

GigaDevice Semiconductor Inc.

GD32G5x3

Arm[®] Cortex[®]-M33 32-bit MCU

**固件库
使用指南**

1.1 版本

(2025 年 2 月)

目录

目录	2
图索引	6
表索引	7
1. 介绍	41
1.1. 文档和固件库规则	41
1.1.1. 外设缩写	41
1.1.2. 命名规则	42
2. 固件库概述	43
2.1. 文件组织结构	43
2.1.1. Examples 文件夹	44
2.1.2. Firmware 文件夹	44
2.1.3. Template 文件夹	44
2.1.4. Utilities 文件夹	46
2.2. 固件库文件描述	46
3. 外设固件库	48
3.1. 外设固件库概述	48
3.2. ADC	48
3.2.1. 外设寄存器描述	48
3.2.2. 外设库函数说明	49
3.3. CAN	91
3.3.1. 外设寄存器说明	91
3.3.2. 外设库函数说明	92
3.4. CAU	139
3.4.1. 外设寄存器说明	139
3.4.2. 外设库函数说明	139
3.5. CMP	168
3.5.1. 外设寄存器说明	168
3.5.2. 外设库函数说明	168
3.6. CPDM	183
3.6.1. 外设寄存器描述	183
3.6.2. 外设库函数说明	183
3.7. CRC	189
3.7.1. 外设寄存器说明	189

3.7.2.	外设库函数说明	189
3.8.	DAC	197
3.8.1.	外设寄存器说明	197
3.8.2.	外设库函数说明	198
3.9.	DBG	224
3.9.1.	外设寄存器说明	225
3.9.2.	外设库函数说明	225
3.10.	DMA / DMAMUX	230
3.10.1.	外设寄存器说明	230
3.10.2.	外设库函数说明	231
3.11.	CLA	277
3.11.1.	外设寄存器说明	277
3.11.2.	外设库函数说明	277
3.12.	EXMC	290
3.12.1.	外设寄存器描述	291
3.12.2.	外设库函数说明	291
3.13.	EXTI	298
3.13.1.	外设寄存器说明	299
3.13.2.	外设库函数说明	299
3.14.	FAC	307
3.14.1.	外设寄存器说明	307
3.14.2.	外设库函数说明	307
3.15.	FFT	326
3.15.1.	外设寄存器说明	326
3.15.2.	外设库函数说明	326
3.16.	FMC	340
3.16.1.	外设寄存器说明	340
3.16.2.	外设库函数说明	341
3.17.	FWDGT	377
3.17.1.	外设寄存器说明	377
3.17.2.	外设库函数说明	377
3.18.	GPIO	383
3.18.1.	外设寄存器说明	383
3.18.2.	外设库函数说明	383
3.19.	HPDF	395
3.19.1.	外设寄存器说明	395
3.19.2.	外设库函数说明	396
3.20.	I2C	453

3.20.1.	外设寄存器说明	453
3.20.2.	外设库函数说明	453
3.21.	LPTIMER.....	492
3.21.1.	外设寄存器说明	492
3.21.2.	外设库函数说明	493
3.22.	MISC	514
3.22.1.	外设寄存器说明	514
3.22.2.	外设库函数说明	516
3.23.	PMU.....	529
3.23.1.	外设寄存器说明	529
3.23.2.	外设库函数说明	529
3.24.	QSPI	546
3.24.1.	外设寄存器说明	546
3.24.2.	外设库函数说明	547
3.25.	RCU	566
3.25.1.	外设寄存器说明	567
3.25.2.	外设库函数说明	567
3.26.	RTC	606
3.26.1.	外设寄存器描述	606
3.26.2.	外设库函数描述	606
3.27.	HRTIMER.....	632
3.27.1.	外设寄存器说明	632
3.27.2.	外设库函数说明	634
3.28.	SPI.....	713
3.28.1.	外设寄存器说明	713
3.28.2.	外设库函数说明	714
3.29.	SYSCFG.....	739
3.29.1.	外设寄存器说明	739
3.29.2.	外设库函数说明	741
3.30.	TIMER	766
3.30.1.	外设寄存器说明	766
3.30.2.	外设库函数说明	767
3.31.	TMU.....	875
3.31.1.	外设寄存器说明	875
3.31.2.	外设库函数说明	875
3.32.	TRIGSEL.....	890
3.32.1.	外设寄存器说明	890
3.32.2.	外设库函数说明	891

3.33.	TRNG.....	901
3.33.1.	外设寄存器说明	901
3.33.2.	外设库函数说明	901
3.34.	USART.....	924
3.34.1.	外设寄存器说明	924
3.34.2.	外设库函数说明	924
3.35.	VREF	978
3.35.1.	外设寄存器说明	978
3.35.2.	外设库函数说明	978
3.36.	WWDGT	983
3.36.1.	外设寄存器说明	983
3.36.2.	外设库函数说明	983
4.	版本历史.....	988

图索引

图 2-1. GD32G5x3 固件库文件组织结构	43
图 2-2. 选择外设例程文件	45
图 2-3. 拷贝外设例程文件	45
图 2-4. 打开工程文件.....	45
图 2-5. 配置工程文件.....	46
图 2-6. 编译调试下载.....	46

表索引

表 1-1. 外设缩写	41
表 2-1. 固件函数库文件描述	46
表 3-1. 外设固件库函数描述格式	48
表 3-2. ADC 寄存器	48
表 3-3. ADC 库函数	49
表 3-4. 函数 adc_deinit	51
表 3-5. 函数 adc_clock_config	51
表 3-6. 函数 adc_special_function_config	53
表 3-7. 函数 adc_data_alignment_config	54
表 3-8. 函数 adc_enable	54
表 3-9. 函数 adc_disable	55
表 3-10. 函数 adc_calibration_mode_config	55
表 3-11. 函数 adc_calibration_number	56
表 3-12. 函数 adc_calibration_enable	57
表 3-13. 函数 adc_resolution_config	57
表 3-14. 函数 adc_resolution_config	58
表 3-15. 函数 adc_gain_mode_enable	59
表 3-16. 函数 adc_gain_mode_disable	59
表 3-17. 函数 adc_gain_factor_set	60
表 3-18. 函数 adc_dma_mode_enable	60
表 3-19. 函数 adc_dma_mode_disable	61
表 3-20. 函数 adc_dma_request_after_last_enable	61
表 3-21. 函数 adc_dma_request_after_last_disable	62
表 3-22. 函数 adc_hpdf_mode_enable	62
表 3-23. 函数 adc_hpdf_mode_disable	63
表 3-24. 函数 adc_discontinuous_mode_config	64
表 3-25. 函数 adc_channel_length_config	64
表 3-26. 函数 adc_routine_channel_config	65
表 3-27. 函数 adc_inserted_channel_config	66
表 3-28. 函数 adc_inserted_channel_offset_config	66
表 3-29. 函数 adc_inserted_channel_mean_value_mode_enable	67
表 3-30. 函数 adc_inserted_channel_mean_value_mode_disable	68
表 3-31. 函数 adc_inserted_channel_mean_value_mode_config	68
表 3-32. 函数 adc_channel_differential_mode_config	69
表 3-33. 函数 adc_external_trigger_config	70
表 3-34. 函数 adc_software_trigger_enable	71
表 3-35. 函数 adc_end_of_conversion_config	71
表 3-36. 函数 adc_routine_data_read	72
表 3-37. 函数 adc_inserted_data_read	72
表 3-38. 函数 adc_watchdog0_single_channel_enable	73

表 3-39. 函数 adc_watchdog0_sequence_channel_enable	74
表 3-40. 函数 adc_watchdog0_disable	75
表 3-41. 函数 adc_watchdog1_channel_config	75
表 3-42. 函数 adc_watchdog2_channel_config	76
表 3-43. 函数 adc_watchdog1_disable	77
表 3-44. 函数 adc_watchdog2_disable	77
表 3-45. 函数 adc_watchdog0_threshold_config	78
表 3-46. 函数 adc_watchdog1_threshold_config	78
表 3-47. 函数 adc_watchdog2_threshold_config	79
表 3-48. 函数 adc_oversample_mode_config	80
表 3-49. 函数 adc_oversample_mode_enable	81
表 3-50. 函数 adc_oversample_mode_disable	82
表 3-51. 函数 adc_flag_get	82
表 3-52. 函数 adc_flag_clear	83
表 3-53. 函数 adc_interrupt_enable	84
表 3-54. 函数 adc_interrupt_disable	84
表 3-55. 函数 adc_interrupt_flag_get	85
表 3-56. 函数 adc_interrupt_flag_clear	86
表 3-57. 函数 adc_sync_mode_config	87
表 3-58. 函数 adc_sync_delay_config	88
表 3-59. 函数 adc_sync_dma_config	89
表 3-60. 函数 adc_sync_dma_request_after_last_enable	89
表 3-61. 函数 adc_sync_dma_request_after_last_disable	90
表 3-62. 函数 adc_sync_master_adc_routine_data_read	90
表 3-63. 函数 adc_sync_slave_adc_routine_data_read	91
表 3-64. CAN 寄存器	91
表 3-65. CAN 库函数	93
表 3-66. 结构体 can_error_counter_struct	94
表 3-67. 结构体 can_parameter_struct	94
表 3-68. 结构体 can_mailbox_descriptor_struct	95
表 3-69. 结构体 can_rx_fifo_struct	95
表 3-70. 结构体 can_fd_parameter_struct	96
表 3-71. 结构体 can_rx_fifo_id_filter_struct	96
表 3-72. 结构体 can_fifo_parameter_struct	96
表 3-73. 结构体 can_pn_mode_filter_struct	96
表 3-74. 结构体 can_pn_mode_config_struct	97
表 3-75. 结构体 can_crc_struct	97
表 3-76. 枚举类型 can_interrupt_enum	97
表 3-77. 枚举类型 can_flag_enum	99
表 3-78. 枚举类型 can_interrupt_flag_enum	101
表 3-79. 枚举类型 can_operation_modes_enum	103
表 3-80. 枚举类型 can_struct_type_enum	104
表 3-81. 枚举类型 can_error_state_enum	104

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

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

表 3-168. 函数 cau_aes_ctr	156
表 3-169. 函数 cau_aes_cfb	157
表 3-170. 函数 cau_aes_ofb	158
表 3-171. 函数 cau_aes_gcm	159
表 3-172. 函数 cau_aes_ccm	160
表 3-173. 函数 cau_tdes_ecb	161
表 3-174. 函数 cau_tdes_cbc	162
表 3-175. 函数 cau_des_ecb	163
表 3-176. 函数 cau_des_cbc	164
表 3-177. 函数 cau_flag_get	164
表 3-178. 函数 cau_interrupt_enable	165
表 3-179. 函数 cau_interrupt_disable	166
表 3-180. 函数 cau_interrupt_flag_get	166
表 3-181. CMP 寄存器	168
表 3-182. CMP 库函数	168
表 3-183. 枚举类型 cmp_enum	169
表 3-184. 函数 cmp_deinit	169
表 3-185. 函数 cmp_mode_init	169
表 3-186. 函数 cmp_noninverting_input_select	171
表 3-187. 函数 cmp_output_init	172
表 3-188. 函数 cmp_output_mux_config	172
表 3-189. 函数 cmp_blanking_init	173
表 3-190. 函数 cmp_enable	174
表 3-191. 函数 cmp_disable	175
表 3-192. 函数 cmp_window_enable	175
表 3-193. 函数 cmp_window_disable	176
表 3-194. 函数 cmp_lock_enable	176
表 3-195. 函数 cmp_voltage_scaler_enable	177
表 3-196. 函数 cmp_voltage_scaler_disable	177
表 3-197. 函数 cmp_scaler_bridge_enable	178
表 3-198. 函数 cmp_scaler_bridge_disable	178
表 3-199. 函数 cmp_output_level_get	179
表 3-200. 函数 cmp_flag_get	179
表 3-201. 函数 cmp_flag_clear	180
表 3-202. 函数 cmp_interrupt_enable	180
表 3-203. 函数 cmp_interrupt_disable	181
表 3-204. 函数 cmp_interrupt_flag_get	182
表 3-205. 函数 cmp_interrupt_flag_clear	182
表 3-206. CPDM 寄存器	183
表 3-207. CPDM 库函数	183
表 3-208. 枚举类型 cpdm_output_phase_enum	183
表 3-209. 函数 cpdm_enable	184
表 3-210. 函数 cpdm_disable	185

表 3-211. 函数 cpdm_delayline_sample_enable	185
表 3-212. 函数 cpdm_delayline_sample_disable	186
表 3-213. 函数 cpdm_output_clock_phase_select	186
表 3-214. 函数 cpdm_delay_step_config	187
表 3-215. 函数 cpdm_delayline_length_valid_flag_get	187
表 3-216. 函数 cpdm_delayline_length_get	188
表 3-217. 函数 cpdm_clock_output	188
表 3-218. CRC 寄存器	189
表 3-219. CRC 库函数	189
表 3-220. 函数 crc_deinit	190
表 3-221. 函数 crc_reverse_output_data_enable	190
表 3-222. 函数 crc_reverse_output_data_disable	191
表 3-223. 函数 crc_data_register_reset	191
表 3-224. 函数 crc_data_register_read	192
表 3-225. 函数 crc_free_data_register_read	192
表 3-226. 函数 crc_free_data_register_write	193
表 3-227. 函数 crc_init_data_register_write	193
表 3-228. 函数 crc_input_data_reverse_config	194
表 3-229. 函数 crc_polynomial_size_set	194
表 3-230. 函数 crc_polynomial_set	195
表 3-231. 函数 crc_single_data_calculate	195
表 3-232. 函数 crc_block_data_calculate	196
表 3-233. DAC 寄存器	197
表 3-234. DAC 库函数	198
表 3-235. 函数 dac_deinit	199
表 3-236. 函数 dac_enable	200
表 3-237. 函数 dac_disable	200
表 3-238. 函数 dac_dma_enable	201
表 3-239. 函数 dac_dma_disable	201
表 3-240. 函数 dac_mode_config	202
表 3-241. 函数 dac_trimming_value_get	203
表 3-242. 函数 dac_trimming_value_set	203
表 3-243. 函数 dac_trimming_enable	204
表 3-244. 函数 dac_trimming_disable	205
表 3-245. 函数 dac_output_value_get	205
表 3-246. 函数 dac_data_format_config	206
表 3-247. 函数 dac_data_set	207
表 3-248. 函数 dac_trigger_enable	207
表 3-249. 函数 dac_trigger_disable	208
表 3-250. 函数 dac_trigger_source_config	208
表 3-251. 函数 dac_software_trigger_enable	209
表 3-252. 函数 dac_wave_mode_config	210
表 3-253. 函数 dac_lfsr_noise_config	211

表 3-254. 函数 dac_triangle_noise_config	211
表 3-255. 函数 dac_sawtooth_reset_trigger_source_config	212
表 3-256. 函数 dac_sawtooth_step_trigger_source_config	213
表 3-257. 函数 dac_sawtooth_step_direction_config	214
表 3-258. 函数 dac_sawtooth_initial_value_set	214
表 3-259. 函数 dac_sawtooth_initial_value_set	215
表 3-260. 函数 dac_sawtooth_step_software_trigger_enable	216
表 3-261. 函数 dac_concurrent_enable	216
表 3-262. 函数 dac_concurrent_disable	217
表 3-263. 函数 dac_concurrent_software_trigger_enable	217
表 3-264. 函数 dac_concurrent_data_set	218
表 3-265. 函数 dac_reset_persist_enable	219
表 3-266. 函数 dac_reset_persist_disable	219
表 3-267. 函数 dac_sample_keep_mode_config	220
表 3-268. 函数 dac_flag_get	220
表 3-269. 函数 dac_flag_clear	221
表 3-270. 函数 dac_interrupt_enable	222
表 3-271. 函数 dac_interrupt_disable	222
表 3-272. 函数 dac_interrupt_flag_get	223
表 3-273. 函数 dac_interrupt_flag_clear	224
表 3-274. DBG 寄存器	225
表 3-275. DBG 库函数	225
表 3-276. 枚举类型 dbg_periph_enum	225
表 3-277. 函数 dbg_deinit	226
表 3-278. 函数 dbg_id_get	226
表 3-279. 函数 dbg_low_power_enable	227
表 3-280. 函数 dbg_low_power_disable	227
表 3-281. 函数 dbg_trace_pin_enable	228
表 3-282. 函数 dbg_trace_pin_disable	228
表 3-283. 函数 dbg_periph_enable	229
表 3-284. 函数 dbg_periph_disable	229
表 3-285. DMA 寄存器	230
表 3-286. DMAMUX 寄存器	231
表 3-287. DMA 库函数	231
表 3-288. DMAMUX 库函数	232
表 3-289. 结构体 dma_parameter_struct	232
表 3-290. 结构体 dmamux_sync_parameter_struct	233
表 3-291. 结构体 dmamux_gen_parameter_struct	233
表 3-292. 枚举 dma_channel_enum	233
表 3-293. 枚举 dmamux_multiplexer_channel_enum	233
表 3-294. 枚举 dmamux_generator_channel_enum	234
表 3-295. 枚举 dmamux_interrupt_enum	234
表 3-296. 枚举 dmamux_flag_enum	235

表 3-297. 枚举 dmamux_interrupt_flag_enum.....	236
表 3-298. 函数 dma_deinit.....	237
表 3-299. 函数 dma_struct_para_init.....	238
表 3-300. 函数 dma_init.....	238
表 3-301. 函数 dma_circulation_enable.....	239
表 3-302. 函数 dma_circulation_disable.....	240
表 3-303. 函数 dma_memory_to_memory_enable.....	240
表 3-304. 函数 dma_memory_to_memory_disable.....	241
表 3-305. 函数 dma_channel_enable.....	242
表 3-306. 函数 dma_channel_disable.....	242
表 3-307. 函数 dma_periph_address_config.....	243
表 3-308. 函数 dma_memory_address_config.....	243
表 3-309. 函数 dma_transfer_number_config.....	244
表 3-310. 函数 dma_transfer_number_get.....	245
表 3-311. 函数 dma_priority_config.....	245
表 3-312. 函数 dma_memory_width_config.....	246
表 3-313. 函数 dma_periph_width_config.....	247
表 3-314. 函数 dma_memory_increase_enable.....	248
表 3-315. 函数 dma_memory_increase_disable.....	248
表 3-316. 函数 dma_periph_increase_enable.....	249
表 3-317. 函数 dma_periph_increase_disable.....	249
表 3-318. 函数 dma_transfer_direction_config.....	250
表 3-319. 函数 dma_flag_get.....	251
表 3-320. 函数 dma_flag_clear.....	251
表 3-321. 函数 dma_interrupt_enable.....	252
表 3-322. 函数 dma_interrupt_disable.....	253
表 3-323. 函数 dma_interrupt_flag_get.....	254
表 3-324. 函数 dma_interrupt_flag_clear.....	254
表 3-325. 函数 dmamux_sync_struct_para_init.....	255
表 3-326. 函数 dmamux_synchronization_init.....	256
表 3-327. 函数 dmamux_synchronization_enable.....	257
表 3-328. 函数 dmamux_synchronization_disable.....	257
表 3-329. 函数 dmamux_event_generation_enable.....	258
表 3-330. 函数 dmamux_event_generation_disable.....	258
表 3-331. 函数 dmamux_gen_struct_para_init.....	259
表 3-332. 函数 dmamux_request_generator_init.....	259
表 3-333. 函数 dmamux_request_generator_channel_enable.....	260
表 3-334. 函数 dmamux_request_generator_channel_disable.....	261
表 3-335. 函数 dmamux_synchronization_polarity_config.....	261
表 3-336. 函数 dmamux_request_forward_number_config.....	262
表 3-337. 函数 dmamux_sync_id_config.....	263
表 3-338. 函数 dmamux_request_id_config.....	264
表 3-339. 函数 dma_interrupt_disable.....	271

表 3-340. 函数 dmamux_request_generate_number_config	272
表 3-341. 函数 dmamux_trigger_id_config	272
表 3-342. 函数 dmamux_flag_get.....	274
表 3-343. 函数 dmamux_flag_clear.....	275
表 3-344. 函数 dmamux_interrupt_enable.....	275
表 3-345. 函数 dmamux_interrupt_disable.....	275
表 3-346. 函数 dmamux_interrupt_flag_get	276
表 3-347. 函数 dmamux_interrupt_flag_clear.....	276
表 3-348. CLA 寄存器	277
表 3-349. CLA 库函数	278
表 3-350. 函数 cla_deinit	278
表 3-351. 函数 cla_enable.....	279
表 3-352. 函数 cla_disable.....	279
表 3-353. 函数 cla_output_state_get	280
表 3-354. 函数 cla_sigs_input_config	280
表 3-355. 函数 cla_lcu_control_config.....	281
表 3-356. 函数 cla_output_config	282
表 3-357. 函数 cla_output_enable.....	282
表 3-358. 函数 cla_output_disable.....	283
表 3-359. 函数 cla_flip_flop_output_reset.....	284
表 3-360. 函数 cla_flip_flop_clockpolarity_config	284
表 3-361. 函数 cla_flip_flop_clocksource_config	285
表 3-362. 函数 cla_flag_get	285
表 3-363. 函数 cla_flag_clear	286
表 3-364. 函数 cla_negedge_interrupt_enable	287
表 3-365. 函数 cla_negedge_interrupt_disable	287
表 3-366. 函数 cla_posedge_interrupt_enable	288
表 3-367. 函数 cla_posedge_interrupt_disable	288
表 3-368. 函数 cla_interrupt_flag_get	289
表 3-369. 函数 cla_interrupt_flag_clear	290
表 3-370. EXMC 寄存器	291
表 3-371. EXMC 库函数	291
表 3-372. 结构体 exmc_norsram_timing_parameter_struct.....	291
表 3-373. 结构体 exmc_norsram_parameter_struct	292
表 3-374. 函数 exmc_norsram_deinit.....	292
表 3-375. 函数 exmc_norsram_struct_para_init	293
表 3-376. 函数 exmc_norsram_init	293
表 3-377. 函数 exmc_norsram_enable.....	295
表 3-378. 函数 exmc_norsram_disable.....	295
表 3-379. 函数 exmc_norsram_page_size_config	296
表 3-380. 函数 exmc_norsram_consecutive_clock_config.....	297
表 3-381. 函数 exmc_norsram_write_fifo_config.....	297
表 3-382. 函数 exmc_norsram_status_get	298

表 3-383. EXTI 寄存器.....	299
表 3-384. EXTI 库函数.....	299
表 3-385. 枚举类型 exti_line_enum	299
表 3-386. 枚举类型 exti_mode_enum	301
表 3-387. 枚举类型 exti_trig_type_enum	301
表 3-388. 函数 exti_deinit	301
表 3-389. 函数 exti_init.....	301
表 3-390. 函数 exti_interrupt_enable.....	302
表 3-391. 函数 exti_interrupt_disable.....	303
表 3-392. 函数 exti_event_enable	303
表 3-393. 函数 exti_event_disable	304
表 3-394. 函数 exti_software_interrupt_enable	304
表 3-395. 函数 exti_software_interrupt_disable.....	305
表 3-396. 函数 exti_flag_get.....	305
表 3-397. 函数 exti_flag_clear.....	306
表 3-398. 函数 exti_interrupt_flag_get	306
表 3-399. 函数 exti_interrupt_flag_clear	307
表 3-400. FAC 寄存器	307
表 3-401. FAC 库函数	308
表 3-402. 结构体 fac_parameter_struct.....	308
表 3-403. 结构体 fac_fixed_data_preload_struct	309
表 3-404. 结构体 fac_float_data_preload_struct	309
表 3-405. 函数 dac_deinit	309
表 3-406. 函数 fac_struct_para_init	310
表 3-407. 函数 fac_fixed_data_preload_init	310
表 3-408. 函数 fac_float_data_preload_init	311
表 3-409. 函数 fac_init.....	312
表 3-410. 函数 fac_fixed_buffer_preload	312
表 3-411. 函数 fac_float_buffer_preload	313
表 3-412. 函数 fac_fixed_data_preload	313
表 3-413. 函数 fac_float_data_preload.....	314
表 3-414. 函数 fac_reset	314
表 3-415. 函数 fac_clip_config	315
表 3-416. 函数 fac_float_enable	315
表 3-417. 函数 fac_float_disable	316
表 3-418. 函数 fac_dma_enable	316
表 3-419. 函数 fac_dma_disable	317
表 3-420. 函数 fac_x0_config.....	317
表 3-421. 函数 fac_x1_config.....	318
表 3-422. 函数 fac_y_config.....	319
表 3-423. 函数 fac_function_config	319
表 3-424. 函数 fac_start	320
表 3-425. 函数 fac_stop	320

表 3-426. 函数 fac_finish_calculate	321
表 3-427. 函数 fac_fixed_data_write.....	321
表 3-428. 函数 fac_fixed_data_read	322
表 3-429. 函数 fac_float_data_write.....	322
表 3-430. 函数 fac_float_data_read	323
表 3-431. 函数 fac_interrupt_enable.....	323
表 3-432. 函数 fac_interrupt_disable.....	324
表 3-433. 函数 fac_interrupt_flag_get	325
表 3-434. 函数 fac_flag_get.....	325
表 3-435. FFT 寄存器.....	326
表 3-436. FFT 库函数.....	327
表 3-437. 结构体 fft_parameter_struct	327
表 3-438. 函数 fft_deinit.....	328
表 3-439. 函数 fft_struct_para_init.....	328
表 3-440. 函数 fft_init	329
表 3-441. 函数 fft_calculation_start	330
表 3-442. 函数 fft_calculation_stop	330
表 3-443. 函数 fft_point_number_config	331
表 3-444. 函数 fft_mode_config	331
表 3-445. 函数 fft_window_enable	332
表 3-446. 函数 fft_window_disable.....	332
表 3-447. 函数 fft_downsample_config.....	333
表 3-448. 函数 fft_image_source_config	333
表 3-449. 函数 fft_real_addr_config.....	334
表 3-450. 函数 fft_image_addr_config	334
表 3-451. 函数 fft_window_addr_config	335
表 3-452. 函数 fft_output_addr_config	335
表 3-453. 函数 fft_loop_buffer_length_config	336
表 3-454. 函数 fft_loop_buffer_index_config.....	336
表 3-455. 函数 fft_flag_get.....	337
表 3-456. 函数 fft_flag_clear	338
表 3-457. 函数 fft_interrupt_enable	338
表 3-458. 函数 fft_interrupt_disable	339
表 3-459. 函数 fft_interrupt_flag_get.....	339
表 3-460. 函数 fft_interrupt_flag_clear.....	340
表 3-461. FMC 寄存器.....	341
表 3-462. FMC 库函数.....	341
表 3-463. 枚举类型 fmc_state_enum	343
表 3-464. 函数 fmc_unlock.....	343
表 3-465. 函数 fmc_lock.....	344
表 3-466. 函数 fmc_wsnt_set.....	344
表 3-467. 函数 fmc_prefetch_enable	345
表 3-468. 函数 fmc_prefetch_disable	345

表 3-469. 函数 fmc_icache_enable	346
表 3-470. 函数 fmc_icache_disable	346
表 3-471. 函数 fmc_icache_reset_enable	347
表 3-472. 函数 fmc_dcache_enable	347
表 3-473. 函数 fmc_dcache_disable	348
表 3-474. 函数 fmc_dcache_reset_enable	348
表 3-475. 函数 fmc_page_erase	349
表 3-476. 函数 fmc_bank0_erase	349
表 3-477. 函数 fmc_bank1_erase	350
表 3-478. 函数 fmc_mass_erase	350
表 3-479. 函数 fmc_doubleword_program	351
表 3-480. 函数 fmc_bank0_base_address_get	352
表 3-481. 函数 fmc_bank1_base_address_get	352
表 3-482. 函数 fmc_page_size_get	353
表 3-483. 函数 fmc_debugger_enable	353
表 3-484. 函数 fmc_debugger_disable	354
表 3-485. 函数 fmc_slp_pd_mode_enable	354
表 3-486. 函数 fmc_slp_pd_mode_disable	355
表 3-487. 函数 fmc_scr_area_enable	355
表 3-488. 函数 ob_unlock	356
表 3-489. 函数 ob_lock	356
表 3-490. 函数 ob_reload	357
表 3-491. 函数 ob_user_write	357
表 3-492. 函数 ob_security_protection_config	360
表 3-493. 函数 ob_dcrp_config	360
表 3-494. 函数 ob_write_protection_config	361
表 3-495. 函数 ob_scr_area_config	362
表 3-496. 函数 ob_boot_lock_config	363
表 3-497. 函数 ob_bank_memory_swap_config	364
表 3-498. 函数 ob_user_get	364
表 3-499. 函数 ob_security_protection_level_get	365
表 3-500. 函数 ob_dcrp_area_get	365
表 3-501. 函数 ob_write_protection_get	366
表 3-502. 函数 ob_scr_area_size_get	367
表 3-503. 函数 ob_boot_config_get	368
表 3-504. 函数 fmc_flag_get	368
表 3-505. 函数 fmc_flag_clear	369
表 3-506. 函数 fmc_interrupt_enable	370
表 3-507. 函数 fmc_interrupt_disable	370
表 3-508. 函数 fmc_ecc_flag_get	371
表 3-509. 函数 fmc_ecc_flag_clear	371
表 3-510. 函数 fmc_eccor_interrupt_enable	372
表 3-511. 函数 fmc_eccor_interrupt_disable	372

