg_io_low_voltage_speed_optimization_enable(void);
功能描述	使能I/O速度优化，低电压下高速功能

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 I/O speed optimization, high-speed at low-voltage */
```

```
syscfg_io_low_voltage_speed_optimization_enable();
```

函数 syscfg_io_low_voltage_speed_optimization_disable

函数syscfg_io_low_voltage_speed_optimization_disable描述见下表：

表 3-1114. 函数 syscfg_io_low_voltage_speed_optimization_disable

函数名称	syscfg_io_low_voltage_speed_optimization_disable
函数原型	void syscfg_io_low_voltage_speed_optimization_disable(void);
功能描述	禁能I/O速度优化，低电压下高速功能
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 I/O speed optimization, high-speed at low-voltage */
```

```
syscfg_io_low_voltage_speed_optimization_disable();
```

函数 syscfg_pnmos_compensation_code_set

函数syscfg_pnmos_compensation_code_set描述见下表：

表 3-1115. 函数 syscfg_pnmos_compensation_code_set

函数名称	syscfg_pnmos_compensation_code_set
函数原型	void syscfg_pnmos_compensation_code_set(uint32_t mos, uint32_t code);
功能描述	设置PMOS/NMOS补偿单元代码
先决条件	-

被调用函数	-
输入参数{in}	
mos	P/N MOS
NMOS_COMPENSATION	NMOS
PMOS_COMPENSATION	PMOS
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* set PMOS compensation value */
```

```
syscfg_pnmos_compensation_code_set(PMOS_COMPENSATION, 0x02);
```

函数 syscfg_secure_sram_size_set

函数syscfg_secure_sram_size_set描述见下表：

表 3-1116. 函数 syscfg_secure_sram_size_set

函数名称	syscfg_secure_sram_size_set
函数原型	void syscfg_secure_sram_size_set(uint32_t size);
功能描述	设置安全SRAM的大小
先决条件	-
被调用函数	-
输入参数{in}	
size	安全SRAM的大小
SECURE_SRAM_SIZE_0KB	安全SRAM的大小为0KB
SECURE_SRAM_SIZE_32KB	安全SRAM的大小为32KB
SECURE_SRAM_SIZE_64KB	安全SRAM的大小为64KB
SECURE_SRAM_SIZE_128KB	安全SRAM的大小为128KB
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* set secure SRAM size */
```

```
syscfg_secure_sram_size_set(SECURE_SRAM_SIZE_32KB);
```

函数 syscfg_secure_sram_size_get

函数syscfg_secure_sram_size_get描述见下表：

表 3-1117. 函数 syscfg_secure_sram_size_get

函数名称	syscfg_secure_sram_size_get
函数原型	uint32_t syscfg_secure_sram_size_get(void);
功能描述	获取安全SRAM的大小
先决条件	-
被调用函数	-
输入参数{in}	
-	-
输出参数{out}	
-	-
返回值	
uint32_t	安全SRAM的大小
SECURE_SRAM_SIZE_0KB	安全SRAM的大小为0KB
SECURE_SRAM_SIZE_32KB	安全SRAM的大小为32KB
SECURE_SRAM_SIZE_64KB	安全SRAM的大小为64KB
SECURE_SRAM_SIZE_128KB	安全SRAM的大小为128KB

例如：

```
/* get secure SRAM size */
uint32_t ret_val = 0U;
ret_val = syscfg_secure_sram_size_get();
```

函数 syscfg_bootmode_get

函数syscfg_bootmode_get描述见下表：

表 3-1118. 函数 syscfg_bootmode_get

函数名称	syscfg_bootmode_get
函数原型	uint32_t syscfg_bootmode_get(void);
功能描述	获取启动方式
先决条件	-
被调用函数	-
输入参数{in}	

-	-
输出参数{out}	
-	-
返回值	
uint32_t	启动方式
BOOT_SRAM	从SRAM (ITCM/DTCM/RAM shared/AXI SRAM)启动
BOOT_SECURITY	从Security启动
BOOT_SYSTEM	从BOOT_SYS (BootLoader)启动
BOOT_USER_FLASH	从BOOT_USER (User flash OSPI0/1)启动

例如:

```
/* get BOOT mode */
uint32_t boot_mod = 0U;

boot_mod = syscfg_bootmode_get();
```

函数 syscfg_tcm_wait_state_enable

函数syscfg_tcm_wait_state_enable描述见下表:

表 3-1119. 函数 syscfg_tcm_wait_state_enable

函数名称	syscfg_tcm_wait_state_enable
函数原型	void syscfg_tcm_wait_state_enable(void);
功能描述	使能TCM等待功能
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 TCM wait state */

syscfg_tcm_wait_state_enable();
```

函数 syscfg_tcm_wait_state_disable

函数syscfg_tcm_wait_state_disable描述见下表:

表 3-1120. 函数 syscfg_tcm_wait_state_disable

函数名称	syscfg_tcm_wait_state_disable
------	-------------------------------

函数原型	void syscfg_tcm_wait_state_disable(void);
功能描述	禁能TCM等待功能
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 TCM wait state */
```

```
syscfg_tcm_wait_state_disable();
```

函数 syscfg_fpu_interrupt_enable

函数syscfg_fpu_interrupt_enable描述见下表:

表 3-1121. 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 fpu_int);
功能描述	使能FPU中断
先决条件	-
被调用函数	-
输入参数{in}	
fpu_int	FPU中断
SYSCFG_FPUINT_I NEXACT	不精确中断
SYSCFG_FPUINT_I NPUT_ABNORMAL	输入异常中断
SYSCFG_FPUINT_ OVERFLOW	溢出中断
SYSCFG_FPUINT_ UNDERFLOW	下溢中断
SYSCFG_FPUINT_ DIV0	除0中断
SYSCFG_FPUINT_I NVALID_OPERATI ON	无效操作中断
输出参数{out}	
-	-
返回值	

-	-
---	---

例如:

```
/* 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 3-1122. 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 fpu_int);
功能描述	禁能FPU中断
先决条件	-
被调用函数	-
输入参数{in}	
fpu_int	FPU中断
SYSCFG_FPUINT_INEXACT	不精确中断
SYSCFG_FPUINT_INPUT_ABNORMAL	输入异常中断
SYSCFG_FPUINT_OVERFLOW	溢出中断
SYSCFG_FPUINT_UNDERFLOW	下溢中断
SYSCFG_FPUINT_DIV0	除0中断
SYSCFG_FPUINT_INVALID_OPERATION	无效操作中断
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* disable FPU inexact interrupt */
```

```
syscfg_fpu_interrupt_disable(SYSCFG_FPUINT_INEXACT);
```

函数 syscfg_compensation_flag_get

函数syscfg_compensation_flag_get描述见下表:

表 3-1123. 函数 syscfg_compensation_flag_get

函数名称	syscfg_compensation_flag_get
函数原型	FlagStatus syscfg_compensation_flag_get(uint32_t cps_flag);
功能描述	获取SYSCFG补偿单元标志
先决条件	-
被调用函数	-
输入参数{in}	
cps_flag	补偿单元标志
SYSCFG_FLAG_IO_LOW_VOLTAGE	I/O低电压状态，产品在2.5V以下工作
SYSCFG_FLAG_COMPENSATION_READY	I/O补偿单元准备好标志
输出参数{out}	
-	-
返回值	
FlagStatus	SET或RESET

例如：

```
/* get compensation cell flags */
```

```
FlagStatus flag;
```

```
flag = syscfg_compensation_flag_get(SYSCFG_FLAG_IO_LOW_VOLTAGE);
```

函数 syscfg_cpu_cache_status_get

函数syscfg_cpu_cache_status_get描述见下表：

表 3-1124. 函数 syscfg_cpu_cache_status_get

函数名称	syscfg_cpu_cache_status_get
函数原型	uint32_t syscfg_cpu_cache_status_get(uint32_t cache, uint32_t status);
功能描述	获取SYSCFG CPU CACHE状态
先决条件	-
被调用函数	-
输入参数{in}	
cache	cache
ICACHE_STATUS	ICACHE状态
DCACHE_STATUS	DCACHE状态
输入参数{in}	
status	状态
CPU_CACHE_ERR_OR_DETECTION	选择错误检测信息
CPU_CACHE_ERR	选择错误库信息

OR_BANK	
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* get the ICACHE detection and error information */
```

```
uint32_t cache;
```

```
cache = syscfg_cpu_cache_status_get(ICACHE_STATUS, CPU_CACHE_ERROR_BANK);
```

函数 syscfg_brownout_reset_threshold_level_get

函数syscfg_brownout_reset_threshold_level_get描述见下表：

表 3-1125. 函数 syscfg_brownout_reset_threshold_level_get

函数名称	syscfg_brownout_reset_threshold_level_get
函数原型	uint32_t syscfg_brownout_reset_threshold_level_get(void);
功能描述	获取BOR掉电复位阈值电平
先决条件	-
被调用函数	-
输入参数{in}	
-	-
输出参数{out}	
-	-
返回值	
uint32_t	BOR掉电复位阈值电平值
BOR_OFF	无BOR掉电复位
BOR_THRESHOLD_VAL1	BOR掉电复位阈值1
BOR_THRESHOLD_VAL2	BOR掉电复位阈值2
BOR_THRESHOLD_VAL3	BOR掉电复位阈值3

例如：

```
/* get brownout reset threshold level */
```

```
uint32_t val;
```

```
val = syscfg_brownout_reset_threshold_level_get();
```


3.32. TIMER

定时器含有可编程的一个无符号计数器，支持输入捕获和输出比较，分为五种类型：高级定时器（TIMERx, x=0, 7），通用定时器L0（TIMERx, x=1~4, 22, 23），通用定时器L3（TIMERx, x=14, 40~44），通用定时器L4（TIMERx, x=15, 16），基本定时器（TIMERx, x=5, 6, 50, 51），不同类型的定时器具体功能有所差别。章节[3.32.1](#)描述了TIMER的寄存器列表，章节[3.32.2](#)对TIMER库函数进行说明。

3.32.1. 外设寄存器说明

TIMER寄存器列表如下表所示：

表 3-1126. 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_CNTL	计数器低位寄存器（仅用于TIMERx, x=50,51）
TIMER_PSC	预分频寄存器
TIMER_CAR	计数器自动重载寄存器
TIMER_CARL	计数器自动重载低位寄存器（仅用于TIMERx, x=50,51）
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_CCHP	互补通道保护寄存器
TIMER_MCHCTL0	TIMER多模式通道控制寄存器0
TIMER_MCHCTL1	TIMER多模式通道控制寄存器1
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_MCH1CV	TIMER多模式通道1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

寄存器名称	寄存器描述
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_FCCHP0	TIMER独立互补通道捕获寄存器0
TIMER_FCCHP1	TIMER独立互补通道捕获寄存器1
TIMER_FCCHP2	TIMER独立互补通道捕获寄存器2
TIMER_FCCHP3	TIMER独立互补通道捕获寄存器3
TIMER_AFCTL0	备用功能控制寄存器0
TIMER_AFCTL1	备用功能控制寄存器1
TIMER_WDGPWR	看门狗计数器周期寄存器
TIMER_CREP1	重复计数寄存器1
TIMER_CNTH	计数器高位寄存器（仅用于TIMERx, x=50,51）
TIMER_CARH	计数器自动重载低高位寄存器（仅用于TIMERx, x=50,51）
TIMER_DMACFG	DMA配置寄存器
TIMER_DMATB	DMA发送缓冲区寄存器
TIMER_CFG	配置寄存器

3.32.2. 外设库函数说明

TIMER库函数列表如下表所示：

表 3-1127. 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

库函数名称	库函数描述
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_delayable_single_pulse_mode_config	配置外设TIMER的可延时的单脉冲模式
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
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_clear_config	配置TIMER的通道输出比较清0功能
timer_channel_output_polarity_config	通道输出极性配置
timer_channel_complementary_output_polarity_config	互补通道输出极性配置
timer_channel_output_state_config	配置通道状态
timer_channel_complementary_output_state_config	配置互补通道输出状态
timer_channel_input_struct_para_init	将TIMER通道输入参数结构体中所有参数初始化为默认值

库函数名称	库函数描述
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
timer_input_trigger_source_select	TIMER的输入触发源选择
timer_master_output0_trigger_source_select	选择TIMER主模式输出0触发源
timer_master_output1_trigger_source_select	选择TIMER主模式输出1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配置为非正交编码器模式
timer_internal_clock_config	TIMER配置为内部时钟模式
timer_internal_trigger_as_external_clock_config	配置TIMER的内部触发为时钟源
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_mod	配置TIMER的复合PWM模式

库函数名称	库函数描述
e_config	
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_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_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
timer_break_lock_config	配置TIMER锁存中止功能
timer_break_lock_release_config	配置TIMER锁存中止功能的释放功能
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_free_complementary_struct_parameter_init	将TIMER通道独立互补参数结构体中所有参数初始化为默认值
timer_channel_free_complementary_config	配置TIMER独立互补通道保护功能
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_upif_backup_config	配置UPIF位备份功能
timer_upifbu_bit_get	获取TIMERx_CNT寄存器中的UPIFBU位
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 3-1128. 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）

成员名称	功能描述
period	周期 (0~0xFFFF(TIMERx(x=0,7,14~16,40~44)), 0~0xFFFFFFFF(TIMERx(x=1~6,22,23)), 0~0xFFFFFFFFFFFFFFFF(TIMERx(x=50,51)))
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, 用于TIMER_CREP0寄存器; 0xFF~0xFFFFFFFF, 用于TIMER_CREP1寄存器)

结构体 timer_break_parameter_struct

表 3-1129. 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_CCHP_PROT_OFF, TIMER_CCHP_PROT_0, TIMER_CCHP_PROT_1, TIMER_CCHP_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)
break0lock	BREAK0输入锁存功能 (TIMER_BREAK0_LK_ENABLE, TIMER_BREAK0_LK_DISABLE)
break0release	BREAK0输入锁存释放功能 (TIMER_BREAK0_RELEASE, TIMER_BREAK0_UNRELEASE)
break1state	BREAK1输入使能 (TIMER_BREAK1_ENABLE, TIMER_BREAK1_DISABLE)
break1filter	BREAK1输入滤波 (0~15)
break1polarity	BREAK1输入极性 (TIMER_BREAK1_POLARITY_LOW, TIMER_BREAK1_POLARITY_HIGH)
break1lock	BREAK1输入锁存功能 (TIMER_BREAK1_LK_ENABLE, TIMER_BREAK1_LK_DISABLE)
break1release	BREAK1输入锁存释放功能 (TIMER_BREAK1_RELEASE, TIMER_BREAK1_UNRELEASE)

结构体 timer_oc_parameter_struct

表 3-1130. 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 3-1131. 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 3-1132. 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)

结构体 timer_free_complementary_parameter_struct

表 3-1133. 结构体 timer_free_complementary_parameter_struct

成员名称	功能描述
freecomstate	独立互补通道保护使能 (TIMER_FCCHP_STATE_ENABLE, TIMER_FCCHP_STATE_DISABLE)

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）

函数 timer_deinit

函数timer_deinit描述见下表：

表 3-1134. 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=0~7,14~16,22,23,40~44,50,51)	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 3-1135. 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 3-1136. 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, 14~16, 22, 23, 40~44, 50, 51)	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(TIMERO, &timer_initpara);
```

函数 timer_enable

函数timer_enable描述见下表:

表 3-1137. 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=0~7,14~16,22,23,40~44,50,51)	TIMER外设选择
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* enable TIMERO */
```

```
timer_enable(TIMERO);
```

函数 timer_disable

函数timer_disable描述见下表:

表 3-1138. 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=0~7,14~16,22,23,40~44,50,51)	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 3-1139. 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, 14~16, 22, 23, 40~44, 50, 51)	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 3-1140. 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
<i>TIMERx</i> (<i>x</i> =0~7,14~16,22,23,40~44,50,51)	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 3-1141. 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
<i>TIMERx</i> (<i>x</i> =0~7,14~16,22,23,40~44,50,51)	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 3-1142. 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, 14~16, 22, 23, 40~44, 50, 51)	TIMER外设选择
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* disable TIMER0 the update event */
timer_update_event_disable(TIMER0);
```

函数 timer_counter_alignment

函数timer_counter_alignment描述见下表：

表 3-1143. 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, 22, 23)	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 3-1144. 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
TIMEx(x=0~4,7,22,23)	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 3-1145. 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=0~4,7,22,23)	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 3-1146. 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=0~7,14~16,22,23,40~44,50,51)	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_RELO AD_NOW	预分频值立即加载
TIMER_PSC_RELO AD_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 3-1147. 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 ccsel, uint32_t repetition)
功能描述	配置外设TIMER的重复计数器
先决条件	-
被调用函数	-
输入参数{in}	
timer_periph	TIMER外设
TIMERx(x=0,7,14~16,40~44)	TIMER外设选择
输入参数{in}	
ccsel	重复计数器选择
TIMER_CREP0_ENABLE	更新速率取决于TIMERx_CREP0寄存器
TIMER_CREP1_ENABLE	更新速率取决于TIMERx_CREP1寄存器
输入参数{in}	
repetition	重复计数器值, 取值范围 (0~0xFF, 用于TIMER_CREP0寄存器; 0xFF~0xFFFFFFFF, 用于TIMER_CREP1寄存器)
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* configure TIMER0 repetition register 0 value */
timer_repetition_value_config(TIMER0, TIMER_CREP0_ENABLE, 98);
```

函数 timer_runtime_repetition_value_read

函数timer_runtime_repetition_value_read描述见下表:

表 3-1148. 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> =0,7,14~16,40~44)	TIMER外设选择
输出参数{out}	
-	-
返回值	
uint32_t	TIMER_CREP1寄存器的计数重复值（0~0xFFFFFFFF）

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 3-1149. 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
<i>TIMERx</i> (<i>x</i> =0~7,14~16,22,23,40~44,50,51)	TIMER外设选择
输入参数{in}	
autoreload	计数器自动重载值 0~0xFFFF, TIMERx(<i>x</i> =0,2,3,7,14~16,40~44) 0~0xFFFFFFFF, TIMERx(<i>x</i> =1,4,5,6,22,23) 0~0xFFFFFFFFFFFFFFFF, TIMERx(<i>x</i> =50,51)
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 3-1150. 函数 timer_autoreload_value_read

函数名称	timer_autoreload_value_read
函数原型	uint64_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=0~7,14~16,22,23,40~44,50,51)	TIMER外设选择
输出参数{out}	
-	-
返回值	
uint64_t	计数器自动重载值 0~0xFFFF, TIMERx(x=0,2,3,7,14~16,40~44) 0~0xFFFFFFFF, TIMERx(x=1,4,5,6,22,23) 0~0xFFFFFFFFFFFFFFFF, TIMERx(x=50,51)

例如：

```
/* get TIMER autoreload register value */
uint64_t i = 0;
i =(uint64_t) timer_autoreload_value_read (TIMER0);
```

函数 timer_counter_value_config

函数timer_counter_value_config描述见下表：

表 3-1151. 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=0~7,14~16,22,23,40~44,50,51)	TIMER外设选择

输入参数{in}	
counter	计数器值 0~0xFFFF, TIMERx(x=0,2,3,7,14~16,40~44) 0~0xFFFFFFFF, TIMERx(x=1,4,5,6,22,23) 0~0xFFFFFFFFFFFFFFFF, TIMERx(x=50,51)
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 3-1152. 函数 timer_counter_read

函数名称	timer_counter_read
函数原型	uint64_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=0~7,14~16,22,23,40~44,50,51)	TIMER外设选择
输出参数{out}	
-	-
返回值	
uint64_t	外设TIMER的计数器值 0~0xFFFF, TIMERx(x=0,2,3,7,14~16,40~44) 0~0xFFFFFFFF, TIMERx(x=1,4,5,6,22,23) 0~0xFFFFFFFFFFFFFFFF, TIMERx(x=50,51)

例如:

```
/* read TIMER0 counter value */
```

```
uint64_t i = 0;
```

```
i = timer_counter_read(TIMER0);
```

函数 timer_prescaler_read

函数timer_prescaler_read描述见下表：

表 3-1153. 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=0~7,14~16,22,23,40~44,50,51)	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 3-1154. 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=0~7,14~16,22,23,40~44,50,51)	TIMER外设选择
输入参数{in}	
spmode	脉冲模式
TIMER_SP_MODE_	单脉冲模式计数

<i>SINGLE</i>	
<i>TIMER_SP_MODE_REPETITIVE</i>	重复模式计数
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* configure TIMER0 single pulse mode */
```

```
timer_single_pulse_mode_config(TIMER0, TIMER_SP_MODE_SINGLE);
```

函数 timer_delayable_single_pulse_mode_config

函数timer_delayable_single_pulse_mode_config描述见下表：

表 3-1155. 函数 timer_delayable_single_pulse_mode_config

函数名称	timer_delayable_single_pulse_mode_config
函数原型	void timer_delayable_single_pulse_mode_config(uint32_t timer_periph, uint16_t channel, uint32_t dspmode, uint16_t cnt_dir);
功能描述	配置外设TIMER的可延时的单脉冲模式
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=0~4,7,14,22,23,40~44)	TIMER外设选择
输入参数{in}	
channel	TIMER通道
TIMER_CH_0	通道0, TIMERx(x=0~4,7,14,22,23,40~44)
TIMER_CH_1	通道1, TIMERx(x=0~4,7,14,22,23,40~44)
TIMER_CH_2	通道2, TIMERx(x=0~4,7,22,23)
TIMER_CH_3	通道3, TIMERx(x=0~4,7,22,23)
TIMER_MCH_0	多模式通道0, TIMERx (x=0,7,14,40~44)
TIMER_MCH_1	多模式通道1, TIMERx (x=0,7)
TIMER_MCH_2	多模式通道2, TIMERx (x=0,7)
TIMER_MCH_3	多模式通道3, TIMERx (x=0,7)
输入参数{in}	
dspmode	可延时单脉冲模式
TIMER_OC_MODE_DSPM0	可延时单脉冲模式0
TIMER_OC_MODE_DSPM1	可延时单脉冲模式1

输入参数{in}	
cnt_dir	计数器方向选择
<i>TIMER_COUNTER_UP</i>	向上计数
<i>TIMER_COUNTER_DOWN</i>	向下计数
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* configure TIMER0_CH0 delayable single pulse mode */
```

```
timer_delayable_single_pulse_mode_config(TIMER0,                                TIMER_CH_0,
TIMER_OC_MODE_DSPM0, TIMER_COUNTER_UP);
```

函数 timer_update_source_config

函数timer_update_source_config描述见下表:

表 3-1156. 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
<i>TIMERx</i> (<i>x</i> =0~7,14~16,22,23,40~44,50,51)	TIMER外设选择
输入参数{in}	
update	更新源
<i>TIMER_UPDATE_SRC_GLOBAL</i>	下述任一事件产生更新中断或DMA请求: – UPG位被置1 – 计数器溢出/下溢 – 从模式控制器产生的更新
<i>TIMER_UPDATE_SRC_REGULAR</i>	只有计数器溢出/ 下溢才产生更新中断或DMA请求
输出参数{out}	
-	-
返回值	
-	-

例如:

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

```
timer_update_source_config(TIMER0, TIMER_UPDATE_SRC_REGULAR);
```

函数 timer_dma_enable

函数timer_dma_enable描述见下表:

表 3-1157. 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=0~7,14~16,22,23,40~44,50,51)	参考具体参数
输入参数{in}	
dma	DMA源
TIMER_DMA_UPD	更新DMA请求, TIMERx(x=0~7,14~16,22,23,40~44,50,51)
TIMER_DMA_CH0D	通道0比较/捕获DMA请求, TIMERx (x=0~4,7,14~16,22,23,40~44)
TIMER_DMA_CH1D	通道1比较/捕获DMA请求, TIMERx (x=0~4,7,14,22,23,40~44)
TIMER_DMA_CH2D	通道2比较/捕获DMA请求, TIMERx (x=0~4,7,22,23)
TIMER_DMA_CH3D	通道3比较/捕获DMA请求, TIMERx (x=0~4,7,22,23)
TIMER_DMA_CMTD	换相DMA更新请求, TIMERx (x=0,7,14~16,40~44)
TIMER_DMA_TRGD	触发DMA请求, TIMERx (x=0~4,7,14,22,23,40~44)
TIMER_DMA_MCH0D	多模式通道0比较/捕获DMA请求, TIMERx (x=0,7,14~16,40~44)
TIMER_DMA_MCH1D	多模式通道1比较/捕获DMA请求, TIMERx (x=0,7)
TIMER_DMA_MCH2D	多模式通道2比较/捕获DMA请求, TIMERx (x=0,7)
TIMER_DMA_MCH3D	多模式通道3比较/捕获DMA请求, TIMERx (x=0,7)

输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* enable the TIMER0 update DMA */
```

```
timer_dma_enable(TIMER0, TIMER_DMA_UPD);
```

函数 timer_dma_disable

函数timer_dma_disable描述见下表：

表 3-1158. 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=0~7,14~16,22,23,40~44,50,51)	参考具体参数
输入参数{in}	
dma	DMA源
TIMER_DMA_UPD	更新DMA请求, TIMERx(x=0~7,14~16,22,23,40~44,50,51)
TIMER_DMA_CH0D	通道0比较/捕获DMA请求, TIMERx (x=0~4,7,14~16,22,23,40~44)
TIMER_DMA_CH1D	通道1比较/捕获DMA请求, TIMERx (x=0~4,7,14,22,23,40~44)
TIMER_DMA_CH2D	通道2比较/捕获DMA请求, TIMERx (x=0~4,7,22,23)
TIMER_DMA_CH3D	通道3比较/捕获DMA请求, TIMERx (x=0~4,7,22,23)
TIMER_DMA_CMTD	换相DMA更新请求, TIMERx (x=0,7,14~16,40~44)
TIMER_DMA_TRGD	触发DMA请求, TIMERx (x=0~4,7,14,22,23,40~44)
TIMER_DMA_MCH0D	多模式通道0比较/捕获DMA请求, TIMERx (x=0,7,14~16,40~44)
TIMER_DMA_MCH1D	多模式通道1比较/捕获DMA请求, TIMERx (x=0,7)

<i>TIMER_DMA_MCH</i> <i>2D</i>	多模式通道2比较/捕获DMA请求, <i>TIMERx</i> (<i>x</i> =0,7)
<i>TIMER_DMA_MCH</i> <i>3D</i>	多模式通道3比较/捕获DMA请求, <i>TIMERx</i> (<i>x</i> =0,7)
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* disable the TIMER0 update DMA */
```

```
timer_dma_disable(TIMER0, TIMER_DMA_UPD);
```

函数 timer_channel_dma_request_source_select

函数timer_channel_dma_request_source_select描述见下表:

表 3-1159. 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
<i>TIMERx</i> (<i>x</i> =0~4, 7, 14~16, 22, 23, 40~44)	TIMER外设选择
输入参数{in}	
dma_request	通道的DMA请求源选择
<i>TIMER_DMAREQUEST_CHANNELEVENT</i>	当通道捕获/比较事件发生时, 发送通道n的DMA请求
<i>TIMER_DMAREQUEST_UPDATEEVENT</i>	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 3-1160. 函数 timer_dma_transfer_config

函数名称	timer_dma_transfer_config
函数原型	void timer_dma_transfer_config(uint32_t timer_periph, uint32_t dma_baseaddr, uint32_t dma_lenth);
功能描述	配置外设TIMER的DMA模式
先决条件	-
被调用函数	-
输入参数{in}	
timer_periph	TIMER外设
TIMERx(x=0~4,7,14~16,22,23,40~44)	参考具体参数
输入参数{in}	
dma_baseaddr	DMA传输起始地址
TIMER_DMACFG_DMATA_CTL0	DMA传输起始地址: TIMER_CTL0, TIMERx(x=0~4,7,14~16,22,23,40~44)
TIMER_DMACFG_DMATA_CTL1	DMA传输起始地址: TIMER_CTL1, TIMERx(x=0~4,7,14~16,22,23,40~44)
TIMER_DMACFG_DMATA_SMCFG	DMA传输起始地址: TIMER_SMCFG, TIMERx(x=0~4,7,14,22,23,40~44)
TIMER_DMACFG_DMATA_DMAINTEN	DMA传输起始地址: TIMER_DMAINTEN, TIMERx(x=0~4,7,14~16,22,23,40~44)
TIMER_DMACFG_DMATA_INTF	DMA传输起始地址: TIMER_INTF, TIMERx(x=0~4,7,14~16,22,23,40~44)
TIMER_DMACFG_DMATA_SWEVG	DMA传输起始地址: TIMER_SWEVG, TIMERx(x=0~4,7,14~16,22,23,40~44)
TIMER_DMACFG_DMATA_CHCTL0	DMA传输起始地址: TIMER_CHCTL0, TIMERx(x=0~4,7,14~16,22,23,40~44)
TIMER_DMACFG_DMATA_CHCTL1	DMA传输起始地址: TIMER_CHCTL1, TIMERx(x=0~4,7,22,23)
TIMER_DMACFG_DMATA_CHCTL2	DMA传输起始地址: TIMER_CHCTL2, TIMERx(x=0~4,7,14~16,22,23,40~44)
TIMER_DMACFG_DMATA_CNT	DMA传输起始地址: TIMER_CNT, TIMERx(x=0~4,7,14~16,22,23,40~44)
TIMER_DMACFG_DMATA_PSC	DMA传输起始地址: TIMER_PSC, TIMERx(x=0~4,7,14~16,22,23,40~44)

<code>TIMER_DMACFG_DMATA_CAR</code>	DMA传输起始地址: <code>TIMER_CAR</code> , <code>TIMERx</code> ($x=0\sim4,7,14\sim16,22,23,40\sim44$)
<code>TIMER_DMACFG_DMATA_CREP0</code>	DMA传输起始地址: <code>TIMER_CREP0</code> , <code>TIMERx</code> ($x=0,7,14\sim16,40\sim44$)
<code>TIMER_DMACFG_DMATA_CH0CV</code>	DMA传输起始地址: <code>TIMER_CH0CV</code> , <code>TIMERx</code> ($x=0\sim4,7,14\sim16,22,23,40\sim44$)
<code>TIMER_DMACFG_DMATA_CH1CV</code>	DMA传输起始地址: <code>TIMER_CH1CV</code> , <code>TIMERx</code> ($x=0\sim4,7,14,22,23$)
<code>TIMER_DMACFG_DMATA_CH2CV</code>	DMA传输起始地址: <code>TIMER_CH2CV</code> , <code>TIMERx</code> ($x=0\sim4,7,22,23$)
<code>TIMER_DMACFG_DMATA_CH3CV</code>	DMA传输起始地址: <code>TIMER_CH3CV</code> , <code>TIMERx</code> ($x=0\sim4,7,22,23$)
<code>TIMER_DMACFG_DMATA_CCHP</code>	DMA传输起始地址: <code>TIMER_CCHP</code> , <code>TIMERx</code> ($x=0,7,14\sim16,40\sim44$)
<code>TIMER_DMACFG_DMATA_MCHCTL0</code>	DMA传输起始地址: <code>TIMER_MCHCTL0</code> , <code>TIMERx</code> ($x=0,7,14\sim16,40\sim44$)
<code>TIMER_DMACFG_DMATA_MCHCTL1</code>	DMA传输起始地址: <code>TIMER_MCHCTL1</code> , <code>TIMERx</code> ($x=0,7$)
<code>TIMER_DMACFG_DMATA_MCHCTL2</code>	DMA传输起始地址: <code>TIMER_MCHCTL2</code> , <code>TIMERx</code> ($x=0,7,14\sim16,40\sim44$)
<code>TIMER_DMACFG_DMATA_MCH0CV</code>	DMA传输起始地址: <code>TIMER_MCH0CV</code> , <code>TIMERx</code> ($x=0,7,14\sim16,40\sim44$)
<code>TIMER_DMACFG_DMATA_MCH1CV</code>	DMA传输起始地址: <code>TIMER_MCH1CV</code> , <code>TIMERx</code> ($x=0,7$)
<code>TIMER_DMACFG_DMATA_MCH2CV</code>	DMA传输起始地址: <code>TIMER_MCH2CV</code> , <code>TIMERx</code> ($x=0,7$)
<code>TIMER_DMACFG_DMATA_MCH3CV</code>	DMA传输起始地址: <code>TIMER_MCH3CV</code> , <code>TIMERx</code> ($x=0,7$)
<code>TIMER_DMACFG_DMATA_CH0COMV_ADD</code>	DMA传输起始地址: <code>TIMER_CH0COMV_ADD</code> , <code>TIMERx</code> ($x=0\sim4,7,14,22,23$)
<code>TIMER_DMACFG_DMATA_CH1COMV_ADD</code>	DMA传输起始地址: <code>TIMER_CH1COMV_ADD</code> , <code>TIMERx</code> ($x=0\sim4,7,14,22,23$)
<code>TIMER_DMACFG_DMATA_CH2COMV_ADD</code>	DMA传输起始地址: <code>TIMER_CH2COMV_ADD</code> , <code>TIMERx</code> ($x=0\sim4,7,22,23$)
<code>TIMER_DMACFG_DMATA_CH3COMV_ADD</code>	DMA传输起始地址: <code>TIMER_CH3COMV_ADD</code> , <code>TIMERx</code> ($x=0\sim4,7,22,23$)
<code>TIMER_DMACFG_DMATA_CTL2</code>	DMA传输起始地址: <code>TIMER_CTL2</code> , <code>TIMERx</code> ($x=0\sim4,7,14\sim16,22,23,40\sim44$)
<code>TIMER_DMACFG_DMATA_FCCHP0</code>	DMA传输起始地址: <code>TIMER_FCCHP0</code> , <code>TIMERx</code> ($x=0,7,14\sim16,40\sim44$)

<i>DMATA_FCCHP0</i>	
<i>TIMER_DMACFG_DMATA_FCCHP1</i>	DMA传输起始地址: TIMER_FCCHP1, TIMERx(x=0,7)
<i>TIMER_DMACFG_DMATA_FCCHP2</i>	DMA传输起始地址: TIMER_FCCHP2, TIMERx(x=0,7)
<i>TIMER_DMACFG_DMATA_FCCHP3</i>	DMA传输起始地址: TIMER_FCCHP3, TIMERx(x=0,7)
<i>TIMER_DMACFG_DMATA_AFCTL0</i>	DMA传输起始地址: TIMER_AFCTL0, TIMERx(x=0,7,14~16,40~44)
<i>TIMER_DMACFG_DMATA_AFCTL1</i>	DMA传输起始地址: TIMER_AFCTL1, TIMERx(x=0,7)
<i>TIMER_DMACFG_DMATA_WDGCNT</i>	DMA传输起始地址: TIMER_WDGCNT, TIMERx(x=0~4,7,22,23)
<i>TIMER_DMACFG_DMATA_CREP1</i>	DMA传输起始地址: TIMER_CREP1, TIMERx(x=0,7,14~16,40~44)
输入参数{in}	
dma_lenth	DMA传输长度
<i>TIMER_DMACFG_DMATC_xTRANSFER</i>	(x=1~38), DMA传输x次
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* configure the TIMER0 DMA transfer */
```

```
timer_dma_transfer_config(TIMER0, TIMER_DMACFG_DMATA_CTL0,  
TIMER_DMACFG_DMATC_5TRANSFER);
```

