

GigaDevice Semiconductor Inc.

GD32H7xx

Arm[®] Cortex[®]-M7 32-bit MCU

**固件库
使用指南**

1.4 版本

(2025 年 1 月)

目录

目录	2
图索引	7
表索引	8
1. 介绍	53
1.1. 文档和固件库规则	53
1.1.1. 外设缩写	53
1.1.2. 命名规则	55
2. 固件库概述	56
2.1. 文件组织结构	56
2.1.1. Examples 文件夹	58
2.1.2. Firmware 文件夹	58
2.1.3. Template 文件夹	58
2.1.4. Utilities 文件夹	60
2.2. 固件库文件描述	60
3. 外设固件库	62
3.1. 外设固件库概述	62
3.2. ADC	62
3.2.1. 外设寄存器描述	62
3.2.2. 外设库函数说明	63
3.3. CAN	102
3.3.1. 外设寄存器说明	102
3.3.2. 外设库函数说明	103
3.4. CAU	149
3.4.1. 外设寄存器说明	149
3.4.2. 外设库函数说明	149
3.5. CMP	178
3.5.1. 外设寄存器说明	178
3.5.2. 外设库函数说明	178
3.6. CPDM	193
3.6.1. 外设寄存器描述	193
3.6.2. 外设库函数说明	193
3.7. CRC	199
3.7.1. 外设寄存器说明	199

3.7.2.	外设库函数说明	199
3.8.	CTC	207
3.8.1.	外设寄存器说明	207
3.8.2.	外设库函数说明	207
3.9.	DAC	221
3.9.1.	外设寄存器说明	221
3.9.2.	外设库函数说明	221
3.10.	DBG	242
3.10.1.	外设寄存器说明	242
3.10.2.	外设库函数说明	242
3.11.	DCI	248
3.11.1.	外设寄存器说明	248
3.11.2.	外设库函数说明	249
3.12.	DMA / DMAMUX	266
3.12.1.	外设寄存器说明	266
3.12.2.	外设库函数说明	267
3.13.	EDOUT	323
3.13.1.	外设寄存器描述	324
3.13.2.	外设库函数说明	324
3.14.	EFUSE	330
3.14.1.	外设寄存器描述	330
3.14.2.	外设库函数说明	331
3.15.	ENET	343
3.15.1.	外设寄存器描述	344
3.15.2.	外设库函数说明	346
3.16.	EXMC	442
3.16.1.	外设寄存器描述	442
3.16.2.	外设库函数说明	443
3.17.	EXTI	469
3.17.1.	外设寄存器说明	469
3.17.2.	外设库函数说明	470
3.18.	FAC	478
3.18.1.	外设寄存器说明	478
3.18.2.	外设库函数说明	478
3.19.	FMC	497
3.19.1.	外设寄存器说明	497
3.19.2.	外设库函数说明	498
3.20.	FWDGT	534

3.20.1.	外设寄存器说明	534
3.20.2.	外设库函数说明	535
3.21.	GPIO	540
3.21.1.	外设寄存器说明	540
3.21.2.	外设库函数说明	541
3.22.	HAU	552
3.22.1.	外设寄存器说明	552
3.22.2.	外设库函数说明	553
3.23.	HPDF	573
3.23.1.	外设寄存器说明	573
3.23.2.	外设库函数说明	573
3.24.	HWSEM	630
3.24.1.	外设寄存器描述	630
3.24.2.	外设库函数说明	631
3.25.	I2C	640
3.25.1.	外设寄存器说明	641
3.25.2.	外设库函数说明	641
3.26.	IPA	680
3.26.1.	外设寄存器描述	680
3.26.2.	外设库函数说明	681
3.27.	LPDTS	705
3.27.1.	外设寄存器说明	705
3.27.2.	外设库函数说明	705
3.28.	MDIO	715
3.28.1.	外设寄存器说明	715
3.28.2.	外设库函数说明	715
3.29.	MDMA	730
3.29.1.	外设寄存器描述	730
3.29.2.	外设库函数说明	731
3.30.	OSPI	760
3.30.1.	外设寄存器描述	760
3.30.2.	外设库函数说明	761
3.31.	OSPIM	801
3.31.1.	外设寄存器描述	802
3.31.2.	外设库函数说明	802
3.32.	MISC	808
3.32.1.	外设寄存器说明	808
3.32.2.	外设库函数说明	810

3.33.	PMU.....	821
3.33.1.	外设寄存器说明	821
3.33.2.	外设库函数说明	821
3.34.	RAMECCUM	841
3.34.1.	外设寄存器描述	841
3.34.2.	外设库函数说明	841
3.35.	RCU.....	856
3.35.1.	外设寄存器说明	856
3.35.2.	外设库函数说明	857
3.36.	RSPDIF	918
3.36.1.	外设寄存器描述	918
3.36.2.	外设库函数说明	919
3.37.	RTC	936
3.37.1.	外设寄存器描述	936
3.37.2.	外设库函数描述	937
3.38.	RTDEC	962
3.38.1.	外设寄存器描述	962
3.38.2.	外设库函数说明	963
3.39.	SAI	974
3.39.1.	外设寄存器描述	974
3.39.2.	外设库函数说明	975
3.40.	SDIO.....	1003
3.40.1.	外设寄存器描述	1003
3.40.2.	外设库函数说明	1004
3.41.	SPI.....	1046
3.41.1.	外设寄存器说明	1046
3.41.2.	外设库函数说明	1047
3.42.	SYSCFG.....	1092
3.42.1.	外设寄存器说明	1092
3.42.2.	外设库函数说明	1093
3.43.	TIMER	1116
3.43.1.	外设寄存器说明	1116
3.43.2.	外设库函数说明	1117
3.44.	TLI.....	1210
3.44.1.	外设寄存器说明	1210
3.44.2.	外设库函数说明	1210
3.45.	TMU.....	1229
3.45.1.	外设寄存器说明	1229
3.45.2.	外设库函数说明	1229

3.46.	TRGSEL.....	1238
3.46.1.	外设寄存器说明	1238
3.46.2.	外设库函数说明	1240
3.47.	TRNG.....	1248
3.47.1.	外设寄存器说明	1248
3.47.2.	外设库函数说明	1248
3.48.	USART	1270
3.48.1.	外设寄存器说明	1270
3.48.2.	外设库函数说明	1270
3.49.	VREF	1323
3.49.1.	外设寄存器说明	1324
3.49.2.	外设库函数说明	1324
3.50.	WWDGT	1329
3.50.1.	外设寄存器说明	1329
3.50.2.	外设库函数说明	1329
4.	版本历史.....	1334

图索引

图 2-1. GD32H7xx 固件库文件组织结构	57
图 2-2. 选择外设例程文件	59
图 2-3. 拷贝外设例程文件	59
图 2-4. 打开工程文件.....	59
图 2-5. 配置工程文件.....	60
图 2-6. 编译调试下载.....	60

表索引

表 1-1. 外设缩写.....	53
表 2-1. 固件函数库文件描述	60
表 3-1. 外设固件库函数描述格式.....	62
表 3-2. ADC 寄存器	62
表 3-3. ADC 库函数	63
表 3-4. 函数 adc_deinit	65
表 3-5. 函数 adc_clock_config	65
表 3-6. 函数 adc_special_function_config	67
表 3-7. 函数 adc_data_alignment_config	67
表 3-8. 函数 adc_enable.....	68
表 3-9. 函数 adc_disable	69
表 3-10. 函数 adc_calibration_mode_config.....	69
表 3-11. 函数 adc_calibration_number	70
表 3-12. 函数 adc_calibration_enable	71
表 3-13. 函数 adc_resolution_config.....	71
表 3-14. 函数 adc_internal_channel_config.....	72
表 3-15. 函数 adc_dma_mode_enable	73
表 3-16. 函数 adc_dma_mode_disable	73
表 3-17. 函数 adc_dma_request_after_last_enable.....	74
表 3-18. 函数 adc_dma_request_after_last_disable.....	74
表 3-19. 函数 adc_hpdf_mode_enable	75
表 3-20. 函数 adc_hpdf_mode_disable.....	75
表 3-21. 函数 adc_discontinuous_mode_config.....	76
表 3-22. 函数 adc_channel_length_config	77
表 3-23. 函数 adc_regular_channel_config.....	77
表 3-24. 函数 adc_inserted_channel_config.....	78
表 3-25. 函数 adc_inserted_channel_offset_config	79
表 3-26. 函数 adc_channel_differential_mode_config.....	80
表 3-27. 函数 adc_external_trigger_config	80
表 3-28. 函数 adc_software_trigger_enable.....	81
表 3-29. 函数 adc_end_of_conversion_config	82
表 3-30. 函数 adc_regular_data_read.....	83
表 3-31. 函数 adc_inserted_data_read.....	83
表 3-32. 函数 adc_watchdog0_single_channel_enable	84
表 3-33. 函数 adc_watchdog0_group_channel_enable	85
表 3-34. 函数 adc_watchdog0_disable	85
表 3-35. 函数 adc_watchdog1_channel_config.....	86
表 3-36. 函数 adc_watchdog2_channel_config.....	87
表 3-37. 函数 adc_watchdog1_disable	88
表 3-38. 函数 adc_watchdog2_disable	88

表 3-39. 函数 adc_watchdog0_threshold_config	89
表 3-40. 函数 adc_watchdog1_threshold_config	89
表 3-41. 函数 adc_watchdog2_threshold_config	90
表 3-42. 函数 adc_oversample_mode_config	90
表 3-43. 函数 adc_oversample_mode_enable	92
表 3-44. 函数 adc_oversample_mode_disable	92
表 3-45. 函数 adc_flag_get	93
表 3-46. 函数 adc_flag_clear	94
表 3-47. 函数 adc_interrupt_enable	94
表 3-48. 函数 adc_interrupt_disable	95
表 3-49. 函数 adc_interrupt_flag_get	96
表 3-50. 函数 adc_interrupt_flag_clear	97
表 3-51. 函数 adc_sync_mode_config	97
表 3-52. 函数 adc_sync_delay_config	98
表 3-53. 函数 adc_sync_dma_config	99
表 3-54. 函数 adc_sync_dma_request_after_last_enable	100
表 3-55. 函数 adc_sync_dma_request_after_last_disable	100
表 3-56. 函数 adc_sync_master_adc_regular_data0_read	101
表 3-57. 函数 adc_sync_slave_adc_regular_data0_read	101
表 3-58. 函数 adc_sync_regular_data1_read	102
表 3-59. CAN 寄存器	102
表 3-60. CAN 库函数	103
表 3-61. 结构体 can_error_counter_struct	105
表 3-62. 结构体 can_parameter_struct	105
表 3-63. 结构体 can_mailbox_descriptor_struct	106
表 3-64. 结构体 can_rx_fifo_struct	106
表 3-65. 结构体 can_fd_parameter_struct	106
表 3-66. 结构体 can_rx_fifo_id_filter_struct	107
表 3-67. 结构体 can_fifo_parameter_struct	107
表 3-68. 结构体 can_pn_mode_filter_struct	107
表 3-69. 结构体 can_pn_mode_config_struct	108
表 3-70. 结构体 can_crc_struct	108
表 3-71. 枚举类型 can_interrupt_enum	108
表 3-72. 枚举类型 can_flag_enum	110
表 3-73. 枚举类型 can_interrupt_flag_enum	112
表 3-74. 枚举类型 can_operation_modes_enum	114
表 3-75. 枚举类型 can_struct_type_enum	114
表 3-76. 枚举类型 can_error_state_enum	115
表 3-77. 函数 can_deinit	115
表 3-78. 函数 can_software_reset	116
表 3-79. 函数 can_init	116
表 3-80. 函数 can_struct_para_init	117
表 3-81. 函数 can_private_filter_config	117

表 3-82. 函数 can_operation_mode_enter.....	118
表 3-83. 函数 can_operation_mode_get.....	119
表 3-84. 函数 can_inactive_mode_exit.....	119
表 3-85. 函数 can_pn_mode_exit.....	120
表 3-86. 函数 can_fd_config.....	120
表 3-87. 函数 can_bitrate_switch_enable.....	121
表 3-88. 函数 can_bitrate_switch_disable.....	122
表 3-89. 函数 can_tdc_get.....	122
表 3-90. 函数 can_tdc_enable.....	123
表 3-91. 函数 can_tdc_disable.....	123
表 3-92. 函数 can_rx_fifo_config.....	124
表 3-93. 函数 can_rx_fifo_filter_table_config.....	124
表 3-94. 函数 can_rx_fifo_read.....	125
表 3-95. 函数 can_rx_fifo_filter_matching_number_get.....	125
表 3-96. 函数 can_rx_fifo_clear.....	126
表 3-97. 函数 can_ram_address_get.....	126
表 3-98. 函数 can_mailbox_config.....	127
表 3-99. 函数 can_mailbox_transmit_abort.....	128
表 3-100. 函数 can_mailbox_transmit_inactive.....	128
表 3-101. 函数 can_mailbox_receive_data_read.....	129
表 3-102. 函数 can_mailbox_receive_lock.....	130
表 3-103. 函数 can_mailbox_receive_unlock.....	130
表 3-104. 函数 can_mailbox_receive_inactive.....	131
表 3-105. 函数 can_mailbox_code_get.....	131
表 3-106. 函数 can_error_counter_config.....	132
表 3-107. 函数 can_error_counter_get.....	133
表 3-108. 函数 can_error_state_get.....	133
表 3-109. 函数 can_crc_get.....	134
表 3-110. 函数 can_pn_mode_config.....	134
表 3-111. 函数 can_pn_mode_filter_config.....	135
表 3-112. 函数 can_pn_mode_num_of_match_get.....	136
表 3-113. 函数 can_pn_mode_data_read.....	136
表 3-114. 函数 can_self_reception_enable.....	137
表 3-115. 函数 can_self_reception_disable.....	138
表 3-116. 函数 can_transmit_abort_enable.....	138
表 3-117. 函数 can_transmit_abort_disable.....	139
表 3-118. 函数 can_auto_busoff_recovery_enable.....	139
表 3-119. 函数 can_auto_busoff_recovery_disable.....	140
表 3-120. 函数 can_time_sync_enable.....	140
表 3-121. 函数 can_time_sync_disable.....	141
表 3-122. 函数 can_edge_filter_mode_enable.....	141
表 3-123. 函数 can_edge_filter_mode_disable.....	142
表 3-124. 函数 can_ped_mode_enable.....	142

表 3-125. 函数 can_ped_mode_disable	143
表 3-126. 函数 can_arbitration_delay_bits_config	143
表 3-127. 函数 can_bsp_mode_config	144
表 3-128. 函数 can_flag_get	144
表 3-129. 函数 can_flag_clear	145
表 3-130. 函数 can_interrupt_enable	145
表 3-131. 函数 can_interrupt_disable	146
表 3-132. 函数 can_interrupt_flag_get	147
表 3-133. 函数 can_interrupt_flag_clear	147
表 3-134. CAU 寄存器	149
表 3-135. CAU 库函数	150
表 3-136. 结构体 cau_key_parameter_struct	151
表 3-137. 结构体 cau_iv_parameter_struct	151
表 3-138. 结构体 cau_context_parameter_struct	151
表 3-139. 结构体 cau_parameter_struct	152
表 3-140. 函数 cau_deinit	152
表 3-141. 函数 cau_struct_para_init	152
表 3-142. 函数 cau_key_struct_para_init	153
表 3-143. 函数 cau_iv_struct_para_init	153
表 3-144. 函数 cau_context_struct_para_init	154
表 3-145. 函数 cau_enable	155
表 3-146. 函数 cau_disable	155
表 3-147. 函数 cau_dma_enable	156
表 3-148. 函数 cau_dma_disable	156
表 3-149. 函数 cau_init	157
表 3-150. 函数 cau_aes_key_select	158
表 3-151. 函数 cau_aes_keysize_config	158
表 3-152. 函数 cau_key_init	159
表 3-153. 函数 cau_iv_init	160
表 3-154. 函数 cau_phase_config	160
表 3-155. 函数 cau_fifo_flush	161
表 3-156. 函数 cau_enable_state_get	161
表 3-157. 函数 cau_data_write	162
表 3-158. 函数 cau_data_read	162
表 3-159. 函数 cau_context_save	163
表 3-160. 函数 cau_context_restore	164
表 3-161. 函数 cau_aes_ecb	164
表 3-162. 函数 cau_aes_cbc	165
表 3-163. 函数 cau_aes_ctr	166
表 3-164. 函数 cau_aes_cfb	167
表 3-165. 函数 cau_aes_ofb	168
表 3-166. 函数 cau_aes_gcm	169
表 3-167. 函数 cau_aes_ccm	170

表 3-168. 函数 cau_tdes_ecb.....	171
表 3-169. 函数 cau_tdes_cbc.....	172
表 3-170. 函数 cau_des_ecb.....	173
表 3-171. 函数 cau_des_cbc.....	173
表 3-172. 函数 cau_interrupt_enable.....	174
表 3-173. 函数 cau_interrupt_disable.....	175
表 3-174. 函数 cau_interrupt_flag_get.....	175
表 3-175. 函数 cau_flag_get.....	176
表 3-176. CMP 寄存器.....	178
表 3-177. CMP 库函数.....	178
表 3-178. 枚举类型 cmp_enum	179
表 3-179. 函数 cmp_deinit.....	179
表 3-180. 函数 cmp_mode_init.....	179
表 3-181. 函数 cmp_noninverting_input_select.....	181
表 3-182. 函数 cmp_output_init.....	182
表 3-183. 函数 cmp_output_mux_config.....	182
表 3-184. 函数 cmp_blanking_init.....	183
表 3-185. 函数 cmp_enable	184
表 3-186. 函数 cmp_disable	185
表 3-187. 函数 cmp_window_enable	185
表 3-188. 函数 cmp_window_disable	186
表 3-189. 函数 cmp_lock_enable	186
表 3-190. 函数 cmp_voltage_scaler_enable.....	187
表 3-191. 函数 cmp_voltage_scaler_disable.....	187
表 3-192. 函数 cmp_scaler_bridge_enable	188
表 3-193. 函数 cmp_scaler_bridge_disable	188
表 3-194. 函数 cmp_output_level_get.....	189
表 3-195. 函数 cmp_flag_get.....	189
表 3-196. 函数 cmp_flag_clear	190
表 3-197. 函数 cmp_interrupt_enable	190
表 3-198. 函数 cmp_interrupt_disable	191
表 3-199. 函数 cmp_interrupt_flag_get.....	192
表 3-200. 函数 cmp_interrupt_flag_clear.....	192
表 3-201. CPDM 寄存器.....	193
表 3-202. CPDM 库函数.....	193
表 3-203. 枚举类型 cpdm_output_phase_enum	193
表 3-204. 函数 cpdm_enable.....	194
表 3-205. 函数 cpdm_disable.....	195
表 3-206. 函数 cpdm_delayline_sample_enable	195
表 3-207. 函数 cpdm_delayline_sample_disable	196
表 3-208. 函数 cpdm_output_clock_phase_select	196
表 3-209. 函数 cpdm_delayline_length_valid_flag_get.....	197
表 3-210. 函数 cpdm_delayline_length_get	197

表 3-211. 函数 cpdm_clock_output	198
表 3-212. CRC 寄存器.....	199
表 3-213. CRC 库函数.....	199
表 3-214. 函数 crc_deinit	199
表 3-215. 函数 crc_reverse_output_data_enable	200
表 3-216. 函数 crc_reverse_output_data_disable.....	200
表 3-217. 函数 crc_data_register_reset	201
表 3-218. 函数 crc_data_register_read	201
表 3-219. 函数 crc_free_data_register_read	202
表 3-220. 函数 crc_free_data_register_write	202
表 3-221. 函数 crc_init_data_register_write.....	203
表 3-222. 函数 crc_input_data_reverse_config	203
表 3-223. 函数 crc_polynomial_size_set	204
表 3-224. 函数 crc_polynomial_set.....	205
表 3-225. 函数 crc_single_data_calculate	205
表 3-226. 函数 crc_block_data_calculate	206
表 3-227. CTC 寄存器	207
表 3-228. CTC 库函数	207
表 3-229. 函数 ctc_deinit	208
表 3-230. 函数 ctc_counter_enable	208
表 3-231. 函数 ctc_counter_disable	209
表 3-232. 函数 ctc_irc48m_trim_value_config.....	209
表 3-233. 函数 ctc_software_refsource_pulse_generate	210
表 3-234. 函数 ctc_hardware_trim_mode_config.....	210
表 3-235. 函数 ctc_refsource_polarity_config	211
表 3-236. 函数 ctc_refsource_signal_select	212
表 3-237. 函数 ctc_refsource_prescaler_config.....	212
表 3-238. 函数 ctc_clock_limit_value_config.....	213
表 3-239. 函数 ctc_counter_reload_value_config.....	214
表 3-240. 函数 ctc_counter_capture_value_read	214
表 3-241. 函数 ctc_counter_direction_read	215
表 3-242. 函数 ctc_counter_reload_value_read	215
表 3-243. 函数 ctc_irc48m_trim_value_read	216
表 3-244. 函数 ctc_flag_get.....	216
表 3-245. 函数 ctc_flag_clear.....	217
表 3-246. 函数 ctc_interrupt_enable.....	218
表 3-247. 函数 ctc_interrupt_disable.....	218
表 3-248. 函数 ctc_interrupt_flag_get	219
表 3-249. 函数 ctc_interrupt_flag_clear	220
表 3-250. DAC 寄存器.....	221
表 3-251. DAC 库函数.....	221
表 3-252. 函数 dac_deinit	222
表 3-253. 函数 dac_enable	223

表 3-254. 函数 dac_disable	223
表 3-255. 函数 dac_dma_enable	224
表 3-256. 函数 dac_dma_disable	225
表 3-257. 函数 dac_mode_config	225
表 3-258. 函数 dac_trimming_value_get	226
表 3-259. 函数 dac_trimming_value_set	227
表 3-260. 函数 dac_trimming_enable	227
表 3-261. 函数 dac_output_value_get	228
表 3-262. 函数 dac_data_set	229
表 3-263. 函数 dac_trigger_enable	229
表 3-264. 函数 dac_trigger_disable	230
表 3-265. 函数 dac_trigger_source_config	231
表 3-266. 函数 dac_software_trigger_enable	231
表 3-267. 函数 dac_wave_mode_config	232
表 3-268. 函数 dac_lfsr_noise_config	233
表 3-269. 函数 dac_triangle_noise_config	233
表 3-270. 函数 dac_concurrent_enable	234
表 3-271. 函数 dac_concurrent_disable	235
表 3-272. 函数 dac_concurrent_software_trigger_enable	235
表 3-273. 函数 dac_concurrent_data_set	236
表 3-274. 函数 dac_sample_keep_mode_config	236
表 3-275. 函数 dac_flag_get	237
表 3-276. 函数 dac_flag_clear	238
表 3-277. 函数 dac_interrupt_enable	239
表 3-278. 函数 dac_interrupt_disable	239
表 3-279. 函数 dac_interrupt_flag_get	240
表 3-280. 函数 dac_interrupt_flag_clear	240
表 3-281. DBG 寄存器	242
表 3-282. DBG 库函数	242
表 3-283. 枚举类型 dbg_periph_enum	242
表 3-284. 函数 dbg_deinit	243
表 3-285. 函数 dbg_id_get	243
表 3-286. 函数 dbg_low_power_enable	244
表 3-287. 函数 dbg_low_power_disable	244
表 3-288. 函数 dbg_periph_enable	245
表 3-289. 函数 dbg_periph_disable	246
表 3-290. 函数 dbg_trace_pin_enable	246
表 3-291. 函数 dbg_trace_pin_disable	247
表 3-292. 函数 dbg_trace_pin_mode_set	247
表 3-293. DCI 寄存器	248
表 3-294. DCI 库函数	249
表 3-295. 结构体 dci_parameter_struct	249
表 3-296. 函数 dci_deinit	250

表 3-297. 函数 dci_init.....	250
表 3-298. 函数 dci_enable	251
表 3-299. 函数 dci_disable	252
表 3-300. 函数 dci_capture_enable.....	252
表 3-301. 函数 dci_capture_disable.....	253
表 3-302. 函数 dci_external_vsync_enable	253
表 3-303. 函数 dci_external_vsync_disable	254
表 3-304. 函数 dci_automatic_error_correction_enable.....	254
表 3-305. 函数 dci_automatic_error_correction_disable.....	255
表 3-306. 函数 dci_jpeg_enable	255
表 3-307. 函数 dci_jpeg_disable	256
表 3-308. 函数 dci_crop_window_enable	256
表 3-309. 函数 dci_crop_window_disable	257
表 3-310. 函数 dci_crop_window_config.....	257
表 3-311. 函数 dci_embedded_sync_enable.....	258
表 3-312. 函数 dci_embedded_sync_disable.....	258
表 3-313. 函数 dci_ccir_enable	259
表 3-314. 函数 dci_ccir_disable	259
表 3-315. 函数 dci_ccir_mode_select.....	260
表 3-316. 函数 dci_sync_codes_config	260
表 3-317. 函数 dci_sync_codes_unmask_config.....	261
表 3-318. 函数 dci_data_read	262
表 3-319. 函数 dci_flag_get.....	262
表 3-320. 函数 dci_interrupt_enable.....	263
表 3-321. 函数 dci_interrupt_disable.....	263
表 3-322. 函数 dci_interrupt_flag_get	264
表 3-323. 函数 dci_interrupt_flag_clear	265
表 3-324. DMA 寄存器	266
表 3-325. DMAMUX 寄存器.....	266
表 3-326. DMA 库函数	267
表 3-327. DMAMUX 库函数	268
表 3-328. 结构体 dma_multi_data_parameter_struct	268
表 3-329. 结构体 dma_single_data_parameter_struct	269
表 3-330. 结构体 dmamux_sync_parameter_struct.....	269
表 3-331. 结构体 dmamux_gen_parameter_struct.....	270
表 3-332. 枚举 dma_channel_enum.....	270
表 3-333. 枚举 dmamux_multiplexer_channel_enum	270
表 3-334. 枚举 dmamux_generator_channel_enum	271
表 3-335. 枚举 dmamux_interrupt_enum	271
表 3-336. 枚举 dmamux_flag_enum	273
表 3-337. 枚举 dmamux_interrupt_flag_enum.....	274
表 3-338. 函数 dma_deinit.....	275
表 3-339. 函数 dma_single_data_para_struct_init.....	276

表 3-340. 函数 dma_multi_data_para_struct_init.....	276
表 3-341. 函数 dma_single_data_mode_init	277
表 3-342. 函数 dma_multi_data_mode_init	278
表 3-343. 函数 dma_periph_address_config	279
表 3-344. 函数 dma_memory_address_config	280
表 3-345. 函数 dma_transfer_number_config	280
表 3-346. 函数 dma_transfer_number_get	281
表 3-347. 函数 dma_priority_config.....	282
表 3-348. 函数 dma_memory_burst_beats_config	282
表 3-349. 函数 dma_periph_burst_beats_config	283
表 3-350. 函数 dma_memory_width_config	284
表 3-351. 函数 dma_periph_width_config	285
表 3-352. 函数 dma_memory_address_generation_config	286
表 3-353. 函数 dma_peripheral_address_generation_config	286
表 3-354. 函数 dma_circulation_enable	287
表 3-355. 函数 dma_circulation_disable.....	288
表 3-356. 函数 dma_channel_enable.....	288
表 3-357. 函数 dma_channel_disable.....	289
表 3-358. 函数 dma_transfer_direction_config	289
表 3-359. 函数 dma_switch_buffer_mode_config	290
表 3-360. 函数 dma_using_memory_get	291
表 3-361. 函数 dma_switch_buffer_mode_enable	292
表 3-362. 函数 dma_switch_buffer_mode_disable	292
表 3-363. 函数 dma_fifo_status_get	293
表 3-364. 函数 dma_flag_get.....	294
表 3-365. 函数 dma_flag_clear	294
表 3-366. 函数 dma_interrupt_enable	295
表 3-367. 函数 dma_interrupt_disable	296
表 3-368. 函数 dma_interrupt_flag_get.....	297
表 3-369. 函数 dma_interrupt_flag_clear.....	297
表 3-370. 函数 dmamux_sync_struct_para_init	298
表 3-371. 函数 dmamux_synchronization_init	299
表 3-372. 函数 dmamux_synchronization_enable	299
表 3-373. 函数 dmamux_synchronization_disable	300
表 3-374. 函数 dmamux_event_generation_enable.....	301
表 3-375. 函数 dmamux_event_generation_disable.....	301
表 3-376. 函数 dmamux_gen_struct_para_init.....	302
表 3-377. 函数 dmamux_request_generator_init	302
表 3-378. 函数 dmamux_request_generator_channel_enable	303
表 3-379. 函数 dmamux_request_generator_channel_disable	304
表 3-380. 函数 dmamux_synchronization_polarity_config	304
表 3-381. 函数 dmamux_request_forward_number_config	305
表 3-382. 函数 dmamux_sync_id_config.....	305

表 3-383. 函数 dmamux_request_id_config.....	307
表 3-384. 函数 dmamux_trigger_polarity_config	317
表 3-385. 函数 dmamux_request_generate_number_config	318
表 3-386. 函数 dmamux_trigger_id_config	318
表 3-387. 函数 dmamux_flag_get.....	320
表 3-388. 函数 dmamux_flag_clear.....	321
表 3-389. 函数 dmamux_interrupt_enable.....	321
表 3-390. 函数 dmamux_interrupt_disable.....	322
表 3-391. 函数 dmamux_interrupt_flag_get	322
表 3-392. 函数 dmamux_interrupt_flag_clear.....	323
表 3-393. EDOUT 寄存器.....	324
表 3-394. EDOUT 库函数.....	324
表 3-395. 函数 edout_deinit	324
表 3-396. 函数 edout_init.....	325
表 3-397. 函数 edout_enable.....	326
表 3-398. 函数 edout_disable	326
表 3-399. 函数 edout_polarity_config	326
表 3-400. 函数 edout_max_location_value_config	327
表 3-401. 函数 edout_output_counter_update	328
表 3-402. 函数 edout_current_location_config	328
表 3-403. 函数 edout_current_location_get	329
表 3-404. 函数 edout_z_output_mode_config	329
表 3-405. 函数 edout_z_output_start_loc_and_width_config.....	330
表 3-406. EFUSE 寄存器	330
表 3-407. EFUSE 库函数	331
表 3-408. 枚举类型 efuse_system_para_size_enum.....	331
表 3-409. 枚举类型 efuse_system_para_index_enum	332
表 3-410. 枚举类型 efuse_state_enum	332
表 3-411. 枚举类型 efuse_interrupt_flag_enum	332
表 3-412. 函数 Function efuse_read	332
表 3-413. 函数 efuse_write	333
表 3-414. 函数 efuse_user_control_write	334
表 3-415. 函数 efuse_write	334
表 3-416. 函数 efuse_write	335
表 3-417. 函数 efuse_write	336
表 3-418. 函数 efuse_write	336
表 3-419. 函数 efuse_aes_key_crc_get.....	337
表 3-420. 函数 efuse_monitor_program_voltage_enable	337
表 3-421. 函数 efuse_monitor_program_voltage_disable	338
表 3-422. 函数 efuse_monitor_program_voltage_get	338
表 3-423. 函数 efuse_ldo_ready_get	339
表 3-424. 函数 efuse_flag_get	339
表 3-425. 函数 efuse_flag_clear	340

表 3-426. 函数 efuse_interrupt_enable	341
表 3-427. 函数 efuse_interrupt_disable	341
表 3-428. 函数 efuse_interrupt_flag_get.....	342
表 3-429. 函数 efuse_interrupt_flag_clear.....	343
表 3-430. ENET 寄存器.....	344
表 3-431. ENET 库函数.....	346
表 3-432. 结构体 enet_initpara_struct.....	349
表 3-433. 结构体 enet_descriptors_struct.....	350
表 3-434. 结构体 enet_ptp_systime_struct	350
表 3-435. 枚举类型 enet_flag_enum	350
表 3-436. 枚举类型 enet_flag_clear_enum.....	352
表 3-437. 枚举类型 enet_int_enum.....	353
表 3-438. 枚举类型 enet_int_flag_enum.....	354
表 3-439. 枚举类型 enet_int_flag_clear_enum.....	355
表 3-440. 枚举类型 enet_desc_reg_enum.....	356
表 3-441. 枚举类型 enet_msc_counter_enum.....	357
表 3-442. 枚举类型 enet_option_enum.....	357
表 3-443. 枚举类型 enet_mediamode_enum	358
表 3-444. 枚举类型 enet_chksumconf_enum	358
表 3-445. 枚举类型 enet_frmrecept_enum.....	358
表 3-446. 枚举类型 enet_registers_type_enum	359
表 3-447. 枚举类型 enet_dmadirection_enum	359
表 3-448. 枚举类型 enet_phydirection_enum	359
表 3-449. 枚举类型 enet_regdirection_enum	359
表 3-450. 枚举类型 enet_macaddress_enum	359
表 3-451. 枚举类型 enet_descstate_enum.....	360
表 3-452. 枚举类型 enet_msc_preset_enum	360
表 3-453. 函数 enet_deinit.....	360
表 3-454. 函数 enet_initpara_config.....	361
表 3-455. 函数 enet_init	364
表 3-456. 函数 enet_software_reset	366
表 3-457. 函数 enet_rxframe_size_get.....	366
表 3-458. 函数 enet_descriptors_chain_init.....	367
表 3-459. 函数 enet_descriptors_ring_init	367
表 3-460. 函数 enet_frame_receive.....	368
表 3-461. 函数 enet_frame_transmit.....	369
表 3-462. 函数 enet_transmit_checksum_config.....	370
表 3-463. 函数 enet_enable	370
表 3-464. 函数 enet_disable	371
表 3-465. 函数 enet_mac_address_set	371
表 3-466. 函数 enet_mac_address_get	373
表 3-467. 函数 enet_flag_get.....	373
表 3-468. 函数 enet_flag_clear	376

表 3-469. 函数 enet_interrupt_enable	377
表 3-470. 函数 enet_interrupt_disable	379
表 3-471. 函数 enet_interrupt_flag_get.....	380
表 3-472. 函数 enet_interrupt_flag_clear.....	382
表 3-473. 函数 enet_tx_enable.....	384
表 3-474. 函数 enet_tx_disable.....	384
表 3-475. 函数 enet_rx_enable	385
表 3-476. 函数 enet_rx_disable	385
表 3-477. 函数 enet_registers_get	386
表 3-478. 函数 enet_debug_status_get.....	387
表 3-479. 函数 enet_address_filter_enable	388
表 3-480. 函数 enet_address_filter_disable	389
表 3-481. 函数 enet_address_filter_config.....	389
表 3-482. 函数 enet_phy_config.....	391
表 3-483. 函数 enet_phy_write_read	391
表 3-484. 函数 enet_phyloopback_enable.....	392
表 3-485. 函数 enet_phyloopback_disable.....	393
表 3-486. 函数 enet_forward_feature_enable	393
表 3-487. 函数 enet_forward_feature_disable	394
表 3-488. 函数 enet_fliter_feature_enable	395
表 3-489. 函数 enet_fliter_feature_disable	396
表 3-490. 函数 enet_pauseframe_generate	397
表 3-491. 函数 enet_pauseframe_detect_config	397
表 3-492. 函数 enet_pauseframe_config	398
表 3-493. 函数 enet_flowcontrol_threshold_config.....	399
表 3-494. 函数 enet_flowcontrol_feature_enable.....	401
表 3-495. 函数 enet_flowcontrol_feature_disable.....	401
表 3-496. 函数 enet_dmaprocess_state_get.....	402
表 3-497. 函数 enet_dmaprocess_resume	403
表 3-498. 函数 enet_rxprocess_check_recovery	404
表 3-499. 函数 enet_txfifo_flush.....	404
表 3-500. 函数 enet_current_desc_address_get.....	405
表 3-501. 函数 enet_desc_information_get.....	406
表 3-502. 函数 enet_missed_frame_counter_get.....	407
表 3-503. 函数 enet_desc_flag_get.....	407
表 3-504. 函数 enet_desc_flag_set.....	409
表 3-505. 函数 enet_desc_flag_clear.....	411
表 3-506. 函数 enet_rx_desc_immediate_receive_complete_interrupt	412
表 3-507. 函数 enet_rx_desc_delay_receive_complete_interrupt	412
表 3-508. 函数 enet_rxframe_drop.....	413
表 3-509. 函数 enet_dma_feature_enable	413
表 3-510. 函数 enet_dma_feature_disable	414
表 3-511. 函数 enet_rx_desc_enhanced_status_get	415

表 3-512. 函数 enet_desc_select_enhanced_mode.....	416
表 3-513. 函数 enet_ptp_enhanced_descriptors_chain_init.....	416
表 3-514. 函数 enet_ptp_enhanced_descriptors_ring_init.....	417
表 3-515. 函数 enet_ptpframe_receive_enhanced_mode.....	418
表 3-516. 函数 enet_ptpframe_transmit_enhanced_mode.....	418
表 3-517. 函数 enet_desc_select_normal_mode.....	419
表 3-518. 函数 enet_ptp_normal_descriptors_chain_init.....	420
表 3-519. 函数 enet_ptp_normal_descriptors_ring_init.....	421
表 3-520. 函数 enet_ptpframe_receive_normal_mode.....	421
表 3-521. 函数 enet_ptpframe_transmit_normal_mode.....	422
表 3-522. 函数 enet_wum_filter_register_pointer_reset.....	423
表 3-523. 函数 enet_wum_filter_config.....	423
表 3-524. 函数 enet_wum_feature_enable.....	424
表 3-525. 函数 enet_wum_feature_disable.....	425
表 3-526. 函数 enet_msc_counters_reset.....	426
表 3-527. 函数 enet_msc_feature_enable.....	426
表 3-528. 函数 enet_msc_feature_disable.....	427
表 3-529. 函数 enet_msc_counters_preset_config.....	428
表 3-530. 函数 enet_msc_counters_get.....	428
表 3-531. 函数 enet_ptp_subsecond_2_nanosecond.....	429
表 3-532. 函数 enet_ptp_nanosecond_2_subsecond.....	430
表 3-533. 函数 enet_ptp_feature_enable.....	430
表 3-534. 函数 enet_ptp_feature_disable.....	431
表 3-535. 函数 enet_ptp_timestamp_function_config.....	432
表 3-536. 函数 enet_ptp_subsecond_increment_config.....	434
表 3-537. 函数 enet_ptp_timestamp_addend_config.....	434
表 3-538. 函数 enet_ptp_timestamp_update_config.....	435
表 3-539. 函数 enet_ptp_expected_time_config.....	436
表 3-540. 函数 enet_ptp_system_time_get.....	436
表 3-541. 函数 enet_ptp_pps_output_frequency_config.....	437
表 3-542. 函数 enet_ptp_start.....	438
表 3-543. 函数 enet_ptp_finecorrection_adjfreq.....	439
表 3-544. 函数 enet_ptp_coarsecorrection_systime_update.....	440
表 3-545. 函数 enet_ptp_finecorrection_settime.....	440
表 3-546. 函数 enet_ptp_flag_get.....	441
表 3-547. 函数 enet_initpara_reset.....	442
表 3-548. EXMC 寄存器.....	442
表 3-549. EXMC 库函数.....	443
表 3-550. 结构体 exmc_norsram_timing_parameter_struct.....	444
表 3-551. 结构体 exmc_norsram_parameter_struct.....	444
表 3-552. 结构体 exmc_nand_timing_parameter_struct.....	445
表 3-553. 结构体 exmc_nand_parameter_struct.....	445
表 3-554. 结构体 exmc_sdram_timing_parameter_struct.....	445

表 3-555. 结构体 exmc_sdram_parameter_struct.....	446
表 3-556. 结构体 exmc_sdram_command_parameter_struct.....	446
表 3-557. 函数 exmc_norsram_deinit.....	446
表 3-558. 函数 exmc_norsram_struct_para_init	447
表 3-559. 函数 exmc_norsram_init	448
表 3-560. 函数 exmc_norsram_enable	449
表 3-561. 函数 exmc_norsram_disable	450
表 3-562. 函数 exmc_nand_deinit.....	450
表 3-563. 函数 exmc_nand_struct_para_init.....	451
表 3-564. 函数 exmc_nand_init	451
表 3-565. 函数 exmc_nand_enable	452
表 3-566. 函数 exmc_nand_disable	453
表 3-567. 函数 exmc_sdram_deinit.....	453
表 3-568. 函数 exmc_sdram_struct_para_init	454
表 3-569. 函数 exmc_sdram_init	454
表 3-570. 函数 exmc_norsram_sdram_remap_config.....	456
表 3-571. 函数 exmc_norsram_sdram_remap_get	456
表 3-572. 函数 exmc_norsram_consecutive_clock_config.....	457
表 3-573. 函数 exmc_norsram_page_size_config	458
表 3-574. 函数 exmc_nand_ecc_config	458
表 3-575. 函数 exmc_ecc_get.....	459
表 3-576. 函数 exmc_sdram_readsample_enable	459
表 3-577. 函数 exmc_sdram_readsample_disable	460
表 3-578. 函数 exmc_sdram_readsample_config	460
表 3-579. 函数 exmc_sdram_command_config	461
表 3-580. 函数 exmc_sdram_refresh_count_set.....	462
表 3-581. 函数 exmc_sdram_autorefresh_number_set	462
表 3-582. 函数 exmc_sdram_write_protection_config.....	463
表 3-583. 函数 exmc_sdram_bankstatus_get.....	463
表 3-584. 函数 exmc_flag_get.....	464
表 3-585. 函数 exmc_flag_clear	465
表 3-586. 函数 exmc_interrupt_enable	466
表 3-587. 函数 exmc_interrupt_disable	467
表 3-588. 函数 exmc_interrupt_flag_get.....	467
表 3-589. 函数 exmc_interrupt_flag_clear.....	468
表 3-590. EXTI 寄存器.....	469
表 3-591. EXTI 库函数.....	470
表 3-592. 枚举类型 exti_line_enum	470
表 3-593. 枚举类型 exti_mode_enum	471
表 3-594. 枚举类型 exti_trig_type_enum	471
表 3-595. 函数 exti_deinit	472
表 3-596. 函数 exti_init.....	472
表 3-597. 函数 exti_interrupt_enable.....	473

表 3-598. 函数 exti_interrupt_disable.....	473
表 3-599. 函数 exti_event_enable	474
表 3-600. 函数 exti_event_disable	474
表 3-601. 函数 exti_software_interrupt_enable	475
表 3-602. 函数 exti_software_interrupt_disable	475
表 3-603. 函数 exti_flag_get.....	476
表 3-604. 函数 exti_flag_clear.....	476
表 3-605. 函数 exti_interrupt_flag_get	477
表 3-606. 函数 exti_interrupt_flag_clear.....	477
表 3-607. FAC 寄存器	478
表 3-608. FAC 库函数	478
表 3-609. 结构体 fac_parameter_struct.....	479
表 3-610. 结构体 fac_fixed_data_preload_struct	479
表 3-611. 结构体 fac_float_data_preload_struct	480
表 3-612. 函数 dac_deinit	480
表 3-613. 函数 fac_struct_para_init	481
表 3-614. 函数 fac_fixed_data_preload_init	481
表 3-615. 函数 fac_float_data_preload_init.....	482
表 3-616. 函数 fac_init.....	482
表 3-617. 函数 fac_fixed_buffer_preload	483
表 3-618. 函数 fac_float_buffer_preload.....	483
表 3-619. 函数 fac_fixed_data_preload	484
表 3-620. 函数 fac_float_data_preload.....	484
表 3-621. 函数 fac_reset	485
表 3-622. 函数 fac_clip_config	485
表 3-623. 函数 fac_float_enable	486
表 3-624. 函数 fac_float_disable	486
表 3-625. 函数 fac_dma_enable	487
表 3-626. 函数 fac_dma_disable	487
表 3-627. 函数 fac_x0_config.....	488
表 3-628. 函数 fac_x1_config.....	489
表 3-629. 函数 fac_y_config.....	489
表 3-630. 函数 fac_function_config	490
表 3-631. 函数 fac_start	490
表 3-632. 函数 fac_stop	491
表 3-633. 函数 fac_finish_calculate	491
表 3-634. 函数 fac_fixed_data_write.....	492
表 3-635. 函数 fac_fixed_data_read	492
表 3-636. 函数 fac_float_data_write.....	493
表 3-637. 函数 fac_float_data_read	493
表 3-638. 函数 fac_interrupt_enable.....	494
表 3-639. 函数 fac_interrupt_disable.....	495
表 3-640. 函数 fac_interrupt_flag_get	495

表 3-641. 函数 fac_flag_get.....	496
表 3-642. FMC 寄存器.....	497
表 3-643. FMC 库函数.....	498
表 3-644. 枚举类型 fmc_state_enum	499
表 3-645. 枚举类型 fmc_flag_enum.....	500
表 3-646. 枚举类型 fmc_interrupt_flag_enum.....	500
表 3-647. 枚举类型 fmc_interrupt_enum.....	501
表 3-648. 函数 fmc_unlock.....	501
表 3-649. 函数 fmc_lock.....	501
表 3-650. 函数 fmc_page_erase	502
表 3-651. 函数 fmc_mass_erase	502
表 3-652. 函数 fmc_protection_removed_mass_erase.....	503
表 3-653. 函数 fmc_word_program.....	503
表 3-654. 函数 fmc_doubleword_program	504
表 3-655. 函数 fmc_check_programming_area_enable.....	505
表 3-656. 函数 fmc_check_programming_area_disable.....	505
表 3-657. 函数 ob_unlock.....	506
表 3-658. 函数 ob_lock.....	506
表 3-659. 函数 ob_start	507
表 3-660. 函数 ob_factory_value_config	507
表 3-661. 函数 ob_secure_access_mode_enable.....	508
表 3-662. 函数 ob_secure_access_mode_enable.....	509
表 3-663. 函数 ob_security_protection_config.....	509
表 3-664. 函数 ob_bor_threshold_config	510
表 3-665. 函数 ob_low_power_config	511
表 3-666. 函数 ob_tcm_ecc_config.....	512
表 3-667. 函数 ob_iospeed_optimize_config	513
表 3-668. 函数 ob_tcm_shared_ram_config	513
表 3-669. 函数 ob_data_program	515
表 3-670. 函数 ob_boot_address_config.....	515
表 3-671. 函数 ob_dcrp_config.....	516
表 3-672. 函数 ob_secure_area_config.....	517
表 3-673. 函数 ob_write_protection_enable.....	518
表 3-674. 函数 ob_write_protection_disable.....	519
表 3-675. 函数 ob_secure_mode_get.....	520
表 3-676. 函数 ob_secure_mode_get.....	520
表 3-677. 函数 ob_bor_threshold_get.....	521
表 3-678. 函数 ob_low_power_get.....	521
表 3-679. 函数 ob_tcm_ecc_get	523
表 3-680. 函数 ob_secure_mode_get.....	524
表 3-681. 函数 ob_tcm_shared_ram_size_get.....	524
表 3-682. 函数 ob_data_get.....	525
表 3-683. 函数 ob_boot_address_get.....	526

表 3-684. 函数 ob_dcrp_area_get	526
表 3-685. 函数 ob_secure_area_get	527
表 3-686. 函数 ob_write_protection_get	528
表 3-687. 函数 fmc_no_rtdec_config	528
表 3-688. 函数 fmc_aes_iv_config	529
表 3-689. 函数 fmc_flash_ecc_get	529
表 3-690. 函数 fmc_no_rtdec_get	530
表 3-691. 函数 fmc_aes_iv_get	530
表 3-692. 函数 fmc_pid_get	531
表 3-693. 函数 fmc_flag_get	531
表 3-694. 函数 fmc_flag_clear	532
表 3-695. 函数 fmc_interrupt_enable	532
表 3-696. 函数 fmc_interrupt_disable	533
表 3-697. 函数 fmc_interrupt_flag_get	533
表 3-698. 函数 fmc_interrupt_flag_clear	534
表 3-699. FWDGT 寄存器	534
表 3-700. FWDGT 库函数	535
表 3-701. 函数 fwdgt_write_enable	535
表 3-702. 函数 fwdgt_write_disable	535
表 3-703. 函数 fwdgt_enable	536
表 3-704. 函数 fwdgt_prescaler_value_config	536
表 3-705. 函数 fwdgt_reload_value_config	537
表 3-706. 函数 fwdgt_window_value_config	538
表 3-707. 函数 fwdgt_counter_reload	538
表 3-708. 函数 fwdgt_config	539
表 3-709. 函数 fwdgt_flag_get	539
表 3-710. GPIO 寄存器	540
表 3-711. GPIO 库函数	541
表 3-712. 函数 gpio_deinit	541
表 3-713. 函数 gpio_mode_set	542
表 3-714. 函数 gpio_output_options_set	543
表 3-715. 函数 gpio_bit_set	544
表 3-716. 函数 gpio_bit_reset	544
表 3-717. 函数 gpio_bit_write	545
表 3-718. 函数 gpio_port_write	546
表 3-719. 函数 gpio_input_filter_set	546
表 3-720. 函数 gpio_input_bit_get	547
表 3-721. 函数 gpio_input_port_get	548
表 3-722. 函数 gpio_output_bit_get	548
表 3-723. 函数 gpio_output_port_get	549
表 3-724. 函数 gpio_af_set	549
表 3-725. 函数 gpio_pin_lock	551
表 3-726. 函数 gpio_bit_toggle	551

表 3-727. 函数 gpio_port_toggle	552
表 3-728. HAU 寄存器	553
表 3-729. HAU 库函数	553
表 3-730. 结构体 hau_init_parameter_struct	554
表 3-731. 结构体 hau_digest_parameter_struct	554
表 3-732. 结构体 hau_context_parameter_struct	554
表 3-733. 函数 hau_deinit	555
表 3-734. 函数 hau_init	555
表 3-735. 函数 hau_init_struct_para_init	556
表 3-736. 函数 hau_reset	556
表 3-737. 函数 hau_last_word_validbits_num_config	557
表 3-738. 函数 hau_data_write	557
表 3-739. 函数 hau_infifo_words_num_get	558
表 3-740. 函数 hau_digest_read	558
表 3-741. 函数 hau_digest_calculation_enable	559
表 3-742. 函数 hau_multiple_single_dma_config	559
表 3-743. 函数 hau_dma_enable	560
表 3-744. 函数 hau_dma_disable	560
表 3-745. 函数 hau_context_struct_para_init	561
表 3-746. 函数 hau_context_save	561
表 3-747. 函数 hau_context_restore	562
表 3-748. 函数 hau_hash_sha_1	563
表 3-749. 函数 hau_hmac_sha_1	563
表 3-750. 函数 hau_hash_sha_224	564
表 3-751. 函数 hau_hmac_sha_224	565
表 3-752. 函数 hau_hash_sha_256	566
表 3-753. 函数 hau_hmac_sha_256	566
表 3-754. 函数 hau_hash_md5	567
表 3-755. 函数 hau_hmac_md5	568
表 3-756. 函数 hau_flag_get	569
表 3-757. 函数 hau_flag_clear	569
表 3-758. 函数 hau_interrupt_enable	570
表 3-759. 函数 hau_interrupt_disable	571
表 3-760. 函数 hau_interrupt_flag_get	571
表 3-761. 函数 hau_interrupt_flag_clear	572
表 3-762. HPDF 寄存器	573
表 3-763. HPDF 库函数	573
表 3-764. 结构体 hpdf_channel_parameter_struct	576
表 3-765. 结构体 hpdf_filter_parameter_struct	576
表 3-766. 结构体 hpdf_rc_parameter_struct	576
表 3-767. 结构体 hpdf_ic_parameter_struct	577
表 3-768. 枚举类型 hpdf_channel_enum	577
表 3-769. 枚举类型 hpdf_filter_enum	577

表 3-770. 枚举类型 hpdf_flag_enum	577
表 3-771. 枚举类型 hpdf_interrupt_flag_enum	578
表 3-772. 枚举类型 hpdf_interrupt_enum	579
表 3-773. 函数 hpdf_deinit.....	579
表 3-774. 函数 hpdf_channel_struct_para_init.....	580
表 3-775. 函数 hpdf_filter_struct_para_init.....	580
表 3-776. 函数 hpdf_rc_struct_para_init.....	581
表 3-777. 函数 hpdf_ic_struct_para_init.....	581
表 3-778. 函数 hpdf_enable.....	582
表 3-779. 函数 hpdf_disable.....	583
表 3-780. 函数 hpdf_channel_init.....	583
表 3-781. 函数 hpdf_filter_init.....	584
表 3-782. 函数 hpdf_rc_init	585
表 3-783. 函数 hpdf_ic_init.....	586
表 3-784. 函数 hpdf_clock_output_config.....	586
表 3-785. 函数 hpdf_clock_output_source_config	587
表 3-786. 函数 hpdf_clock_output_duty_mode_disable.....	588
表 3-787. 函数 hpdf_clock_output_duty_mode_enable.....	588
表 3-788. 函数 hpdf_clock_output_divider_config	589
表 3-789. 函数 hpdf_channel_enable	589
表 3-790. 函数 hpdf_channel_disable	590
表 3-791. 函数 hpdf_spi_clock_source_config	590
表 3-792. 函数 hpdf_serial_interface_type_config	591
表 3-793. 函数 hpdf_malfunction_monitor_disable.....	592
表 3-794. 函数 hpdf_malfunction_monitor_enable.....	592
表 3-795. 函数 hpdf_clock_loss_disable	593
表 3-796. 函数 hpdf_clock_loss_enable	593
表 3-797. 函数 hpdf_channel_pin_redirection_disable.....	594
表 3-798. 函数 hpdf_channel_pin_redirection_enable	594
表 3-799. 函数 hpdf_channel_multiplexer_config.....	595
表 3-800. 函数 hpdf_data_pack_mode_config	595
表 3-801. 函数 hpdf_data_right_bit_shift_config	596
表 3-802. 函数 hpdf_calibration_offset_config.....	596
表 3-803. 函数 hpdf_malfunction_break_signal_config	597
表 3-804. 函数 hpdf_malfunction_counter_config.....	598
表 3-805. 函数 hpdf_write_parallel_data_standard_mode	598
表 3-806. 函数 hpdf_write_parallel_data_interleaved_mode	599
表 3-807. 函数 hpdf_write_parallel_data_dual_mode	599
表 3-808. 函数 hpdf_pulse_skip_update.....	600
表 3-809. 函数 hpdf_pulse_skip_read	601
表 3-810. 函数 hpdf_filter_enable	601
表 3-811. 函数 hpdf_filter_disable	602
表 3-812. 函数 hpdf_filter_config	602

表 3-813. 函数 hpdf_integrator_oversample	603
表 3-814. 函数 hpdf_threshold_monitor_filter_config	604
表 3-815. 函数 hpdf_threshold_monitor_filter_read_data	604
表 3-816. 函数 hpdf_threshold_monitor_fast_mode_disable	605
表 3-817. 函数 hpdf_threshold_monitor_fast_mode_enable	605
表 3-818. 函数 hpdf_threshold_monitor_channel	606
表 3-819. 函数 hpdf_threshold_monitor_high_threshold	607
表 3-820. 函数 hpdf_threshold_monitor_low_threshold	607
表 3-821. 函数 hpdf_high_threshold_break_signal	608
表 3-822. 函数 hpdf_low_threshold_break_signal	608
表 3-823. 函数 hpdf_extremes_monitor_channel	609
表 3-824. 函数 hpdf_extremes_monitor_maximum_get	610
表 3-825. 函数 hpdf_extremes_monitor_minimum_get	610
表 3-826. 函数 hpdf_conversion_time_get	611
表 3-827. 函数 hpdf_rc_continuous_disable	612
表 3-828. 函数 hpdf_rc_continuous_enable	612
表 3-829. 函数 hpdf_rc_start_by_software	613
表 3-830. 函数 hpdf_rc_syn_disable	613
表 3-831. 函数 hpdf_rc_syn_disable	614
表 3-832. 函数 hpdf_rc_dma_disable	614
表 3-833. 函数 hpdf_rc_dma_enable	615
表 3-834. 函数 hpdf_rc_channel_config	615
表 3-835. 函数 hpdf_rc_fast_mode_disable	616
表 3-836. 函数 hpdf_rc_fast_mode_enable	616
表 3-837. 函数 hpdf_rc_data_get	617
表 3-838. 函数 hpdf_rc_channel_get	617
表 3-839. 函数 hpdf_ic_start_by_software	618
表 3-840. 函数 hpdf_ic_syn_disable	618
表 3-841. 函数 hpdf_ic_syn_enable	619
表 3-842. 函数 hpdf_ic_dma_disable	619
表 3-843. 函数 hpdf_ic_dma_enable	620
表 3-844. 函数 hpdf_ic_scan_mode_disable	620
表 3-845. 函数 hpdf_ic_scan_mode_enable	621
表 3-846. 函数 hpdf_ic_trigger_signal_disable	621
表 3-847. 函数 hpdf_ic_trigger_signal_config	622
表 3-848. 函数 hpdf_ic_channel_config	623
表 3-849. 函数 hpdf_ic_data_get	624
表 3-850. 函数 hpdf_ic_channel_get	624
表 3-851. 函数 hpdf_flag_get	625
表 3-852. 函数 hpdf_flag_clear	626
表 3-853. 函数 hpdf_interrupt_enable	627
表 3-854. 函数 hpdf_interrupt_disable	628
表 3-855. 函数 hpdf_interrupt_flag_get	628

表 3-856. 函数 hpdf_interrupt_flag_clear	629
表 3-857. HWSEM 寄存器	630
表 3-858. HWSEM 库函数	631
表 3-859. 枚举类型 hwsem_semaphore_enum	631
表 3-860. 函数 hwsem_lock_set	632
表 3-861. 函数 hwsem_lock_release	633
表 3-862. 函数 hwsem_lock_by_reading	634
表 3-863. 函数 hwsem_unlock_all	634
表 3-864. 函数 hwsem_process_id_get	635
表 3-865. 函数 hwsem_master_id_get	635
表 3-866. 函数 hwsem_lock_status_get	636
表 3-867. 函数 hwsem_key_set	636
表 3-868. 函数 hwsem_key_get	637
表 3-869. 函数 hwsem_flag_get	637
表 3-870. 函数 hwsem_flag_clear	638
表 3-871. 函数 hwsem_interrupt_flag_get	638
表 3-872. 函数 hwsem_interrupt_flag_clear	639
表 3-873. 函数 hwsem_interrupt_enable	639
表 3-874. 函数 hwsem_interrupt_disable	640
表 3-875. I2C 寄存器	641
表 3-876. I2C 库函数	641
表 3-877. 枚举类型 i2c_interrupt_flag_enum	643
表 3-878. 函数 i2c_deinit	643
表 3-879. 函数 i2c_timing_config	644
表 3-880. 函数 i2c_digital_noise_filter_config	644
表 3-881. 函数 i2c_analog_noise_filter_enable	645
表 3-882. 函数 i2c_analog_noise_filter_disable	646
表 3-883. 函数 i2c_master_clock_config	646
表 3-884. 函数 i2c_master_addressing	647
表 3-885. 函数 i2c_address10_header_enable	648
表 3-886. 函数 i2c_address10_header_disable	648
表 3-887. 函数 i2c_address10_enable	649
表 3-888. 函数 i2c_address10_disable	649
表 3-889. 函数 i2c_automatic_end_enable	650
表 3-890. 函数 i2c_automatic_end_disable	650
表 3-891. 函数 i2c_slave_response_to_gcall_enable	651
表 3-892. 函数 i2c_slave_response_to_gcall_disable	651
表 3-893. 函数 i2c_stretch_scl_low_enable	652
表 3-894. 函数 i2c_stretch_scl_low_disable	652
表 3-895. 函数 i2c_address_config	653
表 3-896. 函数 i2c_address_bit_compare_config	653
表 3-897. 函数 i2c_address_disable	654
表 3-898. 函数 i2c_second_address_config	655

表 3-899. 函数 i2c_second_address_disable.....	656
表 3-900. 函数 i2c_receved_address_get.....	656
表 3-901. 函数 i2c_slave_byte_control_enable	657
表 3-902. 函数 i2c_slave_byte_control_disable	657
表 3-903. 函数 i2c_nack_enable.....	658
表 3-904. 函数 i2c_nack_disable.....	658
表 3-905. 函数 i2c_wakeup_from_deepsleep_enable	659
表 3-906. 函数 i2c_wakeup_from_deepsleep_disable	659
表 3-907. 函数 i2c_enable.....	660
表 3-908. 函数 i2c_disable.....	660
表 3-909. 函数 i2c_start_on_bus.....	661
表 3-910. 函数 i2c_stop_on_bus	661
表 3-911. 函数 i2c_data_transmit.....	662
表 3-912. 函数 i2c_data_receive.....	663
表 3-913. 函数 i2c_reload_enable	663
表 3-914. 函数 i2c_reload_disable	664
表 3-915. 函数 i2c_transfer_byte_number_config	664
表 3-916. 函数 i2c_dma_enable.....	665
表 3-917. 函数 i2c_dma_disable.....	665
表 3-918. 函数 i2c_pec_transfer.....	666
表 3-919. 函数 i2c_pec_enable.....	666
表 3-920. 函数 i2c_pec_disable	667
表 3-921. 函数 i2c_pec_value_get	667
表 3-922. 函数 i2c_smbus_alert_enable	668
表 3-923. 函数 i2c_smbus_alert_disable	668
表 3-924. 函数 i2c_smbus_default_addr_enable	669
表 3-925. 函数 i2c_smbus_default_addr_disable	670
表 3-926. 函数 i2c_smbus_host_addr_enable.....	670
表 3-927. 函数 i2c_smbus_host_addr_disable.....	671
表 3-928. 函数 i2c_extented_clock_timeout_enable	671
表 3-929. 函数 i2c_extented_clock_timeout_disable	672
表 3-930. 函数 i2c_clock_timeout_enable	672
表 3-931. 函数 i2c_clock_timeout_disable.....	673
表 3-932. 函数 i2c_bus_timeout_b_config	673
表 3-933. 函数 i2c_bus_timeout_a_config	674
表 3-934. 函数 i2c_idle_clock_timeout_config.....	674
表 3-935. 函数 i2c_flag_get.....	675
表 3-936. 函数 i2c_flag_clear.....	676
表 3-937. 函数 i2c_interrupt_enable.....	677
表 3-938. 函数 i2c_interrupt_disable.....	677
表 3-939. 函数 i2c_interrupt_flag_get	678
表 3-940. 函数 i2c_interrupt_flag_clear	679
表 3-941. IPA 寄存器.....	680

表 3-942. IPA 库函数	681
表 3-943. 结构体 ipa_foreground_parameter_struct	682
表 3-944. 结构体 ipa_background_parameter_struct	683
表 3-945. 结构体 ipa_destination_parameter_struct	683
表 3-946. 结构体 ipa_conversion_parameter_struct	683
表 3-947. 枚举类型 ipa_dpf_enum	684
表 3-948. 枚举类型 ipa_colorspace_enum	684
表 3-949. 函数 ipa_deinit	684
表 3-950. 函数 ipa_transfer_enable	685
表 3-951. 函数 ipa_transfer_hangup_enable	685
表 3-952. 函数 ipa_transfer_hangup_disable	686
表 3-953. 函数 ipa_transfer_stop_enable	686
表 3-954. 函数 ipa_transfer_stop_disable	687
表 3-955. 函数 ipa_foreground_lut_loading_enable	687
表 3-956. 函数 ipa_background_lut_loading_enable	688
表 3-957. 函数 ipa_pixel_format_convert_mode_set	688
表 3-958. 函数 ipa_foreground_interlace_mode_enable	689
表 3-959. 函数 ipa_foreground_interlace_mode_disable	689
表 3-960. 函数 ipa_foreground_struct_para_init	690
表 3-961. 函数 ipa_foreground_init	690
表 3-962. 函数 ipa_background_struct_para_init	691
表 3-963. 函数 ipa_background_init	692
表 3-964. 函数 ipa_destination_struct_para_init	693
表 3-965. 函数 ipa_destination_init	693
表 3-966. 函数 ipa_foreground_lut_init	695
表 3-967. 函数 ipa_background_lut_init	695
表 3-968. 函数 ipa_line_mark_config	696
表 3-969. 函数 ipa_inter_timer_config	697
表 3-970. 函数 ipa_interval_clock_num_config	697
表 3-971. 函数 ipa_color_conversion_struct_para_init	698
表 3-972. 函数 ipa_color_conversion_config	698
表 3-973. 函数 ipa_foreground_scaling_config	699
表 3-974. 函数 ipa_destination_scaling_config	700
表 3-975. 函数	701
表 3-976. 函数	702
表 3-977. 函数 ipa_interrupt_enable	702
表 3-978. 函数 ipa_interrupt_disable	703
表 3-979. 函数 ipa_interrupt_flag_get	704
表 3-980. 函数 ipa_interrupt_flag_clear	704
表 3-981. LPDTS 寄存器	705
表 3-982. LPDTS 库函数	705
表 3-983. 结构体 lpdts_parameter_struct	706
表 3-984. 函数 lpdts_deinit	706

表 3-985. 函数 lpdts_struct_para_init	707
表 3-986. 函数 lpdts_init	707
表 3-987. 函数 lpdts_enable	708
表 3-988. 函数 lpdts_disable	708
表 3-989. 函数 lpdts_soft_trigger_enable	709
表 3-990. 函数 lpdts_soft_trigger_disable	709
表 3-991. 函数 lpdts_high_threshold_set	710
表 3-992. 函数 lpdts_low_threshold_set	710
表 3-993. 函数 lpdts_ref_clock_source_config	711
表 3-994. 函数 lpdts_temperature_get	711
表 3-995. 函数 lpdts_flag_get	712
表 3-996. 函数 lpdts_interrupt_enable	712
表 3-997. 函数 lpdts_interrupt_disable	713
表 3-998. 函数 lpdts_interrupt_flag_get	714
表 3-999. 函数 lpdts_interrupt_flag_clear	714
表 3-1000. MDIO 寄存器	715
表 3-1001. MDIO 库函数	716
表 3-1002. 函数 mdio_deinit	716
表 3-1003. 函数 mdio_software_reset	717
表 3-1004. 函数 mdio_init	717
表 3-1005. 函数 mdio_phy_length_config	718
表 3-1006. 函数 mdio_soft_phyadr_set	719
表 3-1007. 函数 mdio_framefield_phyadr_config	719
表 3-1008. 函数 mdio_framefield_devadd_config	720
表 3-1009. 函数 mdio_phy_pin_read	720
表 3-1010. 函数 mdio_timeout_config	721
表 3-1011. 函数 mdio_timeout_enable	722
表 3-1012. 函数 mdio_timeout_disable	722
表 3-1013. 函数 mdio_op_receive	723
表 3-1014. 函数 mdio_phyadr_receive	723
表 3-1015. 函数 mdio_devadd_receive	724
表 3-1016. 函数 mdio_ta_receive	724
表 3-1017. 函数 mdio_data_receive	725
表 3-1018. 函数 mdio_address_receive	725
表 3-1019. 函数 mdio_data_transmit	726
表 3-1020. 函数 mdio_flag_get	726
表 3-1021. 函数 mdio_flag_clear	727
表 3-1022. 函数 mdio_interrupt_enable	728
表 3-1023. 函数 mdio_interrupt_disable	729
表 3-1024. MDMA 寄存器	730
表 3-1025. MDMA 库函数	731
表 3-1026. 结构体 mdma_parameter_struct	732
表 3-1027. 结构体 mdma_multi_block_parameter_struct	733

表 3-1028. 结构体 mdma_link_node_parameter_struct	733
表 3-1029. 枚举 mdma_add_update_dir_enum	733
表 3-1030. 枚举 mdma_channel_enum	733
表 3-1031. 函数 mdma_deinit	734
表 3-1032. 函数 mdma_channel_deinit	734
表 3-1033. 函数 mdma_para_struct_init	735
表 3-1034. 函数 mdma_multi_block_para_struct_init	735
表 3-1035. 函数 mdma_link_node_para_struct_init	736
表 3-1036. 函数 mdma_init	737
表 3-1037. 函数 mdma_buffer_block_mode_config	738
表 3-1038. 函数 mdma_multi_block_mode_config	739
表 3-1039. 函数 mdma_node_create	739
表 3-1040. 函数 mdma_node_add	740
表 3-1041. 函数 mdma_node_delete	741
表 3-1042. 函数 mdma_destination_address_config	741
表 3-1043. 函数 mdma_source_address_config	742
表 3-1044. 函数 mdma_destination_bus_config	743
表 3-1045. 函数 mdma_source_bus_config	743
表 3-1046. 函数 mdma_priority_config	744
表 3-1047. 函数 mdma_endianness_config	745
表 3-1048. 函数 mdma_alignment_config	745
表 3-1049. 函数 mdma_source_burst_beats_config	746
表 3-1050. 函数 mdma_destination_burst_beats_config	747
表 3-1051. 函数 mdma_source_width_config	748
表 3-1052. 函数 mdma_destination_width_config	749
表 3-1053. 函数 mdma_source_increment_config	750
表 3-1054. 函数 mdma_destination_increment_config	751
表 3-1055. 函数 mdma_channel_bufferable_write_enable	752
表 3-1056. 函数 mdma_channel_bufferable_write_disable	752
表 3-1057. 函数 mdma_channel_software_request_enable	753
表 3-1058. 函数 mdma_channel_enable	753
表 3-1059. 函数 mdma_channel_disable	754
表 3-1060. 函数 mdma_transfer_error_direction_get	754
表 3-1061. 函数 mdma_transfer_error_address_get	755
表 3-1062. 函数 mdma_flag_get	755
表 3-1063. 函数 mdma_flag_clear	756
表 3-1064. 函数 mdma_interrupt_enable	757
表 3-1065. 函数 mdma_interrupt_disable	758
表 3-1066. 函数 mdma_interrupt_flag_get	758
表 3-1067. 函数 mdma_interrupt_flag_clear	759
表 3-1068. OSPI 寄存器	760
表 3-1069. OSPI 库函数	761
表 3-1070. 结构体 ospi_parameter_struct	762

表 3-1071. 结构体 ospi_regular_cmd_struct.....	763
表 3-1072. 结构体 ospi_autopolling_struct.....	764
表 3-1073. 枚举 ospi_interrupt_flag_enum	764
表 3-1074. 函数 ospi_deinit.....	765
表 3-1075. 函数 ospi_struct_init.....	765
表 3-1076. 函数 ospi_init	766
表 3-1077. 函数 ospi_enable	767
表 3-1078. 函数 ospi_disable.....	767
表 3-1079. 函数 ospi_device_memory_type_config	768
表 3-1080. 函数 ospi_device_memory_size_config.....	768
表 3-1081. 函数 ospi_functional_mode_config	769
表 3-1082. 函数 ospi_status_polling_config.....	770
表 3-1083. 函数 ospi_status_mask_config.....	770
表 3-1084. 函数 ospi_status_match_config	771
表 3-1085. 函数 ospi_interval_cycle_config	772
表 3-1086. 函数 ospi_fifo_level_config	772
表 3-1087. 函数 ospi_chip_select_high_cycle_config	773
表 3-1088. 函数 ospi_prescaler_config.....	773
表 3-1089. 函数 ospi_dummy_cycles_config	774
表 3-1090. 函数 ospi_delay_hold_cycle_config	774
表 3-1091. 函数 ospi_sample_shift_config	775
表 3-1092. 函数 ospi_data_length_config	776
表 3-1093. 函数 ospi_instruction_config.....	776
表 3-1094. 函数 ospi_address_config.....	777
表 3-1095. 函数 ospi_alternate_bytes_config.....	778
表 3-1096. 函数 ospi_data_config.....	780
表 3-1097. 函数 ospi_data_transmit	781
表 3-1098. 函数 ospi_data_receive	781
表 3-1099. 函数 ospi_dma_enable	782
表 3-1100. 函数 ospi_dma_disable	782
表 3-1101. 函数 ospi_wrap_size_config.....	783
表 3-1102. 函数 ospi_wrap_instruction_config	783
表 3-1103. 函数 ospi_wrap_address_config	784
表 3-1104. 函数 ospi_wrap_alternate_bytes_config	786
表 3-1105. 函数 ospi_wrap_data_config.....	787
表 3-1106. 函数 ospi_wrap_dummy_cycles_config.....	788
表 3-1107. 函数 ospi_wrap_delay_hold_cycle_config	788
表 3-1108. 函数 ospi_wrap_sample_shift_config.....	789
表 3-1109. 函数 ospi_write_instruction_config	790
表 3-1110. 函数 ospi_write_address_config	791
表 3-1111. 函数 ospi_write_alternate_bytes_config	792
表 3-1112. 函数 ospi_write_data_config	793
表 3-1113. 函数 ospi_write_dummy_cycles_config.....	794

表 3-1114. 函数 <code>ospi_write_dummy_cycles_config</code>	795
表 3-1115. 函数 <code>ospi_transmit</code>	795
表 3-1116. 函数 <code>ospi_receive</code>	796
表 3-1117. 函数 <code>ospi_autopolling_mode</code>	796
表 3-1118. 函数 <code>ospi_interrupt_enable</code>	797
表 3-1119. 函数 <code>ospi_interrupt_disable</code>	798
表 3-1120. 函数 <code>ospi_fifo_level_get</code>	798
表 3-1121. 函数 <code>ospi_flag_get</code>	799
表 3-1122. 函数 <code>ospi_flag_clear</code>	799
表 3-1123. 函数 <code>ospi_interrupt_flag_get</code>	800
表 3-1124. 函数 <code>ospi_interrupt_flag_clear</code>	801
表 3-1125. OSPIM 寄存器	802
表 3-1126. OSPIM 库函数	802
表 3-1127. 函数 <code>ospim_deinit</code>	802
表 3-1128. 函数 <code>ospim_port_sck_config</code>	803
表 3-1129. 函数 <code>ospim_port_sck_source_select</code>	803
表 3-1130. 函数 <code>ospim_port_csn_config</code>	804
表 3-1131. 函数 <code>ospim_port_csn_source_select</code>	805
表 3-1132. 函数 <code>ospim_port_io3_0_config</code>	805
表 3-1133. 函数 <code>ospim_port_io3_0_source_select</code>	806
表 3-1134. 函数 <code>ospim_port_io7_4_config</code>	807
表 3-1135. 函数 <code>ospim_port_io7_4_source_select</code>	808
表 3-1136. NVIC 寄存器	808
表 3-1137. SysTick 寄存器	810
表 3-1138. MISC 库函数	810
表 3-1139. 结构体 <code>mpu_region_init_struct</code>	810
表 3-1140. 枚举类型 <code>IRQn_Type</code>	811
表 3-1141. 函数 <code>nvic_priority_group_set</code>	815
表 3-1142. 函数 <code>nvic_irq_enable</code>	816
表 3-1143. 函数 <code>nvic_irq_disable</code>	816
表 3-1144. 函数 <code>nvic_vector_table_set</code>	817
表 3-1145. 函数 <code>system_lowpower_set</code>	817
表 3-1146. 函数 <code>system_lowpower_reset</code>	818
表 3-1147. 函数 <code>systick_clksource_set</code>	819
表 3-1148. 函数 <code>mpu_region_struct_para_init</code>	819
表 3-1149. 函数 <code>mpu_region_config</code>	820
表 3-1150. 函数 <code>mpu_region_enable</code>	820
表 3-1151. PMU 寄存器	821
表 3-1152. PMU 库函数	821
表 3-1153. 函数 <code>pmu_deinit</code>	822
表 3-1154. 函数 <code>pmu_lvd_select</code>	823
表 3-1155. 函数 <code>pmu_lvd_enable</code>	823
表 3-1156. 函数 <code>pmu_lvd_disable</code>	824

表 3-1157. 函数 pmu_avd_select.....	824
表 3-1158. 函数 pmu_avd_enable	825
表 3-1159. 函数 pmu_avd_disable	825
表 3-1160. 函数 pmu_cvd_enable	826
表 3-1161. 函数 pmu_cvd_disable	826
表 3-1162. 函数 pmu_ldo_output_select.....	827
表 3-1163. 函数 pmu_slido_output_select	828
表 3-1164. 函数 pmu_vbat_charging_select	828
表 3-1165. 函数 pmu_vbat_charging_enable	829
表 3-1166. 函数 pmu_vbat_charging_disable.....	829
表 3-1167. 函数 pmu_vbat_temp_monitor_enable	830
表 3-1168. 函数 pmu_vbat_temp_monitor_disable.....	830
表 3-1169. 函数 pmu_usb_regulator_enable.....	831
表 3-1170. 函数 pmu_usb_regulator_disable.....	831
表 3-1171. 函数 pmu_usb_voltage_detector_enable.....	832
表 3-1172. 函数 pmu_usb_voltage_detector_disable.....	832
表 3-1173. 函数 pmu_smps_ldo_supply_config	833
表 3-1174. 函数 pmu_to_sleepmode	833
表 3-1175. 函数 pmu_to_deepsleepmode.....	834
表 3-1176. 函数 pmu_to_standbymode.....	834
表 3-1177. 函数 pmu_wakeup_pin_enable	835
表 3-1178. 函数 pmu_wakeup_pin_disable	835
表 3-1179. 函数 pmu_backup_write_enable.....	836
表 3-1180. 函数 pmu_backup_write_disable.....	837
表 3-1181. 函数 pmu_backup_voltage_stabilizer_enable.....	837
表 3-1182. 函数 pmu_backup_voltage_stabilizer_disable.....	838
表 3-1183. 函数 pmu_enter_deepsleep_wait_time_config	838
表 3-1184. 函数 pmu_enter_deepsleep_wait_time_config	839
表 3-1185. 函数 pmu_flag_get	839
表 3-1186. 函数 pmu_flag_clear	840
表 3-1187. RAMECCMU 寄存器.....	841
表 3-1188. RAMECCMU 库函数.....	841
表 3-1189. 枚举类型 rameccmu_monitor_enum	842
表 3-1190. 函数 rameccmu_deinit.....	842
表 3-1191. 函数 rameccmu_monitor_failing_address_get	843
表 3-1192. 函数 rameccmu_monitor_failing_data_low_bits_get.....	844
表 3-1193. 函数 rameccmu_monitor_failing_data_high_bits_get.....	845
表 3-1194. 函数 rameccmu_monitor_failing_ecc_error_code_get	846
表 3-1195. 函数 rameccmu_global_interrupt_enable	847
表 3-1196. 函数 rameccmu_global_interrupt_disable	848
表 3-1197. 函数 rameccmu_monitor_interrupt_enable	848
表 3-1198. 函数 rameccmu_monitor_interrupt_disable	850
表 3-1199. 函数 rameccmu_monitor_flag_get	851

表 3-1200. 函数 <code>rameccmu_monitor_flag_clear</code>	852
表 3-1201. 函数 <code>rameccmu_monitor_interrupt_flag_get</code>	853
表 3-1202. 函数 <code>rameccmu_monitor_interrupt_flag_clear</code>	855
表 3-1203. RCU 寄存器	856
表 3-1204. RCU 库函数	858
表 3-1205. 枚举类型 <code>rcu_periph_enum</code>	860
表 3-1206. 枚举类型 <code>rcu_periph_sleep_enum</code>	862
表 3-1207. 枚举类型 <code>rcu_periph_reset_enum</code>	865
表 3-1208. 枚举类型 <code>rcu_flag_enum</code>	868
表 3-1209. 枚举类型 <code>rcu_int_flag_enum</code>	868
表 3-1210. 枚举类型 <code>rcu_int_flag_clear_enum</code>	869
表 3-1211. 枚举类型 <code>rcu_int_enum</code>	870
表 3-1212. 枚举类型 <code>rcu_osci_type_enum</code>	870
表 3-1213. 枚举类型 <code>rcu_clock_freq_enum</code>	870
表 3-1214. 枚举类型 <code>usart_idx_enum</code>	871
表 3-1215. 枚举类型 <code>i2c_idx_enum</code>	871
表 3-1216. 枚举类型 <code>can_idx_enum</code>	871
表 3-1217. 枚举类型 <code>sai_idx_enum</code>	872
表 3-1218. 枚举类型 <code>sai2b_idx_enum</code>	872
表 3-1219. 枚举类型 <code>adc_idx_enum</code>	872
表 3-1220. 枚举类型 <code>usbhs_idx_enum</code>	872
表 3-1221. 枚举类型 <code>pll_idx_enum</code>	872
表 3-1222. 枚举类型 <code>sdio_idx_enum</code>	872
表 3-1223. 枚举类型 <code>spi_idx_enum</code>	873
表 3-1224. 函数 <code>rcu_deinit</code>	873
表 3-1225. 函数 <code>rcu_periph_clock_enable</code>	873
表 3-1226. 函数 <code>rcu_periph_clock_disable</code>	874
表 3-1227. 函数 <code>rcu_periph_clock_sleep_enable</code>	874
表 3-1228. 函数 <code>rcu_periph_clock_sleep_disable</code>	875
表 3-1229. 函数 <code>rcu_periph_reset_enable</code>	875
表 3-1230. 函数 <code>rcu_periph_reset_disable</code>	876
表 3-1231. 函数 <code>rcu_bkp_reset_enable</code>	876
表 3-1232. 函数 <code>rcu_bkp_reset_disable</code>	877
表 3-1233. 函数 <code>rcu_system_clock_source_config</code>	877
表 3-1234. 函数 <code>rcu_system_clock_source_get</code>	878
表 3-1235. 函数 <code>rcu_ahb_clock_config</code>	878
表 3-1236. 函数 <code>rcu_apb1_clock_config</code>	879
表 3-1237. 函数 <code>rcu_apb2_clock_config</code>	880
表 3-1238. 函数 <code>rcu_apb3_clock_config</code>	880
表 3-1239. 函数 <code>rcu_apb4_clock_config</code>	881
表 3-1240. 函数 <code>rcu_ckout0_config</code>	881
表 3-1241. 函数 <code>rcu_ckout1_config</code>	882
表 3-1242. 函数 <code>rcu_pll_input_output_clock_range_config</code>	883

表 3-1243. 函数 rcu_pll_fractional_config	884
表 3-1244. 函数 rcu_pll_fractional_latch_enable	884
表 3-1245. 函数 rcu_pll_fractional_latch_disable	885
表 3-1246. 函数 rcu_pll_source_config	885
表 3-1247. 函数 rcu_pll0_config	886
表 3-1248. 函数 rcu_pll1_config	887
表 3-1249. 函数 rcu_pll2_config	888
表 3-1250. 函数 rcu_pll_clock_output_enable	888
表 3-1251. 函数 rcu_pll_clock_output_disable	889
表 3-1252. 函数 rcu_pllusb0_config	890
表 3-1253. 函数 rcu_pllusb1_config	891
表 3-1254. 函数 rcu_rtc_clock_config	892
表 3-1255. 函数 rcu_rtc_div_config	892
表 3-1256. 函数 rcu_ck48m_clock_config	893
表 3-1257. 函数 rcu_pll48m_clock_config	894
表 3-1258. 函数 rcu_irc64mdiv_clock_config	894
表 3-1259. 函数 rcu_irc64mdiv_freq_get	895
表 3-1260. 函数 rcu_timer_clock_prescaler_config	895
表 3-1261. 函数 rcu_spi_clock_config	896
表 3-1262. 函数 rcu_sdio_clock_config	897
表 3-1263. 函数 rcu_sdio_clock_config	898
表 3-1264. 函数 rcu_tli_clock_div_config	898
表 3-1265. 函数 rcu_usart_clock_config	899
表 3-1266. 函数 rcu_i2c_clock_config	900
表 3-1267. 函数 rcu_can_clock_config	900
表 3-1268. 函数 rcu_adc_clock_config	901
表 3-1269. 函数 rcu_sai_clock_config	902
表 3-1270. 函数 rcu_sai2_block_clock_config	902
表 3-1271. 函数 rcu_rspdif_clock_config	903
表 3-1272. 函数 rcu_exmc_clock_config	904
表 3-1273. 函数 rcu_hpdf_clock_config	905
表 3-1274. 函数 rcu_per_clock_config	905
表 3-1275. 函数 rcu_usbhs_pll1qpsc_config	906
表 3-1276. 函数 rcu_usb48m_clock_config	906
表 3-1277. 函数 rcu_usbhs_clock_config	907
表 3-1278. 函数 rcu_usbhs_clock_selection_enable	908
表 3-1279. 函数 rcu_usbhs_clock_selection_disable	908
表 3-1280. 函数 rcu_lxtal_drive_capability_config	909
表 3-1281. 函数 rcu_osci_stab_wait	909
表 3-1282. 函数 rcu_osci_on	910
表 3-1283. 函数 rcu_osci_off	910
表 3-1284. 函数 rcu_osci_bypass_mode_enable	911
表 3-1285. 函数 rcu_osci_bypass_mode_disable	911

表 3-1286. 函数 rcu_irc64m_adjust_value_set	912
表 3-1287. 函数 rcu_lpirc4m_adjust_value_set	912
表 3-1288. 函数 rcu_hxtal_clock_monitor_enable	913
表 3-1289. 函数 rcu_hxtal_clock_monitor_disable	913
表 3-1290. 函数 rcu_lxtal_clock_monitor_enable	914
表 3-1291. 函数 rcu_lxtal_clock_monitor_disable	914
表 3-1292. 函数 rcu_clock_freq_get	915
表 3-1293. 函数 rcu_flag_get	915
表 3-1294. 函数 rcu_all_reset_flag_clear	916
表 3-1295. 函数 rcu_interrupt_enable	916
表 3-1296. 函数 rcu_interrupt_disable	917
表 3-1297. 函数 rcu_interrupt_flag_get	917
表 3-1298. 函数 rcu_interrupt_flag_clear	918
表 3-1299. RSPDIF 寄存器	919
表 3-1300. RSPDIF 库函数	919
表 3-1301. 结构体 rspdif_parameter_struct	920
表 3-1302. 结构体 rspdif_data_struct	920
表 3-1303. 函数 rspdif_deinit	921
表 3-1304. 函数 rspdif_struct_para_init	921
表 3-1305. 函数 rspdif_init	922
表 3-1306. 函数 rspdif_enable	923
表 3-1307. 函数 rspdif_disable	923
表 3-1308. 函数 rspdif_symbol_clock_enable	924
表 3-1309. 函数 rspdif_symbol_clock_disable	924
表 3-1310. 函数 rspdif_backup_symbol_clock_enable	925
表 3-1311. 函数 rspdif_backup_symbol_clock_disable	925
表 3-1312. 函数 rspdif_dma_enable	926
表 3-1313. 函数 rspdif_dma_disable	926
表 3-1314. 函数 rspdif_control_buffer_dma_enable	927
表 3-1315. 函数 rspdif_control_buffer_dma_disable	927
表 3-1316. 函数 rspdif_data_read	928
表 3-1317. 函数 rspdif_duration_of_symbols_get	928
表 3-1318. 函数 rspdif_user_data_get	929
表 3-1319. 函数 rspdif_channel_status_get	929
表 3-1320. 函数 rspdif_start_block_status_get	930
表 3-1321. 函数 rspdif_low_threshold_get	930
表 3-1322. 函数 rspdif_high_threshold_get	931
表 3-1323. 函数 rspdif_flag_get	931
表 3-1324. 函数 rspdif_flag_clear	932
表 3-1325. 函数 rspdif_interrupt_enable	933
表 3-1326. 函数 rspdif_interrupt_disable	934
表 3-1327. 函数 rspdif_interrupt_flag_get	934
表 3-1328. 函数 rspdif_interrupt_flag_clear	935

表 3-1329. RTC 寄存器	936
表 3-1330. RTC 库函数	937
表 3-1331. 结构体 rtc_parameter_struct	938
表 3-1332. 结构体 rtc_alarm_struct	938
表 3-1333. 结构体 rtc_timestamp_struct	939
表 3-1334. 结构体 rtc_tamper_struct	939
表 3-1335. 函数 rtc_deinit	939
表 3-1336. 函数 rtc_init	940
表 3-1337. 函数 rtc_init_mode_enter	941
表 3-1338. 函数 rtc_init_mode_exit	941
表 3-1339. 函数 rtc_register_sync_wait	942
表 3-1340. 函数 rtc_current_time_get	942
表 3-1341. 函数 rtc_subsecond_get	943
表 3-1342. 函数 rtc_alarm_config	943
表 3-1343. 函数 rtc_alarm_subsecond_config	944
表 3-1344. 函数 rtc_alarm_enable	945
表 3-1345. 函数 rtc_alarm_disable	945
表 3-1346. 函数 rtc_alarm_get	946
表 3-1347. 函数 rtc_alarm_subsecond_get	946
表 3-1348. 函数 rtc_timestamp_enable	947
表 3-1349. 函数 rtc_timestamp_disable	947
表 3-1350. 函数 rtc_timestamp_internalevent_config	948
表 3-1351. 函数 rtc_timestamp_get	948
表 3-1352. 函数 rtc_timestamp_subsecond_get	949
表 3-1353. 函数 rtc_tamper_enable	949
表 3-1354. 函数 rtc_tamper_disable	950
表 3-1355. 函数 rtc_output_pin_select	950
表 3-1356. 函数 rtc_alarm_output_config	951
表 3-1357. 函数 rtc_calibration_output_config	952
表 3-1358. 函数 rtc_hour_adjust	952
表 3-1359. 函数 rtc_second_adjust	953
表 3-1360. 函数 rtc_bypass_shadow_enable	954
表 3-1361. 函数 rtc_bypass_shadow_disable	954
表 3-1362. 函数 rtc_refclock_detection_enable	955
表 3-1363. 函数 rtc_refclock_detection_disable	955
表 3-1364. 函数 rtc_wakeup_enable	956
表 3-1365. 函数 rtc_wakeup_disable	956
表 3-1366. 函数 rtc_wakeup_clock_set	957
表 3-1367. 函数 rtc_wakeup_timer_set	957
表 3-1368. 函数 rtc_wakeup_timer_get	958
表 3-1369. 函数 rtc_smooth_calibration_config	958
表 3-1370. 函数 rtc_interrupt_enable	959
表 3-1371. 函数 rtc_interrupt_disable	960

表 3-1372. 函数 rtc_flag_get	961
表 3-1373. 函数 rtc_flag_clear	961
表 3-1374. RTDEC 寄存器.....	962
表 3-1375. RTDEC 库函数.....	963
表 3-1376. 结构体 rtdec_parameter_struct.....	963
表 3-1377. 函数 rtdec_deinit	964
表 3-1378. 函数 rtdec_struct_para_init	964
表 3-1379. 函数 rtdec_init.....	965
表 3-1380. 函数 rtdec_config	965
表 3-1381. 函数 rtdec_lock.....	966
表 3-1382. 函数 rtdec_addr_init	967
表 3-1383. 函数 rtdec_nonce_init.....	967
表 3-1384. 函数 rtdec_key_init	968
表 3-1385. 函数 rtdec_key_crc_get.....	969
表 3-1386. 函数 rtdec_enable	969
表 3-1387. 函数 rtdec_disable	970
表 3-1388. 函数 rtdec_flag_get.....	970
表 3-1389. 函数 rtdec_flag_clear.....	971
表 3-1390. 函数 rtdec_interrupt_enable	972
表 3-1391. 函数 rtdec_interrupt_disable.....	972
表 3-1392. 函数 rtdec_interrupt_flag_get	973
表 3-1393. 函数 rtdec_interrupt_flag_clear	974
表 3-1394. SAI 寄存器.....	975
表 3-1395. SAI 库函数.....	975
表 3-1396. 结构体 sai_parameter_struct.....	976
表 3-1397. 结构体 sai_frame_parameter_struct	977
表 3-1398. 结构体 sai_slot_parameter_struct.....	977
表 3-1399. 枚举 sai_fifo_state_enum	977
表 3-1400. 函数 sai_deinit	977
表 3-1401. 函数 sai_struct_para_init	978
表 3-1402. 函数 sai_frame_struct_para_init.....	978
表 3-1403. 函数 sai_slot_struct_para_init	979
表 3-1404. 函数 sai_init.....	979
表 3-1405. 函数 sai_frame_init	980
表 3-1406. 函数 sai_slot_init.....	981
表 3-1407. 函数 sai_enable	982
表 3-1408. 函数 sai_disable	983
表 3-1409. 函数 sai_sdoutput_config.....	983
表 3-1410. 函数 sai_monomode_config.....	984
表 3-1411. 函数 sai_companing_config.....	985
表 3-1412. 函数 sai_mute_enable	985
表 3-1413. 函数 sai_disable	986
表 3-1414. 函数 sai_monomode_config.....	987

表 3-1415. 函数 sai_mute_count_config	987
表 3-1416. 函数 sai_data_transmit.....	988
表 3-1417. 函数 sai_data_receive.....	989
表 3-1418. 函数 sai_fifo_status_get.....	989
表 3-1419. 函数 sai_fifo_flush	990
表 3-1420. 函数 sai_dma_enable.....	990
表 3-1421. 函数 sai_dma_disable.....	991
表 3-1422. 函数 sai_sync_input_config	992
表 3-1423. 函数 sai_sync_output_config	992
表 3-1424. 函数 sai_pdm_enable	993
表 3-1425. 函数 sai_pdm_disable	994
表 3-1426. 函数 sai_pdm_microphone_num_config	994
表 3-1427. 函数 sai_pdm_delay_config	995
表 3-1428. 函数 sai_pdm_clk0_enable	996
表 3-1429. 函数 sai_pdm_clk0_disable	996
表 3-1430. 函数 sai_pdm_clk1_enable	997
表 3-1431. 函数 sai_pdm_clk1_disable	997
表 3-1432. 函数 sai_interrupt_enable.....	998
表 3-1433. 函数 sai_interrupt_disable.....	998
表 3-1434. 函数 sai_interrupt_flag_get	999
表 3-1435. 函数 sai_interrupt_flag_clear	1000
表 3-1436. 函数 sai_flag_get.....	1001
表 3-1437. 函数 sai_flag_get.....	1002
表 3-1438. SDIO 寄存器.....	1003
表 3-1439. SDIO 库函数.....	1004
表 3-1440. 函数 sdio_deinit.....	1005
表 3-1441. 函数 sdio_clock_config.....	1006
表 3-1442. 函数 sdio_clock_receive_set	1007
表 3-1443. 函数 sdio_hardware_clock_enable	1007
表 3-1444. 函数 sdio_hardware_clock_disable	1008
表 3-1445. 函数 sdio_bus_mode_set	1008
表 3-1446. 函数 sdio_bus_speed_set.....	1009
表 3-1447. 函数 sdio_data_rate_set.....	1010
表 3-1448. 函数 sdio_direction_polarity_set.....	1010
表 3-1449. 函数 sdio_power_state_set	1011
表 3-1450. 函数 sdio_power_state_get	1012
表 3-1451. 函数 sdio_command_response_config.....	1012
表 3-1452. 函数 sdio_wait_type_set.....	1013
表 3-1453. 函数 sdio_trans_start_enable	1014
表 3-1454. 函数 sdio_trans_start_disable	1015
表 3-1455. 函数 sdio_trans_stop_enable	1015
表 3-1456. 函数 sdio_trans_stop_disable	1016
表 3-1457. 函数 sdio_csm_enable	1016

表 3-1458. 函数 sdio_csm_disable	1017
表 3-1459. 函数 sdio_command_index_get	1017
表 3-1460. 函数 sdio_response_get	1018
表 3-1461. 函数 sdio_hold_enable	1019
表 3-1462. 函数 sdio_hold_disable.....	1019
表 3-1463. 函数 sdio_suspend_enable	1020
表 3-1464. 函数 sdio_suspend_disable	1020
表 3-1465. 函数 sdio_data_config.....	1021
表 3-1466. 函数 sdio_data_transfer_config	1022
表 3-1467. 函数 sdio_dsm_enable	1023
表 3-1468. 函数 sdio_dsm_disable.....	1024
表 3-1469. 函数 sdio_data_write	1024
表 3-1470. 函数 sdio_data_read	1025
表 3-1471. 函数 sdio_data_counter_get	1025
表 3-1472. 函数 sdio_fifo_reset_enable.....	1026
表 3-1473. 函数 sdio_fifo_reset_disable.....	1026
表 3-1474. 函数 sdio_idma_set.....	1027
表 3-1475. 函数 sdio_idma_buffer0_address_set.....	1028
表 3-1476. 函数 sdio_idma_buffer1_address_set.....	1028
表 3-1477. 函数 sdio_buffer_selection_get.....	1029
表 3-1478. 函数 sdio_idma_buffer_select	1029
表 3-1479. 函数 sdio_idma_enable	1030
表 3-1480. 函数 sdio_idma_disable.....	1031
表 3-1481. 函数 sdio_flag_get	1031
表 3-1482. 函数 sdio_flag_clear	1033
表 3-1483. 函数 sdio_interrupt_enable	1035
表 3-1484. 函数 sdio_interrupt_disable	1036
表 3-1485. 函数 sdio_interrupt_flag_get.....	1038
表 3-1486. 函数 sdio_interrupt_flag_clear.....	1039
表 3-1487. 函数 sdio_voltage_switch_enable.....	1041
表 3-1488. 函数 sdio_voltage_switch_disable.....	1041
表 3-1489. 函数 sdio_voltage_switch_sequence_enable	1042
表 3-1490. 函数 sdio_voltage_switch_sequence_disable	1042
表 3-1491. 函数 sdio_boot_mode_set.....	1043
表 3-1492. 函数 sdio_boot_ack_enable	1044
表 3-1493. 函数 sdio_boot_ack_disable	1044
表 3-1494. 函数 sdio_boot_acktimeout_set	1045
表 3-1495. 函数 sdio_boot_enable.....	1045
表 3-1496. 函数 sdio_boot_disable.....	1046
表 3-1497. SPI/I2S 寄存器.....	1047
表 3-1498. SPI/I2S 库函数	1047
表 3-1499. 结构体 spi_parameter_struct.....	1049
表 3-1500. 函数 spi_i2s_deinit.....	1049

表 3-1501. 函数 spi_struct_para_init.....	1050
表 3-1502. 函数 spi_init.....	1050
表 3-1503. 函数 spi_enable	1051
表 3-1504. 函数 spi_disable	1052
表 3-1505. 函数 i2s_init.....	1052
表 3-1506. 函数 i2s_psc_config	1053
表 3-1507. 函数 i2s_enable	1055
表 3-1508. 函数 i2s_disable	1055
表 3-1509. 函数 spi_io_config	1056
表 3-1510. 函数 spi_nss_idleness_delay_set.....	1056
表 3-1511. 函数 spi_data_frame_delay_set.....	1057
表 3-1512. 函数 spi_master_receive_clock_delay_set	1058
表 3-1513. 函数 spi_slave_receive_clock_delay_set	1058
表 3-1514. 函数 spi_master_receive_clock_delay_clear	1059
表 3-1515. 函数 spi_slave_receive_clock_delay_clear.....	1059
表 3-1516. 函数 spi_nss_output_control	1060
表 3-1517. 函数 spi_nss_polarity_set	1060
表 3-1518. 函数 spi_nss_output_enable.....	1061
表 3-1519. 函数 spi_nss_output_disable.....	1062
表 3-1520. 函数 spi_nss_internal_high.....	1062
表 3-1521. 函数 spi_nss_internal_low.....	1063
表 3-1522. 函数 spi_dma_enable	1063
表 3-1523. 函数 spi_dma_disable	1064
表 3-1524. 函数 spi_i2s_data_frame_size_config.....	1064
表 3-1525. 函数 spi_i2s_data_transmit.....	1065
表 3-1526. 函数 spi_i2s_data_receive.....	1066
表 3-1527. 函数 spi_bidirectional_transfer_config.....	1066
表 3-1528. 函数 spi_master_transfer_start	1067
表 3-1529. 函数 spi_current_data_num_config.....	1067
表 3-1530. 函数 spi_reload_data_num_config	1068
表 3-1531. 函数 spi_crc_polynomial_set.....	1069
表 3-1532. 函数 spi_crc_polynomial_get.....	1069
表 3-1533. 函数 spi_crc_length_config.....	1070
表 3-1534. 函数 spi_crc_on.....	1070
表 3-1535. 函数 spi_crc_off.....	1071
表 3-1536. 函数 spi_crc_get.....	1071
表 3-1537. 函数 spi_crc_full_size_enable	1072
表 3-1538. 函数 spi_crc_full_size_enable	1072
表 3-1539. 函数 spi_tcr_init_pattern.....	1073
表 3-1540. 函数 spi_tcr_init_pattern.....	1073
表 3-1541. 函数 spi_ti_mode_enable.....	1074
表 3-1542. 函数 spi_ti_mode_disable.....	1075
表 3-1543. 函数 spi_quad_enable	1075

表 3-1544. 函数 spi_quad_disable	1076
表 3-1545. 函数 spi_quad_write_enable	1076
表 3-1546. 函数 spi_quad_read_enable	1077
表 3-1547. 函数 spi_quad_io23_output_enable.....	1077
表 3-1548. 函数 spi_quad_io23_output_disable	1078
表 3-1549. 函数 spi_underrun_operation	1078
表 3-1550. 函数 spi_underrun_config	1079
表 3-1551. 函数 spi_underrun_data_config	1080
表 3-1552. 函数 spi_suspend_mode_config	1080
表 3-1553. 函数 spi_suspend_request.....	1081
表 3-1554. 函数 spi_related_ios_af_enable.....	1081
表 3-1555. 函数 spi_related_ios_af_enable.....	1082
表 3-1556. 函数 spi_af_gpio_control.....	1082
表 3-1557. 函数 spi_i2s_interrupt_enable	1083
表 3-1558. 函数 spi_i2s_interrupt_disable	1084
表 3-1559. 函数 spi_i2s_interrupt_flag_get.....	1085
表 3-1560. 函数 spi_i2s_flag_get	1086
表 3-1561. 函数 spi_crc_error_clear.....	1087
表 3-1562. 函数 spi_i2s_rxfifo_plevel_get.....	1088
表 3-1563. 函数 spi_i2s_remain_data_num_get.....	1089
表 3-1564. 函数 spi_fifo_threshold_level_set.....	1089
表 3-1565. 函数 spi_enable	1090
表 3-1566. 函数 spi_disable	1090
表 3-1567. 函数 spi_byte_access_enable.....	1091
表 3-1568. 函数 spi_byte_access_disable.....	1091
表 3-1569. SYSCFG 寄存器	1092
表 3-1570. SYSCFG 库函数	1093
表 3-1571. 枚举类型 timer_channel_input_enum	1094
表 3-1572. 函数 syscfg_deinit.....	1100
表 3-1573. 函数 syscfg_i2c_fast_mode_plus_enable	1100
表 3-1574. 函数 syscfg_i2c_fast_mode_plus_disable	1101
表 3-1575. 函数 syscfg_analog_switch_enable.....	1102
表 3-1576. 函数 syscfg_analog_switch_disable	1103
表 3-1577. 函数 syscfg_enet_phy_interface_config.....	1103
表 3-1578. 函数 syscfg_exti_line_config	1104
表 3-1579. 函数 syscfg_lockup_enable	1105
表 3-1580. 函数 syscfg_lockup_disable	1106
表 3-1581. 函数 syscfg_timer_input_source_select.....	1106
表 3-1582. 函数 syscfg_compensation_config.....	1107
表 3-1583. 函数 syscfg_io_low_voltage_speed_optimization_enable	1108
表 3-1584. 函数 syscfg_io_low_voltage_speed_optimization_disable	1108
表 3-1585. 函数 syscfg_pnmos_compensation_code_set	1109
表 3-1586. 函数 syscfg_secure_sram_size_set.....	1109

表 3-1587. 函数 syscfg_secure_sram_size_get	1110
表 3-1588. 函数 syscfg_bootmode_get.....	1111
表 3-1589. 函数 syscfg_tcm_wait_state_enable	1111
表 3-1590. 函数 syscfg_tcm_wait_state_disable	1112
表 3-1591. 函数 syscfg_fpu_interrupt_enable	1112
表 3-1592. 函数 syscfg_fpu_interrupt_disable	1113
表 3-1593. 函数 syscfg_compensation_flag_get	1114
表 3-1594. 函数 syscfg_cpu_cache_status_get	1114
表 3-1595. 函数 syscfg_brownout_reset_threshold_level_get.....	1115
表 3-1596. TIMER 寄存器.....	1116
表 3-1597. TIMER 库函数.....	1117
表 3-1598. 结构体 timer_parameter_struct.....	1121
表 3-1599. 结构体 timer_break_parameter_struct	1121
表 3-1600. 结构体 timer_oc_parameter_struct	1122
表 3-1601. 结构体 timer_omc_parameter_struct.....	1122
表 3-1602. 结构体 timer_ic_parameter_struct	1122
表 3-1603. 结构体 timer_free_complementary_parameter_struct.....	1123
表 3-1604. 函数 timer_deinit	1123
表 3-1605. 函数 timer_struct_para_init	1124
表 3-1606. 函数 timer_init.....	1124
表 3-1607. 函数 timer_enable	1125
表 3-1608. 函数 timer_disable	1125
表 3-1609. 函数 timer_auto_reload_shadow_enable.....	1126
表 3-1610. 函数 timer_auto_reload_shadow_disable.....	1127
表 3-1611. 函数 timer_update_event_enable	1127
表 3-1612. 函数 timer_update_event_disable.....	1128
表 3-1613. 函数 timer_counter_alignment.....	1128
表 3-1614. 函数 timer_counter_up_direction.....	1129
表 3-1615. 函数 timer_counter_down_direction	1129
表 3-1616. 函数 timer_prescaler_config	1130
表 3-1617. 函数 timer_repetition_value_config	1131
表 3-1618. 函数 timer_runtime_repetition_value_read	1132
表 3-1619. 函数 timer_autoreload_value_config	1132
表 3-1620. 函数 timer_autoreload_value_read	1133
表 3-1621. 函数 timer_counter_value_config	1133
表 3-1622. 函数 timer_counter_read	1134
表 3-1623. 函数 timer_prescaler_read.....	1135
表 3-1624. 函数 timer_single_pulse_mode_config.....	1135
表 3-1625. 函数 timer_delayable_single_pulse_mode_config	1136
表 3-1626. 函数 timer_update_source_config	1137
表 3-1627. 函数 timer_dma_enable.....	1138
表 3-1628. 函数 timer_dma_disable.....	1139
表 3-1629. 函数 timer_channel_dma_request_source_select	1140

表 3-1630. 函数 timer_dma_transfer_config	1141
表 3-1631. 函数 timer_event_software_generate	1144
表 3-1632. 函数 timer_break_struct_para_init	1145
表 3-1633. 函数 timer_break_config	1145
表 3-1634. 函数 timer_break_enable	1147
表 3-1635. 函数 timer_break_disable	1147
表 3-1636. 函数 timer_automatic_output_enable	1148
表 3-1637. 函数 timer_automatic_output_disable	1148
表 3-1638. 函数 timer_primary_output_config	1149
表 3-1639. 函数 timer_channel_control_shadow_config	1150
表 3-1640. 函数 timer_channel_control_shadow_update_config	1150
表 3-1641. 函数 timer_channel_output_struct_para_init	1151
表 3-1642. 函数 timer_channel_output_config	1152
表 3-1643. 函数 timer_channel_output_mode_config	1153
表 3-1644. 函数 timer_channel_output_pulse_value_config	1154
表 3-1645. 函数 timer_channel_output_shadow_config	1155
表 3-1646. 函数 timer_channel_output_clear_config	1156
表 3-1647. 函数 timer_channel_output_polarity_config	1157
表 3-1648. 函数 timer_channel_complementary_output_polarity_config	1158
表 3-1649. 函数 timer_channel_output_state_config	1159
表 3-1650. 函数 timer_channel_complementary_output_state_config	1160
表 3-1651. 函数 timer_channel_input_struct_para_init	1161
表 3-1652. 函数 timer_input_capture_config	1161
表 3-1653. 函数 timer_channel_input_capture_prescaler_config	1162
表 3-1654. 函数 timer_channel_capture_value_register_read	1163
表 3-1655. 函数 timer_input_pwm_capture_config	1164
表 3-1656. 函数 timer_hall_mode_config	1165
表 3-1657. 函数 timer_multi_mode_channel_output_parameter_struct_init	1166
表 3-1658. 函数 timer_multi_mode_channel_output_config	1167
表 3-1659. 函数 timer_multi_mode_channel_mode_config	1167
表 3-1660. 函数 timer_input_trigger_source_select	1168
表 3-1661. 函数 timer_master_output0_trigger_source_select	1170
表 3-1662. 函数 timer_master_output1_trigger_source_select	1171
表 3-1663. 函数 timer_slave_mode_select	1172
表 3-1664. 函数 timer_master_slave_mode_config	1173
表 3-1665. 函数 timer_external_trigger_config	1174
表 3-1666. 函数 timer_quadrature_decoder_mode_config	1175
表 3-1667. 函数 timer_non_quadrature_decoder_mode_config	1176
表 3-1668. 函数 timer_internal_clock_config	1177
表 3-1669. 函数 timer_internal_trigger_as_external_clock_config	1177
表 3-1670. 函数 timer_external_trigger_as_external_clock_config	1179
表 3-1671. 函数 timer_external_clock_mode0_config	1180
表 3-1672. 函数 timer_external_clock_mode1_config	1181

表 3-1673. 函数 timer_external_clock_mode1_disable.....	1182
表 3-1674. 函数 timer_write_chxval_register_config	1182
表 3-1675. 函数 timer_output_value_selection_config.....	1183
表 3-1676. 函数 timer_commutation_control_shadow_register_config	1184
表 3-1677. 函数 timer_output_match_pulse_select.....	1185
表 3-1678. 函数 timer_channel_composite_pwm_mode_config.....	1186
表 3-1679. 函数 timer_channel_composite_pwm_mode_output_pulse_value_config	1186
表 3-1680. 函数 timer_channel_additional_compare_value_config	1187
表 3-1681. 函数 timer_channel_additional_output_shadow_config.....	1188
表 3-1682. 函数 timer_channel_additional_compare_value_read	1189
表 3-1683. 函数 timer_break_external_source_config.....	1190
表 3-1684. 函数 timer_break_external_polarity_config	1190
表 3-1685. 函数 timer_break_lock_config	1191
表 3-1686. 函数 timer_break_lock_release_config	1192
表 3-1687. 函数 timer_channel_break_control_config.....	1193
表 3-1688. 函数 timer_channel_dead_time_config.....	1194
表 3-1689. 函数 timer_free_complementary_struct_para_init.....	1194
表 3-1690. 函数 timer_channel_free_complementary_config	1195
表 3-1691. 函数 timer_watchdog_value_config.....	1196
表 3-1692. 函数 timer_watchdog_value_read	1197
表 3-1693. 函数 timer_decoder_disconnection_detection_config.....	1197
表 3-1694. 函数 timer_decoder_jump_detection_config	1198
表 3-1695. 函数 timer_upif_backup_config.....	1198
表 3-1696. 函数 timer_upifbu_bit_get	1199
表 3-1697. 函数 timer_flag_get.....	1200
表 3-1698. 函数 timer_flag_clear.....	1202
表 3-1699. 函数 timer_interrupt_enable.....	1203
表 3-1700. 函数 timer_interrupt_disable.....	1205
表 3-1701. 函数 timer_interrupt_flag_get	1206
表 3-1702. 函数 timer_interrupt_flag_clear	1207
表 3-1703. TLI 寄存器	1210
表 3-1704. TLI 库函数	1211
表 3-1705. 结构体 tli_parameter_struct.....	1211
表 3-1706. 结构体 tli_layer_parameter_struct.....	1212
表 3-1707. 结构体 tli_layer_lut_parameter_struct.....	1212
表 3-1708. 函数 tli_deinit	1213
表 3-1709. 函数 tli_struct_para_init.....	1213
表 3-1710. 函数 tli_init.....	1214
表 3-1711. 函数 tli_dither_config.....	1215
表 3-1712. 函数 tli_enable	1215
表 3-1713. 函数 tli_disable	1216
表 3-1714. 函数 tli_reload_config.....	1216
表 3-1715. 函数 tli_layer_struct_para_init.....	1217

表 3-1716. 函数 tli_layer_init.....	1217
表 3-1717. 函数 tli_layer_window_offset_modify.....	1219
表 3-1718. 函数 tli_lut_struct_para_init	1219
表 3-1719. 函数 tli_lut_init.....	1220
表 3-1720. 函数 tli_color_key_init.....	1221
表 3-1721. 函数 tli_layer_enable	1221
表 3-1722. 函数 tli_layer_disable	1222
表 3-1723. 函数 tli_color_key_enable.....	1222
表 3-1724. 函数 tli_color_key_disable	1223
表 3-1725. 函数 tli_lut_enable.....	1224
表 3-1726. 函数 tli_lut_disable.....	1224
表 3-1727. 函数 tli_line_mark_set	1225
表 3-1728. 函数 tli_current_pos_get.....	1225
表 3-1729. 函数 tli_interrupt_enable.....	1226
表 3-1730. 函数 tli_interrupt_disable.....	1226
表 3-1731. 函数 tli_interrupt_flag_get	1227
表 3-1732. 函数 tli_interrupt_flag_clear	1227
表 3-1733. 函数 tli_flag_get.....	1228
表 3-1734. TMU 寄存器.....	1229
表 3-1735. TMU 库函数.....	1229
表 3-1736. 结构体 tmu_parameter_struct	1230
表 3-1737. 函数 tmu_deinit.....	1230
表 3-1738. 函数 tmu_struct_para_init	1231
表 3-1739. 函数 tmu_init.....	1231
表 3-1740. 函数 tmu_read_interrupt_enable	1232
表 3-1741. 函数 tmu_read_interrupt_disable	1233
表 3-1742. 函数 tmu_dma_read_enable	1233
表 3-1743. 函数 tmu_dma_read_disable.....	1234
表 3-1744. 函数 tmu_dma_write_enable	1234
表 3-1745. 函数 tmu_dma_write_disable	1235
表 3-1746. 函数 tmu_one_q31_write	1235
表 3-1747. 函数 tmu_two_q31_write.....	1236
表 3-1748. 函数 tmu_two_q15_write.....	1236
表 3-1749. 函数 tmu_one_q31_read	1237
表 3-1750. 函数 tmu_two_q31_read.....	1237
表 3-1751. 函数 tmu_two_q15_read.....	1238
表 3-1752. TRIGSEL 寄存器	1238
表 3-1753. TRIGSEL 库函数	1240
表 3-1754. 枚举类型 trigselsource_enum	1240
表 3-1755. 枚举类型 trigselperiph_enum	1244
表 3-1756. 函数 trigsels_init.....	1245
表 3-1757. 函数 trigsels_init.....	1246
表 3-1758. 函数 trigsels_trigger_source_get	1247