表 3-512. 函数 fmc_eccor_interrupt_flag_get.....	373
表 3-513. 函数 fmc_eccor_interrupt_flag_clear	374
表 3-514. 函数 fmc_interrupt_flag_get.....	374
表 3-515. 函数 fmc_interrupt_flag_clear.....	375
表 3-516. 函数 fmc_pd_mode_enter.....	375
表 3-517. 函数 fmc_pd_mode_exit.....	376
表 3-518. 函数 ob_bank_mode_config.....	376
表 3-519. FWDGT 寄存器.....	377
表 3-520. FWDGT 库函数.....	377
表 3-521. 函数 fwdgt_write_enable.....	378
表 3-522. 函数 fwdgt_write_disable.....	378
表 3-523. 函数 fwdgt_enable.....	379
表 3-524. 函数 fwdgt_prescaler_value_config.....	379
表 3-525. 函数 fwdgt_reload_value_config	380
表 3-526. 函数 fwdgt_window_value_config.....	380
表 3-527. 函数 fwdgt_counter_reload	381
表 3-528. 函数 fwdgt_config	381
表 3-529. 函数 fwdgt_flag_get	382
表 3-530. GPIO 寄存器.....	383
表 3-531. GPIO 库函数.....	383
表 3-532. 函数 gpio_deinit.....	384
表 3-533. 函数 gpio_mode_set	384
表 3-534. 函数 gpio_output_options_set.....	385
表 3-535. 函数 gpio_bit_set.....	386
表 3-536. 函数 gpio_bit_reset.....	387
表 3-537. 函数 gpio_bit_write	388
表 3-538. 函数 gpio_port_write	388
表 3-539. 函数 gpio_input_filter_set.....	389
表 3-540. 函数 gpio_input_bit_get	390
表 3-541. 函数 gpio_input_port_get	390
表 3-542. 函数 gpio_output_bit_get.....	391
表 3-543. 函数 gpio_output_port_get.....	392
表 3-544. 函数 gpio_af_set.....	392
表 3-545. 函数 gpio_pin_lock	393
表 3-546. 函数 gpio_bit_toggle.....	394
表 3-547. 函数 gpio_port_toggle	394
表 3-548. HPDF 寄存器.....	395
表 3-549. HPDF 库函数.....	396
表 3-550. 结构体 hpdf_channel_parameter_struct	398
表 3-551. 结构体 hpdf_filter_parameter_struct.....	398
表 3-552. 结构体 hpdf_rc_parameter_struct	399
表 3-553. 结构体 hpdf_ic_parameter_struct.....	399
表 3-554. 枚举类型 hpdf_channel_enum.....	399

表 3-555. 枚举类型 hpdf_filter_enum.....	399
表 3-556. 枚举类型 hpdf_flag_enum	400
表 3-557. 枚举类型 hpdf_interrupt_flag_enum	401
表 3-558. 枚举类型 hpdf_interrupt_enum	401
表 3-559. 函数 hpdf_deinit.....	402
表 3-560. 函数 hpdf_channel_struct_para_init.....	402
表 3-561. 函数 hpdf_filter_struct_para_init.....	403
表 3-562. 函数 hpdf_rc_struct_para_init.....	403
表 3-563. 函数 hpdf_ic_struct_para_init.....	404
表 3-564. 函数 hpdf_enable.....	404
表 3-565. 函数 hpdf_disable.....	405
表 3-566. 函数 hpdf_channel_init.....	405
表 3-567. 函数 hpdf_filter_init.....	406
表 3-568. 函数 hpdf_rc_init	407
表 3-569. 函数 hpdf_ic_init.....	408
表 3-570. 函数 hpdf_clock_output_config.....	409
表 3-571. 函数 hpdf_clock_output_source_config	409
表 3-572. 函数 hpdf_clock_output_duty_mode_disable.....	410
表 3-573. 函数 hpdf_clock_output_duty_mode_enable.....	411
表 3-574. 函数 hpdf_clock_output_divider_config	411
表 3-575. 函数 hpdf_channel_enable	412
表 3-576. 函数 hpdf_channel_disable	412
表 3-577. 函数 hpdf_spi_clock_source_config	413
表 3-578. 函数 hpdf_serial_interface_type_config	413
表 3-579. 函数 hpdf_malfunction_monitor_disable	414
表 3-580. 函数 hpdf_malfunction_monitor_enable.....	414
表 3-581. 函数 hpdf_clock_loss_disable	415
表 3-582. 函数 hpdf_clock_loss_enable	416
表 3-583. 函数 hpdf_channel_pin_redirection_disable.....	416
表 3-584. 函数 hpdf_channel_pin_redirection_enable.....	417
表 3-585. 函数 hpdf_channel_multiplexer_config.....	417
表 3-586. 函数 hpdf_data_pack_mode_config	418
表 3-587. 函数 hpdf_data_right_bit_shift_config	418
表 3-588. 函数 hpdf_calibration_offset_config.....	419
表 3-589. 函数 hpdf_malfunction_break_signal_config	420
表 3-590. 函数 hpdf_malfunction_counter_config.....	420
表 3-591. 函数 hpdf_write_parallel_data_standard_mode	421
表 3-592. 函数 hpdf_write_parallel_data_interleaved_mode	421
表 3-593. 函数 hpdf_write_parallel_data_dual_mode	422
表 3-594. 函数 hpdf_pulse_skip_update.....	423
表 3-595. 函数 hpdf_pulse_skip_read	423
表 3-596. 函数 hpdf_filter_enable	424
表 3-597. 函数 hpdf_filter_disable	424

表 3-598. 函数 hpdf_filter_config	425
表 3-599. 函数 hpdf_integrator_oversample	425
表 3-600. 函数 hpdf_threshold_monitor_filter_config	426
表 3-601. 函数 hpdf_threshold_monitor_filter_read_data	427
表 3-602. 函数 hpdf_threshold_monitor_fast_mode_disable	427
表 3-603. 函数 hpdf_threshold_monitor_fast_mode_enable	428
表 3-604. 函数 hpdf_threshold_monitor_channel	428
表 3-605. 函数 hpdf_threshold_monitor_high_threshold	429
表 3-606. 函数 hpdf_threshold_monitor_low_threshold	430
表 3-607. 函数 hpdf_high_threshold_break_signal	430
表 3-608. 函数 hpdf_low_threshold_break_signal	431
表 3-609. 函数 hpdf_extremes_monitor_channel	432
表 3-610. 函数 hpdf_extremes_monitor_maximum_get	432
表 3-611. 函数 hpdf_extremes_monitor_minimum_get	433
表 3-612. 函数 hpdf_conversion_time_get	433
表 3-613. 函数 hpdf_rc_continuous_disable	434
表 3-614. 函数 hpdf_rc_continuous_enable	435
表 3-615. 函数 hpdf_rc_start_by_software	435
表 3-616. 函数 hpdf_rc_syn_disable	436
表 3-617. 函数 hpdf_rc_syn_disable	436
表 3-618. 函数 hpdf_rc_dma_disable	437
表 3-619. 函数 hpdf_rc_dma_enable	437
表 3-620. 函数 hpdf_rc_channel_config	438
表 3-621. 函数 hpdf_rc_fast_mode_disable	438
表 3-622. 函数 hpdf_rc_fast_mode_enable	439
表 3-623. 函数 hpdf_rc_data_get	439
表 3-624. 函数 hpdf_rc_channel_get	440
表 3-625. 函数 hpdf_ic_start_by_software	440
表 3-626. 函数 hpdf_ic_syn_disable	441
表 3-627. 函数 hpdf_ic_syn_enable	441
表 3-628. 函数 hpdf_ic_dma_disable	442
表 3-629. 函数 hpdf_ic_dma_enable	442
表 3-630. 函数 hpdf_ic_scan_mode_disable	443
表 3-631. 函数 hpdf_ic_scan_mode_enable	443
表 3-632. 函数 hpdf_ic_trigger_signal_disable	444
表 3-633. 函数 hpdf_ic_trigger_signal_config	444
表 3-634. 函数 hpdf_ic_channel_config	445
表 3-635. 函数 hpdf_ic_data_get	446
表 3-636. 函数 hpdf_ic_channel_get	447
表 3-637. 函数 hpdf_flag_get	447
表 3-638. 函数 hpdf_flag_clear	448
表 3-639. 函数 hpdf_interrupt_enable	449
表 3-640. 函数 hpdf_interrupt_disable	450

表 3-641. 函数 hpdf_interrupt_flag_get	451
表 3-642. 函数 hpdf_interrupt_flag_clear	452
表 3-643. I2C 寄存器	453
表 3-644. I2C 库函数	453
表 3-645. 枚举类型 i2c_interrupt_flag_enum	455
表 3-646. 函数 i2c_deinit	455
表 3-647. 函数 i2c_timing_config	456
表 3-648. 函数 i2c_digital_noise_filter_config	457
表 3-649. 函数 i2c_analog_noise_filter_enable	458
表 3-650. 函数 i2c_analog_noise_filter_disable	458
表 3-651. 函数 i2c_master_clock_config	459
表 3-652. 函数 i2c_master_addressing	459
表 3-653. 函数 i2c_address10_header_enable	460
表 3-654. 函数 i2c_address10_header_disable	460
表 3-655. 函数 i2c_address10_enable	461
表 3-656. 函数 i2c_address10_disable	462
表 3-657. 函数 i2c_automatic_end_enable	462
表 3-658. 函数 i2c_automatic_end_disable	463
表 3-659. 函数 i2c_slave_response_to_gcall_enable	463
表 3-660. 函数 i2c_slave_response_to_gcall_disable	464
表 3-661. 函数 i2c_stretch_scl_low_enable	464
表 3-662. 函数 i2c_stretch_scl_low_disable	465
表 3-663. 函数 i2c_address_config	465
表 3-664. 函数 i2c_address_bit_compare_config	466
表 3-665. 函数 i2c_address_disable	467
表 3-666. 函数 i2c_second_address_config	467
表 3-667. 函数 i2c_second_address_disable	468
表 3-668. 函数 i2c_receved_address_get	469
表 3-669. 函数 i2c_slave_byte_control_enable	469
表 3-670. 函数 i2c_slave_byte_control_disable	470
表 3-671. 函数 i2c_nack_enable	470
表 3-672. 函数 i2c_nack_disable	471
表 3-673. 函数 i2c_wakeup_from_deepsleep_enable	471
表 3-674. 函数 i2c_wakeup_from_deepsleep_disable	472
表 3-675. 函数 i2c_enable	472
表 3-676. 函数 i2c_disable	473
表 3-677. 函数 i2c_start_on_bus	473
表 3-678. 函数 i2c_stop_on_bus	474
表 3-679. 函数 i2c_data_transmit	474
表 3-680. 函数 i2c_data_receive	475
表 3-681. 函数 i2c_reload_enable	475
表 3-682. 函数 i2c_reload_disable	476
表 3-683. 函数 i2c_transfer_byte_number_config	476

表 3-684. 函数 i2c_dma_enable.....	477
表 3-685. 函数 i2c_dma_disable.....	478
表 3-686. 函数 i2c_pec_transfer.....	478
表 3-687. 函数 i2c_pec_enable.....	479
表 3-688. 函数 i2c_pec_disable.....	479
表 3-689. 函数 i2c_pec_value_get	480
表 3-690. 函数 i2c_smbus_alert_enable	480
表 3-691. 函数 i2c_smbus_alert_disable	481
表 3-692. 函数 i2c_smbus_default_addr_enable	481
表 3-693. 函数 i2c_smbus_default_addr_disable	482
表 3-694. 函数 i2c_smbus_host_addr_enable.....	482
表 3-695. 函数 i2c_smbus_host_addr_disable.....	483
表 3-696. 函数 i2c_extented_clock_timeout_enable	483
表 3-697. 函数 i2c_extented_clock_timeout_disable	484
表 3-698. 函数 i2c_clock_timeout_enable	484
表 3-699. 函数 i2c_clock_timeout_disable	485
表 3-700. 函数 i2c_bus_timeout_b_config	485
表 3-701. 函数 i2c_bus_timeout_a_config	486
表 3-702. 函数 i2c_idle_clock_timeout_config	486
表 3-703. 函数 i2c_flag_get	487
表 3-704. 函数 i2c_flag_clear	488
表 3-705. 函数 i2c_interrupt_enable	489
表 3-706. 函数 i2c_interrupt_disable.....	489
表 3-707. 函数 i2c_interrupt_flag_get	490
表 3-708. 函数 i2c_interrupt_flag_clear	491
表 3-709. LPTIMER 寄存器	492
表 3-710. LPTIMER 库函数	493
表 3-711. 结构体 lptimer_parameter_struct.....	494
表 3-712. 函数 lptimer_deinit	495
表 3-713. 函数 lptimer_struct_para_init.....	495
表 3-714. 函数 lptimer_init.....	496
表 3-715. 函数 lptimer_inputremap.....	497
表 3-716. 函数 lptimer_register_shadow_enable	498
表 3-717. 函数 lptimer_register_shadow_disable	498
表 3-718. 函数 lptimer_timeout_enable	499
表 3-719. 函数 lptimer_register_shadow_disable	499
表 3-720. 函数 lptimer_counter_sync_reset	500
表 3-721. 函数 lptimer_counter_read_async_reset_enable	500
表 3-722. 函数 lptimer_counter_read_async_reset_disable	501
表 3-723. 函数 lptimer_continue_start.....	501
表 3-724. 函数 lptimer_continue_start.....	502
表 3-725. 函数 lptimer_stop	502
表 3-726. 函数 lptimer_counter_read	503

表 3-727. 函数 lptimer_autoreload_read	503
表 3-728. 函数 lptimer_compare_read	504
表 3-729. 函数 lptimer_autoreload_value_config	504
表 3-730. 函数 lptimer_compare_value_config	505
表 3-731. 函数 lptimer_decodemode0_enable	505
表 3-732. 函数 lptimer_decodemode1_enable	506
表 3-733. 函数 lptimer_decodemode_disable	506
表 3-734. 函数 lptimer_highlevelcounter_enable	507
表 3-735. 函数 lptimer_highlevelcounter_disable	507
表 3-736. 函数 lptimer_flag_get	508
表 3-737. 函数 timer_flag_clear	509
表 3-738. 函数 lptimer_interrupt_enable	510
表 3-739. 函数 lptimer_interrupt_enable	511
表 3-740. 函数 lptimer_interrupt_flag_get	512
表 3-741. 函数 lptimer_interrupt_flag_clear	513
表 3-742. NVIC 寄存器	514
表 3-743. SysTick 寄存器	516
表 3-744. MISC 库函数	516
表 3-745. 结构体 mpu_region_init_struct	516
表 3-746. 结构体 mpu_attribute_init_struct	517
表 3-747. 枚举类型 IRQn_Type	517
表 3-748. 函数 nvic_priority_group_set	520
表 3-749. 函数 nvic_irq_enable	521
表 3-750. 函数 nvic_irq_disable	521
表 3-751. 函数 nvic_system_reset	522
表 3-752. 函数 nvic_vector_table_set	522
表 3-753. 函数 system_lowpower_set	523
表 3-754. 函数 system_lowpower_reset	524
表 3-755. 函数 systick_clksource_set	524
表 3-756. 函数 mpu_enable	525
表 3-757. 函数 mpu_disable	525
表 3-758. 函数 mpu_region_struct_para_init	526
表 3-759. 函数 mpu_attribute_struct_para_init	526
表 3-760. 函数 mpu_region_config	527
表 3-761. 函数 mpu_attribute_config	528
表 3-762. 函数 mpu_region_enable	529
表 3-763. PMU 寄存器	529
表 3-764. PMU 库函数	529
表 3-765. 函数 pmu_deinit	530
表 3-766. 函数 pmu_lvd_select	531
表 3-767. 函数 pmu_lvd_enable	531
表 3-768. 函数 pmu_lvd_disable	532
表 3-769. 函数 pmu_avd_select	532

表 3-770. 函数 pmu_avd_enable	533
表 3-771. 函数 pmu_avd_disable	534
表 3-772. 函数 pmu_ovd_select	534
表 3-773. 函数 pmu_ovd_enable	535
表 3-774. 函数 pmu_ovd_disable	535
表 3-775. 函数 pmu_uvd_select	536
表 3-776. 函数 pmu_uvd_enable	536
表 3-777. 函数 pmu_uvd_disable	537
表 3-778. 函数 pmu_ovd_filter	537
表 3-779. 函数 pmu_uvd_filter	538
表 3-780. 函数 pmu_deepleep_voltage	538
表 3-781. 函数 pmu_vbat_charging_select	539
表 3-782. 函数 pmu_vbat_charging_enable	539
表 3-783. 函数 pmu_vbat_charging_disable	540
表 3-784. 函数 pmu_vbat_temp_monitor_enable	540
表 3-785. 函数 pmu_vbat_temp_monitor_disable	541
表 3-786. 函数 pmu_to_sleepmode	541
表 3-787. 函数 pmu_to_deepsleepmode	542
表 3-788. 函数 pmu_to_standbymode	542
表 3-789. 函数 pmu_wakeup_pin_enable	543
表 3-790. 函数 pmu_wakeup_pin_disable	543
表 3-791. 函数 pmu_backup_write_enable	544
表 3-792. 函数 pmu_backup_write_disable	545
表 3-793. 函数 pmu_flag_get	545
表 3-794. 函数 pmu_flag_clear	546
表 3-795. QSPI 寄存器	547
表 3-796. QSPI 库函数	547
表 3-797. 结构体 qspi_init_struct	548
表 3-798. 结构体 qspi_command_struct	548
表 3-799. 结构体 qspi_polling_struct	549
表 3-800. 函数 qspi_deinit	549
表 3-801. 函数 qspi_struct_para_init	549
表 3-802. 函数 qspi_cmd_struct_para_init	550
表 3-803. 函数 qspi_polling_struct_para_init	550
表 3-804. 函数 qspi_init	551
表 3-805. 函数 qspi_enable	552
表 3-806. 函数 qspi_disable	552
表 3-807. 函数 qspi_dma_enable	553
表 3-808. 函数 qspi_dma_disable	553
表 3-809. 函数 qspi_command_config	554
表 3-810. 函数 qspi_polling_config	554
表 3-811. 函数 qspi_memorymapped_config	555
表 3-812. 函数 qspi_data_transmit	557

表 3-813. 函数 qspi_data_receive	557
表 3-814. 函数 qspi_transmission_abort	558
表 3-815. 函数 qspi_output_clock_delay_enable	558
表 3-816. 函数 qspi_output_clock_delay_disable	559
表 3-817. 函数 qspi_output_clock_delay_config	559
表 3-818. 函数 qspi_sample_shift_config	560
表 3-819. 函数 qspi_receive_clock_sel	561
表 3-820. 函数 qspi_delay_scan_enable	561
表 3-821. 函数 qspi_delay_scan_disable	562
表 3-822. 函数 qspi_csn_edge_cycle	562
表 3-823. 函数 qspi_flag_get	563
表 3-824. 函数 qspi_flag_clear	563
表 3-825. 函数 qspi_interrupt_enable	564
表 3-826. 函数 qspi_interrupt_disable	565
表 3-827. 函数 qspi_interrupt_flag_get	565
表 3-828. 函数 qspi_interrupt_flag_clear	566
表 3-829. RCU 寄存器	567
表 3-830. RCU 库函数	568
表 3-831. 枚举类型 rcu_periph_enum	569
表 3-832. 枚举类型 rcu_periph_sleep_enum	571
表 3-833. 枚举类型 rcu_periph_reset_enum	572
表 3-834. 枚举类型 rcu_flag_enum	574
表 3-835. 枚举类型 rcu_int_flag_enum	574
表 3-836. 枚举类型 rcu_int_flag_clear_enum	575
表 3-837. 枚举类型 rcu_int_enum	575
表 3-838. 枚举类型 rcu_osci_type_enum	575
表 3-839. 枚举类型 rcu_clock_freq_enum	576
表 3-840. 枚举类型 usart_idx_enum	576
表 3-841. 枚举类型 i2c_idx_enum	576
表 3-842. 枚举类型 can_idx_enum	576
表 3-843. 枚举类型 adc_idx_enum	577
表 3-844. 函数 rcu_deinit	577
表 3-845. 函数 rcu_periph_clock_enable	577
表 3-846. 函数 rcu_periph_clock_disable	578
表 3-847. 函数 rcu_periph_clock_sleep_enable	578
表 3-848. 函数 rcu_periph_clock_sleep_disable	579
表 3-849. 函数 rcu_periph_reset_enable	579
表 3-850. 函数 rcu_periph_reset_disable	580
表 3-851. 函数 rcu_bkp_reset_enable	580
表 3-852. 函数 rcu_bkp_reset_disable	581
表 3-853. 函数 rcu_system_clock_source_config	581
表 3-854. 函数 rcu_system_clock_source_get	582
表 3-855. 函数 rcu_ahb_clock_config	582

表 3-856. 函数 rcu_apb1_clock_config	583
表 3-857. 函数 rcu_apb2_clock_config	583
表 3-858. 函数 rcu_apb3_clock_config	584
表 3-859. 函数 rcu_ckout_config	584
表 3-860. 函数 rcu_lsckout_enable.....	585
表 3-861. 函数 rcu_lsckout_disable.....	586
表 3-862. 函数 rcu_lsckout_config	586
表 3-863. 函数 rcu_pll_source_config.....	587
表 3-864. 函数 rcu_pll_config	587
表 3-865. 函数 rcu_pll_clock_output_enable	588
表 3-866. 函数 rcu_pll_clock_output_disable.....	589
表 3-867. 函数 rcu_rtc_clock_config	589
表 3-868. 函数 rcu_usart_clock_config	590
表 3-869. 函数 rcu_i2c_clock_config	591
表 3-870. 函数 rcu_can_clock_config	591
表 3-871. 函数 rcu_adc_clock_config	592
表 3-872. 函数 rcu_hpdcf_clock_config	593
表 3-873. 函数 rcu_hpdcfaudio_clock_config	593
表 3-874. 函数 rcu_trng_clock_config	594
表 3-875. 函数 rcu_lptimer_clock_config	595
表 3-876. 函数 rcu_qspi_clock_config	595
表 3-877. 函数 rcu_hrtimer_clock_config.....	596
表 3-878. 函数 rcu_lxtal_drive_capability_config	596
表 3-879. 函数 rcu_osci_stab_wait.....	597
表 3-880. 函数 rcu_osci_on.....	598
表 3-881. 函数 rcu_osci_off	598
表 3-882. 函数 rcu_osci_bypass_mode_enable	599
表 3-883. 函数 rcu_osci_bypass_mode_disable	599
表 3-884. 函数 rcu_irc8m_adjust_value_set	600
表 3-885. 函数 rcu_hxtal_clock_monitor_enable	600
表 3-886. 函数 rcu_hxtal_clock_monitor_disable	601
表 3-887. 函数 rcu_lxtal_clock_monitor_enable	601
表 3-888. 函数 rcu_lxtal_clock_monitor_disable	602
表 3-889. 函数 rcu_clock_freq_get	602
表 3-890. 函数 rcu_flag_get	603
表 3-891. 函数 rcu_all_reset_flag_clear	603
表 3-892. 函数 rcu_interrupt_enable	604
表 3-893. 函数 rcu_interrupt_disable	604
表 3-894. 函数 rcu_interrupt_flag_get.....	605
表 3-895. 函数 rcu_interrupt_flag_clear.....	605
表 3-896. RTC 寄存器	606
表 3-897. RTC 库函数	606
表 3-898. 结构体 rtc_parameter_struct	608

表 3-899. 结构体 rtc_alarm_struct	608
表 3-900. 结构体 rtc_timestamp_struct.....	608
表 3-901. 结构体 rtc_tamper_struct.....	608
表 3-902. 函数 rtc_deinit.....	609
表 3-903. 函数 rtc_init	609
表 3-904. 函数 rtc_init_mode_enter.....	610
表 3-905. 函数 rtc_init_mode_exit	611
表 3-906. 函数 rtc_register_sync_wait.....	611
表 3-907. 函数 rtc_current_time_get	612
表 3-908. 函数 rtc_subsecond_get	612
表 3-909. 函数 rtc_alarm_config	613
表 3-910. 函数 rtc_alarm_subsecond_config	613
表 3-911. 函数 rtc_alarm_enable	614
表 3-912. 函数 rtc_alarm_disable.....	615
表 3-913. 函数 rtc_alarm_get	615
表 3-914. 函数 rtc_alarm_subsecond_get.....	616
表 3-915. 函数 rtc_timestamp_enable	616
表 3-916. 函数 rtc_timestamp_disable.....	617
表 3-917. 函数 rtc_timestamp_internalevent_config.....	617
表 3-918. 函数 rtc_timestamp_get	618
表 3-919. 函数 rtc_timestamp_subsecond_get	618
表 3-920. 函数 rtc_tamper_enable	619
表 3-921. 函数 rtc_tamper_disable	619
表 3-922. 函数 rtc_output_pin_select.....	620
表 3-923. 函数 rtc_alarm_output_config.....	621
表 3-924. 函数 rtc_calibration_output_config	621
表 3-925. 函数 rtc_hour_adjust	622
表 3-926. 函数 rtc_second_adjust	623
表 3-927. 函数 rtc_bypass_shadow_enable.....	623
表 3-928. 函数 rtc_bypass_shadow_disable.....	624
表 3-929. 函数 rtc_refclock_detection_enable	624
表 3-930. 函数 rtc_refclock_detection_disable	625
表 3-931. 函数 rtc_wakeup_enable	625
表 3-932. 函数 rtc_wakeup_disable	626
表 3-933. 函数 rtc_wakeup_clock_set.....	626
表 3-934. 函数 rtc_wakeup_timer_set	627
表 3-935. 函数 rtc_wakeup_timer_get.....	627
表 3-936. 函数 rtc_smooth_calibration_config.....	628
表 3-937. 函数 rtc_interrupt_enable	629
表 3-938. 函数 rtc_interrupt_disable	629
表 3-939. 函数 rtc_flag_get	630
表 3-940. 函数 rtc_flag_clear	631
表 3-941. HRTIMER 寄存器.....	632

表 3-942. HRTIMER 库函数.....	635
表 3-943. 结构体 hrtimer_baseinit_parameter_struct.....	637
表 3-944. 结构体 hrtimer_timerinit_parameter_struct.....	638
表 3-945. 结构体 hrtimer_timercfg_parameter_struct.....	638
表 3-946. 结构体 hrtimer_capture_value_struct.....	638
表 3-947. 结构体 hrtimer_comparecfg_parameter_struct	639
表 3-948. 结构体 hrtimer_exefilter_parameter_struct	639
表 3-949. 结构体 hrtimer_deadtimecfg_parameter_struct.....	639
表 3-950. 结构体 hrtimer_carriersignalcfg_parameter_struct	639
表 3-951. 结构体 hrtimer_synccfg_parameter_struct.....	640
表 3-952. 结构体 hrtimer_bunchmode_parameter_struct	640
表 3-953. 结构体 hrtimer_exeeventcfg_parameter_struct	640
表 3-954. 结构体 hrtimer_exeeventcnt_parameter_struct	640
表 3-955. 结构体 hrtimer_faultcfg_parameter_struct	640
表 3-956. 结构体 hrtimer_adctrigcfg_parameter_struct.....	641
表 3-957. 结构体 hrtimer_channel_outputcfg_parameter_struct	641
表 3-958. 结构体 hrtimer_roll_over_parameter_struct.....	641
表 3-959. 结构体 hrtimer_double_trigger_parameter_struct	642
表 3-960. 函数 hrtimer_deinit.....	642
表 3-961. 函数 hrtimer_dll_calibration_start.....	642
表 3-962. 函数 hrtimer_baseinit_struct_para_init.....	643
表 3-963. 函数 hrtimer_timers_base_init.....	644
表 3-964. 函数 hrtimer_timers_counter_enable	645
表 3-965. 函数 hrtimer_timers_counter_enable	645
表 3-966. 函数 hrtimer_timers_update_event_enable.....	646
表 3-967. 函数 hrtimer_timers_update_event_enable.....	647
表 3-968. 函数 hrtimer_software_update.....	647
表 3-969. 函数 hrtimer_software_counter_reset	648
表 3-970. 函数 hrtimer_output_exchange	649
表 3-971. 函数 hrtimer_timerinit_struct_para_init	649
表 3-972. 函数 hrtimer_timers_waveform_init.....	650
表 3-973. 函数 hrtimer_timercfg_struct_para_init	651
表 3-974. 函数 hrtimer_slavetimer_waveform_config.....	652
表 3-975. 函数 hrtimer_comparecfg_struct_para_init.....	653
表 3-976. 函数 hrtimer_slavetimer_waveform_compare_config.....	653
表 3-977. 函数 hrtimer_channel_outputcfg_struct_para_init	654
表 3-978. 函数 hrtimer_slavetimer_waveform_channel_config	655
表 3-979. 函数 hrtimer_slavetimer_waveform_channel_software_request	656
表 3-980. 函数 hrtimer_slavetimer_waveform_channel_output_level_get.....	657
表 3-981. 函数 hrtimer_slavetimer_waveform_channel_state_get.....	658
表 3-982. 函数 hrtimer_channel_outputcfg_struct_para_init	659
表 3-983. 函数 hrtimer_slavetimer_waveform_channel_software_request	659
表 3-984. 函数 hrtimer_channel_outputcfg_struct_para_init	660