函数 timer_event_software_generate

函数timer_event_software_generate描述见下表:

表 3-1161. 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
<i>TIMERx(x=0~7,14~16,40~44)</i>	参考具体参数

16,22,23,40~44,50,51)	
输入参数{in}	
event	事件源
TIMER_EVENT_SR_C_UPG	更新事件产生, TIMERx(x=0~7,14~16,22,23,40~44,50,51)
TIMER_EVENT_SR_C_CH0G	通道0捕获或比较事件发生, TIMERx(x=0~4,7,14~16,22,23,40~44)
TIMER_EVENT_SR_C_CH1G	通道1捕获或比较事件发生, TIMERx(x=0~4,7,14,22,23,40~44)
TIMER_EVENT_SR_C_CH2G	通道2捕获或比较事件发生, TIMERx(x=0~4,7,22,23)
TIMER_EVENT_SR_C_CH3G	通道3捕获或比较事件发生, TIMERx(x=0~4,7,22,23)
TIMER_EVENT_SR_C_CMTG	通道换相更新事件发生, TIMERx(x=0,7,14~16,40~44)
TIMER_EVENT_SR_C_TRGG	触发事件产生, IMERx(x=0~4,7,14,22,23,40~44)
TIMER_EVENT_SR_C_BRK0G	产生BREAK0事件, TIMERx(x=0,7,14~16,40~44)
TIMER_EVENT_SR_C_BRK1G	产生BREAK1事件, TIMERx(x=0,7)
TIMER_EVENT_SR_C_MCH0G	多模式通道0捕获或比较事件发生, TIMERx(x=0,7,14~16,40~44)
TIMER_EVENT_SR_C_MCH1G	多模式通道1捕获或比较事件发生, TIMERx(x=0,7)
TIMER_EVENT_SR_C_MCH2G	多模式通道2捕获或比较事件发生, TIMERx(x=0,7)
TIMER_EVENT_SR_C_MCH3G	多模式通道3捕获或比较事件发生, TIMERx(x=0,7)
TIMER_EVENT_SR_C_CH0COMADDG	通道0附加比较事件发生, TIMERx(x=0~4,7,14,22,23)
TIMER_EVENT_SR_C_CH1COMADDG	通道1附加比较事件发生, TIMERx(x=0~4,7,14,22,23)
TIMER_EVENT_SR_C_CH2COMADDG	通道2附加比较事件发生, TIMERx(x=0~4,7,22,23)
TIMER_EVENT_SR_C_CH3COMADDG	通道3附加比较事件发生, TIMERx(x=0~4,7,22,23)
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 3-1162. 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 3-1163. 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=0,7,14~16,40~44)	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_CCHP_PROT_OFF;

timer_breakpara.break0state      = TIMER_BREAK0_ENABLE;

timer_breakpara.break0filter     = 0U;

timer_breakpara.break0polarity   = TIMER_BREAK0_POLARITY_LOW;

timer_breakpara.break0bidirectional = TIMER_BREAK0_LK_ENABLE;

timer_breakpara.break0release    = TIMER_BREAK0_UNRELEASE;

timer_breakpara.break1state      = TIMER_BREAK1_DISABLE;

timer_breakpara.break1filter     = 0U;

timer_breakpara.break1polarity   = TIMER_BREAK1_POLARITY_LOW;

timer_breakpara.break1bidirectional = TIMER_BREAK1_LK_DISABLE;

timer_breakpara.break1release    = TIMER_BREAK1_UNRELEASE;

timer_break_config(TIMER0, &timer_breakpara);
```

函数 timer_break_enable

函数timer_break_enable描述见下表：

表 3-1164. 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_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
<i>TIMERx</i> (<i>x</i> =0,7,14~16,40~44)	TIMER外设选择
输入参数{in}	
break_num	TIMER BREAKx
<i>TIMER_BREAK0</i>	BREAK0输入信号, <i>TIMERx</i> (<i>x</i> =0,7,14~16,40~44)
<i>TIMER_BREAK1</i>	BREAK1输入信号, <i>TIMERx</i> (<i>x</i> =0,7)
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* enable TIMER0 BREAK0 function*/
```

```
timer_break_enable(TIMER0, TIMER_BREAK0);
```

函数 timer_break_disable

函数timer_break_disable描述见下表:

表 3-1165. 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
<i>TIMERx</i> (<i>x</i> =0,7,14~16,40~44)	TIMER外设选择
输入参数{in}	
break_num	TIMER BREAKx
<i>TIMER_BREAK0</i>	BREAK0输入信号, <i>TIMERx</i> (<i>x</i> =0,7,14~16,40~44)
<i>TIMER_BREAK1</i>	BREAK1输入信号, <i>TIMERx</i> (<i>x</i> =0,7)
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* disable TIMER0 BREAK0 function*/
```



```
timer_break_disable(TIMER0, TIMER_BREAK0);
```

函数 timer_automatic_output_enable

函数timer_automatic_output_enable描述见下表：

表 3-1166. 函数 timer_automatic_output_enable

函数名称	timer_automatic_output_enable
函数原型	void timer_automatic_output_enable(uint32_t timer_periph);
功能描述	自动输出使能
先决条件	只有在TIMERx_CCHP寄存器的PROT [1:0] =00 时，才可修改
被调用函数	-
输入参数{in}	
timer_periph	TIMER外设
TIMERx(x=0,7,14~16,40~44)	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 3-1167. 函数 timer_automatic_output_disable

函数名称	timer_automatic_output_disable
函数原型	void timer_automatic_output_disable (uint32_t timer_periph);
功能描述	自动输出禁能
先决条件	只有在TIMERx_CCHP寄存器的PROT [1:0] = 00时，才可修改
被调用函数	-
输入参数{in}	
timer_periph	TIMER外设
TIMERx(x=0,7,14~16,40~44)	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 3-1168. 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=0,7,14~16,40~44)	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 3-1169. 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
TIMEx(x=0,7,14~16,40~44)	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 3-1170. 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
TIMEx(x=0,7,14~16,40~44)	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_CCUNDER	当计数器下溢事件发生时, 影子寄存器更新
TIMER_UPDATECT	当计数器上溢/ 下溢事件发生时, 影子寄存器更新

L_CCUOVERUNDE R	
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 3-1171. 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 3-1172. 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=0~4,7,14~16,22,23,40~44)	参考具体参数
输入参数{in}	
channel	TIMER通道
TIMER_CH_0	通道0, TIMERx(x=0~4,7,14~16,22,23,40~44)
TIMER_CH_1	通道1, TIMERx(x=0~4,7,14,22,23,40~44)
TIMER_CH_2	通道2, TIMERx(x=0~4,7,22,23)
TIMER_CH_3	通道3, TIMERx(x=0~4,7,22,23)
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 3-1173. 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
TIMERx(x=0~4,7,14~16,22,23,40~44)	参考具体参数
输入参数{in}	
channel	TIMER通道
TIMER_CH_0	通道0, TIMERx(x=0~4,7,14~16,22,23,40~44)
TIMER_CH_1	通道1, TIMERx(x=0~4,7,14,22,23,40~44)
TIMER_CH_2	通道2, TIMERx(x=0~4,7,22,23)
TIMER_CH_3	通道3, TIMERx(x=0~4,7,22,23)
TIMER_MCH_0	多模式通道0, TIMERx (x=0,7,14~16,40~44)
TIMER_MCH_1	多模式通道1, TIMERx (x=0,7)
TIMER_MCH_2	多模式通道2, TIMERx (x=0,7)
TIMER_MCH_3	多模式通道3, TIMERx (x=0,7)
输入参数{in}	
ocmode	通道输出比较模式
TIMER_OC_MODE_TIMING	时基模式
TIMER_OC_MODE_ACTIVE	匹配时设置为高
TIMER_OC_MODE_INACTIVE	匹配时设置为低
TIMER_OC_MODE_TOGGLE	匹配时翻转
TIMER_OC_MODE_LOW	强制为低
TIMER_OC_MODE_HIGH	强制为高
TIMER_OC_MODE_PWM0	PWM模式0
TIMER_OC_MODE_PWM1	PWM模式1
TIMER_OC_MODE_DSPM0	可延时的单脉冲模式0
TIMER_OC_MODE_DSPM1	可延时的单脉冲模式1
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* configure TIMER0 channel PWM 0 mode */
```

```
timer_channel_output_mode_config(TIMER0,TIMER_CH_0,TIMER_OC_MODE_PWM0);
```

函数 timer_channel_output_pulse_value_config

函数timer_channel_output_pulse_value_config描述见下表：

表 3-1174. 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=0~4,7,14~16,22,23,40~44)	参考具体参数
输入参数{in}	
channel	TIMER通道
TIMER_CH_0	通道0, TIMERx(x=0~4,7,14~16,22,23,40~44)
TIMER_CH_1	通道1, TIMERx(x=0~4,7,14,22,23,40~44)
TIMER_CH_2	通道2, TIMERx(x=0~4,7,22,23)
TIMER_CH_3	通道3, TIMERx(x=0~4,7,22,23)
TIMER_MCH_0	多模式通道0, TIMERx (x=0,7,14~16,40~44)
TIMER_MCH_1	多模式通道1, TIMERx (x=0,7)
TIMER_MCH_2	多模式通道2, TIMERx (x=0,7)
TIMER_MCH_3	多模式通道3, TIMERx (x=0,7)
输入参数{in}	
pulse	通道输出比较值 (0~65535)
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* configure TIMER0 channel 0 output pulse value */
```

```
timer_channel_output_pulse_value_config(TIMER0, TIMER_CH_0, 399);
```

函数 timer_channel_output_shadow_config

函数timer_channel_output_shadow_config描述见下表:

表 3-1175. 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=0~4,7,14~16,22,23,40~44)	参考具体参数
输入参数{in}	
channel	TIMER通道
TIMER_CH_0	通道0, TIMERx(x=0~4,7,14~16,22,23,40~44)
TIMER_CH_1	通道1, TIMERx(x=0~4,7,14,22,23,40~44)
TIMER_CH_2	通道2, TIMERx(x=0~4,7,22,23)
TIMER_CH_3	通道3, TIMERx(x=0~4,7,22,23)
TIMER_MCH_0	多模式通道0, TIMERx (x=0,7,14~16,40~44)
TIMER_MCH_1	多模式通道1, TIMERx (x=0,7)
TIMER_MCH_2	多模式通道2, TIMERx (x=0,7)
TIMER_MCH_3	多模式通道3, TIMERx (x=0,7)
输入参数{in}	
ocshadow	输出比较影子寄存器功能状态
TIMER_OC_SHADOW_ENABLE	通道输出比较影子寄存器使能
TIMER_OC_SHADOW_DISABLE	通道输出比较影子寄存器禁能
TIMER_OMC_SHADOW_ENABLE	多模式通道输出比较影子寄存器使能
TIMER_OMC_SHADOW_DISABLE	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,
                                   817
```


TIMER_OC_SHADOW_ENABLE);

函数 timer_channel_output_clear_config

函数timer_channel_output_clear_config描述见下表:

表 3-1176. 函数 timer_channel_output_clear_config

函数名称	timer_channel_output_clear_config
函数原型	void timer_channel_output_clear_config(uint32_t timer_periph, uint16_t channel, uint16_t occlear);
功能描述	配置TIMER的通道输出比较清0功能
先决条件	-
被调用函数	-
输入参数{in}	
timer_periph	TIMER外设
TIMERx(x=0~4,7,14~16,22,23,40~44)	参考具体参数
输入参数{in}	
channel	TIMER通道
TIMER_CH_0	通道0, TIMERx(x=0~4,7,14~16,22,23,40~44)
TIMER_CH_1	通道1, TIMERx(x=0~4,7,14,22,23,40~44)
TIMER_CH_2	通道2, TIMERx(x=0~4,7,22,23)
TIMER_CH_3	通道3, TIMERx(x=0~4,7,22,23)
TIMER_MCH_0	多模式通道0, TIMERx (x=0,7,14~16,40~44)
TIMER_MCH_1	多模式通道1, TIMERx (x=0,7)
TIMER_MCH_2	多模式通道2, TIMERx (x=0,7)
TIMER_MCH_3	多模式通道3, TIMERx (x=0,7)
输入参数{in}	
occlear	通道比较输出清0功能状态
TIMER_OC_CLEAR_ENABLE	通道比较输出清0功能使能
TIMER_OC_CLEAR_DISABLE	通道比较输出清0功能禁能
TIMER_OMC_CLEAR_ENABLE	多模式通道比较输出清0功能使能
TIMER_OMC_CLEAR_DISABLE	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 3-1177. 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=0~4,7,14~16,22,23,40~44)	参考具体参数
输入参数{in}	
channel	TIMER通道
TIMER_CH_0	通道0, TIMERx(x=0~4,7,14~16,22,23,40~44)
TIMER_CH_1	通道1, TIMERx(x=0~4,7,14,22,23,40~44)
TIMER_CH_2	通道2, TIMERx(x=0~4,7,22,23)
TIMER_CH_3	通道3, TIMERx(x=0~4,7,22,23)
TIMER_MCH_0	多模式通道0, TIMERx (x=0,7,14~16,40~44)
TIMER_MCH_1	多模式通道1, TIMERx (x=0,7)
TIMER_MCH_2	多模式通道2, TIMERx (x=0,7)
TIMER_MCH_3	多模式通道3, TIMERx (x=0,7)
输入参数{in}	
ocpolarity	通道输出极性
TIMER_OC_POLARITY_HIGH	通道输出极性高电平有效
TIMER_OC_POLARITY_LOW	通道输出极性低电平有效
TIMER_OMC_POLARITY_HIGH	多模式通道输出极性高电平有效
TIMER_OMC_POLARITY_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 3-1178. 函数 timer_channel_complementary_output_polarity_config

函数名称	timer_channel_complementary_output_polarity_config
函数原型	void timer_channel_complementary_output_polarity_config(uint32_t timer_periph, uint16_t channel, uint16_t ocpolarity);
功能描述	互补通道输出极性配置
先决条件	-
被调用函数	-
输入参数{in}	
timer_periph	TIMER外设
TIMERx(x=0~4,7,14~16,22,23,40~44)	TIMER外设选择
输入参数{in}	
channel	TIMER通道
TIMER_CH_0	通道0, TIMERx(x=0~4,7,14~16,22,23,40~44)
TIMER_CH_1	通道1, TIMERx(x=0~4,7,14,22,23,40~44)
TIMER_CH_2	通道2, TIMERx(x=0~4,7,22,23)
TIMER_CH_3	通道3, TIMERx(x=0~4,7,22,23)
输入参数{in}	
ocpolarity	互补通道输出极性
TIMER_OCN_POLARITY_HIGH	互补通道输出极性高电平有效
TIMER_OCN_POLARITY_LOW	互补通道输出极性低电平有效
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* configure TIMER0 channel 0 complementary output polarity */
```

```
timer_channel_complementary_output_polarity_config(TIMER0, TIMER_CH_0,  
TIMER_OCN_POLARITY_HIGH);
```

函数 timer_channel_output_state_config

函数timer_channel_output_state_config描述见下表:

表 3-1179. 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=0~4,7,14~16,22,23,40~44)	参考具体参数
输入参数{in}	
channel	TIMER通道
TIMER_CH_0	通道0, TIMERx(x=0~4,7,14~16,22,23,40~44)
TIMER_CH_1	通道1, TIMERx(x=0~4,7,14,22,23,40~44)
TIMER_CH_2	通道2, TIMERx(x=0~4,7,22,23)
TIMER_CH_3	通道3, TIMERx(x=0~4,7,22,23)
TIMER_MCH_0	多模式通道0, TIMERx (x=0,7,14~16,40~44)
TIMER_MCH_1	多模式通道1, TIMERx (x=0,7)
TIMER_MCH_2	多模式通道2, TIMERx (x=0,7)
TIMER_MCH_3	多模式通道3, TIMERx (x=0,7)
输入参数{in}	
state	通道状态
TIMER_CCX_ENABLE	通道使能
TIMER_CCX_DISABLE	通道禁能
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 3-1180. 函数 `timer_channel_complementary_output_state_config`

函数名称	<code>timer_channel_complementary_output_state_config</code>
函数原型	<code>void timer_channel_complementary_output_state_config(uint32_t timer_periph, uint16_t channel, uint16_t ocnstate);</code>
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}	
<code>timer_periph</code>	TIMER外设
<code>TIMERx(x=0,7,14~16,40~44)</code>	TIMER外设选择
输入参数{in}	
<code>channel</code>	TIMER通道
<code>TIMER_CH_0</code>	通道0, <code>TIMERx(x=0,7,14~16,40~44)</code>
<code>TIMER_CH_1</code>	通道1, <code>TIMERx(x=0,7,14,40~44)</code>
<code>TIMER_CH_2</code>	通道2, <code>TIMERx(x=0,7)</code>
<code>TIMER_CH_3</code>	通道3, <code>TIMERx(x=0,7)</code>
输入参数{in}	
<code>state</code>	互补通道状态
<code>TIMER_CCXN_ENABLER</code>	互补通道使能
<code>TIMER_CCXN_DISABLE</code>	互补通道禁能
输出参数{out}	
-	-
返回值	
-	-

例如:

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

函数 `timer_channel_input_struct_para_init`

函数 `timer_channel_input_struct_para_init` 描述见下表:

表 3-1181. 函数 `timer_channel_input_struct_para_init`

函数名称	<code>timer_channel_input_struct_para_init</code>
函数原型	<code>void timer_channel_input_struct_para_init(timer_ic_parameter_struct* icpara);</code>

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 3-1182. 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=0~4,7,14~16,22,23,40~44)	参考具体参数
输入参数{in}	
channel	TIMER通道
TIMER_CH_0	通道0, TIMERx(x=0~4,7,14~16,22,23,40~44)
TIMER_CH_1	通道1, TIMERx(x=0~4,7,14,22,23,40~44)
TIMER_CH_2	通道2, TIMERx(x=0~4,7,22,23)
TIMER_CH_3	通道3, TIMERx(x=0~4,7,22,23)
TIMER_MCH_0	多模式通道0, TIMERx (x=0,7,14~16,40~44)
TIMER_MCH_1	多模式通道1, TIMERx (x=0,7)
TIMER_MCH_2	多模式通道2, TIMERx (x=0,7)
TIMER_MCH_3	多模式通道3, TIMERx (x=0,7)
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.icselection = 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 3-1183. 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=0~4,7,14~16,22,23,40~44)	参考具体参数
输入参数{in}	
channel	TIMER通道
TIMER_CH_0	通道0, TIMERx(x=0~4,7,14~16,22,23,40~44)
TIMER_CH_1	通道1, TIMERx(x=0~4,7,14,22,23,40~44)
TIMER_CH_2	通道2, TIMERx(x=0~4,7,22,23)
TIMER_CH_3	通道3, TIMERx(x=0~4,7,22,23)
TIMER_MCH_0	多模式通道0, TIMERx (x=0,7,14~16,40~44)
TIMER_MCH_1	多模式通道1, TIMERx (x=0,7)
TIMER_MCH_2	多模式通道2, TIMERx (x=0,7)
TIMER_MCH_3	多模式通道3, TIMERx (x=0,7)
输入参数{in}	

prescaler	通道输入捕获预分频值
<i>TIMER_IC_PSC_DIV1</i>	不分频
<i>TIMER_IC_PSC_DIV2</i>	2分频
<i>TIMER_IC_PSC_DIV4</i>	4分频
<i>TIMER_IC_PSC_DIV8</i>	8分频
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* configure TIMER0 channel 0 input capture prescaler value */
timer_channel_input_capture_prescaler_config(TIMER0, TIMER_CH_0,
TIMER_IC_PSC_DIV2);
```

函数 timer_channel_capture_value_register_read

函数timer_channel_capture_value_register_read描述见下表：

表 3-1184. 函数 timer_channel_capture_value_register_read

函数名称	timer_channel_capture_value_register_read
函数原型	uint32_t timer_channel_capture_value_register_read(uint32_t timer_periph, uint16_t channel);
功能描述	读取通道捕获值
先决条件	-
被调用函数	-
输入参数{in}	
timer_periph	TIMER外设
<i>TIMERx</i> (<i>x</i> =0~4, 7, 14~16, 22, 23, 40~44)	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, 14~16, 22, 23, 40~44)
<i>TIMER_CH_1</i>	通道1, <i>TIMERx</i> (<i>x</i> =0~4, 7, 14, 22, 23, 40~44)
<i>TIMER_CH_2</i>	通道2, <i>TIMERx</i> (<i>x</i> =0~4, 7, 22, 23)
<i>TIMER_CH_3</i>	通道3, <i>TIMERx</i> (<i>x</i> =0~4, 7, 22, 23)
<i>TIMER_MCH_0</i>	多模式通道0, <i>TIMERx</i> (<i>x</i> =0, 7, 14~16, 40~44)
<i>TIMER_MCH_1</i>	多模式通道1, <i>TIMERx</i> (<i>x</i> =0, 7)
<i>TIMER_MCH_2</i>	多模式通道2, <i>TIMERx</i> (<i>x</i> =0, 7)

<i>TIMER_MCH_3</i>	多模式通道3, TIMERx (x=0,7)
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 3-1185. 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
<i>TIMERx</i> (x=0~4, 7, 14~16, 22, 23, 40~44)	TIMER外设选择
输入参数{in}	
channel	TIMER通道
<i>TIMER_CH_0</i>	通道0
<i>TIMER_CH_1</i>	通道1
输入参数{in}	
icpwm	输入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_icinitpara;

timer_icinitpara.icpolarity = TIMER_IC_POLARITY_RISING;
```

```

timer_icinitpara.icselection = TIMER_IC_SELECTION_DIRECTTTI;

timer_icinitpara.icprescaler = TIMER_IC_PSC_DIV1;

timer_icinitpara.icfilter      = 0x0;

timer_input_pwm_capture_config(TIMER0, TIMER_CH_0, &timer_icinitpara);

```

函数 timer_hall_mode_config

函数timer_hall_mode_config描述见下表:

表 3-1186. 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
TIMERx(x=0~4,7,14 ~16,22,23,40~44)	TIMER外设选择
输入参数{in}	
hallmode	HALL接口功能状态
TIMER_HALLINTE RFACE_ENABLE	HALL接口使能
TIMER_HALLINTE RFACE_DISABLE	HALL接口禁能
输出参数{out}	
-	-
返回值	
-	-

例如:

```

/* configure TIMER0 hall sensor mode */

timer_hall_mode_config(TIMER0, TIMER_HALLINTERFACE_ENABLE);

```

函数 timer_multi_mode_channel_output_parameter_struct_init

函数timer_multi_mode_channel_output_parameter_struct_init描述见下表:

表 3-1187. 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_paramet

	er_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 3-1188. 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=0,7,14~16,40~44)	参考具体参数
输入参数{in}	
channel	TIMER通道
TIMER_MCH_0	多模式通道0，TIMERx (x=0,7,14~16,40~44)
TIMER_MCH_1	多模式通道1，TIMERx (x=0,7)
TIMER_MCH_2	多模式通道2，TIMERx (x=0,7)
TIMER_MCH_3	多模式通道3，TIMERx (x=0,7)
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 TIMER0 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(TIMER0,          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 3-1189. 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
TIMERx(x=0,7,14~16,40~44)	参考具体参数
输入参数{in}	
channel	TIMER通道
TIMER_MCH_0	多模式通道0, TIMERx (x=0,7,14~16,40~44)
TIMER_MCH_1	多模式通道1, TIMERx (x=0,7)
TIMER_MCH_2	多模式通道2, TIMERx (x=0,7)
TIMER_MCH_3	多模式通道3, TIMERx (x=0,7)
输入参数{in}	
ocmode	通道输出比较模式
TIMER_MCH_MODE_INDEPENDENTLY	多模式通道为独立模式
TIMER_MCH_MODE_COMPLEMENTARY	多模式通道为互补模式
输出参数{out}	

-	-
返回值	
-	-

例如：

```
/* select TIMER0 multi mode channel 0 mode */
```

```
timer_multi_mode_channel_mode_config      (TIMER0,          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 3-1190. 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=0~4,7,14,22,23,40~44)	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=0~4,7,14,22,23,40~44))
TIMER_SMCFG_T RGSEL_ITI1	内部触发输入1(ITI1, TIMERx(x=0~4,7,14,22,23,40~44))
TIMER_SMCFG_T RGSEL_ITI2	内部触发输入2(ITI2, TIMERx(x=0~4,7,14,22,23,40~44))
TIMER_SMCFG_T RGSEL_ITI3	内部触发输入3(ITI3, TIMERx(x=0~4,7,14,22,23,40~44))
TIMER_SMCFG_T RGSEL_CIOF_ED	TI0的边沿检测(CIOF_ED, TIMERx(x=0~4,7,14,22,23,40~44))
TIMER_SMCFG_T RGSEL_CIOFE0	滤波后的通道0输入(CIOFE0, TIMERx(x=0~4,7,14,22,23,40~44))
TIMER_SMCFG_T RGSEL_C1FE1	滤波后的通道1输入(C1FE1, TIMERx(x=0~4,7,14,22,23,40~44))
TIMER_SMCFG_T RGSEL_ETIFP	滤波后的外部触发输入(ETIFP, TIMERx(x=0~4,7,22,23))
TIMER_SMCFG_T	滤波后的通道2输入(TIMERx(x=0,7))

<i>RGSEL_CI2FE2</i>	
<i>TIMER_SMCFG_T</i> <i>RGSEL_CI3FE3</i>	滤波后的通道3输入(TIMERx(x=0,7))
<i>TIMER_SMCFG_T</i> <i>RGSEL_MCI0FEM0</i>	滤波后的多模式通道0输入(TIMERx(x=0,7,14,40~44))
<i>TIMER_SMCFG_T</i> <i>RGSEL_MCI1FEM1</i>	滤波后的多模式通道1输入(TIMERx(x=0,7))
<i>TIMER_SMCFG_T</i> <i>RGSEL_MCI2FEM2</i>	滤波后的多模式通道2输入(TIMERx(x=0,7))
<i>TIMER_SMCFG_T</i> <i>RGSEL_MCI3FEM3</i>	滤波后的多模式通道3输入(TIMERx(x=0,7))
<i>TIMER_L0_SMCFG</i> <i>_TRGSEL_ITI4</i>	内部触发输入4对于通用L0定时器(TIMERx(x=1,2,22,23))
<i>TIMER_L0_SMCFG</i> <i>_TRGSEL_ITI5</i>	内部触发输入5对于通用L0定时器(TIMERx(x=1,22,23))
<i>TIMER_L0_SMCFG</i> <i>_TRGSEL_ITI6</i>	内部触发输入6对于通用L0定时器
<i>TIMER_L0_SMCFG</i> <i>_TRGSEL_ITI7</i>	内部触发输入7对于通用L0定时器
<i>TIMER_L0_SMCFG</i> <i>_TRGSEL_ITI8</i>	内部触发输入8对于通用L0定时器
<i>TIMER_L0_SMCFG</i> <i>_TRGSEL_ITI9</i>	内部触发输入9对于通用L0定时器(TIMERx(x=22,23))
<i>TIMER_L0_SMCFG</i> <i>_TRGSEL_ITI10</i>	内部触发输入10对于通用L0定时器(TIMERx(x=22,23))
<i>TIMER_L0_SMCFG</i> <i>_TRGSEL_ITI11</i>	内部触发输入11对于通用L0定时器(TIMERx(x=22,23))
<i>TIMER_SMCFG_T</i> <i>RGSEL_ITI12</i>	内部触发输入12(ITI12, TIMERx(x=0~4,7,23))
<i>TIMER_SMCFG_T</i> <i>RGSEL_ITI13</i>	内部触发输入13(ITI13, TIMERx(x=0~4,7,22))
<i>TIMER_SMCFG_T</i> <i>RGSEL_ITI14</i>	内部触发输入14(ITI14, TIMERx(x=0~4,7,14,22,23,40~44))
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_output0_trigger_source_select

函数timer_master_output0_trigger_source_select描述见下表:

表 3-1191. 函数 timer_master_output0_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, 14, 22, 23, 50, 51)	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, 14, 22, 23, 50, 51))
TIMER_TRI_OUT0_SRC_ENABLE	TIMER使能信号作为TRGO0(TIMERx(x=0~7, 14, 22, 23, 50, 51))
TIMER_TRI_OUT0_SRC_UPDATE	更新事件作为TRGO0 (TIMERx(x=0~7, 14, 22, 23, 50, 51))
TIMER_TRI_OUT0_SRC_CH0	通道0捕获/比较事件作为TRGO0 (TIMERx(x=0~4, 7, 14, 22, 23, 40~44))
TIMER_TRI_OUT0_SRC_O0CPRE	O0CPRE作为触发输出TRGO0(TIMERx(x=0~4, 7, 14, 22, 23, 40~44))
TIMER_TRI_OUT0_SRC_O1CPRE	O1CPRE作为触发输出TRGO0(TIMERx(x=0~4, 7, 14, 22, 23, 40~44))
TIMER_TRI_OUT0_SRC_O2CPRE	O2CPRE作为触发输出TRGO0(TIMERx(x=0~4, 7, 22, 23))
TIMER_TRI_OUT0_SRC_O3CPRE	O3CPRE作为触发输出TRGO0(TIMERx(x=0~4, 7, 22, 23))
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* select TIMER0 master mode output 0 trigger source */
```

```
timer_master_output0_trigger_source_select(TIMER0, TIMER_TRI_OUT0_SRC_RESET);
```

函数 timer_master_output1_trigger_source_select

函数timer_master_output1_trigger_source_select描述见下表:

表 3-1192. 函数 timer_master_output1_trigger_source_select

函数名称	timer_master_output1_trigger_source_select
函数原型	void timer_master_output1_trigger_source_select(uint32_t timer_periph, uint32_t outrigger);
功能描述	选择TIMER主模式输出1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)	TIMER外设选择
输入参数{in}	
outrigger	输出触发源
TIMER_TRI_OUT1_SRC_RESET	UPG位作为TRGO1
TIMER_TRI_OUT1_SRC_ENABLE	TIMER使能信号作为TRGO1
TIMER_TRI_OUT1_SRC_UPDATE	更新事件作为TRGO1
TIMER_TRI_OUT1_SRC_CH0	通道0捕获/比较事件作为TRGO1
TIMER_TRI_OUT1_SRC_O0CPRE	O0CPRE作为触发输出TRGO1
TIMER_TRI_OUT1_SRC_O1CPRE	O1CPRE作为触发输出TRGO1
TIMER_TRI_OUT1_SRC_O2CPRE	O2CPRE作为触发输出TRGO1
TIMER_TRI_OUT1_SRC_O3CPRE	O3CPRE作为触发输出TRGO1
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* select TIMER0 master mode output 1 trigger source */
```

```
timer_master_output1_trigger_source_select(TIMER0, TIMER_TRI_OUT1_SRC_RESET);
```


函数 timer_slave_mode_select

函数timer_slave_mode_select描述见下表:

表 3-1193. 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=0~4,7,14,22,23,40~44)	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=0~4,7,14,22,23,40~44)
TIMER_ENCODER_MODE0	编码器模式0, TIMERx(x=0~4,7,22,23)
TIMER_ENCODER_MODE1	编码器模式1, TIMERx(x=0~4,7,22,23)
TIMER_ENCODER_MODE2	编码器模式2, TIMERx(x=0~4,7,22,23)
TIMER_SLAVE_MODE_RESTART	复位模式, TIMERx(x=0~4,7,14,22,23,40~44)
TIMER_SLAVE_MODE_PAUSE	暂停模式, TIMERx(x=0~4,7,14,22,23,40~44)
TIMER_SLAVE_MODE_EVENT	事件模式, TIMERx(x=0~4,7,14,22,23,40~44)
TIMER_SLAVE_MODE_EXTERNAL0	外部时钟模式0, TIMERx(x=0~4,7,14,22,23,40~44)
TIMER_SLAVE_MODE_RESTART_EVENT	复位+事件模式, TIMERx(x=0~4,7,14,22,23,40~44)
TIMER_NONQUAD_MODE0	非正交编码器模式0, TIMERx(x=0~4,7,22,23)
TIMER_NONQUAD_MODE1	非正交编码器模式1, TIMERx(x=0~4,7,22,23)
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* select TIMER0 slave mode */
```

```
timer_slave_mode_select(TIMER0, TIMER_ENCODER_MODE0);
```

函数 timer_master_slave_mode_config

函数timer_master_slave_mode_config描述见下表：

表 3-1194. 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
TIMEx(x=0~4,7,14,22,23,40~44)	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 3-1195. 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=0~4,7,22,23)	TIMER外设选择
输入参数{in}	
extprescaler	外部触发预分频
TIMER_EXT_TRI_PSC_OFF	不分频
TIMER_EXT_TRI_PSC_DIV2	2分频
TIMER_EXT_TRI_PSC_DIV4	4分频
TIMER_EXT_TRI_PSC_DIV8	8分频
输入参数{in}	
expolarity	外部触发输入极性
TIMER_ETP_FALLING	低电平或者下降沿有效
TIMER_ETP_RISING	高电平或者上升沿有效
输入参数{in}	
extfilter	外部触发滤波控制（0~15）
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* configure TIMER0 external trigger input */
```

```
timer_external_trigger_config(TIMER0, TIMER_EXT_TRI_PSC_DIV2,  
TIMER_ETP_FALLING, 10);
```

函数 timer_quadrature_decoder_mode_config

函数timer_quadrature_decoder_mode_config描述见下表：

表 3-1196. 函数 timer_quadrature_decoder_mode_config

函数名称	timer_quadrature_decoder_mode_config
函数原型	void timer_quadrature_decoder_mode_config(uint32_t timer_periph, uint32_t

	decomode, uint16_t ic0polarity, uint16_t ic1polarity);
功能描述	TIMER配置为编码器模式
先决条件	-
被调用函数	-
输入参数{in}	
timer_periph	TIMER外设
TIMERx(x=0~4, 7, 22, 23)	TIMER外设选择
输入参数{in}	
decomode	编码器模式
TIMER_ENCODER_MODE0	根据CI0FE0的电平，计数器在CI1FE1的边沿向上/下计数
TIMER_ENCODER_MODE1	根据CI1FE1的电平，计数器在CI0FE0的边沿向上/下计数
TIMER_ENCODER_MODE2	根据另一个信号的输入电平，计数器在CI0FE0和CI1FE1的边沿向上/下计数
输入参数{in}	
ic0polarity	IC0输入极性
TIMER_IC_POLARITY_RISING	捕获上升边沿
TIMER_IC_POLARITY_FALLING	捕获下降边沿
TIMER_IC_POLARITY_BOTH_EDGE	双边沿有效
输入参数{in}	
ic1polarity	IC1输入极性
TIMER_IC_POLARITY_RISING	捕获上升边沿
TIMER_IC_POLARITY_FALLING	捕获下降边沿
TIMER_IC_POLARITY_BOTH_EDGE	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描述见下表:

表 3-1197. 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 decompode, uint16_t ic1polarity);
功能描述	TIMER配置为非正交编码器模式
先决条件	-
被调用函数	-
输入参数{in}	
timer_periph	TIMER外设
TIMERx(x=0~4, 7, 22, 23)	TIMER外设选择
输入参数{in}	
decompode	编码器模式
TIMER_NONQUAD_MODE0	非正交编码器模式0
TIMER_NONQUAD_MODE1	非正交编码器模式1
输入参数{in}	
ic1polarity	IC1输入极性
TIMER_IC_POLARITY_RISING	捕获上升边沿
TIMER_IC_POLARITY_FALLING	捕获下降边沿
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* configure TIMER0 non-quadrature decoder mode */
```

```
timer_non_quadrature_decoder_mode_config (TIMER0, TIMER_NONQUAD_MODE0,
TIMER_IC_POLARITY_RISING);
```

函数 timer_internal_clock_config

函数timer_internal_clock_config描述见下表:

表 3-1198. 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=0~4,7,14,22,23,40~44)	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 3-1199. 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=0~4,7,14,22,23,40~44)	TIMER外设选择
输入参数{in}	
intrigger	被选择的内部触发源
TIMER_SMCFG_TRGSEL_ITI0	内部触发输入0(ITI0, TIMERx(x=0~4,7,14,22,23,40~44))
TIMER_SMCFG_TRGSEL_ITI1	内部触发输入1(ITI1, TIMERx(x=0~4,7,14,22,23,40~44))
TIMER_SMCFG_TRGSEL_ITI2	内部触发输入2(ITI2, TIMERx(x=0~4,7,14,22,23,40~44))
TIMER_SMCFG_TRGSEL_ITI3	内部触发输入3(ITI3, TIMERx(x=0~4,7,14,22,23,40~44))
TIMER_L0_SMCFG_TRGSEL_ITI4	内部触发输入4对于通用L0定时器(TIMERx(x=1,2,22,23))

<code>TIMER_L0_SMCFG_TRGSEL_ITI5</code>	内部触发输入5对于通用L0定时器(TIMERx(x=1,22,23))
<code>TIMER_L0_SMCFG_TRGSEL_ITI7</code>	内部触发输入7对于通用L0定时器(TIMERx(x=4))
<code>TIMER_L0_SMCFG_TRGSEL_ITI9</code>	内部触发输入9对于通用L0定时器(TIMERx(x=22,23))
<code>TIMER_L0_SMCFG_TRGSEL_ITI10</code>	内部触发输入10对于通用L0定时器(TIMERx(x=22,23))
<code>TIMER_L0_SMCFG_TRGSEL_ITI11</code>	内部触发输入11对于通用L0定时器(TIMERx(x=22,23))
<code>TIMER_SMCFG_TRGSEL_ITI12</code>	内部触发输入12(ITI12, TIMERx(x=0~4,7,23))
<code>TIMER_SMCFG_TRGSEL_ITI13</code>	内部触发输入13(ITI13, TIMERx(x=0~4,7,22))
<code>TIMER_SMCFG_TRGSEL_ITI14</code>	内部触发输入14(ITI14, TIMERx(x=0~4,7,14,22,23,40~44))
输出参数{out}	
-	-
返回值	
-	-

例如:

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

```
timer_internal_trigger_as_external_clock_config(TIMER0, TIMER_SMCFG_TRGSEL_ITI0);
```

函数 timer_external_trigger_as_external_clock_config

函数timer_external_trigger_as_external_clock_config描述见下表:

表 3-1200. 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=0~4,7,14,22,23,40~44)	TIMER外设选择
输入参数{in}	
extrigger	外部触发源
TIMER_SMCFG_T	TIO的边沿检测(CIOF_ED, TIMERx(x=0~4,7,14,22,23,40~44))

<i>RGSEL_CIOF_ED</i>	
<i>TIMER_SMCFG_T RGSEL_CIOFE0</i>	滤波后的通道0输入(CIOFE0, TIMERx(x=0~4,7,14,22,23,40~44))
<i>TIMER_SMCFG_T RGSEL_C1FE1</i>	滤波后的通道1输入(C1FE1, TIMERx(x=0~4,7,14,22,23,40~44))
<i>TIMER_SMCFG_T RGSEL_C2FE2</i>	滤波后的通道2输入(TIMERx(x=0,7))
<i>TIMER_SMCFG_T RGSEL_C3FE3</i>	滤波后的通道3输入(TIMERx(x=0,7))
<i>TIMER_SMCFG_T RGSEL_MCIOFEM0</i>	滤波后的多模式通道0输入(TIMERx(x=0,7,14,40~44))
<i>TIMER_SMCFG_T RGSEL_MCI1FEM1</i>	滤波后的多模式通道1输入(TIMERx(x=0,7))
<i>TIMER_SMCFG_T RGSEL_MCI2FEM2</i>	滤波后的多模式通道2输入(TIMERx(x=0,7))
<i>TIMER_SMCFG_T RGSEL_MCI3FEM3</i>	滤波后的多模式通道3输入(TIMERx(x=0,7))
输入参数{in}	
expolarity	外部触发源极性
<i>TIMER_IC_POLARI TY_RISING</i>	外部触发源高电平或者上升沿有效
<i>TIMER_IC_POLARI TY_FALLING</i>	外部触发源低电平或者下降沿有效
<i>TIMER_IC_POLARI TY_BOTH_EDGE</i>	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 CIOFE0 as external clock input */
```

```
timer_external_trigger_as_external_clock_config(TIMER0,  
TIMER_SMCFG_TRGSEL_CIOFE0, TIMER_IC_POLARITY_RISING, 0);
```

函数 timer_external_clock_mode0_config

函数timer_external_clock_mode0_config描述见下表:

表 3-1201. 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=0~4,7,14,22,23,40~44)	TIMER外设选择		
输入参数{in}			
extprescaler	ETI触发源预分频值		
TIMER_EXT_TRI_PSC_OFF	不分频		
TIMER_EXT_TRI_PSC_DIV2	2分频		
TIMER_EXT_TRI_PSC_DIV4	4分频		
TIMER_EXT_TRI_PSC_DIV8	8分频		
输入参数{in}			
expolarity	ETI触发源极性		
TIMER_ETP_FALLING	下降沿或者低电平有效		
TIMER_ETP_RISING	上升沿或者高电平有效		
输入参数{in}			
extfilter	ETI触发源滤波参数（0~15）		
输出参数{out}			
-	-		
返回值			
-	-		

例如：

```
/* configure TIMER0 the external clock mode0 */
```

```
timer_external_clock_mode0_config(TIMER0, TIMER_EXT_TRI_PSC_DIV2,
TIMER_ETP_FALLING, 0);
```

函数 timer_external_clock_mode1_config

函数timer_external_clock_mode1_config描述见下表：

表 3-1202. 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=0~4,7,22,23)	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 3-1203. 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, 22, 23)	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 3-1204. 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, 14~16, 22, 23, 40~44)	TIMER外设选择
输入参数{in}	
ccsel	写CHxVAL寄存器选择位
TIMER_CHVSEL_DISABLE	无影响
TIMER_CHVSEL_ENABLE	当写入捕获比较寄存器的值与寄存器当前值相等时，写入操作无效。
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* configure TIMER0 write CHxVAL register selection */
```

```
timer_write_chxval_register_config(TIMER0, TIMER_CHVSEL_ENABLE);
```

函数 timer_output_value_selection_config

函数timer_output_value_selection_config描述见下表：

表 3-1205. 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=0,7,14~16,40~44)	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 3-1206. 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 ccsel);
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=0,7,14~16,40~44)	TIMER外设选择
输入参数{in}	
cssel	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 3-1207. 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=0~4,7,14,22,23,40~44)	参考具体参数
输入参数{in}	
channel	TIMER通道
TIMER_CH_0	通道0, TIMERx(x=0~4,7,14,22,23,40~44)
TIMER_CH_1	通道1, TIMERx(x=0~4,7,14,22,23,40~44)
TIMER_CH_2	通道2, TIMERx(x=0~4,7,22,23)
TIMER_CH_3	通道3, TIMERx(x=0~4,7,22,23)
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 3-1208. 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
TIMERx(x=0~4,7,14,22,23,40~44)	参考具体参数
输入参数{in}	
channel	TIMER通道
TIMER_CH_0	通道0，TIMERx(x=0~4,7,14,22,23,40~44)
TIMER_CH_1	通道1，TIMERx(x=0~4,7,14,22,23,40~44)
TIMER_CH_2	通道2，TIMERx(x=0~4,7,22,23)
TIMER_CH_3	通道3，TIMERx(x=0~4,7,22,23)
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 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 3-1209. 函数 timer_channel_composite_pwm_mode_output_pulse_value_config

函数名称	timer_channel_composite_pwm_mode_output_pulse_value_config
函数原型	void timer_channel_composite_pwm_mode_output_pulse_value_config(uint32_t timer_periph, uint32_t channel, uint32_t pulse, uint32_t add_pulse);
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
TIMERx(x=0~4,7,14,22,23,40~44)	参考具体参数
输入参数{in}	
channel	TIMER通道
TIMER_CH_0	通道0, TIMERx(x=0~4,7,14,22,23,40~44)
TIMER_CH_1	通道1, TIMERx(x=0~4,7,14,22,23,40~44)
TIMER_CH_2	通道2, TIMERx(x=0~4,7,22,23)
TIMER_CH_3	通道3, TIMERx(x=0~4,7,22,23)
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(TIMERO, 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 3-1210. 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
TIMEx(x=0~4,7,14,22,23,40~44)	参考具体参数
输入参数{in}	
channel	TIMER通道
TIMER_CH_0	通道0, TIMEx(x=0~4,7,14,22,23,40~44)
TIMER_CH_1	通道1, TIMEx(x=0~4,7,14,22,23,40~44)
TIMER_CH_2	通道2, TIMEx(x=0~4,7,22,23)
TIMER_CH_3	通道3, TIMEx(x=0~4,7,22,23)
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 TIMERO channel 0 additional compare value */
timer_channel_additional_compare_value_config (TIMERO, TIMER_CH_0, 399);
```

函数 timer_channel_additional_output_shadow_config

函数timer_channel_additional_output_shadow_config描述见下表:

表 3-1211. 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
TIMERx(x=0~4,7,14,22,23,40~44)	参考具体参数
输入参数{in}	
channel	TIMER通道
TIMER_CH_0	通道0, TIMERx(x=0~4,7,14,22,23,40~44)
TIMER_CH_1	通道1, TIMERx(x=0~4,7,14,22,23,40~44)
TIMER_CH_2	通道2, TIMERx(x=0~4,7,22,23)
TIMER_CH_3	通道3, TIMERx(x=0~4,7,22,23)
输入参数{in}	
aocshadow	通道附加输出比较影子寄存器状态
TIMER_ADD_SHADOW_ENABLE	通道附加输出比较影子寄存器使能
TIMER_ADD_SHADOW_DISABLE	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 3-1212. 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
TIMERx(x=0~4,7,14,22,23,40~44)	TIMER外设选择
输入参数{in}	

channel	TIMER通道
<i>TIMER_CH_0</i>	通道0, TIMERx(x=0~4,7,14,22,23,40~44)
<i>TIMER_CH_1</i>	通道1, TIMERx(x=0~4,7,14,22,23,40~44)
<i>TIMER_CH_2</i>	通道2, TIMERx(x=0~4,7,22,23)
<i>TIMER_CH_3</i>	通道3, TIMERx(x=0~4,7,22,23)
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 3-1213. 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, uint16_t break_num, 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=0,7,14~16,40~44)	参考具体参数
输入参数{in}	
break_num	TIMER BREAKx
TIMER_BREAK0	BREAK0输入信号, TIMERx(x=0,7,14~16,40~44)
TIMER_BREAK1	BREAK1输入信号, TIMERx(x=0,7)
输入参数{in}	
break_src	break source
TIMER_BRKIN0	BRKIN0备用功能输入使能, TIMERx(x=0,7,14~16,40~44)
TIMER_BRKIN1	BRKIN1备用功能输入使能, TIMERx(x=0,7)
TIMER_BRKIN2	BRKIN2备用功能输入使能, TIMERx(x=0,7)
TIMER_BRKCOMP0	CMP0输入使能, TIMERx(x=0,7,14~16,40~44)
TIMER_BRKCOMP1	CMP1输入使能, TIMERx(x=0,7,14~16,40~44)
TIMER_BRKHPDF	HPDF输入使能, TMERx(x=0,7,14~16,40~44)

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 break source */
```

```
timer_break_external_source_config(TIMER0,    TIMER_BREAK0,    TIMER_BRKIN0,
ENABLE);
```

函数 timer_break_external_polarity_config

函数timer_break_external_polarity_config描述见下表：

表 3-1214. 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
<i>TIMERx</i> (x=0,7,14~16,40~44)	参考具体参数
输入参数{in}	
break_num	TIMER BREAKx
<i>TIMER_BREAK0</i>	BREAK0输入信号，TIMERx(x=0,7,14~16,40~44)
<i>TIMER_BREAK1</i>	BREAK1输入信号，TIMERx(x=0,7)
输入参数{in}	
break_src	break source
<i>TIMER_BRKIN0</i>	BRKIN0备用功能输入使能，TIMERx(x=0,7,14~16,40~44)
<i>TIMER_BRKIN1</i>	BRKIN1备用功能输入使能，TIMERx(x=0,7)
<i>TIMER_BRKIN2</i>	BRKIN2备用功能输入使能，TIMERx(x=0,7)
<i>TIMER_BRKCOMP0</i>	CMP0输入使能，TIMERx(x=0,7,14~16,40~44)
<i>TIMER_BRKCOMP1</i>	CMP1输入使能，TIMERx(x=0,7,14~16,40~44)
输入参数{in}	
bkinpolarity	中止极性
<i>TIMER_BRKIN_PO</i>	低电平有效

LARITY_LOW	
TIMER_BRKIN_PO LARITY_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_BREAK0,    TIMER_BRKIN0,  
TIMER_BRKIN_POLARITY_HIGH);
```

函数 timer_break_lock_config

函数timer_break_lock_config描述见下表：

表 3-1215. 函数 timer_break_lock_config

函数名称	timer_break_lock_config
函数原型	void timer_break_lock_config(uint32_t timer_periph, uint16_t break_num, ControlStatus newvalue);
功能描述	配置TIMER锁存中止功能
先决条件	-
被调用函数	-
输入参数{in}	
timer_periph	TIMER外设
TIMERx(x=0,7,14~16,40~44)	参考具体参数
输入参数{in}	
break_num	TIMER BREAKx
TIMER_BREAK0	BREAK0输入信号，TIMERx(x=0,7,14~16,40~44)
TIMER_BREAK1	BREAK1输入信号，TIMERx(x=0,7)
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 TIMER break lock function */
```

```
timer_break_lock_config(TIMER0, TIMER_BREAK0, ENABLE);
```

函数 timer_break_lock_release_config

函数timer_break_lock_release_config描述见下表：

表 3-1216. 函数 timer_break_lock_release_config

函数名称	timer_break_lock_release_config
函数原型	void timer_break_lock_release_config(uint32_t timer_periph, uint16_t break_num, ControlStatus newvalue);
功能描述	配置TIMER锁存中止功能的释放功能
先决条件	-
被调用函数	-
输入参数{in}	
timer_periph	TIMER外设
TIMERx(x=0,7,14~16,40~44)	参考具体参数
输入参数{in}	
break_num	TIMER BREAKx
TIMER_BREAK0	BREAK0输入信号, TIMERx(x=0,7,14~16,40~44)
TIMER_BREAK1	BREAK1输入信号, TIMERx(x=0,7)
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：

```
/* release the TIMER break lock function */
```

```
timer_break_lock_release_config (TIMER0, TIMER_BREAK0, ENABLE);
```

函数 timer_channel_break_control_config

函数timer_channel_break_control_config描述见下表：

表 3-1217. 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 3-1218. 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
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 free dead time function */
```

```
timer_channel_dead_time_config (TIMER0, TIMER_CH_0, ENABLE);
```

函数 timer_free_complementary_struct_para_init

函数timer_free_complementary_struct_para_init描述见下表：

表 3-1219. 函数 timer_free_complementary_struct_para_init

函数名称	timer_free_complementary_struct_para_init
函数原型	void timer_free_complementary_struct_para_init(timer_free_complementary_parameter_struct *freecompara);
功能描述	将TIMER通道独立互补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}	
freecompara	独立互补参数结构体，详见 结构体 timer_free_complementary_parameter_struct 。
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* initialize TIMER channel free complementary parameter struct with a default value */
```

```
timer_free_complementary_parameter_struct timer_freecompara;
```

```
timer_free_complementary_struct_para_init (&timer_freecompara);
```

函数 timer_channel_free_complementary_config

函数timer_channel_free_complementary_config描述见下表：

表 3-1220. 函数 timer_channel_free_complementary_config

函数名称	timer_channel_free_complementary_config
函数原型	void timer_channel_free_complementary_config(uint32_t timer_periph, uint16_t channel, timer_free_complementary_parameter_struct *fcpara);
功能描述	配置TIMER独立互补通道保护功能
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}	
freecompara	独立互补参数结构体，详见 结构体 timer_free_complementary_parameter_struct 。
输出参数{out}	
-	-
返回值	
-	-

例如：

```

/* initialize TIMER break parameter struct with a default value */
timer_free_complementary_parameter_struct timer_freecompara;
timer_freecompara.freecomstate = TIMER_FCCHP_STATE_ENABLE;
timer_freecompara.runoffstate   = TIMER_ROS_STATE_ENABLE;
timer_freecompara.ideloffstate  = TIMER_IOS_STATE_ENABLE;
timer_freecompara.deadtime      = 255;
timer_channel_free_complementary_config(&timer_freecompara);

```

函数 timer_watchdog_value_config

函数timer_watchdog_value_config描述见下表：

表 3-1221. 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=0~4,7,22,23)	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 3-1222. 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=0~4,7,22,23)	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 3-1223. 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=0~4, 7, 22, 23)	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 3-1224. 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=0~4, 7, 22, 23)	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_upif_backup_config

函数timer_upif_backup_config描述见下表：

表 3-1225. 函数 timer_upif_backup_config

函数名称	timer_upif_backup_config
函数原型	void timer_upif_backup_config(uint32_t timer_periph, ControlStatus newvalue);
功能描述	配置UPIF位备份功能
先决条件	-
被调用函数	-
输入参数{in}	
timer_periph	TIMER外设
TIMERx(x=0~7, 14~16, 22, 23, 40~44, 50, 51)	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 UPIF bit backup function */
```

```
timer_upif_backup_config(TIMER0, ENABLE);
```

函数 timer_upifbu_bit_get

函数timer_upifbu_bit_get描述见下表：

表 3-1226. 函数 timer_upifbu_bit_get

函数名称	timer_upifbu_bit_get
函数原型	UPIFBUSStatus timer_upifbu_bit_get(uint32_t timer_periph);
功能描述	获取TIMERx_CNT寄存器中的UPIFBU位
先决条件	-
被调用函数	-
输入参数{in}	
timer_periph	TIMER外设
TIMERx(x=0~7,14~16,22,23,40~44,50,51)	TIMER外设选择
输出参数{out}	
-	-
返回值	
UPIFBUSStatus	UPIFBU位状态
VALID_SET	UPIFBU位有效, 返回值为1
VALID_RESET	UPIFBU位有效, 返回值为0
INVALID	UPIFBU位无效

例如:

```
/* get the UPIFBU bit in the TIMERx_CNT register */
```

```
UPIFBUSStatus upstatus = INVALID;
```

```
upstatus = timer_upifbu_bit_get (TIMER0);
```

函数 timer_flag_get

函数timer_flag_get描述见下表:

表 3-1227. 函数 timer_flag_get

函数名称	timer_flag_get
函数原型	FlagStatus timer_flag_get(uint32_t timer_periph, uint32_t flag);
功能描述	获取外设TIMERx的状态标志
先决条件	-
被调用函数	-
输入参数{in}	
timer_periph	TIMER外设
TIMERx(x=0~7,14~16,22,23,40~44,50,51)	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=0~7,14~16,22,23,40~44,50,51)

<i>TIMER_FLAG_CH0</i>	通道0比较/捕获标志, <i>TIMERx</i> (<i>x</i> =0~4,7,14~16,22,23,40~44)
<i>TIMER_FLAG_CH1</i>	通道1比较/捕获标志, <i>TIMERx</i> (<i>x</i> =0~4,7,14,22,23,40~44)
<i>TIMER_FLAG_CH2</i>	通道2比较/捕获标志, <i>TIMERx</i> (<i>x</i> =0~4,7,22,23)
<i>TIMER_FLAG_CH3</i>	通道3比较/捕获标志, <i>TIMERx</i> (<i>x</i> =0~4,7,22,23)
<i>TIMER_FLAG_CMT</i>	通道换相更新标志, <i>TIMERx</i> (<i>x</i> =0,7,14~16,40~44)
<i>TIMER_FLAG_TRG</i>	触发标志, <i>TIMERx</i> (<i>x</i> =0~4,7,14,22,23,40~44)
<i>TIMER_FLAG_BRK</i> 0	BREAK标志, <i>TIMERx</i> (<i>x</i> =0,7,14~16,40~44)
<i>TIMER_FLAG_BRK</i> 1	BREAK1标志, <i>TIMERx</i> (<i>x</i> =0,7)
<i>TIMER_FLAG_CH0</i> 0	通道0捕获溢出标志, <i>TIMERx</i> (<i>x</i> =0~4,7,14~16,22,23,40~44)
<i>TIMER_FLAG_CH1</i> 0	通道1捕获溢出标志, <i>TIMERx</i> (<i>x</i> =0~4,7,14,22,23,40~44)
<i>TIMER_FLAG_CH2</i> 0	通道2捕获溢出标志, <i>TIMERx</i> (<i>x</i> =0~4,7,22,23)
<i>TIMER_FLAG_CH3</i> 0	通道3捕获溢出标志, <i>TIMERx</i> (<i>x</i> =0~4,7,22,23)
<i>TIMER_FLAG_SYS</i> B	系统源中止标志, <i>TIMERx</i> (<i>x</i> =0,7)
<i>TIMER_FLAG_DEC</i> J	正交编码器跳变标志, <i>TIMERx</i> (<i>x</i> =0~4,7,22,23)
<i>TIMER_FLAG_DEC</i> DIS	正交编码器断线标志, <i>TIMERx</i> (<i>x</i> =0~4,7,22,23)
<i>TIMER_FLAG_MCH</i> 0	多模式通道0比较/捕获标志, <i>TIMERx</i> (<i>x</i> =0,7,14~16,40~44)
<i>TIMER_FLAG_MCH</i> 1	多模式通道1比较/捕获标志, <i>TIMERx</i> (<i>x</i> =0,7)
<i>TIMER_FLAG_MCH</i> 2	多模式通道2比较/捕获标志, <i>TIMERx</i> (<i>x</i> =0,7)
<i>TIMER_FLAG_MCH</i> 3	多模式通道3比较/捕获标志, <i>TIMERx</i> (<i>x</i> =0,7)
<i>TIMER_FLAG_MCH</i> 00	多模式通道0捕获溢出标志, <i>TIMERx</i> (<i>x</i> =0,7,14~16,40~44)
<i>TIMER_FLAG_MCH</i> 10	多模式通道1捕获溢出标志, <i>TIMERx</i> (<i>x</i> =0,7)
<i>TIMER_FLAG_MCH</i> 20	多模式通道2捕获溢出标志, <i>TIMERx</i> (<i>x</i> =0,7)
<i>TIMER_FLAG_MCH</i> 30	多模式通道3捕获溢出标志, <i>TIMERx</i> (<i>x</i> =0,7)
<i>TIMER_FLAG_CH0</i> COMADD	通道0附加比较标志, <i>TIMERx</i> (<i>x</i> =0~4,7,14,22,23)

<i>TIMER_FLAG_CH1</i> COMADD	通道1附加比较标志, TIMERx(x=0~4,7,14,22,23)
<i>TIMER_FLAG_CH2</i> COMADD	通道2附加比较标志, TIMERx(x=0~4,7,22,23)
<i>TIMER_FLAG_CH3</i> COMADD	通道3附加比较标志, TIMERx(x=0~4,7,22,23)
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 3-1228. 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=0~7,14~16,22,23,40~44,50,51)	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=0~7,14~16,22,23,40~44,50,51)
TIMER_FLAG_CH0	通道0比较/捕获标志, TIMERx(x=0~4,7,14~16,22,23,40~44)
TIMER_FLAG_CH1	通道1比较/捕获标志, TIMERx(x=0~4,7,14,22,23,40~44)
TIMER_FLAG_CH2	通道2比较/捕获标志, TIMERx(x=0~4,7,22,23)
TIMER_FLAG_CH3	通道3比较/捕获标志, TIMERx(x=0~4,7,22,23)
TIMER_FLAG_CMT	通道换相更新标志, TIMERx(x=0,7,14~16,40~44)
TIMER_FLAG_TRG	触发标志, TIMERx(x=0~4,7,14,22,23,40~44)
TIMER_FLAG_BRK 0	BREAK0标志, TIMERx(x=0,7,14~16,40~44)

<code>TIMER_FLAG_BRK</code> 1	BREAK1标志, <code>TIMERx(x=0,7)</code>
<code>TIMER_FLAG_CH0</code> 0	通道0捕获溢出标志, <code>TIMERx(x=0~4,7,14~16,22,23,40~44)</code>
<code>TIMER_FLAG_CH1</code> 0	通道1捕获溢出标志, <code>TIMERx(x=0~4,7,14,22,23,40~44)</code>
<code>TIMER_FLAG_CH2</code> 0	通道2捕获溢出标志, <code>TIMERx(x=0~4,7,22,23)</code>
<code>TIMER_FLAG_CH3</code> 0	通道3捕获溢出标志, <code>TIMERx(x=0~4,7,22,23)</code>
<code>TIMER_FLAG_SYS</code> B	系统源中止标志, <code>TIMERx(x=0,7)</code>
<code>TIMER_FLAG_DEC</code> J	正交编码器跳变标志, <code>TIMERx(x=0~4,7,22,23)</code>
<code>TIMER_FLAG_DEC</code> DIS	正交编码器断线标志, <code>TIMERx(x=0~4,7,22,23)</code>
<code>TIMER_FLAG_MCH</code> 0	多模式通道0比较/捕获标志, <code>TIMERx(x=0,7,14~16,40~44)</code>
<code>TIMER_FLAG_MCH</code> 1	多模式通道1比较/捕获标志, <code>TIMERx(x=0,7)</code>
<code>TIMER_FLAG_MCH</code> 2	多模式通道2比较/捕获标志, <code>TIMERx(x=0,7)</code>
<code>TIMER_FLAG_MCH</code> 3	多模式通道3比较/捕获标志, <code>TIMERx(x=0,7)</code>
<code>TIMER_FLAG_MCH</code> 00	多模式通道0捕获溢出标志, <code>TIMERx(x=0,7,14~16,40~44)</code>
<code>TIMER_FLAG_MCH</code> 10	多模式通道1捕获溢出标志, <code>TIMERx(x=0,7)</code>
<code>TIMER_FLAG_MCH</code> 20	多模式通道2捕获溢出标志, <code>TIMERx(x=0,7)</code>
<code>TIMER_FLAG_MCH</code> 30	多模式通道3捕获溢出标志, <code>TIMERx(x=0,7)</code>
<code>TIMER_FLAG_CH0</code> COMADD	通道0附加比较标志, <code>TIMERx(x=0~4,7,14,22,23)</code>
<code>TIMER_FLAG_CH1</code> COMADD	通道1附加比较标志, <code>TIMERx(x=0~4,7,14,22,23)</code>
<code>TIMER_FLAG_CH2</code> COMADD	通道2附加比较标志, <code>TIMERx(x=0~4,7,22,23)</code>
<code>TIMER_FLAG_CH3</code> COMADD	通道3附加比较标志, <code>TIMERx(x=0~4,7,22,23)</code>
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 3-1229. 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=0~7,14~16,22,23,40~44,50,51)	参考具体参数
输入参数{in}	
interrupt	中断源
TIMER_INT_UP	更新中断, TIMERx(x=0~7,14~16,22,23,40~44,50,51)
TIMER_INT_CH0	通道0比较/捕获中断, TIMERx(x=0~4,7,14~16,22,23,40~44)
TIMER_INT_CH1	通道1比较/捕获中断, TIMERx(x=0~4,7,14,22,23,40~44)
TIMER_INT_CH2	通道2比较/捕获中断, TIMERx(x=0~4,7,22,23)
TIMER_INT_CH3	通道3比较/捕获中断, TIMERx(x=0~4,7,22,23)
TIMER_INT_CMT	换相更新中断, TIMERx(x=0,7,14~16,40~44)
TIMER_INT_TRG	触发中断, TIMERx(x=0~4,7,14,22,23,40~44)
TIMER_INT_BRK	中止中断, TIMERx(x=0,7,14~16,40~44)
TIMER_INT_DECJ	正交编码器跳变中断, TIMERx(x=0~4,7,22,23)
TIMER_INT_DECDIS	正交编码器断线中断, TIMERx(x=0~4,7,22,23)
TIMER_INT_MCH0	多模式通道0比较/捕获中断, TIMERx(x=0,7,14~16,40~44)
TIMER_INT_MCH1	多模式通道1比较/捕获中断, TIMERx(x=0,7)
TIMER_INT_MCH2	多模式通道2比较/捕获中断, TIMERx(x=0,7)
TIMER_INT_MCH3	多模式通道3比较/捕获中断, TIMERx(x=0,7)
TIMER_INT_CH0C OMADD	通道0附加比较中断, TIMERx(x=0~4,7,14,22,23,40~44)
TIMER_INT_CH0C OMADD	通道1附加比较中断, TIMERx(x=0~4,7,14,22,23,40~44)
TIMER_INT_CH0C	通道2附加比较中断, TIMERx(x=0~4,7,22,23)

OMADD	
TIMER_INT_CH0C OMADD	通道3附加比较中断, TIMERx(x=0~4,7,22,23)
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 3-1230. 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=0~7,14~16,22,23,40~44,50,51)	参考具体参数
输入参数{in}	
interrupt	中断源
TIMER_INT_UP	更新中断, TIMERx(x=0~7,14~16,22,23,40~44,50,51)
TIMER_INT_CH0	通道0比较/捕获中断, TIMERx(x=0~4,7,14~16,22,23,40~44)
TIMER_INT_CH1	通道1比较/捕获中断, TIMERx(x=0~4,7,14,22,23,40~44)
TIMER_INT_CH2	通道2比较/捕获中断, TIMERx(x=0~4,7,22,23)
TIMER_INT_CH3	通道3比较/捕获中断, TIMERx(x=0~4,7,22,23)
TIMER_INT_CMT	换相更新中断, TIMERx(x=0,7,14~16,40~44)
TIMER_INT_TRG	触发中断, TIMERx(x=0~4,7,14,22,23,40~44)
TIMER_INT_BRK	中止中断, TIMERx(x=0,7,14~16,40~44)
TIMER_INT_DECJ	正交编码器跳变中断, TIMERx(x=0~4,7,22,23)
TIMER_INT_DECDIS	正交编码器断线中断, TIMERx(x=0~4,7,22,23)
TIMER_INT_MCH0	多模式通道0比较/捕获中断, TIMERx(x=0,7,14~16,40~44)
TIMER_INT_MCH1	多模式通道1比较/捕获中断, TIMERx(x=0,7)
TIMER_INT_MCH2	多模式通道2比较/捕获中断, TIMERx(x=0,7)

<code>TIMER_INT_MCH3</code>	多模式通道3比较/捕获中断, <code>TIMERx(x=0,7)</code>
<code>TIMER_INT_CH0C</code> <code>OMADD</code>	通道0附加比较中断, <code>TIMERx(x=0~4,7,14,22,23,40~44)</code>
<code>TIMER_INT_CH0C</code> <code>OMADD</code>	通道1附加比较中断, <code>TIMERx(x=0~4,7,14,22,23,40~44)</code>
<code>TIMER_INT_CH0C</code> <code>OMADD</code>	通道2附加比较中断, <code>TIMERx(x=0~4,7,22,23)</code>
<code>TIMER_INT_CH0C</code> <code>OMADD</code>	通道3附加比较中断, <code>TIMERx(x=0~4,7,22,23)</code>
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 3-1231. 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TIMERx中断标志
先决条件	-
被调用函数	-
输入参数{in}	
timer_periph	TIMER外设
<code>TIMERx(x=0~7,14~16,22,23,40~44,50,51)</code>	参考具体参数
输入参数{in}	
int_flag	中断源
<code>TIMER_INT_FLAG_UP</code>	更新中断标志, <code>TIMERx(x=0~7,14~16,22,23,40~44,50,51)</code>
<code>TIMER_INT_FLAG_CH0</code>	通道0比较/捕获中断标志, <code>TIMERx(x=0~4,7,14~16,22,23,40~44)</code>
<code>TIMER_INT_FLAG_CH1</code>	通道1比较/捕获中断标志, <code>TIMERx(x=0~4,7,14,22,23,40~44)</code>
<code>TIMER_INT_FLAG_CH2</code>	通道2比较/捕获中断标志, <code>TIMERx(x=0~4,7,22,23)</code>