表 3-1759. 函数 trngsel_trigger_source_set	1247
表 3-1760. 函数 trngsel_trigger_lock_get	1248
表 3-1761. TRNG 寄存器	1248
表 3-1762. TRNG 库函数	1249
表 3-1763. 枚举 trng_inmod_enum	1249
表 3-1764. 枚举 trng_outmod_enum	1249
表 3-1765. 枚举 trng_modsel_enum	1250
表 3-1766. 枚举 trng_flag_enum	1250
表 3-1767. 枚举 trng_int_flag_enum	1250
表 3-1768. 函数 trng_deinit	1250
表 3-1769. 函数 trng_enable	1251
表 3-1770. 函数 trng_disable	1251
表 3-1771. 函数 trng_lock	1252
表 3-1772. 函数 trng_mode_config	1252
表 3-1773. 函数 trng_postprocessing_enable	1253
表 3-1774. 函数 trng_postprocessing_disable	1254
表 3-1775. 函数 trng_conditioning_enable	1254
表 3-1776. 函数 trng_conditioning_disable	1255
表 3-1777. 函数 trng_conditioning_input_bitwidth	1256
表 3-1778. 函数 trng_conditioning_output_bitwidth	1257
表 3-1779. 函数 trng_replace_test_enable	1258
表 3-1780. 函数 trng_replace_test_disable	1258
表 3-1781. 函数 trng_hash_init_enable	1259
表 3-1782. 函数 trng_hash_init_disable	1260
表 3-1783. 函数 trng_powermode_config	1260
表 3-1784. 函数 trng_clockdiv_config	1261
表 3-1785. 函数 trng_clockerror_detection_enable	1262
表 3-1786. 函数 trng_clockerror_detection_disable	1262
表 3-1787. 函数 trng_get_true_random_data	1263
表 3-1788. 函数 trng_conditioning_reset_enable	1263
表 3-1789. 函数 trng_conditioning_reset_disable	1264
表 3-1790. 函数 trng_conditioning_algo_config	1265
表 3-1791. 函数 trng_health_tests_config	1266
表 3-1792. 函数 trng_flag_get	1267
表 3-1793. 函数 trng_interrupt_enable	1267
表 3-1794. 函数 trng_interrupt_disable	1268
表 3-1795. 函数 trng_interrupt_flag_get	1268
表 3-1796. 函数 trng_interrupt_flag_clear	1269
表 3-1797. USART 寄存器	1270
表 3-1798. USART 库函数	1270
表 3-1799. 枚举类型 usart_flag_enum	1273
表 3-1800. 枚举类型 usart_interrupt_flag_enum	1273
表 3-1801. 枚举类型 usart_interrupt_enum	1274

表 3-1802. 枚举类型 usart_invert_enum	1275
表 3-1803. 函数 usart_deinit	1275
表 3-1804. 函数 usart_baudrate_set.....	1276
表 3-1805. 函数 usart_parity_config.....	1276
表 3-1806. 函数 usart_word_length_set	1277
表 3-1807. 函数 usart_stop_bit_set	1278
表 3-1808. 函数 usart_enable	1278
表 3-1809. 函数 usart_disable	1279
表 3-1810. 函数 usart_transmit_config	1279
表 3-1811. 函数 usart_receive_config	1280
表 3-1812. 函数 usart_data_first_config.....	1281
表 3-1813. 函数 usart_invert_config.....	1281
表 3-1814. 函数 usart_overrun_enable	1282
表 3-1815. 函数 usart_overrun_disable	1282
表 3-1816. 函数 usart_oversample_config.....	1283
表 3-1817. 函数 usart_sample_bit_config	1284
表 3-1818. 函数 usart_receiver_timeout_enable	1284
表 3-1819. 函数 usart_receiver_timeout_disable	1285
表 3-1820. 函数 usart_receiver_timeout_threshold_config	1285
表 3-1821. 函数 usart_data_transmit.....	1286
表 3-1822. 函数 usart_data_receive.....	1287
表 3-1823. 函数 usart_command_enable	1287
表 3-1824. 函数 usart_address_0_match_mode_enable	1288
表 3-1825. 函数 usart_address_0_match_mode_disable	1288
表 3-1826. 函数 usart_address_1_match_mode_enable	1289
表 3-1827. 函数 usart_address_1_match_mode_disable	1289
表 3-1828. 函数 usart_address_0_config	1290
表 3-1829. 函数 usart_address_1_config	1291
表 3-1830. 函数 usart_address_0_detection_mode_config	1291
表 3-1831. 函数 usart_address_1_detection_mode_config	1292
表 3-1832. 函数 usart_mute_mode_enable	1293
表 3-1833. 函数 usart_mute_mode_disable	1293
表 3-1834. 函数 usart_mute_mode_wakeup_config.....	1294
表 3-1835. 函数 usart_lin_mode_enable.....	1294
表 3-1836. 函数 usart_lin_mode_disable.....	1295
表 3-1837. 函数 usart_lin_break_detection_length_config	1295
表 3-1838. 函数 usart_halfduplex_enable	1296
表 3-1839. 函数 usart_halfduplex_disable	1297
表 3-1840. 函数 usart_clock_enable.....	1297
表 3-1841. 函数 usart_clock_disable.....	1298
表 3-1842. 函数 usart_synchronous_clock_config	1298
表 3-1843. 函数 usart_guard_time_config.....	1299
表 3-1844. 函数 usart_smartcard_mode_enable	1300

表 3-1845. 函数 usart_smartcard_mode_disable	1300
表 3-1846. 函数 usart_smartcard_mode_nack_enable	1301
表 3-1847. 函数 usart_smartcard_mode_nack_disable	1301
表 3-1848. 函数 usart_smartcard_mode_early_nack_enable	1302
表 3-1849. 函数 usart_smartcard_mode_early_nack_disable	1302
表 3-1850. 函数 usart_smartcard_autoretry_config	1303
表 3-1851. 函数 usart_block_length_config	1303
表 3-1852. 函数 usart_irda_mode_enable	1304
表 3-1853. 函数 usart_irda_mode_disable	1304
表 3-1854. 函数 usart_prescaler_config	1305
表 3-1855. 函数 usart_irda_lowpower_config	1305
表 3-1856. 函数 usart_hardware_flow_rts_config	1306
表 3-1857. 函数 usart_hardware_flow_cts_config	1307
表 3-1858. 函数 usart_hardware_flow_coherence_config	1307
表 3-1859. 函数 usart_rs485_driver_enable	1308
表 3-1860. 函数 usart_rs485_driver_disable	1308
表 3-1861. 函数 usart_driver_asserttime_config	1309
表 3-1862. 函数 usart_driver_deasserttime_config	1310
表 3-1863. 函数 usart_depolarity_config	1310
表 3-1864. 函数 usart_dma_receive_config	1311
表 3-1865. 函数 usart_dma_transmit_config	1311
表 3-1866. 函数 usart_reception_error_dma_disable	1312
表 3-1867. 函数 usart_reception_error_dma_enable	1313
表 3-1868. 函数 usart_wakeup_enable	1313
表 3-1869. 函数 usart_wakeup_disable	1314
表 3-1870. 函数 usart_wakeup_mode_config	1314
表 3-1871. 函数 usart_fifo_enable	1315
表 3-1872. 函数 usart_fifo_disable	1315
表 3-1873. 函数 usart_transmit_fifo_threshold_config	1316
表 3-1874. 函数 usart_receive_fifo_threshold_config	1317
表 3-1875. 函数 usart_receive_fifo_counter_number	1318
表 3-1876. 函数 usart_flag_get	1318
表 3-1877. 函数 usart_flag_clear	1319
表 3-1878. 函数 usart_interrupt_enable	1320
表 3-1879. 函数 usart_interrupt_disable	1321
表 3-1880. 函数 usart_interrupt_flag_get	1321
表 3-1881. 函数 usart_interrupt_flag_clear	1322
表 3-1882. VREF 寄存器	1324
表 3-1883. VREF 库函数	1324
表 3-1884. 函数 vref_deinit	1324
表 3-1885. 函数 vref_enable	1325
表 3-1886. 函数 vref_disable	1325
表 3-1887. 函数 vref_high_impedance_mode_enable	1326

表 3-1888. 函数 <code>vref_high_impedance_mode_disable</code>	1326
表 3-1889. 函数 <code>vref_status_get</code>	1327
表 3-1890. 函数 <code>vref_voltage_select</code>	1327
表 3-1891. 函数 <code>vref_calib_value_set</code>	1328
表 3-1892. 函数 <code>vref_calib_value_get</code>	1328
表 3-1893. WWDGT 寄存器.....	1329
表 3-1894. WWDGT 库函数.....	1329
表 3-1895. 函数 <code>wwdgt_deinit</code>	1329
表 3-1896. 函数 <code>wwdgt_enable</code>	1330
表 3-1897. 函数 <code>wwdgt_counter_update</code>	1330
表 3-1898. 函数 <code>wwdgt_config</code>	1331
表 3-1899. 函数 <code>wwdgt_interrupt_enable</code>	1332
表 3-1900. 函数 <code>wwdgt_flag_get</code>	1332
表 3-1901. 函数 <code>wwdgt_flag_clear</code>	1333
表 4-1. 版本历史	1334

1. 介绍

本手册介绍了32位基于ARM微控制器GD32H7xx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GD32H7xx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
CAU	加密处理器
CMP	比较器
CPDM	时钟相位延迟模块
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
DCI	数字摄像头接口
DMA	直接存储器访问控制器

外设缩写	说明
DMAMUX	DMA请求多路复用器
EDOUT	编码器分频输出控制器
EFUSE	熔丝
ENET	以太网
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
HAU	哈希处理器
HPDF	高性能数字滤波器
HWSEM	硬件信号量
I2C	内部集成电路总线接口
IPA	图像处理加速器
LPDTS	低功耗数字温度传感器
MDIO	管理数据输入 / 输出接口
MDMA	主机直接存储器访问控制器
OSPI	八线SPI接口
OSPIM	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
RSPDIF	S/PD数字音频接口接收器
RTC	实时时钟
RTDEC	实时解密
SAI	串行音频接口
SDIO	SDIO接口
SPI/I2S	串行外设接口/片上音频接口
SYSCFG	系统配置
TIMER	定时器
TLI	TFT-LCD接口
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

1.1.2. 命名规则

固件库遵从以下命名规则：

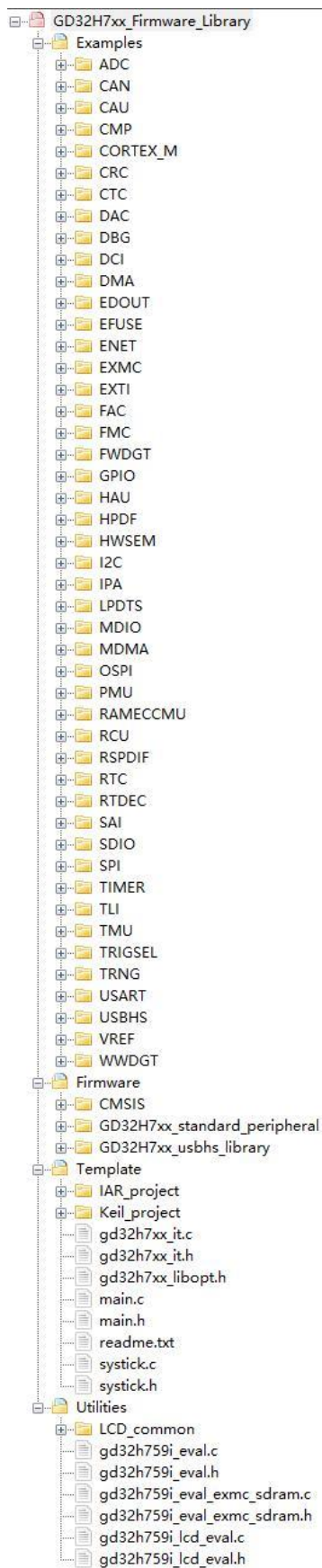
- XXX表示任一外设缩写，例如：ADC。更多缩写相关信息参阅[外设缩写](#)；
- 源文件和头文件命名都以“gd32h7xx_”作为开头，例如：gd32h7xx_adc.h；
- 常量仅被应用于一个文件的，定义于该文件中；被应用于多个文件的，在对应头文件中定义。所有常量都由英文字母大写书写；
- 寄存器作为常量处理。他们的命名都由英文字母大写书写。在大多数情况下，寄存器缩写规范与本用户手册一致；
- 变量名采用全部小写，有多个单词组成的，在单词之间以下划线分隔；
- 外设函数的命名以该外设的缩写加下划线为开头，有多个单词组成的，在单词之间以下划线分隔，所有外设函数都由英文字母小写书写。

2. 固件库概述

2.1. 文件组织结构

GD32H7xx_Firmware_Library，文件组织结构见下图：

图 2-1. GD32H7xx 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；
- **gd32h7xx_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）；
- **gd32h7xx_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）；
- **gd32h7xx_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**GD32H7xx**的全局头文件和系统配置文件；
- **GD32H7xx_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。

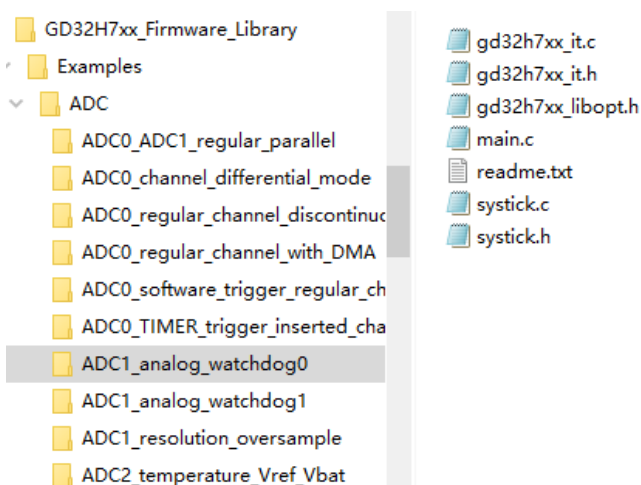
2.1.3. Template 文件夹

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：

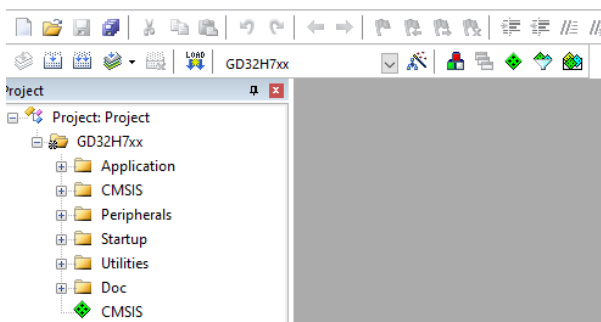
图 2-3. 拷贝外设例程文件



打开工程

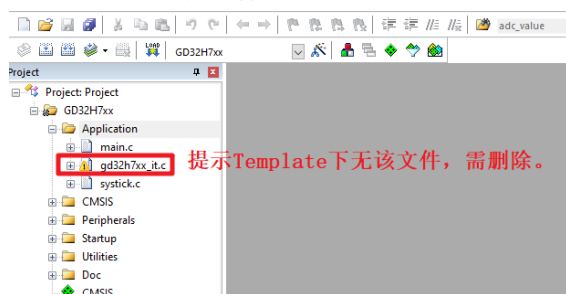
GD 提供 Keil 和 IAR 两种版本的工程，根据客户所安装的软件，打开不同的 project，如“Keil_project”，打开\Template\Keil_project\Project.uvprojx，如下图所示：

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：

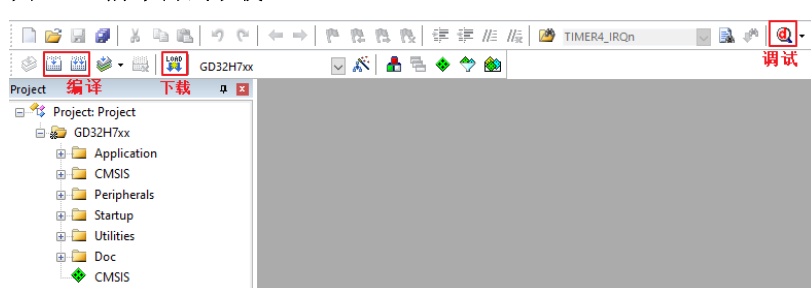
图 2-5. 配置工程文件



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
gd32h7xx_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。
gd32h7xx_it.h	头文件，包含所有中断处理函数原形。
gd32h7xx_it.c	外设中断函数文件。用户可以加入自己的中断程序代码。对于指向同一个中断向量的多个不同中断请求，可以利用函数通过判断外设的中断标志位来确定准

	确的中断源。固件库提供了这些函数的名称。
gd32h7xx_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。
gd32h7xx_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
ADCx(x=0,1,2)	ADC外设选择
输入参数{in}	
data_alignment	数据对齐方式选择
ADC_DATAALIGN_RIGHT	最低有效位对齐
ADC_DATAALIGN_LEFT	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
ADCx(x=0,1,2)	ADC外设选择
输出参数{out}	
-	-
返回值	
-	-

例如:

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

```
adc_enable(ADC0);
```

函数 adc_disable

函数adc_disable描述见下表:

表 3-9. 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	adc_inserted_channel_offset_config
函数原形	void adc_inserted_channel_offset_config(uint32_t adc_periph, uint8_t inserted_channel, uint32_t offset);
功能描述	配置ADC注入通道组数据偏移值
先决条件	-
被调用函数	-
输入参数{in}	
adc_periph	ADC外设
ADCx(x=0,1,2)	ADC外设选择
输入参数{in}	
inserted_channel	注入通道选择
ADC_INSERTED_CHANNEL_x(x=0..3)	注入通道，x=0,1,2,3
输入参数{in}	
offset	数据偏移值
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_CHANNELx (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
ADCx(x=0,1,2)	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
ADCx(x=0,1,2)	ADC外设选择
输入参数{in}	
selection_channel	选择ADC通道
ADC_AWD1_2_SELECTION_CHANNEL_x (x=0..20), ADC_AWD1_2_SELECTION_CHANNEL_ALL	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
ENABLE	使能
DISABLE	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..0xFFFFF, 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..0xFFFFF, 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..0xFFFFF, 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..0xFFFFF, 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	<code>adc_watchdog2_threshold_config</code>
函数原形	<code>void adc_watchdog2_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2阈值
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}	
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	<code>adc_oversample_mode_config</code>
函数原形	<code>void adc_oversample_mode_config(uint32_t adc_periph, uint32_t mode, uint16_t shift, uint16_t ratio);</code>
功能描述	配置ADC过采样模式

先决条件	-
被调用函数	-
输入参数{in}	
adc_periph	ADC外设
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_CONVERSION	在一个触发之后，对一个通道连续进行过采样转换
ADC_OVERSAMPLING_ONE_CONVERSION	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	<code>adc_oversample_mode_enable</code>
函数原形	<code>void adc_oversample_mode_enable(uint32_t adc_periph);</code>
功能描述	使能ADC过采样
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	
-	-

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	<code>adc_oversample_mode_disable</code>
函数原形	<code>void adc_oversample_mode_disable(uint32_t adc_periph);</code>
功能描述	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	<code>adc_interrupt_disable</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	<code>adc_interrupt_flag_get</code>
函数原形	<code>FlagStatus adc_interrupt_flag_get(uint32_t adc_periph, uint32_t int_flag);</code>
功能描述	获取ADC中断标志位
先决条件	-
被调用函数	-
输入参数{in}	
adc_periph	ADC外设
<code>ADCx(x=0,1,2)</code>	ADC外设选择
输入参数{in}	
int_flag	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	
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	adc_sync_regular_data1_read
函数原形	uint32_t adc_sync_regular_data1_read(void);
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	
uint32_t	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
CAN_CTL0	控制寄存器 0
CAN_CTL1	控制寄存器 1
CAN_TIMER	计数器寄存器
CAN_RMPUBF	接收邮箱公有过滤寄存器
CAN_ERR0	错误寄存器 0
CAN_ERR1	错误寄存器 1
CAN_INTEN	中断使能寄存器
CAN_STAT	状态寄存器
CAN_CTL2	控制寄存器 2
CAN_CRCC	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_id_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	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
timeout_int	使能或禁能超时唤醒中断
match_int	使能或禁能匹配唤醒中断
num_matches	设置消息匹配次数
match_timeout	设置超时唤醒时间值
frame_filter	设置帧的过滤类型
id_filter	设置 ID 域的过滤类型
data_filter	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
classical_frm_mb_number	发送了 CRC 值为 CRCTC[14:0]的邮箱的编号
classical_frm_transmitted_crc	最新发送的常规帧的 CRC 计算值
classical_fd_frm_mb_number	发送常规帧或者 FD 帧时, CRC 值为 CRCTC[20:0]的邮箱的编号
classical_fd_frm_transmitted_crc	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
CAN_INT_RX_WARNING	Rx 错误警告中断
CAN_INT_TX_WARNING	Tx 错误警告中断
CAN_INT_ERR_SUMMARY	错误汇总中断
CAN_INT_BUSOFF	离线中断
CAN_INT_BUSOFF_RECOVERY	离线恢复中断
CAN_INT_ERR_SUMMARY_FD	FD 帧数据位时间的错误汇总中断
CAN_INT_MB0	邮箱 0 成功发送或接收帧中断
CAN_INT_MB1	邮箱 1 成功发送或接收帧中断
CAN_INT_MB2	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_ERR</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	
<code>ErrStatus</code>	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	
<code>ErrStatus</code>	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
CANx(x=0,1,2)	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
CANx(x=0,1,2)	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. CAU

加密处理单元支持处理DES，三重DES或AES（128，192或256）算法，对数据进行加密或解密。CAU寄存器列举在章节[3.4.1](#)，CAU固件库函数介绍在章节[3.4.2](#)。

3.4.1. 外设寄存器说明

CAU寄存器列表如下表所示：

表 3-134. CAU 寄存器

寄存器名称	寄存器描述
CAU_CTL	CAU控制寄存器
CAU_STAT0	CAU状态寄存器0
CAU_DI	CAU数据输入寄存器
CAU_DO	CAU数据输出寄存器
CAU_DMAEN	CAU DMA使能寄存器
CAU_INTEN	CAU中断使能寄存器
CAU_STAT1	CAU状态寄存器1
CAU_INTF	CAU中断标志寄存器
CAU_KEY0H	CAU密钥0高位寄存器
CAU_KEY0L	CAU密钥0低位寄存器
CAU_KEY1H	CAU密钥1高位寄存器
CAU_KEY1L	CAU密钥1低位寄存器
CAU_KEY2H	CAU密钥2高位寄存器
CAU_KEY2L	CAU密钥2低位寄存器
CAU_KEY3H	CAU密钥3高位寄存器
CAU_KEY3L	CAU密钥3低位寄存器
CAU_IV0H	CAU初始化向量0高位寄存器
CAU_IV0L	CAU初始化向量0低位寄存器
CAU_IV1H	CAU初始化向量1高位寄存器
CAU_IV1L	CAU初始化向量1低位寄存器
CAU_GCMCCMCT XSx (x = 0..7)	GCM或CCM模式上下文交换寄存器x
CAU_GCMCTXSx (x = 0..7)	GCM模式上下文交换寄存器x

3.4.2. 外设库函数说明

CAU库函数列表如下表所示：

表 3-135. CAU 库函数

库函数名称	库函数描述
cau_deinit	复位CAU外设
cau_struct_para_init	初始化CAU加密解密结构体
cau_key_parameter_init	初始化密钥结构体
cau_iv_parameter_init	初始化矢量结构体
cau_context_struct_para_init	初始化上下文结构体
cau_enable	使能CAU外设
cau_disable	除能CAU外设
cau_dma_enable	使能CAU DMA接口
cau_dma_disable	除能CAU DMA接口
cau_init	初始化CAU
cau_aes_key_select	配置AES密钥来源选项
cau_aes_keysize_config	在使用AES算法的情况下配置密钥大小
cau_key_init	初始化密钥参数
cau_iv_init	初始化矢量参数
cau_phase_config	阶段配置
cau_fifo_flush	清除FIFO内容
cau_enable_state_get	返回CAU外设是否使能的状态值
cau_data_write	将数据写入IN FIFO
cau_data_read	返回最近进入OUT FIFO的数据
cau_context_save	上下文交换之前保存上下文
cau_context_restore	上下文交换之后恢复上下文
cau_aes_ecb	ECB模式下使用AES算法加密和解密
cau_aes_cbc	CBC模式下使用AES算法加密和解密
cau_aes_ctr	CTR模式下使用AES算法加密和解密
cau_aes_cfb	CFB模式下使用AES算法加密和解密
cau_aes_ofb	OFB模式下使用AES算法加密和解密
cau_aes_gcm	GCM模式下使用AES算法加密和解密
cau_aes_ccm	CCM模式下使用AES算法加密和解密
cau_tdes_ecb	ECB模式下使用TDES算法加密和解密
cau_tdes_cbc	CBC模式下使用TDES算法加密和解密
cau_des_ecb	ECB模式下使用DES算法加密和解密
cau_des_cbc	CBC模式下使用DES算法加密和解密
cau_flag_get	获取CAU标志状态
cau_interrupt_enable	使能CAU中断
cau_interrupt_disable	除能CAU中断
cau_interrupt_flag_get	获取中断标志

结构体 cau_key_parameter_struct

表 3-136. 结构体 cau_key_parameter_struct

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

结构体 cau_iv_parameter_struct

表 3-137. 结构体 cau_iv_parameter_struct

成员名称	功能描述
iv_0_high	矢量0高位
iv_0_low	矢量0低位
iv_1_high	矢量1高位
iv_1_low	矢量1低位

结构体 cau_context_parameter_struct

表 3-138. 结构体 cau_context_parameter_struct

成员名称	功能描述
ctl_config	当前配置
iv_0_high	矢量0高位
iv_0_low	矢量0低位
iv_1_high	矢量1高位
iv_1_low	矢量1低位
key_0_high	密钥0高位
key_0_low	密钥0低位
key_1_high	密钥1高位
key_1_low	密钥1低位
key_2_high	密钥2高位
key_2_low	密钥2低位
key_3_high	密钥3高位
key_3_low	密钥3低位
gcmccmctxs[8]	GCM或CCM模式上下文
gcmctxs[8]	GCM模式上下文

结构体 cau_parameter_struct

表 3-139. 结构体 cau_parameter_struct

成员名称	功能描述
alg_dir	算法方向
*key	密钥
key_size	密钥字节长度
*iv	初始化矢量
iv_size	初始化矢量字节长度
*input	输入数据
in_length	输入数据字节长度
*aad	附加身份验证数据
aad_size	附加身份验证数据长度

函数 cau_deinit

函数cau_deinit描述见下表：

表 3-140. 函数 cau_deinit

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

例如：

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

```
cau_deinit();
```

函数 cau_struct_para_init

函数cau_struct_para_init描述见下表：

表 3-141. 函数 cau_struct_para_init

函数名称	cau_struct_para_init
函数原形	void cau_struct_para_init(cau_parameter_struct *cau_parameter)
功能描述	初始化CAU加密解密结构体
先决条件	-

被调用函数	-
输入参数{in}	
-	-
输出参数{out}	
cau_parameter	参考结构体 表3-139. 结构体cau_parameter_struct
返回值	
-	-

例如：

```
cau_parameter_struct text;
```

```
/* initialize CAU encrypt and decrypt parameter struct */
```

```
cau_struct_para_init(&text);
```

函数 cau_key_struct_para_init

The description of cau_key_struct_para_init描述见下表：

表 3-142. 函数 cau_key_struct_para_init

函数名称	cau_key_struct_para_init
函数原形	void cau_key_struct_para_init(cau_key_parameter_struct *key_initpara)
功能描述	初始化密钥结构体
先决条件	-
被调用函数	-
输入参数{in}	
-	-
输出参数{out}	
key_initpara	参考结构体 表3-136. 结构体cau_key_parameter_struct
返回值	
-	-

例如：

```
/* initialize the key parameter struct */
```

```
cau_key_parameter_struct key_initpara;
```

```
cau_key_struct_para_init(&key_initpara);
```

函数 cau_iv_struct_para_init

函数cau_iv_struct_para_init描述见下表：

表 3-143. 函数 cau_iv_struct_para_init

函数名称	cau_iv_struct_para_init
函数原形	void cau_iv_struct_para_init(cau_iv_parameter_struct *iv_initpara)

功能描述	初始化矢量结构体
先决条件	-
被调用函数	-
输入参数{in}	
-	-
输出参数{out}	
iv_initpara	参考结构体 表3-137. 结构体cau iv parameter struct
返回值	
-	-

例如：

```
/* initialize the vectors parameter struct */
```

```
cau_iv_parameter_struct iv_initpara;
```

```
cau_iv_struct_para_init(&iv_initpara);
```

函数 cau_context_struct_para_init

函数cau_context_struct_para_init描述见下表：

表 3-144. 函数 cau_context_struct_para_init

函数名称	cau_context_struct_para_init
函数原形	void cau_context_struct_para_init(cau_context_parameter_struct *cau_context)
功能描述	初始化上下文结构体
先决条件	-
被调用函数	-
输入参数{in}	
-	-
输出参数{out}	
cau_context	参考结构体 表3-138. 结构体cau_context parameter struct
返回值	
-	-

例如：

```
/* initialize the context parameter struct */
```

```
cau_context_parameter_struct context_initpara;
```

```
cau_context_struct_para_init (&context_initpara);
```

函数 cau_enable

函数cau_enable描述见下表：

表 3-145. 函数 cau_enable

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

例如：

```
/* enable the CAU peripheral */
cau_enable();
```

函数 cau_disable

函数cau_disable描述见下表：

表 3-146. 函数 cau_disable

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

例如：

```
/* disable the CAU peripheral */
cau_disable();
```

函数 cau_dma_enable

函数cau_dma_enable描述见下表：

表 3-147. 函数 cau_dma_enable

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

例如：

```
/* enable the CAU DMA interface */
cau_dma_enable(CAU_DMA_INFIFO);
```

函数 cau_dma_disable

函数cau_dma_disable描述见下表：

表 3-148. 函数 cau_dma_disable

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

例如：

```
/* disable the CAU DMA interface */
cau_dma_disable(CAU_DMA_INFIFO);
```


函数 `cau_init`

函数 `cau_init` 描述见下表:

表 3-149. 函数 `cau_init`

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

<i>T</i>	
CAU_SWAPPING_16BIT	半字交换
<i>T</i>	
CAU_SWAPPING_8BIT	字节交换
CAU_SWAPPING_1BIT	位交换
输出参数{out}	
-	-
返回值	
-	-

例如：

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

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

函数 cau_aes_key_select

函数cau_aes_key_select描述见下表：

表 3-150. 函数 cau_aes_key_select

函数名称	cau_aes_key_select
函数原形	void cau_aes_key_select(uint32_t key_selection);
功能描述	配置AES密钥来源选项
先决条件	-
被调用函数	-
输入参数{in}	
key_selection	密钥来源选项
CAU_KEY	使用来自于CAU寄存器中的密钥
CAU_EFUSE_KEY	使用来自于EFUSE中的密钥
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* configure the key source as CAU registers */
```

```
cau_aes_key_select(CAU_KEY);
```

函数 cau_aes_keysize_config

函数cau_aes_keysize_config描述见下表：

表 3-151. 函数 cau_aes_keysize_config

函数名称	cau_aes_keysize_config
------	------------------------

函数原形	void cau_aes_keysize_config(uint32_t key_size);
功能描述	在使用AES算法的情况下配置密钥大小
先决条件	-
被调用函数	-
输入参数{in}	
key_size	密钥长度
CAU_KEYSIZE_128BIT	128位密钥长度
CAU_KEYSIZE_192BIT	192位密钥长度
CAU_KEYSIZE_256BIT	256位密钥长度
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* configure key size if used AES algorithm */
cau_aes_keysize_config (CAU_KEYSIZE_128BIT);
```

函数 cau_key_init

函数cau_key_init描述见下表：

表 3-152. 函数 cau_key_init

函数名称	cau_key_init
函数原形	void cau_key_init(cau_key_parameter_struct* key_initpara);
功能描述	初始化密钥参数
先决条件	-
被调用函数	-
输入参数{in}	
key_initpara	参考结构体 表3-136. 结构体cau_key_parameter_struct
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* initialize the key parameters */
cau_key_parameter_struct key_initpara;
cau_key_init(&key_initpara);
```

函数 cau_iv_init

函数cau_iv_init描述见下表:

表 3-153. 函数 cau_iv_init

函数名称	cau_iv_init
函数原形	void cau_iv_init(cau_iv_parameter_struct* iv_initpara);
功能描述	初始化矢量参数
先决条件	-
被调用函数	-
输入参数{in}	
iv_initpara	参考结构体 表3-137. 结构体cau_iv_parameter_struct
输出参数{out}	
-	-
返回值	
-	-

例如:

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

函数 cau_phase_config

函数cau_phase_config描述见下表:

表 3-154. 函数 cau_phase_config

函数名称	cau_phase_config
函数原形	void cau_phase_config(uint32_t phase)
功能描述	阶段配置
先决条件	-
被调用函数	-
输入参数{in}	
phase	GCM或CCM阶段
CAU_PREPARE_PHASE	准备阶段
CAU_AAD_PHASE	附加身份验证数据阶段
CAU_ENCRYPT_DECRYPT_PHASE	加密解密阶段
CAU_TAG_PHASE	标签阶段
输出参数{out}	
-	-
返回值	

-	-
---	---

例如:

```
/* select prepare phase */
```

```
cau_phase_config(CAU_PREPARE_PHASE);
```

函数 cau_fifo_flush

函数cau_fifo_flush描述见下表:

表 3-155. 函数 cau_fifo_flush

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

例如:

```
/* flush the IN and OUT FIFOs */
```

```
cau_fifo_flush();
```

函数 cau_enable_state_get

函数cau_enable_state_get描述见下表:

表 3-156. 函数 cau_enable_state_get

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

例如:

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

```
ControlStatus state = DISABLE;
```

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

函数 cau_data_write

函数cau_data_write描述见下表:

表 3-157. 函数 cau_data_write

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

例如:

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

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

函数 cau_data_read

函数cau_data_read描述见下表:

表 3-158. 函数 cau_data_read

函数名称	cau_data_read
函数原形	uint32_t cau_data_read(void);
功能描述	返回最近进入OUT FIFO的数据
先决条件	-
被调用函数	-
输入参数{in}	
-	-
输出参数{out}	
-	-
返回值	
uint32_t	0x0 – 0xFFFFFFFF

例如:

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

```
uint32_t data = 0;
```

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

函数 `cau_context_save`

函数 `cau_context_save` 描述见下表:

表 3-159. 函数 `cau_context_save`

函数名称	<code>cau_context_save</code>
函数原形	<code>void cau_context_save(cau_context_parameter_struct *cau_context, cau_key_parameter_struct* key_initpara)</code>
功能描述	上下文交换之前保存上下文
先决条件	-
被调用函数	-
输入参数{in}	
<code>key_initpara</code>	参考结构体 表3-136. 结构体 <code>cau_key_parameter_struct</code>
输出参数{out}	
<code>cau_context</code>	参考结构体 表3-138. 结构体 <code>cau_context_parameter_struct</code>
返回值	
-	-

例如:

```
cau_context_parameter_struct context;
```

```
cau_key_parameter_struct key;
```

```
cau_parameter_struct cau_parameter;
```

```
uint32_t keyaddr;
```

```
.....
```

```
keyaddr = (uint32_t)(cau_parameter->key);
```

```
cau_key_struct_para_init(&key);
```

```
key.key_1_high = __REV(*(uint32_t*)(keyaddr));
```

```
keyaddr += 4U;
```

```
key.key_1_low = __REV(*(uint32_t*)(keyaddr));
```

```
/* save context before context switching */
```

```
cau_context_save(&context, &key);
```

函数 **cau_context_restore**

函数cau_context_restore描述见下表:

表 3-160. 函数 **cau_context_restore**

函数名称	cau_context_restore
函数原形	void cau_context_restore(cau_context_parameter_struct *cau_context);
功能描述	上下文交换之后恢复上下文
先决条件	-
被调用函数	-
输入参数{in}	
cau_context	参考结构体 表3-138. 结构体cau_context_parameter_struct
输出参数{out}	
-	-
返回值	
-	-

例如:

```
cau_context_parameter_struct context;

.....

cau_context_save(&context, &key);

.....

/* restore context after context switching */

cau_context_restore (&context);
```

函数 **cau_aes_ecb**

函数cau_aes_ecb描述见下表:

表 3-161. 函数 **cau_aes_ecb**

函数名称	cau_aes_ecb
函数原形	ErrStatus cau_aes_ecb(cau_parameter_struct *cau_parameter, uint8_t *output);
功能描述	ECB模式下使用AES算法加密和解密
先决条件	-
被调用函数	-
输入参数{in}	
cau_parameter	参考结构体 表3-139. 结构体cau_parameter_struct
输出参数{out}	
output	指针指向返回数组
返回值	
ErrStatus	SUCCESS或ERROR

例如:

```
cau_parameter_struct text;

uint8_t encrypt_result[TEXT_SIZE];

ErrStatus status;

.....

key_addr = key_select[i];

key_size = keysize[i];

text.alg_dir    = CAU_ENCRYPT;

text.key        = key_addr;

text.key_size   = key_size;

text.input      = plaintext;

text.in_length  = TEXT_SIZE;

/* encryption in ECB mode */

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

函数 cau_aes_cbc

函数cau_aes_cbc描述见下表:

表 3-162. 函数 cau_aes_cbc

函数名称	cau_aes_cbc
函数原形	ErrStatus cau_aes_cbc(cau_parameter_struct *cau_parameter, uint8_t *output)
功能描述	CBC模式下使用AES算法加密和解密
先决条件	-
被调用函数	-
输入参数{in}	
algo_dir	算法方向
CAU_ENCRYPT	加密
CAU_DECRYPT	解密
输入参数{in}	
cau_parameter	参考结构体 表3-139. 结构体cau_parameter_struct
输出参数{out}	
output	指针指向返回数组
返回值	
ErrStatus	SUCCESS或ERROR

例如:

```

cau_parameter_struct text;

uint8_t encrypt_result[TEXT_SIZE];

ErrStatus status;

.....

key_addr = key_select[i];

key_size = keysize[i];

text.alg_dir    = CAU_ENCRYPT;

text.key        = key_addr;

text.key_size   = key_size;

text.iv         = vectors;

text.input      = plaintext;

text.in_length  = TEXT_SIZE;

/* encryption in CBC mode */

status = cau_aes_cbc(&text, encrypt_result);

```

函数 cau_aes_ctr

函数cau_aes_ctr描述见下表：

表 3-163. 函数 cau_aes_ctr

函数名称	cau_aes_ctr
函数原形	ErrStatus cau_aes_ctr(cau_parameter_struct *cau_parameter, uint8_t *output)
功能描述	CTR模式下使用AES算法加密和解密
先决条件	-
被调用函数	-
输入参数{in}	
cau_parameter	参考结构体 表3-139. 结构体cau_parameter_struct
输出参数{out}	
output	指针指向返回数组
返回值	
ErrStatus	SUCCESS或ERROR

例如：

```

cau_parameter_struct text;

uint8_t encrypt_result[TEXT_SIZE];

```

```

ErrStatus status;

.....

key_addr = key_select[i];

key_size = keysize[i];

text.alg_dir    = CAU_ENCRYPT;

text.key        = key_addr;

text.key_size   = key_size;

text.iv         = vectors;

text.input      = plaintext;

text.in_length  = TEXT_SIZE;

/* encryption in CTR mode */

status = cau_aes_ctr(&text, encrypt_result);

```

函数 cau_aes_cfb

函数cau_aes_cfb描述见下表：

表 3-164. 函数 cau_aes_cfb

函数名称	cau_aes_cfb
函数原形	ErrStatus cau_aes_cfb(cau_parameter_struct *cau_parameter, uint8_t *output)
功能描述	CFB模式下使用AES算法加密和解密
先决条件	-
被调用函数	-
输入参数{in}	
cau_parameter	参考结构体 表3-139. 结构体cau_parameter_struct
输出参数{out}	
output	指针指向返回数组
返回值	
ErrStatus	SUCCESS或ERROR

例如：

```

cau_parameter_struct cau_cfb_parameter;

uint8_t encrypt_result[TEXT_SIZE];

ErrStatus status;

.....

```

```
/* encryption in CFB mode */

cau_cfb_parameter.alg_dir   = CAU_ENCRYPT;

cau_cfb_parameter.key       = (uint8_t *)key_128;

cau_cfb_parameter.key_size  = KEY_SIZE;

cau_cfb_parameter.iv        = (uint8_t *)vectors;

cau_cfb_parameter.iv_size   = IV_SIZE;

cau_cfb_parameter.input     = (uint8_t *)plaintext;

cau_cfb_parameter.in_length = PLAINTEXT_SIZE;

status = cau_aes_cfb(&cau_cfb_parameter, encrypt_result);
```

函数 cau_aes_ofb

函数cau_aes_ofb描述见下表:

表 3-165. 函数 cau_aes_ofb

函数名称	cau_aes_ofb
函数原形	ErrStatus cau_aes_ofb(cau_parameter_struct *cau_parameter, uint8_t *output)
功能描述	OFB模式下使用AES算法加密和解密
先决条件	-
被调用函数	-
输入参数{in}	
cau_parameter	参考结构体 表3-139. 结构体cau_parameter_struct
输出参数{out}	
output	指针指向返回数组
返回值	
ErrStatus	SUCCESS或ERROR

例如:

```
cau_parameter_struct cau_ofb_parameter;

uint8_t encrypt_result[TEXT_SIZE];

ErrStatus status;

.....

/* encryption in OFB mode */

cau_ofb_parameter.alg_dir   = CAU_ENCRYPT;

cau_ofb_parameter.key       = (uint8_t *)key_128;
```

```

cau_ofb_parameter.key_size = KEY_SIZE;

cau_ofb_parameter.iv       = (uint8_t *)vectors;

cau_ofb_parameter.iv_size  = IV_SIZE;

cau_ofb_parameter.input    = (uint8_t *)plaintext;

cau_ofb_parameter.in_length = PLAINTEXT_SIZE;

status = cau_aes_ofb(&cau_ofb_parameter, encrypt_result);

```

函数 cau_aes_gcm

函数cau_aes_gcm描述见下表:

表 3-166. 函数 cau_aes_gcm

函数名称	cau_aes_gcm
函数原形	ErrStatus cau_aes_gcm(cau_parameter_struct *cau_parameter, uint8_t *output, uint8_t *tag)
功能描述	GCM模式下使用AES算法加密和解密
先决条件	-
被调用函数	-
输入参数{in}	
cau_parameter	参考结构体 表3-139. 结构体cau_parameter_struct
输出参数{out}	
output	指针指向返回数组
输出参数{out}	
tag	指针指向返回标签数组
返回值	
ErrStatus	SUCCESS或ERROR

例如:

```

cau_parameter_struct cau_gcm_parameter;

uint8_t encrypt_result[TEXT_SIZE];

uint8_t gcm_tag[GCM_TAG_SIZE];

ErrStatus status;

.....

/* encryption in GCM mode */

cau_gcm_parameter.alg_dir = CAU_ENCRYPT;

cau_gcm_parameter.key     = (uint8_t *)key_128;

cau_gcm_parameter.key_size = KEY_SIZE;

```

```

cau_gcm_parameter.iv          = (uint8_t *)vectors;

cau_gcm_parameter.iv_size     = IV_SIZE;

cau_gcm_parameter.input       = (uint8_t *)plaintext;

cau_gcm_parameter.in_length   = PLAINTEXT_SIZE;

cau_gcm_parameter.aad         = (uint8_t *)aadmessage;

cau_gcm_parameter.aad_size = AAD_SIZE;

status = cau_aes_gcm(&cau_gcm_parameter, encrypt_result, gcm_tag);

```

函数 cau_aes_ccm

函数cau_aes_ccm描述见下表：

表 3-167. 函数 cau_aes_ccm

函数名称	cau_aes_ccm
函数原形	ErrStatus cau_aes_ccm(cau_parameter_struct *cau_parameter, uint8_t *output, uint8_t tag[], uint32_t tag_size, uint8_t aad_buf[])
功能描述	CCM模式下使用AES算法加密和解密
先决条件	-
被调用函数	-
输入参数{in}	
cau_parameter	参考结构体 表3-139. 结构体cau_parameter_struct
输入参数{in}	
tag_size	标签字节长度
输出参数{out}	
output	指针指向返回数组
输出参数{out}	
tag	指针指向返回标签数组
输出参数{out}	
aad_buf	指针指向用户定义的用于填充附加身份验证数据的数组
返回值	
ErrStatus	SUCCESS或ERROR

例如：

```

cau_parameter_struct cau_ccm_parameter;

uint8_t encrypt_result[TEXT_SIZE];

uint8_t ccm_tag[CCM_TAG_SIZE];

uint8_t aad_buf[AAD_SIZE + 21];

ErrStatus status;

```

.....

/* encryption in CCM mode */

cau_ccm_parameter.alg_dir = CAU_ENCRYPT;

cau_ccm_parameter.key = (uint8_t *)ccm_key_128;

cau_ccm_parameter.key_size = KEY_SIZE;

cau_ccm_parameter.iv = (uint8_t *)ccm_vectors;

cau_ccm_parameter.iv_size = CCM_IV_SIZE;

cau_ccm_parameter.input = (uint8_t *)plaintext;

cau_ccm_parameter.in_length = PLAINTEXT_SIZE;

cau_ccm_parameter.aad = (uint8_t *)aadmessage;

cau_ccm_parameter.aad_size = AAD_SIZE;

status = cau_aes_ccm(&cau_ccm_parameter, encrypt_result, ccm_tag, CCM_TAG_SIZE,
(uint8_t *)aad_buf);

函数 cau_tdes_ecb

函数cau_tdes_ecb描述见下表:

表 3-168. 函数 cau_tdes_ecb

函数名称	cau_tdes_ecb
函数原形	ErrStatus cau_tdes_ecb(cau_parameter_struct *cau_parameter, uint8_t *output)
功能描述	ECB模式下使用TDES算法加密和解密
先决条件	-
被调用函数	-
输入参数{in}	
cau_parameter	参考结构体 表3-139. 结构体cau_parameter_struct
输出参数{out}	
output	指针指向返回数组
返回值	
ErrStatus	SUCCESS或ERROR

例如:

cau_parameter_struct text;

uint8_t encrypt_result[DATA_SIZE];

ErrStatus status;

.....

```
text.alg_dir   = CAU_ENCRYPT;

text.key       = tdes_key;

text.input     = plaintext;

text.in_length = DATA_SIZE;

/* encryption in ECB mode */

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

函数 cau_tdes_cbc

函数cau_tdes_cbc描述见下表:

表 3-169. 函数 cau_tdes_cbc

函数名称	cau_tdes_cbc
函数原形	ErrStatus cau_tdes_cbc(cau_parameter_struct *cau_parameter, uint8_t *output)
功能描述	CBC模式下使用TDES算法加密和解密
先决条件	-
被调用函数	-
输入参数{in}	
cau_parameter	参考结构体 表3-139. 结构体cau_parameter_struct
输出参数{out}	
output	指针指向返回数组
返回值	
ErrStatus	SUCCESS或ERROR

例如:

```
cau_parameter_struct text;

uint8_t encrypt_result[DATA_SIZE];

ErrStatus status;

.....

text.alg_dir   = CAU_ENCRYPT;

text.key       = tdes_key;

text.iv        = vectors;

text.input     = plaintext;

text.in_length = DATA_SIZE;

/* encryption in CBC mode */
```



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

函数 cau_des_ecb

函数cau_des_ecb描述见下表:

表 3-170. 函数 cau_des_ecb

函数名称	cau_des_ecb
函数原形	ErrStatus cau_des_ecb(cau_parameter_struct *cau_parameter, uint8_t *output)
功能描述	ECB模式下使用DES算法加密和解密
先决条件	-
被调用函数	-
输入参数{in}	
cau_parameter	参考结构体 表3-139. 结构体cau_parameter_struct
输出参数{out}	
output	指针指向返回数组
返回值	
ErrStatus	SUCCESS或ERROR

例如:

```
cau_parameter_struct text;

uint8_t encrypt_result[DATA_SIZE];

ErrStatus status;

.....

text.alg_dir   = CAU_ENCRYPT;

text.key       = des_key;

text.input     = plaintext;

text.in_length = DATA_SIZE;

/* encryption in ECB mode */

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

函数 cau_des_cbc

函数cau_des_cbc描述见下表:

表 3-171. 函数 cau_des_cbc

函数名称	cau_des_cbc
函数原形	ErrStatus cau_des_cbc(cau_parameter_struct *cau_parameter, uint8_t *output)

功能描述	CBC模式下使用DES算法加密和解密
先决条件	-
被调用函数	-
输入参数{in}	
cau_parameter	参考结构体 表3-139. 结构体cau_parameter_struct
输出参数{out}	
output	指针指向返回数组
返回值	
ErrStatus	SUCCESS或ERROR

例如：

```
cau_parameter_struct text;

uint8_t encrypt_result[DATA_SIZE];

ErrStatus status;

.....

text.alg_dir   = CAU_ENCRYPT;

text.key       = des_key;

text.iv        = vectors;

text.input     = plaintext;

text.in_length = DATA_SIZE;

/* encryption in CBC mode */

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

函数 cau_interrupt_enable

函数cau_interrupt_enable描述见下表：

表 3-172. 函数 cau_interrupt_enable

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

返回值	
-	-

例如：

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

```
cau_interrupt_enable(CAU_INT_INFIFO);
```

函数 `cau_interrupt_disable`

函数 `cau_interrupt_disable` 描述见下表：

表 3-173. 函数 `cau_interrupt_disable`

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

例如：

```
/* disable cau interrupt */
```

```
cau_interrupt_disable(CAU_INT_INFIFO);
```

函数 `cau_interrupt_flag_get`

函数 `cau_interrupt_flag_get` 描述见下表：

表 3-174. 函数 `cau_interrupt_flag_get`

函数名称	<code>cau_interrupt_flag_get</code>
函数原形	<code>FlagStatus cau_interrupt_flag_get(uint32_t int_flag)</code>
功能描述	获取中断标志
先决条件	-
被调用函数	-
输入参数{in}	
<code>int_flag</code>	CAU中断标志
<code>CAU_INT_FLAG_INFIFO</code>	输入FIFO中断

0	
CAU_INT_FLAG_OUTFIFO	输出FIFO中断
输出参数{out}	
-	-
返回值	
FlagStatus	SET或RESET

例如:

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

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

函数 cau_flag_get

函数cau_flag_get描述见下表:

表 3-175. 函数 cau_flag_get

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

例如:

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

```
FlagStatus status;
```

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

3.5. 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[3.5.1](#)描述了CMP的寄存器列表，章节[3.5.2](#)对CMP库函数进行说明。

3.5.1. 外设寄存器说明

CMP寄存器列表如下表所示：

表 3-176. 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

3.5.2. 外设库函数说明

CMP库函数列表如下表所示：

表 3-177. 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
cmp_flag_clear	清除CMP标志位
cmp_interrupt_enable	CMP中断使能

库函数名称	库函数描述
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 3-178. 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 3-179. 函数 **cmp_deinit**

函数名称	cmp_deinit
函数原型	void cmp_deinit(cmp_enum cmp_periph);
功能描述	复位CMP
先决条件	-
被调用函数	rcu_periph_reset_enable / rcu_periph_reset_disable
输入参数{in}	
cmp_periph	参考枚举 表 3-178. 枚举类型cmp_enum
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 3-180. 函数 **cmp_mode_init**

函数名称	cmp_mode_init
函数原型	void cmp_mode_init(cmp_enum cmp_periph, uint32_t operating_mode, uint32_t inverting_input, uint32_t output_hysteresis);
功能描述	CMP工作模式初始化
先决条件	-
被调用函数	-

输入参数{in}	
cmp_periph	参考枚举 表 3-178. 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 _HIGH	高迟滞
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 3-181. 函数 cmp_noninverting_input_select

函数名称	cmp_noninverting_input_select
函数原型	void cmp_noninverting_input_select(cmp_enum cmp_periph, uint32_t noninverting_input);
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}	
cmp_periph	参考枚举 表 3-178. 枚举类型cmp_enum
输入参数{in}	
noninverting_input	正相输入源
CMP_NONINVERTING_INPUT_PB0_PE9	PB0作为CMP0输入源或者PE9作为CMP1输入源
CMP_NONINVERTING_INPUT_PB2_PE12	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 3-182. 函数 cmp_output_init

函数名称	cmp_output_init
函数原型	void cmp_output_init(cmp_enum cmp_periph, uint32_t output_polarity);
功能描述	CMP输出初始化
先决条件	-
被调用函数	-
输入参数{in}	
cmp_periph	参考枚举 表 3-178. 枚举类型cmp_enum
输入参数{in}	
output_polarity	CMP输出极性
CMP_OUTPUT_POLARITY_INVERTED	输出反相
CMP_OUTPUT_POLARITY_NONINVERTED	输出正相
输出参数{out}	
-	-
返回值	
-	-

例如:

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

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

函数 cmp_output_mux_config

函数cmp_output_mux_config描述见下表:

表 3-183. 函数 cmp_output_mux_config

函数名称	cmp_output_mux_config
函数原型	void cmp_output_mux_config(cmp_enum cmp_periph, uint32_t cmp_output_sel);
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}	
cmp_periph	参考枚举 表 3-178. 枚举类型cmp_enum
输入参数{in}	
cmp_output_sel	CMP备用输出选择
CMP_AFSE_GPIO_PA6	备用输出选择PA6
CMP_AFSE_GPIO_PA8	备用输出选择PA8

<code>CMP_AFSE_GPIO_</code> <code>PB12</code>	备用输出选择PB12
<code>CMP_AFSE_GPIO_</code> <code>PE6</code>	备用输出选择PE6
<code>CMP_AFSE_GPIO_</code> <code>PE15</code>	备用输出选择PE15
<code>CMP_AFSE_GPIO_</code> <code>PG2</code>	备用输出选择PG2
<code>CMP_AFSE_GPIO_</code> <code>PG3</code>	备用输出选择PG3
<code>CMP_AFSE_GPIO_</code> <code>PG4</code>	备用输出选择PG4
<code>CMP_AFSE_GPIO_</code> <code>PK0</code>	备用输出选择PK0
<code>CMP_AFSE_GPIO_</code> <code>PK1</code>	备用输出选择PK1
<code>CMP_AFSE_GPIO_</code> <code>PK2</code>	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 3-184. 函数 `cmp_blanking_init`

函数名称	<code>cmp_blanking_init</code>
函数原型	<code>void cmp_blanking_init(cmp_enum cmp_periph, uint32_t blanking_source_selection);</code>
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}	
<code>cmp_periph</code>	参考枚举 表 3-178. 枚举类型 <code>cmp_enum</code>
输入参数{in}	
<code>blanking_source_selection</code>	消隐源选择

<code>CMP_BLANKING_NONE</code>	无消隐
<code>CMP_BLANKING_TIMER0_OC0</code>	选择TIMER0_CH0输出比较信号为消隐源
<code>CMP_BLANKING_TIMER1_OC2</code>	选择TIMER1_CH2输出比较信号为消隐源
<code>CMP_BLANKING_TIMER2_OC2</code>	选择TIMER2_CH2输出比较信号为消隐源
<code>CMP_BLANKING_TIMER2_OC3</code>	选择TIMER2_CH3输出比较信号为消隐源
<code>CMP_BLANKING_TIMER7_OC0</code>	选择TIMER7_CH0输出比较信号为消隐源
<code>CMP_BLANKING_TIMER14_OC0</code>	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 3-185. 函数 `cmp_enable`

函数名称	<code>cmp_enable</code>
函数原型	<code>void cmp_enable(cmp_enum cmp_periph);</code>
功能描述	使能CMP
先决条件	-
被调用函数	-
输入参数{in}	
<code>cmp_periph</code>	参考枚举 表 3-178. 枚举类型 <code>cmp_enum</code>
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 3-186. 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 3-178. 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 3-187. 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 3-188. 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 3-189. 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 3-178. 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 3-190. 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 3-178. 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 3-191. 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 3-178. 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 3-192. 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 3-178. 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 3-193. 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 3-178. 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 3-194. 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 3-178. 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 3-195. 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 3-178. 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 3-196. 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 3-178. 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 3-197. 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 3-178. 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 3-198. 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 3-178. 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 3-199. 函数 cmp_interrupt_flag_get

函数名称	cmp_interrupt_flag_get
函数原形	FlagStatus cmp_interrupt_flag_get(cmp_enum cmp_periph, uint32_t flag);
功能描述	获取CMP中断标志位
先决条件	-
被调用函数	-
输入参数{in}	
cmp_periph	参考枚举 表 3-178. 枚举类型cmp_enum
输入参数{in}	
flag	CMP中断标志位
CMP_INT_FLAG_C OMPARE	CMP比较中断标志位
输出参数{out}	
-	-
返回值	
FlagStatus	SET或RESET

例如:

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

```
FlagStatus flag_value;
```

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

函数 cmp_interrupt_flag_clear

函数cmp_interrupt_flag_clear描述见下表:

表 3-200. 函数 cmp_interrupt_flag_clear

函数名称	cmp_interrupt_flag_clear
函数原形	void cmp_interrupt_flag_clear(cmp_enum cmp_periph, uint32_t flag);
功能描述	清除CMP中断标志位
先决条件	-
被调用函数	-
输入参数{in}	
cmp_periph	参考枚举 表 3-178. 枚举类型cmp_enum
输入参数{in}	
flag	CMP中断标志位
CMP_INT_FLAG_C OMPARE	CMP比较中断标志位
输出参数{out}	

-	-
返回值	
-	-

例如：

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

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

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

3.6.1. 外设寄存器描述

CPDM寄存器列表如下表所示：

表 3-201. 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

3.6.2. 外设库函数说明

CPDM库函数列表如下表所示：

表 3-202. 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 3-203. 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 3-204. 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 3-205. 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 3-206. 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 3-207. 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 3-208. 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3-203. 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 3-209. 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 3-210. 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>	x = 0,1
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 3-211. 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>	x = 0,1
输入参数{in}	
output_clock_phase	输出时钟相位，请参考 表3-203. 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);
```

3.7. CRC

循环冗余校验码是一种用在数字网络和存储设备上的差错校验码，可以校验原始数据的偶然误差。章节[3.7.1](#)描述了CRC的寄存器列表，章节[3.7.2](#)对CRC库函数进行说明。

3.7.1. 外设寄存器说明

CRC寄存器列表如下表所示：

表 3-212. CRC 寄存器

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

3.7.2. 外设库函数说明

CRC库函数列表如下表所示：

表 3-213. CRC 库函数

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

函数 crc_deinit

函数crc_deinit描述见下表：

表 3-214. 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_reverse_output_data_enable`

函数 `crc_reverse_output_data_enable` 描述见下表：

表 3-215. 函数 `crc_reverse_output_data_enable`

函数名称	<code>crc_reverse_output_data_enable</code>
函数原形	<code>void crc_reverse_output_data_enable(void);</code>
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_reverse_output_data_disable`

函数 `crc_reverse_output_data_disable` 描述见下表：

表 3-216. 函数 `crc_reverse_output_data_disable`

函数名称	<code>crc_reverse_output_data_disable</code>
函数原形	<code>void crc_reverse_output_data_disable(void);</code>
功能描述	除能输出数据翻转功能

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

例如：

```
/* disable crc reverse operation of output data */
```

```
crc_reverse_output_data_disable();
```

函数 `crc_data_register_reset`

函数 `crc_data_register_reset` 描述见下表：

表 3-217. 函数 `crc_data_register_reset`

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

例如：

```
/* reset crc data register */
```

```
crc_data_register_reset();
```

函数 `crc_data_register_read`

函数 `crc_data_register_read` 描述见下表：

表 3-218. 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 3-219. 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 3-220. 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_init_data_register_write

函数crc_init_data_register_write描述见下表：

表 3-221. 函数 crc_init_data_register_write

函数名称	crc_init_data_register_write
函数原形	void crc_init_data_register_write(uint32_t init_data)
功能描述	写初值寄存器
先决条件	-
被调用函数	-
输入参数{in}	
init_data	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_input_data_reverse_config

函数crc_input_data_reverse_config描述见下表：

表 3-222. 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
<i>CRC_INPUT_DATA_NOT</i>	输入数据不翻转
<i>CRC_INPUT_DATA_BYTE</i>	输入数据按字节翻转
<i>CRC_INPUT_DATA_HALFWORD</i>	输入数据按半字翻转
<i>CRC_INPUT_DATA_WORD</i>	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_polynomial_size_set**

函数 **crc_polynomial_size_set** 描述见下表：

表 3-223. 函数 **crc_polynomial_size_set**

函数名称	crc_polynomial_size_set
函数原形	void crc_polynomial_size_set(uint32_t poly_size)
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
<i>CRC_CTL_PS_32</i>	32位多项式值用于CRC计算
<i>CRC_CTL_PS_16</i>	16位多项式值用于CRC计算
<i>CRC_CTL_PS_8</i>	8位多项式值用于CRC计算
<i>CRC_CTL_PS_7</i>	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 3-224. 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 3-225. 函数 `crc_single_data_calculate`

函数名称	<code>crc_single_data_calculate</code>
函数原形	<code>uint32_t crc_single_data_calculate(uint32_t sdata, uint8_t data_format);</code>
功能描述	CRC 计算一个 32 位数据
先决条件	-
被调用函数	-
输入参数{in}	
sdata	设定的 32 位数据
输入参数{in}	
data_format	数据格式
<code>INPUT_FORMAT_WORD</code>	输入数据格式为字
<code>INPUT_FORMAT_HALFWORD</code>	输入数据格式为半字
<code>INPUT_FORMAT_BYTE</code>	输入数据格式为字节

YTE	
输出参数{out}	
-	-
返回值	
uint32_t	32位CRC计算结果（0-0xFFFFFFFF）

例如：

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

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

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

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

函数 crc_block_data_calculate

函数crc_block_data_calculate描述见下表：

表 3-226. 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32位数组
先决条件	-
被调用函数	-
输入参数{in}	
array	32位数据数组
输入参数{in}	
size	数据长度
输入参数{in}	
data_format	数据格式
INPUT_FORMAT_WORD	输入数据格式为字
INPUT_FORMAT_HALFWORD	输入数据格式为半字
INPUT_FORMAT_BYTE	输入数据格式为字节
输出参数{out}	
-	-
返回值	
uint32_t	32位CRC计算结果（0-0xFFFFFFFF）

例如：

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

```
#define BUFFER_SIZE    6

uint32_t valcrc = 0;

static const uint32_t data_buffer[BUFFER_SIZE] = {

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

valcrc = crc_block_data_calculate(data_buffer, BUFFER_SIZE, INPUT_FORMAT_WORD);
```

3.8. CTC

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

3.8.1. 外设寄存器说明

CTC寄存器列表如下表所示：

表 3-227. CTC 寄存器

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

3.8.2. 外设库函数说明

CTC库函数列表如下表所示：

表 3-228. 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

库函数名称	库函数描述
ctc_counter_direction_read	读取CTC校准时钟计数方向
ctc_counter_reload_value_read	读取CTC计数器重载值
ctc_irc48m_trim_value_read	读取IRC48M校准值
ctc_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 3-229. 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 3-230. 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 3-231. 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 3-232. 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 3-233. 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 3-234. 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_	硬件校准开启

TRIM_MODE_ENABLE	
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 3-235. 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 3-236. 函数 ctc_refsource_signal_select

函数名称	ctc_refsource_signal_select
函数原形	void ctc_refsource_signal_select(uint32_t refs);
功能描述	CTC参考信号源选择
先决条件	-
被调用函数	-
输入参数{in}	
refs	参考信号源
CTC_REFSOURCE_GPIO	选择GPIO输入信号
CTC_REFSOURCE_LXTAL	选择LXTAL时钟
输出参数{out}	
-	-
返回值	
-	-

例如:

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

```
ctc_refsource_signal_select (CTC_REFSOURCE_LXTAL);
```

函数 ctc_refsource_prescaler_config

函数ctc_refsource_prescaler_config描述见下表:

表 3-237. 函数 ctc_refsource_prescaler_config

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

<code>_PSC_DIV8</code>	
<code>CTC_REFSOURCE</code> <code>_PSC_DIV16</code>	参考信号16分频
<code>CTC_REFSOURCE</code> <code>_PSC_DIV32</code>	参考信号32分频
<code>CTC_REFSOURCE</code> <code>_PSC_DIV64</code>	参考信号64分频
<code>CTC_REFSOURCE</code> <code>_PSC_DIV128</code>	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 3-238. 函数 `ctc_clock_limit_value_config`

函数名称	<code>ctc_clock_limit_value_config</code>
函数原形	<code>void ctc_clock_limit_value_config(uint8_t limit_value);</code>
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}	
<code>limit_value</code>	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 3-239. 函数 `ctc_counter_reload_value_config`

函数名称	<code>ctc_counter_reload_value_config</code>
函数原形	<code>void ctc_counter_reload_value_config(uint16_t reload_value);</code>
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}	
<code>reload_value</code>	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 3-240. 函数 `ctc_counter_capture_value_read`

函数名称	<code>ctc_counter_capture_value_read</code>
函数原形	<code>uint16_t ctc_counter_capture_value_read(void);</code>
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	
<code>uint16_t</code>	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 3-241. 函数 `ctc_counter_direction_read`

函数名称	<code>ctc_counter_direction_read</code>
函数原形	<code>FlagStatus ctc_counter_direction_read(void);</code>
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 3-242. 函数 `ctc_counter_reload_value_read`

函数名称	<code>ctc_counter_reload_value_read</code>
函数原形	<code>uint16_t ctc_counter_reload_value_read(void);</code>
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 3-243. 函数 ctc_irc48m_trim_value_read

函数名称	ctc_irc48m_trim_value_read
函数原形	uint8_t ctc_irc48m_trim_value_read(void);
功能描述	读取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	
uint8_t	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 3-244. 函数 ctc_flag_get

函数名称	ctc_flag_get
函数原形	FlagStatus ctc_flag_get (uint32_t flag);
功能描述	获取CTC状态标志位
先决条件	-
被调用函数	-
输入参数{in}	
flag	CTC状态标志
CTC_FLAG_CKOK	时钟校准完成标志
CTC_FLAG_CKWARN	时钟校准警告标志
CTC_FLAG_ERR	错误标志
CTC_FLAG_EREFS	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	
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 3-245. 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_CKER R	时钟校准错误标志
CTC_FLAG_REFM SS	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 3-246. 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
<i>CTC_INT_CKOK</i>	时钟校准完成中断
<i>CTC_INT_CKWARN</i>	时钟校准警告中断
<i>CTC_INT_ERR</i>	错误中断
<i>CTC_INT_EREFS</i>	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 3-247. 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
<i>CTC_INT_CKOK</i>	时钟校准完成中断
<i>CTC_INT_CKWARN</i>	时钟校准警告中断
<i>CTC_INT_ERR</i>	错误中断
<i>CTC_INT_EREFS</i>	期望参考信号中断

输出参数{out}	
-	-
返回值	
-	-

例如:

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

```
ctc_interrupt_disable (CTC_INT_CKOK);
```

函数 ctc_interrupt_flag_get

函数ctc_interrupt_flag_get描述见下表:

表 3-248. 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_ERR	错误中断标志
CTC_INT_FLAG_REF	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_EFMIS	参考同步脉冲信号丢失标志
CTC_INT_FLAG_T_RIMERR	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 3-249. 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_ERRRR</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_REFMISST</code>	参考同步脉冲信号丢失标志
<code>CTC_INT_FLAG_TIMERR</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);
```


3.9. DAC

数字/模拟转换器可以将12位的数字数据转换为外部引脚上的电压输出，章节[3.9.1](#)描述了DAC的寄存器列表，章节[3.9.2](#)对DAC库函数进行说明。

3.9.1. 外设寄存器说明

DAC寄存器列表如下表所示：

表 3-250. 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

3.9.2. 外设库函数说明

DAC库函数列表如下表所示：

表 3-251. 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 3-252. 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 3-253. 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 3-254. 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 3-255. 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 3-256. 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 3-257. 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 3-258. 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 3-259. 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 3-260. 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 3-261. 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 3-262. 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 3-263. 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 3-264. 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 3-265. 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 3-266. 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 3-267. 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 3-268. 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 3-269. 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 3-270. 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 3-271. 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 3-272. 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 3-273. 函数 `dac_concurrent_data_set`

函数名称	<code>dac_concurrent_data_set</code>
函数原型	<code>void dac_concurrent_data_set(uint32_t dac_periph, uint32_t dac_align, uint16_t data0, uint16_t data1);</code>
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}	
<code>dac_periph</code>	DAC外设
<code>DACx</code>	DAC外设选择（ <code>x = 0</code> ）
输入参数{in}	
<code>dac_align</code>	DAC对齐模式
<code>DAC_ALIGN_8B_R</code>	8位数据右对齐
<code>DAC_ALIGN_12B_R</code>	12位数据右对齐
<code>DAC_ALIGN_12B_L</code>	12位数据左对齐
输入参数{in}	
<code>data0</code>	写入DACx_OUT0的数据（0~4095）
输入参数{in}	
<code>data1</code>	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 3-274. 函数 `dac_sample_keep_mode_config`

函数名称	<code>dac_sample_keep_mode_config</code>
函数原型	<code>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);</code>
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 3-275. 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 3-276. 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
DACx	DAC外设选择（x = 0）
输入参数{in}	
flag	DAC状态标志位
DAC_FLAG_DDUD R0	DACx_OUT0 DMA欠载标志位
DAC_FLAG_DDUD R1	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 3-277. 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>	DACx_OUT0 DMA欠载中断
<code>DAC_INT_DDUDR1</code>	DACx_OUT1 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 3-278. 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>	DACx_OUT0 DMA欠载中断
<code>DAC_INT_DDUDR1</code>	DACx_OUT1 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 3-279. 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 3-280. 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);
```

3.10. DBG

调试系统帮助调试者在低功耗模式下调试或者进行一些外设调试。章节[3.10.1](#)描述了DBG的寄存器列表，章节[3.10.2](#)对DBG库函数进行说明。

3.10.1. 外设寄存器说明

DBG寄存器列表如下表所示：

表 3-281. 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

3.10.2. 外设库函数说明

DBG库函数列表如下表所示：

表 3-282. 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 3-283. 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 3-284. 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 3-285. 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 3-286. 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 3-287. 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 3-288. 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3-283. 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 3-289. 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3-283. 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 3-290. 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 3-291. 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 3-292. 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);
```

3.11. DCI

数字摄像头接口是一个同步并行接口，可以从数字摄像头捕获视频和图像信息。DCI寄存器列举在章节[3.11.1](#)，DCI固件库函数列举在章节[3.11.2](#)。

3.11.1. 外设寄存器说明

DCI寄存器列表如下表所示：

表 3-293. DCI 寄存器

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

寄存器名称	寄存器描述
DCI_DATA	DCI数据寄存器

3.11.2. 外设库函数说明

DCI库函数列表如下表所示：

表 3-294. DCI 库函数

库函数名称	库函数描述
dc_i_deinit	复位DCI
dc_i_struct_para_init	初始化DCI参数结构体为默认值
dc_i_init	初始化DCI寄存器
dc_i_enable	使能DCI功能
dc_i_disable	除能DCI功能
dc_i_capture_enable	使能DCI捕获功能
dc_i_capture_disable	除能DCI捕获功能
dc_i_external_vsync_enable	在CCIR逐行扫描模式下使能DCI外部同步功能
dc_i_external_vsync_disable	在CCIR逐行扫描模式下除能DCI外部同步功能
dc_i_automatic_error_correction_enable	在CCIR隔行扫描模式下使能DCI自动纠错功能
dc_i_automatic_error_correction_disable	在CCIR隔行扫描模式下除能DCI自动纠错功能
dc_i_jpeg_enable	使能DCI JPEG模式
dc_i_jpeg_disable	除能DCI JPEG模式
dc_i_crop_window_enable	使能裁剪窗口功能
dc_i_crop_window_disable	除能裁剪窗口功能
dc_i_ccir_enable	使能CCIR模式
dc_i_ccir_disable	除能CCIR模式
dc_i_ccir_mode_select	选择CCIR模式
dc_i_crop_window_config	配置DCI裁剪窗口功能
dc_i_embedded_sync_enable	使能内嵌同步模式
dc_i_embedded_sync_disable	除能内嵌同步模式
dc_i_sync_codes_config	在内嵌同步模式下配置同步码
dc_i_sync_codes_unmask_config	在内嵌同步模式下配置非屏蔽同步码
dc_i_data_read	读取DCI数据寄存器
dc_i_flag_get	获取指定标志
dc_i_interrupt_enable	使能指定的DCI中断
dc_i_interrupt_disable	除能指定的DCI中断
dc_i_interrupt_flag_get	获取指定的中断标志
dc_i_interrupt_flag_clear	清除指定的中断标志

结构体 dc_i_parameter_struct

表 3-295. 结构体 dc_i_parameter_struct

成员名称	功能描述
------	------

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

函数 dci_deinit

函数dci_deinit描述见下表:

表 3-296. 函数 dci_deinit

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

例如:

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

```
dci_deinit();
```

函数 dci_init

函数dci_init描述见下表:

表 3-297. 函数 dci_init

函数名称	dci_init
函数原形	void dci_init(dci_parameter_struct* dci_struct);
功能描述	初始化DCI寄存器

先决条件	-
被调用函数	-
输入参数{in}	
dci_struct	DCI参数初始化结构体的地址
输出参数{out}	
-	-
返回值	
-	-

例如：

```

/* initialize DCI registers */

dci_parameter_struct dci_struct;

dci_struct.capture_mode = DCI_CAPTURE_MODE_CONTINUOUS;

dci_struct.clock_polarity = DCI_CK_POLARITY_RISING;

dci_struct.hsync_polarity = DCI_HSYNC_POLARITY_LOW;

dci_struct.vsync_polarity = DCI_VSYNC_POLARITY_LOW;

dci_struct.frame_rate = DCI_FRAME_RATE_ALL;

dci_struct.interface_format = DCI_INTERFACE_FORMAT_8BITS;

dci_init(&dci_struct);

```

函数 dci_enable

函数dci_enable描述见下表：

表 3-298. 函数 dci_enable

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

例如：

```

/* enable DCI function */

```

```
dc_i_enable();
```

函数 dci_disable

函数dci_disable描述见下表：

表 3-299. 函数 dci_disable

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

例如：

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

```
dc_i_disable();
```

函数 dci_capture_enable

函数dci_capture_enable描述见下表：

表 3-300. 函数 dci_capture_enable

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

例如：

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

```
dc_i_capture_enable();
```


函数 dci_capture_disable

函数dci_capture_disable描述见下表：

表 3-301. 函数 dci_capture_disable

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

例如：

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

```
dci_capture_disable();
```

函数 dci_external_vsync_enable

函数dci_external_vsync_enable描述见下表：

表 3-302. 函数 dci_external_vsync_enable

函数名称	dci_external_vsync_enable
函数原形	void dci_external_vsync_enable(void);
功能描述	在CCIR逐行扫描模式下使能DCI外部同步功能
先决条件	-
被调用函数	-
输入参数{in}	
-	-
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* enable DCI external vsync in CCIR progressive mode */
```

```
dci_external_vsync_enable();
```

函数 dci_external_vsync_disable

函数dci_external_vsync_disable描述见下表:

表 3-303. 函数 dci_external_vsync_disable

函数名称	dci_external_vsync_disable
函数原形	void dci_external_vsync_disable(void);
功能描述	在CCIR逐行扫描模式下除能DCI外部同步功能
先决条件	-
被调用函数	-
输入参数{in}	
-	-
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* disable DCI external vsync in CCIR progressive mode */  
  
dci_external_vsync_disable();
```

函数 dci_automatic_error_correction_enable

函数dci_automatic_error_correction_enable描述见下表:

表 3-304. 函数 dci_automatic_error_correction_enable

函数名称	dci_automatic_error_correction_enable
函数原形	void dci_automatic_error_correction_enable(void);
功能描述	在CCIR隔行扫描模式下使能DCI自动纠错功能
先决条件	-
被调用函数	-
输入参数{in}	
-	-
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* enable DCI automatic error correction in CCIR interlaced mode */  
  
dci_automatic_error_correction_enable();
```

函数 dci_automatic_error_correction_disable

函数dci_automatic_error_correction_disable描述见下表：

表 3-305. 函数 dci_automatic_error_correction_disable

函数名称	dci_automatic_error_correction_disable
函数原形	void dci_automatic_error_correction_disable (void);
功能描述	在CCIR隔行扫描模式下除能DCI自动纠错功能
先决条件	-
被调用函数	-
输入参数{in}	
-	-
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* disable DCI automatic error correction in CCIR interlaced mode */  
dci_automatic_error_correction_disable();
```

函数 dci_jpeg_enable

函数dci_jpeg_enable描述见下表：

表 3-306. 函数 dci_jpeg_enable

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

例如：

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

函数 dci_jpeg_disable

函数dci_jpeg_disable描述见下表:

表 3-307. 函数 dci_jpeg_disable

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

例如:

```
/* disable DCI jpeg mode */  
  
dci_jpeg_disable();
```

函数 dci_crop_window_enable

函数dci_crop_window_enable描述见下表:

表 3-308. 函数 dci_crop_window_enable

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

例如:

```
/* enable cropping window function */  
  
dci_crop_window_enable();
```

函数 dci_crop_window_disable

函数dci_crop_window_disable描述见下表：

表 3-309. 函数 dci_crop_window_disable

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

例如：

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

```
dci_crop_window_disable();
```

函数 dci_crop_window_config

函数dci_crop_window_config描述见下表：

表 3-310. 函数 dci_crop_window_config

函数名称	dci_crop_window_config
函数原形	void dci_crop_window_config(uint16_t start_x, uint16_t start_y, uint16_t size_width, uint16_t size_height);
功能描述	配置DCI裁剪窗口功能
先决条件	-
被调用函数	-
输入参数{in}	
start_x	窗口水平起始地址
输入参数{in}	
start_y	窗口垂直起始地址
输入参数{in}	
size_width	窗口水平长度大小
输入参数{in}	
size_height	窗口垂直长度大小
输出参数{out}	
-	-
返回值	

例如：

```
/* config DCI cropping window */
```

```
dci_crop_window_config(0x800, 0x600, 0x100, 0x100);
```

函数 dci_embedded_sync_enable

函数dci_embedded_sync_enable描述见下表：

表 3-311. 函数 dci_embedded_sync_enable

函数名称	dci_embedded_sync_enable
函数原形	void dci_embedded_sync_enable(void);
功能描述	使能内嵌同步模式
先决条件	-
被调用函数	-
输入参数{in}	
-	-
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* enable embedded synchronous mode */
```

```
dci_embedded_sync_enable();
```

函数 dci_embedded_sync_disable

函数dci_embedded_sync_disable描述见下表：

表 3-312. 函数 dci_embedded_sync_disable

函数名称	dci_embedded_sync_disable
函数原形	void dci_embedded_sync_disable(void);
功能描述	除能内嵌同步模式
先决条件	-
被调用函数	-
输入参数{in}	
-	-
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* disable embedded synchronous mode */
```

```
dc_i_embedded_sync_disable()
```

函数 **dc_i_ccir_enable**

函数dc_i_ccir_enable描述见下表：

表 3-313. 函数 dc_i_ccir_enable

函数名称	dc_i_ccir_enable
函数原形	void dc_i_ccir_enable (void);
功能描述	使能CCIR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：

```
/* CCIR mode enable */
```

```
dc_i_ccir_enable();
```

函数 **dc_i_ccir_disable**

函数dc_i_ccir_disable描述见下表：

表 3-314. 函数 dc_i_ccir_disable

函数名称	dc_i_ccir_disable
函数原形	void dc_i_ccir_disable (void);
功能描述	除能CCIR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：

```
/* CCIR mode disable */
```

```
dc_i_ccir_disable();
```

函数 dc_i_ccir_mode_select

函数dc_i_ccir_mode_select描述见下表：

表 3-315. 函数 dc_i_ccir_mode_select

函数名称	dc_i_ccir_mode_select
函数原形	void dc_i_ccir_mode_select(uint32_t ccir_mode);
功能描述	选择CCIR模式
先决条件	-
被调用函数	-
输入参数{in}	
ccir_mode	CCIR模式
CCIR_PROGRESSIVE_MODE	CCIR逐行扫描模式
CCIR_INTERLACE_MODE	CCIR隔行扫描模式
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* CCIR mode disable */
```

```
dc_i_ccir_mode_select (CCIR_Progressive_Mode);
```

函数 dc_i_sync_codes_config

函数dc_i_sync_codes_config描述见下表：

表 3-316. 函数 dc_i_sync_codes_config

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

输入参数{in}	
line_end	在内嵌同步模式行终止码
输入参数{in}	
frame_end	在内嵌同步模式帧终止码
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* config synchronous codes in embedded synchronous mode */
```

```
dc_i_sync_codes_config(0x10, 0x10, 0x20, 0x20);
```

函数 dc_i_sync_codes_unmask_config

函数dc_i_sync_codes_unmask_config描述见下表：

表 3-317. 函数 dc_i_sync_codes_unmask_config

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

例如：

```
/* config synchronous codes unmask in embedded synchronous mode */
```

```
dc_i_sync_codes_unmask_config(0x10, 0x10, 0x20, 0x20);
```

函数 dci_data_read

函数dci_data_read描述见下表:

表 3-318. 函数 dci_data_read

函数名称	dci_data_read
函数原形	uint32_t dci_data_read(void);
功能描述	读取DCI数据寄存器
先决条件	-
被调用函数	-
输入参数{in}	
-	-
输出参数{out}	
-	-
返回值	
uint32_t	0x00 – 0xFFFFFFFF

例如:

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

函数 dci_flag_get

函数dci_flag_get描述见下表:

表 3-319. 函数 dci_flag_get

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

返回值	
FlagStatus	SET或RESET

例如:

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

```
FlagStatus status = dci_flag_get (DCI_FLAG_HS);
```

函数 dci_interrupt_enable

函数dci_interrupt_enable描述见下表:

表 3-320. 函数 dci_interrupt_enable

函数名称	dci_interrupt_enable
函数原形	void dci_interrupt_enable(uint32_t interrupt);
功能描述	使能指定的DCI中断
先决条件	-
被调用函数	-
输入参数{in}	
interrupt	DCI中断
DCI_INT_EF	帧结束中断
DCI_INT_OVR	FIFO溢出中断
DCI_INT_ESE	内嵌码同步错误中断
DCI_INT_VSYNC	垂直同步中断
DCI_INT_EL	行结束中断
DCI_INT_F0	CCIR场0中断
DCI_INT_F1	CCIR场1中断
DCI_INT_COF	场切换中断
DCI_INT_CCE	CCIR错误中断
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* enable specified DCI interrupt */
```

```
dci_interrupt_enable(DCI_INT_EF);
```

函数 dci_interrupt_disable

函数dci_interrupt_disable描述见下表:

表 3-321. 函数 dci_interrupt_disable

函数名称	dci_interrupt_disable
------	-----------------------

函数原形	void dci_interrupt_disable(uint32_t interrupt);
功能描述	除能指定的DCI中断
先决条件	-
被调用函数	-
输入参数{in}	
interrupt	DCI interrupt
DCI_INT_EF	帧结束中断
DCI_INT_OVR	FIFO溢出中断
DCI_INT_ESE	内嵌码同步错误中断
DCI_INT_VSYNC	垂直同步中断
DCI_INT_EL	行结束中断
DCI_INT_F0	CCIR场0中断
DCI_INT_F1	CCIR场1中断
DCI_INT_COF	场切换中断
DCI_INT_CCE	CCIR错误中断
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* disable specified DCI interrupt */
```

```
dci_interrupt_disable(DCI_INT_EF);
```

函数 dci_interrupt_flag_get

函数dci_interrupt_flag_get描述见下表：

表 3-322. 函数 dci_interrupt_flag_get

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

DCI_INT_FLAG_CCE	CCIR错误中断标志
输出参数{out}	
-	-
返回值	
FlagStatus	SET或RESET

例如：

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

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

函数 dci_interrupt_flag_clear

函数dci_interrupt_flag_clear描述见下表：

表 3-323. 函数 dci_interrupt_flag_clear

函数名称	dci_interrupt_flag_clear
函数原形	void dci_interrupt_flag_clear(uint32_t interrupt);
功能描述	清除指定的中断标志
先决条件	-
被调用函数	-
输入参数{in}	
interrupt	DCI中断
DCI_INT_FLAG_EF	帧结束中断标志
DCI_INT_FLAG_OVR	FIFO溢出中断标志
DCI_INT_FLAG_ESE	内嵌码同步错误中断标志
DCI_INT_FLAG_VSYN C	垂直同步中断标志
DCI_INT_FLAG_EL	行结束中断标志
DCI_INT_F0	CCIR场0中断标志
DCI_INT_F1	CCIR场1中断标志
DCI_INT_COF	场切换中断标志
DCI_INT_CCE	CCIR错误中断标志
输出参数{out}	
-	-
返回值	
-	-

例如：

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

```
dci_interrupt_flag_clear(DCI_INT_FLAG_EF);
```

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

3.12.1. 外设寄存器说明

DMA寄存器列表如下表所示：

表 3-324. 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
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	通道x FIFO控制寄存器

DMAMUX寄存器列表如下表所示：

表 3-325. 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

寄存器名称	寄存器描述
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

3.12.2. 外设库函数说明

DMA库函数列表如下表所示：

表 3-326. 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
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

库函数名称	库函数描述
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 3-327. 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
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 3-328. 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 3-329. 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 3-330. 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 3-331. 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 3-332. 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 3-333. 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

成员名称	功能描述
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 3-334. 枚举 dmamux_generator_channel_enum

成员名称	功能描述
DMAMUX_GENCH0	DMAMUX请求生成通道0
DMAMUX_GENCH1	DMAMUX请求生成通道1
DMAMUX_GENCH2	DMAMUX请求生成通道2
DMAMUX_GENCH3	DMAMUX请求生成通道3
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 3-335. 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

成员名称	功能描述
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 XCH15_SO	DMAMUX请求路由通道15同步溢出中断标志
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 3-336. 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
DMAMUX_FLAG_M UXCH10_SO	DMAMUX请求路由通道10同步溢出标志
DMAMUX_FLAG_M UXCH11_SO	DMAMUX请求路由通道11同步溢出标志
DMAMUX_FLAG_M UXCH12_SO	DMAMUX请求路由通道12同步溢出标志
DMAMUX_FLAG_M UXCH13_SO	DMAMUX请求路由通道13同步溢出标志
DMAMUX_FLAG_M UXCH14_SO	DMAMUX请求路由通道14同步溢出标志
DMAMUX_FLAG_M UXCH15_SO	DMAMUX请求路由通道15同步溢出标志
DMAMUX_FLAG_G ENCH0_TO	DMAMUX请求生成通道0触发溢出标志
DMAMUX_FLAG_G ENCH1_TO	DMAMUX请求生成通道1触发溢出标志
DMAMUX_FLAG_G ENCH2_TO	DMAMUX请求生成通道2触发溢出标志
DMAMUX_FLAG_G	DMAMUX请求生成通道3触发溢出标志

成员名称	功能描述
ENCH3_TO	
DMAMUX_FLAG_G ENCH4_TO	DMAMUX请求生成通道4触发溢出标志
DMAMUX_FLAG_G ENCH5_TO	DMAMUX请求生成通道5触发溢出标志
DMAMUX_FLAG_G ENCH6_TO	DMAMUX请求生成通道6触发溢出标志
DMAMUX_FLAG_G ENCH7_TO	DMAMUX请求生成通道7触发溢出标志

枚举 dmamux_interrupt_flag_enum

表 3-337. 枚举 dmamux_interrupt_flag_enum

成员名称	功能描述
DMAMUX_INT_FL G_MUXCH0_SO	DMAMUX请求路由通道0同步溢出中断标志
DMAMUX_INT_FL G_MUXCH1_SO	DMAMUX请求路由通道1同步溢出中断标志
DMAMUX_INT_FL G_MUXCH2_SO	DMAMUX请求路由通道2同步溢出中断标志
DMAMUX_INT_FL G_MUXCH3_SO	DMAMUX请求路由通道3同步溢出中断标志
DMAMUX_INT_FL G_MUXCH4_SO	DMAMUX请求路由通道4同步溢出中断标志
DMAMUX_INT_FL G_MUXCH5_SO	DMAMUX请求路由通道5同步溢出中断标志
DMAMUX_INT_FL G_MUXCH6_SO	DMAMUX请求路由通道6同步溢出中断标志
DMAMUX_INT_FL G_MUXCH7_SO	DMAMUX请求路由通道7同步溢出中断标志
DMAMUX_INT_FL G_MUXCH8_SO	DMAMUX请求路由通道8同步溢出中断标志
DMAMUX_INT_FL G_MUXCH9_SO	DMAMUX请求路由通道9同步溢出中断标志
DMAMUX_INT_FL G_MUXCH10_SO	DMAMUX请求路由通道10同步溢出中断标志
DMAMUX_INT_FL G_MUXCH11_SO	DMAMUX请求路由通道11同步溢出中断标志
DMAMUX_INT_FL G_MUXCH12_SO	DMAMUX请求路由通道12同步溢出中断标志
DMAMUX_INT_FL G_MUXCH13_SO	DMAMUX请求路由通道13同步溢出中断标志

成员名称	功能描述
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 3-338. 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3-332. 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 3-339. 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3-329. 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 3-340. 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3-328. 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 3-341. 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3-332. 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3-329. 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 3-342. 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
DMAx	(x=0,1)
输入参数{in}	
channelx	DMA通道，参考 表3-332. 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3-328. 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 3-343. 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3-332. 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 3-344. 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3-332. 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 3-345. 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
<i>DMAx</i>	(x=0,1)
输入参数{in}	
channelx	DMA通道，参考 表3-332. 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 3-346. 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
<i>DMAx</i>	(x=0,1)
输入参数{in}	
channelx	DMA通道，参考 表3-332. 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 3-347. 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3-332. 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_MEDIUM	中优先级
DMA_PRIORITY_HIGH	高优先级
DMA_PRIORITY_ULTRA_HIGH	极高优先级
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* configure 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 3-348. 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
<i>DMAx</i>	(x=0,1)
输入参数{in}	
channelx	DMA通道, 参考 表3-332. 枚举dma_channel_enum 。
输入参数{in}	
mbeat	存储器增量突发类型
<i>DMA_MEMORY_BURST_SINGLE</i>	存储器单一传输
<i>DMA_MEMORY_BURST_4_BEAT</i>	存储器4拍增量突发传输
<i>DMA_MEMORY_BURST_8_BEAT</i>	存储器8拍增量突发传输
<i>DMA_MEMORY_BURST_16_BEAT</i>	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 3-349. 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
<i>DMAx</i>	(x=0,1)
输入参数{in}	
channelx	DMA通道, 参考 表3-332. 枚举dma_channel_enum 。
输入参数{in}	
mbeat	外设增量突发类型
<i>DMA_PERIPH_BURST_SINGLE</i>	外设单一传输

<code>_SINGLE</code>	
<code>DMA_PERIPH_BURST_4_BEAT</code>	外设4拍增量突发传输
<code>DMA_PERIPH_BURST_8_BEAT</code>	外设8拍增量突发传输
<code>DMA_PERIPH_BURST_16_BEAT</code>	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 3-350. 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
<i>DMAx</i>	(x=0,1)
输入参数{in}	
channelx	DMA通道，参考 表3-332. 枚举dma_channel_enum 。
输入参数{in}	
msize	存储器传输数据宽度
<i>DMA_MEMORY_WIDTH_H_8BIT</i>	存储器传输数据8-bit宽度
<i>DMA_MEMORY_WIDTH_H_16BIT</i>	存储器传输数据16-bit宽度
<i>DMA_MEMORY_WIDTH_H_32BIT</i>	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 3-351. 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3-332. 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 3-352. 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3-332. 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_INCREASE_ENABLE);
```

函数 dma_peripheral_address_generation_config

函数dma_peripheral_address_generation_config描述见下表:

表 3-353. 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3-332. 枚举dma_channel_enum 。
输入参数{in}	
generation_algorithm	外设地址生成算法
DMA_PERIPH_INCREASE_ENABLE	外设增量地址模式
DMA_PERIPH_INCREASE_DISABLE	外设固定地址模式
DMA_PERIPH_INCREASE_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_INCREASE_DISABLE);
```

函数 dma_circulation_enable

函数dma_circulation_enable描述见下表：

表 3-354. 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3-332. 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 3-355. 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3-332. 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 3-356. 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3-332. 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 3-357. 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3-332. 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 3-358. 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3-332. 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 3-359. 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
<i>DMAx</i>	(x=0,1)
输入参数{in}	
channelx	DMA通道, 参考 表3-332. 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

<i>DMA_MEMORY_0</i>	DMA存储器0
<i>DMA_MEMORY_1</i>	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 3-360. 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, dma_channel_enum channelx);
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}	
dma_periph	DMA外设
<i>DMAx</i>	(x=0,1)
输入参数{in}	
channelx	DMA通道，参考 表3-332. 枚举dma_channel_enum 。
输出参数{out}	
-	-
返回值	
uint32_t	使用的存储器
<i>DMA_MEMORY_0</i>	DMA存储器0
<i>DMA_MEMORY_1</i>	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 3-361. 函数 dma_switch_buffer_mode_enable

函数名称	dma_switch_buffer_mode_enable
函数原型	void dma_switch_buffer_mode_enable(uint32_t dma_periph, dma_channel_enum channelx);
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}	
dma_periph	DMA外设
DMAx	(x=0,1)
输入参数{in}	
channelx	DMA通道, 参考 表3-332. 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 3-362. 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3-332. 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 3-363. 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3-332. 枚举dma_channel_enum 。
输出参数{out}	
-	-
返回值	
uint32_t	存储在FIFO中的字数
DMA_FIFO_CNT_NO_DATA	无数据
DMA_FIFO_CNT_1_WORD	1个字
DMA_FIFO_CNT_2_WORD	2个字
DMA_FIFO_CNT_3_WORD	3个字
DMA_FIFO_CNT_EMPTY	FIFO空
DMA_FIFO_CNT_FULL	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 3-364. 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
DMAx	(x=0,1)
输入参数{in}	
channelx	DMA通道, 参考 表3-332. 枚举dma_channel_enum 。
输入参数{in}	
flag	指定获取的标志位
DMA_FLAG_FEE	FIFO错误与异常标志
DMA_FLAG_SDE	单数据传输模式异常标志
DMA_FLAG_TAE	传输错误标志
DMA_FLAG_HTF	半传输完成标志
DMA_FLAG_FTF	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 3-365. 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3-332. 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 3-366. 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
<i>DMAx</i>	(x=0,1)
输入参数{in}	
channelx	DMA通道, 参考 表3-332. 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：

```
/* 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 3-367. 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
DMAx	(x=0,1)
输入参数{in}	
channelx	DMA通道，参考 表3-332. 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：

```
/* 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 3-368. 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
DMAx	(x=0,1)
输入参数{in}	
channelx	DMA通道, 参考 表3-332. 枚举dma_channel_enum 。
输入参数{in}	
int_flag	指定获取的中断标志
DMA_INT_FLAG_FEE	FIFO错误与异常中断标志
DMA_INT_FLAG_SDE	单数据传输模式异常中断标志
DMA_INT_FLAG_TAE	传输错误中断标志
DMA_INT_FLAG_HTF	半传输完成中断标志
DMA_INT_FLAG_FTF	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 3-369. 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

DMAx	(x=0,1)
输入参数{in}	
channelx	DMA通道，参考 表3-332. 枚举dma_channel_enum 。
输入参数{in}	
int_flag	指定获取的中断标志
DMA_INT_FLAG_FEE	FIFO错误与异常中断标志
DMA_INT_FLAG_SDE	单数据传输模式异常中断标志
DMA_INT_FLAG_TAE	传输错误中断标志
DMA_INT_FLAG_HTF	半传输完成中断标志
DMA_INT_FLAG_FTF	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 3-370. 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3-330. 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 3-371. 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3-333. 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3-330. 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 3-372. 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3-333. 枚举 dmamux_mux_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 3-373. 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3-333. 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 3-374. 函数 `dmamux_event_generation_enable`

函数名称	<code>dmamux_event_generation_enable</code>
函数原型	void <code>dmamux_event_generation_enable(dmamux_mux_channel_enum channelx);</code>
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3-333. 枚举 <code>dmamux_mux_channel_enum</code> 。
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 3-375. 函数 `dmamux_event_generation_disable`

函数名称	<code>dmamux_event_generation_disable</code>
函数原型	void <code>dmamux_event_generation_disable(dmamux_mux_channel_enum channelx);</code>
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3-333. 枚举 <code>dmamux_mux_channel_enum</code> 。
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 3-376. 函数 `dmamux_gen_struct_para_init`

函数名称	<code>dmamux_gen_struct_para_init</code>
函数原型	<code>void dmamux_gen_struct_para_init(dmamux_gen_parameter_struct *init_struct);</code>
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}	
<code>init_struct</code>	初始化DMAMUX生成通道需要的数据，参考 表3-331. 结构体 <code>dmamux_gen_parameter_struct</code> 。
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 3-377. 函数 `dmamux_request_generator_init`

函数名称	<code>dmamux_request_generator_init</code>
函数原型	<code>void dmamux_request_generator_init(dmamux_generator_channel_enum channelx, dmamux_gen_parameter_struct *init_struct);</code>
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}	
<code>channelx</code>	DMAMUX请求生成通道，参考 表3-334. 枚举 <code>dmamux_generator_channel_enum</code> 。
输入参数{in}	
<code>init_struct</code>	初始化DMAMUX生成通道需要的数据，参考 表3-331. 结构体 <code>dmamux_gen_parameter_struct</code> 。
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 3-378. 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3-334. 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 3-379. 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3-334. 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 3-380. 函数 dmamux_synchronization_polarity_config

函数名称	dmamux_synchronization_polarity_config
函数原型	void dmamux_synchronization_polarity_config(dmamux_multiplexer_channel_enum channelx, uint32_t polarity);
功能描述	配置DMAMUX同步输入的有效边沿
先决条件	-
被调用函数	-
输入参数{in}	
channelx	DMAMUX请求路由通道，参考 表3-333. 枚举 dmamux_multiplexer_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 3-381. 函数 dmamux_request_forward_number_config

函数名称	dmamux_request_forward_number_config
函数原型	void dmamux_request_forward_number_config(dmamux_multiplexer_channel_enum channelx, uint32_t number);
功能描述	配置DMAMUX通道x要传输多少个DMA请求
先决条件	-
被调用函数	-
输入参数{in}	
channelx	DMAMUX请求路由通道，参考 表3-333. 枚举 dmamux_multiplexer_channel_enum 。
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 3-382. 函数 dmamux_sync_id_config

函数名称	dmamux_sync_id_config
------	-----------------------

函数原型	void dmamux_sync_id_config(dmamux_multiplexer_channel_enum channelx, uint32_t id);
功能描述	配置DMAMUX同步输入标识
先决条件	-
被调用函数	-
输入参数{in}	
channelx	DMAMUX请求路由通道，参考 表3-333. 枚举 dmamux_multiplexer_channel_enum 。
输入参数{in}	
id	同步输入标识
DMAMUX_SYNC_EXTI0	同步输入信号为EXTI0
DMAMUX_SYNC_EXTI1	同步输入信号为EXTI1
DMAMUX_SYNC_EXTI2	同步输入信号为EXTI2
DMAMUX_SYNC_EXTI3	同步输入信号为EXTI3
DMAMUX_SYNC_EXTI4	同步输入信号为EXTI4
DMAMUX_SYNC_EXTI5	同步输入信号为EXTI5
DMAMUX_SYNC_EXTI6	同步输入信号为EXTI6
DMAMUX_SYNC_EXTI7	同步输入信号为EXTI7
DMAMUX_SYNC_EXTI8	同步输入信号为EXTI8
DMAMUX_SYNC_EXTI9	同步输入信号为EXTI9
DMAMUX_SYNC_EXTI10	同步输入信号为EXTI10
DMAMUX_SYNC_EXTI11	同步输入信号为EXTI11
DMAMUX_SYNC_EXTI12	同步输入信号为EXTI12
DMAMUX_SYNC_EXTI13	同步输入信号为EXTI13
DMAMUX_SYNC_EXTI14	同步输入信号为EXTI14
DMAMUX_SYNC_EXTI15	同步输入信号为EXTI15

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_CMP 0_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 3-383. 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3-333. 枚举 dmamux_mux_channel_enum 。
输入参数{in}	
id	DMA请求输入标识
DMA_REQUEST_M2M	内存到内存传输
DMA_REQUEST_GENERATOR0	DMAMUX请求生成通道0请求
DMA_REQUEST_GENERATOR1	DMAMUX请求生成通道1请求
DMA_REQUEST_GENERATOR2	DMAMUX请求生成通道2请求
DMA_REQUEST_GENERATOR3	DMAMUX请求生成通道3请求
DMA_REQUEST_GENERATOR4	DMAMUX请求生成通道4请求
DMA_REQUEST_GENERATOR5	DMAMUX请求生成通道5请求
DMA_REQUEST_GENERATOR6	DMAMUX请求生成通道6请求
DMA_REQUEST_GENERATOR7	DMAMUX请求生成通道7请求
DMA_REQUEST_ADC0	DMAMUX ADC0请求
DMA_REQUEST_ADC1	DMAMUX ADC1请求
DMA_REQUEST_TIMER0_CH0	DMAMUX TIMER0 CH0请求
DMA_REQUEST_TIMER0_CH1	DMAMUX TIMER0 CH1请求
DMA_REQUEST_TIMER0_CH2	DMAMUX TIMER0 CH2请求
DMA_REQUEST_TIMER0_CH3	DMAMUX TIMER0 CH3请求
DMA_REQUEST_TIMER0_MCH0	DMAMUX TIMER0 MCH0请求
DMA_REQUEST_TIMER0_MCH1	DMAMUX TIMER0 MCH1请求
DMA_REQUEST_TIMER0_MCH2	DMAMUX TIMER0 MCH2请求

<i>R0_MCH2</i>	
<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> <i>_RX</i>	DMAMUX SPI1 RX请求
<i>DMA_REQUEST_SPI1</i> <i>_TX</i>	DMAMUX SPI1 TX请求
<i>DMA_REQUEST_USA</i> <i>RT0_RX</i>	DMAMUX USART0 RX请求
<i>DMA_REQUEST_USA</i> <i>RT0_TX</i>	DMAMUX USART0 TX请求
<i>DMA_REQUEST_USA</i> <i>RT1_RX</i>	DMAMUX USART1 RX请求
<i>DMA_REQUEST_USA</i> <i>RT1_TX</i>	DMAMUX USART1 TX请求
<i>DMA_REQUEST_USA</i> <i>RT2_RX</i>	DMAMUX USART2 RX请求
<i>DMA_REQUEST_USA</i> <i>RT2_TX</i>	DMAMUX USART2 TX请求
<i>DMA_REQUEST_TIME</i> <i>R7_CH0</i>	DMAMUX TIMER7 CH0请求
<i>DMA_REQUEST_TIME</i> <i>R7_CH1</i>	DMAMUX TIMER7 CH1请求
<i>DMA_REQUEST_TIME</i> <i>R7_CH2</i>	DMAMUX TIMER7 CH2请求
<i>DMA_REQUEST_TIME</i> <i>R7_CH3</i>	DMAMUX TIMER7 CH3请求
<i>DMA_REQUEST_TIME</i> <i>R7_MCH0</i>	DMAMUX TIMER7 MCH0请求
<i>DMA_REQUEST_TIME</i>	DMAMUX TIMER7 MCH1请求

<i>R7_MCH1</i>	
<i>DMA_REQUEST_TIME</i> <i>R7_MCH2</i>	DMAMUX TIMER7 MCH2请求
<i>DMA_REQUEST_TIME</i> <i>R7_MCH3</i>	DMAMUX TIMER7 MCH3请求
<i>DMA_REQUEST_TIME</i> <i>R7_UP</i>	DMAMUX TIMER7 UP请求
<i>DMA_REQUEST_TIME</i> <i>R7_TRG</i>	DMAMUX TIMER7 TRG请求
<i>DMA_REQUEST_TIME</i> <i>R7_CMT</i>	DMAMUX TIMER7 CMT请求
<i>DMA_REQUEST_TIME</i> <i>R4_CH0</i>	DMAMUX TIMER4 CH0请求
<i>DMA_REQUEST_TIME</i> <i>R4_CH1</i>	DMAMUX TIMER4 CH1请求
<i>DMA_REQUEST_TIME</i> <i>R4_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_USART5_RX</i>	DMAMUX USART5 RX请求
<i>DMA_REQUEST_USART5_TX</i>	DMAMUX USART5 TX请求
<i>DMA_REQUEST_I2C2_RX</i>	DMAMUX I2C2 RX请求
<i>DMA_REQUEST_I2C2_TX</i>	DMAMUX I2C2 TX请求
<i>DMA_REQUEST_DCI</i>	DMAMUX DCI请求
<i>DMA_REQUEST_CAU_IN</i>	DMAMUX CAU IN请求
<i>DMA_REQUEST_CAU_OUT</i>	DMAMUX CAU OUT请求
<i>DMA_REQUEST_HAU_IN</i>	DMAMUX HAU IN请求
<i>DMA_REQUEST_UART6_RX</i>	DMAMUX UART6 RX请求
<i>DMA_REQUEST_UART6_TX</i>	DMAMUX UART6 TX请求
<i>DMA_REQUEST_UART7_RX</i>	DMAMUX UART7 RX请求
<i>DMA_REQUEST_UART7_TX</i>	DMAMUX UART7 TX请求
<i>DMA_REQUEST_SPI3_RX</i>	DMAMUX SPI3 RX请求
<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_SAI0_B0</i>	DMAMUX SAI0 B0请求
<i>DMA_REQUEST_SAI0_B1</i>	DMAMUX SAI0 B1请求
<i>DMA_REQUEST_RSPDIF_DATA</i>	DMAMUX RSPDIF DATA请求
<i>DMA_REQUEST_RSPDIF_CS</i>	DMAMUX RSPDIF CS请求
<i>DMA_REQUEST_HPDF_F_FLT0</i>	DMAMUX HPDF FLT0请求
<i>DMA_REQUEST_HPDF_F_FLT1</i>	DMAMUX HPDF FLT1请求

<i>DMA_REQUEST_HPDI F_FLT2</i>	DMAMUX HPDI FLT2请求
<i>DMA_REQUEST_HPDI F_FLT3</i>	DMAMUX HPDI FLT3请求
<i>DMA_REQUEST_TIME R14_CH0</i>	DMAMUX TIMER14 CH0请求
<i>DMA_REQUEST_TIME R14_CH1</i>	DMAMUX TIMER14 CH1 请求
<i>DMA_REQUEST_TIME R14_MCH0</i>	DMAMUX TIMER14 MCH0请求
<i>DMA_REQUEST_TIME R14_UP</i>	DMAMUX TIMER14 UP请求
<i>DMA_REQUEST_TIME R14_TRG</i>	DMAMUX TIMER14 TRG请求
<i>DMA_REQUEST_TIME R14_CMT</i>	DMAMUX TIMER14 CMT请求
<i>DMA_REQUEST_TIME R15_CH0</i>	DMAMUX TIMER15 CH0请求
<i>DMA_REQUEST_TIME R15_MCH0</i>	DMAMUX TIMER15 MCH0请求
<i>DMA_REQUEST_TIME R15_UP</i>	DMAMUX TIMER15 UP请求
<i>DMA_REQUEST_TIME R16_CH0</i>	DMAMUX TIMER16 CH0请求
<i>DMA_REQUEST_TIME R16_MCH0</i>	DMAMUX TIMER16 MCH0请求
<i>DMA_REQUEST_TIME R16_UP</i>	DMAMUX TIMER16 TRG请求
<i>DMA_REQUEST_ADC 2</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</i>	DMAMUX TIMER22 CH2请求

<i>R22_CH2</i>	
<i>DMA_REQUEST_TIME</i> <i>R22_CH3</i>	DMAMUX TIMER22 CH3请求
<i>DMA_REQUEST_TIME</i> <i>R22_UP</i>	DMAMUX TIMER22 UP请求
<i>DMA_REQUEST_TIME</i> <i>R22_TRG</i>	DMAMUX TIMER22 TRG请求
<i>DMA_REQUEST_TIME</i> <i>R23_CH0</i>	DMAMUX TIMER23 CH0请求
<i>DMA_REQUEST_TIME</i> <i>R23_CH1</i>	DMAMUX TIMER23 CH1 请求
<i>DMA_REQUEST_TIME</i> <i>R23_CH2</i>	DMAMUX TIMER23 CH2请求
<i>DMA_REQUEST_TIME</i> <i>R23_CH3</i>	DMAMUX TIMER23 CH3请求
<i>DMA_REQUEST_TIME</i> <i>R23_UP</i>	DMAMUX TIMER23 UP请求
<i>DMA_REQUEST_TIME</i> <i>R23_TRG</i>	DMAMUX TIMER23 TRG请求
<i>DMA_REQUEST_TIME</i> <i>R30_CH0</i>	DMAMUX TIMER30 CH0请求
<i>DMA_REQUEST_TIME</i> <i>R30_CH1</i>	DMAMUX TIMER30 CH1 请求
<i>DMA_REQUEST_TIME</i> <i>R30_CH2</i>	DMAMUX TIMER30 CH2请求
<i>DMA_REQUEST_TIME</i> <i>R30_CH3</i>	DMAMUX TIMER30 CH3请求
<i>DMA_REQUEST_TIME</i> <i>R30_UP</i>	DMAMUX TIMER30 UP请求
<i>DMA_REQUEST_TIME</i> <i>R30_TRG</i>	DMAMUX TIMER30 TRG请求
<i>DMA_REQUEST_TIME</i> <i>R31_CH0</i>	DMAMUX TIMER31 CH0请求
<i>DMA_REQUEST_TIME</i> <i>R31_CH1</i>	DMAMUX TIMER31 CH1 请求
<i>DMA_REQUEST_TIME</i> <i>R31_CH2</i>	DMAMUX TIMER31 CH2请求
<i>DMA_REQUEST_TIME</i> <i>R31_CH3</i>	DMAMUX TIMER31 CH3请求
<i>DMA_REQUEST_TIME</i> <i>R31_UP</i>	DMAMUX TIMER31 UP请求
<i>DMA_REQUEST_TIME</i> <i>R31_TRG</i>	DMAMUX TIMER31 TRG请求

<i>DMA_REQUEST_TIME</i> <i>R40_CH0</i>	DMAMUX TIMER40 CH0请求
<i>DMA_REQUEST_TIME</i> <i>R40_MCH0</i>	DMAMUX TIMER40 MCH0请求
<i>DMA_REQUEST_TIME</i> <i>R40_CMT</i>	DMAMUX TIMER40 CMT请求
<i>DMA_REQUEST_TIME</i> <i>R40_UP</i>	DMAMUX TIMER40 UP请求
<i>DMA_REQUEST_TIME</i> <i>R41_CH0</i>	DMAMUX TIMER41 CH0请求
<i>DMA_REQUEST_TIME</i> <i>R41_MCH0</i>	DMAMUX TIMER41 MCH0请求
<i>DMA_REQUEST_TIME</i> <i>R41_CMT</i>	DMAMUX TIMER41 CMT请求
<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>	DMAMUX TIMER51 UP请求

<i>R51_UP</i>	
<i>DMA_REQUEST_SAI1</i> <i>_B0</i>	DMAMUX SAI1 B0请求
<i>DMA_REQUEST_SAI1</i> <i>_B1</i>	DMAMUX SAI1 B1请求
<i>DMA_REQUEST_SAI2</i> <i>_B0</i>	DMAMUX SAI2 B0请求
<i>DMA_REQUEST_SAI2</i> <i>_B1</i>	DMAMUX SAI2 B1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
<i>DMA_REQUEST_CAN</i> <i>2</i>	DMAMUX CAN2请求
<i>DMA_REQUEST_TIME</i> <i>R40_CH1</i>	DMAMUX TIMER40 CH1 请求
<i>DMA_REQUEST_TIME</i> <i>R40_TRG</i>	DMAMUX TIMER40 TRG请求
<i>DMA_REQUEST_TIME</i> <i>R41_CH1</i>	DMAMUX TIMER41 CH1 请求
<i>DMA_REQUEST_TIME</i> <i>R41_TRG</i>	DMAMUX TIMER41 TRG请求
<i>DMA_REQUEST_TIME</i> <i>R42_CH1</i>	DMAMUX TIMER42 CH1 请求
<i>DMA_REQUEST_TIME</i> <i>R42_TRG</i>	DMAMUX TIMER42 TRG请求
<i>DMA_REQUEST_TIME</i> <i>R43_CH1</i>	DMAMUX TIMER43 CH1 请求
<i>DMA_REQUEST_TIME</i> <i>R43_TRG</i>	DMAMUX TIMER43 TRG请求
<i>DMA_REQUEST_TIME</i> <i>R44_CH1</i>	DMAMUX TIMER44 CH1 请求
<i>DMA_REQUEST_TIME</i> <i>R44_TRG</i>	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 3-384. 函数 dmamux_trigger_polarity_config

函数名称	dmamux_trigger_polarity_config
函数原型	void dmamux_trigger_polarity_config(dmamux_generator_channel_enum channelx, uint32_t polarity);
功能描述	配置DMAMUX触发输入的有效边沿
先决条件	-
被调用函数	-
输入参数{in}	
channelx	DMAMUX请求生成通道，参考 表3-334. 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_EDGE	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 3-385. 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3-334. 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 3-386. 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3-334. 枚举 dmamux_generator_channel_enum 。
输入参数{in}	
id	触发输入标识
DMAMUX_TRIGGER_EXTI0	触发输入为EXTI0

DMAMUX_TRIGGER_ EXTI1	触发输入为EXTI1
DMAMUX_TRIGGER_ EXTI2	触发输入为EXTI2
DMAMUX_TRIGGER_ EXTI3	触发输入为EXTI3
DMAMUX_TRIGGER_ EXTI4	触发输入为EXTI4
DMAMUX_TRIGGER_ EXTI5	触发输入为EXTI5
DMAMUX_TRIGGER_ EXTI6	触发输入为EXTI6
DMAMUX_TRIGGER_ EXTI7	触发输入为EXTI7
DMAMUX_TRIGGER_ EXTI8	触发输入为EXTI8
DMAMUX_TRIGGER_ EXTI9	触发输入为EXTI9
DMAMUX_TRIGGER_ EXTI10	触发输入为EXTI10
DMAMUX_TRIGGER_ EXTI11	触发输入为EXTI11
DMAMUX_TRIGGER_ EXTI12	触发输入为EXTI12
DMAMUX_TRIGGER_ EXTI13	触发输入为EXTI13
DMAMUX_TRIGGER_ EXTI14	触发输入为EXTI14
DMAMUX_TRIGGER_ EXTI15	触发输入为EXTI15
DMAMUX_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

<i>EVT_OUT6</i>	
<i>DMAMUX_TRIGGER_</i> <i>EVT_OUT7</i>	触发输入为Evt7_out
<i>DMAMUX_TRIGGER_</i> <i>RTC_WAKEUP</i>	触发输入为唤醒
<i>DMAMUX_TRIGGER_</i> <i>CMP0_OUTPUT</i>	触发输入为CMP0输出
<i>DMAMUX_TRIGGER_</i> <i>CMP1_OUTPUT</i>	触发输入为CMP1输出
<i>DMAMUX_TRIGGER_I</i> <i>2C0_WAKEUP</i>	触发输入为I2C0唤醒
<i>DMAMUX_TRIGGER_I</i> <i>2C1_WAKEUP</i>	触发输入为I2C1唤醒
<i>DMAMUX_TRIGGER_I</i> <i>2C2_WAKEUP</i>	触发输入为I2C2唤醒
<i>DMAMUX_TRIGGER_I</i> <i>2C3_WAKEUP</i>	触发输入为I2C3唤醒
<i>DMAMUX_TRIGGER_I</i> <i>2C0_INT_EVENT</i>	触发输入为I2C0中断事件
<i>DMAMUX_TRIGGER_I</i> <i>2C1_INT_EVENT</i>	t触发输入为I2C1中断事件
<i>DMAMUX_TRIGGER_I</i> <i>2C2_INT_EVENT</i>	触发输入为I2C2中断事件
<i>DMAMUX_TRIGGER_I</i> <i>2C3_INT_EVENT</i>	t触发输入为I2C3中断事件
<i>DMAMUX_TRIGGER_</i> <i>ADC2_INT</i>	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 3-387. 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3-336. 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 3-388. 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3-336. 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 3-389. 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3-335. 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 3-390. 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3-335. 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 3-391. 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3-337. 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 3-392. 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3-337. 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);
```