表 3-985. 函数 hrtimer_slavetimer_carriersignal_config.....	661
表 3-986. 函数 hrtimer_output_channel_enable	662
表 3-987. 函数 hrtimer_output_channel_disable	662
表 3-988. 函数 hrtimer_slavetimer_waveform_compare_config	663
表 3-989. 函数 hrtimer_slavetimer_compare_value_get.....	664
表 3-990. 函数 hrtimer_mastertimer_compare_value_config	665
表 3-991. 函数 hrtimer_slavetimer_compare_value_get.....	665
表 3-992. 函数 hrtimer_timers_counter_value_config	666
表 3-993. 函数 hrtimer_timers_counter_value_get.....	667
表 3-994. 函数 hrtimer_timers_autoreload_value_config.....	667
表 3-995. 函数 hrtimer_timers_autoreload_value_get	668
表 3-996. 函数 hrtimer_timers_repetition_value_config.....	669
表 3-997. 函数 hrtimer_timers_repetition_value_get	669
表 3-998. 函数 hrtimer_exefilter_struct_para_init.....	670
表 3-999. 函数 hrtimer_slavetimer_exeevent_filtering_config	671
表 3-1000. 函数 hrtimer_exeeventcfg_struct_para_init	672
表 3-1001. 函数 hrtimer_exeevent_config	672
表 3-1002. 函数 hrtimer_exeevent_prescaler.....	673
表 3-1003. 函数 hrtimer_exeeventx_counter_struct_para_init.....	674
表 3-1004. 函数 hrtimer_exeeventx_counter_config	675
表 3-1005. 函数 hrtimer_software_reset_exeeventx_counter.....	675
表 3-1006. 函数 hrtimer_exeeventx_counter_enable	676
表 3-1007. 函数 hrtimer_exeeventx_counter_disable	677
表 3-1008. 函数 hrtimer_exeeventx_counter_read	677
表 3-1009. 函数 hrtimer_synccfg_struct_para_init.....	678
表 3-1010. 函数 hrtimer_synchronization_config	679
表 3-1011. 函数 hrtimer_double_channel_struct_para_init.....	679
表 3-1012. 函数 hrtimer_double_trigger_config.....	680
表 3-1013. 函数 hrtimer_roll_over_struct_para_init	681
表 3-1014. 函数 hrtimer_roll_over_mode_config	681
表 3-1015. 函数 hrtimer_faultcfg_struct_para_init	682
表 3-1016. 函数 hrtimer_fault_config	683
表 3-1017. 函数 hrtimer_fault_prescaler_config	684
表 3-1018. 函数 hrtimer_fault_input_enable	684
表 3-1019. 函数 hrtimer_fault_input_disable	685
表 3-1020. 函数 hrtimer_fault_counter_reset	686
表 3-1021. 函数 hrtimer_fault_blank_enable.....	686
表 3-1022. 函数 hrtimer_fault_blank_disable	687
表 3-1023. 函数 hrtimer_timers_dma_enable	687
表 3-1024. 函数 hrtimer_timers_dma_enable	688
表 3-1025. 函数 hrtimer_dmamode_config	690
表 3-1026. 函数 hrtimer_bunchmode_struct_para_init.....	691
表 3-1027. 函数 hrtimer_bunchmode_config.....	692

表 3-1028. 函数 hrtimer_bunchmode_enable	693
表 3-1029. 函数 hrtimer_bunchmode_disable	693
表 3-1030. 函数 hrtimer_bunchmode_flag_get.....	694
表 3-1031. 函数 hrtimer_bunchmode_software_start	694
表 3-1032. 函数 hrtimer_slavetimer_capture_config.....	695
表 3-1033. 函数 hrtimer_slavetimer_capture_software	696
表 3-1034. 函数 hrtimer_slavetimer_capture_value_read	697
表 3-1035. 函数 hrtimer_adctrigcfg_struct_para_init.....	698
表 3-1036. 函数 hrtimer_adc_trigger0_3_config	698
表 3-1037. 函数 hrtimer_adc_trigger4_9_config	699
表 3-1038. 函数 hrtimer_adc_prescaler_config.....	700
表 3-1039. 函数 hrtimer_slavetimer_counter_direction_get	701
表 3-1040. 函数 hrtimer_timers_flag_get.....	701
表 3-1041. 函数 hrtimer_timers_flag_clear	703
表 3-1042. 函数 hrtimer_common_flag_get	704
表 3-1043. 函数 hrtimer_common_flag_clear	705
表 3-1044. 函数 hrtimer_timers_interrupt_enable	706
表 3-1045. 函数 hrtimer_timers_interrupt_disable	707
表 3-1046. 函数 hrtimer_timers_interrupt_flag_get.....	708
表 3-1047. 函数 hrtimer_timers_interrupt_flag_clear.....	709
表 3-1048. 函数 hrtimer_common_interrupt_enable	710
表 3-1049. 函数 hrtimer_common_interrupt_disable	711
表 3-1050. 函数 hrtimer_common_interrupt_flag_get.....	712
表 3-1051. 函数 hrtimer_common_interrupt_flag_clear	713
表 3-1052. SPI/寄存器.....	713
表 3-1053. SPI 库函数.....	714
表 3-1054. 结构体 spi_parameter_struct.....	715
表 3-1055. 函数 spi_deinit	715
表 3-1056. 函数 spi_struct_para_init.....	716
表 3-1057. 函数 spi_init.....	716
表 3-1058. 函数 spi_enable	717
表 3-1059. 函数 spi_disable	718
表 3-1060. 函数 spi_nss_output_enable.....	718
表 3-1061. 函数 spi_nss_output_disable.....	719
表 3-1062. 函数 spi_nss_internal_high.....	719
表 3-1063. 函数 spi_nss_internal_low.....	720
表 3-1064. 函数 spi_dma_enable	720
表 3-1065. 函数 spi_dma_disable	721
表 3-1066. 函数 spi_transmit_odd_config.....	722
表 3-1067. 函数 spi_receive_odd_config.....	722
表 3-1068. 函数 spi_data_frame_format_config.....	723
表 3-1069. 函数 spi_fifo_access_size_config.....	723
表 3-1070. 函数 spi_bidirectional_transfer_config.....	724

表 3-1071. 函数 spi_data_transmit	725
表 3-1072. 函数 spi_data_receive	725
表 3-1073. 函数 spi_crc_polynomial_set	726
表 3-1074. 函数 spi_crc_polynomial_get	726
表 3-1075. 函数 spi_crc_length_set	727
表 3-1076. 函数 spi_crc_on	728
表 3-1077. 函数 spi_crc_off	728
表 3-1078. 函数 spi_crc_next	729
表 3-1079. 函数 spi_crc_get	729
表 3-1080. 函数 spi_crc_error_clear	730
表 3-1081. 函数 spi_ti_mode_enable	730
表 3-1082. 函数 spi_ti_mode_disable	731
表 3-1083. 函数 spi_nssp_mode_enable	731
表 3-1084. 函数 spi_nssp_mode_disable	732
表 3-1085. 函数 spi_quad_enable	732
表 3-1086. 函数 spi_quad_disable	733
表 3-1087. 函数 spi_quad_write_enable	733
表 3-1088. 函数 spi_quad_read_enable	734
表 3-1089. 函数 spi_quad_io23_output_enable	734
表 3-1090. 函数 spi_quad_io23_output_disable	735
表 3-1091. 函数 spi_format_error_clear	735
表 3-1092. 函数 spi_flag_get	736
表 3-1093. 函数 spi_interrupt_enable	737
表 3-1094. 函数 spi_interrupt_disable	738
表 3-1095. 函数 spi_interrupt_flag_get	738
表 3-1096. SYSCFG 寄存器	740
表 3-1097. SYSCFG 库函数	741
表 3-1098. 枚举类型 syscfg_interrupt_enum	742
表 3-1099. 枚举类型 syscfg_flag_enum	742
表 3-1100. 枚举类型 syscfg_sram_serrbits_enum	742
表 3-1101. 枚举类型 syscfg_sram_erraddr_enum	743
表 3-1102. 枚举类型 timer_channel_input_enum	743
表 3-1103. 函数 syscfg_deinit	751
表 3-1104. 函数 syscfg_i2c_fast_mode_plus_enable	751
表 3-1105. 函数 syscfg_i2c_fast_mode_plus_disable	752
表 3-1106. 函数 syscfg_exti_line_config	753
表 3-1107. 函数 syscfg_pin_reset_mode_config	754
表 3-1108. 函数 syscfg_trigsel_cla_reset_mode_config	754
表 3-1109. 函数 syscfg_lockup_enable	755
表 3-1110. 函数 syscfg_timer_input_source_select	756
表 3-1111. 函数 syscfg_flash_bank_remap_set	756
表 3-1112. 函数 syscfg_bootmode_memmap_select	757
表 3-1113. 函数 syscfg_sram_ecc_single_correctable_bit_get	757

表 3-1114. 函数 syscfg_sram_ecc_error_address_get	758
表 3-1115. 函数 syscfg_tcmsram_erase	759
表 3-1116. 函数 syscfg_tcmsram_erase_lock.....	759
表 3-1117. 函数 syscfg_tcmsram_erase_unlock	760
表 3-1118. 函数 syscfg_tcmsram_page_wp_enable	760
表 3-1119. 函数 syscfg_io_compensation_config	761
表 3-1120. 函数 syscfg_fpu_interrupt_enable	761
表 3-1121. 函数 syscfg_fpu_interrupt_disable	762
表 3-1122. 函数 syscfg_interrupt_flag_get.....	763
表 3-1123. 函数 syscfg_interrupt_flag_clear.....	763
表 3-1124. 函数 syscfg_interrupt_enable	764
表 3-1125. 函数 syscfg_interrupt_disable	764
表 3-1126. 函数 syscfg_tcmsram_busy_flag_get.....	765
表 3-1127. 函数 syscfg_compensation_cell_ready_flag_get	765
表 3-1128. TIMER 寄存器	766
表 3-1129. TIMER 库函数	767
表 3-1130. 结构体 timer_parameter_struct	771
表 3-1131. 结构体 timer_break_parameter_struct.....	772
表 3-1132. 结构体 timer_oc_parameter_struct	773
表 3-1133. 结构体 timer_omc_parameter_struct	773
表 3-1134. 结构体 timer_ic_parameter_struct.....	773
表 3-1135. 结构体 timer_free_complementary_parameter_struct.....	773
表 3-1136. 函数 timer_deinit.....	774
表 3-1137. 函数 timer_struct_para_init	774
表 3-1138. 函数 timer_init.....	775
表 3-1139. 函数 timer_enable.....	776
表 3-1140. 函数 timer_disable.....	776
表 3-1141. 函数 timer_auto_reload_shadow_enable	777
表 3-1142. 函数 timer_auto_reload_shadow_disable	777
表 3-1143. 函数 timer_update_event_enable	778
表 3-1144. 函数 timer_update_event_disable	778
表 3-1145. 函数 timer_counter_alignment.....	779
表 3-1146. 函数 timer_counter_up_direction	780
表 3-1147. 函数 timer_counter_down_direction.....	780
表 3-1148. 函数 timer_adjustment_mode_config	781
表 3-1149. 函数 timer_prescaler_config	781
表 3-1150. 函数 timer_repetition_value_config	782
表 3-1151. 函数 timer_runtime_repetition_value_read.....	783
表 3-1152. 函数 timer_autoreload_value_config	783
表 3-1153. 函数 timer_auto_reload_fract_value_config.....	784
表 3-1154. 函数 timer_autoreload_value_read	785
表 3-1155. 函数 timer_auto_reload_fract_value_read	785
表 3-1156. 函数 timer_counter_value_config.....	786

表 3-1157. 函数 timer_counter_read.....	786
表 3-1158. 函数 timer_prescaler_read.....	787
表 3-1159. 函数 timer_single_pulse_mode_config.....	788
表 3-1160. 函数 timer_delayable_single_pulse_mode_config.....	788
表 3-1161. 函数 timer_update_source_config.....	789
表 3-1162. 函数 timer_ocpre_clear_source_config.....	790
表 3-1163. 函数 timer_ocpre_clear_input_config.....	791
表 3-1164. 函数 timer_pulse_on_compare_config.....	791
表 3-1165. 函数 timer_dma_enable.....	792
表 3-1166. 函数 timer_dma_disable.....	793
表 3-1167. 函数 timer_channel_dma_request_source_select.....	794
表 3-1168. 函数 timer_dma_transfer_config.....	795
表 3-1169. 函数 timer_event_software_generate.....	798
表 3-1170. 函数 timer_break_struct_para_init.....	799
表 3-1171. 函数 timer_break_config.....	800
表 3-1172. 函数 timer_break_enable.....	801
表 3-1173. 函数 timer_break_disable.....	801
表 3-1174. 函数 timer_automatic_output_enable.....	802
表 3-1175. 函数 timer_automatic_output_disable.....	803
表 3-1176. 函数 timer_primary_output_config.....	803
表 3-1177. 函数 timer_channel_control_shadow_config.....	804
表 3-1178. 函数 timer_channel_control_shadow_update_config.....	804
表 3-1179. 函数 timer_channel_output_struct_para_init.....	805
表 3-1180. 函数 timer_channel_output_config.....	806
表 3-1181. 函数 timer_channel_output_mode_config.....	807
表 3-1182. 函数 timer_channel_output_pulse_value_config.....	808
表 3-1183. 函数 timer_channel_output_pulse_fract_value_config.....	809
表 3-1184. 函数 timer_channel_output_shadow_config.....	810
表 3-1185. 函数 timer_channel_output_clear_config.....	811
表 3-1186. 函数 timer_channel_output_compare_fast_config.....	812
表 3-1187. 函数 timer_channel_output_polarity_config.....	813
表 3-1188. 函数 timer_channel_complementary_output_polarity_config.....	814
表 3-1189. 函数 timer_channel_output_state_config.....	815
表 3-1190. 函数 timer_channel_complementary_output_state_config.....	816
表 3-1191. 函数 timer_channel_input_struct_para_init.....	817
表 3-1192. 函数 timer_input_capture_config.....	817
表 3-1193. 函数 timer_channel_input_capture_prescaler_config.....	818
表 3-1194. 函数 timer_channel_capture_value_register_read.....	820
表 3-1195. 函数 timer_input_pwm_capture_config.....	820
表 3-1196. 函数 timer_hall_mode_config.....	821
表 3-1197. 函数 timer_multi_mode_channel_output_parameter_struct_init.....	822
表 3-1198. 函数 timer_multi_mode_channel_output_config.....	822
表 3-1199. 函数 timer_multi_mode_channel_mode_config.....	823

表 3-1200. 函数 timer_input_trigger_source_select.....	824
表 3-1201. 函数 timer_master_output0_trigger_source_select.....	826
表 3-1202. 函数 timer_master_output1_trigger_source_select.....	827
表 3-1203. 函数 timer_slave_mode_select.....	828
表 3-1204. 函数 timer_pause_reset_polarity_config.....	829
表 3-1205. 函数 timer_master_slave_mode_config.....	830
表 3-1206. 函数 timer_external_trigger_config.....	831
表 3-1207. 函数 timer_quadrature_decoder_mode_config.....	832
表 3-1208. 函数 timer_decoder_mode_config.....	833
表 3-1209. 函数 timer_internal_clock_config.....	834
表 3-1210. 函数 timer_internal_trigger_as_external_clock_config.....	835
表 3-1211. 函数 timer_external_trigger_as_external_clock_config.....	836
表 3-1212. 函数 timer_external_clock_mode0_config.....	837
表 3-1213. 函数 timer_external_clock_mode1_config.....	838
表 3-1214. 函数 timer_external_clock_mode1_disable.....	839
表 3-1215. 函数 timer_write_chxval_register_config.....	839
表 3-1216. 函数 timer_output_value_selection_config.....	840
表 3-1217. 函数 timer_commutation_control_shadow_register_config.....	841
表 3-1218. 函数 timer_output_match_pulse_select.....	841
表 3-1219. 函数 timer_channel_composite_pwm_mode_config.....	842
表 3-1220. 函数 timer_channel_composite_pwm_mode_output_pulse_value_config.....	843
表 3-1221. 函数 timer_channel_additional_compare_value_config.....	844
表 3-1222. 函数 timer_channel_additional_output_shadow_config.....	845
表 3-1223. 函数 timer_channel_additional_output_update_select.....	846
表 3-1224. 函数 timer_channel_additional_compare_value_read.....	846
表 3-1225. 函数 timer_break_external_source_config.....	847
表 3-1226. 函数 timer_break_external_polarity_config.....	848
表 3-1227. 函数 timer_break_lock_config.....	849
表 3-1228. 函数 timer_break_lock_release_config.....	850
表 3-1229. 函数 timer_dead_time_falling_edge_config.....	851
表 3-1230. 函数 timer_dead_time_different_config.....	851
表 3-1231. 函数 timer_dead_time_modify_config.....	852
表 3-1232. 函数 timer_channel_break_control_config.....	853
表 3-1233. 函数 timer_channel_dead_time_config.....	853
表 3-1234. 函数 timer_free_complementary_struct_para_init.....	854
表 3-1235. 函数 timer_channel_free_complementary_config.....	855
表 3-1236. 函数 timer_watchdog_value_config.....	856
表 3-1237. 函数 timer_watchdog_value_read.....	856
表 3-1238. 函数 timer_decoder_disconnection_detection_config.....	857
表 3-1239. 函数 timer_decoder_jump_detection_config.....	857
表 3-1240. 函数 timer_decoder_jump_detection_config.....	858
表 3-1241. 函数 timer_decoder_mode_update_source_config.....	859
表 3-1242. 函数 timer_index_reset_counter_config.....	859

表 3-1243. 函数 timer_index_reset_direction_config	860
表 3-1244. 函数 timer_decoder_index_position_config	861
表 3-1245. 函数 timer_first_index_reset_counter_config	862
表 3-1246. 函数 timer_upif_backup_config	862
表 3-1247. 函数 timer_upifbu_bit_get	863
表 3-1248. 函数 timer_counter_initial_register_config	864
表 3-1249. 函数 timer_counter_initial_config	864
表 3-1250. 函数 timer_synchronization_event_generate	865
表 3-1251. 函数 timer_flag_get	866
表 3-1252. 函数 timer_flag_clear	867
表 3-1253. 函数 timer_interrupt_enable	869
表 3-1254. 函数 timer_interrupt_disable	871
表 3-1255. 函数 timer_interrupt_flag_get	872
表 3-1256. 函数 timer_interrupt_flag_clear	873
表 3-1257. TMU 寄存器	875
表 3-1258. TMU 库函数	876
表 3-1259. 结构体 tmu_parameter_struct	876
表 3-1260. 函数 tmu_deinit	877
表 3-1261. 函数 tmu_struct_para_init	877
表 3-1262. 函数 tmu_init	878
表 3-1263. 函数 tmu_dma_read_enable	879
表 3-1264. 函数 tmu_dma_read_disable	879
表 3-1265. 函数 tmu_dma_write_enable	880
表 3-1266. 函数 tmu_dma_write_disable	880
表 3-1267. 函数 tmu_one_q31_write	881
表 3-1268. 函数 tmu_two_q31_write	881
表 3-1269. 函数 tmu_two_q15_write	882
表 3-1270. 函数 tmu_one_f32_write	882
表 3-1271. 函数 tmu_two_f32_write	883
表 3-1272. 函数 tmu_one_q31_read	883
表 3-1273. 函数 tmu_two_q31_read	884
表 3-1274. 函数 tmu_two_q15_read	885
表 3-1275. 函数 tmu_one_f32_read	885
表 3-1276. 函数 tmu_two_f32_read	886
表 3-1277. 函数 tmu_flag_get	886
表 3-1278. 函数 tmu_flag_clear	887
表 3-1279. 函数 tmu_interrupt_enable	887
表 3-1280. 函数 tmu_interrupt_disable	888
表 3-1281. 函数 tmu_interrupt_flag_get	888
表 3-1282. 函数 tmu_interrupt_flag_clear	889
表 3-1283. TRIGSEL 寄存器	890
表 3-1284. TRIGSEL 库函数	891
表 3-1285. 枚举类型 trigsel_source_enum	891

表 3-1286. 枚举类型 trngsel_periph_enum	896
表 3-1287. 函数 trngsel_init.....	898
表 3-1288. 函数 trngsel_init.....	898
表 3-1289. 函数 trngsel_trigger_source_get	899
表 3-1290. 函数 trngsel_trigger_source_set	900
表 3-1291. 函数 trngsel_trigger_lock_get.....	900
表 3-1292. TRNG 寄存器	901
表 3-1293. TRNG 库函数	901
表 3-1294. 枚举 trng_inmod_enum.....	902
表 3-1295. 枚举 trng_outmod_enum	902
表 3-1296. 枚举 trng_modsel_enum.....	902
表 3-1297. 枚举 trng_nist_seed_enum.....	902
表 3-1298. 枚举 trng_flag_enum	902
表 3-1299. 枚举 trng_int_flag_enum.....	903
表 3-1300. 函数 trng_deinit	903
表 3-1301. 函数 trng_enable	903
表 3-1302. 函数 trng_disable	904
表 3-1303. 函数 trng_lock.....	904
表 3-1304. 函数 trng_mode_config.....	905
表 3-1305. 函数 trng_nist_seed_config	906
表 3-1306. 函数 trng_postprocessing_enable	906
表 3-1307. 函数 trng_postprocessing_disable	907
表 3-1308. 函数 trng_conditioning_enable.....	908
表 3-1309. 函数 trng_conditioning_disable	909
表 3-1310. 函数 trng_conditioning_input_bitwidth.....	910
表 3-1311. 函数 trng_conditioning_output_bitwidth	910
表 3-1312. 函数 trng_replace_test_enable	911
表 3-1313. 函数 trng_replace_test_disable	912
表 3-1314. 函数 trng_hash_init_enable.....	912
表 3-1315. 函数 trng_hash_init_disable.....	913
表 3-1316. 函数 trng_powermode_config.....	914
表 3-1317. 函数 trng_clockdiv_config.....	914
表 3-1318. 函数 trng_clockerror_detection_enable.....	915
表 3-1319. 函数 trng_clockerror_detection_disable.....	916
表 3-1320. 函数 trng_get_true_random_data.....	916
表 3-1321. 函数 trng_conditioning_reset_enable	917
表 3-1322. 函数 trng_conditioning_reset_disable	918
表 3-1323. 函数 trng_conditioning_algo_config	918
表 3-1324. 函数 trng_health_tests_config.....	919
表 3-1325. 函数 trng_flag_get.....	920
表 3-1326. 函数 trng_interrupt_enable.....	921
表 3-1327. 函数 trng_interrupt_disable.....	921
表 3-1328. 函数 trng_interrupt_flag_get	922

表 3-1329. 函数 trng_interrupt_flag_clear	922
表 3-1330. USART 寄存器	924
表 3-1331. USART 库函数	924
表 3-1332. 枚举类型 usart_flag_enum	927
表 3-1333. 枚举类型 usart_interrupt_flag_enum	928
表 3-1334. 枚举类型 usart_interrupt_enum	928
表 3-1335. 枚举类型 usart_invert_enum	929
表 3-1336. 函数 usart_deinit	929
表 3-1337. 函数 usart_baudrate_set	930
表 3-1338. 函数 usart_parity_config	930
表 3-1339. 函数 usart_word_length_set	931
表 3-1340. 函数 usart_stop_bit_set	932
表 3-1341. 函数 usart_enable	932
表 3-1342. 函数 usart_disable	933
表 3-1343. 函数 usart_transmit_config	933
表 3-1344. 函数 usart_receive_config	934
表 3-1345. 函数 usart_data_first_config	935
表 3-1346. 函数 usart_invert_config	935
表 3-1347. 函数 usart_overrun_enable	936
表 3-1348. 函数 usart_overrun_disable	937
表 3-1349. 函数 usart_oversample_config	937
表 3-1350. 函数 usart_sample_bit_config	938
表 3-1351. 函数 usart_receiver_timeout_enable	938
表 3-1352. 函数 usart_receiver_timeout_disable	939
表 3-1353. 函数 usart_receiver_timeout_threshold_config	939
表 3-1354. 函数 usart_data_transmit	940
表 3-1355. 函数 usart_data_receive	941
表 3-1356. 函数 usart_command_enable	941
表 3-1357. 函数 usart_address_0_match_mode_enable	942
表 3-1358. 函数 usart_address_0_match_mode_disable	943
表 3-1359. 函数 usart_address_1_match_mode_enable	943
表 3-1360. 函数 usart_address_1_match_mode_disable	944
表 3-1361. 函数 usart_address_0_config	944
表 3-1362. 函数 usart_address_1_config	945
表 3-1363. 函数 usart_address_0_detection_mode_config	945
表 3-1364. 函数 usart_address_1_detection_mode_config	946
表 3-1365. 函数 usart_mute_mode_enable	947
表 3-1366. 函数 usart_mute_mode_disable	947
表 3-1367. 函数 usart_mute_mode_wakeup_config	948
表 3-1368. 函数 usart_lin_mode_enable	948
表 3-1369. 函数 usart_lin_mode_disable	949
表 3-1370. 函数 usart_lin_break_detection_length_config	949
表 3-1371. 函数 usart_halfduplex_enable	950

表 3-1372. 函数 usart_halfduplex_disable	951
表 3-1373. 函数 usart_clock_enable.....	951
表 3-1374. 函数 usart_clock_disable.....	952
表 3-1375. 函数 usart_synchronous_clock_config	952
表 3-1376. 函数 usart_guard_time_config.....	953
表 3-1377. 函数 usart_smartcard_mode_enable	954
表 3-1378. 函数 usart_smartcard_mode_disable	954
表 3-1379. 函数 usart_smartcard_mode_nack_enable	955
表 3-1380. 函数 usart_smartcard_mode_nack_disable	955
表 3-1381. 函数 usart_smartcard_mode_early_nack_enable	956
表 3-1382. 函数 usart_smartcard_mode_early_nack_disable	956
表 3-1383. 函数 usart_smartcard_autoretry_config	957
表 3-1384. 函数 usart_block_length_config.....	957
表 3-1385. 函数 usart_irda_mode_enable	958
表 3-1386. 函数 usart_irda_mode_disable	958
表 3-1387. 函数 usart_prescaler_config	959
表 3-1388. 函数 usart_irda_lowpower_config.....	959
表 3-1389. 函数 usart_hardware_flow_rts_config.....	960
表 3-1390. 函数 usart_hardware_flow_cts_config	961
表 3-1391. 函数 usart_hardware_flow_coherence_config	961
表 3-1392. 函数 usart_rs485_driver_enable	962
表 3-1393. 函数 usart_rs485_driver_disable.....	963
表 3-1394. 函数 usart_driver_asserttime_config.....	963
表 3-1395. 函数 usart_driver_deasserttime_config	964
表 3-1396. 函数 usart_depolarity_config	964
表 3-1397. 函数 usart_dma_receive_config	965
表 3-1398. 函数 usart_dma_transmit_config	966
表 3-1399. 函数 usart_reception_error_dma_disable	966
表 3-1400. 函数 usart_reception_error_dma_enable	967
表 3-1401. 函数 usart_wakeup_enable	967
表 3-1402. 函数 usart_wakeup_disable	968
表 3-1403. 函数 usart_wakeup_mode_config.....	968
表 3-1404. 函数 usart_fifo_enable	969
表 3-1405. 函数 usart_fifo_disable	970
表 3-1406. 函数 usart_transmit_fifo_threshold_config.....	970
表 3-1407. 函数 usart_receive_fifo_threshold_config.....	971
表 3-1408. 函数 usart_receive_fifo_counter_number	972
表 3-1409. 函数 usart_flag_get.....	972
表 3-1410. 函数 usart_flag_clear.....	973
表 3-1411. 函数 usart_interrupt_enable	974
表 3-1412. 函数 usart_interrupt_disable.....	975
表 3-1413. 函数 usart_interrupt_flag_get	976
表 3-1414. 函数 usart_interrupt_flag_clear	976