<i>TIMER_INT_FLAG_CH3</i>	通道3比较/捕获中断标志，TIMERx(x=0~4,7,22,23)
<i>TIMER_INT_FLAG_CMT</i>	换相更新中断标志，TIMERx(x=0,7,14~16,40~44)
<i>TIMER_INT_FLAG_TRG</i>	触发中断标志，TIMERx(x=0~4,7,14,22,23,40~44)
<i>TIMER_INT_FLAG_BRK0</i>	BREAK0中断标志，TIMERx(x=0,7,14~16,40~44)
<i>TIMER_INT_FLAG_BRK1</i>	BREAK1中断标志，TIMERx(x=0,7)
<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=0~4,7,22,23)
<i>TIMER_INT_FLAG_DECDIS</i>	正交编码器断线中断标志，TIMERx(x=0~4,7,22,23)
<i>TIMER_INT_FLAG_MCH0</i>	多模式通道0比较/捕获中断标志，TIMERx(x=0,7,14~16,40~44)
<i>TIMER_INT_FLAG_MCH1</i>	多模式通道1比较/捕获中断标志，TIMERx(x=0,7)
<i>TIMER_INT_FLAG_MCH2</i>	多模式通道2比较/捕获中断标志，TIMERx(x=0,7)
<i>TIMER_INT_FLAG_MCH3</i>	多模式通道3比较/捕获中断标志，TIMERx(x=0,7)
<i>TIMER_INT_FLAG_CH0COMADD</i>	通道0附加比较中断标志，TIMERx(x=0~4,7,14,22,23)
<i>TIMER_INT_FLAG_CH1COMADD</i>	通道1附加比较中断标志，TIMERx(x=0~4,7,14,22,23)
<i>TIMER_INT_FLAG_CH2COMADD</i>	通道2附加比较中断标志，TIMERx(x=0~4,7,22,23)
<i>TIMER_INT_FLAG_CH3COMADD</i>	通道3附加比较中断标志，TIMERx(x=0~4,7,22,23)
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 3-1232. 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=0~7,14~16,22,23,40~44,50,51)	参考具体参数
输入参数{in}	
int_flag	中断源
TIMER_INT_FLAG_UP	更新中断标志, TIMERx(x=0~7,14~16,22,23,40~44,50,51)
TIMER_INT_FLAG_CH0	通道0比较/捕获中断标志, TIMERx(x=0~4,7,14~16,22,23,40~44)
TIMER_INT_FLAG_CH1	通道1比较/捕获中断标志, TIMERx(x=0~4,7,14,22,23,40~44)
TIMER_INT_FLAG_CH2	通道2比较/捕获中断标志, TIMERx(x=0~4,7,22,23)
TIMER_INT_FLAG_CH3	通道3比较/捕获中断标志, TIMERx(x=0~4,7,22,23)
TIMER_INT_FLAG_CMT	换相更新中断标志, TIMERx(x=0,7,14~16,40~44)
TIMER_INT_FLAG_TRG	触发中断标志, TIMERx(x=0~4,7,14,22,23,40~44)
TIMER_INT_FLAG_BRK0	BREAK0中断标志, TIMERx(x=0,7,14~16,40~44)
TIMER_INT_FLAG_BRK1	BREAK1中断标志, TIMERx(x=0,7)
TIMER_INT_FLAG_SYSB	系统源中止中断标志, TIMERx(x=0,7)
TIMER_INT_FLAG_DECJ	正交编码器跳变中断标志, TIMERx(x=0~4,7,22,23)
TIMER_INT_FLAG_DECDIS	正交编码器断线中断标志, TIMERx(x=0~4,7,22,23)
TIMER_INT_FLAG_MCH0	多模式通道0比较/捕获中断标志, TIMERx(x=0,7,14~16,40~44)

<i>TIMER_INT_FLAG_</i> <i>MCH1</i>	多模式通道1比较/捕获中断标志，TIMERx(x=0,7)
<i>TIMER_INT_FLAG_</i> <i>MCH2</i>	多模式通道2比较/捕获中断标志，TIMERx(x=0,7)
<i>TIMER_INT_FLAG_</i> <i>MCH3</i>	多模式通道3比较/捕获中断标志，TIMERx(x=0,7)
<i>TIMER_INT_FLAG_</i> <i>CH0COMADD</i>	通道0附加比较中断标志，TIMERx(x=0~4,7,14,22,23)
<i>TIMER_INT_FLAG_</i> <i>CH1COMADD</i>	通道1附加比较中断标志，TIMERx(x=0~4,7,14,22,23)
<i>TIMER_INT_FLAG_</i> <i>CH2COMADD</i>	通道2附加比较中断标志，TIMERx(x=0~4,7,22,23)
<i>TIMER_INT_FLAG_</i> <i>CH3COMADD</i>	通道3附加比较中断标志，TIMERx(x=0~4,7,22,23)
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* clear TIMER0 update interrupt flag */
```

```
timer_interrupt_flag_clear(TIMER0, TIMER_INT_FLAG_UP);
```

3.33. TMU

三角函数加速器（TMU）是一个完全可配置的单元，可执行常见的三角运算和算术运算操作。TMU可以减轻CPU的负担。章节[3.33.1](#)描述了TMU的寄存器列表，章节[3.33.2](#)对TMU库函数进行说明。

3.33.1. 外设寄存器说明

TMU寄存器列表如下表所示：

表 3-1233. TMU 寄存器

寄存器名称	寄存器描述
TMU_CS	TMU控制和状态寄存器
TMU_IDATA	TMU输入数据寄存器
TMU_ODATA	TMU输出数据寄存器

3.33.2. 外设库函数说明

TMU库函数列表如下表所示：

表 3-1234. TMU 库函数

函数名称	功能描述
tmu_deinit	复位 TMU 的所有寄存器
tmu_struct_para_init	使用默认值初始化 tmu_parameter_struct 结构体
tmu_init	初始化 TMU
tmu_read_interrupt_enable	使能 TMU 读中断
tmu_read_interrupt_disable	禁能 TMU 读中断
tmu_dma_read_enable	使能 TMU 读请求（DMA）
tmu_dma_read_disable	禁能 TMU 读请求（DMA）
tmu_dma_write_enable	使能 TMU 写中断
tmu_dma_write_disable	禁能 TMU 写中断
tmu_one_q31_write	写一个 q1.31 格式数据
tmu_two_q31_write	写两个 q1.31 格式数据
tmu_two_q15_write	写两个 q1.15 格式数据
tmu_one_q31_read	读一个 q1.31 格式数据
tmu_two_q31_read	读两个 q1.31 格式数据
tmu_two_q15_read	读两个 q1.15 格式数据

结构体 tmu_parameter_struct

表 3-1235. 结构体 tmu_parameter_struct

成员名称	功能描述
mode	TMU运行模式

成员名称	功能描述
	(TMU_MODE_COS,TMU_MODE_SIN,TMU_MODE_ATAN2,TMU_MODE_MODULUS,TMU_MODE_ATAN,TMU_MODE_COSH,TMU_MODE_SINH,TMU_MODE_ATANH,TMU_MODE_LN,TMU_MODE_SQRT)
iterations_number	迭代次数(TMU_ITERATION_STEPS_x(x=4,8,12,...24))
scale	缩放因子(TMU_SCALING_FACTOR_x(x=1,2,4,8,16,32,64,128))
dma_read	读TMU_ODATA寄存器DMA请求(TMU_READ_DMA_DISABLE, TMU_READ_DMA_ENABLE)
dma_write	写TMU_IDATA寄存器DMA请求(TMU_WRITE_DMA_DISABLE, TMU_WRITE_DMA_ENABLE)
read_times	读TMU_ODATA寄存器的次数(TMU_READ_TIMES_1, TMU_READ_TIMES_2)
write_times	写TMU_IDATA寄存器的次数(TMU_WRITE_TIMES_1, TMU_WRITE_TIMES_2)
output_width	输出数据宽度(TMU_OUTPUT_WIDTH_32, TMU_OUTPUT_WIDTH_16)
input_width	输入数据宽度(TMU_INPUT_WIDTH_32, TMU_INPUT_WIDTH_16)

函数 tmu_deinit

函数tmu_deinit的描述如下表:

表 3-1236. 函数 tmu_deinit

函数名称	tmu_deinit
函数原型	void tmu_deinit(void);
功能描述	复位TMU的所有寄存器
先决条件	-
被调用函数	rcu_periph_reset_enable / rcu_periph_reset_disable
输入参数{in}	
-	-
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* reset TMU */
tmu_deinit();
```

函数 tmu_struct_para_init

函数tmu_struct_para_init的描述如下表:

表 3-1237. 函数 tmu_struct_para_init

函数名称	tmu_struct_para_init
函数原型	void tmu_struct_para_init(tmu_parameter_struct* init_struct);

功能描述	使用默认值初始化tmu_parameter_struct结构体
先决条件	-
被调用函数	-
输入参数{in}	
init_struct	指向tmu_parameter_struct结构体指针，结构体成员参考 结构体 tmu_parameter_struct 。
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* initialize TMU init parameter struct with a default value */
```

```
tmu_parameter_struct tmu_initpara;
```

```
tmu_struct_para_init(&tmu_initpara);
```

函数 tmu_init

函数tmu_init的描述如下表：

表 3-1238. 函数 tmu_init

函数名称	tmu_init
函数原型	void tmu_init(tmu_parameter_struct* init_struct);
功能描述	初始化TMU
先决条件	-
被调用函数	-
输入参数{in}	
init_struct	指向 tmu_parameter_struct结构体指针，结构体成员参考 结构体 tmu_parameter_struct 。
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* initialize TMU */
```

```
tmu_parameter_struct tmu_init_struct;
```

```
tmu_init_struct.mode = TMU_MODE_COS;
```

```
tmu_init_struct.iterations_number = TMU_ITERATION_STEPS_24;
```

```
tmu_init_struct.scale = TMU_SCALING_FACTOR_1;
```



```

tmu_init_struct.dma_read = TMU_READ_DMA_ENABLE;

tmu_init_struct.dma_write = TMU_WRITE_DMA_ENABLE;

tmu_init_struct.read_times = TMU_READ_TIMES_2;

tmu_init_struct.write_times = TMU_WRITE_TIMES_2;

tmu_init_struct.output_width = TMU_OUTPUT_WIDTH_32;

tmu_init_struct.input_width = TMU_INPUT_WIDTH_32;

tmu_init(&tmu_init_struct);

```

函数 tmu_read_interrupt_enable

函数tmu_read_interrupt_enable的描述如下表：

表 3-1239. 函数 tmu_read_interrupt_enable

函数名称	tmu_read_interrupt_enable
函数原型	void tmu_read_interrupt_enable(void);
功能描述	使能TMU读中断
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 TMU read interrupt */

tmu_read_interrupt_enable();

```

函数 tmu_read_interrupt_disable

函数tmu_read_interrupt_disable的描述如下表：

表 3-1240. 函数 tmu_read_interrupt_disable

函数名称	tmu_read_interrupt_disable
函数原型	void tmu_read_interrupt_disable(void);
功能描述	禁能TMU读中断
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 TMU read interrupt */
tmu_read_interrupt_disable();
```

函数 tmu_dma_read_enable

函数tmu_dma_read_enable的描述如下表：

表 3-1241. 函数 tmu_dma_read_enable

函数名称	tmu_dma_read_enable
函数原型	void tmu_dma_read_enable(void);
功能描述	使能TMU读请求（DMA）
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 TMU DMA read request */
tmu_dma_read_enable();
```

函数 tmu_dma_read_disable

函数tmu_dma_read_disable的描述如下表：

表 3-1242. 函数 tmu_dma_read_disable

函数名称	tmu_dma_read_disable
函数原型	void tmu_dma_read_disable(void);
功能描述	禁能TMU读请求（DMA）
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 TMU DMA read request */
```

```
tmu_dma_read_disable();
```

函数 tmu_dma_write_enable

函数tmu_dma_write_enable的描述如下表：

表 3-1243. 函数 tmu_dma_write_enable

函数名称	tmu_dma_write_enable
函数原型	void tmu_dma_write_enable(void);
功能描述	使能TMU写请求（DMA）
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 TMU DMA write request */
```

```
tmu_dma_write_enable();
```

函数 tmu_dma_write_disable

函数tmu_dma_write_disable的描述如下表：

表 3-1244. 函数 tmu_dma_write_disable

函数名称	tmu_dma_write_disable
函数原型	void tmu_dma_write_disable(void);
功能描述	禁能TMU写请求（DMA）
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 TMU DMA write request */
```

```
tmu_dma_write_disable();
```

函数 tmu_one_q31_write

函数tmu_one_q31_write的描述如下表：

表 3-1245. 函数 tmu_one_q31_write

函数名称	tmu_one_q31_write
函数原型	void tmu_one_q31_write(uint32_t data);
功能描述	写一个q1.31格式数据
先决条件	-
被调用函数	-
输入参数{in}	
data	q1.31格式的输入数据(0x00000000~0xFFFFFFFF)
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* write onedata in q1.31 format */
```

```
int32_t in = -2000;
```

```
tmu_one_q31_write((uint32_t)in);
```

函数 tmu_two_q31_write

函数tmu_two_q31_write的描述如下表：

表 3-1246. 函数 tmu_two_q31_write

函数名称	tmu_two_q31_write
函数原型	void tmu_two_q31_write(uint32_t data1, uint32_t data2);
功能描述	写两个q1.31格式数据
先决条件	-
被调用函数	-
输入参数{in}	
data1	q1.31格式的第一个输入数据(0x00000000~0xFFFFFFFF)
输入参数{in}	
data2	q1.31格式的第二个输入数据(0x00000000~0xFFFFFFFF)

返回值	
-	-

例如：

```
/* write two data in q1.31 format */
```

```
int32_t in1 = -2000;
```

```
int32_t in2 = 3000;
```

```
tmu_two_q31_write((uint32_t)in1, (uint32_t)in2);
```

函数 tmu_two_q15_write

函数tmu_two_q15_write的描述如下表：

表 3-1247. 函数 tmu_two_q15_write

函数名称	tmu_two_q15_write
函数原型	void tmu_two_q15_write(uint16_t data1, uint16_t data2);
功能描述	写两个q1.15格式数据
先决条件	-
被调用函数	-
输入参数{in}	
data1	q1.15格式的第二个输入数据(0x0000~0xFFFF)
输入参数{in}	
data2	q1.15格式的第二个输入数据(0x0000~0xFFFF)
返回值	
-	-

例如：

```
/* write two data in q1.15 format */
```

```
int16_t in1 = -2000;
```

```
int16_t in2 = 3000;
```

```
tmu_two_q15_write((uint16_t)in1, (uint16_t)in2);
```

函数 tmu_one_q31_read

函数tmu_one_q31_read的描述如下表：

表 3-1248. 函数 tmu_one_q31_read

函数名称	tmu_one_q31_read
函数原型	void tmu_one_q31_read(uint32_t* p);
功能描述	读一个q1.31格式数据
先决条件	-

被调用函数	-
输入参数{in}	
p	指向输出数据的指针（q1.31格式）
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* read one data in q1.31 format */
uint32_t out = 0;
tmu_one_q31_read (&out);
```

函数 tmu_two_q31_read

函数tmu_two_q31_read的描述如下表：

表 3-1249. 函数 tmu_two_q31_read

函数名称	tmu_two_q31_read
函数原型	void tmu_two_q31_read(uint32_t* p1, uint32_t* p2);
功能描述	读两个q1.31格式数据
先决条件	-
被调用函数	-
输入参数{in}	
p1	指向第一个输出数据的指针（q1.31格式）
输入参数{in}	
p2	指向第二个输出数据的指针（q1.31格式）
返回值	
-	-

例如：

```
/* read two data in q1.31 format */
uint32_t out1 = 0;
uint32_t out2 = 0;
tmu_two_q31_read(&out1, &out2);
```

函数 tmu_two_q15_read

函数tmu_two_q15_read的描述如下表：

表 3-1250. 函数 tmu_two_q15_read

函数名称	tmu_two_q15_read
------	------------------

函数原型	void tmu_two_q15_read(uint16_t* p1, uint16_t* p2);
功能描述	读两个q1.15格式数据
先决条件	-
被调用函数	-
输入参数{in}	
p1	指向第一个输出数据的指针（q1.15格式）
输入参数{in}	
p2	指向第二个输出数据的指针（q1.15格式）
返回值	
-	-

例如：

```
/* read two data in q1.15 format */
uint16_t out1 = 0;
uint16_t out2 = 0;
tmu_two_q15_read(&out1, &out2);
```

3.34. 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 [3.34.1](#) 描述了 TRIGSEL 的寄存器列表，章节 [3.34.2](#) 对 TRIGSEL 库函数进行说明。

3.34.1. 外设寄存器说明

TRIGSEL 寄存器列表如下表所示：

表 3-1251. TRIGSEL 寄存器

寄存器名称	寄存器描述
TRIGSEL_EXTOUT0	EXTOUT0触发选择寄存器
TRIGSEL_EXTOUT1	EXTOUT1触发选择寄存器
TRIGSEL_EXTOUT2	EXTOUT2触发选择寄存器
TRIGSEL_EXTOUT3	EXTOUT3触发选择寄存器
TRIGSEL_ADC0	ADC0触发选择寄存器
TRIGSEL_ADC1	ADC1触发选择寄存器
TRIGSEL_ADC2	ADC2触发选择寄存器
TRIGSEL_DACOUT0	DAC_OUT0触发选择寄存器
TRIGSEL_DACOUT1	DAC_OUT1触发选择寄存器
TRIGSEL_TIMER0BRKIN	TIMER0_BRKIN触发选择寄存器
TRIGSEL_TIMER7BRKIN	TIMER7_BRKIN触发选择寄存器
TRIGSEL_TIMER14BRKIN	TIMER14_BRKIN触发选择寄存器

寄存器名称	寄存器描述
TRIGSEL_TIMER15BRKIN	TIMER15_BRKIN触发选择寄存器
TRIGSEL_TIMER16BRKIN	TIMER16_BRKIN触发选择寄存器
TRIGSEL_TIMER40BRKIN	TIMER40_BRKIN触发选择寄存器
TRIGSEL_TIMER41BRKIN	TIMER41_BRKIN触发选择寄存器
TRIGSEL_TIMER42BRKIN	TIMER42_BRKIN触发选择寄存器
TRIGSEL_TIMER43BRKIN	TIMER43_BRKIN触发选择寄存器
TRIGSEL_TIMER44BRKIN	TIMER44_BRKIN触发选择寄存器
TRIGSEL_CAN0	CAN0触发选择寄存器
TRIGSEL_CAN1	CAN1触发选择寄存器
TRIGSEL_CAN2	CAN2触发选择寄存器
TRIGSEL_LPDS	LPDS触发选择寄存器
TRIGSEL_TIMER0ETI	TIMER0_ETI触发选择寄存器
TRIGSEL_TIMER1ETI	TIMER1_ETI触发选择寄存器
TRIGSEL_TIMER2ETI	TIMER2_ETI触发选择寄存器
TRIGSEL_TIMER3ETI	TIMER3_ETI触发选择寄存器
TRIGSEL_TIMER4ETI	TIMER4_ETI触发选择寄存器
TRIGSEL_TIMER7ETI	TIMER7_ETI触发选择寄存器
TRIGSEL_TIMER22ETI	TIMER22_ETI触发选择寄存器
TRIGSEL_TIMER23ETI	TIMER23_ETI触发选择寄存器
TRIGSEL_EDOUT	EDOUT触发选择寄存器
TRIGSEL_HPDF	HPDF触发选择寄存器
TRIGSEL_TIMER0ITI14	TIMER0_ITI14触发选择寄存器
TRIGSEL_TIMER1ITI14	TIMER1_ITI14触发选择寄存器
TRIGSEL_TIMER2ITI14	TIMER2_ITI14触发选择寄存器
TRIGSEL_TIMER3ITI14	TIMER3_ITI14触发选择寄存器
TRIGSEL_TIMER4ITI14	TIMER4_ITI14触发选择寄存器
TRIGSEL_TIMER7ITI14	TIMER7_ITI14触发选择寄存器
TRIGSEL_TIMER14ITI14	TIMER14_ITI14触发选择寄存器
TRIGSEL_TIMER22ITI14	TIMER22_ITI14触发选择寄存器
TRIGSEL_TIMER23ITI14	TIMER23_ITI14触发选择寄存器
TRIGSEL_TIMER40ITI14	TIMER40_ITI14触发选择寄存器
TRIGSEL_TIMER41ITI14	TIMER41_ITI14触发选择寄存器
TRIGSEL_TIMER42ITI14	TIMER42_ITI14触发选择寄存器
TRIGSEL_TIMER43ITI14	TIMER43_ITI14触发选择寄存器
TRIGSEL_TIMER44ITI14	TIMER44_ITI14触发选择寄存器

3.34.2. 外设库函数说明

TRIGSEL库函数列表如下表所示：

表 3-1252. 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 3-1253. 枚举类型 trigsel_source_enum

成员名称	功能描述
TRIGSEL_INPUT_0	触发输入源为0
TRIGSEL_INPUT_1	触发输入源为1
TRIGSEL_INPUT_TRIGSEL_IN0	触发输入源为TRIGSEL_IN0引脚
TRIGSEL_INPUT_TRIGSEL_IN1	触发输入源为TRIGSEL_IN1引脚
TRIGSEL_INPUT_TRIGSEL_IN2	触发输入源为TRIGSEL_IN2引脚
TRIGSEL_INPUT_TRIGSEL_IN3	触发输入源为TRIGSEL_IN3引脚
TRIGSEL_INPUT_TRIGSEL_IN4	触发输入源为TRIGSEL_IN4引脚
TRIGSEL_INPUT_TRIGSEL_IN5	触发输入源为TRIGSEL_IN5引脚
TRIGSEL_INPUT_TRIGSEL_IN6	触发输入源为TRIGSEL_IN6引脚
TRIGSEL_INPUT_TRIGSEL_IN7	触发输入源为TRIGSEL_IN7引脚
TRIGSEL_INPUT_TRIGSEL_IN8	触发输入源为TRIGSEL_IN8引脚
TRIGSEL_INPUT_TRIGSEL_IN9	触发输入源为TRIGSEL_IN9引脚
TRIGSEL_INPUT_TRIGSEL_IN10	触发输入源为TRIGSEL_IN10引脚
TRIGSEL_INPUT_TRIGSEL_IN11	触发输入源为TRIGSEL_IN11引脚
TRIGSEL_INPUT_TRIGSEL_IN12	触发输入源为TRIGSEL_IN12引脚
TRIGSEL_INPUT_TRIGSEL_IN13	触发输入源为TRIGSEL_IN13引脚
TRIGSEL_INPUT_LXTAL_TRG	触发输入源为LXTAL_TRG
TRIGSEL_INPUT_TIMER0_TRGO0	触发输入源为TIMER0 TRGO0
TRIGSEL_INPUT_TIMER0_TRGO1	触发输入源为TIMER0 TRGO1
TRIGSEL_INPUT_TIMER0_CH0	触发输入源为TIMER0 CH0
TRIGSEL_INPUT_TIMER0_CH1	触发输入源为TIMER0 CH1
TRIGSEL_INPUT_TIMER0_CH2	触发输入源为TIMER0 CH2
TRIGSEL_INPUT_TIMER0_CH3	触发输入源为TIMER0 CH3
TRIGSEL_INPUT_TIMER0_MCH0	触发输入源为TIMER0 MCH0
TRIGSEL_INPUT_TIMER0_MCH1	触发输入源为TIMER0 MCH1
TRIGSEL_INPUT_TIMER0_MCH2	触发输入源为TIMER0 MCH2
TRIGSEL_INPUT_TIMER0_MCH3	触发输入源为TIMER0 MCH3
TRIGSEL_INPUT_TIMER0_BRKIN0	触发输入源为TIMER0 BRKIN0
TRIGSEL_INPUT_TIMER0_BRKIN1	触发输入源为TIMER0 BRKIN1
TRIGSEL_INPUT_TIMER0_BRKIN2	触发输入源为TIMER0 BRKIN2

成员名称	功能描述
TRIGSEL_INPUT_TIMER0_ETI	触发输入源为TIMER0 ETI
TRIGSEL_INPUT_TIMER1_TRGO0	触发输入源为TIMER1 TRGO0
TRIGSEL_INPUT_TIMER1_CH0	触发输入源为TIMER1 CH0
TRIGSEL_INPUT_TIMER1_CH1	触发输入源为TIMER1 CH1
TRIGSEL_INPUT_TIMER1_CH2	触发输入源为TIMER1 CH2
TRIGSEL_INPUT_TIMER1_CH3	触发输入源为TIMER1 CH3
TRIGSEL_INPUT_TIMER1_ETI	触发输入源为TIMER1 ETI
TRIGSEL_INPUT_TIMER2_TRGO0	触发输入源为TIMER2 TRGO0
TRIGSEL_INPUT_TIMER2_CH0	触发输入源为TIMER2 CH0
TRIGSEL_INPUT_TIMER2_CH1	触发输入源为TIMER2 CH1
TRIGSEL_INPUT_TIMER2_CH2	触发输入源为TIMER2 CH2
TRIGSEL_INPUT_TIMER2_CH3	触发输入源为TIMER2 CH3
TRIGSEL_INPUT_TIMER2_ETI	触发输入源为TIMER2 ETI
TRIGSEL_INPUT_TIMER3_TRGO0	触发输入源为TIMER3 TRGO0
TRIGSEL_INPUT_TIMER3_CH0	触发输入源为TIMER3 CH0
TRIGSEL_INPUT_TIMER3_CH1	触发输入源为TIMER3 CH1
TRIGSEL_INPUT_TIMER3_CH2	触发输入源为TIMER3 CH2
TRIGSEL_INPUT_TIMER3_CH3	触发输入源为TIMER3 CH3
TRIGSEL_INPUT_TIMER3_ETI	触发输入源为TIMER3 ETI
TRIGSEL_INPUT_TIMER4_TRGO0	触发输入源为TIMER4 TRGO0
TRIGSEL_INPUT_TIMER4_CH0	触发输入源为TIMER4 CH0
TRIGSEL_INPUT_TIMER4_CH1	触发输入源为TIMER4 CH1
TRIGSEL_INPUT_TIMER4_CH2	触发输入源为TIMER4 CH2
TRIGSEL_INPUT_TIMER4_CH3	触发输入源为TIMER4 CH3
TRIGSEL_INPUT_TIMER4_ETI	触发输入源为TIMER4 ETI
TRIGSEL_INPUT_TIMER5_TRGO0	触发输入源为TIMER5 TRGO0
TRIGSEL_INPUT_TIMER6_TRGO0	触发输入源为TIMER6 TRGO0
TRIGSEL_INPUT_TIMER7_TRGO0	触发输入源为TIMER7 TRGO0
TRIGSEL_INPUT_TIMER7_TRGO1	触发输入源为TIMER7 TRGO1
TRIGSEL_INPUT_TIMER7_CH0	触发输入源为TIMER7 CH0
TRIGSEL_INPUT_TIMER7_CH1	触发输入源为TIMER7 CH1
TRIGSEL_INPUT_TIMER7_CH2	触发输入源为TIMER7 CH2
TRIGSEL_INPUT_TIMER7_CH3	触发输入源为TIMER7 CH3
TRIGSEL_INPUT_TIMER7_MCH0	触发输入源为TIMER7 MCH0
TRIGSEL_INPUT_TIMER7_MCH1	触发输入源为TIMER7 MCH1
TRIGSEL_INPUT_TIMER7_MCH2	触发输入源为TIMER7 MCH2
TRIGSEL_INPUT_TIMER7_MCH3	触发输入源为TIMER7 MCH3
TRIGSEL_INPUT_TIMER7_BRKIN0	触发输入源为TIMER7 BRKIN0
TRIGSEL_INPUT_TIMER7_BRKIN1	触发输入源为TIMER7 BRKIN1
TRIGSEL_INPUT_TIMER7_BRKIN2	触发输入源为TIMER7 BRKIN2
TRIGSEL_INPUT_TIMER7_ETI	触发输入源为TIMER7 ETI

成员名称	功能描述
TRIGSEL_INPUT_TIMER14_TRGO0	触发输入源为TIMER14 TRGO0
TRIGSEL_INPUT_TIMER14_CH0	触发输入源为TIMER14 CH0
TRIGSEL_INPUT_TIMER14_CH1	触发输入源为TIMER14 CH1
TRIGSEL_INPUT_TIMER14_MCH0	触发输入源为TIMER14 MCH0
TRIGSEL_INPUT_TIMER14_BRKIN	触发输入源为TIMER14 BRKIN
TRIGSEL_INPUT_TIMER15_CH0	触发输入源为TIMER15 CH0
TRIGSEL_INPUT_TIMER15_MCH0	触发输入源为TIMER15 MCH0
TRIGSEL_INPUT_TIMER15_BRKIN	触发输入源为TIMER15 BRKIN
TRIGSEL_INPUT_TIMER16_CH0	触发输入源为TIMER16 CH0
TRIGSEL_INPUT_TIMER16_MCH0	触发输入源为TIMER16 MCH0
TRIGSEL_INPUT_TIMER16_BRKIN	触发输入源为TIMER16 BRKIN
TRIGSEL_INPUT_TIMER22_TRGO0	触发输入源为TIMER22 TRGO0
TRIGSEL_INPUT_TIMER22_CH0	触发输入源为TIMER22 CH0
TRIGSEL_INPUT_TIMER22_CH1	触发输入源为TIMER22 CH1
TRIGSEL_INPUT_TIMER22_CH2	触发输入源为TIMER22 CH2
TRIGSEL_INPUT_TIMER22_CH3	触发输入源为TIMER22 CH3
TRIGSEL_INPUT_TIMER22_ETI	触发输入源为TIMER22 ETI
TRIGSEL_INPUT_TIMER23_TRGO0	触发输入源为TIMER23 TRGO0
TRIGSEL_INPUT_TIMER23_CH0	触发输入源为TIMER23 CH0
TRIGSEL_INPUT_TIMER23_CH1	触发输入源为TIMER23 CH1
TRIGSEL_INPUT_TIMER23_CH2	触发输入源为TIMER23 CH2
TRIGSEL_INPUT_TIMER23_CH3	触发输入源为TIMER22 CH3
TRIGSEL_INPUT_TIMER23_ETI	触发输入源为TIMER23 ETI
TRIGSEL_INPUT_TIMER40_TRGO0	触发输入源为TIMER40 TRGO0
TRIGSEL_INPUT_TIMER40_CH0	触发输入源为TIMER40 CH0
TRIGSEL_INPUT_TIMER40_CH1	触发输入源为TIMER40 CH1
TRIGSEL_INPUT_TIMER40_MCH0	触发输入源为TIMER40 MCH0
TRIGSEL_INPUT_TIMER40_BRKIN	触发输入源为TIMER40 BRKIN
TRIGSEL_INPUT_TIMER41_TRGO0	触发输入源为TIMER41 TRGO0
TRIGSEL_INPUT_TIMER41_CH0	触发输入源为TIMER41 CH0
TRIGSEL_INPUT_TIMER41_CH1	触发输入源为TIMER41 CH1
TRIGSEL_INPUT_TIMER41_MCH0	触发输入源为TIMER41 MCH0
TRIGSEL_INPUT_TIMER41_BRKIN	触发输入源为TIMER41 BRKIN
TRIGSEL_INPUT_TIMER42_TRGO0	触发输入源为TIMER42 TRGO0
TRIGSEL_INPUT_TIMER42_CH0	触发输入源为TIMER42 CH0
TRIGSEL_INPUT_TIMER42_CH1	触发输入源为TIMER42 CH1
TRIGSEL_INPUT_TIMER42_MCH0	触发输入源为TIMER42 MCH0
TRIGSEL_INPUT_TIMER42_BRKIN	触发输入源为TIMER42 BRKIN
TRIGSEL_INPUT_TIMER43_TRGO0	触发输入源为TIMER43 TRGO0
TRIGSEL_INPUT_TIMER43_CH0	触发输入源为TIMER43 CH0
TRIGSEL_INPUT_TIMER43_CH1	触发输入源为TIMER43 CH1

成员名称	功能描述
TRIGSEL_INPUT_TIMER43_MCH0	触发输入源为TIMER43 MCH0
TRIGSEL_INPUT_TIMER43_BRKIN	触发输入源为TIMER43 BRKIN
TRIGSEL_INPUT_TIMER44_TRGO0	触发输入源为TIMER44 TRGO0
TRIGSEL_INPUT_TIMER44_CH0	触发输入源为TIMER44 CH0
TRIGSEL_INPUT_TIMER44_CH1	触发输入源为TIMER44 CH1
TRIGSEL_INPUT_TIMER44_MCH0	触发输入源为TIMER44 MCH0
TRIGSEL_INPUT_TIMER44_BRKIN	触发输入源为TIMER44 BRKIN
TRIGSEL_INPUT_TIMER50_TRGO0	触发输入源为TIMER50 TRGO0
TRIGSEL_INPUT_TIMER51_TRGO0	触发输入源为TIMER51 TRGO0
TRIGSEL_INPUT_RTC_ALARM	触发输入源为RTC alarm
TRIGSEL_INPUT_RTC_TPTS	触发输入源为RTC TPTS
TRIGSEL_INPUT_ADC0_WD0_OUT	触发输入源为ADC0 watchdog0 output
TRIGSEL_INPUT_ADC0_WD1_OUT	触发输入源为ADC0 watchdog1 output
TRIGSEL_INPUT_ADC0_WD2_OUT	触发输入源为ADC0 watchdog2 output
TRIGSEL_INPUT_ADC1_WD0_OUT	触发输入源为ADC1 watchdog0 output
TRIGSEL_INPUT_ADC1_WD1_OUT	触发输入源为ADC1 watchdog1 output
TRIGSEL_INPUT_ADC1_WD2_OUT	触发输入源为ADC1 watchdog2 output
TRIGSEL_INPUT_ADC2_WD0_OUT	触发输入源为ADC2 watchdog0 output
TRIGSEL_INPUT_ADC2_WD1_OUT	触发输入源为ADC2 watchdog1 output
TRIGSEL_INPUT_ADC2_WD2_OUT	触发输入源为ADC2 watchdog2 output
TRIGSEL_INPUT_CMP0_OUT	触发输入源为CMP0_OUT
TRIGSEL_INPUT_CMP1_OUT	触发输入源为CMP1_OUT
TRIGSEL_INPUT_SAI0_AFS_OUT	触发输入源为SAI0_AFS_OUT
TRIGSEL_INPUT_SAI0_BFS_OUT	触发输入源为SAI0_BFS_OUT
TRIGSEL_INPUT_SAI2_AFS_OUT	触发输入源为SAI2_AFS_OUT
TRIGSEL_INPUT_SAI2_BFS_OUT	触发输入源为SAI2_BFS_OUT

枚举类型 `trigsel_periph_enum`

表 3-1254. 枚举类型 `trigsel_periph_enum`

成员名称	功能描述
TRIGSEL_OUTPUT_TRIGSEL_OUT0	输出到目标外设TRIGSEL_OUT0引脚
TRIGSEL_OUTPUT_TRIGSEL_OUT1	输出到目标外设TRIGSEL_OUT1引脚
TRIGSEL_OUTPUT_TRIGSEL_OUT2	输出到目标外设TRIGSEL_OUT2引脚
TRIGSEL_OUTPUT_TRIGSEL_OUT3	输出到目标外设TRIGSEL_OUT3引脚
TRIGSEL_OUTPUT_TRIGSEL_OUT4	输出到目标外设TRIGSEL_OUT4引脚
TRIGSEL_OUTPUT_TRIGSEL_OUT5	输出到目标外设TRIGSEL_OUT5引脚
TRIGSEL_OUTPUT_TRIGSEL_OUT6	输出到目标外设TRIGSEL_OUT6引脚
TRIGSEL_OUTPUT_TRIGSEL_OUT7	输出到目标外设TRIGSEL_OUT7引脚
TRIGSEL_OUTPUT_ADC0_REGTRG	输出到目标外设ADC0_REGTRG
TRIGSEL_OUTPUT_ADC0_INSTRG	输出到目标外设ADC0_INSTRG