3.13. 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[3.13.1](#)描述了EDOUT的寄存器列表，章节[3.13.2](#)对EDOUT库函数进

行说明。

3.13.1. 外设寄存器描述

EDOUT寄存器列表如下表所示：

表 3-393. 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

3.13.2. 外设库函数说明

EDOUT库函数列表如下表所示：

表 3-394. EDOUT 库函数

库函数名称	库函数描述
edout_deinit	复位EDOUT
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 3-395. 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 3-396. 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LOC_MAX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 3-397. 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 3-398. 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 3-399. 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 3-400. 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 3-401. 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 3-402. 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LOC_MAX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 3-403. 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 3-404. 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 3-405. 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);
```

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

3.14.1. 外设寄存器描述

EFUSE寄存器列表如下表所示:

表 3-406. 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)

寄存器名称	寄存器描述
EFUSE_AES_KEYx (x = 0...3)	AES 密钥寄存器 x(x = 0...3)
EFUSE_USER_DATAx (x = 0...3)	用户数据寄存器 x(x = 0...3)

3.14.2. 外设库函数说明

EFUSE库函数列表如下表所示：

表 3-407. 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
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 3-408. 枚举类型 efuse_system_para_size_enum

成员名称	功能描述
USER_CTL_SIZE	用户控制段大小
MCU_RESERVED_SIZE	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 3-409. 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 3-410. 枚举类型 `efuse_state_enum`

Member name	Function description
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 3-411. 枚举类型 `efuse_interrupt_flag_enum`

Member name	Function description
EFUSE_INT_FLAG_ILLEGAL_ACCESS_ERR	illegal access error interrupt flag
EFUSE_INT_FLAG_PROGRAM_OPERATION_COMPLETE	programming operation completion interrupt flag
EFUSE_INT_FLAG_READ_OPERATION_COMPLETE	read operation completion interrupt flag
EFUSE_INT_FLAG_PROGRAM_VOLTAGE_SETTING_ERROR	program voltage setting error flag

函数 `efuse_read`

函数 `efuse_read` 描述见下表：

表 3-412. 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	-
被调用函数	-

输入参数{in}	
ef_addr	要读取的系统参数的起始地址
<i>USER_CTL_EFADDR</i>	用户控制段起始地址
<i>MCU_RESERVED_EFADDR</i>	MCU 保留段起始地址
<i>DP_EFADDR</i>	调试密钥起始地址
<i>USER_DATA_EFADDR</i>	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）
<i>USER_CTL_SIZE</i>	用户控制段大小
<i>MCU_RESERVED_SIZE</i>	MCU 保留段大小
<i>DP_SIZE</i>	调试密钥大小
<i>USER_DATA_SIZE</i>	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 3-413. 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 3-414. 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 3-415. 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 3-416. 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	写所有的调试秘钥
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 3-417. 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 3-418. 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 3-419. 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 3-420. 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 3-421. 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 3-422. 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 3-423. 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 3-424. 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 3-425. 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
EFUSE_FLAG_ILLEGAL_ACCESS_ERR_CLR	非法访问错误标志位
EFUSE_FLAG_PROG RAM_COMPLETE_CLR	清除写操作完成标志位
EFUSE_FLAG_READ_ COMPLETE_CLR	清除读操作完成标志位
EFUSE_FLAG_PROG RAM_VOLTAGE_ERR _CLR	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 3-426. 函数 `efuse_interrupt_enable`

函数名称	<code>efuse_interrupt_enable</code>
函数原型	<code>void efuse_interrupt_enable(uint32_t source);</code>
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
<code>EFUSE_INT_ILLEGAL_ACCESS_ERR</code>	使能非法访问错误中断
<code>EFUSE_INT_PROGRAM_COMPLETE</code>	使能写操作完成中断
<code>EFUSE_INT_READ_COMPLETE</code>	使能读操作完成中断
<code>EFUSE_INT_PROGRAM_VOLTAGE_ERR</code>	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 3-427. 函数 `efuse_interrupt_disable`

函数名称	<code>efuse_interrupt_disable</code>
函数原型	<code>void efuse_interrupt_disable(uint32_t source);</code>
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
<code>EFUSE_INT_ILLEGAL_ACCESS_ERR</code>	失能非法访问错误中断
<code>EFUSE_INT_PROGRAM_COMPLETE</code>	失能写操作完成中断

<i>M_COMPLETE</i>	
<i>EFUSE_INT_READ_COMPLETE</i>	失能读操作完成中断
<i>EFUSE_INT_PROGRAM_VOLTAGE_ERR</i>	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 3-428. 函数 `efuse_interrupt_flag_get`

函数名称	<code>efuse_interrupt_flag_get</code>
函数原型	<code>FlagStatus efuse_interrupt_flag_get(uint32_t int_flag);</code>
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
<i>EFUSE_INT_FLAG_ILLEGAL_ACCESS_ERR</i>	非法访问错误中断标志
<i>EFUSE_INT_FLAG_PROGRAM_COMPLETE</i>	写操作完成中断标志
<i>EFUSE_INT_FLAG_READ_COMPLETE</i>	读操作完成中断标志
<i>EFUSE_INT_FLAG_PROGRAM_VOLTAGE_ERR</i>	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 3-429. 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_READ_COMPLETE_CLR	清除读操作完成中断标志
EFUSE_INT_FLAG_PROGRAM_VOLTAGE_ERROR_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);
```

3.15. ENET

该以太网模块包含两个10/100Mbps以太网MAC（媒体访问控制器），采用DMA优化数据帧的发送与接收性能，支持MII（媒体独立接口）与RMII（简化的媒体独立接口）两种与物理层（PHY）通讯的标准接口，实现以太网数据帧的发送与接收。章节[3.15.1](#)描述了ENET的寄存器列表，章节[3.15.2](#)对ENET库函数进行说明。

3.15.1. 外设寄存器描述

ENET寄存器列表如下表所示：

表 3-430. ENET 寄存器

寄存器名称	寄存器描述
ENET_MAC_CFG	MAC配置寄存器
ENET_MAC_FRMF	MAC帧过滤器寄存器
ENET_MAC_HLH	MAC hash列表高寄存器
ENET_MAC_HLL	MAC hash列表低寄存器
ENET_MAC_PHY_CTL	MAC PHY控制寄存器
ENET_MAC_PHY_DATA	MAC PHY数据寄存器
ENET_MAC_FCTL	MAC流控寄存器
ENET_MAC_VLT	MAC VLAN标签寄存器
ENET_MAC_RWFF	MAC远程唤醒帧过滤器寄存器
ENET_MAC_WUM	MAC唤醒管理寄存器
ENET_MAC_DBG	MAC调试寄存器
ENET_MAC_INTF	MAC中断状态寄存器
ENET_MAC_INTMSK	MAC中断屏蔽寄存器
ENET_MAC_ADDR0H	MAC地址0高寄存器
ENET_MAC_ADDR0L	MAC地址0低寄存器
ENET_MAC_ADDR1H	MAC地址1高寄存器
ENET_MAC_ADDR1L	MAC地址1低寄存器
ENET_MAC_ADDT2H	MAC地址2高寄存器
ENET_MAC_ADDR2L	MAC地址2低寄存器
ENET_MAC_ADDR3H	MAC地址3高寄存器
ENET_MAC_ADDR3L	MAC地址3低寄存器
ENET_MAC_FCTH	MAC流控阈值寄存器
ENET_MSC_CTL	MSC控制寄存器
ENET_MSC_RINTF	MSC接收中断状态寄存器
ENET_MSC_TINTF	MSC发送中断状态寄存器
ENET_MSC_RINT	MSC接收中断屏蔽寄存器

寄存器名称	寄存器描述
MSK	
ENET_MSC_TINTMSK	MSC发送中断屏蔽寄存器
ENET_MSC_SCCNT	MSC 1次冲突后发送"好"帧的计数器寄存器
ENET_MSC_MSCCNT	MSC 1次以上冲突后发送"好"帧的计数器寄存器
ENET_MSC_TGFCNT	MSC发送"好"帧计数器寄存器
ENET_MSC_RFCECNT	MSC CRC错误接收帧计数器寄存器
ENET_MSC_RFAECNT	MSC对齐错误接收帧计数器寄存器
ENET_MSC_RGUFcnt	MSC"好"单播帧接收帧计数器寄存器
ENET_PTP_TSCTL	PTP时间戳控制寄存器
ENET_PTP_SSINC	PTP亚秒递增寄存器
ENET_PTP_TSH	PTP时间戳高寄存器
ENET_PTP_TSL	PTP时间戳低寄存器
ENET_PTP_TSUH	PTP时间戳高更新寄存器
ENET_PTP_TSUL	PTP时间戳低更新寄存器
ENET_PTP_TSADDEND	PTP时间戳加数寄存器
ENET_PTP_ETH	PTP期望时间高寄存器
ENET_PTP_ETL	PTP期望时间低寄存器
ENET_PTP_TSF	PTP时间戳标志寄存器
ENET_PTP_PPSCTL	PTP PPS控制寄存器
ENET_DMA_BCTL	DMA总线控制寄存器
ENET_DMA_TPEN	DMA发送查询使能寄存器
ENET_DMA_RPEN	DMA接收查询使能寄存器
ENET_DMA_RDTADDR	DMA接收描述符列表地址寄存器
ENET_DMA_TDTADDR	DMA发送描述符列表地址寄存器
ENET_DMA_STAT	DMA状态寄存器
ENET_DMA_CTL	DMA控制寄存器
ENET_DMA_INTEN	DMA中断使能寄存器
ENET_DMA_MFBOCNT	DMA丢失帧和缓存溢出计数器寄存器
ENET_DMA_RSWD	DMA接收接收状态看门狗计数器寄存器

寄存器名称	寄存器描述
C	
ENET_DMA_CTDA DDR	DMA当前发送描述符地址寄存器
ENET_DMA_CRDA DDR	DMA当前接收描述符地址寄存器
ENET_DMA_CTBA DDR	DMA当前发送缓存地址寄存器
ENET_DMA_CRBA DDR	DMA当前接收缓存地址寄存器

3.15.2. 外设库函数说明

ENET库函数列表如下表所示：

表 3-431. ENET 库函数

库函数名称	库函数描述
常用函数	
enet_deinit	复位ENET模块及相关软件初始化所需结构体
enet_initpara_config	配置ENET模块的各类不常用功能。当enet_init()函数无法满足所需实现功能时调用，必须在enet_init()函数之前调用
enet_init	ENET模块初始化，配置用户最关心的功能
enet_software_reset	复位ENET寄存器，并检测CLK_TX/CLK_RX信号
enet_rxfree_size_get	检测接收帧是否有错误，正确时返回帧长度
enet_descriptors_chain_init	初始化DMA接收/发送描述符为链模式
enet_descriptors_ring_init	初始化DMA接收/发送描述符为环模式
enet_frame_receive	处理当前接收到的帧，并将当前描述符中存储的接收帧数据拷贝到指定区域
enet_frame_transmit	将指定区域内的数据拷贝到当前发送描述符中，并发送
enet_transmit_checksum_config	配置发送帧校验和模式
enet_enable	ENET Tx/Rx功能使能（包括ENET外设内的MAC和DMA模块）
enet_disable	ENET Tx/Rx功能禁能（包括ENET外设内的MAC和DMA模块）
enet_mac_address_set	配置MAC地址
enet_mac_address_get	获取MAC地址
MAC功能函数	
enet_tx_enable	ENET发送功能使能（包括ENET外设内的MAC和DMA模块）
enet_tx_disable	ENET发送功能禁能（包括ENET外设内的MAC和DMA模块）
enet_rx_enable	ENET接收功能使能（包括ENET外设内的MAC和DMA模块）
enet_rx_disable	ENET接收功能禁能（包括ENET外设内的MAC和DMA模块）
enet_registers_get	获取指定范围ENET寄存器值
enet_debug_status_get	获取ENET调试状态信息

库函数名称	库函数描述
enet_address_filter_enable	MAC地址过滤器使能
enet_address_filter_disable	MAC地址过滤器禁能
enet_address_filter_config	配置MAC地址过滤器模式
enet_phy_config	PHY接口配置（配置SMI时钟并复位PHY芯片）
enet_phy_write_read	写/读PHY寄存器
enet_phyloopback_enable	使能PHY芯片回环模式
enet_phyloopback_disable	禁能PHY芯片回环模式
enet_forward_feature_enable	使能ENET帧通过相关功能
enet_forward_feature_disable	禁能ENET帧通过相关功能
enet_fliter_feature_enable	使能ENET帧过滤器相关功能
enet_fliter_feature_disable	禁能ENET帧过滤器相关功能
流控功能函数	
enet_pauseframe_generate	生成暂停帧，使能发送流控功能后ENET模块将发送暂停帧
enet_pauseframe_detect_config	配置暂停帧检测类型
enet_pauseframe_config	配置暂停帧参数
enet_flowcontrol_threshold_config	配置流控阈值
enet_flowcontrol_feature_enable	使能ENET流控相关功能
enet_flowcontrol_feature_disable	禁能ENET流控相关功能
DMA功能函数	
enet_dmaprocess_state_get	获取DMA发送/接收流程状态
enet_dmaprocess_resume	DMA发送/接收查询使能
enet_rxprocess_check_recovery	检测并恢复接收流程
enet_txfifo_flush	刷新ENET发送FIFO，并等待刷新操作完成
enet_current_desc_address_get	获取当前发送/接收描述符地址、当前缓冲区地址、描述符列表首地址
enet_desc_information_get	获取发送/接收描述符详细信息
enet_missed_frame_counter_get	获取接收丢弃帧数
描述符功能函数	
enet_desc_flag_get	获取ENET模块DMA描述符标志位
enet_desc_flag_set	设置ENET模块DMA描述符标志位
enet_desc_flag_clear	清除ENET模块DMA描述符标志位
enet_rx_desc_immediate_receive_complete_interrupt	接收完成后立即置位ENET_DMA_STAT寄存器的RS位
enet_rx_desc_delay_receive_complete_interrupt	接收完成后延迟指定时间再置位ENET_DMA_STAT寄存器的RS位
enet_rxframe_drop	丢弃当前接收到的帧
enet_dma_feature_enable	使能ENET模块DMA相关功能
enet_dma_feature_disable	禁能ENET模块DMA相关功能
增强型描述符功能函数	
enet_rx_desc_enhanced_status_get	获取接收描述符增强状态标志位信息
enet_desc_select_enhanced_mode	配置DMA描述符为增强型描述符

库函数名称	库函数描述
enet_ptp_enhanced_descriptors_chain_init	初始化具有PTP功能的增强型DMA接收/发送描述符为链模式
enet_ptp_enhanced_descriptors_ring_init	初始化具有PTP功能的增强型DMA接收/发送描述符为环模式
enet_ptpframe_receive_enhanced_mode	在PTP模式下处理当前接收到的帧，并将当前增强型描述符中存储的接收帧数据和时间戳拷贝到指定区域
enet_ptpframe_transmit_enhanced_mode	在PTP模式下将指定区域内的数据拷贝到当前增强型发送描述符中，并同时时间戳一起发送
常规型描述符功能函数	
enet_desc_select_normal_mode	配置DMA描述符为常规型描述符
enet_ptp_normal_descriptors_chain_init	初始化具有PTP功能的普通型DMA接收/发送描述符为链模式
enet_ptp_normal_descriptors_ring_init	初始化具有PTP功能的普通型DMA接收/发送描述符为环模式
enet_ptpframe_receive_normal_mode	在PTP模式下处理当前接收到的帧，并将当前普通型描述符中存储的接收帧数据和时间戳拷贝到指定区域
enet_ptpframe_transmit_normal_mode	在PTP模式下将指定区域内的数据拷贝到当前普通型发送描述符中，并同时时间戳一起发送
WUM功能函数	
enet_wum_filter_register_pointer_reset	远程唤醒帧过滤器寄存器指针复位
enet_wum_filter_config	配置远程唤醒帧寄存器
enet_wum_feature_enable	使能ENET模块唤醒管理相关功能
enet_wum_feature_disable	禁能ENET模块唤醒管理相关功能
MSC功能函数	
enet_msc_counters_reset	复位MAC统计计数器组
enet_msc_feature_enable	使能MAC统计计数器相关功能
enet_msc_feature_disable	禁能MAC统计计数器相关功能
enet_msc_counters_preset_config	配置MAC统计计数器的预设模式
enet_msc_counters_get	获取MAC相关统计计数器值
PTP功能函数	
enet_ptp_subsecond_2_nanosecond	将亚秒值变为纳秒值
enet_ptp_nanosecond_2_subsecond	将纳秒值变为亚秒值
enet_ptp_feature_enable	使能PTP相关功能
enet_ptp_feature_disable	禁能PTP相关功能
enet_ptp_timestamp_function_config	配置PTP时间戳相关功能
enet_ptp_subsecond_increment_config	配置PTP系统时间亚秒增加值
enet_ptp_timestamp_addend_config	精调模式下PTP时钟频率校准配置
enet_ptp_timestamp_update_config	初始化时用于替换系统时间，在更新时表示在系统时间上加上或减去的秒值
enet_ptp_expected_time_config	配置PTP期望时间

库函数名称	库函数描述
enet_ptp_system_time_get	获取PTP当前系统时间
enet_ptp_pps_output_frequency_config	配置PPS输出频率
enet_ptp_start	配置和启动PTP时间戳计数器
enet_ptp_finecorrection_adjfreq	通过PTP时间戳加数寄存器精调系统时间
enet_ptp_coarsecorrection_systime_update	粗调系统时间
enet_ptp_finecorrection_settime	精调系统时间
enet_ptp_flag_get	获取PTP标志位状态
内部函数	
enet_initpara_reset	复位 ENET initpara struct, 需在enet_initpara_config()函数前调用
中断和标志位相关函数	
enet_flag_get	获取ENET模块MAC/MSC/PTP/DMA状态标志位
enet_flag_clear	清除ENET状态标志位
enet_interrupt_enable	使能ENET模块MAC/MSC/DMA中断
enet_interrupt_disable	禁能ENET模块MAC/MSC/DMA中断
enet_interrupt_flag_get	获取ENET模块MAC/MSC/DMA中断标志位
enet_interrupt_flag_clear	禁能ENET模块DMA中断标志位

结构体 enet_initpara_struct

表 3-432. 结构体 enet_initpara_struct

成员名称	功能描述
option_enable	选择配置功能
forward_frame	发送帧相关配置参数
dmabus_mode	DMA总线模式相关配置参数
dma_maxburst	DMA最大传输相关配置参数
dma_arbitration	DMA接收/发送仲裁相关配置参数
store_forward_mode	存储转发模式相关配置参数
dma_function	DMA控制相关配置参数
vlan_config	VLAN标签相关配置参数
flow_control	流控相关配置参数
hashtable_high	hash列表高32位相关配置参数
hashtable_low	hash列表低32位相关配置参数
framesfilter_mode	帧过滤器控制相关配置参数
halfduplex_param	半双工相关配置参数
timer_config	帧发送计数器相关配置参数
interframegap	帧间隔相关配置参数

结构体 `enet_descriptors_struct`

表 3-433. 结构体 `enet_descriptors_struct`

成员名称	功能描述
status	描述符状态位
control_buffer_size	描述符控制位及缓冲区1、2长度
buffer1_addr	缓冲区1地址指针/时间戳低
buffer2_next_desc_addr	缓冲区2或下一描述符地址指针/时间戳高
extended_status	增强型描述符状态
reserved	保留
timestamp_low	增强型描述符时间戳低位
timestamp_high	增强型描述符时间戳高位

结构体 `enet_ptp_systime_struct`

表 3-434. 结构体 `enet_ptp_systime_struct`

成员名称	功能描述
second	系统时间（单位秒）
subsecond	系统时间（单位纳秒）
sign	系统时间符号位

枚举类型 `enet_flag_enum`

表 3-435. 枚举类型 `enet_flag_enum`

成员名称	功能描述
ENET_MAC_FLAG_MPKR	接收到魔术帧标志位
ENET_MAC_FLAG_WUFR	接收到唤醒帧标志位
ENET_MAC_FLAG_FLOWCONTROL	流控状态标志位
ENET_MAC_FLAG_WUM	WUM状态标志位
ENET_MAC_FLAG_MSC	MSC状态标志位
ENET_MAC_FLAG_MSCR	MSC接收状态标志位
ENET_MAC_FLAG_MSCT	MSC发送状态标志位
ENET_MAC_FLAG_TMST	时间戳触发状态标志位
ENET_PTP_FLAG_	时间戳秒计数溢出标志位

TSSCO	
ENET_PTP_FLAG_TTM	目标时间匹配标志位
ENET_MSC_FLAG_RFCE	接收帧CRC错误标志位
ENET_MSC_FLAG_RFAE	接收帧对齐错误标志位
ENET_MSC_FLAG_RGUF	接收到“好”的单播帧标志位
ENET_MSC_FLAG_TGFSC	发送“好”的帧时仅遇到1个冲突标志位
ENET_MSC_FLAG_TGFMS	发送“好”的帧时遇到1个以上冲突
ENET_MSC_FLAG_TGF	发送“好”的帧标志位
ENET_DMA_FLAG_TS	发送状态标志位
ENET_DMA_FLAG_TPS	发送流程停止状态标志位
ENET_DMA_FLAG_TBU	发送缓冲区不可用状态标志位
ENET_DMA_FLAG_TJT	发送jabber超时状态标志位
ENET_DMA_FLAG_RO	接收溢出状态标志位
ENET_DMA_FLAG_TU	发送下溢状态标志位
ENET_DMA_FLAG_RS	接收状态标志位
ENET_DMA_FLAG_RBU	接收缓冲区不可用状态标志位
ENET_DMA_FLAG_RPS	接受流程停止状态标志位
ENET_DMA_FLAG_RWT	接收看门狗超时状态标志位
ENET_DMA_FLAG_ET	早发送状态标志位
ENET_DMA_FLAG_FBE	致命总线错误状态标志位
ENET_DMA_FLAG_ER	早接收状态标志位
ENET_DMA_FLAG_AI	异常中断汇总标志位

ENET_DMA_FLAG_NI	正常中断汇总标志位
ENET_DMA_FLAG_EB_DMA_ERROR	DMA错误标志位
ENET_DMA_FLAG_EB_TRANSFER_ERROR	发送错误标志位
ENET_DMA_FLAG_EB_ACCESS_ERROR	DMA访问错误标志位
ENET_DMA_FLAG_MSC	MSC状态标志位
ENET_DMA_FLAG_WUM	WUM状态标志位
ENET_DMA_FLAG_TST	时间戳触发状态标志位

枚举类型 `enet_flag_clear_enum`

表 3-436. 枚举类型 `enet_flag_clear_enum`

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

ENET_DMA_FLAG_FBE_CLR	致命总线错误状态标志位清除
ENET_DMA_FLAG_ER_CLR	早接收状态标志位清除
ENET_DMA_FLAG_AI_CLR	异常中断汇总标志位清除
ENET_DMA_FLAG_NI_CLR	正常中断汇总标志位清除

枚举类型 `enet_int_enum`

表 3-437. 枚举类型 `enet_int_enum`

成员名称	功能描述
ENET_MAC_INT_WUMIM	WUM中断屏蔽
ENET_MAC_INT_TSTMIM	时间戳触发中断屏蔽
ENET_MSC_INT_RFCEIM	接收帧CRC错误中断屏蔽
ENET_MSC_INT_RFAEIM	接收帧对齐错误中断屏蔽
ENET_MSC_INT_RGUFIM	接收“好”单播帧中断屏蔽
ENET_MSC_INT_TGFSCIM	仅遇到1个冲突后发送“好”帧中断屏蔽
ENET_MSC_INT_TGFMSCIM	遇到1个以上冲突后发送“好”帧中断屏蔽
ENET_MSC_INT_TGFIM	发送“好”的帧的中断屏蔽
ENET_DMA_INT_TIE	发送中断使能
ENET_DMA_INT_TPSIE	发送流程停止中断使能
ENET_DMA_INT_TBUIE	发送缓冲区不可用中断使能
ENET_DMA_INT_TJTIE	发送jabber超时中断使能
ENET_DMA_INT_ROIE	接收溢出中断使能
ENET_DMA_INT_TUIE	发送下溢中断使能
ENET_DMA_INT_RIE	接收中断使能

ENET_DMA_INT_RBUIE	接收缓冲区不可用中断使能
ENET_DMA_INT_RPSIE	接收流程停止中断使能
ENET_DMA_INT_RWTIE	接收看门狗超时中断使能
ENET_DMA_INT_ETIE	早发送中断使能
ENET_DMA_INT_FBEIE	致命总线错误中断使能
ENET_DMA_INT_ERIE	早接收中断使能
ENET_DMA_INT_AIE	异常中断汇总使能
ENET_DMA_INT_NIE	正常中断汇总使能

枚举类型 enet_int_flag_enum

表 3-438. 枚举类型 enet_int_flag_enum

成员名称	功能描述
ENET_MAC_INT_FLAG_WUM	WUM中断标志位
ENET_MAC_INT_FLAG_MSC	MSC中断标志位
ENET_MAC_INT_FLAG_MSCR	MSC接收中断标志位
ENET_MAC_INT_FLAG_MSCT	MSC发送中断标志位
ENET_MAC_INT_FLAG_TMST	时间戳触发中断标志位
ENET_MSC_INT_FLAG_RFCE	接收帧CRC错误中断标志位
ENET_MSC_INT_FLAG_RFAE	接收帧对齐错误中断标志位
ENET_MSC_INT_FLAG_RGUF	接收“好”单播帧中断标志位
ENET_MSC_INT_FLAG_TGFSC	仅遇到1个冲突后发送“好”帧中断标志位
ENET_MSC_INT_FLAG_TGFMSC	遇到1个以上冲突后发送“好”帧中断标志位
ENET_MSC_INT_FLAG_TGF	发送“好”的帧的中断标志位

ENET_DMA_INT_F LAG_TS	发送中断标志位
ENET_DMA_INT_F LAG_TPS	发送流程停止中断标志位
ENET_DMA_INT_F LAG_TBU	发送缓冲区不可用中断标志位
ENET_DMA_INT_F LAG_TJT	发送jabber超时中断标志位
ENET_DMA_INT_F LAG_RO	接收溢出中断标志位
ENET_DMA_INT_F LAG_TU	发送下溢中断标志位
ENET_DMA_INT_F LAG_RS	接收中断标志位
ENET_DMA_INT_F LAG_RBU	接收缓冲区不可用中断标志位
ENET_DMA_INT_F LAG_RPS	接收流程停止中断标志位
ENET_DMA_INT_F LAG_RWT	接收看门狗超时中断标志位
ENET_DMA_INT_F LAG_ET	早发送中断标志位
ENET_DMA_INT_F LAG_FBE	致命总线错误中断标志位
ENET_DMA_INT_F LAG_ER	早接收中断标志位
ENET_DMA_INT_F LAG_AI	异常中断汇总中断标志位
ENET_DMA_INT_F LAG_NI	正常中断汇总中断标志位
ENET_DMA_INT_F LAG_MSC	MSC中断标志位
ENET_DMA_INT_F LAG_WUM	WUM中断标志位
ENET_DMA_INT_F LAG_TST	时间戳触发中断标志位

枚举类型 enet_int_flag_clear_enum

表 3-439. 枚举类型 enet_int_flag_clear_enum

成员名称	功能描述
ENET_DMA_INT_F LAG_TS_CLR	发送中断标志位清除

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

枚举类型 enet_desc_reg_enum

表 3-440. 枚举类型 enet_desc_reg_enum

成员名称	功能描述
ENET_RX_DESC_T ABLE	接收描述符列表首地址
ENET_RX_CURRE NT_DESC	当前DMA控制器使用的接收描述符地址
ENET_RX_CURRE NT_BUFFER	当前DMA控制器使用的接收描述符缓冲区地址
ENET_TX_DESC_T ABLE	发送描述符列表首地址
ENET_TX_CURRE NT_DESC	当前DMA控制器使用的发送描述符地址

ENET_TX_CURRE NT_BUFFER	当前DMA控制器使用的发送描述符缓冲区地址
----------------------------	-----------------------

枚举类型 `enet_msc_counter_enum`

表 3-441. 枚举类型 `enet_msc_counter_enum`

成员名称	功能描述
ENET_MSC_TX_S CCNT	MSC 1次冲突后发送“好”帧的计数器
ENET_MSC_TX_M SCCNT	MSC 1次以上冲突后发送“好”帧的计数器
ENET_MSC_TX_T GFCNT	MSC发送“好”帧计数器
ENET_MSC_RX_R FCECNT	MSC CRC错误接收帧计数器
ENET_MSC_RX_R FAECNT	MSC对齐错误接收帧计数器
ENET_MSC_RX_R GUFCNT	MSC“好”单播帧接收帧计数器

枚举类型 `enet_option_enum`

表 3-442. 枚举类型 `enet_option_enum`

成员名称	功能描述
FORWARD_OPTIO N	选择配置帧通过功能相关参数
DMABUS_OPTION	选择配置DMA总线模式相关参数
DMA_MAXBURST_ OPTION	选择配置DMA最大突发传输相关参数
DMA_ARBITRATIO N_OPTION	选择配置DMA仲裁相关参数
STORE_OPTION	选择配置存储转发模式相关参数
DMA_OPTION	选择配置DMA相关参数
VLAN_OPTION	选择配置VLAN相关参数
FLOWCTL_OPTION	选择配置流控相关参数
HASHH_OPTION	选择配置HASH_H相关参数
HASHL_OPTION	选择配置HASH_L相关参数
FILTER_OPTION	选择配置帧过滤器相关参数
HALFDUPLEX_OPT ION	选择配置半双工模式相关参数
TIMER_OPTION	选择配置计数器相关参数
INTERFRAMEGAP_ OPTION	选择配置帧间隔相关参数

枚举类型 `enet_mediamode_enum`

表 3-443. 枚举类型 `enet_mediamode_enum`

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

枚举类型 `enet_chksumconf_enum`

表 3-444. 枚举类型 `enet_chksumconf_enum`

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

枚举类型 `enet_frmrecept_enum`

表 3-445. 枚举类型 `enet_frmrecept_enum`

成员名称	功能描述
<code>ENET_PROMISCUOUS_MODE</code>	使能混杂模式
<code>ENET_RECEIVEALL</code>	接收所有帧
<code>ENET_BROADCAST_FRAMES_PASS</code>	接收广播帧
<code>ENET_BROADCAST_FRAMES_DROP</code>	禁止接收广播帧

枚举类型 `enet_registers_type_enum`表 3-446. 枚举类型 `enet_registers_type_enum`

成员名称	功能描述
<code>ALL_MAC_REG</code>	寄存器范围从ENET_MAC_CFG到ENET_MAC_FCTH
<code>ALL_MSC_REG</code>	寄存器范围从ENET_MSC_CTL到ENET_MSC_RGUFCNT
<code>ALL_PTP_REG</code>	寄存器范围从ENET_PTP_TSCTL到ENET_PTP_PPSCTL
<code>ALL_DMA_REG</code>	寄存器范围从ENET_DMA_BCTL到ENET_DMA_CRBADDR

枚举类型 `enet_dmadirection_enum`表 3-447. 枚举类型 `enet_dmadirection_enum`

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

枚举类型 `enet_phydirection_enum`表 3-448. 枚举类型 `enet_phydirection_enum`

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

枚举类型 `enet_regdirection_enum`表 3-449. 枚举类型 `enet_regdirection_enum`

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

枚举类型 `enet_macaddress_enum`表 3-450. 枚举类型 `enet_macaddress_enum`

成员名称	功能描述
<code>ENET_MAC_ADDR ESS0</code>	配置MAC address 0过滤器
<code>ENET_MAC_ADDR ESS1</code>	配置MAC address 1过滤器
<code>ENET_MAC_ADDR ESS2</code>	配置MAC address 2过滤器
<code>ENET_MAC_ADDR ESS3</code>	配置MAC address 3过滤器

枚举类型 `enet_descstate_enum`

表 3-451. 枚举类型 `enet_descstate_enum`

成员名称	功能描述
<code>TXDESC_COLLISION_COUNT</code>	帧发送出去前出现的冲突次数
<code>TXDESC_BUFFER_1_ADDR</code>	发送帧的缓冲区地址
<code>RXDESC_FRAME_LENGTH</code>	接收帧长度
<code>RXDESC_BUFFER_1_SIZE</code>	接收缓冲区1大小
<code>RXDESC_BUFFER_2_SIZE</code>	接收缓冲区2大小
<code>RXDESC_BUFFER_1_ADDR</code>	接收帧的缓冲区地址

枚举类型 `enet_msc_preset_enum`

表 3-452. 枚举类型 `enet_msc_preset_enum`

成员名称	功能描述
<code>ENET_MSC_PRESET_NONE</code>	关闭MSC计数器预设功能
<code>ENET_MSC_PRESET_HALF</code>	预设为最大值一半
<code>ENET_MSC_PRESET_FULL</code>	预设为最大值一半

函数 `enet_deinit`

函数`enet_deinit`描述见下表：

表 3-453. 函数 `enet_deinit`

函数名称	<code>enet_deinit</code>
函数原型	<code>void enet_deinit(uint32_t enet_periph);</code>
功能描述	复位ENET模块及相关软件初始化所需结构体
先决条件	-
被调用函数	<code>rcu_periph_reset_enable()/rcu_periph_reset_disable()/enet_initpara_reset()</code>
输入参数{in}	
<code>enet_periph</code>	以太网外设
<code>ENET0</code>	以太网0
<code>ENET1</code>	以太网1
输出参数{out}	
-	-

返回值	
-	-

例如：

```
/* deinitialize the ENET */
```

```
enet_deinit(ENET0);
```

函数 enet_initpara_config

函数enet_initpara_config描述见下表：

表 3-454. 函数 enet_initpara_config

函数名称	enet_initpara_config
函数原型	void enet_initpara_config(enet_option_enum option, uint32_t para)
功能描述	配置ENET模块的各类不常用功能。当enet_init()函数无法满足所需实现功能时调用，必须在enet_init()函数之前调用
先决条件	enet_initpara_reset(void)
被调用函数	-
输入参数{in}	
option	ENET模块功能选项，根据选择需要使用不同参数进行配置，参考 表3-442. 枚举类型enet_option_enum ，下列参数仅可选择一个
FORWARD_OPTION	选择配置帧通过功能相关参数
DMABUS_OPTION	选择配置DMA总线模式相关参数
DMA_MAXBURST_OPTION	选择配置DMA最大突发传输相关参数
DMA_ARBITRATION_OPTION	选择配置DMA仲裁相关参数
STORE_OPTION	选择配置存储转发模式相关参数
DMA_OPTION	选择配置DMA相关参数
VLAN_OPTION	选择配置VLAN相关参数
FLOWCTL_OPTION	选择配置流控相关参数
HASHH_OPTION	选择配置HASH_H相关参数
HASHL_OPTION	选择配置HASH_L相关参数
FILTER_OPTION	选择配置帧过滤器相关参数
HALFDUPLEX_OPTION	选择配置半双工模式相关参数
TIMER_OPTION	选择配置计数器相关参数
INTERFRAMEGAP_OPTION	选择配置帧间隔相关参数
输入参数{in}	
para	下列参数值均可以被配置
(该参数值需根据	例如：para = (value1 value2 value3...)

option参数对应值进行选取)	
当option参数值为FORWARD_OPTION时	
value1	ENET_AUTO_PADCRC_DROP_ENABLE / ENET_AUTO_PADCRC_DROP_DISABLE
value2	ENET_TYPEFRAME_CRC_DROP_ENABLE / ENET_TYPEFRAME_CRC_DROP_DISABLE
value3	ENET_FORWARD_ERRFRAMES_ENABLE / ENET_FORWARD_ERRFRAMES_DISABLE
value4	ENET_FORWARD_UNDEERSZ_GOODFRAMES_ENABLE / ENET_FORWARD_UNDEERSZ_GOODFRAMES_DISABLE
当option参数值为DMABUS_OPTION时	
value1	ENET_ADDRESS_ALIGN_ENABLE / ENET_ADDRESS_ALIGN_DISABLE
value2	ENET_FIXED_BURST_ENABLE / ENET_FIXED_BURST_DISABLE
value3	ENET_MIXED_BURST_ENABLE / ENET_MIXED_BURST_DISABLE
当option参数值为DMA_MAXBURST_OPTION时	
value1	ENET_RXDP_1BEAT / ENET_RXDP_2BEAT / ENET_RXDP_4BEAT / ENET_RXDP_8BEAT / ENET_RXDP_16BEAT / ENET_RXDP_32BEAT / ENET_RXDP_4xPGBL_4BEAT / ENET_RXDP_4xPGBL_8BEAT / ENET_RXDP_4xPGBL_16BEAT / ENET_RXDP_4xPGBL_32BEAT / ENET_RXDP_4xPGBL_64BEAT / ENET_RXDP_4xPGBL_128BEAT
value2	ENET_PGBL_1BEAT / ENET_PGBL_2BEAT / ENET_PGBL_4BEAT / ENET_PGBL_8BEAT / ENET_PGBL_16BEAT / ENET_PGBL_32BEAT / ENET_PGBL_4xPGBL_4BEAT / ENET_PGBL_4xPGBL_8BEAT / ENET_PGBL_4xPGBL_16BEAT / ENET_PGBL_4xPGBL_32BEAT / ENET_PGBL_4xPGBL_64BEAT / ENET_PGBL_4xPGBL_128BEAT
value3	ENET_RXTX_DIFFERENT_PGBL / ENET_RXTX_SAME_PGBL
当option参数值为DMA_ARBITRATION_OPTION时	
value1	ENET_ARBITRATION_RXPRIORTX
value2	ENET_ARBITRATION_RXTX_1_1 / ENET_ARBITRATION_RXTX_2_1 / ENET_ARBITRATION_RXTX_3_1 / ENET_ARBITRATION_RXTX_4_1
当option参数值为STORE_OPTION时	
value1	ENET_RX_MODE_STOREFORWARD / ENET_RX_MODE_CUTTHROUGH
value2	ENET_TX_MODE_STOREFORWARD / ENET_TX_MODE_CUTTHROUGH
value3	ENET_RX_THRESHOLD_64BYTES / ENET_RX_THRESHOLD_32BYTES / ENET_RX_THRESHOLD_96BYTES / ENET_RX_THRESHOLD_128BYTES
value4	ENET_TX_THRESHOLD_64BYTES / ENET_TX_THRESHOLD_128BYTES / ENET_TX_THRESHOLD_192BYTES / ENET_TX_THRESHOLD_256BYTES / ENET_TX_THRESHOLD_40BYTES / ENET_TX_THRESHOLD_32BYTES / ENET_TX_THRESHOLD_24BYTES / ENET_TX_THRESHOLD_16BYTES
当option参数值为DMA_OPTION时	
value1	ENET_FLUSH_RXFRAME_ENABLE / ENET_FLUSH_RXFRAME_DISABLE

value2	ENET_SECONDFRAME_OPT_ENABLE / ENET_SECONDFRAME_OPT_DISABLE
value3	ENET_ENHANCED_DESCRIPTOR/ ENET_NORMAL_DESCRIPTOR
当 option 参数值为VLAN_OPTION时	
value1	ENET_VLANTAGCOMPARISON_12BIT/ ENET_VLANTAGCOMPARISON_16BIT
value2	MAC_VLT_VLTI(regval)
当 option 参数值为FLOWCTL_OPTION时	
value1	MAC_FCTL_PTM(regval)
value2	ENET_ZERO_QUANTA_PAUSE_ENABLE / ENET_ZERO_QUANTA_PAUSE_DISABLE
value3	ENET_PAUSETIME_MINUS4 / ENET_PAUSETIME_MINUS28 / ENET_PAUSETIME_MINUS144/ENET_PAUSETIME_MINUS256
value4	ENET_MAC0_AND_UNIQUE_ADDRESS_PAUSEDetect / ENET_UNIQUE_PAUSEDetect
value5	ENET_RX_FLOWCONTROL_ENABLE / ENET_RX_FLOWCONTROL_DISABLE
value6	ENET_TX_FLOWCONTROL_ENABLE / ENET_TX_FLOWCONTROL_DISABLE
当 option 参数值为HASHH_OPTION时	
value1	0x0~0xFFFF FFFFU
当 option 参数值为HASHL_OPTION时	
value1	0x0~0xFFFF FFFFU
当 option 参数值为FILTER_OPTION时	
value1	ENET_SRC_FILTER_NORMAL_ENABLE / ENET_SRC_FILTER_INVERSE_ENABLE / ENET_SRC_FILTER_DISABLE
value2	ENET_DEST_FILTER_INVERSE_ENABLE / ENET_DEST_FILTER_INVERSE_DISABLE
value3	ENET_MULTICAST_FILTER_HASH_OR_PERFECT / ENET_MULTICAST_FILTER_HASH / ENET_MULTICAST_FILTER_PERFECT / ENET_MULTICAST_FILTER_NONE
value4	ENET_UNICAST_FILTER_EITHER / ENET_UNICAST_FILTER_HASH / ENET_UNICAST_FILTER_PERFECT
value5	ENET_PCFRM_PREVENT_ALL / ENET_PCFRM_PREVENT_PAUSEFRAME / ENET_PCFRM_FORWARD_ALL / ENET_PCFRM_FORWARD_FILTERED
当 option 参数值为HALFDUPLEX_OPTION时	
value1	ENET_CARRIERSENSE_ENABLE / ENET_CARRIERSENSE_DISABLE
value2	ENET_RECEIVEOWN_ENABLE / ENET_RECEIVEOWN_DISABLE
value3	ENET_RETRYTRANSMISSION_ENABLE / ENET_RETRYTRANSMISSION_DISABLE
value4	ENET_BACKOFFLIMIT_10 / ENET_BACKOFFLIMIT_8 /

	<i>ENET_BACKOFFLIMIT_4 / ENET_BACKOFFLIMIT_1</i>
value5	<i>ENET_DEFERRALCHECK_ENABLE / ENET_DEFERRALCHECK_DISABLE</i>
当 option 参数值为 <i>TIMER_OPTION</i> 时	
value1	<i>ENET_WATCHDOG_ENABLE / ENET_WATCHDOG_DISABLE</i>
value2	<i>ENET_JABBER_ENABLE / ENET_JABBER_DISABLE</i>
当 option 参数值为 <i>INTERFRAMEGAP_OPTION</i> 时	
value1	<i>ENET_INTERFRAMEGAP_96BIT / ENET_INTERFRAMEGAP_88BIT / ENET_INTERFRAMEGAP_80BIT / ENET_INTERFRAMEGAP_72BIT / ENET_INTERFRAMEGAP_64BIT / ENET_INTERFRAMEGAP_56BIT / ENET_INTERFRAMEGAP_48BIT / ENET_INTERFRAMEGAP_40BIT</i>
输出参数{out}	
-	-
返回值	
-	-

例如：

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

```
enet_initpara_reset();
```

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

函数 enet_init

函数enet_init描述见下表：

表 3-455. 函数 enet_init

函数名称	enet_init
函数原型	ErrStatus enet_init(uint32_t enet_periph, enet_mediamode_enum mediamode, enet_chksumconf_enum checksum, enet_frmrecept_enum recept);
功能描述	ENET模块初始化，配置用户最关心的功能
先决条件	enet_deinit ()
被调用函数	enet_phy_config()/enet_phy_write_read()/enet_default_init()
输入参数{in}	
enet_periph	以太网外设
ENET0	以太网0
ENET1	以太网1
输入参数{in}	
mediamode	ENET通讯方式配置，参考 表3-443. 枚举类型enet_mediamode_enum ，仅可选择唯一参数
ENET_AUTO_NEGOTIATION	PHY自协商
ENET_100M_FULLD	100Mbit/s，全双工

UPLEX	
ENET_100M_HALF DUPLEX	100Mbit/s, 半双工
ENET_10M_FULLD UPLEX	10Mbit/s, 全双工
ENET_10M_HALFD UPLEX	10Mbit/s, 半双工
ENET_LOOPBACK MODE	MII模式下的回环模式
输入参数{in}	
checksum	IP帧数据校验和功能, 参考 表3-444. 枚举类型enet_chksumconf_enum , 仅可选择唯一参数
ENET_NO_AUTOCH ECKSUM	关闭IP帧校验和功能
ENET_AUTOCHECK SUM_DROP_FAILF RAMES	使能IP帧校验和功能
ENET_AUTOCHECK SUM_ACCEPT_FAIL FRAMES	使能IP帧校验和功能, 不丢弃仅有载荷错误的帧
输入参数{in}	
recept	帧过滤功能, 参考 表3-445. 枚举类型enet_frmrecept_enum , 仅可选择唯一参数
ENET_PROMISCUO US_MODE	使能混杂模式
ENET_RECEIVEALL	接收所有帧
ENET_BROADCAST _FRAMES_PASS	接收广播帧
ENET_BROADCAST _FRAMES_DROP	禁止接收广播帧
输出参数{out}	
-	-
返回值	
ErrStatus	ERROR or SUCCESS

例如:

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

```
ErrStatus enet_init_status;
```

```
enet_init_status = enet_init(ENET0, ENET_AUTO_NEGOTIATION, ENET_AUTOCHECK  
SUM_DROP_FAILFRAMES, ENET_BROADCAST_FRAMES_PASS);
```

函数 **enet_software_reset**

函数 **enet_software_reset** 描述见下表：

表 3-456. 函数 **enet_software_reset**

函数名称	enet_software_reset
函数原型	ErrStatus enet_software_reset(uint32_t enet_periph);
功能描述	复位ENET寄存器，并检测CLK_TX/CLK_RX信号
先决条件	-
被调用函数	-
输入参数{in}	
enet_periph	以太网外设
ENET0	以太网0
ENET1	以太网1
输出参数{out}	
-	-
返回值	
ErrStatus	ERROR or SUCCESS

例如：

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

```
ErrStatus reval_state = ERROR;
```

```
reval_state = enet_software_reset(ENET0);
```

函数 **enet_rxframe_size_get**

函数 **enet_rxframe_size_get** 描述见下表：

表 3-457. 函数 **enet_rxframe_size_get**

函数名称	enet_rxframe_size_get
函数原型	uint32_t enet_rxframe_size_get(uint32_t enet_periph);
功能描述	检测接收帧是否有错误，正确时返回帧长度
先决条件	-
被调用函数	enet_rxframe_drop()
输入参数{in}	
enet_periph	以太网外设
ENET0	以太网0
ENET1	以太网1
输出参数{out}	
-	-
返回值	
uint32_t	取值范围：0x0 - 0x3FFF

例如：

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

```
uint32_t reval;
```

```
reval = enet_rxframe_size_get(ENET0);
```

函数 enet_descriptors_chain_init

函数enet_descriptors_chain_init描述见下表：

表 3-458. 函数 enet_descriptors_chain_init

函数名称	enet_descriptors_chain_init
函数原型	void enet_descriptors_chain_init(uint32_t enet_periph, enet_dmadirection_enum direction);
功能描述	初始化DMA接收/发送描述符为链模式
先决条件	-
被调用函数	-
输入参数{in}	
enet_periph	以太网外设
ENET0	以太网0
ENET1	以太网1
输入参数{in}	
direction	想要初始化的描述符类型，参考 表3-447. 枚举类型 enet_dmadirection_enum , 下列参数仅可选择一个
ENET_DMA_TX	DMA Tx描述符
ENET_DMA_RX	DMA Rx描述符
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* initialize the DMA Tx/Rx descriptors's parameters in chain mode */
```

```
enet_descriptors_chain_init(ENET0, ENET_DMA_TX);
```

函数 enet_descriptors_ring_init

函数enet_descriptors_ring_init描述见下表：

表 3-459. 函数 enet_descriptors_ring_init

函数名称	enet_descriptors_ring_init
函数原型	void enet_descriptors_ring_init(uint32_t enet_periph, enet_dmadirection_enum direction);
功能描述	初始化DMA接收/发送描述符为环模式

先决条件	-
被调用函数	-
输入参数{in}	
enet_periph	以太网外设
ENET0	以太网0
ENET1	以太网1
输入参数{in}	
direction	想要初始化的描述符类型，参考 表3-447. 枚举类型 enet_dmadirection_enum , 下列参数仅可选择一个
ENET_DMA_TX	DMA Tx描述符
ENET_DMA_RX	DMA Rx描述符
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* initialize the DMA Tx/Rx descriptors's parameters in ring mode */
```

```
enet_descriptors_ring_init(ENET0, ENET_DMA_TX);
```

函数 enet_frame_receive

函数enet_frame_receive描述见下表：

表 3-460. 函数 enet_frame_receive

函数名称	enet_frame_receive
函数原型	ErrStatus enet_frame_receive(uint32_t enet_periph, uint8_t buffer[], uint32_t bufsize);
功能描述	处理当前接收到的帧，并将当前描述符中存储的接收帧数据拷贝到指定区域
先决条件	-
被调用函数	-
输入参数{in}	
enet_periph	以太网外设
ENET0	以太网0
ENET1	以太网1
输入参数{in}	
bufsize	缓冲区长度，范围(0~1524)
输出参数{out}	
buffer	接收帧数据的缓冲区地址指针。如果输入NULL，用户需要在调用该函数之前将数据拷贝到用户缓冲区内
返回值	
ErrStatus	ERROR or SUCCESS

例如：

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

uint8_t data_buffer[1500];

uint32_t data_size;

enet_frame_receive(ENET0, data_buffer, &data_size);
```

函数 enet_frame_transmit

函数enet_frame_transmit描述见下表：

表 3-461. 函数 enet_frame_transmit

函数名称	enet_frame_transmit
函数原型	ErrStatus enet_frame_transmit(uint32_t enet_periph, uint8_t buffer[], uint32_t length);
功能描述	将指定区域内的数据拷贝到当前发送描述符中，并发送
先决条件	-
被调用函数	-
输入参数{in}	
enet_periph	以太网外设
ENET0	以太网0
ENET1	以太网1
输入参数{in}	
buffer	等待发送的帧数据缓冲区地址指针。如果输入NULL，用户需要在调用该函数之前将待发送数据拷贝到描述符指定的位置
输入参数{in}	
length	待发送数据长度，范围(0~1524)
输出参数{out}	
-	-
返回值	
ErrStatus	ERROR or SUCCESS

例如：

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

uint8_t data_buffer[1500];

uint32_t data_size = 800;

enet_frame_transmit (ENET0, data_buffer, data_size);
```

函数 enet_transmit_checksum_config

函数enet_transmit_checksum_config描述见下表：

表 3-462. 函数 `enet_transmit_checksum_config`

函数名称	<code>enet_transmit_checksum_config</code>
函数原型	<code>ErrStatus enet_transmit_checksum_config(enet_descriptors_struct *desc, uint32_t checksum);</code>
功能描述	配置发送帧校验和模式
先决条件	-
被调用函数	-
输入参数{in}	
desc	需要配置的描述符地址指针，结构体成员介绍参考 表3-433. 结构体 <code>enet_descriptors_struct</code>
输入参数{in}	
checksum	IP帧校验和配置，下列参数仅可选择一个
<code>ENET_CHECKSUM_DISABLE</code>	禁能校验和自动插入
<code>ENET_CHECKSUM_IPV4HEADER</code>	仅使能IP头校验和计算和插入
<code>ENET_CHECKSUM_TCPUDPICMP_SEGMENT</code>	TCP/UDP/ICMP校验和（除去伪报头）计算和插入
<code>ENET_CHECKSUM_TCPUDPICMP_FULL</code>	TCP/UDP/ICMP校验和计算和插入
输出参数{out}	
返回值	
ErrStatus	ERROR or SUCCESS

例如：

```
/* configure the transmit IP frame checksum offload calculation and insertion */
enet_descriptors_struct rx_desc;
enet_transmit_checksum_config(rx_desc, ENET_CHECKSUM_TCPUDPICMP_FULL);
```

函数 `enet_enable`

函数`enet_enable`描述见下表：

表 3-463. 函数 `enet_enable`

函数名称	<code>enet_enable</code>
函数原型	<code>void enet_enable(uint32_t enet_periph);</code>
功能描述	ENET Tx/Rx功能使能（包括ENET外设内的MAC和DMA模块）
先决条件	-
被调用函数	<code>enet_tx_enable()/enet_rx_enable()</code>

输入参数{in}	
enet_periph	以太网外设
<i>ENET0</i>	以太网0
<i>ENET1</i>	以太网1
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* enable the ENET */
enet_enable(ENET0);
```

函数 enet_disable

函数enet_disable描述见下表：

表 3-464. 函数 enet_disable

函数名称	enet_disable
函数原型	void enet_disable(uint32_t enet_periph);
功能描述	ENET Tx/Rx功能禁能（包括ENET外设内的MAC和DMA模块）
先决条件	-
被调用函数	enet_tx_disable()/enet_rx_disable()
输入参数{in}	
enet_periph	以太网外设
<i>ENET0</i>	以太网0
<i>ENET1</i>	以太网1
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* disable the ENET */
enet_disable(ENET0);
```

函数 enet_mac_address_set

函数enet_mac_address_set描述见下表：

表 3-465. 函数 enet_mac_address_set

函数名称	enet_mac_address_set
函数原型	void enet_mac_address_set(uint32_t enet_periph, enet_macaddress_enum

	mac_addr, uint8_t paddr[]);
功能描述	配置MAC地址
先决条件	-
被调用函数	-
输入参数{in}	
enet_periph	以太网外设
ENET0	以太网0
ENET1	以太网1
输入参数{in}	
mac_addr	选择何组MAC地址将被配置，参考 表3-450. 枚举类型 enet_macaddress_enum ，下列参数仅可选择一个
ENET_MAC_ADDRES0	配置MAC address 0过滤器
ENET_MAC_ADDRES1	配置MAC address 1过滤器
ENET_MAC_ADDRES2	配置MAC address 2过滤器
ENET_MAC_ADDRES3	配置MAC address 3过滤器
输入参数{in}	
paddr	存储MAC地址的缓冲区指针，小端存储 例如MAC地址为aa:bb:cc:dd:ee:22，缓冲区内数据为{22, ee, dd, cc, bb, aa}
输出参数{out}	
-	-
返回值	
-	-

例如：

```

/* config mac address */

netif->hwaddr[0] = 0x02;

netif->hwaddr[1] = 0xaa;

netif->hwaddr[2] = 0xbb;

netif->hwaddr[3] = 0xcc;

netif->hwaddr[4] = 0xdd;

netif->hwaddr[5] = 0xee;

enet_mac_address_set(ENET0, ENET_MAC_ADDRESS0, netif->hwaddr);

```

函数 enet_mac_address_get

函数enet_mac_address_get描述见下表：

表 3-466. 函数 `enet_mac_address_get`

函数名称	<code>enet_mac_address_get</code>
函数原型	<code>ErrStatus enet_mac_address_get(uint32_t enet_periph, enet_macaddress_enum mac_addr, uint8_t paddr[], uint8_t bufsize);</code>
功能描述	获取MAC地址
先决条件	-
被调用函数	-
输入参数{in}	
<code>enet_periph</code>	以太网外设
<code>ENET0</code>	以太网0
<code>ENET1</code>	以太网1
输入参数{in}	
<code>mac_addr</code>	选择何组MAC地址将被配置，参考 表3-450. 枚举类型 <code>enet_macaddress_enum</code> ，下列参数仅可选择一个
<code>ENET_MAC_ADDRESS0</code>	配置MAC address 0过滤器
<code>ENET_MAC_ADDRESS1</code>	配置MAC address 1过滤器
<code>ENET_MAC_ADDRESS2</code>	配置MAC address 2过滤器
<code>ENET_MAC_ADDRESS3</code>	配置MAC address 3过滤器
输出参数{out}	
<code>paddr</code>	存储MAC地址的缓冲区指针，小端存储 例如MAC地址为aa:bb:cc:dd:ee:22，缓冲区内数据为{22, ee, dd, cc, bb, aa}
输入参数{in}	
<code>bufsize</code>	缓存大小，6-255
返回值	
-	-

例如：

```
/* get mac address */
```

```
enet_mac_address_get (ENET0, ENET_MAC_ADDRESS0, netif->hwaddr, 0x100);
```

函数 `enet_flag_get`

函数`enet_flag_get`描述见下表：

表 3-467. 函数 `enet_flag_get`

函数名称	<code>enet_flag_get</code>
函数原型	<code>FlagStatus enet_flag_get(uint32_t enet_periph, enet_flag_enum enet_flag);</code>
功能描述	获取ENET模块MAC/MSC/PTP/DMA状态标志位
先决条件	-

被调用函数	-
输入参数{in}	
enet_periph	以太网外设
ENET0	以太网0
ENET1	以太网1
输入参数{in}	
enet_flag	ENET状态标志位，参考 表3-435. 枚举类型enet_flag_enum ，下列参数仅可选择一个
ENET_MAC_FLAG_MPKR	接收到魔术帧标志位
ENET_MAC_FLAG_WUFR	接收到唤醒帧标志位
ENET_MAC_FLAG_FLOWCONTROL	流控状态标志位
ENET_MAC_FLAG_WUM	WUM状态标志位
ENET_MAC_FLAG_MSC	MSC状态标志位
ENET_MAC_FLAG_MSCR	MSC接收状态标志位
ENET_MAC_FLAG_MSCT	MSC发送状态标志位
ENET_MAC_FLAG_TMST	时间戳触发状态标志位
ENET_PTP_FLAG_TSSCO	时间戳秒计数溢出标志位
ENET_PTP_FLAG_TTM	目标时间匹配标志位
ENET_MSC_FLAG_RFCE	接收帧CRC错误标志位
ENET_MSC_FLAG_RFAE	接收帧对齐错误标志位
ENET_MSC_FLAG_RGUF	接收到“好”的单播帧标志位
ENET_MSC_FLAG_TGFSC	发送“好”的帧时仅遇到1个冲突标志位
ENET_MSC_FLAG_TGFMSC	发送“好”的帧时遇到1个以上冲突
ENET_MSC_FLAG_TGF	发送“好”的帧标志位
ENET_DMA_FLAG_TS	发送状态标志位

ENET_DMA_FLAG_TPS	发送流程停止状态标志位
ENET_DMA_FLAG_TBU	发送缓冲区不可用状态标志位
ENET_DMA_FLAG_TJT	发送jabber超时状态标志位
ENET_DMA_FLAG_RO	接收溢出状态标志位
ENET_DMA_FLAG_TU	发送下溢状态标志位
ENET_DMA_FLAG_RS	接收状态标志位
ENET_DMA_FLAG_RBU	接收缓冲区不可用状态标志位
ENET_DMA_FLAG_RPS	接受流程停止状态标志位
ENET_DMA_FLAG_RWT	接收看门狗超时状态标志位
ENET_DMA_FLAG_ET	早发送状态标志位
ENET_DMA_FLAG_FBE	致命总线错误状态标志位
ENET_DMA_FLAG_ER	早接收状态标志位
ENET_DMA_FLAG_AI	异常中断汇总标志位
ENET_DMA_FLAG_NI	正常中断汇总标志位
ENET_DMA_FLAG_EB_DMA_ERROR	DMA错误标志位
ENET_DMA_FLAG_EB_TRANSFER_ERROR	发送错误标志位
ENET_DMA_FLAG_EB_ACCESS_ERROR	DMA访问错误标志位
ENET_DMA_FLAG_MSC	MSC状态标志位
ENET_DMA_FLAG_WUM	WUM状态标志位
ENET_DMA_FLAG_TST	时间戳触发状态标志位
输出参数{out}	

-	-
返回值	
FlagStatus	SET or RESET

例如：

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

```
enet_flag_get (ENET0, ENET_DMA_FLAG_RS);
```

函数 enet_flag_clear

函数enet_flag_clear描述见下表：

表 3-468. 函数 enet_flag_clear

函数名称	enet_flag_clear
函数原型	void enet_flag_clear(uint32_t enet_periph, enet_flag_clear_enum enet_flag);
功能描述	清除ENET状态标志位
先决条件	-
被调用函数	-
输入参数{in}	
enet_periph	以太网外设
ENET0	以太网0
ENET1	以太网1
输入参数{in}	
enet_flag	ENET模块DMA标志位清除，参考 表3-436. 枚举类型enet_flag_clear_enum ， 下列参数仅可选择一个
ENET_DMA_FLAG_TS_CLR	发送状态标志位清除
ENET_DMA_FLAG_TPS_CLR	发送流程停止状态标志位清除
ENET_DMA_FLAG_TBU_CLR	发送缓冲区不可用状态标志位清除
ENET_DMA_FLAG_TJT_CLR	发送jabber超时状态标志位清除
ENET_DMA_FLAG_RO_CLR	接收溢出状态标志位清除
ENET_DMA_FLAG_TU_CLR	发送下溢状态标志位清除
ENET_DMA_FLAG_RS_CLR	接收状态标志位清除
ENET_DMA_FLAG_RBU_CLR	接收缓冲区不可用状态标志位清除
ENET_DMA_FLAG_RPS_CLR	接受流程停止状态标志位清除

ENET_DMA_FLAG_ RWT_CLR	接收看门狗超时状态标志位清除
ENET_DMA_FLAG_ ET_CLR	早发送状态标志位清除
ENET_DMA_FLAG_ FBE_CLR	致命总线错误状态标志位清除
ENET_DMA_FLAG_ ER_CLR	早接收状态标志位清除
ENET_DMA_FLAG_ AI_CLR	异常中断汇总标志位清除
ENET_DMA_FLAG_ NI_CLR	正常中断汇总标志位清除
输出参数{out}	
-	-
返回值	
-	-

例如：

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

```
enet_flag_clear(ENET0, ENET_DMA_FLAG_RS_CLR);
```

函数 enet_interrupt_enable

函数enet_interrupt_enable描述见下表：

表 3-469. 函数 enet_interrupt_enable

函数名称	enet_interrupt_enable
函数原型	void enet_interrupt_enable(uint32_t enet_periph, enet_int_enum enet_int);
功能描述	使能ENET模块MAC/MSC/DMA中断
先决条件	-
被调用函数	-
输入参数{in}	
enet_periph	以太网外设
ENET0	以太网0
ENET1	以太网1
输入参数{in}	
enet_int	ENET中断，参考 表3-437. 枚举类型enet_int_enum ，下列参数仅可选择一个
ENET_MAC_INT_W UMIM	WUM中断屏蔽
ENET_MAC_INT_T MSTIM	时间戳触发中断屏蔽
ENET_MSC_INT_RF CEIM	接收帧CRC错误中断屏蔽

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

例如：

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

```
enet_interrupt_enable(ENET0, ENET_DMA_INT_NIE);
```

函数 `enet_interrupt_disable`

函数 `enet_interrupt_disable` 描述见下表：

表 3-470. 函数 `enet_interrupt_disable`

函数名称	<code>enet_interrupt_disable</code>
函数原型	<code>void enet_interrupt_disable(uint32_t enet_periph, enet_int_enum enet_int);</code>
功能描述	禁能ENET模块MAC/MSC/DMA中断
先决条件	-
被调用函数	-
输入参数{in}	
enet_periph	以太网外设
<i>ENET0</i>	以太网0
<i>ENET1</i>	以太网1
输入参数{in}	
enet_int	ENET中断，参考 表3-437. 枚举类型 <code>enet_int_enum</code> ，下列参数仅可选择一个
<i>ENET_MAC_INT_WUMIM</i>	WUM中断屏蔽
<i>ENET_MAC_INT_TSTMIM</i>	时间戳触发中断屏蔽
<i>ENET_MSC_INT_RFCEIM</i>	接收帧CRC错误中断屏蔽
<i>ENET_MSC_INT_RFAEIM</i>	接收帧对齐错误中断屏蔽
<i>ENET_MSC_INT_RGUFIM</i>	接收“好”单播帧中断屏蔽
<i>ENET_MSC_INT_TGFSCIM</i>	仅遇到1个冲突后发送“好”帧中断屏蔽
<i>ENET_MSC_INT_TGFMSCIM</i>	遇到1个以上冲突后发送“好”帧中断屏蔽
<i>ENET_MSC_INT_TGFIM</i>	发送“好”的帧的中断屏蔽
<i>ENET_DMA_INT_TIE</i>	发送中断使能
<i>ENET_DMA_INT_TPSIE</i>	发送流程停止中断使能
<i>ENET_DMA_INT_TBT</i>	发送缓冲区不可用中断使能

<i>UIE</i>	
<i>ENET_DMA_INT_TJTIE</i>	发送jabber超时中断使能
<i>ENET_DMA_INT_ROIE</i>	接收溢出中断使能
<i>ENET_DMA_INT_TUIE</i>	发送下溢中断使能
<i>ENET_DMA_INT_RIE</i>	接收中断使能
<i>ENET_DMA_INT_RBUIE</i>	接收缓冲区不可用中断使能
<i>ENET_DMA_INT_RPSIE</i>	接收流程停止中断使能
<i>ENET_DMA_INT_RWTIE</i>	接收看门狗超时中断使能
<i>ENET_DMA_INT_ETIE</i>	早发送中断使能
<i>ENET_DMA_INT_FBEIE</i>	致命总线错误中断使能
<i>ENET_DMA_INT_ERIE</i>	早接收中断使能
<i>ENET_DMA_INT_AIE</i>	异常中断汇总使能
<i>ENET_DMA_INT_NIE</i>	正常中断汇总使能
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* disable normal interrupt summary */
```

```
enet_interrupt_disable(ENET0, ENET_DMA_INT_NIE);
```

函数 **enet_interrupt_flag_get**

函数 **enet_interrupt_flag_get** 描述见下表：

表 3-471. 函数 **enet_interrupt_flag_get**

函数名称	enet_interrupt_flag_get
函数原型	FlagStatus enet_interrupt_flag_get(uint32_t enet_periph, enet_int_flag_enum int_flag);
功能描述	获取ENET模块MAC/MSD/DMA中断标志位

先决条件	-
被调用函数	-
输入参数{in}	
enet_periph	以太网外设
ENET0	以太网0
ENET1	以太网1
输入参数{in}	
int_flag	ENET中断标志位，参考 表3-438. 枚举类型enet_int_flag_enum ，下列参数仅可选择一个
ENET_MAC_INT_FL AG_WUM	WUM中断标志位
ENET_MAC_INT_FL AG_MSC	MSC中断标志位
ENET_MAC_INT_FL AG_MSCR	MSC接收中断标志位
ENET_MAC_INT_FL AG_MSCT	MSC发送中断标志位
ENET_MAC_INT_FL AG_TMST	时间戳触发中断标志位
ENET_MSC_INT_FL AG_RFCE	接收帧CRC错误中断标志位
ENET_MSC_INT_FL AG_RFAE	接收帧对齐错误中断标志位
ENET_MSC_INT_FL AG_RGUF	接收“好”单播帧中断标志位
ENET_MSC_INT_FL AG_TGFSC	仅遇到1个冲突后发送“好”帧中断标志位
ENET_MSC_INT_FL AG_TGFMSC	遇到1个以上冲突后发送“好”帧中断标志位
ENET_MSC_INT_FL AG_TGF	发送“好”的帧的中断标志位
ENET_DMA_INT_FL AG_TS	发送中断标志位
ENET_DMA_INT_FL AG_TPS	发送流程停止中断标志位
ENET_DMA_INT_FL AG_TBU	发送缓冲区不可用中断标志位
ENET_DMA_INT_FL AG_TJT	发送jabber超时中断标志位
ENET_DMA_INT_FL AG_RO	接收溢出中断标志位
ENET_DMA_INT_FL	发送下溢中断标志位

AG_TU	
ENET_DMA_INT_FL AG_RS	接收中断标志位
ENET_DMA_INT_FL AG_RBU	接收缓冲区不可用中断标志位
ENET_DMA_INT_FL AG_RPS	接收流程停止中断标志位
ENET_DMA_INT_FL AG_RWT	接收看门狗超时中断标志位
ENET_DMA_INT_FL AG_ET	早发送中断标志位
ENET_DMA_INT_FL AG_FBE	致命总线错误中断标志位
ENET_DMA_INT_FL AG_ER	早接收中断标志位
ENET_DMA_INT_FL AG_AI	异常中断汇总中断标志位
ENET_DMA_INT_FL AG_NI	正常中断汇总中断标志位
ENET_DMA_INT_FL AG_MSC	MSC中断标志位
ENET_DMA_INT_FL AG_WUM	WUM中断标志位
ENET_DMA_INT_FL AG_TST	时间戳触发中断标志位
输出参数{out}	
-	-
返回值	
FlagStatus	SET or RESET

例如:

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

```
enet_interrupt_flag_get(ENET0, ENET_DMA_INT_FLAG_RS);
```

函数 enet_interrupt_flag_clear

函数enet_interrupt_flag_clear描述见下表:

表 3-472. 函数 enet_interrupt_flag_clear

函数名称	enet_interrupt_flag_clear
函数原型	void enet_interrupt_flag_clear(uint32_t enet_periph, enet_int_flag_clear_enum int_flag_clear);
功能描述	禁能ENET中断标志位

先决条件	-
被调用函数	-
输入参数{in}	
enet_periph	以太网外设
ENET0	以太网0
ENET1	以太网1
输入参数{in}	
int_flag_clear	ENET中断标志位清除，参考 表3-439. 枚举类型enet_int_flag_clear_enum ，下列参数仅可选择一个
ENET_DMA_INT_FL AG_TS_CLR	发送中断标志位清除
ENET_DMA_INT_FL AG_TPS_CLR	发送流程停止中断标志位清除
ENET_DMA_INT_FL AG_TBU_CLR	发送缓冲区不可用中断标志位清除
ENET_DMA_INT_FL AG_TJT_CLR	发送jabber超时中断标志位清除
ENET_DMA_INT_FL AG_RO_CLR	接收溢出中断标志位清除
ENET_DMA_INT_FL AG_TU_CLR	发送下溢中断标志位清除
ENET_DMA_INT_FL AG_RS_CLR	接收中断标志位清除
ENET_DMA_INT_FL AG_RBU_CLR	接收缓冲区不可用中断标志位清除
ENET_DMA_INT_FL AG_RPS_CLR	接收流程停止中断标志位清除
ENET_DMA_INT_FL AG_RWT_CLR	接收看门狗超时中断标志位清除
ENET_DMA_INT_FL AG_ET_CLR	早发送中断标志位清除
ENET_DMA_INT_FL AG_FBE_CLR	致命总线错误中断标志位清除
ENET_DMA_INT_FL AG_ER_CLR	早接收中断标志位清除
ENET_DMA_INT_FL AG_AI_CLR	异常中断汇总中断标志位清除
ENET_DMA_INT_FL AG_NI_CLR	正常中断汇总中断标志位清除
输出参数{out}	
-	-
返回值	

-	-
---	---

例如：

```
/* clear receive status flag bit */
```

```
enet_interrupt_flag_clear(ENET0, ENET_DMA_INT_FLAG_RS);
```

函数 enet_tx_enable

函数enet_tx_enable描述见下表：

表 3-473. 函数 enet_tx_enable

函数名称	enet_tx_enable
函数原型	void enet_tx_enable(uint32_t enet_periph);
功能描述	ENET发送功能使能（包括ENET外设内的MAC和DMA模块）
先决条件	-
被调用函数	enet_txfifo_flush()
输入参数{in}	
enet_periph	以太网外设
ENET0	以太网0
ENET1	以太网1
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* enable transport function of MAC and DMA */
```

```
enet_tx_enable(ENET0);
```

函数 enet_tx_disable

函数enet_tx_disable描述见下表：

表 3-474. 函数 enet_tx_disable

函数名称	enet_tx_disable
函数原型	void enet_tx_disable(uint32_t enet_periph);
功能描述	ENET发送功能禁能（包括ENET外设内的MAC和DMA模块）
先决条件	-
被调用函数	enet_txfifo_flush()
输入参数{in}	
enet_periph	以太网外设
ENET0	以太网0
ENET1	以太网1

输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* disable transport function of MAC and DMA */
```

```
enet_tx_disable(ENET0);
```

函数 enet_rx_enable

函数enet_rx_enable描述见下表：

表 3-475. 函数 enet_rx_enable

函数名称	enet_rx_enable
函数原型	void enet_rx_enable(uint32_t enet_periph);
功能描述	ENET接收功能使能（包括ENET外设内的MAC和DMA模块）
先决条件	-
被调用函数	-
输入参数{in}	
enet_periph	以太网外设
ENET0	以太网0
ENET1	以太网1
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* enable reception function of MAC and DMA */
```

```
enet_rx_enable(ENET0);
```

函数 enet_rx_disable

函数enet_rx_disable描述见下表：

表 3-476. 函数 enet_rx_disable

函数名称	enet_rx_disable
函数原型	void enet_rx_disable(uint32_t enet_periph);
功能描述	ENET发送功能禁能（包括ENET外设内的MAC和DMA模块）
先决条件	-
被调用函数	-
输入参数{in}	

enet_periph	以太网外设
<i>ENET0</i>	以太网0
<i>ENET1</i>	以太网1
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* disable reception function of MAC and DMA */
```

```
enet_rx_disable(ENET0);
```

函数 **enet_registers_get**

函数 **enet_registers_get** 描述见下表：

表 3-477. 函数 **enet_registers_get**

函数名称	enet_registers_get
函数原型	void enet_registers_get(uint32_t enet_periph, enet_registers_type_enum type, uint32_t *preg, uint32_t num);
功能描述	获取指定范围ENET寄存器值
先决条件	-
被调用函数	-
输入参数{in}	
enet_periph	以太网外设
<i>ENET0</i>	以太网0
<i>ENET1</i>	以太网1
输入参数{in}	
type	寄存器类型，参考 表3-446. 枚举类型enet_registers_type_enum ，下列参数仅可选择一个
<i>ALL_MAC_REG</i>	寄存器范围从ENET_MAC_CFG到ENET_MAC_FCTH
<i>ALL_MSC_REG</i>	寄存器范围从ENET_MSC_CTL到ENET_MSC_RGUF CNT
<i>ALL_PTP_REG</i>	寄存器范围从ENET_PTP_TSCTL到ENET_PTP_PPSCTL
<i>ALL_DMA_REG</i>	寄存器范围从ENET_DMA_BCTL到ENET_DMA_CRBADDR
输入参数{in}	
num	想要获取的寄存器个数，范围(0~54)
输出参数{out}	
preg	存储寄存器值的应用缓冲区指针
返回值	
-	-

例如：

```
/* get all mac registers value */
```

```
uint32_t register_buffer[5];
```

```
enet_registers_get(ENET0, ALL_MAC_REG, 5, register_buffer);
```

函数 `enet_debug_status_get`

函数 `enet_debug_status_get` 描述见下表：

表 3-478. 函数 `enet_debug_status_get`

函数名称	<code>enet_debug_status_get</code>
函数原型	<code>uint32_t enet_debug_status_get(uint32_t enet_periph, uint32_t mac_debug);</code>
功能描述	获取ENET调试状态信息
先决条件	-
被调用函数	-
输入参数{in}	
<code>enet_periph</code>	以太网外设
<code>ENET0</code>	以太网0
<code>ENET1</code>	以太网1
输入参数{in}	
<code>mac_debug</code>	ENET调试信息选项，下列参数仅可选择一个
<code>ENET_MAC_RECEIVER_NOT_IDLE</code>	MAC接收器非空
<code>ENET_RX_ASYNCERONOUS_FIFO_STATUS</code>	异步接收FIFO状态
<code>ENET_RXFIFO_WRITING</code>	接收FIFO正在执行写操作
<code>ENET_RXFIFO_READ_STATUS</code>	读取接收FIFO操作状态
<code>ENET_RXFIFO_STATUS</code>	接收FIFO状态
<code>ENET_MAC_TRANSMITTER_NOT_IDLE</code>	MAC发送器非空
<code>ENET_MAC_TRANSMITTER_STATUS</code>	MAC发送器状态
<code>ENET_PAUSE_CONDITION_STATUS</code>	暂停条件状态
<code>ENET_TXFIFO_READ_STATUS</code>	读取发送FIFO操作状态
<code>ENET_TXFIFO_WRITING</code>	发送FIFO正在执行写操作
<code>ENET_TXFIFO_NOT_EMPTY</code>	发送FIFO非空
<code>ENET_TXFIFO_FULL</code>	发送FIFO已满

<i>L</i>	
输出参数{out}	
-	-
返回值	
uint32_t	相关状态值

例如：

```
/* get debug message of RxFIFO state */

uint32_t debug_value;

debug_value = enet_debug_status_get (ENET0, ENET_RXFIFO_STATE);
```

函数 enet_address_filter_enable

函数enet_address_filter_enable描述见下表：

表 3-479. 函数 enet_address_filter_enable

函数名称	enet_address_filter_enable
函数原型	void enet_address_filter_enable(uint32_t enet_periph, enet_macaddress_enum mac_addr);
功能描述	MAC地址过滤器使能
先决条件	-
被调用函数	--
输入参数{in}	
enet_periph	以太网外设
ENET0	以太网0
ENET1	以太网1
输入参数{in}	
mac_addr	选择被使能的MAC地址组，参考 表3-450. 枚举类型 enet_macaddress_enum ，下列参数仅可选择一个
ENET_MAC_ADDRESSES1	使能MAC地址组1过滤器
ENET_MAC_ADDRESSES2	使能MAC地址组2过滤器
ENET_MAC_ADDRESSES3	使能MAC地址组3过滤器
输出参数{out}	
-	-
返回值	
-	-

例如：

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



```
enet_address_filter_enable(ENET0, ENET_MAC_ADDRESS1);
```

函数 `enet_address_filter_disable`

函数 `enet_address_filter_disable` 描述见下表：

表 3-480. 函数 `enet_address_filter_disable`

函数名称	<code>enet_address_filter_disable</code>
函数原型	<code>void enet_address_filter_disable(uint32_t enet_periph, enet_macaddress_enum mac_addr);</code>
功能描述	MAC地址过滤器禁能
先决条件	-
被调用函数	-
输入参数{in}	
<code>enet_periph</code>	以太网外设
<code>ENET0</code>	以太网0
<code>ENET1</code>	以太网1
输入参数{in}	
<code>mac_addr</code>	选择被使能的MAC地址组，参考 表3-450. 枚举类型 <code>enet_macaddress_enum</code> ，下列参数仅可选择一个
<code>ENET_MAC_ADDRESS1</code>	使能MAC地址组1过滤器
<code>ENET_MAC_ADDRESS2</code>	使能MAC地址组2过滤器
<code>ENET_MAC_ADDRESS3</code>	使能MAC地址组3过滤器
输出参数{out}	
-	-
返回值	
-	-

例如：

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

```
enet_address_filter_disable(ENET0, ENET_MAC_ADDRESS1);
```

函数 `enet_address_filter_config`

函数 `enet_address_filter_config` 描述见下表：

表 3-481. 函数 `enet_address_filter_config`

函数名称	<code>enet_address_filter_config</code>
函数原型	<code>void enet_address_filter_config(uint32_t enet_periph, enet_macaddress_enum mac_addr, uint32_t addr_mask, uint32_t filter_type);</code>
功能描述	配置MAC地址过滤器模式

先决条件	-
被调用函数	-
输入参数{in}	
enet_periph	以太网外设
ENET0	以太网0
ENET1	以太网1
输入参数{in}	
mac_addr	选择被使能的MAC地址组，参考 表3-450. 枚举类型 enet_macaddress_enum ，下列参数仅可选择一个
ENET_MAC_ADDRESSES1	使能MAC地址组1过滤器
ENET_MAC_ADDRESSES2	使能MAC地址组2过滤器
ENET_MAC_ADDRESSES3	使能MAC地址组3过滤器
输入参数{in}	
addr_mask	选择MAC地址哪些字节将被屏蔽，下列参数可以选择多个
ENET_ADDRESS_MASK_BYTE0	屏蔽ENET_MAC_ADDR1L[7:0] bits
ENET_ADDRESS_MASK_BYTE1	屏蔽ENET_MAC_ADDR1L[15:8] bits
ENET_ADDRESS_MASK_BYTE2	屏蔽ENET_MAC_ADDR1L[23:16] bits
ENET_ADDRESS_MASK_BYTE3	屏蔽ENET_MAC_ADDR1L [31:24] bits
ENET_ADDRESS_MASK_BYTE4	屏蔽ENET_MAC_ADDR1H [7:0] bits
ENET_ADDRESS_MASK_BYTE5	屏蔽ENET_MAC_ADDR1H [15:8] bits
输入参数{in}	
filter_type	选择MAC地址过滤器类型， 下列参数仅可选择一个
ENET_ADDRESS_FILTER_SA	过滤器比对接收帧MAC地址的源地址域
ENET_ADDRESS_FILTER_DA	过滤器比对接收帧MAC地址的目的地址域
输出参数{out}	
-	-
返回值	
-	-

例如：

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

```
enet_address_filter_config(ENET0, ENET_MAC_ADDRESS1, ENET_ADDRESS_MASK_BYTE0 | ENET_ADDRESS_MASK_BYTE1 | ENET_ADDRESS_MASK_BYTE2, ENET_ADDRESS_FILTER_DA);
```

函数 `enet_phy_config`

函数 `enet_phy_config` 描述见下表：

表 3-482. 函数 `enet_phy_config`

函数名称	<code>enet_phy_config</code>
函数原型	<code>ErrStatus enet_phy_config(uint32_t enet_periph);</code>
功能描述	PHY接口配置（配置SMI时钟并复位PHY芯片）
先决条件	-
被调用函数	<code>rcu_clock_freq_get()/enet_phy_write_read()</code>
输入参数{in}	
<code>enet_periph</code>	以太网外设
<code>ENET0</code>	以太网0
<code>ENET1</code>	以太网1
输出参数{out}	
-	-
返回值	
<code>ErrStatus</code>	ERROR or SUCCESS

例如：

```
/* config PHY interface */
enet_phy_config(ENET0);
```

函数 `enet_phy_write_read`

函数 `enet_phy_write_read` 描述见下表：

表 3-483. 函数 `enet_phy_write_read`

函数名称	<code>enet_phy_write_read</code>
函数原型	<code>ErrStatus enet_phy_write_read(uint32_t enet_periph, enet_phydirection_enum direction, uint16_t phy_address, uint16_t phy_reg, uint16_t *pvalue);</code>
功能描述	写/读PHY寄存器
先决条件	-
被调用函数	-
输入参数{in}	
<code>enet_periph</code>	以太网外设
<code>ENET0</code>	以太网0
<code>ENET1</code>	以太网1
输入参数{in}	
<code>direction</code>	参考 表3-448. 枚举类型 <code>enet_phydirection_enum</code> ，下列参数仅可选择一个

<i>ENET_PHY_WRITE</i>	向PHY寄存器写数据
<i>ENET_PHY_READ</i>	从PHY寄存器读数据
输入参数{in}	
phy_address	0x0 - 0x1F
输入参数{in}	
phy_reg	0x0 - 0x1F
输入参数{in}	
pvalue	当 direction 选择ENET_PHY_WRITE时，表示将写入PHY寄存器的值
输出参数{out}	
pvalue-	当 direction 选择ENET_PHY_READ时，表示将存储PHY寄存器读出的值
返回值	
ErrStatus	ERROR or SUCCESS

例如：

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

```
uint16_t temp_phy = 0U;
```

```
phy_state = enet_phy_write_read(ENET0, ENET_PHY_WRITE, PHY_ADDRESS,
PHY_REG_BCR, &temp_phy);
```

函数 enet_phyloopback_enable

函数enet_phyloopback_enable描述见下表：

表 3-484. 函数 enet_phyloopback_enable

函数名称	enet_phyloopback_enable
函数原型	ErrStatus enet_phyloopback_enable(uint32_t enet_periph);
功能描述	使能PHY芯片回环模式
先决条件	-
被调用函数	enet_phy_write_read()
输入参数{in}	
enet_periph	以太网外设
<i>ENET0</i>	以太网0
<i>ENET1</i>	以太网1
输出参数{out}	
-	-
返回值	
ErrStatus	ERROR or SUCCESS

例如：

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

```
ErrStatus phy_state = ERROR;
```

```
phy_state = enet_phyloopback_enable(ENET0);
```

函数 enet_phyloopback_disable

函数enet_phyloopback_disable描述见下表：

表 3-485. 函数 enet_phyloopback_disable

函数名称	enet_phyloopback_disable
函数原型	ErrStatus enet_phyloopback_disable(uint32_t enet_periph);
功能描述	禁能PHY芯片回环模式
先决条件	-
被调用函数	enet_phy_write_read
输入参数{in}	
enet_periph	以太网外设
ENET0	以太网0
ENET1	以太网1
输出参数{out}	
-	-
返回值	
ErrStatus	ERROR or SUCCESS

例如：

```
/* disable the loopback function of PHY chip */
ErrStatus phy_state = ERROR;
phy_state = enet_phyloopback_disable(ENET0);
```

函数 enet_forward_feature_enable

函数enet_forward_feature_enable描述见下表：

表 3-486. 函数 enet_forward_feature_enable

函数名称	enet_forward_feature_enable
函数原型	void enet_forward_feature_enable(uint32_t enet_periph, uint32_t feature);
功能描述	使能ENET帧通过相关功能
先决条件	-
被调用函数	-
输入参数{in}	
enet_periph	以太网外设
ENET0	以太网0
ENET1	以太网1
输入参数{in}	
feature	ENET帧通过功能，下列参数可以选择多个
ENET_AUTO_PAD_C	自动去除接收帧填充字节和FCS域

<i>RC_DROP</i>	
<i>ENET_TYPEFRAME_CRC_DROP</i>	帧通过前自动去除FCS域最后四个字节的CRC校验和
<i>ENET_FORWARD_ERRFRAMES</i>	除了过短帧外的其他错误帧都会转发给应用
<i>ENET_FORWARD_UNDSZ_GOODFRAMES</i>	帧长小于64字节但没有错误的帧将转发给应用
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* enable the function that forwarding undersized good frames */
```

```
enet_forward_feature_enable(ENET0, ENET_FORWARD_UNDSZ_GOODFRAMES);
```

函数 `enet_forward_feature_disable`

函数 `enet_forward_feature_disable` 描述见下表：

表 3-487. 函数 `enet_forward_feature_disable`

函数名称	<code>enet_forward_feature_disable</code>
函数原型	<code>void enet_forward_feature_disable(uint32_t enet_periph, uint32_t feature);</code>
功能描述	禁能ENET帧通过相关功能
先决条件	-
被调用函数	-
输入参数{in}	
enet_periph	以太网外设
<i>ENET0</i>	以太网0
<i>ENET1</i>	以太网1
输入参数{in}	
feature	ENET帧通过功能，下列参数可以选择多个
<i>ENET_AUTO_PAD_CRC_DROP</i>	自动去除接收帧填充字节和FCS域
<i>ENET_TYPEFRAME_CRC_DROP</i>	帧通过前自动去除FCS域最后四个字节的CRC校验和
<i>ENET_FORWARD_ERRFRAMES</i>	除了过短帧外的其他错误帧都会转发给应用
<i>ENET_FORWARD_UNDSZ_GOODFRAMES</i>	帧长小于64字节但没有错误的帧将转发给应用
输出参数{out}	

-	-
返回值	
-	-

例如：

```
/* enable the function that forwarding undersized good frames */
```

```
enet_forward_feature_enable(ENET0, ENET_FORWARD_UNDERSZ_GOODFRAMES);
```

函数 **enet_fliter_feature_enable**

函数 **enet_fliter_feature_enable** 描述见下表：

表 3-488. 函数 enet_fliter_feature_enable

函数名称	enet_fliter_feature_enable
函数原型	void enet_fliter_feature_enable(uint32_t enet_periph, uint32_t feature);
功能描述	使能ENET帧过滤器相关功能
先决条件	-
被调用函数	-
输入参数{in}	
enet_periph	以太网外设
ENET0	以太网0
ENET1	以太网1
输入参数{in}	
feature	ENET过滤器功能，下列参数可以选择多个
ENET_SRC_FILTER	源地址过滤器
ENET_SRC_FILTER_INVERSE	源地址过滤结果逆转
ENET_DEST_FILTER_INVERSE	目的地址过滤结果逆转
ENET_MULTICAST_FILTER_PASS	接收多播帧
ENET_MULTICAST_FILTER_HASH_MODE	HASH多播过滤器
ENET_UNICAST_FILTER_HASH_MODE	HASH单播过滤器
ENET_FILTER_MODE_EITHER	HASH或完美过滤器功能
输出参数{out}	
-	-
返回值	
-	-

例如：

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

```
enet_fliter_feature_enable(ENET0, ENET_SRC_FILTER);
```

函数 `enet_fliter_feature_disable`

函数 `enet_fliter_feature_disable` 描述见下表：

表 3-489. 函数 `enet_fliter_feature_disable`

函数名称	<code>enet_fliter_feature_disable</code>
函数原型	<code>void enet_fliter_feature_disable(uint32_t enet_periph, uint32_t feature);</code>
功能描述	禁能ENET帧过滤器相关功能
先决条件	-
被调用函数	-
输入参数{in}	
<code>enet_periph</code>	以太网外设
<code>ENET0</code>	以太网0
<code>ENET1</code>	以太网1
输入参数{in}	
<code>feature</code>	ENET过滤器功能，下列参数可以选择多个
<code>ENET_SRC_FILTER</code>	源地址过滤器
<code>ENET_SRC_FILTER_INVERSE</code>	源地址过滤结果逆转
<code>ENET_DEST_FILTER_INVERSE</code>	目的地址过滤结果逆转
<code>ENET_MULTICAST_FILTER_PASS</code>	接收多播帧
<code>ENET_MULTICAST_FILTER_HASH_MODE</code>	HASH多播过滤器
<code>ENET_UNICAST_FILTER_HASH_MODE</code>	HASH单播过滤器
<code>ENET_FILTER_MODE_EITHER</code>	HASH或完美过滤器功能
输出参数{out}	
-	-
返回值	
-	-

例如：

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

```
enet_fliter_feature_enable(ENET0, ENET_SRC_FILTER);
```


函数 `enet_pauseframe_generate`

函数 `enet_pauseframe_generate` 描述见下表：

表 3-490. 函数 `enet_pauseframe_generate`

函数名称	<code>enet_pauseframe_generate</code>
函数原型	<code>ErrStatus enet_pauseframe_generate(uint32_t enet_periph);</code>
功能描述	生成暂停帧，使能发送流控功能后ENET模块将发送暂停帧
先决条件	-
被调用函数	-
输入参数{in}	
<code>enet_periph</code>	以太网外设
<code>ENET0</code>	以太网0
<code>ENET1</code>	以太网1
输出参数{out}	
-	-
返回值	
<code>ErrStatus</code>	ERROR or SUCCESS

例如：

```
/* generate the pause frame */
```

```
ErrStatus reval;
```

```
reval = enet_pauseframe_generate(ENET0);
```

函数 `enet_pauseframe_detect_config`

函数 `enet_pauseframe_detect_config` 描述见下表：

表 3-491. 函数 `enet_pauseframe_detect_config`

函数名称	<code>enet_pauseframe_detect_config</code>
函数原型	<code>void enet_pauseframe_detect_config(uint32_t enet_periph, uint32_t detect);</code>
功能描述	配置暂停帧检测类型
先决条件	-
被调用函数	-
输入参数{in}	
<code>enet_periph</code>	以太网外设
<code>ENET0</code>	以太网0
<code>ENET1</code>	以太网1
输入参数{in}	
<code>detect</code>	暂停帧检测，下列参数仅可选择一个
<code>ENET_MAC0_AND_UNIQUE_ADDRESS_PAUSEDetect</code>	除了唯一多播地址的暂停帧，MAC同时还会使用MAC0地址（ENET_MAC_ADDR0H寄存器和ENET_MAC_ADDR0L寄存器）来检测暂停帧

ENET_UNIQUE_PA USEDETECT	MAC只接收符合IEEE802.3规范定义的唯一多播地址的暂停帧
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* config the pause frame detect type as ENET_UNIQUE_PAUSEDDETECT */
enet_pauseframe_detect_config(ENET0, ENET_UNIQUE_PAUSEDDETECT);
```

函数 enet_pauseframe_config

函数enet_pauseframe_config描述见下表：

表 3-492. 函数 enet_pauseframe_config

函数名称	enet_pauseframe_config
函数原型	void enet_pauseframe_config(uint32_t enet_periph, uint32_t pausetime, uint32_t pause_threshold);
功能描述	配置暂停帧参数
先决条件	-
被调用函数	-
输入参数{in}	
enet_periph	以太网外设
ENET0	以太网0
ENET1	以太网1
输入参数{in}	
pausetime	暂停控制帧时间域，范围(0~0xFFFF)
输入参数{in}	
pause_threshold	设置了自动重发暂停帧的定时器阈值。这个阈值应当大于0，小于位[31:16]定义的暂停时间。低阈值的计算公式为PTM-PLTS。例如，PTM = 0x80（128个时间间隙），PLTS = 0x1（28个时间间隙），那么在第一个暂停帧发出100(128-28)个时间间隙后，将自动重发第二个暂停帧，下列参数仅可选择一个
ENET_PAUSETIME _MINUS4	暂停时间 - 4个时间间隙
ENET_PAUSETIME _MINUS28	暂停时间 - 28个时间间隙
ENET_PAUSETIME _MINUS144	暂停时间 - 144个时间间隙
ENET_PAUSETIME _MINUS256	暂停时间 - 256个时间间隙
输出参数{out}	
-	-

返回值	
-	-

例如：

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

```
enet_pauseframe_config(ENET0, 30, ENET_PAUSETIME_MINUS4);
```

函数 enet_flowcontrol_threshold_config

函数enet_flowcontrol_threshold_config描述见下表：

表 3-493. 函数 enet_flowcontrol_threshold_config

函数名称	enet_flowcontrol_threshold_config
函数原型	void enet_flowcontrol_threshold_config(uint32_t enet_periph, uint32_t deactive, uint32_t active);
功能描述	配置流控阈值
先决条件	-
被调用函数	-
输入参数{in}	
enet_periph	以太网外设
ENET0	以太网0
ENET1	以太网1
输入参数{in}	
deactive	流控失效的阈值。这个值应当小于位[2:0]定义的流控激活阈值。当RxFIFO中未处理的数据低于这些位所设置的值，流控功能将自动失效，下列参数仅可选择 一个
ENET_DEACTIVE_THRESHOLD_256BYTES	256字节
ENET_DEACTIVE_THRESHOLD_512BYTES	512字节
ENET_DEACTIVE_THRESHOLD_768BYTES	768字节
ENET_DEACTIVE_THRESHOLD_1024BYTES	1024字节
ENET_DEACTIVE_THRESHOLD_1280BYTES	1280字节
ENET_DEACTIVE_THRESHOLD_1536B	1536字节

YTES	
ENET_DEACTIVE_THRESHOLD_1792BYTES	1792字节
YTES	
输入参数{in}	
active	流控激活的阈值。若使能了流控功能，当RxFIFO中未处理的数据超过了这些位所设置的值，流控功能将被激活，下列参数仅可选择一个
ENET_ACTIVE_THRESHOLD_256BYTES	256字节
ENET_ACTIVE_THRESHOLD_512BYTES	512字节
ENET_ACTIVE_THRESHOLD_768BYTES	768字节
ENET_ACTIVE_THRESHOLD_1024BYTES	1024字节
ENET_ACTIVE_THRESHOLD_1280BYTES	1280字节
ENET_ACTIVE_THRESHOLD_1536BYTES	1536字节
ENET_ACTIVE_THRESHOLD_1792BYTES	1792字节
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* configure the threshold of the flow control */
```

```
enet_flowcontrol_threshold_config(ENET0, ENET_DEACTIVE_THRESHOLD_256BYTES,
ENET_ACTIVE_THRESHOLD_256BYTES);
```

函数 enet_flowcontrol_feature_enable

函数enet_flowcontrol_feature_enable描述见下表：

表 3-494. 函数 `enet_flowcontrol_feature_enable`

函数名称	<code>enet_flowcontrol_feature_enable</code>
函数原型	<code>void enet_flowcontrol_feature_enable(uint32_t enet_periph, uint32_t feature);</code>
功能描述	使能ENET流控相关功能
先决条件	-
被调用函数	-
输入参数{in}	
<code>enet_periph</code>	以太网外设
<code>ENET0</code>	以太网0
<code>ENET1</code>	以太网1
输入参数{in}	
<code>feature</code>	ENET流控功能模式，下列参数可以选择多个
<code>ENET_ZERO_QUANTA_PAUSE</code>	零时间片暂停控制帧自动生成
<code>ENET_TX_FLOWCONTROL</code>	发送流控功能
<code>ENET_RX_FLOWCONTROL</code>	接收流控功能
<code>ENET_BACKPRESSURE</code>	背压功能（仅在半双工模式下）
输出参数{out}	
-	-
返回值	
-	-

例如：

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

```
enet_flowcontrol_feature_enable(ENET0, ENET_ZERO_QUANTA_PAUSE);
```

函数 `enet_flowcontrol_feature_disable`

函数`enet_flowcontrol_feature_disable`描述见下表：

表 3-495. 函数 `enet_flowcontrol_feature_disable`

函数名称	<code>enet_flowcontrol_feature_disable</code>
函数原型	<code>void enet_flowcontrol_feature_disable(uint32_t enet_periph, uint32_t feature);</code>
功能描述	禁能ENET流控相关功能
先决条件	-
被调用函数	-
输入参数{in}	
<code>enet_periph</code>	以太网外设
<code>ENET0</code>	以太网0
<code>ENET1</code>	以太网1

输入参数{in}	
feature	ENET流控功能模式，下列参数可以选择多个
<i>ENET_ZERO_QUANTA_PAUSE</i>	零时间片暂停控制帧自动生成
<i>ENET_TX_FLOWCONTROL</i>	发送流控功能
<i>ENET_RX_FLOWCONTROL</i>	接收流控功能
<i>ENET_BACKPRESSURE</i>	背压功能（仅在半双工模式下）
输出参数{out}	
-	-
返回值	
-	-

例如：

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

```
enet_flowcontrol_feature_disable(ENET0, ENET_ZERO_QUANTA_PAUSE);
```

函数 `enet_dmaprocess_state_get`

函数 `enet_dmaprocess_state_get` 描述见下表：

表 3-496. 函数 `enet_dmaprocess_state_get`

函数名称	<code>enet_dmaprocess_state_get</code>
函数原型	<code>uint32_t enet_dmaprocess_state_get(uint32_t enet_periph, enet_dmadirection_enum direction);</code>
功能描述	获取DMA发送/接收流程状态
先决条件	-
被调用函数	-
输入参数{in}	
enet_periph	以太网外设
<i>ENET0</i>	以太网0
<i>ENET1</i>	以太网1
输入参数{in}	
direction	DMA传输方向
<i>ENET_DMA_TX</i>	DMA发送进程
<i>ENET_DMA_RX</i>	DMA接收进程
输出参数{out}	
-	-
返回值	
uint32_t	DMA流程状态，可取值如下： <i>ENET_RX_STATE_STOPPED</i> / <i>ENET_RX_STATE_FETCHING</i> /

	<i>ENET_RX_STATE_WAITING / ENET_RX_STATE_SUSPENDED / ENET_RX_STATE_CLOSING / ENET_RX_STATE_QUEUING / ENET_TX_STATE_STOPPED / ENET_TX_STATE_FETCHING / ENET_TX_STATE_WAITING / ENET_TX_STATE_READING / ENET_TX_STATE_SUSPENDED / ENET_TX_STATE_CLOSING</i>
--	---

例如：

```
/* get the dma receive process state */

uint32_t reval;

reval = enet_dmaprocess_state_get(ENET0, ENET_DMA_RX);

if(ENET_RX_STATE_SUSPENDED == reval){

    do...

}
```

函数 enet_dmaprocess_resume

函数enet_dmaprocess_resume描述见下表：

表 3-497. 函数 enet_dmaprocess_resume

函数名称	enet_dmaprocess_resume
函数原型	void enet_dmaprocess_resume(uint32_t enet_periph, enet_dmadiirection_enum direction);
功能描述	DMA发送/接收查询使能
先决条件	-
被调用函数	-
输入参数{in}	
enet_periph	以太网外设
<i>ENET0</i>	以太网0
<i>ENET1</i>	以太网1
输入参数{in}	
direction	描述符类型，下列参数仅可选择一个
<i>ENET_DMA_TX</i>	DMA发送进程
<i>ENET_DMA_RX</i>	DMA接收进程
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* enable DMA receive process */

enet_dmaprocess_resume(ENET0, ENET_DMA_RX);
```

函数 enet_rxprocess_check_recovery

函数enet_rxprocess_check_recovery描述见下表：

表 3-498. 函数 enet_rxprocess_check_recovery

函数名称	enet_rxprocess_check_recovery
函数原型	void enet_rxprocess_check_recovery(uint32_t enet_periph);
功能描述	检测并恢复接收流程
先决条件	-
被调用函数	-
输入参数{in}	
enet_periph	以太网外设
ENET0	以太网0
ENET1	以太网1
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* check and recover the Rx process */
enet_rxprocess_check_recovery(ENET0);
```

函数 enet_txfifo_flush

函数enet_txfifo_flush描述见下表：

表 3-499. 函数 enet_txfifo_flush

函数名称	enet_txfifo_flush
函数原型	ErrStatus enet_txfifo_flush(uint32_t enet_periph);
功能描述	刷新ENET发送FIFO，并等待刷新操作完成
先决条件	-
被调用函数	-
输入参数{in}	
enet_periph	以太网外设
ENET0	以太网0
ENET1	以太网1
输出参数{out}	
-	-
返回值	
ErrStatus	ERROR or SUCCESS

例如：

```
/* flush the ENET transmit FIFO */
```



```
ErrStatus reval = ERROR;
```

```
reval = enet_txfifo_flush(ENET0);
```

函数 enet_current_desc_address_get

函数enet_current_desc_address_get描述见下表：

表 3-500. 函数 enet_current_desc_address_get

函数名称	enet_current_desc_address_get
函数原型	uint32_t enet_current_desc_address_get(uint32_t enet_periph, enet_desc_reg_enum addr_get);
功能描述	获取当前发送/接收描述符地址、当前缓冲区地址、描述符列表首地址
先决条件	-
被调用函数	-
输入参数{in}	
enet_periph	以太网外设
ENET0	以太网0
ENET1	以太网1
输入参数{in}	
addr_get	可获取的描述符地址类型，参考 表3-440. 枚举类型enet_desc_reg_enum ，下列参数仅可选择一个
ENET_RX_DESC_TABLE	接收描述符列表首地址
ENET_RX_CURRENT_DESC	当前DMA控制器使用的接收描述符地址
ENET_RX_CURRENT_BUFFER	当前DMA控制器使用的接收描述符缓冲区地址
ENET_TX_DESC_TABLE	发送描述符列表首地址
ENET_TX_CURRENT_DESC	当前DMA控制器使用的发送描述符地址
ENET_TX_CURRENT_BUFFER	当前DMA控制器使用的发送描述符缓冲区地址
输出参数{out}	
-	-
返回值	
uint32_t	0- 0xFFFFFFFF

例如：

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

```
uint32_t reval;
```

```
reval = enet_current_desc_address_get(ENET0, ENET_RX_DESC_TABLE);
```

函数 `enet_desc_information_get`

函数 `enet_desc_information_get` 描述见下表：

表 3-501. 函数 `enet_desc_information_get`

函数名称	<code>enet_desc_information_get</code>
函数原型	<code>uint32_t enet_desc_information_get(uint32_t enet_periph, enet_descriptors_struct *desc, enet_descstate_enum info_get);</code>
功能描述	获取发送/接收描述符详细信息
先决条件	-
被调用函数	-
输入参数{in}	
<code>enet_periph</code>	以太网外设
<code>ENET0</code>	以太网0
<code>ENET1</code>	以太网1
输入参数{in}	
<code>desc</code>	描述符指针，结构体成员介绍请参考 表3-433. 结构体 <code>enet_descriptors_struct</code>
输入参数{in}	
<code>info_get</code>	可选择的描述符信息类型，参考 表3-451. 枚举类型 <code>enet_descstate_enum</code> ，下列参数仅可选择一个
<code>TXDESC_COLLISION_COUNT</code>	帧发送出去前出现的冲突次数
<code>TXDESC_BUFFER_1_ADDR</code>	发送帧的缓冲区地址
<code>RXDESC_FRAME_LENGTH</code>	接收帧长度
<code>RXDESC_BUFFER_1_SIZE</code>	接收缓冲区1大小
<code>RXDESC_BUFFER_2_SIZE</code>	接收缓冲区2大小
<code>RXDESC_BUFFER_1_ADDR</code>	接收帧的缓冲区地址
输出参数{out}	
-	-
返回值	
<code>uint32_t</code>	描述符信息，如果返回值为0xFFFFFFFFU，说明输入参数有误

例如：

```
/* get the reception buffer 1 size */
```

```
uint32_t reval;
```

```
reval = enet_desc_information_get(ENET0, rx_desc, RXDESC_BUFFER_1_SIZE);
```

函数 `enet_missed_frame_counter_get`

函数 `enet_missed_frame_counter_get` 描述见下表：

表 3-502. 函数 `enet_missed_frame_counter_get`

函数名称	<code>enet_missed_frame_counter_get</code>
函数原型	<code>void enet_missed_frame_counter_get(uint32_t enet_periph, uint32_t *rxfifo_drop, uint32_t *rxdma_drop);</code>
功能描述	获取接收丢弃帧数
先决条件	-
被调用函数	-
输入参数{in}	
<code>enet_periph</code>	以太网外设
<code>ENET0</code>	以太网0
<code>ENET1</code>	以太网1
输出参数{out}	
<code>rxfifo_drop</code>	存储由于过短帧或接收FIFO溢出而丢弃帧数的指针
输出参数{out}	
<code>rxdma_drop</code>	存储由于接收描述符不可用而丢弃帧数的指针
返回值	
-	-

例如：

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

```
uint32_t rxcnt, txcnt;
```

```
enet_missed_frame_counter_get(ENET0, &rxcnt, &txcnt);
```

函数 `enet_desc_flag_get`

函数 `enet_desc_flag_get` 描述见下表：

表 3-503. 函数 `enet_desc_flag_get`

函数名称	<code>enet_desc_flag_get</code>
函数原型	<code>FlagStatus enet_desc_flag_get(enet_descriptors_struct *desc, uint32_t desc_flag);</code>
功能描述	获取ENET模块DMA描述符标志位
先决条件	-
被调用函数	-
输入参数{in}	
<code>desc</code>	描述符指针，结构体成员介绍请参考 表3-433. 结构体 <code>enet_descriptors_struct</code>
输入参数{in}	
<code>desc_flag</code> (the value according	描述符标志位，下列参数仅可选择一个

to the parameter desc)	
当 desc 参数为发送描述符时	
<i>ENET_TDES0_DB</i>	顺延位
<i>ENET_TDES0_UFE</i>	数据下溢错误位
<i>ENET_TDES0_EXD</i>	过度顺延位
<i>ENET_TDES0_VFR</i> <i>M</i>	VLAN帧位
<i>ENET_TDES0_ECO</i>	过度冲突位
<i>ENET_TDES0_LCO</i>	延迟冲突位
<i>ENET_TDES0_NCA</i>	无载波位
<i>ENET_TDES0_LCA</i>	载波丢失位
<i>ENET_TDES0_IPPE</i>	IP数据错误位
<i>ENET_TDES0_FRM</i> <i>F</i>	帧清空位
<i>ENET_TDES0_JT</i>	Jabber超时位
<i>ENET_TDES0_ES</i>	错误汇总
<i>ENET_TDES0_IPHE</i>	IP报头错误位
<i>ENET_TDES0_TTM</i> <i>SS</i>	发送时间戳状态位
<i>ENET_TDES0_TCH</i> <i>M</i>	第二地址链表模式位
<i>ENET_TDES0_TER</i> <i>M</i>	环形发送结束模式位
<i>ENET_TDES0_TTS</i> <i>EN</i>	使能发送时间戳位
<i>ENET_TDES0_DPA</i> <i>D</i>	不填充位
<i>ENET_TDES0_DCR</i> <i>C</i>	不计算CRC位
<i>ENET_TDES0_FSG</i>	第一分块位
<i>ENET_TDES0_LSG</i>	最后分块位
<i>ENET_TDES0_INTC</i>	完成时中断位
<i>ENET_TDES0_DAV</i>	DAV位
当 desc 参数为接收描述符时	
<i>ENET_RDES0_PCE</i> <i>RR</i>	数据校验和错误
<i>ENET_RDES0_CER</i> <i>R</i>	CRC错误
<i>ENET_RDES0_DBE</i> <i>RR</i>	Dribble位错误
<i>ENET_RDES0_RER</i>	接收错误

<i>R</i>	
<i>ENET_RDES0_RWD</i> <i>T</i>	接收看门狗超时
<i>ENET_RDES0_FRM</i> <i>T</i>	帧类型
<i>ENET_RDES0_LCO</i>	延迟冲突位
<i>ENET_RDES0_IPHE</i> <i>RR</i>	IP帧报头校验和错误
<i>ENET_RDES0_LDE</i> <i>S</i>	最后一个描述符
<i>ENET_RDES0_FDE</i> <i>S</i>	第一个描述符
<i>ENET_RDES0_VTA</i> <i>G</i>	VLAN标签位
<i>ENET_RDES0_OER</i> <i>R</i>	溢出错误位
<i>ENET_RDES0_LER</i> <i>R</i>	长度错误位
<i>ENET_RDES0_SAF</i> <i>F</i>	未通过源地址过滤器位
<i>ENET_RDES0_DER</i> <i>R</i>	描述符错误位
<i>ENET_RDES0_ERR</i> <i>S</i>	错误汇总位
<i>ENET_RDES0_DAF</i> <i>F</i>	未通过目标地址过滤器位
<i>ENET_RDES0_DAV</i>	描述符可用位
输出参数{out}	
-	-
返回值	
FlagStatus	SET or RESET