表 3-1415. VREF 寄存器.....	978
表 3-1416. VREF 库函数.....	978
表 3-1417. 函数 vref_deinit.....	978
表 3-1418. 函数 vref_enable.....	979
表 3-1419. 函数 vref_disable.....	979
表 3-1420. 函数 vref_high_impedance_mode_enable.....	980
表 3-1421. 函数 vref_high_impedance_mode_disable.....	980
表 3-1422. 函数 vref_status_get.....	981
表 3-1423. 函数 vref_voltage_select.....	981
表 3-1424. 函数 vref_calib_value_set.....	982
表 3-1425. 函数 vref_calib_value_get.....	982
表 3-1426. WWDGT 寄存器.....	983
表 3-1427. WWDGT 库函数.....	983
表 3-1428. 函数 wwdgt_deinit.....	984
表 3-1429. 函数 wwdgt_enable.....	984
表 3-1430. 函数 wwdgt_counter_update.....	985
表 3-1431. 函数 wwdgt_config.....	985
表 3-1432. 函数 wwdgt_interrupt_enable.....	986
表 3-1433. 函数 wwdgt_flag_get.....	986
表 3-1434. 函数 wwdgt_flag_clear.....	987
表 4-1. 版本历史.....	988

1. 介绍

本手册介绍了32位基于ARM微控制器GD32G5x3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GD32G5x3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
CLA	可配置逻辑阵列
CMP	比较器
CRC	循环冗余校验计算单元
DAC	数模转换器
DBG	调试模块
DMA	直接存储器访问控制器
DMAMUX	DMA请求多路复用器
EXMC	外部存储器控制器

外设缩写	说明
EXTI	外部中断事件控制器
FAC	滤波算法加速器
FFT	快速傅里叶变换
FMC	闪存控制器
FWDGT	独立看门狗
GPIO	通用和备用输入
HPDF	高性能数字滤波器
I2C	内部集成电路总线接口
LPTIMER	低功耗定时器
MISC	嵌套中断向量列表控制器
PMU	电源管理单元
QSPI	四线SPI接口
RCU	复位和时钟单元
RTC	实时时钟
HRTIMER	高分辨率定时器
SPI/I2S	串行外设接口/片上音频接口
SYSCFG	系统配置
TIMER	定时器
TMU	三角函数加速器
TRIGSEL	触发选择控制器
TRNG	真随机数生成器
USART	通用同步异步收发器
VREF	VREF
WWDGT	窗口看门狗

1.1.2. 命名规则

固件库遵从以下命名规则：

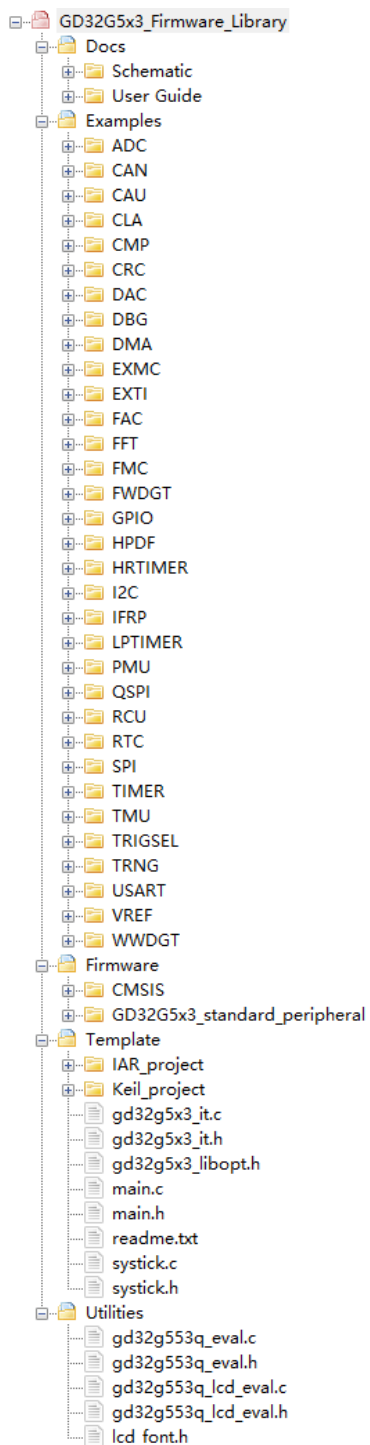
- XXX表示任一外设缩写，例如：ADC。更多缩写相关信息参阅[外设缩写](#)；
- 源文件和头文件命名都以“gd32g5x3_”作为开头，例如：gd32g5x3_adc.h；
- 常量仅被应用于一个文件的，定义于该文件中；被应用于多个文件的，在对应头文件中定义。所有常量都由英文字母大写书写；
- 寄存器作为常量处理。他们的命名都由英文字母大写书写。在大多数情况下，寄存器缩写规范与本用户手册一致；
- 变量名采用全部小写，有多个单词组成的，在单词之间以下划线分隔；
- 外设函数的命名以该外设的缩写加下划线为开头，有多个单词组成的，在单词之间以下划线分隔，所有外设函数都由英文字母小写书写。

2. 固件库概述

2.1. 文件组织结构

GD32G5x3_Firmware_Library，文件组织结构见下图：

图 2-1. GD32G5x3 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；
- **gd32g5x3_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）；
- **gd32g5x3_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）；
- **gd32g5x3_it.h**: 该头文件包含了所有的中断处理程序的原形；
- **systick.c**: 该源文件包含了使用**systick**的精准延时程序；
- **systick.h**: 该头文件包含了使用**systick**的精准延时程序的原形；
- **main.c**: 例程代码注：所有的例程的使用，都不受不同软件开发环境的影响。

2.1.2. Firmware 文件夹

Firmware文件夹包含组成固件库核心的所有子文件夹和文件：

- **CMSIS**子文件夹包含有**Cortex® M33**内核的支持文件、基于**Cortex® M33**内核处理器的启动代码和库引导文件以及基于**GD32G5x3**的全局头文件和系统配置文件；
- **GD32G5x3_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。

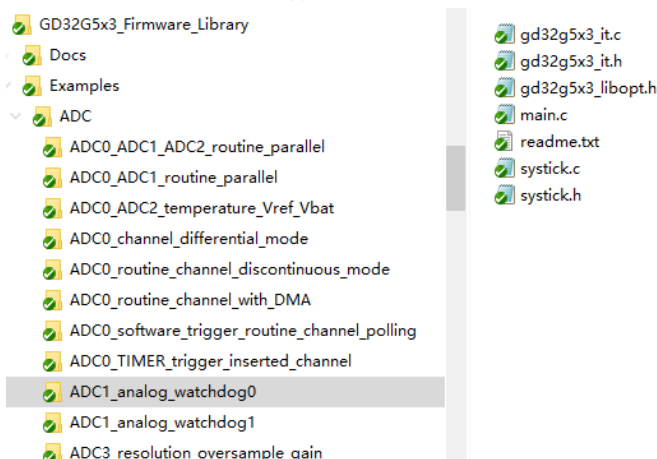
2.1.3. Template 文件夹

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：

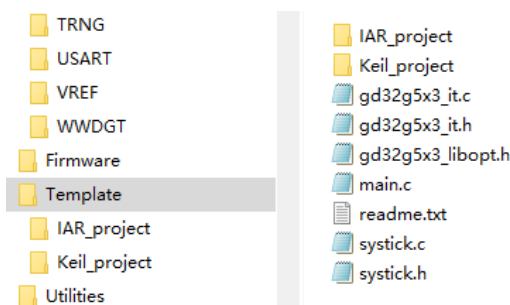
图 2-2. 选择外设例程文件



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：

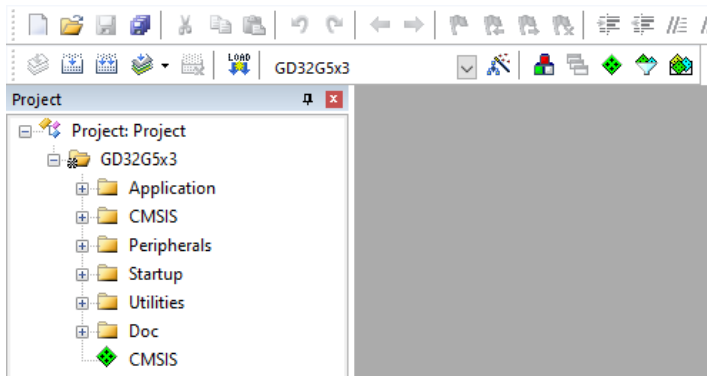
图 2-3. 拷贝外设例程文件



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：

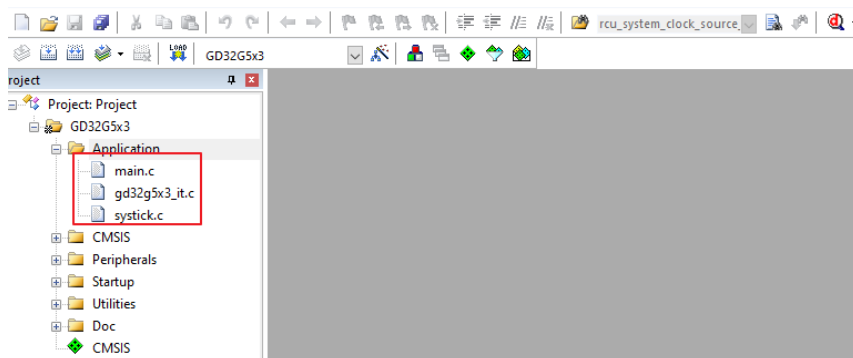
图 2-4. 打开工程文件



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：

- gd32g553q_eval.h文件是运行固件库例程所需关于评估板的头文件；
- gd32g553q_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
gd32g5x3_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。

文件名	描述
main.c	主函数体示例。
gd32g5x3_it.h	头文件，包含所有中断处理函数原形。
gd32g5x3_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。
gd32g5x3_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。
gd32g5x3_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	常规序列寄存器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	常规数据寄存器
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_GAINCFG	ADC增益配置寄存器
ADC_SSTAT	摘要状态寄存器
ADC_SYNCCTL	同步控制寄存器
ADC_SYNCDATA	同步常规数据寄存器

3.2.2. 外设库函数说明

ADC库函数列表如下表所示：

表 3-3. ADC 库函数

库函数名称	库函数描述
adc_deinit	复位ADC外设
adc_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_gain_mode_enable	增益模式使能
adc_gain_mode_disable	增益模式失能
adc_gain_factor_set	增益因子配置
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，在每个常规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	配置常规序列或注入序列的长度

库函数名称	库函数描述
adc_routine_channel_config	配置ADC常规通道
adc_inserted_channel_config	配置ADC注入通道
adc_inserted_channel_offset_config	配置ADC注入通道数据偏移值
adc_inserted_channel_mean_value_mode_enable	注入通道均值模式使能
adc_inserted_channel_mean_value_mode_disable	注入通道均值模式失能
adc_inserted_channel_mean_value_mode_config	注入通道均值模式配置
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	配置常规序列转换结束模式
adc_routine_data_read	读ADC常规序列数据寄存器
adc_inserted_data_read	读ADC注入序列数据寄存器
adc_watchdog0_single_channel_enable	配置ADC模拟看门狗0单通道有效
adc_watchdog0_sequence_channel_enable	配置ADC模拟看门狗0在序列有效
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_en	根据SYNCDMA位来产生DMA请求

库函数名称	库函数描述
able	
adc_sync_dma_request_after_last_di sable	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_routine_data_r ead	ADC同步模式主ADC常规数据寄存器读取
adc_sync_slave_adc_routine_data_re ad	ADC同步模式从ADC常规数据寄存器读取

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=0,1,2,3)	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=0,1,2,3)	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	<code>adc_special_function_config</code>
函数原形	<code>void adc_special_function_config(uint32_t adc_periph, uint32_t function, ControlStatus newvalue);</code>
功能描述	使能或禁能ADC特殊功能
先决条件	-
被调用函数	-
输入参数{in}	
<code>adc_periph</code>	ADC外设
<code>ADCx(x=0,1,2,3)</code>	ADC外设选择
输入参数{in}	
<code>function</code>	功能配置
<code>ADC_SCAN_MODE</code>	扫描模式选择
<code>ADC_INSERTED_CHANNEL_AUTO</code>	注入序列自动转换
<code>ADC_CONTINUOUS_MODE</code>	连续模式选择
输入参数{in}	
<code>newvalue</code>	功能使能禁能
<code>ENABLE</code>	使能
<code>DISABLE</code>	禁能
输出参数{out}	
-	-
返回值	
-	-

例如：

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

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

函数 **adc_data_alignment_config**

函数adc_alignment_config描述见下表:

表 3-7. 函数 **adc_data_alignment_config**

函数名称	adc_data_alignment_config
函数原形	void adc_data_alignment_config(uint32_t adc_periph, uint32_t data_alignment);
功能描述	配置ADC数据对齐方式
先决条件	-
被调用函数	-
输入参数{in}	
adc_periph	ADC外设
ADCx(x=0,1,2,3)	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=0,1,2,3)	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	<code>adc_disable</code>
函数原形	<code>void adc_disable(uint32_t adc_periph);</code>
功能描述	禁能ADC外设
先决条件	-
被调用函数	-
输入参数{in}	
<code>adc_periph</code>	ADC外设
<code>ADCx(x=0,1,2,3)</code>	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	<code>adc_calibration_mode_config</code>
函数原形	<code>void adc_calibration_mode_config(uint32_t adc_periph, uint32_t clb_mode);</code>
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}	
<code>adc_periph</code>	ADC外设
<code>ADCx(x=0,1)</code>	ADC外设选择

输入参数{in}	
clb_mode	校准模式
<i>ADC_CALIBRATION_OFFSET_MISMATCH</i>	偏移、不匹配模式
<i>ADC_CALIBRATION_OFFSET</i>	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
<i>ADCx(x=0,1)</i>	ADC外设选择
输入参数{in}	
clb_num	校准次数
<i>ADC_CALIBRATION_NUM1</i>	校准1次
<i>ADC_CALIBRATION_NUM2</i>	校准2次
<i>ADC_CALIBRATION_NUM4</i>	校准4次
<i>ADC_CALIBRATION_NUM8</i>	校准8次
<i>ADC_CALIBRATION_NUM16</i>	校准16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	<code>adc_calibration_enable</code>
函数原形	<code>void adc_calibration_enable(uint32_t adc_periph);</code>
功能描述	ADC校准复位
先决条件	-
被调用函数	-
输入参数{in}	
<code>adc_periph</code>	ADC外设
<code>ADCx(x=0,1,2,3)</code>	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	<code>adc_resolution_config</code>
函数原形	<code>void adc_resolution_config(uint32_t adc_periph , uint32_t resolution);</code>
功能描述	配置ADC分辨率
先决条件	-
被调用函数	-
输入参数{in}	
<code>adc_periph</code>	ADC外设
<code>ADCx(x=0,1,2,3)</code>	ADC外设选择
输入参数{in}	
<code>resolution</code>	ADC分辨率
<code>ADC_RESOLUTION_12B</code>	12位分辨率

ADC_RESOLUTION_10B	10位分辨率
ADC_RESOLUTION_8B	8位分辨率
ADC_RESOLUTION_6B	6位分辨率
输出参数{out}	
-	-
返回值	
-	-

Example:

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

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

函数 adc_internal_channel_config

函数adc_internal_channel_config描述见下表:

表 3-14. 函数 adc_resolution_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_gain_mode_enable`

函数`adc_gain_mode_enable`描述见下表:

表 3-15. 函数 `adc_gain_mode_enable`

函数名称	<code>adc_gain_mode_enable</code>
函数原形	<code>void adc_gain_mode_enable (uint32_t adc_periph);</code>
功能描述	ADC增益模式使能
先决条件	-
被调用函数	-
输入参数{in}	
<code>adc_periph</code>	ADC外设
<code>ADCx(x=0,1,2,3)</code>	ADC外设选择
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* enable ADC0 gain mode */
```

```
adc_gain_mode_enable (ADC0);
```

函数 `adc_gain_mode_disable`

函数`adc_gain_mode_disable`描述见下表:

表 3-16. 函数 `adc_gain_mode_disable`

函数名称	<code>adc_gain_mode_disable</code>
函数原形	<code>void adc_gain_mode_disable(uint32_t adc_periph);</code>
功能描述	ADC增益模式禁能
先决条件	-
被调用函数	-
输入参数{in}	
<code>adc_periph</code>	ADC外设
<code>ADCx(x=0,1,2,3)</code>	ADC外设选择

输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* disable ADC0 gain mode */
adc_gain_mode_disable(ADC0);
```

函数 `adc_gain_factor_set`

函数`adc_gain_factor_set`描述见下表:

表 3-17. 函数 `adc_gain_factor_set`

函数名称	<code>adc_gain_factor_set</code>
函数原形	<code>void adc_gain_factor_set(uint32_t adc_periph, uint32_t factor);</code>
功能描述	配置ADC增益模式系数
先决条件	-
被调用函数	-
输入参数{in}	
adc_periph	ADC外设
<code>ADCx(x=0,1,2,3)</code>	ADC外设选择
输入参数{in}	
factor	系数, 0..0x3FFF
输出参数{out}	
-	-
返回值	
-	-

Example:

```
/* configure ADC0 gain factor */
adc_gain_factor_set (ADC0,0x1);
```

函数 `adc_dma_mode_enable`

函数`adc_dma_mode_enable`描述见下表:

表 3-18. 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}	
adc_periph	ADC外设
<i>ADCx(x=0,1,2,3)</i>	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 3-19. 函数 `adc_dma_mode_disable`

函数名称	adc_dma_mode_disable
函数原形	void adc_dma_mode_disable(uint32_t adc_periph);
功能描述	ADC DMA请求禁能
先决条件	-
被调用函数	-
输入参数{in}	
adc_periph	ADC外设
<i>ADCx(x=0,1,2,3)</i>	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 3-20. 函数 `adc_dma_request_after_last_enable`

函数名称	adc_dma_request_after_last_enable
函数原形	void adc_dma_request_after_last_enable(uint32_t adc_periph);
功能描述	当DMA=1时，在每个常规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
ADCx(x=0,1,2,3)	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 routine 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 3-21. 函数 adc_dma_request_after_last_disable

函数名称	adc_dma_request_after_last_disable
函数原形	void adc_dma_request_after_last_disable(uint32_t adc_periph);
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
ADCx(x=0,1,2,3)	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 3-22. 函数 adc_hpdf_mode_enable

函数名称	adc_hpdf_mode_enable
------	----------------------

函数原形	void adc_hpdf_mode_enable(uint32_t adc_periph);
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
ADCx(x=0,1,2,3)	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 3-23. 函数 adc_hpdf_mode_disable

函数名称	adc_hpdf_mode_disable
函数原形	void adc_hpdf_mode_disable(uint32_t adc_periph);
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=0,1,2,3)	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 3-24. 函数 `adc_discontinuous_mode_config`

函数名称	<code>adc_discontinuous_mode_config</code>
函数原形	<code>void adc_discontinuous_mode_config(uint32_t adc_periph, uint8_t adc_sequence, uint32_t length);</code>
功能描述	配置ADC中断模式
先决条件	-
被调用函数	-
输入参数{in}	
<code>adc_periph</code>	ADC外设
<code>ADCx(x=0,1,2,3)</code>	ADC外设选择
输入参数{in}	
<code>adc_sequence</code>	序列选择
<code>ADC_ROUTINE_CHANNEL</code>	常规序列
<code>ADC_INSERTED_CHANNEL</code>	注入序列
<code>ADC_CHANNEL_DISCON_DISABLE</code>	常规序列和注入序列中断模式禁能
输入参数{in}	
<code>length</code>	中断模式下的转换数目，常规序列取值为1..8，注入序列取值无意义
输出参数{out}	
-	-
返回值	
-	-

例如：

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

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

函数 `adc_channel_length_config`

函数`adc_channel_length_config`描述见下表：

表 3-25. 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_sequence, uint32_t length);</code>
功能描述	配置常规通道序列或注入通道序列的长度
先决条件	-
被调用函数	-
输入参数{in}	
<code>adc_periph</code>	ADC外设
<code>ADCx(x=0,1,2,3)</code>	ADC外设选择

输入参数{in}	
adc_sequence	通道序列选择
ADC_ROUTINE_CHANNEL	常规序列
ADC_INSERTED_CHANNEL	注入序列
输入参数{in}	
length	通道长度，常规序列为1-16，注入序列为1-4
输出参数{out}	
-	-
返回值	
-	-

例如：

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

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

函数 adc_routine_channel_config

函数adc_routine_channel_config描述见下表：

表 3-26. 函数 adc_routine_channel_config

函数名称	adc_routine_channel_config
函数原形	void adc_routine_channel_config(uint32_t adc_periph, uint8_t rank, uint8_t adc_channel, uint32_t sample_time);
功能描述	配置ADC常规通道序列
先决条件	-
被调用函数	-
输入参数{in}	
adc_periph	ADC外设
ADCx(x=0,1,2,3)	ADC外设选择
输入参数{in}	
rank	常规通道序列，取值范围为0~15
输入参数{in}	
adc_channel	ADC通道选择
ADC_CHANNEL_x(x=0..21)	ADC 通道x
输入参数{in}	
sample_time	采样时间，0..638
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* configure ADC0 routine channel */
```

```
adc_routine_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 3-27. 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注入通道序列
先决条件	-
被调用函数	-
输入参数{in}	
adc_periph	ADC外设
ADCx(x=0,1,2,3)	ADC外设选择
输入参数{in}	
rank	注入通道序列，取值范围为0~3
输入参数{in}	
adc_channel	ADC通道选择
ADC_CHANNEL_x(x=0..21)	ADC 通道x
输入参数{in}	
sample_time	采样时间，0..638
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 3-28. 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=0,1,2,3)	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_inserted_channel_mean_value_mode_enable

函数adc_inserted_channel_mean_value_mode_enable描述见下表:

表 3-29. 函数 adc_inserted_channel_mean_value_mode_enable

函数名称	adc_inserted_channel_mean_value_mode_enable
函数原形	void adc_inserted_channel_mean_value_mode_enable (uint32_t adc_periph);
功能描述	注入通道均值模式使能
先决条件	-
被调用函数	-
输入参数{in}	
adc_periph	ADC外设
ADCx(x=0,1,2,3)	ADC外设选择
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* enable ADC0 mean value mode for inserted channel */
```

```
adc_inserted_channel_mean_value_mode_enable (ADC0);
```

函数 adc_inserted_channel_mean_value_mode_disable

函数adc_inserted_channel_mean_value_mode_disable描述见下表：

表 3-30. 函数 adc_inserted_channel_mean_value_mode_disable

函数名称	adc_inserted_channel_mean_value_mode_disable
函数原形	void adc_inserted_channel_mean_value_mode_disable (uint32_t adc_periph);
功能描述	注入通道均值模式失能
先决条件	-
被调用函数	-
输入参数{in}	
adc_periph	ADC外设
ADCx(x=0,1,2,3)	ADC外设选择
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* disable ADC0 mean value mode for inserted channel */
```

```
adc_inserted_channel_mean_value_mode_disable (ADC0);
```

函数 adc_inserted_channel_mean_value_mode_config

函数adc_inserted_channel_mean_value_mode_config描述见下表：

表 3-31. 函数 adc_inserted_channel_mean_value_mode_config

函数名称	adc_inserted_channel_mean_value_mode_config
函数原形	void adc_inserted_channel_mean_value_mode_config (uint32_t adc_periph, uint32_t mode);
功能描述	配置ADC均值模式
先决条件	-
被调用函数	-
输入参数{in}	
adc_periph	ADC外设
ADCx(x=0,1,2,3)	ADC外设选择
输入参数{in}	
mode	均值模式选择
ADC_MEAN_VALUE_MODE_4X	4x模式
ADC_MEAN_VALUE_MODE_8X	8x模式
输出参数{out}	

-	-
返回值	
-	-

Example:

```
/* configure ADC0 mean value mode for inserted channel */
```

```
adc_inserted_channel_mean_value_mode_config(ADC0, ADC_MEAN_VALUE_MODE_4X);
```

函数 `adc_channel_differential_mode_config`

函数 `adc_channel_differential_mode_config` 描述见下表:

表 3-32. 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}	
<code>adc_periph</code>	ADC外设
<code>ADCx(x=0..2)</code>	ADC外设选择
输入参数{in}	
<code>inserted_channel</code>	通道使用差分模式
<code>ADC_DIFFERENTIAL_MODE_CHANNEL_EL_x (x=0..21),</code> <code>ADC_DIFFERENTIAL_MODE_CHANNEL_EL_ALL</code>	ADC通道差分模式
输入参数{in}	
<code>newvalue</code>	通道使能/禁能
<code>ENABLE</code>	使能
<code>DISABLE</code>	禁能
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* 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 3-33. 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_sequence, 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}	
<code>adc_periph</code>	ADC外设
<code>ADCx(x=0,1,2,3)</code>	ADC外设选择
输入参数{in}	
<code>adc_sequence</code>	序列选择
<code>ADC_ROUTINE_CHANNEL</code>	常规序列
<code>ADC_INSERTED_CHANNEL</code>	注入序列
输入参数{in}	
<code>trigger_mode</code>	外部触发模式
<code>EXTERNAL_TRIGGER_DISABLE</code>	外部触发禁能
<code>EXTERNAL_TRIGGER_RISING</code>	上升沿触发
<code>EXTERNAL_TRIGGER_FALLING</code>	下降沿触发
<code>EXTERNAL_TRIGGER_RISING_FALLING</code>	双边沿触发
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* enable ADC0 inserted sequence external trigger */
adc_external_trigger_config(ADC0,ADC_INSERTED_CHANNEL,
EXTERNAL_TRIGGER_RISING);
```

函数 adc_software_trigger_enable

函数adc_software_trigger_enable描述见下表：

表 3-34. 函数 adc_software_trigger_enable

函数名称	adc_software_trigger_enable
函数原形	void adc_software_trigger_enable(uint32_t adc_periph, uint8_t adc_sequence);
功能描述	ADC软件触发使能
先决条件	-
被调用函数	-
输入参数{in}	
adc_periph	ADC外设
ADCx(x=0,1,2,3)	ADC外设选择
输入参数{in}	
adc_sequence	通道序列
ADC_ROUTINE_CHANNEL	常规序列
ADC_INSERTED_CHANNEL	注入序列
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* enable ADC0 routine sequence software trigger */
```

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

函数 adc_end_of_conversion_config

函数adc_end_of_conversion_config描述见下表：

表 3-35. 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=0,1,2,3)	ADC外设选择
输入参数{in}	

end_selection	转换结束模式选择
ADC_EOC_SET_SEQUENCE	只有在常规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	在每个常规转换结束时，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 routine conversion, the EOC bit is set. */
```

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

函数 adc_routine_data_read

函数adc_inserted_routine_data_read描述见下表：

表 3-36. 函数 adc_routine_data_read

函数名称	adc_routine_data_read
函数原形	uint32_t adc_routine_data_read(uint32_t adc_periph);
功能描述	读ADC常规数据寄存器
先决条件	-
被调用函数	-
输入参数{in}	
adc_periph	ADC外设
ADCx(x=0,1,2,3)	ADC外设选择
输出参数{out}	
-	-
返回值	
uint32_t	ADC转换值

例如：

```
/* read ADC0 routine data register */
```

```
uint32_t adc_value = 0;
```

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

函数 adc_inserted_data_read

函数adc_inserted_routine_data_read描述见下表：

表 3-37. 函数 adc_inserted_data_read

函数名称	adc_inserted_data_read
-------------	------------------------

函数原形	uint32_t adc_inserted_data_read(uint32_t adc_periph, uint8_t inserted_channel);
功能描述	读ADC注入数据寄存器
先决条件	-
被调用函数	-
输入参数{in}	
adc_periph	ADC外设
ADCx(x=0,1,2,3)	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

例如:

```
/* read ADC0 inserted 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 3-38. 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=0,1,2,3)	ADC外设选择
输入参数{in}	
adc_channel	选择ADC通道
ADC_CHANNEL_x (x=0..21)	ADC通道x
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_sequence_channel_enable`

函数`adc_watchdog0_sequence_channel_enable`描述见下表：

表 3-39. 函数 `adc_watchdog0_sequence_channel_enable`

函数名称	<code>adc_watchdog0_sequence_channel_enable</code>
函数原形	<code>void adc_watchdog0_sequence_channel_enable (uint32_t adc_periph, uint8_t adc_sequence);</code>
功能描述	配置ADC模拟看门狗0在序列有效
先决条件	-
被调用函数	-
输入参数{in}	
<code>adc_periph</code>	ADC外设
<code>ADCx(x=0,1,2,3)</code>	ADC外设选择
输入参数{in}	
<code>adc_sequence</code>	序列使用模拟看门狗
<code>ADC_ROUTINE_CHANNEL</code>	常规序列
<code>ADC_INSERTED_CHANNEL</code>	注入序列
<code>ADC_ROUTINE_INSERTED_CHANNEL</code>	常规和注入序列
输出参数{out}	
-	-
返回值	
-	-