成员名称	功能描述
TRIGSEL_OUTPUT_ADC1_REGTRG	输出到目标外设ADC1_REGTRG
TRIGSEL_OUTPUT_ADC1_INSTRG	输出到目标外设ADC1_INSTRG
TRIGSEL_OUTPUT_ADC2_REGTRG	输出到目标外设ADC2_REGTRG
TRIGSEL_OUTPUT_ADC2_INSTRG	输出到目标外设ADC2_INSTRG
TRIGSEL_OUTPUT_DAC_OUT0_EXTRG	输出到目标外设DAC_OUT0_EXTRG
TRIGSEL_OUTPUT_DAC_OUT1_EXTRG	输出到目标外设DAC_OUT1_EXTRG
TRIGSEL_OUTPUT_TIMER0_BRKIN0	输出到目标外设TIMER0_BRKIN0
TRIGSEL_OUTPUT_TIMER0_BRKIN1	输出到目标外设TIMER0_BRKIN1
TRIGSEL_OUTPUT_TIMER0_BRKIN2	输出到目标外设TIMER0_BRKIN2
TRIGSEL_OUTPUT_TIMER7_BRKIN0	输出到目标外设TIMER7_BRKIN0
TRIGSEL_OUTPUT_TIMER7_BRKIN1	输出到目标外设TIMER7_BRKIN1
TRIGSEL_OUTPUT_TIMER7_BRKIN2	输出到目标外设TIMER7_BRKIN2
TRIGSEL_OUTPUT_TIMER14_BRKIN0	输出到目标外设TIMER14_BRKIN0
TRIGSEL_OUTPUT_TIMER15_BRKIN0	输出到目标外设TIMER15_BRKIN0
TRIGSEL_OUTPUT_TIMER16_BRKIN0	输出到目标外设TIMER16_BRKIN0
TRIGSEL_OUTPUT_TIMER40_BRKIN0	输出到目标外设TIMER40_BRKIN0
TRIGSEL_OUTPUT_TIMER41_BRKIN0	输出到目标外设TIMER41_BRKIN0
TRIGSEL_OUTPUT_TIMER42_BRKIN0	输出到目标外设TIMER42_BRKIN0
TRIGSEL_OUTPUT_TIMER43_BRKIN0	输出到目标外设TIMER43_BRKIN0
TRIGSEL_OUTPUT_TIMER44_BRKIN0	输出到目标外设TIMER44_BRKIN0
TRIGSEL_OUTPUT_CAN0_EX_TIME_TICK	输出到目标外设CAN0_EX_TIME_TICK
TRIGSEL_OUTPUT_CAN1_EX_TIME_TICK	输出到目标外设CAN1_EX_TIME_TICK
TRIGSEL_OUTPUT_CAN2_EX_TIME_TICK	输出到目标外设CAN2_EX_TIME_TICK
TRIGSEL_OUTPUT_LPPTS_TRG	输出到目标外设LPPTS_TRG
TRIGSEL_OUTPUT_TIMER0_ETI	输出到目标外设TIMER0_ETI
TRIGSEL_OUTPUT_TIMER1_ETI	输出到目标外设TIMER1_ETI
TRIGSEL_OUTPUT_TIMER2_ETI	输出到目标外设TIMER2_ETI
TRIGSEL_OUTPUT_TIMER3_ETI	输出到目标外设TIMER3_ETI
TRIGSEL_OUTPUT_TIMER4_ETI	输出到目标外设TIMER4_ETI
TRIGSEL_OUTPUT_TIMER7_ETI	输出到目标外设TIMER7_ETI
TRIGSEL_OUTPUT_TIMER22_ETI	输出到目标外设TIMER22_ETI
TRIGSEL_OUTPUT_TIMER23_ETI	输出到目标外设TIMER23_ETI
TRIGSEL_OUTPUT_EDOUT_TRG	输出到目标外设EDOUT_TRG
TRIGSEL_OUTPUT_HPDF_ITR	输出到目标外设HPDF_ITR
TRIGSEL_OUTPUT_TIMER0_ITI14	输出到目标外设TIMER0_ITI14
TRIGSEL_OUTPUT_TIMER1_ITI14	输出到目标外设TIMER1_ITI14
TRIGSEL_OUTPUT_TIMER2_ITI14	输出到目标外设TIMER2_ITI14
TRIGSEL_OUTPUT_TIMER3_ITI14	输出到目标外设TIMER3_ITI14
TRIGSEL_OUTPUT_TIMER4_ITI14	输出到目标外设TIMER4_ITI14
TRIGSEL_OUTPUT_TIMER7_ITI14	输出到目标外设TIMER7_ITI14
TRIGSEL_OUTPUT_TIMER14_ITI14	输出到目标外设TIMER14_ITI14

成员名称	功能描述
TRIGSEL_OUTPUT_TIMER22_ITI14	输出到目标外设TIMER22_ITI14
TRIGSEL_OUTPUT_TIMER23_ITI14	输出到目标外设TIMER23_ITI14
TRIGSEL_OUTPUT_TIMER40_ITI14	输出到目标外设TIMER40_ITI14
TRIGSEL_OUTPUT_TIMER41_ITI14	输出到目标外设TIMER41_ITI14
TRIGSEL_OUTPUT_TIMER42_ITI14	输出到目标外设TIMER42_ITI14
TRIGSEL_OUTPUT_TIMER43_ITI14	输出到目标外设TIMER43_ITI14
TRIGSEL_OUTPUT_TIMER44_ITI14	输出到目标外设TIMER44_ITI14

函数 trigsel_deinit

函数trigsel_init描述见下表：

表 3-1255. 函数 trigsel_init

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 3-1256. 函数 trigsel_init

函数名称	trigsel_init
函数原型	void trigsel_init(trigsel_periph_enum target_periph, trigsel_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 3-1254. 枚举类型 trigsel_periph_enum
输入参数{in}	

trigger_source	触发源，参考 表 3-1253. 枚举类型 trigselsource_enum
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* select TIMER0_CH2 to trigger ADC0 */
trigsels_init(TRIGSEL_OUTPUT_ADC0_REGTRG, TRIGSEL_INPUT_TIMER0_CH2);
```

函数 trigsels_trigger_source_get

函数trigsels_trigger_source_get描述见下表：

表 3-1257. 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 3-1254. 枚举类型 trigsels_periph_enum
输出参数{out}	
-	-
返回值	
trigger_source	触发源，参考 表 3-1253. 枚举类型 trigselsource_enum

例如：

```
/* get the trigger input signal for ADC0 */
trigsels_source_enum input_signal;
input_signal = trigsels_trigger_source_get(TRIGSEL_OUTPUT_ADC0_REGTRG);
```

函数 trigsels_register_lock_set

函数trigsels_register_lock_set描述见下表：

表 3-1258. 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 3-1254 . 枚举类型 trigsel_periph_enum
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* lock the trigger register for ADC0 */
trigsel_register_lock_set(TRIGSEL_OUTPUT_ADC0_REGTRG);
```

函数 trigsel_register_lock_get

函数trigsel_register_lock_get描述见下表:

表 3-1259. 函数 trigsel_trigger_lock_get

函数名称	trigsel_register_lock_get
函数原型	FlagStatus trigsel_register_lock_get(trigsel_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 3-1254 . 枚举类型 trigsel_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 = trigsel_register_lock_get(TRIGSEL_OUTPUT_ADC0_REGTRG);
```

3.35. 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 [3.35.1](#), TRNG固件库函数介绍在章节 [3.35.2](#)。

3.35.1. 外设寄存器说明

TRNG寄存器列表如下表所示:

表 3-1260. TRNG 寄存器

寄存器名称	寄存器描述
TRNG_CTL	控制寄存器
TRNG_STAT	状态寄存器
TRNG_DATA	数据寄存器

3.35.2. 外设库函数说明

TRNG库函数列表如下表所示：

表 3-1261. 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	除能TRNG接口
trng_lock	锁定TRNG控制位域
trng_mode_config	配置TRNG工作模式
trng_postprocessing_enable	使能TRNG后处理模块
trng_postprocessing_disable	失能TRNG后处理模块
trng_conditioning_enable	使能TRNG训练单元
trng_conditioning_disable	失能TRNG训练单元
trng_conditioning_input_bitwidth	配置训练单元输入位宽
trng_conditioning_output_bitwidth	配置训练单元输出位宽
trng_replace_test_enable	使能替换测试
trng_replace_test_disable	失能替换测试
trng_hash_init_enable	当训练单元使能时使能哈希算法初始化
trng_hash_init_disable	当训练单元使能时失能哈希算法初始化
trng_powermode_config	配置功耗模式
trng_clockdiv_config	配置时钟分频系数
trng_clockerror_detection_enable	使能时钟错误检测
trng_clockerror_detection_disable	失能时钟错误检测
trng_get_true_random_data	获取真随机值
trng_conditioning_reset_enable	复位训练单元使能
trng_conditioning_reset_disable	复位训练单元失能
trng_conditioning_algo_config	配置训练单元算法
trng_health_tests_config	配置健康测试
trng_flag_get	获取TRNG状态标志
trng_interrupt_enable	使能TRNG中断
trng_interrupt_disable	除能TRNG中断
trng_interrupt_flag_get	获取TRNG中断标志
trng_interrupt_flag_clear	清除TRNG中断标志

枚举 trng_inmod_enum

表 3-1262. 枚举 trng_inmod_enum

成员名称	功能描述
TRNG_INMOD_256BIT	训练单元256比特输入位宽
TRNG_INMOD_440BIT	训练单元440比特输入位宽

枚举 trng_outmod_enum

表 3-1263. 枚举 trng_outmod_enum

成员名称	功能描述
TRNG_OUTMOD_128BIT	训练单元128比特输出位宽
TRNG_OUTMOD_256BIT	训练单元256比特输出位宽

枚举 trng_modsel_enum

表 3-1264. 枚举 trng_modsel_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_flag_enum

表 3-1265. 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 3-1266. 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 3-1267. 函数 trng_deinit

函数名称	trng_deinit
函数原形	void trng_deinit (void);
功能描述	复位TRNG

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

例如：

```
/* TRNG deinit */
```

```
trng_deinit( );
```

函数 trng_enable

函数trng_enable描述见下表：

表 3-1268. 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 the TRNG interface */
```

```
trng_enable( );
```

函数 trng_disable

函数trng_disable描述见下表：

表 3-1269. 函数 trng_disable

函数名称	trng_disable
函数原形	void trng_disable(void);
功能描述	除能TRNG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 TRNG interface */
```

```
trng_disable( );
```

函数 trng_lock

函数trng_lock描述见下表:

表 3-1270. 函数 trng_lock

函数名称	trng_lock
函数原形	void trng_lock(void);
功能描述	锁定TRNG控制位域
先决条件	-
被调用函数	-
输入参数{in}	
-	-
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* lock the TRNG control bits */
```

```
trng_lock( );
```

函数 trng_mode_config

函数trng_mode_config描述见下表:

表 3-1271. 函数 trng_mode_config

函数名称	trng_mode_config
函数原形	void trng_mode_config(trng_modsel_enum mode_select);
功能描述	配置TRNG工作模式
先决条件	trng_conditioning_reset_enable
被调用函数	-

输入参数{in}	
mode_select	TRNG工作模式，参考 表3-1264. 枚举trng_modsel_enum
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* configure TRNG module */

trng_deinit();

trng_conditioning_reset_enable();

trng_mode_config(TRNG_MODSEL_NIST);

trng_postprocessing_enable();

trng_conditioning_enable();

trng_conditioning_input_bitwidth(TRNG_INMOD_440BIT);

trng_conditioning_output_bitwidth(TRNG_OUTMOD_256BIT);

trng_clockerror_detection_enable();

trng_enable();

trng_conditioning_reset_disable();
```

函数 trng_postprocessing_enable

函数trng_postprocessing_enable描述见下表：

表 3-1272. 函数 trng_postprocessing_enable

函数名称	trng_postprocessing_enable
函数原形	void trng_postprocessing_enable (void);
功能描述	使能TRNG后处理模块
先决条件	trng_conditioning_reset_enable
被调用函数	-
输入参数{in}	
-	-
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* configure TRNG module */
```

```

trng_deinit();

trng_conditioning_reset_enable();

trng_mode_config(TRNG_MODSEL_NIST);

trng_postprocessing_enable();

trng_conditioning_enable();

trng_conditioning_input_bitwidth(TRNG_INMOD_440BIT);

trng_conditioning_output_bitwidth(TRNG_OUTMOD_256BIT);

trng_clockerror_detection_enable();

trng_enable();

trng_conditioning_reset_disable();

```

函数 trng_postprocessing_disable

函数trng_postprocessing_disable描述见下表：

表 3-1273. 函数 trng_postprocessing_disable

函数名称	trng_postprocessing_disable
函数原形	void trng_postprocessing_disable(void);
功能描述	失能TRNG后处理模块
先决条件	trng_conditioning_reset_enable
被调用函数	-
输入参数{in}	
-	-
输出参数{out}	
-	-
返回值	
-	-

例如：

```

/* configure TRNG module */

trng_deinit();

trng_conditioning_reset_enable();

trng_mode_config(TRNG_MODSEL_NIST);

trng_postprocessing_disable();

trng_conditioning_enable();

trng_conditioning_input_bitwidth(TRNG_INMOD_440BIT);

```

```
trng_conditioning_output_bitwidth(TRNG_OUTMOD_256BIT);
```

```
trng_clockerror_detection_enable();
```

```
trng_enable();
```

```
trng_conditioning_reset_disable();
```

函数 trng_conditioning_enable

函数trng_conditioning_enable描述见下表：

表 3-1274. 函数 trng_conditioning_enable

函数名称	trng_conditioning_enable
函数原形	void trng_conditioning_enable(void);
功能描述	使能TRNG训练单元
先决条件	trng_conditioning_reset_enable
被调用函数	-
输入参数{in}	
-	-
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* configure TRNG module */

trng_deinit();

trng_conditioning_reset_enable();

trng_mode_config(TRNG_MODSEL_NIST);

trng_postprocessing_enable();

trng_conditioning_enable();

trng_conditioning_input_bitwidth(TRNG_INMOD_440BIT);

trng_conditioning_output_bitwidth(TRNG_OUTMOD_256BIT);

trng_clockerror_detection_enable();

trng_enable();

trng_conditioning_reset_disable();
```

函数 trng_conditioning_disable

函数trng_conditioning_disable描述见下表：

表 3-1275. 函数 `trng_conditioning_disable`

函数名称	<code>trng_conditioning_disable</code>
函数原形	<code>void trng_conditioning_disable(void);</code>
功能描述	失能TRNG训练单元
先决条件	<code>trng_conditioning_reset_enable</code>
被调用函数	-
输入参数{in}	
-	-
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* configure TRNG module */

trng_deinit();

trng_conditioning_reset_enable();

trng_mode_config(TRNG_MODSEL_NIST);

trng_postprocessing_enable();

trng_conditioning_disable();

trng_conditioning_input_bitwidth(TRNG_INMOD_440BIT);

trng_conditioning_output_bitwidth(TRNG_OUTMOD_256BIT);

trng_clockerror_detection_enable();

trng_enable();

trng_conditioning_reset_disable();
```

函数 `trng_conditioning_input_bitwidth`

函数`trng_conditioning_input_bitwidth`描述见下表：

表 3-1276. 函数 `trng_conditioning_input_bitwidth`

函数名称	<code>trng_conditioning_input_bitwidth</code>
函数原形	<code>void trng_conditioning_input_bitwidth(trng_inmod_enum input_bitwidth);</code>
功能描述	配置训练单元输入位宽
先决条件	<code>trng_conditioning_reset_enable</code>
被调用函数	-
输入参数{in}	
<code>input_bitwidth</code>	输入位宽，参考 表3-1262. 枚举<code>trng_inmod_enum</code>
输出参数{out}	

-	-
返回值	
-	-

例如：

```
/* configure TRNG module */

trng_deinit();

trng_conditioning_reset_enable();

trng_mode_config(TRNG_MODSEL_NIST);

trng_postprocessing_enable();

trng_conditioning_enable();

trng_conditioning_input_bitwidth(TRNG_INMOD_440BIT);

trng_conditioning_output_bitwidth(TRNG_OUTMOD_256BIT);

trng_clockerror_detection_enable();

trng_enable();

trng_conditioning_reset_disable();
```

函数 `trng_conditioning_output_bitwidth`

函数 `trng_conditioning_output_bitwidth` 描述见下表：

表 3-1277. 函数 `trng_conditioning_output_bitwidth`

函数名称	<code>trng_conditioning_output_bitwidth</code>
函数原形	<code>void trng_conditioning_output_bitwidth(trng_outmod_enum output_bitwidth);</code>
功能描述	配置训练单元输出位宽
先决条件	<code>trng_conditioning_reset_enable</code>
被调用函数	-
输入参数{in}	
output_bitwidth	输出位宽，参考 表3-1263. 枚举 <code>trng_outmod_enum</code>
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* configure TRNG module */

trng_deinit();

trng_conditioning_reset_enable();
```

```

trng_mode_config(TRNG_MODSEL_NIST);

trng_postprocessing_enable();

trng_conditioning_enable();

trng_conditioning_input_bitwidth(TRNG_INMOD_440BIT);

trng_conditioning_output_bitwidth(TRNG_OUTMOD_256BIT);

trng_clockerror_detection_enable();

trng_enable();

trng_conditioning_reset_disable();

```

函数 trng_replace_test_enable

函数trng_replace_test_enable描述见下表：

表 3-1278. 函数 trng_replace_test_enable

函数名称	trng_replace_test_enable
函数原形	void trng_replace_test_enable(void);
功能描述	使能替换测试
先决条件	trng_conditioning_reset_enable
被调用函数	-
输入参数{in}	
-	-
输出参数{out}	
-	-
返回值	
-	-

例如：

```

/* enable TRNG replace test */

trng_deinit();

trng_conditioning_reset_enable();

trng_replace_test_enable();

trng_enable();

trng_conditioning_reset_disable();

```

函数 trng_replace_test_disable

函数trng_replace_test_disable描述见下表：

表 3-1279. 函数 `trng_replace_test_disable`

函数名称	<code>trng_replace_test_disable</code>
函数原形	<code>void trng_replace_test_disable(void);</code>
功能描述	失能替换测试
先决条件	<code>trng_conditioning_reset_enable</code>
被调用函数	-
输入参数{in}	
-	-
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* disable TRNG replace test */

trng_deinit();

trng_conditioning_reset_enable();

trng_replace_test_disable();

trng_enable();

trng_conditioning_reset_disable();
```

函数 `trng_hash_init_enable`

函数`trng_hash_init_enable`描述见下表：

表 3-1280. 函数 `trng_hash_init_enable`

函数名称	<code>trng_hash_init_enable</code>
函数原形	<code>void trng_hash_init_enable(void);</code>
功能描述	当训练单元使能时使能哈希算法初始化
先决条件	<code>trng_conditioning_reset_enable</code> / <code>trng_conditioning_enable</code>
被调用函数	-
输入参数{in}	
-	-
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* enable hash algorithm init when conditioning module enabled */

trng_deinit();
```

```
trng_conditioning_reset_enable();
```

```
trng_conditioning_enable();
```

```
trng_hash_init_enable();
```

```
trng_enable();
```

```
trng_conditioning_reset_disable();
```

函数 trng_hash_init_disable

函数trng_hash_init_disable描述见下表:

表 3-1281. 函数 trng_hash_init_disable

函数名称	trng_hash_init_disable
函数原形	void trng_hash_init_disable(void);
功能描述	当训练单元使能时失能哈希算法初始化
先决条件	trng_conditioning_reset_enable / trng_conditioning_enable
被调用函数	-
输入参数{in}	
-	-
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* disable the TRNG interface */
```

```
trng_deinit();
```

```
trng_conditioning_reset_enable();
```

```
trng_conditioning_enable();
```

```
trng_hash_init_disable();
```

```
trng_enable();
```

```
trng_conditioning_reset_disable();
```

函数 trng_powermode_config

函数trng_powermode_config描述见下表:

表 3-1282. 函数 trng_powermode_config

函数名称	trng_powermode_config
函数原形	void trng_powermode_config(uint32_t powermode);
功能描述	配置功耗模式

先决条件	trng_conditioning_reset_enable
被调用函数	-
输入参数{in}	
powermode	功耗模式选项
TRNG_NR_ULATRL0 W	极低功耗模式
TRNG_NR_LOW	低功耗模式
TRNG_NR_MEDIUM	中等功耗模式
TRNG_NR_HIGH	高功耗模式
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* configure the TRNG analog power mode as high */
trng_deinit();
trng_conditioning_reset_enable();
trng_powermode_config(TRNG_NR_HIGH);
trng_enable();
trng_conditioning_reset_disable();
```

函数 trng_clockdiv_config

函数trng_clockdiv_config描述见下表：

表 3-1283. 函数 trng_clockdiv_config

函数名称	trng_clockdiv_config
函数原形	void trng_clockdiv_config(uint32_t clkdiv);
功能描述	配置时钟分频系数
先决条件	trng_conditioning_reset_enable
被调用函数	-
输入参数{in}	
clkdiv	时钟分频系数
TRNG_CLK_DIVx	对TRNG时钟x分频，x = 1, 2, 4, 8, 16, 32, 64, 128, 256, 512, 1024, 2048, 4096, 8192, 16384, 32768
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* configure the TRNG clock frequency division coefficient to 64 */  
trng_deinit();  
trng_conditioning_reset_enable();  
trng_clockdiv_config(TRNG_CLK_DIV64);  
trng_enable();  
trng_conditioning_reset_disable();
```

函数 trng_clockerror_detection_enable

函数trng_clockerror_detection_enable描述见下表:

表 3-1284. 函数 trng_clockerror_detection_enable

函数名称	trng_clockerror_detection_enable
函数原形	void trng_clockerror_detection_enable(void);
功能描述	使能时钟错误检测
先决条件	trng_conditioning_reset_enable
被调用函数	-
输入参数{in}	
-	-
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* enable the TRNG clock error detection */  
trng_deinit();  
trng_conditioning_reset_enable();  
trng_clockerror_detection_enable();  
trng_enable();  
trng_conditioning_reset_disable();
```

函数 trng_clockerror_detection_disable

函数trng_clockerror_detection_disable描述见下表:

表 3-1285. 函数 trng_clockerror_detection_disable

函数名称	trng_clockerror_detection_disable
------	-----------------------------------

函数原形	void trng_clockerror_detection_disable(void);
功能描述	失能时钟错误检测
先决条件	trng_conditioning_reset_enable
被调用函数	-
输入参数{in}	
-	-
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* disable the TRNG clock error detection */

trng_deinit();

trng_conditioning_reset_enable();

trng_clockerror_detection_disable();

trng_enable();

trng_conditioning_reset_disable();
```

函数 trng_get_true_random_data

函数trng_get_true_random_data描述见下表:

表 3-1286. 函数 trng_get_true_random_data

函数名称	trng_get_true_random_data
函数原形	uint32_t trng_get_true_random_data(void);
功能描述	获取真随机值
先决条件	-
被调用函数	-
输入参数{in}	
-	-
输出参数{out}	
-	-
返回值	
uint32_t	0x0 – 0xFFFFFFFF

例如:

```
/* get the true random data */

uint32_t data = 0;

data = trng_get_true_random_data();
```

函数 trng_conditioning_reset_enable

函数trng_conditioning_reset_enable描述见下表：

表 3-1287. 函数 trng_conditioning_reset_enable

函数名称	trng_conditioning_reset_enable
函数原形	void trng_conditioning_reset_enable(void)
功能描述	复位训练单元失能
先决条件	-
被调用函数	-
输入参数{in}	
-	-
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* configure TRNG module */  
  
trng_deinit();  
  
trng_conditioning_reset_enable();  
  
trng_mode_config(TRNG_MODSEL_NIST);  
  
trng_postprocessing_enable();  
  
trng_conditioning_enable();  
  
trng_conditioning_input_bitwidth(TRNG_INMOD_440BIT);  
  
trng_conditioning_output_bitwidth(TRNG_OUTMOD_256BIT);  
  
trng_clockerror_detection_enable();  
  
trng_enable();  
  
trng_conditioning_reset_disable();
```

函数 trng_conditioning_reset_disable

函数trng_conditioning_reset_disable描述见下表：

表 3-1288. 函数 trng_conditioning_reset_disable

函数名称	trng_conditioning_reset_disable
函数原形	void trng_conditioning_reset_disable(void)
功能描述	复位训练单元使能
先决条件	-

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

例如：

```
/* configure TRNG module */

trng_deinit();

trng_conditioning_reset_enable();

trng_mode_config(TRNG_MODSEL_NIST);

trng_postprocessing_enable();

trng_conditioning_enable();

trng_conditioning_input_bitwidth(TRNG_INMOD_440BIT);

trng_conditioning_output_bitwidth(TRNG_OUTMOD_256BIT);

trng_clockerror_detection_enable();

trng_enable();

trng_conditioning_reset_disable();
```

函数 trng_conditioning_algo_config

函数trng_conditioning_algo_config描述见下表：

表 3-1289. 函数 trng_conditioning_algo_config

函数名称	trng_conditioning_algo_config
函数原形	void trng_conditioning_algo_config(uint32_t module_algo)
功能描述	配置训练单元算法
先决条件	trng_conditioning_reset_enable
被调用函数	-
输入参数{in}	
module_algo	模块算法
TRNG_ALGO_SHA1	训练模块使用SHA1算法
TRNG_ALGO_MD5	训练模块使用MD5算法
TRNG_ALGO_SHA224	训练模块使用SHA224算法
TRNG_ALGO_SHA256	训练模块使用SHA256算法
输出参数{out}	
-	-

返回值	
-	-

例如：

```
/* configure TRNG module */

trng_deinit();

trng_conditioning_reset_enable();

trng_mode_config(TRNG_MODSEL_NIST);

trng_postprocessing_enable();

trng_conditioning_enable();

trng_conditioning_input_bitwidth(TRNG_INMOD_440BIT);

trng_conditioning_output_bitwidth(TRNG_OUTMOD_256BIT);

trng_clockerror_detection_enable();

trng_conditioning_algo_config(TRNG_ALGO_SHA1);

trng_enable();

trng_conditioning_reset_disable();
```

函数 trng_health_tests_config

函数trng_health_tests_config描述见下表：

表 3-1290. 函数 trng_health_tests_config

函数名称	trng_health_tests_config
函数原形	void trng_health_tests_config(uint32_t adpo_threshold, uint8_t rep_threshold)
功能描述	配置健康测试
先决条件	trng_conditioning_reset_enable
被调用函数	-
输入参数{in}	
adpo_threshold	自适应比例测试阈值
输入参数{in}	
rep_threshold	重复测试（00 / 11）阈值
输出参数{out}	
-	-
返回值	
-	-

例如：

```

/* configure TRNG module */

trng_deinit();

trng_conditioning_reset_enable();

trng_mode_config(TRNG_MODSEL_NIST);

trng_postprocessing_enable();

trng_conditioning_enable();

trng_conditioning_input_bitwidth(TRNG_INMOD_440BIT);

trng_conditioning_output_bitwidth(TRNG_OUTMOD_256BIT);

trng_clockerror_detection_enable();

trng_health_tests_config(700, 50);

trng_enable();

trng_conditioning_reset_disable();

```

函数 trng_flag_get

函数trng_flag_get描述见下表：

表 3-1291. 函数 trng_flag_get

函数名称	trng_flag_get
函数原形	FlagStatus trng_flag_get(trng_flag_enum flag);
功能描述	获取TRNG状态标志
先决条件	-
被调用函数	-
输入参数{in}	
flag	TRNG状态标志，参考 表3-1265. 枚举trng_flag_enum
输出参数{out}	
-	-
返回值	
FlagStatus	SET或RESET

例如：

```

/* get the trng status flags */

FlagStatus status;

status = trng_flag_get(TRNG_FLAG_DRDY);

```

函数 trng_interrupt_enable

函数trng_interrupt_enable描述见下表：

表 3-1292. 函数 trng_interrupt_enable

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

例如:

```
/* enable the TRNG interrupt */
```

```
trng_interrupt_enable ( );
```

函数 trng_interrupt_disable

函数trng_interrupt_disable描述见下表:

表 3-1293. 函数 trng_interrupt_disable

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

例如:

```
/* disable the TRNG interrupt */
```

```
trng_interrupt_disable( );
```

函数 trng_interrupt_flag_get

函数trng_interrupt_flag_get描述见下表:

表 3-1294. 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, 参考 表3-1266. 枚举trng_int_flag_enum
输出参数{out}	
-	-
返回值	
FlagStatus	SET或RESET

例如:

```
/* get the trng interrupt flag */
```

```
FlagStatus status = RESET;
```

```
status = 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 3-1295. 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, 参考 表3-1266. 枚举trng_int_flag_enum
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/*clear the trng interrupt flag */
```

```
trng_interrupt_flag_clear (TRNG_INT_FLAG_CEIF);
```

3.36. 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[3.36.1](#)描述了USART的寄存器列表，章节[3.36.2](#)对USART库函数进行说明。

3.36.1. 外设寄存器说明

USART寄存器列表如下表所示：

表 3-1296. USART 寄存器

寄存器名称	寄存器描述
USART_CTL0	控制寄存器0
USART_CTL1	控制寄存器1
USART_CTL2	控制寄存器2
USART_BAUD	波特率发生寄存器
USART_GP	保护时间和预分频器寄存器
USART_RT	接收超时寄存器
USART_CMD	请求寄存器
USART_STAT	状态寄存器
USART_INTC	中断标志清除寄存器
USART_RDATA	数据接收寄存器
USART_TDATA	数据发送寄存器
USART_CHC	兼容性控制寄存器
USART_FCS	FIFO控制和状态寄存器

3.36.2. 外设库函数说明

USART库函数列表如下表所示：

表 3-1297. 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	禁能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_overrun_enable	使能USART溢出禁止功能

库函数名称	库函数描述
usart_overrun_disable	禁能USART溢出禁止功能
usart_oversample_config	配置USART过采样模式
usart_sample_bit_config	配置USART单次采样方式
usart_receiver_timeout_enable	使能USART接收超时
usart_receiver_timeout_disable	禁能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_command_enable	使能USART请求
usart_address_0_match_mode_enable	使能地址0匹配模式
usart_address_0_match_mode_disable	禁能地址0匹配模式
usart_address_1_match_mode_enable	使能地址1匹配模式
usart_address_1_match_mode_disable	禁能地址1匹配模式
usart_address_0_config	配置USART地址0
usart_address_1_config	配置USART地址1
usart_address_0_detection_mode_config	配置USART地址0检测模式
usart_address_1_detection_mode_config	配置USART地址1检测模式
usart_mute_mode_enable	使能USART静默模式
usart_mute_mode_disable	禁能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	禁能USART LIN模式
usart_lin_break_detection_length_config	配置USART LIN模式中中断帧长度
usart_halfduplex_enable	使能USART半双工模式
usart_halfduplex_disable	禁能USART半双工模式
usart_clock_enable	使能USART CK引脚
usart_clock_disable	禁能USART 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	禁能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智能卡模式下禁能NACK

库函数名称	库函数描述
usart_smartcard_mode_early_nack_enable	使能USART智能卡模式提前NACK
usart_smartcard_mode_early_nack_disable	禁能USART智能卡模式提前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	禁能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_hardware_flow_coherence_config	配置硬件流控兼容模式
usart_rs485_driver_enable	使能USART rs485驱动
usart_rs485_driver_disable	禁能USART rs485驱动
usart_driver_asserttime_config	配置USART驱动使能置位时间
usart_driver_deasserttime_config	配置USART驱动使能置低时间
usart_depolarity_config	配置USART驱动使能极性模式
usart_dma_receive_config	配置USART DMA接收
usart_dma_transmit_config	配置USART DMA发送
usart_reception_error_dma_disable	USART接收错误时禁能DMA
usart_reception_error_dma_enable	USART接收错误时使能DMA
usart_wakeup_enable	使能USART唤醒
usart_wakeup_disable	禁能USART唤醒
usart_wakeup_mode_config	配置USART唤醒模式
usart_fifo_enable	使能FIFO
usart_fifo_disable	禁能FIFO
usart_transmit_fifo_threshold_config	配置发送FIFO阈值
usart_receive_fifo_threshold_config	配置接收FIFO阈值
usart_receive_fifo_counter_number	读取接收FIFO计数器的值
usart_flag_get	获取STAT/RFCs寄存器中的标志
usart_flag_clear	清除USART状态
usart_interrupt_enable	使能USART中断
usart_interrupt_disable	禁能USART中断
usart_interrupt_flag_get	获取USART中断和标志状态
usart_interrupt_flag_clear	清除USART中断标志位

枚举类型 `usart_flag_enum`

表 3-1298. 枚举类型 `usart_flag_enum`

成员名称	功能描述
USART_FLAG_REA	接收使能通知标志
USART_FLAG_TEA	发送使能通知标志
USART_FLAG_WU	从深度睡眠模式唤醒标志
USART_FLAG_RWU	接收器从静默模式唤醒
USART_FLAG_SB	断开信号发送标志
USART_FLAG_AM0	地址0匹配标志
USART_FLAG_BSY	忙标志
USART_FLAG_AM1	地址1匹配标志
USART_FLAG_EB	块结束标志
USART_FLAG_RT	接收超时标志
USART_FLAG_CTS	CTS电平
USART_FLAG_CTSF	CTS变化标志
USART_FLAG_LBD	LIN断开检测标志
USART_FLAG_TBE	发送数据寄存器空
USART_FLAG_TFN	发送FIFO非满
USART_FLAG_TC	发送完成
USART_FLAG_RBNE	读数据缓冲区非空
USART_FLAG_RFN	接收FIFO非空
USART_FLAG_IDLE	空闲线检测标志
USART_FLAG_ORERR	溢出错误
USART_FLAG_NERR	噪声错误标志
USART_FLAG_FERR	帧错误
USART_FLAG_PERR	校验错误
USART_FLAG_EPERR	校验错误超前检测标志
USART_FLAG_RFF	接收FIFO满标志
USART_FLAG_RFE	接收FIFO空标志
USART_FLAG_TFF	发送FIFO满标志
USART_FLAG_TFE	发送FIFO空标志
USART_FLAG_TFT	发送FIFO阈值到达标志
USART_FLAG_RFT	接收FIFO阈值到达标志

枚举类型 `usart_interrupt_flag_enum`

表 3-1299. 枚举类型 `usart_interrupt_flag_enum`

成员名称	功能描述
USART_INT_FLAG_AM1	地址1匹配中断标志
USART_INT_FLAG_EB	块结束中断标志
USART_INT_FLAG_RT	接收超时中断标志
USART_INT_FLAG_AM0	地址匹配中断标志

成员名称	功能描述
USART_INT_FLAG_PERR	奇偶校验错误中断标志
USART_INT_FLAG_TBE	发送寄存器空中断标志
USART_INT_FLAG_TFNF	发送FIFO非满中断标志
USART_INT_FLAG_TC	发送完成中断标志
USART_INT_FLAG_RBNE	读缓冲区非空中断标志
USART_INT_FLAG_RFNE	接收FIFO非空中断标志
USART_INT_FLAG_RBNE_ORE RR	读缓冲区非空和溢出中断标志
USART_INT_FLAG_RFNE_ORE RR	接收FIFO非空和溢出中断标志
USART_INT_FLAG_IDLE	空闲线检测中断标志
USART_INT_FLAG_LBD	LIN断开检测中断标志
USART_INT_FLAG_WU	从深度睡眠模式唤醒中断标志
USART_INT_FLAG_CTS	CTS中断标志
USART_INT_FLAG_ERR_NERR	噪声错误中断标志
USART_INT_FLAG_ERR_ORER R	溢出错误中断标志
USART_INT_FLAG_ERR_FERR	帧错误中断标志
USART_INT_FLAG_TFT	发送FIFO阈值到达中断标志
USART_INT_FLAG_TFE	发送FIFO空中断标志
USART_INT_FLAG_RFT	接收FIFO阈值到达中断标志
USART_INT_FLAG_RFF	接收FIFO满中断标志

枚举类型 `usart_interrupt_enum`

表 3-1300. 枚举类型 `usart_interrupt_enum`

成员名称	功能描述
USART_INT_AM1	地址1匹配中断使能
USART_INT_EB	块结束中断使能
USART_INT_RT	接收超时中断使能
USART_INT_AM0	地址0匹配中断使能
USART_INT_PERR	奇偶校验错误中断使能
USART_INT_TBE	发送寄存器空中断使能
USART_INT_TFNF	发送FIFO非满中断使能
USART_INT_TC	发送完成中断使能
USART_INT_RBNE	读缓冲区非空中断和溢出错误中断使能
USART_INT_RFNE	接收FIFO非空中断使能
USART_INT_IDLE	空闲线检测中断使能
USART_INT_LBD	LIN断开检测中断使能
USART_INT_WU	从深度睡眠模式唤醒中断使能
USART_INT_CTS	CTS中断使能
USART_INT_ERR	错误中断使能

成员名称	功能描述
USART_INT_TFE	发送FIFO空中断使能
USART_INT_TFT	发送FIFO阈值到达中断使能
USART_INT_RFT	接收FIFO阈值到达中断使能
USART_INT_RFF	接收FIFO满中断使能

枚举类型 `usart_invert_enum`

表 3-1301. 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
USART_SWAP_ENABLE	交换TX/RX管脚
USART_SWAP_DISABLE	不交换TX/RX管脚

函数 `usart_deinit`

函数`usart_deinit`描述见下表：

表 3-1302. 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
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}	
usart_periph	外设USARTx
<i>USARTx</i>	x = 0, 1, 2, 5
<i>UARTx</i>	x = 3, 4, 6, 7
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* reset USART0 */
usart_deinit(USART0);
```