例如：

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

```
FlagStatus reval;
```

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

函数 enet_desc_flag_set

函数enet_desc_flag_set描述见下表：

表 3-504. 函数 enet_desc_flag_set

函数名称	enet_desc_flag_set
------	--------------------

函数原型	void enet_desc_flag_set(enet_descriptors_struct *desc, uint32_t desc_flag);
功能描述	设置ENET模块DMA描述符标志位
先决条件	-
被调用函数	-
输入参数{in}	
desc	描述符指针，结构体成员介绍请参考 表3-433. 结构体enet_descriptors_struct
输入参数{in}	
desc_flag (the value according to the parameter desc)	描述符标志位，下列参数仅可选择一个
当desc参数为发送描述符时	
ENET_TDES0_VFRM	VLAN帧位
ENET_TDES0_FRM	帧清空位
ENET_TDES0_TCHM	第二地址链表模式位
ENET_TDES0_TERM	环形发送结束模式位
ENET_TDES0_TTS	使能发送时间戳位
ENET_TDES0_DPA	不填充位
ENET_TDES0_DCR	不计算CRC位
ENET_TDES0_FSG	第一分块位
ENET_TDES0_LSG	最后分块位
ENET_TDES0_INTC	完成时中断位
ENET_TDES0_DAV	DAV位
当desc参数为接收描述符时	
ENET_RDES0_DAV	描述符可用位
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* set VLAN frame bit flag of ENET DMA descriptor */
enet_desc_flag_set(p_txdesc, ENET_TDES0_VFRM);
```

函数 `enet_desc_flag_clear`

函数 `enet_desc_flag_clear` 描述见下表：

表 3-505. 函数 `enet_desc_flag_clear`

函数名称	<code>enet_desc_flag_clear</code>
函数原型	<code>void enet_desc_flag_clear(enet_descriptors_struct *desc, uint32_t desc_flag);</code>
功能描述	清除ENET模块DMA描述符标志位
先决条件	-
被调用函数	-
输入参数{in}	
desc	描述符指针，结构体成员介绍请参考 表3-433. 结构体 <code>enet_descriptors_struct</code>
输入参数{in}	
desc_flag (the value according to the parameter desc)	描述符标志位，下列参数仅可选择一个
当 desc 参数为发送描述符时	
<code>ENET_TDES0_VFRM</code>	VLAN帧位
<code>ENET_TDES0_FRMF</code>	帧清空位
<code>ENET_TDES0_TCHM</code>	第二地址链表模式位
<code>ENET_TDES0_TERM</code>	环形发送结束模式位
<code>ENET_TDES0_TTS</code>	使能发送时间戳位
<code>ENET_TDES0_DPAD</code>	不填充位
<code>ENET_TDES0_DCR</code>	不计算CRC位
<code>ENET_TDES0_FSG</code>	第一分块位
<code>ENET_TDES0_LSG</code>	最后分块位
<code>ENET_TDES0_INTC</code>	完成时中断位
<code>ENET_TDES0_DAV</code>	DAV位
当 desc 参数为接收描述符时	
<code>ENET_RDES0_DAV</code>	描述符可用位
输出参数{out}	
-	-
返回值	
-	-

例如：

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

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

函数 `enet_rx_desc_immediate_receive_complete_interrupt`

函数 `enet_rx_desc_immediate_receive_complete_interrupt` 描述见下表：

表 3-506. 函数 `enet_rx_desc_immediate_receive_complete_interrupt`

函数名称	<code>enet_rx_desc_immediate_receive_complete_interrupt</code>
函数原型	<code>void enet_rx_desc_immediate_receive_complete_interrupt(enet_descriptors_struct *desc);</code>
功能描述	当接收完成时，立即置位 <code>ENET_DMA_STAT</code> 寄存器的 RS 位
先决条件	-
被调用函数	-
输入参数{in}	
desc	描述符指针，结构体成员介绍请参考 表3-433. 结构体 <code>enet_descriptors_struct</code>
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* set RS bit in ENET_DMA_STAT register immediately when receiving completed */
```

```
enet_rx_desc_immediate_receive_complete_interrupt(p_rxdesc);
```

函数 `enet_rx_desc_delay_receive_complete_interrupt`

函数 `enet_rx_desc_delay_receive_complete_interrupt` 描述见下表：

表 3-507. 函数 `enet_rx_desc_delay_receive_complete_interrupt`

函数名称	<code>enet_rx_desc_delay_receive_complete_interrupt</code>
函数原型	<code>void enet_rx_desc_delay_receive_complete_interrupt(uint32_t enet_periph, enet_descriptors_struct *desc, uint32_t delay_time);</code>
功能描述	当接收完成时，延迟指定时间再置位 <code>ENET_DMA_STAT</code> 寄存器的 RS 位
先决条件	-
被调用函数	-
输入参数{in}	
enet_periph	以太网外设
<code>ENET0</code>	以太网0
<code>ENET1</code>	以太网1
输入参数{in}	
desc	描述符指针，结构体成员介绍请参考 表3-433. 结构体 <code>enet_descriptors_struct</code>

输入参数{in}	
delay_time	延迟时间，实际延迟时间为256*delay_time个HCLK（0~0xFF）
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* when receiving completed, RS bit in ENET_DMA_STAT register will be set after 256*16 HCLK */
```

```
enet_rx_desc_delay_receive_complete_interrupt(ENET0, p_rxdesc, 0x00000010);
```

函数 enet_rxframe_drop

函数enet_rxframe_drop描述见下表：

表 3-508. 函数 enet_rxframe_drop

函数名称	enet_rxframe_drop
函数原型	void enet_rxframe_drop(uint32_t enet_periph);
功能描述	丢弃当前接收到的帧
先决条件	-
被调用函数	-
输入参数{in}	
enet_periph	以太网外设
ENET0	以太网0
ENET1	以太网1
输出参数{out}	
-	-
返回值	
-	-

例如：

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

```
enet_rxframe_drop(ENET0);
```

函数 enet_dma_feature_enable

函数enet_dma_feature_enable描述见下表：

表 3-509. 函数 enet_dma_feature_enable

函数名称	enet_dma_feature_enable
函数原型	void enet_dma_feature_enable(uint32_t enet_periph, uint32_t feature);
功能描述	使能ENET模块DMA相关功能

先决条件	-
被调用函数	-
输入参数{in}	
enet_periph	以太网外设
ENET0	以太网0
ENET1	以太网1
输入参数{in}	
feature	DMA功能，下列参数可以选择多个
ENET_NO_FLUSH_RXFRAME	描述符不可用时，RxDMA控制器清空接收帧功能
ENET_SECONDFRAME_OPT	TxDMA控制器第二帧功能
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* enable RxDMA does not flushes frames function */
```

```
enet_dma_feature_enable(ENET0, ENET_NO_FLUSH_RXFRAME);
```

函数 enet_dma_feature_disable

函数enet_dma_feature_disable描述见下表：

表 3-510. 函数 enet_dma_feature_disable

函数名称	enet_dma_feature_disable
函数原型	void enet_dma_feature_disable(uint32_t enet_periph, uint32_t feature);
功能描述	禁能ENET模块DMA相关功能
先决条件	-
被调用函数	-
输入参数{in}	
enet_periph	以太网外设
ENET0	以太网0
ENET1	以太网1
输入参数{in}	
feature	DMA功能，下列参数可以选择多个
ENET_NO_FLUSH_RXFRAME	描述符不可用时，RxDMA控制器清空接收帧功能
ENET_SECONDFRAME_OPT	TxDMA控制器第二帧功能
输出参数{out}	
-	-

返回值	
-	-

例如：

```
/* disable RxDMA does not flushes frames function */
```

```
enet_dma_feature_disable(ENET0, ENET_NO_FLUSH_RXFRAME);
```

函数 enet_rx_desc_enhanced_status_get

函数enet_rx_desc_enhanced_status_get描述见下表：

表 3-511. 函数 enet_rx_desc_enhanced_status_get

函数名称	enet_rx_desc_enhanced_status_get
函数原型	uint32_t enet_rx_desc_enhanced_status_get(enet_descriptors_struct *desc, uint32_t desc_status);
功能描述	获取接收描述符增强状态标志位信息
先决条件	-
被调用函数	-
输入参数{in}	
desc	描述符指针，结构体成员介绍请参考 表3-433. 结构体enet_descriptors_struct
输入参数{in}	
desc_status	想要获取的状态信息
ENET_RDES4_IPPLDT	IP帧有效载荷
ENET_RDES4_IPHERR	IP帧头错误
ENET_RDES4_IPPLDERR	IP帧有效载荷错误
ENET_RDES4_IPCKSB	IP帧旁路校验和
ENET_RDES4_IPF4	Ipv4帧
ENET_RDES4_IPF6	Ipv6帧
ENET_RDES4_PTPMT	PTP消息类型
ENET_RDES4_PTPOEF	PTP网络帧
ENET_RDES4_PTPVF	PTP版本格式
输出参数{out}	
-	-
返回值	
uint32_t	相应的状态值

例如：

```
/* get the IP frame payload type in ENET DMA descriptor */
uint32_t status;

status = enet_rx_desc_enhanced_status_get(p_rxdesc, ENET_RDES4_IPPLDT);
```

函数 enet_desc_select_enhanced_mode

函数enet_desc_select_enhanced_mode描述见下表：

表 3-512. 函数 enet_desc_select_enhanced_mode

函数名称	enet_desc_select_enhanced_mode
函数原型	void enet_desc_select_enhanced_mode(uint32_t enet_periph);
功能描述	配置DMA描述符为增强型描述符
先决条件	-
被调用函数	-
输入参数{in}	
enet_periph	以太网外设
ENET0	以太网0
ENET1	以太网1
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* configure descriptor to work in enhanced mode */
enet_desc_select_enhanced_mode(ENET0);
```

函数 enet_ptp_enhanced_descriptors_chain_init

函数enet_ptp_enhanced_descriptors_chain_init描述见下表：

表 3-513. 函数 enet_ptp_enhanced_descriptors_chain_init

函数名称	enet_ptp_enhanced_descriptors_chain_init
函数原型	void enet_ptp_enhanced_descriptors_chain_init(uint32_t enet_periph, enet_dmadirection_enum direction);
功能描述	初始化具有PTP功能的增强型DMA接收/发送描述符为链模式
先决条件	-
被调用函数	-
输入参数{in}	
enet_periph	以太网外设
ENET0	以太网0

<i>ENET1</i>	以太网1
输入参数{in}	
direction	描述符类型，下列参数仅可选择一个
<i>ENET_DMA_TX</i>	DMA Tx描述符
<i>ENET_DMA_RX</i>	DMA Rx描述符
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* initialize the DMA Tx descriptors's parameters in enhanced chain mode with ptp function */
enet_ptp_enhanced_descriptors_chain_init(ENET0, ENET_DMA_TX);
```

函数 `enet_ptp_enhanced_descriptors_ring_init`

函数 `enet_ptp_enhanced_descriptors_ring_init` 描述见下表：

表 3-514. 函数 `enet_ptp_enhanced_descriptors_ring_init`

函数名称	<code>enet_ptp_enhanced_descriptors_ring_init</code>
函数原型	<code>void enet_ptp_enhanced_descriptors_ring_init(uint32_t enet_periph, enet_dmadirection_enum direction);</code>
功能描述	初始化具有PTP功能的DMA接收/发送描述符为环模式
先决条件	-
被调用函数	-
输入参数{in}	
enet_periph	以太网外设
<i>ENET0</i>	以太网0
<i>ENET1</i>	以太网1
输入参数{in}	
direction	描述符类型，下列参数仅可选择一个
<i>ENET_DMA_TX</i>	DMA Tx描述符
<i>ENET_DMA_RX</i>	DMA Rx描述符
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* initialize the DMA Rx descriptors's parameters in enhanced ring mode with ptp function */
enet_ptp_enhanced_descriptors_ring_init(ENET0, ENET_DMA_RX);
```

函数 enet_ptpframe_receive_enhanced_mode

函数enet_ptpframe_receive_enhanced_mode描述见下表:

表 3-515. 函数 enet_ptpframe_receive_enhanced_mode

函数名称	enet_ptpframe_receive_enhanced_mode
函数原型	ErrStatus enet_ptpframe_receive_enhanced_mode(uint32_t enet_periph, uint8_t buffer[], uint32_t bufsize, uint32_t timestamp[]);
功能描述	在PTP模式下处理当前接收到的帧，并将当前增强型描述符中存储的接收帧数据和时间戳拷贝到指定区域
先决条件	-
被调用函数	-
输入参数{in}	
enet_periph	以太网外设
ENET0	以太网0
ENET1	以太网1
输入参数{in}	
bufsize	缓冲区大小
输出参数{out}	
buffer	存放数据的用户缓冲区指针，如果输入NULL，用户需要在调用该函数之前将数据拷贝到自己指定的位置
输出参数{out}	
timestamp	存放时间戳指针
返回值	
ErrStatus	SUCCESS or ERROR

例如:

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

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

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

```
ErrStatus status;
```

```
status = enet_ptpframe_receive_enhanced_mode (ENET0, rx_buffer, 500, time_stamp);
```

函数 enet_ptpframe_transmit_enhanced_mode

函数enet_ptpframe_transmit_enhanced_mode描述见下表:

表 3-516. 函数 enet_ptpframe_transmit_enhanced_mode

函数名称	enet_ptpframe_transmit_enhanced_mode
函数原型	ErrStatus enet_ptpframe_transmit_enhanced_mode(uint32_t enet_periph, uint8_t buffer[], uint32_t length, uint32_t timestamp[]);

功能描述	在PTP模式下将制定区域内的数据拷贝到当前增强型发送描述符中，并同时间戳一起发送
先决条件	-
被调用函数	-
输入参数{in}	
enet_periph	以太网外设
ENET0	以太网0
ENET1	以太网1
输入参数{in}	
buffer	存放数据的用户缓冲区指针，如果输入NULL，用户需要在调用该函数之前将数据拷贝到指定的位置
输入参数{in}	
length	发送数据大小
输出参数{out}	
timestamp	存放时间戳的指针，如果输入为NULL，则忽略时间戳
返回值	
ErrStatus	SUCCESS or ERROR

例如：

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

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

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

```
ErrStatus status;
```

```
status = enet_ptpframe_transmit_enhanced_mode(ENET0, tx_buffer, 500, time_stamp);
```

函数 enet_desc_select_normal_mode

函数enet_desc_select_normal_mode描述见下表：

表 3-517. 函数 enet_desc_select_normal_mode

函数名称	enet_desc_select_normal_mode
函数原型	void enet_desc_select_normal_mode(uint32_t enet_periph);
功能描述	配置DMA描述符为常规型描述符
先决条件	-
被调用函数	-
输入参数{in}	
enet_periph	以太网外设
ENET0	以太网0
ENET1	以太网1
输出参数{out}	

-	-
返回值	
-	-

例如：

```
/* configure descriptor to work in normal mode */
```

```
enet_desc_select_normal_mode(ENET0);
```

函数 enet_ptp_normal_descriptors_chain_init

函数enet_ptp_normal_descriptors_chain_init描述见下表：

表 3-518. 函数 enet_ptp_normal_descriptors_chain_init

函数名称	enet_ptp_normal_descriptors_chain_init
函数原型	void enet_ptp_normal_descriptors_chain_init(uint32_t enet_periph, enet_dmadirection_enum direction, enet_descriptors_struct *desc_ptptab);
功能描述	初始化具有PTP功能的DMA接收/发送描述符为链模式
先决条件	-
被调用函数	-
输入参数{in}	
enet_periph	以太网外设
ENET0	以太网0
ENET1	以太网1
输入参数{in}	
direction	描述符类型，下列参数仅可选择一个
ENET_DMA_TX	DMA Tx描述符
ENET_DMA_RX	DMA Rx描述符
输入参数{in}	
desc_ptptab	描述符指针，结构体成员介绍请参考 表3-433. 结构体enet_descriptors_struct
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* initialize the DMA Rx descriptors's parameters in normal chain mode with PTP function */
```

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

```
enet_ptp_normal_descriptors_chain_init(ENET0, ENET_DMA_RX, ptp_rxdesc_tab);
```

函数 enet_ptp_normal_descriptors_ring_init

函数enet_ptp_normal_descriptors_ring_init描述见下表：

表 3-519. 函数 `enet_ptp_normal_descriptors_ring_init`

函数名称	<code>enet_ptp_normal_descriptors_ring_init</code>
函数原型	<code>void enet_ptp_normal_descriptors_ring_init(uint32_t enet_periph, enet_dmadiirection_enum direction, enet_descriptors_struct *desc_ptptab);</code>
功能描述	初始化具有PTP功能的DMA接收/发送描述符为环模式
先决条件	-
被调用函数	-
输入参数{in}	
<code>enet_periph</code>	以太网外设
<code>ENET0</code>	以太网0
<code>ENET1</code>	以太网1
输入参数{in}	
<code>direction</code>	描述符类型，下列参数仅可选择一个
<code>ENET_DMA_TX</code>	DMA Tx描述符
<code>ENET_DMA_RX</code>	DMA Rx描述符
输入参数{in}	
<code>desc_ptptab</code>	描述符指针，结构体成员介绍请参考 表3-433. 结构体 <code>enet_descriptors_struct</code>
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* initialize the DMA Rx descriptors's parameters in normal ring mode with PTP function */
enet_descriptors_struct ptp_rxdesc_tab[ENET_RXBUF_NUM];

enet_ptp_normal_descriptors_ring_init(ENET0, ENET_DMA_RX, ptp_rxdesc_tab);
```

函数 `enet_ptpframe_receive_normal_mode`

函数`enet_ptpframe_receive_normal_mode`描述见下表：

表 3-520. 函数 `enet_ptpframe_receive_normal_mode`

函数名称	<code>enet_ptpframe_receive_normal_mode</code>
函数原型	<code>ErrStatus enet_ptpframe_receive_normal_mode(uint32_t enet_periph, uint8_t buffer[], uint32_t bufsize, uint32_t timestamp[]);</code>
功能描述	在PTP模式下处理当前接收到的帧，并将当前描述符中存储的接收帧数据和时 间戳拷贝到指定区域
先决条件	-
被调用函数	-
输入参数{in}	
<code>enet_periph</code>	以太网外设
<code>ENET0</code>	以太网0

<i>ENET1</i>	以太网1
输入参数{in}	
bufsize	缓冲区大小
输出参数{out}	
timestamp	存放时间戳指针
输出参数{out}	
buffer	存放数据的用户缓冲区指针，如果输入NULL，用户需要在调用该函数之前将数据拷贝到自己指定的位置
返回值	
ErrStatus	ERROR or SUCCESS

例如：

```

/* receive a packet data with timestamp values to application buffer in DMA normal mode */
uint32_t rx_buffer[500];

uint32_t time_stamp[2];

ErrStatus status;

status = enet_ptpframe_receive_normal_mode (ENET0, rx_buffer, 500, time_stamp);

```

函数 enet_ptpframe_transmit_normal_mode

函数enet_ptpframe_transmit_normal_mode描述见下表：

表 3-521. 函数 enet_ptpframe_transmit_normal_mode

函数名称	enet_ptpframe_transmit_normal_mode
函数原型	ErrStatus enet_ptpframe_transmit_normal_mode(uint32_t enet_periph, uint8_t buffer[], uint32_t length, uint32_t timestamp[]);
功能描述	在PTP模式下将指定区域内的数据拷贝到当前发送描述符中，并同时时间戳一起发送
先决条件	-
被调用函数	--
输入参数{in}	
enet_periph	以太网外设
<i>ENET0</i>	以太网0
<i>ENET1</i>	以太网1
输入参数{in}	
buffer	存放数据的用户缓冲区指针，如果输入NULL，用户需要在调用该函数之前将数据拷贝到指定的位置
输入参数{in}	
length	发送数据大小
输出参数{out}	
timestamp	存放时间戳的指针，如果输入为NULL，则忽略时间戳

返回值	
ErrStatus	ERROR or SUCCESS

例如:

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

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

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

```
ErrStatus status;
```

```
status = enet_ptpframe_transmit_normal_mode(ENET0, tx_buffer, 500, time_stamp);
```

函数 enet_wum_filter_register_pointer_reset

函数enet_wum_filter_register_pointer_reset描述见下表:

表 3-522. 函数 enet_wum_filter_register_pointer_reset

函数名称	enet_wum_filter_register_pointer_reset
函数原型	void enet_wum_filter_register_pointer_reset(uint32_t enet_periph);
功能描述	远程唤醒帧过滤器寄存器指针复位
先决条件	-
被调用函数	-
输入参数{in}	
enet_periph	以太网外设
ENET0	以太网0
ENET1	以太网1
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* reset wakeup frame filter register pointer */
```

```
enet_wum_filter_register_pointer_reset(ENET0);
```

函数 enet_wum_filter_config

函数enet_wum_filter_config描述见下表:

表 3-523. 函数 enet_wum_filter_config

函数名称	enet_wum_filter_config
函数原型	void enet_wum_filter_config(uint32_t enet_periph, uint32_t pdata[]);
功能描述	配置远程唤醒帧寄存器

先决条件	-
被调用函数	-
输入参数{in}	
enet_periph	以太网外设
ENET0	以太网0
ENET1	以太网1
输入参数{in}	
pdata	存放将写入远程唤醒帧寄存器组的数据指针（总共8字节）
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* set the remote wakeup frame registers */
uint32_t wum_data[8];
enet_wum_filter_config (ENET0, wum_data);
```

函数 enet_wum_feature_enable

函数enet_wum_feature_enable描述见下表：

表 3-524. 函数 enet_wum_feature_enable

函数名称	enet_wum_feature_enable
函数原型	void enet_wum_feature_enable(uint32_t enet_periph, uint32_t feature);
功能描述	使能ENET模块唤醒管理相关功能
先决条件	-
被调用函数	-
输入参数{in}	
enet_periph	以太网外设
ENET0	以太网0
ENET1	以太网1
输入参数{in}	
feature	下列参数可以选择多个
ENET_WUM_POWER_DOWN	掉电模式
ENET_WUM_MAGIC_PACKET_FRAME	使能接收到魔法帧的唤醒事件
ENET_WUM_WAKEUP_FRAME	使能接收到唤醒帧的唤醒事件
ENET_WUM_GLOBAL_UNICAST	任何通过过滤器的单播帧均作为唤醒帧

输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* enable power down mode */
```

```
enet_wum_feature_enable(ENET0, ENET_WUM_POWER_DOWN);
```

函数 enet_wum_feature_disable

函数enet_wum_feature_disable描述见下表：

表 3-525. 函数 enet_wum_feature_disable

函数名称	enet_wum_feature_disable
函数原型	void enet_wum_feature_disable(uint32_t enet_periph, uint32_t feature);
功能描述	禁能ENET模块唤醒管理相关功能
先决条件	-
被调用函数	-
输入参数{in}	
enet_periph	以太网外设
ENET0	以太网0
ENET1	以太网1
输入参数{in}	
feature	下列参数可以选择多个
ENET_WUM_MAGIC_PACKET_FRAME	使能接收到魔法帧的唤醒事件
ENET_WUM_WAKEUP_FRAME	使能接收到唤醒帧的唤醒事件
ENET_WUM_GLOBAL_UNICAST	任何通过过滤器的单播帧均作为唤醒帧
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* disable power down mode */
```

```
enet_wum_feature_disable(ENET0, ENET_WUM_POWER_DOWN);
```

函数 enet_msc_counters_reset

函数enet_msc_counters_reset描述见下表：

表 3-526. 函数 `enet_msc_counters_reset`

函数名称	<code>enet_msc_counters_reset</code>
函数原型	<code>void enet_msc_counters_reset(uint32_t enet_periph);</code>
功能描述	复位MAC统计计数器组
先决条件	-
被调用函数	-
输入参数{in}	
enet_periph	以太网外设
<i>ENET0</i>	以太网0
<i>ENET1</i>	以太网1
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* reset the MAC statistics counters */
enet_msc_counters_reset(ENET0);
```

函数 `enet_msc_feature_enable`

函数`enet_msc_feature_enable`描述见下表：

表 3-527. 函数 `enet_msc_feature_enable`

函数名称	<code>enet_msc_feature_enable</code>
函数原型	<code>void enet_msc_feature_enable(uint32_t enet_periph, uint32_t feature);</code> <code>_feature_enable(uint32_t feature);</code>
功能描述	使能MAC统计计数器相关功能
先决条件	-
被调用函数	-
输入参数{in}	
enet_periph	以太网外设
<i>ENET0</i>	以太网0
<i>ENET1</i>	以太网1
输入参数{in}	
feature	下列参数可以选择多个
<i>ENET_MSC_COUNTER_STOP_ROLLOVER</i>	计数器停止回转
<i>ENET_MSC_RESET_ON_READ</i>	读时复位
<i>ENET_MSC_COUNTERS_FREEZE</i>	MSC计数器冻结位

输出参数{out}	
-	-
返回值	
-	-

例如：

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

```
enet_msc_feature_enable(ENET0, ENET_MSC_COUNTER_STOP_ROLLOVER);
```

函数 enet_msc_feature_disable

函数enet_msc_feature_disable描述见下表：

表 3-528. 函数 enet_msc_feature_disable

函数名称	enet_msc_feature_disable
函数原型	void enet_msc_feature_disable(uint32_t enet_periph, uint32_t feature);
功能描述	禁能MAC统计计数器相关功能
先决条件	-
被调用函数	-
输入参数{in}	
enet_periph	以太网外设
ENET0	以太网0
ENET1	以太网1
输入参数{in}	
feature	下列参数可以选择多个
ENET_MSC_COUNTER_STOP_ROLLOVER	计数器停止回转
ENET_MSC_RESET_ON_READ	读时复位
ENET_MSC_COUNTERS_FREEZE	MSC计数器冻结位
输出参数{out}	
-	-
返回值	
-	-

例如：

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

```
enet_msc_feature_disable(ENET0, ENET_MSC_COUNTER_STOP_ROLLOVER);
```

函数 **enet_msc_counters_preset_config**

函数 **enet_msc_counters_preset_config** 描述见下表：

表 3-529. 函数 **enet_msc_counters_preset_config**

函数名称	enet_msc_counters_preset_config
函数原型	void enet_msc_counters_preset_config(uint32_t enet_periph, enet_msc_preset_enum mode);
功能描述	配置MAC统计计数器的预设模式
先决条件	-
被调用函数	-
输入参数{in}	
enet_periph	以太网外设
ENET0	以太网0
ENET1	以太网1
输入参数{in}	
mode	参考 表3-452. 枚举类型enet_msc_preset_enum ，下列参数可以选择多个
ENET_MSC_PRESET_T_NONE	关闭MSC计数器预设功能
ENET_MSC_PRESET_T_HALF	预设值为最大值一半
ENET_MSC_PRESET_T_FULL	预设值为最大值一半
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* preset all MSC counters to almost-half */
```

```
enet_msc_counters_preset_config (ENET0, ENET_MSC_PRESET_HALF);
```

函数 **enet_msc_counters_get**

函数 **enet_msc_counters_get** 描述见下表：

表 3-530. 函数 **enet_msc_counters_get**

函数名称	enet_msc_counters_get
函数原型	uint32_t enet_msc_counters_get(uint32_t enet_periph, enet_msc_counter_enum counter);
功能描述	获取MAC相关统计计数器值
先决条件	-
被调用函数	-
输入参数{in}	

enet_periph	以太网外设
<i>ENET0</i>	以太网0
<i>ENET1</i>	以太网1
输入参数{in}	
counter	MSC计数器，参考 表3-441. 枚举类型enet_msc_counter_enum ，下列参数仅可选择一个
<i>ENET_MSC_TX_SCNT</i>	MSC 1次冲突后发送“好”帧的计数器
<i>ENET_MSC_TX_MSCCNT</i>	MSC 1次以上冲突后发送“好”帧的计数器
<i>ENET_MSC_TX_TGFCNT</i>	MSC发送“好”帧计数器
<i>ENET_MSC_RX_RFCENT</i>	MSC CRC错误接收帧计数器
<i>ENET_MSC_RX_RFAECNT</i>	MSC对齐错误接收帧计数器
<i>ENET_MSC_RX_RGUFCNT</i>	MSC“好”单播帧接收帧计数器
输出参数{out}	
-	-
返回值	
uint32_t	MSC计数器值

例如：

```
/* get MSC transmitted good frames after a single collision counter value*/
uint32_t reval;

reval = enet_msc_counters_get(ENET0, ENET_MSC_TX_SCNT);
```

函数 enet_ptp_subsecond_2_nanosecond

函数enet_ptp_subsecond_2_nanosecond描述见下表：

表 3-531. 函数 enet_ptp_subsecond_2_nanosecond

函数名称	enet_ptp_subsecond_2_nanosecond
函数原型	uint32_t enet_ptp_subsecond_2_nanosecond(uint32_t subsecond);
功能描述	将亚秒值变为纳秒值
先决条件	-
被调用函数	-
输入参数{in}	
subsecond	亚秒值
输出参数{out}	
-	-

返回值	
uint32_t	纳秒值

例如：

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

```
uint32_t reval;
```

```
reval = enet_ptp_subsecond_2_nanosecond (2);
```

函数 enet_ptp_nanosecond_2_subsecond

函数enet_ptp_nanosecond_2_subsecond描述见下表：

表 3-532. 函数 enet_ptp_nanosecond_2_subsecond

函数名称	enet_ptp_nanosecond_2_subsecond
函数原型	uint32_t enet_ptp_nanosecond_2_subsecond(uint32_t nanosecond);
功能描述	将纳秒值变为亚秒值
先决条件	-
被调用函数	-
输入参数{in}	
nanosecond	纳秒值
输出参数{out}	
-	-
返回值	
uint32_t	亚秒值

例如：

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

```
uint32_t reval;
```

```
reval = enet_ptp_nanosecond_2_subsecond (2);
```

函数 enet_ptp_feature_enable

函数enet_ptp_feature_enable描述见下表：

表 3-533. 函数 enet_ptp_feature_enable

函数名称	enet_ptp_feature_enable
函数原型	void enet_ptp_feature_enable(uint32_t enet_periph, uint32_t feature);
功能描述	使能PTP相关功能
先决条件	-
被调用函数	-
输入参数{in}	
enet_periph	以太网外设

<i>ENET0</i>	以太网0
<i>ENET1</i>	以太网1
输入参数{in}	
feature	ENET模块PTP功能，下列参数可以选择多个
<i>ENET_RXTX_TIMES TAMP</i>	发送/接收帧时间戳
<i>ENET_PTP_TIMEST AMP_INT</i>	时间戳中断触发
<i>ENET_ALL_RX_TIM ESTAMP</i>	所有接收帧时间戳快照使能
<i>ENET_NONTYPE_F RAME_SNAPSHOT</i>	接收以太网帧时时间戳使能
<i>ENET_IPV6_FRAME _SNAPSHOT</i>	接收Ipv6帧时时间戳使能
<i>ENET_IPV4_FRAME _SNAPSHOT</i>	接收Ipv4帧时时间戳使能
<i>ENET_PTP_FRAME _USE_MACADDRES S_FILTER</i>	PTP帧MAC地址过滤使能
输出参数{out}	
-	-
返回值	
-	-

例如：

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

```
enet_ptp_feature_enable(ENET0, ENET_ALL_RX_TIMESTAMP);
```

函数 `enet_ptp_feature_disable`

函数 `enet_ptp_feature_disable` 描述见下表：

表 3-534. 函数 `enet_ptp_feature_disable`

函数名称	<code>enet_ptp_feature_disable</code>
函数原型	<code>void enet_ptp_feature_disable(uint32_t enet_periph, uint32_t feature);</code>
功能描述	禁能PTP相关功能
先决条件	-
被调用函数	-
输入参数{in}	
enet_periph	以太网外设
<i>ENET0</i>	以太网0
<i>ENET1</i>	以太网1
输入参数{in}	

feature	ENET模块PTP功能，下列参数可以选择多个
ENET_RXTX_TIMES TAMP	发送/接收帧时间戳
ENET_PTP_TIMEST AMP_INT	时间戳中断触发
ENET_ALL_RX_TIM ESTAMP	所有接收帧时间戳快照使能
ENET_NONTYPE_F RAME_SNAPSHOT	接收以太网帧时时间戳使能
ENET_IPV6_FRAME _SNAPSHOT	接收Ipv6帧时时间戳使能
ENET_IPV4_FRAME _SNAPSHOT	接收Ipv4帧时时间戳使能
ENET_PTP_FRAME _USE_MACADDRES S_FILTER	PTP帧MAC地址过滤使能
输出参数{out}	
-	-
返回值	
-	-

例如：

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

```
enet_ptp_feature_disable(ENET_ALL_RX_TIMESTAMP);
```

函数 enet_ptp_timestamp_function_config

函数enet_ptp_timestamp_function_config描述见下表：

表 3-535. 函数 enet_ptp_timestamp_function_config

函数名称	enet_ptp_timestamp_function_config
函数原型	ErrStatus enet_ptp_timestamp_function_config(uint32_t enet_periph, enet_ptp_function_enum func);
功能描述	配置PTP时间戳相关功能
先决条件	-
被调用函数	-
输入参数{in}	
enet_periph	以太网外设
ENET0	以太网0
ENET1	以太网1
输入参数{in}	
func	下列参数仅可选择一个
ENET_CKNT_ORDI	时间戳时钟节点类型为普通时钟

NARY	
ENET_CKNT_BOUNDARY	时间戳时钟节点类型为边界时钟
ENET_CKNT_END_TO_END	时间戳时钟节点类型为端对端透明时钟
ENET_CKNT_PEER_TO_PEER	时间戳时钟节点类型为点对点透明时钟
ENET_PTP_ADDEND_UPDATE	加数寄存器更新
ENET_PTP_SYSTIME_UPDATE	时间戳更新
ENET_PTP_SYSTIME_INIT	时间戳初始化
ENET_PTP_FINEMODE	精调模式更新系统时间戳
ENET_PTP_COARSEMODE	粗调模式更新系统时间戳
ENET_SUBSECOND_DIGITAL_ROLLOVER	十进制回转模式
ENET_SUBSECOND_BINARY_ROLLOVER	二进制回转模式
ENET_SNOOPING_PTP_VERSION_2	监听PTP帧版本2
ENET_SNOOPING_PTP_VERSION_1	监听PTP帧版本1
ENET_EVENT_TYPE_MESSAGES_SNAPSHOT	只接收事件类型消息使能时间戳
ENET_ALL_TYPE_MESSAGES_SNAPSHOT	接收到除了Announce, Management和Signaling以外的所有其他类型的消息时, 时间戳快照使能
ENET_MASTER_NODE_MESSAGE_SNAPSHOT	主节点消息时间戳快照使能
ENET_SLAVE_NODE_MESSAGE_SNAPSHOT	从节点消息时间戳快照使能
输出参数{out}	
-	-
返回值	
ErrStatus	SUCCESS or ERROR

例如：

```
/* config addend register update function */
```

```
ErrStatus reval = ERROR;
```

```
reval = enet_ptp_timestamp_function_config(ENET0, ENET_PTP_ADDEND_UPDATE);
```

函数 enet_ptp_subsecond_increment_config

函数enet_ptp_subsecond_increment_config描述见下表：

表 3-536. 函数 enet_ptp_subsecond_increment_config

函数名称	enet_ptp_subsecond_increment_config
函数原型	void enet_ptp_subsecond_increment_config(uint32_t enet_periph, uint32_t subsecond);
功能描述	配置PTP系统时间亚秒增加值
先决条件	-
被调用函数	-
输入参数{in}	
enet_periph	以太网外设
ENET0	以太网0
ENET1	以太网1
输入参数{in}	
subsecond	该值将被加到系统时间的亚秒值，范围（0~0xFF）
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* configure 0x1F as system time subsecond increment value */
```

```
enet_ptp_subsecond_increment_config(ENET0, 0x1F);
```

函数 enet_ptp_timestamp_addend_config

函数enet_ptp_timestamp_addend_config描述见下表：

表 3-537. 函数 enet_ptp_timestamp_addend_config

函数名称	enet_ptp_timestamp_addend_config
函数原型	void enet_ptp_timestamp_addend_config(uint32_t enet_periph, uint32_t add);
功能描述	精调模式下PTP时钟频率校准配置
先决条件	-
被调用函数	-
输入参数{in}	

enet_periph	以太网外设
<i>ENET0</i>	以太网0
<i>ENET1</i>	以太网1
输入参数{in}	
add	通过将该值加到累加器用于时间同步，范围(0~0xFFFF FFFF)
输出参数{out}	
-	-
返回值	
-	-

例如：

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

```
enet_ptp_timestamp_addend_config(ENET0, 0x1FFF);
```

函数 enet_ptp_timestamp_update_config

函数enet_ptp_timestamp_update_config描述见下表：

表 3-538. 函数 enet_ptp_timestamp_update_config

函数名称	enet_ptp_timestamp_update_config
函数原型	void enet_ptp_timestamp_update_config(uint32_t enet_periph, uint32_t sign, uint32_t second, uint32_t subsecond);
功能描述	初始化时用于替换系统时间，在更新时表示在系统时间上加上或减去的秒值
先决条件	-
被调用函数	-
输入参数{in}	
enet_periph	以太网外设
<i>ENET0</i>	以太网0
<i>ENET1</i>	以太网1
输入参数{in}	
sign	时间戳更新正或负符号位，下列参数仅可选择一个
<i>ENET_PTP_ADD_TO_TIME</i>	更新值加到系统时间
<i>ENET_PTP_SUBSTRACT_FROM_TIME</i>	将系统时间减去更新值
输入参数{in}	
second	秒值，范围(0~0xFFFF FFFF)
输入参数{in}	
subsecond	亚秒值，精度为0.46 ns，范围(0~0x7FFF FFFF)
输出参数{out}	
-	-
返回值	
-	-

例如：

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

```
enet_ptp_timestamp_update_config(ENET0, ENET_PTP_ADD_TO_TIME, 0, 0);
```

函数 enet_ptp_expected_time_config

函数enet_ptp_expected_time_config描述见下表：

表 3-539. 函数 enet_ptp_expected_time_config

函数名称	enet_ptp_expected_time_config
函数原型	void enet_ptp_expected_time_config(uint32_t enet_periph, uint32_t second, uint32_t nanosecond);
功能描述	配置PTP期望时间
先决条件	-
被调用函数	-
输入参数{in}	
enet_periph	以太网外设
ENET0	以太网0
ENET1	以太网1
输入参数{in}	
second	目标时间秒值，范围(0~0xFFFF FFFF)
输入参数{in}	
nanosecond	目标时间纳秒值，范围(0~0xFFFF FFFF)
输出参数{out}	
-	-
返回值	
-	-

例如：

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

```
enet_ptp_expected_time_config(ENET0, 2000, 0);
```

函数 enet_ptp_system_time_get

函数enet_ptp_system_time_get描述见下表：

表 3-540. 函数 enet_ptp_system_time_get

函数名称	enet_ptp_system_time_get
函数原型	void enet_ptp_system_time_get(uint32_t enet_periph, enet_ptp_systime_struct *systime_struct);
功能描述	获取PTP当前系统时间
先决条件	-
被调用函数	-

输入参数{in}	
enet_periph	以太网外设
<i>ENET0</i>	以太网0
<i>ENET1</i>	以太网1
输出参数{out}	
systime_struct	PTP系统时间结构体指针，结构体成员介绍请参考 表3-434. 结构体 <i>enet_ptp_systime_struct</i>
返回值	
-	-

例如：

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

```
enet_ptp_systime_struct systime;
```

```
enet_ptp_system_time_get(ENET0, &systime);
```

函数 **enet_ptp_pps_output_frequency_config**

函数enet_ptp_pps_output_frequency_config描述见下表：

表 3-541. 函数 enet_ptp_pps_output_frequency_config

函数名称	enet_ptp_pps_output_frequency_config
函数原型	void enet_ptp_pps_output_frequency_config(uint32_t enet_periph, uint32_t freq);
功能描述	配置PPS输出频率
先决条件	-
被调用函数	-
输入参数{in}	
enet_periph	以太网外设
<i>ENET0</i>	以太网0
<i>ENET1</i>	以太网1
输入参数{in}	
freq	输出频率
<i>ENET_PPISOFC_1HZ</i>	PPS输出1Hz
<i>ENET_PPISOFC_2HZ</i>	PPS输出2Hz
<i>ENET_PPISOFC_4HZ</i>	PPS输出4Hz
<i>ENET_PPISOFC_8HZ</i>	PPS输出8Hz
<i>ENET_PPISOFC_16HZ</i>	PPS输出16Hz

ENET_PPISOFC_32 HZ	PPS输出32Hz
ENET_PPISOFC_64 HZ	PPS输出64Hz
ENET_PPISOFC_128 HZ	PPS输出128Hz
ENET_PPISOFC_256 HZ	PPS输出256Hz
ENET_PPISOFC_512 HZ	PPS输出512Hz
ENET_PPISOFC_1024 HZ	PPS输出1024Hz
ENET_PPISOFC_2048 HZ	PPS输出2048Hz
ENET_PPISOFC_4096 HZ	PPS输出4096Hz
ENET_PPISOFC_8192 HZ	PPS输出8192Hz
ENET_PPISOFC_16384 HZ	PPS输出16384Hz
ENET_PPISOFC_32768 HZ	PPS输出32768Hz
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* configure the PPS output frequency as 1Hz */
```

```
enet_ptp_pps_output_frequency_config(ENET0, ENET_PPISOFC_1HZ);
```

函数 enet_ptp_start

函数The enet_ptp_start描述见下表：

表 3-542. 函数 enet_ptp_start

函数名称	enet_ptp_start
函数原型	void enet_ptp_start(uint32_t enet_periph, int32_t updatemethod, uint32_t init_sec, uint32_t init_subsec, uint32_t carry_cfg, uint32_t accuracy_cfg);
功能描述	配置和启动PTP时间戳计数器
先决条件	-
被调用函数	-
输入参数{in}	

enet_periph	以太网外设
<i>ENET0</i>	以太网0
<i>ENET1</i>	以太网1
输入参数{in}	
updatemethod	更新方式
<i>ENET_PTP_FINEMODE</i>	精细校正
<i>ENET_PTP_COARSEMODE</i>	粗校正
输入参数{in}	
init_sec	初始化系统时间秒值
输入参数{in}	
init_subsec	初始化系统时间亚秒值
输入参数{in}	
carry_cfg	增加到时间戳加数寄存器的值（使用精细校正）
输入参数{in}	
accuracy_cfg	系统时间增加的亚秒值
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* gconfigure and start PTP timestamp counter*/
```

```
enet_ptp_start(ENET0, ENET_PTP_FINEMODE, 10, 10, 10, 10);
```

函数 **enet_ptp_finecorrection_adjfreq**

函数 **enet_ptp_finecorrection_adjfreq** 描述见下表：

表 3-543. 函数 **enet_ptp_finecorrection_adjfreq**

函数名称	enet_ptp_finecorrection_adjfreq
函数原型	<code>void enet_ptp_finecorrection_adjfreq(uint32_t enet_periph, int32_t carry_cfg);</code>
功能描述	通过PTP时间戳加数寄存器精调系统时间
先决条件	-
被调用函数	-
输入参数{in}	
enet_periph	以太网外设
<i>ENET0</i>	以太网0
<i>ENET1</i>	以太网1
输入参数{in}	
carry_cfg	增加到PTP加数寄存器的数值

输出参数{out}	
-	-
返回值	
-	-

例如：

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

```
enet_ptp_finecorrection_adjfreq(ENET0, 10);
```

函数 enet_ptp_coarsecorrection_systime_update

函数enet_ptp_coarsecorrection_systime_update描述见下表：

表 3-544. 函数 enet_ptp_coarsecorrection_systime_update

函数名称	enet_ptp_coarsecorrection_systime_update
函数原型	void enet_ptp_coarsecorrection_systime_update(uint32_t enet_periph, enet_ptp_systime_struct *systime_struct);
功能描述	粗调系统时间
先决条件	-
被调用函数	-
输入参数{in}	
enet_periph	以太网外设
ENET0	以太网0
ENET1	以太网1
输入参数{in}	
systime_struct	初始化结构体，结构体成员参考 表3-434. 结构体enet_ptp_systime_struct
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* update system time in coarse method */
```

```
enet_ptp_systime_struct systime_struct;
```

```
enet_ptp_coarsecorrection_systime_update (ENET0, &systime_struct);
```

函数 enet_ptp_finecorrection_settime

函数enet_ptp_finecorrection_settime描述见下表：

表 3-545. 函数 enet_ptp_finecorrection_settime

函数名称	enet_ptp_finecorrection_settime
函数原形	void enet_ptp_finecorrection_settime(uint32_t enet_periph,

	enet_ptp_systime_struct * systime_struct);
功能描述	精调系统时间
先决条件	-
被调用函数	-
输入参数{in}	
enet_periph	以太网外设
ENET0	以太网0
ENET1	以太网1
输入参数{in}	
systime_struct	初始化结构体，结构体成员参考 表3-434. 结构体enet_ptp_systime_struct
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* set system time in fine method */
```

```
enet_ptp_systime_struct systime_struct;
```

```
enet_ptp_finecorrection_settime (ENET0, &systime_struct);
```

函数 enet_ptp_flag_get

函数enet_ptp_flag_get描述见下表：

表 3-546. 函数 enet_ptp_flag_get

函数名称	enet_ptp_flag_get
函数原形	FlagStatus enet_ptp_flag_get(uint32_t enet_periph, uint32_t flag);
功能描述	获取PTP标志位状态
先决条件	-
被调用函数	-
输入参数{in}	
enet_periph	以太网外设
ENET0	以太网0
ENET1	以太网1
输入参数{in}	
flag	需要检查的PTP标志位
ENET_PTP_ADDEND_UPDATE	加数寄存器更新
ENET_PTP_SYSTIME_UPDATE	时间戳更新
ENET_PTP_SYSTIME_INIT	时间戳初始化

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

例如：

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

```
FlagStatus status = enet_ptp_flag_get(ENET0, ENET_PTP_ADDEND_UPDATE);
```

函数 enet_initpara_reset

函数enet_initpara_reset描述见下表：

表 3-547. 函数 enet_initpara_reset

函数名称	enet_initpara_reset
函数原型	void enet_initpara_reset(void);
功能描述	复位 ENET initpara struct, 需在enet_initpara_config()函数前调用
先决条件	-
被调用函数	-
输入参数{in}	
-	-
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* reset the ENET initpara struct */
```

```
enet_initpara_reset();
```

3.16. EXMC

外部存储器控制器EXMC，用来访问各种片外存储器。章节[3.16.1](#)描述了EXMC的寄存器列表，章节[3.16.2](#)对EXMC库函数进行说明。

3.16.1. 外设寄存器描述

EXMC寄存器列表如下表所示：

表 3-548. EXMC 寄存器

寄存器名称	寄存器描述
EXMC_SNCTL	SRAM/NOR flash控制寄存器

寄存器名称	寄存器描述
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

3.16.2. 外设库函数说明

EXMC库函数列表如下表所示：

表 3-549. 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

库函数名称	库函数描述
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 3-550. 结构体 exmc_norsram_timing_parameter_struct

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

结构体 exmc_norsram_parameter_struct

表 3-551. 结构体 exmc_norsram_parameter_struct

成员名称	功能描述
norsram_region	选择EXMC NOR/SRAM Region
write_mode	写模式（同步模式或者异步模式）
extended_mode	使能或者禁用扩展模式
asyn_wait	使能或者禁用异步等待功能
nwait_signal	在同步突发模式中，使能或者禁用NWAIT信号
memory_write	使能或者禁用写操作
nwait_config	配置NWAIT信号
nwait_polarity	指定NWAIT的极性
burst_mode	使能或者禁用突发模式

成员名称	功能描述
databus_width	指定外部存储器数据总线宽度
memory_type	指定外部存储器的类型
address_data_mux	数据线/地址线复用是否复用
cram_page_size	指定CRAM页大小
read_write_timing	未用扩展模式时，读时序参数和写时序参数；或采用扩展模式时，读时序参数
write_timing	未用扩展模式时，写时序参数

结构体 `exmc_nand_timing_parameter_struct`

表 3-552. 结构体 `exmc_nand_timing_parameter_struct`

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

结构体 `exmc_nand_parameter_struct`

表 3-553. 结构体 `exmc_nand_parameter_struct`

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

结构体 `exmc_sdram_timing_parameter_struct`

表 3-554. 结构体 `exmc_sdram_timing_parameter_struct`

成员名称	功能描述
row_to_column_delay	行至列的延迟
row_precharge_delay	行预充电延迟
write_recovery_delay	写恢复延迟
auto_refresh_delay	自动刷新延迟
row_address_select	行地址选择延迟

成员名称	功能描述
_delay	
exit_selfrefresh_delay	退出自刷新的延迟
load_mode_register_delay	加载模式寄存器延迟

结构体 `exmc_sdram_parameter_struct`

表 3-555. 结构体 `exmc_sdram_parameter_struct`

成员名称	功能描述
sdram_device	选择EXMC SDRAM device x
pipeline_read_delay	流水线读数据延迟
burst_read_switch	突发读开关
sdclock_config	SDRAM时钟配置
write_protection	配置写保护功能
cas_latency	配置CAS延迟
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 3-556. 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 x
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 3-557. 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}	
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：

```
/* 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 3-558. 函数 exmc_norsram_struct_para_init

函数名称	exmc_norsram_struct_para_init
函数原型	void exmc_norsram_struct_para_init(exmc_norsram_parameter_struct* exmc_norsram_init_struct);
功能描述	初始化结构体exmc_norsram_parameter_struct
先决条件	-
被调用函数	-
输入参数{in}	
-	-
输出参数{out}	
exmc_norsram_init_struct	初始化结构体，结构体成员参考 表3-551. 结构体 exmc_norsram_parameter_struct
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 3-559. 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3-551. 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 3-560. 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 3-561. 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 3-562. 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 3-563. 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3-553. 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 3-564. 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3-553. 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 3-565. 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 3-566. 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 3-567. 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_DEVICE	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 3-568. 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3-555. 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 3-569. 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3-555. 结构体 exmc_sdram_parameter_struct

struct	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 3-570. 函数 `exmc_norsram_sdram_remap_config`

函数名称	<code>exmc_norsram_sdram_remap_config</code>
函数原型	<code>void exmc_norsram_sdram_remap_config(uint32_t bank_remap);</code>
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
<code>EXMC_BANK_REMAP_DEFAULT</code>	默认映射
<code>EXMC_BANK_NORPSRAM_SDRAM_SWAP</code>	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 3-571. 函数 `exmc_norsram_sdram_remap_get`

函数名称	<code>exmc_norsram_sdram_remap_get</code>
函数原型	<code>uint32_t exmc_norsram_sdram_remap_get(void);</code>
功能描述	获取EXMC NOR/SRAM 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 3-572. 函数 exmc_norsram_consecutive_clock_config

函数名称	exmc_norsram_consecutive_clock_config
函数原型	void exmc_norsram_consecutive_clock_config(uint32_t clock_mode);
功能描述	配置连续时钟
先决条件	-
被调用函数	-
输入参数{in}	
clock_mode	连续时钟模式
<i>EXMC_CLOCK_SYN_MODE</i>	EXMC_CLK只在同步模式产生
<i>EXMC_CLOCK_UNCONDITIONALLY</i>	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 3-573. 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}	
page_size	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
<code>EXMC_CRAM_PAGE_SIZE_1024_BYTES</code>	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 3-574. 函数 `exmc_nand_ecc_config`

函数名称	<code>exmc_nand_ecc_config</code>
函数原型	<code>void exmc_nand_ecc_config(ControlStatus newvalue);</code>
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 3-575. 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 3-576. 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 3-577. 函数 exmc_sdram_readsample_disable

函数名称	exmc_sdram_readsample_disable
函数原型	void exmc_sdram_readsample_enable(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 3-578. 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_	x=0...15

DELAY_CELL	
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 3-579. 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_com mand_init_struct	初始化结构体，结构体成员参考 表3-556. 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;
```

```
sdr_command_init_struct.auto_refresh_number = EXMC_SDRAM_AUTO_REFRESH_1_SDCLK;
```

```
sdr_command_init_struct.mode_register_content = 0;
```

```
exmc_sdram_command_config(&sdr_command_init_struct);
```

函数 `exmc_sdram_refresh_count_set`

函数 `exmc_sdram_refresh_count_set` 描述见下表：

表 3-580. 函数 `exmc_sdram_refresh_count_set`

函数名称	<code>exmc_sdram_refresh_count_set</code>
函数原型	<code>void exmc_sdram_refresh_count_set(uint32_t exmc_count);</code>
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}	
<code>exmc_count</code>	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 3-581. 函数 `exmc_sdram_autorefresh_number_set`

函数名称	<code>exmc_sdram_autorefresh_number_set</code>
函数原型	<code>void exmc_sdram_autorefresh_number_set(uint32_t exmc_number);</code>
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}	
<code>exmc_number</code>	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 3-582. 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
EXMC_SDRAM_DEVICE	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 3-583. 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

ce	
<i>EXMC_SDRAM_DE VICE_x</i>	x=0,1
输出参数{out}	
-	-
返回值	
uint32_t	SDRAM device _x 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 3-584. 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_NA ND</i>	NAND bank2
<i>EXMC_SDRAM_DE VICE0</i>	SDRAM device0
<i>EXMC_SDRAM_DE VICE1</i>	SDRAM device1
输入参数{in}	
flag	EXMC标志状态
<i>EXMC_NAND_FLA G_LEVEL</i>	高电平状态
<i>EXMC_NAND_FLA G_RISE</i>	上升沿状态
<i>EXMC_NAND_FLA G_FALL</i>	下降沿状态
<i>EXMC_SDRAM_FL AG_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 3-585. 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
EXMC_BANK2_NAND	NAND bank2
EXMC_SDRAM_DEVICE0	SDRAM device0
EXMC_SDRAM_DEVICE1	SDRAM device1
输入参数{in}	
flag	EXMC标志状态
EXMC_NAND_FLAG_LEVEL	高电平状态
EXMC_NAND_FLAG_RISE	上升沿状态
EXMC_NAND_FLAG_FALL	下降沿状态
EXMC_SDRAM_FLAG_REFRESH	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 3-586. 函数 `exmc_interrupt_enable`

函数名称	<code>exmc_interrupt_enable</code>
函数原型	<code>void exmc_interrupt_enable(uint32_t exmc_bank, uint32_t interrupt);</code>
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
<code>EXMC_BANK2_NAND</code>	NAND bank2
<code>EXMC_SDRAM_DEVICE0</code>	SDRAM device0
<code>EXMC_SDRAM_DEVICE1</code>	SDRAM device1
输入参数{in}	
interrupt	EXMC中断
<code>EXMC_NAND_INT_FLAG_LEVEL</code>	高电平中断
<code>EXMC_NAND_INT_FLAG_RISE</code>	上升沿中断
<code>EXMC_NAND_INT_FLAG_FALL</code>	下降沿中断
<code>EXMC_SDRAM_INT_FLAG_REFRESH</code>	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 3-587. 函数 `exmc_interrupt_disable`

函数名称	<code>exmc_interrupt_disable</code>
函数原型	<code>void exmc_interrupt_disable(uint32_t exmc_bank, uint32_t interrupt);</code>
功能描述	禁用EXMC中断
先决条件	-
被调用函数	-
输入参数{in}	
<code>exmc_bank</code>	EXMC外设
<code>EXMC_BANK2_NAND</code>	NAND bank2
<code>EXMC_SDRAM_DEVICE0</code>	SDRAM device0
<code>EXMC_SDRAM_DEVICE1</code>	SDRAM device1
输入参数{in}	
<code>interrupt</code>	EXMC中断
<code>EXMC_NAND_INT_FLAG_LEVEL</code>	高电平中断
<code>EXMC_NAND_INT_FLAG_RISE</code>	上升沿中断
<code>EXMC_NAND_INT_FLAG_FALL</code>	下降沿中断
<code>EXMC_SDRAM_INTERRUPT_FLAG_REFRESH</code>	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 3-588. 函数 `exmc_interrupt_flag_get`

函数名称	<code>exmc_interrupt_flag_get</code>
函数原型	<code>FlagStatus exmc_interrupt_flag_get(uint32_t exmc_bank, uint32_t interrupt);</code>
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>EXMC_SDRAM_FLAG_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 3-589. 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

VICE0	
EXMC_SDRAM_DE VICE1	SDRAM device1
输入参数{in}	
interrupt	EXMC中断状态
EXMC_NAND_FL A_G_LEVEL	高电平中断
EXMC_NAND_FL A_G_RISE	上升沿中断
EXMC_NAND_FL A_G_FALL	下降沿中断
EXMC_SDRAM_FL AG_REFRESH	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);
```

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

3.17.1. 外设寄存器说明

EXTI寄存器列表如下表所示:

表 3-590. 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

寄存器名称	寄存器描述
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

3.17.2. 外设库函数说明

EXTI库函数列表如下表所示：

表 3-591. EXTI 库函数

库函数名称	库函数描述
exti_deinit	复位EXTI
exti_init	初始化EXTI线x
exti_interrupt_enable	EXTI线x中断使能
exti_interrupt_disable	EXTI线x中断禁能
exti_event_enable	EXTI线x事件使能
exti_event_disable	EXTI线x事件禁能
exti_software_interrupt_enable	EXTI线x软件中断使能
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 3-592. 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

枚举名称	枚举描述
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
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 3-593. 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 3-594. 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 3-595. 函数 exti_deinit

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

例如：

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

```
exti_deinit();
```

函数 exti_init

函数exti_init描述见下表：

表 3-596. 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3-592. 枚举类型exti_line_enum
输入参数{in}	
mode	EXTI模式，参考 表3-593. 枚举类型exti_mode_enum
输入参数{in}	
trig_type	触发类型，参考 表3-594. 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 3-597. 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3-592. 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 3-598. 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3-592. 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 3-599. 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3-592. 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 3-600. 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3-592. 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 3-601. 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3-592. 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 3-602. 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3-592. 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 3-603. 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3-592. 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 3-604. 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3-592. 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 3-605. 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3-592. 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 3-606. 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3-592. 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);
```

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

3.18.1. 外设寄存器说明

FAC寄存器列表如下表所示：

表 3-607. 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
FAC_WDATA	FAC写数据寄存器
FAC_RDATA	FAC读数据寄存器

3.18.2. 外设库函数说明

FAC库函数列表如下表所示：

表 3-608. 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

库函数名称	库函数描述
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 3-609. 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 3-610. 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

成员名称	功能描述
*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 3-611. 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 3-612. 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 3-613. 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 3-614. 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 3-615. 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 3-616. 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 3-617. 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 3-618. 函数 fac_float_buffer_preload

函数名称	fac_float_buffer_preload
函数原型	void fac_float_buffer_preload(fac_float_data_preload_struct* init_struct);
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}	
init_struct	FAC初始化参数结构体，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：

```
/* 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 3-619. 函数 fac_fixed_data_preload

函数名称	fac_fixed_data_preload
函数原型	void fac_fixed_data_preload(uint8_t size, int16_t *data);
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}	
size	数据长度
输入参数{in}	
*data	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 3-620. 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 3-621. 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 3-622. 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

<code>FAC_CP_ENABLE</code>	使能限幅
<code>FAC_CP_DISABLE</code>	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 3-623. 函数 `fac_float_enable`

函数名称	<code>fac_float_enable</code>
函数原型	<code>void fac_float_enable(void);</code>
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 3-624. 函数 `fac_float_disable`

函数名称	<code>fac_float_disable</code>
函数原型	<code>void fac_float_disable(void);</code>
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 3-625. 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 3-626. 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 3-627. 函数 fac_x0_config

函数名称	fac_x0_config
函数原型	void fac_x0_config(uint32_t watermark, uint8_t baseaddr, uint8_t bufsize);
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
FAC_THRESHOLD _x	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 3-628. 函数 fac_x1_config

函数名称	fac_x1_config
函数原型	void fac_x1_config(uint8_t baseaddr, uint8_t bufsize);
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 3-629. 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 3-630. 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 3-631. 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 3-632. 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 3-633. 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 3-634. 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 3-635. 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 3-636. 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 3-637. 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 3-638. 函数 fac_interrupt_enable

函数名称	fac_interrupt_enable
函数原型	void fac_interrupt_enable(uint32_t interrupt);
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
FAC_CTL_RIE	FAC读缓冲区中断
FAC_CTL_WIE	FAC写缓冲区中断
FAC_CTL_OFEIE	FAC上溢错误中断
FAC_CTL_UFEIE	FAC下溢错误中断
FAC_CTL_STEIE	FAC饱和错误中断
FAC_CTL_GSTEIE	FAC增益饱和和错误中断
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 3-639. 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 3-640. 函数 `fac_interrupt_flag_get`

函数名称	<code>fac_interrupt_disable</code>
函数原型	<code>FlagStatus fac_interrupt_flag_get(uint8_t interrupt);</code>
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
<code>FAC_INT_FLAG_YBEF</code>	Y缓冲区空中断标志
<code>FAC_INT_FLAG_X0BFF</code>	X0缓冲区满中断标志
<code>FAC_INT_FLAG_OFEF</code>	上溢错误中断标志
<code>FAC_INT_FLAG_UFEF</code>	下溢错误中断标志

<i>FAC_INT_FLAG_STEF</i>	饱和错误中断标志
<i>FAC_INT_FLAG_GSTEF</i>	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 3-641. 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
<i>FAC_FLAG_YBEF</i>	Y缓冲区空标志
<i>FAC_FLAG_X0BFF</i>	X0缓冲区满标志
<i>FAC_FLAG_OFEF</i>	上溢错误标志
<i>FAC_FLAG_UFEF</i>	下溢错误标志
<i>FAC_FLAG_STEF</i>	饱和错误标志
<i>FAC_FLAG_GSTEF</i>	增益饱和和错误标志
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);
```

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

3.19.1. 外设寄存器说明

FMC寄存器列表如下表所示：

表 3-642. 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
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	AES初始向量生效寄存器x (x = 0...2)

寄存器名称	寄存器描述
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)

3.19.2. 外设库函数说明

FMC库函数列表如下表所示：

表 3-643. 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
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

库函数名称	库函数描述
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
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 3-644. 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_PGERR	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 3-645. 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 3-646. 枚举类型 `fmc_interrupt_flag_enum`

枚举名称	枚举描述
FMC_INT_FLAG_E ND	闪存操作结束中断标志
FMC_INT_FLAG_W PERR	闪存擦除/编程保护错误中断标志
FMC_INT_FLAG_P GSERR	闪存编程顺序错误中断标志
FMC_INT_FLAG_R PERR	闪存读保护错误中断标志
FMC_INT_FLAG_R SERR	闪存读安全错误中断错误
FMC_INT_FLAG_E CCCOR	检测并纠正单个位错误中断标志
FMC_INT_FLAG_E CCDET	检测到双位错误中断标志
FMC_INT_FLAG_O BMERR	选项字节修改错误中断标志

枚举类型 **fmc_interrupt_enum**表 3-647. 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_PGSERR	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_ECCCOR	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_OBMER	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 3-648. 函数 **fmc_unlock**

函数名称	fmc_unlock
函数原型	void fmc_unlock(void);
功能描述	解锁FMC_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 3-649. 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 3-650. 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3-644. 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 3-651. 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3-644. 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 3-652. 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3-644. 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 3-653. 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3-644. 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 3-654. 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3-644. 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 3-655. 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3-644. 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 3-656. 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3-644. 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 3-657. 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 3-658. 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 3-659. 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3-644. 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 3-660. 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3-644. 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 3-661. 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3-644. 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 3-662. 函数 **ob_secure_access_mode_disable**

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3-644. 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 3-663. 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3-644. 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 3-664. 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3-644. 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 3-665. 函数 `ob_low_power_config`

函数名称	<code>ob_low_power_config</code>
函数原型	<code>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);</code>
功能描述	配置低功耗相关选项字节
先决条件	<code>ob_unlock</code>
被调用函数	-
输入参数{in}	
<code>ob_fwdgt</code>	看门狗选项
<code>OB_FWDGT_SW</code>	软件控制开门狗
<code>OB_FWDGT_HW</code>	硬件控制开门狗
输入参数{in}	
<code>ob_deepsleep</code>	深度睡眠模式复位选项
<code>OB_DEEPSLEEP_NRS</code> <code>T</code>	进入深度睡眠模式时不产生复位
<code>OB_DEEPSLEEP_RST</code>	进入深度睡眠模式时产生复位
输入参数{in}	
<code>ob_stdby</code>	待机模式复位选项
<code>OB_STDBY_NRS</code>	进入待机模式时不产生复位
<code>OB_STDBY_RST</code>	进入待机模式时产生复位
输入参数{in}	
<code>ob_fwdg_suspend_deepsleep</code>	深度睡眠模式下独立看门狗暂停选项
<code>OB_DPSLP_FWDGT_SUSPEND</code>	在深度睡眠状态下暂停独立看门狗
<code>OB_DPSLP_FWDGT_RUN</code>	在深度睡眠状态下运行独立看门狗
输入参数{in}	
<code>ob_fwdg_suspend_standby</code>	待机模式下独立看门狗暂停选项配置位
<code>OB_STDBY_FWDGT_SUSPEND</code>	在待机状态下暂停独立看门狗
<code>OB_STDBY_FWDGT_RUN</code>	在待机状态下运行独立看门狗
输出参数{out}	
-	-
返回值	
<code>fmc_state_enum</code>	FMC状态，参考 表3-644. 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 3-666. 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_itcmecc, uint32_t ob_dtcmecc, uint32_t ob_dtcmecc);
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_itcmecc	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_dtcmecc	DTCM0的ECC功能使能选项
OB_DTCMECCEN_DISABLE	失能DTCM0的ECC功能
OB_DTCMECCEN_ENABLE	使能DTCM0的ECC功能
输入参数{in}	
ob_dtcmecc	DTCM1的ECC功能使能选项
OB_DTCMECCEN_DISABLE	失能DTCM1的ECC功能
OB_DTCMECCEN_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3-644. 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_DTC
M0ECCEN_DISABLE, OB_DTCM1ECCEN_DISABLE);

```

函数 ob_iospeed_optimize_config

函数ob_iospeed_optimize_config描述见下表

表 3-667. 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3-644. 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 3-668. 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_RAM_0KB	0-KB DTCM
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3-644. 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 3-669. 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3-644. 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 3-670. 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3-644. 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 3-671. 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
<i>OB_DCRP_AREA_ER_ASE_DISABLE</i>	DCRP区域擦除失能
<i>OB_DCRP_AREA_ER_ASE_ENABLE</i>	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3-644. 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 3-672. 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
OB_SCR_AREA_ERASE_ENABLE	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3-644. 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 3-673. 函数 `ob_write_protection_enable`

函数名称	<code>ob_write_protection_enable</code>
函数原型	<code>fmc_state_enum ob_write_protection_enable(uint32_t ob_wp);</code>
功能描述	使能扇区编程/擦除保护
先决条件	<code>ob_unlock</code>
被调用函数	-
输入参数{in}	
ob_wp	使能编程/擦除保护的扇区
<code>OB_WP_0</code>	编程/擦除保护扇区0 ~ 扇区15
<code>OB_WP_1</code>	编程/擦除保护扇区16 ~ 扇区31
<code>OB_WP_2</code>	编程/擦除保护扇区32 ~ 扇区47
<code>OB_WP_3</code>	编程/擦除保护扇区48 ~ 扇区63
<code>OB_WP_4</code>	编程/擦除保护扇区64 ~ 扇区79
<code>OB_WP_5</code>	编程/擦除保护扇区80 ~ 扇区95
<code>OB_WP_6</code>	编程/擦除保护扇区96 ~ 扇区111
<code>OB_WP_7</code>	编程/擦除保护扇区112 ~ 扇区127
<code>OB_WP_8</code>	编程/擦除保护扇区128 ~ 扇区143
<code>OB_WP_9</code>	编程/擦除保护扇区144 ~ 扇区159
<code>OB_WP_10</code>	编程/擦除保护扇区160 ~ 扇区175
<code>OB_WP_11</code>	编程/擦除保护扇区176 ~ 扇区191
<code>OB_WP_12</code>	编程/擦除保护扇区192 ~ 扇区207
<code>OB_WP_13</code>	编程/擦除保护扇区208 ~ 扇区223
<code>OB_WP_14</code>	编程/擦除保护扇区224 ~ 扇区239
<code>OB_WP_15</code>	编程/擦除保护扇区240 ~ 扇区255
<code>OB_WP_16</code>	编程/擦除保护扇区256 ~ 扇区383
<code>OB_WP_17</code>	编程/擦除保护扇区384 ~ 扇区511
<code>OB_WP_18</code>	编程/擦除保护扇区512 ~ 扇区639
<code>OB_WP_19</code>	编程/擦除保护扇区640 ~ 扇区767
<code>OB_WP_20</code>	编程/擦除保护扇区768 ~ 扇区895
<code>OB_WP_21</code>	编程/擦除保护扇区896 ~ 扇区959
<code>OB_WP_ALL</code>	all sectors
输出参数{out}	
-	-
返回值	
fmc_state_enum	FMC状态，参考 表3-644. 枚举类型 <code>fmc_state_enum</code>

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 3-674. 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_enumFMC状态，参考 [表3-644. 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 3-675. 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 3-676. 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 3-677. 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
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

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 3-678. 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}	
fwdg_suspend_standby	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 3-679. 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 *itcmecc_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}	
itcmecc_option	CPU ITCM ECC功能选项
OB_ITCMECCEN_DISABLE	失能ITCM的ECC功能
OB_ITCMECCEN_ENABLE	使能ITCM的ECC功能
输出参数{out}	
dtcm0ecc_option	CPU DTCM0 ECC功能选项
OB_DTCM0ECCEN_DISABLE	失能DTCM0的ECC功能
OB_DTCM0ECCEN_ENABLE	使能DTCM0的ECC功能
输出参数{out}	
dtcm1ecc_option	CPU DTCM1 ECC功能选项
OB_DTCM1ECCEN_DISABLE	失能DTCM1的ECC功能
OB_DTCM1ECCEN_ENABLE	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 3-680. 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 3-681. 函数 **ob_tcm_shared_ram_size_get**

函数名称	ob_tcm_shared_ram_size_get
函数原型	void ob_tcm_shared_ram_size_get(uint32_t *itcm_shared_ram_kb_size, uint32_t *dcm_shared_ram_kb_size);
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}	
itcm_shared_ram_kb_size	ITCM shared RAM size in KB unit
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

输出参数{out}	
dtcm_shared_ram_kb_size	DTCM shared RAM size in KB unit
<i>OB_DTCM_SHARED_RAM_0KB</i>	0-KB DTCM
<i>OB_DTCM_SHARED_RAM_64KB</i>	64-KB DTCM
<i>OB_DTCM_SHARED_RAM_128KB</i>	128-KB DTCM
<i>OB_DTCM_SHARED_RAM_256KB</i>	256-KB DTCM
<i>OB_DTCM_SHARED_RAM_512KB</i>	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 3-682. 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 3-683. 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 3-684. 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 3-685. 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 3-686. 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 - 0x3FFFFFFF)

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 3-687. 函数 fmc_no_rtdec_config

函数名称	fmc_no_rtdec_config
函数原型	fmc_state_enum fmc_no_rtdec_config(uint32_t nodec_area_start, uint32_t nodec_area_end);
功能描述	配置NO-RTDEC区域
先决条件	fmc_unlock
被调用函数	-
输入参数{in}	
nodec_area_start	NO-RTDEC区域起始地址
输入参数{in}	
nodec_area_end	NO-RTDEC区域结束地址
输出参数{out}	
-	-
返回值	
fmc_state_enum	FMC状态, 参考 表3-644. 枚举类型fmc_state_enum

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 3-688. 函数 fmc_aes_iv_config

函数名称	fmc_aes_iv_config
函数原型	fmc_state_enum fmc_aes_iv_config(uint32_t *aes_iv);
功能描述	配置AES初始向量的高96位
先决条件	fmc_unlock
被调用函数	-
输入参数{in}	
aes_iv	AES初始向量的高96位
输出参数{out}	
-	-
返回值	
fmc_state_enum	FMC状态，参考 表3-644. 枚举类型fmc_state_enum

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 3-689. 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 3-690. 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 3-691. 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 3-692. 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 3-693. 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3-645. 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 3-694. 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3-645. 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 3-695. 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3-647. 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 3-696. 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3-647. 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 3-697. 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3-646. 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 3-698. 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3-646. 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);
```

3.20. 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 [3.20.1](#) 描述了FWDGT的寄存器列表，章节 [3.20.2](#) 对FWDGT库函数进行说明。

3.20.1. 外设寄存器说明

FWDGT寄存器列表如下表所示：

表 3-699. 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	重载寄存器
FWDGT_STAT	状态寄存器
FWDGT_WND	窗口寄存器

3.20.2. 外设库函数说明

FWDGT库函数列表如下表所示：

表 3-700. FWDGT 库函数

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

函数 fwdgt_write_enable

函数fwdgt_write_enable描述见下表：

表 3-701. 函数 fwdgt_write_enable

函数名称	fwdgt_write_enable
函数原型	void fwdgt_write_enable(void);
功能描述	使能对寄存器FWDGT_PSC, 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 3-702. 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 3-703. 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 3-704. 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 3-705. 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装载值
先决条件	-
被调用函数	-
输入参数{in}	
reload_value	重装载值, 数值范围为0x0000 - 0x0FFF
输出参数{out}	
-	-
返回值	
ErrStatus	ERROR / SUCCESS

例如:

```
/* set FWDGT reload value to 0xFFFF */
```

```
ErrStatus flag;
```

```
flag = fwdgt_reloadr_value_config(0xFFFF);
```

函数 fwdgt_window_value_reload

函数fwdgt_window_value_config描述见下表:

表 3-706. 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 3-707. 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 3-708. 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
FWDGT_PSC_DIV32	FWDGT预分频值设为32
FWDGT_PSC_DIV64	FWDGT预分频值设为64
FWDGT_PSC_DIV128	FWDGT预分频值设为128
FWDGT_PSC_DIV256	FWDGT预分频值设为256
输出参数{out}	
-	-
返回值	
ErrStatus	ERROR 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 3-709. 函数 fwdgt_flag_get

函数名称	fwdgt_flag_get
函数原型	FlagStatus fwdgt_flag_get(uint16_t flag);
功能描述	获取FWDGT标志位状态
先决条件	-
被调用函数	-
输入参数{in}	
flag	需要获取状态的FWDGT标志位

<i>FWDGT_FLAG_PUD</i>	预分频值更新进行中
<i>FWDGT_FLAG_RU</i> <i>D</i>	重装载值更新进行中
<i>FWDGT_FLAG_WU</i> <i>D</i>	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);
```

3.21. GPIO

GPIO用来实现各片上设备的逻辑输入/输出功能。章节[3.21.1](#)描述了GPIO的寄存器列表，章节[3.21.2](#)对GPIO库函数进行说明。

3.21.1. 外设寄存器说明

GPIO寄存器列表如下表所示:

表 3-710. 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

3.21.2. 外设库函数说明

GPIO库函数列表如下表所示：

表 3-711. 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 3-712. 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 3-713. 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_ANALOG	模拟模式
输入参数{in}	
pull_up_down	GPIO引脚上拉下拉电阻设置
GPIO_PUPD_NONE	悬空模式, 无上拉和下拉
GPIO_PUPD_PULLUP	带上拉电阻
GPIO_PUPD_PULLDOWN	带下拉电阻
输入参数{in}	
pin	GPIO pin
GPIO_PIN_x	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 3-714. 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 MHZ	最大输出速度为12MHz
GPIO_OSPEED_60 MHZ	最大输出速度为60MHz
GPIO_OSPEED_85 MHZ	最大输出速度为85MHz
GPIO_OSPEED_100_220MHZ	最大输出速度为100/220MHz
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：

```
/* 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 3-715. 函数 gpio_bit_set

函数名称	gpio_bit_set
函数原型	void gpio_bit_set(uint32_t gpio_periph, uint32_t pin);
功能描述	置位引脚值
先决条件	-
被调用函数	-
输入参数{in}	
gpio_periph	GPIO端口
GPIOx	端口选择 (x = A,B,C,D,E,F,G,H,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:

```
/* 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 3-716. 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

<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:

```
/* 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 3-717. 函数 gpio_bit_write

函数名称	gpio_bit_write
函数原型	void gpio_bit_write(uint32_t gpio_periph,uint32_t pin, bit_status bit_value);
功能描述	将特定的值写入引脚
先决条件	-
被调用函数	-
输入参数{in}	
gpio_periph	GPIO端口
<i>GPIOx</i>	端口选择 (x = A,B,C,D,E,F,G,H,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 3-718. 函数 **gpio_port_write**

函数名称	gpio_port_write
函数原型	void gpio_port_write(uint32_t gpio_periph, uint16_t data);
功能描述	将特定的值写入端口
先决条件	-
被调用函数	-
输入参数{in}	
gpio_periph	GPIO端口
GPIOx	端口选择 (x = A,B,C,D,E,F,G,H,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 3-719. 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

<i>GPIO_IFTYPE_SYNC</i>	同步类型
<i>GPIO_IFTYPE_3_SAMPLES</i>	过滤（3个采样点）
<i>GPIO_IFTYPE_6_SAMPLES</i>	过滤（6个采样点）
<i>GPIO_IFTYPE_ASYNC</i>	异步类型
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 GPIO input filter */
```

```
gpio_input_filter_set(GPIOA, GPIO_IPERIOD(100), GPIO_IFTYPE_SYNC);
```

函数 gpio_input_bit_get

函数gpio_input_bit_get描述见下表：

表 3-720. 函数 gpio_input_bit_get

函数名称	gpio_input_bit_get
函数原型	FlagStatus gpio_input_bit_get(uint32_t gpio_periph, uint32_t pin);
功能描述	获取引脚的输入值
先决条件	-
被调用函数	-
输入参数{in}	
gpio_periph	GPIO端口
<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	
FlagStatus	SET / RESET

例如:

```
/* get status of PA0*/
```

```
FlagStatus bit_state;
```

```
bit_state = gpio_input_bit_get(GPIOA, GPIO_PIN_0);
```

函数 gpio_input_port_get

函数gpio_input_port_get描述见下表:

表 3-721. 函数 gpio_input_port_get

函数名称	gpio_input_port_get
函数原型	uint16_t gpio_input_port_get(uint32_t gpio_periph);
功能描述	获取端口的输入值
先决条件	-
被调用函数	-
输入参数{in}	
gpio_periph	GPIO端口
GPIOx	端口选择 (x = A,B,C,D,E,F,G,H,J,K)
输出参数{out}	
-	-
返回值	
uint16_t	0x0000-0xFFFF

例如:

```
/* get input value of Port A */
```

```
uint16_t port_state;
```

```
port_state = gpio_input_port_get(GPIOA);
```

函数 gpio_output_bit_get

函数gpio_output_bit_get描述见下表:

表 3-722. 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
<i>GPIO_PIN_x</i>	引脚选择 (x = 0..15)
<i>GPIO_PIN_ALL</i>	所有引脚
输出参数{out}	
-	-
返回值	
FlagStatus	SET / RESET

例如:

```
/* get 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 3-723. 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 3-724. 函数 gpio_af_set

函数名称	gpio_af_set
-------------	-------------

函数原型	void gpio_af_set(uint32_t gpio_periph, uint32_t alt_func_num, uint32_t pin);
功能描述	设置GPIO的备用功能
先决条件	-
被调用函数	-
输入参数{in}	
gpio_periph	GPIO端口
GPIOx	端口选择 (x = A,B,C,D,E,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
GPIO_AF_0	SYSTEM, TIMER40, TIMER41, TIMER42, TIMER43, TIMER44
GPIO_AF_1	TIMER0, TIMER1, TIMER15, TIMER16, EXMC, SAI1, SAI2
GPIO_AF_2	TIMER2, TIMER3, TIMER4, TIMER7, TIMER14, TLI, CAN2, SAI0, EXMC
GPIO_AF_3	TIMER7, TIMER9, EDOUT, EXMC, TLI, HPDF, OSPIM
GPIO_AF_4	TIMER14, TIMER30, TIMER31, I2C0, I2C1, I2C2, I2C3, USART0, HPDF, OSPIM, TLI
GPIO_AF_5	SPI0, SPI1, SPI2, SPI3, SPI4, SPI5, CAN2
GPIO_AF_6	UART3, SPI2, I2C3, HPDF, SAI0, ETH1, EDOUT, OSPIM
GPIO_AF_7	USART0, USART1, USART2, USART5, UART6, TIMER40, TIMER41, TIMER42, TIMER43, SPI1, SPI2, SPI5, SDIO0, USBHS1
GPIO_AF_8	UART3, UART4, UART7, SPI5, SDIO0, RSPDIF, TIMER44, USBHS1, SAI1, SAI2
GPIO_AF_9	SDIO1, TRGSEL, CAN0, CAN1, TLI, OPSIM, EXMC, RSPDIF, SAI2
GPIO_AF_10	SAI1, SAI2, OTG0, SDIO1, CMP, USBHS0, OPSIM, EXMC
GPIO_AF_11	ETH0, MDIO, CMP, UART6, EXMC, HPDF, I2C3, TLI, SDIO1, OPSIM
GPIO_AF_12	TIMER0, MDIOS, SDIO0, EXMC, OPSIM, CMP, TLI, USBHS1
GPIO_AF_13	TRGSEL, DCI, COMP0, CMP, TIMER22
GPIO_AF_14	TLI, UART4, TIMER23
GPIO_AF_15	EVENTOUT
输入参数{in}	
pin	GPIO引脚
GPIO_PIN_x	引脚选择 (x = 0..15)
GPIO_PIN_ALL	所有引脚
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/*set PA0 alternate function 0*/
```

```
gpio_af_set(GPIOA, GPIO_AF_0, GPIO_PIN_0);
```

函数 **gpio_pin_lock**

函数gpio_pin_lock描述见下表:

表 3-725. 函数 **gpio_pin_lock**

函数名称	gpio_pin_lock
函数原型	void gpio_pin_lock(uint32_t gpio_periph, uint32_t pin);
功能描述	相应的引脚配置被锁定
先决条件	-
被调用函数	-
输入参数{in}	
gpio_periph	GPIO端口
GPIOx	端口选择 (x = A,B,C,D,E,F,G,H,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:

```
/* 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 3-726. 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 3-727. 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);
```

3.22. HAU

哈希处理器应用于信息安全。支持应用于多种场合的安全哈希算法（SHA-1，SHA-224和SHA-256），消息摘要算法（MD5）和哈希运算消息认证码（HMAC）。HAU寄存器列举在章节[3.22.1](#)，HAU固件库函数介绍在章节[3.22.2](#)。

3.22.1. 外设寄存器说明

HAU寄存器列表如下表所示：

表 3-728. HAU 寄存器

寄存器名称	寄存器描述
HAU_CTL	控制寄存器
HAU_DI	数据输入寄存器
HAU_CFG	配置寄存器
HAU_DO0	数据输出寄存器0
HAU_DO1	数据输出寄存器1
HAU_DO2	数据输出寄存器2
HAU_DO3	数据输出寄存器3
HAU_DO4	数据输出寄存器4
HAU_DO5	数据输出寄存器5
HAU_DO6	数据输出寄存器6
HAU_DO7	数据输出寄存器7
HAU_INTEN	中断使能寄存器
HAU_STAT	状态寄存器
HAU_CTXSx (x = 0..53)	上下文交换寄存器

3.22.2. 外设库函数说明

HAU库函数列表如下表所示：

表 3-729. HAU 库函数

库函数名称	库函数描述
hau_deinit	复位HAU外设
hau_init	初始化HAU外设参数
hau_init_struct_para_init	初始化结构体hau_initpara
hau_reset	复位HAU内核
hau_last_word_validbits_num_config	配置消息最新字有效位数
hau_data_write	写数据到IN FIFO
hau_infifo_words_num_get	返回已经写入IN FIFO的字数目
hau_digest_read	读消息摘要结果
hau_digest_calculation_enable	使能摘要计算
hau_multiple_single_dma_config	配置使用多DMA或单DMA，从未确定是否在DMA传输结束后计算摘要
hau_dma_enable	使能HAU DMA接口
hau_dma_disable	除能HAU DMA接口
hau_context_struct_para_init	初始化上下文结构体
hau_context_save	保存HAU外设上下文
hau_context_restore	恢复HAU外设上下文
hau_hash_sha_1	在HASH模式下使用SHA1计算摘要
hau_hmac_sha_1	在HMAC模式下使用SHA1计算摘要
hau_hash_sha_224	在HASH模式下使用SHA224计算摘要

库函数名称	库函数描述
hau_hmac_sha_224	在HMAC模式下使用SHA224计算摘要
hau_hash_sha_256	在HASH模式下使用SHA256计算摘要
hau_hmac_sha_256	在HMAC模式下使用SHA256计算摘要
hau_hash_md5	在HASH模式下使用MD5计算摘要
hau_hmac_md5	在HMAC模式下使用MD5计算摘要
hau_flag_get	获取HAU标志状态
hau_flag_clear	清除HAU标志状态
hau_interrupt_enable	使能HAU中断
hau_interrupt_disable	除能HAU中断
hau_interrupt_flag_get	获取HAU中断标志状态
hau_interrupt_flag_clear	清除HAU中断标志状态

结构体 hau_init_parameter_struct

表 3-730. 结构体 hau_init_parameter_struct

成员名称	功能描述
algo	算法选择: HAU_ALGO_SHA1, HAU_ALGO_SHA224, HAU_ALGO_SHA256, HAU_ALGO_MD5
mode	HAU模式选择: HAU_MODE_HASH, HAU_MODE_HMAC
datatype	数据类型模式: HAU_SWAPPING_32BIT, HAU_SWAPPING_16BIT, HAU_SWAPPING_8BIT, HAU_SWAPPING_1BIT
keytype	密钥长度模式: HAU_KEY_SHORTER_64, HAU_KEY_LONGGER_64

结构体 hau_digest_parameter_struct

表 3-731. 结构体 hau_digest_parameter_struct

成员名称	功能描述
out[x] (x = 0...7)	消息摘要结果0-7

结构体 hau_context_parameter_struct

表 3-732. 结构体 hau_context_parameter_struct

成员名称	功能描述
hau_ctl_bak	HAU_CTL寄存器的备份
hau_cfg_bak	HAU_CFG寄存器的备份
hau_inten_bak	HAU_INTEN寄存器的备份
hau_ctxs_bak[54]	HAU_CTXSx寄存器的备份

函数 hau_deinit

函数hau_deinit描述见下表:

表 3-733. 函数 hau_deinit

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

例如：

```
/* reset the HAU peripheral */
```

```
hau_deinit();
```

函数 hau_init

函数hau_init描述见下表：

表 3-734. 函数 hau_init

函数名称	hau_init
函数原形	void hau_init(hau_init_parameter_struct* initpara);
功能描述	初始化HAU外设参数
先决条件	-
被调用函数	-
输入参数{in}	
initpara	参考结构体 表3-730. 结构体hau_init_parameter_struct
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* initialize the HAU peripheral parameters */
```

```
hau_init_parameter_struct hau_initpara;
```

```
...
```

```
hau_initpara.algo = algo;
```

```
hau_initpara.mode = HAU_MODE_HMAC;
```

```
hau_initpara.datatype = HAU_SWAPPING_8BIT;
```

```
if(key_len > 64U){  
    hau_initpara.keytype = HAU_KEY_LONGGER_64;  
}  
else{  
    hau_initpara.keytype = HAU_KEY_SHORTER_64;  
}  
  
hau_init(&hau_initpara);
```

函数 hau_init_struct_para_init

函数hau_init_struct_para_init描述见下表:

表 3-735. 函数 hau_init_struct_para_init

函数名称	hau_init_struct_para_init
函数原形	void hau_init_struct_para_init(hau_init_parameter_struct* initpara)
功能描述	初始化结构体hau_initpara
先决条件	-
被调用函数	-
输入参数{in}	
-	-
输出参数{out}	
initpara	参考结构体 表3-730. 结构体hau_init_parameter_struct
返回值	
-	-

例如:

```
/* initialize the HAU peripheral parameters */  
  
hau_init_parameter_struct hau_initpara;  
  
hau_init_struct_para_init(&hau_initpara);
```

函数 hau_reset

函数hau_reset描述见下表:

表 3-736. 函数 hau_reset

函数名称	hau_reset
函数原形	void hau_reset(void);
功能描述	复位HAU内核
先决条件	-
被调用函数	-
输入参数{in}	
-	-

输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* reset the HAU processor core */
hau_reset();
```

函数 hau_last_word_validbits_num_config

函数hau_last_word_validbits_num_config描述见下表：

表 3-737. 函数 hau_last_word_validbits_num_config

函数名称	hau_last_word_validbits_num_config
函数原形	void hau_last_word_validbits_num_config(uint32_t valid_num);
功能描述	配置消息最新字有效位数
先决条件	-
被调用函数	-
输入参数{in}	
valid_num	消息最新字有效位数(0x00 – 0x1F)
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* configure the number of valid bits in last word of the message */
hau_last_word_validbits_num_config(0x10);
```

函数 hau_data_write

函数hau_data_write描述见下表：

表 3-738. 函数 hau_data_write

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

-	-
返回值	
-	-

例如：

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

```
hau_data_write(0x10);
```

函数 `hau_infifo_words_num_get`

函数 `hau_infifo_words_num_get` 描述见下表：

表 3-739. 函数 `hau_infifo_words_num_get`

函数名称	<code>hau_infifo_words_num_get</code>
函数原形	<code>uint32_t hau_infifo_words_num_get(void);</code>
功能描述	返回已经写入IN FIFO的字数目
先决条件	-
被调用函数	-
输入参数{in}	
-	-
输出参数{out}	
-	-
返回值	
<code>uint32_t</code>	0x0 – 0xFFFFFFFF

例如：

```
/* return the number of words already written into the IN FIFO */
```

```
uint32_t num = hau_infifo_words_num_get();
```

函数 `hau_digest_read`

函数 `hau_digest_read` 描述见下表：

表 3-740. 函数 `hau_digest_read`

函数名称	<code>hau_digest_read</code>
函数原形	<code>void hau_digest_read(hau_digest_parameter_struct* digestpara);</code>
功能描述	读消息摘要结果
先决条件	-
被调用函数	-
输入参数{in}	
-	-
输出参数{out}	
<code>digestpara</code>	参考结构体 表3-731. 结构体 <code>hau_digest_parameter_struct</code>

返回值	
-	-

例如：

```
/* read the message digest result */
hau_digest_parameter_struct digestpara;
hau_digest_read(&digestpara);
```

函数 hau_digest_calculation_enable

函数hau_digest_calculation_enable描述见下表：

表 3-741. 函数 hau_digest_calculation_enable

函数名称	hau_digest_calculation_enable
函数原形	void hau_digest_calculation_enable(void);
功能描述	使能摘要计算
先决条件	-
被调用函数	-
输入参数{in}	
-	-
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* enable digest calculation */
hau_digest_calculation_enable();
```

函数 hau_multiple_single_dma_config

函数hau_multiple_single_dma_config描述见下表：

表 3-742. 函数 hau_multiple_single_dma_config

函数名称	hau_multiple_single_dma_config
函数原形	void hau_multiple_single_dma_config(uint32_t multi_single);
功能描述	配置使用多DMA或单DMA，从未确定是否在DMA传输结束后计算摘要
先决条件	-
被调用函数	-
输入参数{in}	
multi_single	Multiple or single
SINGLE_DMA_AUTO_DIGEST	在DMA传输完成后消息填充和计算摘要

<i>MULTIPLE_DMA_NO_DIGEST</i>	需要多次DMA传输，在DMA传输结束时硬件不自动将CALEN位置1
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* configure multiple or single DMA is used, and digest calculation at the end of a DMA transfer or not */
```

```
hau_multiple_single_dma_config(SINGLE_DMA_AUTO_DIGEST);
```

函数 `hau_dma_enable`

函数 `hau_dma_enable` 描述见下表：

表 3-743. 函数 `hau_dma_enable`

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

例如：

```
/* enable the HAU DMA interface */
```

```
hau_dma_enable();
```

函数 `hau_dma_disable`

函数 `hau_dma_disable` 描述见下表：

表 3-744. 函数 `hau_dma_disable`

函数名称	<code>hau_dma_disable</code>
函数原形	<code>void hau_dma_disable(void);</code>
功能描述	除能HAU DMA接口
先决条件	-
被调用函数	-

输入参数{in}	
-	-
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* disable the HAU DMA interface */
```

```
hau_dma_disable();
```

函数 `hau_context_struct_para_init`

函数 `hau_context_struct_para_init` 描述见下表：

表 3-745. 函数 `hau_context_struct_para_init`

函数名称	<code>hau_context_struct_para_init</code>
函数原形	<code>void hau_context_struct_para_init(hau_context_parameter_struct* context)</code>
功能描述	初始化上下文结构体
先决条件	-
被调用函数	-
输入参数{in}	
-	-
输出参数{out}	
context	参考结构体 表3-732. 结构体 <code>hau_context_parameter_struct</code>
返回值	
-	-

例如：

```
/* initialize the struct context */
```

```
hau_context_parameter_struct context_para;
```

```
hau_context_struct_para_init(&context_para);
```

函数 `hau_context_save`

函数 `hau_context_save` 描述见下表：

表 3-746. 函数 `hau_context_save`

函数名称	<code>hau_context_save</code>
函数原形	<code>void hau_context_save(hau_context_parameter_struct* context_save)</code>
功能描述	保存HAU外设上下文
先决条件	-
被调用函数	-

输入参数{in}	
-	-
输出参数{out}	
context	参考结构体 表3-732. 结构体hau_context_parameter_struct
返回值	
-	-

例如：

```
hau_context_parameter_struct context_para;

hau_context_struct_para_init(&context_para);

/* save HAU context */

hau_context_save(&context_para);
```

函数 hau_context_restore

函数hau_context_restore描述见下表：

表 3-747. 函数 hau_context_restore

函数名称	hau_context_restore
函数原形	void hau_context_restore(hau_context_parameter_struct* context_restore)
功能描述	恢复HAU外设上下文
先决条件	-
被调用函数	-
输入参数{in}	
context	参考结构体 表3-732. 结构体hau_context_parameter_struct
输出参数{out}	
-	-
返回值	
-	-

例如：

```
hau_context_parameter_struct context_para;

hau_context_struct_para_init(&context_para);

hau_context_save(&context_para);

.....

/* restore HAU context */

hau_context_restore(&context_para);
```

函数 hau_hash_sha_1

函数hau_hash_sha_1描述见下表:

表 3-748. 函数 hau_hash_sha_1

函数名称	hau_hash_sha_1
函数原形	ErrStatus hau_hash_sha_1(uint8_t *input, uint32_t in_length, uint8_t output[]);
功能描述	在HASH模式下使用SHA1计算摘要
先决条件	-
被调用函数	-
输入参数{in}	
input	指向输入缓存区
输入参数{in}	
in_length	输入缓存区长度
输出参数{out}	
output	摘要结果
返回值	
ErrStatus	SUCCESS或ERROR

例如:

```
/* calculate digest using SHA1 in HASH mode */
ErrStatus status = ERROR;

uint8_t output[20];

uint8_t input[] = {0x01, 0x02, 0x03, 0x04, 0x05, 0x06, 0x07, 0x08,
                   0x01, 0x02, 0x03, 0x04, 0x05, 0x06, 0x07, 0x08;};

status = hau_hash_sha_1(input, 0x10, output);
```

函数 hau_hmac_sha_1

函数hau_hmac_sha_1描述见下表:

表 3-749. 函数 hau_hmac_sha_1

函数名称	hau_hmac_sha_1
函数原形	ErrStatus hau_hmac_sha_1(uint8_t *key, uint32_t keysize, uint8_t *input, uint32_t in_length, uint8_t output[]);
功能描述	在HMAC模式下使用SHA1计算摘要
先决条件	-
被调用函数	-
输入参数{in}	
key	指向用于HMAC模式的密钥
输入参数{in}	

keysize	用于HMAC模式的密钥长度
输入参数{in}	
input	指向输入缓存区
输入参数{in}	
in_length	输入缓存区长度
输出参数{out}	
output	摘要结果
返回值	
ErrStatus	SUCCESS或ERROR

例如:

```
/* calculate digest using SHA1 in HMAC mode */
```

```
ErrStatus status = ERROR;
```

```
uint8_t output[20];
```

```
uint8_t key[] = {0x01, 0x02, 0x03, 0x04, 0x05, 0x06, 0x07, 0x08,
                 0x01, 0x02, 0x03, 0x04, 0x05, 0x06, 0x07, 0x08};
```

```
uint8_t input[] = {0x01, 0x02, 0x03, 0x04, 0x05, 0x06, 0x07, 0x08,
                  0x01, 0x02, 0x03, 0x04, 0x05, 0x06, 0x07, 0x08};
```

```
status = hau_hmac_sha_1(key, 0x10, input, 0x10, output);
```

函数 hau_hash_sha_224

函数hau_hash_sha_224描述见下表:

表 3-750. 函数 hau_hash_sha_224

函数名称	hau_hash_sha_224
函数原形	ErrStatus hau_hash_sha_224(uint8_t *input, uint32_t in_length, uint8_t output[]);
功能描述	在HASH模式下使用SHA224计算摘要
先决条件	-
被调用函数	-
输入参数{in}	
input	指向输入缓存区
输入参数{in}	
in_length	输入缓存区长度
输出参数{out}	
output	摘要结果
返回值	
ErrStatus	SUCCESS或ERROR

例如:


```
/* calculate digest using SHA224 in HASH mode */
```

```
ErrStatus status = ERROR;
```

```
uint8_t output[28];
```

```
uint8_t input[] = {0x01, 0x02, 0x03, 0x04, 0x05, 0x06, 0x07, 0x08,  
                  0x01, 0x02, 0x03, 0x04, 0x05, 0x06, 0x07, 0x08};
```

```
status = hau_hash_sha_224 (input, 0x10, output);
```

函数 hau_hmac_sha_224

函数hau_hmac_sha_224描述见下表：

表 3-751. 函数 hau_hmac_sha_224

函数名称	hau_hmac_sha_224
函数原形	ErrStatus hau_hmac_sha_224(uint8_t *key, uint32_t keysize, uint8_t *input, uint32_t in_length, uint8_t output[]);
功能描述	在HMAC模式下使用SHA224计算摘要
先决条件	-
被调用函数	-
输入参数{in}	
key	指向用于HMAC模式的密钥
输入参数{in}	
keysize	用于HMAC模式的密钥长度
输入参数{in}	
input	指向输入缓存区
输入参数{in}	
in_length	输入缓存区长度
输出参数{out}	
output	摘要结果
返回值	
ErrStatus	SUCCESS或ERROR

例如：

```
/* calculate digest using SHA224 in HMAC mode */
```

```
ErrStatus status = ERROR;
```

```
uint8_t output[28];
```

```
uint8_t key[] = {0x01, 0x02, 0x03, 0x04, 0x05, 0x06, 0x07, 0x08,  
                0x01, 0x02, 0x03, 0x04, 0x05, 0x06, 0x07, 0x08};
```

```
uint8_t input[] = {0x01, 0x02, 0x03, 0x04, 0x05, 0x06, 0x07, 0x08,
```

0x01, 0x02, 0x03, 0x04, 0x05, 0x06, 0x07, 0x08};

status = hau_hmac_sha_224 (key, 0x10, input, 0x10, output);

函数 hau_hash_sha_256

函数hau_hash_sha_256描述见下表:

表 3-752. 函数 hau_hash_sha_256

函数名称	hau_hash_sha_256
函数原形	ErrStatus hau_hash_sha_256(uint8_t *input, uint32_t in_length, uint8_t output[]);
功能描述	在HASH模式下使用SHA256计算摘要
先决条件	-
被调用函数	-
输入参数{in}	
input	指向输入缓存区
输入参数{in}	
in_length	输入缓存区长度
输出参数{out}	
output	摘要结果
返回值	
ErrStatus	SUCCESS或ERROR

例如:

```
/* calculate digest using SHA256 in HASH mode */
```

```
ErrStatus status = ERROR;
```

```
uint8_t output[32];
```

```
uint8_t input[] = {0x01, 0x02, 0x03, 0x04, 0x05, 0x06, 0x07, 0x08,  
0x01, 0x02, 0x03, 0x04, 0x05, 0x06, 0x07, 0x08};
```

```
status = hau_hash_sha_256 (input, 0x10, output);
```

函数 hau_hmac_sha_256

函数hau_hmac_sha_256描述见下表:

表 3-753. 函数 hau_hmac_sha_256

函数名称	hau_hmac_sha_256
函数原形	ErrStatus hau_hmac_sha_256(uint8_t *key, uint32_t keysize, uint8_t *input, uint32_t in_length, uint8_t output[]);
功能描述	在HMAC模式下使用SHA256计算摘要
先决条件	-

被调用函数	-
输入参数{in}	
key	指向用于HMAC模式的密钥
输入参数{in}	
keysize	用于HMAC模式的密钥长度
输入参数{in}	
input	指向输入缓存区
输入参数{in}	
in_length	输入缓存区长度
输出参数{out}	
output	摘要结果
返回值	
ErrStatus	SUCCESS或ERROR

例如：

```
/* calculate digest using SHA256 in HMAC mode */
```

```
ErrStatus status = ERROR;
```

```
uint8_t output[32];
```

```
uint8_t key[] = {0x01, 0x02, 0x03, 0x04, 0x05, 0x06, 0x07, 0x08,  
                0x01, 0x02, 0x03, 0x04, 0x05, 0x06, 0x07, 0x08};
```

```
uint8_t input[] = {0x01, 0x02, 0x03, 0x04, 0x05, 0x06, 0x07, 0x08,  
                  0x01, 0x02, 0x03, 0x04, 0x05, 0x06, 0x07, 0x08};
```

```
status = hau_hmac_sha_256 (key, 0x10, input, 0x10, output);
```

函数 hau_hash_md5

函数hau_hash_md5描述见下表：

表 3-754. 函数 hau_hash_md5

函数名称	hau_hash_md5
函数原形	ErrStatus hau_hash_md5(uint8_t *input, uint32_t in_length, uint8_t output[]);
功能描述	在HASH模式下使用MD5计算摘要
先决条件	-
被调用函数	-
输入参数{in}	
input	指向输入缓存区
输入参数{in}	
in_length	输入缓存区长度
输出参数{out}	

output	摘要结果
返回值	
ErrStatus	SUCCESS或ERROR

例如:

```
/* calculate digest using MD5 in HASH mode */
```

```
ErrStatus status = ERROR;
```

```
uint8_t output[16];
```

```
uint8_t input[] = {0x01, 0x02, 0x03, 0x04, 0x05, 0x06, 0x07, 0x08,  
                  0x01, 0x02, 0x03, 0x04, 0x05, 0x06, 0x07, 0x08};
```

```
status = hau_hash_md5 (input, 0x10, output);
```

函数 hau_hmac_md5

函数hau_hmac_md5描述见下表:

表 3-755. 函数 hau_hmac_md5

函数名称	hau_hmac_md5
函数原形	ErrStatus hau_hmac_md5(uint8_t *key, uint32_t keysize, uint8_t *input, uint32_t in_length, uint8_t output[]);
功能描述	在HMAC模式下使用MD5计算摘要
先决条件	-
被调用函数	-
输入参数{in}	
key	指向用于HMAC模式的密钥
输入参数{in}	
keysize	用于HMAC模式的密钥长度
输入参数{in}	
input	指向输入缓存区
输入参数{in}	
in_length	输入缓存区长度
输出参数{out}	
output	摘要结果
返回值	
ErrStatus	SUCCESS或ERROR

例如:

```
/* calculate digest using MD5 in HMAC mode */
```

```
ErrStatus status = ERROR;
```

```
uint8_t output[16];
```

```
uint8_t key[] = {0x01, 0x02, 0x03, 0x04, 0x05, 0x06, 0x07, 0x08,
                 0x01, 0x02, 0x03, 0x04, 0x05, 0x06, 0x07, 0x08};

uint8_t input[] = {0x01, 0x02, 0x03, 0x04, 0x05, 0x06, 0x07, 0x08,
                  0x01, 0x02, 0x03, 0x04, 0x05, 0x06, 0x07, 0x08};

status = hau_hmac_md5 (key, 0x10, input, 0x10, output);
```

函数 hau_flag_get

函数hau_flag_get描述见下表:

表 3-756. 函数 hau_flag_get

函数名称	hau_flag_get
函数原形	FlagStatus hau_flag_get(uint32_t flag);
功能描述	获取HAU标志状态
先决条件	-
被调用函数	-
输入参数{in}	
flag	HAU标志状态
HAU_FLAG_DATA_INP UT	输入FIFO有足够空间
HAU_FLAG_CALCULA TION_COMPLETE	摘要计算完整
HAU_FLAG_DMA	DMA被使能或传输正在处理
HAU_FLAG_BUSY	数据块正在处理
HAU_FLAG_INFIFO_N O_EMPTY	输入FIFO非空
输出参数{out}	
-	-
返回值	
FlagStatus	SET或RESET

例如:

```
/* get the HAU flag status */

FlagStatus status;

status = hau_flag_get(HAU_FLAG_DMA);
```

函数 hau_flag_clear

函数hau_flag_clear描述见下表:

表 3-757. 函数 hau_flag_clear

函数名称	hau_flag_clear
------	----------------

函数原形	void hau_flag_clear(uint32_t flag);
功能描述	清除HAU标志状态
先决条件	-
被调用函数	-
输入参数{in}	
flag	HAU标志状态
HAU_FLAG_DATA_INPUT	输入FIFO有足够空间
HAU_FLAG_CALCULATION_COMPLETE	摘要计算完整
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* clear the HAU flag status */
```

```
hau_flag_clear(HAU_FLAG_DATA_INPUT);
```

函数 hau_interrupt_enable

函数hau_interrupt_enable描述见下表：

表 3-758. 函数 hau_interrupt_enable

函数名称	hau_interrupt_enable
函数原形	void hau_interrupt_enable(uint32_t interrupt);
功能描述	使能HAU中断
先决条件	-
被调用函数	-
输入参数{in}	
interrupt	指定需要使能的HAU中断
HAU_INT_DATA_INPUT	新数据块进入IN缓存区
HAU_INT_CALCULATION_COMPLETE	计算完成
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* enable hau interrupt */
```

```
hau_interrupt_enable(HAU_INT_DATA_INPUT);
```

函数 `hau_interrupt_disable`

函数 `hau_interrupt_disable` 描述见下表：

表 3-759. 函数 `hau_interrupt_disable`

函数名称	<code>hau_interrupt_disable</code>
函数原形	<code>void hau_interrupt_disable(uint32_t interrupt);</code>
功能描述	除能HAU中断
先决条件	-
被调用函数	-
输入参数{in}	
Interrupt	指定需要失能的HAU中断
<code>HAU_INT_DATA_INPUT</code>	新数据块进入IN缓存区
<code>HAU_INT_CALCULATION_COMPLETE</code>	计算完成
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* disable hau interrupt */
```

```
hau_interrupt_disable(HAU_INT_DATA_INPUT);
```

函数 `hau_interrupt_flag_get`

函数 `hau_interrupt_flag_get` 描述见下表：

表 3-760. 函数 `hau_interrupt_flag_get`

函数名称	<code>hau_interrupt_flag_get</code>
函数原形	<code>FlagStatus hau_interrupt_flag_get(uint32_t int_flag)</code>
功能描述	获取HAU中断标志状态
先决条件	-
被调用函数	-
输入参数{in}	
int_flag	HAU标志状态
<code>HAU_INT_FLAG_DATA_INPUT</code>	输入FIFO有足够空间
<code>HAU_INT_FLAG_CALCULATION_COMPLETE</code>	摘要计算完整
输出参数{out}	

-	-
返回值	
FlagStatus	SET或RESET

例如：

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

```
FlagStatus status = hau_interrupt_flag_get (HAU_INT_FLAG_DATA_INPUT);
```

函数 `hau_interrupt_flag_clear`

函数 `hau_interrupt_flag_clear` 描述见下表：

表 3-761. 函数 `hau_interrupt_flag_clear`

函数名称	<code>hau_interrupt_flag_clear</code>
函数原形	<code>void hau_interrupt_flag_clear(uint32_t int_flag)</code>
功能描述	清除HAU中断标志状态
先决条件	-
被调用函数	-
输入参数{in}	
int_flag	HAU标志状态
<code>HAU_INT_FLAG_DATA_INPUT</code>	输入FIFO有足够空间
<code>HAU_INT_FLAG_CALCULATION_COMPLETE</code>	摘要计算完整
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* clear the HAU interrupt flag status */
```

```
hau_interrupt_flag_clear(HAU_INT_FLAG_DATA_INPUT);
```


3.23. HPDF

GD32W51x 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 [3.23.1](#) 描述了 HPDF 的寄存器列表，章节 [3.23.2](#) 对 HPDF 库函数进行说明。

3.23.1. 外设寄存器说明

HPDF 寄存器列表如下表所示：

表 3-762. 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

3.23.2. 外设库函数说明

HPDF库函数列表如下表所示：

表 3-763. 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

库函数名称	库函数描述
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
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

库函数名称	库函数描述
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
hpdf_interrupt_disable	禁止HPDF中断
hpdf_interrupt_flag_get	获取HPDF中断标志位
hpdf_interrupt_flag_clear	清除HPDF中断标志位

结构体 `hpdf_channel_parameter_struct`

表 3-764. 结构体 `hpdf_channel_parameter_struct`

成员名称	功能描述
<code>data_packing_mode</code>	并行数据寄存器的数据封装模式
<code>channel_muxlexer</code>	复用通道输入数据源
<code>channel_pin_select</code>	通道输入引脚选择
<code>ck_loss_detector</code>	时钟丢失检测
<code>malfunction_monitor</code>	故障监视器
<code>spl_ck_source</code>	SPI接口时钟源
<code>serial_interface</code>	串行接口类型
<code>calibration_offset</code>	24位校准偏移
<code>right_bit_shift</code>	右移位数
<code>tm_filter</code>	阈值监视器滤波器阶数选择
<code>tm_filter_oversample</code>	阈值监视器滤波器过采样率
<code>mm_break_signal</code>	分配故障监视器断路信号
<code>mm_counter_threshold</code>	故障监视器计数阈值
<code>plsk_value</code>	跳频脉冲数

结构体 `hpdf_filter_parameter_struct`

表 3-765. 结构体 `hpdf_filter_parameter_struct`

成员名称	功能描述
<code>tm_fast_mode</code>	阈值监视器快速模式
<code>tm_channel</code>	阈值监视器通道
<code>tm_high_threshold</code>	阈值监视器上限阈值
<code>tm_low_threshold</code>	阈值监视器下限阈值
<code>extreme_monitor_channel</code>	极值监视器通道
<code>sinc_filter</code>	Sinc滤波器阶数
<code>sinc_oversample</code>	Sinc滤波器过采样率
<code>integrator_oversample</code>	积分器过采样率
<code>ht_break_signal</code>	分配上限阈值断路信号
<code>lt_break_signal</code>	分配下限阈值断路信号

结构体 `hpdf_rc_parameter_struct`

表 3-766. 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>rds_channel</code>	规则转换通道
<code>rdsmaen</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 3-767. 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 3-768. 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3-769. 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3-770. 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

成员名称	功能描述
HPDF_FLAG_FLT0_CKLF0	通道0时钟丢失标志
HPDF_FLAG_FLT0_CKLF1	通道1时钟丢失标志
HPDF_FLAG_FLT0_CKLF2	通道2时钟丢失标志
HPDF_FLAG_FLT0_CKLF3	通道3时钟丢失标志
HPDF_FLAG_FLT0_CKLF4	通道4时钟丢失标志
HPDF_FLAG_FLT0_CKLF5	通道5时钟丢失标志
HPDF_FLAG_FLT0_CKLF6	通道6时钟丢失标志
HPDF_FLAG_FLT0_CKLF7	通道7时钟丢失标志
HPDF_FLAG_FLT0_MMF0	通道0故障事件标志
HPDF_FLAG_FLT0_MMF1	通道1故障事件标志
HPDF_FLAG_FLT0_MMF2	通道2故障事件标志
HPDF_FLAG_FLT0_MMF3	通道3故障事件标志
HPDF_FLAG_FLT0_MMF4	通道4故障事件标志
HPDF_FLAG_FLT0_MMF5	通道5故障事件标志
HPDF_FLAG_FLT0_MMF6	通道6故障事件标志
HPDF_FLAG_FLT0_MMF7	通道7故障事件标志
HPDF_FLAG_FLTy_RCHPDT	规则通道等待处理数据
HPDF_FLAG_FLTy_LTF0	通道0阈值监视器下限阈值标志
HPDF_FLAG_FLTy_LTF1	通道1阈值监视器下限阈值标志
HPDF_FLAG_FLTy_LTF2	通道2阈值监视器下限阈值标志
HPDF_FLAG_FLTy_LTF3	通道3阈值监视器下限阈值标志
HPDF_FLAG_FLTy_LTF4	通道4阈值监视器下限阈值标志
HPDF_FLAG_FLTy_LTF5	通道5阈值监视器下限阈值标志
HPDF_FLAG_FLTy_LTF6	通道6阈值监视器下限阈值标志
HPDF_FLAG_FLTy_LTF7	通道7阈值监视器下限阈值标志
HPDF_FLAG_FLTy_HTF0	通道0阈值监视器上限阈值标志
HPDF_FLAG_FLTy_HTF1	通道1阈值监视器上限阈值标志
HPDF_FLAG_FLTy_HTF2	通道2阈值监视器上限阈值标志
HPDF_FLAG_FLTy_HTF3	通道3阈值监视器上限阈值标志
HPDF_FLAG_FLTy_HTF4	通道4阈值监视器上限阈值标志
HPDF_FLAG_FLTy_HTF5	通道5阈值监视器上限阈值标志
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3-771. 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

成员名称	功能描述
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3-772. 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 3-773. 函数 `hpdf_deinit`

函数名称	<code>hpdf_deinit</code>
函数原型	<code>void hpdf_deinit(void);</code>
功能描述	复位外设HPDF
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 HPDF */
```

```
hpdf_deinit();
```

函数 hpdf_channel_struct_para_init

函数hpdf_channel_struct_para_init描述见下表：

表 3-774. 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3-768. 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 3-775. 函数 hpdf_filter_struct_para_init

函数名称	hpdf_filter_struct_para_init
函数原型	void hpdf_filter_struct_para_init(hpdf_filter_parameter_struct* init_struct);
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}	
*init_struct	HPDF滤波器初始化结构体，参考 表3-765. 结构体 hpdf_filter_parameter_struct
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 3-776. 函数 hpdf_rc_struct_para_init

函数名称	hpdf_rc_struct_para_init
函数原型	void hpdf_rc_struct_para_init(hpdf_rc_parameter_struct* init_struct);
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}	
*init_struct	HPDF规则转换初始化结构体，参考 表3-766. 结构体 hpdf_rc_parameter_struct
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 3-777. 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3-767. 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 3-778. 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 3-779. 函数 `hpdf_disable`

函数名称	<code>hpdf_disable</code>
函数原型	<code>void hpdf_disable (void);</code>
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 3-780. 函数 `hpdf_channel_init`

函数名称	<code>hpdf_channel_init</code>
函数原型	<code>void hpdf_channel_init(hpdf_channel_enum channelx, hpdf_channel_parameter_struct* init_struct);</code>
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
<code>CHANNELx(x=0..7)</code>	HPDF通道选择，参考 表3-768. 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3-764. 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 3-781. 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3-769. 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3-765. 结构体 hpdf_filter_parameter_struct
输出参数{out}	
-	-
返回值	
-	-

例如：