例如：

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

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

函数 `adc_watchdog0_disable`

函数`adc_watchdog0_disable`描述见下表：

表 3-40. 函数 `adc_watchdog0_disable`

函数名称	<code>adc_watchdog0_disable</code>
函数原形	<code>void adc_watchdog0_disable(uint32_t adc_periph);</code>
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}	
<code>adc_periph</code>	ADC外设
<code>ADCx(x=0,1,2,3)</code>	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 3-41. 函数 `adc_watchdog1_channel_config`

函数名称	<code>adc_watchdog1_channel_config</code>
函数原形	<code>void adc_watchdog1_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1单通道有效
先决条件	-
被调用函数	-
输入参数{in}	
<code>adc_periph</code>	ADC外设
<code>ADCx(x=0,1,2,3)</code>	ADC外设选择
输入参数{in}	
<code>selection_channel</code>	选择ADC通道
<code>ADC_AWD1_2_SELECTION_CHANNEL_x (x=0..21), ADC_AWD1_2_SELECTION_CHANNEL_ALL</code>	ADC通道模拟看门狗1选择(x=0..21)
输入参数{in}	
<code>newvalue</code>	使能/禁能控制
<code>ENABLE</code>	使能

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

例如:

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

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

函数 `adc_watchdog2_channel_config`

函数`adc_watchdog2_channel_config`描述见下表:

表 3-42. 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
<i>ADCx</i> (<i>x</i> =0,1,2,3)	ADC外设选择
输入参数{in}	
selection_channel	选择ADC通道
<i>ADC_AWD1_2_SELECTION_CHANNEL_x</i> (<i>x</i> =0..20), <i>ADC_AWD1_2_SELECTION_CHANNEL_ALL</i>	ADC通道模拟看门狗2选择(<i>x</i> =0..21)
输入参数{in}	
newvalue	使能/禁能控制
<i>ENABLE</i>	使能
<i>DISABLE</i>	禁能
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* configure ADC analog watchdog 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 3-43. 函数 adc_watchdog1_disable

函数名称	adc_watchdog1_disable
函数原形	void adc_watchdog1_disable(uint32_t adc_periph);
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}	
adc_periph	ADC外设
ADCx(x=0,1,2,3)	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 3-44. 函数 adc_watchdog2_disable

函数名称	adc_watchdog2_disable
函数原形	void adc_watchdog2_disable(uint32_t adc_periph);
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}	
adc_periph	ADC外设
ADCx(x=0,1,2,3)	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 3-45. 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}	
adc_periph	ADC外设
<code>ADCx(x=0,1,2,3)</code>	ADC外设选择
输入参数{in}	
low_threshold	模拟看门狗0低阈值, 范围为0..0x3FFFFFF
输入参数{in}	
high_threshold	模拟看门狗0高阈值, 范围为0..0x3FFFFFF
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 3-46. 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}	

adc_periph	ADC外设
<i>ADCx(x=0,1,2,3)</i>	ADC外设选择
输入参数{in}	
low_threshold	模拟看门狗1低阈值，范围为0..0x3FFFFFF
输入参数{in}	
high_threshold	模拟看门狗1高阈值，范围为0..0x3FFFFFF
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 3-47. 函数 **adc_watchdog2_threshold_config**

函数名称	adc_watchdog2_threshold_config
函数原形	void adc_watchdog2_threshold_config(uint32_t adc_periph, uint32_t low_threshold, uint32_t high_threshold);
功能描述	配置ADC模拟看门狗2阈值
先决条件	-
被调用函数	-
输入参数{in}	
adc_periph	ADC外设
<i>ADCx(x=0,1,2,3)</i>	ADC外设选择
输入参数{in}	
low_threshold	模拟看门狗2低阈值，范围为0..0x3FFFFFF
输入参数{in}	
high_threshold	模拟看门狗2高阈值，范围为0..0x3FFFFFF
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 3-48. 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
<code>ADCx(x=0,1,2,3)</code>	ADC外设选择
输入参数{in}	
mode	ADC过采样触发模式
<code>ADC_OVERSAMPLING_ALL_CONVERSIONS</code>	在一个触发之后，对一个通道连续进行过采样转换
<code>ADC_OVERSAMPLING_ONE_CONVERSION</code>	在一个触发之后，对一个通道只进行一次过采样转换
输入参数{in}	
shift	ADC过滤采样移位
<code>ADC_OVERSAMPLING_SHIFT_NONE</code>	不移位
<code>ADC_OVERSAMPLING_SHIFT_1B</code>	移1位
<code>ADC_OVERSAMPLING_SHIFT_2B</code>	移2位
<code>ADC_OVERSAMPLING_SHIFT_3B</code>	移3位
<code>ADC_OVERSAMPLING_SHIFT_4B</code>	移4位
<code>ADC_OVERSAMPLING_SHIFT_5B</code>	移5位
<code>ADC_OVERSAMPLING_SHIFT_6B</code>	移6位
<code>ADC_OVERSAMPLING_SHIFT_7B</code>	移7位
<code>ADC_OVERSAMPLING_SHIFT_8B</code>	移8位
<code>ADC_OVERSAMPLING_SHIFT_9B</code>	移9位

NG_SHIFT_9B	
ADC_OVERSAMPLING_SHIFT_10B	移10位
ADC_OVERSAMPLING_SHIFT_11B	移11位
输入参数{in}	
ratio	ADC过采样率, 0..1023对应于1x~1024x
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 3-49. 函数 adc_oversample_mode_enable

函数名称	adc_oversample_mode_enable
函数原形	void adc_oversample_mode_enable(uint32_t adc_periph);
功能描述	使能ADC过采样
先决条件	-
被调用函数	-
输入参数{in}	
adc_periph	ADC外设
ADCx(x=0,1,2,3)	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 3-50. 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}	
<code>adc_periph</code>	ADC外设
<code>ADCx(x=0,1,2,3)</code>	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 3-51. 函数 `adc_flag_get`

函数名称	<code>adc_flag_get</code>
函数原形	<code>FlagStatus adc_flag_get(uint32_t adc_periph , uint32_t flag);</code>
功能描述	获取ADC标志位
先决条件	-
被调用函数	-
输入参数{in}	
<code>adc_periph</code>	ADC外设
<code>ADCx(x=0,1,2,3)</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>	序列转换结束标志位
<code>ADC_FLAG_EOIC</code>	注入序列转换结束标志位
<code>ADC_FLAG_STIC</code>	注入序列转换开始标志位
<code>ADC_FLAG_STRC</code>	常规序列转换开始标志位
<code>ADC_FLAG_ROVF</code>	常规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	
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 3-52. 函数 adc_flag_clear

函数名称	adc_flag_clear
函数原形	void adc_flag_clear(uint32_t adc_periph, uint32_t flag);
功能描述	清除ADC标志位
先决条件	-
被调用函数	-
输入参数{in}	
adc_periph	ADC外设
ADCx(x=0,1,2,3)	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	序列转换结束标志位
ADC_FLAG_EOIC	注入序列转换结束标志位
ADC_FLAG_STIC	注入序列转换开始标志位
ADC_FLAG_STRC	常规序列转换开始标志位
ADC_FLAG_ROVF	常规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	
-	-

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 3-53. 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}	
<code>adc_periph</code>	ADC外设
<code>ADCx(x=0,1,2,3)</code>	ADC外设选择
输入参数{in}	
<code>interrupt</code>	ADC中断标志位
<code>ADC_INT_WDE0</code>	模拟看门狗0中断标志位
<code>ADC_INT_EOC</code>	序列转换结束中断标志位
<code>ADC_INT_EOIC</code>	注入序列转换结束中断标志位
<code>ADC_INT_ROVF</code>	常规数据寄存器溢出中断标志位
<code>ADC_INT_WDE1</code>	模拟看门狗1中断标志位
<code>ADC_INT_WDE2</code>	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 3-54. 函数 `adc_interrupt_disable`

函数名称	<code>adc_interrupt_disable</code>
函数原形	<code>void adc_interrupt_disable(uint32_t adc_periph, uint32_t interrupt);</code>
功能描述	ADC中断禁能
先决条件	-
被调用函数	-
输入参数{in}	
<code>adc_periph</code>	ADC外设
<code>ADCx(x=0,1,2,3)</code>	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>	序列转换结束中断标志位
<i>ADC_INT_EOIC</i>	注入序列转换结束中断标志位
<i>ADC_INT_ROVF</i>	常规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 3-55. 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
<i>ADCx(x=0,1,2,3)</i>	ADC外设选择
输入参数{in}	
int_flag	ADC中断标志位
<i>ADC_INT_FLAG_WDE0</i>	模拟看门狗0中断标志位
<i>ADC_INT_FLAG_EOC</i>	序列转换结束中断标志位
<i>ADC_INT_FLAG_EOIC</i>	注入序列转换结束中断标志位
<i>ADC_INT_FLAG_ROVF</i>	常规数据寄存器溢出中断标志位
<i>ADC_INT_FLAG_WDE1</i>	模拟看门狗1中断标志位
<i>ADC_INT_FLAG_WDE2</i>	模拟看门狗2中断标志位

DE2	
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 3-56. 函数 adc_interrupt_flag_clear

函数名称	adc_interrupt_flag_clear
函数原形	void adc_interrupt_flag_clear(uint32_t adc_periph, uint32_t int_flag);
功能描述	清除ADC中断标志位
先决条件	-
被调用函数	-
输入参数{in}	
adc_periph	ADC外设
ADCx(x=0,1,2,3)	ADC外设选择
输入参数{in}	
int_flag	ADC中断标志位
ADC_INT_FLAG_WDE0	模拟看门狗0中断标志位
ADC_INT_FLAG_EOC	序列转换结束中断标志位
ADC_INT_FLAG_EOIC	注入序列转换结束中断标志位
ADC_INT_FLAG_ROVF	常规数据寄存器溢出中断标志位
ADC_INT_FLAG_WDE1	模拟看门狗1中断标志位
ADC_INT_FLAG_WDE2	模拟看门狗2中断标志位
输出参数{out}	
-	-
返回值	
-	-

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 3-57. 函数 `adc_sync_mode_config`

函数名称	<code>adc_sync_mode_config</code>
函数原形	<code>void adc_sync_mode_config(uint32_t sync_mode);</code>
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}	
<code>adc_periph</code>	ADC外设
<code>ADCx(x=0..2)</code>	ADC外设选择
输入参数{in}	
<code>sync_mode</code>	同步模式
<code>ADC_SYNC_MODE_INDEPENDENT</code>	ADC同步模式失能，所有的ADC都独立工作
<code>ADC_DUAL_REGULAR_PARALLEL_INSERTED_PARALLEL</code>	ADC0和ADC1工作在常规并行和注入并行的组合模式。
<code>ADC_DUAL_REGULAR_PARALLEL_INSERTED_ROTATION</code>	ADC0和ADC1工作在常规并行和交替触发的组合模式。
<code>ADC_DUAL_INSERTED_PARALLEL</code>	ADC0和ADC1工作在注入并行模式。
<code>ADC_DUAL_REGULAR_PARALLEL</code>	ADC0和ADC1工作在常规并行模式。
<code>ADC_DUAL_REGULAR_FOLLOW_UP</code>	ADC0和ADC1工作在跟随模式。
<code>ADC_DUAL_INSERTED_TRIGGER_ROTATION</code>	ADC0和ADC1工作在交替触发模式。
<code>ADC_TRIPLE_REGULAR_INSERTED_PARALLEL</code>	ADC0、ADC1和ADC2工作在常规并行和注入并行的组合模式。
<code>ADC_TRIPLE_REGULAR_FOLLOW_UP</code>	ADC0、ADC1和ADC2工作在常规并行和交替触发的组合模式。

TINE_PARALLEL_INSERTED_ROTATION	
ADC_TRIPLE_INSERTED_PARALLEL	ADC0、ADC1和ADC2工作在注入并行模式。
ADC_TRIPLE_ROUTINE_PARALLEL	ADC0、ADC1和ADC2工作在常规并行模式。
ADC_TRIPLE_ROUTINE_FOLLOW_UP	ADC0、ADC1和ADC2工作在跟随模式。
ADC_TRIPLE_INSERTED_TRIGGER_ROTATION	ADC0、ADC1和ADC2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 routine parallel & inserted parallel mode */
```

```
adc_sync_mode_config(ADC_DAUL_REGULAL_PARALLEL_INSERTED_PARALLEL);
```

函数 adc_sync_delay_config

函数adc_sync_delay_config描述见下表：

表 3-58. 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_xCYCLE (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 3-59. 函数 `adc_sync_dma_config`

函数名称	<code>adc_sync_dma_config</code>
函数原形	<code>void adc_sync_dma_config(uint32_t dma_mode);</code>
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}	
<code>dma_mode</code>	DMA模式
<code>ADC_SYNC_DMA_DISABLE</code>	ADC同步DMA失能
<code>ADC_SYNC_DMA_MODE0</code>	ADC同步DMA模式0
<code>ADC_SYNC_DMA_MODE1</code>	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 3-60. 函数 `adc_sync_dma_request_after_last_enable`

函数名称	<code>adc_sync_dma_request_after_last_enable</code>
函数原形	<code>void adc_sync_dma_request_after_last_enable(void);</code>
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 SYNC DMA is not equal to 2'b00, the DMA engine issues requests according to the
SYNC DMA 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 3-61. 函数 `adc_sync_dma_request_after_last_disable`

函数名称	<code>adc_sync_dma_request_after_last_disable</code>
函数原形	<code>void adc_sync_dma_request_after_last_disable(void);</code>
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_routine_data_read`

函数 `adc_sync_master_adc_routine_data_read` 描述见下表:

表 3-62. 函数 `adc_sync_master_adc_routine_data_read`

函数名称	<code>adc_sync_master_adc_routine_data_read</code>
函数原形	<code>uint16_t adc_sync_master_adc_routine_data_read(void);</code>
功能描述	ADC同步模式主ADC常规数据寄存器读取
先决条件	-
被调用函数	-
输入参数{in}	
-	-
输出参数{out}	
-	-

返回值	
uint16_t	返回同步模式的ADC常规通道数据寄存器的值

例如：

```
/* read ADC sync master adc routine data register */
```

```
adc_sync_master_adc_routine_data_read ();
```

函数 adc_sync_slave_adc_routine_data_read

函数adc_sync_slave_adc_routine_data_read描述见下表：

表 3-63. 函数 adc_sync_slave_adc_routine_data_read

函数名称	adc_sync_slave_adc_routine_data_read
函数原形	uint16_t adc_sync_slave_adc_routine_data_read(void);
功能描述	ADC同步模式从ADC常规数据寄存器读取
先决条件	-
被调用函数	-
输入参数{in}	
-	-
输出参数{out}	
-	-
返回值	
uint16_t	返回同步模式的ADC常规通道数据寄存器的值

例如：

```
/* read ADC sync slave adc routine data register */
```

```
adc_sync_slave_adc_routine_data_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 3-64. 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

寄存器名称	寄存器描述
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
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 3-65. 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
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

库函数名称	库函数描述
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
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_bsp_syn_config	配置位采样同步模式
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 3-66. 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 3-67. 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_idr_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

成员名称	功能描述
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
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 3-68. 结构体 can_mailbox_descriptor_struct

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

结构体 can_rx_fifo_struct

表 3-69. 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

成员名称	功能描述
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 3-70. 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 3-71. 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 3-72. 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 3-73. 结构体 can_pn_mode_filter_struct

成员名称	功能描述
rtr	远程帧

ide	扩展帧
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 3-74. 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 3-75. 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 3-76. 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

成员名称	功能描述
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
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	PN 模式匹配唤醒中断

成员名称	功能描述
_MATCH	
CAN_INT_WAKEUP_TIMEOUT	PN 模式超时唤醒中断

枚举类型 `can_flag_enum`

表 3-77. 枚举类型 `can_flag_enum`

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

成员名称	功能描述
UMMARY_FD	
CAN_FLAG_ERR_Overrun	错误溢出状态
CAN_FLAG_STUFFERR_FD	D 帧 BRS 位为隐性位时数据阶段的填充错误状态
CAN_FLAG_FORMERR_FD	FD 帧 BRS 位为隐性位时数据阶段的格式错误状态
CAN_FLAG_CRCERR_FD	FD 帧 BRS 位为隐性位时数据阶段的 CRC 错误状态
CAN_FLAG_BITDOMINANTERR_FD	FD 帧 BRS 位为隐性位时数据阶段的位显性错误状态
CAN_FLAG_BITRECESSIVEERR_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

成员名称	功能描述
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
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 3-78. 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

成员名称	功能描述
B3	
CAN_INT_FLAG_M B4	邮箱 4 成功发送或接收帧中断标志
CAN_INT_FLAG_M B5	邮箱 5 成功发送或接收帧中断标志
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	邮箱 24 成功发送或接收帧中断标志

成员名称	功能描述
B24	
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
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 3-79. 枚举类型 `can_operation_modes_enum`

成员名称	功能描述
CAN_NORMAL_MO DE	正常模式
CAN_MONITOR_M ODE	监听模式
CAN_LOOPBACK_ SILENT_MODE	回环静默模式
CAN_INACTIVE_M ODE	暂停模式
CAN_DISABLE_MO DE	禁能模式
CAN_PN_MODE	虚拟联网模式

枚举类型 `can_struct_type_enum`

表 3-80. 枚举类型 `can_struct_type_enum`

成员名称	功能描述
<code>CAN_INIT_STRUCT</code>	CAN 初始化参数结构体
<code>CAN_FD_INIT_STRUCT</code>	CAN FD 参数结构体
<code>CAN_FIFO_INIT_STRUCT</code>	CAN FIFO 参数结构体
<code>CAN_PN_MODE_INIT_STRUCT</code>	虚拟联网模式参数结构体
<code>CAN_PN_MODE_FILTER_STRUCT</code>	虚拟联网模式过滤器参数结构体
<code>CAN_MDSC_STRUCT</code>	邮箱描述符结构体
<code>CAN_FDES_STRUCT</code>	Rx FIFO 描述符结构体
<code>CAN_FIFO_ID_FILTER_STRUCT</code>	Rx FIFO ID 过滤器结构体
<code>CAN_CRC_STRUCT</code>	CRC 结构体
<code>CAN_ERRCNT_STRUCT</code>	错误计数结构体

枚举类型 `can_error_state_enum`

表 3-81. 枚举类型 `can_error_state_enum`

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

函数 `can_deinit`

函数 `can_deinit` 描述见下表：

表 3-82. 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	rcu_periph_reset_enable / rcu_periph_reset_disable
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：

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

```
can_deinit(CAN0);
```

函数 can_software_reset

函数can_software_reset描述见下表：

表 3-83. 函数 can_software_reset

函数名称	can_software_reset
函数原型	ErrStatus can_software_reset(uint32_t can_periph);
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}	
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;
```

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

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

函数 can_init

函数can_init描述见下表：

表 3-84. 函数 can_init

函数名称	can_init
函数原型	ErrStatus can_init(uint32_t can_periph, can_parameter_struct*

	can_parameter_init);
功能描述	CAN 模块初始化
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_parameter_init	参见 表 3-67. 结构体 can_parameter_struct
输出参数{out}	
-	-
返回值	
ErrStatus	ERROR 或 SUCCESS

例如:

```
can_parameter_struct can_parameter;
```

```
ErrStatus err;
```

```
.....
```

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

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

函数 can_struct_para_init

函数can_struct_para_init描述见下表:

表 3-85. 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 3-80. 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 3-86. 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 3-87. 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 3-79. 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 3-88. 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 3-79. 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 3-89. 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 3-90. 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 3-91. 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 3-70. 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 3-92. 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 3-93. 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 3-94. 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 3-95. 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 3-96. 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 3-97. 函数 can_rx_fifo_config

函数名称	can_rx_fifo_config
函数原型	void can_rx_fifo_config(uint32_t can_periph, can_fifo_parameter_struct *can_fifo_para_init);
功能描述	配置接收 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
输入参数{in}	
can_fifo_para_init	参见结构体 表 3-72. 结构体 can_fifo_parameter_struct
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 3-98. 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 3-71. 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 3-99. 函数 `can_rx_fifo_read`

函数名称	<code>can_rx_fifo_read</code>
函数原型	<code>void can_rx_fifo_read(uint32_t can_periph, can_rx_fifo_struct *rx_fifo);</code>
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}	
<code>can_periph</code>	CAN 外设
<code>CANx(x=0,1,2)</code>	CAN 外设选择
输出参数{out}	
<code>rx_fifo</code>	参见结构体 表 3-69. 结构体 <code>can_rx_fifo_struct</code>
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 3-100. 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 3-101. 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 3-102. 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 3-103. 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 3-68. 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 3-104. 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 3-105. 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 3-106. 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 3-68. 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 3-107. 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 3-108. 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 3-109. 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 3-110. 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 3-111. 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 3-66. 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 3-112. 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 3-66. 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 3-113. 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 3-81. 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 3-114. 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	
can_crc_struct	参见结构体 表 3-75. 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 3-115. 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 3-74. 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 3-116. 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 3-73. 结构体 can_pn_mode_filter_struct
返回值	
filter	参见结构体 表 3-73. 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 3-117. 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 3-118. 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 3-68. 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 3-119. 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 3-120. 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 3-121. 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 3-122. 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 3-123. 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 3-124. 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 3-125. 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 3-126. 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 3-127. 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 3-128. 函数 can_edge_filter_mode_disable

函数名称	can_edge_filter_mode_disable
函数原型	void can_edge_filter_mode_disable(uint32_t can_periph);
功能描述	禁用边沿过滤模式
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：

```
/* 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 3-129. 函数 can_ped_mode_enable

函数名称	can_ped_mode_enable
------	---------------------

函数原型	void can_ped_mode_enable(uint32_t can_periph);
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}	
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 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 3-130. 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 3-131. 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 3-132. 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_TRHEE_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_bsp_syn_config

函数can_bsp_syn_config描述见下表：

表 3-133. 函数 can_bsp_syn_config

函数名称	can_bsp_syn_config
函数原型	void can_bsp_syn_config(uint32_t can_periph, uint32_t sys_mode);
功能描述	配置位采样同步模式
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
输入参数{in}	
sys_mode	位采样同步模式
CAN_BSP_TWO_STAGES_SYN	位采样两阶段同步
CAN_BSP_ONE_STAGE_SYN	位采样单级同步
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* configure bit sampling synchronization */
```

```
can_bsp_syn_config(CAN0, CAN_BSP_TWO_STAGES_SYN);
```

函数 can_flag_get

函数can_flag_get描述见下表：

表 3-134. 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 3-77. 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 3-135. 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 3-77. 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 3-136. 函数 `can_interrupt_enable`

函数名称	<code>can_interrupt_enable</code>
函数原型	<code>void can_interrupt_enable(uint32_t can_periph, can_interrupt_enum interrupt);</code>
功能描述	使能 CAN 中断
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>interrupt</code>	参见枚举类型 表 3-76. 枚举类型 <code>can_interrupt_enum</code>
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 3-137. 函数 `can_interrupt_disable`

函数名称	<code>can_interrupt_disable</code>
函数原型	<code>void can_interrupt_disable(uint32_t can_periph, can_interrupt_enum interrupt);</code>
功能描述	禁能 CAN 中断
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>interrupt</code>	参见枚举类型 表 3-76. 枚举类型 <code>can_interrupt_enum</code>
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 3-138. 函数 can_interrupt_flag_get

函数名称	can_interrupt_flag_get
函数原型	FlagStatus can_interrupt_flag_get(uint32_t can_periph, can_interrupt_flag_enum int_flag);
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}	
can_periph	CAN 外设
CANx(x=0, 1, 2)	CAN 外设选择
输入参数{in}	
int_flag	参见枚举类型 表 3-78. 枚举类型 can_interrupt_flag_enum
输出参数{out}	
-	-
返回值	
FlagStatus	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 3-139. 函数 can_interrupt_flag_clear

函数名称	can_interrupt_flag_clear
函数原型	void can_interrupt_flag_clear(uint32_t can_periph, can_interrupt_flag_enum int_flag);
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}	
can_periph	CAN 外设
CANx(x=0,1,2)	CAN 外设选择
输入参数{in}	
int_flag	参见枚举类型 表 3-78. 枚举类型 can_interrupt_flag_enum
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 3-140. 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 3-141. CAU 库函数

库函数名称	库函数描述
cau_deinit	复位CAU外设
cau_struct_para_init	初始化CAU加密解密结构体
cau_key_struct_para_init	初始化密钥结构体
cau_iv_struct_para_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_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 3-142. 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 3-143. 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 3-144. 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 3-145. 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 3-146. 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	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 CAU peripheral */
```

```
cau_deinit();
```

函数 cau_struct_para_init

函数cau_struct_para_init描述见下表：

表 3-147. 函数 cau_struct_para_init

函数名称	cau_struct_para_init
函数原形	void cau_struct_para_init(cau_parameter_struct *cau_parameter);
功能描述	初始化CAU加密解密结构体
先决条件	-
被调用函数	-
输入参数{in}	
-	-
输出参数{out}	
cau_parameter	CAU加密解密结构体，参考结构体 表3-145. 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

函数cau_key_struct_para_init描述见下表：

表 3-148. 函数 cau_key_struct_para_init

函数名称	cau_key_struct_para_init
函数原形	void cau_key_struct_para_init(cau_key_parameter_struct *key_initpara);
功能描述	初始化密钥结构体
先决条件	-
被调用函数	-
输入参数{in}	
-	-
输出参数{out}	
key_initpara	密钥结构体，参考结构体 表3-142. 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 3-149. 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	矢量结构体，参考结构体 表3-143. 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 3-150. 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	上下文结构体，参考结构体 表3-144. 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 3-151. 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 3-152. 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 3-153. 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
<i>CAU_DMA_INFIFO</i>	DMA用于接收数据
<i>CAU_DMA_OUTFIFO</i>	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 3-154. 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
<i>CAU_DMA_INFIFO</i>	DMA用于接收数据
<i>CAU_DMA_OUTFIFO</i>	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 3-155. 函数 cau_init

函数名称	cau_init
------	----------

函数原形	void cau_init(uint32_t alg_dir, uint32_t algo_mode, uint32_t swapping);
功能描述	初始化CAU
先决条件	-
被调用函数	-
输入参数{in}	
alg_dir	算法方向
CAU_ENCRYPT	加密
CAU_DECRYPT	解密
输入参数{in}	
algo_mode	算法模式选择
CAU_MODE_TDES_ECB	TDES-ECB
CAU_MODE_TDES_CBC	TDES-CBC
CAU_MODE_DES_ECB	DES-ECB
CAU_MODE_DES_CBC	DES-CBC
CAU_MODE_AES_ECB	AES-ECB
CAU_MODE_AES_CBC	AES-CBC
CAU_MODE_AES_CTR	AES-CTR
CAU_MODE_AES_KEY	AES解密密钥准备模式
CAU_MODE_AES_GCM	AES-GCM
CAU_MODE_AES_CCM	AES-CCM
CAU_MODE_AES_CFB	AES-CFB
CAU_MODE_AES_OFB	AES-OFB
输入参数{in}	
swapping	数据交换选择
CAU_SWAPPING_32BIT	无交换
CAU_SWAPPING_16BIT	半字交换
CAU_SWAPPING_8BIT	字节交换
CAU_SWAPPING_1BIT	位交换

输出参数{out}	
-	-
返回值	
-	-

例如：

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

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

函数 cau_aes_keysize_config

函数cau_aes_keysize_config描述见下表：

表 3-156. 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 3-157. 函数 cau_key_init

函数名称	cau_key_init
函数原形	void cau_key_init(cau_key_parameter_struct* key_initpara);
功能描述	初始化密钥参数
先决条件	-
被调用函数	-

输入参数{in}	
key_initpara	密钥参数，参考结构体 表3-142. 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;

key_initpara->key_0_high = 0x12345678;

key_initpara->key_0_low = 0x12345678;

key_initpara->key_1_high = 0x12345678;

key_initpara->key_1_low = 0x12345678;

key_initpara->key_2_high = 0x12345678;

key_initpara->key_3_low = 0x12345678;

key_initpara->key_3_high = 0x12345678;

key_initpara->key_3_low = 0x12345678;

cau_key_init(&key_initpara);
```

函数 cau_iv_init

函数cau_iv_init描述见下表：

表 3-158. 函数 cau_iv_init

函数名称	cau_iv_init
函数原形	void cau_iv_init(cau_iv_parameter_struct* iv_initpara);
功能描述	初始化矢量参数
先决条件	-
被调用函数	-
输入参数{in}	
iv_initpara	矢量参数，参考结构体 表3-143. 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;

iv_initpara->iv_0_high = 0x12345678;

iv_initpara->iv_0_low = 0x12345678;

iv_initpara->iv_1_high = 0x12345678;

iv_initpara->iv_1_low = 0x12345678;

cau_iv_init(&iv_initpara);