函数 usart_baudrate_set

函数usart_baudrate_set描述见下表：

表 3-1303. 函数 usart_baudrate_set

函数名称	usart_baudrate_set
函数原型	void usart_baudrate_set(uint32_t usart_periph, uint32_t baudval);
功能描述	配置USART波特率
先决条件	-
被调用函数	rcu_clock_freq_get
输入参数{in}	
usart_periph	外设USARTx
USARTx	x = 0, 1, 2, 5
UARTx	x = 3, 4, 6, 7
输入参数{in}	
baudval	波特率值
输出参数{out}	
-	-
返回值	
-	-

例如：

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

函数 usart_parity_config

函数usart_parity_config描述见下表：

表 3-1304. 函数 usart_parity_config

函数名称	usart_parity_config
函数原型	void usart_parity_config(uint32_t usart_periph, uint32_t paritycfg);
功能描述	配置USART奇偶校验
先决条件	-
被调用函数	-
输入参数{in}	
usart_periph	外设USARTx
USARTx	x = 0, 1, 2, 5
UARTx	x = 3, 4, 6, 7
输入参数{in}	
paritycfg	配置USART奇偶校验
USART_PM_NONE	无校验
USART_PM_ODD	奇校验

USART_PM_EVEN	偶校验
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* configure USART0 parity */
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 3-1305. 函数 usart_word_length_set

函数名称	usart_word_length_set
函数原型	void usart_word_length_set(uint32_t usart_periph, uint32_t wlen);
功能描述	配置USART字长
先决条件	-
被调用函数	-
输入参数{in}	
usart_periph	外设USARTx
USARTx	x = 0, 1, 2, 5
UARTx	x = 3, 4, 6, 7
输入参数{in}	
wlen	配置USART字长
USART_WL_8BIT	8位
USART_WL_9BIT	9位
USART_WL_7BIT	7位
USART_WL_10BIT	10位
输出参数{out}	
-	-
返回值	
-	-

例如：

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

函数 usart_stop_bit_set

函数usart_stop_bit_set描述见下表：

表 3-1306. 函数 usart_stop_bit_set

函数名称	usart_stop_bit_set
函数原型	void usart_stop_bit_set(uint32_t usart_periph, uint32_t stblen);
功能描述	配置USART停止位
先决条件	-
被调用函数	-
输入参数{in}	
usart_periph	外设USARTx
USARTx	x = 0, 1, 2, 5
UARTx	x = 3, 4, 6, 7
输入参数{in}	
stblen	配置USART停止位
USART_STB_1BIT	1位
USART_STB_0_5BIT	0.5位
USART_STB_2BIT	2位
USART_STB_1_5BIT	1.5位
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 3-1307. 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
USARTx	x = 0, 1, 2, 5
UARTx	x = 3, 4, 6, 7
输出参数{out}	

-	-
返回值	
-	-

例如：

```
/* enable USART0 */
```

```
usart_enable(USART0);
```

函数 usart_disable

函数usart_disable描述见下表：

表 3-1308. 函数 usart_disable

函数名称	usart_disable
函数原型	void usart_disable(uint32_t usart_periph);
功能描述	禁能USART
先决条件	-
被调用函数	-
输入参数{in}	
usart_periph	外设USARTx
USARTx	x = 0, 1, 2, 5
UARTx	x = 3, 4, 6, 7
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* disable USART0 */
```

```
usart_disable(USART0);
```

函数 usart_transmit_config

函数usart_transmit_config描述见下表：

表 3-1309. 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

<i>USARTx</i>	<i>x</i> = 0, 1, 2, 5
<i>UARTx</i>	<i>x</i> = 3, 4, 6, 7
输入参数{in}	
txconfig	使能/禁能USART发送器
<i>USART_TRANSMIT_ENABLE</i>	使能USART发送
<i>USART_TRANSMIT_DISABLE</i>	禁能USART发送
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* configure USART0 transmitter */
```

```
usart_transmit_config(USART0,USART_TRANSMIT_ENABLE);
```

函数 **usart_receive_config**

函数usart_receive_config描述见下表：

表 3-1310. 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
<i>USARTx</i>	<i>x</i> = 0, 1, 2, 5
<i>UARTx</i>	<i>x</i> = 3, 4, 6, 7
输入参数{in}	
rxconfig	使能/禁能USART接收器
<i>USART_RECEIVE_ENABLE</i>	使能USART接收
<i>USART_RECEIVE_DISABLE</i>	禁能USART接收
输出参数{out}	
-	-
返回值	
-	-

例如：


```
/* configure USART0 receiver */
```

```
usart_receive_config(USART0, USART_RECEIVE_ENABLE);
```

函数 usart_data_first_config

函数usart_data_first_config描述见下表：

表 3-1311. 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, 5
UARTx	x = 3, 4, 6, 7
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 3-1312. 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
<i>USARTx</i>	x = 0, 1, 2, 5
<i>UARTx</i>	x = 3, 4, 6, 7
输入参数{in}	
invertpara	参考 表3-1301. 枚举类型usart_invert_enum
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* configure USART0 inversion */
```

```
usart_invert_config(USART0, USART_DINV_ENABLE);
```

函数 usart_overrun_enable

函数usart_overrun_enable描述见下表：

表 3-1313. 函数 usart_overrun_enable

函数名称	usart_overrun_enable
函数原型	void usart_overrun_enable(uint32_t usart_periph);
功能描述	使能USART溢出禁止功能
先决条件	-
被调用函数	-
输入参数{in}	
usart_periph	外设USARTx
<i>USARTx</i>	x = 0, 1, 2, 5
<i>UARTx</i>	x = 3, 4, 6, 7
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* enable USART0 overrun */
```

```
usart_overrun_enable(USART0);
```

函数 usart_overrun_disable

函数usart_overrun_disable描述见下表：

表 3-1314. 函数 usart_overrun_disable

函数名称	usart_overrun_disable
-------------	-----------------------

函数原型	void usart_oversample_config(uint32_t usart_periph);
功能描述	配置USART过采样模式
先决条件	-
被调用函数	-
输入参数{in}	
usart_periph	外设USARTx
USARTx	x = 0, 1, 2, 5
UARTx	x = 3, 4, 6, 7
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* disable USART0 overrun */
```

```
usart_oversample_config(USART0);
```

函数 usart_oversample_config

函数usart_oversample_config描述见下表：

表 3-1315. 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
USARTx	x = 0, 1, 2, 5
UARTx	x = 3, 4, 6, 7
输入参数{in}	
oversamp	过采样值
USART_OVSMOD_8	8倍过采样
USART_OVSMOD_16	16倍过采样
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* configure USART0 oversampling by 8 */
```

```
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 3-1316. 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 osb);
功能描述	配置USART单次采样方式
先决条件	-
被调用函数	-
输入参数{in}	
usart_periph	外设USARTx
USARTx	x = 0, 1, 2, 5
UARTx	x = 3, 4, 6, 7
输入参数{in}	
osb	单次采样方式
USART_OSB_1BIT	1次采样方法
USART_OSB_3BIT	3次采样方法
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* config USART0 1 bit sample mode */
```

```
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 3-1317. 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, 5

输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* enable USART0 receiver timeout */
```

```
usart_receiver_timeout_enable(USART0);
```

函数 usart_receiver_timeout_disable

函数usart_receiver_timeout_disable描述见下表：

表 3-1318. 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	禁能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, 5
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* disable USART0 receiver timeout */
```

```
usart_receiver_timeout_disable(USART0);
```

函数 usart_receiver_timeout_threshold_config

函数usart_receiver_timeout_threshold_config描述见下表：

表 3-1319. 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
<i>USARTx</i>	x = 0, 1, 2, 5
输入参数{in}	
rtimeout	超时时间（0x00000000-0x00FFFFFF）
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 3-1320. 函数 usart_data_transmit

函数名称	usart_data_transmit
函数原型	void usart_data_transmit(uint32_t usart_periph, uint16_t data);
功能描述	USART发送数据功能
先决条件	-
被调用函数	-
输入参数{in}	
usart_periph	外设USARTx
<i>USARTx</i>	x = 0, 1, 2, 5
<i>UARTx</i>	x = 3, 4, 6, 7
输入参数{in}	
data	发送的数据（0x0000-0x01FF）
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 3-1321. 函数 `usart_data_receive`

函数名称	<code>usart_data_receive</code>
函数原型	<code>uint16_t usart_data_receive(uint32_t usart_periph);</code>
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}	
<code>usart_periph</code>	外设USARTx
<code>USARTx</code>	x = 0, 1, 2, 5
<code>UARTx</code>	x = 3, 4, 6, 7
输出参数{out}	
-	-
返回值	
<code>uint16_t</code>	接收到的数据 (0x0000-0x01FF)

例如:

```
/* USART0 receive data */
```

```
uint16_t temp;
```

```
temp = usart_data_receive(USART0);
```

函数 `usart_command_enable`

函数`usart_command_enable`描述见下表:

表 3-1322. 函数 `usart_command_enable`

函数名称	<code>usart_command_enable</code>
函数原型	<code>void usart_command_enable(uint32_t usart_periph, uint32_t cmdtype);</code>
功能描述	使能USART请求
先决条件	-
被调用函数	-
输入参数{in}	
<code>usart_periph</code>	外设USARTx
<code>USARTx</code>	x = 0, 1, 2, 5
<code>UARTx</code>	x = 3, 4, 6, 7
输入参数{in}	
<code>cmdtype</code>	请求类型
<code>USART_CMD_SBK_CMD</code>	发送断开帧请求
<code>USART_CMD_MM_CMD</code>	静模式请求
<code>USART_CMD_RXF_CMD</code>	接收数据清空请求

USART_CMD_TXF CMD	发送数据清空请求
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* enable USART0 command */
```

```
usart_command_enable(USART0, USART_CMD_SBKCMD);
```

函数 usart_address_0_match_mode_enable

函数usart_address_0_match_mode_enable描述见下表：

表 3-1323. 函数 usart_address_0_match_mode_enable

函数名称	usart_address_0_match_mode_enable
函数原型	void usart_address_0_match_mode_enable(uint32_t usart_periph);
功能描述	使能地址0匹配模式
先决条件	-
被调用函数	-
输入参数{in}	
usart_periph	外设USARTx
USARTx	x = 0, 1, 2, 5
UARTx	x = 3, 4, 6, 7
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* enable address 0 match mode */
```

```
usart_address_0_match_mode_enable (USART0);
```

函数 usart_address_0_match_mode_disable

函数usart_address_0_match_mode_disable描述见下表：

表 3-1324. 函数 usart_address_0_match_mode_disable

函数名称	usart_address_0_match_mode_disable
函数原型	void usart_address_0_match_mode_disable(uint32_t usart_periph);
功能描述	禁能地址0匹配模式
先决条件	-

被调用函数	-
输入参数{in}	
usart_periph	外设USARTx
USARTx	x = 0, 1, 2, 5
UARTx	x = 3, 4, 6, 7
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* disable address 0 match mode */
```

```
usart_address_0_match_mode_disable(USART0);
```

函数 usart_address_1_match_mode_enable

函数usart_address_1_match_mode_enable描述见下表：

表 3-1325. 函数 usart_address_1_match_mode_enable

函数名称	usart_address_1_match_mode_enable
函数原型	void usart_address_1_match_mode_enable(uint32_t usart_periph);
功能描述	使能地址1匹配模式
先决条件	-
被调用函数	-
输入参数{in}	
usart_periph	外设USARTx
USARTx	x = 0, 1, 2, 5
UARTx	x = 3, 4, 6, 7
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* enable address 1 match mode */
```

```
usart_address_1_match_mode_enable (USART0);
```

函数 usart_address_1_match_mode_disable

函数usart_address_1_match_mode_disable描述见下表：

表 3-1326. 函数 usart_address_1_match_mode_disable

函数名称	usart_address_1_match_mode_disable
------	------------------------------------

函数原型	void usart_address_1_match_mode_disable(uint32_t usart_periph);
功能描述	禁能地址1匹配模式
先决条件	-
被调用函数	-
输入参数{in}	
usart_periph	外设USARTx
USARTx	x = 0, 1, 2, 5
UARTx	x = 3, 4, 6, 7
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* disable address 1 match mode */
```

```
usart_address_1_match_mode_disable(USART0);
```

函数 usart_address_0_config

函数usart_address_0_config描述见下表：

表 3-1327. 函数 usart_address_0_config

函数名称	usart_address_0_config
函数原型	void usart_address_0_config (uint32_t usart_periph, uint8_t addr);
功能描述	配置USART地址0
先决条件	-
被调用函数	-
输入参数{in}	
usart_periph	外设USARTx
USARTx	x = 0, 1, 2, 5
UARTx	x = 3, 4, 6, 7
输入参数{in}	
addr	USART地址（0x00-0xFF）
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* configure address 0 of the USART0 */
```

```
usart_address_0_config (USART0, 0x00);
```

函数 usart_address_1_config

函数usart_address_1_config描述见下表:

表 3-1328. 函数 usart_address_1_config

函数名称	usart_address_1_config
函数原型	void usart_address_1_config (uint32_t usart_periph, uint8_t addr);
功能描述	配置USART地址1
先决条件	-
被调用函数	-
输入参数{in}	
usart_periph	外设USARTx
USARTx	x = 0, 1, 2, 5
UARTx	x = 3, 4, 6, 7
输入参数{in}	
addr	USART地址 (0x00-0xFF)
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* configure address 1 of the USART0 */
```

```
usart_address_1_config (USART0, 0x00);
```

函数 usart_address_0_detection_mode_config

函数usart_address_0_detection_mode_config描述见下表:

表 3-1329. 函数 usart_address_0_detection_mode_config

函数名称	usart_address_0_detection_mode_config
函数原型	void usart_address_0_detection_mode_config(uint32_t usart_periph, uint32_t addmod)
功能描述	配置USART地址检测模式
先决条件	-
被调用函数	-
输入参数{in}	
usart_periph	外设USARTx
USARTx	x = 0, 1, 2, 5
UARTx	x = 3, 4, 6, 7
输入参数{in}	
addmod	地址检测模式
USART_ADDM_4BI	4位地址检测

<i>T</i>	
<i>USART_ADDM_FU</i> <i>LLBIT</i>	全位地址检测
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* configure address 0 detection mode */
```

```
usart_address_0_detection_mode_config (USART0, USART_ADDM_4BIT);
```

函数 usart_address_1_detection_mode_config

函数usart_address_1_detection_mode_config描述见下表：

表 3-1330. 函数 usart_address_1_detection_mode_config

函数名称	usart_address_1_detection_mode_config
函数原型	void usart_address_1_detection_mode_config(uint32_t usart_periph, uint32_t addmod)
功能描述	配置USART地址检测模式
先决条件	-
被调用函数	-
输入参数{in}	
usart_periph	外设USARTx
<i>USARTx</i>	x = 0, 1, 2, 5
<i>UARTx</i>	x = 3, 4, 6, 7
输入参数{in}	
addmod	地址检测模式
<i>USART_ADDM_4BIT</i>	4位地址检测
<i>USART_ADDM_FU</i> <i>LLBIT</i>	全位地址检测
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* configure address 1 detection mode */
```

```
usart_address_1_detection_mode_config (USART0, USART_ADDM_4BIT);
```

函数 `usart_mute_mode_enable`

函数 `usart_mute_mode_enable` 描述见下表：

表 3-1331. 函数 `usart_mute_mode_enable`

函数名称	<code>usart_mute_mode_enable</code>
函数原型	<code>void usart_mute_mode_enable(uint32_t usart_periph);</code>
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}	
<code>usart_periph</code>	外设USARTx
<code>USARTx</code>	x = 0, 1, 2, 5
<code>UARTx</code>	x = 3, 4, 6, 7
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* 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 3-1332. 函数 `usart_mute_mode_disable`

函数名称	<code>usart_mute_mode_disable</code>
函数原型	<code>void usart_mute_mode_disable(uint32_t usart_periph);</code>
功能描述	禁能USART静默模式
先决条件	-
被调用函数	-
输入参数{in}	
<code>usart_periph</code>	外设USARTx
<code>USARTx</code>	x = 0, 1, 2, 5
<code>UARTx</code>	x = 3, 4, 6, 7
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 3-1333. 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
USARTx	x = 0, 1, 2, 5
UARTx	x = 3, 4, 6, 7
输入参数{in}	
wmethod	两种方法用于进入或退出静默模式
USART_WM_IDLE	空闲线唤醒
USART_WM_ADDR	地址匹配唤醒
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* configure USART0 wakeup method in mute mode */
```

```
usart_mute_mode_wakeup_config(USART0, USART_WM_IDLE);
```

函数 usart_lin_mode_enable

函数usart_lin_mode_enable描述见下表：

表 3-1334. 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
USARTx	x = 0, 1, 2, 5
输出参数{out}	

-	-
返回值	
-	-

例如：

```
/* enable USART0 LIN mode */
```

```
usart_lin_mode_enable(USART0);
```

函数 usart_lin_mode_disable

函数usart_lin_mode_disable描述见下表：

表 3-1335. 函数 usart_lin_mode_disable

函数名称	usart_lin_mode_disable
函数原型	void usart_lin_mode_disable(uint32_t usart_periph);
功能描述	禁能USART LIN模式
先决条件	-
被调用函数	-
输入参数{in}	
usart_periph	外设USARTx
USARTx	x = 0, 1, 2, 5
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* disable USART0 LIN mode */
```

```
usart_lin_mode_disable(USART0);
```

函数 usart_lin_break_dection_length_config

函数usart_lin_break_dection_length_config描述见下表：

表 3-1336. 函数 usart_lin_break_dection_length_config

函数名称	usart_lin_break_dection_length_config
函数原型	void usart_lin_break_dection_length_config(uint32_t usart_periph, uint32_t 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

<i>USARTx</i>	<i>x</i> = 0, 1, 2, 5
输入参数{in}	
lblen	LIN模式中断帧长度
<i>USART_LBLEN_10</i> <i>B</i>	断开帧长度为10位
<i>USART_LBLEN_11</i> <i>B</i>	断开帧长度为11位
输出参数{out}	
-	-
返回值	
-	-

例如：

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

```
usart_lin_break_dection_length_config(USART0, USART_LBLEN_10B);
```

函数 usart_halfduplex_enable

函数usart_halfduplex_enable描述见下表：

表 3-1337. 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
<i>USARTx</i>	<i>x</i> = 0, 1, 2, 5
<i>UARTx</i>	<i>x</i> = 3, 4, 6, 7
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* enable USART0 half duplex mode */
```

```
usart_halfduplex_enable(USART0);
```

函数 usart_halfduplex_disable

函数usart_halfduplex_disable描述见下表：

表 3-1338. 函数 `usart_halfduplex_disable`

函数名称	<code>usart_halfduplex_disable</code>
函数原型	<code>void usart_halfduplex_disable(uint32_t usart_periph);</code>
功能描述	禁能USART半双工模式
先决条件	-
被调用函数	-
输入参数{in}	
<code>usart_periph</code>	外设USARTx
<code>USARTx</code>	x = 0, 1, 2, 5
<code>UARTx</code>	x = 3, 4, 6, 7
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* disable USART0 half duplex mode */
```

```
usart_halfduplex_disable(USART0);
```

函数 `usart_clock_enable`

函数`usart_clock_enable`描述见下表:

表 3-1339. 函数 `usart_clock_enable`

函数名称	<code>usart_clock_enable</code>
函数原型	<code>void usart_clock_enable(uint32_t usart_periph);</code>
功能描述	使能USART CK引脚
先决条件	-
被调用函数	-
输入参数{in}	
<code>usart_periph</code>	外设USARTx
<code>USARTx</code>	x = 0, 1, 2, 5
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* enable clock */
```

```
usart_clock_enable(USART0);
```

函数 usart_clock_disable

函数usart_clock_disable描述见下表:

表 3-1340. 函数 usart_clock_disable

函数名称	usart_clock_disable
函数原型	void usart_clock_disable(uint32_t usart_periph);
功能描述	禁能USART CK引脚
先决条件	-
被调用函数	-
输入参数{in}	
usart_periph	外设USARTx
USARTx	x = 0, 1, 2, 5
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* disable clock */
```

```
usart_clock_disable(USART0);
```

函数 usart_synchronous_clock_config

函数usart_synchronous_clock_config描述见下表:

表 3-1341. 函数 usart_synchronous_clock_config

函数名称	usart_synchronous_clock_config
函数原型	void usart_synchronous_clock_config(uint32_t usart_periph, uint32_t clen, uint32_t cph, uint32_t cpl);
功能描述	配置USART同步通讯模式参数
先决条件	-
被调用函数	-
输入参数{in}	
usart_periph	外设USARTx
USARTx	x = 0, 1, 2, 5
输入参数{in}	
clen	CK信号长度
USART_CLEN_NO NE	8位数据帧中有7个CK脉冲, 9位数据帧中有8个CK脉冲
USART_CLEN_EN	8位数据帧中有8个CK脉冲, 9位数据帧中有9个CK脉冲
输入参数{in}	
cph	时钟相位

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 3-1342. 函数 usart_guard_time_config

函数名称	usart_guard_time_config
函数原型	void usart_guard_time_config(uint32_t usart_periph,uint32_t guat);
功能描述	在USART智能卡模式下配置保护时间值
先决条件	-
被调用函数	-
输入参数{in}	
usart_periph	外设USARTx
USARTx	x = 0, 1, 2, 5
输入参数{in}	
guat	保护时间值（0x00-0x000000FF）
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* configure USART0 guard time value in smartcard mode */
```

```
usart_guard_time_config(USART0, 0x0000 0055);
```

函数 usart_smartcard_mode_enable

函数usart_smartcard_mode_enable描述见下表：

表 3-1343. 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, 5
输出参数{out}	
-	-
返回值	
-	-

例如：

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

函数 usart_smartcard_mode_disable

函数usart_smartcard_mode_disable描述见下表：

表 3-1344. 函数 usart_smartcard_mode_disable

函数名称	usart_smartcard_mode_disable
函数原型	void usart_smartcard_mode_disable(uint32_t usart_periph);
功能描述	禁能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, 5
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* disable USART0 smartcard mode */
usart_smartcard_mode_disable(USART0);
```

函数 usart_smartcard_mode_nack_enable

函数usart_smartcard_mode_nack_enable描述见下表：

表 3-1345. 函数 `usart_smartcard_mode_nack_enable`

函数名称	<code>usart_smartcard_mode_nack_enable</code>
函数原型	<code>void usart_smartcard_mode_nack_enable(uint32_t usart_periph);</code>
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}	
<code>usart_periph</code>	外设USARTx
<code>USARTx</code>	<code>x = 0, 1, 2, 5</code>
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 3-1346. 函数 `usart_smartcard_mode_nack_disable`

函数名称	<code>usart_smartcard_mode_nack_disable</code>
函数原型	<code>void usart_smartcard_mode_nack_disable(uint32_t usart_periph);</code>
功能描述	在USART智能卡模式下禁能NACK
先决条件	-
被调用函数	-
输入参数{in}	
<code>usart_periph</code>	外设USARTx
<code>USARTx</code>	<code>x = 0, 1, 2, 5</code>
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_mode_early_nack_enable`

函数`usart_smartcard_mode_early_nack_enable`描述见下表：

表 3-1347. 函数 usart_smartcard_mode_early_nack_enable

函数名称	usart_smartcard_mode_early_nack_enable
函数原型	void usart_smartcard_mode_early_nack_enable(uint32_t usart_periph);
功能描述	使能USART智能卡模式提前NACK
先决条件	-
被调用函数	-
输入参数{in}	
usart_periph	外设USARTx
USARTx	x = 0, 1, 2, 5
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* enable USART0 early NACK in smartcard mode */
usart_smartcard_mode_early_nack_enable(USART0);
```

函数 usart_smartcard_mode_early_nack_disable

函数usart_smartcard_mode_early_nack_disable描述见下表：

表 3-1348. 函数 usart_smartcard_mode_early_nack_disable

函数名称	usart_smartcard_mode_early_nack_disable
函数原型	void usart_smartcard_mode_early_nack_disable(uint32_t usart_periph);
功能描述	禁能USART智能卡模式提前NACK
先决条件	-
被调用函数	-
输入参数{in}	
usart_periph	外设USARTx
USARTx	x = 0, 1, 2, 5
输出参数{out}	
-	-
返回值	
-	-

例如：

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

函数 usart_smartcard_autoretry_config

函数usart_smartcard_autoretry_config描述见下表：

表 3-1349. 函数 `usart_smartcard_autoretry_config`

函数名称	<code>usart_smartcard_autoretry_config</code>
函数原型	<code>void usart_smartcard_autoretry_config(uint32_t usart_periph, uint32_t scrtnum);</code>
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}	
<code>usart_periph</code>	外设USARTx
<code>USARTx</code>	x = 0, 1, 2, 5
输入参数{in}	
<code>scrtnum</code>	智能卡自动重试次数（0x00-0x00000007）
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* configure smartcard auto-retry number */
```

```
usart_smartcard_autoretry_config(USART0, 0x00000007);
```

函数 `usart_block_length_config`

函数`usart_block_length_config`描述见下表：

表 3-1350. 函数 `usart_block_length_config`

函数名称	<code>usart_block_length_config</code>
函数原型	<code>void usart_block_length_config(uint32_t usart_periph, uint32_t bl);</code>
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}	
<code>usart_periph</code>	外设USARTx
<code>USARTx</code>	x = 0, 1, 2, 5
输入参数{in}	
<code>bl</code>	块长度（0x00-0x000000FF）
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* configure block length in Smartcard T=1 reception */
```

```
usart_block_length_config(USART0, 0x000000FF);
```

函数 usart_irda_mode_enable

函数usart_irda_mode_enable描述见下表:

表 3-1351. 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, 5
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 3-1352. 函数 usart_irda_mode_disable

函数名称	usart_irda_mode_disable
函数原型	void usart_irda_mode_disable(uint32_t usart_periph);
功能描述	禁能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, 5
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 3-1353. 函数 usart_prescaler_config

函数名称	usart_prescaler_config
函数原型	void usart_prescaler_config(uint32_t usart_periph, uint32_t psc);
功能描述	在USART IrDA低功耗模式下配置外设时钟分频系数
先决条件	-
被调用函数	-
输入参数{in}	
usart_periph	外设USARTx
USARTx	x = 0, 1, 2, 5
输入参数{in}	
psc	时钟分频系数（0x00-0xFF）
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* configure the USART0 peripheral clock prescaler in USART IrDA low-power mode */
```

```
usart_prescaler_config(USART0, 0x00);
```

函数 usart_irda_lowpower_config

函数usart_irda_lowpower_config描述见下表：

表 3-1354. 函数 usart_irda_lowpower_config

函数名称	usart_irda_lowpower_config
函数原型	void usart_irda_lowpower_config(uint32_t usart_periph, uint32_t irlp);
功能描述	配置USART IrDA低功耗模式
先决条件	-
被调用函数	-
输入参数{in}	
usart_periph	外设USARTx
USARTx	x = 0, 1, 2, 5
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 3-1355. 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, 5
UARTx	x = 3, 4, 6, 7
输入参数{in}	
rtsconfig	使能/禁能RTS
USART_RTS_ENA BLE	使能RTS
USART_RTS_DISA BLE	禁能RTS
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* configure USART0 hardware flow control RTS */
```

```
usart_hardware_flow_rts_config(USART0, USART_RTS_ENABLE);
```

函数 usart_hardware_flow_cts_config

函数usart_hardware_flow_cts_config描述见下表：

表 3-1356. 函数 `usart hardware_flow_cts_config`

函数名称	<code>usart hardware_flow_cts_config</code>
函数原型	<code>void usart hardware_flow_cts_config(uint32_t usart_periph, uint32_t ctsconfig);</code>
功能描述	配置USART CTS硬件控制流
先决条件	-
被调用函数	-
输入参数{in}	
<code>usart_periph</code>	外设USARTx
<code>USARTx</code>	x = 0, 1, 2, 5
<code>UARTx</code>	x = 3, 4, 6, 7
输入参数{in}	
<code>ctsconfig</code>	使能/禁能CTS
<code>USART_CTS_ENABLE</code>	使能CTS
<code>USART_CTS_DISABLE</code>	禁能CTS
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* configure USART0 hardware flow control CTS */
```

```
usart hardware_flow_cts_config(USART0, USART_CTS_ENABLE);
```

函数 `usart hardware_flow_coherence_config`

函数`usart hardware_flow_coherence_config`描述见下表:

表 3-1357. 函数 `usart hardware_flow_coherence_config`

函数名称	<code>usart hardware_flow_coherence_config</code>
函数原型	<code>void usart hardware_flow_coherence_config(uint32_t usart_periph, uint32_t hcm);</code>
功能描述	配置硬件流控兼容模式
先决条件	-
被调用函数	-
输入参数{in}	
<code>usart_periph</code>	外设USARTx
<code>USARTx</code>	x = 0, 1, 2, 5
<code>UARTx</code>	x = 3, 4, 6, 7
输入参数{in}	
<code>hcm</code>	硬件流控制兼容模式
<code>USART_HCM_NON</code>	nRTS信号与USART_STAT0寄存器中RBNE位相同

<i>E</i>	
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 hardware flow control coherence mode */
```

```
usart_hardware_flow_coherence_config(USART0, USART_HCM_NONE);
```

函数 usart_rs485_driver_enable

函数usart_rs485_driver_enable描述见下表：

表 3-1358. 函数 usart_rs485_driver_enable

函数名称	usart_rs485_driver_enable
函数原型	void usart_rs485_driver_enable(uint32_t usart_periph);
功能描述	使能USART rs485驱动
先决条件	-
被调用函数	-
输入参数{in}	
usart_periph	外设USARTx
USARTx	x = 0, 1, 2, 5
UARTx	x = 3, 4, 6, 7
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* enable USART0 RS485 driver */
```

```
usart_rs485_driver_enable(USART0);
```

函数 usart_rs485_driver_disable

函数usart_rs485_driver_disable描述见下表：

表 3-1359. 函数 usart_rs485_driver_disable

函数名称	usart_rs485_driver_disable
函数原型	void usart_rs485_driver_disable(uint32_t usart_periph);
功能描述	禁能USART rs485驱动
先决条件	-

被调用函数	-
输入参数{in}	
usart_periph	外设USARTx
<i>USARTx</i>	x = 0, 1, 2, 5
<i>UARTx</i>	x = 3, 4, 6, 7
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* disable USART0 RS485 driver */
```

```
usart_rs485_driver_disable(USART0);
```

函数 usart_driver_assertime_config

函数usart_driver_assertime_config描述见下表：

表 3-1360. 函数 usart_driver_assertime_config

函数名称	usart_driver_assertime_config
函数原型	void usart_driver_assertime_config(uint32_t usart_periph, uint32_t deatime);
功能描述	配置USART驱动使能置位时间
先决条件	-
被调用函数	-
输入参数{in}	
usart_periph	外设USARTx
<i>USARTx</i>	x = 0, 1, 2, 5
<i>UARTx</i>	x = 3, 4, 6, 7
输入参数{in}	
deatime	驱动使能置位时间（0x00000000-0x0000001F）
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* set USART0 driver assertime */
```

```
usart_driver_assertime_config(USART0, 0x0000001F);
```

函数 usart_driver_deassertime_config

函数usart_driver_deassertime_config描述见下表：

表 3-1361. 函数 usart_driver_deasserttime_config

函数名称	usart_driver_deasserttime_config
函数原型	void usart_driver_deasserttime_config(uint32_t usart_periph, uint32_t dedtime);
功能描述	配置USART驱动使能置低时间
先决条件	-
被调用函数	-
输入参数{in}	
usart_periph	外设USARTx
USARTx	x = 0, 1, 2, 5
UARTx	x = 3, 4, 6, 7
输入参数{in}	
dedtime	驱动使能置低时间（0x00000000-0x0000001F）
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* set USART0 driver deasserttime */
```

```
usart_driver_deasserttime_config(USART0, 0x0000001F);
```

函数 usart_depolarity_config

函数usart_depolarity_config描述见下表：

表 3-1362. 函数 usart_depolarity_config

函数名称	usart_depolarity_config
函数原型	void usart_depolarity_config(uint32_t usart_periph, uint32_t dep);
功能描述	配置USART驱动使能极性模式
先决条件	-
被调用函数	-
输入参数{in}	
usart_periph	外设USARTx
USARTx	x = 0, 1, 2, 5
UARTx	x = 3, 4, 6, 7
输入参数{in}	
dep	驱动使能的极性选择模式
USART_DEP_HIGH	DE信号高有效
USART_DEP_LOW	DE信号低有效
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* configure driver enable polarity mode */
```

```
usart_driver_depolarity_config(USART0, USART_DEP_HIGH);
```