```

/* initialize the HPDF fliter0 */

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 3-782. 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
FLTy(y=0..3)	HPDF滤波器选择，参考 表3-769. 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3-766. 结构体 hpdf_rc_parameter_struct
输出参数{out}	
-	-
返回值	
-	-

例如：

```

/* initialize regular conversion of the HPDF fliter0 */

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 3-783. 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3-767. 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 3-784. 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 3-785. 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 3-786. 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 3-787. 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 3-788. 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 3-789. 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3-768. 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 3-790. 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3-768. 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 3-791. 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
CHANNELx(x=0..7)	HPDF通道选择, 参考 表3-768. 枚举类型hpdf_channel_enum
输入参数{in}	
clock_source	SPI接口时钟源
EXTERNAL_CKIN	外部输入时钟
INTERNAL_CKOUT	内部CKOUT时钟
HALF_CKOUT_FALLIN G_EDGE	内部每第二个CKOUT时钟的下降沿
HALF_CKOUT_RISING _EDGE	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 3-792. 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
CHANNELx(x=0..7)	HPDF通道选择，参考 表3-768. 枚举类型hpdf_channel_enum
输入参数{in}	
type	串行接口类型
SPI_RISING_EDGE	SPI接口上升沿采样
SPI_FALLING_EDGE	SPI接口下降沿采样
MANCHESTER_CODE 0	曼切斯特编码输入：上升沿=逻辑0，下降沿=逻辑1
MANCHESTER_CODE 1	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 3-793. 函数 `hpdf_malfunction_monitor_disable`

函数名称	<code>hpdf_malfunction_monitor_disable</code>
函数原型	<code>void hpdf_malfunction_monitor_disable(hpdf_channel_enum channelx);</code>
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
<i>CHANNELx(x=0..7)</i>	HPDF通道选择, 参考 表3-768. 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 3-794. 函数 `hpdf_malfunction_monitor_enable`

函数名称	<code>hpdf_malfunction_monitor_enable</code>
函数原型	<code>void hpdf_malfunction_monitor_enable(hpdf_channel_enum channelx);</code>
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
<i>CHANNELx(x=0..7)</i>	HPDF通道选择, 参考 表3-768. 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 3-795. 函数 `hpdf_clock_loss_disable`

函数名称	<code>hpdf_clock_loss_disable</code>
函数原型	<code>void hpdf_clock_loss_disable(hpdf_channel_enum channelx);</code>
功能描述	禁止时钟检测
先决条件	<code>CHANNELx(x=0...7)</code> 禁止
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3-768. 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 3-796. 函数 `hpdf_clock_loss_enable`

函数名称	<code>hpdf_clock_loss_enable</code>
函数原型	<code>void hpdf_clock_loss_enable(hpdf_channel_enum channelx);</code>
功能描述	使能时钟检测
先决条件	<code>CHANNELx(x=0...7)</code> 禁止
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3-768. 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 3-797. 函数 `hpdf_channel_pin_redirection_disable`

函数名称	<code>hpdf_channel_pin_redirection_disable</code>
函数原型	<code>void hpdf_channel_pin_redirection_disable(hpdf_channel_enum channelx);</code>
功能描述	禁止通道引脚重定向
先决条件	<code>CHANNELx(x=0...7)</code> 禁止
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3-768. 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 3-798. 函数 `hpdf_channel_pin_redirection_enable`

函数名称	<code>hpdf_channel_pin_redirection_enable</code>
函数原型	<code>void hpdf_channel_pin_redirection_enable(hpdf_channel_enum channelx);</code>
功能描述	使能通道引脚重定向
先决条件	<code>CHANNELx(x=0...7)</code> 禁止
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3-768. 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_multiplexer_config`

函数 `hpdf_channel_multiplexer_config` 描述见下表:

表 3-799. 函数 `hpdf_channel_mux_config`

函数名称	<code>hpdf_channel_mux_config</code>
函数原型	<code>void hpdf_channel_mux_config(hpdf_channel_enum channelx, uint32_t data_source);</code>
功能描述	配置复用通道输入数据源
先决条件	<code>CHANNELx(x=0...7)</code> 禁止
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3-768. 枚举类型hpdf_channel_enum
输入参数{in}	
data_source	输入数据源
<code>SERIAL_INPUT</code>	输入数据源为串行输入数据
<code>INTERNAL_INPUT</code>	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 multiplexer 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 3-800. 函数 `hpdf_data_pack_mode_config`

函数名称	<code>hpdf_data_pack_mode_config</code>
函数原型	<code>void hpdf_data_pack_mode_config(hpdf_channel_enum channelx, uint32_t mode);</code>
功能描述	配置数据封装模式
先决条件	<code>CHANNELx(x=0...7)</code> 禁止
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3-768. 枚举类型hpdf_channel_enum
输入参数{in}	
mode	数据封装模式
<code>DPM_STANDARD_MODE</code>	标准模式
<code>DPM_INTERLEAVED_MODE</code>	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 3-801. 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3-768. 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 3-802. 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3-768. 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 3-803. 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
CHANNELx(x=0..7)	HPDF通道选择, 参考 表3-768. 枚举类型hpdf_channel_enum
输入参数{in}	
break_signal	数据封装模式
NO_MM_BREAK	无断路信号被分配
MM_BREAK0	断路信号0被分配至所监视的通道
MM_BREAK1	断路信号1被分配至所监视的通道
MM_BREAK2	断路信号2被分配至所监视的通道
MM_BREAK3	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 3-804. 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
CHANNELx(x=0..7)	HPDF通道选择，参考 表3-768. 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 3-805. 函数 hpdf_write_parallel_data_standard_mode

函数名称	hpdf_write_parallel_data_standard_mode
函数原型	void hpdf_write_parallel_data_standard_mode(hpdf_channel_enum channelx, int16_t data);
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
CHANNELx(x=0..7)	HPDF通道选择，参考 表3-768. 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 3-806. 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3-768. 枚举类型 <code>hpdf_channel_enum</code>
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 3-807. 函数 `hpdf_write_parallel_data_dual_mode`

函数名称	<code>hpdf_write_parallel_data_dual_mode</code>
函数原型	<code>void hpdf_write_parallel_data_dual_mode(hpdf_channel_enum channelx,</code>

	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3-768. 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 3-808. 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
CHANNELx(x=0..7)	HPDF通道选择, 参考 表3-768. 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 3-809. 函数 `hpdf_pulse_skip_read`

函数名称	<code>hpdf_pulse_skip_read</code>
函数原型	<code>uint8_t hpdf_pulse_skip_read(hpdf_channel_enum channelx);</code>
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3-768. 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 3-810. 函数 `hpdf_filter_enable`

函数名称	<code>hpdf_filter_enable</code>
函数原型	<code>void hpdf_filter_enable(hpdf_filter_enum filtery);</code>
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
<i>FLTy(y=0..3)</i>	HPDF滤波器选择，参考 表3-769. 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 3-811. 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3-769. 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 3-812. 函数 hpdf_filter_config

函数名称	hpdf_filter_config
函数原型	void hpdf_filter_config(hpdf_filter_enum filtery, uint32_t order, uint16_t oversample);
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
FLTy(y=0..3)	HPDF滤波器选择, 参考 表3-769. 枚举类型hpdf_filter_enum
输入参数{in}	
order	滤波器阶数
FLT_FASTSINC	FastSinc型滤波器
FLT_SINC1	Sinc1型滤波器
FLT_SINC2	Sinc2型滤波器
FLT_SINC3	Sinc3型滤波器

<i>FLT_SINC4</i>	Sinc4型滤波器
<i>FLT_SINC5</i>	Sinc5型滤波器
oversample	滤波器过采样率（1-1024）
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 3-813. 函数 hpdf_integrator_oversample

函数名称	hpdf_integrator_oversample
函数原型	void hpdf_integrator_oversample(hpdf_filter_enum filtery, uint8_t oversample);
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>FLT_y</i> (y=0..3)	HPDF滤波器选择，参考 表3-769. 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 3-814. 函数 `hpdf_threshold_monitor_filter_config`

函数名称	<code>hpdf_threshold_monitor_filter_config</code>
函数原型	<code>void hpdf_threshold_monitor_filter_config(hpdf_channel_enum channelx, uint32_t order, uint8_t oversample);</code>
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
<code>CHANNELx(x=0..7)</code>	HPDF通道选择, 参考 表3-768. 枚举类型hpdf_channel_enum
输入参数{in}	
order	滤波器阶数
<code>TM_FASTSINC</code>	FastSinc型滤波器
<code>TM_SINC1</code>	Sinc1型滤波器
<code>TM_SINC2</code>	Sinc2型滤波器
<code>TM_SINC3</code>	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 3-815. 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3-768. 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 3-816. 函数 hpdf_threshold_monitor_fast_mode_disable

函数名称	hpdf_threshold_monitor_fast_mode_disable
函数原型	void hpdf_threshold_monitor_fast_mode_disable(hpdf_filter_enum filtery);
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
<i>FLTy(y=0..3)</i>	HPDF滤波器选择，参考 表3-769. 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 3-817. 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
<i>FLTy(y=0..3)</i>	HPDF滤波器选择，参考 表3-769. 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 3-818. 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3-769. 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 3-819. 函数 `hpdf_threshold_monitor_high_threshold`

函数名称	<code>hpdf_threshold_monitor_high_threshold</code>
函数原型	<code>void hpdf_threshold_monitor_high_threshold(hpdf_filter_enum filtery, int32_t value);</code>
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
<i>FLTy(y=0..3)</i>	HPDF滤波器选择, 参考 表3-769. 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 3-820. 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3-769. 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 3-821. 函数 hpdf_high_threshold_break_signal

函数名称	hpdf_high_threshold_break_signal
函数原型	void hpdf_high_threshold_break_signal(hpdf_filter_enum filtery, uint32_t break_signal);
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
FLTy(y=0..3)	HPDF滤波器选择，参考 表3-769. 枚举类型hpdf_filter_enum
输入参数{in}	
break_signal	HPDF断路信号
NO_TM_HT_BREAK	禁止断路信号分配给阈值监视器的上限阈值事件
TM_HT_BREAK0	断路信号0分配给阈值监视器的上限阈值事件
TM_HT_BREAK1	断路信号1分配给阈值监视器的上限阈值事件
TM_HT_BREAK2	断路信号2分配给阈值监视器的上限阈值事件
TM_HT_BREAK3	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 3-822. 函数 hpdf_low_threshold_break_signal

函数名称	hpdf_low_threshold_break_signal
------	---------------------------------

函数原型	void hpdf_low_threshold_break_signal(hpdf_filter_enum filtery, uint32_t break_signal);
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
FLTy(y=0..3)	HPDF滤波器选择, 参考 表3-769. 枚举类型hpdf_filter_enum
输入参数{in}	
break_signal	HPDF断路信号
NO_TM_LT_BREAK	禁止断路信号分配给阈值监视器的下限阈值事件
TM_LT_BREAK0	断路信号0分配给阈值监视器的下限阈值事件
TM_LT_BREAK1	断路信号1分配给阈值监视器的下限阈值事件
TM_LT_BREAK2	断路信号2分配给阈值监视器的下限阈值事件
TM_LT_BREAK3	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 3-823. 函数 hpdf_extremes_monitor_channel

函数名称	hpdf_extremes_monitor_channel
函数原型	void hpdf_extremes_monitor_channel(hpdf_filter_enum filtery, uint32_t channel);
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
FLTy(y=0..3)	HPDF滤波器选择, 参考 表3-769. 枚举类型hpdf_filter_enum
输入参数{in}	
channel	极值监视器所监视的通道
EM_CHANNEL_DISABLE	禁止所有极值监视y接收通道的数据
EM_CHANNEL0	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 3-824. 函数 hpdf_extremes_monitor_maximum_get

函数名称	hpdf_extremes_monitor_maximum_get
函数原型	int32_t hpdf_extremes_monitor_maximum_get(hpdf_filter_enum filtery);
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3-769. 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 3-825. 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
FLTy(y=0..3)	HPDF滤波器选择, 参考 表3-769. 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 3-826. 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3-769. 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 3-827. 函数 `hpdf_rc_continuous_disable`

函数名称	<code>hpdf_rc_continuous_disable</code>
函数原型	<code>void hpdf_rc_continuous_disable(hpdf_filter_enum filtery);</code>
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
<i>FLTy(y=0..3)</i>	HPDF滤波器选择, 参考 表3-769. 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 3-828. 函数 `hpdf_rc_continuous_enable`

函数名称	<code>hpdf_rc_continuous_enable</code>
函数原型	<code>void hpdf_rc_continuous_enable(hpdf_filter_enum filtery);</code>
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
<i>FLTy(y=0..3)</i>	HPDF滤波器选择, 参考 表3-769. 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 3-829. 函数 `hpdf_rc_start_by_software`

函数名称	<code>hpdf_rc_start_by_software</code>
函数原型	<code>void hpdf_rc_start_by_software(hpdf_filter_enum filtery);</code>
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
<i>FLTy(y=0..3)</i>	HPDF滤波器选择, 参考 表3-769. 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 3-830. 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 <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3-769. 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 3-831. 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3-769. 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 3-832. 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3-769. 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 3-833. 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3-769. 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 3-834. 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3-769. 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3-768. 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 3-835. 函数 hpdf_rc_fast_mode_disable

函数名称	hpdf_rc_fast_mode_disable
函数原型	void hpdf_rc_fast_mode_disable(hpdf_filter_enum filtery);
功能描述	禁止规则转换快速模式
先决条件	禁止FLT _y (y=0..3)
被调用函数	-
输入参数{in}	
filtery	HPDF模块滤波器
FLT _y (y=0..3)	HPDF滤波器选择, 参考 表3-769. 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 3-836. 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FLT _y (y=0..3)
被调用函数	-
输入参数{in}	
filtery	HPDF模块滤波器
FLT _y (y=0..3)	HPDF滤波器选择, 参考 表3-769. 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 3-837. 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3-769. 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 3-838. 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3-769. 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 3-839. 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3-769. 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 3-840. 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3-769. 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 3-841. 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3-769. 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 3-842. 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3-769. 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 3-843. 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3-769. 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 3-844. 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3-769. 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 3-845. 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3-769. 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 3-846. 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3-769. 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 3-847. 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FLT _y (y=0..3)
被调用函数	-
输入参数{in}	
filtery	HPDF模块滤波器
FLT _y (y=0..3)	HPDF滤波器选择，参考 表3-769. 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 3-848. 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3-769. 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 3-849. 函数 `hpdf_ic_data_get`

函数名称	<code>hpdf_ic_data_get</code>
函数原型	<code>int32_t hpdf_ic_data_get(hpdf_filter_enum filtery);</code>
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
<i>FLTy(y=0..3)</i>	HPDF滤波器选择, 参考 表3-769. 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 3-850. 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3-769. 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 3-851. 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3-769. 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 DOF</i>	注入转换溢出标志
<i>HPDF_FLAG_FLTy_RC DOF</i>	规则转换溢出标志
<i>HPDF_FLAG_FLTy_TM EOF</i>	阈值监视器事件标志
<i>HPDF_FLAG_FLTy_IC PF</i>	注入转换正在进行标志
<i>HPDF_FLAG_FLTy_RC PF</i>	规则转换正在进行标志
<i>HPDF_FLAG_FLT0_CK LFx</i>	通道(x = 0..7)时钟丢失标志
<i>HPDF_FLAG_FLT0_M MFx</i>	通道(x = 0..7)故障事件标志
<i>HPDF_FLAG_FLTy_RC HPDT</i>	规则通道等待处理数据
<i>HPDF_FLAG_FLTy_LT Fx</i>	通道(x = 0..7)阈值监视器下限阈值标志
<i>HPDF_FLAG_FLTy_HT 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 3-852. 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
FLTy(y=0..3)	HPDF滤波器选择，参考 表3-769. 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 3-853. 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3-769. 枚举类型hpdf_filter_enum
输入参数{in}	
interrupt	HPDF模块中断
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_CKLE	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_TMEOIE);
```

函数 **hpdf_interrupt_disable**

函数hpdf_interrupt_disable描述见下表:

表 3-854. 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3-769. 枚举类型hpdf_filter_enum
输入参数{in}	
interrupt	HPDF模块中断
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_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(FTL0, HPDF_INT_FLT0_CKLI E);
```

函数 **hpdf_interrupt_flag_get**

函数hpdf_interrupt_flag_get描述见下表:

表 3-855. 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3-769. 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 3-856. 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3-769. 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_FLT y_ICEF	注入转换结束中断标志
HPDF_INT_FLAG_FLT y_RCEF	规则转换结束中断标志
HPDF_INT_FLAG_FLT y_ICDOF	注入转换溢出中断标志
HPDF_INT_FLAG_FLT y_RCDOF	规则转换溢出中断标志
HPDF_INT_FLAG_FLT y_TMEOF	阈值监视器事件中断标志
HPDF_INT_FLAG_FLT 0_CKLFx	通道(x = 0..7)时钟丢失中断标志
HPDF_INT_FLAG_FLT 0_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);
```

3.24. HWSEM

硬件信号量（HWSEM）模块提供了一种非阻塞机制来保证不同进程之间的同步。章节[3.24.1](#)描述了HWSEM的寄存器列表，章节[3.24.2](#)对HWSEM库函数进行说明。

3.24.1. 外设寄存器描述

HWSEM寄存器列表如下表所示:

表 3-857. HWSEM 寄存器

寄存器名称	寄存器描述
HWSEM_CTLx (x =	控制寄存器 x

寄存器名称	寄存器描述
0..31)	
HWSEM_RLKx (x = 0..31)	读锁定寄存器 x
HWSEM_INTEN	中断使能寄存器
HWSEM_INTC	中断状态清除寄存器
HWSEM_STAT	状态寄存器
HWSEM_INTF	中断状态寄存器
HWSEM_UNLK	解锁寄存器
HWSEM_KEY	键值寄存器

3.24.2. 外设库函数说明

HWSEM库函数列表如下表所示：

表 3-858. HWSEM 库函数

库函数名称	库函数描述
hwsem_lock_set	尝试通过写进程 ID 来锁定指定的信号量
hwsem_lock_release	尝试通过写进程 ID 来解锁指定的信号量
hwsem_lock_by_reading	尝试通过读来锁定信号量
hwsem_unlock_all	解锁主控总线 ID 下的所有信号量
hwsem_process_id_get	获取指定信号量的进程 ID
hwsem_master_id_get	获取指定信号量的主控总线 ID
hwsem_lock_status_get	获取信号量的锁状态
hwsem_key_set	设置解锁密钥
hwsem_key_get	获取解锁密钥
hwsem_flag_get	获取 HWSEM 解锁标志状态
hwsem_flag_clear	清除 HWSEM 解锁标志状态
hwsem_interrupt_flag_get	获取 HWSEM 解锁中断标志状态
hwsem_interrupt_flag_clear	清除 HWSEM 解锁中断标志状态
hwsem_interrupt_enable	使能 HWSEM 中断
hwsem_interrupt_disable	禁能 HWSEM 中断

枚举类型 hwsem_semaphore_enum

表 3-859. 枚举类型 hwsem_semaphore_enum

枚举名称	枚举描述
SEM0	信号量 0
SEM1	信号量 1
SEM2	信号量 2
SEM3	信号量 3
SEM4	信号量 4
SEM5	信号量 5

枚举名称	枚举描述
SEM6	信号量 6
SEM7	信号量 7
SEM8	信号量 8
SEM9	信号量 9
SEM10	信号量 10
SEM11	信号量 11
SEM12	信号量 12
SEM13	信号量 13
SEM14	信号量 14
SEM15	信号量 15
SEM16	信号量 16
SEM17	信号量 17
SEM18	信号量 18
SEM19	信号量 19
SEM20	信号量 20
SEM21	信号量 21
SEM22	信号量 22
SEM23	信号量 23
SEM24	信号量 24
SEM25	信号量 25
SEM26	信号量 26
SEM27	信号量 27
SEM28	信号量 28
SEM29	信号量 29
SEM30	信号量 30
SEM31	信号量 31

函数 hwsem_lock_set

函数hwsem_lock_set描述见下表：

表 3-860. 函数 hwsem_lock_set

函数名称	hwsem_lock_set
函数原型	ErrStatus hwsem_lock_set(hwsem_semaphore_enum semaphore, uint8_t process);
功能描述	尝试通过写进程 ID 来锁定指定的信号量
先决条件	-
被调用函数	-
输入参数{in}	
semaphore	信号量索引，参考枚举 表 3-859. 枚举类型 hwsem_semaphore_enum
SEMx	x=0..31
输入参数{in}	

process	锁定信号量的进程
0..0xFF	进程 ID

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

例如:

```
ErrStatus state;
```

```
/* lock semaphore 0 by process 0 */
```

```
state = hwsem_lock_set(SEM0, 0);
```

函数 hwsem_lock_release

函数hwsem_lock_release描述见下表:

表 3-861. 函数 hwsem_lock_release

函数名称	hwsem_lock_release
函数原型	ErrStatus hwsem_lock_release(hwsem_semaphore_enum semaphore, uint8_t process);
功能描述	尝试通过写进程 ID 来解锁指定的信号量
先决条件	-
被调用函数	-
输入参数{in}	
semaphore	信号量索引, 参考枚举 表 3-859. 枚举类型 hwsem_semaphore_enum
SEMx	x=0..31
输入参数{in}	
process	解锁信号量的进程
0..0xFF	进程 ID
输出参数{out}	
-	-
返回值	
ErrStatus	SUCCESS 或 ERROR

例如:

```
ErrStatus state;
```

```
/* unlock semaphore 0 by process 0 */
```

```
state = hwsem_lock_release(SEM0, 0);
```

函数 hwsem_lock_by_reading

函数hwsem_lock_by_reading描述见下表：

表 3-862. 函数 hwsem_lock_by_reading

函数名称	hwsem_lock_by_reading
函数原型	ErrStatus hwsem_lock_by_reading(hwsem_semaphore_enum semaphore);
功能描述	尝试通过读来锁定信号量
先决条件	-
被调用函数	-
输入参数{in}	
semaphore	信号量索引，参考枚举 表 3-859. 枚举类型 hwsem_semaphore_enum
SEMX	x=0..31
输出参数{out}	
-	-
返回值	
ErrStatus	SUCCESS 或 ERROR

例如：

```
ErrStatus state;
```

```
/* unlock semaphore 0 by reading */
```

```
state = hwsem_lock_by_reading(SEM0);
```

函数 hwsem_unlock_all

函数hwsem_unlock_all描述见下表：

表 3-863. 函数 hwsem_unlock_all

函数名称	hwsem_unlock_all
函数原型	ErrStatus hwsem_unlock_all(uint16_t key);
功能描述	解锁主控总线 ID 下的所有信号量
先决条件	-
被调用函数	-
输入参数{in}	
key	密钥值
0 - 0xFFFF	用于解锁的密钥值
输出参数{out}	
-	-
返回值	
ErrStatus	SUCCESS 或 ERROR

例如：

```
ErrStatus state;
```

```
/* unlock all semaphores */
```

```
state = hwsem_unlock_all(0);
```

函数 hwsem_process_id_get

函数hwsem_process_id_get描述见下表：

表 3-864. 函数 hwsem_process_id_get

函数名称	hwsem_process_id_get
函数原型	uint32_t hwsem_process_id_get(hwsem_semaphore_enum semaphore);
功能描述	获取指定信号量的进程 ID
先决条件	-
被调用函数	-
输入参数{in}	
semaphore	信号量索引，参考枚举 表 3-859. 枚举类型 hwsem_semaphore_enum
SEMx	x=0..31
输出参数{out}	
-	-
返回值	
uint32_t	0 - 0xFF

例如：

```
uint32_t pid = 0;
```

```
/* get process ID of semaphore 0 */
```

```
pid = hwsem_process_id_get(SEM0);
```

函数 hwsem_master_id_get

函数hwsem_master_id_get描述见下表：

表 3-865. 函数 hwsem_master_id_get

函数名称	hwsem_master_id_get
函数原型	uint32_t hwsem_master_id_get(hwsem_semaphore_enum semaphore);
功能描述	获取指定信号量的主控总线 ID
先决条件	-
被调用函数	-
输入参数{in}	
semaphore	信号量索引，参考枚举 表 3-859. 枚举类型 hwsem_semaphore_enum
SEMx	x=0..31
输出参数{out}	
-	-
返回值	
uint32_t	0 - 0xF

例如：

```
uint32_t mid = 0;

/* get master ID of semaphore 0 */

mid = hwsem_master_id_get(SEM0);
```

函数 hwsem_lock_status_get

函数hwsem_lock_status_get描述见下表：

表 3-866. 函数 hwsem_lock_status_get

函数名称	hwsem_lock_status_get
函数原型	FlagStatus hwsem_lock_status_get(hwsem_semaphore_enum semaphore);
功能描述	获取信号量的锁状态
先决条件	-
被调用函数	-
输入参数{in}	
semaphore	信号量索引，参考枚举 表 3-859. 枚举类型 hwsem_semaphore_enum
SEMx	x=0..31
输出参数{out}	
-	-
返回值	
FlagStatus	SET 或 RESET

例如：

```
FlagStatus flag;

/* get the lock status */

flag = hwsem_lock_status_get(SEM0);
```

函数 hwsem_key_set

函数hwsem_key_set描述见下表：

表 3-867. 函数 hwsem_key_set

函数名称	hwsem_key_set
函数原型	void hwsem_key_set(uint16_t key);
功能描述	设置解锁密钥
先决条件	-
被调用函数	-
输入参数{in}	
key	密钥值
0 - 0xFFFF	要设置的密钥值

输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* set unlock key */
```

```
hwsem_key_set(0);
```

函数 hwsem_key_get

函数hwsem_key_get描述见下表：

表 3-868. 函数 hwsem_key_get

函数名称	hwsem_key_get
函数原型	uint16_t hwsem_key_get(void);
功能描述	获取解锁密钥
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：

```
uint16_t key;
```

```
/* get the unlock key */
```

```
key = hwsem_key_get();
```

函数 hwsem_flag_get

函数hwsem_flag_get描述见下表：

表 3-869. 函数 hwsem_flag_get

函数名称	hwsem_flag_get
函数原型	FlagStatus hwsem_flag_get(hwsem_semaphore_enum semaphore);
功能描述	获取 HWSEM 解锁标志状态
先决条件	-
被调用函数	-
输入参数{in}	
semaphore	信号量索引，参考枚举 表 3-859. 枚举类型 hwsem_semaphore_enum

<i>SEM</i> x	x=0..31
输出参数{out}	
-	-
返回值	
FlagStatus	SET 或 RESET

例如:

```
FlagStatus flag;
```

```
/* get the HWSEM unlock flag status */
```

```
flag = hwsem_flag_get(SEM0);
```

函数 hwsem_flag_clear

函数hwsem_flag_clear描述见下表:

表 3-870. 函数 hwsem_flag_clear

函数名称	hwsem_flag_clear
函数原型	void hwsem_flag_clear(hwsem_semaphore_enum semaphore);
功能描述	清除 HWSEM 解锁标志状态
先决条件	-
被调用函数	-
输入参数{in}	
semaphore	信号量索引, 参考枚举 表 3-859. 枚举类型 hwsem_semaphore_enum
<i>SEM</i> x	x=0..31
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* clear HWSEM unlock flag status */
```

```
hwsem_flag_clear(SEM0);
```

函数 hwsem_interrupt_flag_get

函数hwsem_interrupt_flag_get描述见下表:

表 3-871. 函数 hwsem_interrupt_flag_get

函数名称	hwsem_interrupt_flag_get
函数原型	FlagStatus hwsem_interrupt_flag_get(hwsem_semaphore_enum semaphore);
功能描述	获取 HWSEM 解锁中断标志状态
先决条件	-

被调用函数	-
输入参数{in}	
semaphore	信号量索引，参考枚举 表 3-859. 枚举类型 hwsem_semaphore_enum
SEMx	x=0..31
输出参数{out}	
-	-
返回值	
FlagStatus	SET 或 RESET

例如：

FlagStatus flag;

/* get the HWSEM unlock interrupt flag status */

flag = hwsem_interrupt_flag_get(SEM0);

函数 hwsem_interrupt_flag_clear

函数hwsem_interrupt_flag_clear描述见下表：

表 3-872. 函数 hwsem_interrupt_flag_clear

函数名称	hwsem_interrupt_flag_clear
函数原型	void hwsem_interrupt_flag_clear(hwsem_semaphore_enum semaphore);
功能描述	清除 HWSEM 解锁中断标志状态
先决条件	-
被调用函数	-
输入参数{in}	
semaphore	信号量索引，参考枚举 表 3-859. 枚举类型 hwsem_semaphore_enum
SEMx	x=0..31
输出参数{out}	
-	-
返回值	
-	-

例如：

/* clear HWSEM unlock interrupt flag status */

hwsem_interrupt_flag_clear(SEM0);

函数 hwsem_interrupt_enable

函数hwsem_interrupt_enable描述见下表：

表 3-873. 函数 hwsem_interrupt_enable

函数名称	hwsem_interrupt_enable
函数原型	void hwsem_interrupt_enable(hwsem_semaphore_enum semaphore);

功能描述	使能 HWSEM 解锁中断
先决条件	-
被调用函数	-
输入参数{in}	
semaphore	信号量索引, 参考枚举 表 3-859. 枚举类型 hwsem_semaphore_enum
SEMx	x=0..31
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* enable HWSEM unlock interrupt */
```

```
hwsem_interrupt_enable(SEM0);
```

函数 hwsem_interrupt_disable

函数hwsem_interrupt_disable描述见下表:

表 3-874. 函数 hwsem_interrupt_disable

函数名称	hwsem_interrupt_disable
函数原型	void hwsem_interrupt_disable(hwsem_semaphore_enum semaphore);
功能描述	禁能 HWSEM 解锁中断
先决条件	-
被调用函数	-
输入参数{in}	
semaphore	信号量索引, 参考枚举 表 3-859. 枚举类型 hwsem_semaphore_enum
SEMx	x=0..31
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* disable HWSEM unlock interrupt */
```

```
hwsem_interrupt_disable(SEM0);
```

3.25. 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[3.25.1](#)描述了I2C的寄存器列表，章节[3.25.2](#)对I2C库函数进行说明。

3.25.1. 外设寄存器说明

I2C寄存器列表如下表所示：

表 3-875. 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

3.25.2. 外设库函数说明

I2C库函数列表如下表所示：

表 3-876. I2C 库函数

库函数名称	库函数描述
i2c_deinit	复位外设 I2C
i2c_timing_config	配置时序参数
i2c_digital_noise_filter_config	配置数字噪声过滤器
i2c_analog_noise_filter_enable	使能数字噪声过滤器
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

库函数名称	库函数描述
i2c_address_bit_compare_config	定义 ADDRESS[7:1]的哪些位和接收到的地址进行比较
i2c_address_disable	禁能从机模式下 I2C 地址
i2c_second_address_config	配置 I2C 从机第二个地址
i2c_second_address_disable	禁能 I2C 从机第二个地址
i2c_received_address_get	获取从机模式下匹配成功的地址
i2c_slave_byte_control_enable	使能从机字节控制
i2c_slave_byte_control_disable	禁能从机字节控制
i2c_nack_enable	从机模式下产生 NACK
i2c_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
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

库函数名称	库函数描述
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 3-877. 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 3-878. 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 3-879. 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 3-880. 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  $t_{I2CCLK}$  */
```

```
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 3-881. 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 3-882. 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 3-883. 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 3-884. 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 3-885. 函数 i2c_address10_header_enable

函数名称	i2c_address10_header_enable
函数原型	void i2c_address10_header_enable(uint32_t i2c_periph);
功能描述	主机接收模式下，10 位地址头只执行读操作
先决条件	-
被调用函数	-
输入参数{in}	
i2c_periph	I2C 外设
I2Cx	(x=0,1,2,3)
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 3-886. 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 3-887. 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 3-888. 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 3-889. 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 3-890. 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 3-891. 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 3-892. 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 3-893. 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 3-894. 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 3-895. 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 3-896. 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
<i>I2Cx</i>	(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 3-897. 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
<i>I2Cx</i>	(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 3-898. 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 3-899. 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 3-900. 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_receivied_address_get(I2C0);
```

函数 i2c_slave_byte_control_enable

函数i2c_slave_byte_control_enable描述见下表：

表 3-901. 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 3-902. 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 3-903. 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 3-904. 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
<i>I2Cx</i>	(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 3-905. 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
<i>I2Cx</i>	(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 3-906. 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 3-907. 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 3-908. 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 3-909. 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 3-910. 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 3-911. 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 3-912. 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 3-913. 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 3-914. 函数 i2c_reload_disable

函数名称	i2c_reload_disable
函数原型	void i2c_reload_disable(uint32_t i2c_periph);
功能描述	禁能 I2C 重载模式
先决条件	-
被调用函数	-
输入参数{in}	
i2c_periph	I2C 外设
I2Cx	(x=0,1,2,3)
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 3-915. 函数 i2c_transfer_byte_number_config

函数名称	i2c_transfer_byte_number_config
函数原型	void i2c_transfer_byte_number_config(uint32_t i2c_periph, uint32_t byte_number);
功能描述	配置待发送字节数
先决条件	-
被调用函数	-
输入参数{in}	
i2c_periph	I2C 外设
I2Cx	(x=0,1,2,3)
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 3-916. 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
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：

```
/* 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 3-917. 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
<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：

```
/* 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 3-918. 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
<i>I2Cx</i>	(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 3-919. 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 3-920. 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 3-921. 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 3-922. 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 3-923. 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 3-924. 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 3-925. 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 3-926. 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 3-927. 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 3-928. 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 3-929. 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 3-930. 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 3-931. 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 3-932. 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 3-933. 函数 `i2c_bus_timeout_a_config`

函数名称	<code>i2c_bus_timeout_a_config</code>
函数原型	<code>void i2c_bus_timeout_a_config(uint32_t i2c_periph, uint32_t timeout);</code>
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}	
<code>i2c_periph</code>	I2C 外设
<code>I2Cx</code>	(x=0,1,2,3)
输入参数{in}	
<code>timeout</code>	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 3-934. 函数 `i2c_idle_clock_timeout_config`

函数名称	<code>i2c_idle_clock_timeout_config</code>
函数原型	<code>void i2c_idle_clock_timeout_config(uint32_t i2c_periph, uint32_t timeout);</code>
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}	
<code>i2c_periph</code>	I2C 外设
<code>I2Cx</code>	(x=0,1,2,3)
输入参数{in}	
<code>timeout</code>	总线超时 A
<code>BUSTOA_DETECT_SCL_LOW</code>	BUSTOA 用于检测 SCL 低电平超时
<code>BUSTOA_DETECT_IDLE</code>	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 3-935. 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
I2Cx	(x=0,1,2,3)
输入参数{in}	
flag	I2C 标志位
I2C_FLAG_TBE	发送期间 I2C_TDATA 寄存器空标志
I2C_FLAG_TI	发送中断标志
I2C_FLAG_RBNE	接收期间 I2C_RDATA 非空标志
I2C_FLAG_ADDSEND	从机模式下，接收到的地址与自身地址匹配
I2C_FLAG_NACK	NACK 标志
I2C_FLAG_STPDET	从机模式下检测到 STOP 信号
I2C_FLAG_TC	主机模式下传输完成标志
I2C_FLAG_TCR	传输完成重载标志
I2C_FLAG_BERR	总线错误标志
I2C_FLAG_LOSTARB	仲裁丢失标志
I2C_FLAG_OUERR	从机模式下，过载/欠载错误标志
I2C_FLAG_PECERR	PEC 错误标志
I2C_FLAG_TIMEOUT	超时标志
I2C_FLAG_SMBALT	SMBus 报警标志
I2C_FLAG_I2CBSY	忙标志
I2C_FLAG_TR	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 3-936. 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
I2Cx	(x=0,1,2,3)
输入参数{in}	
flag	I2C 标志位
I2C_FLAG_ADDSEND	从机模式下, 接收到的地址与自身地址匹配
I2C_FLAG_NACK	NACK 标志
I2C_FLAG_STPDET	从机模式下检测到 STOP 信号
I2C_FLAG_BERR	总线错误标志
I2C_FLAG_LOSTARB	仲裁丢失标志
I2C_FLAG_OUERR	从机模式下, 过载/欠载错误标志
I2C_FLAG_PECERR	PEC 错误标志
I2C_FLAG_TIMEOUT	超时标志
I2C_FLAG_SMBALT	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 3-937. 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 3-938. 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 3-939. 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 3-877. 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_STPD ET	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_OUER	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 3-940. 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 3-877. 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
I2C_INT_FLAG_NACK	NACK 中断标志
I2C_INT_FLAG_STPD	从机模式下检测到 STOP 信号中断标志

<i>ET</i>	
<i>I2C_INT_FLAG_BERR</i>	总线错误中断标志
<i>I2C_INT_FLAG_LOSTARB</i>	仲裁丢失中断标志
<i>I2C_INT_FLAG_OUER</i> <i>R</i>	从机模式下，过载/欠载错误中断标志
<i>I2C_INT_FLAG_PECERR</i>	PEC 错误中断标志
<i>I2C_INT_FLAG_TIMEOUT</i> <i>UT</i>	超时中断标志
<i>I2C_INT_FLAG_SMBAL</i> <i>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);
```

3.26. IPA

IPA提供从某一个或两个源图像到目标图像的可配置的，灵活的图像处理功能。章节[3.26.1](#)描述了IPA的寄存器列表，章节[3.26.2](#)对IPA库函数进行说明。

3.26.1. 外设寄存器描述

IPA 寄存器列表如下表所示：

表 3-941. IPA 寄存器

寄存器名称	寄存器描述
IPA_CTL	控制寄存器
IPA_INTF	中断状态寄存器
IPA_INTC	中断标志清除寄存器
IPA_FMADDR	前景层存储区基地址寄存器
IPA_FLOFF	前景层行偏移寄存器
IPA_BMADDR	背景层存储区基地址寄存器
IPA_BLOFF	背景层行偏移寄存器
IPA_FPCTL	前景层像素控制寄存器
IPA_FPV	前景层像素值寄存器
IPA_BPCTL	背景层像素控制寄存器

寄存器名称	寄存器描述
IPA_BPV	背景层像素值寄存器
IPA_FLMADDR	前景层 LUT 存储区基地址寄存器
IPA_BLMADDR	背景层 LUT 存储区基地址寄存器
IPA_DPCTL	目标像素控制寄存器
IPA_DPV	目标像素值寄存器
IPA_DMADDR	目标存储区基地址寄存器
IPA_DLOFF	目标行偏移寄存器
IPA_IMS	图像大小寄存器
IPA_LM	行标记寄存器
IPA_ITCTL	内部定时器控制寄存器
IPA_BSCTL	双线性缩放控制寄存器
IPA_DIMS	缩放目标图像大小寄存器
IPA_EF_UV_MAD DR	前景层偶数帧/UV 存储区基地址寄存器
IPA_CSCC_CFG0	色彩空间转换系数配置寄存器0
IPA_CSCC_CFG1	色彩空间转换系数配置寄存器1
IPA_CSCC_CFG2	色彩空间转换系数配置寄存器2

3.26.2. 外设库函数说明

IPA 库函数列表如下表所示：

表 3-942. IPA 库函数

库函数名称	库函数描述
ipa_deinit	重新初始化IPA
ipa_transfer_enable	使能IPA传输
ipa_transfer_hangup_enable	使能IPA传输挂起
ipa_transfer_hangup_disable	禁能IPA传输挂起
ipa_transfer_stop_enable	使能IPA传输停止
ipa_transfer_stop_disable	禁能IPA传输停止
ipa_foreground_lut_loading_enable	使能前景层LUT加载
ipa_background_lut_loading_enable	使能背景层LUT加载
ipa_pixel_format_convert_mode_set	设置像素格式转换模式，该函数在IPA传输使能的情况下调用无效
ipa_foreground_interlace_mode_enable	使能IPA前景层隔行模式
ipa_foreground_interlace_mode_disable	禁能IPA前景层隔行模式
ipa_foreground_struct_para_init	用默认值初始化IPA前景层参数结构体，建议在定义一个ipa_foreground_parameter_struct结构体后调用该接口实现对结构体的初始化
ipa_foreground_init	初始化前景层参数

库函数名称	库函数描述
ipa_background_struct_para_init	用默认值初始化IPA背景层参数结构体，建议在定义一个ipa_background_parameter_struct结构体后调用该接口实现对结构体的初始化
ipa_background_init	初始化背景层参数
ipa_destination_struct_para_init	用默认值初始化IPA目标参数结构体，建议在定义一个ipa_destination_parameter_struct结构体后调用该接口实现对结构体的初始化
ipa_destination_init	初始化目标参数
ipa_foreground_lut_init	初始化IPA前景层LUT参数
ipa_background_lut_init	初始化IPA背景层LUT参数
ipa_line_mark_config	配置IPA行标记
ipa_inter_timer_config	配置IPA内部定时器使能或禁能
ipa_interval_clock_num_config	配置间隔时钟周期数
ipa_color_conversion_struct_para_init	用默认值初始化IPA颜色转换结构体，建议在定义一个ipa_foreground_parameter_struct结构体后调用该接口实现对结构体的初始化
ipa_color_conversion_config	配置IPA颜色转换功能
ipa_foreground_scaling_config	配置IPA前景层缩放功能
ipa_destination_scaling_config	配置IPA目标层缩放功能
ipa_flag_get	从IPA_INTF寄存器获取IPA标志位状态
ipa_flag_clear	清除IPA标志位
ipa_interrupt_enable	使能IPA中断
ipa_interrupt_disable	禁能IPA中断
ipa_interrupt_flag_get	获取IPA中断标志位
ipa_interrupt_flag_clear	清除IPA中断标志位

结构体 ipa_foreground_parameter_struct

表 3-943. 结构体 ipa_foreground_parameter_struct

成员名称	功能描述
foreground_memaddr	前景层存储区基地址
foreground_lineoff	前景层行偏移
foreground_prealpha	前景层预定义alpha通道值
foreground_alpha_algorithm	前景层alpha值计算算法
foreground_pf	前景层像素格式
foreground_prered	前景层预定义红色值
foreground_pregreen	前景层预定义绿色值
foreground_preblue	前景层预定义蓝色值
foreground_interlace_mode	前景层隔行输入模式
foreground_efuv_memaddr	前景层偶数帧/UV存储区基地址

结构体 ipa_background_parameter_struct

表 3-944. 结构体 ipa_background_parameter_struct

成员名称	功能描述
background_memaddr	背景层存储区基地址
background_lineoff	背景层行偏移
background_prealpha	背景层预定义alpha通道值
background_alpha_algorithm	背景层alpha值计算算法
background_pf	背景层像素格式
background_prered	背景层预定义红色值
background_pregreen	背景层预定义绿色值
background_preblue	背景层预定义蓝色值

结构体 ipa_destination_parameter_struct

表 3-945. 结构体 ipa_destination_parameter_struct

成员名称	功能描述
destination_memaddr	目标存储区基地址
destination_lineoff	目标存储区行偏移
destination_prealpha	目标存储区预定义alpha通道值
destination_pf	目标存储区像素格式
destination_prered	目标存储区预定义红色值
destination_pregreen	目标存储区预定义绿色值
destination_preblue	目标存储区预定义蓝色值
image_width	图像宽度
image_height	图像高度
image_rotate	图像旋转
image_hor_decimation	图像水平预抽取
image_ver_decimation	图像垂直预抽取
image_bilinear_xscale	图像X缩放因子
image_bilinear_yscale	图像Y缩放因子
image_scaling_width	缩放后图像宽度
image_scaling_height	缩放后图像高度

结构体 ipa_conversion_parameter_struct

表 3-946. 结构体 ipa_conversion_parameter_struct

成员名称	功能描述
color_space	颜色空间转换模式
y_offset	Y数据中的偏移量
uv_offset	UV数据中的偏移量
coef_c0	Y乘数系数
coef_c1	V/Cr红色乘数系数

coef_c2	V/Cr绿色乘数系数
coef_c3	U/Cb绿色乘数系数
coef_c4	U/Cb蓝色乘数系数

枚举类型 `ipa_dpf_enum`

表 3-947. 枚举类型 `ipa_dpf_enum`

枚举名称	枚举描述
IPA_DPF_ARGB8888	目标像素格式ARGB8888
IPA_DPF_RGB888	目标像素格式RGB888
IPA_DPF_RGB565	目标像素格式RGB565
IPA_DPF_ARGB1555	目标像素格式ARGB1555
IPA_DPF_ARGB4444	目标像素格式ARGB4444

枚举类型 `ipa_colorspace_enum`

表 3-948. 枚举类型 `ipa_colorspace_enum`

枚举名称	枚举描述
IPA_COLORSPACE_YUV	IPA颜色转换使用YUV参数
IPA_COLORSPACE_YCBCR	IPA颜色转换使用YCbCr参数

函数 `ipa_deinit`

函数ipa_deinit描述见下表：

表 3-949. 函数 `ipa_deinit`

函数名称	ipa_deinit
函数原型	void ipa_deinit(void);
功能描述	重新初始化IPA
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 IPA */
```

```
ipa_deinit();
```


函数 ipa_transfer_enable

函数ipa_transfer_enable描述见下表：

表 3-950. 函数 ipa_transfer_enable

函数名称	ipa_transfer_enable
函数原型	void ipa_transfer_enable(void);
功能描述	使能IPA传输
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 IPA transfer */
```

```
ipa_transfer_enable();
```

函数 ipa_transfer_hangup_enable

函数ipa_transfer_hangup_enable描述见下表：

表 3-951. 函数 ipa_transfer_hangup_enable

函数名称	ipa_transfer_hangup_enable
函数原型	void ipa_transfer_hangup_enable(void);
功能描述	使能IPA传输挂起
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 IPA transfer hang up */
```

```
ipa_transfer_hangup_enable();
```

函数 ipa_transfer_hangup_disable

函数ipa_transfer_hangup_disable描述见下表：

表 3-952. 函数 ipa_transfer_hangup_disable

函数名称	ipa_transfer_hangup_disable
函数原型	void ipa_transfer_hangup_disable(void);
功能描述	禁能IPA传输挂起
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 IPA transfer hang up */
```

```
ipa_transfer_hangup_disable();
```

函数 ipa_transfer_stop_enable

函数ipa_transfer_stop_enable描述见下表：

表 3-953. 函数 ipa_transfer_stop_enable

函数名称	ipa_transfer_stop_enable
函数原型	void ipa_transfer_stop_enable(void);
功能描述	使能IPA传输停止
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 IPA transfer stop */
```

```
ipa_transfer_stop_enable();
```

函数 ipa_transfer_stop_disable

函数ipa_transfer_stop_disable描述见下表:

表 3-954. 函数 ipa_transfer_stop_disable

函数名称	ipa_transfer_stop_disable
函数原型	void ipa_transfer_stop_disable(void);
功能描述	禁能IPA传输停止
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 IPA transfer stop */
```

```
ipa_transfer_stop_disable();
```

函数 ipa_foreground_lut_loading_enable

函数ipa_foreground_lut_loading_enable描述见下表:

表 3-955. 函数 ipa_foreground_lut_loading_enable

函数名称	ipa_foreground_lut_loading_enable
函数原型	void ipa_foreground_lut_loading_enable(void);
功能描述	使能前景层LUT加载
先决条件	-
被调用函数	-
输入参数{in}	
-	-
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* enable IPA foreground LUT loading */
```

```
ipa_foreground_lut_loading_enable();
```

函数 ipa_background_lut_loading_enable

函数ipa_background_lut_loading_enable描述见下表：

表 3-956. 函数 ipa_background_lut_loading_enable

函数名称	ipa_background_lut_loading_enable
函数原型	void ipa_background_lut_loading_enable(void);
功能描述	使能背景层LUT加载
先决条件	-
被调用函数	-
输入参数{in}	
-	-
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* enable IPA background LUT loading */
```

```
ipa_background_lut_loading_enable();
```

函数 ipa_pixel_format_convert_mode_set

函数ipa_pixel_format_convert_mode_set描述见下表：

表 3-957. 函数 ipa_pixel_format_convert_mode_set

函数名称	ipa_pixel_format_convert_mode_set
函数原型	void ipa_pixel_format_convert_mode_set(uint32_t pfcmm);
功能描述	设置像素格式转换模式，该函数在IPA传输使能的情况下调用无效
先决条件	-
被调用函数	-
输入参数{in}	
pfcmm	像素格式转换模式
IPA_FGTCODE	前景层存储区到目标存储区无像素格式转换
IPA_FGTCODE_PF_CONVERT	前景层存储区到目标存储区有像素格式转换
IPA_FBGTCODE	混合前景层和背景层存储区到目标存储区
IPA_FILL_UP_DE	用特定的颜色填充目标存储区
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* configure foreground memory to destination memory with pixel format convert */
```

```
ipa_pixel_format_convert_mode_set(IPA_FGTODE_PF_CONVERT);
```

函数 ipa_foreground_interlace_mode_enable

函数ipa_foreground_interlace_mode_enable描述见下表:

表 3-958. 函数 ipa_foreground_interlace_mode_enable

函数名称	ipa_foreground_interlace_mode_enable
函数原型	void ipa_foreground_interlace_mode_enable(void);
功能描述	使能IPA前景层隔行模式
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 foreground interlace mode */
```

```
ipa_foreground_interlace_mode_enable();
```

函数 ipa_foreground_interlace_mode_disable

函数ipa_foreground_interlace_mode_disable描述见下表:

表 3-959. 函数 ipa_foreground_interlace_mode_disable

函数名称	ipa_foreground_interlace_mode_disable
函数原型	void ipa_foreground_interlace_mode_disable(void);
功能描述	禁能IPA前景层隔行模式
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 foreground interlace mode */
```

```
ipa_foreground_interlace_mode_disable();
```

函数 ipa_foreground_struct_para_init

函数ipa_foreground_struct_para_init描述见下表：

表 3-960. 函数 ipa_foreground_struct_para_init

函数名称	ipa_foreground_struct_para_init
函数原型	void ipa_foreground_struct_para_init(ipa_foreground_parameter_struct* foreground_struct);
功能描述	用默认值初始化IPA前景层参数结构体，建议在定义一个ipa_foreground_parameter_struct结构体后调用该接口实现对结构体的初始化
先决条件	-
被调用函数	-
输入参数{in}	
*foreground_struct	指向ipa_foreground_parameter_struct结构体的指针，参考 表3-943. 结构体 ipa_foreground_parameter_struct
输出参数{out}	
-	-
返回值	
-	-

例如：

```
ipa_foreground_parameter_struct fg_struct;
```

```
/* initialize the structure of IPA foreground parameter struct with the default values */
```

```
ipa_foreground_struct_para_init(&fg_struct);
```

函数 ipa_foreground_init

函数ipa_foreground_init描述见下表：

表 3-961. 函数 ipa_foreground_init

函数名称	ipa_foreground_init
函数原型	void ipa_foreground_init(ipa_foreground_parameter_struct* foreground_struct);
功能描述	初始化前景层参数
先决条件	-
被调用函数	-
输入参数{in}	
*foreground_struct	指向 ipa_foreground_parameter_struct结构体的指针，参考 表3-943. 结构体 ipa_foreground_parameter_struct
输出参数{out}	
-	-
返回值	

例如：

```
ipa_foreground_parameter_struct ipa_fg_init_struct;

ipa_foreground_struct_para_init(&ipa_fg_init_struct);

/* configure IPA foreground */

ipa_fg_init_struct.foreground_memaddr = (uint32_t)&gBuffer;

ipa_fg_init_struct.foreground_pf = FOREGROUND_PPF_RGB565;

ipa_fg_init_struct.foreground_alpha_algorithm = IPA_FG_ALPHA_MODE_1;

ipa_fg_init_struct.foreground_prealpha = 0x75;

ipa_fg_init_struct.foreground_lineoff = 0x00;

ipa_fg_init_struct.foreground_preblue = 0x00;

ipa_fg_init_struct.foreground_pregreen = 0x00;

ipa_fg_init_struct.foreground_prered = 0x00;

ipa_fg_init_struct.foreground_efuv_memaddr = 0x00;

ipa_fg_init_struct.foreground_interlace_mode = DISABLE;

/* foreground initialization */

ipa_foreground_init(&ipa_fg_init_struct);
```

函数 ipa_background_struct_para_init

函数ipa_background_struct_para_init描述见下表：

表 3-962. 函数 ipa_background_struct_para_init

函数名称	ipa_background_struct_para_init
函数原型	void ipa_background_struct_para_init(ipa_background_parameter_struct* background_struct);
功能描述	用默认值初始化IPA背景层参数结构体，建议在定义一个 ipa_background_parameter_struct结构体后调用该接口实现对结构体的初始化
先决条件	-
被调用函数	-
输入参数{in}	
*background_struct	指向ipa_background_parameter_struct结构体的指针，参考 表3-944. 结构体 ipa_background_parameter_struct
输出参数{out}	
-	-
返回值	

-	-
---	---

例如：

```
ipa_background_parameter_struct bg_struct;
```

```
/* initialize the structure of IPA background parameter struct with the default values */
```

```
ipa_background_struct_para_init(&bg_struct);
```

函数 ipa_background_init

函数ipa_background_init描述见下表：

表 3-963. 函数 ipa_background_init

函数名称	ipa_background_init
函数原型	void ipa_background_init(ipa_background_parameter_struct* background_struct);
功能描述	初始化背景层参数
先决条件	-
被调用函数	-
输入参数{in}	
*background_struct	指向ipa_background_parameter_struct结构体的指针，参考 表3-944. 结构体 ipa_background_parameter_struct
输出参数{out}	
-	-
返回值	
-	-

例如：

```
ipa_background_parameter_struct ipa_bg_init_struct;
```

```
ipa_background_struct_para_init(&ipa_bg_init_struct);
```

```
/* configure IPA background */
```

```
ipa_bg_init_struct.background_memaddr = (uint32_t)&gBuffer;
```

```
ipa_bg_init_struct.background_pf = BACKGROUND_PPF_RGB565;
```

```
ipa_bg_init_struct.background_alpha_algorithm = IPA_BG_ALPHA_MODE_0;
```

```
ipa_bg_init_struct.background_prealpha = 255;
```

```
ipa_bg_init_struct.background_lineoff = 0x00;
```

```
ipa_bg_init_struct.background_preblue = 0x00;
```

```
ipa_bg_init_struct.background_pregreen = 0x00;
```

```
ipa_bg_init_struct.background_prered = 0x00;
```



```
/* background initialization */
```

```
ipa_background_init(&ipa_bg_init_struct);
```

函数 ipa_destination_struct_para_init

函数ipa_destination_struct_para_init描述见下表：

表 3-964. 函数 ipa_destination_struct_para_init

函数名称	ipa_destination_struct_para_init
函数原型	void ipa_destination_struct_para_init(ipa_destination_parameter_struct* destination_struct);
功能描述	用默认值初始化IPA目标参数结构体，建议在定义一个ipa_destination_parameter_struct结构体后调用该接口实现对结构体的初始化
先决条件	-
被调用函数	-
输入参数{in}	
*destination_struct	指向ipa_destination_parameter_struct结构体的指针，参考 表3-945. 结构体 ipa_destination_parameter_struct
输出参数{out}	
-	-
返回值	
-	-

例如：

```
ipa_destination_parameter_struct destination_struct;
```

```
/* initialize the structure of IPA destination parameter struct with the default values */
```

```
ipa_destination_struct_para_init(&destination_struct);
```

函数 ipa_destination_init

函数ipa_destination_init描述见下表：

表 3-965. 函数 ipa_destination_init

函数名称	ipa_destination_init
函数原型	void ipa_destination_init(ipa_destination_parameter_struct* destination_struct);
功能描述	初始化目标参数
先决条件	-
被调用函数	-
输入参数{in}	
*destination_struct	指向ipa_destination_parameter_struct结构体的指针，参考 表3-945. 结构体 ipa_destination_parameter_struct
输出参数{out}	
-	-

返回值	
-	-

例如:

```

ipa_destination_parameter_struct ipa_destination_init_struct;

ipa_destination_struct_para_init(&ipa_destination_init_struct);

/* configure destination pixel format */

ipa_destination_init_struct.destination_pf = IPA_DPF_RGB565;

/* configure destination memory base address */

ipa_destination_init_struct.destination_memaddr = (uint32_t)&gBuffer;

/* configure destination pre-defined alpha value RGB */

ipa_destination_init_struct.destination_pregreen = 0;

ipa_destination_init_struct.destination_preblue = 0;

ipa_destination_init_struct.destination_prered = 0;

ipa_destination_init_struct.destination_prealpha = 0;

/* configure destination line offset */

ipa_destination_init_struct.destination_lineoff = 0;

/* configure height of the image to be processed */

ipa_destination_init_struct.image_height = 160;

/* configure width of the image to be processed */

ipa_destination_init_struct.image_width = 229;

ipa_destination_init_struct.image_rotate = DESTINATION_ROTATE_0;

ipa_destination_init_struct.image_hor_decimation                      =
DESTINATION_HORDECIMATE_DISABLE;

ipa_destination_init_struct.image_ver_decimation                    =
DESTINATION_VERDECIMATE_DISABLE;

ipa_destination_init_struct.image_scaling_height = 160;

ipa_destination_init_struct.image_scaling_width = 229;

ipa_destination_init_struct.image_bilinear_xscale = 0x1000;

ipa_destination_init_struct.image_bilinear_yscale = 0x1000;

/* ipa destination initialization */

```

```
ipa_destination_init(&ipa_destination_init_struct);
```

函数 ipa_foreground_lut_init

函数ipa_foreground_lut_init描述见下表：

表 3-966. 函数 ipa_foreground_lut_init

函数名称	ipa_foreground_lut_init
函数原型	void ipa_foreground_lut_init(uint8_t fg_lut_num, uint8_t fg_lut_pf, uint32_t fg_lut_addr);
功能描述	初始化IPA前景层LUT参数
先决条件	-
被调用函数	-
输入参数{in}	
fg_lut_num	前景层LUT像素数目
输入参数{in}	
fg_lut_pf	前景层LUT像素格式
IPA_LUT_PF_ARG B8888	LUT像素格式是ARGB8888
IPA_LUT_PF_RGB 888	LUT像素格式是RGB888
输入参数{in}	
fg_lut_addr	前景层LUT存储区基地址
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* initialize the LUT foreground LUT parameters */
```

```
ipa_foreground_lut_init(1, IPA_LUT_PF_ARGB8888, 0x20002000);
```

函数 ipa_background_lut_init

函数ipa_background_lut_init描述见下表：

表 3-967. 函数 ipa_background_lut_init

函数名称	ipa_background_lut_init
函数原型	void ipa_background_lut_init(uint8_t bg_lut_num, uint8_t bg_lut_pf, uint32_t bg_lut_addr);
功能描述	初始化IPA背景层LUT参数
先决条件	-
被调用函数	-
输入参数{in}	

bg_lut_num	背景层LUT像素数目
输入参数{in}	
bg_lut_pf	背景层LUT像素格式
<i>IPA_LUT_PF_ARG B8888</i>	LUT像素格式是ARGB8888
<i>IPA_LUT_PF_RGB 888</i>	LUT像素格式是RGB888
输入参数{in}	
bg_lut_addr	背景层LUT存储区基地址
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* initialize the LUT background LUT parameters */
```

```
ipa_background_lut_init(2, IPA_LUT_PF_RGB888, 0x20001000);
```

函数 ipa_line_mark_config

函数ipa_line_mark_config描述见下表：

表 3-968. 函数 ipa_line_mark_config

函数名称	ipa_line_mark_config
函数原型	void ipa_line_mark_config(uint16_t line_num);
功能描述	配置IPA行标记
先决条件	-
被调用函数	-
输入参数{in}	
line_num	行数目
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* configure line mark */
```

```
ipa_line_mark_config(10);
```

函数 ipa_inter_timer_config

函数ipa_inter_timer_config描述见下表：

表 3-969. 函数 ipa_inter_timer_config

函数名称	ipa_inter_timer_config
函数原型	void ipa_inter_timer_config(uint8_t timer_cfg);
功能描述	配置IPA内部定时器使能或禁能
先决条件	-
被调用函数	-
输入参数{in}	
timer_cfg	内部定时器配置
IPA_INTER_TIMER_ENABLE	使能内部定时器
IPA_INTER_TIMER_DISABLE	禁能内部定时器
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* enable inter-timer */
```

```
ipa_inter_timer_config(IPA_INTER_TIMER_ENABLE);
```

函数 ipa_interval_clock_num_config

函数ipa_interval_clock_num_config描述见下表:

表 3-970. 函数 ipa_interval_clock_num_config

函数名称	ipa_interval_clock_num_config
函数原型	void ipa_interval_clock_num_config(uint8_t clk_num);
功能描述	配置间隔时钟周期数
先决条件	-
被调用函数	-
输入参数{in}	
clk_num	间隔时钟周期数
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* configure the number of clock cycles interval and it has no meaning if ITEN is '0' */
```

```
ipa_interval_clock_num_config(1);
```

函数 `ipa_color_conversion_struct_para_init`

函数 `ipa_color_conversion_struct_para_init` 描述见下表：

表 3-971. 函数 `ipa_color_conversion_struct_para_init`

函数名称	<code>ipa_color_conversion_struct_para_init</code>
函数原型	<code>void ipa_color_conversion_struct_para_init(ipa_conversion_parameter_struct* conversion_struct, ipa_colorspace_enum colorspace);</code>
功能描述	用默认值初始化IPA颜色转换结构体，建议在定义一个 <code>ipa_foreground_parameter_struct</code> 结构体后调用该接口实现对结构体的初始化
先决条件	-
被调用函数	-
输入参数{in}	
colorspace	颜色空间，参考 表3-948. 枚举类型ipa_colorspace_enum
<code>IPA_COLORSPACE_YUV</code>	IPA颜色转换使用YUV参数
<code>IPA_COLORSPACE_YCBCR</code>	IPA颜色转换使用YCbCr参数
输入参数{out}	
conversion_struct	指向 <code>ipa_conversion_parameter_struct</code> 结构体的指针，参考 表3-946. 结构体 ipa_conversion_parameter_struct
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* initialize the structure of IPA ipa_conversion_parameter_struct with the default values */
ipa_conversion_parameter_struct conversion_struct;

ipa_color_conversion_struct_para_init(&conversion_struct, IPA_COLORSPACE_YUV);
```

函数 `ipa_color_conversion_config`

函数 `ipa_color_conversion_config` 描述见下表：

表 3-972. 函数 `ipa_color_conversion_config`

函数名称	<code>ipa_color_conversion_config</code>
函数原型	<code>void ipa_color_conversion_config(ipa_conversion_parameter_struct* conversion_struct);</code>
功能描述	配置IPA颜色转换功能
先决条件	-
被调用函数	-
输入参数{in}	

*foreground_struct	指向ipa_foreground_parameter_struct结构体的指针，参考 表3-946. 结构体 ipa_conversion_parameter_struct
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* configure the color space conversion parameters */
ipa_conversion_parameter_struct conversion_struct;
conversion_struct.color_space = IPA_COLORSPACE_YUV;
conversion_struct.y_offset = 0x0;
conversion_struct.uv_offset = 0x0;
conversion_struct.coef_c0 = 0x100;
conversion_struct.coef_c1 = 0x123;
conversion_struct.coef_c2 = 0x76B;
conversion_struct.coef_c3 = 0x79C;
conversion_struct.coef_c4 = 0x208;
ipa_color_conversion_config(&conversion_struct);
```

函数 ipa_foreground_scaling_config

函数ipa_foreground_scaling_config描述见下表：

表 3-973. 函数 ipa_foreground_scaling_config

函数名称	ipa_foreground_scaling_config
函数原型	void ipa_foreground_scaling_config(uint32_t horizontal_decimation, uint32_t vertical_decimation, uint32_t image_scaling_width, uint32_t image_scaling_height);
功能描述	配置IPA前景层缩放功能
先决条件	-
被调用函数	-
输入参数{in}	
horizontal_decimation	水平缩放值
DESTINATION_HORDECIMATE_DISABLE	禁能水平抽取
DESTINATION_HO	水平抽取为2

<i>RDECIMATE_2</i>	
<i>DESTINATION_HORDECIMATE_4</i>	水平抽取为4
<i>DESTINATION_HORDECIMATE_8</i>	水平抽取为8
输入参数{in}	
vertical_decimation	垂直抽取值
<i>DESTINATION_VERDECIMATE_DISABLE</i>	禁能垂直抽取
<i>DESTINATION_VERDECIMATE_2</i>	垂直抽取为2
<i>DESTINATION_VERDECIMATE_4</i>	垂直抽取为4
<i>DESTINATION_VERDECIMATE_8</i>	垂直抽取为8
输入参数{in}	
image_scaling_width	图像宽度缩放因子
输入参数{in}	
image_scaling_height	图像高度缩放因子
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* configure IPA foreground scaling */
```

```
ipa_foreground_scaling_config(DESTINATION_HORDECIMATE_2,  
DESTINATION_VERDECIMATE_2, 320, 240);
```

函数 ipa_destination_scaling_config

函数ipa_destination_scaling_config描述见下表：

表 3-974. 函数 ipa_destination_scaling_config

函数名称	ipa_destination_scaling_config
函数原型	void ipa_destination_scaling_config(uint32_t dest_scaling_width, uint32_t dest_scaling_height);
功能描述	配置IPA目标层缩放功能
先决条件	-

被调用函数	-
输入参数{in}	
dest_scaling_width	缩放后的图像宽度
输入参数{in}	
dest_scaling_height	缩放后的图像高度
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* configure IPA destination scaling */
ipa_destination_scaling_config(320, 240);
```

函数 ipa_flag_get

函数ipa_flag_get描述见下表：

表 3-975. 函数

函数名称	ipa_flag_get
函数原型	FlagStatus ipa_flag_get(uint32_t flag);
功能描述	从IPA_INTF寄存器获取IPA标志位状态
先决条件	-
被调用函数	-
输入参数{in}	
flag	IPA标志
IPA_FLAG_TAE	传输访问错误中断标志
IPA_FLAG_FTF	传输完成中断标志
IPA_FLAG_TLM	传输行标记中断标志
IPA_FLAG_LAC	LUT访问冲突中断标志位
IPA_FLAG_LLF	LUT加载完成中断标志LUT
IPA_FLAG_WCF	配置错误中断标志
输出参数{out}	
-	-
返回值	
FlagStatus	SET或RESET

例如：

```
/* wait for full transfer finish flag set */
while(RESET == ipa_flag_get(IPA_FLAG_FTF));
```

函数 ipa_flag_clear

函数ipa_flag_clear描述见下表:

表 3-976. 函数

函数名称	ipa_flag_clear
函数原型	void ipa_flag_clear(uint32_t flag);
功能描述	清除IPA标志位
先决条件	-
被调用函数	-
输入参数{in}	
flag	IPA标志
IPA_FLAG_TAE	传输访问错误中断标志
IPA_FLAG_FTF	传输完成中断标志
IPA_FLAG_TLM	传输行标记中断标志
IPA_FLAG_LAC	LUT访问冲突中断标志位
IPA_FLAG_LLF	LUT加载完成中断标志LUT
IPA_FLAG_WCF	配置错误中断标志
输出参数{out}	
-	-
返回值	
-	-

例如:

Example:

```
/* wait for full transfer finish flag set */
while(RESET ==ipa_flag_get(IPA_FLAG_FTF));
/* clear full transfer finish flag */
ipa_flag_clear(IPA_FLAG_FTF);
```

函数 ipa_interrupt_enable

函数ipa_interrupt_enable描述见下表:

表 3-977. 函数 ipa_interrupt_enable

函数名称	ipa_interrupt_enable
函数原型	void ipa_interrupt_enable(uint32_t int_flag);
功能描述	使能IPA中断
先决条件	-
被调用函数	-
输入参数{in}	
int_flag	IPA中断标志位

<i>IPA_INT_TAE</i>	传输访问错误中断标志
<i>IPA_INT_FTF</i>	传输完成中断标志
<i>IPA_INT_TLM</i>	传输行标记中断标志
<i>IPA_INT_LAC</i>	LUT访问冲突中断标志位
<i>IPA_INT_LLF</i>	LUT加载完成中断标志LUT
<i>IPA_INT_WCF</i>	配置错误中断标志
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* enable full transfer finish interrupt */
```

```
ipa_interrupt_enable(IPA_INT_FTF);
```

函数 ipa_interrupt_disable

函数ipa_interrupt_disable描述见下表：

表 3-978. 函数 ipa_interrupt_disable

函数名称	ipa_interrupt_disable
函数原型	void ipa_interrupt_disable(uint32_t int_flag);
功能描述	禁能IPA中断
先决条件	-
被调用函数	-
输入参数{in}	
int_flag	IPA中断标志位
<i>IPA_INT_TAE</i>	传输访问错误中断标志
<i>IPA_INT_FTF</i>	传输完成中断标志
<i>IPA_INT_TLM</i>	传输行标记中断标志
<i>IPA_INT_LAC</i>	LUT访问冲突中断标志位
<i>IPA_INT_LLF</i>	LUT加载完成中断标志LUT
<i>IPA_INT_WCF</i>	配置错误中断标志
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* disable full transfer finish interrupt */
```

```
ipa_interrupt_disable(IPA_INT_FTF);
```

函数 ipa_interrupt_flag_get

函数ipa_interrupt_flag_get描述见下表:

表 3-979. 函数 ipa_interrupt_flag_get

函数名称	ipa_interrupt_flag_get
函数原型	FlagStatus ipa_interrupt_flag_get(uint32_t int_flag);
功能描述	获取IPA中断标志位
先决条件	-
被调用函数	-
输入参数{in}	
int_flag	IPA中断标志位
IPA_INT_FLAG_TAE	传输访问错误中断标志
IPA_INT_FLAG_FTF	传输完成中断标志
IPA_INT_FLAG_TLM	传输行标记中断标志
IPA_INT_FLAG_LAC	LUT访问冲突中断标志位
IPA_INT_FLAG_LLF	LUT加载完成中断标志LUT
IPA_INT_FLAG_WCF	配置错误中断标志
输出参数{out}	
-	-
返回值	
FlagStatus	SET或RESET

例如:

```
/* check whether full transfer finish interrupt flag is SET */
while(RESET == ipa_interrupt_flag_get(IPA_INT_FLAG_FTF));
```

函数 ipa_interrupt_flag_clear

函数ipa_interrupt_flag_clear描述见下表:

表 3-980. 函数 ipa_interrupt_flag_clear

函数名称	ipa_interrupt_flag_clear
函数原型	void ipa_interrupt_flag_clear(uint32_t int_flag);
功能描述	清除IPA中断标志位
先决条件	-
被调用函数	-
输入参数{in}	
int_flag	IPA中断标志位
IPA_INT_FLAG_TAE	传输访问错误中断标志
IPA_INT_FLAG_FTF	传输完成中断标志
IPA_INT_FLAG_TLM	传输行标记中断标志
IPA_INT_FLAG_LAC	LUT访问冲突中断标志位

IPA_INT_FLAG_LLF	LUT加载完成中断标志LUT
IPA_INT_FLAG_WCF	配置错误中断标志
输出参数{out}	
-	-
返回值	
-	-

例如：

```
if(ipa_interrupt_flag_get(IPA_INT_FLAG_FTF) != RESET){
    /* clear full transfer finish interrupt flag */
    ipa_interrupt_flag_clear(IPA_INT_FLAG_FTF);
}
```

3.27. 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[3.27.1](#)描述了LPDTS的寄存器列表，章节[3.27.2](#)对LPDTS库函数进行说明。

3.27.1. 外设寄存器说明

LPDTS寄存器列表如下表所示：

表 3-981. 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
LPDTS_INTC	中断标志清除寄存器
LPDTS_OP	选择寄存器

3.27.2. 外设库函数说明

LPDTS库函数列表如下表所示：

表 3-982. LPDTS 库函数

库函数名称	库函数描述
lpdts_deinit	复位外设LPDTS

库函数名称	库函数描述
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 3-983. 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 3-984. 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 3-985. 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3-983. 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 3-986. 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3-983. 结构体 lpdts_parameter_struct

	lpdts_parameter_struct
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* initilize 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 3-987. 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 3-988. 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 3-989. 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 3-990. 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 3-991. 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 3-992. 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 3-993. 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 3-994. 函数 lpdts_temperature_get

函数名称	lpdts_temperature_get
函数原形	int32_t lpdts_temperature_get(void);
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	
int32_t	最终获得的温度值

例如:

```
int32_t temperature;
```

```
/* wait till TSRF flag is set */
```

```
while(RESET == lpdts_flag_get(LPDTTS_FLAG_TSRF)){
}
```

```
temperature = lpdts_temperature_get();
```

函数 lpdts_flag_get

函数lpdts_flag_get描述见下表:

表 3-995. 函数 lpdts_flag_get

函数名称	lpdts_flag_get
函数原形	FlagStatus lpdts_flag_get(uint32_t flag);
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}	
flag	标志位
LPDTS_FLAG_TSR	温度传感器准备标志位
输出参数{out}	
-	-
返回值	
FlagStatus	SET或RESET

例如:

```
/* wait till TSRF flag is set */
```

```
while(RESET == lpdts_flag_get(LPDTTS_FLAG_TSR)) {
}
```

函数 lpdts_interrupt_enable

函数lpdts_interrupt_enable描述见下表:

表 3-996. 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(LPDTs_INT_EM);
```

函数 lpdts_interrupt_disable

函数lpdts_interrupt_disable描述见下表：

表 3-997. 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(LPPTS_INT_EM);
```

函数 lpdts_interrupt_flag_get

函数lpdts_interrupt_flag_get描述见下表：

表 3-998. 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LPPTS中断状态
先决条件	-
被调用函数	-
输入参数{in}	
flag	中断标志位
LPPTS_INT_FLAG_EM	测量完成中断标志位
LPPTS_INT_FLAG_LT	低阈值中断标志位
LPPTS_INT_FLAG_HT	高阈值中断标志位
LPPTS_INT_FLAG_EMA	测量完成异步中断标志位
LPPTS_INT_FLAG_LTA	低阈值异步中断标志位
LPPTS_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(LPPTS_INT_FLAG_HTA)) {
}
```

函数 lpdts_interrupt_flag_clear

函数lpdts_interrupt_flag_clear描述见下表：

表 3-999. 函数 lpdts_interrupt_flag_clear

函数名称	lpdts_interrupt_flag_clear
函数原形	void lpdts_flag_interrupt_clear(uint32_t flag);
功能描述	清除外设LPPTS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	
-	-

例如：

```
/* clear HTA flag */
lpdts_flag_clear(LPDTS_INT_FLAG-HTA);
```

3.28. MDIO

MDIO 接口可以接收完整的 MDIO 帧。章节 [3.28.1](#) 描述了 MDIO 的寄存器列表，章节 [3.28.2](#) 对 MDIO 库函数进行说明。

3.28.1. 外设寄存器说明

MDIO 寄存器列表如下表所示：

表 3-1000. MDIO 寄存器

寄存器名称	寄存器描述
MDIO_CTL	MDIO控制寄存器
MDIO_RFRM	MDIO接收帧信息寄存器
MDIO_RDATA	MDIO数据接收寄存器
MDIO_RADDR	MDIO地址接收寄存器
MDIO_TDATA	MDIO数据发送寄存器
MDIO_CFG	MDIO配置寄存器
MDIO_STAT	MDIO状态寄存器
MDIO_INTEN	MDIO中断使能寄存器
MDIO_PIN	MDIO引脚数值寄存器
MDIO_TO	MDIO超时寄存器

3.28.2. 外设库函数说明

MDIO 库函数列表如下表所示：

表 3-1001. MDIO 库函数

库函数名称	库函数描述
mdio_deinit	复位MDIO寄存器
mdio_software_reset	复位MDIO模块
mdio_init	初始化MDIO接口通讯功能
mdio_phy_length_config	配置MDIO PHY位长度
mdio_soft_phyadr_set	设置软件PHYADR值
mdio_framefield_phyadr_config	选择期望的PHYADR配置
mdio_framefield_devadd_config	配置期望的DEVADD
mdio_phy_pin_read	读取硬件PRTADR引脚数值
mdio_timeout_config	配置期望的帧位与位之间的超时
mdio_timeout_enable	使能帧位超时
mdio_timeout_disable	禁能帧位超时
mdio_op_receive	读取接收到的帧OP位信息
mdio_phyadr_receive	读取接收到的帧PHYADR位信息
mdio_devadd_receive	读取接收到的帧DEVADD位信息
mdio_ta_receive	读取接收到的帧TA位信息
mdio_data_receive	读取接收到的帧DATA位信息
mdio_address_receive	读取接收到的帧ADDRESS位信息
mdio_data_transmit	发送帧DATA位数据
mdio_flag_get	获取接收到的帧状态标志
mdio_flag_clear	清除状态标志
mdio_interrupt_enable	使能MDIO中断
mdio_interrupt_disable	禁能MDIO中断

函数 mdio_deinit

函数mdio_deinit描述见下表：

表 3-1002. 函数 mdio_deinit

函数名称	mdio_deinit
函数原型	void mdio_deinit(void);
功能描述	复位MDIO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 MDIO */
```

```
mdio_deinit ();
```

函数 mdio_software_reset

函数mdio_software_reset描述见下表：

表 3-1003. 函数 mdio_software_reset

函数名称	mdio_software_reset
函数原型	void mdio_software_reset(void);
功能描述	复位MDIO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：

```
/* reset MDIO block */
```

```
mdio_software_reset ();
```

函数 mdio_init

函数mdio_init描述见下表：

表 3-1004. 函数 mdio_init

函数名称	mdio_init
函数原型	uint32_t mdio_init(uint32_t phy_size, uint32_t phy_softaddr, uint32_t phy_sel, uint16_t devadd);
功能描述	初始化MDIO接口通讯功能
先决条件	-
被调用函数	-
输入参数{in}	
phy_size	PHY位长度
MDIO_PHY_BITS_3	PHY为3位
MDIO_PHY_BITS_5	PHY为5位
输入参数{in}	
phy_softaddr	软件PHYADR
0-31	软件PHYADR值
输入参数{in}	

phy_sel	选择PHYADR
<i>MDIO_PHYADR_HARDWARE</i>	设置期望的PHYADR为PHYPIN[4:0]
<i>MDIO_PHYADR_SOFTWARE</i>	设置期望的PHYADR为PHYSW[4:0]
1-30	期望的PHYADR选择
输入参数{in}	
devadd	设备类型
<i>DEVADD_PMA_PMD</i>	PMA/PMD设备类型
<i>DEVADD_WIS</i>	WIS设备类型
<i>DEVADD_PCS</i>	PCS设备类型
<i>DEVADD_PHY_XS</i>	PHY XS设备类型
<i>DEVADD_DTE_XS</i>	DTE XS设备类型
输出参数{out}	
-	-
返回值	
uint32_t	从机响应的PHYADR地址 0 - 31

例如：

```
/* PHY use 5 bits, select software provided address '0x0' as PHYADR, DEVADD is 0x1 */
mdio_init(MDIO_PHY_BITS_5, 0x0, MDIO_PHYADR_SOFTWARE, DEVADD_PMA_PMD);
```

函数 mdio_phy_length_config

函数mdio_phy_length_config描述见下表：

表 3-1005. 函数 mdio_phy_length_config

函数名称	mdio_phy_length_config
函数原型	void mdio_phy_length_config(uint32_t phy_bit);
功能描述	配置MDIO PHY位长度
先决条件	-
被调用函数	-
输入参数{in}	
phy_bit	PHY位长度
<i>MDIO_PHY_BITS_3</i>	PHY为3位
<i>MDIO_PHY_BITS_5</i>	PHY为5位
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* configure MDIO phy bit length */
```

```
mdio_phy_length_config (MDIO_PHY_BITS_3);
```

函数 mdio_soft_phyadr_set

函数mdio_soft_phyadr_set描述见下表：

表 3-1006. 函数 mdio_soft_phyadr_set

函数名称	mdio_soft_phyadr_set
函数原型	void mdio_soft_phyadr_set(uint32_t phy_soft);
功能描述	设置软件PHYADR值
先决条件	-
被调用函数	-
输入参数{in}	
phy_soft	软件PHYADR
0-31	软件PHYADR值
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* set the software PHYADR value */
```

```
mdio_soft_phyadr_set (1);
```

函数 mdio_framefield_phyadr_config

函数mdio_framefield_phyadr_config描述见下表：

表 3-1007. 函数 mdio_framefield_phyadr_config

函数名称	mdio_framefield_phyadr_config
函数原型	void mdio_framefield_phyadr_config(uint32_t phy_sel);
功能描述	选择期望的PHYADR配置
先决条件	-
被调用函数	-
输入参数{in}	
phy_sel	选择PHYADR
MDIO_PHYADR_HARDWARE	设置期望的PHYADR为PHYPIN[4:0]
MDIO_PHYADR_SOFTWARE	设置期望的PHYADR为PHYSW[4:0]
1-30	期望的PHYADR选择
输出参数{out}	

-	-
返回值	
-	-

例如:

```
/* select the expected PHYADR */
```

```
mdio_framefield_phyadr_config (MDIO_PHYADR_HARDWARE);
```

函数 mdio_framefield_devadd_config

函数mdio_framefield_devadd_config描述见下表:

表 3-1008. 函数 mdio_framefield_devadd_config

函数名称	mdio_framefield_devadd_config
函数原型	void mdio_framefield_devadd_config(uint16_t type);
功能描述	配置期望的DEVADD
先决条件	-
被调用函数	-
输入参数{in}	
type	设备类型
DEVADD_PMA_PMD	PMA/PMD设备类型
DEVADD_WIS	WIS设备类型
DEVADD_PCS	PCS设备类型
DEVADD_PHY_XS	PHY XS设备类型
DEVADD_DTE_XS	DTE XS设备类型
输入参数{in}	
-	-
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* configure DEVADD */
```

```
mdio_framefield_devadd_config (DEVADD_PMA_PMD);
```

函数 mdio_phy_pin_read

函数mdio_phy_pin_read描述见下表:

表 3-1009. 函数 mdio_phy_pin_read

函数名称	mdio_phy_pin_read
------	-------------------

函数原型	uint32_t mdio_phy_pin_read(void);
功能描述	读取硬件PRTADR引脚数值
先决条件	-
被调用函数	-
输入参数{in}	
-	-
输出参数{out}	
-	-
返回值	
uint32_t	PRTADR[4:0]管脚值0 – 0x1F

例如:

```
uint32_t hard_phy = 0;

/* read the hardware PRTADR[4:0] value */

hard_phy = mdio_phy_pin_read ();
```

函数 mdio_timeout_config

函数mdio_timeout_config描述见下表:

表 3-1010. 函数 mdio_timeout_config

函数名称	mdio_timeout_config
函数原型	void mdio_timeout_config(uint16_t timeout);
功能描述	配置期望的帧位与位之间的超时
先决条件	-
被调用函数	-
输入参数{in}	
timeout	帧位与位之间的超时计数
0 - 0xFFFF	超时值
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* configure the frame bit timeout */

mdio_timeout_config (0xFFFF);
```

函数 mdio_timeout_enable

函数mdio_timeout_enable描述见下表:

表 3-1011. 函数 mdio_timeout_enable

函数名称	mdio_timeout_enable
函数原型	void mdio_timeout_enable(void);
功能描述	使能帧位超时
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 MDIO frame bit timeout */
mdio_timeout_enable ();
```

函数 mdio_timeout_disable

函数mdio_timeout_disable描述见下表：

表 3-1012. 函数 mdio_timeout_disable

函数名称	mdio_timeout_disable
函数原型	void mdio_timeout_disable(void);
功能描述	禁能帧位超时
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 MDIO frame bit timeout */
mdio_timeout_disable ();
```

函数 mdio_op_receive

函数mdio_op_receive描述见下表：

表 3-1013. 函数 mdio_op_receive

函数名称	mdio_op_receive
函数原型	uint16_t mdio_op_receive(void);
功能描述	读取接收到的帧OP位信息
先决条件	-
被调用函数	-
输入参数{in}	
-	-
输出参数{out}	
-	-
返回值	
uint16_t	接收到的OP值0 – 3

例如：

```
uint16_t rx_op = 0;

/* read the received frame field OP */

rx_op = mdio_op_receive ();
```

函数 mdio_phyadr_receive

函数mdio_phyadr_receive描述见下表：

表 3-1014. 函数 mdio_phyadr_receive

函数名称	mdio_phyadr_receive
函数原型	uint16_t mdio_phyadr_receive(void);
功能描述	读取接收到的帧PHYADR位信息
先决条件	-
被调用函数	-
输入参数{in}	
-	-
输出参数{out}	
-	-
返回值	
uint16_t	接收到的PHYADR值 0 – 0x1F

例如：

```
uint16_t rx_phyadr = 0;

/* read the received frame field PHYADR */

rx_phyadr = mdio_phyadr_receive ();
```

函数 mdio_devadd_receive

函数mdio_devadd_receive描述见下表：

表 3-1015. 函数 mdio_devadd_receive

函数名称	mdio_devadd_receive
函数原型	uint16_t mdio_devadd_receive(void);
功能描述	读取接收到的帧DEVADD位信息
先决条件	-
被调用函数	-
输入参数{in}	
-	-
输出参数{out}	
-	-
返回值	
uint16_t	接收到的DEVADD值 0 – 0x1F

例如：

```
/* read the received frame field DEVADD */
```

```
uint16_t rx_devadd = 0;
```

```
rx_devadd = mdio_devadd_receive ();
```

函数 mdio_ta_receive

函数mdio_ta_receive描述见下表：

表 3-1016. 函数 mdio_ta_receive

函数名称	mdio_ta_receive
函数原型	uint16_t mdio_ta_receive(void);
功能描述	读取接收到的帧TA位信息
先决条件	-
被调用函数	-
输入参数{in}	
-	-
输出参数{out}	
-	-
返回值	
uint16_t	接收到的TA值0 – 3

例如：

```
/* read the received frame field TA */
```

```
uint16_t rx_ta = 0;
```

```
rx_ta = mdio_ta_receive ();
```


函数 mdio_data_receive

函数mdio_data_receive描述见下表：

表 3-1017. 函数 mdio_data_receive

函数名称	mdio_data_receive
函数原型	uint16_t mdio_data_receive(void);
功能描述	读取接收到的帧DATA位信息
先决条件	-
被调用函数	-
输入参数{in}	
-	-
输出参数{out}	
-	-
返回值	
uint16_t	接收到的DATA值0 – 0xFFFF

例如：

```
/* read the received frame field DATA */
```

```
uint16_t rx_data = 0;
```

```
rx_data = mdio_data_receive ();
```

函数 mdio_address_receive

函数mdio_address_receive描述见下表：

表 3-1018. 函数 mdio_address_receive

函数名称	mdio_address_receive
函数原型	uint16_t mdio_address_receive(void);
功能描述	读取接收到的帧ADDRESS位信息
先决条件	-
被调用函数	-
输入参数{in}	
-	-
输出参数{out}	
-	-
返回值	
uint16_t	接收到的ADDRESS值0 – 0xFFFF

例如：

```
/* read the received frame field ADDRESS */
```

```
uint16_t rx_address = 0;
```

```
rx_address = mdio_address_receive ();
```

函数 **mdio_data_transmit**

函数mdio_data_transmit描述见下表:

表 3-1019. 函数 **mdio_data_transmit**

函数名称	mdio_data_transmit
函数原型	void mdio_data_transmit(uint16_t data);
功能描述	发送帧DATA位数据
先决条件	-
被调用函数	-
输入参数{in}	
data	由后一个读数据帧或者读后增量地址帧发送的数据
0 - 0xFFFF	发送的数据
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* transmit data 1 */
mdio_data_transmit (0x1);
```

函数 **mdio_flag_get**

函数mdio_flag_get描述见下表:

表 3-1020. 函数 **mdio_flag_get**

函数名称	mdio_flag_get
函数原型	FlagStatus mdio_flag_get(uint32_t flag);
功能描述	获取接收到的帧状态标志
先决条件	-
被调用函数	-
输入参数{in}	
flag	要获取的状态标志
MDIO_FLAG_WFRM	写数据帧匹配标志
MDIO_FLAG_ADDRFM	地址帧匹配标志
MDIO_FLAG_RDINCFM	读后增量地址帧匹配标志
MDIO_FLAG_RDFRM	读数据帧匹配标志
MDIO_FLAG_DEVCM	DEVADD位匹配标志

<i>MDIO_FLAG_DEVN</i> <i>M</i>	DEVADD位不匹配标志
<i>MDIO_FLAG_PHY</i> <i>M</i>	PHYADR匹配标志
<i>MDIO_FLAG_PHYN</i> <i>M</i>	PHYADR不匹配标志
<i>MDIO_FLAG_TAN</i> <i>M</i>	TA不匹配标志
<i>MDIO_FLAG_TIME</i> <i>OUT</i>	超时标志
<i>MDIO_FLAG_TX_U</i> <i>NDERRUN</i>	发送下溢标志
<i>MDIO_FLAG_RX_O</i> <i>VERRUN</i>	接收上溢标志
<i>MDIO_FLAG_RBNE</i>	接收缓存非空标志
输出参数{out}	
-	-
返回值	
FlagStatus	SET或RESET

例如:

```
/* get the address frame state */
```

```
FlagStatus flag;
```

```
flag = mdio_flag_get (MDIO_FLAG_ADDRFRM);
```

函数 mdio_flag_clear

函数mdio_flag_clear描述见下表:

表 3-1021. 函数 mdio_flag_clear

函数名称	mdio_flag_clear
函数原型	void mdio_flag_clear(uint32_t flag);
功能描述	清除帧状态标志
先决条件	-
被调用函数	-
输入参数{in}	
flag	要清除的状态标志
<i>MDIO_FLAG_WRF</i> <i>RM</i>	写数据帧匹配状态标志位
<i>MDIO_FLAG_ADD</i> <i>RFRM</i>	地址帧匹配状态标志位
<i>MDIO_FLAG_RDIN</i> <i>CFRM</i>	读后增量地址帧匹配状态标志位

MDIO_FLAG_RDFR M	读数据帧匹配状态标志位
MDIO_FLAG_DEV M	DEVADD匹配状态标志位
MDIO_FLAG_DEVN M	DEVADD不匹配状态标志位
MDIO_FLAG_PHY M	PHYADR匹配状态标志位
MDIO_FLAG_PHYN M	PHYADR不匹配状态标志位
MDIO_FLAG_TAN M	TA不匹配状态标志位
MDIO_FLAG_TIME OUT	超时状态标志位
MDIO_FLAG_TX_U NDERRUN	数据发送下溢标志位
MDIO_FLAG_RX_O VERRUN	数据接收上溢标志位
MDIO_FLAG_RBNE	数据接收缓冲区非空标志位
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* clear read data buffer not empty flag */
mdio_flag_clear(MDIO_FLAG_RBNE);
```

函数 mdio_interrupt_enable

函数mdio_interrupt_enable描述见下表：

表 3-1022. 函数 mdio_interrupt_enable

函数名称	mdio_interrupt_enable
函数原型	void mdio_interrupt_enable(uint32_t interrupt);
功能描述	使能MDIO中断
先决条件	-
被调用函数	-
输入参数{in}	
interrupt	要使能的中断
MDIO_INT_WRFR M	写数据帧匹配中断
MDIO_INT_ADDRF	地址帧匹配中断

<i>RM</i>	
<i>MDIO_INT_RDINCF</i> <i>RM</i>	读后增量地址帧匹配中断
<i>MDIO_INT_RDFRM</i>	读数据帧匹配中断
<i>MDIO_INT_DEVM</i>	DEVADD位匹配中断
<i>MDIO_INT_DEVNM</i>	DEVADD位不匹配中断
<i>MDIO_INT_PHYM</i>	PHYADR匹配中断
<i>MDIO_INT_PHYNM</i>	PHYADR不匹配中断
<i>MDIO_INT_TANM</i>	TA不匹配中断
<i>MDIO_INT_TIMEO</i> <i>UT</i>	超时中断
<i>MDIO_INT_TX_UN</i> <i>DERRUN</i>	发送下溢中断
<i>MDIO_INT_RX_OV</i> <i>ERRUN</i>	接收上溢中断
<i>MDIO_INT_RBNE</i>	接收缓存非空中断
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* enable address frame interrupt */
mdio_interrupt_enable (MDIO_INT_ADDRFRM);
```

函数 mdio_interrupt_disable

函数mdio_interrupt_disable描述见下表：

表 3-1023. 函数 mdio_interrupt_disable

函数名称	mdio_interrupt_disable
函数原型	void mdio_interrupt_disable(uint32_t interrupt);
功能描述	禁能MDIO中断
先决条件	-
被调用函数	-
输入参数{in}	
interrupt	要禁能的中断
<i>MDIO_INT_WRFR</i> <i>M</i>	写数据帧匹配中断
<i>MDIO_INT_ADDRF</i> <i>RM</i>	地址帧匹配中断
<i>MDIO_INT_RDINCF</i> <i>RM</i>	读后增量地址帧匹配中断

MDIO_INT_RDFRM	读数据帧匹配中断
MDIO_INT_DEVM	DEVADD位匹配中断
MDIO_INT_DEVNM	DEVADD位不匹配中断
MDIO_INT_PHYM	PHYADR匹配中断
MDIO_INT_PHYNM	PHYADR不匹配中断
MDIO_INT_TANM	TA不匹配中断
MDIO_INT_TIMEO UT	超时中断
MDIO_INT_TX_UN DERRUN	发送下溢中断
MDIO_INT_RX_OV ERRUN	接收上溢中断
MDIO_INT_RBNE	接收缓存非空中断
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* disable address frame interrupt */
mdio_interrupt_disable (MDIO_INT_ADDRFRM);
```

3.29. 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[3.29.1](#)描述了MDMA的寄存器列表，章节[3.29.2](#)对MDMA库函数进行说明。

3.29.1. 外设寄存器描述

MDMA寄存器列表如下表所示：

表 3-1024. MDMA 寄存器

寄存器名称	寄存器描述
MDMA_GINTF	全局中断标志寄存器
MDMA_CHxSTAT0	通道x状态寄存器0
MDMA_CHxSTATC	通道x状态清除寄存器
MDMA_CHxSTAT1	通道x状态寄存器1
MDMA_CHxCTL0	通道x控制寄存器 0
MDMA_CHxCFG	通道x配置寄存器
MDMA_CHxBTCFG	通道x块传输配置寄存器
MDMA_CHxSADDR	通道x源地址寄存器

寄存器名称	寄存器描述
MDMA_CHxDADDR	通道x目的地址寄存器
MDMA_CHxMBAD DRU	通道x多块地址更新寄存器
MDMA_CHxLADDR	通道x链路地址寄存器
MDMA_CHxCTL1	通道x控制寄存器 1
MDMA_CHxMADD R	通道x掩码地址寄存器
MDMA_CHxMDATA	通道x掩码数据寄存器

3.29.2. 外设库函数说明

MDMA库函数列表如下表所示：

表 3-1025. 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
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

库函数名称	库函数描述
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 3-1026. 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
source_data_size	指定源数据大小
dest_data_dize	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
destination_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
destination_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 3-1027. 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 3-1028. 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
chxbtcf_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 3-1029. 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 3-1030. 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

成员名称	功能描述
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 3-1031. 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 3-1032. 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3-1030. 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 3-1033. 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3-1026. 结构体mdma_parameter_struct 。
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 3-1034. 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 表3-1027. 结构体mdma_multi_block_parameter_struct 。
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_parameter_struct_init(&mdma_multi_block_init_struct);
```

函数 mdma_link_node_parameter_struct_init

函数mdma_link_node_parameter_struct_init描述见下表：

表 3-1035. 函数 mdma_link_node_parameter_struct_init

函数名称	mdma_link_node_parameter_struct_init
函数原型	void mdma_link_node_parameter_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3-1028. 结构体mdma_link_node_parameter_struct 。
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_parameter_struct_init (&node);
```

函数 mdma_init

函数mdma_init描述见下表:

表 3-1036. 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3-1030. 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3-1026. 结构体mdma_parameter_struct 。
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.destination_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_dize = 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.destination_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 3-1037. 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3-1030. 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 3-1038. 函数 `mdma_multi_block_mode_config`

函数名称	<code>mdma_multi_block_mode_config</code>
函数原型	<code>void mdma_multi_block_mode_config(mdma_channel_enum channelx, uint32_t tbnun, mdma_multi_block_parameter_struct *block_init_struct);</code>
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
<code>MDMA_CHx(x=0..15)</code>	MDMA通道选择, 参考 表3-1030. 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3-1027. 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 3-1039. 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}	
block_init_struct	初始化MDMA通道所需的初始化数据, 参考 表3-1027. 结构体

	mdma_multi_block_parameter_struct 。
输入参数{in}	
init_struct	初始化MDMA通道所需的初始化数据，参考 表3-1026. 结构体 mdma_parameter_struct 。
输出参数{out}	
node	链路节点结构体，结构体成员可参考 表3-1028. 结构体 mdma_link_node_parameter_struct 。
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 3-1040. 函数 mdma_node_add

函数名称	mdma_node_add
函数原型	void mdma_node_add(mdma_link_node_parameter_struct *pre_node, mdma_link_node_parameter_struct *new_node);
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3-1028. 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3-1028. 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 3-1041. 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3-1028. 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3-1028. 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 3-1042. 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3-1030. 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 3-1043. 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3-1030. 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 3-1044. 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3-1030. 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 3-1045. 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3-1030. 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	源总线是AHB总线或TCM

<code>_TCM</code>	
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 3-1046. 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3-1030. 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
MDMA_PRIORITY_HIGH	高优先级
MDMA_PRIORITY_ULTRA_HIGH	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 3-1047. 函数 `mdma_endianness_config`

函数名称	<code>mdma_endianness_config</code>
函数原型	<code>void mdma_endianness_config(mdma_channel_enum channelx, uint32_t endianness);</code>
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
<code>MDMA_CHx(x=0..15)</code>	MDMA通道选择，参考 表3-1030. 枚举mdma_channel_enum 。
输入参数{in}	
endianness	MDMA字节顺序
<code>MDMA_LITTLE_ENDIANNESS</code>	小端
<code>MDMA_BYTE_ENDIANNESS_EXCHANGE</code>	交换半字中字节顺序
<code>MDMA_HALFWORD_ENDIANNESS_EXCHANGE</code>	交换字中半字顺序
<code>MDMA_WORD_ENDIANNESS_EXCHANGE</code>	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 3-1048. 函数 `mdma_alignment_config`

函数名称	<code>mdma_alignment_config</code>
函数原型	<code>void mdma_endianness_config(mdma_channel_enum channelx, uint32_t endianness);</code>
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3-1030. 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 3-1049. 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3-1030. 枚举mdma_channel_enum 。
输入参数{in}	
sbeat	源传输突发类型
MDMA_SOURCE_BUR ST_SINGLE	源单一传输
MDMA_SOURCE_BUR ST_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 3-1050. 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3-1030 。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 3-1051. 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3-1030. 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	源地址数据长度为64位

ASIZE_64BIT	
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 3-1052. 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3-1030. 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_DATASIZE_16BIT	目的地址数据长度为16位
MDMA_DESTINATION_DATASIZE_32BIT	目的地址数据长度为32位
MDMA_DESTINATION_DATASIZE_64BIT	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 3-1053. 函数 `mdma_source_increment_config`

函数名称	<code>mdma_source_increment_config</code>
函数原型	<code>void mdma_source_increment_config(mdma_channel_enum channelx, uint32_t sinc);</code>
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
<code>MDMA_CHx(x=0..15)</code>	MDMA通道选择，参考 表3-1030. 枚举mdma_channel_enum 。
输入参数{in}	
sinc	源地址增长模式
<code>MDMA_SOURCE_INC REASE_DISABLE</code>	无增长
<code>MDMA_SOURCE_INC REASE_8BIT</code>	源地址按8位增长
<code>MDMA_SOURCE_INC REASE_16BIT</code>	源地址按16位增长
<code>MDMA_SOURCE_INC REASE_32BIT</code>	源地址按32位增长
<code>MDMA_SOURCE_INC REASE_64BIT</code>	源地址按64位增长
<code>MDMA_SOURCE_DEC REASE_8BIT</code>	源地址按8位减少
<code>MDMA_SOURCE_DEC REASE_16BIT</code>	源地址按16位减少
<code>MDMA_SOURCE_DEC REASE_32BIT</code>	源地址按32位减少
<code>MDMA_SOURCE_DEC REASE_64BIT</code>	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 3-1054. 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3-1030. 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_DECREASE_16BIT	目的地址按16位减少
MDMA_DESTINATION_DECREASE_32BIT	目的地址按32位减少
MDMA_DESTINATION_DECREASE_64BIT	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 3-1055. 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3-1030. 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 3-1056. 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3-1030. 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 3-1057. 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3-1030. 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 3-1058. 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3-1030. 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 3-1059. 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3-1030. 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 3-1060. 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3-1030. 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
R	

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 3-1061. 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3-1030. 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 3-1062. 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3-1030. 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_TCF	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 3-1063. 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3-1030. 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 3-1064. 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3-1030. 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 3-1065. 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3-1030. 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
MDMA_INT_BTC	块传输完成中断
MDMA_INT_TC	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 3-1066. 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
MDMA_CHx(x=0..15)	MDMA通道选择, 参考 表3-1030. 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	
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 3-1067. 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3-1030. 枚举mdma_channel_enum 。
输入参数{in}	
int_flag	MDMA中断标志
MDMA_INT_FLAG_ERR	传输错误中断标志
MDMA_INT_FLAG_CHANNEL_TCF	通道传输完成中断标志
MDMA_INT_FLAG_BLOCK_TCF	多块传输完成中断标志
MDMA_INT_FLAG_PACKET_TCF	块传输完成中断标志
MDMA_INT_FLAG_BUFFER_TCF	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);
```

3.30. 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[3.30.1](#)描述了MDIO的寄存器列表, 章节[3.30.2](#)对MDIO库函数进行说明。

3.30.1. 外设寄存器描述

OSPI寄存器列表如下表所示:

表 3-1068. 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

寄存器名称	寄存器描述
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

3.30.2. 外设库函数说明

OSPI库函数列表如下表所示：

表 3-1069. OSPI 库函数

库函数名称	库函数描述
ospi_deinit	复位OSPI
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

库函数名称	库函数描述
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
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 3-1070. 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 3-1071. 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)
nbdta	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 3-1072. 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 3-1073. 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

成员名称	功能描述
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 3-1074. 函数 ospi_deinit

函数名称	ospi_deinit
函数原形	void ospi_deinit(uint32_t ospi_periph);
功能描述	复位OSPI
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 OSPI peripheral */
```

```
ospi_deinit();
```

函数 ospi_struct_init

函数ospi_struct_init描述见下表：

表 3-1075. 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3-1070. 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 3-1076. 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3-1070. 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;
```

```
ospi_init(OSPI0, &ospi_struct);
```

函数 `ospi_enable`

函数`ospi_enable`描述见下表：

表 3-1077. 函数 `ospi_enable`

函数名称	<code>ospi_enable</code>
函数原形	<code>void ospi_enable(uint32_t ospi_periph);</code>
功能描述	使能OSPI
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：

```
/* enable OSPI0 */
ospi_enable(OSPI0);
```

函数 `ospi_disable`

函数`ospi_disable`描述见下表：

表 3-1078. 函数 `ospi_disable`

函数名称	<code>ospi_disable</code>
函数原形	<code>void ospi_disable(uint32_t ospi_periph);</code>
功能描述	失能OSPI
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 OSPI0 */
ospi_disable(OSPI0);
```

函数 ospi_device_memory_type_config

函数ospi_device_memory_type_config描述见下表:

表 3-1079. 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 3-1080. 函数 ospi_device_memory_size_config

函数名称	ospi_device_memory_size_config
函数原形	void ospi_device_memory_size_config(uint32_t ospi_periph, uint32_t mesz);
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}	
ospi_periph	OSPIx (x=0,1)
输入参数{in}	

mesz	设备内存大小
<i>OSPI_MESE_x_BYTES</i>	x = 2, 4, 8, ..., 512, 1024
<i>OSPI_MESE_x_KBS</i>	x = 2, 4, 8, ..., 512, 1024
<i>OSPI_MESE_x_MBS</i>	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 3-1081. 函数 ospi_functional_mode_config

函数名称	ospi_functional_mode_config
函数原形	void ospi_functional_mode_config(uint32_t ospi_periph, uint32_t fmod);
功能描述	选择功能模式
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}	
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 3-1082. 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
OSPI_AUTOMATIC_STOP_MATCH	状态轮询模式在匹配时停止
输入参数{in}	
mode	OSPI匹配模式
OSPI_MATCH_MODE_AND	状态轮询匹配与模式
OSPI_MATCH_MODE_OR	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 3-1083. 函数 **ospi_status_mask_config**

函数名称	ospi_status_mask_config
------	-------------------------

函数原形	void ospi_status_mask_config(uint32_t ospi_periph, uint32_t mask);
功能描述	配置状态屏蔽
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}	
mask	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 3-1084. 函数 ospi_status_match_config

函数名称	ospi_status_match_config
函数原形	void ospi_status_match_config(uint32_t ospi_periph, uint32_t match);
功能描述	配置状态匹配
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}	
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 3-1085. 函数 `ospi_interval_cycle_config`

函数名称	<code>ospi_interval_cycle_config</code>
函数原形	<code>void ospi_interval_cycle_config(uint32_t ospi_periph, uint16_t interval);</code>
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 3-1086. 函数 `ospi_fifo_level_config`

函数名称	<code>ospi_fifo_level_config</code>
函数原形	<code>void ospi_fifo_level_config(uint32_t ospi_periph, uint32_t ftl)</code>
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
<code>OSPI_FIFO_THRES_HOLD_x</code>	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 3-1087. 函数 `ospi_chip_select_high_cycle_config`

函数名称	<code>ospi_chip_select_high_cycle_config</code>
函数原形	<code>void ospi_chip_select_high_cycle_config(uint32_t ospi_periph, uint32_t cshc);</code>
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}	
<code>ospi_periph</code>	OSPIx (x=0,1)
输入参数{in}	
<code>cshc</code>	OSPI片选高电平周期
<code>OSPI_CS_HIGH_TIME_x_CYCLE</code>	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 3-1088. 函数 `ospi_prescaler_config`

函数名称	<code>ospi_prescaler_config</code>
函数原形	<code>void ospi_prescaler_config(uint32_t ospi_periph, uint32_t psc);</code>
功能描述	配置OSPI预分频
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>psc</code>	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 3-1089. 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 3-1090. 函数 ospi_delay_hold_cycle_config

函数名称	ospi_delay_hold_cycle_config
函数原形	void ospi_delay_hold_cycle_config(uint32_t ospi_periph, uint32_t dehq);
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}	
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无延迟保持周期
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 */
```

```
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 3-1091. 函数 ospi_sample_shift_config

函数名称	ospi_sample_shift_config
函数原形	void ospi_sample_shift_config(uint32_t ospi_periph, uint32_t ssample);
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 3-1092. 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 3-1093. 函数 ospi_instruction_config

函数名称	ospi_instruction_config
函数原形	void ospi_instruction_config(uint32_t ospi_periph, uint32_t imod, uint32_t inssz, uint32_t instruction);
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
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
输入参数{in}	
输入参数{in}	
inssz	OSPI指令大小
OSPI_INSTRUCTION_8_BITS	8位指令
OSPI_INSTRUCTION_16_BITS	16位指令
OSPI_INSTRUCTION_24_BITS	24位指令
OSPI_INSTRUCTION_32_BITS	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 3-1094. 函数 ospi_address_config

函数名称	ospi_address_config
函数原形	void ospi_address_config(uint32_t ospi_periph, uint32_t addrmod, uint32_t addrdrtr, uint32_t addrsz, uint32_t addr);
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
OSPI_ADDRESS_N	无地址模式

ONE	
OSPI_ADDRESS_1 _LINE	单线地址模式
OSPI_ADDRESS_2 _LINES	双线地址模式
OSPI_ADDRESS_4 _LINES	四线地址模式
OSPI_ADDRESS_8 _LINES	八线地址模式
输入参数{in}	
addrdtr	OSPI地址DTR模式
OSPI_ADDRDTR_ MODE_DISABLE	地址阶段禁用DTR
OSPI_ADDTR_MO DE_ENABLE	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_1 6_BITS	16位地址
OSPI_ADDRESS_2 4_BITS	24位地址
OSPI_ADDRESS_3 2_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 address mode */
```

```
ospi_address_config(OSPI0, OSPI_ADDRESS_8_LINES,
OSPI_ADDRDTR_MODE_DISABLE, OSPI_ADDRESS_32_BITS, 0xFFFFFFFF);
```

函数 `ospi_alternate_bytes_config`

函数 `ospi_alternate_bytes_config` 描述见下表：

表 3-1095. 函数 `ospi_alternate_bytes_config`

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

函数原形	void ospi_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 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 3-1096. 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
OSPI_DATA_NONE	无数据模式
OSPI_DATA_1_LINE	单线数据模式
OSPI_DATA_2_LINES	双线数据模式
OSPI_DATA_4_LINES	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述见下表：

表 3-1097. 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 3-1098. 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 3-1099. 函数 `ospi_dma_enable`

函数名称	<code>ospi_dma_enable</code>
函数原形	<code>void ospi_dma_enable(uint32_t ospi_periph);</code>
功能描述	使能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:

```
/* enable OSPI DMA */
```

```
ospi_dma_enable(OSPI0);
```

函数 `ospi_dma_disable`

函数 `ospi_dma_disable` 描述见下表:

表 3-1100. 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 3-1101. 函数 **ospi_wrap_size_config**

函数名称	ospi_wrap_size_config
函数原形	void ospi_wrap_size_config(uint32_t ospi_periph, uint32_t wpsz);
功能描述	配置回卷大小
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}	
wpsz	OSPI回卷大小设置
OSPI_DIRECT	外部存储器设备不支持回卷读取
OSPI_WRAP_16BYTES	外部存储器设备支持16字节回卷大小
OSPI_WRAP_32BYTES	外部存储器设备支持32字节回卷大小
OSPI_WRAP_64BYTES	外部存储器设备支持64字节回卷大小
OSPI_WRAP_128BYTES	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 3-1102. 函数 **ospi_wrap_instruction_config**

函数名称	ospi_wrap_instruction_config
函数原形	void ospi_wrap_instruction_config(uint32_t ospi_periph, uint32_t imod, uint32_t inssz, uint32_t instruction);
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
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
输入参数{in}	
inssz	OSPI指令大小
OSPI_INSTRUCTION_8_BITS	8位指令
OSPI_INSTRUCTION_16_BITS	16位指令
OSPI_INSTRUCTION_24_BITS	24位指令
OSPI_INSTRUCTION_32_BITS	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 3-1103. 函数 ospi_wrap_address_config

函数名称	ospi_wrap_address_config
函数原形	void ospi_wrap_address_config (uint32_t ospi_periph, uint32_t addrmod, uint32_t addrdtr, uint32_t addrsz, uint32_t addr);

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
OSPI_ADDRESS_N ONE	无地址模式
OSPI_ADDRESS_1 _LINE	单线地址模式
OSPI_ADDRESS_2 _LINES	双线地址模式
OSPI_ADDRESS_4 _LINES	四线地址模式
OSPI_ADDRESS_8 _LINES	八线地址模式
输入参数{in}	
addrdtr	OSPI地址DTR模式
OSPI_ADDRDTR_ MODE_DISABLE	地址阶段禁用DTR
OSPI_ADDTR_MO DE_ENABLE	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_1 6_BITS	16位地址
OSPI_ADDRESS_2 4_BITS	24位地址
OSPI_ADDRESS_3 2_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 3-1104. 函数 `ospi_wrap_alternate_bytes_config`

函数名称	<code>ospi_wrap_alternate_bytes_config</code>
函数原形	<code>void ospi_wrap_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
<code>OSPI_ALTERNATE_BYTES_NONE</code>	无交替字节模式
<code>OSPI_ALTERNATE_BYTES_1_LINE</code>	单线交替字节模式
<code>OSPI_ALTERNATE_BYTES_2_LINES</code>	双线交替字节模式
<code>OSPI_ALTERNATE_BYTES_4_LINES</code>	四线交替字节模式
<code>OSPI_ALTERNATE_BYTES_8_LINES</code>	八线交替字节模式
输入参数{in}	
abdtr	OSPI交替字节DTR模式
<code>OSPI_ABDTR_MODE_DISABLE</code>	交替字节阶段禁用DTR
<code>OSPI_ABDTR_MODE_ENABLE</code>	交替字节阶段使能DTR
输入参数{in}	
altesz	OSPI交替字节大小
<code>OSPI_ALTERNATE_BYTES_8_BITS</code>	8位交替字节
<code>OSPI_ALTERNATE_BYTES_16_BITS</code>	16位交替字节
<code>OSPI_ALTERNATE_BYTES_24_BITS</code>	24位交替字节
<code>OSPI_ALTERNATE_BYTES_32_BITS</code>	32位交替字节

<code>_BYTES_32_BITS</code>	
输入参数{in}	
<code>alte</code>	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 3-1105. 函数 `ospi_wrap_data_config`

函数名称	<code>ospi_wrap_data_config</code>
函数原形	<code>void ospi_wrap_data_config(uint32_t ospi_periph, uint32_t datamod, uint32_t dadtr);</code>
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}	
<code>ospi_periph</code>	OSPIx (x=0,1)
输入参数{in}	
<code>datamod</code>	OSPI数据模式
<code>OSPI_DATA_NONE</code>	无数据模式
<code>OSPI_DATA_1_LINE</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}	
<code>dadtr</code>	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 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 3-1106. 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 3-1107. 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
<i>OSPI_DELAY_HOLD_NONE</i>	OSPI无延迟保持
<i>OSPI_DELAY_HOLD_QUARTER_CYCLE</i>	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 3-1108. 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
<i>OSPI_SAMPLE_SHIFTING_NONE</i>	OSPI无采样移位
<i>OSPI_SAMPLE_SHIFTING_HALF_CYCLE</i>	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 3-1109. 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
输入参数{in}	
inssz	OSPI指令大小
OSPI_INSTRUCTION_8_BITS	8位指令
OSPI_INSTRUCTION_16_BITS	16位指令
OSPI_INSTRUCTION_24_BITS	24位指令
OSPI_INSTRUCTION_32_BITS	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 3-1110. 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
OSPI_ADDRESS_NONE	无地址模式
OSPI_ADDRESS_1_LINE	单线地址模式
OSPI_ADDRESS_2_LINES	双线地址模式
OSPI_ADDRESS_4_LINES	四线地址模式
OSPI_ADDRESS_8_LINES	八线地址模式
输入参数{in}	
addrdtr	OSPI地址DTR模式
OSPI_ADDRDTR_MODE_DISABLE	地址阶段禁用DTR
OSPI_ADDTR_MODE_ENABLE	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_2 4_BITS	24位地址
OSPI_ADDRESS_3 2_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 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 3-1111. 函数 `ospi_write_alternate_bytes_config`

函数名称	ospi_write_alternate_bytes_config
函数原形	void ospi_write_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 3-1112. 函数 ospi_write_data_config

函数名称	ospi_write_data_config
函数原形	void ospi_write_data_config(uint32_t ospi_periph, uint32_t datamod, uint32_t dadtr);
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
OSPI_DATA_NONE	无数据模式
OSPI_DATA_1_LINE	单线数据模式

<i>E</i>	
<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 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 3-1113. 函数 ospi_write_dummy_cycles_config

函数名称	ospi_write_dummy_cycles_config
函数原形	void ospi_write_dummy_cycles_config(uint32_t ospi_periph, uint32_t dumyc);
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}	
ospi_periph	OSPIx (x=0,1)
输入参数{in}	
dumyc	空指令周期数
<i>OSPI_DUMYC_CYCLES_x</i>	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 3-1114. 函数 **ospi_write_dummy_cycles_config**

函数名称	ospi_write_dummy_cycles_config
函数原形	void ospi_command_config(uint32_t ospi_periph, ospi_parameter_struct *ospi_struct, ospi_regular_cmd_struct *cmd_struct);
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}	
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 3-1070. 结构体 ospi_parameter_struct
输入参数{in}	
cmd_struct	OSPI 命令, 结构体成员参考 表 3-1071. 结构体 ospi_regular_cmd_struct
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 3-1115. 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 3-1116. 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 3-1117. 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 3-1070. 结构体 <i>ospi_parameter_struct</i>
输入参数{in}	
ospi_autopolling_struct	OSPI 状态轮询结构体, 结构体成员参考 表 3-1072. 结构体 <i>ospi_autopolling_struct</i>
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 3-1118. 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 3-1119. 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 3-1120. 函数 ospi_fifo_level_get

函数名称	ospi_fifo_level_get
函数原形	uint32_t ospi_fifo_level_get(uint32_t ospi_periph);
功能描述	get OSPI fifo level
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	
uint32_t	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 3-1121. 函数 ospi_flag_get

函数名称	ospi_flag_get
函数原形	FlagStatus ospi_flag_get(uint32_t ospi_periph, uint32_t flag);
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}	
ospi_periph	OSPIx (x=0,1)
输入参数{in}	
flag	OSPI状态标志
OSPI_FLAG_TERR	传输错误标志
OSPI_FLAG_TC	传输完成标志
OSPI_FLAG_FT	FIFO阈值标志
OSPI_FLAG_SM	状态匹配
OSPI_FLAG_BUSY	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 3-1122. 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 3-1123. 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_FIT	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 3-1124. 函数 `ospi_interrupt_flag_clear`

函数名称	<code>ospi_interrupt_flag_clear</code>
函数原形	<code>void ospi_interrupt_flag_clear(uint32_t ospi_periph, uint32_t int_flag);</code>
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
<code>OSPI_INT_FLAG_TERR</code>	传输错误中断标志
<code>OSPI_INT_FLAG_TC</code>	传输完成中断标志
<code>OSPI_INT_FLAG_SM</code>	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);
```