```

函数 cau_phase_config

函数cau_phase_config描述见下表:

表 3-159. 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 3-160. 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 3-161. 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 3-162. 函数 cau_data_write

函数名称	cau_data_write
函数原形	void cau_data_write(uint32_t data);

功能描述	将数据写入IN FIFO
先决条件	-
被调用函数	-
输入参数{in}	
data	所写的的数据0 - 0xFFFFFFFF
输出参数{out}	
-	-
返回值	
-	-

例如：

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

```
cau_data_write(0x0010);
```

函数 cau_data_read

函数cau_data_read描述见下表：

表 3-163. 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 = 0U;
```

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

函数 cau_context_save

函数cau_context_save描述见下表：

表 3-164. 函数 cau_context_save

函数名称	cau_context_save
函数原形	void cau_context_save(cau_context_parameter_struct *cau_context,

	cau_key_parameter_struct* key_initpara);
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}	
key_initpara	密钥参数，参考结构体 表3-142. 结构体cau_key_parameter_struct
输出参数{out}	
cau_context	上下文结构体，参考结构体 表3-144. 结构体cau_context_parameter_struct
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 3-165. 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	上下文结构体，参考结构体 表3-144. 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 3-166. 函数 cau_aes_ecb

函数名称	cau_aes_ecb
函数原形	ErrStatus cau_aes_ecb(cau_parameter_struct *cau_parameter, uint8_t *output);
功能描述	ECB模式下使用AES算法加密和解密
先决条件	-
被调用函数	-
输入参数{in}	
cau_parameter	CAU加密和解密参数, 参考结构体 表3-145. 结构体cau_parameter_struct
输出参数{out}	
output	指针指向返回数组
返回值	
ErrStatus	SUCCESS或ERROR

例如:

```
cau_parameter_struct text;

uint8_t encrypt_result[TEXT_SIZE];

ErrStatus status;

.....

cau_struct_para_init(&text);

key_addr = key_select[i];
```

```
key_size = keysize[i];

text.alg_dir    = CAU_ENCRYPT;

text.key        = key_addr;

text.key_size   = key_size;

text.input      = plaintext;

text.in_length  = TEXT_SIZE;

/* encryption in ECB mode */

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

函数 cau_aes_cbc

函数cau_aes_cbc描述见下表：

表 3-167. 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	CAU加密和解密参数，参考结构体 表3-145. 结构体cau_parameter_struct
输出参数{out}	
output	指针指向返回数组
返回值	
ErrStatus	SUCCESS或ERROR

例如：

```
cau_parameter_struct text;

uint8_t encrypt_result[TEXT_SIZE];

ErrStatus status;

.....

cau_struct_para_init(&text);

key_addr = key_select[i];
```

```
key_size = keysize[i];

text.alg_dir    = CAU_ENCRYPT;

text.key        = key_addr;

text.key_size   = key_size;

text.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 3-168. 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	CAU加密和解密参数，参考结构体 表3-145. 结构体cau_parameter_struct
输出参数{out}	
output	指针指向返回数组
返回值	
ErrStatus	SUCCESS或ERROR

例如：

```
cau_parameter_struct text;

uint8_t encrypt_result[TEXT_SIZE];

ErrStatus status;

.....

cau_struct_para_init(&text);

key_addr = key_select[i];

key_size = keysize[i];
```

```

text.alg_dir    = CAU_ENCRYPT;

text.key        = key_addr;

text.key_size   = key_size;

text.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 3-169. 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	CAU加密和解密参数，参考结构体 表3-145. 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;

.....

cau_struct_para_init(&cau_cfb_parameter);

/* 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 3-170. 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	CAU加密和解密参数，参考结构体 表3-145. 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;

.....

cau_struct_para_init(&cau_ofb_parameter);

/* 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 3-171. 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	CAU加密和解密参数，参考结构体 表3-145. 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;

.....

cau_struct_para_init(&cau_gcm_parameter);

/* 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 3-172. 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	CAU加密和解密参数，参考结构体 表3-145. 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;

.....

```



```

cau_struct_para_init(&cau_ccm_parameter);

/* 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 3-173. 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	CAU加密和解密参数, 参考结构体 表3-145. 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;

.....

```

```
cau_struct_para_init(&text);

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 3-174. 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	CAU加密和解密参数, 参考结构体 表3-145. 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;

.....

cau_struct_para_init(&text);

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 3-175. 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	CAU加密和解密参数，参考结构体 表3-145. 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;

.....

cau_struct_para_init(&text);

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 3-176. 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	CAU加密和解密参数，参考结构体 表3-145. 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;

.....

cau_struct_para_init(&text);

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_flag_get

函数cau_flag_get描述见下表：

表 3-177. 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滿标志
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 = RESET;
```

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

函数 cau_interrupt_enable

函数cau_interrupt_enable描述见下表:

表 3-178. 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 3-179. 函数 cau_interrupt_disable

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

例如：

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

```
cau_interrupt_disable(CAU_INT_INFIFO);
```

函数 cau_interrupt_flag_get

函数cau_interrupt_flag_get描述见下表：

表 3-180. 函数 cau_interrupt_flag_get

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

-	-
返回值	
FlagStatus	SET或RESET

例如：

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

```
FlagStatus status = RESET;
```

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

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 3-181. 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 3-182. 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 3-183. 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 3-184. 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 3-183. 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 3-185. 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 3-183. 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 3-186. 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 3-183. 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 3-187. 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 3-183. 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 3-188. 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 3-183. 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 3-189. 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 3-183. 枚举类型 <code>cmp_enum</code>
输入参数{in}	
<code>blanking_source_selection</code>	消隐源选择

CMP_BLANKING_NONE	无消隐
CMP_BLANKING_TIMER0_OC0	选择TIMER0_CH0输出比较信号为消隐源
CMP_BLANKING_TIMER1_OC2	选择TIMER1_CH2输出比较信号为消隐源
CMP_BLANKING_TIMER2_OC2	选择TIMER2_CH2输出比较信号为消隐源
CMP_BLANKING_TIMER2_OC3	选择TIMER2_CH3输出比较信号为消隐源
CMP_BLANKING_TIMER7_OC0	选择TIMER7_CH0输出比较信号为消隐源
CMP_BLANKING_TIMER14_OC0	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 3-190. 函数 cmp_enable

函数名称	cmp_enable
函数原型	void cmp_enable(cmp_enum cmp_periph);
功能描述	使能CMP
先决条件	-
被调用函数	-
输入参数{in}	
cmp_periph	参考枚举 表 3-183. 枚举类型cmp_enum
输出参数{out}	
-	-
返回值	
-	-

例如：

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

```
cmp_enable(CMP0);
```

函数 cmp_disable

函数cmp_disable描述见下表：

表 3-191. 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 3-183. 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 3-192. 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 3-193. 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 3-194. 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 3-183. 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 3-195. 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 3-183. 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 3-196. 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 3-183. 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 3-197. 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 3-183. 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 3-198. 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 3-183. 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 3-199. 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 3-183. 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 3-200. 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 3-183. 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	
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 3-201. 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 3-183. 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 3-202. 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 3-183. 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 3-203. 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 3-183. 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 3-204. 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 3-183. 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 3-205. 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 3-183. 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 3-206. 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 3-207. 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_delay_step_config	配置CPDM延迟步长
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 3-208. 枚举类型 cpdm_output_phase_enum

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

成员名称	功能描述
TION_0	
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 3-209. 函数 cpdm_enable

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

函数 cpdm_disable

函数cpdm_disable描述见下表：

表 3-210. 函数 cpdm_disable

函数名称	cpdm_disable
函数原形	void cpdm_disable(void);
功能描述	禁能CPDM
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 CPDM */  
  
cpdm_disable();
```

函数 cpdm_delayline_sample_enable

函数cpdm_delayline_sample_enable描述见下表：

表 3-211. 函数 cpdm_delayline_sample_enable

函数名称	cpdm_delayline_sample_enable
函数原形	void cpdm_delayline_sample_enable(void);
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}	
-	-
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* enable CPDM delay line sample module */
```

```
cpdm_delayline_sample_enable();
```

函数 cpdm_delayline_sample_disable

函数cpdm_delayline_sample_disable描述见下表：

表 3-212. 函数 cpdm_delayline_sample_disable

函数名称	cpdm_delayline_sample_disable
函数原形	void cpdm_delayline_sample_disable(void);
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}	
-	-
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* disable CPDM delay line sample module */
```

```
cpdm_delayline_sample_disable();
```

函数 cpdm_output_clock_phase_select

函数cpdm_output_clock_phase_select描述见下表：

表 3-213. 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(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}	
output_clock_phase	输出时钟相位，请参考 表3-208. 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_OUTPUT_PHASE_SELECTION_0);
```

函数 cpdm_delay_step_config

函数cpdm_delay_step_config描述见下表：

表 3-214. 函数 cpdm_delay_step_config

函数名称	cpdm_delay_step_config
函数原形	void cpdm_delay_step_config(uint8_t delay_step)
功能描述	配置CPDM延迟步长
先决条件	-
被调用函数	-
输入参数{in}	
delay_step	0~127
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* configure CPDM delay step */
```

```
cpdm_delay_step_config(1);
```

函数 cpdm_delayline_length_valid_flag_get

函数cpdm_delayline_length_valid_flag_get描述见下表：

表 3-215. 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(void);
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}	
-	-
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();
```

函数 cpdm_delayline_length_get

函数cpdm_delayline_length_get描述见下表：

表 3-216. 函数 cpdm_delayline_length_get

函数名称	cpdm_delayline_length_get
函数原形	uint16_t cpdm_delayline_length_get(void);
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}	
-	-
输出参数{out}	
-	-
返回值	
uint16_t	0x00~0xFFF

例如：

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

```
uint16_t len;
```

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

函数 cpdm_clock_output

函数cpdm_clock_output描述见下表：

表 3-217. 函数 cpdm_clock_output

函数名称	cpdm_clock_output
函数原形	void cpdm_clock_output(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}	
output_clock_phase	输出时钟相位，请参考 表3-208. 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_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 3-218. 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 3-219. 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 3-220. 函数 `crc_deinit`

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

例如：

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

```
crc_deinit();
```

函数 `crc_reverse_output_data_enable`

函数 `crc_reverse_output_data_enable` 描述见下表：

表 3-221. 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 3-222. 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 3-223. 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 3-224. 函数 crc_data_register_read

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

例如：

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

函数 crc_free_data_register_read

函数crc_free_data_register_read描述见下表：

表 3-225. 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 3-226. 函数 `crc_free_data_register_write`

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

例如：

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

函数 `crc_init_data_register_write`

函数 `crc_init_data_register_write` 描述见下表：

表 3-227. 函数 `crc_init_data_register_write`

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

例如：

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

函数 `crc_input_data_reverse_config`

函数 `crc_input_data_reverse_config` 描述见下表：

表 3-228. 函数 `crc_input_data_reverse_config`

函数名称	<code>crc_input_data_reverse_config</code>
函数原形	<code>void crc_input_data_reverse_config(uint32_t data_reverse)</code>
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}	
<code>data_reverse</code>	设定的输入数据翻转功能
<code>CRC_INPUT_DATA_NOT</code>	输入数据不翻转
<code>CRC_INPUT_DATA_BYTE</code>	输入数据按字节翻转
<code>CRC_INPUT_DATA_HALFWORD</code>	输入数据按半字翻转
<code>CRC_INPUT_DATA_WORD</code>	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 3-229. 函数 `crc_polynomial_size_set`

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

CRC_CTL_PS_7	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 3-230. 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 3-231. 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
<i>INPUT_FORMAT_WORD</i>	输入数据格式为字
<i>INPUT_FORMAT_HALFWORD</i>	输入数据格式为半字
<i>INPUT_FORMAT_BYTE</i>	输入数据格式为字节
输出参数{out}	
-	-
返回值	
uint32_t	32位CRC计算结果（0-0xFFFFFFFF）

例如：

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

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

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

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

函数 **crc_block_data_calculate**

函数 **crc_block_data_calculate** 描述见下表：

表 3-232. 函数 **crc_block_data_calculate**

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

数字/模拟转换器可以将12位的数字数据转换为外部引脚上的电压输出，章节[3.8.1](#)描述了DAC的寄存器列表，章节[3.8.2](#)对DAC库函数进行说明。

3.8.1. 外设寄存器说明

DAC寄存器列表如下表所示：

表 3-233. 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

寄存器名称	寄存器描述
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
DAC_OUT0_SAW	DACx_OUT0锯齿波寄存器
DAC_OUT1_SAW	DACx_OUT1锯齿波寄存器
DAC_SAWMDR	DACx锯齿波模式寄存器

3.8.2. 外设库函数说明

DAC库函数列表如下表所示：

表 3-234. DAC 库函数

库函数名称	库函数描述
dac_deinit	DAC外设复位
dac_enable	DAC使能
dac_disable	DAC禁能
dac_dma_enable	DAC的DMA功能使能
dac_dma_disable	DAC的DMA功能禁能
dac_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_trimming_disable	DACx校准禁止
dac_output_value_get	DAC输出数据获取
dac_data_format_config	DAC数据格式设置
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_sawtooth_reset_trigger_source_config	DAC锯齿波复位触发源配置
dac_sawtooth_step_trigger_source_config	DAC锯齿波步进触发源配置
dac_sawtooth_step_direction_config	DAC锯齿波步进方向配置
dac_sawtooth_initial_value_set	DAC锯齿波初始值设置
dac_sawtooth_step_value_set	DAC锯齿波步进值设置
dac_sawtooth_step_software_trigger_enable	DAC锯齿波步进软件触发使能

库函数名称	库函数描述
enable	
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_reset_pesist_enable	DAC复位保持使能
dac_reset_pesist_disable	DAC复位保持禁能
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 3-235. 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 = 0,1,2,3)
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 3-236. 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 = 0,1,2,3)
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 3-237. 函数 **dac_disable**

函数名称	dac_disable
函数原型	void dac_disable(uint32_t dac_periph, uint8_t dac_out);
功能描述	DAC禁能
先决条件	-
被调用函数	-
输入参数{in}	
dac_periph	DAC外设
DACx	DAC外设选择 (x = 0,1,2,3)
输入参数{in}	
dac_out	DAC输出
DAC_OUTx	DAC输出通道选择 (x = 0,1)
输出参数{out}	
-	-
返回值	
-	-

例如:


```
/* disable DAC0_OUT0 */
```

```
dac_disable(DAC0, DAC_OUT0);
```

函数 **dac_dma_enable**

函数 **dac_dma_enable** 描述见下表：

表 3-238. 函数 **dac_dma_enable**

函数名称	dac_dma_enable
函数原型	void dac_dma_enable(uint32_t dac_periph, uint8_t dac_out);
功能描述	DAC的DMA功能使能
先决条件	-
被调用函数	-
输入参数{in}	
dac_periph	DAC外设
DACx	DAC外设选择 (x = 0,1,2,3)
输入参数{in}	
dac_out	DAC输出
DAC_OUTx	DAC输出通道选择 (x = 0,1)
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* enable DAC0_OUT0 DMA function */
```

```
dac_dma_enable(DAC0, DAC_OUT0);
```

函数 **dac_dma_disable**

函数 **dac_dma_disable** 描述见下表：

表 3-239. 函数 **dac_dma_disable**

函数名称	dac_dma_disable
函数原型	void dac_dma_disable(uint32_t dac_periph, uint8_t dac_out);
功能描述	DAC的DMA功能禁能
先决条件	-
被调用函数	-
输入参数{in}	
dac_periph	DAC外设
DACx	DAC外设选择 (x = 0,1,2,3)
输入参数{in}	
dac_out	DAC输出

<i>DAC_OUTx</i>	DAC输出通道选择 (x = 0,1)
输出参数{out}	
-	-
返回值	
-	-

例如:

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

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

函数 **dac_mode_config**

函数dac_mode_config描述见下表:

表 3-240. 函数 **dac_mode_config**

函数名称	dac_mode_config
函数原型	void dac_mode_config(uint32_t dac_periph, uint32_t dac_out, uint32_t mode);
功能描述	DAC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 = 0,1,2,3)
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}	
mode	DAC工作模式
<i>NORMAL_PIN_BUFFERON</i>	普通模式下, DAC_OUTx连接到外部引脚, 缓冲区启用
<i>NORMAL_PIN_PERIPH_BUFFERON</i>	普通模式下, DAC_OUTx连接到外部引脚和片上外设, 缓冲区启用
<i>NORMAL_PIN_BUFFEROFF</i>	普通模式下, DAC_OUTx连接到外部引脚, 缓冲区禁用
<i>NORMAL_PERIPH_BUFFEROFF</i>	普通模式下, DAC_OUTx连接到片上外设, 缓冲区禁用
<i>SAMPLEKEEP_PIN_BUFFERON</i>	采样保持模式下, DAC_OUTx连接到外部引脚, 缓冲区启用
<i>SAMPLEKEEP_PIN_PERIPH_BUFFERON</i>	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
<i>SAMPLEKEEP_PIN_BUFFEROFF</i>	采样保持模式下, DAC_OUTx连接到外部引脚, 缓冲区禁用
<i>SAMPLEKEEP_PERIPH_BUFFEROFF</i>	采样保持模式下, DAC_OUTx连接到片上外设, 缓冲区禁用

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

例如:

```
/* configure DAC0_OUT0 working in NORMAL_PIN_BUFFON mode */
dac_mode_config(DAC0, DAC_OUT0, NORMAL_PIN_BUFFON);
```

函数 `dac_trimming_value_get`

函数 `dac_trimming_value_get` 描述见下表:

表 3-241. 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 = 0,1,2,3)
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:

```
/* 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 3-242. 函数 `dac_trimming_value_set`

函数名称	<code>dac_trimming_value_set</code>
函数原型	<code>void dac_trimming_value_set(uint32_t dac_periph, uint32_t dac_out, uint32_t trim_value);</code>

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
<i>DACx</i>	DAC外设选择 (x = 0,1,2,3)
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}	
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 3-243. 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 = 0,1,2,3)
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_trimming_disable

函数dac_trimming_disable描述见下表:

表 3-244. 函数 dac_trimming_disable

函数名称	dac_trimming_disable
函数原型	void dac_trimming_disable(uint32_t dac_periph, uint32_t dac_out);
功能描述	DACx校准禁止
先决条件	-
被调用函数	-
输入参数{in}	
dac_periph	DAC外设
DACx	DAC外设选择 (x = 0,1,2,3)
输入参数{in}	
dac_out	DAC输出
DAC_OUTx	DAC输出通道选择 (x = 0,1)
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* disable the DAC0_OUT0 trimming */
```

```
dac_trimming_disable (DAC0, DAC_OUT0);
```

函数 dac_output_value_get

函数dac_output_value_get描述见下表:

表 3-245. 函数 dac_output_value_get

函数名称	dac_output_value_get
函数原型	uint16_t dac_output_value_get(uint32_t dac_periph);
功能描述	DAC输出数据获取
先决条件	-
被调用函数	-
输入参数{in}	
dac_periph	DAC外设
DACx	DAC外设选择 (x = 0,1,2,3)
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	
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 */
```

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

函数 dac_data_format_config

函数dac_data_format_config描述见下表:

表 3-246. 函数 dac_data_format_config

函数名称	dac_data_format_config
函数原型	void dac_data_format_config(uint32_t dac_periph, uint8_t dac_out, uint32_t format);
功能描述	DAC输出数据格式配置
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 = 0,1,2,3）
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}	
format	DAC对齐模式
DAC_DATA_FORMAT_UNSIGNED	无符号数据格式
DAC_DATA_FORMAT_SIGNED	有符号数据格式
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* set DAC0_OUT0 data signed format */
```

```
dac_data_format_config(DAC0, DAC_OUT0, DAC_DATA_FORMAT_SIGNED);
```

函数 dac_data_set

函数dac_data_set描述见下表:

表 3-247. 函数 `dac_data_set`

函数名称	<code>dac_data_set</code>
函数原型	<code>void dac_data_set(uint32_t dac_periph, uint8_t dac_out, uint32_t dac_align, uint16_t data);</code>
功能描述	DAC输出数据设置
先决条件	-
被调用函数	-
输入参数{in}	
<code>dac_periph</code>	DAC外设
<code>DACx</code>	DAC外设选择 (x = 0,1,2,3)
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>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>data</code>	写入DACx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 3-248. 函数 `dac_trigger_enable`

函数名称	<code>dac_trigger_enable</code>
函数原型	<code>void dac_trigger_enable(uint32_t dac_periph, uint8_t dac_out);</code>
功能描述	DAC触发使能
先决条件	-
被调用函数	-
输入参数{in}	
<code>dac_periph</code>	DAC外设
<code>DACx</code>	DAC外设选择 (x = 0,1,2,3)

输入参数{in}	
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 3-249. 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 = 0,1,2,3)
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 3-250. 函数 dac_trigger_source_config

函数名称	dac_trigger_source_config
------	---------------------------

函数原型	void dac_trigger_source_config(uint32_t dac_periph, uint8_t dac_out, uint32_t triggersource);
功能描述	DAC触发源选择
先决条件	-
被调用函数	-
输入参数{in}	
dac_periph	DAC外设
DACx	DAC外设选择 (x = 0,1,2,3)
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	外部触发来自TRIGSEL
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_EXTERNAL);
```

函数 dac_software_trigger_enable

函数dac_software_trigger_enable描述见下表:

表 3-251. 函数 dac_software_trigger_enable

函数名称	dac_software_trigger_enable
函数原型	void dac_software_trigger_enable(uint32_t dac_periph, uint8_t dac_out);
功能描述	DAC软件触发使能
先决条件	-
被调用函数	-
输入参数{in}	
dac_periph	DAC外设
DACx	DAC外设选择 (x = 0,1,2,3)
输入参数{in}	
dac_out	DAC输出
DAC_OUTx	DAC输出通道选择 (x = 0,1)
输出参数{out}	

-	-
返回值	
-	-

例如:

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

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

函数 **dac_wave_mode_config**

函数 **dac_wave_mode_config** 描述见下表:

表 3-252. 函数 **dac_wave_mode_config**

函数名称	dac_wave_mode_config
函数原型	void dac_wave_mode_config(uint32_t dac_periph, uint8_t dac_out, uint32_t wave_mode);
功能描述	DAC噪声波模式选择
先决条件	-
被调用函数	-
输入参数{in}	
dac_periph	DAC外设
DACx	DAC外设选择 (x = 0,1,2,3)
输入参数{in}	
dac_out	DAC输出
DAC_OUTx	DAC输出通道选择 (x = 0,1)
输入参数{in}	
wave_mode	噪声波模式选择
DAC_WAVE_DISABLE	噪声波模式禁能
DAC_WAVE_MODE_LFSR	LFSR噪声波模式
DAC_WAVE_MODE_TRIANGLE	三角波噪声波模式
DAC_WAVE_MODE_SAWTOOTH	锯齿波噪声波模式
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 3-253. 函数 dac_lfsr_noise_config

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

例如:

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

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

函数 dac_triangle_noise_config

函数dac_triangle_noise_config描述见下表:

表 3-254. 函数 dac_triangle_noise_config

函数名称	dac_triangle_noise_config
函数原型	void dac_triangle_noise_config(uint32_t dac_periph, uint8_t dac_out, uint32_t amplitude);
功能描述	DAC三角波模式设置
先决条件	-
被调用函数	-
输入参数{in}	

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

例如:

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

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

函数 **dac_sawtooth_reset_trigger_source_config**

函数dac_sawtooth_reset_trigger_source_config描述见下表:

表 3-255. 函数 **dac_sawtooth_reset_trigger_source_config**

函数名称	dac_sawtooth_reset_trigger_source_config
函数原型	void dac_sawtooth_reset_trigger_source_config(uint32_t dac_periph, uint8_t dac_out, uint32_t triggersource);
功能描述	DAC锯齿波复位触发源配置
先决条件	-
被调用函数	-
输入参数{in}	
dac_periph	DAC外设
<i>DACx</i>	DAC外设选择 (x = 0,1,2,3)
输入参数{in}	
dac_out	DAC输出
<i>DAC_OUTx</i>	DAC输出通道选择 (x = 0,1)
输入参数{in}	
triggersource	锯齿波复位触发源
<i>DAC_TRIGGER_EXTERNAL</i>	外部触发来自TRIGSEL
<i>DAC_TRIGGER_SOFTWARE</i>	软件触发
输出参数{out}	
-	-

返回值	
-	-

例如:

```
/* configure DAC0_OUT0 sawtooth reset trigger source */
```

```
dac_sawtooth_reset_trigger_source_config(DAC0,DAC_OUT0,DAC_TRIGGER_EXTERNAL);
```

函数 dac_sawtooth_step_trigger_source_config

函数dac_sawtooth_step_trigger_source_config描述见下表:

表 3-256. 函数 dac_sawtooth_step_trigger_source_config

函数名称	dac_sawtooth_step_trigger_source_config
函数原型	void dac_sawtooth_step_trigger_source_config(uint32_t dac_periph, uint8_t dac_out, uint32_t triggersource);
功能描述	DAC锯齿波步进触发源配置
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 = 0,1,2,3)
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	锯齿波步进触发源
DAC_TRIGGER_EXTERNAL	外部触发来自TRIGSEL
DAC_TRIGGER_SOFTWARE	软件触发
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* configure DAC0_OUT0 sawtooth step trigger source */
```

```
dac_sawtooth_step_trigger_source_config(DAC0,DAC_OUT0,DAC_TRIGGER_EXTERNAL);
```

函数 dac_sawtooth_step_direction_config

函数dac_sawtooth_step_direction_config描述见下表：

表 3-257. 函数 dac_sawtooth_step_direction_config

函数名称	dac_sawtooth_step_direction_config
函数原型	void dac_sawtooth_step_direction_config(uint32_t dac_periph, uint8_t dac_out, uint32_t direction);
功能描述	DAC锯齿波步进方向配置
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 = 0,1,2,3）
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}	
direction	步进方向
DAC_SAWTOOTH_STEP_DOWN	向下递减
DAC_SAWTOOTH_STEP_UP	向上递增
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* configure DAC0_OUT0 sawtooth step direction */
```

```
dac_sawtooth_step_direction_config(DAC0,DAC_OUT0,DAC_SAWTOOTH_STEP_DOWN);
```

函数 dac_sawtooth_initial_value_set

函数dac_sawtooth_initial_value_set描述见下表：

表 3-258. 函数 dac_sawtooth_initial_value_set

函数名称	dac_sawtooth_initial_value_set
函数原型	void dac_sawtooth_initial_value_set (uint32_t dac_periph, uint8_t dac_out, uint32_t init_value);
功能描述	DAC锯齿波初始值设置
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 = 0,1,2,3)
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}	
init_value	DAC锯齿波初始值 (0~4095)
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* set DAC0_OUT0 sawtooth initial value */
```

```
dac_sawtooth_initial_value_set(DAC0, DAC_OUT0, 0x0F);
```

函数 **dac_sawtooth_step_value_set**

函数dac_sawtooth_step_value_set描述见下表:

表 3-259. 函数 **dac_sawtooth_initial_value_set**

函数名称	dac_sawtooth_step_value_set
函数原型	void dac_sawtooth_step_value_set (uint32_t dac_periph, uint8_t dac_out, uint32_t step_value);
功能描述	DAC锯齿波步进值设置
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 = 0,1,2,3)
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}	
step_value	DAC锯齿波步进值 (0x0~0xFFFF)
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* set DAC0_OUT0 sawtooth step value */
```

```
dac_sawtooth_step_value_set(DAC0, DAC_OUT0, 0x0F);
```

函数 `dac_sawtooth_step_software_trigger_enable`

函数 `dac_sawtooth_step_software_trigger_enable` 描述见下表：

表 3-260. 函数 `dac_sawtooth_step_software_trigger_enable`

函数名称	<code>dac_sawtooth_step_software_trigger_enable</code>
函数原型	<code>void dac_sawtooth_step_software_trigger_enable(uint32_t dac_periph, uint8_t dac_out);</code>
功能描述	DAC 锯齿波步进软件触发使能
先决条件	-
被调用函数	-
输入参数{in}	
<code>dac_periph</code>	DAC 外设
<code>DACx</code>	DAC 外设选择 (x = 0,1,2,3)
输入参数{in}	
<code>dac_out</code>	DAC 输出
<code>DAC_OUTx</code>	DAC 输出通道选择 (x = 0,1)
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* enable DAC0_OUT0 sawtooth step trigger */
```

```
dac_sawtooth_step_software_trigger_enable(DAC0, DAC_OUT0);
```