函数 usart_dma_receive_config

函数usart_dma_receive_config描述见下表:

表 3-1363. 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
先决条件	-
被调用函数	-
输入参数{in}	
usart_periph	外设USARTx
USARTx	x = 0, 1, 2, 5
UARTx	x = 3, 4, 6, 7
输入参数{in}	
dmacmd	USART DMA模式
USART_RECEIVE_DMA_ENABLE	USART DMA接收使能
USART_RECEIVE_DMA_DISABLE	USART DMA接收禁能
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* USART0 DMA enable for reception */
```

```
usart_dma_receive_config(USART0, USART_RECEIVE_DMA_ENABLE);
```

函数 usart_dma_transmit_config

函数usart_dma_transmit_config描述见下表:

表 3-1364. 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
先决条件	-
被调用函数	-

输入参数{in}	
usart_periph	外设USARTx
<i>USARTx</i>	x = 0, 1, 2, 5
<i>UARTx</i>	x = 3, 4, 6, 7
输入参数{in}	
dmacmd	USART DMA模式
<i>USART_TRANSMIT_DMA_ENABLE</i>	USART DMA发送使能
<i>USART_TRANSMIT_DMA_DISABLE</i>	USART DMA发送禁能
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* USART0 DMA disable for transmission */
```

```
usart_dma_transmit_config(USART0, USART_TRANSMIT_DMA_DISABLE);
```

函数 **usart_reception_error_dma_disable**

函数usart_reception_error_dma_disable描述见下表：

表 3-1365. 函数 usart_reception_error_dma_disable

函数名称	usart_reception_error_dma_disable
函数原型	void usart_reception_error_dma_disable(uint32_t usart_periph);
功能描述	USART接收错误时禁能DMA
先决条件	-
被调用函数	-
输入参数{in}	
usart_periph	外设USARTx
<i>USARTx</i>	x = 0, 1, 2, 5
<i>UARTx</i>	x = 3, 4, 6, 7
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* disable DMA on reception error */
```

```
usart_reception_error_dma_disable(USART0);
```


函数 usart_reception_error_dma_enable

函数usart_reception_error_dma_enable描述见下表：

表 3-1366. 函数 usart_reception_error_dma_enable

函数名称	usart_reception_error_dma_enable
函数原型	void usart_reception_error_dma_enable(uint32_t usart_periph);
功能描述	USART接收错误时使能DMA
先决条件	-
被调用函数	-
输入参数{in}	
usart_periph	外设USARTx
USARTx	x = 0, 1, 2, 5
UARTx	x = 3, 4, 6, 7
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* enable DMA on reception error */
usart_reception_error_dma_enable(USART0);
```

函数 usart_wakeup_enable

函数usart_wakeup_enable描述见下表：

表 3-1367. 函数 usart_wakeup_enable

函数名称	usart_wakeup_enable
函数原型	void usart_wakeup_enable(uint32_t usart_periph);
功能描述	使能USART唤醒
先决条件	-
被调用函数	-
输入参数{in}	
usart_periph	外设USARTx
USARTx	x = 0, 1, 2, 5
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* enable USART0 wake up */
```

```
usart_wakeup_enable(USART0);
```

函数 usart_wakeup_disable

函数usart_wakeup_disable描述见下表：

表 3-1368. 函数 usart_wakeup_disable

函数名称	usart_wakeup_disable
函数原型	void usart_wakeup_disable(uint32_t usart_periph);
功能描述	禁能USART唤醒
先决条件	-
被调用函数	-
输入参数{in}	
usart_periph	外设USARTx
USARTx	x = 0, 1, 2, 5
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* disable USART0 wake up */
```

```
usart_wakeup_disable(USART0);
```

函数 usart_wakeup_mode_config

函数usart_wakeup_mode_config描述见下表：

表 3-1369. 函数 usart_wakeup_mode_config

函数名称	usart_wakeup_mode_config
函数原型	void usart_wakeup_mode_config(uint32_t usart_periph, uint32_t wum);
功能描述	配置USART唤醒模式
先决条件	-
被调用函数	-
输入参数{in}	
usart_periph	外设USARTx
USARTx	x = 0, 1, 2, 5
输入参数{in}	
wum	唤醒模式
USART_WUM_ADD R	WUF在地址匹配时置位
USART_WUM_STA RTB	WUF在检测到起始位时置位
USART_WUM_RBN	WUF在检测到RBNE时置位

<i>E</i>	
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* configure USART0 wake up mode */
usart_wakeup_mode_config(USART0, USART_WUM_ADDR);
```

函数 usart_fifo_enable

函数usart_fifo_enable描述见下表：

表 3-1370. 函数 usart_fifo_enable

函数名称	usart_fifo_enable
函数原型	void usart_fifo_enable(uint32_t usart_periph);
功能描述	使能FIFO
先决条件	-
被调用函数	-
输入参数{in}	
usart_periph	外设USARTx
USARTx	x = 0, 1, 2, 5
UARTx	x = 3, 4, 6, 7
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* enable FIFO */
usart_fifo_enable(USART0);
```

函数 usart_fifo_disable

函数usart_fifo_disable描述见下表：

表 3-1371. 函数 usart_fifo_disable

函数名称	usart_fifo_disable
函数原型	void usart_fifo_disable (uint32_t usart_periph);
功能描述	禁能FIFO
先决条件	-
被调用函数	-

输入参数{in}	
usart_periph	外设USARTx
<i>USARTx</i>	x = 0, 1, 2, 5
<i>UARTx</i>	x = 3, 4, 6, 7
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* disable FIFO */
```

```
usart_fifo_disable(USART0);
```

函数 usart_transmit_fifo_threshold_config

函数usart_transmit_fifo_threshold_config描述见下表:

表 3-1372. 函数 usart_transmit_fifo_threshold_config

函数名称	usart_transmit_fifo_threshold_config
函数原型	void usart_transmit_fifo_threshold_config(uint32_t usart_periph, uint32_t txthreshold);
功能描述	配置发送FIFO阈值
先决条件	-
被调用函数	-
输入参数{in}	
usart_periph	外设USARTx
<i>USARTx</i>	x = 0, 1, 2, 5
<i>UARTx</i>	x = 3, 4, 6, 7
输入参数{in}	
txthreshold	发送FIFO阈值
<i>USART_TFTCFG_T HRESHOLD_1_8</i>	发送FIFO到达其深度的1/8
<i>USART_TFTCFG_T HRESHOLD_1_4</i>	发送FIFO到达其深度的1/4
<i>USART_TFTCFG_T HRESHOLD_1_2</i>	发送FIFO到达其深度的1/2
<i>USART_TFTCFG_T HRESHOLD_3_4</i>	发送FIFO到达其深度的3/4
<i>USART_TFTCFG_T HRESHOLD_7_8</i>	发送FIFO到达其深度的7/8
<i>USART_TFTCFG_T HRESHOLD_EMPTY</i>	发送FIFO变为空

输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* configure transmit FIFO threshold as empty */
```

```
usart_transmit_fifo_threshold_config (USART0, USART_TFTCFG_THRESHOLD_EMPTY);
```

函数 usart_receive_fifo_threshold_config

函数usart_receive_fifo_threshold_config描述见下表:

表 3-1373. 函数 usart_receive_fifo_threshold_config

函数名称	usart_receive_fifo_threshold_config
函数原型	void usart_receive_fifo_threshold_config(uint32_t usart_periph, uint32_t rxthreshold);
功能描述	配置接收FIFO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, 5
UARTx	x = 3, 4, 6, 7
输入参数{in}	
rxthreshold	接收FIFO阈值
USART_RFTCFG_THRESHOLD_1_8	接收FIFO到达其深度的1/8
USART_RFTCFG_THRESHOLD_1_4	接收FIFO到达其深度的1/4
USART_RFTCFG_THRESHOLD_1_2	接收FIFO到达其深度的1/2
USART_RFTCFG_THRESHOLD_3_4	接收FIFO到达其深度的3/4
USART_RFTCFG_THRESHOLD_7_8	接收FIFO到达其深度的7/8
USART_RFTCFG_THRESHOLD_FULL	接收FIFO变为满
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* configure receiveFIFO threshold as full */
```

```
usart_receive_fifo_threshold_config (USART0, USART_RFTCFG_THRESHOLD_FULL);
```

函数 usart_receive_fifo_counter_number

函数usart_receive_fifo_counter_number描述见下表：

表 3-1374. 函数 usart_receive_fifo_counter_number

函数名称	usart_receive_fifo_counter_number
函数原型	uint8_t usart_receive_fifo_counter_number(uint32_t usart_periph);
功能描述	读取接收FIFO计数器的值
先决条件	-
被调用函数	-
输入参数{in}	
usart_periph	外设USARTx
USARTx	x = 0, 1, 2, 5
UARTx	x = 3, 4, 6, 7
输出参数{out}	
-	-
返回值	
uint8_t	接收FIFO计数器的值

例如：

```
/* read receive FIFO counter number */
```

```
uint8_t temp;
```

```
temp = usart_receive_fifo_counter_number(USART0);
```

函数 usart_flag_get

函数usart_flag_get描述见下表：

表 3-1375. 函数 usart_flag_get

函数名称	usart_flag_get
函数原型	FlagStatus usart_flag_get(uint32_t usart_periph, usart_flag_enum flag);
功能描述	获取USART STAT/CHC/FCS寄存器标志位
先决条件	-
被调用函数	-
输入参数{in}	
usart_periph	外设USARTx
USARTx	x = 0, 1, 2, 5
UARTx	x = 3, 4, 6, 7

输入参数{in}	
flag	USART标志位, 参考 表3-1298. 枚举类型usart_flag_enum 只能选择一个参数
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 3-1376. 函数 usart_flag_clear

函数名称	usart_flag_clear
函数原型	void usart_flag_clear(uint32_t usart_periph, usart_flag_enum flag);
功能描述	清除USART状态寄存器标志位
先决条件	-
被调用函数	-
输入参数{in}	
usart_periph	外设USARTx
USARTx	x = 0, 1, 2, 5
UARTx	x = 3, 4, 6, 7
输入参数{in}	
flag	USART标志位, 参考 表3-1298. 枚举类型usart_flag_enum 只能选择一个参数
USART_FLAG_PERR	校验错误标志
USART_FLAG_FER	帧错误标志
USART_FLAG_NERR	噪声错误标志
USART_FLAG_ORERR	溢出错误标志
USART_FLAG_IDLE	空闲线检测标志
USART_FLAG_TC	发送完成标志
USART_FLAG_LBD	发送完成标志

USART_FLAG_CTS F	CTS变化标志
USART_FLAG_RT	接收超时标志
USART_FLAG_EB	块结束标志
USART_FLAG_AM 0	ADDR0匹配标志
USART_FLAG_AM 1	ADDR1匹配标志
USART_FLAG_WU	从深度睡眠模式唤醒标志
USART_FLAG_EPE RR	校验错误超前检测标志
USART_FLAG_TFE	发送FIFO空标志
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 3-1377. 函数 usart_interrupt_enable

函数名称	usart_interrupt_enable
函数原型	void usart_interrupt_enable(uint32_t usart_periph, usart_interrupt_enum interrupt);
功能描述	使能USART中断
先决条件	-
被调用函数	-
输入参数{in}	
usart_periph	外设USARTx
USARTx	x = 0, 1, 2, 5
UARTx	x = 3, 4, 6, 7
输入参数{in}	
interrupt	USART中断USART标志位，参考 表3-1300. 枚举类型usart_interrupt_enum 只能选择一个参数
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 3-1378. 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	禁能USART中断
先决条件	-
被调用函数	-
输入参数{in}	
usart_periph	外设USARTx
USARTx	x = 0, 1, 2, 5
UARTx	x = 3, 4, 6, 7
输入参数{in}	
interrupt	USART中断USART标志位，参考 表3-1300. 枚举类型usart_interrupt_enum 只能选择一个参数
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 3-1379. 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
<i>USARTx</i>	x = 0, 1, 2, 5
<i>UARTx</i>	x = 3, 4, 6, 7
输入参数{in}	
int_flag	USART中断标志, 参考 表3-1299. 枚举类型usart_interrupt_flag_enum 只能选择一个参数
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 3-1380. 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
<i>USARTx</i>	x = 0, 1, 2, 5
<i>UARTx</i>	x = 3, 4, 6, 7
输入参数{in}	
int_flag	USART中断标志, 参考 表3-1299. 枚举类型usart_interrupt_flag_enum 只能选择一个参数
<i>USART_INT_FLAG_PERR</i>	校验错误中断标志
<i>USART_INT_FLAG_ERR_FERR</i>	帧错误中断标志
<i>USART_INT_FLAG_ERR_NERR</i>	噪声错误中断标志
<i>USART_INT_FLAG_RBNE_ORERR</i>	读数据缓冲区非空中断和溢出错误中断标志

USART_INT_FLAG_ERR_ORERR	过载错误中断标志
USART_INT_FLAG_IDLE	IDLE线检测中断标志
USART_INT_FLAG_TC	发送完成中断标志
USART_INT_FLAG_LBD	LIN断开检测中断标志
USART_INT_FLAG_CTS	CTS变化中断标志
USART_INT_FLAG_RT	接收超时事件中断标志
USART_INT_FLAG_EB	块结束事件中断标志
USART_INT_FLAG_AM0	ADDR0匹配中断标志
USART_INT_FLAG_AM1	ADDR1匹配中断标志
USART_INT_FLAG_WU	从深度睡眠模式唤醒中断标志
USART_INT_FLAG_RFT	接受FIFO阈值到达中断标志
USART_INT_FLAG_RFF	接受FIFO满中断标志
USART_INT_FLAG_TFE	发送FIFO空中断标志
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* clear the USART0 interrupt flag */
```

```
usart_interrupt_flag_clear(USART0, USART_INT_FLAG_TC);
```

3.37. VREF

VREF用于为ADC / DAC提供基准电压，或由连接到VREFP引脚的片外电路使用。章节[3.37.1](#)描述了VREF的寄存器列表，章节[3.37.2](#)对VREF库函数进行说明。

3.37.1. 外设寄存器说明

VREF寄存器列表如下表所示：

表 3-1381. VREF 寄存器

寄存器名称	寄存器描述
VREF_CS	控制和状态寄存器
VREF_CALIB	校准寄存器

3.37.2. 外设库函数说明

VREF库函数列表如下表所示：

表 3-1382. VREF 库函数

库函数名称	库函数说明
vref_deinit	复位VREF
vref_enable	使能VREF
vref_disable	失能VREF
vref_high_impedance_mode_enable	使能VREF高阻模式
vref_high_impedance_mode_disable	失能VREF高阻模式
vref_status_get	获取VREF状态
vref_voltage_select	选择VREF参考电压值
vref_calib_value_set	设置VREF校准值
vref_calib_value_get	获取VREF校准值

函数 vref_deinit

函数vref_deinit描述见下表：

表 3-1383. 函数 vref_deinit

函数名称	vref_deinit
函数原型	void vref_deinit(void);
功能描述	复位VREF
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 the VREF */
```

vref_deinit();

函数 vref_enable

函数vref_enable描述见下表:

表 3-1384. 函数 vref_enable

函数名称	vref_enable
函数原型	void vref_enable(void);
功能描述	使能VREF
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 VREF */
```

```
vref_enable();
```

函数 vref_disable

函数vref_disable描述见下表:

表 3-1385. 函数 vref_disable

函数名称	vref_disable
函数原型	void vref_disable(void);
功能描述	失能VREF
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 VREF */
```

```
vref_disable();
```

函数 vref_high_impedance_mode_enable

函数vref_high_impedance_mode_enable描述见下表：

表 3-1386. 函数 vref_high_impedance_mode_enable

函数名称	vref_high_impedance_mode_enable
函数原型	void vref_high_impedance_mode_enable(void);
功能描述	使能VREF高阻模式
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 VREF high impedance mode */  
vref_high_impedance_mode_enable();
```

函数 vref_high_impedance_mode_disable

函数vref_high_impedance_mode_disable描述见下表：

表 3-1387. 函数 vref_high_impedance_mode_disable

函数名称	vref_high_impedance_mode_disable
函数原型	void vref_high_impedance_mode_disable(void);
功能描述	失能VREF高阻模式
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 VREF high impedance mode */  
vref_high_impedance_mode_disable();
```

函数 vref_status_get

函数vref_status_get描述见下表:

表 3-1388. 函数 vref_status_get

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

例如:

```
/* get the status of VREF */
```

```
FlagStatus status;
```

```
status = vref_status_get();
```

函数 vref_voltage_select

函数vref_voltage_select描述见下表:

表 3-1389. 函数 vref_voltage_select

函数名称	vref_voltage_select
函数原型	void vref_voltage_select(uint32_t vref_voltage);
功能描述	选择VREF参考电压值
先决条件	-
被调用函数	-
输入参数{in}	
vref_voltage	VREF参考电压选择
VREF_VOLTAGE_S EL_2_5V	VREF参考电压选择为2.5 V
VREF_VOLTAGE_S EL_2_048V	VREF参考电压选择为2.048 V
VREF_VOLTAGE_S EL_1_8V	VREF参考电压选择为1.8 V
VREF_VOLTAGE_S EL_1_5V	VREF参考电压选择为1.5 V
输出参数{out}	

-	-
返回值	
-	-

例如：

```
/* select 2.5V as the VREF voltage reference */
```

```
vref_voltage_select(VREF_VOLTAGE_SEL_2_5V);
```

函数 vref_calib_value_set

函数vref_calib_value_set描述见下表：

表 3-1390. 函数 vref_calib_value_set

函数名称	vref_calib_value_set
函数原型	void vref_calib_value_set(uint8_t value);
功能描述	设置VREF校准值
先决条件	-
被调用函数	-
输入参数{in}	
value	校准值(0x00 - 0x3F)
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* set the calibration value of VREF */
```

```
vref_calib_value_set(0x0A);
```

函数 vref_calib_value_get

函数vref_calib_value_get描述见下表：

表 3-1391. 函数 vref_calib_value_get

函数名称	vref_calib_value_get
函数原型	uint8_t vref_calib_value_get(void);
功能描述	获取VREF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	校准值 (0x00-0x3F)

例如:

```
/* get the calibration value of VREF */
uint8_t cal_val;
cal_val = vref_calib_value_get();
```

3.38. 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[3.38.1](#)描述了WWDGT的寄存器列表，章节[3.38.2](#)对WWDGT库函数进行说明。

3.38.1. 外设寄存器说明

WWDGT寄存器列表如下表所示:

表 3-1392. 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

3.38.2. 外设库函数说明

WWDGT库函数列表如下表所示:

表 3-1393. WWDGT 库函数

库函数名称	库函数说明
wwdgt_deinit	将WWDGT寄存器重设为缺省值
wwdgt_enable	使能WWDGT
wwdgt_counter_update	设置WWDGT计数器更新值
wwdgt_config	设置WWDGT计数器值、窗口值和预分频值
wwdgt_interrupt_enable	使能WWDGT提前唤醒中断
wwdgt_flag_get	检查WWDGT提前唤醒中断标志位是否置位
wwdgt_flag_clear	清除WWDGT提前唤醒中断标志位状态

函数 wwdgt_deinit

函数wwdgt_deinit描述见下表:

表 3-1394. 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 WWdGT configuration */
```

```
wwdgt_deinit();
```

函数 wwdgt_enable

函数wwdgt_enable描述见下表：

表 3-1395. 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 WWdGT counter */
```

```
wwdgt_enable();
```

函数 wwdgt_counter_update

函数wwdgt_counter_update描述见下表：

表 3-1396. 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	计数器值，数值范围为0x0000 - 0x007F
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 3-1397. 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	定时器计数值，数值范围0x0000 - 0x007F
输入参数{in}	
window	窗口值，数值范围0x0000 - 0x007F
输入参数{in}	
prescaler	WWDGT预分频值
WWDGT_CFG_PSC_DIV1	WWDGT计数器时钟为（PCLK3/4096）/1
WWDGT_CFG_PSC_DIV2	WWDGT计数器时钟为（PCLK3/4096）/2
WWDGT_CFG_PSC_DIV4	WWDGT计数器时钟为（PCLK3/4096）/4
WWDGT_CFG_PSC_DIV8	WWDGT计数器时钟为（PCLK3/4096）/8
输出参数{out}	
-	-
Return value	
-	-

例如:

```
/* configure WWDGT counter value to 0x7F, window value to 0x50, prescaler divider value to 8 */
```

```
wwdgt_config(127, 80, WWDGT_CFG_PSC_DIV8);
```

函数 wwdgt_interrupt_enable

函数wwdgt_interrupt_enable描述见下表:

表 3-1398. 函数 wwdgt_interrupt_enable

函数名称	wwdgt_interrupt_enable
函数原型	void wwdgt_interrupt_enable(void);
功能描述	使能WWDGT提前唤醒中断
先决条件	-
被调用函数	-
输入参数{in}	
-	-
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* enable early wakeup interrupt of WWDGT */
```

```
wwdgt_interrupt_enable();
```

函数 wwdgt_flag_get

函数wwdgt_flag_get描述见下表:

表 3-1399. 函数 wwdgt_flag_get

函数名称	wwdgt_flag_get
函数原型	FlagStatus wwdgt_flag_get(void);
功能描述	检查WWDGT提前唤醒中断标志位是否置位
先决条件	-
被调用函数	-
输入参数{in}	
-	-
输出参数{out}	
-	-
返回值	
FlagStatus	SET or RESET

例如：

```
/* test if the counter value update has reached the 0x40 */
```

```
FlagStatus status;
```

```
status = wwdgt_flag_get();
```

函数 `wwdgt_flag_clear`

函数 `wwdgt_flag_clear` 描述见下表：

表 3-1400. 函数 `wwdgt_flag_clear`

函数名称	<code>wwdgt_flag_clear</code>
函数原型	<code>void wwdgt_flag_clear(void);</code>
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();
```

3.39. ESC_INTC

器件提供了可编程的多层中断结构,通过中断控制器(INTC)进行控制。章节[3.39.1](#)描述了INTC的寄存器列表,章节[3.39.2](#)对INTC库函数进行说明。

3.39.1. 外设寄存器描述

ESC_INTC寄存器列表如下表所示:

表 3-1401. ESC_INTC 寄存器

寄存器名称	寄存器描述
INTC_CTL	INTC控制寄存器
INTC_FLAG	INTC标志寄存器
INTC_EN	INTC使能寄存器

3.39.2. 外设库函数说明

ESC_INTC库函数列表如下表所示:

表 3-1402. ESC_INTC 库函数

库函数名称	库函数描述
intc_deas_config	配置INTC中断无效间隔
intc_deasc_config	配置INTC中断无效间隔清除
intc_irq_config	配置IRQ引脚输出使能、极性、时钟和模式
intc_interrupt_enable	使能INTC中断
intc_interrupt_disable	失能INTC中断
intc_interrupt_flag_get	获取INTC中断标志
intc_interrupt_flag_clear	清除INTC中断标志
intc_deasstat_get	获取中断无效间隔状态
intc_irqstat_get	获取内部IRQ中断线状态

枚举类型 intc_enable_enum

表 3-1403.枚举 intc_enable_enum

成员名	描述
SW_INT_ENABLE	软件中断使能
READY_INT_ENABLE	设备就绪中断使能
PHYB_INT_ENABLE	以太网 PHY B 中断使能
PHYA_INT_ENABLE	以太网 PHY A 中断使能
TIM_INT_ENABLE	定时器中断使能
PME_INT_ENABLE	PME 中断使能
AHB2OPB_INT_ENABLE	AHB2OP 桥连接中断使能

成员名	描述
ECAT_INT_ENABLE	etherCAT 中断使能

枚举类型 `intc_disable_enum`

表 3-1404.枚举 `intc_disable_enum`

成员名	描述
SW_INT_DISABLE	软件中断失能
READY_INT_DISABLE	设备就绪中断失能
PHYB_INT_DISABLE	以太网 PHY B 中断失能
PHYA_INT_DISABLE	以太网 PHY A 中断失能
TIM_INT_DISABLE	定时器中断失能
PME_INT_DISABLE	PME 中断失能
AHB2OPB_INT_DISABLE	AHB2OP 桥连接中断失能
ECAT_INT_DISABLE	etherCAT 中断失能

枚举类型 `intc_get_flag_enum`

表 3-1405.枚举 `intc_get_flag_enum`

成员名	描述
SW_INT_FLAG_GET	获取软件中断标志
READY_INT_FLAG_GET	获取设备就绪中断标志
PHYB_INT_FLAG_GET	获取以太网 PHY B 中断标志
PHYA_INT_FLAG_GET	获取以太网 PHY A 中断标志
TIM_INT_FLAG_GET	获取定时器中断标志
PME_INT_FLAG_GET	获取 PME 中断标志
AHB2OPB_INT_FLAG_GET	获取 AHB2OPB 桥连接中断标志
ECAT_INT_FLAG_GET	获取 etherCAT 中断标志

枚举类型 `intc_clear_flag_enum`

表 3-1406.枚举 `intc_clear_flag_enum`

成员名	描述
SW_INT_FLAG_CLEAR	清除软件中断标志
READY_INT_FLAG_CLEAR	清除设备就绪中断标志
TIM_INT_FLAG_CLEAR	清除定时器中断标志
PME_INT_FLAG_CLEAR	清除 PME 中断标志

函数 `intc_deas_config`

函数 `intc_deas_config` 描述见下表：

表 3-1407. 函数 `intc_deas_config`

函数名称	<code>intc_deas_config</code>
函数原型	<code>void intc_deas_config(uint8_t deas);</code>
功能描述	INTC中断无效间隔配置
先决条件	-
被调用函数	<code>read_intc_register</code> <code>write_intc_register</code>
输入参数{in}	
<code>uint8_t</code>	配置中断无效间隔，以10us为单位
<code>deas</code>	0x00 – 0xFF
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* configure INTC interrupt de-assertion interval */
```

```
intc_deas_config(0x10);
```

函数 `intc_deasc_config`

函数 `intc_deasc_config` 描述见下表：

表 3-1408. 函数 `intc_deasc_config`

函数名称	<code>intc_deasc_config</code>
函数原型	<code>void intc_deasc_config(uint32_t deasc);</code>
功能描述	清除INTC中断无效间隔配置
先决条件	-
被调用函数	<code>fw_debug_report_err</code> <code>read_intc_register</code> <code>write_intc_register</code>
输入参数{in}	
<code>deasc</code>	清除中断无效间隔计数器
<code>INTC_DEAS</code>	无影响
<code>INTC_DEAS_CLEA R</code>	清除中断无效间隔
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* clear INTC interrupt de-assertion interval */
```

```
intc_deasc_config(INTC_DEAS_CLEAR);
```

函数 intc_irq_config

函数intc_irq_config描述见下表：

表 3-1409. 函数 intc_irq_config

函数名称	intc_irq_config
函数原型	void intc_irq_config(uint32_t irqen, uint32_t irqpol, uint32_t irqckout, uint32_t irqmode);
功能描述	IRQ 引脚输出使能、极性、时钟和模式配置
先决条件	-
被调用函数	fw_debug_report_err read_intc_register write_intc_register
输入参数{in}	
irqen	IRQ 引脚输出使能
IRQ_DISABLE	失能 IRQ 引脚输出
IRQ_ENABLE	使能 IRQ 引脚输出
输入参数{in}	
irqpol	IRQ 引脚输出极性
IRQ_POL_LOW	IRQ 引脚激活电平为低
IRQ_POL_HIGH	IRQ 引脚激活电平为高
输入参数{in}	
irqckout	IRQ 时钟输出
IRQ_NO_CKOUT	无时钟输出
IRQ_CKOUT	IRQ 引脚输出晶振时钟
输入参数{in}	
irqmode	IRQ 引脚输出模式
IRQ_OPEN_DRAIN_MODE	输出开漏模式
IRQ_PUSH_PULL_MODE	输出推挽模式
Output parameter{out}	
-	-
Return value	
-	-

例如：

```
/* config IRQ pin out enable, polarity, clock and mode */
```

```
intc_irq_config(IRQ_ENABLE, IRQ_POL_LOW, IRQ_CKOUT, IRQ_OPEN_DRAIN_MODE);
```

函数 intc_interrupt_enable

函数intc_interrupt_enable描述见下表：

表 3-1410. 函数 intc_interrupt_enable

函数名称	intc_interrupt_enable
函数原型	void intc_interrupt_enable(intc_enable_enum int_type);
功能描述	使能INTC中断
先决条件	-
被调用函数	fw_debug_report_err read_intc_register write_intc_register
输入参数{in}	
intc_enable_enum	INTC中断使能参考 表3-1403.枚举intc_enable_enum
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* enable software interrupt */
intc_interrupt_enable(SW_INT_ENABLE);
```

函数 intc_interrupt_disable

函数intc_interrupt_disable描述见下表：

表 3-1411. 函数 intc_interrupt_disable

函数名称	intc_interrupt_disable
函数原型	void intc_interrupt_disable(intc_disable_enum int_type);
功能描述	失能INTC中断
先决条件	-
被调用函数	fw_debug_report_err read_intc_register write_intc_register
输入参数{in}	
intc_disable_enum	INTC中断失能参考 表3-1404.枚举intc_disable_enum
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* disable software interrupt */
intc_interrupt_disble(SW_INT_DISABLE);
```

函数 intc_interrupt_flag_get

函数intc_interrupt_flag_get描述见下表：

表 3-1412. 函数 intc_interrupt_flag_get

函数名称	intc_interrupt_flag_get
函数原型	FlagStatus intc_interrupt_flag_get(intc_get_flag_enum flag_type);
功能描述	获取 INTC 中断标志
先决条件	-
被调用函数	fw_debug_report_err read_intc_register
输入参数{in}	
intc_get_flag_enum	INTC 中断标志获取参考 表 3-1405.枚举 intc_get_flag_enum
输出参数{out}	
-	-
返回值	
FlagStatus	RESET 或 SET

例如：

```
/* get software interrupt flag */
FlagStatus sw_flag;
sw_flag = intc_interrupt_flag_get(SW_INT_FLAG_GET);
```

函数 intc_interrupt_flag_clear

函数intc_interrupt_flag_clear描述见下表：

表 3-1413. 函数 intc_interrupt_flag_clear

函数名称	intc_interrupt_flag_clear
函数原型	void intc_interrupt_flag_clear(intc_clear_flag_enum flag_type);
功能描述	清除 INTC 中断标志
先决条件	-
被调用函数	fw_debug_report_err write_intc_register
输入参数{in}	
intc_clear_flag_enum	INTC 中断标志清除参考 表 3-1406.枚举 intc_clear_flag_enum
输出参数{out}	
-	-

返回值	
-	-

例如：

```
/* clear software interrupt flag */
```

```
intc_interrupt_flag_clear(SW_INT_FLAG_CLEAR);
```

函数 intc_deasstat_get

函数intc_deasstat_get描述见下表：

表 3-1414. 函数 intc_deasstat_get

函数名称	intc_deasstat_get
函数原型	FlagStatus intc_deasstat_get(void);
功能描述	获取中断无效间隔状态
先决条件	-
被调用函数	read_intc_register
输入参数{in}	
-	-
输出参数{out}	
-	-
返回值	
FlagStatus	RESET 或 SET

例如：

```
/* get interrupt de-assertion interval status */
```

```
FlagStatus deas_flag;
```

```
deas_flag = intc_deasstat_get();
```

函数 intc_irqstat_get

函数intc_irqstat_get描述见下表：

表 3-1415. 函数 intc_irqstat_get

函数名称	intc_irqstat_get
函数原型	FlagStatus intc_irqstat_get(void);
功能描述	获取内部 IRQ 中断线状态
先决条件	-
被调用函数	read_intc_register
输入参数{in}	
-	-
输出参数{out}	
-	-

返回值	
FlagStatus	RESET 或 SET

例如：

```
/* get internal IRQ line status */
```

```
FlagStatus irq_flag;
```

```
irq_flag = intc_irqstat_get();
```

3.40. ESC_OSPI

EtherCAT支持SPI/QSPI/OSPI从模块。章节[3.40.1](#)描述了ESC_OSPI的寄存器列表，章节[3.40.2](#)对ESC_OSPI库函数进行说明。

3.40.1. 外设寄存器描述

ESC_OSPI寄存器列表如下表所示：

表 3-1416. ESC_OSPI 寄存器

寄存器名称	寄存器描述
OSPI_CTL	OSPI控制寄存器
OSPI_DCFG0	OSPI设备配置寄存器0
OSPI_DCFG1	OSPI设备配置寄存器1
OSPI_STAT	OSPI状态寄存器
OSPI_STATC	OSPI状态清除寄存器
OSPI_DTLEN	OSPI数据长度寄存器
OSPI_ADDR	OSPI地址寄存器
OSPI_DATA	OSPI数据寄存器
OSPI_STATMK	OSPI状态屏蔽寄存器
OSPI_STATMATCH	OSPI状态匹配寄存器
OSPI_INTERVAL	OSPI间隔寄存器
OSPI_TCFG	OSPI传输配置寄存器
OSPI_TIMCFG	OSPI时需配置寄存器
OSPI_INS	OSPI指令寄存器
OSPI_ALTE	OSPI交替字节寄存器
OSPI_WPTCFG	OSPI回卷传输配置寄存器
OSPI_WPTIMCFG	OSPI回卷时序配置寄存器
OSPI_WPINS	OSPI回卷指令寄存器
OSPI_WPALTE	OSPI回卷交替字节寄存器
OSPI_WTCFG	OSPI写入传输配置寄存器
OSPI_WTIMCFG	OSPI写入时序配置寄存器
OSPI_WINS	OSPI写入指令寄存器

寄存器名称	寄存器描述
OSPI_WALTE	OSPI写入交替字节寄存器

3.40.2. 外设库函数说明

ESC_OSPI库函数列表如下表所示：

表 3-1417. ESC_OSPI 库函数

库函数名称	库函数描述
ospi_hw_init	硬件OSPI初始化
ospi_enable_qspi_mode	使能QSPI模式
ospi_enable_ospi_mode	使能OSPI模式
ospi_reset_spi_mode	复位SPI模式
ospi_write	OSPI写数据
ospi_read	OSPI读数据
OSPIWriteRegUsingCSR	OSPI使用CSR写寄存器
OSPIReadRegUsingCSR	OSPI使用CSR读寄存器
OSPIWritePDRamRegister	OSPI写PDRam寄存器
OSPIReadPDRamRegister	OSPI读PDRam寄存器
OSPIReadRegister	OSPI读寄存器
OSPIWriteRegister	OSPI写寄存器