3.31. OSPI

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[3.31.1](#)描述了MDIO的寄存器列表，章节[3.31.2](#)对MDIO库函数进行说明。

3.31.1. 外设寄存器描述

OSPIM寄存器列表如下表所示：

表 3-1125. 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
OSPIM_PCFG1	OSPIM端口配置寄存器1

3.31.2. 外设库函数说明

OSPIM库函数列表如下表所示：

表 3-1126. OSPIM 库函数

库函数名称	库函数描述
ospim_deinit	复位OSPIM
ospim_port_sck_config	配置端口的SCK
ospim_port_sck_source_select	选择端口的SCK源
ospim_port_csn_config	配置端口的CSN
ospim_port_csn_source_select	选择端口的CSN源
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 3-1127. 函数 ospim_deinit

函数名称	ospim_deinit
函数原形	void ospim_deinit();
功能描述	复位OSPIM
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 OSPIM peripheral */
```

ospim_deinit();

函数 ospim_port_sck_config

函数ospim_port_sck_config描述见下表:

表 3-1128. 函数 ospim_port_sck_config

函数名称	ospim_port_sck_config
函数原形	void ospim_port_sck_config(uint8_t port, uint32_t sckconfig);
功能描述	配置端口的SCK
先决条件	-
被调用函数	-
输入参数{in}	
port	端口号
OSPIM_PORT0	端口0
OSPIM_PORT1	端口1
输入参数{in}	
sckconfig	使能/失能SCK
OSPIM_PORT_SCK_DISABLE	失能SCK
OSPIM_PORT_SCK_ENABLE	使能SCK
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* configurate SCK for port */
```

```
ospim_port_sck_config(OSPIM_PORT0, OSPIM_PORT_SCK_ENABLE);
```

函数 ospim_port_sck_source_select

函数ospim_port_sck_source_select描述见下表:

表 3-1129. 函数 ospim_port_sck_source_select

函数名称	ospim_port_sck_source_select
函数原形	void ospim_port_sck_source_select(uint8_t port, uint32_t sck_source);
功能描述	选择端口的SCK源
先决条件	-
被调用函数	-
输入参数{in}	
port	端口号
OSPIM_PORT0	端口0

OSPIM_PORT1	端口1
输入参数{in}	
sck_source	SCK源
OSPIM_SCK_SOU RCE_OSPI0_SCK	SCK的源为OSPI0_SCK
OSPIM_SCK_SOU RCE_OSPI1_SCK	SCKSCK的源为OSPI1_SCK
输出参数{out}	
-	-
返回值	
-	-

例如:

Example:

```
/* select source of SCK for port */
```

```
ospim_port_sck_source_select(OSPIM_PORT0, OSPIM_SCK_SOURCE_OSPI0_SCK);
```

函数 ospim_port_csn_config

函数ospim_port_csn_config描述见下表:

表 3-1130. 函数 ospim_port_csn_config

函数名称	ospim_port_csn_config
函数原形	void ospim_port_csn_config(uint8_t ospim_port, 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}	
port	端口号
OSPIM_PORT0	端口0
OSPIM_PORT1	端口1
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:


```
/* configurate CSN for port */
```

```
ospim_port_csn_config(OSPIM_PORT0, OSPIM_PORT_CSN_ENABLE);
```

函数 ospim_port_csn_source_select

函数ospim_port_csn_source_select描述见下表：

表 3-1131. 函数 ospim_port_csn_source_select

函数名称	ospim_port_csn_source_select
函数原形	void ospim_port_csn_source_select(uint8_t port, uint32_t csn_source);
功能描述	select source of CSN for port
先决条件	-
被调用函数	-
输入参数{in}	
port	端口号
OSPIM_PORT0	端口0
OSPIM_PORT1	端口1
输入参数{in}	
csn_source	CSN源
OSPIM_CSN_SOURCE_OSPI0_CSN	CSN的源为OSPI0_CSN
OSPIM_CSN_SOURCE_OSPI1_CSN	CSN的源为OSPI1_CSN
输出参数{out}	
-	-
返回值	
-	-

例如：

Example:

```
/* select source of CSN for port */
```

```
ospim_port_csn_source_select(OSPIM_PORT0, OSPIM_CSN_SOURCE_OSPI0_CSN);
```

函数 ospim_port_io3_0_config

函数ospim_port_io3_0_config描述见下表：

表 3-1132. 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(uint8_t port, 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}	
port	端口号
<i>OSPIM_PORT0</i>	端口0
<i>OSPIM_PORT1</i>	端口1
输入参数{in}	
ioconfig	使能/失能IO[3:0]
<i>OSPIM_IO_LOW_DISABLE</i>	失能IO[3:0]
<i>OSPIM_IO_LOW_ENABLE</i>	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_PORT0, 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 3-1133. 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(uint8_t port, 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}	
port	端口号
<i>OSPIM_PORT0</i>	端口0
<i>OSPIM_PORT1</i>	端口1
输入参数{in}	
csn_source	IO[3:0]源
<i>OSPIM_SRCPLIO_OSPIO_IO_LOW</i>	选择OSPI0_IO[3:0]
<i>OSPIM_SRCPLIO_OSPIO_IO_HIGH</i>	选择OSPI0_IO[7:4]
<i>OSPIM_SRCPLIO_OSPI1_IO_LOW</i>	选择OSPI1_IO[3:0]
<i>OSPIM_SRCPLIO_OSPI1_IO_HIGH</i>	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_PORT0, OSPIM_IO_LOW_ENABLE);
```

函数 ospim_port_io7_4_config

函数ospim_port_io7_4_config描述见下表：

表 3-1134. 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(uint8_t port, 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}	
port	端口号
OSPIM_PORT0	端口0
OSPIM_PORT1	端口1
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_PORT0, 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 3-1135. 函数 `ospim_port_io7_4_source_select`

函数名称	<code>ospim_port_io7_4_source_select</code>
函数原形	<code>void ospim_port_io7_4_source_select(uint8_t port, uint32_t io_source);</code>
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}	
port	端口号
<code>OSPIM_PORT0</code>	端口0
<code>OSPIM_PORT1</code>	端口1
输入参数{in}	
csn_source	source of IO[3:0]
<code>OSPIM_SRCPHIO_OSPI0_IO_LOW</code>	选择OSPI0_IO[3:0]
<code>OSPIM_SRCPHIO_OSPI0_IO_HIGH</code>	选择OSPI0_IO[7:4]
<code>OSPIM_SRCPHIO_OSPI1_IO_LOW</code>	选择OSPI1_IO[3:0]
<code>OSPIM_SRCPHIO_OSPI1_IO_HIGH</code>	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_PORT0, OSPIM_SRCPHIO_OSPI0_IO_LOW);
```

3.32. 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 [3.32.1](#) 描述了 NVIC 和 SysTick 的寄存器列表，章节 [3.32.2](#) 对 MISC 库函数进行说明。

3.32.1. 外设寄存器说明

表 3-1136. NVIC 寄存器

寄存器名称	寄存器描述
<code>ISER⁽¹⁾</code>	中断使能寄存器
<code>ICER⁽¹⁾</code>	中断禁能寄存器
<code>ISPR⁽¹⁾</code>	中断挂起寄存器

寄存器名称	寄存器描述
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
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

寄存器名称	寄存器描述
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 3-1137. 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

3.32.2. 外设库函数说明

MISC库函数列表如下表所示:

表 3-1138. 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
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 3-1139. 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

成员名称	功能描述
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 3-1140. 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
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

成员名称	功能描述
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
SDIO0_IRQn	SDIO0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_DACOVRR_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
ENET0_IRQn	以太网0全局中断
ENET0_WKUP_IRQn	连接到EXTI线的以太网0唤醒中断
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
DCI_IRQn	DCI全局中断
CAU_IRQn	CAU全局中断

成员名称	功能描述
HAU_TRNG_IRQn	HAU和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
SAI0_IRQn	SAI0全局中断
TLI_IRQn	TLI全局中断
TLI_ER_IRQn	TLI错误中断
IPA_IRQn	IPA全局中断
SAI1_IRQn	SAI1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
RSPDIF_IRQn	RSPDIF全局中断
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
SAI2_IRQn	SAI2全局中断
TIMER14_IRQn	TIMER14全局中断
TIMER15_IRQn	TIMER15全局中断
TIMER16_IRQn	TIMER16全局中断
MDIO_IRQn	MDIO全局中断
MDMA_IRQn	MDMA全局中断
SDIO1_IRQn	SDIO1全局中断
HWSEM_IRQn	HWSEM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
RTDEC0_IRQn	RTDEC0全局中断
RTDEC1_IRQn	RTDEC1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

成员名称	功能描述
TIMER23_IRQn	TIMER23全局中断
TIMER30_IRQn	TIMER30全局中断
TIMER31_IRQn	TIMER31全局中断
TIMER40_IRQn	TIMER40全局中断
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
ENET1_IRQn	以太网1全局中断
ENET1_WKUP_IRQn	连接到EXTI线的以太网1唤醒中断
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

成员名称	功能描述
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
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
TIMER30_DEC_IRQn	TIMER30译码器检测中断
TIMER31_DEC_IRQn	TIMER31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 3-1141. 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 3-1142. 函数 nvic_irq_enable

函数名称	nvic_irq_enable
函数原型	void nvic_irq_enable(uint8_t nvic_irq, uint8_t nvic_irq_pre_priority, uint8_t nvic_irq_sub_priority);
功能描述	使能NVIC中断
先决条件	-
被调用函数	nvic_priority_group_set
输入参数{in}	
nvic_irq	NVIC中断，参考枚举类型 表3-1140. 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 3-1143. 函数 nvic_irq_disable

函数名称	nvic_irq_disable
函数原型	void nvic_irq_disable(uint8_t 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3-1140. 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 3-1144. 函数 nvic_vector_table_set

函数名称	nvic_vector_table_set
函数原型	void nvic_vector_table_set(uint32_t nvic_vect_tab, uint32_t offset);
功能描述	设置向量表基地址
先决条件	-
被调用函数	-
输入参数{in}	
nvic_vect_tab	RAM或者FLASH基地址
NVIC_VECTTAB_RAM	RAM基地址
NVIC_VECTTAB_FLASH	FLASH基地址
输入参数{in}	
offset	向量表偏移量 (向量表地址=基地址+偏移量)
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* set vector table address = NVIC_VECTTAB_FLASH + 0x200 */
```

```
nvic_vector_table_set(NVIC_VECTTAB_FLASH, 0x200);
```

函数 system_lowpower_set

函数system_lowpower_set描述见下表:

表 3-1145. 函数 system_lowpower_set

函数名称	system_lowpower_set
函数原型	void system_lowpower_set(uint8_t lowpower_mode);
功能描述	设置系统低功耗模式状态
先决条件	-
被调用函数	-

输入参数{in}	
lowpower_mode	系统低功耗模式的状态
SCB_LPM_SLEEP_EXIT_ISR	该位为1时，退出ISR时一直处于低功耗模式
SCB_LPM_DEEPSLEEP	该位为1时，系统处于deep sleep模式
SCB_LPM_WAKE_BY_ALL_INT	该位为1时，低功耗模式可以被所有中断唤醒（无论中断是否被使能）
输出参数{out}	
-	-
返回值	
-	-

例如：

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

```
system_lowpower_set(SCB_LPM_SLEEP_EXIT_ISR);
```

函数 **system_lowpower_reset**

函数system_lowpower_reset描述见下表：

表 3-1146. 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 3-1147. 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
SYSTICK_CLKSOURCE_CKSYS	systick时钟源为CK_SYS时钟
SYSTICK_CLKSOURCE_CKSYS_DIV2	systick时钟源为CK_SYS时钟的2分频
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* systick clock source is CK_SYS/2 */
```

```
systick_clksource_set(SYSTICK_CLKSOURCE_CKSYS_DIV2);
```

函数 mpu_region_struct_para_init

函数mpu_region_struct_para_init描述见下表:

表 3-1148. 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3-1139. 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 3-1149. 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3-1139. 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 3-1150. 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();
```

3.33. 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 [3.33.1](#) 描述了 PMU 的寄存器列表，章节 [3.33.2](#) 对 PMU 库函数进行说明。

3.33.1. 外设寄存器说明

PMU 寄存器列表如下表所示：

表 3-1151. 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

3.33.2. 外设库函数说明

PMU 库函数列表如下表所示：

表 3-1152. 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_avd_select	选择模拟电压检测器阈值
pmu_avd_enable	打开模拟电压检测器
pmu_avd_disable	关闭模拟电压检测器
pmu_cvd_enable	打开V _{0.9V} 内核电压检测器

库函数名称	库函数描述
pmu_cvd_disable	关闭V _{0.9V} 内核电压检测器
pmu_ldo_output_select	选择LDO V _{0.9V} 供电电压
pmu_sldo_output_select	选择LDO深度模式V _{0.9V} 供电电压
pmu_vbat_charging_select	选择VBAT电池充电电阻
pmu_vbat_charging_enable	使能VBAT电池充电
pmu_vbat_charging_disable	失能VBAT电池充电
pmu_vbat_temp_monitor_enable	使能VBAT和温度检测器
pmu_vbat_temp_monitor_disable	失能VBAT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_smpps_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 3-1153. 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 3-1154. 函数 pmu_lvd_select

函数名称	pmu_lvd_select
函数原型	void pmu_lvd_select(uint32_t lvdt_n);
功能描述	选择低压检测阈值
先决条件	-
被调用函数	-
输入参数{in}	
lvdt_n	电压阈值
PMU_LVDT_0	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 3-1155. 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 3-1156. 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_avd_select

函数pmu_avd_select描述见下表：

表 3-1157. 函数 pmu_avd_select

函数名称	pmu_avd_select
函数原型	void pmu_avd_select(uint32_t avdt_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_avd_select(PMU_VAVDVC_3);
```

函数 pmu_avd_enable

函数pmu_avd_enable描述见下表：

表 3-1158. 函数 pmu_avd_enable

函数名称	pmu_avd_enable
函数原型	void pmu_avd_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_avd_enable();
```

函数 pmu_avd_disable

函数pmu_avd_disable描述见下表：

表 3-1159. 函数 pmu_avd_disable

函数名称	pmu_avd_disable
------	-----------------

函数原型	void pmu_avd_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_avd_disable();
```

函数 pmu_cvd_enable

函数pmu_cvd_enable描述见下表：

表 3-1160. 函数 pmu_cvd_enable

函数名称	pmu_cvd_enable
函数原型	void pmu_cvd_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_cvd_enable();
```

函数 pmu_cvd_disable

函数pmu_cvd_disable描述见下表：

表 3-1161. 函数 pmu_cvd_disable

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

功能描述	关闭V0.9V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_cvd_disable();
```

函数 pmu_ldo_output_select

函数pmu_ldo_output_select描述见下表：

表 3-1162. 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 3-1163. 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}	
slido_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_slido_output_select(PMU_SLDOVS_3);
```

函数 **pmu_vbat_charging_select**

函数pmu_vbat_charging_select描述见下表:

表 3-1164. 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 3-1165. 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 3-1166. 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 3-1167. 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 3-1168. 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 3-1169. 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 3-1170. 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 3-1171. 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 3-1172. 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 3-1173. 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	V0.9V电源域由LDO供电
PMU_DIRECT_SM PS_SUPPLY	V0.9V电源域由SMPS供电
PMU_BYPASS	SMPS使能、LDO旁路，外部电源供电V0.9V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 3-1174. 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 3-1175. 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 3-1176. 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 3-1177. 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 3-1178. 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 3-1179. 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 3-1180. 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 3-1181. 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 3-1182. 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 3-1183. 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(uint32_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_enter_deepsleep_wait_time_config`

函数 `pmu_enter_deepsleep_wait_time_config` 描述见下表：

表 3-1184. 函数 `pmu_enter_deepsleep_wait_time_config`

函数名称	<code>pmu_enter_deepsleep_wait_time_config</code>
函数原型	<code>void pmu_enter_deepsleep_wait_time_config(uint32_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 3-1185. 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 3-1186. 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);
```

3.34. 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 [3.34.1](#) 描述了 RAMECCMU 的寄存器列表，章节 [3.34.2](#) 对 RAMECCMU 库函数进行说明。

3.34.1. 外设寄存器描述

RAMECCMU 寄存器列表如下表所示:

表 3-1187. 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

3.34.2. 外设库函数说明

RAMECCMU 库函数列表如下表所示:

表 3-1188. 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 3-1189. 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 3-1190. 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 3-1191. 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 3-1192. 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 3-1193. 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 3-1194. 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 3-1195. 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 3-1196. 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 3-1197. 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_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}	
monitor_interrupt	监视器中断
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
RAMECCMU_INT_ECC_ERROR_LATCHING	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 3-1198. 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 3-1199. 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 3-1200. 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 3-1201. 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 3-1202. 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);
```

3.35. 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 [3.35.1](#) 描述了 RCU 的寄存器列表，章节 [3.35.2](#) 对 RCU 库函数进行说明。

3.35.1. 外设寄存器说明

RCU寄存器列表如下表所示：

表 3-1203. 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

3.35.2. 外设库函数说明

RCU库函数列表如下表所示：

表 3-1204. 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_sdio_clock_config	配置SDIO时钟源时钟

库函数名称	库函数描述
rcu_deepsleep_wakeup_sys_clock_config	配置深度睡眠模式下系统唤醒时钟源选择
rcu_tli_clock_div_config	配置TLI时钟分频系数
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_sai_clock_config	配置SAI时钟源时钟
rcu_sai2_block_clock_config	配置SAI2时钟源时钟
rcu_rspdif_clock_config	配置RSPDIF时钟源时钟
rcu_exmc_clock_config	配置EXMC时钟源时钟
rcu_hpdc_clock_config	配置HPDC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_oscstb_wait	等待振荡器稳定标志位置位或振荡器起振超时
rcu_oscstb_on	打开振荡器
rcu_oscstb_off	关闭振荡器
rcu_oscstb_bypass_mode_enable	使能时钟旁路模式
rcu_oscstb_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 3-1205. 枚举类型 `rcu_periph_enum`

成员名称	功能描述
<code>RCU_ENET1</code>	ENET1时钟
<code>RCU_ENET1TX</code>	ENET1 TX时钟
<code>RCU_ENET1RX</code>	ENET1 RX时钟
<code>RCU_ENET1PTP</code>	ENET1 PTP时钟
<code>RCU_USBHS0</code>	USBHS0时钟
<code>RCU_USBHS0ULPI</code>	USBHS0 ULPI时钟
<code>RCU_DMA0</code>	DMA0时钟
<code>RCU_DMA1</code>	DMA1时钟
<code>RCU_DMAMUX</code>	IPA时钟
<code>RCU_ENET0</code>	ENET0时钟
<code>RCU_ENET0TX</code>	ENET0 TX时钟
<code>RCU_ENET0RX</code>	ENET0 RX时钟
<code>RCU_ENET0PTP</code>	ENET0 PTP时钟
<code>RCU_USBHS1</code>	USBHS1时钟
<code>RCU_USBHS1ULPI</code>	USBHS1 ULPI时钟
<code>RCU_DCI</code>	DCI时钟
<code>RCU_FAC</code>	FAC时钟
<code>RCU_SDIO1</code>	SDIO1时钟
<code>RCU_CAU</code>	CAU时钟
<code>RCU_HAU</code>	HAU时钟
<code>RCU_TRNG</code>	TRNG时钟
<code>RCU_TMU</code>	TMU时钟
<code>RCU_RAMECCMU1</code>	RAMECCMU1时钟
<code>RCU_EXMC</code>	EXMC时钟
<code>RCU_IPA</code>	IPA时钟
<code>RCU_SDIO0</code>	SDIO0时钟
<code>RCU_MDMA</code>	MDMMA时钟
<code>RCU_OSPIM</code>	OSPIM时钟
<code>RCU_OSPI0</code>	OSPI0时钟
<code>RCU_OSPI1</code>	OSPI1时钟
<code>RCU_RTDEC0</code>	RTDEC0时钟
<code>RCU_RTDEC1</code>	RTDEC1时钟
<code>RCU_RAMECCMU0</code>	RAMECCMU0时钟
<code>RCU_CPU</code>	CPU时钟
<code>RCU_GPIOA</code>	GPIOA时钟
<code>RCU_GPIOB</code>	GPIOB时钟
<code>RCU_GPIOC</code>	GPIOC时钟
<code>RCU_GPIOD</code>	GPIOD时钟

成员名称	功能描述
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_HWSEM	HWSEM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_TIMER30	TIMER30时钟
RCU_TIMER31	TIMER31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
RCU_MDIO	MDIO时钟
RCU_USART1	USART1时钟
RCU_USART2	USART2时钟
RCU_UART3	UART3时钟
RCU_UART4	UART4时钟
RCU_I2C0	I2C0时钟
RCU_I2C1	I2C1时钟
RCU_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

成员名称	功能描述
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_HPDPF	HPDPF时钟
RCU_SPI4	SPI4时钟
RCU_SPI5	SPI5时钟
RCU_SAI0	SAI0时钟
RCU_SAI1	SAI1时钟
RCU_SAI2	SAI2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_TLI	TLI时钟
RCU_WWDGT	WWDGT时钟
RCU_SYSCFG	SYSCFG时钟
RCU_CMP	CMP时钟
RCU_VREF	VREF时钟
RCU_LPPTS	LPPTS时钟
RCU_PMU	PMU时钟
RCU_RTC	RTC时钟
RCU_CAN0	CAN0时钟
RCU_CAN1	CAN1时钟
RCU_CAN2	CAN2时钟

枚举类型 `rcu_periph_sleep_enum`

表 3-1206. 枚举类型 `rcu_periph_sleep_enum`

成员名称	功能描述
RCU_ENET1_SLP	ENET1时钟
RCU_ENET1TX_SLP	ENET1 TX时钟
RCU_ENET1RX_SLP	ENET1 RX时钟

成员名称	功能描述
RCU_ENET1PTP_SLP	ENET1 PTP时钟
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_ENET0_SLP	ENET0时钟
RCU_ENET0TX_SLP	ENET0 TX时钟
RCU_ENET0RX_SLP	ENET0 RX时钟
RCU_ENET0PTP_SLP	ENET0 PTP时钟
RCU_USBHS1_SLP	USBHS1时钟
RCU_USBHS1ULPI_SLP	USBHS1ULPI时钟
RCU_DCI_SLP	DCI时钟
RCU_FAC_SLP	FAC时钟
RCU_SDIO1_SLP	SDIO1时钟
RCU_CAU_SLP	CAU时钟
RCU_HAU_SLP	HAU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_IPA_SLP	IPA时钟
RCU_SDIO0_SLP	SDIO0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

成员名称	功能描述
RCU_GPIOH_SLP	GPIOH时钟
RCU_GPIOJ_SLP	GPIOJ时钟
RCU_GPIOK_SLP	GPIOK时钟
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_TIMER30_SLP	TIMER30时钟
RCU_TIMER31_SLP	TIMER31时钟
RCU_TIMER50_SLP	TIMER50时钟
RCU_TIMER51_SLP	TIMER51时钟
RCU_RSPDIF_SLP	RSPDIF时钟
RCU_SPI1_SLP	SPI1时钟
RCU_SPI2_SLP	SPI2时钟
RCU_MDIO_SLP	MDIO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

成员名称	功能描述
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_HPDPF_SLP	HPDPF时钟
RCU_SPI4_SLP	SPI4时钟
RCU_SPI5_SLP	SPI5时钟
RCU_SAI0_SLP	SAI0时钟
RCU_SAI1_SLP	SAI1时钟
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_TLI_SLP	TLI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_LPPTS_SLP	LPPTS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 3-1207. 枚举类型 `rcu_periph_reset_enum`

成员名称	功能描述
RCU_ENET1RST	复位ENET1时钟
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_ENET0RST	复位ENET时钟
RCU_USBHS1RST	复位USBHS1HS时钟
RCU_DCIRST	复位DCI时钟

成员名称	功能描述
RCU_FACRST	复位FAC时钟
RCU_SDIO1RST	复位SDIO1时钟
RCU_CAURST	复位CAU时钟
RCU_HAURST	复位HAU时钟
RCU_TRNGRST	复位TRNG时钟
RCU_TMURST	复位TMU时钟
RCU_EXMCRST	复位EXMC时钟
RCU_IPARST	复位IPA时钟
RCU_SDIO0RST	复位SDIO0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
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_HWSEMRST	复位HWSEM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_TIMER30RST	复位TIMER30时钟
RCU_TIMER31RST	复位TIMER31时钟
RCU_TIMER50RST	复位TIMER50时钟
RCU_TIMER51RST	复位TIMER51时钟
RCU_RSPDIFRST	复位RSPDIF时钟
RCU_SPI1RST	复位SPI1时钟

成员名称	功能描述
RCU_SPI2RST	复位SPI2时钟
RCU_MDIORST	复位MDIO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
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HPDR时钟
RCU_SPI4RST	复位SPI4时钟
RCU_SPI5RST	复位SPI5时钟
RCU_SAI0RST	复位SAI0时钟
RCU_SAI1RST	复位SAI1时钟
RCU_SAI2RST	复位SAI2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_TLIRST	复位TLI时钟

成员名称	功能描述
RCU_WWDGTRST	复位WWDGT时钟
RCU_SYSCFGTRST	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 3-1208. 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
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 3-1209. 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

成员名称	功能描述
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 3-1210. 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
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 3-1211. 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 3-1212. 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 3-1213. 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

成员名称	功能描述
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 3-1214. 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 3-1215. 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
IDX_I2C2	I2C2索引
IDX_I2C3	I2C3索引

枚举类型 `can_idx_enum`

表 3-1216. 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 `sai_idx_enum`

表 3-1217. 枚举类型 `sai_idx_enum`

成员名称	功能描述
IDX_SAI0	SAI0索引
IDX_SAI1	SAI1索引

枚举类型 `sai2b_idx_enum`

表 3-1218. 枚举类型 `sai2b_idx_enum`

成员名称	功能描述
IDX_SAI2B0	SAI2B0索引
IDX_SAI2B1	SAI2B1索引

枚举类型 `adc_idx_enum`

表 3-1219. 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 3-1220. 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 3-1221. 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 `sdio_idx_enum`

表 3-1222. 枚举类型 `sdio_idx_enum`

成员名称	功能描述
IDX_SDIO0	SDIO0索引
IDX_SDIO1	SDIO1索引

枚举类型 `spi_idx_enum`表 3-1223. 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
IDX_SPI5	SPI5索引

函数 `rcu_deinit`

函数`rcu_deinit`描述见下表：

表 3-1224. 函数 `rcu_deinit`

函数名称	<code>rcu_deinit</code>
函数原形	<code>void rcu_deinit(void);</code>
功能描述	复位RCU，将RCU所有寄存器的值复位成初始值
先决条件	-
被调用函数	-
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 3-1225. 函数 `rcu_periph_clock_enable`

函数名称	<code>rcu_periph_clock_enable</code>
函数原形	<code>void rcu_periph_clock_enable(rcu_periph_enum periph);</code>
功能描述	使能外设时钟
先决条件	-
被调用函数	-
输入参数{in}	
periph	RCU外设，具体参考 表3-1205. 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 3-1226. 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3-1205. 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 3-1227. 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3-1206. 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 3-1228. 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3-1206. 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 3-1229. 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3-1229. 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 3-1230. 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3-1229. 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 3-1231. 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 3-1232. 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 3-1233. 函数 rcu_system_clock_source_config

函数名称	rcu_system_clock_source_config
函数原形	void rcu_system_clock_source_config(uint32_t ck_sys);
功能描述	配置选择系统时钟源
先决条件	-
被调用函数	-
输入参数{in}	
ck_sys	系统时钟源选择
RCU_CKSYSSRC_1 RC64MDIV	选择CK_IRC64MDIV时钟作为CK_SYS时钟源
RCU_CKSYSSRC_0 HXTAL	选择CK_HXTAL时钟作为CK_SYS时钟源

<i>RCU_CKSYSSRC_</i> <i>LPIRC4M</i>	选择CK_LPIRC4M时钟作为CK_SYS时钟源
<i>RCU_CKSYSSRC_</i> <i>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_CKSYSSRC_HXTAL);
```

函数 rcu_system_clock_source_get

函数rcu_system_clock_source_get描述见下表：

表 3-1234. 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 3-1235. 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 3-1236. 函数 rcu_apb1_clock_config

函数名称	rcu_apb1_clock_config
函数原形	void rcu_apb1_clock_config(uint32_t ck_apb1);
功能描述	配置APB1时钟预分频选择
先决条件	-
被调用函数	-
输入参数{in}	
ck_apb1	APB1预分频选择
RCU_APB1_CKAH B_DIVx	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_CKAHB_DIV16);
```

函数 rcu_apb2_clock_config

函数rcu_apb2_clock_config描述见下表:

表 3-1237. 函数 rcu_apb2_clock_config

函数名称	rcu_apb2_clock_config
函数原形	void rcu_apb2_clock_config(uint32_t ck_apb2);
功能描述	配置APB2时钟预分频选择
先决条件	-
被调用函数	-
输入参数{in}	
ck_apb2	APB2预分频选择
RCU_APB2_CKAHB_DIVx	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_CKAHB_DIV8);
```

函数 rcu_apb3_clock_config

函数rcu_apb3_clock_config描述见下表:

表 3-1238. 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
RCU_APB3_CKAHB_DIVx	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_CKAHB_DIV8);
```

函数 **rcu_apb4_clock_config**

函数rcu_apb4_clock_config描述见下表:

表 3-1239. 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
<i>RCU_APB4_CKAHB_DIVx</i>	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 3-1240. 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
<i>RCU_CKOUT0SRC_IRC64MDIV</i>	选择内部IRC64MDIV时钟
<i>RCU_CKOUT0SRC_LXTAL</i>	选择外部低速晶体振荡器时钟 (LXTAL)
<i>RCU_CKOUT0SRC_HXTAL</i>	选择外部高速晶体振荡器时钟 (HXTAL)
<i>RCU_CKOUT0SRC_PLL0P</i>	选择CK_PLL0P时钟
<i>RCU_CKOUT0SRC</i>	选择内部48M RC振荡器时钟

<code>_IRC48M</code>	
<code>RCU_CKOUT0SRC</code> <code>_PER</code>	选择外设时钟
<code>RCU_CKOUT0SRC</code> <code>_USBHS060M</code>	选择USBHS0 60M时钟
<code>RCU_CKOUT0SRC</code> <code>_USBHS160M</code>	选择USBHS1 60M时钟
输入参数{in}	
<code>ckout1_div</code>	CKOUT1分频系数
<code>RCU_CKOUT0_DIV</code> <code>x</code>	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 3-1241. 函数 `rcu_ckout1_config`

函数名称	<code>rcu_ckout1_config</code>
函数原形	<code>void rcu_ckout1_config(uint32_t ckout1_src, uint32_t ckout1_div);</code>
功能描述	配置CKOUT1时钟源选择及分频系数
先决条件	-
被调用函数	-
输入参数{in}	
<code>ckout1_src</code>	CKOUT1时钟源选择
<code>RCU_CKOUT1SRC</code> <code>_SYSTEMCLOCK</code>	选择系统时钟CK_SYS
<code>RCU_CKOUT1SRC</code> <code>_PLL1R</code>	选择CK_PLL1R时钟
<code>RCU_CKOUT1SRC</code> <code>_HXTAL</code>	选择外部高速速晶体振荡器时钟 (HXTAL)
<code>RCU_CKOUT1SRC</code> <code>_PLL0P</code>	选择CK_PLL0P时钟
<code>RCU_CKOUT1SRC</code> <code>_LPIRC4M</code>	选择内部低功耗4M RC振荡器时钟
<code>RCU_CKOUT1SRC</code>	选择内部低速32K振荡器时钟 (IRC32K)

<code>_IRC32K</code>	
<code>RCU_CKOUT1SRC</code> <code>_PLL2R</code>	选择CK_PLL2R时钟
输入参数{in}	
<code>ckout1_div</code>	CKOUT1分频系数
<code>RCU_CKOUT1_DIV</code> <code>x</code>	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 3-1242. 函数 `rcu_pll_input_output_clock_range_config`

函数名称	<code>rcu_pll_input_output_clock_range_config</code>
函数原形	<code>void rcu_pll_input_output_clock_range_config(pll_idx_enum pll_idx, uint32_t ck_input, uint32_t ck_output);</code>
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}	
<code>pll_idx</code>	pll索引, 参考 表3-1221. 枚举类型pll_idx_enum
输入参数{in}	
<code>ck_input</code>	输入时钟范围
<code>RCU_PLLRNG_1M_2M</code>	输入时钟频率: 1-2MHz
<code>RCU_PLLRNG_2M_4M</code>	输入时钟频率: 2-4MHz
<code>RCU_PLLRNG_4M_8M</code>	输入时钟频率: 4-8MHz
<code>RCU_PLLRNG_8M_16M</code>	输入时钟频率: 8-16MHz
输入参数{in}	
<code>RCU_PLLVCO_192M_836M</code>	选择宽输出时钟范围: 192-836MHz
<code>RCU_PLLVCO_150</code>	选择窄输出时钟范围: 150-420MHz

<i>M_420M</i>	
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 3-1243. 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 3-1244. 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 3-1245. 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 3-1246. 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
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 3-1247. 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

<i>uint32_t</i>	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 3-1248. 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
<i>uint32_t</i>	1-63
输入参数{in}	
pll1_n	PLL1时钟倍频因子
<i>uint32_t</i>	9-512
输入参数{in}	
pll1_p	PLL1P时钟分频因子
<i>uint32_t</i>	1-128
输入参数{in}	
pll1_q	PLL1Q时钟分频因子
<i>uint32_t</i>	1-128
输入参数{in}	
pll1_r	PLL1R时钟分频因子
<i>uint32_t</i>	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 3-1249. 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 3-1250. 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
RCU_PLL0P	选择PLL0P分频输出使能
RCU_PLL0Q	选择PLL0Q分频输出使能
RCU_PLL0R	选择PLL0R分频输出使能
RCU_PLL1P	选择PLL1P分频输出使能
RCU_PLL1Q	选择PLL1Q分频输出使能
RCU_PLL1R	选择PLL1R分频输出使能
RCU_PLL2P	选择PLL2P分频输出使能
RCU_PLL2Q	选择PLL2Q分频输出使能
RCU_PLL2R	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 3-1251. 函数 rcu_pll_clock_output_disable

函数名称	rcu_pll_clock_output_disable
函数原形	void rcu_pll_clock_output_disable(uint32_t pllxy);
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
RCU_PLL0P	选择PLL0P分频输出使能
RCU_PLL0Q	选择PLL0Q分频输出使能
RCU_PLL0R	选择PLL0R分频输出使能
RCU_PLL1P	选择PLL1P分频输出使能
RCU_PLL1Q	选择PLL1Q分频输出使能
RCU_PLL1R	选择PLL1R分频输出使能
RCU_PLL2P	选择PLL2P分频输出使能
RCU_PLL2Q	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 3-1252. 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PLLUSBHS0时钟
先决条件	-
被调用函数	-
输入参数{in}	
pllusb_presel	PLLUSBHSPRE时钟选择
<i>RCU_PLLUSBHSPRE_HXTAL</i>	PLLUSBHSPRE选择HXTAL作为时钟
<i>RCU_PLLUSBHSPRE_IRC48M</i>	PLLUSBHSPRE选择IRC48M作为时钟
输入参数{in}	
pllusb_predv	PLLUSBHS时钟分频因子
<i>RCU_PLLUSBHSPRE_DIVx (x= 1...15)</i>	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
<i>RCU_PLLUSBHSMULx (x = 16,17...127)</i>	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
<i>RCU_USBHSDV_DIVx (x = 2,4...16)</i>	选择 RCU_USBHSDV_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 PLLUSBHSPRE */
```

```
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 3-1253. 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_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 PLLUSBHS1 */
```

```
rcu_pllusb1_config(RCU_PLLUSBHSPRE_IRC48M, RCU_PLLUSBHSPRE_DIV1,  
RCU_PLLUSBHS_MUL1, RCU_USBHS_DIV1);
```

函数 rcu_rtc_clock_config

函数rcu_rtc_clock_config描述见下表:

表 3-1254. 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 3-1255. 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_NONE	未选择时钟
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 3-1256. 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 3-1257. 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 3-1258. 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 3-1259. 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 3-1260. 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_	如果CK_APBx = CK_AHB 或者 CK_APBx = CK_AHB / 2, CK_TIMERx =

MUL2	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 3-1261. 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3-1223. 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

<i>RCU_SPISRC_LPI RC4M</i>	选择CK_LPIRC4M时钟作为SPI时钟，适用于SPI3 / SPI4 / SPI5
<i>RCU_SPISRC_HXT AL</i>	选择CK_HXTAL时钟作为SPI时钟，适用于SPI3 / SPI4 / SPI5
<i>RCU_SPI5SRC_I2S _CKIN</i>	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_SPISRC_PLLQ);
```

函数 rcu_sdio_clock_config

函数rcu_sdio_clock_config描述见下表：

表 3-1262. 函数 rcu_sdio_clock_config

函数名称	rcu_sdio_clock_config
函数原形	void rcu_sdio_clock_config(sdio_idx_enum sdio_idx, uint32_t ck_sdio);
功能描述	配置SDIO时钟
先决条件	-
被调用函数	-
输入参数{in}	
sdio_idx	SDIO索引，参考 表3-1222. 枚举类型sdio_idx_enum
输入参数{in}	
ck_sdio	SDIOx时钟源
<i>RCU_SDIO0SRC_P LL0Q</i>	选择CK_PLL0Q时钟作为SDIO0时钟
<i>RCU_SDIO0SRC_P LL1R</i>	选择CK_PLL1R时钟作为SDIO0时钟
<i>RCU_SDIO1SRC_P LLQ</i>	选择CK_PLLQ时钟作为SDIO1时钟
<i>RCU_SDIO1SRC_P LL1R</i>	选择CK_PLL1R时钟作为SDIO1时钟
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* configure the PLLQ as SDIO0 clock */
```

```
rcu_sdio_clock_config(IDX_SDIO0, RCU_SDIO0SRC_PLLQ);
```

函数 rcu_deepsleep_wakeup_sys_clock_config

函数rcu_deepsleep_wakeup_sys_clock_config描述见下表：

表 3-1263. 函数 rcu_sdio_clock_config

函数名称	rcu_deepsleep_wakeup_sys_clock_config
函数原形	void rcu_deepsleep_wakeup_sys_clock_config(uint32_t ck_dspwussel);
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}	
ck_dspwussel	深度睡眠模式下系统唤醒时钟源
RCU_DSPWUSSEL_IRC64MDIV	选择CK_IRC64MDIV作为深度睡眠模式下系统唤醒时钟源
RCU_DSPWUSSEL_LPIRC4M	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_tli_clock_div_config

函数rcu_tli_clock_div_config描述见下表：

表 3-1264. 函数 rcu_tli_clock_div_config

函数名称	rcu_tli_clock_div_config
函数原形	void rcu_tli_clock_div_config(uint32_t pllsai_r_div);
功能描述	配置TLI分频系数
先决条件	-
被调用函数	-
输入参数{in}	
pllsai_r_div	TLI分频系数
RCU_PLL2R_DIV2	PLL2R / 2
RCU_PLL2R_DIV4	PLL2R / 4
RCU_PLL2R_DIV8	PLL2R / 8

RCU_PLL2R_DIV16	PLL2R / 16
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* configure the TLI prescaler factor from PLL2R clock */
```

```
rcu_tli_clock_div_config(RCU_PLL2R_DIV4);
```

函数 rcu_usart_clock_config

函数rcu_usart_clock_config描述见下表：

表 3-1265. 函数 rcu_usart_clock_config

函数名称	rcu_usart_clock_config
函数原形	void rcu_usart_clock_config(usart_idx_enum usart_idx, uint32_t ck_usart);
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3-1214. 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 3-1266. 函数 `rcu_i2c_clock_config`

函数名称	<code>rcu_i2c_clock_config</code>
函数原形	<code>void rcu_i2c_clock_config(i2c_idx_enum i2c_idx, uint32_t ck_i2c);</code>
功能描述	配置I2C时钟
先决条件	-
被调用函数	-
输入参数{in}	
<code>i2c_idx</code>	I2C索引，参考 表3-1215. 枚举类型<i>i2c_idx_enum</i>
输入参数{in}	
<code>ck_i2c</code>	I2Cx时钟源
<code>RCU_I2CSRC_APB1</code>	选择CK_APB1时钟作为I2C时钟
<code>RCU_I2CSRC_PLL2R</code>	选择CK_PLL2R时钟作为I2C时钟
<code>RCU_I2CSRC_IRC64MDIV</code>	选择CK_IRC64MDIV时钟作为I2C时钟
<code>RCU_I2CSRC_LPIRC4M</code>	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 3-1267. 函数 `rcu_can_clock_config`

函数名称	<code>rcu_can_clock_config</code>
函数原形	<code>void rcu_can_clock_config(can_idx_enum can_idx, uint32_t ck_can);</code>
功能描述	配置CAN时钟
先决条件	-
被调用函数	-
输入参数{in}	
<code>can_idx_enum</code>	CAN索引，参考 表3-1216. 枚举类型<i>can_idx_enum</i>
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 3-1268. 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3-1219. 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_PEL2R	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_sai_clock_config

函数rcu_sai_clock_config描述见下表：

表 3-1269. 函数 rcu_sai_clock_config

函数名称	rcu_sai_clock_config
函数原形	void rcu_sai_clock_config(sai_idx_enum sai_idx, uint32_t ck_sai);
功能描述	配置sai时钟
先决条件	-
被调用函数	-
输入参数{in}	
sai_idx	sai索引，参考 表3-1217. 枚举类型sai_idx_enum
输入参数{in}	
ck_sai	SAIx时钟源
RCU_SAISRC_PLL0Q	选择CK_PLL0Q时钟作为SAI时钟
RCU_SAISRC_PLL1P	选择CK_PLL1P时钟作为SAI时钟
RCU_SAISRC_PLL2P	选择CK_PLL2P时钟作为SAI时钟
RCU_SAISRC_CKIN	选择I2S_CKIN时钟作为SAI时钟
RCU_SAISRC_PER	选择CK_PER时钟作为SAI时钟
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* configure the CK_PLLQ as the SAI0 clock source */
```

```
rcu_sai_clock_config(IDX_SAI0, RCU_SAISRC_PLLQ);
```

函数 rcu_sai2_block_clock_config

函数rcu_sai2_block_clock_config描述见下表：

表 3-1270. 函数 rcu_sai2_block_clock_config

函数名称	rcu_sai2_block_clock_config
函数原形	void rcu_sai2_block_clock_config(sai2b_idx_enum sai2b_idx, uint32_t

	ck_sai2b);
功能描述	配置sai2时钟源时钟
先决条件	-
被调用函数	-
输入参数{in}	
sai2b_idx	sai2b索引, 参考 表3-1218. 枚举类型sai2b_idx_enum
输入参数{in}	
ck_sai2b	SAI2B时钟源
RCU_SAI2BSRC_P LL0Q	选择CK_PLL0Q时钟作为SAI2B时钟
RCU_SAI2BSRC_P LL1P	选择CK_PLL1P时钟作为SAI2B时钟
RCU_SAI2BSRC_P LL2P	选择CK_PLL2P时钟作为SAI2B时钟
RCU_SAI2BSRC_C KIN	选择I2S_CKIN时钟作为SAI2B时钟
RCU_SAI2BSRC_P ER	选择CK_PER时钟作为SAI2B时钟
RCU_SAI2BSRC_R SPDIF_SYMB	选择CK_RSPDIF_SYMB时钟作为SAI2B时钟
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* configure the CK_PLLQ as the SAI2B0 clock source */
```

```
rcu_sai2_block_clock_config(IDX_SAI2B0, RCU_SAI2SRC_PLLQ);
```

函数 rcu_rspdif_clock_config

函数rcu_rspdif_clock_config描述见下表:

表 3-1271. 函数 rcu_rspdif_clock_config

函数名称	rcu_rspdif_clock_config
函数原形	void rcu_rspdif_clock_config(uint32_t ck_rspdif);
功能描述	配置rspdif时钟源时钟
先决条件	-
被调用函数	-
输入参数{in}	
ck_rspdif	RSPDIF时钟源
RCU_RSPDIFSRC_ PLL0Q	选择CK_PLL0Q时钟作为RSPDIF时钟

<i>RCU_RSPDIFSRC_PLL1R</i>	选择CK_PLL1R时钟作为RSPDIF时钟
<i>RCU_RSPDIFSRC_PLL2R</i>	选择CK_PLL2R时钟作为RSPDIF时钟
<i>RCU_RSPDIFSRC_IRC64MDIV</i>	选择CK_IRC64MDIV时钟作为RSPDIF时钟
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* configure the CK_PLLQ as the RSPDIF clock source */
```

```
rcu_rspdif_clock_config(RCU_RSPDIFSRC_PLLQ);
```

函数 rcu_exmc_clock_config

函数rcu_exmc_clock_config描述见下表:

表 3-1272. 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
<i>RCU_EXMCSRC_AHB</i>	选择CK_AHB时钟作为EXMC时钟
<i>RCU_EXMCSRC_PLLQ</i>	选择CK_PLLQ时钟作为EXMC时钟
<i>RCU_EXMCSRC_PLL1R</i>	选择CK_PLL1R时钟作为EXMC时钟
<i>RCU_EXMCSRC_PER</i>	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 3-1273. 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_HPDSRC_A PB2	选择CK_APB2时钟作为HPDF时钟
RCU_HPDSRC_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 3-1274. 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_IRC 64MDIV	选择CK_IRC64MDIV时钟作为PER时钟
RCU_PERSRC_LPI RC4M	选择CK_LPIRC4M时钟作为PER时钟
RCU_PERSRC_HX TAL	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 3-1275. 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3-1220. 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 3-1276. 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 3-1277. 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_48M	选择48M作为USBHS时钟源
RCU_USBHSSEL_60M	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 3-1278. 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 3-1279. 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 3-1280. 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 3-1281. 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3-1212. 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 3-1282. 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3-1212. 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 3-1283. 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3-1212. 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 3-1284. 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3-1212. 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_osci_bypass_mode_enable(RCU_HXTAL);
```

函数 rcu_osci_bypass_mode_disable

函数rcu_osci_bypass_mode_disable描述见下表：

表 3-1285. 函数 rcu_osci_bypass_mode_disable

函数名称	rcu_osci_bypass_mode_disable
函数原形	void rcu_osci_bypass_mode_disable(rcu_osci_type_enum osci);
功能描述	禁能振荡器时钟旁路模式
先决条件	HXTALEN或LXTALEN应在使能振荡器时钟旁路模式前先复位

被调用函数	-
输入参数{in}	
osci	振荡器类型，参考 表3-1212. 枚举类型rcu_osci_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 3-1286. 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 3-1287. 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 3-1288. 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 3-1289. 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 3-1290. 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 3-1291. 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 3-1292. 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3-1213. 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 3-1293. 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3-1208. 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 3-1294. 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 3-1295. 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3-1211. 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 3-1296. 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3-1211. 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 3-1297. 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3-1211. 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 3-1298. 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3-1210. 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);
```

3.36. RSPDIF

S/PD数字音频接口接收器（RSPDIF）模块提供了接收及解码SPDIF音频数据流的功能。章节[3.36.1](#)描述了RSPDIF的寄存器列表，章节[3.36.2](#)对RSPDIF库函数进行说明。

3.36.1. 外设寄存器描述

RSPDIF寄存器列表如下表所示：

表 3-1299. RSPDIF 寄存器

寄存器名称	寄存器描述
RSPDIF_CTL	RSPDIF控制寄存器
RSPDIF_INTEN	RSPDIF中断使能寄存器
RSPDIF_STAT	RSPDIF状态寄存器
RSPDIF_STATC	RSPDIF状态清除寄存器
RSPDIF_DATA	RSPDIF接收数据寄存器
RSPDIF_CHSTAT	RSPDIF接收通道状态寄存器
RSPDIF_DTH	RSPDIF接收阈值寄存器

3.36.2. 外设库函数说明

RSPDIF库函数列表如下表所示：

表 3-1300. RSPDIF 库函数

库函数名称	库函数描述
rspdif_deinit	复位RSPDIF
rspdif_struct_para_init	初始化RSPDIF结构体中所有参数为默认值
rspdif_init	初始化RSPDIF参数
rspdif_enable	配置RSPDIF外设状态
rspdif_disable	禁能RSPDIF
rspdif_symbol_clock_enable	使能RSPDIF符号时钟
rspdif_symbol_clock_disable	禁能RSPDIF符号时钟
rspdif_backup_symbol_clock_enable	使能RSPDIF备份符号时钟
rspdif_backup_symbol_clock_disable	禁能RSPDIF备份符号时钟
rspdif_dma_enable	使能RSPDIF接收DMA
rspdif_dma_disable	禁能RSPDIF接收DMA
rspdif_control_buffer_dma_enable	使能RSPDIF控制流接收DMA
rspdif_control_buffer_dma_disable	禁能RSPDIF控制流接收DMA
rspdif_data_read	RSPDIF读数据
rspdif_duration_of_symbols_get	获取使用rspdif_ck统计的5个符号的持续时间
rspdif_user_data_get	获取用户数据信息
rspdif_channel_status_get	获取通道状态信息
rspdif_start_block_status_get	获取bolck起始
rspdif_low_threshold_get	获取阈值下限估计值
rspdif_high_threshold_get	获取阈值上限估计值
rspdif_flag_get	获取RSPDIF状态标志
rspdif_flag_clear	清除RSPDIF状态标志
rspdif_interrupt_enable	使能RSPDIF中断
rspdif_interrupt_disable	禁能RSPDIF中断
rspdif_interrupt_flag_get	获取RSPDIF中断状态标志
rspdif_interrupt_flag_clear	清除RSPDIF中断状态标志

结构体 `rspdif_parameter_struct`

表 3-1301. 结构体 `rspdif_parameter_struct`

成员名称	功能描述
<code>input_sel</code>	RSPDIF输入选择 (<code>RSPDIF_INPUT_INx</code> , $x = 0,1,2,3$)
<code>max_retrie</code>	RSPDIF在同步阶段允许的最大重试次数 (<code>RSPDIF_MAXRETRIES_NONE</code> , <code>RSPDIF_MAXRETRIES_3</code> , <code>RSPDIF_MAXRETRIES_15</code> , <code>RSPDIF_MAXRETRIES_63</code>)
<code>wait_activity</code>	RSPDIF是否等待选定输入上的活动 (<code>RSPDIF_WAIT_FOR_ACTIVITY_OFF</code> , <code>RSPDIF_WAIT_FOR_ACTIVITY_ON</code>)
<code>channel_sel</code>	是否从通道A或通道B交换通道状态 (<code>RSPDIF_CHANNEL_A</code> , <code>RSPDIF_CHANNEL_B</code>)
<code>sample_format</code>	RSPDIF数据样本格式 (<code>RSPDIF_DATAFORMAT_LSB</code> , <code>RSPDIF_DATAFORMAT_MSB</code> , <code>RSPDIF_DATAFORMAT_32BITS</code>)
<code>sound_mode</code>	RSPDIF为立体声或单声道模式 (<code>RSPDIF_STEREOMODE_DISABLE</code> , <code>RSPDIF_STEREOMODE_ENABLE</code>)
<code>pre_type</code>	是否将序言类型值复制到RSPDIF_DATA (<code>RSPDIF_PREAMBLE_TYPE_MASK_OFF</code> , <code>RSPDIF_PREAMBLE_TYPE_MASK_ON</code>)
<code>channel_status_bit</code>	是否将信道状态和用户位复制到接收帧中 (<code>RSPDIF_CHANNEL_STATUS_MASK_OFF</code> , <code>RSPDIF_CHANNEL_STATUS_MASK_ON</code>)
<code>validity_bit</code>	有效性位是否被复制到接收帧中 (<code>RSPDIF_VALIDITY_MASK_OFF</code> , <code>RSPDIF_VALIDITY_MASK_ON</code>)
<code>parity_error_bit</code>	是否将奇偶校验错误位复制到接收帧中 (<code>RSPDIF_PARITY_ERROR_MASK_OFF</code> , <code>RSPDIF_PARITY_ERROR_MASK_ON</code>)
<code>symbol_clk</code>	RSPDIF是否生成符号时钟 (<code>RSPDIF_SYMBOL_CLK_OFF</code> , <code>RSPDIF_SYMBOL_CLK_ON</code>)
<code>bak_symbol_clk</code>	RSPDIF是否生成备份符号时钟 (<code>RSPDIF_BACKUP_SYMBOL_CLK_OFF</code> , <code>RSPDIF_BACKUP_SYMBOL_CLK_ON</code>)

结构体 `rspdif_data_struct`

表 3-1302. 结构体 `rspdif_data_struct`

成员名称	功能描述
<code>format</code>	数据格式
<code>preamble</code>	序言类型
<code>channel_status</code>	通道状态位
<code>user_bit</code>	用户位

成员名称	功能描述
validity	有效位
parity_err	校验位
data0	数据0
data1	数据1

函数 rspdif_deinit

函数rspdif_deinit描述见下表：

表 3-1303. 函数 rspdif_deinit

函数名称	rspdif_deinit
函数原型	void rspdif_deinit(void);
功能描述	复位RSPDIF
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 RSPDIF */
```

```
rspdif_deinit ();
```

函数 rspdif_struct_para_init

函数rspdif_struct_para_init描述见下表：

表 3-1304. 函数 rspdif_struct_para_init

函数名称	rspdif_struct_para_init
函数原型	void rspdif_struct_para_init(rspdif_parameter_struct *rspdif_struct);
功能描述	初始化RSPDIF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}	
rspdif_parameter_struct	初始化的结构体rspdif_parameter_struct指针，结构体成员可以引用该结构体的成员 表3-1301. 结构体rspdif_parameter_struct
返回值	
-	-

例如:

```
/* initialize the parameters of RSPDIF structure with the default values */
```

```
rspdif_parameter_struct rspdif_structure;
```

```
rspdif_struct_para_init(&rspdif_structure);
```

函数 `rspdif_init`

函数 `rspdif_init` 描述见下表:

表 3-1305. 函数 `rspdif_init`

函数名称	<code>rspdif_init</code>
函数原型	<code>void rspdif_init(rspdif_parameter_struct *init_struct);</code>
功能描述	初始化RSPDIF参数
先决条件	-
被调用函数	-
输入参数{in}	
<code>rspdif_parameter_struct</code>	初始化的结构体 <code>rspdif_parameter_struct</code> 指针, 结构体成员可以引用该结构体的成员 表3-1301. 结构体 <code>rspdif_parameter_struct</code>
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* initialize the RSPDIF */
```

```
rspdif_parameter_struct rspdif_structure;
```

```
rspdif_struct_para_init(&rspdif_structure);
```

```
rspdif_structure.input_sel      = RSPDIF_INPUT_IN0;
```

```
rspdif_structure.max_retrie    = RSPDIF_MAXRETRIES_15;
```

```
rspdif_structure.wait_activity = RSPDIF_WAIT_FOR_ACTIVITY_ON;
```

```
rspdif_structure.channel_sel   = RSPDIF_CHANNEL_A;
```

```
rspdif_structure.sample_format = RSPDIF_DATAFORMAT_MSB;
```

```
rspdif_structure.sound_mode    = RSPDIF_STEREOMODE_ENABLE;
```

```
rspdif_structure.pre_type      = RSPDIF_PREAMBLE_TYPE_MASK_OFF;
```

```
rspdif_structure.channel_status_bit = RSPDIF_CHANNEL_STATUS_MASK_OFF;
```

```
rspdif_structure.validity_bit   = RSPDIF_VALIDITY_MASK_OFF;
```

```

rspdif_structure.parity_error_bit = RSPDIF_PARITY_ERROR_MASK_OFF;

rspdif_structure.symbol_clk       = RSPDIF_SYMBOL_CLK_OFF;

rspdif_structure.bak_symbol_clk   = RSPDIF_BACKUP_SYMBOL_CLK_OFF;

rspdif_init(&rspdif_structure);

```

函数 rspdif_enable

函数rspdif_enable描述见下表：

表 3-1306. 函数 rspdif_enable

函数名称	rspdif_enable
函数原型	void rspdif_enable(uint32_t mode);
功能描述	配置RSPDIF外设状态
先决条件	-
被调用函数	-
输入参数{in}	
mode	RSPDIF外设状态
RSPDIF_STATE_SYN C	仅使能RSPDIF同步
RSPDIF_STATE_RCV	使能RSPDIF
输出参数{out}	
-	-
返回值	
-	-

例如：

```

/* enable RSPDIF */

rspdif_enable(RSPDIF_STATE_RCV);

```

函数 rspdif_disable

函数rspdif_disable描述见下表：

表 3-1307. 函数 rspdif_disable

函数名称	rspdif_disable
函数原型	void rspdif_disable(void);
功能描述	禁能RSPDIF
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 RSPDIF */
```

```
rspdif_disable();
```

函数 `rspdif_symbol_clock_enable`

函数 `rspdif_symbol_clock_enable` 描述见下表:

表 3-1308. 函数 `rspdif_symbol_clock_enable`

函数名称	<code>rspdif_symbol_clock_enable</code>
函数原型	<code>void rspdif_symbol_clock_enable(void);</code>
功能描述	使能RSPDIF符号时钟
先决条件	-
被调用函数	-
输入参数{in}	
-	-
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* enable RSPDIF symbol clock */
```

```
rspdif_symbol_clock_enable();
```

函数 `rspdif_symbol_clock_disable`

函数 `rspdif_symbol_clock_disable` 描述见下表:

表 3-1309. 函数 `rspdif_symbol_clock_disable`

函数名称	<code>rspdif_symbol_clock_disable</code>
函数原型	<code>void rspdif_symbol_clock_disable(void);</code>
功能描述	禁能RSPDIF符号时钟
先决条件	-
被调用函数	-
输入参数{in}	
-	-
输出参数{out}	
-	-

返回值	
-	-

例如:

```
/* disable RSPDIF symbol clock */
```

```
rspdif_symbol_clock_disable();
```

函数 `rspdif_backup_symbol_clock_enable`

函数 `rspdif_backup_symbol_clock_enable` 描述见下表:

表 3-1310. 函数 `rspdif_backup_symbol_clock_enable`

函数名称	<code>rspdif_backup_symbol_clock_enable</code>
函数原型	<code>void rspdif_backup_symbol_clock_enable(void);</code>
功能描述	使能RSPDIF备份符号时钟
先决条件	-
被调用函数	-
输入参数{in}	
-	-
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* enable RSPDIF backup symbol clock */
```

```
rspdif_backup_symbol_clock_enable();
```

函数 `rspdif_backup_symbol_clock_disable`

函数 `rspdif_backup_symbol_clock_disable` 描述见下表:

表 3-1311. 函数 `rspdif_backup_symbol_clock_disable`

函数名称	<code>rspdif_backup_symbol_clock_disable</code>
函数原型	<code>void rspdif_backup_symbol_clock_disable(void);</code>
功能描述	禁能RSPDIF备份符号时钟
先决条件	-
被调用函数	-
输入参数{in}	
-	-
输出参数{out}	
-	-
返回值	

-	-
---	---

例如:

```
/* disable RSPDIF backup symbol clock */
```

```
rspdif_backup_symbol_clock_disable();
```

函数 `rspdif_dma_enable`

函数`rspdif_dma_enable`描述见下表:

表 3-1312. 函数 `rspdif_dma_enable`

函数名称	<code>rspdif_dma_enable</code>
函数原型	<code>void rspdif_dma_enable(void);</code>
功能描述	使能RSPDIF接收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 the RSPDIF receiver DMA */
```

```
rspdif_dma_enable();
```

函数 `rspdif_dma_disable`

函数`rspdif_dma_disable`描述见下表:

表 3-1313. 函数 `rspdif_dma_disable`

函数名称	<code>rspdif_dma_disable</code>
函数原型	<code>void rspdif_dma_disable(void);</code>
功能描述	禁能RSPDIF接收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 the RSPDIF receiver DMA */  
  
rspdif_dma_disable();
```

函数 `rspdif_control_buffer_dma_enable`

函数 `rspdif_control_buffer_dma_enable` 描述见下表:

表 3-1314. 函数 `rspdif_control_buffer_dma_enable`

函数名称	<code>rspdif_control_buffer_dma_enable</code>
函数原型	<code>void rspdif_control_buffer_dma_enable(void);</code>
功能描述	使能RSPDIF控制流接收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 the RSPDIF control buffer DMA */  
  
rspdif_control_buffer_dma_enable();
```

函数 `rspdif_control_buffer_dma_disable`

函数 `rspdif_control_buffer_dma_disable` 描述见下表:

表 3-1315. 函数 `rspdif_control_buffer_dma_disable`

函数名称	<code>rspdif_control_buffer_dma_disable</code>
函数原型	<code>void rspdif_control_buffer_dma_disable(void);</code>
功能描述	禁能RSPDIF控制流接收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 the RSPDIF control buffer DMA */
```

```
rspdif_control_buffer_dma_disable();
```

函数 `rspdif_data_read`

函数 `rspdif_data_read` 描述见下表：

表 3-1316. 函数 `rspdif_data_read`

函数名称	<code>rspdif_data_read</code>
函数原型	<code>void rspdif_data_read(rspdif_data_struct *data_struct);</code>
功能描述	RSPDIF 读数据
先决条件	-
被调用函数	-
输入参数{in}	
-	-
输出参数{out}	
<code>data_struct</code>	初始化的结构体 <code>rspdif_data_struct</code> 指针，结构体成员可以引用该结构体的成员 表3-1302. 结构体 <code>rspdif_data_struct</code>
返回值	
-	-

例如：

```
/* RSPDIF read data */
```

```
rspdif_data_struct data_structure;
```

```
rspdif_data_read(&data_structure);
```

函数 `rspdif_duration_of_symbols_get`

函数 `rspdif_duration_of_symbols_get` 描述见下表：

表 3-1317. 函数 `rspdif_duration_of_symbols_get`

函数名称	<code>rspdif_duration_of_symbols_get</code>
函数原型	<code>uint32_t rspdif_duration_of_symbols_get(void);</code>
功能描述	获取使用 <code>rspdif_ck</code> 统计的 5 个符号的持续时间
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>	0x0-0x7FFF

例如:

```
/* get duration of 5 symbols counted using rspdif_ck */
```

```
uint32_t cnt;
```

```
cnt = rspdif_duration_of_symbols_get();
```

函数 `rspdif_user_data_get`

函数 `rspdif_user_data_get` 描述见下表:

表 3-1318. 函数 `rspdif_user_data_get`

函数名称	<code>rspdif_user_data_get</code>
函数原型	<code>uint32_t rspdif_user_data_get(void);</code>
功能描述	获取用户数据信息
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>	0x0-0xFFFF

例如:

```
/* get user data information */
```

```
uint32_t user_data;
```

```
user_data = rspdif_user_data_get();
```

函数 `rspdif_channel_status_get`

函数 `rspdif_channel_status_get` 描述见下表:

表 3-1319. 函数 `rspdif_channel_status_get`

函数名称	<code>rspdif_channel_status_get</code>
函数原型	<code>uint32_t rspdif_channel_status_get(void);</code>
功能描述	获取通道状态信息
先决条件	-
被调用函数	-
输入参数{in}	
-	-
输出参数{out}	
-	-
返回值	

uint32_t	0x0-0xFF
----------	----------

例如:

```
/* get channel status information */

uint32_t ch_status;

ch_status = rspdif_channel_status_get();
```

函数 rspdif_start_block_status_get

函数rspdif_start_block_status_get描述见下表:

表 3-1320. 函数 rspdif_start_block_status_get

函数名称	rspdif_start_block_status_get
函数原型	FlagStatus rspdif_start_block_status_get(void);
功能描述	获取bolck起始
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 start of block */

FlagStatus flag;

flag = rspdif_start_block_status_get();
```

函数 rspdif_low_threshold_get

函数rspdif_low_threshold_get描述见下表:

表 3-1321. 函数 rspdif_low_threshold_get

函数名称	rspdif_low_threshold_get
函数原型	uint32_t rspdif_low_threshold_get(void);
功能描述	获取阈值下限估计值
先决条件	-
被调用函数	-
输入参数{in}	
-	-
输出参数{out}	

-	-
返回值	
uint32_t	0x0~0x1FF

例如:

```
/* get threshold low estimation */
```

```
uint32_t low_th;
```

```
low_th = rspdif_low_threshold_get();
```

函数 `rspdif_high_threshold_get`

函数`rspdif_high_threshold_get`描述见下表:

表 3-1322. 函数 `rspdif_high_threshold_get`

函数名称	<code>rspdif_high_threshold_get</code>
函数原型	<code>uint32_t rspdif_high_threshold_get(void);</code>
功能描述	获取阈值上限估计值
先决条件	-
被调用函数	-
输入参数{in}	
-	-
输出参数{out}	
-	-
返回值	
uint32_t	0x0~0x1FF

例如:

```
/* get threshold high estimation */
```

```
uint32_t high_th;
```

```
high_th = rspdif_high_threshold_get();
```

函数 `rspdif_flag_get`

函数`rspdif_flag_get`描述见下表:

表 3-1323. 函数 `rspdif_flag_get`

函数名称	<code>rspdif_flag_get</code>
函数原型	<code>FlagStatus rspdif_flag_get(uint16_t flag);</code>
功能描述	获取RSPDIF状态标志
先决条件	-
被调用函数	-
输入参数{in}	

flag	RSPDIF状态标志
RSPDIF_FLAG_RBNE	RSPDIF RX buffer非空
RSPDIF_FLAG_CBNE	RSPDIF RX控制流缓存非空
RSPDIF_FLAG_PERR	RSPDIF校验错误
RSPDIF_FLAG_RXOR ER	RSPDIF RX上溢错误
RSPDIF_FLAG_SYND B	RSPDIF已检测到同步块
RSPDIF_FLAG_SYND O	RSPDIF同步完成
RSPDIF_FLAG_FRER R	RSPDIF帧错误
RSPDIF_FLAG_SYNE RR	RSPDIF同步错误
RSPDIF_FLAG_TMOU TERR	RSPDIF超时错误
输出参数{out}	
-	-
返回值	
FlagStatus	SET或RESET

例如:

```
/* get RSPDIF flag status */
```

```
FlagStatus flag;
```

```
flag = rspdif_flag_get(RSPDIF_FLAG_TMOUTERR);
```

函数 `rspdif_flag_clear`

函数`rspdif_flag_clear`描述见下表:

表 3-1324. 函数 `rspdif_flag_clear`

函数名称	<code>rspdif_flag_clear</code>
函数原型	<code>void rspdif_flag_clear(uint16_t flag);</code>
功能描述	清除RSPDIF状态标志
先决条件	-
被调用函数	-
输入参数{in}	
flag	RSPDIF状态标志
RSPDIF_FLAG_PERR	RSPDIF校验错误
RSPDIF_FLAG_RXOR ER	RSPDIF RX上溢错误
RSPDIF_FLAG_SYND	RSPDIF已检测到同步块

B	
RSPDIF_FLAG_SYND O	RSPDIF同步完成
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* clear RSPDIF flag */
```

```
rspdif_flag_clear(RSPDIF_FLAG_PERR);
```

函数 `rspdif_interrupt_enable`

函数 `rspdif_interrupt_enable` 描述见下表:

表 3-1325. 函数 `rspdif_interrupt_enable`

函数名称	<code>rspdif_interrupt_enable</code>
函数原型	<code>void rspdif_interrupt_enable(uint8_t interrupt);</code>
功能描述	使能RSPDIF中断
先决条件	-
被调用函数	-
输入参数{in}	
interrupt	RSPDIF中断
RSPDIF_INT_RBNE	RSPDIF RX buffer非空中断
RSPDIF_INT_CBNE	RSPDIF RX控制流缓存非空中断
RSPDIF_INT_PERR	RSPDIF校验错误中断
RSPDIF_INT_RXORER R	RSPDIF RX上溢错误中断
RSPDIF_INT_SYNDB	RSPDIF已检测到同步块中断
RSPDIF_INT_SYNDO	RSPDIF同步完成中断
RSPDIF_INT_RXDCER R	RSPDIF解码错误中断
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* enable RSPDIF interrupt */
```

```
rspdif_interrupt_enable(RSPDIF_INT_RBNE);
```

函数 rspdif_interrupt_disable

函数rspdif_interrupt_disable描述见下表：

表 3-1326. 函数 rspdif_interrupt_disable

函数名称	rspdif_interrupt_disable
函数原型	void rspdif_interrupt_disable(uint8_t interrupt);
功能描述	禁能RSPDIF中断
先决条件	-
被调用函数	-
输入参数{in}	
interrupt	RSPDIF中断
RSPDIF_INT_RBNE	RSPDIF RX buffer非空中断
RSPDIF_INT_CBNE	RSPDIF RX控制流缓存非空中断
RSPDIF_INT_PERR	RSPDIF校验错误中断
RSPDIF_INT_RXORERR	RSPDIF RX上溢错误中断
RSPDIF_INT_SYNDB	RSPDIF已检测到同步块中断
RSPDIF_INT_SYNDO	RSPDIF同步完成中断
RSPDIF_INT_RXDCERR	RSPDIF解码错误中断
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* disable RSPDIF interrupt */
rspdif_interrupt_disable(RSPDIF_INT_RBNE);
```

函数 rspdif_interrupt_flag_get

函数rspdif_interrupt_flag_get描述见下表：

表 3-1327. 函数 rspdif_interrupt_flag_get

函数名称	rspdif_interrupt_flag_get
函数原型	FlagStatus rspdif_interrupt_flag_get(uint16_t int_flag);
功能描述	获取RSPDIF中断状态标志
先决条件	-
被调用函数	-
输入参数{in}	
int_flag	RSPDIF中断状态标志
RSPDIF_INT_FLAG_	RSPDIF RX buffer非空

RBNE	
RSPDIF_INT_FLAG_C BNE	RSPDIF RX控制流缓存非空
RSPDIF_INT_FLAG_P ERR	RSPDIF校验错误
RSPDIF_INT_FLAG_R XORER	RSPDIF RX上溢错误
RSPDIF_INT_FLAG_S YNCDB	RSPDIF已检测到同步块
RSPDIF_INT_FLAG_S YND0	RSPDIF同步完成
RSPDIF_INT_FLAG_F RERR	RSPDIF帧错误
RSPDIF_INT_FLAG_S YNERR	RSPDIF同步错误
RSPDIF_INT_FLAG_T MOUTERR	RSPDIF超时错误
输出参数{out}	
-	-
返回值	
FlagStatus	SET或RESET

例如:

```
/* get RSPDIF interrupt flag status */
```

```
FlagStatus flag;
```

```
flag = rspdif_interrupt_flag_get(RSPDIF_INT_FLAG_TMOUTERR);
```

函数 `rspdif_interrupt_flag_clear`

函数`rspdif_interrupt_flag_clear`描述见下表:

表 3-1328. 函数 `rspdif_interrupt_flag_clear`

函数名称	<code>rspdif_interrupt_flag_clear</code>
函数原型	<code>void rspdif_interrupt_flag_clear(uint16_t int_flag);</code>
功能描述	清除RSPDIF中断状态标志
先决条件	-
被调用函数	-
输入参数{in}	
int_flag	RSPDIF中断状态标志
RSPDIF_INT_FLAG_P ERR	RSPDIF校验错误
RSPDIF_INT_FLAG_R	RSPDIF RX上溢错误

XORER	
RSPDIF_INT_FLAG_S YNDB	RSPDIF已检测到同步块
RSPDIF_INT_FLAG_S YNDO	RSPDIF同步完成
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* clear RSPDIF interrupt flag status */
```

```
rspdif_interrupt_flag_clear(RSPDIF_INT_FLAG_PERR);
```

3.37. 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[3.37.1](#)描述了RTC的寄存器列表，章节[3.37.2](#)对RTC库函数进行说明。

3.37.1. 外设寄存器描述

RTC寄存器列表如下表所示:

表 3-1329. 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

寄存器名称	寄存器描述
RTC_ALRM0SS	RTC闹钟0亚秒寄存器
RTC_ALRM1SS	RTC闹钟1亚秒寄存器
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备份域寄存器4

3.37.2. 外设库函数描述

RTC库函数列表如下表所示：

表 3-1330. 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
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_tamper_mask	RTC侵入检测屏蔽配置
rtc_tamper_without_bkp_reset	RTC侵入检测事件不擦除备份域寄存器配置
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

库函数名称	库函数描述
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 3-1331. 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 3-1332. 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 3-1333. 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 3-1334. 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 3-1335. 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 3-1336. 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3-1331. 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 3-1337. 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 3-1338. 函数 rtc_init_mode_exit

函数名称	rtc_init_mode_exit
函数原型	void rtc_init_mode_exit(void);
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 3-1339. 函数 rtc_register_sync_wait

函数名称	rtc_register_sync_wait
函数原型	ErrStatus rtc_register_sync_wait(void);
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 3-1340. 函数 rtc_current_time_get

函数名称	rtc_current_time_get
函数原型	void rtc_current_time_get(rtc_parameter_struct* rtc_initpara_struct);
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}	
rtc_initpara_struct	初始化结构体，结构体成员参考 表3-1331. 结构体rtc_parameter_struct
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 3-1341. 函数 rtc_subsecond_get

函数名称	rtc_subsecond_get
函数原型	uint32_t rtc_subsecond_get(void);
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	
uint32_t	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 3-1342. 函数 rtc_alarm_config

函数名称	rtc_alarm_config
函数原型	void rtc_alarm_config(uint8_t rtc_alarm, rtc_alarm_struct *rtc_alarm_time);
功能描述	配置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
输入参数{in}	
rtc_alarm_time	闹钟结构体，结构体成员参考 表3-1332. 结构体rtc_alarm_struct
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 3-1343. 函数 rtc_alarm_subsecond_config

函数名称	rtc_alarm_subsecond_config
函数原型	void rtc_alarm_subsecond_config(uint8_t rtc_alarm, uint32_t mask_subsecond, uint32_t subsecond)
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}	
rtc_alarm	RTC_ALARM0或者RTC_ALARM1
输入参数{in}	
mask_subsecond	闹钟亚秒屏蔽位
RTC_MASKSSC_0_14	屏蔽闹钟亚秒设置
RTC_MASKSSC_1_14	屏蔽RTC_ALRM0SS_SSC[14:1], SSC[0]位用于时间匹配
RTC_MASKSSC_2_14	屏蔽RTC_ALRM0SS_SSC[14:2], SSC[1:0]位用于时间匹配
RTC_MASKSSC_3_14	屏蔽RTC_ALRM0SS_SSC[14:3], SSC[2:0]位用于时间匹配
RTC_MASKSSC_4_14	屏蔽RTC_ALRM0SS_SSC[14:4], SSC[3:0]位用于时间匹配
RTC_MASKSSC_5_14	屏蔽RTC_ALRM0SS_SSC[14:5], SSC[4:0]位用于时间匹配
RTC_MASKSSC_6_14	屏蔽RTC_ALRM0SS_SSC[14:6], SSC[5:0]位用于时间匹配
RTC_MASKSSC_7_14	屏蔽RTC_ALRM0SS_SSC[14:7], SSC[6:0]位用于时间匹配
RTC_MASKSSC_8_14	屏蔽RTC_ALRM0SS_SSC[14:8], SSC[7:0]位用于时间匹配
RTC_MASKSSC_9_14	屏蔽RTC_ALRM0SS_SSC[14:9], SSC[8:0]位用于时间匹配
RTC_MASKSSC_10_14	屏蔽RTC_ALRM0SS_SSC[14:10], SSC[9:0]位用于时间匹配
RTC_MASKSSC_11_14	屏蔽RTC_ALRM0SS_SSC[14:11], SSC[10:0]位用于时间匹配
RTC_MASKSSC_12_14	屏蔽RTC_ALRM0SS_SSC[14:12], SSC[11:0]位用于时间匹配
RTC_MASKSSC_13_14	屏蔽RTC_ALRM0SS_SSC[14:13], SSC[12:0]位用于时间匹配
RTC_MASKSSC_14	屏蔽RTC_ALRM0SS_SSC[14], SSC[13:0]位用于时间匹配
RTC_MASKSSC_NONE	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 3-1344. 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 3-1345. 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 3-1346. 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3-1332. 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 3-1347. 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 3-1348. 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 3-1349. 函数 rtc_timestamp_disable

函数名称	rtc_timestamp_disable
函数原型	void rtc_timestamp_disable(void);
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 3-1350. 函数 rtc_timestamp_internalevent_config

函数名称	rtc_timestamp_internalevent_config
函数原型	void rtc_timestamp_internalevent_config(uint32_t mode)
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
RTC_ITSEN_DISABLE	失能RTC时间戳内部事件
RTC_ITSEN_ENABLE	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 3-1351. 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3-1333. 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 3-1352. 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 3-1353. 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3-1334. 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 3-1354. 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
<i>RTC_TAMPER0</i>	RTC 侵入检测0
<i>RTC_TAMPER1</i>	RTC 侵入检测1
<i>RTC_TAMPER2</i>	RTC 侵入检测2
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 3-1355. 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
<i>RTC_OUT_PC13</i>	RTC输出引脚为PC13
<i>RTC_OUT_PB2</i>	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 3-1356. 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_OUTPU T_OD</i>	开漏输出
<i>RTC_ALARM_OUTPU 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 3-1357. 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
RTC_CALIBRATION_5 12HZ	当LSE频率为32768Hz并且RTC_PSC 为默认值，输出512Hz信号
RTC_CALIBRATION_1 HZ	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 3-1358. 函数 rtc_hour_adjust

函数名称	rtc_hour_adjust
函数原型	void rtc_hour_adjust(uint32_t operation);
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

<i>RTC_CTL_A1H</i>	增加一个小时
<i>RTC_CTL_S1H</i>	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 3-1359. 函数 rtc_second_adjust

函数名称	rtc_second_adjust
函数原型	ErrStatus rtc_second_adjust(uint32_t add, uint32_t minus);
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
<i>RTC_SHIFT_ADD1S_RESET</i>	无影响
<i>RTC_SHIFT_ADD1S_SET</i>	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 3-1360. 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 3-1361. 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 3-1362. 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 3-1363. 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:

```
/* disable RTC 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 3-1364. 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 3-1365. 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 3-1366. 函数 `rtc_wakeup_clock_set`

函数名称	<code>rtc_wakeup_clock_set</code>
函数原型	<code>ErrStatus rtc_wakeup_clock_set(uint8_t wakeup_clock);</code>
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}	
<code>wakeup_clock</code>	时钟选择
<code>WAKEUP_RTCK_DIV16</code>	RTC时钟的16分频
<code>WAKEUP_RTCK_DIV8</code>	RTC时钟的8分频
<code>WAKEUP_RTCK_DIV4</code>	RTC时钟的4分频
<code>WAKEUP_RTCK_DIV2</code>	RTC时钟的2分频
<code>WAKEUP_CKSPRE</code>	<code>ck_spre</code> (默认1Hz)时钟
<code>WAKEUP_CKSPRE_2EXP16</code>	<code>ck_spre</code> (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 3-1367. 函数 `rtc_wakeup_timer_set`

函数名称	<code>rtc_wakeup_timer_set</code>
函数原型	<code>ErrStatus rtc_wakeup_timer_set(uint16_t wakeup_timer);</code>
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}	
<code>wakeup_timer</code>	定时器选择
<code>uint16_t</code>	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 3-1368. 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 3-1369. 函数 rtc_smooth_calibration_config

函数名称	rtc_smooth_calibration_config
函数原型	ErrStatus rtc_smooth_calibration_config(uint32_t window, uint32_t plus, uint32_t minus);
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
RTC_CALIBRATION_	采用32S校准周期

<i>WINDOW_32S</i>	
<i>RTC_CALIBRATION_WINDOW_16S</i>	采用16S校准周期
<i>RTC_CALIBRATION_WINDOW_8S</i>	采用8S校准周期
输入参数{in}	
plus	增加脉冲
<i>RTC_CALIBRATION_PLUS_SET</i>	每2048个脉冲增加一个RTCCLK脉冲
<i>RTC_CALIBRATION_PLUS_RESET</i>	无影响
输入参数{in}	
minus	校准窗口校准周期RTCCLK脉冲屏蔽数（0x0-0xFF）
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 3-1370. 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
<i>RTC_INT_TIMESTAMP</i>	时间戳中断
<i>RTC_INT_ALARM0</i>	闹钟0中断
<i>RTC_INT_ALARM1</i>	闹钟1中断
<i>RTC_INT_TAMP_ALL</i>	所有侵入检测中断
<i>RTC_INT_TAMP0</i>	侵入检测0中断
<i>RTC_INT_TAMP1</i>	侵入检测1中断

<i>RTC_INT_TAMP2</i>	侵入检测2中断
<i>RTC_INT_WAKEUP</i>	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 3-1371. 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
<i>RTC_INT_TIMESTAMP</i>	时间戳中断
<i>RTC_INT_ALARM0</i>	闹钟0中断
<i>RTC_INT_ALARM1</i>	闹钟1中断
<i>RTC_INT_TAMP_ALL</i>	所有侵入检测中断
<i>RTC_INT_TAMP0</i>	侵入检测0中断
<i>RTC_INT_TAMP1</i>	侵入检测1中断
<i>RTC_INT_TAMP2</i>	侵入检测2中断
<i>RTC_INT_WAKEUP</i>	唤醒定时器中断
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* disble 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 3-1372. 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 3-1373. 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_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_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);
```

3.38. RTDEC

根据读请求地址的信息，实时解密（RTDEC）模块可以进行实时解密。RTDEC 可以配置四个独立、不同的加密区域。每个区域都可被选择配置为只执行或从不执行。RTDEC 通常与外部存储配合使用。章节 [3.38.1](#) 描述了 RTC 的寄存器列表，章节 [3.38.2](#) 对 RTC 库函数进行说明。

3.38.1. 外设寄存器描述

RTDEC寄存器列表如下表所示:

表 3-1374. RTDEC 寄存器

寄存器名称	寄存器描述
RTDEC_ARE_CFG	区域配置寄存器
RTDEC_ARE_SADDR	区域起始地址寄存器
RTDEC_ARE_EADDR	区域结束地址寄存器
RTDEC_ARE_NONCE0	区域随机数寄存器0
RTDEC_ARE_NONCE1	区域随机数寄存器1
RTDEC_ARE_KEY0	区域密钥寄存器0
RTDEC_ARE_KEY1	区域密钥寄存器1
RTDEC_ARE_KEY2	区域密钥寄存器2

寄存器名称	寄存器描述
RTDEC_ARE_KEY3	区域密钥寄存器3
RTDEC_INTF	中断标志寄存器
RTDEC_INTC	中断标志清除寄存器
RTDEC_INTEN	中断使能寄存器

3.38.2. 外设库函数说明

RTDEC库函数列表如下表所示：

表 3-1375. RTDEC 库函数

库函数名称	库函数描述
rtdec_deinit	复位RTDEC
rtdec_struct_para_init	使用默认值初始化RTDEC结构体参数
rtdec_init	初始化RTDEC
rtdec_config	配置RTDEC区域数据属性
rtdec_lock	锁定RTDEC密钥或寄存器
rtdec_addr_init	初始化RTDEC区域地址
rtdec_nonce_init	初始化RTDEC区域随机数，遵循小端模式
rtdec_key_init	初始化RTDEC区域密钥，遵循小端模式
rtdec_key_crc_get	获取RTDEC密钥CRC
rtdec_enable	使能RTDEC区域
rtdec_disable	禁能RTDEC区域
rtdec_flag_get	获取RTDEC错误标志
rtdec_flag_clear	清除RTDEC错误标志
rtdec_interrupt_enable	使能RTDEC中断
rtdec_interrupt_disable	禁能RTDEC中断
rtdec_interrupt_flag_get	获取RTDEC中断标志
rtdec_interrupt_flag_clear	清除RTDEC中断标志

结构体 rtdec_parameter_struct

表 3-1376. 结构体 rtdec_parameter_struct

成员名称	功能描述
access_mode	区域访问模式
key_crc	密钥CRC值
fw_version	区域固件版本
key	区域密钥
nonce	区域随机数
start_addr	区域起始地址
end_addr	区域结束地址

函数 rtdec_deinit

函数 rtdec_deinit 描述见下表：

表 3-1377. 函数 rtdec_deinit

函数名称	rtdec_deinit
函数原型	void rtdec_deinit(uint32_t rtdec_periph);
功能描述	复位RTDEC
先决条件	无
被调用函数	无
输入参数{in}	
rtdec_periph	RTDEC外设
RTDECx	x=1, 2
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* reset RTDEC */
rtdec_deinit (RTDEC0);
```

函数 rtdec_struct_para_init

函数rtdec_struct_para_init描述见下表：

表 3-1378. 函数 rtdec_struct_para_init

函数名称	rtdec_struct_para_init
函数原型	void rtdec_struct_para_init(rtdec_parameter_struct *rtdec_struct);
功能描述	使用默认值初始化RTDEC结构体参数
先决条件	无
被调用函数	无
输入参数{in}	
-	-
输出参数{out}	
rtdec_struct	RTDEC初始化结构体参数，参考 表3-1374. RTDEC寄存器
返回值	
-	-

例如：

```
rtdec_parameter_struct rtdec_struct;

/* initialize rtdec_struct parameter wit default values */

rtdec_struct_para_init(&rtdec_struct);
```


函数 `rtdec_init`

函数`rtdec_init`描述见下表:

表 3-1379. 函数 `rtdec_init`

函数名称	<code>rtdec_init</code>
函数原型	<code>ErrStatus rtdec_init(uint32_t rtdec_periph, uint32_t rtdec_area, rtdec_parameter_struct *rtdec_struct);</code>
功能描述	初始化RTDEC
先决条件	无
被调用函数	无
输入参数{in}	
<code>rtdec_periph</code>	RTDEC外设
<code>RTDECx</code>	<code>x=1, 2</code>
<code>rtdec_area</code>	RTDEC区域
<code>RTDEC_AREAx</code>	<code>x=0, 1, 2, 3</code>
<code>rtdec_struct</code>	RTDEC初始化结构体参数, 参考 表3-1374. RTDEC寄存器
输出参数{out}	
-	-
返回值	
<code>ErrStatus</code>	ERROR或SUCCESS

例如:

```
rtdec_parameter_struct rtdec_struct;

/* configure RTDEC area0 */

rtdec_init(RTDEC_USE, RTDEC_AREA0, &rtdec_struct);
```

函数 `rtdec_config`

函数`rtdec_config`描述见下表:

表 3-1380. 函数 `rtdec_config`

函数名称	<code>rtdec_config</code>
函数原型	<code>void rtdec_config(uint32_t rtdec_periph, uint32_t rtdec_area, uint8_t access_mode, uint16_t firmware_version);</code>
功能描述	配置区域数据属性
先决条件	无
被调用函数	无
输入参数{in}	
<code>rtdec_periph</code>	RTDEC外设
<code>RTDECx</code>	<code>x=1, 2</code>
<code>rtdec_area</code>	RTDEC区域
<code>RTDEC_AREAx</code>	<code>x=0, 1, 2, 3</code>

access_mode	允许的数据访问模式
<i>RTDEC_MODE_CODE_ACCESS</i>	仅代码/指令访问
<i>RTDEC_MODE_DATA_ACCESS</i>	仅数据访问
<i>RTDEC_MODE_BOTH_ACCESS</i>	代码和数据均可访问
firmware_version	16位区域版本
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* configure RTDEC area0 */
```

```
rtdec_config(RTDEC0, RTDEC_AREA0, RTDEC_MODE_CODE_ACCESS, 0x1234);
```

函数 rtdec_lock

函数rtdec_lock描述见下表：

表 3-1381. 函数 rtdec_lock

函数名称	rtdec_lock
函数原型	void rtdec_lock(uint32_t rtdec_periph, uint32_t rtdec_area, uint32_t lock_type);
功能描述	锁定RTDEC密钥或寄存器
先决条件	无
被调用函数	无
输入参数{in}	
rtdec_periph	RTDEC外设
<i>RTDECx</i>	x=1, 2
rtdec_area	RTDEC区域
<i>RTDEC_AREAx</i>	x=0, 1, 2, 3
lock_type	密钥或区域寄存器锁定
<i>RTDEC_ARE_CFG_LK</i>	区域寄存器锁定
<i>RTDEC_ARE_K_LK</i>	密钥锁定
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* lock RTDEC area0 key */
```

```
rtdec_lock(RTDEC0, RTDEC_AREA0, RTDEC_ARE_K_LK);
```

函数 rtdec_addr_init

函数rtdec_addr_init描述见下表：

表 3-1382. 函数 rtdec_addr_init

函数名称	rtdec_addr_init
函数原型	void rtdec_addr_init(uint32_t rtdec_periph, uint32_t rtdec_area, uint32_t saddr, uint32_t eaddr);
功能描述	初始化RTDEC区域地址
先决条件	无
被调用函数	无
输入参数{in}	
rtdec_periph	RTDEC外设
RTDECx	x=1, 2
rtdec_area	RTDEC区域
RTDEC_AREAx	x=0, 1, 2, 3
saddr	区域起始地址，忽略高4位和低12位
eaddr	区域结束地址，忽略高4位和低12位
输出参数{out}	
-	-
返回值	
-	-

例如：

```
uint32_t start_addr = 0x90000000;
```

```
uint32_t end_addr = 0x90001000;
```

```
/* initialize RTDEC area0 address */
```

```
rtdec_addr_init(RTDEC0, RTDEC_AREA0, start_addr, end_addr);
```

函数 rtdec_nonce_init

函数rtdec_nonce_init描述见下表：

表 3-1383. 函数 rtdec_nonce_init

函数名称	rtdec_nonce_init
函数原型	void rtdec_nonce_init(uint32_t rtdec_periph, uint32_t rtdec_area, uint32_t* nonce);
功能描述	初始化RTDEC区域随机数，遵循小端模式
先决条件	无
被调用函数	无
输入参数{in}	

rtdec_periph	RTDEC外设
<i>RTDECx</i>	x=1, 2
rtdec_area	RTDEC区域
<i>RTDEC_AREAx</i>	x=0, 1, 2, 3
nonce	包含64位随机数的数组，遵循小端模式
输出参数{out}	
-	-
返回值	
-	-

例如：

```
uint32_t nonce[2] = {0x89ABCDEF, 0x01234567};

/* initialize RTDEC area0 nonce */

rtdec_addr_init(RTDEC0, RTDEC_AREA0, nonce);
```

函数 rtdec_key_init

函数rtdec_key_init描述见下表：

表 3-1384. 函数 rtdec_key_init

函数名称	rtdec_key_init
函数原型	void rtdec_key_init(uint32_t rtdec_periph, uint32_t rtdec_area, uint32_t* key);
功能描述	初始化RTDEC密钥，遵循小端模式
先决条件	无
被调用函数	无
输入参数{in}	
rtdec_periph	RTDEC外设
<i>RTDECx</i>	x=1, 2
rtdec_area	RTDEC区域
<i>RTDEC_AREAx</i>	x=0, 1, 2, 3
key	包含128位密钥的数组，遵循小端模式
输出参数{out}	
-	-
返回值	
-	-

例如：

```
uint32_t rtdec_key[4] = {0x01234567, 0x89ABCDEF, 0x01234567, 0x89ABCDEF};

/* initialize RTDEC area0 key */

rtdec_key_init(RTDEC0, RTDEC_AREA0, rtdec_key);
```

函数 `rtdec_key_crc_get`

函数 `rtdec_key_crc_get` 描述见下表：

表 3-1385. 函数 `rtdec_key_crc_get`

函数名称	<code>rtdec_key_crc_get</code>
函数原型	<code>uint8_t rtdec_key_crc_get(uint32_t rtdec_periph, uint32_t rtdec_area);</code>
功能描述	获取RTDEC密钥CRC值
先决条件	无
被调用函数	无
输入参数{in}	
<code>rtdec_periph</code>	RTDEC外设
<code>RTDECx</code>	$x=1, 2$
<code>rtdec_area</code>	RTDEC区域
<code>RTDEC_AREAx</code>	$x=0, 1, 2, 3$
输出参数{out}	
-	-
返回值	
<code>uint8_t</code>	CRC值

例如：

```
uint8_t key_crc;
```

```
/* get RTDEC area0 key crc */
```

```
key_crc = rtdec_key_crc_get(RTDEC0, RTDEC_AREA0);
```

函数 `rtdec_enable`

函数 `rtdec_enable` 描述见下表：

表 3-1386. 函数 `rtdec_enable`

函数名称	<code>rtdec_enable</code>
函数原型	<code>void rtdec_enable(uint32_t rtdec_periph, uint32_t rtdec_area);</code>
功能描述	使能RTDEC区域
先决条件	无
被调用函数	无
输入参数{in}	
<code>rtdec_periph</code>	RTDEC外设
<code>RTDECx</code>	$x=1, 2$
<code>rtdec_area</code>	RTDEC区域
<code>RTDEC_AREAx</code>	$x=0, 1, 2, 3$
输出参数{out}	
-	-
返回值	

-	-
---	---

例如：

```
/* enable RTDEC area0 */
```

```
rtdec_enable(RTDEC0, RTDEC_AREA0);
```

函数 rtdec_disable

函数rtdec_enable描述见下表：

表 3-1387. 函数 rtdec_disable

函数名称	rtdec_disable
函数原型	void rtdec_disable(uint32_t rtdec_periph, uint32_t rtdec_area);
功能描述	禁能RTDEC区域
先决条件	无
被调用函数	无
输入参数{in}	
rtdec_periph	RTDEC外设
RTDECx	x=1, 2
rtdec_area	RTDEC区域
RTDEC_AREAx	x=0, 1, 2, 3
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* disable RTDEC area0 */
```

```
rtdec_disable(RTDEC0, RTDEC_AREA0);
```

函数 rtdec_flag_get

函数rtdec_flag_get描述见下表：

表 3-1388. 函数 rtdec_flag_get

函数名称	rtdec_flag_get
函数原型	FlagStatus rtdec_flag_get(uint32_t rtdec_periph, uint32_t flag);
功能描述	获取RTDEC错误标志
先决条件	无
被调用函数	无
输入参数{in}	
rtdec_periph	RTDEC外设
RTDECx	x=1, 2

flag	错误标志
<i>RTDEC_FLAG_SEC_ERROR</i>	安全错误标志
<i>RTDEC_FLAG_MODE_ERROR</i>	访问模式错误标志
<i>RTDEC_FLAG_KEY_ERROR</i>	密钥错误标志
输出参数{out}	
-	-
返回值	
FlagStatus	SET或RESET

例如：

```
FlagStatus status;
```

```
/* get RTDEC err flag */
```

```
status = rtdec_flag_get(RTDEC0, RTDEC_FLAG_SEC_ERROR);
```

函数 `rtdec_flag_clear`

函数 `rtdec_flag_clear` 描述见下表：

表 3-1389. 函数 `rtdec_flag_clear`

函数名称	<code>rtdec_flag_clear</code>
函数原型	<code>void rtdec_flag_clear(uint32_t rtdec_periph, uint32_t flag);</code>
功能描述	清除RTDEC错误标志
先决条件	无
被调用函数	无
输入参数{in}	
rtdec_periph	RTDEC外设
<i>RTDECx</i>	x=1, 2
flag	错误标志
<i>RTDEC_FLAG_SEC_ERROR</i>	安全错误标志
<i>RTDEC_FLAG_MODE_ERROR</i>	访问模式错误标志
<i>RTDEC_FLAG_KEY_ERROR</i>	密钥错误标志
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* clear RTDEC err flag */
```

```
rtdec_flag_clear(RTDEC0, RTDEC_FLAG_SEC_ERROR);
```

函数 rtdec_interrupt_enable

函数rtdec_interrupt_enable描述见下表：

表 3-1390. 函数 rtdec_interrupt_enable

函数名称	rtdec_interrupt_enable
函数原型	void rtdec_interrupt_enable(uint32_t rtdec_periph, uint32_t interrupt);
功能描述	使能RTDEC中断
先决条件	无
被调用函数	无
输入参数{in}	
rtdec_periph	RTDEC外设
RTDECx	x=1, 2
interrupt	中断类型
RTDEC_INT_SEC	安全错误中断
RTDEC_INT_MODE E	访问模式错误中断
RTDEC_INT_KEY	密钥错误中断
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* enable security error interrupt */
```

```
rtdec_interrupt_enable(RTDEC0, RTDEC_INT_SEC);
```

函数 rtdec_interrupt_disable

函数rtdec_interrupt_disable描述见下表：

表 3-1391. 函数 rtdec_interrupt_disable

函数名称	rtdec_interrupt_disable
函数原型	void rtdec_interrupt_disable(uint32_t rtdec_periph, uint32_t interrupt);
功能描述	禁能RTDEC中断
先决条件	无
被调用函数	无
输入参数{in}	
rtdec_periph	RTDEC外设
RTDECx	x=1, 2

interrupt	中断类型
RTDEC_INT_SEC	安全错误中断
RTDEC_INT_MODE E	访问模式错误中断
RTDEC_INT_KEY	密钥错误中断
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* disable security error interrupt */
```

```
rtdec_interrupt_disable(RTDEC0, RTDEC_INT_SEC);
```

函数 rtdec_interrupt_flag_get

函数rtdec_interrupt_flag_get描述见下表：

表 3-1392. 函数 rtdec_interrupt_flag_get

函数名称	rtdec_interrupt_flag_get
函数原型	FlagStatus rtdec_interrupt_flag_get(uint32_t rtdec_periph, uint32_t int_flag);
功能描述	获取RTDEC中断标志
先决条件	无
被调用函数	无
输入参数{in}	
rtdec_periph	RTDEC外设
RTDECx	x=1, 2
int_flag	中断标志
RTDEC_INT_FLAG _SEC_ERROR	安全错误中断标志
RTDEC_INT_FLAG _MODE_ERROR	访问模式错误中断标志
RTDEC_INT_FLAG _KEY_ERROR	密钥错误中断标志
输出参数{out}	
-	-
返回值	
FlagStatus	SET或RESET

例如：

```
FlagStatus status;
```

```
/* get RTDEC interrupt flag */
```

```
status = rtdec_interrupt_flag_get(RTDEC0, RTDEC_INT_FLAG_SEC_ERROR);
```

函数 `rtdec_interrupt_flag_clear`

函数 `rtdec_interrupt_flag_clear` 描述见下表：

表 3-1393. 函数 `rtdec_interrupt_flag_clear`

函数名称	<code>rtdec_interrupt_flag_clear</code>
函数原型	<code>void rtdec_interrupt_flag_clear(uint32_t rtdec_periph, uint32_t int_flag);</code>
功能描述	清除RTDEC中断标志
先决条件	无
被调用函数	无
输入参数{in}	
<code>rtdec_periph</code>	RTDEC外设
<code>RTDECx</code>	$x=1, 2$
<code>int_flag</code>	中断标志
<code>RTDEC_INT_FLAG_SEC_ERROR</code>	安全错误中断标志
<code>RTDEC_INT_FLAG_MODE_ERROR</code>	访问模式错误中断标志
<code>RTDEC_INT_FLAG_KEY_ERROR</code>	密钥错误中断标志
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* clear RTDEC interrupt flag */
```

```
rtdec_interrupt_flag_clear(RTDEC0, RTDEC_INT_FLAG_SEC_ERROR);
```

3.39. SAI

串行音频接口（SAI）用于支持各种通用的音频协议，如 I2S、PCM/DSP、AC' 97、LSB 或 MSB 对齐和 TDM，它适用于单声道和立体声。章节 [3.39.1](#) 描述了 SAI 的寄存器列表，章节 [3.39.2](#) 对 SAI 库函数进行说明。

3.39.1. 外设寄存器描述

SAI寄存器列表如下表所示：

表 3-1394. SAI 寄存器

寄存器名称	寄存器描述
SAI_SYNCFG	同步配置寄存器
SAI_B0CFG0	子模块0配置寄存器0
SAI_B0CFG1	子模块0配置寄存器1
SAI_B0FCFG	子模块0帧配置寄存器
SAI_B0SCFG	子模块0slot配置寄存器
SAI_B0INTEN	子模块0中断使能寄存器
SAI_B0STAT	子模块0状态寄存器
SAI_B0INTC	子模块0中断标志清除寄存器
SAI_B0DATA	子模块0数据寄存器
SAI_B1CFG0	子模块1配置寄存器0
SAI_B1CFG1	子模块1配置寄存器1
SAI_B1FCFG	子模块1帧配置寄存器
SAI_B1SCFG	子模块1slot配置寄存器
SAI_B1INTEN	子模块1中断使能寄存器
SAI_B1STAT	子模块1状态寄存器
SAI_B1INTC	子模块1中断标志清除寄存器
SAI_B1DATA	子模块1数据寄存器
SAI_PDMCTL	PDM控制寄存器
SAI_PDMCFG	PDM配置寄存器

3.39.2. 外设库函数说明

SAI库函数列表如下表所示：

表 3-1395. SAI 库函数

库函数名称	库函数描述
sai_deinit	复位SAI
sai_struct_para_init	初始化SAI结构体参数为默认值
sai_frame_struct_para_init	初始化SAI帧结构体参数为默认值
sai_slot_struct_para_init	初始化SAI slot结构体参数为默认值
sai_init	初始化SAI结构体
sai_frame_init	初始化SAI帧结构体
sai_slot_init	初始化SAI slot结构体
sai_enable	使能SAI
sai_disable	失能SAI
sai_sdoutput_config	配置SAI串行数据输出管理
sai_monomode_config	配置SAI单声道模式
sai_companing_config	配置SAI压缩扩展模式
sai_mute_enable	使能SAI静音
sai_mute_disable	使能SAI静音

库函数名称	库函数描述
sai_mute_value_config	配置SAI静音值
sai_mute_count_config	配置SAI静音帧计数器
sai_data_transmit	SAI数据传输
sai_data_receive	SAI数据接收
sai_fifo_status_get	获取SAI FIFO 状态
sai_fifo_flush	SAI FIFO刷新
sai_dma_enable	使能SAI DMA
sai_dma_disable	失能SAI DMA
sai_sync_input_config	配置SAI同步输入选择
sai_sync_output_config	配置SAI同步输出选择
sai_pdm_enable	使能SAI PDM
sai_pdm_disable	失能SAI PDM
sai_pdm_microphone_number_config	配置SAI PDM 麦克风数量
sai_pdm_delay_config	配置SAI PDM 麦克风数据流延迟周期
sai_pdm_clk0_enable	使能SAI PDM时钟线0
sai_pdm_clk0_disable	失能SAI PDM时钟线0
sai_pdm_clk1_enable	使能SAI PDM时钟线1
sai_pdm_clk1_disable	失能SAI PDM时钟线1
sai_interrupt_enable	使能SAI中断
sai_interrupt_disable	失能SAI中断
sai_interrupt_flag_get	获取SAI中断标志
sai_interrupt_flag_clear	清除SAI中断标志
sai_flag_get	获取SAI标志
sai_flag_clear	清除SAI标志

结构体 sai_parameter_struct

表 3-1396. 结构体 sai_parameter_struct

成员名称	功能描述
operating_mode	工作模式
protocol	协议选择
data_width	数据宽度
shift_dir	移动方向
sample_edge	采样时钟边沿
sync_mode	同步模式
output_drive	输出驱动
clk_div_bypass	时钟分频器旁路
mclk_div	主时钟分频器
mclk_oversampling	主时钟过采样率
mclk_enable	主时钟使能
fifo_threshold	FIFO阈值

结构体 sai_frame_parameter_struct

表 3-1397. 结构体 sai_frame_parameter_struct

成员名称	功能描述
frame_width	帧宽度
frame_sync_width	帧同步有效宽度
frame_sync_function	帧同步功能
frame_sync_polarity	帧同步有效极性
frame_sync_offset	帧同步偏移

结构体 sai_slot_parameter_struct

表 3-1398. 结构体 sai_slot_parameter_struct

成员名称	功能描述
slot_number	一个帧中slot个数
slot_width	slot宽度
data_offset	数据偏移
slot_active	slot激活向量

枚举 sai_fifo_state_enum

表 3-1399. 枚举 sai_fifo_state_enum

成员名称	功能描述
FIFO_EMPTY	空
FIFO_EMPTY_TO_1_4_FULL	空 <FIFO级别<= 1/4满
FIFO_1_4_FULL_TO_1_2_FULL	1/4满 <FIFO级别<= 1/2满
FIFO_1_2_FULL_TO_3_4_FULL	1/2满 <FIFO级别<= 3/4满
FIFO_3_4_FULL_TO_FULL	3/4满 <FIFO级别< 全满
FIFO_FULL	全满

函数 sai_deinit

函数 sai_deinit 描述见下表:

表 3-1400. 函数 sai_deinit

函数名称	sai_deinit
函数原型	void sai_deinit(uint32_t sai_periph);
功能描述	复位RTDEC
先决条件	无
被调用函数	rcu_periph_reset_enable / rcu_periph_reset_disable
输入参数[in]	

sai_periph	SAI外设
SAIx	x=0, 1, 2
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* deinitialize the SAI0 */
```

```
sai_deinit(SAI0);
```

函数 sai_struct_para_init

函数 sai_struct_para_init 描述见下表：

表 3-1401. 函数 sai_struct_para_init

函数名称	sai_struct_para_init
函数原型	void sai_struct_para_init(sai_parameter_struct* initpara);
功能描述	初始化SAI结构体参数为默认值
先决条件	无
被调用函数	无
输入参数{in}	
initpara	初始化SAI所需的初始化数据，参考 表3-1396. 结构体sai_parameter_struct
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* initialize the parameters of SAI structure */
```

```
sai_struct_para_init sai_init_struct;
```

```
sai_struct_para_init (&sai_init_struct);
```

函数 sai_frame_struct_para_init

函数 sai_frame_struct_para_init 描述见下表：

表 3-1402. 函数 sai_frame_struct_para_init

函数名称	sai_frame_struct_para_init
函数原型	void sai_frame_struct_para_init(sai_frame_parameter_struct* initpara);
功能描述	初始化SAI帧结构体参数为默认值
先决条件	无
被调用函数	无

输入参数{in}	
initpara	初始化 SAI帧所需的初始化数据，参考 表3-1397. 结构体 sai frame parameter struct
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* deinitialize the SAI0 */
```

```
sai_deinit(SAI0);
```

函数 sai_slot_struct_para_init

函数 sai_slot_struct_para_init 描述见下表：

表 3-1403. 函数 sai_slot_struct_para_init

函数名称	sai_slot_struct_para_init
函数原型	void sai_slot_struct_para_init(sai_slot_parameter_struct* initpara);
功能描述	初始化SAI slot结构体参数为默认值
先决条件	无
被调用函数	无
输入参数{in}	
initpara	初始化 SAI slot所需的初始化数据，参考 表3-1398. 结构体 sai slot parameter struct
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* deinitialize the SAI0 */
```

```
sai_deinit(SAI0);
```

函数 sai_init

函数 sai_init 描述见下表：

表 3-1404. 函数 sai_init

函数名称	sai_init
函数原型	void sai_init(uint32_t sai_periph, uint32_t block, sai_parameter_struct* initpara);
功能描述	复位RTDEC
先决条件	无

被调用函数	无
输入参数{in}	
sai_periph	SAI外设
SAIx	x=0, 1, 2
输入参数{in}	
block	SAI块
SAI_BLOCKx	x=0, 1
输入参数{in}	
initpara	初始化 SAI 所需的初始化数据, 参考 表3-1396. 结构体sai_parameter_struct
输出参数{out}	
-	-
返回值	
-	-

例如:

```

/* initialize SAI frame */
sai_parameter_struct sai_init_structure;
sai_init_structure.operating_mode = SAI_MASTER_TRANSMITTER;
sai_init_structure.protocol = SAI_PROTOCOL_POLYMORPHIC;
sai_init_structure.data_width = SAI_DATAWIDTH_16BIT;
sai_init_structure.shift_dir = SAI_SHIFT_MSB;
sai_init_structure.sample_edge = SAI_SAMPEDGE_FALLING;
sai_init_structure.sync_mode = SAI_SYNCMODE_ASYNC;
sai_init_structure.output_drive = SAI_OUTPUT_WITH_SAIEN;
sai_init_structure.clk_div_bypass = SAI_CLKDIV_BYPASS_OFF;
sai_init_structure.mclk_div = SAI_MCLKDIV_2;
sai_init_structure.mclk_oversampling = SAI_MASTERCLK_OVERSAMP_256;
sai_init_structure.mclk_enable = SAI_MASTERCLK_DISABLE;
sai_init_structure.fifo_threshold = SAI_FIFOTH_QUARTER;
sai_init(SAI0, SAI_BLOCK0, sai_init_structure);

```

函数 sai_frame_init

函数 sai_frame_init 描述见下表:

表 3-1405. 函数 sai_frame_init

函数名称	sai_frame_init
函数原型	void sai_frame_init(uint32_t sai_periph, uint32_t block, sai_frame_parameter_struct* initpara);
功能描述	初始化SAI帧
先决条件	无
被调用函数	无
输入参数{in}	
sai_periph	SAI外设

SAI/x	x=0, 1, 2
输入参数{in}	
block	SAI块
SAI_BLOCKx	x=0, 1
输入参数{in}	
initpara	初始化SAI帧所需的初始化数据，参考 表3-1397. 结构体 sai frame parameter struct
输出参数{out}	
-	-
返回值	
-	-

例如：

```

/* initialize SAI frame */
sai_frame_parameter_struct sai_frame_init_structure;
sai_frame_init_structure.frame_width = 128;
sai_frame_init_structure.frame_sync_width = 1;
sai_frame_init_structure.frame_sync_function = SAI_FSFUNC_START;
sai_frame_init_structure.frame_sync_polarity = SAI_FSPOLR_LOW;
sai_frame_init_structure.frame_sync_offset = SAI_FSOST_BEGINNING;
sai_frame_init(SAI0, SAI_BLOCK0, sai_frame_init_structure);

```

函数 sai_slot_init

函数 sai_slot_init 描述见下表：

表 3-1406. 函数 sai_slot_init

函数名称	sai_slot_init
函数原型	void sai_slot_init(uint32_t sai_periph, uint32_t block, sai_slot_parameter_struct* initpara);
功能描述	初始化SAI slot
先决条件	无
被调用函数	无
输入参数{in}	
sai_periph	SAI外设
SAI/x	x=0, 1, 2
输入参数{in}	
block	SAI块
SAI_BLOCKx	x=0, 1
输入参数{in}	
initpara	初始化 SAI slot所需的初始化数据，参考 表3-1398. 结构体 sai slot parameter struct
输出参数{out}	