函数 `dac_concurrent_enable`

函数 `dac_concurrent_enable` 描述见下表：

表 3-261. 函数 `dac_concurrent_enable`

函数名称	<code>dac_concurrent_enable</code>
函数原型	<code>void dac_concurrent_enable(uint32_t dac_periph);</code>
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}	
<code>dac_periph</code>	DAC 外设
<code>DACx</code>	DAC 外设选择 (x = 0,1,2,3)
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 3-262. 函数 dac_concurrent_disable

函数名称	dac_concurrent_disable
函数原型	void dac_concurrent_disable(uint32_t dac_periph);
功能描述	并发DAC模式禁能
先决条件	-
被调用函数	-
输入参数{in}	
dac_periph	DAC外设
DACx	DAC外设选择 (x = 0,1,2,3)
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 3-263. 函数 dac_concurrent_software_trigger_enable

函数名称	dac_concurrent_software_trigger_enable
函数原型	void dac_concurrent_software_trigger_enable(uint32_t dac_periph);
功能描述	并发DAC模式软件触发使能
先决条件	-
被调用函数	-
输入参数{in}	
dac_periph	DAC外设
DACx	DAC外设选择 (x = 0,1,2,3)
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 3-264. 函数 **dac_concurrent_data_set**

函数名称	dac_concurrent_data_set
函数原型	void dac_concurrent_data_set(uint32_t dac_periph, uint32_t dac_align, uint16_t data0, uint16_t data1);
功能描述	并发DAC模式输出数据设置
先决条件	-
被调用函数	-
输入参数{in}	
dac_periph	DAC外设
<i>DACx</i>	DAC外设选择 (x = 0,1,2,3)
输入参数{in}	
dac_align	DAC对齐模式
<i>DAC_ALIGN_8B_R</i>	8位数据右对齐
<i>DAC_ALIGN_12B_R</i>	12位数据右对齐
<i>DAC_ALIGN_12B_L</i>	12位数据左对齐
输入参数{in}	
data0	写入DAC0的数据 (0~4095)
输入参数{in}	
data1	写入DAC1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_reset_persist_enable**

函数dac_reset_persist_enable描述见下表:

表 3-265. 函数 `dac_reset_persist_enable`

函数名称	<code>dac_reset_persist_enable</code>
函数原型	<code>void dac_reset_persist_enable(uint32_t dac_periph, uint8_t dac_out);</code>
功能描述	DAC复位保持使能
先决条件	-
被调用函数	-
输入参数{in}	
<code>dac_periph</code>	DAC外设
<code>DACx</code>	DAC外设选择 (x = 0,1,2,3)
输入参数{in}	
<code>dac_out</code>	DAC输出
<code>DAC_OUTx</code>	DAC输出通道选择 (x = 0,1)
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* enable DAC0_OUT0 reset persisting mode */
```

```
dac_reset_persist_enable(DAC0, DAC_OUT0);
```

函数 `dac_reset_persist_disable`

函数`dac_reset_persist_disable`描述见下表:

表 3-266. 函数 `dac_reset_persist_disable`

函数名称	<code>dac_reset_persist_disable</code>
函数原型	<code>void dac_reset_persist_disable(uint32_t dac_periph, uint8_t dac_out);</code>
功能描述	DAC复位保持禁能
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 = 0,1,2,3)
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 reset persisting mode */
```

```
dac_reset_persist_disable(DAC0, DAC_OUT0);
```

函数 dac_sample_keep_mode_config

函数dac_sample_keep_mode_config描述见下表：

表 3-267. 函数 dac_sample_keep_mode_config

函数名称	dac_sample_keep_mode_config
函数原型	void dac_sample_keep_mode_config(uint32_t dac_periph, uint32_t dac_out, uint32_t sample_time, uint32_t keep_time, uint32_t refresh_time);
功能描述	DAC采样和保持时间值设置
先决条件	-
被调用函数	-
输入参数{in}	
dac_periph	DAC外设
DACx	DAC外设选择 (x = 0,1,2,3)
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}	
sample_time	DAC采样时间(0~1023)
输入参数{in}	
keep_time	DAC保持时间(0~1023)
输入参数{in}	
refresh_time	DAC刷新时间(0~255)
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 3-268. 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 = 0,1,2,3)
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> <i>R1</i>	DACx_OUT1 DMA欠载标志位
<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 3-269. 函数 **dac_flag_clear**

函数名称	dac_flag_clear
函数原型	void dac_flag_clear(uint32_t dac_periph, uint32_t flag);
功能描述	DAC标志位清除
先决条件	-
被调用函数	-
输入参数{in}	
dac_periph	DAC外设
<i>DACx</i>	DAC外设选择 (x = 0,1,2,3)
输入参数{in}	
flag	DAC状态标志位
<i>DAC_FLAG_DDUD</i> <i>R0</i>	DACx_OUT0 DMA欠载标志位
<i>DAC_FLAG_DDUD</i> <i>R1</i>	DACx_OUT1 DMA欠载标志位

输出参数{out}	
-	-
返回值	
-	-

例如:

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

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

函数 dac_interrupt_enable

函数dac_interrupt_enable描述见下表:

表 3-270. 函数 dac_interrupt_enable

函数名称	dac_interrupt_enable
函数原型	void dac_interrupt_enable(uint32_t dac_periph, uint32_t interrupt);
功能描述	DAC中断使能
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 = 0,1,2,3)
输入参数{in}	
interrupt	DAC中断
DAC_INT_DDUDRIE0	DACx_OUT0 DMA欠载中断
DAC_INT_DDUDRIE1	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 3-271. 函数 dac_interrupt_disable

函数名称	dac_interrupt_disable
------	-----------------------

函数原型	void dac_interrupt_disable(uint32_t dac_periph, uint32_t interrupt);
功能描述	DAC中断禁能
先决条件	-
被调用函数	-
输入参数{in}	
dac_periph	DAC外设
DACx	DAC外设选择 (x = 0,1,2,3)
输入参数{in}	
interrupt	DAC中断
DAC_INT_DDUDRI E0	DACx_OUT0 DMA欠载中断
DAC_INT_DDUDRI E1	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 3-272. 函数 dac_interrupt_flag_get

函数名称	dac_interrupt_flag_get
函数原型	FlagStatus dac_interrupt_flag_get(uint32_t dac_periph, uint32_t int_flag);
功能描述	DAC中断标志位获取
先决条件	-
被调用函数	-
输入参数{in}	
dac_periph	DAC外设
DACx	DAC外设选择 (x = 0,1,2,3)
输入参数{in}	
int_flag	DAC中断标志位
DAC_INT_FLAG_D DUDR0	DACx_OUT0 DMA欠载中断标志位
DAC_INT_FLAG_D DUDR1	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 */
```

```
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 3-273. 函数 dac_interrupt_flag_clear

函数名称	dac_interrupt_flag_clear
函数原型	FlagStatus dac_interrupt_flag_clear(uint32_t dac_periph, uint32_t int_flag);
功能描述	DAC中断标志位清除
先决条件	-
被调用函数	-
输入参数{in}	
dac_periph	DAC外设
DACx	DAC外设选择（x = 0,1,2,3）
输入参数{in}	
int_flag	DAC中断标志位
DAC_INT_FLAG_D DUDR0	DACx_OUT0 DMA欠载中断标志位
DAC_INT_FLAG_D DUDR1	DACx_OUT1 DMA欠载中断标志位
输出参数{out}	
-	-
返回值	
-	-

例如:

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

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

3.9. DBG

调试系统帮助调试者在低功耗模式下调试或者进行一些外设调试。章节[3.9.1](#)描述了DBG的寄存器列表，章节[3.9.2](#)对DBG库函数进行说明。

3.9.1. 外设寄存器说明

DBG寄存器列表如下表所示：

表 3-274. 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

3.9.2. 外设库函数说明

DBG库函数列表如下表所示：

表 3-275. 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_trace_pin_enable	使能跟踪引脚分配
dbg_trace_pin_disable	禁能跟踪引脚分配
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

枚举类型 dbg_periph_enum

表 3-276. 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=0,1,2,3,4,5,6,7,14,15,16,19)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
DBG_LPTIMER_HOLD	当内核停止时，保持LPTIMER计数器计数值不变
DBG_HRTIMER_HOLD	当内核停止时，保持HRTIMER计数器计数值不变

函数 dbg_deinit

函数dbg_deinit描述见下表:

表 3-277. 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 3-278. 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 3-279. 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 3-280. 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

<i>R_SLEEP</i>	
<i>DBG_LOW_POWER_SLEEP</i>	在深度睡眠模式下，不保持调试器连接，无法进行调试
<i>DBG_LOW_POWER_STANDBY</i>	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_trace_pin_enable**

函数dbg_trace_pin_enable描述见下表：

表 3-281. 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 3-282. 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_periph_enable

函数dbg_periph_enable描述见下表：

表 3-283. 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3-276. 枚举类型dbg_periph_enum
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 3-284. 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3-276. 枚举类型dbg_periph_enum
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);
```

3.10. DMA / DMAMUX

DMA控制器提供了一种硬件的方式在外设和存储器之间或者存储器和存储器之间传输数据，而无需CPU的介入，从而使CPU可以专注在处理其他系统功能上。章节[3.10.1](#)描述了DMA的寄存器列表，章节[3.10.2](#)对DMA库函数进行说明。

DMAMUX是DMA请求的传输调度器。可编程的DMA请求多路复用器DMAMUX，可在外设和DMA控制器之间路由DMA请求线路，或者DMAMUX也可以将可编程事件连入到输入触发信号上，作为一个DMAMUX请求发生器，再由DMAMUX请求路由器在DMAMUX请求生成器产生的DMA请求和DMA控制器之间路由DMA请求线路。章节[3.10.1](#)描述了DMAMUX的寄存器列表，章节[3.10.2](#)对DMAMUX库函数进行说明。

3.10.1. 外设寄存器说明

DMA寄存器列表如下表所示：

表 3-285. DMA 寄存器

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

DMAMUX寄存器列表如下表所示：

表 3-286. DMAMUX 寄存器

寄存器名称	寄存器描述
DMAMUX_RM_CHx CFG (x=0..13)	请求路由通道x配置寄存器
DMAMUX_RM_INT F	请求路由通道中断标志位寄存器
DMAMUX_RM_INT C	请求路由通道中断标志位清除寄存器
DMAMUX_RG_CHx CFG (x=0..3)	请求生成通道x配置寄存器
DMAMUX_RG_INT F	请求生成通道中断标志位寄存器
DMAMUX_RG_INT C	请求生成通道中断标志位清除寄存器

3.10.2. 外设库函数说明

DMA库函数列表如下表所示：

表 3-287. DMA 库函数

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

库函数名称	库函数描述
dma_flag_clear	清除DMA通道x标志位状态
dma_interrupt_enable	使能DMA通道x中断
dma_interrupt_disable	禁能DMA通道x中断
dma_interrupt_flag_get	获取DMA通道x中断标志位状态
dma_interrupt_flag_clear	清除DMA通道x中断标志位状态

DMAMUX库函数列表如下表所示：

表 3-288. DMAMUX 库函数

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

结构体 dma_parameter_struct

表 3-289. 结构体 dma_parameter_struct

成员名称	功能描述
------	------

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

结构体 dmamux_sync_parameter_struct

表 3-290. 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 3-291. 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 3-292. 枚举 dma_channel_enum

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

枚举 dmamux_multiplexer_channel_enum

表 3-293. 枚举 dmamux_multiplexer_channel_enum

成员名称	功能描述
DMAMUX_MUXCH	DMAMUX请求路由通道0

0	
DMAMUX_MUXCH 1	DMAMUX请求路由通道1
DMAMUX_MUXCH 2	DMAMUX请求路由通道2
DMAMUX_MUXCH 3	DMAMUX请求路由通道3
DMAMUX_MUXCH 4	DMAMUX请求路由通道4
DMAMUX_MUXCH 5	DMAMUX请求路由通道5
DMAMUX_MUXCH 6	DMAMUX请求路由通道6
DMAMUX_MUXCH 7	DMAMUX请求路由通道7
DMAMUX_MUXCH 8	DMAMUX请求路由通道8
DMAMUX_MUXCH 9	DMAMUX请求路由通道9
DMAMUX_MUXCH 10	DMAMUX请求路由通道10
DMAMUX_MUXCH 11	DMAMUX请求路由通道11
DMAMUX_MUXCH 12	DMAMUX请求路由通道12
DMAMUX_MUXCH 13	DMAMUX请求路由通道13

枚举 dmamux_generator_channel_enum

表 3-294. 枚举 dmamux_generator_channel_enum

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

枚举 dmamux_interrupt_enum

表 3-295. 枚举 dmamux_interrupt_enum

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

DMAMUX_INT_MU XCH1_SO	DMAMUX请求路由通道1同步溢出中断
DMAMUX_INT_MU XCH2_SO	DMAMUX请求路由通道2同步溢出中断
DMAMUX_INT_MU XCH3_SO	DMAMUX请求路由通道3同步溢出中断
DMAMUX_INT_MU XCH4_SO	DMAMUX请求路由通道4同步溢出中断
DMAMUX_INT_MU XCH5_SO	DMAMUX请求路由通道5同步溢出中断
DMAMUX_INT_MU XCH6_SO	DMAMUX请求路由通道6同步溢出中断
DMAMUX_INT_MU XCH7_SO	DMAMUX请求路由通道7同步溢出中断
DMAMUX_INT_MU XCH8_SO	DMAMUX请求路由通道8同步溢出中断
DMAMUX_INT_MU XCH9_SO	DMAMUX请求路由通道9同步溢出中断
DMAMUX_INT_MU XCH10_SO	DMAMUX请求路由通道10同步溢出中断
DMAMUX_INT_MU XCH11_SO	DMAMUX请求路由通道11同步溢出中断
DMAMUX_INT_MU XCH12_SO	DMAMUX请求路由通道12同步溢出中断
DMAMUX_INT_MU XCH13_SO	DMAMUX请求路由通道13同步溢出中断
DMAMUX_INT_GE NCH0_TO	DMAMUX请求生成通道0触发溢出中断
DMAMUX_INT_GE NCH1_TO	DMAMUX请求生成通道1触发溢出中断
DMAMUX_INT_GE NCH2_TO	DMAMUX请求生成通道2触发溢出中断
DMAMUX_INT_GE NCH3_TO	DMAMUX请求生成通道3触发溢出中断

枚举 dmamux_flag_enum

表 3-296. 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_G ENCH0_TO	DMAMUX请求生成通道0触发溢出标志
DMAMUX_FLAG_G ENCH1_TO	DMAMUX请求生成通道1触发溢出标志
DMAMUX_FLAG_G ENCH2_TO	DMAMUX请求生成通道2触发溢出标志
DMAMUX_FLAG_G ENCH3_TO	DMAMUX请求生成通道3触发溢出标志

枚举 dmamux_interrupt_flag_enum

表 3-297. 枚举 dmamux_interrupt_flag_enum

成员名称	功能描述
DMAMUX_INT_FLAG_MUXCH0_SO	DMAMUX请求路由通道0同步溢出中断标志
DMAMUX_INT_FLAG_MUXCH1_SO	DMAMUX请求路由通道1同步溢出中断标志
DMAMUX_INT_FLAG_MUXCH2_SO	DMAMUX请求路由通道2同步溢出中断标志

DMAMUX_INT_FLAG_MUXCH3_SO	DMAMUX请求路由通道3同步溢出中断标志
DMAMUX_INT_FLAG_MUXCH4_SO	DMAMUX请求路由通道4同步溢出中断标志
DMAMUX_INT_FLAG_MUXCH5_SO	DMAMUX请求路由通道5同步溢出中断标志
DMAMUX_INT_FLAG_MUXCH6_SO	DMAMUX请求路由通道6同步溢出中断标志
DMAMUX_INT_FLAG_MUXCH7_SO	DMAMUX请求路由通道7同步溢出中断标志
DMAMUX_INT_FLAG_MUXCH8_SO	DMAMUX请求路由通道8同步溢出中断标志
DMAMUX_INT_FLAG_MUXCH9_SO	DMAMUX请求路由通道9同步溢出中断标志
DMAMUX_INT_FLAG_MUXCH10_SO	DMAMUX请求路由通道10同步溢出中断标志
DMAMUX_INT_FLAG_MUXCH11_SO	DMAMUX请求路由通道11同步溢出中断标志
DMAMUX_INT_FLAG_GENCH0_TO	DMAMUX请求生成通道0触发溢出中断标志
DMAMUX_INT_FLAG_GENCH1_TO	DMAMUX请求生成通道1触发溢出中断标志
DMAMUX_INT_FLAG_GENCH2_TO	DMAMUX请求生成通道2触发溢出中断标志
DMAMUX_INT_FLAG_GENCH3_TO	DMAMUX请求生成通道3触发溢出中断标志

函数 dma_deinit

函数dma_deinit描述见下表：

表 3-298. 函数 dma_deinit

函数名称	dma_deinit
函数原型	void dma_deinit(uint32_t dma_periph, dma_channel_enum channelx);
功能描述	复位外设 DMAx 的通道 y 的所有寄存器
先决条件	无
被调用函数	无
输入参数{in}	
dma_periph	DMA 外设
DMAx(x=0,1)	DMA 外设选择
输入参数{in}	
channelx	DMA 通道
DMA_CHx	DMA 通道选择，参考 表 3-292. 枚举 dma_channel_enum

输出参数{out}	
-	-
返回值	
-	-

例如：

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

函数 dma_struct_para_init

函数dma_struct_para_init描述见下表：

表 3-299. 函数 dma_struct_para_init

函数名称	dma_struct_para_init
函数原型	void dma_struct_para_init(dma_parameter_struct* init_struct);
功能描述	将 DMA 结构体中所有参数初始化为默认值
先决条件	无
被调用函数	无
输入参数{in}	
*init_struct	一个已经定义的 dma_parameter_struct 结构体变量地址，参考 表 3-289. 结构体 dma_parameter_struct
输出参数{out}	
-	-
返回值	
-	-

例如：

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

函数 dma_init

函数dma_init描述见下表：

表 3-300. 函数 dma_init

函数名称	dma_init
函数原型	void dma_init(uint32_t dma_periph, dma_channel_enum channelx, dma_parameter_struct* init_struct);
功能描述	初始化外设 DMAx 的通道 y
先决条件	无
被调用函数	无
输入参数{in}	

dma_periph	DMA 外设
<i>DMAx</i>	DMA 外设选择
输入参数{in}	
channelx	DMA 通道
<i>DMA_CHx</i>	DMA 通道选择, 参考 表 3-292. 枚举 dma_channel_enum
输入参数{in}	
init_struct	初始化结构体, 结构体成员参考 表 3-289. 结构体 dma_parameter_struct
输出参数{out}	
-	-
返回值	
-	-

例如:

```

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

dma_init_struct.direction = DMA_PERIPHERAL_TO_MEMORY;
dma_init_struct.memory_addr = (uint32_t)g_destbuf;
dma_init_struct.memory_inc = DMA_MEMORY_INCREASE_ENABLE;
dma_init_struct.memory_width = DMA_MEMORY_WIDTH_32BIT;
dma_init_struct.number = TRANSFER_NUM/4;
dma_init_struct.periph_addr = (uint32_t) FLASH_WRITE_START_ADDR;
dma_init_struct.periph_inc = DMA_PERIPH_INCREASE_ENABLE;
dma_init_struct.periph_width = DMA_PERIPHERAL_WIDTH_32BIT;
dma_init_struct.priority = DMA_PRIORITY_ULTRA_HIGH;
dma_init(DMA0, DMA_CH0, &dma_init_struct);

```

函数 dma_circulation_enable

函数 dma_circulation_enable 描述见下表:

表 3-301. 函数 dma_circulation_enable

函数名称	dma_circulation_enable
函数原型	void dma_circulation_enable(uint32_t dma_periph, dma_channel_enum channelx);
功能描述	DMA 循环模式使能
先决条件	无
被调用函数	无
输入参数{in}	
dma_periph	DMA 外设
<i>DMAx</i>	DMA 外设选择
输入参数{in}	

channelx	DMA 通道
<i>DMA_CHx</i>	DMA 通道选择, 参考 表 3-292. 枚举 dma_channel_enum
输出参数{out}	
-	-
返回值	
-	-

例如:

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

函数 dma_circulation_disable

函数dma_circulation_disable描述见下表:

表 3-302. 函数 dma_circulation_disable

函数名称	dma_circulation_disable
函数原型	void dma_circulation_disable(uint32_t dma_periph, dma_channel_enum channelx);
功能描述	DMA 循环模式禁能
先决条件	无
被调用函数	无
输入参数{in}	
dma_periph	DMA 外设
<i>DMAx</i>	DMA 外设选择
输入参数{in}	
channelx	DMA 通道
<i>DMA_CHx</i>	DMA 通道选择, 参考 表 3-292. 枚举 dma_channel_enum
输出参数{out}	
-	-
返回值	
-	-

例如:

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

函数 dma_memory_to_memory_enable

函数dma_memory_to_memory_enable描述见下表:

表 3-303. 函数 dma_memory_to_memory_enable

函数名称	dma_memory_to_memory_enable
函数原型	void dma_memory_to_memory_enable(uint32_t dma_periph,

	<code>dma_channel_enum channelx);</code>
功能描述	存储器到存储器 DMA 传输使能
先决条件	无
被调用函数	无
输入参数{in}	
<code>dma_periph</code>	DMA 外设
<code>DMAx</code>	DMA 外设选择
输入参数{in}	
<code>channelx</code>	DMA 通道
<code>DMA_CHx</code>	DMA 通道选择, 参考 表 3-292. 枚举 dma_channel_enum
输出参数{out}	
-	-
返回值	
-	-

例如:

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

函数 dma_memory_to_memory_disable

函数dma_memory_to_memory_disable描述见下表:

表 3-304. 函数 dma_memory_to_memory_disable

函数名称	<code>dma_memory_to_memory_disable</code>
函数原形	<code>void dma_memory_to_memory_disable(uint32_t dma_periph, dma_channel_enum channelx);</code>
功能描述	存储器到存储器 DMA 传输禁能
先决条件	无
被调用函数	无
输入参数{in}	
<code>dma_periph</code>	DMA 外设
<code>DMAx</code>	DMA 外设选择
输入参数{in}	
<code>channelx</code>	DMA 通道
<code>DMA_CHx</code>	DMA 通道选择, 参考 表 3-292. 枚举 dma_channel_enum
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* disable DMA0 channel0 memory to memory mode */
```

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

函数 dma_channel_enable

函数dma_channel_enable描述见下表：

表 3-305. 函数 dma_channel_enable

函数名称	dma_channel_enable
函数原型	void dma_channel_enable(uint32_t dma_periph, dma_channel_enum channelx);
功能描述	外设 DMAx 的通道 y 传输使能
先决条件	无
被调用函数	无
输入参数{in}	
dma_periph	DMA 外设
DMAx	DMA 外设选择
输入参数{in}	
channelx	DMA 通道
DMA_CHx	DMA 通道选择，参考 表 3-292. 枚举 dma_channel_enum
输出参数{out}	
-	-
返回值	
-	-

例如：

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

函数 dma_channel_disable

函数dma_channel_disable描述见下表：

表 3-306. 函数 dma_channel_disable

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

<i>DMA_CHx</i>	DMA 通道选择, 参考 表 3-292. 枚举 dma_channel_enum
输出参数{out}	
-	-
返回值	
-	-

例如:

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

函数 dma_periph_address_config

函数dma_periph_address_config描述见下表:

表 3-307. 函数 dma_periph_address_config

函数名称	dma_periph_address_config
函数原型	void dma_periph_address_config(uint32_t dma_periph, dma_channel_enum channelx, uint32_t address);
功能描述	DMAx 通道 y 传输的外设基地址配置
先决条件	无
被调用函数	无
输入参数{in}	
dma_periph	DMA 外设
<i>DMAx</i>	DMA 外设选择
输入参数{in}	
channelx	DMA 通道
<i>DMA_CHx</i>	DMA 通道选择, 参考 表 3-292. 枚举 dma_channel_enum
输入参数{in}	
address	外设基地址
输出参数{out}	
-	-
返回值	
-	-

例如:

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

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

函数 dma_memory_address_config

函数dma_memory_address_config描述见下表:

表 3-308. 函数 dma_memory_address_config

函数名称	dma_memory_address_config
------	---------------------------

函数原型	void dma_memory_address_config(uint32_t dma_periph, dma_channel_enum channelx, uint32_t address);
功能描述	DMAx 通道 y 传输的存储器基地址配置
先决条件	无
被调用函数	无
输入参数{in}	
dma_periph	DMA 外设
DMAx	DMA 外设选择
输入参数{in}	
channelx	DMA 通道
DMA_CHx	DMA 通道选择, 参考 表 3-292. 枚举 dma_channel_enum
输入参数{in}	
address	存储器基地址
输出参数{out}	
-	-
返回值	
-	-

例如:

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

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

函数 dma_transfer_number_config

函数 dma_transfer_number_config 描述见下表:

表 3-309. 函数 dma_transfer_number_config

函数名称	dma_transfer_number_config
函数原型	void dma_transfer_number_config(uint32_t dma_periph, dma_channel_enum channelx, uint32_t number);
功能描述	配置 DMAx 通道 y 还有多少数据要传输
先决条件	无
被调用函数	无
输入参数{in}	
dma_periph	DMA 外设
DMAx	DMA 外设选择
输入参数{in}	
channelx	DMA 通道
DMA_CHx	DMA 通道选择, 参考 表 3-292. 枚举 dma_channel_enum
输入参数{in}	
number	数据传输数量 (0x00000000 – 0x0000FFFF)
输出参数{out}	
-	-

返回值	
-	-

例如：

```
#define TRANSFER_NUM                0x400

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

函数 dma_transfer_number_get

函数dma_transfer_number_get描述见下表：

表 3-310. 函数 dma_transfer_number_get

函数名称	dma_transfer_number_get
函数原型	uint32_t dma_transfer_number_get(uint32_t dma_periph, dma_channel_enum channelx);
功能描述	获取 DMAx 通道 y 还有多少数据要传输
先决条件	无
被调用函数	无
输入参数{in}	
dma_periph	DMA 外设
DMAx	DMA 外设选择
输入参数{in}	
channelx	DMA 通道
DMA_CHx	DMA 通道选择，参考 表 3-292. 枚举 dma_channel_enum
输出参数{out}	
-	-
返回值	
uint32_t	DMA 数据传输剩余数量（0x00000000 – 0x0000FFFF）

例如：

```
uint32_t number = 0;

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

函数 dma_priority_config

函数dma_priority_config描述见下表：

表 3-311. 函数 dma_priority_config

函数名称	dma_priority_config
函数原型	void dma_priority_config(uint32_t dma_periph, dma_channel_enum channelx, uint32_t priority);
功能描述	DMAx 通道 y 的传输软件优先级配置
先决条件	无

被调用函数	无
输入参数{in}	
dma_periph	DMA 外设
<i>DMAx</i>	DMA 外设选择
输入参数{in}	
channelx	DMA 通道
<i>DMA_CHx</i>	DMA 通道选择, 参考 表 3-292. 枚举 dma_channel_enum
输入参数{in}	
priority	DMA 通道软件优先级
<i>DMA_PRIORITY_LOW</i>	低优先级
<i>DMA_PRIORITY_MEDIUM</i>	中优先级
<i>DMA_PRIORITY_HIGH</i>	高优先级
<i>DMA_PRIORITY_ULTRA_HIGH</i>	极高优先级
输出参数{out}	
-	-
返回值	
-	-

例如:

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

函数 dma_memory_width_config

函数dma_memory_width_config描述见下表:

表 3-312. 函数 dma_memory_width_config

函数名称	dma_memory_width_config
函数原型	void dma_memory_width_config (uint32_t dma_periph, dma_channel_enum channelx, uint32_t mwidth);
功能描述	DMAx 通道 y 传输的存储器数据宽度配置
先决条件	无
被调用函数	无
输入参数{in}	
dma_periph	DMA 外设
<i>DMAx</i>	DMA 外设选择
输入参数{in}	
channelx	DMA 通道
<i>DMA_CHx</i>	DMA 通道选择, 参考 表 3-292. 枚举 dma_channel_enum
输入参数{in}	

mwidth	存储器数据传输宽度
<i>DMA_MEMORY_WIDTH_8BIT</i>	8 位数据传输宽度
<i>DMA_MEMORY_WIDTH_16BIT</i>	16 位数据传输宽度
<i>DMA_MEMORY_WIDTH_32BIT</i>	32 位数据传输宽度
输出参数{out}	
-	-
返回值	
-	-