函数 ospi_hw_init

函数ospi_hw_init描述见下表：

表 3-1418. 函数 ospi_hw_init

函数名称	ospi_hw_init
函数原型	void ospi_hw_init(uint32_t ospi_periph, ospi_parameter_struct *ospi_struct);
功能描述	初始化硬件OSPI
先决条件	-
被调用函数	ospi_deinit, ospim_deinit, rcu_periph_clock_enable, gpio_af_set, gpio_mode_set, gpio_output_options_set, ospim_port_sck_config, ospim_port_csn_config, ospim_port_io3_0_config, ospim_port_io7_4_config, ospi_init, ospi_enable
输入参数{in}	
ospi_periph	外设OSPIx
OSPIx	x=0,1
输入参数{in}	
ospi_struct	OSPI参数初始化结构的成员
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* initialize OSPI/OSPI1 and GPIO */

ospi_parameter_struct ospi_struct = {0};

ospi_hw_init(OSPI1, &ospi_struct);
```

函数 **ospi_enable_qspi_mode**

函数ospi_enable_qspi_mode描述见下表：

表 3-1419. 函数 ospi_enable_qspi_mode

函数名称	ospi_enable_qspi_mode
函数原型	void ospi_enable_qspi_mode(uint32_t ospi_periph, ospi_parameter_struct *ospi_struct, interface_mode mode);
功能描述	使能QSPI模式
先决条件	-
被调用函数	ospi_command_config
输入参数{in}	
ospi_periph	外设OSPIx
OSPIx	x=0,1
输入参数{in}	
ospi_struct	OSPI参数初始化结构的成员
输入参数{in}	
interface_mode	SPI接口
SPI_MODE	标准SPI
QSPI_MODE	四线SPI
OSPI_MODE	八线SPI
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* enable qspi mode */

ospi_parameter_struct ospi_struct = {0};

ospi_enable_qspi_mode (OSPI1, &ospi_struct, SPI_MODE);
```

函数 **ospi_enable_ospi_mode**

函数ospi_enable_ospi_mode描述见下表：

表 3-1420. 函数 ospi_enable_ospi_mode

函数名称	ospi_enable_ospi_mode
------	-----------------------

函数原型	void ospi_enable_ospi_mode(uint32_t ospi_periph, ospi_parameter_struct *ospi_struct, interface_mode mode);
功能描述	使能OSPI模式
先决条件	-
被调用函数	ospi_command_config
输入参数{in}	
ospi_periph	外设OSPIx
OSPIx	x=0,1
输入参数{in}	
ospi_struct	OSPI参数初始化结构的成员
输入参数{in}	
interface_mode	SPI接口
SPI_MODE	标准SPI
QSPI_MODE	四线SPI
OSPI_MODE	八线SPI
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* enable ospi mode */
```

```
ospi_parameter_struct ospi_struct = {0};
```

```
ospi_enable_ospi_mode (OSPI1, &ospi_struct, SPI_MODE);
```

函数 ospi_reset_spi_mode

函数ospi_reset_spi_mode描述见下表：

表 3-1421. 函数 ospi_reset_spi_mode

函数名称	ospi_reset_spi_mode
函数原型	void ospi_reset_spi_mode(uint32_t ospi_periph, ospi_parameter_struct *ospi_struct, interface_mode mode);
功能描述	复位SPI模式
先决条件	-
被调用函数	ospi_command_config
输入参数{in}	
ospi_periph	外设OSPIx
OSPIx	x=0,1
输入参数{in}	
ospi_struct	OSPI参数初始化结构的成员
输入参数{in}	

interface_mode	SPI接口
<i>SPI_MODE</i>	标准SPI
<i>QSPI_MODE</i>	四线SPI
<i>OSPI_MODE</i>	八线SPI
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* enable ospi mode */
```

```
ospi_parameter_struct ospi_struct = {0};
```

```
ospi_reset_spi_mode(OSPI1, &ospi_struct, SPI_MODE);
```

函数 **ospi_write**

函数ospi_write描述见下表：

表 3-1422. 函数 ospi_write

函数名称	ospi_write
函数原型	void ospi_write(uint32_t ospi_periph, ospi_parameter_struct *ospi_struct, interface_mode mode, uint8_t cmd, addr_inc_mode inc_mode, uint32_t addr, uint8_t *pdata, uint32_t data_size);
功能描述	OSPI写数据
先决条件	-
被调用函数	ospi_command_config, ospi_transmit
输入参数{in}	
ospi_periph	外设OSPIx
<i>OSPIx</i>	x=0,1
输入参数{in}	
<i>ospi_struct</i>	OSPI参数初始化结构的成员
输入参数{in}	
interface_mode	SPI接口
<i>SPI_MODE</i>	标准SPI
<i>QSPI_MODE</i>	四线SPI
<i>OSPI_MODE</i>	八线SPI
输入参数{in}	
<i>cmd</i>	命令
输入参数{in}	
inc_mode	地址模式
<i>ADDR_NO_INC</i>	地址不变
<i>ADDR_INC</i>	地址递增

<i>ADDR_DEC</i>	地址递减
输入参数{in}	
<i>addr</i>	地址
输入参数{in}	
<i>pdata</i>	指向要写入的数据
输入参数{in}	
<i>data_size</i>	数据长度
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* write data */
```

```
uint32_t temp = 0U;
```

```
ospi_parameter_struct ospi_struct = {0};
```

```
ospi_write(OSPI_INTERFACE, &ospi_struct, OSPI_MODE, 0x01U, 0x4000U, 0x1000U,
(uint8_t *)&temp, 4);
```

函数 **ospi_read**

函数ospi_read描述见下表:

表 3-1423. 函数 **ospi_read**

函数名称	ospi_read
函数原型	void ospi_read(uint32_t ospi_periph, ospi_parameter_struct *ospi_struct, interface_mode mode, uint8_t cmd, addr_inc_mode inc_mode, uint32_t addr, uint8_t *pdata, uint32_t data_size);
功能描述	OSPI读数据
先决条件	-
被调用函数	ospi_command_config, ospi_transmit
输入参数{in}	
ospi_periph	外设OSPIx
<i>OSPIx</i>	x=0,1
输入参数{in}	
<i>ospi_struct</i>	OSPI参数初始化结构的成员
输入参数{in}	
interface_mode	SPI接口
<i>SPI_MODE</i>	标准SPI
<i>QSPI_MODE</i>	四线SPI
<i>OSPI_MODE</i>	八线SPI
输入参数{in}	

<i>cmd</i>	命令
输入参数{in}	
inc_mode	地址模式
<i>ADDR_NO_INC</i>	地址不变
<i>ADDR_INC</i>	地址递增
<i>ADDR_DEC</i>	地址递减
输入参数{in}	
<i>addr</i>	地址
输入参数{in}	
<i>pdata</i>	指向要读入的数据
输入参数{in}	
<i>data_size</i>	数据长度
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* read data */
```

```
uint32_t temp = 0U;
```

```
ospi_parameter_struct ospi_struct = {0};
```

```
ospi_read(OSPI_INTERFACE, &ospi_struct, OSPI_MODE, 0x01U, 0x4000U, 0x1000U,
(uint8_t *)&temp, 4);
```

函数 OSPIWriteRegUsingCSR

函数OSPIWriteRegUsingCSR描述见下表:

表 3-1424. 函数 OSPIWriteRegUsingCSR

函数名称	OSPIWriteRegUsingCSR
函数原型	void OSPIWriteRegUsingCSR(uint8_t *WriteBuffer, uint16_t Address, uint8_t Count, uint32_t ospi_periph, ospi_parameter_struct *ospi_struct, interface_mode mode, uint8_t write_cmd, uint8_t read_cmd, addr_inc_mode inc_mode);
功能描述	OSPI使用CSR写寄存器
先决条件	-
被调用函数	ospi_write, ospi_read
输入参数{in}	
WriteBuffer	指向要被写入的数据
输入参数{in}	
Address	写入的起始地址
输入参数{in}	

Count	写入的数据长度
输入参数{in}	
ospi_periph	外设OSPIx
OSPIx	x=0,1
输入参数{in}	
ospi_struct	OSPI参数初始化结构的成员
输入参数{in}	
interface_mode	SPI接口
SPI_MODE	标准SPI
QSPI_MODE	四线SPI
OSPI_MODE	八线SPI
输入参数{in}	
write_cmd	写命令
输入参数{in}	
read_cmd	读命令
输入参数{in}	
inc_mode	地址模式
ADDR_NO_INC	地址不变
ADDR_INC	地址递增
ADDR_DEC	地址递减
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* write data */
```

```
uint8_t *WriteBuffer;
```

```
uint16_t Address = 0x10;
```

```
ospi_parameter_struct ospi_struct = {0};
```

```
OSPIWriteRegUsingCSR(WriteBuffer, Address, 2, OSPI1, &ospi_init_struct, OSPI_MODE,
1U, 0x0BU, 0U);
```

函数 OSPIReadRegUsingCSR

函数OSPIReadRegUsingCSR描述见下表：

表 3-1425. 函数 OSPIReadRegUsingCSR

函数名称	OSPIReadRegUsingCSR
函数原型	void OSPIReadRegUsingCSR (uint8_t *ReadBuffer, uint16_t Address, uint8_t Count, uint32_t ospi_periph, ospi_parameter_struct *ospi_struct,

	interface_mode mode, uint8_t write_cmd, uint8_t read_cmd, addr_inc_mode inc_mode);
功能描述	OSPI使用CSR读寄存器
先决条件	-
被调用函数	ospi_write, ospi_read
输入参数{in}	
ReadBuffer	指向要被读入的数据
输入参数{in}	
Address	读入的起始地址
输入参数{in}	
Count	读入的数据长度
输入参数{in}	
ospi_periph	外设OSPIx
OSPIx	x=0,1
输入参数{in}	
ospi_struct	OSPI参数初始化结构的成员
输入参数{in}	
interface_mode	SPI接口
SPI_MODE	标准SPI
QSPI_MODE	四线SPI
OSPI_MODE	八线SPI
输入参数{in}	
write_cmd	写命令
输入参数{in}	
read_cmd	读命令
输入参数{in}	
inc_mode	地址模式
ADDR_NO_INC	地址不变
ADDR_INC	地址递增
ADDR_DEC	地址递减
输出参数{out}	
-	-
返回值	
-	-

例如:

```

/* read data */

uint8_t *ReadBuffer;

uint16_t Address = 0x10;

ospi_parameter_struct ospi_struct = {0};

```

OSPIReadRegUsingCSR(ReadBuffer, Address, 2, OSPI1, &ospi_init_struct, OSPI_MODE, 1U, 0x0BU, 0U);

函数 OSPIReadRegister

函数OSPIReadRegister描述见下表:

表 3-1426. 函数 OSPIReadRegister

函数名称	OSPIReadRegister
函数原型	void OSPIReadRegister(uint8_t *ReadBuffer, uint16_t Address, uint16_t Count);
功能描述	OSPI读寄存器
先决条件	-
被调用函数	OSPIReadPDRamRegister, OSPIReadRegUsingCSR
输入参数{in}	
ReadBuffer	指向要被读入的数据
输入参数{in}	
Address	读入的起始地址
输入参数{in}	
Count	读入的数据长度
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* read data */
uint8_t *ReadBuffer;
uint16_t Address = 0x10;
ospi_parameter_struct ospi_struct = {0};
OSPIReadRegister(ReadBuffer, Address, 2);
```

函数 OSPIWriteRegister

函数OSPIWriteRegister描述见下表:

表 3-1427. 函数 OSPIWriteRegister

函数名称	OSPIWriteRegister
函数原型	void OSPIWriteRegister(uint8_t *WriteBuffer, uint16_t Address, uint16_t Count);
功能描述	OSPI写寄存器
先决条件	-

被调用函数	OSPIReadPDRamRegister, OSPIReadRegUsingCSR
输入参数{in}	
WriteBuffer	指向要被写入的数据
输入参数{in}	
Address	写入的起始地址
输入参数{in}	
Count	写入的数据长度
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* write data */
uint8_t *WriteBuffer;
uint16_t Address = 0x10;
ospi_parameter_struct ospi_struct = {0};
OSPIWriteRegister (WriteBuffer, Address, 2);
```

3.41. ESC_PHY

EtherCAT从站控制器控制, 它包含 PHY A 和 PHY B, 两者在功能上是相同的。章节[3.42.1](#)描述了ESC PHY 控制寄存器, 章节[3.41.2](#)控制库函数进行说明。

3.41.1. 外设寄存器描述

ESC_PHY寄存器列表如下表所示:

表 3-1428.ESC_PHY 控制寄存器

寄存器名称	寄存器描述
ESC_MII_CONTROL	ESC MII管理控制寄存器
ESC_MII_STATUS	IESC MII管理状态寄存器
ESC_PHY_ADDR	ESC PHY 地址寄存器
ESC_PHY_DATA	ESC PHY 数据寄存器
ESC_MII_PDI_STATE	MII 管理 EtherCAT 访问状态寄存器
ESC_PHY_PORT0_STA	PHY 端口0状态寄存器
ESC_PHY_PORT1_STA	PHY 端口1状态寄存器

3.41.2. 外设库函数说明

PHY库函数列表如下表所示:

表 3-1429. ESC_PHY 库函数

库函数名称	库函数描述
esc_phy_read	读phy寄存器值
esc_phy_write	写phy寄存器值
esc_mmd_read	读phy MMD寄存器值
esc_mmd_write	写phy MMD寄存器值

函数 esc_phy_read

函数esc_phy_read描述见下表：

表 3-1430. 函数 esc_phy_read

函数名称	esc_phy_read
函数原型	unsigned short esc_phy_read(unsigned char phy_addr, unsigned char phy_reg);
功能描述	读PHY寄存器的值
先决条件	-
被调用函数	ospi_read
输入参数{in}	
phy_addr	phy的物理地址(0-15)
phy_reg	phy寄存器(Address offset)
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* read PHYA control register value */
uint32_t temp_value;
esc_phy_write(0x0,0x1F,0x0);
temp_value = esc_phy_read (0x00,0x00);
```

函数 esc_phy_write

函数write_intc_register描述见下表：

表 3-1431. 函数 esc_phy_write

函数名称	esc_phy_write
函数原型	void esc_phy_write(unsigned char phy_addr, unsigned char phy_reg, unsigned short phy_data);
功能描述	写PHY寄存器值
先决条件	-
被调用函数	ospi_write

输入参数{in}	
<i>phy_addr</i>	phy的物理地址(0-15)
<i>phy_reg</i>	phy寄存器(Address offset)
<i>phy_data</i>	Phy数据
输入参数{in}	
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* write phy PHYA control Register Half Duplex data value */
```

```
esc_phy_write(0x0,0x1F,0x0);
```

```
esc_phy_write(0x0,0x0,0x3000);
```

函数 **esc_mmd_read**

函数esc_mmd_read描述见下表:

表 3-1432. 函数 esc_mmd_read

函数名称	esc_mmd_read
函数原型	unsigned short esc_mmd_read(unsigned char phy_addr, unsigned char MMD_addr, unsigned short MMD_addr_index);
功能描述	从MMD寄存器读取PHY数据
先决条件	-
被调用函数	esc_phy_write esc_phy_read
输入参数{in}	
<i>phy_addr</i>	PHY的物理地址(0-15)
<i>MMD_addr</i>	MMD的地址(3,7)
<i>MMD_addr_index</i>	MMD 地址索引(Address offset)
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* read phyA specified MMD AN Control Registe */
```

```
uint32_t temp_value;
```

```
temp_value = esc_mmd_read(0x00, 0x07,0x00);
```

函数 **esc_mmd_write**

函数esc_mmd_write描述见下表:

表 3-1433. 函数 **esc_mmd_write**

函数名称	esc_mmd_write
函数原型	void esc_mmd_write(unsigned char phy_addr, unsigned char MMD_addr, unsigned short MMD_addr_index, unsigned short MMD_data);
功能描述	向MMD寄存器写入PHY数据
先决条件	-
被调用函数	esc_phy_write
输入参数{in}	
phy_addr	PHY的物理地址(0-15)
MMD_addr	MMD的地址(3,7)
MMD_addr_index	MMD 地址索引(Address offset)
MMD_data	MMD写数据
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* write phy specified MMD AN Control Register Auto-Negotiation Enable */
esc_mmd_write(0x00,0x07,0x00,0x300);
```

3.42. ESC_PMU

ESC电源管理单元(PMU)管理ESC设备的设备级和模块级功耗。章节[3.42.1](#)描述了ESC PMU控制寄存器，章节[3.42.2](#) ESC PMU控制库函数进行说明。

3.42.1. 外设寄存器描述

ESC_PMU寄存器列表如下表所示:

表 3-1434. ESC_PMU 寄存器

寄存器名称	寄存器描述
PMU_CTRL0	ESC_PMU 控制寄存器 0
PMU_CTRL1	ESC_PMU 控制寄存器 1
PMU_PDIREFVAL	ESC_PMU 处理数据接口参考值寄存器

3.42.2. 外设库函数说明

ESC_PMU库函数列表如下表所示:

表 3-1435. ESC_PMU 库函数

库函数名称	库函数描述
pmu_esc_power_mang_mode_config	配置电源管理模式
pmu_esc_wake_up_mode_config	配置 PMU 唤醒模式
pmu_esc_led_wm_config	配置 PMU LED 工作模式
pmu_esc_led_inact_stat_config	配置 PMU LED 非活动状态
pmu_esc_ready_stat_get	获取设备就绪状态
pmu_esc_edwol_stat_get	获取能量检测/WoL 端口状态
pmu_esc_edwol_stat_clear	清除能量检测/WoL 端口状态
pmu_esc_fun_config	配置 PMU 功能
pmu_esc_byte_test	测试 PMU 字节
pmu_esc_sleep_mode_enable	启用电源管理睡眠模式
pmu_esc_sleep_mode_disable	禁用电源管理睡眠模式

枚举类型 pmu_esc_fun_enum

表 3-1436. 枚举 pmu_esc_fun_enum

成员名	描述
PMU_ESC_FUN_EDWOLA	PMU 能量检测/WoL 端口 A
PMU_ESC_FUN_EDWOLB	PMU 能量检测/WoL 端口 B
PMU_ESC_FUN_ECATCHK	PMU EtherCAT 核心时钟
PMU_ESC_FUN_LEDOUT	PMU LED 输出

函数 pmu_esc_power_mang_mode_config

函数 pmu_esc_power_mang_mode_config 描述见下表：

表 3-1437. 函数 pmu_esc_power_mang_mode_config

函数名称	pmu_esc_power_mang_mode_config
函数原型	void pmu_esc_power_mang_mode_config(uint32_t pm_mode)
功能描述	配置电源管理模式

先决条件	-
被调用函数	-
输入参数{in}	
pm_mode	电源管理模式
<i>PMU_ESC_PMMOD0</i>	电源管理模式 0
<i>PMU_ESC_PMMOD1</i>	电源管理模式 1
<i>PMU_ESC_PMMOD2</i>	电源管理模式 2
<i>PMU_ESC_PMMOD3</i>	电源管理模式 3
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* write PMU ESC register vaule */
pmu_esc_power_mang_mode_config(PMU_ESC_PMMOD0);
```

函数 pmu_esc_wake_up_mode_config

函数pmu_esc_wake_up_mode_config描述见下表：

表 3-1438.函数 pmu_esc_wake_up_mode_config

函数名称	pmu_esc_wake_up_mode_config
函数原型	void pmu_esc_wake_up_mode_config(uint32_t wu_mode)
功能描述	配置 PMU 唤醒模式
先决条件	-
被调用函数	-
输入参数{in}	
wu_mode	唤醒模式
<i>PMU_WAKE_UP_MASTER</i>	PMU 由主控唤醒
<i>PMU_WAKE_UP_PME_MASTER</i>	PMU 由 PME 或主控唤醒
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* configure PMU wake up mode */
pmu_esc_wake_up_mode_config(PMU_WAKE_UP_MASTER);
```

函数 pmu_esc_led_wm_config

函数pmu_esc_led_wm_config描述见下表:

表 3-1439. 函数 pmu_esc_led_wm_config

函数名称	pmu_esc_led_wm_config
函数原型	void pmu_esc_led_wm_config(uint32_t led_wm)
功能描述	配置 PMU LED 工作模式
先决条件	-
被调用函数	-
输入参数{in}	
led_wm	LEDs 的工作模式
PMU_LED_OPEN_DR AIN	LEDs 的工作模式是开漏/开源
PMU_LED_PUSH_PUL L	LEDs 的工作模式是推挽
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* configure PMU LED work mode */
```

```
pmu_esc_led_wm_config(PMU_LED_OPEN_DRAIN);
```

函数 pmu_esc_led_inact_stat_config

函数pmu_esc_led_inact_stat_config描述见下表:

表 3-1440. 函数 pmu_esc_led_inact_stat_config

函数名称	pmu_esc_led_inact_stat_config
函数原型	void pmu_esc_led_inact_stat_config(uint32_t led_inact_stat)
功能描述	配置 PMU LED 非活动状态
先决条件	-
被调用函数	-
输入参数{in}	
led_inact_stat	LED 非活动状态
PMU_LED_INACT_STA T0	0 表示非活动状态
PMU_LED_INACT_STA T1	1 表示非活动状态
输出参数{out}	
-	-
返回值	

-	-
---	---

例如：

```
/* configure PMU LED inactive state */
```

```
pmu_esc_led_inact_stat_config(PMU_LED_INACT_STAT0);
```

函数 pmu_esc_ready_stat_get

函数pmu_esc_ready_stat_get描述见下表：

表 3-1441. 函数 pmu_esc_ready_stat_get

函数名称	pmu_esc_ready_stat_get
函数原型	FlagStatus pmu_esc_ready_stat_get(void)
功能描述	获取设备就绪状态
先决条件	-
被调用函数	-
输入参数{in}	
-	-
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* get device ready status */
```

```
FlagStatus state = pmu_esc_ready_stat_get();
```

函数 pmu_esc_edwol_stat_get

函数pmu_esc_edwol_stat_get描述见下表：

表 3-1442. 函数 pmu_esc_edwol_stat_get

函数名称	pmu_esc_edwol_stat_get
函数原型	FlagStatus pmu_esc_edwol_stat_get(uint32_t edwol_port)
功能描述	获取能量检测/WoL 端口状态
先决条件	-
被调用函数	-
输入参数{in}	
edwol_port	能量检测/WoL 端口
PMU_EDWOL_PORTA	能量检测/WoL 端口 A
PMU_EDWOL_PORTB	能量检测/WoL 端口 B
输出参数{out}	
-	-

返回值	
FlagStatus	SET 或 RESET

例如：

```
/* get energy detect / WoL port status */
```

```
FlagStatus state = pmu_esc_edwol_stat_get();
```

函数 pmu_esc_edwol_stat_clear

函数pmu_esc_edwol_stat_clear描述见下表：

表 3-1443. 函数 pmu_esc_edwol_stat_clear

函数名称	pmu_esc_edwol_stat_clear
函数原型	void pmu_esc_edwol_stat_clear(uint32_t edwol_port)
功能描述	清除能量检测/WoL 端口状态
先决条件	-
被调用函数	-
输入参数{in}	
edwol_port	能量检测/WoL 端口
PMU_EDWOL_PORTA	能量检测/WoL 端口 A
PMU_EDWOL_PORTB	能量检测/WoL 端口 B
输出参数{out}	
-	-
返回值	
FlagStatus	SET 或 RESET

例如：

```
/* clear energy detect / WoL port status */
```

```
pmu_esc_edwol_stat_clear(PMU_EDWOL_PORTA);
```

函数 pmu_esc_fun_config

函数pmu_esc_fun_config描述见下表：

表 3-1444. 函数 pmu_esc_fun_config

函数名称	pmu_esc_fun_config
函数原型	void pmu_esc_fun_config(pmu_esc_fun_enum fun_type, ControlStatus newvalue)
功能描述	配置 PMU 功能
先决条件	-
被调用函数	-
输入参数{in}	

fun_type	PMU 功能参考 m 表 3-1436. 枚举 pmu_esc_fun_en
输入参数{in}	
newvalue	ENABLE 或 DISABLE
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* configure PMU function */
pmu_esc_fun_config(PMU_ESC_FUN_EDWOLB, ENABLE);
```

函数 pmu_esc_byte_test

函数 pmu_esc_byte_test 描述见下表:

表 3-1445. 函数 pmu_esc_byte_test

函数名称	pmu_esc_byte_test
函数原型	FlagStatus pmu_esc_byte_test(void)
功能描述	测试 PMU 字节
先决条件	-
被调用函数	-
输入参数{in}	
-	-
输入参数{in}	
newvalue	ENABLE 或 DISABLE
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* test PMU byte */
pmu_esc_byte_test();
```

函数 pmu_esc_sleep_mode_enable

函数 pmu_esc_sleep_mode_enable 描述见下表:

表 3-1446. 函数 pmu_esc_sleep_mode_enable

函数名称	pmu_esc_sleep_mode_enable
函数原型	void pmu_esc_sleep_mode_enable(void)

功能描述	启用电源管理睡眠模式
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 power management sleep mode */
```

```
pmu_esc_sleep_mode_enable();
```

函数 pmu_esc_sleep_mode_disable

函数pmu_esc_sleep_mode_disable描述见下表：

表 3-1447. 函数 pmu_esc_sleep_mode_disable

函数名称	pmu_esc_sleep_mode_disable
函数原型	void pmu_esc_sleep_mode_disable(void)
功能描述	禁用电源管理睡眠模式
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 power management sleep mode */
```

```
pmu_esc_sleep_mode_disable();
```

3.43. ESC_SYSCFG

ESC系统配置 (ESC_SYSCFG)用于对ESC模块ID和ESC内核控制器进行配置。章节[3.43.1](#)描述了ESC_SYSCFG的寄存器列表，章节[3.43.2](#)对ESC_SYSCFG库函数进行说明。

3.43.1. 外设寄存器说明

ESC_SYSCFG寄存器列表如下表所示:

表 3-1448. ESC_SYSCFG 寄存器

寄存器名称	寄存器描述
ESC_EF_CHIP_ID	EFUSE芯片ID寄存器
ESC_EFUSE_UID_READ	ESC UID寄存器
ESC_CCTL_DATA	数据寄存器
ESC_CCTL_CMD	命令寄存器
ESC_PRAM_FIFO_DR	PRAM数据读寄存器
ESC_PRAM_ALR	PRAM地址和长度读寄存器
ESC_PRAM_CR	PRAM命令读寄存器
ESC_PRAM_FIFO_DW	PRAM FIFO数据读寄存器
ESC_PRAM_ALW	PRAM地址和长度写寄存器
ESC_PRAM_CW	PRAM命令写寄存器
ESC_OPB_CS	ESC OPB控制和状态寄存器
ESC_SYSCFG_CFG0	ESC系统配置寄存器0
ESC_SYSCFG_CHIPID	芯片ID寄存器
ESC_SYSCFG_CHIPVER	芯片版本寄存器
ESC_SYSCFG_RESERVED	保留寄存器

3.43.2. 外设库函数说明

WWDGT库函数列表如下表所示:

表 3-1449. ESC_SYSCFG 库函数

库函数名称	库函数说明
syscfg_get_chip_id	获取ESC芯片ID
syscfg_get_chip_version	获取ESC芯片版本
syscfg_efuse_read	读取efuse

函数 syscfg_get_chip_id

函数syscfg_get_chip_id描述见下表:

表 3-1450. 函数 syscfg_get_chip_id

函数名称	syscfg_get_chip_id
函数原型	uint32_t syscfg_get_chip_id(void);
功能描述	获取ESC芯片ID
先决条件	-
被调用函数	-
输入参数{in}	
-	-
输出参数{out}	
-	-
返回值	
uint32_t	temp_value

例如:

```
uint32_t chip_id = 0;

/* get chip id */

chip_id = syscfg_get_chip_id();
```

函数 syscfg_get_chip_version

函数syscfg_get_chip_version描述见下表:

表 3-1451. 函数 syscfg_get_chip_version

函数名称	syscfg_get_chip_version
函数原型	uint32_t syscfg_get_chip_version(void);
功能描述	获取ESC芯片ID
先决条件	-
被调用函数	-
输入参数{in}	
-	-
输出参数{out}	
-	-
返回值	
uint32_t	temp_value

例如:

```
uint32_t chip_ver = 0;

/* get chip version */

chip_ver = syscfg_get_chip_version();
```

函数 syscfg_efuse_read

函数syscfg_efuse_read描述见下表：

表 3-1452. 函数 syscfg_efuse_read

函数名称	syscfg_efuse_read
函数原型	uint32_t syscfg_efuse_read(uint32_t reg);
功能描述	读取EFUSE
先决条件	-
被调用函数	-
输入参数{in}	
reg	需访问的寄存器
ESC_EF_CHIP_ID	读取ESC_EF_CHIP_ID
ESC_EFUSE_UID_READ	读取ESC_EFUSE_UID_READ
输出参数{out}	
-	-
返回值	
uint32_t	temp_value

例如：

```
uint32_t temp_value = 0;
```

```
/* read efuse content */
```

```
temp_value = syscfg_efuse_read(ESC_EF_CHIP_ID);
```

3.44. ESC_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，分为两种类型：基本定时器（ESC_BASIC_TIMER），独立运行计数器（ESC_FRC），不同类型的定时器具体功能有所差别。章节[3.44.1](#)描述了ESC_TIMER的寄存器列表，章节[3.44.2](#)对ESC_TIMER库函数进行说明。

3.44.1. 外设寄存器描述

ESC_TIMER寄存器列表如下表所示：

表 3-1453. ESC_TIMER 寄存器

寄存器名称	寄存器描述
ESC_TIMER_CTL0	ESC_BASIC_TIMER控制寄存器
ESC_TIMER_CNT	ESC_BASIC_TIMER计数寄存器
ESC_FRC_CNT	ESC_FRC计数寄存器

3.44.2. 外设库函数说明

ESC_TIMER库函数列表如下表所示：

表 3-1454. ESC_TIMER 库函数

库函数名称	库函数描述
esc_timer_enable	ESC_BASIC_TIMER使能
esc_timer_disable	ESC_BASIC_TIMER禁能
esc_timer_autoreload_value_config	配置ESC_BASIC_TIMER自动重装值
esc_timer_counter_read	读取ESC_BASIC_TIMER计数值
esc_frc_counter_read	读取ESC_FRC计数值

函数 esc_timer_enable

函数esc_timer_enable描述见下表：

表 3-1455. 函数 esc_timer_enable

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

例如：

```
/* enable a ESC_BASIC_TIMER */
```

```
esc_timer_enable();
```

函数 esc_timer_disable

函数esc_timer_disable描述见下表：

表 3-1456. 函数 esc_timer_disable

函数名称	esc_timer_disable
函数原型	void esc_timer_disable(void);
功能描述	ESC_BASIC_TIMER禁能
先决条件	-
被调用函数	ospi_write

输入参数{in}	
-	-
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* disable a ESC_BASIC_TIMER */
```

```
esc_timer_disable();
```

函数 **esc_timer_autoreload_value_config**

函数esc_timer_autoreload_value_config描述见下表：

表 3-1457. 函数 esc_timer_autoreload_value_config

函数名称	esc_timer_autoreload_value_config
函数原型	void esc_timer_autoreload_value_config(uint16_t autoreload);
功能描述	配置ESC_BASIC_TIMER自动重装值
先决条件	-
被调用函数	ospi_write
输入参数{in}	
autoreload	自动重装计数值
<i>uint16_t</i>	0x0000 – 0xFFFF
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* configure ESC_BASIC_TIMER autoreload register value */
```

```
esc_timer_autoreload_value_config(100);
```

函数 **esc_timer_counter_read**

函数esc_timer_counter_read描述见下表：

表 3-1458. 函数 esc_timer_counter_read

函数名称	esc_timer_counter_read
函数原型	uint32_t esc_timer_counter_read(void);
功能描述	读取ESC_BASIC_TIMER计数值
先决条件	-
被调用函数	ospi_read

输入参数{in}	
-	-
输出参数{out}	
-	-
返回值	
uint32_t	计数值 0~0xFFFF

例如:

```
/* read ESC_BASIC_TIMER counter value */
```

```
uint32_t val = 0;
```

```
val = esc_timer_counter_read();
```

函数 esc_frc_counter_read

函数esc_frc_counter_read描述见下表:

表 3-1459. 函数 esc_frc_counter_read

函数名称	esc_frc_counter_read
函数原型	uint32_t esc_frc_counter_read(void);
功能描述	读取ESC_FRC计数值
先决条件	-
被调用函数	ospi_read
输入参数{in}	
-	-
输出参数{out}	
-	-
返回值	
uint32_t	计数值 0~0xFFFFFFFF

例如:

```
/* read ESC_FRC counter value */
```

```
uint32_t val = 0;
```

```
val = esc_frc_counter_read();
```

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	2024 年 11 月 08 日
1.1	1. 删除 <u>表 3-864. 函数 <code>pmu smps ldo supply config</code></u> 中 <code>PMU_SMPS_1V8_SUPPLIES_LDO</code> , <code>PMU_SMPS_2V5_SUPPLIES_LDO</code> , <code>PMU_SMPS_1V8_SUPPLIES_EXT_AND_LDO</code> , <code>PMU_SMPS_2V5_SUPPLIES_EXT_AND_LDO</code> , <code>PMU_SMPS_1V8_SUPPLIES_EXT</code> 和 <code>PMU_SMPS_2V5_SUPPLIES_EXT</code> 。	2025 年 1 月 24 日

Important Notice

This document is the property of GigaDevice Semiconductor Inc. and its subsidiaries (the "Company"). This document, including any product of the Company described in this document (the "Product"), is owned by the Company under the intellectual property laws and treaties of the People's Republic of China and other jurisdictions worldwide. The Company reserves all rights under such laws and treaties and does not grant any license under its patents, copyrights, trademarks, or other intellectual property rights. The names and brands of third party referred thereto (if any) are the property of their respective owner and referred to for identification purposes only.

The Company makes no warranty of any kind, express or implied, with regard to this document or any Product, including, but not limited to, the implied warranties of merchantability and fitness for a particular purpose. The Company does not assume any liability arising out of the application or use of any Product described in this document. Any information provided in this document is provided only for reference purposes. It is the responsibility of the user of this document to properly design, program, and test the functionality and safety of any application made of this information and any resulting product. Except for customized products which has been expressly identified in the applicable agreement, the Products are designed, developed, and/or manufactured for ordinary business, industrial, personal, and/or household applications only. The Products are not designed, intended, or authorized for use as components in systems designed or intended for the operation of weapons, weapons systems, nuclear installations, atomic energy control instruments, combustion control instruments, airplane or spaceship instruments, transportation instruments, traffic signal instruments, life-support devices or systems, other medical devices or systems (including resuscitation equipment and surgical implants), pollution control or hazardous substances management, or other uses where the failure of the device or Product could cause personal injury, death, property or environmental damage ("Unintended Uses"). Customers shall take any and all actions to ensure using and selling the Products in accordance with the applicable laws and regulations. The Company is not liable, in whole or in part, and customers shall and hereby do release the Company as well as its suppliers and/or distributors from any claim, damage, or other liability arising from or related to all Unintended Uses of the Products. Customers shall indemnify and hold the Company as well as its suppliers and/or distributors harmless from and against all claims, costs, damages, and other liabilities, including claims for personal injury or death, arising from or related to any Unintended Uses of the Products.

Information in this document is provided solely in connection with the Products. The Company reserves the right to make changes, corrections, modifications or improvements to this document and Products and services described herein at any time, without notice.