-	-
返回值	
-	-

例如：

```
/* initialize SAI frame */
sai_slot_parameter_struct sai_slot_init_structure;
sai_slot_init_structure.slot_number = 8;
sai_slot_init_structure.slot_width = SAI_SLOTWIDTH_16BIT;
sai_slot_init_structure.data_offset = 0;
sai_slot_init_structure.slot_active = SAI_SLOT_ACTIVE_ALL;

void sai_slot_init(SAI0, SAI_BLOCK0, sai_slot_init_structure);
```

函数 sai_enable

函数 sai_enable 描述见下表：

表 3-1407. 函数 sai_enable

函数名称	sai_slot_init
函数原型	void sai_enable(uint32_t sai_periph, uint32_t block);
功能描述	使能SAI
先决条件	无
被调用函数	无
输入参数{in}	
sai_periph	SAI外设
SAIx	x=0, 1, 2
输入参数{in}	
block	SAI块
SAI_BLOCKx	x=0, 1
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* enable SAI0 */

sai_enable(SAI0, SAI_BLOCK0);
```

函数 sai_disable

函数 sai_disable 描述见下表：

表 3-1408. 函数 sai_disable

函数名称	sai_disable
函数原型	void sai_disable(uint32_t sai_periph, uint32_t block);
功能描述	失能SAI
先决条件	无
被调用函数	无
输入参数{in}	
sai_periph	SAI外设
SAIx	x=0, 1, 2
输入参数{in}	
block	SAI块
SAI_BLOCKx	x=0, 1
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* disable SAI0 */
```

```
sai_disable (SAI0, SAI_BLOCK0);
```

函数 sai_sdoutput_config

函数 sai_sdoutput_config 描述见下表:

表 3-1409. 函数 sai_sdoutput_config

函数名称	sai_sdoutput_config
函数原型	void sai_sdoutput_config(uint32_t sai_periph, uint32_t block, uint32_t sdout);
功能描述	配置SAI串行数据输出管理
先决条件	无
被调用函数	无
输入参数{in}	
sai_periph	SAI外设
SAIx	x=0, 1, 2
输入参数{in}	
block	SAI块
SAI_BLOCKx	x=0, 1
输入参数{in}	
sdout	串行数据输出管理模式选择
SAI_SDLIN_DRIVE	在音频帧期间, 完全由SAI驱动SD输出
SAI_SDLIN_RELEASE	SD输出在无效slot附近释放

输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* configure SAI0 serial data output management mode selection */
sai_sdoutput_config (SAI0, SAI_BLOCK0, SAI_SDLINE_DRIVE);
```

函数 sai_monomode_config

函数 sai_monomode_config 描述见下表：

表 3-1410. 函数 sai_monomode_config

函数名称	sai_monomode_config
函数原型	void sai_monomode_config(uint32_t sai_periph, uint32_t block, uint32_t mono);
功能描述	配置SAI单声道模式
先决条件	无
被调用函数	无
输入参数{in}	
sai_periph	SAI外设
SAIx	x=0, 1, 2
输入参数{in}	
block	SAI块
SAI_BLOCKx	x=0, 1
输入参数{in}	
mono	单声道和立体声模式选择
SAI_STEREO_MODE	立体声模式
SAI_MONO_MODE	单声道模式
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* configure SAI0 mono mode */
sai_monomode_config(SAI0, SAI_BLOCK0, SAI_STEREO_MODE);
```

函数 sai_companding_config

函数 sai_companding_config 描述见下表：

表 3-1411. 函数 sai_companding_config

函数名称	sai_companding_config
函数原型	void sai_companding_config(uint32_t sai_periph, uint32_t block, uint32_t compander, uint32_t complement);
功能描述	配置SAI压缩扩展模式
先决条件	无
被调用函数	无
输入参数{in}	
sai_periph	SAI外设
SAIx	x=0, 1, 2
输入参数{in}	
block	SAI块
SAI_BLOCKx	x=0, 1
输入参数{in}	
compander	压缩扩展模式
SAI_COMPANDER_OFF	不使用压缩扩展模式
SAI_COMPANDER_ULAW	Mu-law算法
SAI_COMPANDER_ALAW	A-law算法
输入参数{in}	
complement	压缩扩展模式
SAI_COMPLEMEN T_1S	数据以1的补码表示
AI_COMPLEMENT_ 2S	数据以2的补码表示
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* configure SAI0 companding mode */
```

```
sai_companding_config(SAI0, SAI0_BLOCK0, SAI_MONO_MODE, SAI_COMPLEMENT_1S);
```

函数 sai_mute_enable

函数 sai_mute_enable 描述见下表：

表 3-1412. 函数 sai_mute_enable

函数名称	sai_mute_enable
------	-----------------

函数原型	oid sai_mute_enable(uint32_t sai_periph, uint32_t block);
功能描述	使能SAI静音
先决条件	无
被调用函数	无
输入参数{in}	
sai_periph	SAI外设
SAIx	x=0, 1, 2
输入参数{in}	
block	SAI块
SAI_BLOCKx	x=0, 1
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* enable SAI0 mute */
```

```
sai_mute_enable(SAI0, SAI_BLOCK0);
```

函数 sai_mute_disable

函数 sai_mute_disable 描述见下表:

表 3-1413. 函数 sai_disable

函数名称	sai_mute_disable
函数原型	void sai_mute_disable(uint32_t sai_periph, uint32_t block);
功能描述	失能SAI静音
先决条件	无
被调用函数	无
输入参数{in}	
sai_periph	SAI外设
SAIx	x=0, 1, 2
输入参数{in}	
block	SAI块
SAI_BLOCKx	x=0, 1
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* disable SAI0 mute */
```

```
sai_mute_disable(SAI0, SAI_BLOCK0);
```

函数 sai_mute_value_config

函数 sai_mute_value_config 描述见下表：

表 3-1414. 函数 sai_monomode_config

函数名称	sai_mute_value_config
函数原型	void sai_mute_value_config(uint32_t sai_periph, uint32_t block, uint32_t value);
功能描述	配置SAI静音值
先决条件	无
被调用函数	无
输入参数{in}	
sai_periph	SAI外设
SAIx	x=0, 1, 2
输入参数{in}	
block	SAI块
SAI_BLOCKx	x=0, 1
输入参数{in}	
value	静音值
SAI_MUTESENT_0	当静音打开时，串行数据线上发送0
SAI_MUTESENT_L ASTFREAM	当SLOTNB小于或等于2时，如果静音打开，则在重新数据线上重发上一个帧，
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* configure SAI0 mute value is 0*/
```

```
sai_mute_value_config(SAI0, SAI_BLOCK0, SAI_MUTESENT_0);
```

函数 sai_mute_count_config

函数 sai_mute_count_config 描述见下表：

表 3-1415. 函数 sai_mute_count_config

函数名称	sai_mute_count_config
函数原型	void sai_mute_count_config(uint32_t sai_periph, uint32_t block, uint32_t count);
功能描述	配置SAI静音帧计数器
先决条件	无
被调用函数	无
输入参数{in}	

sai_periph	SAI外设
<i>SAIx</i>	x=0, 1, 2
输入参数{in}	
block	SAI块
<i>SAI_BLOCKx</i>	x=0, 1
输入参数{in}	
count	0~63, 静音帧计数器
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* configure SAI0 mute frame count 16 */
```

```
sai_mute_count_config (SAI0, SAI_BLOCK0, 16);
```

函数 sai_data_transmit

函数 sai_data_transmit 描述见下表:

表 3-1416. 函数 sai_data_transmit

函数名称	sai_data_transmit
函数原型	void sai_data_transmit(uint32_t sai_periph, uint32_t block, uint32_t data);
功能描述	SAI数据传输
先决条件	无
被调用函数	无
输入参数{in}	
sai_periph	SAI外设
<i>SAIx</i>	x=0, 1, 2
输入参数{in}	
block	SAI块
<i>SAI_BLOCKx</i>	x=0, 1
输入参数{in}	
data	32位数据
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* SAI0 transmit data */
```

```
sai_data_transmit (SAI0, SAI_BLOCK0, sai0_send_array[send_n]);
```


函数 sai_data_receive

函数 sai_data_receive 描述见下表:

表 3-1417. 函数 sai_data_receive

函数名称	sai_data_receive
函数原型	uint32_t sai_data_receive(uint32_t sai_periph, uint32_t block);
功能描述	SAI数据接收
先决条件	无
被调用函数	无
输入参数{in}	
sai_periph	SAI外设
SAIx	x=0, 1, 2
输入参数{in}	
block	SAI块
SAI_BLOCKx	x=0, 1
输出参数{out}	
-	-
返回值	
uint32_t	0x00000000-0xFFFFFFFF

例如:

```
/* SAI0 receive data */
```

```
uint32_t sai_receiver;
```

```
sai_receiver = sai_data_receive (SAI0);
```

函数 sai_fifo_status_get

函数 sai_fifo_status_get 描述见下表:

表 3-1418. 函数 sai_fifo_status_get

函数名称	sai_fifo_status_get
函数原型	sai_fifo_state_enum sai_fifo_status_get(uint32_t sai_periph, uint32_t block);
功能描述	获取SAI fifo状态
先决条件	无
被调用函数	无
输入参数{in}	
sai_periph	SAI外设
SAIx	x=0, 1, 2
输入参数{in}	
block	SAI块
SAI_BLOCKx	x=0, 1
输出参数{out}	

-	-
返回值	
sai_fifo_state_enum	fifo 状态, 参考 表3-1399. 枚举sai_fifo_state_enum

例如:

```
/* get SAI0 fifo status */
```

```
sai_fifo_state_enum fifo_state;
```

```
fifo_state =sai_fifo_status_get(SAI0, SAI_BLOCK0);
```

函数 sai_fifo_flush

函数 sai_fifo_flush 描述见下表:

表 3-1419. 函数 sai_fifo_flush

函数名称	sai_fifo_flush
函数原型	void sai_fifo_flush(uint32_t sai_periph, uint32_t block);
功能描述	SAI fifo 刷新
先决条件	无
被调用函数	无
输入参数{in}	
sai_periph	SAI外设
SAI/x	x=0, 1, 2
输入参数{in}	
block	SAI块
SAI_BLOCKx	x=0, 1
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* SAI0 fifo flush */
```

```
sai_fifo_flush(SAI0);
```

函数 sai_dma_enable

函数 sai_dma_enable 描述见下表:

表 3-1420. 函数 sai_dma_enable

函数名称	sai_dma_enable
函数原型	void sai_dma_enable(uint32_t sai_periph, uint32_t block);
功能描述	使能SAI DMA

先决条件	无
被调用函数	无
输入参数{in}	
sai_periph	SAI外设
SAIx	x=0, 1, 2
输入参数{in}	
block	SAI块
SAI_BLOCKx	x=0, 1
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* enable SAI dma */
```

```
sai_dma_enable (SAI0, SAI_BLOCK0);
```

函数 sai_dma_disable

函数 sai_dma_disable 描述见下表:

表 3-1421. 函数 sai_dma_disable

函数名称	sai_dma_disable
函数原型	void sai_dma_disable(uint32_t sai_periph, uint32_t block);
功能描述	失能SAI DMA
先决条件	无
被调用函数	无
输入参数{in}	
sai_periph	SAI外设
SAIx	x=0, 1, 2
输入参数{in}	
block	SAI块
SAI_BLOCKx	x=0, 1
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* disable SAI dma */
```

```
sai_dma_disable (SAI0, SAI_BLOCK0);
```

函数 sai_sync_input_config

函数 sai_sync_input_config 描述见下表：

表 3-1422. 函数 sai_sync_input_config

函数名称	sai_sync_input_config
函数原型	void sai_sync_input_config(uint32_t sai_periph, uint32_t input);
功能描述	SAI同步输入选择
先决条件	无
被调用函数	无
输入参数{in}	
sai_periph	SAI外设
SAIx	x=0, 1, 2
输入参数{in}	
input	指定要选择哪个外部 SAI 进行同步
SAI_SYNCINPUT_SAI0	SAI1或SAI2选择来自SAI0的同步
SAI_SYNCINPUT_SAI1	SAI0或SAI2选择来自SAI1的同步
SAI_SYNCINPUT_SAI2	SAI0或SAI1选择来自SAI2的同步
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* configure SAI1 selects the synchronization coming from SAI0 */
```

```
sai_sync_input_config (SAI1, SAI_SYNCINPUT_SAI0);
```

函数 sai_sync_output_config

函数 sai_sync_output_config 描述见下表：

表 3-1423. 函数 sai_sync_output_config

函数名称	sai_sync_output_config
函数原型	void sai_sync_output_config(uint32_t sai_periph, uint32_t output);
功能描述	SAI同步输出选择
先决条件	无
被调用函数	无
输入参数{in}	
sai_periph	SAI外设
SAIx	x=0, 1, 2

输入参数{in}	
output	指定要用于其他 SAI 进一步同步的块
SAI_SYNCOUTPUT _OFF	无同步输出信号
SAI_SYNCOUTPUT _BLOCK0	音频模块0与其他SAI进行同步
SAI_SYNCOUTPUT _BLOCK1	音频模块1与其他SAI进行同步
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* configure SAI1 selects the synchronization coming from SAI0 */
```

```
sai_sync_input_config (SAI1, SAI_SYNCINPUT_SAI0);
```

函数 sai_pdm_enable

函数 sai_pdm_enable 描述见下表:

表 3-1424. 函数 sai_pdm_enable

函数名称	sai_pdm_enable
函数原型	void sai_pdm_enable(uint32_t sai_periph);
功能描述	使能SAI pdm模式
先决条件	无
被调用函数	无
输入参数{in}	
sai_periph	SAI外设
SAIx	x=0, 1, 2
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* enable SAI0 pdm mode */
```

```
sai_pdm_enable(SAI0);
```

函数 sai_pdm_disable

函数 sai_pdm_disable 描述见下表:

表 3-1425. 函数 sai_pdm_disable

函数名称	sai_pdm_disable
函数原型	void sai_pdm_disable(uint32_t sai_periph);
功能描述	失能SAI pdm模式
先决条件	无
被调用函数	无
输入参数{in}	
sai_periph	SAI外设
SAI/x	x=0, 1, 2
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* disable SAI0 pdm mode */
```

```
sai_pdm_disable(SAI0);
```

函数 sai_pdm_microphone_number_config

函数 sai_pdm_microphone_num_conf 描述见下表:

表 3-1426. 函数 sai_pdm_microphone_num_config

函数名称	sai_pdm_disable
函数原型	void sai_pdm_microphone_num_config(uint32_t sai_periph, uint32_t microphonenum);
功能描述	配置SAI pdm 模式麦克风数量
先决条件	无
被调用函数	无
输入参数{in}	
sai_periph	SAI外设
SAI/x	x=0, 1, 2
输入参数{in}	
microphonenum	2, 4, 6, 选择麦克风数量
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* configure SAI0 pdm mode microphone number 2 */
```

```
sai_pdm_microphone_num_config(SAI0, 2);
```

函数 **sai_pdm_delay_config**

函数 **sai_pdm_delay_config** 描述见下表：

表 3-1427. 函数 **sai_pdm_delay_config**

函数名称	sai_pdm_delay_config
函数原型	void sai_pdm_delay_config(uint32_t sai_periph, uint32_t microphone, uint32_t delay);
功能描述	配置SAI PDM 麦克风数据流延迟周期
先决条件	无
被调用函数	无
输入参数{in}	
sai_periph	SAI外设
SAIx	x=0, 1, 2
输入参数{in}	
microphone	指定要配置的麦克风延迟参数
SAI_PDM_MICROPHONE0_L	麦克风0左声道
SAI_PDM_MICROPHONE0_R	麦克风0右声道
SAI_PDM_MICROPHONE1_L	麦克风1左声道
SAI_PDM_MICROPHONE1_R	麦克风1右声道
SAI_PDM_MICROPHONE2_L	麦克风2左声道
SAI_PDM_MICROPHONE2_R	麦克风2右声道
SAI_PDM_MICROPHONE3_L	麦克风3左声道
SAI_PDM_MICROPHONE3_R	麦克风3右声道
输入参数{in}	
delay	0~7, 麦克风数据流延迟周期
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* configure SAI pdm mode microphone 0 left delay 1 */
```

```
sai_pdm_delay_config(SAI0, SAI_PDM_MICROPHONE0_L, 1);
```

函数 sai_pdm_clk0_enable

函数 sai_pdm_clk0_enable 描述见下表:

表 3-1428. 函数 sai_pdm_clk0_enable

函数名称	sai_pdm_clk0_enable
函数原型	void sai_pdm_clk0_disable(uint32_t sai_periph);
功能描述	使能SAI pdm模式时钟线0
先决条件	无
被调用函数	无
输入参数{in}	
sai_periph	SAI外设
SAI/x	x=0, 1, 2
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* enable SAI0 pdm mode clock line 0 */  
sai_pdm_clk0_enable(SAI0);
```

函数 sai_pdm_clk0_disable

函数 sai_pdm_clk0_disable 描述见下表:

表 3-1429. 函数 sai_pdm_clk0_disable

函数名称	sai_pdm_clk0_disable
函数原型	void sai_pdm_clk0_disable(uint32_t sai_periph);
功能描述	失能SAI pdm模式时钟线0
先决条件	无
被调用函数	无
输入参数{in}	
sai_periph	SAI外设
SAI/x	x=0, 1, 2
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* disable SAI0 pdm mode clock line 0 */  
sai_pdm_clk0_disable(SAI0);
```


函数 sai_pdm_clk1_enable

函数 sai_pdm_clk1_enable 描述见下表:

表 3-1430. 函数 sai_pdm_clk1_enable

函数名称	sai_pdm_clk1_enable
函数原型	void sai_pdm_clk1_disable(uint32_t sai_periph);
功能描述	使能SAI pdm模式时钟线1
先决条件	无
被调用函数	无
输入参数{in}	
sai_periph	SAI外设
SAI/x	x=0, 1, 2
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* enable SAI0 pdm mode clock line 1 */  
sai_pdm_clk1_enable(SAI0);
```

函数 sai_pdm_clk1_disable

函数 sai_pdm_clk1_disable 描述见下表:

表 3-1431. 函数 sai_pdm_clk1_disable

函数名称	sai_pdm_clk1_disable
函数原型	void sai_pdm_clk1_disable(uint32_t sai_periph);
功能描述	失能SAI pdm模式时钟线1
先决条件	无
被调用函数	无
输入参数{in}	
sai_periph	SAI外设
SAI/x	x=0, 1, 2
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* disable SAI0 pdm mode clock line 1 */  
sai_pdm_clk1_disable(SAI0);
```

函数 sai_interrupt_enable

函数 sai_interrupt_enable 描述见下表：

表 3-1432. 函数 sai_interrupt_enable

函数名称	sai_interrupt_enable
函数原型	void sai_interrupt_enable(uint32_t sai_periph, uint32_t block, uint32_t interrupt);
功能描述	使能SAI pdm模式时钟线1
先决条件	无
被调用函数	无
输入参数{in}	
sai_periph	SAI外设
SAIx	x=0, 1, 2
输入参数{in}	
block	SAI块
SAI_BLOCKx	x=0,1
输入参数{in}	
interrupt	指定要启用的中断
SAI_INT_OUERR	FIFO上溢或下溢中断
SAI_INT_MTDET	静音检测中断
SAI_INT_ERRCK	错误时钟中断
SAI_INT_FFREQ	FIFO请求中断
SAI_INT_ACNRDY	音频编解码器未就绪中断
SAI_INT_FSADET	帧同步提前检测中断
SAI_INT_FSPDET	帧同步滞后检测中断
输入参数{in}	
-	-
返回值	
-	-

例如：

```
/* enable SAI0 block0 mute detection interrupt */
```

```
sai_interrupt_enable(SAI0, SAI_BLOCK0, SAI_INT_MTDET);
```

函数 sai_interrupt_disable

函数 sai_interrupt_disable 描述见下表：

表 3-1433. 函数 sai_interrupt_disable

函数名称	sai_interrupt_disable
函数原型	void sai_interrupt_disable(uint32_t sai_periph, uint32_t block, uint32_t interrupt);

功能描述	失能SAI中断
先决条件	无
被调用函数	无
输入参数{in}	
sai_periph	SAI外设
SAIx	x=0, 1, 2
输入参数{in}	
block	SAI块
SAI_BLOCKx	x=0,1
输入参数{in}	
interrupt	指定要启用的中断
SAI_INT_OUERR	FIFO上溢或下溢中断
SAI_INT_MTDET	静音检测中断
SAI_INT_ERRCK	错误时钟中断
SAI_INT_FFREQ	FIFO请求中断
SAI_INT_ACNRDY	音频编解码器未就绪中断
SAI_INT_FSADET	帧同步提前检测中断
SAI_INT_FSPDET	帧同步滞后检测中断
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* disable SAI0 block0 mute detection interrupt */
```

```
sai_interrupt_disable(SAI0, SAI_BLOCK0, SAI_INT_MTDET);
```

函数 sai_interrupt_flag_get

函数 sai_interrupt_flag_get 描述见下表：

表 3-1434. 函数 sai_interrupt_flag_get

函数名称	sai_interrupt_flag_get
函数原型	FlagStatus sai_interrupt_flag_get(uint32_t sai_periph, uint32_t block, uint32_t interrupt);
功能描述	获取SAI中断标志
先决条件	无
被调用函数	无
输入参数{in}	
sai_periph	SAI外设
SAIx	x=0, 1, 2
输入参数{in}	
block	SAI块

SAI_BLOCKx	x=0,1
输入参数{in}	
interrupt	指定要获取的中断标志位状态
SAI_FLAG_OUERR	FIFO上溢或下溢中断标志
SAI_FLAG_MTDET	静音检测中断标志
SAI_FLAG_ERRCK	错误时钟中断标志
SAI_FLAG_FFREQ	FIFO请求中断标志
SAI_FLAG_ACNRDY	音频编解码器未就绪中断标志
SAI_FLAG_FSADET	帧同步提前检测中断标志
SAI_FLAG_FSPDET	帧同步滞后检测中断标志
输出参数{out}	
-	-
返回值	
FlagStatus	SET或RESET

例如：

```
/* get SAI0 block0 interrupt flag status*/
```

```
FlagStatus flag_state = RESET;
```

```
flag_state = sai_interrupt_flag_get(SAI0, SAI_BLOCK0, SAI_FLAG_MTDET);
```

函数 sai_interrupt_flag_clear

函数 sai_interrupt_flag_clear 描述见下表：

表 3-1435. 函数 sai_interrupt_flag_clear

函数名称	sai_interrupt_flag_clear
函数原型	void sai_interrupt_flag_clear(uint32_t sai_periph, uint32_t block, uint32_t interrupt);
功能描述	清除SAI中断标志
先决条件	无
被调用函数	无
输入参数{in}	
sai_periph	SAI外设
SAIx	x=0, 1, 2
输入参数{in}	
block	SAI块
SAI_BLOCKx	x=0,1
输入参数{in}	
interrupt	指定要获取的中断标志位状态

<code>SAI_FLAG_OUERR</code>	FIFO上溢或下溢中断标志
<code>SAI_FLAG_MTDET</code>	静音检测中断标志
<code>SAI_FLAG_ERRCK</code>	错误时钟中断标志
<code>SAI_FLAG_FFREQ</code>	FIFO请求中断标志
<code>SAI_FLAG_ACNRD</code> Y	音频编解码器未就绪中断标志
<code>SAI_FLAG_FSADE</code> T	帧同步提前检测中断标志
<code>SAI_FLAG_FSPDE</code> T	帧同步滞后检测中断标志
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* clear SAI0 block 0 error clock interrupt flag */
```

```
sai_flag_clear (SAI0, SAI_BLOCK0, SAI_FLAG_ERRCK);
```

函数 sai_flag_get

函数 sai_flag_get 描述见下表:

表 3-1436. 函数 sai_flag_get

函数名称	sai_flag_get
函数原型	FlagStatus sai_flag_get(uint32_t sai_periph, uint32_t block, uint32_t flag);
功能描述	获取SAI标志
先决条件	无
被调用函数	无
输入参数{in}	
sai_periph	SAI外设
SAIx	x=0, 1, 2
输入参数{in}	
block	SAI 块
SAI_BLOCKx	x=0,1
输入参数{in}	
interrupt	指定要获取的标志
<code>SAI_FLAG_OUERR</code>	FIFO上溢或下溢标志
<code>SAI_FLAG_MTDET</code>	静音检测标志
<code>SAI_FLAG_ERRCK</code>	错误时钟标志
<code>SAI_FLAG_FFREQ</code>	FIFO请求标志
<code>SAI_FLAG_ACNRD</code> Y	音频编解码器未就绪标志

SAI_FLAG_FSADE <i>T</i>	帧同步提前检测标志
SAI_FLAG_FSPDE <i>T</i>	帧同步滞后检测标志
输出参数{out}	
-	-
返回值	
FlagStatus	SET或RESET

例如:

```
/* get SAI0 block0 flag status*/
```

```
FlagStatus flag_state = RESET;
```

```
flag_state = sai_flag_get(SAI0, SAI_BLOCK0, SAI_FLAG_MTDET);
```

函数 sai_flag_clear

函数 sai_flag_clear 描述见下表:

表 3-1437. 函数 sai_flag_get

函数名称	sai_flag_clear
函数原型	void sai_flag_clear(uint32_t sai_periph, uint32_t block, uint32_t flag);
功能描述	清除SAI标志
先决条件	无
被调用函数	无
输入参数{in}	
sai_periph	SAI外设
<i>SAIx</i>	x=0, 1, 2
输入参数{in}	
block	SAI 块
<i>SAI_BLOCKx</i>	x=0,1
输入参数{in}	
interrupt	指定要清除的标志
<i>SAI_FLAG_OUERR</i>	FIFO上溢或下溢标志
<i>SAI_FLAG_MTDET</i>	静音检测标志
<i>SAI_FLAG_ERRCK</i>	错误时钟标志
<i>SAI_FLAG_ACNRD</i> <i>Y</i>	音频编解码器未就绪标志
SAI_FLAG_FSADE <i>T</i>	帧同步提前检测标志
SAI_FLAG_FSPDE <i>T</i>	帧同步滞后检测标志
输出参数{out}	

-	-
返回值	
FlagStatus	SET 或 RESET

例如:

```
/* clear SAI0 block 0 error clock flag */
```

```
sai_flag_clear (SAI0, SAI_BLOCK0, SAI_FLAG_ERRCK);
```

3.40. SDIO

安全的数字输入/输出接口（SDIO）定义了SD卡、SD I/O卡和嵌入式多媒体卡（e•MMC）主机接口，提供AHB系统总线与SD存储卡、SD I/O卡以及e•MMC设备之间的数据传输。章节[3.40.1](#)描述了SDIO的寄存器列表，章节[3.40.2](#)对SDIO库函数进行说明。

3.40.1. 外设寄存器描述

SDIO寄存器列表如下表所示:

表 3-1438. SDIO 寄存器

寄存器名称	寄存器描述
SDIO_PWRCTL	电源控制寄存器
SDIO_CLKCTL	时钟控制寄存器
SDIO_CMDAGMT	命令参数寄存器
SDIO_CMDCTL	命令控制寄存器
SDIO_RSPCMDIDX	命令索引响应寄存器
SDIO_RESPx x=0..3	响应寄存器
SDIO_DATATO	数据超时寄存器
SDIO_DATALEN	数据长度寄存器
SDIO_DATACTL	数据控制寄存器
SDIO_DATACNT	数据计数寄存器
SDIO_STAT	状态寄存器
SDIO_INTC	中断清除寄存器
SDIO_INTEN	中断使能寄存器
SDIO_ACKTIO	ACK超时寄存器
SDIO_FIFO	FIFO数据寄存器
SDIO_IDMACTL	内部DMA（IDMA）控制寄存器
SDIO_IDMASIZE	内部DMA（IDMA）缓冲大小寄存器
SDIO_IDMAADDR0	IDMA缓冲区0基地址寄存器
SDIO_IDMAADDR1	IDMA缓冲区1基地址寄存器

3.40.2. 外设库函数说明

SDIO库函数列表如下表所示：

表 3-1439. SDIO 库函数

库函数名称	库函数描述
sdio_deinit	复位SDIO
sdio_clock_config	配置SDIO时钟
sdio_clock_receive_set	设置接收时钟
sdio_hardware_clock_enable	使能硬件时钟控制
sdio_hardware_clock_disable	禁能硬件时钟控制
sdio_bus_mode_set	设置多种SDIO卡总线模式
sdio_bus_speed_set	设置SDIO总线速度
sdio_data_rate_set	设置SDIO数据率
sdio_direction_polarity_set	设置数据和命令的方向极性
sdio_power_state_set	设置SDIO电源状态
sdio_power_state_get	获取SDIO电源状态
sdio_command_response_config	配置命令和响应
sdio_wait_type_set	设置命令状态机等待类型
sdio_trans_start_enable	使能CSM传输命令模式（CSM将所有命令当做传输命令）
sdio_trans_start_disable	禁能CSM传输命令模式（CSM将所有命令当做传输命令）
sdio_trans_stop_enable	使能CSM将命令当做停止传输命令且终止DSM
sdio_trans_stop_disable	禁能CSM将命令当做停止传输命令且终止DSM
sdio_csm_enable	使能命令状态机
sdio_csm_disable	禁能命令状态机
sdio_command_index_get	获取上一次响应的命令索引
sdio_response_get	获取上一次响应的接收命令
sdio_hold_enable	使能DSM状态保持
sdio_hold_disable	禁能DSM状态保持
sdio_suspend_enable	使能命令挂起模式（CSM将命令当做挂起或恢复命令）
sdio_suspend_disable	禁能命令挂起模式（CSM将命令当做挂起或恢复命令）
sdio_data_config	配置数据超时、数据长度和数据块大小
sdio_data_transfer_config	配置数据传输模式和方向
sdio_dsm_enable	使能数据传输的数据状态机
sdio_dsm_disable	禁能数据传输的数据状态机
sdio_data_write	在发送FIFO里写入数据（一个字）
sdio_data_read	在接收FIFO里读取数据（一个字）
sdio_data_counter_get	获取要传输到卡的剩余数据字节的数目
sdio_fifo_reset_enable	使能FIFO复位
sdio_fifo_reset_disable	禁能FIFO复位
sdio_idma_set	设置IDMA的缓冲模式和大小
sdio_idma_buffer0_address_set	设置IDMA缓冲区0地址

库函数名称	库函数描述
sdio_idma_buffer1_address_set	设置IDMA缓冲区1地址
sdio_buffer_selection_get	获取IDMA缓冲区选择
sdio_idma_buffer_select	选择IDMA缓冲区
sdio_idma_enable	使能SDIO的IDMA请求
sdio_idma_disable	禁能SDIO的IDMA请求
sdio_flag_get	获取SDIO的标志位状态
sdio_flag_clear	清除SDIO的标志位状态
sdio_interrupt_enable	使能SDIO中断
sdio_interrupt_disable	禁能SDIO中断
sdio_interrupt_flag_get	获取SDIO的中断标志位状态
sdio_interrupt_flag_clear	清除SDIO的中断标志位状态
sdio_voltage_switch_enable	使能电压切换
sdio_voltage_switch_disable	禁能电压切换
sdio_voltage_switch_sequence_enable	使能电压切换序列
sdio_voltage_switch_sequence_disable	禁能电压切换序列
sdio_boot_mode_set	设置引导模式
sdio_boot_ack_enable	使能DSM引导应答
sdio_boot_ack_disable	禁能DSM引导应答
sdio_boot_acktimeout_set	设置引导应答超时周期
sdio_boot_enable	使能引导操作
sdio_boot_disable	禁能引导操作

函数 sdio_deinit

函数sdio_deinit描述见下表：

表 3-1440. 函数 sdio_deinit

函数名称	sdio_deinit
函数原形	void sdio_deinit(uint32_t sdio_periph);
功能描述	复位SDIO
先决条件	-
被调用函数	rcu_periph_reset_enable / rcu_periph_reset_disable
输入参数{in}	
sdio_periph	SDIO外设
SDIO0	选择SDIO0
SDIO1	选择SDIO1
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* deinitialize the SDIO0 */
```

```
sdio_deinit(SDIO0);
```

函数 **sdio_clock_config**

函数sdio_clock_config描述见下表：

表 3-1441. 函数 sdio_clock_config

函数名称	sdio_clock_config
函数原形	void sdio_clock_config(uint32_t sdio_periph, uint32_t clock_edge, uint32_t clock_powersave, uint32_t clock_division);
功能描述	配置SDIO时钟
先决条件	-
被调用函数	-
输入参数{in}	
sdio_periph	SDIO外设
SDIO0	选择SDIO0
SDIO1	选择SDIO1
输入参数{in}	
clock_edge	SDIO_CLK时钟边沿选择
SDIO_SDIOLCKEDGE_RISING	选择SDIOCLK的上升沿产生SDIO_CLK
SDIO_SDIOLCKEDGE_FALLING	选择SDIOCLK的下降沿产生SDIO_CLK
输入参数{in}	
clock_powersave	SDIO_CLK时钟动态开启/关闭以节省功耗
SDIO_CLOCKPWRSERVE_ENABLE	SDIO_CLK时钟在总线空闲时关闭
SDIO_CLOCKPWRSERVE_DISABLE	SDIO_CLK时钟总是开启
输入参数{in}	
clock_division	时钟分频，0-1023
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* configure the SDIO0 clock */
```

```
sdio_clock_config(SDIO0, SDIO_SDIOLCKEDGE_RISING, SDIO_CLOCKPWRSERVE_DISABLE, SD_CLK_DIV_TRANS_DSPEED);
```

函数 sdio_clock_receive_set

函数sdio_clock_receive_set描述见下表：

表 3-1442. 函数 sdio_clock_receive_set

函数名称	sdio_clock_receive_set
函数原形	void sdio_clock_receive_set(uint32_t sdio_periph, uint32_t clock_receive);
功能描述	设置接收时钟
先决条件	-
被调用函数	-
输入参数{in}	
sdio_periph	SDIO外设
SDIO0	选择SDIO0
SDIO1	选择SDIO1
输入参数{in}	
clock_receive	接收时钟选择
SDIO_RECEIVECLOCK_INCLK	选择SDIO_IN_CLK
SDIO_RECEIVECLOCK_CLKIN	选择SDIO_CLKIN
SDIO_RECEIVECLOCK_FBCLK	选择SDIO_FB_CLK
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* SDIO0 select SDIO_FB_CLK as receive clock */
```

```
sdio_clock_receive_set(SDIO0, SDIO_RECEIVECLOCK_FBCLK);
```

函数 sdio_hardware_clock_enable

函数sdio_hardware_clock_enable描述见下表：

表 3-1443. 函数 sdio_hardware_clock_enable

函数名称	sdio_hardware_clock_enable
函数原形	void sdio_hardware_clock_enable(uint32_t sdio_periph);
功能描述	使能硬件时钟控制
先决条件	-
被调用函数	-
输入参数{in}	
sdio_periph	SDIO外设

<i>SDIO0</i>	选择SDIO0
<i>SDIO1</i>	选择SDIO1
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* enable hardware clock control */
```

```
sdio_hardware_clock_enable(SDIO0);
```

函数 **sdio_hardware_clock_disable**

函数sdio_hardware_clock_disable描述见下表：

表 3-1444. 函数 sdio_hardware_clock_disable

函数名称	sdio_hardware_clock_disable
函数原形	void sdio_hardware_clock_disable(uint32_t sdio_periph);
功能描述	禁能硬件时钟控制
先决条件	-
被调用函数	-
输入参数{in}	
sdio_periph	SDIO外设
<i>SDIO0</i>	选择SDIO0
<i>SDIO1</i>	选择SDIO1
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* disable hardware clock control */
```

```
sdio_hardware_clock_disable(SDIO0);
```

函数 **sdio_bus_mode_set**

函数sdio_bus_mode_set描述见下表：

表 3-1445. 函数 sdio_bus_mode_set

函数名称	sdio_bus_mode_set
函数原形	void sdio_bus_mode_set(uint32_t sdio_periph, uint32_t bus_mode);
功能描述	设置多种SDIO卡总线模式
先决条件	-

被调用函数	-
输入参数{in}	
sdio_periph	SDIO外设
<i>SDIO0</i>	选择SDIO0
<i>SDIO1</i>	选择SDIO1
输入参数{in}	
bus_mode	SDIO卡总线模式
<i>SDIO_BUSMODE_1 BIT</i>	1位SDIO卡总线模式
<i>SDIO_BUSMODE_4 BIT</i>	4位SDIO卡总线模式
<i>SDIO_BUSMODE_8 BIT</i>	8位SDIO卡总线模式
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* set SDIO0 1-bit bus mode */
```

```
sdio_bus_mode_set(SDIO0, SDIO_BUSMODE_1BIT);
```

函数 **sdio_bus_speed_set**

函数sdio_bus_speed_set描述见下表：

表 3-1446. 函数 **sdio_bus_speed_set**

函数名称	sdio_bus_speed_set
函数原形	void sdio_bus_speed_set(uint32_t sdio_periph, uint32_t bus_speed);
功能描述	设置多种SDIO卡总线模式
先决条件	-
被调用函数	-
输入参数{in}	
sdio_periph	SDIO外设
<i>SDIO0</i>	选择SDIO0
<i>SDIO1</i>	选择SDIO1
输入参数{in}	
bus_speed	SDIO卡总线模式
<i>SDIO_BUSSPEED_ LOW</i>	DS, HS, SDR12, SDR25总线速度
<i>SDIO_BUSSPEED_ HIGH</i>	SDR50, SDR104, DDR50总线速度
输出参数{out}	

-	-
返回值	
-	-

例如：

```
/* set SDIO0 bus speed mode */
```

```
sdio_bus_speed_set(SDIO0, SDIO_BUSSPEED_HIGH);
```

函数 sdio_data_rate_set

函数sdio_data_rate_set描述见下表：

表 3-1447. 函数 sdio_data_rate_set

函数名称	sdio_data_rate_set
函数原形	void sdio_data_rate_set(uint32_t sdio_periph, uint32_t data_rate);
功能描述	设置SDIO数据率
先决条件	-
被调用函数	-
输入参数{in}	
sdio_periph	SDIO外设
SDIO0	选择SDIO0
SDIO1	选择SDIO1
输入参数{in}	
data_rate	SDIO卡总线数据率
SDIO_DATA_RATE_SDR	选择SDR模式
SDIO_DATA_RATE_DDR	选择DDR模式
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* set SDIO0 bus data rate */
```

```
sdio_data_rate_set(SDIO0, SDIO_DATA_RATE_DDR);
```

函数 sdio_direction_polarity_set

函数sdio_direction_polarity_set描述见下表：

表 3-1448. 函数 sdio_direction_polarity_set

函数名称	sdio_direction_polarity_set
------	-----------------------------

函数原形	void sdio_direction_polarity_set(uint32_t sdio_periph, uint32_t dirpl);
功能描述	设置数据和命令的方向极性
先决条件	-
被调用函数	-
输入参数{in}	
sdio_periph	SDIO外设
SDIO0	选择SDIO0
SDIO1	选择SDIO1
输入参数{in}	
dirpl	SDIO总线方向极性
SDIO_BUSSPEED_LOW	方向信号为低电平时，电压收发器驱动IOs作为输出
SDIO_BUSSPEED_HIGH	方向信号为高电平时，电压收发器驱动IOs作为输出
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* set SDIO0 bus direction polarity */
```

```
sdio_direction_polarity_set(SDIO, SDIO_DIRECTION_SIGNAL_HIGH);
```

函数 sdio_power_state_set

函数sdio_power_state_set描述见下表：

表 3-1449. 函数 sdio_power_state_set

函数名称	sdio_power_state_set
函数原形	void sdio_power_state_set(uint32_t sdio_periph, uint32_t power_state);
功能描述	设置SDIO电源状态
先决条件	-
被调用函数	-
输入参数{in}	
sdio_periph	SDIO外设
SDIO0	选择SDIO0
SDIO1	选择SDIO1
输入参数{in}	
power_state	SDIO电源状态
SDIO_POWER_ON	SDIO上电
SDIO_POWER_CYCLE	SDIO掉电再上电
SDIO_POWER_OFF	SDIO断电

<i>F</i>	
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* set SDIO0 power state */
```

```
sdio_power_state_set(SDIO0, SDIO_POWER_ON);
```

函数 **sdio_power_state_get**

函数sdio_power_state_get描述见下表:

表 3-1450. 函数 sdio_power_state_get

函数名称	sdio_power_state_get
函数原形	uint32_t sdio_power_state_get(uint32_t sdio_periph);
功能描述	获取SDIO电源状态
先决条件	-
被调用函数	-
输入参数{in}	
sdio_periph	SDIO外设
SDIO0	选择SDIO0
SDIO1	选择SDIO1
输出参数{out}	
-	-
返回值	
uint32_t	SDIO_POWER_ON / SDIO_POWER_CYCLE / SDIO_POWER_OFF

例如:

```
/* get the SDIO0 power state */
```

```
uint32_t sdio_power_value;
```

```
sdio_power_value = sdio_power_state_get(SDIO0);
```

函数 **sdio_command_response_config**

函数sdio_command_response_config描述见下表:

表 3-1451. 函数 sdio_command_response_config

函数名称	sdio_command_response_config
函数原形	void sdio_command_response_config(uint32_t sdio_periph, uint32_t cmd_index, uint32_t cmd_argument, uint32_t response_type);
功能描述	配置命令和响应

先决条件	-
被调用函数	-
输入参数{in}	
sdio_periph	SDIO外设
SDIO0	选择SDIO0
SDIO1	选择SDIO1
输入参数{in}	
cmd_index	命令索引，请参阅相关规范
输入参数{in}	
cmd_argument	命令参数，请参阅相关规范
输入参数{in}	
response_type	命令响应类型
SDIO_RESPONSETYPE_NO	无响应
SDIO_RESPONSETYPE_SHORT	短响应
SDIO_RESPONSETYPE_SHORT_NOCRC	无CRC的短响应
SDIO_RESPONSETYPE_LONG	长响应
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* CMD2(SD_CMD_ALL_SEND_CID) command response config*/
```

```
sdio_command_response_config(SDIO0, SD_CMD_ALL_SEND_CID, (uint32_t)0x0, SDIO_RESPONSETYPE_LONG);
```

函数 sdio_wait_type_set

函数sdio_wait_type_set描述见下表：

表 3-1452. 函数 sdio_wait_type_set

函数名称	sdio_wait_type_set
函数原形	void sdio_wait_type_set(uint32_t sdio_periph, uint32_t wait_type);
功能描述	设置命令状态机等待类型
先决条件	-
被调用函数	-
输入参数{in}	
sdio_periph	SDIO外设

<i>SDIO0</i>	选择SDIO0
<i>SDIO1</i>	选择SDIO1
输入参数{in}	
wait_type	等待类型
<i>SDIO_WAITTYPE_NO</i>	不等待中断
<i>SDIO_WAITTYPE_INTERRUPT</i>	等待中断
<i>SDIO_WAITTYPE_DATAEND</i>	等待数据传输结束
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* set the SDIO0 command state machine wait type */
```

```
sdio_wait_type_set(SDIO0, SDIO_WAITTYPE_NO);
```

函数 **sdio_trans_start_enable**

函数sdio_trans_start_enable描述见下表：

表 3-1453. 函数 sdio_trans_start_enable

函数名称	sdio_trans_start_enable
函数原形	void sdio_trans_start_enable(uint32_t sdio_periph);
功能描述	使能CSM传输命令模式（CSM将所有命令当做传输命令）
先决条件	-
被调用函数	-
输入参数{in}	
sdio_periph	SDIO外设
<i>SDIO0</i>	选择SDIO0
<i>SDIO1</i>	选择SDIO1
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* enable CSM transfer command */
```

```
sdio_trans_start_enable(SDIO0);
```

函数 `sdio_trans_start_disable`

函数 `sdio_trans_start_disable` 描述见下表:

表 3-1454. 函数 `sdio_trans_start_disable`

函数名称	<code>sdio_trans_start_disable</code>
函数原形	<code>void sdio_trans_start_disable(uint32_t sdio_periph);</code>
功能描述	禁能CSM传输命令模式（CSM将所有命令当做传输命令）
先决条件	-
被调用函数	-
输入参数{in}	
<code>sdio_periph</code>	SDIO外设
<code>SDIO0</code>	选择SDIO0
<code>SDIO1</code>	选择SDIO1
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* disable CSM transfer command mode*/
```

```
sdio_trans_start_disable(SDIO0);
```

函数 `sdio_trans_stop_enable`

函数 `sdio_trans_stop_enable` 描述见下表:

表 3-1455. 函数 `sdio_trans_stop_enable`

函数名称	<code>sdio_trans_stop_enable</code>
函数原形	<code>void sdio_trans_stop_enable(uint32_t sdio_periph);</code>
功能描述	使能CSM将命令当做停止传输命令且终止DSM
先决条件	-
被调用函数	-
输入参数{in}	
<code>sdio_periph</code>	SDIO外设
<code>SDIO0</code>	选择SDIO0
<code>SDIO1</code>	选择SDIO1
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* enable CSM stop command mode */
```

```
sdio_trans_stop_enable(SDIO0);
```

函数 sdio_trans_stop_disable

函数sdio_trans_stop_disable描述见下表：

表 3-1456. 函数 sdio_trans_stop_disable

函数名称	sdio_trans_stop_disable
函数原形	void sdio_trans_stop_disable(uint32_t sdio_periph);
功能描述	禁能CSM将命令当做停止传输命令且终止DSM
先决条件	-
被调用函数	-
输入参数{in}	
sdio_periph	SDIO外设
SDIO0	选择SDIO0
SDIO1	选择SDIO1
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* disable CSM stop command mode*/
```

```
sdio_trans_stop_disable(SDIO0);
```

函数 sdio_csm_enable

函数sdio_csm_enable描述见下表：

表 3-1457. 函数 sdio_csm_enable

函数名称	sdio_csm_enable
函数原形	void sdio_csm_enable(uint32_t sdio_periph);
功能描述	使能命令状态机
先决条件	-
被调用函数	-
输入参数{in}	
sdio_periph	SDIO外设
SDIO0	选择SDIO0
SDIO1	选择SDIO1
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* enable the CSM(command state machine) */
```

```
sdio_csm_enable(SDIO);
```

函数 **sdio_csm_disable**

函数sdio_csm_disable描述见下表:

表 3-1458. 函数 sdio_csm_disable

函数名称	sdio_csm_disable
函数原形	v void sdio_csm_disable(uint32_t sdio_periph);
功能描述	禁能命令状态机
先决条件	-
被调用函数	-
输入参数{in}	
sdio_periph	SDIO外设
SDIO0	选择SDIO0
SDIO1	选择SDIO1
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* disable the CSM(command state machine) */
```

```
sdio_csm_disable(SDIO0);
```

函数 **sdio_command_index_get**

函数sdio_command_index_get描述见下表:

表 3-1459. 函数 sdio_command_index_get

函数名称	sdio_command_index_get
函数原形	uint8_t sdio_command_index_get(uint32_t sdio_periph);
功能描述	获取上一次响应的命令索引
先决条件	-
被调用函数	-
输入参数{in}	
sdio_periph	SDIO外设
SDIO0	选择SDIO0
SDIO1	选择SDIO1
输出参数{out}	
-	-

返回值	
uint8_t	上一次响应的命令索引

例如：

```
/* get SDIO0 command index */
```

```
uint8_t sdio_commond_value;
```

```
sdio_commond_value = sdio_command_index_get(SDIO0);
```

函数 sdio_response_get

函数sdio_response_get描述见下表：

表 3-1460. 函数 sdio_response_get

函数名称	sdio_response_get
函数原形	uint32_t sdio_response_get(uint32_t sdio_periph, uint32_t sdio_responsex);
功能描述	获取上一次响应的接收命令
先决条件	-
被调用函数	-
输入参数{in}	
sdio_periph	SDIO外设
SDIO0	选择SDIO0
SDIO1	选择SDIO1
输入参数{in}	
responsex	SDIO响应
SDIO_RESPONSE0	卡响应 [31:0]/卡响应 [127:96]
SDIO_RESPONSE1	卡响应 [95:64]
SDIO_RESPONSE2	卡响应 [63:32]
SDIO_RESPONSE3	卡响应 [31:1]，加上位0
输出参数{out}	
-	-
返回值	
uint32_t	上一次响应的接收命令

例如：

```
/* store the CID0 numbers of SDIO0*/
```

```
uint32_t sdio_cid[0];
```

```
sdio_cid[0] = sdio_response_get(SDIO, SDIO_RESPONSE0);
```

函数 sdio_hold_enable

函数sdio_hold_enable描述见下表：

表 3-1461. 函数 `sdio_hold_enable`

函数名称	<code>sdio_hold_enable</code>
函数原形	<code>void sdio_hold_enable(uint32_t sdio_periph);</code>
功能描述	使能DSM状态保持
先决条件	-
被调用函数	-
输入参数{in}	
<code>sdio_periph</code>	SDIO外设
<code>SDIO0</code>	选择SDIO0
<code>SDIO1</code>	选择SDIO1
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* enable DSM status hold */
```

```
sdio_hold_enable (SDIO);
```

函数 `sdio_hold_disable`

函数`sdio_hold_disable`描述见下表：

表 3-1462. 函数 `sdio_hold_disable`

函数名称	<code>sdio_hold_disable</code>
函数原形	<code>void sdio_hold_disable(uint32_t sdio_periph);</code>
功能描述	禁能DSM状态保持
先决条件	-
被调用函数	-
输入参数{in}	
<code>sdio_periph</code>	SDIO外设
<code>SDIO0</code>	选择SDIO0
<code>SDIO1</code>	选择SDIO1
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* disable DSM status hold */
```

```
sdio_hold_disable (SDIO)
```

函数 `sdio_suspend_enable`

函数`sdio_suspend_enable`描述见下表:

表 3-1463. 函数 `sdio_suspend_enable`

函数名称	<code>sdio_suspend_enable</code>
函数原形	<code>Void sdio_suspend_enable(uint32_t sdio_periph);</code>
功能描述	使能命令挂起模式（CSM将命令当做挂起或恢复命令）
先决条件	-
被调用函数	-
输入参数{in}	
<code>sdio_periph</code>	SDIO外设
<code>SDIO0</code>	选择SDIO0
<code>SDIO1</code>	选择SDIO1
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* enable CSM suspend mode */
```

```
sdio_suspend_enable(SDIO0);
```

函数 `sdio_suspend_disable`

函数`sdio_suspend_disable`描述见下表:

表 3-1464. 函数 `sdio_suspend_disable`

函数名称	<code>sdio_suspend_disable</code>
函数原形	<code>void sdio_suspend_disable (uint32_t sdio_periph);</code>
功能描述	禁能命令挂起模式（CSM将命令当做挂起或恢复命令）
先决条件	-
被调用函数	-
输入参数{in}	
<code>sdio_periph</code>	SDIO外设
<code>SDIO0</code>	选择SDIO0
<code>SDIO1</code>	选择SDIO1
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/*disable CSM suspend mode */
```


sdio_suspend_disable(SDIO0);

函数 sdio_data_config

函数sdio_data_config描述见下表:

表 3-1465. 函数 sdio_data_config

函数名称	sdio_data_config
函数原形	void sdio_data_config(uint32_t sdio_periph, uint32_t data_timeout, uint32_t data_length, uint32_t data_blocksize);
功能描述	配置数据超时、数据长度和数据块大小
先决条件	-
被调用函数	-
输入参数{in}	
sdio_periph	SDIO外设
SDIO0	选择SDIO0
SDIO1	选择SDIO1
输入参数{in}	
data_timeout	卡总线时钟周期中的数据超时周期
输入参数{in}	
data_length	要传输的数据字节数
输入参数{in}	
data_blocksize	块传输中数据块的大小
SDIO_DATABLOCK SIZE_1BYTE	块大小 = 1字节
SDIO_DATABLOCK SIZE_2BYTES	块大小 = 2字节
SDIO_DATABLOCK SIZE_4BYTES	块大小 = 4字节
SDIO_DATABLOCK SIZE_8BYTES	块大小 = 8字节
SDIO_DATABLOCK SIZE_16BYTES	块大小 = 16字节
SDIO_DATABLOCK SIZE_32BYTES	块大小 = 32字节
SDIO_DATABLOCK SIZE_64BYTES	块大小 = 64字节
SDIO_DATABLOCK SIZE_128BYTES	块大小 = 128字节
SDIO_DATABLOCK SIZE_256BYTES	块大小 = 256字节
SDIO_DATABLOCK SIZE_512BYTES	块大小 = 512字节

SDIO_DATABLOCK SIZE_1024BYTES	块大小 = 1024字节
SDIO_DATABLOCK SIZE_2048BYTES	块大小 = 2048字节
SDIO_DATABLOCK SIZE_4096BYTES	块大小 = 4096字节
SDIO_DATABLOCK SIZE_8192BYTES	块大小 = 8192字节
SDIO_DATABLOCK SIZE_16384BYTES	块大小 = 16384字节
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* configure SDIO0 data */
```

```
sdio_data_config(SDIO0, 0, 0, SDIO_DATABLOCKSIZE_1BYTE);
```

函数 sdio_data_transfer_config

函数sdio_data_transfer_config描述见下表：

表 3-1466. 函数 sdio_data_transfer_config

函数名称	sdio_data_transfer_config
函数原形	void sdio_data_transfer_config(uint32_t sdio_periph, uint32_t transfer_mode, uint32_t transfer_direction);
功能描述	配置数据传输模式和方向
先决条件	-
被调用函数	-
输入参数{in}	
sdio_periph	SDIO外设
SDIO0	选择SDIO0
SDIO1	选择SDIO1
输入参数{in}	
transfer_mode	数据传输模式
SDIO_TRANSMOD E_BLOCK	块传输模式
SDIO_TRANSMOD E_MULTIBYTE	多字节传输模式（只适用于 SD/SD I/O 卡）
SDIO_TRANSMOD E_STREAM	流传输模式（只适用于 eMMC 卡）
SDIO_TRANSMOD	需要CMD12终止的多块传输模式

<i>E_BLOCKCMD12</i>	
输入参数{in}	
transfer_direction	数据传输方向
<i>SDIO_TRANSDIRECTION_TOCARD</i>	写数据到卡上
<i>SDIO_TRANSDIRECTION_TOSDIO</i>	从卡中读取数据
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* configure SDIO0 data transmisson */
```

```
sdio_data_transfer_config(SDIO0, SDIO_TRANSDIRECTION_TOCARD,
SDIO_TRANSMODE_BLOCKCOUNT);
```

函数 **sdio_dsm_enable**

函数sdio_dsm_enable描述见下表:

表 3-1467. 函数 sdio_dsm_enable

函数名称	sdio_dsm_enable
函数原形	void sdio_dsm_enable(uint32_t sdio_periph);
功能描述	使能数据传输的数据状态机
先决条件	-
被调用函数	-
输入参数{in}	
sdio_periph	SDIO外设
<i>SDIO0</i>	选择SDIO0
<i>SDIO1</i>	选择SDIO1
输入参数{in}	
-	-
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* enable the DSM(data state machine) */
```

```
sdio_dsm_enable(SDIO0);
```

函数 **sdio_dsm_disable**

函数sdio_dsm_disable描述见下表:

表 3-1468. 函数 **sdio_dsm_disable**

函数名称	sdio_dsm_disable
函数原形	void sdio_dsm_disable(uint32_t sdio_periph);
功能描述	禁能数据传输的数据状态机
先决条件	-
被调用函数	-
输入参数{in}	
sdio_periph	SDIO外设
SDIO0	选择SDIO0
SDIO1	选择SDIO1
输入参数{in}	
-	-
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* disable the DSM(data state machine) */
```

```
sdio_dsm_disable(SDIO0);
```

函数 **sdio_data_write**

函数sdio_data_write描述见下表:

表 3-1469. 函数 **sdio_data_write**

函数名称	sdio_data_write
函数原形	void sdio_data_write(uint32_t sdio_periph ,uint32_t data);
功能描述	在发送FIFO里写入数据（一个字）
先决条件	-
被调用函数	-
输入参数{in}	
sdio_periph	SDIO外设
SDIO0	选择SDIO0
SDIO1	选择SDIO1
输入参数{in}	
data	往卡里写入32位数据
输出参数{out}	
-	-

返回值	
-	-

例如：

```
/* write data(one word) to the transmit FIFO */
```

```
sdio_data_write(SDIO0, 0x0000 0001);
```

函数 sdio_data_read

函数sdio_data_read描述见下表：

表 3-1470. 函数 sdio_data_read

函数名称	sdio_data_read
函数原形	uint32_t sdio_data_read(uint32_t sdio_periph);
功能描述	在接收FIFO里读取数据（一个字）
先决条件	-
被调用函数	-
输入参数{in}	
sdio_periph	SDIO外设
SDIO0	选择SDIO0
SDIO1	选择SDIO1
输入参数{in}	
-	-
输出参数{out}	
-	-
返回值	
uint32_t	接收的数据

例如：

```
/* read data(one word) from the receive FIFO */
```

```
sdio_data_read(SDIO0);
```

函数 sdio_data_counter_get

函数sdio_data_counter_get描述见下表：

表 3-1471. 函数 sdio_data_counter_get

函数名称	sdio_data_counter_get
函数原形	uint32_t sdio_data_counter_get(uint32_t sdio_periph);
功能描述	获取要传输到卡的剩余数据字节的数目
先决条件	-
被调用函数	-
输入参数{in}	

sdio_periph	SDIO外设
<i>SDIO0</i>	选择SDIO0
<i>SDIO1</i>	选择SDIO1
输出参数{out}	
-	-
返回值	
uint32_t	要传输的剩余数据字节数

例如：

```
/* get the number of remaining data bytes to be transferred to card */
```

```
uint32_t sdio_data_value;
```

```
sdio_data_value = sdio_data_counter_get(SDIO0);
```

函数 **sdio_fifo_reset_enable**

函数sdio_fifo_reset_enable描述见下表：

表 3-1472. 函数 sdio_fifo_reset_enable

函数名称	sdio_fifo_reset_enable
函数原形	void sdio_fifo_reset_enable(uint32_t sdio_periph);
功能描述	使能FIFO复位
先决条件	-
被调用函数	-
输入参数{in}	
sdio_periph	SDIO外设
<i>SDIO0</i>	选择SDIO0
<i>SDIO1</i>	选择SDIO1
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* enable reset FIFO */
```

```
sdio_fifo_reset_enable(SDIO0);
```

函数 **sdio_fifo_reset_disable**

函数sdio_fifo_reset_disable描述见下表：

表 3-1473. 函数 sdio_fifo_reset_disable

函数名称	sdio_fifo_reset_disable
函数原形	void sdio_fifo_reset_disable(uint32_t sdio_periph);

功能描述	禁能FIFO复位
先决条件	-
被调用函数	-
输入参数{in}	
sdio_periph	SDIO外设
<i>SDIO0</i>	选择SDIO0
<i>SDIO1</i>	选择SDIO1
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* disable reset FIFO */
```

```
sdio_fifo_reset_disable(SDIO0);
```

函数 sdio_idma_set

函数sdio_idma_set描述见下表：

表 3-1474. 函数 sdio_idma_set

函数名称	sdio_idma_set
函数原形	void sdio_idma_set(uint32_t sdio_periph, uint32_t buffer_mode, uint32_t buffer_size);
功能描述	设置IDMA的缓冲模式和大小
先决条件	-
被调用函数	-
输入参数{in}	
sdio_periph	SDIO外设
<i>SDIO0</i>	选择SDIO0
<i>SDIO1</i>	选择SDIO1
输入参数{in}	
buffer_mode	设置SDIO的缓冲模式
<i>SDIO_IDMA_SINGL E_BUFFER</i>	选择单缓冲模式
<i>SDIO_IDMA_DOUB LE_BUFFER</i>	选择双缓冲模式
输入参数{in}	
buffer_size	设置IDMA缓冲区的大小，范围：0-0xFF
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* set the SDIO0 IDMA */
```

```
sdio_idma_set(SDIO, SDIO_IDMA_SINGLE_BUFFER, (uint32_t)0x20);
```

函数 **sdio_idma_buffer0_address_set**

函数sdio_idma_buffer0_address_set描述见下表：

表 3-1475. 函数 sdio_idma_buffer0_address_set

函数名称	sdio_idma_buffer0_address_set
函数原形	void sdio_idma_buffer0_address_set(uint32_t sdio_periph, uint32_t buffer_address);
功能描述	设置IDMA缓冲区0地址
先决条件	-
被调用函数	-
输入参数{in}	
sdio_periph	SDIO外设
SDIO0	选择SDIO0
SDIO1	选择SDIO1
输入参数{in}	
buffer_address	缓冲区0的地址，值是8的倍数
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* set IDMA buffer0 address */
```

```
sdio_idma_buffer0_address_set(SDIO0, (uint32_t)(0x0000 0010));
```

函数 **sdio_idma_buffer1_address_set**

函数sdio_idma_buffer1_address_set描述见下表：

表 3-1476. 函数 sdio_idma_buffer1_address_set

函数名称	sdio_idma_buffer1_address_set
函数原形	void sdio_idma_buffer1_address_set(uint32_t sdio_periph, uint32_t buffer_address);
功能描述	设置IDMA缓冲区1地址
先决条件	-
被调用函数	-
输入参数{in}	
sdio_periph	SDIO外设

<i>SDIO0</i>	选择SDIO0
<i>SDIO1</i>	选择SDIO1
输入参数{in}	
buffer_address	缓冲区1的地址，值是8的倍数
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* set IDMA buffer1 address */
```

```
sdio_idma_buffer1_address_set(SDIO0, (uint32_t)(0x0000 0010));
```

函数 **sdio_buffer_selection_get**

函数sdio_buffer_selection_get描述见下表：

表 3-1477. 函数 **sdio_buffer_selection_get**

函数名称	sdio_buffer_selection_get
函数原形	void sdio_buffer_selection_get(uint32_t sdio_periph);
功能描述	获取IDMA缓冲区选择
先决条件	-
被调用函数	-
输入参数{in}	
sdio_periph	SDIO外设
<i>SDIO0</i>	选择SDIO0
<i>SDIO1</i>	选择SDIO1
输出参数{out}	
-	-
返回值	
uint32_t	SDIO_IDMA_BUFER0_SELECTION/SDIO_IDMA_BUFER1_SELECTION

例如：

```
/* get the IDMA of SDIO0 */
```

```
uint32_t sdio_buffer_selection_value;
```

```
sdio_buffer_selection_value = sdio_buffer_selection_get(SDIO0);
```

函数 **sdio_idma_buffer_select**

函数sdio_idma_buffer_select描述见下表：

表 3-1478. 函数 **sdio_idma_buffer_select**

函数名称	sdio_idma_buffer_select
------	-------------------------

函数原形	void sdio_idma_buffer_select(uint32_t sdio_periph, uint32_t buffer_select);
功能描述	选择IDMA缓冲区
先决条件	-
被调用函数	-
输入参数{in}	
sdio_periph	SDIO外设
SDIO0	选择SDIO0
SDIO1	选择SDIO1
输入参数{in}	
buffer_select	选择缓冲地址
SDIO_IDMA_BUFF ER0	选择缓冲区0
SDIO_IDMA_BUFF ER0	选择缓冲区1
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* set the IDMA of SDIO0 */
```

```
sdio_idma_buffer_select(SDIO0, SDIO_IDMA_BUFFER0);
```

函数 sdio_idma_enable

函数sdio_idma_enable描述见下表：

表 3-1479. 函数 sdio_idma_enable

函数名称	sdio_idma_enable
函数原形	void sdio_idma_enable(uint32_t sdio_periph);
功能描述	使能SDIO的IDMA请求
先决条件	-
被调用函数	-
输入参数{in}	
sdio_periph	SDIO外设
SDIO0	选择SDIO0
SDIO1	选择SDIO1
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* enable the IDMA of SDIO0 */
```

```
sdio_idma_enable(SDIO0);
```

函数 **sdio_idma_disable**

函数sdio_idma_disable描述见下表：

表 3-1480. 函数 sdio_idma_disable

函数名称	sdio_idma_disable
函数原形	void sdio_idma_disable(uint32_t sdio_periph);
功能描述	禁能SDIO的IDMA请求
先决条件	-
被调用函数	-
输入参数{in}	
sdio_periph	SDIO外设
SDIO0	选择SDIO0
SDIO1	选择SDIO1
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* disable the IDMA of SDIO0 */
```

```
sdio_idma_disable(SDIO0);
```

函数 **sdio_flag_get**

函数sdio_flag_get描述见下表：

表 3-1481. 函数 sdio_flag_get

函数名称	sdio_flag_get
函数原形	FlagStatus sdio_flag_get(uint32_t sdio_periph, uint32_t flag);
功能描述	获取SDIO的标志位状态
先决条件	-
被调用函数	-
输入参数{in}	
sdio_periph	SDIO外设
SDIO0	选择SDIO0
SDIO1	选择SDIO1
输入参数{in}	
flag	SDIO标志位状态
SDIO_FLAG_CCRC	命令响应已接收（CRC检测失败）

ERR	
SDIO_FLAG_DTCRCERR	数据块已发送/已接收（CRC检测失败）
SDIO_FLAG_CMDTMO	命令响应超时
SDIO_FLAG_DTTMO	数据超时
SDIO_FLAG_TXUR	发送FIFO下溢错误发生
SDIO_FLAG_RXOR	接收FIFO上溢错误发生
SDIO_FLAG_CMDRECV	命令响应已接收（CRC检测通过）
SDIO_FLAG_CMDS	命令已发送（不需响应）
SDIO_FLAG_DTHOLD	数据传输保持
SDIO_FLAG_DTBLKEND	数据块已发送/已接收（CRC检测通过）
SDIO_FLAG_DTBABORT	数据传输被CMD12中止
SDIO_FLAG_CMDS	命令通道激活状态
SDIO_FLAG_DATS	数据通道激活状态
SDIO_FLAG_TFH	发送FIFO半空：至少还有8个字可被写入到FIFO中
SDIO_FLAG_RFH	接收FIFO半满：FIFO中至少还有8个字可被读取
SDIO_FLAG_TFF	发送FIFO为满
SDIO_FLAG_RFF	接收FIFO为满
SDIO_FLAG_TFE	发送FIFO为空
SDIO_FLAG_RFE	接收FIFO为空
SDIO_FLAG_DAT0BSY	DAT0线信号保持低电平
SDIO_FLAG_DAT0BSYEND	DAT0线信号从繁忙到转备好
SDIO_FLAG_SDIOINT	SD I/O中断已接收
SDIO_FLAG_ACKFAIL	引导确认接收且检查错误
SDIO_FLAG_ACKTO	引导确认超时
SDIO_FLAG_	电压切换时关键时序完成

<i>VOLSWEND</i>	
<i>SDIO_FLAG_CLKS TOP</i>	电压切换期间SDIO_CLK停止
<i>SDIO_FLAG_IDMA ERR</i>	IDMA传输错误
<i>SDIO_FLAG_IDMA END</i>	IDMA传输结束
输出参数{out}	
-	-
返回值	
FlagStatus	SET 或 RESET

例如:

```
/* get the flags state of SDIO0 */
```

```
FlagStatus flag_value;
```

```
flag_value = sdio_flag_get(SDIO0, SDIO_FLAG_RFH);
```

函数 **sdio_flag_clear**

函数sdio_flag_clear描述见下表:

表 3-1482. 函数 sdio_flag_clear

函数名称	sdio_flag_clear
函数原形	void sdio_flag_clear(uint32_t sdio_periph, uint32_t flag);
功能描述	清除SDIO的标志位状态
先决条件	-
被调用函数	-
输入参数{in}	
sdio_periph	SDIO外设
<i>SDIO0</i>	选择SDIO0
<i>SDIO1</i>	选择SDIO1
输入参数{in}	
flag	SDIO标志位状态
<i>SDIO_FLAG_CCRC ERR</i>	命令响应已接收 (CRC检测失败)
<i>SDIO_FLAG_DTCR CERR</i>	数据块已发送/已接收 (CRC检测失败)
<i>SDIO_FLAG_CMDT MOUT</i>	命令响应超时
<i>SDIO_FLAG_DTTM OUT</i>	数据超时
<i>SDIO_FLAG_TXUR</i>	发送FIFO下溢错误发生

<i>E</i>	
<i>SDIO_FLAG_RXOR</i> <i>E</i>	接收FIFO上溢错误发生
<i>SDIO_FLAG_CMDR</i> <i>ECV</i>	命令响应已接收（CRC检测通过）
<i>SDIO_FLAG_CMDS</i> <i>END</i>	命令已发送（不需响应）
<i>SDIO_FLAG_DTEN</i> <i>D</i>	数据结束（数据计数器，SDIO_DATACNT为零）
<i>SDIO_FLAG_DTHO</i> <i>LD</i>	数据传输保持
<i>SDIO_FLAG_DTB</i> <i>KEND</i>	数据块已发送/已接收（CRC检测通过）
<i>SDIO_FLAG_DTAB</i> <i>ORT</i>	数据传输被CMD12中止
<i>SDIO_FLAG_DAT0</i> <i>BSYEND</i>	DAT0线信号从繁忙到准备好
<i>SDIO_FLAG_SDIOI</i> <i>NT</i>	SD I/O中断已接收
<i>SDIO_FLAG_ACKF</i> <i>AIL</i>	引导确认接收且检查错误
<i>SDIO_FLAG_ACKT</i> <i>O</i>	引导确认超时
<i>SDIO_FLAG_VOLS</i> <i>WEND</i>	电压切换时关键时序完成
<i>SDIO_FLAG_CLKS</i> <i>TOP</i>	电压切换期间SDIO_CLK 停止
<i>SDIO_FLAG_IDMA</i> <i>ERR</i>	IDMA传输错误
<i>SDIO_FLAG_IDMA</i> <i>END</i>	IDMA传输结束
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* clear the pending flags of SDIO0 */
```

```
sdio_flag_clear(SDIO0, SDIO_FLAG_DTCRCERR);
```

函数 sdio_interrupt_enable

函数sdio_interrupt_enable描述见下表：

表 3-1483. 函数 `sdio_interrupt_enable`

函数名称	<code>sdio_interrupt_enable</code>
函数原形	<code>void sdio_interrupt_enable(uint32_t sdio_periph, uint32_t int_flag);</code>
功能描述	使能SDIO中断
先决条件	-
被调用函数	-
输入参数{in}	
sdio_periph	SDIO外设
<code>SDIO0</code>	选择SDIO0
<code>SDIO1</code>	选择SDIO1
输入参数{in}	
int_flag	SDIO中断标志位状态
<code>SDIO_INT_CCRCE</code> <code>RR</code>	命令响应CRC错误中断
<code>SDIO_INT_DTCRC</code> <code>ERR</code>	数据CRC错误中断
<code>SDIO_INT_CMDTM</code> <code>OUT</code>	命令响应超时中断
<code>SDIO_INT_DTTMO</code> <code>UT</code>	数据超时中断
<code>SDIO_INT_TXURE</code>	发送FIFO下溢错误中断
<code>SDIO_INT_RXORE</code>	接收FIFO上溢错误中断
<code>SDIO_INT_CMDRE</code> <code>CV</code>	命令响应已接收中断
<code>SDIO_INT_CMDSE</code> <code>ND</code>	命令已发送中断
<code>SDIO_INT_DTEND</code>	数据结束中断
<code>SDIO_INT_DTHOL</code> <code>D</code>	数据保持中断
<code>SDIO_INT_DTBLE</code> <code>ND</code>	数据块已发送/已接收中断
<code>SDIO_INT_DTABO</code> <code>RT</code>	CMD12终止中断
<code>SDIO_INT_TFH</code>	发送FIFO半空中断
<code>SDIO_INT_RFH</code>	接收FIFO半满中断
<code>SDIO_INT_RFF</code>	接收FIFO满中断
<code>SDIO_INT_RFE</code>	接收FIFO空中断
<code>SDIO_INT_DAT0BS</code> <code>YEND</code>	DAT0线信号从繁忙到准备好中断
<code>SDIO_INT_SDIOIN</code> <code>T</code>	SD I/O中断已接收中断中断
<code>SDIO_INT_ACKFAI</code>	引导确认接收且检查错误中断

<i>L</i>	
<i>SDIO_INT_ACKTO</i>	引导确认超时中断
<i>SDIO_INT_VOLSW END</i>	电压切换时关键时序完成中断
<i>SDIO_INT_CLKSTO P</i>	电压切换期间SDIO_CLK停止中断
<i>SDIO_INT_IDMAER R</i>	IDMA传输错误中断
<i>SDIO_INT_IDMAEN D</i>	IDMA传输错误中断
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* enable the SDIO0 corresponding interrupts */
```

```
sdio_interrupt_enable(SDIO0, SDIO_INT_CCRCERR | SDIO_INT_DTTMOUT |  
SDIO_INT_RXORE | SDIO_INT_DTEND | SDIO_INT_STBITE);
```

函数 `sdio_interrupt_disable`

函数 `sdio_interrupt_disable` 描述见下表：

表 3-1484. 函数 `sdio_interrupt_disable`

函数名称	<code>sdio_interrupt_disable</code>
函数原形	<code>void sdio_interrupt_disable(uint32_t sdio_periph, uint32_t int_flag);</code>
功能描述	禁能SDIO中断
先决条件	-
被调用函数	-
输入参数{in}	
sdio_periph	SDIO外设
<i>SDIO0</i>	选择SDIO0
<i>SDIO1</i>	选择SDIO1
输入参数{in}	
int_flag	SDIO中断标志位状态
<i>SDIO_INT_CCRCE RR</i>	命令响应CRC错误中断
<i>SDIO_INT_DTCRC ERR</i>	数据CRC错误中断
<i>SDIO_INT_CMDTM OUT</i>	命令响应超时中断
<i>SDIO_INT_DTTMO</i>	数据超时中断

UT	
SDIO_INT_TXURE	发送FIFO下溢错误中断
SDIO_INT_RXORE	接收FIFO上溢错误中断
SDIO_INT_CMDRE CV	命令响应已接收中断
SDIO_INT_CMDSE ND	命令已发送中断
SDIO_INT_DTEND	数据结束中断
SDIO_INT_DTHOL D	数据保持中断
SDIO_INT_DTBLKE ND	数据块已发送/已接收中断
SDIO_INT_DTABO RT	CMD12终止中断
SDIO_INT_TFH	发送FIFO半空中断
SDIO_INT_RFH	接收FIFO半满中断
SDIO_INT_RFF	接收FIFO满中断
SDIO_INT_TFE	发送FIFO空中断
SDIO_INT_DAT0BS YEND	DAT0线信号从繁忙到准备好中断
SDIO_INT_SDIOIN T	SD I/O中断已接收中断中断
SDIO_INT_ACKFAI L	引导确认接收且检查错误中断
SDIO_INT_ACKTO	引导确认超时中断
SDIO_INT_VOLSW END	电压切换时关键时序完成中断
SDIO_INT_CLKSTO P	电压切换期间SDIO_CLK停止中断
SDIO_INT_IDMAER R	IDMA传输错误中断
SDIO_INT_IDMAEN D	IDMA传输错误中断
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* disable the SDIO0 interrupt */
```

```
sdio_interrupt_disable(SDIO0, SDIO_INT_DTCCRERR);
```

函数 **sdio_interrupt_flag_get**

函数sdio_interrupt_flag_get描述见下表:

表 3-1485. 函数 **sdio_interrupt_flag_get**

函数名称	sdio_interrupt_flag_get
函数原形	FlagStatus sdio_interrupt_flag_get(uint32_t sdio_periph, uint32_t int_flag);
功能描述	获取SDIO的中断标志位状态
先决条件	-
被调用函数	-
输入参数{in}	
sdio_periph	SDIO外设
SDIO0	选择SDIO0
SDIO1	选择SDIO1
输入参数{in}	
int_flag	SDIO中断标志位状态
SDIO_INT_FLAG_C CRCERR	命令响应CRC错误中断
SDIO_INT_FLAG_D TCRCERR	数据CRC错误中断
SDIO_INT_FLAG_C MDTMOUT	命令响应超时中断
SDIO_INT_FLAG_D TTMOUT	数据超时中断
SDIO_INT_FLAG_T XURE	发送FIFO下溢错误中断
SDIO_INT_FLAG_R XORE	接收FIFO上溢错误中断
SDIO_INT_FLAG_C MDRECV	命令响应已接收中断
SDIO_INT_FLAG_C MDSEND	命令已发送中断
SDIO_INT_FLAG_D TEND	数据结束中断
SDIO_INT_FLAG_D THOLD	数据保持中断
SDIO_INT_FLAG_D TBLKEND	数据块已发送/已接收中断
SDIO_INT_FLAG_D TABORT	CMD12终止中断
SDIO_INT_FLAG_T FH	发送FIFO半空中断
SDIO_INT_FLAG_R	接收FIFO半满中断

<i>FH</i>	
<i>SDIO_INT_FLAG_RFF</i>	接收FIFO满中断
<i>SDIO_INT_FLAG_TFE</i>	发送FIFO空中断
<i>SDIO_INT_FLAG_DAT0BSYEND</i>	DAT0线信号从繁忙到准备好中断
<i>SDIO_INT_FLAG_SDIOINT</i>	SD I/O中断已接收中断
<i>SDIO_INT_FLAG_ACKFAIL</i>	引导确认接收且检查错误中断
<i>SDIO_INT_FLAG_ACKTO</i>	引导确认超时中断
<i>SDIO_INT_FLAG_VOLSWEND</i>	电压切换时关键时序完成中断
<i>SDIO_INT_FLAG_CLKSTOP</i>	电压切换期间SDIO_CLK停止中断
<i>SDIO_INT_FLAG_IDMAEND</i>	IDMA传输错误中断
输出参数{out}	
-	-
返回值	
FlagStatus	SET 或 RESET

例如：

```
/* get the interrupt flags state of SDIO0 */
```

```
FlagStatus flag_value;
```

```
flag_value = sdio_interrupt_flag_get(SDIO0, SDIO_INT_FLAG_DTEND);
```

函数 sdio_interrupt_flag_clear

函数sdio_interrupt_flag_clear描述见下表：

表 3-1486. 函数 sdio_interrupt_flag_clear

函数名称	sdio_interrupt_flag_clear
函数原形	void sdio_interrupt_flag_clear(uint32_t sdio_periph, uint32_t int_flag);
功能描述	清除SDIO的中断标志位状态
先决条件	-
被调用函数	-
输入参数{in}	
sdio_periph	SDIO外设
SDIO0	选择SDIO0

SDIO1	选择SDIO1
输入参数{in}	
int_flag	SDIO中断标志位状态
SDIO_INT_FLAG_C CRCERR	命令响应CRC错误中断
SDIO_INT_FLAG_D TCRCERR	数据CRC错误中断
SDIO_INT_FLAG_C MDTMOUT	命令响应超时中断
SDIO_INT_FLAG_D TTMOUT	数据超时中断
SDIO_INT_FLAG_T XURE	发送FIFO下溢错误中断
SDIO_INT_FLAG_R XORE	接收FIFO上溢错误中断
SDIO_INT_FLAG_C MDRECV	命令响应已接收中断
SDIO_INT_FLAG_C MDSEND	命令已发送中断
SDIO_INT_FLAG_D TEND	数据结束中断
SDIO_INT_FLAG_D THOLD	起始位错误中断
SDIO_INT_FLAG_D TBLKEND	数据块已发送/已接收中断
SDIO_INT_FLAG_D TABORT	CMD12终止中断
SDIO_INT_FLAG_ DAT0BSYEND	DAT0线信号从繁忙到准备好中断
SDIO_INT_FLAG_S DIOINT	SD I/O中断已接收中断
SDIO_INT_FLAG_A CKFAIL	引导确认接收且检查错误中断
SDIO_INT_FLAG_A CKTO	引导确认超时中断
SDIO_INT_FLAG_ VOLSWEND	电压切换时关键时序完成中断
SDIO_INT_FLAG_C LKSTOP	电压切换期间SDIO_CLK停止中断
SDIO_INT_FLAG_I DMAERR	IDMA传输错误中断
SDIO_INT_FLAG_I DMAEND	IDMA传输结束中断

输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* clear the interrupt pending flags of SDIO0 */
```

```
sdio_interrupt_flag_clear(SDIO0, SDIO_INT_FLAG_DTEND);
```

函数 sdio_voltage_switch_enable

函数sdio_voltage_switch_enable描述见下表：

表 3-1487. 函数 sdio_voltage_switch_enable

函数名称	sdio_voltage_switch_enable
函数原形	void sdio_voltage_switch_enable(uint32_t sdio_periph);
功能描述	使能电压切换
先决条件	-
被调用函数	-
输入参数{in}	
sdio_periph	SDIO外设
SDIO0	选择SDIO0
SDIO1	选择SDIO1
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* enable SDIO0 voltage switch */
```

```
sdio_voltage_switch_enable(SDIO0);
```

函数 sdio_voltage_switch_disable

函数sdio_voltage_switch_disable描述见下表：

表 3-1488. 函数 sdio_voltage_switch_disable

函数名称	sdio_voltage_switch_disable
函数原形	void sdio_voltage_switch_disable(uint32_t sdio_periph);
功能描述	禁能电压切换
先决条件	-
被调用函数	-
输入参数{in}	

sdio_periph	SDIO外设
<i>SDIO0</i>	选择SDIO0
<i>SDIO1</i>	选择SDIO1
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* disable SDIO0 voltage switch */
```

```
sdio_voltage_switch_disable(SDIO0);
```

函数 **sdio_voltage_switch_sequence_enable**

函数sdio_voltage_switch_sequence_enable描述见下表：

表 3-1489. 函数 sdio_voltage_switch_sequence_enable

函数名称	sdio_voltage_switch_sequence_enable
函数原形	void sdio_voltage_switch_sequence_enable(uint32_t sdio_periph);
功能描述	使能电压切换序列
先决条件	-
被调用函数	-
输入参数{in}	
sdio_periph	SDIO外设
<i>SDIO0</i>	选择SDIO0
<i>SDIO1</i>	选择SDIO1
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* enable SDIO0 voltage switch sequence */
```

```
sdio_voltage_switch_sequence_enable(SDIO0);
```

函数 **sdio_voltage_switch_sequence_disable**

函数sdio_voltage_switch_sequence_disable描述见下表：

表 3-1490. 函数 sdio_voltage_switch_sequence_disable

函数名称	sdio_voltage_switch_sequence_disable
函数原形	void sdio_voltage_switch_sequence_disable(uint32_t sdio_periph);
功能描述	禁能电压切换序列

先决条件	-
被调用函数	-
输入参数{in}	
sdio_periph	SDIO外设
<i>SDIO0</i>	选择SDIO0
<i>SDIO1</i>	选择SDIO1
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* disable SDIO0 voltage switch sequence */
```

```
sdio_voltage_switch_sequence_disable(SDIO0);
```

函数 **sdio_boot_mode_set**

函数sdio_boot_mode_set描述见下表：

表 3-1491. 函数 sdio_boot_mode_set

函数名称	sdio_boot_mode_set
函数原形	void sdio_boot_mode_set(uint32_t sdio_periph, uint32_t boot_mode);
功能描述	设置引导模式
先决条件	-
被调用函数	-
输入参数{in}	
sdio_periph	SDIO外设
<i>SDIO0</i>	选择SDIO0
<i>SDIO1</i>	选择SDIO1
输入参数{in}	
boot_mode	SDIO引导模式
<i>SDIO_BOOT_NOR MAL</i>	正常的引导模式
<i>SDIO_BOOT_ALTE RNATIVE</i>	备用的引导模式
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* set SDIO0 boot mode*/
```

```
sdio_boot_mode_set (SDIO0, SDIO_BOOT_NORMAL);
```

函数 **sdio_boot_ack_enable**

函数sdio_boot_ack_enable描述见下表：

表 3-1492. 函数 sdio_boot_ack_enable

函数名称	sdio_boot_ack_enable
函数原形	void sdio_boot_ack_enable(uint32_t sdio_periph);
功能描述	使能DSM引导应答
先决条件	-
被调用函数	-
输入参数{in}	
sdio_periph	SDIO外设
SDIO0	选择SDIO0
SDIO1	选择SDIO1
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* enable SDIO0 boot acknowledgment */
```

```
sdio_boot_ack_enable(SDIO0);
```

函数 **sdio_boot_ack_disable**

函数sdio_boot_ack_disable描述见下表：

表 3-1493. 函数 sdio_boot_ack_disable

函数名称	sdio_boot_ack_disable
函数原形	void sdio_boot_ack_disable(uint32_t sdio_periph);
功能描述	禁能DSM引导应答
先决条件	-
被调用函数	-
输入参数{in}	
sdio_periph	SDIO外设
SDIO0	选择SDIO0
SDIO1	选择SDIO1
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* disable SDIO0 boot acknowledgment */
```

```
sdio_boot_ack_disable(SDIO0);
```

函数 **sdio_boot_acktimeout_set**

函数sdio_boot_acktimeout_set描述见下表：

表 3-1494. 函数 sdio_boot_acktimeout_set

函数名称	sdio_boot_acktimeout_set
函数原形	void sdio_boot_acktimeout_set(uint32_t sdio_periph, uint32_t timeout);
功能描述	设置引导应答超时周期
先决条件	-
被调用函数	-
输入参数{in}	
sdio_periph	SDIO外设
SDIO0	选择SDIO0
SDIO1	选择SDIO1
输入参数{in}	
timeout	引导超时时间
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* set SDIO0 boot ACK timeout period */
```

```
void sdio_boot_acktimeout_set(SDIO0, uint32_t (0xFFFF));
```

函数 **sdio_boot_enable**

函数sdio_boot_enable描述见下表：

表 3-1495. 函数 sdio_boot_enable

函数名称	sdio_boot_enable
函数原形	void sdio_boot_enable(uint32_t sdio_periph);
功能描述	使能引导操作
先决条件	-
被调用函数	-
输入参数{in}	
sdio_periph	SDIO外设
SDIO0	选择SDIO0
SDIO1	选择SDIO1

输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* enable SDIO0 boot operation */
sdio_boot_enable(SDIO0);
```

函数 sdio_boot_disable

函数sdio_boot_disable描述见下表：

表 3-1496. 函数 sdio_boot_disable

函数名称	sdio_boot_disable
函数原形	void sdio_boot_disable(uint32_t sdio_periph);
功能描述	禁能引导操作
先决条件	-
被调用函数	-
输入参数{in}	
sdio_periph	SDIO外设
SDIO0	选择SDIO0
SDIO1	选择SDIO1
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* disable SDIO0 boot operation */
sdio_boot_disable(SDIO0);
```

3.41. 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[3.41.1](#)描述了SPI/I2S的寄存器列表，章节[3.41.2](#)对SPI/I2S库函数进行说明。

3.41.1. 外设寄存器说明

SPI/I2S寄存器列表如下表所示：

表 3-1497. 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
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

3.41.2. 外设库函数说明

SPI/I2S库函数列表如下表所示：

表 3-1498. 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

库函数名称	库函数描述
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
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

库函数名称	库函数描述
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
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 3-1499. 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 3-1500. 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 3-1501. 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3-1499. 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 3-1502. 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>	x=0,1...5
输入参数{in}	
spi_struct	初始化结构体，结构体成员参考 表3-1499. 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 3-1503. 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>	x=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 3-1504. 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 3-1505. 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
<i>I2S_MODE_SLAVE_TX</i>	I2S从机发送模式
<i>I2S_MODE_SLAVE_RX</i>	I2S从机接收模式
<i>I2S_MODE_MASTERTX</i>	I2S主机发送模式
<i>I2S_MODE_MASTERRX</i>	I2S主机接收模式
输入参数{in}	
i2s_standard	I2S标准选择
<i>I2S_STD_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_PCMSHORT</i>	I2S PCM短帧标准
<i>I2S_STD_PCMLONG</i>	I2S PCM长帧标准
输入参数{in}	
i2s_ckpl	I2S空闲状态时钟极性
<i>I2S_CKPL_LOW</i>	I2S_CK空闲状态为低电平
<i>I2S_CKPL_HIGH</i>	I2S_CK空闲状态为高电平
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* initialize 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 3-1506. 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
SPIx	x=0,1,2,5
输入参数{in}	
i2s_audiosample	I2S音频采样频率
I2S_AUDIOSAMPL E_8K	音频采样频率为8KHz
I2S_AUDIOSAMPL E_11K	音频采样频率为11KHz
I2S_AUDIOSAMPL E_16K	音频采样频率为16KHz
I2S_AUDIOSAMPL E_22K	音频采样频率为22KHz
I2S_AUDIOSAMPL E_32K	音频采样频率为32KHz
I2S_AUDIOSAMPL E_44K	音频采样频率为44KHz
I2S_AUDIOSAMPL E_48K	音频采样频率为48KHz
I2S_AUDIOSAMPL E_96K	音频采样频率为96KHz
I2S_AUDIOSAMPL E_192K	音频采样频率为192KHz
输入参数{in}	
i2s_frameformat	I2S数据长度和通道长度
I2S_FRAMEFORMA T_DT16B_CH16B	I2S数据长度为16位，通道长度为16位
I2S_FRAMEFORMA T_DT16B_CH32B	I2S数据长度为16位，通道长度为32位
I2S_FRAMEFORMA T_DT24B_CH32B	I2S数据长度为24位，通道长度为32位
I2S_FRAMEFORMA T_DT32B_CH32B	I2S数据长度为32位，通道长度为32位
输入参数{in}	
i2s_mckout	I2S_MCK输出
I2S_MCKOUT_ENA BLE	I2S_MCK输出使能
I2S_MCKOUT_DIS ABLE	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 3-1507. 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 3-1508. 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 3-1509. 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 3-1510. 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
<i>SPIx</i>	x=0,1...5
输入参数{in}	
delay_cycle	时钟延时
<i>SPI_NSS_IDLENES</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 3-1511. 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 3-1512. 函数 spi_master_receive_clock_delay_set

函数名称	spi_master_receive_clock_delay_set
函数原形	void spi_master_receive_clock_delay_set(uint32_t spi_periph, uint32_t delay_unit);
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
SPIx	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 3-1513. 函数 spi_slave_receive_clock_delay_set

函数名称	spi_slave_receive_clock_delay_set
函数原形	spi_slave_receive_clock_delay_set (uint32_t spi_periph, uint32_t delay_unit);
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
SPIx	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 3-1514. 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 3-1515. 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 3-1516. 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 3-1517. 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
<i>SPIx</i>	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 3-1518. 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 3-1519. 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 3-1520. 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 3-1521. 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 3-1522. 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 3-1523. 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 3-1524. 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
<i>SPIx</i>	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 3-1525. 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 3-1526. 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
SPIx	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 3-1527. 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
SPIx	x=0,1...5
输入参数{in}	
transfer_direction	SPI双向传输输出使能
SPI_BIDIRECTIONAL_TRANSMIT	SPI工作在只发送模式
SPI_BIDIRECTIONAL_RECEIVE	SPI工作在只接收模式
输出参数{out}	

-	-
返回值	
-	-

例如：

```
/* SPI0 works in transmit-only mode */
```

```
spi_bidirectional_transfer_config(SPI0, SPI_BIDIRECTIONAL_TRANSMIT);
```

函数 spi_master_transfer_start

函数spi_master_transfer_start描述见下表：

表 3-1528. 函数 spi_master_transfer_start

函数名称	spi_master_transfer_start
函数原形	void spi_master_transfer_start(uint32_t spi_periph, uint32_t transfer_start);
功能描述	主机启动传输
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}	
transfer_start	传输启动
SPI_TRANS_START	主机开始传输，或者被自动挂起功能临时挂起
SPI_TRANS_IDLE	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 3-1529. 函数 spi_current_data_num_config

函数名称	spi_current_data_num_config
函数原形	void spi_current_data_num_config(uint32_t spi_periph, uint32_t current_num);
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}	
spi_periph	外设SPIx
<i>SPIx</i>	x=0,1...5
输入参数{in}	
current_num	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 3-1530. 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
<i>SPIx</i>	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 3-1531. 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
SPIx	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 3-1532. 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
SPIx	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 3-1533. 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 3-1534. 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 3-1535. 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 3-1536. 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 3-1537. 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 3-1538. 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

<i>SPIx</i>	<i>x</i> =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_tcr_init_pattern**

函数spi_tcr_init_pattern描述见下表：

表 3-1539. 函数 spi_tcr_init_pattern

函数名称	spi_tcr_init_pattern
函数原形	void spi_tcr_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
<i>SPIx</i>	<i>x</i> =0,1...5
输入参数{in}	
init_pattern	初始模式
<i>SPI_TCRC_INIT_1</i>	全1模式
<i>SPI_TCRC_INIT_0</i>	全0模式
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* config SPI0 TCRC all 1 initial pattern */
```

```
spi_tcr_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 3-1540. 函数 spi_tcr_init_pattern

函数名称	spi_tcr_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
SPIx	x=0,1...5
输入参数{in}	
init_pattern	初始模式
SPI_RCRC_INIT_1	全1模式
SPI_RCRC_INIT_0	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 3-1541. 函数 spi_ti_mode_enable

函数名称	spi_ti_mode_enable
函数原形	void spi_ti_mode_enable(uint32_t spi_periph);
功能描述	使能SPI TI模式
先决条件	-
被调用函数	-
输入参数{in}	
spi_periph	外设SPIx
SPIx	x=0,1...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 3-1542. 函数 spi_ti_mode_disable

函数名称	spi_ti_mode_disable
函数原形	void spi_ti_mode_disable(uint32_t spi_periph);
功能描述	禁能SPI TI模式
先决条件	-
被调用函数	-
输入参数{in}	
spi_periph	外设SPIx
SPIx	x=0,1...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 3-1543. 函数 spi_quad_enable

函数名称	spi_quad_enable
函数原形	void spi_quad_enable(uint32_t spi_periph);
功能描述	使能四线SPI模式
先决条件	-
被调用函数	-
输入参数{in}	
spi_periph	外设SPIx
SPIx	x=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 3-1544. 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 3-1545. 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 3-1546. 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 3-1547. 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 3-1548. 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 3-1549. 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_REG1 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 3-1550. 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 3-1551. 函数 `spi_underrun_data_config`

函数名称	<code>spi_underrun_data_config</code>
函数原形	<code>void spi_underrun_data_config(uint32_t spi_periph, uint32_t udata);</code>
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}	
<code>spi_periph</code>	外设SPIx
<code>SPIx</code>	x=0,1...5
输入参数{in}	
<code>udata</code>	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 3-1552. 函数 `spi_suspend_mode_config`

函数名称	<code>spi_suspend_mode_config</code>
函数原形	<code>void spi_suspend_mode_config(uint32_t spi_periph, uint32_t sus_mode);</code>
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}	
<code>spi_periph</code>	外设SPIx
<code>SPIx</code>	x=0,1...5
输入参数{in}	
<code>sus_mode</code>	挂起模式
<code>SPI_AUTO_SUSPEND</code>	当上溢出现之前，当RxFIFO已满时，SPI数据流被挂起
<code>SPI_CONTINUOUS</code>	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 3-1553. 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 3-1554. 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 3-1555. 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 3-1556. 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

<i>SPI_GPIO_FREE</i>	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 3-1557. 函数 `spi_i2s_interrupt_enable`

函数名称	<code>spi_i2s_interrupt_enable</code>
函数原形	<code>void spi_i2s_interrupt_enable(uint32_t spi_periph, uint8_t interrupt);</code>
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
<i>SPIx</i>	x=0,1...5
输入参数{in}	
interrupt	SPI/I2S中断
<i>SPI_I2S_INT_RP</i>	RP中断
<i>SPI_I2S_INT_TP</i>	TP中断
<i>SPI_I2S_INT_DP</i>	DP中断
<i>SPI_I2S_INT_ESTC</i>	传输结束、挂起、TxFIFO清空中断
<i>SPI_I2S_INT_TXF</i>	传输已填充中断
<i>SPI_I2S_INT_TXUR</i> <i>E</i>	下溢错误中断
<i>SPI_I2S_INT_RXO</i> <i>RE</i>	上溢错误中断
<i>SPI_I2S_INT_CRC</i> <i>ER</i>	CRC错误中断
<i>SPI_INT_FE</i>	TI帧错误中断
<i>SPI_I2S_INT_CON</i> <i>FE</i>	SPI配置错误中断
<i>SPI_I2S_INT_TXSE</i> <i>RF</i>	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 3-1558. 函数 spi_i2s_interrupt_disable

函数名称	spi_i2s_interrupt_disable
函数原形	void spi_i2s_interrupt_disable(uint32_t spi_periph, uint8_t interrupt);
功能描述	禁能外设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 3-1559. 函数 spi_i2s_interrupt_flag_get

函数名称	spi_i2s_interrupt_flag_get
函数原形	FlagStatus spi_i2s_interrupt_flag_get(uint32_t spi_periph, uint8_t interrupt);
功能描述	获取外设SPI/I2S中断状态
先决条件	-
被调用函数	-
输入参数{in}	
spi_periph	外设SPIx
SPIx	x=0,1...5
输入参数{in}	
interrupt	SPI/I2S中断状态
SPI_I2S_INT_FLAG _RP	RP中断标志
SPI_I2S_INT_FLAG _TP	TP中断标志
SPI_I2S_INT_FLAG _DP	DP中断标志
SPI_I2S_INT_FLAG _ET	发送或接收结束中断标志
SPI_I2S_INT_FLAG _TXF	传输已填充中断标志
SPI_I2S_INT_FLAG _TXURERR	下溢错误中断标志
SPI_I2S_INT_FLAG _RXORERR	上溢错误中断标志
SPI_I2S_INT_FLAG _CRCERR	CRC错误中断标志
SPI_I2S_INT_FLAG _FERR	TI帧错误中断标志
SPI_I2S_INT_FLAG _CONFERR	SPI配置错误中断标志
SPI_I2S_INT_FLAG _TXSERF	TXSER重载中断标志
SPI_I2S_INT_FLAG	挂起中断标志

<code>_SPD</code>	
<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 3-1560. 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_TXURER	SPI传输下溢错误标志
SPI_FLAG_RXORER	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

<code>SPI_FLAG_CONFER</code>	SPI配置错误标志
<code>SPI_FLAG_TXSERF</code>	SPI TXSER已重载标志
<code>SPI_FLAG_SPD</code>	SPI挂起标志
<code>SPI_FLAG_TC</code>	SPI TxFIFO清空标志
<code>SPI_FLAG_RWNE</code>	SPI RxFIFO中字长非空标志
<code>I2S_FLAG_RP</code>	I2S RP标志
<code>I2S_FLAG_TP</code>	I2S TP标志
<code>I2S_FLAG_DP</code>	I2S DP标志
<code>I2S_FLAG_ET</code>	I2S发送或接收结束标志
<code>I2S_FLAG_TXF</code>	I2S TxFIFO已填充标志
<code>I2S_FLAG_TXURERR</code>	I2S传输下溢错误标志
<code>I2S_FLAG_RXORERR</code>	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 3-1561. 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
SPI_STATC_ETC	清除传输/接收结束标志
SPI_STATC_TXFC	清除TxFIFO传输填充标志
SPI_STATC_TXURERRC	清除传输下溢错误标志
SPI_STATC_RXORERRC	清除接收上溢错误标志
SPI_STATC_CRCEERRC	清除CRC错误标志
SPI_STATC_FERRC	清除SPI TI格式错误标志
SPI_STATC_CONFERRC	清除配置错误标志
SPI_STATC_TXSERF RFC	清除TXSERF标志
SPI_STATC_SPDC	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 3-1562. 函数 spi_i2s_rxfifo_plevel_get

函数名称	spi_i2s_rxfifo_plevel_get
函数原形	uint32_t spi_i2s_rxfifo_plevel_get(uint32_t spi_periph);
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}	
spi_periph	外设SPIx
SPIx	x=0,1...5
输出参数{out}	
-	-
返回值	
uint32_t	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 3-1563. 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 3-1564. 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
<i>SPI_FIFO_TH_xDATA</i> <i>TA</i>	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)
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 3-1565. 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
<i>SPIx</i>	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 3-1566. 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
<i>SPIx</i>	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 3-1567. 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
<i>SPIx</i>	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 3-1568. 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	禁用外设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);
```

3.42. SYSCFG

章节 [3.42.1](#) 描述了SYSCFG的寄存器列表，章节 [3.42.2](#) 对SYSCFG库函数进行说明。

3.42.1. 外设寄存器说明

SYSCFG寄存器列表如下表所示：

表 3-1569. 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	TIMER输入选择寄存器4

寄存器名称	寄存器描述
SEL4	
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 C	CPU ICACHE错误状态寄存器
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

3.42.2. 外设库函数说明

SYSCFG库函数列表如下表所示：

表 3-1570. 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

库函数名称	库函数描述
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
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 3-1571. 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_C11_INPUT_TIMER7_CH1	选择TIMER7 CH1作为TIMER7 C11输入
TIMER7_C12_INPUT_TIMER7_CH2	选择TIMER7 CH2作为TIMER7 C12输入
TIMER7_C13_INPUT_TIMER7_CH3	选择TIMER7 CH3作为TIMER7 C13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_C11_INPUT_TIMER0_CH1	选择TIMER0 CH1作为TIMER0 C11输入
TIMER0_C12_INPUT_TIMER0_CH2	选择TIMER0 CH2作为TIMER0 C12输入
TIMER0_C13_INPUT_TIMER0_CH3	选择TIMER0 CH3作为TIMER0 C13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

枚举名称	枚举描述
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
TIMER1_CI0_INPUT_TIMER1_CH0	选择TIMER1 CH0作为TIMER1 CI0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_CI0_INPUT_TIMER4_CH0	选择TIMER4 CH0作为TIMER4 CI0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_CI0_INPUT_TIMER3_CH0	选择TIMER3 CH0作为TIMER3 CI0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_CI0_INPUT_TIMER23_CH0	选择TIMER23 CH0作为TIMER23 CI0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

枚举名称	枚举描述
TIMER23_CI3_INPUT_TIMER23_CH3	选择TIMER23 CH3作为TIMER23 CI3输入
TIMER22_CI0_INPUT_TIMER22_CH0	选择TIMER22 CH0作为TIMER22 CI0输入
TIMER22_CI1_INPUT_TIMER22_CH1	选择TIMER22 CH1作为TIMER22 CI1输入
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
TIMER31_CI0_INPUT_TIMER31_CH0	选择TIMER31 CH0作为TIMER31 CI0输入
TIMER31_CI0_INPUT_CMP0_OUT	选择CMP0输出作为TIMER31 CI0输入
TIMER31_CI0_INPUT_CMP1_OUT	选择CMP1输出作为TIMER31 CI0输入
TIMER31_CI0_INPUT_CMP0_OUT_CMP1_OUT	选择CMP0或CMP1输出作为TIMER31 CI0输入
TIMER31_CI1_INPUT_TIMER31_CH1	选择TIMER31 CH1作为TIMER31 CI1输入
TIMER31_CI2_INPUT_TIMER31_CH2	选择TIMER31 CH2作为TIMER31 CI2输入
TIMER31_CI3_INPUT_TIMER31_CH3	选择TIMER31 CH3作为TIMER31 CI3输入
TIMER30_CI0_INPUT_TIMER30_CH0	选择TIMER30 CH0作为TIMER30 CI0输入
TIMER30_CI0_INPUT_CMP0_OUT	选择CMP0输出作为TIMER30 CI0输入
TIMER30_CI0_INPUT_CMP1_OUT	选择CMP1输出作为TIMER30 CI0输入
TIMER30_CI0_INPUT_CMP0_OUT_CMP1_OUT	选择CMP0或CMP1输出作为TIMER30 CI0输入
TIMER30_CI1_INPUT_TIMER30_CH1	选择TIMER30 CH1作为TIMER30 CI1输入
TIMER30_CI2_INPUT_TIMER30_CH2	选择TIMER30 CH2作为TIMER30 CI2输入

枚举名称	枚举描述
TIMER30_CI3_INPUT_TIMER30_CH3	选择TIMER30 CH3作为TIMER30 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
TIMER40_CI1_INPUT_TIMER40_CH1	选择TIMER40 CH1作为TIMER40 CI0输入
TIMER40_CI1_INPUT_TIMER2_CH1	选择TIMER2 CH1作为TIMER40 CI0输入
TIMER40_CI1_INPUT_TIMER3_CH1	选择TIMER3 CH1作为TIMER40 CI0输入
TIMER40_CI1_INPUT_TIMER4_CH1	选择TIMER4 CH1作为TIMER40 CI0输入

枚举名称	枚举描述
TIMER41_CIO_INPUT_TIMER41_CH0	选择TIMER41 CH0作为TIMER41 CIO输入
TIMER41_CIO_INPUT_TIMER3_CH0	选择TIMER3 CH0作为TIMER41 CIO输入
TIMER41_CIO_INPUT_TIMER4_CH0	选择TIMER4 CH0作为TIMER41 CIO输入
TIMER41_CIO_INPUT_TIMER22_CH0	选择TIMER22 CH0作为TIMER41 CIO输入
TIMER41_CIO_INPUT_LXTAL	选择LXTAL作为TIMER41 CIO输入
TIMER41_CIO_INPUT_LPIRC4M	选择LPIRC4M作为TIMER41 CIO输入
TIMER41_CIO_INPUT_CKOUT1	选择CKOUT1作为TIMER41 CIO输入
TIMER41_C11_INPUT_TIMER41_CH1	选择TIMER41 CH1作为TIMER41 C11输入
TIMER41_C11_INPUT_TIMER3_CH1	选择TIMER3 CH1作为TIMER41 C11输入
TIMER41_C11_INPUT_TIMER4_CH1	选择TIMER4 CH1作为TIMER41 C11输入
TIMER41_C11_INPUT_TIMER22_CH1	选择TIMER22 CH1作为TIMER41 C11输入
TIMER42_CIO_INPUT_TIMER42_CH0	选择TIMER42 CH0作为TIMER42 CIO输入
TIMER42_CIO_INPUT_TIMER4_CH0	选择TIMER4 CH0作为TIMER42 CIO输入
TIMER42_CIO_INPUT_TIMER22_CH0	选择TIMER22 CH0作为TIMER42 CIO输入
TIMER42_CIO_INPUT_TIMER23_CH0	选择TIMER23 CH0作为TIMER42 CIO输入
TIMER42_CIO_INPUT_LXTAL	选择LXTAL作为TIMER42 CIO输入
TIMER42_CIO_INPUT_LPIRC4M	选择LPIRC4M作为TIMER42 CIO输入
TIMER42_CIO_INPUT_CKOUT1	选择CKOUT1作为TIMER42 CIO输入
TIMER42_C11_INPUT_TIMER42_CH1	选择TIMER42 CH1作为TIMER42 C11输入
TIMER42_C11_INPUT_TIMER4_CH1	选择TIMER4 CH1作为TIMER42 C11输入
TIMER42_C11_INPUT_TIMER22_CH1	选择TIMER22 CH1作为TIMER42 C11输入
TIMER42_C11_INPUT_TIMER23_CH1	选择TIMER23 CH1作为TIMER42 C11输入
TIMER15_CIO_INPUT_TIMER15_CH0	选择TIMER15 CH0作为TIMER15 CIO输入

枚举名称	枚举描述
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_TIMER30_CH0	选择TIMER30 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_CIO_INPUT_TIMER43_CH1	选择TIMER43 CH1作为TIMER43 CIO输入
TIMER43_CIO_INPUT_TIMER22_CH1	选择TIMER22 CH1作为TIMER43 CIO输入
TIMER43_CIO_INPUT_TIMER23_CH1	选择TIMER23 CH1作为TIMER43 CIO输入
TIMER43_CIO_INPUT_TIMER30_CH1	选择TIMER30 CH1作为TIMER43 CIO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_TIMER30_CH0	选择TIMER30 CH0作为TIMER44 CIO输入
TIMER44_CIO_INPUT_TIMER31_CH0	选择TIMER31 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
TIMER44_CI1_INPUT_TIMER30_CH1	选择TIMER30 CH1作为TIMER44 CI1输入
TIMER44_CI1_INPUT_TIMER31_CH1	选择TIMER31 CH1作为TIMER44 CI1输入

函数 syscfg_deinit

函数syscfg_deinit描述见下表：

表 3-1572. 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 3-1573. 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_FM	I2C0 Fm+模式

<i>P</i>	
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	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 3-1574. 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	I2C3 Fm+模式

<i>P</i>	
<code>SYSCFG_I2C_FMP_PB6</code>	PB6引脚Fm+模式
<code>SYSCFG_I2C_FMP_PB7</code>	PB7引脚Fm+模式
<code>SYSCFG_I2C_FMP_PB8</code>	PB8引脚Fm+模式
<code>SYSCFG_I2C_FMP_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 3-1575. 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}	
gpio_answ	GPIO模拟开关
<code>SYSCFG_PA0_AN ALOG_SWITCH</code>	PA0模拟开关
<code>SYSCFG_PA1_AN ALOG_SWITCH</code>	PA1模拟开关
<code>SYSCFG_PC2_AN ALOG_SWITCH</code>	PC2模拟开关
<code>SYSCFG_PC3_AN 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 3-1576. 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 3-1577. 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
SYSCFG_ENET_PHY_MII	选择MII模式
SYSCFG_ENET_PHY_RMII	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 3-1578. 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
EXTI_SOURCE_GPIOx	x = A,B,C,D,E,F,G,H,J,K
输入参数{in}	
exti_pin	EXTI引脚
EXTI_SOURCE_PINx	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 3-1579. 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 3-1580. 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
<i>SYSCFG_LVD_LO CKUP</i>	LVD锁定
<i>SYSCFG_CPU_LO CKUP</i>	CPU锁定
<i>SYSCFG_BKPRAM _LOCKUP</i>	Region 2备份SRAM ECC双差错锁定
<i>SYSCFG_SRAM1_ LOCKUP</i>	Region 1 SRAM1 ECC双差错锁定
<i>SYSCFG_SRAM0_ LOCKUP</i>	Region 1 SRAM0 ECC双差错锁定
<i>SYSCFG_DTCM_L OCKUP</i>	Region 0 DTCM ECC双差错锁定
<i>SYSCFG_ITCM_LO CKUP</i>	Region 0 ITCM-RAM ECC双差错锁定
<i>SYSCFG_AXIRAM_ LOCKUP</i>	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 3-1581. 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3-1571. 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 3-1582. 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 3-1583. 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 3-1584. 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 3-1585. 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 3-1586. 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

ZE_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 3-1587. 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 3-1588. 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 3-1589. 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 3-1590. 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 3-1591. 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_ INVALID_OPERATION	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 3-1592. 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 3-1593. 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 3-1594. 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 OR_BANK	选择错误库信息
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 3-1595. 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

<code>BOR_THRESHOLD</code> <code>_VAL3</code>	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();
```

3.43. 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, 30, 31），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[3.43.1](#)描述了TIMER的寄存器列表，章节[3.43.2](#)对TIMER库函数进行说明。

3.43.1. 外设寄存器说明

TIMER寄存器列表如下表所示:

表 3-1596. 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

寄存器名称	寄存器描述
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
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_WDGP	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

3.43.2. 外设库函数说明

TIMER库函数列表如下表所示：

表 3-1597. 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

库函数名称	库函数描述
timer_counter_alignment	设置外设TIMER的对齐模式
timer_counter_up_direction	设置外设TIMER向上计数
timer_counter_down_direction	设置外设TIMER向下计数
timer_prescaler_config	配置外设TIMER预分频器
timer_repetition_value_config	配置外设TIMER的重复计数器
timer_runtime_repetition_value_read	配置外设TIMER实时重复计数器值
timer_autoreload_value_config	配置外设TIMER的自动重载寄存器
timer_autoreload_value_read	读取TIMER自动重载寄存器计数器值
timer_counter_value_config	配置外设TIMER的计数器值
timer_counter_read	读取外设TIMER的计数器值
timer_prescaler_read	读取外设TIMER的预分频器值
timer_single_pulse_mode_config	配置外设TIMER的单脉冲模式
timer_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

库函数名称	库函数描述
timer_channel_output_polarity_config	通道输出极性配置
timer_channel_complementary_output_polarity_config	互补通道输出极性配置
timer_channel_output_state_config	配置通道状态
timer_channel_complementary_output_state_config	配置互补通道输出状态
timer_channel_input_struct_para_init	将TIMER通道输入参数结构体中所有参数初始化为默认值
timer_input_capture_config	配置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

库函数名称	库函数描述
timer_external_clock_mode1_disable	TIMER外部时钟模式1禁能
timer_write_chxval_register_config	配置TIMER写CHxVAL选择位
timer_output_value_selection_config	配置TIMER输出值选择位
timer_commutation_control_shadow_register_config	配置换相控制影子寄存器更新选择
timer_output_match_pulse_select	配置TIMER输出匹配脉冲选择
timer_channel_composite_pwm_mode_config	配置TIMER的复合PWM模式
timer_channel_composite_pwm_mode_output_pulse_value_config	配置TIMER的复合PWM模式输出脉冲值
timer_channel_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 3-1598. 结构体 timer_parameter_struct

成员名称	功能描述
prescaler	预分频值 (0~65535)
alignedmode	对齐模式 (TIMER_COUNTER_EDGE, TIMER_COUNTER_CENTER_DOWN, TIMER_COUNTER_CENTER_UP, TIMER_COUNTER_CENTER_BOTH)
counterdirection	计数方向 (TIMER_COUNTER_UP, TIMER_COUNTER_DOWN)
period	周期 (0~0xFFFF(TIMERx(x=0,7,14~16,40~44)), 0~0xFFFFFFFF(TIMERx(x=1~6,22,23,30,31)), 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 3-1599. 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)

成员名称	功能描述
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 3-1600. 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 3-1601. 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 3-1602. 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)

成员名称	功能描述
icfilter	通道输入捕获滤波（0~15）

结构体 timer_free_complementary_parameter_struct

表 3-1603. 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）
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 3-1604. 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=0~7,14~16,22,23,30,31,40~44,50,51)	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 3-1605. 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 3-1606. 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, 30, 31, 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 3-1607. 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=0~7,14~16,22,23,30,31,40~44,50,51)	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 3-1608. 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=0~7,14~16,22,23,30,31,40~44,50,51)	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 3-1609. 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=0~7,14~16,22,23,30,31,40~44,50,51)	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 3-1610. 函数 timer_auto_reload_shadow_disable

函数名称	timer_auto_reload_shadow_disable
函数原型	void timer_auto_reload_shadow_disable (uint32_t timer_periph);
功能描述	TIMER自动重载影子禁能
先决条件	-
被调用函数	-
输入参数{in}	
timer_periph	TIMER外设
TIMERx(x=0~7, 14~16, 22, 23, 30, 31, 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 3-1611. 函数 timer_update_event_enable

函数名称	timer_update_event_enable
函数原型	void timer_update_event_enable(uint32_t timer_periph);
功能描述	TIMER更新使能
先决条件	-
被调用函数	-
输入参数{in}	
timer_periph	TIMER外设
TIMERx(x=0~7, 14~16, 22, 23, 30, 31, 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 3-1612. 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=0~7,14~16,22,23,30,31,40~44,50,51)	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 3-1613. 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=0~4,7,22,23,30,31)	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

<code>TIMER_COUNTER_CENTER_UP</code>	中央对齐向上计数模式
<code>TIMER_COUNTER_CENTER_BOTH</code>	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 3-1614. 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
<code>TIMERx(x=0~4,7,22,23,30,31)</code>	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 3-1615. 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=0~4,7,22,23,30,31)	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 3-1616. 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=0~7,14~16,22,23,30,31,40~44,50,51)	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 3-1617. 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 3-1618. 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
TIMERx(x=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 3-1619. 函数 timer_autoreload_value_config

函数名称	timer_autoreload_value_config
函数原型	void timer_autoreload_value_config(uint32_t timer_periph, uint64_t autoreload);
功能描述	配置外设TIMER的自动重载寄存器
先决条件	-
被调用函数	-
输入参数{in}	
timer_periph	TIMER外设
TIMERx(x=0~7,14~16,22,23,30,31,40~44,50,51)	TIMER外设选择
输入参数{in}	
autoreload	计数器自动重载值 0~0xFFFF, TIMERx(x=0,2,3,7,14~16,30,31,40~44) 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 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 3-1620. 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=0~7,14~16,22,23,30,31,40~44,50,51)	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=0,2,3,7,14~16,30,31,40~44) 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 3-1621. 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=0~7,14~16,22,23,30,31,40~44,50,51)	TIMER外设选择
输入参数{in}	
counter	计数器值 0~0xFFFF, TIMERx(x=0,2,3,7,14~16,30,31,40~44) 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 3-1622. 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=0~7,14~16,22,23,30,31,40~44,50,51)	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=0,2,3,7,14~16,30,31,40~44) 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 3-1623. 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, 30, 31, 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 3-1624. 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

<i>TIMERx</i> (<i>x</i> =0~7,14~16,22,23,30,31,40~44,50,51)	TIMER外设选择
输入参数{in}	
spmode	脉冲模式
<i>TIMER_SP_MODE_SINGLE</i>	单脉冲模式计数
<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 3-1625. 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
<i>TIMERx</i> (<i>x</i> =0~4,7,14,22,23,30,31,40~44)	TIMER外设选择
输入参数{in}	
channel	TIMER通道
<i>TIMER_CH_0</i>	通道0, <i>TIMERx</i> (<i>x</i> =0~4,7,14,22,23,30,31,40~44)
<i>TIMER_CH_1</i>	通道1, <i>TIMERx</i> (<i>x</i> =0~4,7,14,22,23,30,31,40~44)
<i>TIMER_CH_2</i>	通道2, <i>TIMERx</i> (<i>x</i> =0~4,7,22,23,30,31)
<i>TIMER_CH_3</i>	通道3, <i>TIMERx</i> (<i>x</i> =0~4,7,22,23,30,31)
<i>TIMER_MCH_0</i>	多模式通道0, <i>TIMERx</i> (<i>x</i> =0,7,14,40~44)
<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, <i>TIMERx</i> (<i>x</i> =0,7)

输入参数{in}	
dspmode	可延时单脉冲模式
<i>TIMER_OC_MODE_DSPM0</i>	可延时单脉冲模式0
<i>TIMER_OC_MODE_DSPM1</i>	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 3-1626. 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(x=0~7,14~16,22,23,30,31,40~44,50,51)</i>	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： <ul style="list-style-type: none"> – UPG位被置1 – 计数器溢出/下溢 – 从模式控制器产生的更新

TIMER_UPDATE_SRC_REGULAR	只有计数器溢出/ 下溢才产生更新中断或DMA请求
输出参数{out}	
-	-
返回值	
-	-

例如:

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

```
timer_update_source_config(TIMER0, TIMER_UPDATE_SRC_REGULAR);
```

函数 timer_dma_enable

函数timer_dma_enable描述见下表:

表 3-1627. 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=0~7,14~16,22,23,30,31,40~44,50,51)	参考具体参数
输入参数{in}	
dma	DMA源
TIMER_DMA_UPD	更新DMA请求, TIMERx(x=0~7,14~16,22,23,30,31,40~44,50,51)
TIMER_DMA_CH0D	通道0比较/捕获DMA请求, TIMERx (x=0~4,7,14~16,22,23,30,31,40~44)
TIMER_DMA_CH1D	通道1比较/捕获DMA请求, TIMERx (x=0~4,7,14,22,23,30,31,40~44)
TIMER_DMA_CH2D	通道2比较/捕获DMA请求, TIMERx (x=0~4,7,22,23,30,31)
TIMER_DMA_CH3D	通道3比较/捕获DMA请求, TIMERx (x=0~4,7,22,23,30,31)
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=0~4,7,14,22,23,30,31,40~44)
TIMER_DMA_MCH0D	多模式通道0比较/捕获DMA请求, TIMERx (x=0,7,14~16,40~44)

<i>TIMER_DMA_MCH</i> 1 <i>D</i>	多模式通道1比较/捕获DMA请求, TIMERx (x=0,7)
<i>TIMER_DMA_MCH</i> 2 <i>D</i>	多模式通道2比较/捕获DMA请求, TIMERx (x=0,7)
<i>TIMER_DMA_MCH</i> 3 <i>D</i>	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 3-1628. 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
<i>TIMERx</i> (x=0~7,14~16,22,23,30,31,40~44,50,51)	参考具体参数
输入参数{in}	
dma	DMA源
<i>TIMER_DMA_UPD</i>	更新DMA请求, TIMERx(x=0~7,14~16,22,23,30,31,40~44,50,51)
<i>TIMER_DMA_CH0</i> <i>D</i>	通道0比较/捕获DMA请求, TIMERx (x=0~4,7,14~16,22,23,30,31,40~44)
<i>TIMER_DMA_CH1</i> <i>D</i>	通道1比较/捕获DMA请求, TIMERx (x=0~4,7,14,22,23,30,31,40~44)
<i>TIMER_DMA_CH2</i> <i>D</i>	通道2比较/捕获DMA请求, TIMERx (x=0~4,7,22,23,30,31)
<i>TIMER_DMA_CH3</i> <i>D</i>	通道3比较/捕获DMA请求, TIMERx (x=0~4,7,22,23,30,31)
<i>TIMER_DMA_CMT</i> <i>D</i>	换相DMA更新请求, TIMERx (x=0,7,14~16,40~44)

<i>TIMER_DMA_TRG D</i>	触发DMA请求, TIMERx (x=0~4,7,14,22,23,30,31,40~44)
<i>TIMER_DMA_MCH 0D</i>	多模式通道0比较/捕获DMA请求, TIMERx (x=0,7,14~16,40~44)
<i>TIMER_DMA_MCH 1D</i>	多模式通道1比较/捕获DMA请求, TIMERx (x=0,7)
<i>TIMER_DMA_MCH 2D</i>	多模式通道2比较/捕获DMA请求, TIMERx (x=0,7)
<i>TIMER_DMA_MCH 3D</i>	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:

```
/* 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 3-1629. 函数 timer_channel_dma_request_source_select

函数名称	timer_channel_dma_request_source_select
函数原型	void timer_channel_dma_request_source_select(uint32_t timer_periph, uint32_t dma_request);
功能描述	外设TIMER的通道DMA请求源选择
先决条件	-
被调用函数	-
输入参数{in}	
timer_periph	TIMER外设
TIMERx(x=0~4,7,14~16,22,23,30,31,40~44)	TIMER外设选择
输入参数{in}	
dma_request	通道的DMA请求源选择
TIMER_DMAREQUEST_CHANNELEVENT	当通道捕获/比较事件发生时, 发送通道n的DMA请求
TIMER_DMAREQUEST_UPDATEEVENT	当更新事件发生时, 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 3-1630. 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=0~4,7,14~16,22,23,30,31,40~44)	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=0~4,7,14~16,22,23,30,31,40~44)
TIMER_DMACFG_DMATA_CTL1	DMA传输起始地址：TIMER_CTL1，TIMERx(x=0~4,7,14~16,22,23,30,31,40~44)
TIMER_DMACFG_DMATA_SMCFG	DMA传输起始地址：TIMER_SMCFG，TIMERx(x=0~4,7,14,22,23,30,31,40~44)
TIMER_DMACFG_DMATA_DMAINTEN	DMA传输起始地址：TIMER_DMAINTEN，TIMERx(x=0~4,7,14~16,22,23,30,31,40~44)
TIMER_DMACFG_DMATA_INTF	DMA传输起始地址：TIMER_INTF，TIMERx(x=0~4,7,14~16,22,23,30,31,40~44)
TIMER_DMACFG_DMATA_SWEVG	DMA传输起始地址：TIMER_SWEVG，TIMERx(x=0~4,7,14~16,22,23,30,31,40~44)
TIMER_DMACFG_DMATA_CHCTL0	DMA传输起始地址：TIMER_CHCTL0，TIMERx(x=0~4,7,14~16,22,23,30,31,40~44)

<i>TIMER_DMACFG_DMATA_CHCTL1</i>	DMA传输起始地址: TIMER_CHCTL1, TIMERx(x=0~4,7,22,23,30,31)
<i>TIMER_DMACFG_DMATA_CHCTL2</i>	DMA传输起始地址: TIMER_CHCTL2, TIMERx(x=0~4,7,14~16,22,23,30,31,40~44)
<i>TIMER_DMACFG_DMATA_CNT</i>	DMA传输起始地址: TIMER_CNT, TIMERx(x=0~4,7,14~16,22,23,30,31,40~44)
<i>TIMER_DMACFG_DMATA_PSC</i>	DMA传输起始地址: TIMER_PSC, TIMERx(x=0~4,7,14~16,22,23,30,31,40~44)
<i>TIMER_DMACFG_DMATA_CAR</i>	DMA传输起始地址: TIMER_CAR, TIMERx(x=0~4,7,14~16,22,23,30,31,40~44)
<i>TIMER_DMACFG_DMATA_CREP0</i>	DMA传输起始地址: TIMER_CREP0, TIMERx(x=0,7,14~16,40~44)
<i>TIMER_DMACFG_DMATA_CH0CV</i>	DMA传输起始地址: TIMER_CH0CV, TIMERx(x=0~4,7,14~16,22,23,30,31,40~44)
<i>TIMER_DMACFG_DMATA_CH1CV</i>	DMA传输起始地址: TIMER_CH1CV, TIMERx(x=0~4,7,14,22,23,30,31)
<i>TIMER_DMACFG_DMATA_CH2CV</i>	DMA传输起始地址: TIMER_CH2CV, TIMERx(x=0~4,7,22,23,30,31)
<i>TIMER_DMACFG_DMATA_CH3CV</i>	DMA传输起始地址: TIMER_CH3CV, TIMERx(x=0~4,7,22,23,30,31)
<i>TIMER_DMACFG_DMATA_CCHP</i>	DMA传输起始地址: TIMER_CCHP, TIMERx(x=0,7,14~16,40~44)
<i>TIMER_DMACFG_DMATA_MCHCTL0</i>	DMA传输起始地址: TIMER_MCHCTL0, TIMERx(x=0,7,14~16,40~44)
<i>TIMER_DMACFG_DMATA_MCHCTL1</i>	DMA传输起始地址: TIMER_MCHCTL1, TIMERx(x=0,7)
<i>TIMER_DMACFG_DMATA_MCHCTL2</i>	DMA传输起始地址: TIMER_MCHCTL2, TIMERx(x=0,7,14~16,40~44)
<i>TIMER_DMACFG_DMATA_MCH0CV</i>	DMA传输起始地址: TIMER_MCH0CV, TIMERx(x=0,7,14~16,40~44)
<i>TIMER_DMACFG_DMATA_MCH1CV</i>	DMA传输起始地址: TIMER_MCH1CV, TIMERx(x=0,7)
<i>TIMER_DMACFG_DMATA_MCH2CV</i>	DMA传输起始地址: TIMER_MCH2CV, TIMERx(x=0,7)
<i>TIMER_DMACFG_DMATA_MCH3CV</i>	DMA传输起始地址: TIMER_MCH3CV, TIMERx(x=0,7)
<i>TIMER_DMACFG_DMATA_CH0COMV_ADD</i>	DMA传输起始地址: TIMER_CH0COMV_ADD, TIMERx(x=0~4,7,14,22,23,30,31)
<i>TIMER_DMACFG_DMATA_CH1COMV_ADD</i>	DMA传输起始地址: TIMER_CH1COMV_ADD, TIMERx(x=0~4,7,14,22,23,30,31)
<i>TIMER_DMACFG_DMATA_CH2COMV_ADD</i>	DMA传输起始地址: TIMER_CH2COMV_ADD, TIMERx(x=0~4,7,22,23,30,31)

<code>DMATA_CH2COMV_ADD</code>	
<code>TIMER_DMACFG_DMATA_CH3COMV_ADD</code>	DMA传输起始地址: <code>TIMER_CH3COMV_ADD</code> , <code>TIMERx</code> ($x=0\sim4,7,22,23,30,31$)
<code>TIMER_DMACFG_DMATA_CTL2</code>	DMA传输起始地址: <code>TIMER_CTL2</code> , <code>TIMERx</code> ($x=0\sim4,7,14\sim16,22,23,30,31,40\sim44$)
<code>TIMER_DMACFG_DMATA_FCCHP0</code>	DMA传输起始地址: <code>TIMER_FCCHP0</code> , <code>TIMERx</code> ($x=0,7,14\sim16,40\sim44$)
<code>TIMER_DMACFG_DMATA_FCCHP1</code>	DMA传输起始地址: <code>TIMER_FCCHP1</code> , <code>TIMERx</code> ($x=0,7$)
<code>TIMER_DMACFG_DMATA_FCCHP2</code>	DMA传输起始地址: <code>TIMER_FCCHP2</code> , <code>TIMERx</code> ($x=0,7$)
<code>TIMER_DMACFG_DMATA_FCCHP3</code>	DMA传输起始地址: <code>TIMER_FCCHP3</code> , <code>TIMERx</code> ($x=0,7$)
<code>TIMER_DMACFG_DMATA_AFCTL0</code>	DMA传输起始地址: <code>TIMER_AFCTL0</code> , <code>TIMERx</code> ($x=0,7,14\sim16,40\sim44$)
<code>TIMER_DMACFG_DMATA_AFCTL1</code>	DMA传输起始地址: <code>TIMER_AFCTL1</code> , <code>TIMERx</code> ($x=0,7$)
<code>TIMER_DMACFG_DMATA_WDGCNT</code>	DMA传输起始地址: <code>TIMER_WDGCNT</code> , <code>TIMERx</code> ($x=0\sim4,7,22,23,30,31$)
<code>TIMER_DMACFG_DMATA_CREP1</code>	DMA传输起始地址: <code>TIMER_CREP1</code> , <code>TIMERx</code> ($x=0,7,14\sim16,40\sim44$)
输入参数{in}	
<code>dma_lenth</code>	DMA传输长度
<code>TIMER_DMACFG_DMATC_xTRANSFER</code>	($x=1\sim38$), 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 3-1631. 函数 timer_event_software_generate

函数名称	timer_event_software_generate
函数原型	void timer_event_software_generate(uint32_t timer_periph, uint32_t event);
功能描述	软件产生事件
先决条件	-
被调用函数	-
输入参数{in}	
timer_periph	TIMER外设
TIMERx(x=0~7,14~16,22,23,30,31,40~44,50,51)	参考具体参数
输入参数{in}	
event	事件源
TIMER_EVENT_SR_C_UPG	更新事件产生, TIMERx(x=0~7,14~16,22,23,30,31,40~44,50,51)
TIMER_EVENT_SR_C_CH0G	通道0捕获或比较事件发生, TIMERx(x=0~4,7,14~16,22,23,30,31,40~44)
TIMER_EVENT_SR_C_CH1G	通道1捕获或比较事件发生, TIMERx(x=0~4,7,14,22,23,30,31,40~44)
TIMER_EVENT_SR_C_CH2G	通道2捕获或比较事件发生, TIMERx(x=0~4,7,22,23,30,31)
TIMER_EVENT_SR_C_CH3G	通道3捕获或比较事件发生, TIMERx(x=0~4,7,22,23,30,31)
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=0~4,7,14,22,23,30,31,40~44)
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=0~4,7,14,22,23,30,31)
TIMER_EVENT_SR	通道1附加比较事件发生, TIMERx(x=0~4,7,14,22,23,30,31)

C_CH1COMADDG	
TIMER_EVENT_SR C_CH2COMADDG	通道2附加比较事件发生, TIMERx(x=0~4,7,22,23,30,31)
TIMER_EVENT_SR C_CH3COMADDG	通道3附加比较事件发生, TIMERx(x=0~4,7,22,23,30,31)
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 3-1632. 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 3-1633. 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 3-1634. 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
TIMERx(x=0,7,14~16,40~44)	TIMER外设选择
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 3-1635. 函数 timer_break_disable

函数名称	timer_break_disable
函数原型	void timer_break_disable(uint32_t timer_periph, uint16_t break_num);
功能描述	禁能TIMER的中止功能
先决条件	只有在TIMERx_CCHP寄存器的PROT [1:0] =00 时, 才可修改
被调用函数	-
输入参数{in}	
timer_periph	TIMER外设
TIMERx(x=0,7,14~16,40~44)	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 3-1636. 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 <i>TIMERx_CCHP</i> 寄存器的 <i>PROT</i> [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 3-1637. 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 <i>TIMERx_CCHP</i> 寄存器的 <i>PROT</i> [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 3-1638. 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 3-1639. 函数 timer_channel_control_shadow_config

函数名称	timer_channel_control_shadow_config
函数原型	void timer_channel_control_shadow_config(uint32_t timer_periph, ControlStatus newvalue);
功能描述	配置通道控制影子寄存器
先决条件	-
被调用函数	-
输入参数{in}	
timer_periph	TIMER外设
TIMERx(x=0,7,14~16,40~44)	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 3-1640. 函数 timer_channel_control_shadow_update_config

函数名称	timer_channel_control_shadow_update_config
函数原型	void timer_channel_control_shadow_update_config(uint32_t timer_periph, uint32_t ccuctl);
功能描述	配置TIMER通道控制影子寄存器更新控制
先决条件	-
被调用函数	-
输入参数{in}	
timer_periph	TIMER外设
TIMERx(x=0,7,14~16,40~44)	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
<i>TIMER_UPDATECTL_CCUC</i>	CMTG位被置1时更新影子寄存器
<i>TIMER_UPDATECTL_CCUTRI</i>	当CMTG位被置1或检测到TRIGI上升沿时，影子寄存器更新
<i>TIMER_UPDATECTL_CCUCOVER</i>	当计数器上溢事件发生时，影子寄存器更新
<i>TIMER_UPDATECTL_CCUCUNDER</i>	当计数器下溢事件发生时，影子寄存器更新
<i>TIMER_UPDATECTL_CCUCOVERUNDER</i>	当计数器上溢/ 下溢事件发生时，影子寄存器更新
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* configure TIMER0 channel control shadow register update when CMTG bit is set */
timer_channel_control_shadow_update_config(TIMER0, TIMER_UPDATECTL_CCUC);
```

函数 timer_channel_output_struct_para_init

函数timer_channel_output_struct_para_init描述见下表：

表 3-1641. 函数 timer_channel_output_struct_para_init

函数名称	timer_channel_output_struct_para_init
函数原型	void timer_channel_output_struct_para_init(timer_oc_parameter_struct* ocpa);
功能描述	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}	
ocpa	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 3-1642. 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=0~4,7,14~16,22,23,30,31,40~44)	参考具体参数
输入参数{in}	
channel	TIMER通道
TIMER_CH_0	通道0, TIMERx(x=0~4,7,14~16,22,23,30,31,40~44)
TIMER_CH_1	通道1, TIMERx(x=0~4,7,14,22,23,30,31,40~44)
TIMER_CH_2	通道2, TIMERx(x=0~4,7,22,23,30,31)
TIMER_CH_3	通道3, TIMERx(x=0~4,7,22,23,30,31)
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(TIMERO, 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 3-1643. 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
TIMEx(x=0~4,7,14~16,22,23,30,31,40~44)	参考具体参数
输入参数{in}	
channel	TIMER通道
TIMER_CH_0	通道0, TIMEx(x=0~4,7,14~16,22,23,30,31,40~44)
TIMER_CH_1	通道1, TIMEx(x=0~4,7,14,22,23,30,31,40~44)
TIMER_CH_2	通道2, TIMEx(x=0~4,7,22,23,30,31)
TIMER_CH_3	通道3, TIMEx(x=0~4,7,22,23,30,31)
TIMER_MCH_0	多模式通道0, TIMEx (x=0,7,14~16,40~44)
TIMER_MCH_1	多模式通道1, TIMEx (x=0,7)
TIMER_MCH_2	多模式通道2, TIMEx (x=0,7)
TIMER_MCH_3	多模式通道3, TIMEx (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	PWM模式0

<code>_PWM0</code>	
<code>TIMER_OC_MODE_PWM1</code>	PWM模式1
<code>TIMER_OC_MODE_DSPM0</code>	可延时的单脉冲模式0
<code>TIMER_OC_MODE_DSPM1</code>	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 3-1644. 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=0~4,7,14~16,22,23,30,31,40~44)	参考具体参数
输入参数{in}	
channel	TIMER通道
TIMER_CH_0	通道0, TIMERx(x=0~4,7,14~16,22,23,30,31,40~44)
TIMER_CH_1	通道1, TIMERx(x=0~4,7,14,22,23,30,31,40~44)
TIMER_CH_2	通道2, TIMERx(x=0~4,7,22,23,30,31)
TIMER_CH_3	通道3, TIMERx(x=0~4,7,22,23,30,31)
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 3-1645. 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=0~4,7,14~16,22,23,30,31,40~44)	参考具体参数
输入参数{in}	
channel	TIMER通道
TIMER_CH_0	通道0, TIMERx(x=0~4,7,14~16,22,23,30,31,40~44)
TIMER_CH_1	通道1, TIMERx(x=0~4,7,14,22,23,30,31,40~44)
TIMER_CH_2	通道2, TIMERx(x=0~4,7,22,23,30,31)
TIMER_CH_3	通道3, TIMERx(x=0~4,7,22,23,30,31)
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

<i>DOW_ENABLE</i>	
<i>TIMER_OMC_SHA</i> <i>DOW_DISABLE</i>	多模式通道输出比较影子寄存器禁能
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/*configure TIMER0 channel 0 output shadow function */
```

```
timer_channel_output_shadow_config(TIMER0,                                TIMER_CH_0,  
TIMER_OC_SHADOW_ENABLE);
```

函数 timer_channel_output_clear_config

函数timer_channel_output_clear_config描述见下表：

表 3-1646. 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=0~4,7,14~16,22,23,30,31,40~44)	参考具体参数
输入参数{in}	
channel	TIMER通道
TIMER_CH_0	通道0, TIMERx(x=0~4,7,14~16,22,23,30,31,40~44)
TIMER_CH_1	通道1, TIMERx(x=0~4,7,14,22,23,30,31,40~44)
TIMER_CH_2	通道2, TIMERx(x=0~4,7,22,23,30,31)
TIMER_CH_3	通道3, TIMERx(x=0~4,7,22,23,30,31)
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

<code>TIMER_OC_CLEAR_DISABLE</code>	通道比较输出清0功能禁能
<code>TIMER_OMC_CLEAR_ENABLE</code>	多模式通道比较输出清0功能使能
<code>TIMER_OMC_CLEAR_DISABLE</code>	多模式通道比较输出清0功能禁能
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* configure TIMER0 channel 0 output clear function */
```

```
timer_channel_output_clear_config(TIMER0, TIMER_CH_0,  
TIMER_OC_CLEAR_ENABLE);
```

函数 timer_channel_output_polarity_config

函数timer_channel_output_polarity_config描述见下表：

表 3-1647. 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=0~4,7,14~16,22,23,30,31,40~44)	参考具体参数
输入参数{in}	
channel	TIMER通道
TIMER_CH_0	通道0, TIMERx(x=0~4,7,14~16,22,23,30,31,40~44)
TIMER_CH_1	通道1, TIMERx(x=0~4,7,14,22,23,30,31,40~44)
TIMER_CH_2	通道2, TIMERx(x=0~4,7,22,23,30,31)
TIMER_CH_3	通道3, TIMERx(x=0~4,7,22,23,30,31)
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
<i>TIMER_OC_POLARITY_HIGH</i>	通道输出极性高电平有效
<i>TIMER_OC_POLARITY_LOW</i>	通道输出极性低电平有效
<i>TIMER_OMC_POLARITY_HIGH</i>	多模式通道输出极性高电平有效
<i>TIMER_OMC_POLARITY_LOW</i>	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 3-1648. 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
<i>TIMERx(x=0~4,7,14~16,22,23,30,31,40~44)</i>	TIMER外设选择
输入参数{in}	
channel	TIMER通道
<i>TIMER_CH_0</i>	通道0, TIMERx(x=0~4,7,14~16,22,23,30,31,40~44)
<i>TIMER_CH_1</i>	通道1, TIMERx(x=0~4,7,14,22,23,30,31,40~44)
<i>TIMER_CH_2</i>	通道2, TIMERx(x=0~4,7,22,23,30,31)
<i>TIMER_CH_3</i>	通道3, TIMERx(x=0~4,7,22,23,30,31)
输入参数{in}	
ocpolarity	互补通道输出极性

<i>TIMER_OCN_POLARITY_HIGH</i>	互补通道输出极性高电平有效
<i>TIMER_OCN_POLARITY_LOW</i>	互补通道输出极性低电平有效
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* configure TIMER0 channel 0 complementary output polarity */
```

```
timer_channel_complementary_output_polarity_config(TIMER0, TIMER_CH_0,  
TIMER_OCN_POLARITY_HIGH);
```

函数 timer_channel_output_state_config

函数timer_channel_output_state_config描述见下表：

表 3-1649. 函数 timer_channel_output_state_config

函数名称	timer_channel_output_state_config
函数原型	void timer_channel_output_state_config(uint32_t timer_periph, uint16_t channel, uint32_t state);
功能描述	配置通道状态
先决条件	-
被调用函数	-
输入参数{in}	
timer_periph	TIMER外设
<i>TIMERx</i> (<i>x</i> =0~4,7,14~16,22,23,30,31,40~44)	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> =0~4,7,14~16,22,23,30,31,40~44)
<i>TIMER_CH_1</i>	通道1, <i>TIMERx</i> (<i>x</i> =0~4,7,14,22,23,30,31,40~44)
<i>TIMER_CH_2</i>	通道2, <i>TIMERx</i> (<i>x</i> =0~4,7,22,23,30,31)
<i>TIMER_CH_3</i>	通道3, <i>TIMERx</i> (<i>x</i> =0~4,7,22,23,30,31)
<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, <i>TIMERx</i> (<i>x</i> =0,7)
输入参数{in}	
state	通道状态
<i>TIMER_CCX_ENAB</i>	通道使能

LE	
TIMER_CCX_DISABLE	通道禁能
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 3-1650. 函数 timer_channel_complementary_output_state_config

函数名称	timer_channel_complementary_output_state_config
函数原型	void timer_channel_complementary_output_state_config(uint32_t timer_periph, uint16_t channel, uint16_t ocnstate);
功能描述	配置互补通道输出状态
先决条件	-
被调用函数	-
输入参数{in}	
timer_periph	TIMER外设
TIMERx(x=0,7,14~16,40~44)	TIMER外设选择
输入参数{in}	
channel	TIMER通道
TIMER_CH_0	通道0, TIMERx(x=0,7,14~16,40~44)
TIMER_CH_1	通道1, TIMERx(x=0,7,14,40~44)
TIMER_CH_2	通道2, TIMERx(x=0,7)
TIMER_CH_3	通道3, TIMERx(x=0,7)
输入参数{in}	
state	互补通道状态
TIMER_CCXN_ENABLE	互补通道使能
TIMER_CCXN_DISABLE	互补通道禁能

输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* configure TIMER0 channel 0 complementary output enable state */
```

```
timer_channel_complementary_output_state_config(TIMER0, TIMER_CH_0,  
TIMER_CCXN_ENABLE);
```

函数 timer_channel_input_struct_para_init

函数timer_channel_input_struct_para_init描述见下表：

表 3-1651. 函数 timer_channel_input_struct_para_init

函数名称	timer_channel_input_struct_para_init
函数原型	void timer_channel_input_struct_para_init(timer_ic_parameter_struct* icpara);
功能描述	将TIMER通道输入参数结构体中所有参数初始化为默认值
先决条件	-
被调用函数	-
输入参数{in}	
icpara	通道输入结构体，详见 结构体timer_ic_parameter_struct 。
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* initialize TIMER channel input parameter struct with a default value */
```

```
timer_ic_parameter_struct timer_icinitpara;
```

```
timer_channel_input_struct_para_init(&timer_icinitpara);
```

函数 timer_input_capture_config

函数timer_input_capture_config描述见下表：

表 3-1652. 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
<i>TIMERx(x=0~4,7,14~16,22,23,30,31,40~44)</i>	参考具体参数
输入参数{in}	
channel	TIMER通道
<i>TIMER_CH_0</i>	通道0, <i>TIMERx(x=0~4,7,14~16,22,23,30,31,40~44)</i>
<i>TIMER_CH_1</i>	通道1, <i>TIMERx(x=0~4,7,14,22,23,30,31,40~44)</i>
<i>TIMER_CH_2</i>	通道2, <i>TIMERx(x=0~4,7,22,23,30,31)</i>
<i>TIMER_CH_3</i>	通道3, <i>TIMERx(x=0~4,7,22,23,30,31)</i>
<i>TIMER_MCH_0</i>	多模式通道0, <i>TIMERx(x=0,7,14~16,40~44)</i>
<i>TIMER_MCH_1</i>	多模式通道1, <i>TIMERx(x=0,7)</i>
<i>TIMER_MCH_2</i>	多模式通道2, <i>TIMERx(x=0,7)</i>
<i>TIMER_MCH_3</i>	多模式通道3, <i>TIMERx(x=0,7)</i>
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 3-1653. 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=0~4,7,14~16,22,23,30,31,40~44)	参考具体参数
输入参数{in}	
channel	TIMER通道
TIMER_CH_0	通道0, TIMERx(x=0~4,7,14~16,22,23,30,31,40~44)
TIMER_CH_1	通道1, TIMERx(x=0~4,7,14,22,23,30,31,40~44)
TIMER_CH_2	通道2, TIMERx(x=0~4,7,22,23,30,31)
TIMER_CH_3	通道3, TIMERx(x=0~4,7,22,23,30,31)
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
TIMER_IC_PSC_DIV1	不分频
TIMER_IC_PSC_DIV2	2分频
TIMER_IC_PSC_DIV4	4分频
TIMER_IC_PSC_DIV8	8分频
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* configure TIMER0 channel 0 input capture prescaler value */
timer_channel_input_capture_prescaler_config(TIMER0, TIMER_CH_0,
TIMER_IC_PSC_DIV2);
```

函数 timer_channel_capture_value_register_read

函数timer_channel_capture_value_register_read描述见下表:

表 3-1654. 函数 timer_channel_capture_value_register_read

函数名称	timer_channel_capture_value_register_read
------	---

函数原型	uint32_t timer_channel_capture_value_register_read(uint32_t timer_periph, uint16_t channel);
功能描述	读取通道捕获值
先决条件	-
被调用函数	-
输入参数{in}	
timer_periph	TIMER外设
TIMERx(x=0~4, 7, 14~16, 22, 23, 30, 31, 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, 30, 31, 40~44)
TIMER_CH_1	通道1, TIMERx(x=0~4, 7, 14, 22, 23, 30, 31, 40~44)
TIMER_CH_2	通道2, TIMERx(x=0~4, 7, 22, 23, 30, 31)
TIMER_CH_3	通道3, TIMERx(x=0~4, 7, 22, 23, 30, 31)
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 3-1655. 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(x=0~4, 7, 14~16, 22, 23, 30, 31, 40~44)</i>	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 3-1656. 函数 timer_hall_mode_config

函数名称	timer_hall_mode_config
函数原型	void timer_hall_mode_config(uint32_t timer_periph, uint32_t hallmode);
功能描述	配置TIMER的HALL接口功能
先决条件	-
被调用函数	-
输入参数{in}	
timer_periph	TIMER外设
<i>TIMERx(x=0~4, 7, 14~16, 22, 23, 30, 31, 40~44)</i>	TIMER外设选择
输入参数{in}	

hallmode	HALL接口功能状态
<i>TIMER_HALLINTE RFACE_ENABLE</i>	HALL接口使能
<i>TIMER_HALLINTE RFACE_DISABLE</i>	HALL接口禁能
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* configure TIMER0 hall sensor mode */
```

```
timer_hall_mode_config(TIMER0, TIMER_HALLINTERFACE_ENABLE);
```

函数 timer_multi_mode_channel_output_parameter_struct_init

函数timer_multi_mode_channel_output_parameter_struct_init描述见下表：

表 3-1657. 函数 timer_multi_mode_channel_output_parameter_struct_init

函数名称	timer_multi_mode_channel_output_parameter_struct_init
函数原型	void timer_multi_mode_channel_output_parameter_struct_init(timer_omc_parameter_struct *omcpara);
功能描述	将TIMER多模式通道输出参数结构体中所有参数初始化为默认值
先决条件	-
被调用函数	-
输入参数{in}	
omcpara	多模式通道输出参数结构体，详见 结构体timer_omc_parameter_struct 。
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* initialize TIMER multi mode channel output parameter struct with a default value */
```

```
timer_omc_parameter_struct timer_omcinitpara;
```

```
timer_multi_mode_channel_output_parameter_struct_init(&timer_omcinitpara);
```

函数 timer_multi_mode_channel_output_config

函数timer_multi_mode_channel_output_config描述见下表：

表 3-1658. 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 3-1659. 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 3-1660. 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=0~4,7,14,22,23,30,31,40~44)	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=0~4,7,14,22,23,30,31,40~44))
TIMER_SMCFG_T RGSEL_ITI1	内部触发输入1(ITI1, TIMERx(x=0~4,7,14,22,23,30,31,40~44))
TIMER_SMCFG_T RGSEL_ITI2	内部触发输入2(ITI2, TIMERx(x=0~4,7,14,22,23,30,31,40~44))
TIMER_SMCFG_T RGSEL_ITI3	内部触发输入3(ITI3, TIMERx(x=0~4,7,14,22,23,30,31,40~44))
TIMER_SMCFG_T RGSEL_CIOF_ED	TI0的边沿检测(CIOF_ED, TIMERx(x=0~4,7,14,22,23,30,31,40~44))
TIMER_SMCFG_T RGSEL_CIOFE0	滤波后的通道0输入(CIOFE0, TIMERx(x=0~4,7,14,22,23,30,31,40~44))
TIMER_SMCFG_T RGSEL_CI1FE1	滤波后的通道1输入(CI1FE1, TIMERx(x=0~4,7,14,22,23,30,31,40~44))
TIMER_SMCFG_T RGSEL_ETIFP	滤波后的外部触发输入(ETIFP, TIMERx(x=0~4,7,22,23,30,31))
TIMER_SMCFG_T RGSEL_CI2FE2	滤波后的通道2输入(TIMERx(x=0,7))
TIMER_SMCFG_T RGSEL_CI3FE3	滤波后的通道3输入(TIMERx(x=0,7))
TIMER_SMCFG_T RGSEL_MCI0FEM0	滤波后的多模式通道0输入(TIMERx(x=0,7,14,40~44))
TIMER_SMCFG_T RGSEL_MCI1FEM1	滤波后的多模式通道1输入(TIMERx(x=0,7))
TIMER_SMCFG_T RGSEL_MCI2FEM2	滤波后的多模式通道2输入(TIMERx(x=0,7))
TIMER_SMCFG_T RGSEL_MCI3FEM3	滤波后的多模式通道3输入(TIMERx(x=0,7))
TIMER_L0_SMCFG _TRGSEL_ITI4	内部触发输入4对于通用L0定时器(TIMERx(x=1,2,22,23,30,31))
TIMER_L0_SMCFG _TRGSEL_ITI5	内部触发输入5对于通用L0定时器(TIMERx(x=1,22,23,30))
TIMER_L0_SMCFG _TRGSEL_ITI6	内部触发输入6对于通用L0定时器
TIMER_L0_SMCFG _TRGSEL_ITI7	内部触发输入7对于通用L0定时器
TIMER_L0_SMCFG _TRGSEL_ITI8	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=0~4,7,23,30,31))
<i>TIMER_SMCFG_T</i> <i>RGSEL_ITI13</i>	内部触发输入13(ITI13, TIMERx(x=0~4,7,22,30,31))
<i>TIMER_SMCFG_T</i> <i>RGSEL_ITI14</i>	内部触发输入14(ITI14, TIMERx(x=0~4,7,14,22,23,30,31,40~44))
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 3-1661. 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=0~7,14,22,23,30,31,50,51)	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=0~7,14,22,23,30,31,50,51))
TIMER_TRI_OUT0_SRC_ENABLE	TIMER使能信号作为TRGO0(TIMERx(x=0~7,14,22,23,30,31,50,51))
TIMER_TRI_OUT0_	更新事件作为TRGO0 (TIMERx(x=0~7,14,22,23,30,31,50,51))

<i>SRC_UPDATE</i>	
<i>TIMER_TRI_OUT0_SRC_CH0</i>	通道0捕获/比较事件作为TRGO0 (TIMERx(x=0~4,7,14,22,23,30,31,40~44))
<i>TIMER_TRI_OUT0_SRC_O0CPRE</i>	O0CPRE作为触发输出TRGO0(TIMERx(x=0~4,7,14,22,23,30,31,40~44))
<i>TIMER_TRI_OUT0_SRC_O1CPRE</i>	O1CPRE作为触发输出TRGO0(TIMERx(x=0~4,7,14,22,23,30,31,40~44))
<i>TIMER_TRI_OUT0_SRC_O2CPRE</i>	O2CPRE作为触发输出TRGO0(TIMERx(x=0~4,7,22,23,30,31))
<i>TIMER_TRI_OUT0_SRC_O3CPRE</i>	O3CPRE作为触发输出TRGO0(TIMERx(x=0~4,7,22,23,30,31))
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 3-1662. 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_	通道0捕获/比较事件作为TRGO1

<i>SRC_CH0</i>	
<i>TIMER_TRI_OUT1_SRC_O0CPRE</i>	O0CPRE作为触发输出TRGO1
<i>TIMER_TRI_OUT1_SRC_O1CPRE</i>	O1CPRE作为触发输出TRGO1
<i>TIMER_TRI_OUT1_SRC_O2CPRE</i>	O2CPRE作为触发输出TRGO1
<i>TIMER_TRI_OUT1_SRC_O3CPRE</i>	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 3-1663. 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
<i>TIMERx</i> (<i>x</i> =0~4,7,14,22,23,30,31,40~44)	TIMER外设选择
输入参数{in}	
slavemode	从模式
<i>TIMER_SLAVE_MODE_DISABLE</i>	关闭从模式， <i>TIMERx</i> (<i>x</i> =0~4,7,14,22,23,30,31,40~44)
<i>TIMER_ENCODER_MODE0</i>	编码器模式0， <i>TIMERx</i> (<i>x</i> =0~4,7,22,23,30,31)
<i>TIMER_ENCODER_MODE1</i>	编码器模式1， <i>TIMERx</i> (<i>x</i> =0~4,7,22,23,30,31)
<i>TIMER_ENCODER_MODE2</i>	编码器模式2， <i>TIMERx</i> (<i>x</i> =0~4,7,22,23,30,31)
<i>TIMER_SLAVE_MODE</i>	复位模式， <i>TIMERx</i> (<i>x</i> =0~4,7,14,22,23,30,31,40~44)

<i>DE_RESTART</i>	
<i>TIMER_SLAVE_MODE_PAUSE</i>	暂停模式, <i>TIMERx</i> (<i>x</i> =0~4,7,14,22,23,30,31,40~44)
<i>TIMER_SLAVE_MODE_EVENT</i>	事件模式, <i>TIMERx</i> (<i>x</i> =0~4,7,14,22,23,30,31,40~44)
<i>TIMER_SLAVE_MODE_EXTERNAL0</i>	外部时钟模式0, <i>TIMERx</i> (<i>x</i> =0~4,7,14,22,23,30,31,40~44)
<i>TIMER_SLAVE_MODE_RESTART_EVENT</i>	复位+事件模式, <i>TIMERx</i> (<i>x</i> =0~4,7,14,22,23,30,31,40~44)
<i>TIMER_NONQUAD_MODE0</i>	非正交编码器模式0, <i>TIMERx</i> (<i>x</i> =0~4,7,22,23,30,31)
<i>TIMER_NONQUAD_MODE1</i>	非正交编码器模式1, <i>TIMERx</i> (<i>x</i> =0~4,7,22,23,30,31)
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 3-1664. 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
<i>TIMERx</i> (<i>x</i> =0~4,7,14,22,23,30,31,40~44)	TIMER外设选择
输入参数{in}	
masterslave	主从模式使能状态
<i>TIMER_MASTER_SLAVE_MODE_ENABLE</i>	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 3-1665. 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=0~4,7,22,23,30,31)	TIMER外设选择
输入参数{in}	
extprescaler	外部触发预分频
TIMER_EXT_TRIP_SC_OFF	不分频
TIMER_EXT_TRIP_SC_DIV2	2分频
TIMER_EXT_TRIP_SC_DIV4	4分频
TIMER_EXT_TRIP_SC_DIV8	8分频
输入参数{in}	
expolarity	外部触发输入极性
TIMER_ETP_FALLING	低电平或者下降沿有效
TIMER_ETP_RISING	高电平或者上升沿有效

输入参数{in}	
extfilter	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 3-1666. 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=0~4,7,22,23,30,31)	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
<i>TIMER_IC_POLARITY_RISING</i>	捕获上升边沿
<i>TIMER_IC_POLARITY_FALLING</i>	捕获下降边沿
<i>TIMER_IC_POLARITY_BOTH_EDGE</i>	双边沿有效
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* configure TIMER0 quadrature decoder mode */
```

```
timer_quadrature_decoder_mode_config(TIMER0, TIMER_ENCODER_MODE0,  
TIMER_IC_POLARITY_RISING, TIMER_IC_POLARITY_RISING);
```

函数 timer_non_quadrature_decoder_mode_config

函数timer_non_quadrature_decoder_mode_config描述见下表：

表 3-1667. 函数 timer_non_quadrature_decoder_mode_config

函数名称	timer_non_quadrature_decoder_mode_config
函数原型	void timer_non_quadrature_decoder_mode_config(uint32_t timer_periph, uint32_t decomode, uint16_t 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
<i>TIMERx(x=0~4,7,22,23,30,31)</i>	TIMER外设选择
输入参数{in}	
decomode	编码器模式
<i>TIMER_NONQUAD_MODE0</i>	非正交编码器模式0
<i>TIMER_NONQUAD_MODE1</i>	非正交编码器模式1
输入参数{in}	
ic1polarity	IC1输入极性
<i>TIMER_IC_POLARITY_RISING</i>	捕获上升边沿

<i>TIMER_IC_POLARITY_FALLING</i>	捕获下降边沿
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 3-1668. 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
<i>TIMERx(x=0~4,7,14,22,23,30,31,40~44)</i>	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 3-1669. 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=0~4,7,14,22,23,30,31,40~44)	TIMER外设选择
输入参数{in}	
intrigger	被选择的内部触发源
TIMER_SMCFG_T RGSEL_ITI0	内部触发输入0(ITI0, TIMERx(x=0~4,7,14,22,23,30,31,40~44))
TIMER_SMCFG_T RGSEL_ITI1	内部触发输入1(ITI1, TIMERx(x=0~4,7,14,22,23,30,31,40~44))
TIMER_SMCFG_T RGSEL_ITI2	内部触发输入2(ITI2, TIMERx(x=0~4,7,14,22,23,30,31,40~44))
TIMER_SMCFG_T RGSEL_ITI3	内部触发输入3(ITI3, TIMERx(x=0~4,7,14,22,23,30,31,40~44))
TIMER_L0_SMCFG _TRGSEL_ITI4	内部触发输入4对于通用L0定时器(TIMERx(x=1,2,22,23,30,31))
TIMER_L0_SMCFG _TRGSEL_ITI5	内部触发输入5对于通用L0定时器(TIMERx(x=1,22,23,30))
TIMER_L0_SMCFG _TRGSEL_ITI7	内部触发输入7对于通用L0定时器(TIMERx(x=4))
TIMER_L0_SMCFG _TRGSEL_ITI9	内部触发输入9对于通用L0定时器(TIMERx(x=22,23))
TIMER_L0_SMCFG _TRGSEL_ITI10	内部触发输入10对于通用L0定时器(TIMERx(x=22,23))
TIMER_L0_SMCFG _TRGSEL_ITI11	内部触发输入11对于通用L0定时器(TIMERx(x=22,23))
TIMER_SMCFG_T RGSEL_ITI12	内部触发输入12(ITI12, TIMERx(x=0~4,7,23,30,31))
TIMER_SMCFG_T RGSEL_ITI13	内部触发输入13(ITI13, TIMERx(x=0~4,7,22,30,31))
TIMER_SMCFG_T RGSEL_ITI14	内部触发输入14(ITI14, TIMERx(x=0~4,7,14,22,23,30,31,40~44))
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 3-1670. 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=0~4,7,14,22,23,30,31,40~44)	TIMER外设选择
输入参数{in}	
extrigger	外部触发源
TIMER_SMCFG_TRGSEL_CIOF_ED	TI0的边沿检测(CIOF_ED, TIMERx(x=0~4,7,14,22,23,30,31,40~44))
TIMER_SMCFG_TRGSEL_CIOFE0	滤波后的通道0输入(CIOFE0, TIMERx(x=0~4,7,14,22,23,30,31,40~44))
TIMER_SMCFG_TRGSEL_CI1FE1	滤波后的通道1输入(CI1FE1, TIMERx(x=0~4,7,14,22,23,30,31,40~44))
TIMER_SMCFG_TRGSEL_CI2FE2	滤波后的通道2输入(TIMERx(x=0,7))
TIMER_SMCFG_TRGSEL_CI3FE3	滤波后的通道3输入(TIMERx(x=0,7))
TIMER_SMCFG_TRGSEL_MCI0FEM0	滤波后的多模式通道0输入(TIMERx(x=0,7,14,40~44))
TIMER_SMCFG_TRGSEL_MCI1FEM1	滤波后的多模式通道1输入(TIMERx(x=0,7))
TIMER_SMCFG_TRGSEL_MCI2FEM2	滤波后的多模式通道2输入(TIMERx(x=0,7))
TIMER_SMCFG_TRGSEL_MCI3FEM3	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
TIMER_IC_POLARITY_RISING	外部触发源高电平或者上升沿有效
TIMER_IC_POLARITY_FALLING	外部触发源低电平或者下降沿有效

<i>TIMER_IC_POLARITY_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 CI0FE0 as external clock input */
```

```
timer_external_trigger_as_external_clock_config(TIMER0,  
TIMER_SMCFG_TRGSEL_CI0FE0, TIMER_IC_POLARITY_RISING, 0);
```

函数 timer_external_clock_mode0_config

函数timer_external_clock_mode0_config描述见下表：

表 3-1671. 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
<i>TIMERx(x=0~4,7,14,22,23,30,31,40~44)</i>	TIMER外设选择
输入参数{in}	
extprescaler	ETI触发源预分频值
<i>TIMER_EXT_TRI_PSC_OFF</i>	不分频
<i>TIMER_EXT_TRI_PSC_DIV2</i>	2分频
<i>TIMER_EXT_TRI_PSC_DIV4</i>	4分频
<i>TIMER_EXT_TRI_PSC_DIV8</i>	8分频
输入参数{in}	
expolarity	ETI触发源极性
<i>TIMER_ETP_FALLING</i>	下降沿或者低电平有效

<i>TIMER_ETP_RISING</i>	上升沿或者高电平有效
输入参数{in}	
extfilter	ETI触发源滤波参数（0~15）
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* configure TIMER0 the external clock 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 3-1672. 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
<i>TIMERx</i> (<i>x</i> =0~4,7,22,23,30,31)	TIMER外设选择
输入参数{in}	
extprescaler	ETI触发源预分频值
<i>TIMER_EXT_TRI_PSC_OFF</i>	不分频
<i>TIMER_EXT_TRI_PSC_DIV2</i>	2分频
<i>TIMER_EXT_TRI_PSC_DIV4</i>	4分频
<i>TIMER_EXT_TRI_PSC_DIV8</i>	8分频
输入参数{in}	
expolarity	ETI触发源极性
<i>TIMER_ETP_FALLING</i>	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 3-1673. 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, 30, 31)	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 3-1674. 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, 30, 31, 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 3-1675. 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

<i>TIMER_OUTSEL_ENABLE</i>	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 3-1676. 函数 timer_commutation_control_shadow_register_config

函数名称	timer_commutation_control_shadow_register_config
函数原型	void timer_commutation_control_shadow_register_config(uint32_t timer_periph, uint16_t ccssel);
功能描述	配置换相控制影子寄存器更新选择
先决条件	-
被调用函数	-
输入参数{in}	
timer_periph	TIMER外设
TIMERx(x=0,7,14~16,40~44)	TIMER外设选择
输入参数{in}	
ccssel	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 3-1677. 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=0~4,7,14,22,23,30,31,40~44)	参考具体参数
输入参数{in}	
channel	TIMER通道
TIMER_CH_0	通道0, TIMERx(x=0~4,7,14,22,23,30,31,40~44)
TIMER_CH_1	通道1, TIMERx(x=0~4,7,14,22,23,30,31,40~44)
TIMER_CH_2	通道2, TIMERx(x=0~4,7,22,23,30,31)
TIMER_CH_3	通道3, TIMERx(x=0~4,7,22,23,30,31)
输入参数{in}	
pulsesel	输出匹配脉冲选择
TIMER_PULSE_OUTPUT_NORMAL	通道输出正常
TIMER_PULSE_OUTPUT_CNT_UP	仅在向上计数时, 通道输出脉冲
TIMER_PULSE_OUTPUT_CNT_DOWN	仅在向下计数时, 通道输出脉冲
TIMER_PULSE_OUTPUT_CNT_BOTH	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 3-1678. 函数 `timer_channel_composite_pwm_mode_config`

函数名称	<code>timer_channel_composite_pwm_mode_config</code>
函数原型	<code>void timer_channel_composite_pwm_mode_config(uint32_t timer_periph, uint32_t channel, ControlStatus newvalue);</code>
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}	
<code>timer_periph</code>	TIMER外设
<code>TIMERx(x=0~4, 7, 14, 22, 23, 30, 31, 40~44)</code>	参考具体参数
输入参数{in}	
<code>channel</code>	TIMER通道
<code>TIMER_CH_0</code>	通道0, <code>TIMERx(x=0~4, 7, 14, 22, 23, 30, 31, 40~44)</code>
<code>TIMER_CH_1</code>	通道1, <code>TIMERx(x=0~4, 7, 14, 22, 23, 30, 31, 40~44)</code>
<code>TIMER_CH_2</code>	通道2, <code>TIMERx(x=0~4, 7, 22, 23, 30, 31)</code>
<code>TIMER_CH_3</code>	通道3, <code>TIMERx(x=0~4, 7, 22, 23, 30, 31)</code>
输入参数{in}	
<code>newvalue</code>	控制状态
<code>ENABLE</code>	使能
<code>DISABLE</code>	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 3-1679. 函数 `timer_channel_composite_pwm_mode_output_pulse_value_config`

函数名称	<code>timer_channel_composite_pwm_mode_output_pulse_value_config</code>
函数原型	<code>void timer_channel_composite_pwm_mode_output_pulse_value_config(uint32_t timer_periph, uint32_t channel, uint32_t pulse, uint32_t add_pulse);</code>
功能描述	配置TIMER的复合PWM模式输出脉冲值

先决条件	-
被调用函数	-
输入参数{in}	
timer_periph	TIMER外设
TIMERx(x=0~4,7,14,22,23,30,31,40~44)	参考具体参数
输入参数{in}	
channel	TIMER通道
TIMER_CH_0	通道0, TIMERx(x=0~4,7,14,22,23,30,31,40~44)
TIMER_CH_1	通道1, TIMERx(x=0~4,7,14,22,23,30,31,40~44)
TIMER_CH_2	通道2, TIMERx(x=0~4,7,22,23,30,31)
TIMER_CH_3	通道3, TIMERx(x=0~4,7,22,23,30,31)
输入参数{in}	
pulse	通道比较值 (0~0xFFFFFFFF)
输入参数{in}	
add_pulse	通道附加比较值 (0~0xFFFFFFFF)
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* configure TIMER0 channel 0 output pulse value */
```

```
timer_channel_output_pulse_value_config(TIMER0, TIMER_CH_0, 399, 3999);
```

函数 timer_channel_additional_compare_value_config

函数timer_channel_additional_compare_value_config描述见下表:

表 3-1680. 函数 timer_channel_additional_compare_value_config

函数名称	timer_channel_additional_compare_value_config
函数原型	void timer_channel_additional_compare_value_config(uint32_t timer_periph, uint16_t channel, uint32_t value)
功能描述	配置TIMER通道附加比较寄存器
先决条件	-
被调用函数	-
输入参数{in}	
timer_periph	TIMER外设
TIMERx(x=0~4,7,14,22,23,30,31,40~44)	参考具体参数
输入参数{in}	
channel	TIMER通道
TIMER_CH_0	通道0, TIMERx(x=0~4,7,14,22,23,30,31,40~44)

<i>TIMER_CH_1</i>	通道1, <i>TIMERx</i> (<i>x</i> =0~4,7,14,22,23,30,31,40~44)
<i>TIMER_CH_2</i>	通道2, <i>TIMERx</i> (<i>x</i> =0~4,7,22,23,30,31)
<i>TIMER_CH_3</i>	通道3, <i>TIMERx</i> (<i>x</i> =0~4,7,22,23,30,31)
输入参数{in}	
value	通道附加比较值 (0~0xFFFFFFFF)
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* configure TIMER0 channel 0 additional compare value */
```

```
timer_channel_additional_compare_value_config (TIMER0, TIMER_CH_0, 399);
```

函数 timer_channel_additional_output_shadow_config

函数timer_channel_additional_output_shadow_config描述见下表:

表 3-1681. 函数 timer_channel_additional_output_shadow_config

函数名称	timer_channel_additional_output_shadow_config
函数原型	void timer_channel_additional_output_shadow_config(uint32_t timer_periph, uint16_t channel, uint16_t aocshadow);
功能描述	配置TIMER通道附加输出比较影子寄存器功能
先决条件	-
被调用函数	-
输入参数{in}	
timer_periph	TIMER外设
<i>TIMERx</i> (<i>x</i> =0~4,7,14,22,23,30,31,40~44)	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> =0~4,7,14,22,23,30,31,40~44)
<i>TIMER_CH_1</i>	通道1, <i>TIMERx</i> (<i>x</i> =0~4,7,14,22,23,30,31,40~44)
<i>TIMER_CH_2</i>	通道2, <i>TIMERx</i> (<i>x</i> =0~4,7,22,23,30,31)
<i>TIMER_CH_3</i>	通道3, <i>TIMERx</i> (<i>x</i> =0~4,7,22,23,30,31)
输入参数{in}	
aocshadow	通道附加输出比较影子寄存器状态
<i>TIMER_ADD_SHADOW_ENABLE</i>	通道附加输出比较影子寄存器使能
<i>TIMER_ADD_SHADOW_DISABLE</i>	通道附加输出比较影子寄存器禁能
输出参数{out}	
-	-

返回值	
-	-

例如：

```
/*configure TIMER0 channel 0 additional output shadow function */
```

```
timer_channel_additional_output_shadow_config      (TIMER0,          TIMER_CH_0,
TIMER_OC_SHADOW_ENABLE);
```

函数 timer_channel_additional_compare_value_read

函数timer_channel_additional_compare_value_read描述见下表：

表 3-1682. 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=0~4,7,14,22,23,30,31,40~44)	TIMER外设选择
输入参数{in}	
channel	TIMER通道
TIMER_CH_0	通道0, TIMERx(x=0~4,7,14,22,23,30,31,40~44)
TIMER_CH_1	通道1, TIMERx(x=0~4,7,14,22,23,30,31,40~44)
TIMER_CH_2	通道2, TIMERx(x=0~4,7,22,23,30,31)
TIMER_CH_3	通道3, TIMERx(x=0~4,7,22,23,30,31)
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 3-1683. 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
ENABLE	使能
DISABLE	禁能
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* configure the TIMER break source */
```

```
timer_break_external_source_config(TIMER0,    TIMER_BREAK0,    TIMER_BRKIN0,
ENABLE);
```

函数 timer_break_external_polarity_config

函数timer_break_external_polarity_config描述见下表:

表 3-1684. 函数 timer_break_external_polarity_config

函数名称	timer_break_external_polarity_config
------	--------------------------------------

例如：

```
timer_break_external_polarity_config(TIMER0,    TIMER_BREAK0,    TIMER_BRKIN0,  
TIMER BRKIN POLARITY HIGH);
```

函数timer break lock config描述见下表:

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 3-1686. 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)

<i>TIMER_BREAK1</i>	BREAK1输入信号, <i>TIMERx</i> (<i>x</i> =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 3-1687. 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
<i>TIMERx</i> (<i>x</i> =0,7)	参考具体参数
输入参数{in}	
channel	TIMER通道
<i>TIMER_CH_0</i>	通道0
<i>TIMER_CH_1</i>	通道1
<i>TIMER_CH_2</i>	通道2
<i>TIMER_CH_3</i>	通道3
输入参数{in}	
newvalue	控制状态
ENABLE	使能
DISABLE	禁能
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* configure TIMER0 channel 0 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 3-1688. 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 3-1689. 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 3-1690. 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
TIMEx(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 3-1691. 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=0~4,7,22,23,30,31)	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 3-1692. 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=0~4,7,22,23,30,31)	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 3-1693. 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=0~4,7,22,23,30,31)	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 3-1694. 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=0~4,7,22,23,30,31)	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 3-1695. 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, 30, 31, 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 3-1696. 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UPIFBUS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, 30, 31, 40~44, 50, 51)	TIMER外设选择
输出参数{out}	
-	-
返回值	
UPIFBUSStatus	UPIFBUS位状态
VALID_SET	UPIFBUS位有效，返回值为1
VALID_RESET	UPIFBUS位有效，返回值为0

INVALID	UPIFBU位无效
---------	-----------

例如:

```
/* get the UPIFBU bit in the TIMEx_CNT register */
```

```
UPIFBUStatus upstatus = INVALID;
```

```
upstatus = timer_upifbu_bit_get (TIMER0);
```

函数 timer_flag_get

函数timer_flag_get描述见下表:

表 3-1697. 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=0~7,14~16,22,23,30,31,40~44,50,51)	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=0~7,14~16,22,23,30,31,40~44,50,51)
TIMER_FLAG_CH0	通道0比较/捕获标志, TIMERx(x=0~4,7,14~16,22,23,30,31,40~44)
TIMER_FLAG_CH1	通道1比较/捕获标志, TIMERx(x=0~4,7,14,22,23,30,31,40~44)
TIMER_FLAG_CH2	通道2比较/捕获标志, TIMERx(x=0~4,7,22,23,30,31)
TIMER_FLAG_CH3	通道3比较/捕获标志, TIMERx(x=0~4,7,22,23,30,31)
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=0~4,7,14,22,23,30,31,40~44)
TIMER_FLAG_BRK0	BREAK标志, TIMERx(x=0,7,14~16,40~44)
TIMER_FLAG_BRK1	BREAK1标志, TIMERx(x=0,7)
TIMER_FLAG_CH0O	通道0捕获溢出标志, TIMERx(x=0~4,7,14~16,22,23,30,31,40~44)
TIMER_FLAG_CH1O	通道1捕获溢出标志, TIMERx(x=0~4,7,14,22,23,30,31,40~44)
TIMER_FLAG_CH2O	通道2捕获溢出标志, TIMERx(x=0~4,7,22,23,30,31)
TIMER_FLAG_CH3O	通道3捕获溢出标志, TIMERx(x=0~4,7,22,23,30,31)

0	
TIMER_FLAG_SYSB	系统源中止标志, TIMERx(x=0,7)
TIMER_FLAG_DECJ	正交编码器跳变标志, TIMERx(x=0~4,7,22,23,30,31)
TIMER_FLAG_DECDIS	正交编码器断线标志, TIMERx(x=0~4,7,22,23,30,31)
TIMER_FLAG_MCH0	多模式通道0比较/捕获标志, TIMERx(x=0,7,14~16,40~44)
TIMER_FLAG_MCH1	多模式通道1比较/捕获标志, TIMERx(x=0,7)
TIMER_FLAG_MCH2	多模式通道2比较/捕获标志, TIMERx(x=0,7)
TIMER_FLAG_MCH3	多模式通道3比较/捕获标志, TIMERx(x=0,7)
TIMER_FLAG_MCH00	多模式通道0捕获溢出标志, TIMERx(x=0,7,14~16,40~44)
TIMER_FLAG_MCH10	多模式通道1捕获溢出标志, TIMERx(x=0,7)
TIMER_FLAG_MCH20	多模式通道2捕获溢出标志, TIMERx(x=0,7)
TIMER_FLAG_MCH30	多模式通道3捕获溢出标志, TIMERx(x=0,7)
TIMER_FLAG_CH0COMADD	通道0附加比较标志, TIMERx(x=0~4,7,14,22,23,30,31)
TIMER_FLAG_CH1COMADD	通道1附加比较标志, TIMERx(x=0~4,7,14,22,23,30,31)
TIMER_FLAG_CH2COMADD	通道2附加比较标志, TIMERx(x=0~4,7,22,23,30,31)
TIMER_FLAG_CH3COMADD	通道3附加比较标志, TIMERx(x=0~4,7,22,23,30,31)
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 3-1698. 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=0~7,14~16,22,23,30,31,40~44,50,51)	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=0~7,14~16,22,23,30,31,40~44,50,51)
TIMER_FLAG_CH0	通道0比较/捕获标志, TIMERx(x=0~4,7,14~16,22,23,30,31,40~44)
TIMER_FLAG_CH1	通道1比较/捕获标志, TIMERx(x=0~4,7,14,22,23,30,31,40~44)
TIMER_FLAG_CH2	通道2比较/捕获标志, TIMERx(x=0~4,7,22,23,30,31)
TIMER_FLAG_CH3	通道3比较/捕获标志, TIMERx(x=0~4,7,22,23,30,31)
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=0~4,7,14,22,23,30,31,40~44)
TIMER_FLAG_BRK0	BREAK0标志, TIMERx(x=0,7,14~16,40~44)
TIMER_FLAG_BRK1	BREAK1标志, TIMERx(x=0,7)
TIMER_FLAG_CH0O	通道0捕获溢出标志, TIMERx(x=0~4,7,14~16,22,23,30,31,40~44)
TIMER_FLAG_CH1O	通道1捕获溢出标志, TIMERx(x=0~4,7,14,22,23,30,31,40~44)
TIMER_FLAG_CH2O	通道2捕获溢出标志, TIMERx(x=0~4,7,22,23,30,31)
TIMER_FLAG_CH3O	通道3捕获溢出标志, TIMERx(x=0~4,7,22,23,30,31)
TIMER_FLAG_SYSB	系统源中止标志, TIMERx(x=0,7)
TIMER_FLAG_DECJ	正交编码器跳变标志, TIMERx(x=0~4,7,22,23,30,31)
TIMER_FLAG_DECDIS	正交编码器断线标志, TIMERx(x=0~4,7,22,23,30,31)
TIMER_FLAG_MCH	多模式通道0比较/捕获标志, TIMERx(x=0,7,14~16,40~44)

0	
TIMER_FLAG_MCH1	多模式通道1比较/捕获标志, TIMERx(x=0,7)
TIMER_FLAG_MCH2	多模式通道2比较/捕获标志, TIMERx(x=0,7)
TIMER_FLAG_MCH3	多模式通道3比较/捕获标志, TIMERx(x=0,7)
TIMER_FLAG_MCH00	多模式通道0捕获溢出标志, TIMERx(x=0,7,14~16,40~44)
TIMER_FLAG_MCH10	多模式通道1捕获溢出标志, TIMERx(x=0,7)
TIMER_FLAG_MCH20	多模式通道2捕获溢出标志, TIMERx(x=0,7)
TIMER_FLAG_MCH30	多模式通道3捕获溢出标志, TIMERx(x=0,7)
TIMER_FLAG_CH0COMADD	通道0附加比较标志, TIMERx(x=0~4,7,14,22,23,30,31)
TIMER_FLAG_CH1COMADD	通道1附加比较标志, TIMERx(x=0~4,7,14,22,23,30,31)
TIMER_FLAG_CH2COMADD	通道2附加比较标志, TIMERx(x=0~4,7,22,23,30,31)
TIMER_FLAG_CH3COMADD	通道3附加比较标志, TIMERx(x=0~4,7,22,23,30,31)
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 3-1699. 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=0~7,14~16,22,23,30,31,40~44,50,51)	参考具体参数
输入参数{in}	
interrupt	中断源
TIMER_INT_UP	更新中断, TIMERx(x=0~7,14~16,22,23,30,31,40~44,50,51)
TIMER_INT_CH0	通道0比较/捕获中断, TIMERx(x=0~4,7,14~16,22,23,30,31,40~44)
TIMER_INT_CH1	通道1比较/捕获中断, TIMERx(x=0~4,7,14,22,23,30,31,40~44)
TIMER_INT_CH2	通道2比较/捕获中断, TIMERx(x=0~4,7,22,23,30,31)
TIMER_INT_CH3	通道3比较/捕获中断, TIMERx(x=0~4,7,22,23,30,31)
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=0~4,7,14,22,23,30,31,40~44)
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=0~4,7,22,23,30,31)
TIMER_INT_DECDIS	正交编码器断线中断, TIMERx(x=0~4,7,22,23,30,31)
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_CH0COMADD	通道0附加比较中断, TIMERx(x=0~4,7,14,22,23,30,31,40~44)
TIMER_INT_CH1COMADD	通道1附加比较中断, TIMERx(x=0~4,7,14,22,23,30,31,40~44)
TIMER_INT_CH2COMADD	通道2附加比较中断, TIMERx(x=0~4,7,22,23,30,31)
TIMER_INT_CH3COMADD	通道3附加比较中断, TIMERx(x=0~4,7,22,23,30,31)
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 3-1700. 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=0~7,14~16,22,23,30,31,40~44,50,51)	参考具体参数
输入参数{in}	
interrupt	中断源
TIMER_INT_UP	更新中断, TIMERx(x=0~7,14~16,22,23,30,31,40~44,50,51)
TIMER_INT_CH0	通道0比较/捕获中断, TIMERx(x=0~4,7,14~16,22,23,30,31,40~44)
TIMER_INT_CH1	通道1比较/捕获中断, TIMERx(x=0~4,7,14,22,23,30,31,40~44)
TIMER_INT_CH2	通道2比较/捕获中断, TIMERx(x=0~4,7,22,23,30,31)
TIMER_INT_CH3	通道3比较/捕获中断, TIMERx(x=0~4,7,22,23,30,31)
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=0~4,7,14,22,23,30,31,40~44)
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=0~4,7,22,23,30,31)
TIMER_INT_DECDIS	正交编码器断线中断, TIMERx(x=0~4,7,22,23,30,31)
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=0~4,7,14,22,23,30,31,40~44)
TIMER_INT_CH0C OMADD	通道1附加比较中断, TIMERx(x=0~4,7,14,22,23,30,31,40~44)
TIMER_INT_CH0C OMADD	通道2附加比较中断, TIMERx(x=0~4,7,22,23,30,31)
TIMER_INT_CH0C OMADD	通道3附加比较中断, TIMERx(x=0~4,7,22,23,30,31)
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 3-1701. 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
TIMERx(x=0~7,14~16,22,23,30,31,40~44,50,51)	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=0~7,14~16,22,23,30,31,40~44,50,51)
TIMER_INT_FLAG_CH0	通道0比较/捕获中断标志，TIMERx(x=0~4,7,14~16,22,23,30,31,40~44)
TIMER_INT_FLAG_CH1	通道1比较/捕获中断标志，TIMERx(x=0~4,7,14,22,23,30,31,40~44)
TIMER_INT_FLAG_CH2	通道2比较/捕获中断标志，TIMERx(x=0~4,7,22,23,30,31)
TIMER_INT_FLAG_CH3	通道3比较/捕获中断标志，TIMERx(x=0~4,7,22,23,30,31)
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=0~4,7,14,22,23,30,31,40~44)
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=0~4,7,22,23,30,31)
TIMER_INT_FLAG_	正交编码器断线中断标志，TIMERx(x=0~4,7,22,23,30,31)

DECDIS	
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)
TIMER_INT_FLAG_MCH1	多模式通道1比较/捕获中断标志, TIMERx(x=0,7)
TIMER_INT_FLAG_MCH2	多模式通道2比较/捕获中断标志, TIMERx(x=0,7)
TIMER_INT_FLAG_MCH3	多模式通道3比较/捕获中断标志, TIMERx(x=0,7)
TIMER_INT_FLAG_CH0COMADD	通道0附加比较中断标志, TIMERx(x=0~4,7,14,22,23,30,31)
TIMER_INT_FLAG_CH1COMADD	通道1附加比较中断标志, TIMERx(x=0~4,7,14,22,23,30,31)
TIMER_INT_FLAG_CH2COMADD	通道2附加比较中断标志, TIMERx(x=0~4,7,22,23,30,31)
TIMER_INT_FLAG_CH3COMADD	通道3附加比较中断标志, TIMERx(x=0~4,7,22,23,30,31)
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 3-1702. 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=0~7,14~16,22,23,30,31,40~44,50,51)	参考具体参数

输入参数{in}	
int_flag	中断源
<i>TIMER_INT_FLAG_UP</i>	更新中断标志, $TIMERx(x=0\sim7,14\sim16,22,23,30,31,40\sim44,50,51)$
<i>TIMER_INT_FLAG_CH0</i>	通道0比较/捕获中断标志, $TIMERx(x=0\sim4,7,14\sim16,22,23,30,31,40\sim44)$
<i>TIMER_INT_FLAG_CH1</i>	通道1比较/捕获中断标志, $TIMERx(x=0\sim4,7,14,22,23,30,31,40\sim44)$
<i>TIMER_INT_FLAG_CH2</i>	通道2比较/捕获中断标志, $TIMERx(x=0\sim4,7,22,23,30,31)$
<i>TIMER_INT_FLAG_CH3</i>	通道3比较/捕获中断标志, $TIMERx(x=0\sim4,7,22,23,30,31)$
<i>TIMER_INT_FLAG_CMT</i>	换相更新中断标志, $TIMERx(x=0,7,14\sim16,40\sim44)$
<i>TIMER_INT_FLAG_TRG</i>	触发中断标志, $TIMERx(x=0\sim4,7,14,22,23,30,31,40\sim44)$
<i>TIMER_INT_FLAG_BRK0</i>	BREAK0中断标志, $TIMERx(x=0,7,14\sim16,40\sim44)$
<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, $TIMERx(x=0\sim4,7,22,23,30,31)$
<i>TIMER_INT_FLAG_DECDIS</i>	正交编码器断线中断标志, $TIMERx(x=0\sim4,7,22,23,30,31)$
<i>TIMER_INT_FLAG_MCH0</i>	多模式通道0比较/捕获中断标志, $TIMERx(x=0,7,14\sim16,40\sim44)$
<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, $TIMERx(x=0\sim4,7,14,22,23,30,31)$
<i>TIMER_INT_FLAG_CH1COMADD</i>	通道1附加比较中断标志, $TIMERx(x=0\sim4,7,14,22,23,30,31)$
<i>TIMER_INT_FLAG_CH2COMADD</i>	通道2附加比较中断标志, $TIMERx(x=0\sim4,7,22,23,30,31)$
<i>TIMER_INT_FLAG_CH3COMADD</i>	通道3附加比较中断标志, $TIMERx(x=0\sim4,7,22,23,30,31)$
输出参数{out}	

-	-
返回值	
-	-

例如：

```
/* clear TIMER0 update interrupt flag */
```

```
timer_interrupt_flag_clear(TIMER0, TIMER_INT_FLAG_UP);
```

3.44. TLI

TLI(TFT-LCD 接口)连接同步的 LCD 接口，并且为无源 LCD 显示屏提供提供像素数据，时钟以及时序信号。TLI 寄存器列举在章节 [3.44.1](#)，TLI 固件库函数介绍在章节 [3.44.2](#)。

3.44.1. 外设寄存器说明

TLI 寄存器列表如下表所示：

表 3-1703. TLI 寄存器

寄存器名称	寄存器描述
TLI_SPSZ	TLI 同步脉冲宽度寄存器
TLI_BPSZ	TLI 后沿宽度寄存器
TLI_ASZ	TLI 有效宽度寄存器
TLI_TSZ	TLI 总宽度寄存器
TLI_CTL	TLI 控制寄存器
TLI_RL	TLI 重载配置寄存器
TLI_BGC	TLI 背景色配置寄存器
TLI_INTEN	TLI 中断使能寄存器
TLI_INTF	TLI 中断标志寄存器
TLI_INTC	TLI 中断标志清除寄存器
TLI_LM	TLI 行标记寄存器
TLI_CPPOS	TLI 当前像素寄存器
TLI_STAT	TLI 状态寄存器
TLI_LxCTL	TLI x 层控制寄存器
TLI_LxHPOS	TLI x 层水平位置参数寄存器
TLI_LxVPOS	TLI x 层垂直位置参数寄存器
TLI_LxCKEY	TLI x 层色键值寄存器
TLI_LxPPF	TLI x 层像素格式寄存器
TLI_LxSA	TLI x 层特定 alpha 寄存器
TLI_LxDC	TLI x 层默认颜色寄存器
TLI_LxBLEND	TLI x 层混合寄存器
TLI_LxFBADDR	TLI x 层帧基地址寄存器
TLI_LxFLLN	TLI x 层行数寄存器
TLI_LxFTLN	TLI x 层帧总行数寄存器
TLI_LxLUT	TLI x 层颜色查找表寄存器

3.44.2. 外设库函数说明

TLI 库函数列表如下表所示：

表 3-1704. TLI 库函数

库函数名称	库函数描述
tli_deinit	复位 TLI
tli_struct_para_init	用默认值初始化 TLI 参数结构体，建议在定义一个 tli_parameter_struct 结构体后调用该接口实现对结构体的初始化
tli_init	初始化 TLI
tli_dither_config	配置 TLI 抖动功能
tli_enable	使能 TLI
tli_disable	禁能 TLI
tli_reload_config	配置 TLI 重载模式
tli_layer_struct_para_init	用默认参数初始化 TLI 层结构体，建议在定义一个 tli_layer_parameter_struct 结构体后调用该接口实现对结构体的初始化
tli_layer_init	初始化 TLI 层
tli_layer_window_offset_modify	重新配置窗口位置
tli_lut_struct_para_init	用默认参数初始化 TLI 层 LUT 结构体，建议在定义一个 tli_layer_lut_parameter_struct 结构体后调用该接口实现对结构体的初始化
tli_lut_init	初始化 TLI 层颜色查表
tli_color_key_init	初始化 TLI 层色键
tli_layer_enable	使能 TLI 层
tli_layer_disable	禁能 TLI 层
tli_color_key_enable	使能 TLI 层色键
tli_color_key_disable	禁能 TLI 层色键
tli_lut_enable	使能 TLI 层颜色查找
tli_lut_disable	禁能 TLI 层颜色查找
tli_line_mark_set	设置行标记值
tli_current_pos_get	获取当前像素位置
tli_interrupt_enable	使能 TLI 中断
tli_interrupt_disable	禁能 TLI 中断
tli_interrupt_flag_get	获取 TLI 中断标志
tli_interrupt_flag_clear	清除 TLI 中断标志
tli_flag_get	从 TLI_INTF 寄存器或 TLI_STAT 寄存器获取 TLI 标志或状态

结构体 tli_parameter_struct

表 3-1705. 结构体 tli_parameter_struct

成员名称	功能描述
synpsz_vpsz	垂直同步脉冲宽度
synpsz_hpsz	水平同步脉冲宽度
backpsz_vbpsz	垂直后沿加同步脉冲的宽度
backpsz_hbpsz	水平后沿加同步脉冲的宽度

activesz_vasz	垂直有效宽度加后沿像素和水平同步像素宽度
activesz_hasz	水平有效宽度加后沿像素和垂直同步像素宽度
totalsz_vtsz	显示器的垂直总宽度
totalsz_htsz	显示器的水平总宽度
backcolor_red	背景色红色值
backcolor_green	背景色绿色值
backcolor_blue	背景色蓝色值
signalpolarity_hs	水平脉冲极性选择
signalpolarity_vs	垂直脉冲极性选择
signalpolarity_de	数据使能极性选择
signalpolarity_pixelck	像素时钟极性选择

结构体 tli_layer_parameter_struct

表 3-1706. 结构体 tli_layer_parameter_struct

成员名称	功能描述
layer_window_rightpos	窗口右边位置
layer_window_leftpos	窗口左边位置
layer_window_bottompos	窗口下边位置
layer_window_toppos	窗口上边位置
layer_ppf	包像素格式
layer_sa	特定 alpha
layer_default_alpha	默认颜色 alpha
layer_default_red	默认红色值
layer_default_green	默认绿色值
layer_default_blue	默认蓝色值
layer_acf1	Alpha 混合计算因子 1
layer_acf2	Alpha 混合计算因子 2
layer_frame_bufaddr	帧缓存区起始地址
layer_frame_buf_stride_offset	帧缓存区步幅偏移
layer_frame_line_length	帧行长度
layer_frame_total_line_number	帧总行数

结构体 tli_layer_lut_parameter_struct

表 3-1707. 结构体 tli_layer_lut_parameter_struct

成员名称	功能描述
layer_table_addr	查表写地址
layer_lut_channel_red	颜色查找表条目红色通道
layer_lut_channel_green	颜色查找表条目绿色通道
layer_lut_channel_blue	颜色查找表条目蓝色通道

函数 tli_deinit

函数tli_deinit描述见下表：

表 3-1708. 函数 tli_deinit

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

例如：

```
/* deinitialize TLI registers */
```

```
tli_deinit ();
```

函数 tli_struct_para_init

函数tli_struct_para_init描述见下表：

表 3-1709. 函数 tli_struct_para_init

函数名称	tli_struct_para_init
函数原形	void tli_struct_para_init(tli_parameter_struct *tli_struct);
功能描述	用默认值初始化 TLI 参数结构体，建议在定义一个 tli_parameter_struct 结构体后调用该接口实现对结构体的初始化
先决条件	-
被调用函数	-
输入参数{in}	
-	-
输出参数{out}	
*tli_struct	指向 TLI 参数结构体的指针
返回值	
-	-

例如：

```
tli_parameter_struct tli_init_struct;
```

```
/* initialize the parameters of TLI parameter structure with the default values */
```

```
tli_struct_para_init(&tli_init_struct);
```

函数 tli_init

函数tli_init描述见下表:

表 3-1710. 函数 tli_init

函数名称	tli_init
函数原形	void tli_init(tli_parameter_struct *tli_struct);
功能描述	初始化 TLI
先决条件	-
被调用函数	-
输入参数{in}	
*tli_struct	指向 TLI 参数结构体的指针
输出参数{out}	
-	-
返回值	
-	-

例如:

```
tli_parameter_struct  tli_init_struct;

/* initialize the parameters of TLI parameter structure with the default values */
tli_struct_para_init(&tli_init_struct);

/* configure TLI parameter struct */
tli_init_struct.signalpolarity_hs = TLI_HSYN_ACTLIVE_LOW;
tli_init_struct.signalpolarity_vs = TLI_VSYN_ACTLIVE_LOW;
tli_init_struct.signalpolarity_de = TLI_DE_ACTLIVE_LOW;
tli_init_struct.signalpolarity_pixelclk = TLI_PIXEL_CLOCK_TLI;

/* LCD display timing configuration */
tli_init_struct.synpsz_hpsz = 40;
tli_init_struct.synpsz_vpsz = 9;
tli_init_struct.backpsz_hbpsz = 42;
tli_init_struct.backpsz_vbpsz = 11;
tli_init_struct.activesz_hasz = 522;
tli_init_struct.activesz_vasz = 283;
tli_init_struct.totalsz_htsz = 524;
tli_init_struct.totalsz_vtsz = 285;
```

```
/* configure LCD background R,G,B values */
```

```
tli_init_struct.backcolor_red = 0xFF;
```

```
tli_init_struct.backcolor_green = 0xFF;
```

```
tli_init_struct.backcolor_blue = 0xFF;
```

```
tli_init(&tli_init_struct);
```

函数 tli_dither_config

函数tli_dither_config描述见下表:

表 3-1711. 函数 tli_dither_config

函数名称	tli_dither_config
函数原形	void tli_dither_config(uint8_t ditherstat);
功能描述	配置 TLI 抖动功能
先决条件	-
被调用函数	-
输入参数{in}	
ditherstat	抖动状态
TLI_DITHER_ENABLE	使能 TLI 抖动
TLI_DITHER_DISABLE	禁能 TLI 抖动
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* enable TLI dither function */
```

```
tli_dither_config(TLI_DITHER_ENABLE);
```

函数 tli_enable

函数tli_enable描述见下表:

表 3-1712. 函数 tli_enable

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

-	-
返回值	
-	-

例如：

```
/* enable TLI */
```

```
tli_enable( );
```

函数 tli_disable

函数tli_disable描述见下表：

表 3-1713. 函数 tli_disable

函数名称	tli_disable
函数原形	void tli_disable(void);
功能描述	禁能 TLI
先决条件	-
被调用函数	-
输入参数{in}	
-	-
Output parameter{out}	
-	-
返回值	
-	-

例如：

```
/* disable TLI */
```

```
tli_disable( );
```

函数 tli_reload_config

函数tli_reload_config描述见下表：

表 3-1714. 函数 tli_reload_config

函数名称	tli_reload_config
函数原形	void tli_reload_config(uint8_t reloadmode);
功能描述	configure TLI reload mode
先决条件	-
被调用函数	-
输入参数{in}	
reloadmode	Reload mode
TLI_FRAME_BLANK_RELOAD_EN	层配置在帧间隔被重载

<code>TLI_REQUEST_RELO AD_EN</code>	层配置在置位后被重载
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* configure TLI reload mode */
```

```
tli_reload_config (TLI_FRAME_BLANK_RELOAD_EN);
```

函数 `tli_layer_struct_para_init`

函数 `tli_layer_struct_para_init` 描述见下表：

表 3-1715. 函数 `tli_layer_struct_para_init`

函数名称	<code>tli_layer_struct_para_init</code>
函数原形	<code>void tli_layer_struct_para_init(tli_layer_parameter_struct *layer_struct);</code>
功能描述	用默认参数初始化 TLI 层结构体，建议在定义一个 <code>tli_layer_parameter_struct</code> 结构体后调用该接口实现对结构体的初始化
先决条件	-
被调用函数	-
输入参数{in}	
-	-
输出参数{out}	
<code>*layer_struct</code>	指向 <code>tli_layer_parameter_struct</code> 结构体的指针
返回值	
-	-

例如：

```
tli_layer_parameter_struct tli_layer_init_struct;
```

```
/* initialize the parameters of TLI layer structure with the default values */
```

```
tli_layer_struct_para_init(&tli_layer_init_struct);
```

函数 `tli_layer_init`

函数 `tli_layer_init` 描述见下表：

表 3-1716. 函数 `tli_layer_init`

函数名称	<code>tli_layer_init</code>
函数原形	<code>void tli_layer_init(uint32_t layerx, tli_layer_parameter_struct *layer_struct)</code>
功能描述	初始化 TLI 层
先决条件	-

被调用函数	-
输入参数{in}	
layerx	层基地址
<i>LAYER0</i>	0 层基地址
<i>LAYER1</i>	1 层基地址
输入参数{in}	
*layer_struct	指向 TLI 层参数结构体的指针
输出参数{out}	
-	-
返回值	
-	-

例如：

```

tli_layer_parameter_struct tli_layer_init_struct;

tli_layer_struct_para_init(&tli_layer_init_struct);

/* TLI layer0 configuration */

tli_layer_init_struct.layer_window_leftpos = 20 + 43;

tli_layer_init_struct.layer_window_rightpos = (20 + 440 + 43 - 1);

tli_layer_init_struct.layer_window_toppos = 40 + 12;

tli_layer_init_struct.layer_window_bottompos = (40 + 182 + 12 - 1);

tli_layer_init_struct.layer_ppf = LAYER_PPF_RGB565;

/* TLI window specified alpha configuration */

tli_layer_init_struct.layer_sa = 255;

/* TLI layer default alpha R,G,B value configuration */

tli_layer_init_struct.layer_default_blue = 0xFF;

tli_layer_init_struct.layer_default_green = 0xFF;

tli_layer_init_struct.layer_default_red = 0xFF;

tli_layer_init_struct.layer_default_alpha = 0xFF;

/* TLI window blend configuration */

tli_layer_init_struct.layer_acf1 = LAYER_ACF1_PASA;

tli_layer_init_struct.layer_acf2 = LAYER_ACF2_PASA;

/* TLI layer frame buffer base address configuration */

tli_layer_init_struct.layer_frame_bufaddr = (uint32_t)&gBackground;

```

```
tli_layer_init_struct.layer_frame_line_length = ((440 * 2) + 3);
```

```
tli_layer_init_struct.layer_frame_buf_stride_offset = (440 * 2);
```

```
tli_layer_init_struct.layer_frame_total_line_number = 182;
```

```
tli_layer_init(LAYER0, &tli_layer_init_struct);
```

函数 tli_layer_window_offset_modify

函数tli_layer_window_offset_modify描述见下表：

表 3-1717. 函数 tli_layer_window_offset_modify

函数名称	tli_layer_window_offset_modify
函数原形	void tli_layer_window_offset_modify(uint32_t layerx,uint16_t offset_x,uint16_t offset_y);
功能描述	重新配置窗口位置
先决条件	-
被调用函数	-
输入参数{in}	
layerx	层基地址
LAYER0	0 层基地址
LAYER1	1 层基地址
输入参数{in}	
offset_x	新水平偏移
输入参数{in}	
offset_y	新垂直偏移
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* reconfigure LAYER1 window position */
```

```
tli_layer_window_offset_modify(LAYER1, 20, 20);
```

函数 tli_lut_struct_para_init

函数tli_lut_struct_para_init描述见下表：

表 3-1718. 函数 tli_lut_struct_para_init

函数名称	tli_lut_struct_para_init
函数原形	void tli_lut_struct_para_init(tli_layer_lut_parameter_struct *lut_struct);
功能描述	用默认参数初始化 TLI 层 LUT 结构体，建议在定义一个 tli_layer_lut_parameter_struct 结构体后调用该接口实现对结构体的初始化

先决条件	-
被调用函数	-
输入参数{in}	
-	-
输出参数{out}	
*lut_struct	指向 tli_layer_lut_parameter_struct 结构体的指针
返回值	
-	-

例如：

```
tli_layer_lut_parameter_struct tli_lut_struct;
```

```
/* initialize the parameters of TLI layer LUT structure with the default values */
```

```
tli_lut_struct_para_init(&tli_lut_struct);
```

函数 tli_lut_init

函数tli_lut_init描述见下表：

表 3-1719. 函数 tli_lut_init

函数名称	tli_lut_init
函数原形	void tli_lut_init(uint32_t layerx,tli_layer_lut_parameter_struct *lut_struct)
功能描述	初始化 TLI 层颜色查表
先决条件	-
被调用函数	-
输入参数{in}	
layerx	层基地址
LAYER0	0 层基地址
LAYER1	1 层基地址
输入参数{in}	
*lut_struct	指向 TLI 层 LUT 参数结构体的指针
输出参数{out}	
-	-
返回值	
-	-

例如：

```
tli_layer_lut_parameter_struct tli_lut_struct;
```

```
tli_lut_struct_para_init(&tli_lut_struct);
```

```
/* initialize TLI layer0 LUT */
```

```
tli_lut_struct.layer_table_addr = 0x20003000;
```

```
tli_lut_struct.layer_lut_channel_red = 0x20;
```



```
tli_lut_struct.layer_lut_channel_green = 0x30;
```

```
tli_lut_struct.layer_lut_channel_blue = 0xFF;
```

```
tli_lut_init(LAYER0, &tli_lut_struct);
```

函数 tli_color_key_init

函数tli_color_key_init描述见下表：

表 3-1720. 函数 tli_color_key_init

函数名称	tli_color_key_init
函数原形	void tli_color_key_init(uint32_t layerx,uint8_t redkey,uint8_t greenkey,uint8_t bluekey);
功能描述	初始化 TLI 层色键
先决条件	-
被调用函数	-
输入参数{in}	
layerx	层基地址
LAYER0	0 层基地址
LAYER1	1 层基地址
输入参数{in}	
redkey	红色键
输入参数{in}	
greenkey	绿色键
输入参数{in}	
bluekey	蓝色键
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* initialize TLI layer0 color key */
```

```
tli_color_key_init(LAYER0, 0xAA, 0xFF, 0x00);
```

函数 tli_layer_enable

函数tli_layer_enable描述见下表：

表 3-1721. 函数 tli_layer_enable

函数名称	tli_layer_enable
函数原形	void tli_layer_enable(uint32_t layerx);
功能描述	使能 TLI 层
先决条件	-

被调用函数	-
输入参数{in}	
layerx	层基地址
LAYER0	0 层基地址
LAYER1	1 层基地址
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* TLI layer enable */
```

```
tli_layer_enable(LAYER0);
```

函数 tli_layer_disable

函数tli_layer_disable描述见下表：

表 3-1722. 函数 tli_layer_disable

函数名称	tli_layer_disable
函数原形	void tli_layer_disable(uint32_t layerx);
功能描述	禁能 TLI 层
先决条件	-
被调用函数	-
输入参数{in}	
layerx	层基地址
LAYER0	0 层基地址
LAYER1	1 层基地址
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* TLI layer disable */
```

```
tli_layer_disable(LAYER0);
```

函数 tli_color_key_enable

函数tli_color_key_enable描述见下表：

表 3-1723. 函数 tli_color_key_enable

函数名称	tli_color_key_enable
------	----------------------

函数原形	void tli_color_key_enable(uint32_t layerx);
功能描述	使能 TLI 层色键
先决条件	-
被调用函数	-
输入参数{in}	
layerx	层基地址
LAYER0	0 层基地址
LAYER1	1 层基地址
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* enable TLI layer0 color keying */
```

```
tli_color_key_enable(LAYER0);
```

函数 tli_color_key_disable

函数tli_color_key_disable描述见下表：

表 3-1724. 函数 tli_color_key_disable

函数名称	tli_color_key_disable
函数原形	void tli_color_key_disable(uint32_t layerx);
功能描述	禁能 TLI 层色键
先决条件	-
被调用函数	-
输入参数{in}	
layerx	层基地址
LAYER0	0 层基地址
LAYER1	1 层基地址
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* disable TLI layer0 color keying */
```

```
tli_color_key_disable(LAYER0);
```

函数 tli_lut_enable

函数tli_lut_enable描述见下表：

表 3-1725. 函数 tli_lut_enable

函数名称	tli_lut_enable
函数原形	void tli_lut_enable(uint32_t layerx);
功能描述	使能 TLI 层颜色查找
先决条件	-
被调用函数	-
输入参数{in}	
layerx	层基地址
LAYER0	0 层基地址
LAYER1	1 层基地址
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* enable TLI layer0 LUT */
```

```
tli_lut_enable(LAYER0);
```

函数 tli_lut_disable

函数tli_lut_disable描述见下表:

表 3-1726. 函数 tli_lut_disable

函数名称	tli_lut_disable
函数原形	void tli_lut_disable(uint32_t layerx);
功能描述	禁能 TLI 层颜色查找
先决条件	-
被调用函数	-
输入参数{in}	
layerx	层基地址
LAYER0	0 层基地址
LAYER1	1 层基地址
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* disable TLI layer0 LUT */
```

```
tli_lut_disable(LAYER0);
```

函数 tli_line_mark_set

函数tli_line_mark_set描述见下表：

表 3-1727. 函数 tli_line_mark_set

函数名称	tli_line_mark_set
函数原形	void tli_line_mark_set(uint16_t line_num);
功能描述	设置行标记值
先决条件	-
被调用函数	-
输入参数{in}	
line_num	行编号
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* set line mark value */
```

```
tli_line_mark_set(0x20);
```

函数 tli_current_pos_get

函数tli_current_pos_get描述见下表：

表 3-1728. 函数 tli_current_pos_get

函数名称	tli_current_pos_get
函数原形	uint32_t tli_current_pos_get(void);
功能描述	获取当前像素位置
先决条件	-
被调用函数	-
输入参数{in}	
-	-
输出参数{out}	
-	-
返回值	
uint32_t	0x0 – 0xFFFFFFFF

例如：

```
uint32_t pos;
```

```
/* get current pixel position */
```

```
pos = tli_current_pos_get();
```

函数 **tli_interrupt_enable**

函数tli_interrupt_enable描述见下表：

表 3-1729. 函数 tli_interrupt_enable

函数名称	tli_interrupt_enable
函数原形	void tli_interrupt_enable(uint32_t int_flag);
功能描述	使能 TLI 中断
先决条件	-
被调用函数	-
输入参数{in}	
int_flag	TLI 中断标志
TLI_INT_LM	行标记中断
TLI_INT_FE	FIFO 错误中断
TLI_INT_TE	事务错误中断
TLI_INT_LCR	层配置重载中断
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* enable TLI line mark interrupt */
tli_interrupt_enable(TLI_INT_LM);
```

函数 **tli_interrupt_disable**

函数tli_interrupt_disable描述见下表：

表 3-1730. 函数 tli_interrupt_disable

函数名称	tli_interrupt_disable
函数原形	void tli_interrupt_disable(uint32_t int_flag);
功能描述	禁能 TLI 中断
先决条件	-
被调用函数	-
输入参数{in}	
int_flag	TLI 中断标志
TLI_INT_LM	行标记中断
TLI_INT_FE	FIFO 错误中断
TLI_INT_TE	事务错误中断
TLI_INT_LCR	层配置重载中断
输出参数{out}	
-	-

返回值	
-	-

例如：

```
/* disable TLI line mark interrupt */
tli_interrupt_disable(TLI_INT_LM);
```

函数 tli_interrupt_flag_get

函数tli_interrupt_flag_get描述见下表：

表 3-1731. 函数 tli_interrupt_flag_get

函数名称	tli_interrupt_flag_get
函数原形	FlagStatus tli_interrupt_flag_get(uint32_t int_flag);
功能描述	获取 TLI 中断标志
先决条件	-
被调用函数	-
输入参数{in}	
int_flag	TLI 中断标志
TLI_INT_FLAG_LM	行标记中断
TLI_INT_FLAG_FE	FIFO 错误中断
TLI_INT_FLAG_TE	事务错误中断
TLI_INT_FLAG_LCR	层配置重载中断
输出参数{out}	
-	-
返回值	
FlagStatus	SET 或 RESET

例如：

```
/* get TLI interrupt flag */
if(SET == tli_interrupt_flag_get(TLI_INT_FLAG_LM)){
    tli_interrupt_flag_clear(TLI_INT_FLAG_LM);
}
```

函数 tli_interrupt_flag_clear

函数tli_interrupt_flag_clear描述见下表：

表 3-1732. 函数 tli_interrupt_flag_clear

函数名称	tli_interrupt_flag_clear
函数原形	void tli_interrupt_flag_clear(uint32_t int_flag)
功能描述	清除 TLI 中断标志

先决条件	-
被调用函数	-
输入参数{in}	
int_flag	TLI 中断标志
<i>TLI_INT_FLAG_LM</i>	行标记中断
<i>TLI_INT_FLAG_FE</i>	FIFO 错误中断
<i>TLI_INT_FLAG_TE</i>	事务错误中断
<i>TLI_INT_FLAG_LCR</i>	层配置重载中断
输出参数{out}	
-	-
返回值	
-	-

例如：

```
if(SET == tli_interrupt_flag_get(TLI_INT_FLAG_LM)){
    /* clear TLI interrupt flag */
    tli_interrupt_flag_clear(TLI_INT_FLAG_LM);
}
```

函数 tli_flag_get

函数tli_flag_get描述见下表：

表 3-1733. 函数 tli_flag_get

函数名称	tli_flag_get
函数原形	FlagStatus tli_flag_get(uint32_t flag);
功能描述	从 TLI_INTF 寄存器或 TLI_STAT 寄存器获取 TLI 标志或状态
先决条件	-
被调用函数	-
输入参数{in}	
flag	TLI flags
<i>TLI_FLAG_VDE</i>	当前 VDE 状态
<i>TLI_FLAG_HDE</i>	当前 HDE 状态
<i>TLI_FLAG_VS</i>	当前 vs 状态
<i>TLI_FLAG_HS</i>	当前 hs 状态
<i>TLI_FLAG_LM</i>	行标记中断标志
<i>TLI_FLAG_FE</i>	FIFO 错误中断标志
<i>TLI_FLAG_TE</i>	事务错误中断标志
<i>TLI_FLAG_LCR</i>	层配置重载中断标志
输出参数{out}	
-	-
返回值	

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

例如:

```
/* wait the TLI_FLAG_LM flag set */
while(RESET == tli_flag_get(TLI_FLAG_LM)){
}
```

3.45. 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[3.45.1](#)描述了TMU的寄存器列表，章节[3.45.2](#)对TMU库函数进行说明。

3.45.1. 外设寄存器说明

TMU寄存器列表如下表所示:

表 3-1734. 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

3.45.2. 外设库函数说明

TMU库函数列表如下表所示:

表 3-1735. 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

函数名称	功能描述
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 3-1736. 结构体 tmu_parameter_struct

成员名称	功能描述
mode	TMU运行模式 (TMU_MODE_COS,TMU_MODE_SIN,TMU_MODE_ATAN2,TMU_MODE_MODULEUS,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 3-1737. 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 3-1738. 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 3-1739. 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 3-1740. 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 3-1741. 函数 `tmu_read_interrupt_disable`

函数名称	<code>tmu_read_interrupt_disable</code>
函数原型	<code>void tmu_read_interrupt_disable(void);</code>
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 3-1742. 函数 `tmu_dma_read_enable`

函数名称	<code>tmu_dma_read_enable</code>
函数原型	<code>void tmu_dma_read_enable(void);</code>
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 3-1743. 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 3-1744. 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 3-1745. 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 3-1746. 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 3-1747. 函数 `tmu_two_q31_write`

函数名称	<code>tmu_two_q31_write</code>
函数原型	<code>void tmu_two_q31_write(uint32_t data1, uint32_t data2);</code>
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 3-1748. 函数 `tmu_two_q15_write`

函数名称	<code>tmu_two_q15_write</code>
函数原型	<code>void tmu_two_q15_write(uint16_t data1, uint16_t data2);</code>
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 3-1749. 函数 `tmu_one_q31_read`

函数名称	<code>tmu_one_q31_read</code>
函数原型	<code>void tmu_one_q31_read(uint32_t* p);</code>
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 3-1750. 函数 `tmu_two_q31_read`

函数名称	<code>tmu_two_q31_read</code>
函数原型	<code>void tmu_two_q31_read(uint32_t* p1, uint32_t* p2);</code>
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 3-1751. 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);
```

3.46. TRGSEL

TRGSEL 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 [3.46.1](#) 描述了 TRGSEL 的寄存器列表，章节 [3.46.2](#) 对 TRGSEL 库函数进行说明。

3.46.1. 外设寄存器说明

TRGSEL寄存器列表如下表所示：

表 3-1752. TRGSEL 寄存器

寄存器名称	寄存器描述
TRGSEL_EXTOUT0	EXTOUT0触发选择寄存器
TRGSEL_EXTOUT1	EXTOUT1触发选择寄存器
TRGSEL_EXTOUT2	EXTOUT2触发选择寄存器

寄存器名称	寄存器描述
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
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_TIMER30ETI	TIMER30_ETI触发选择寄存器
TRIGSEL_TIMER31ETI	TIMER31_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_ITI14register
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

寄存器名称	寄存器描述
TRIGSEL_TIMER30ITI14	TIMER30_ITI14触发选择寄存器
TRIGSEL_TIMER31ITI14	TIMER31_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

3.46.2. 外设库函数说明

TRGSEL库函数列表如下表所示：

表 3-1753. 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 3-1754. 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

成员名称	功能描述
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
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

成员名称	功能描述
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
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_TIMER30_TRGO0	触发输入源为TIMER30 TRGO0
TRIGSEL_INPUT_TIMER30_CH0	触发输入源为TIMER30 CH0
TRIGSEL_INPUT_TIMER30_CH1	触发输入源为TIMER30 CH1
TRIGSEL_INPUT_TIMER30_CH2	触发输入源为TIMER30 CH2
TRIGSEL_INPUT_TIMER30_CH3	触发输入源为TIMER30 CH3
TRIGSEL_INPUT_TIMER30_ETI	触发输入源为TIMER30 ETI

成员名称	功能描述
TRIGSEL_INPUT_TIMER31_TRGO0	触发输入源为TIMER31 TRGO0
TRIGSEL_INPUT_TIMER31_CH0	触发输入源为TIMER31 CH0
TRIGSEL_INPUT_TIMER31_CH1	触发输入源为TIMER31 CH1
TRIGSEL_INPUT_TIMER31_CH2	触发输入源为TIMER31 CH2
TRIGSEL_INPUT_TIMER31_CH3	触发输入源为TIMER31 CH3
TRIGSEL_INPUT_TIMER31_ETI	触发输入源为TIMER31 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
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

成员名称	功能描述
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 3-1755. 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
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

成员名称	功能描述
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_LPDS_TRG	输出到目标外设LPDS_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_TIMER30_ETI	输出到目标外设TIMER30_ETI
TRIGSEL_OUTPUT_TIMER31_ETI	输出到目标外设TIMER31_ETI
TRIGSEL_OUTPUT_EDOUT_TRG	输出到目标外设EDOUT_TRG
TRIGSEL_OUTPUT_HPDI_IIR	输出到目标外设HPDI_IIR
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
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_TIMER30_ITI14	输出到目标外设TIMER30_ITI14
TRIGSEL_OUTPUT_TIMER31_ITI14	输出到目标外设TIMER31_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 3-1756. 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 3-1757. 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 3-1755. 枚举类型 trigsel_periph_enum
输入参数{in}	
trigger_source	触发源，参考 表 3-1754. 枚举类型 trigsel_source_enum
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* select TIMER0_CH2 to trigger ADC0 */
```

```
trigsel_init(TRIGSEL_OUTPUT_ADC0_REGTRG, TRIGSEL_INPUT_TIMER0_CH2);
```

函数 trigsel_trigger_source_get

函数trigsel_trigger_source_get描述见下表：

表 3-1758. 函数 `trigsel_trigger_source_get`

函数名称	<code>trigsel_trigger_source_get</code>
函数原型	<code>trigsel_source_enum trigsel_trigger_source_get(trigsel_periph_enum target_periph);</code>
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 3-1755. 枚举类型 <code>trigsel_periph_enum</code>
输出参数{out}	
-	-
返回值	
trigger_source	触发源, 参考 表 3-1754. 枚举类型 <code>trigsel_source_enum</code>

例如:

```
/* get the trigger input signal for ADC0 */
trigsel_source_enum input_signal;
input_signal = trigsel_trigger_source_get(TRIGSEL_OUTPUT_ADC0_REGTRG);
```

函数 `trigsel_register_lock_set`

函数 `trigsel_register_lock_set` 描述见下表:

表 3-1759. 函数 `trigsel_trigger_source_set`

函数名称	<code>trigsel_register_lock_set</code>
函数原型	<code>void trigsel_register_lock_set(trigsel_periph_enum target_periph);</code>
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 3-1755. 枚举类型 <code>trigsel_periph_enum</code>
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 3-1760. 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 3-1755. 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);
```

3.47. 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[3.47.1](#), TRNG固件库函数介绍在章节[3.47.2](#)。

3.47.1. 外设寄存器说明

TRNG寄存器列表如下表所示:

表 3-1761. 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

3.47.2. 外设库函数说明

TRNG库函数列表如下表所示:

表 3-1762. 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 3-1763. 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 3-1764. 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 3-1765. 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 3-1766. 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 3-1767. 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 3-1768. 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 3-1769. 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 3-1770. 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 3-1771. 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 3-1772. 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3-1765. 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 3-1773. 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 3-1774. 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 3-1775. 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 3-1776. 函数 trng_conditioning_disable

函数名称	trng_conditioning_disable
函数原形	void trng_conditioning_disable(void);
功能描述	失能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_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 3-1777. 函数 trng_conditioning_input_bitwidth

函数名称	trng_conditioning_input_bitwidth
函数原形	void trng_conditioning_input_bitwidth(trng_inmod_enum input_bitwidth);
功能描述	配置训练单元输入位宽
先决条件	trng_conditioning_reset_enable
被调用函数	-
输入参数{in}	
input_bitwidth	输入位宽，参考 表3-1763. 枚举trng_inmod_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_conditioning_output_bitwidth

函数trng_conditioning_output_bitwidth描述见下表：

表 3-1778. 函数 trng_conditioning_output_bitwidth

函数名称	trng_conditioning_output_bitwidth
函数原形	void trng_conditioning_output_bitwidth(trng_outmod_enum output_bitwidth);
功能描述	配置训练单元输出位宽
先决条件	trng_conditioning_reset_enable
被调用函数	-
输入参数{in}	
output_bitwidth	输出位宽，参考 表3-1764. 枚举trng_outmod_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_replace_test_enable

函数trng_replace_test_enable描述见下表：

表 3-1779. 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 3-1780. 函数 trng_replace_test_disable

函数名称	trng_replace_test_disable
函数原形	void trng_replace_test_disable(void);
功能描述	失能替换测试
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:

```
/* 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 3-1781. 函数 trng_hash_init_enable

函数名称	trng_hash_init_enable
函数原形	void trng_hash_init_enable(void);
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	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:

```
/* 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 3-1782. 函数 `trng_hash_init_disable`

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

```
/* 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 3-1783. 函数 `trng_powermode_config`

函数名称	<code>trng_powermode_config</code>
函数原形	<code>void trng_powermode_config(uint32_t powermode);</code>
功能描述	配置功耗模式
先决条件	<code>trng_conditioning_reset_enable</code>
被调用函数	-
输入参数{in}	
powermode	功耗模式选项
<code>TRNG_NR_ULATRL0W</code>	极低功耗模式
<code>TRNG_NR_LOW</code>	低功耗模式
<code>TRNG_NR_MEDIUM</code>	中等功耗模式
<code>TRNG_NR_HIGH</code>	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 3-1784. 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 3-1785. 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 3-1786. 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 3-1787. 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 3-1788. 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 3-1789. 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 3-1790. 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 3-1791. 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 3-1792. 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3-1766. 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 3-1793. 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 3-1794. 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 3-1795. 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3-1767. 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 3-1796. 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3-1767. 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);
```

3.48. 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[3.48.1](#)描述了USART的寄存器列表，章节[3.48.2](#)对USART库函数进行说明。

3.48.1. 外设寄存器说明

USART寄存器列表如下表所示：

表 3-1797. 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

3.48.2. 外设库函数说明

USART库函数列表如下表所示：

表 3-1798. 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 3-1799. 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 3-1800. 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 3-1801. 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 3-1802. 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 3-1803. 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 3-1804. 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 3-1805. 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 3-1806. 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 3-1807. 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 3-1808. 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 3-1809. 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 3-1810. 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 3-1811. 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 3-1812. 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 B	数据传输时低位在前
USART_MSBF_MS B	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 3-1813. 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3-1802. 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 3-1814. 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 3-1815. 函数 usart_overrun_disable

函数名称	usart_overrun_disable
-------------	-----------------------

函数原型	void usart_oversample_disable(uint32_t usart_periph);
功能描述	禁能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
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_disable(USART0);
```

函数 usart_oversample_config

函数usart_oversample_config描述见下表：

表 3-1816. 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 3-1817. 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 3-1818. 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 3-1819. 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 3-1820. 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 3-1821. 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 3-1822. 函数 usart_data_receive

函数名称	usart_data_receive
函数原型	uint16_t usart_data_receive(uint32_t usart_periph);
功能描述	USART接收数据功能
先决条件	-
被调用函数	-
输入参数{in}	
usart_periph	外设USARTx
USARTx	x = 0, 1, 2, 5
UARTx	x = 3, 4, 6, 7
输出参数{out}	
-	-
返回值	
uint16_t	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 3-1823. 函数 usart_command_enable

函数名称	usart_command_enable
函数原型	void usart_command_enable(uint32_t usart_periph, uint32_t cmdtype);
功能描述	使能USART请求
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}	
cmdtype	请求类型
USART_CMD_SBK CMD	发送断开帧请求
USART_CMD_MM CMD	静模式请求
USART_CMD_RXF CMD	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 3-1824. 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 3-1825. 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
<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 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 3-1826. 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
<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 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 3-1827. 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 3-1828. 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 3-1829. 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 3-1830. 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 3-1831. 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 3-1832. 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 3-1833. 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	禁用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 3-1834. 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 3-1835. 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 3-1836. 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 3-1837. 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 3-1838. 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 3-1839. 函数 usart_halfduplex_disable

函数名称	usart_halfduplex_disable
函数原型	void usart_halfduplex_disable(uint32_t usart_periph);
功能描述	禁能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
USARTx	x = 0, 1, 2, 5
UARTx	x = 3, 4, 6, 7
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* disable USART0 half duplex mode */
```

```
usart_halfduplex_disable(USART0);
```

函数 usart_clock_enable

函数usart_clock_enable描述见下表：

表 3-1840. 函数 usart_clock_enable

函数名称	usart_clock_enable
函数原型	void usart_clock_enable(uint32_t usart_periph);
功能描述	使能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：

```
/* enable clock */
```

```
usart_clock_enable(USART0);
```

函数 usart_clock_disable

函数usart_clock_disable描述见下表:

表 3-1841. 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 3-1842. 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 3-1843. 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 3-1844. 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 3-1845. 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 3-1846. 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>	x = 0, 1, 2, 5
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* enable USART0 NACK in smartcard mode */
usart_smartcard_mode_nack_enable(USART0);
```

函数 `usart_smartcard_mode_nack_disable`

函数`usart_smartcard_mode_nack_disable`描述见下表：

表 3-1847. 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>	x = 0, 1, 2, 5
输出参数{out}	
-	-
返回值	
-	-

例如：

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

函数 `usart_smartcard_mode_early_nack_enable`

函数`usart_smartcard_mode_early_nack_enable`描述见下表：

表 3-1848. 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 3-1849. 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 3-1850. 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 3-1851. 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 3-1852. 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 3-1853. 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 3-1854. 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 3-1855. 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 3-1856. 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 3-1857. 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 3-1858. 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 3-1859. 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 3-1860. 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 3-1861. 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 3-1862. 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 3-1863. 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 3-1864. 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 3-1865. 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 3-1866. 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 3-1867. 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 3-1868. 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 3-1869. 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 3-1870. 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 3-1871. 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 3-1872. 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 3-1873. 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 3-1874. 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 3-1875. 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 3-1876. 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3-1799. 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 3-1877. 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3-1799. 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 3-1878. 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3-1801. 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 3-1879. 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3-1801. 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 3-1880. 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3-1800. 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 3-1881. 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3-1800. 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);
```

3.49. 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[3.49.1](#)描述了VREF的寄存器列表，章节[3.49.2](#)对VREF库函数进行说明。

3.49.1. 外设寄存器说明

VREF寄存器列表如下表所示：

表 3-1882. 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

3.49.2. 外设库函数说明

VREF库函数列表如下表所示：

表 3-1883. 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 3-1884. 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 3-1885. 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 3-1886. 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 3-1887. 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 3-1888. 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 3-1889. 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 3-1890. 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 3-1891. 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 3-1892. 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();
```

3.50. 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[3.50.1](#)描述了WWDGT的寄存器列表，章节[3.50.2](#)对WWDGT库函数进行说明。

3.50.1. 外设寄存器说明

WWDGT寄存器列表如下表所示:

表 3-1893. 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

3.50.2. 外设库函数说明

WWDGT库函数列表如下表所示:

表 3-1894. 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 3-1895. 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 3-1896. 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 3-1897. 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 3-1898. 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 3-1899. 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 3-1900. 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 3-1901. 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();
```

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	2023 年 3 月 31 日
1.1	1. 修改函数接口，将表 3-233 中函数 <code>crc_block_data_calculate_byte</code> ，表 3-234 中函数 <code>crc_block_data_calculate_byte_halfword</code> ，表 3-235 中函数 <code>crc_block_data_calculate_byte_word</code> 更新为表 3-233 中函数 <code>crc_block_data_calculate</code> 。	2023 年 6 月 21 日
1.2	1. 删除表 3-1799 枚举 <code>usart_flag_enum</code> 中部分成员， <code>usart_interrupt_flag_clear</code> 函数接口中添加枚举。 2. 章节 3.5 中 CMP 一致性更新。 3. 章节 3.9 中 DAC 一致性更新。	2024 年 1 月 5 日
1.3	1. 更新表 3-1051 到表 3-1054 实参描述。	2024 年 7 月 10 日
1.4	1. 删除表 3-1173. 函数 <code>pmu_smmps_ldo_supply_config</code> 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.