例如：

```
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 3-313. 函数 dma_periph_width_config

函数名称	dma_periph_width_config
函数原型	void dma_periph_width_config (uint32_t dma_periph, dma_channel_enum channelx, uint32_t pwidth);
功能描述	DMAx 通道 y 传输的外设数据宽度配置
先决条件	无
被调用函数	无
输入参数{in}	
dma_periph	DMA 外设
<i>DMAx</i>	DMA 外设选择
输入参数{in}	
channelx	DMA 通道
<i>DMA_CHx</i>	DMA 通道选择，参考 表 3-292. 枚举 dma_channel_enum
输入参数{in}	
pwidth	外设数据传输宽度
<i>DMA_PERIPHERAL_WIDTH_8BIT</i>	8 位数据传输宽度
<i>DMA_PERIPHERAL_WIDTH_16BIT</i>	16 位数据传输宽度
<i>DMA_PERIPHERAL_WIDTH_32BIT</i>	32 位数据传输宽度
输出参数{out}	
-	-
返回值	

-	-
---	---

例如：

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

函数 dma_memory_increase_enable

函数dma_memory_increase_enable描述见下表：

表 3-314. 函数 dma_memory_increase_enable

函数名称	dma_memory_increase_enable
函数原型	void dma_memory_increase_enable(uint32_t dma_periph, dma_channel_enum channelx);
功能描述	DMAx 通道 y 传输的存储器地址生成算法增量模式使能
先决条件	无
被调用函数	无
输入参数{in}	
dma_periph	DMA 外设
DMAx	DMA 外设选择
输入参数{in}	
channelx	DMA 通道
DMA_CHx	DMA 通道选择，参考 表 3-292. 枚举 dma_channel_enum
输出参数{out}	
-	-
返回值	
-	-

例如：

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

函数 dma_memory_increase_disable

函数dma_memory_increase_disable描述见下表：

表 3-315. 函数 dma_memory_increase_disable

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

输入参数{in}	
channelx	DMA 通道
<i>DMA_CHx</i>	DMA 通道选择, 参考 表 3-292. 枚举 dma_channel_enum
输出参数{out}	
-	-
返回值	
-	-

例如:

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

函数 dma_periph_increase_enable

函数dma_periph_increase_enable描述见下表:

表 3-316. 函数 dma_periph_increase_enable

函数名称	dma_periph_increase_enable
函数原型	void dma_periph_increase_enable(uint32_t dma_periph, dma_channel_enum channelx);
功能描述	DMAx 通道 y 传输的外设地址生成算法增量模式使能
先决条件	无
被调用函数	无
输入参数{in}	
dma_periph	DMA 外设
<i>DMAx</i>	DMA 外设选择
输入参数{in}	
channelx	DMA 通道
<i>DMA_CHx</i>	DMA 通道选择, 参考 表 3-292. 枚举 dma_channel_enum
输出参数{out}	
-	-
返回值	
-	-

例如:

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

函数 dma_periph_increase_disable

函数dma_periph_increase_disable描述见下表:

表 3-317. 函数 dma_periph_increase_disable

函数名称	dma_periph_increase_disable
函数原型	void dma_periph_increase_disable(uint32_t dma_periph, dma_channel_enum channelx);

功能描述	DMAx 通道 y 传输的外设地址生成算法增量模式禁能
先决条件	无
被调用函数	无
输入参数{in}	
dma_periph	DMA 外设
<i>DMAx</i>	DMA 外设选择
输入参数{in}	
channelx	DMA 通道
<i>DMA_CHx</i>	DMA 通道选择, 参考 表 3-292. 枚举 dma_channel_enum
输出参数{out}	
-	-
返回值	
-	-

例如:

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

函数 dma_transfer_direction_config

函数 dma_transfer_direction_config 描述见下表:

表 3-318. 函数 dma_transfer_direction_config

函数名称	dma_transfer_direction_config
函数原型	void dma_transfer_direction_config(uint32_t dma_periph, dma_channel_enum channelx, uint32_t direction);
功能描述	DMAx 通道 y 的传输方向配置
先决条件	无
被调用函数	无
输入参数{in}	
dma_periph	DMA 外设
<i>DMAx</i>	DMA 外设选择
输入参数{in}	
channelx	DMA 通道
<i>DMA_CHx</i>	DMA 通道选择, 参考 表 3-292. 枚举 dma_channel_enum
输入参数{in}	
direction	数据传输方向
<i>DMA_PERIPHERAL_TO_MEMORY</i>	读取外设中数据, 写入存储器
<i>DMA_MEMORY_TO_PERIPHERAL</i>	读取存储器中数据, 写入外设
输出参数{out}	
-	-
返回值	

例如：

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

函数 dma_flag_get

函数dma_flag_get描述见下表：

表 3-319. 函数 dma_flag_get

函数名称	dma_flag_get
函数原型	FlagStatus dma_flag_get(uint32_t dma_periph, dma_channel_enum channelx, uint32_t flag);
功能描述	获取 DMAx 通道 y 标志位状态
先决条件	无
被调用函数	无
输入参数{in}	
dma_periph	DMA 外设
DMAx	DMA 外设选择
输入参数{in}	
channelx	DMA 通道
DMA_CHx	DMA 通道选择，参考 表 3-292. 枚举 dma_channel_enum
输入参数{in}	
flag	DMA 标志
DMA_FLAG_G	DMA 通道全局中断标志
DMA_FLAG_FTF	DMA 通道传输完成标志
DMA_FLAG_HTF	DMA 通道半传输完成标志
DMA_FLAG_ERR	DMA 通道错误标志
输出参数{out}	
-	-
返回值	
FlagStatus	SET 或 RESET

例如：

```
FlagStatus flag = RESET;
```

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

函数 dma_flag_clear

函数dma_flag_clear描述见下表：

表 3-320. 函数 dma_flag_clear

函数名称	dma_flag_clear
函数原型	void dma_flag_clear(uint32_t dma_periph, dma_channel_enum channelx,

	uint32_t flag);
功能描述	清除 DMAx 通道 y 标志位状态
先决条件	无
被调用函数	无
输入参数{in}	
dma_periph	DMA 外设
DMAx	DMA 外设选择
输入参数{in}	
channelx	DMA 通道
DMA_CHx	DMA 通道选择, 参考 表 3-292. 枚举 dma_channel_enum
输入参数{in}	
flag	DMA 标志
DMA_FLAG_G	DMA 通道全局中断标志
DMA_FLAG_FTF	DMA 通道传输完成标志
DMA_FLAG_HTF	DMA 通道半传输完成标志
DMA_FLAG_ERR	DMA 通道错误标志
输出参数{out}	
-	-
返回值	
-	-

例如:

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

函数 dma_interrupt_enable

函数dma_interrupt_enable描述见下表:

表 3-321. 函数 dma_interrupt_enable

函数名称	dma_interrupt_enable
函数原型	void dma_interrupt_enable(uint32_t dma_periph, dma_channel_enum channelx, uint32_t source);
功能描述	DMAx 通道 y 中断使能
先决条件	无
被调用函数	无
输入参数{in}	
dma_periph	DMA 外设
DMAx	DMA 外设选择
输入参数{in}	
channelx	DMA 通道
DMA_CHx	DMA 通道选择, 参考 表 3-292. 枚举 dma_channel_enum
输入参数{in}	
source	DMA 中断源

<i>DMA_INT_FTF</i>	DMA 通道传输完成中断
<i>DMA_INT_HTF</i>	DMA 通道半传输完成中断
<i>DMA_INT_ERR</i>	DMA 通道错误中断
输出参数{out}	
-	-
返回值	
-	-

例如：

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

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

函数 dma_interrupt_disable

函数dma_interrupt_disable描述见下表：

表 3-322. 函数 dma_interrupt_disable

函数名称	dma_interrupt_disable
函数原型	void dma_interrupt_disable(uint32_t dma_periph, dma_channel_enum channelx, uint32_t source);
功能描述	DMAx 通道 y 中断禁能
先决条件	无
被调用函数	无
输入参数{in}	
dma_periph	DMA 外设
<i>DMAx</i>	DMA 外设选择
输入参数{in}	
channelx	DMA 通道
<i>DMA_CHx</i>	DMA 通道选择，参考 表 3-292. 枚举 dma_channel_enum
输入参数{in}	
source	DMA 中断源
<i>DMA_INT_FTF</i>	DMA 通道传输完成中断
<i>DMA_INT_HTF</i>	DMA 通道半传输完成中断
<i>DMA_INT_ERR</i>	DMA 通道错误中断
输出参数{out}	
-	-
返回值	
-	-

例如：

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

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

函数 dma_interrupt_flag_get

函数dma_interrupt_flag_get描述见下表:

表 3-323. 函数 dma_interrupt_flag_get

函数名称	dma_interrupt_flag_get
函数原型	FlagStatus dma_interrupt_flag_get(uint32_t dma_periph, dma_channel_enum channelx, uint32_t flag);
功能描述	获取 DMAx 通道 y 中断标志位状态
先决条件	无
被调用函数	无
输入参数{in}	
dma_periph	DMA 外设
DMAx	DMA 外设选择
输入参数{in}	
channelx	DMA 通道
DMA_CHx	DMA 通道选择, 参考 表 3-292. 枚举 dma_channel_enum
输入参数{in}	
flag	DMA 标志
DMA_INT_FLAG_FTF	DMA 通道传输完成中断标志
DMA_INT_FLAG_HTF	DMA 通道半传输完成中断标志
DMA_INT_FLAG_ERR	DMA 通道错误中断标志
输出参数{out}	
-	-
返回值	
FlagStatus	SET 或 RESET

例如:

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

函数 dma_interrupt_flag_clear

函数dma_interrupt_flag_clear描述见下表:

表 3-324. 函数 dma_interrupt_flag_clear

函数名称	dma_interrupt_flag_clear
函数原型	void dma_interrupt_flag_clear(uint32_t dma_periph, dma_channel_enum channelx, uint32_t flag);
功能描述	清除 DMAx 通道 y 中断标志位状态

先决条件	无
被调用函数	无
输入参数{in}	
dma_periph	DMA 外设
<i>DMAx</i>	DMA 外设选择
输入参数{in}	
channelx	DMA 通道
<i>DMA_CHx</i>	DMA 通道选择, 参考 表 3-292. 枚举 dma_channel_enum
输入参数{in}	
flag	DMA 标志
<i>DMA_INT_FLAG_G</i>	DMA 通道全局中断标志
<i>DMA_INT_FLAG_FTF</i>	DMA 通道传输完成中断标志
<i>DMA_INT_FLAG_HTF</i>	DMA 通道半传输完成中断标志
<i>DMA_INT_FLAG_ERR</i>	DMA 通道错误中断标志
输出参数{out}	
-	-
返回值	
-	-

例如:

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

函数 dmamux_sync_struct_para_init

函数 dmamux_sync_struct_para_init 描述见下表:

表 3-325. 函数 dmamux_sync_struct_para_init

函数名称	dmamux_sync_struct_para_init
函数原型	void dmamux_sync_struct_para_init(dmamux_sync_parameter_struct *init_struct);
功能描述	将DMAMUX同步结构体中所有参数初始化为默认值
先决条件	无
被调用函数	无
输入参数{in}	
-	-
输出参数{out}	
init_struct	一个已经定义的dmamux_sync_parameter_struct结构体变量地址, 参考 表 3-290. 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 3-326. 函数 dmamux_synchronization_init

函数名称	dmamux_synchronization_init
函数原型	void dmamux_synchronization_init(dmamux_multiplexer_channel_enum channelx, dmamux_sync_parameter_struct *init_struct);
功能描述	初始化DMAMUX同步结构体通道x
先决条件	无
被调用函数	无
输入参数{in}	
channelx	指定要初始化的DMAMUX请求路由通道
DMAMUX_MUXCH x	DMAMUX通道选择，参考 表3-293. 枚举 dmamux_multiplexer_channel_enum
输入参数{in}	
init_struct	一个已经定义的dmamux_sync_parameter_struct结构体变量地址，参考 表 3-290. 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 3-327. 函数 `dmamux_synchronization_enable`

函数名称	<code>dmamux_synchronization_enable</code>
函数原型	<code>void dmamux_synchronization_enable(dmamux_multiplexer_channel_enum channelx);</code>
功能描述	使能DMAMUX同步模式
先决条件	无
被调用函数	无
输入参数{in}	
channelx	指定要初始化的DMAMUX请求路由通道
<code>DMAMUX_MUXCHx</code>	DMAMUX通道选择, 参考 表3-293. 枚举 <code>dmamux_multiplexer_channel_enum</code>
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 3-328. 函数 `dmamux_synchronization_disable`

函数名称	<code>dmamux_synchronization_disable</code>
函数原型	<code>void dmamux_synchronization_disable(dmamux_multiplexer_channel_enum channelx);</code>
功能描述	禁能DMAMUX同步模式
先决条件	无
被调用函数	无
输入参数{in}	
channelx	指定要初始化的DMAMUX请求路由通道
<code>DMAMUX_MUXCHx</code>	DMAMUX通道选择, 参考 表3-293. 枚举 <code>dmamux_multiplexer_channel_enum</code>
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 3-329. 函数 dmamux_event_generation_enable

函数名称	dmamux_event_generation_enable
函数原型	void dmamux_event_generation_enable(dmamux_multiplexer_channel_enum channelx);
功能描述	使能DMAMUX事件输出
先决条件	无
被调用函数	无
输入参数{in}	
channelx	指定要初始化的DMAMUX请求路由通道
DMAMUX_MUXCH x	DMAMUX通道选择，参考 表3-293. 枚举 dmamux_multiplexer_channel_enum
输出参数{out}	
-	-
返回值	
-	-

例如：

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

函数 dmamux_event_generation_disable

函数 dmamux_event_generation_disable 描述见下表：

表 3-330. 函数 dmamux_event_generation_disable

函数名称	dmamux_event_generation_disable
函数原型	void dmamux_event_generation_disable(dmamux_multiplexer_channel_enum channelx);
功能描述	禁能DMAMUX事件输出
先决条件	无
被调用函数	无
输入参数{in}	
channelx	指定要初始化的DMAMUX请求路由通道
DMAMUX_MUXCH x	DMAMUX通道选择，参考 表3-293. 枚举 dmamux_multiplexer_channel_enum
输出参数{out}	
-	-
返回值	
-	-

例如：

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

函数 dmamux_gen_struct_para_init

函数 dmamux_gen_struct_para_init 描述见下表：

表 3-331. 函数 dmamux_gen_struct_para_init

函数名称	dmamux_gen_struct_para_init
函数原型	void dmamux_gen_struct_para_init(dmamux_gen_parameter_struct *init_struct);
功能描述	将DMAMUX请求生成结构体中所有参数初始化为默认值
先决条件	无
被调用函数	无
输入参数{in}	
-	-
输出参数{out}	
init_struct	一个已经定义的dmamux_gen_parameter_struct结构体变量地址，参考 表3-291. 结构体dmamux_gen_parameter_struct
返回值	
-	-

例如：

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

函数 dmamux_request_generator_init

函数 dmamux_request_generator_init 描述见下表：

表 3-332. 函数 dmamux_request_generator_init

函数名称	dmamux_request_generator_init
函数原型	void dmamux_request_generator_init(dmamux_generator_channel_enum channelx, dmamux_gen_parameter_struct *init_struct);
功能描述	初始化DMAMUX请求生成结构体通道x
先决条件	无
被调用函数	无
输入参数{in}	
channelx	指定要初始化的DMAMUX请求生成通道
DMAMUX_GENCHx	DMAMUX请求生成通道选择，参考 表3-294. 枚举 dmamux_generator_channel_enum
输入参数{in}	
init_struct	一个已经定义的dmamux_gen_parameter_struct结构体变量地址，参考 表3-291.

	结构体dmamux_gen_parameter_struct
输出参数{out}	
-	-
返回值	
-	-

例如：

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

函数 dmamux_request_generator_channel_enable

函数 dmamux_request_generator_channel_enable 描述见下表：

表 3-333. 函数 dmamux_request_generator_channel_enable

函数名称	dmamux_request_generator_channel_enable
函数原型	void dmamux_request_generator_channel_enable(dmamux_generator_channel_enum channelx);
功能描述	使能DMAMUX请求生成通道x
先决条件	无
被调用函数	无
输入参数{in}	
channelx	指定要初始化的DMAMUX请求生成通道
DMAMUX_GENCHx	DMAMUX请求生成通道选择，参考 表3-294. 枚举 dmamux_generator_channel_enum
输出参数{out}	
-	-
返回值	
-	-

例如：

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

函数 dmamux_request_generator_channel_disable

函数 dmamux_request_generator_channel_disable 描述见下表：

表 3-334. 函数 dmamux_request_generator_channel_disable

函数名称	dmamux_request_generator_channel_disable
函数原型	void dmamux_request_generator_channel_disable(dmamux_generator_channel_enum channelx);
功能描述	禁能DMAMUX请求生成通道x
先决条件	无
被调用函数	无
输入参数{in}	
channelx	指定要初始化的DMAMUX请求生成通道
DMAMUX_GENCHx	DMAMUX请求生成通道选择, 参考 表3-294. 枚举 dmamux_generator_channel_enum
输出参数{out}	
-	-
返回值	
-	-

例如:

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

函数 dmamux_synchronization_polarity_config

函数 dmamux_synchronization_polarity_config 描述见下表:

表 3-335. 函数 dmamux_synchronization_polarity_config

函数名称	dmamux_synchronization_polarity_config
函数原型	void dmamux_synchronization_polarity_config(dmamux_multiplexer_channel_enum channelx, uint32_t polarity);
功能描述	配置DMAMUX同步输入的有效边沿
先决条件	无
被调用函数	无
输入参数{in}	
channelx	指定要初始化的DMAMUX请求路由通道
DMAMUX_MUXCHx(x=0..6)	DMAMUX通道选择, 参考 表3-293. 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 3-336. 函数 dmamux_request_forward_number_config

函数名称	dmamux_request_forward_number_config
函数原型	void dmamux_request_forward_number_config(dmamux_mux_channel_enum channelx, uint32_t number);
功能描述	配置DMAMUX通道x要传输多少个DMA请求
先决条件	无
被调用函数	无
输入参数{in}	
channelx	指定要初始化的DMAMUX请求路由通道
DMAMUX_MUXCHx	DMAMUX通道选择，参考 表3-293. 枚举 dmamux_mux_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 3-337. 函数 dmamux_sync_id_config

函数名称	dmamux_sync_id_config
函数原型	void dmamux_sync_id_config(dmamux_multiplexer_channel_enum channelx, uint32_t id);
功能描述	配置DMAMUX同步输入标识
先决条件	无
被调用函数	无
输入参数{in}	
channelx	指定要初始化的DMAMUX请求路由通道
DMAMUX_MUXCH x	DMAMUX通道选择, 参考 表3-293. 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

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

例如：

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

函数 dmamux_request_id_config

函数 dmamux_request_id_config 描述见下表：

表 3-338. 函数 dmamux_request_id_config

函数名称	dmamux_request_id_config
函数原型	void dmamux_request_id_config(dmamux_multiplexer_channel_enum channelx, uint32_t id);
功能描述	配置DMAMUX请求路由通道输入标识
先决条件	无
被调用函数	无
输入参数{in}	
channelx	指定要初始化的DMAMUX请求路由通道
DMAMUX_MUXCH x	DMAMUX通道选择，参考 表3-293. 枚举 dmamux_multiplexer_channel_enum
输入参数{in}	
id	DMA请求输入标识
DMA_REQUEST_M 2M	内存到内存传输
DMA_REQUEST_G ENERATOR0	DMAMUX请求生成通道0请求

<i>DMA_REQUEST_GENERATOR1</i>	DMAMUX请求生成通道1请求
<i>DMA_REQUEST_GENERATOR2</i>	DMAMUX请求生成通道2请求
<i>DMA_REQUEST_GENERATOR3</i>	DMAMUX请求生成通道3请求
<i>DMA_REQUEST_ADC0</i>	请求DMAMUX ADC0
<i>DMA_REQUEST_DAC0_CH0</i>	请求DMAMUX DAC0 CH0
<i>DMA_REQUEST_DAC0_CH1</i>	请求DMAMUX DAC0 CH1
<i>DMA_REQUEST_TIMER5_UP</i>	请求DMAMUX TIMER5_UP
<i>DMA_REQUEST_TIMER6_UP</i>	请求DMAMUX TIMER6_UP
<i>DMA_REQUEST_SPI0_RX</i>	请求DMAMUX SPI0_RX
<i>DMA_REQUEST_SPI0_TX</i>	请求DMAMUX SPI0_TX
<i>DMA_REQUEST_SPI1_RX</i>	请求DMAMUX SPI1_RX
<i>DMA_REQUEST_SPI1_TX</i>	请求DMAMUX SPI1_TX
<i>DMA_REQUEST_SPI2_RX</i>	请求DMAMUX SPI2_RX
<i>DMA_REQUEST_SPI2_TX</i>	请求DMAMUX SPI2_TX
<i>DMA_REQUEST_I2C0_RX</i>	请求DMAMUX I2C0_RX
<i>DMA_REQUEST_I2C0_TX</i>	请求DMAMUX I2C0_TX
<i>DMA_REQUEST_I2C1_RX</i>	请求DMAMUX I2C1_RX
<i>DMA_REQUEST_I2C1_TX</i>	请求DMAMUX I2C1_TX
<i>DMA_REQUEST_I2C2_RX</i>	请求DMAMUX I2C2_RX
<i>DMA_REQUEST_I2C2_TX</i>	请求DMAMUX I2C2_TX
<i>DMA_REQUEST_I2C3_RX</i>	请求DMAMUX I2C3_RX
<i>DMA_REQUEST_I2C3_TX</i>	请求DMAMUX I2C3_TX

C3_TX	
DMA_REQUEST_USART0_RX	请求DMAMUX USART0_RX
DMA_REQUEST_USART0_TX	请求DMAMUX USART0_TX
DMA_REQUEST_USART1_RX	请求DMAMUX USART1_RX
DMA_REQUEST_USART1_TX	请求DMAMUX USART1_TX
DMA_REQUEST_USART2_RX	请求DMAMUX USART2_RX
DMA_REQUEST_USART2_TX	请求DMAMUX USART2_TX
DMA_REQUEST_UART3_RX	请求DMAMUX UART3_RX
DMA_REQUEST_UART3_TX	请求DMAMUX UART3_TX
DMA_REQUEST_UART4_RX	请求DMAMUX UART4_RX
DMA_REQUEST_UART4_TX	请求DMAMUX UART4_TX
DMA_REQUEST_ADC1	请求DMAMUX ADC1
DMA_REQUEST_ADC2	请求DMAMUX ADC2
DMA_REQUEST_ADC3	请求DMAMUX ADC3
DMA_REQUEST_QSPI	请求DMAMUX QSPI
DMA_REQUEST_DAC1_CH0	请求DMAMUX DAC1_CH0
DMA_REQUEST_DAC1_CH1	请求DMAMUX DAC1_CH1
DMA_REQUEST_TIMER0_CH0	请求DMAMUX TIMER0_CH0
DMA_REQUEST_TIMER0_CH1	请求DMAMUX TIMER0_CH1
DMA_REQUEST_TIMER0_CH2	请求DMAMUX TIMER0_CH2
DMA_REQUEST_TIMER0_CH3	请求DMAMUX TIMER0_CH3
DMA_REQUEST_TIMER0_CH0N	请求DMAMUX TIMER0_CH0N

DMA_REQUEST_TIMER0_CH1N	请求DMAMUX TIMER0_CH1N
DMA_REQUEST_TIMER0_CH2N	请求DMAMUX TIMER0_CH2N
DMA_REQUEST_TIMER0_CH3N	请求DMAMUX TIMER0_CH3N
DMA_REQUEST_TIMER0_UP	请求DMAMUX TIMER0_UP
DMA_REQUEST_TIMER0_TRIG	请求DMAMUX TIMER0_TRIG
DMA_REQUEST_TIMER0_COM	请求DMAMUX TIMER0_COM
DMA_REQUEST_TIMER7_CH0	请求DMAMUX TIMER7_CH0
DMA_REQUEST_TIMER7_CH1	请求DMAMUX TIMER7_CH1
DMA_REQUEST_TIMER7_CH2	请求DMAMUX TIMER7_CH2
DMA_REQUEST_TIMER7_CH3	请求DMAMUX TIMER7_CH3
DMA_REQUEST_TIMER7_CH0N	请求DMAMUX TIMER7_CH0N
DMA_REQUEST_TIMER7_CH1N	请求DMAMUX TIMER7_CH1N
DMA_REQUEST_TIMER7_CH2N	请求DMAMUX TIMER7_CH2N
DMA_REQUEST_TIMER7_CH3N	请求DMAMUX TIMER7_CH3N
DMA_REQUEST_TIMER7_UP	请求DMAMUX TIMER7_UP
DMA_REQUEST_TIMER7_TRIG	请求DMAMUX TIMER7_TRIG
DMA_REQUEST_TIMER7_COM	请求DMAMUX TIMER7_COM
DMA_REQUEST_TIMER1_CH0	请求DMAMUX TIMER1_CH0
DMA_REQUEST_TIMER1_CH1	请求DMAMUX TIMER1_CH1
DMA_REQUEST_TIMER1_CH2	请求DMAMUX TIMER1_CH2
DMA_REQUEST_TIMER1_CH3	请求DMAMUX TIMER1_CH3
DMA_REQUEST_TIMER1_UP	请求DMAMUX TIMER1_UP

MER1_UP	
DMA_REQUEST_TIMER1_TRIG	请求DMAMUX TIMER1_TRIG
DMA_REQUEST_TIMER2_CH0	请求DMAMUX TIMER2_CH0
DMA_REQUEST_TIMER2_CH1	请求DMAMUX TIMER2_CH1
DMA_REQUEST_TIMER2_CH2	请求DMAMUX TIMER2_CH2
DMA_REQUEST_TIMER2_CH3	请求DMAMUX TIMER2_CH3
DMA_REQUEST_TIMER2_UP	请求DMAMUX TIMER2_UP
DMA_REQUEST_TIMER2_TRIG	请求DMAMUX TIMER2_TRIG
DMA_REQUEST_TIMER3_CH0	请求DMAMUX TIMER3_CH0
DMA_REQUEST_TIMER3_CH1	请求DMAMUX TIMER3_CH1
DMA_REQUEST_TIMER3_CH2	请求DMAMUX TIMER3_CH2
DMA_REQUEST_TIMER3_CH3	请求DMAMUX TIMER3_CH3
DMA_REQUEST_TIMER3_UP	请求DMAMUX TIMER3_UP
DMA_REQUEST_TIMER3_TRIG	请求DMAMUX TIMER3_TRIG
DMA_REQUEST_TIMER4_CH0	请求DMAMUX TIMER4_CH0
DMA_REQUEST_TIMER4_CH1	请求DMAMUX TIMER4_CH1
DMA_REQUEST_TIMER4_CH2	请求DMAMUX TIMER4_CH2
DMA_REQUEST_TIMER4_CH3	请求DMAMUX TIMER4_CH3
DMA_REQUEST_TIMER4_UP	请求DMAMUX TIMER4_UP
DMA_REQUEST_TIMER4_TRIG	请求DMAMUX TIMER4_TRIG
DMA_REQUEST_TIMER14_CH0	请求DMAMUX TIMER14_CH0
DMA_REQUEST_TIMER14_CH1	请求DMAMUX DMAMUXTIMER14_CH1

DMA_REQUEST_TIMER14_CH0N	请求DMAMUX TIMER14_CH0N
DMA_REQUEST_TIMER14_UP	请求DMAMUX TIMER14_UP
DMA_REQUEST_TIMER14_TRIG	请求DMAMUX TIMER14_TRIG
DMA_REQUEST_TIMER14_COM	请求DMAMUX TIMER14_COM
DMA_REQUEST_TIMER15_CH0	请求DMAMUX TIMER15_CH0
DMA_REQUEST_TIMER15_CH0N	请求DMAMUX TIMER15_CH0N
DMA_REQUEST_TIMER15_UP	请求DMAMUX TIMER15_UP
DMA_REQUEST_TIMER16_CH0	请求DMAMUX TIMER16_CH0
DMA_REQUEST_TIMER16_CH0N	请求DMAMUX TIMER16_CH0N
DMA_REQUEST_TIMER16_UP	请求DMAMUX TIMER16_UP
DMA_REQUEST_TIMER19_CH0	请求DMAMUX TIMER19_CH0
DMA_REQUEST_TIMER19_CH1	请求DMAMUX TIMER19_CH1
DMA_REQUEST_TIMER19_CH2	请求DMAMUX TIMER19_CH2
DMA_REQUEST_TIMER19_CH3	请求DMAMUX TIMER19_CH3
DMA_REQUEST_TIMER19_CH0N	请求DMAMUX TIMER19_CH0N
DMA_REQUEST_TIMER19_CH1N	请求DMAMUX TIMER19_CH1N
DMA_REQUEST_TIMER19_CH2N	请求DMAMUX TIMER19_CH2N
DMA_REQUEST_TIMER19_CH3N	请求DMAMUX TIMER19_CH3N
DMA_REQUEST_TIMER19_UP	请求DMAMUX TIMER19_UP
DMA_REQUEST_TIMER19_TRIG	请求DMAMUX TIMER19_TRIG
DMA_REQUEST_TIMER19_COM	请求DMAMUX TIMER19_COM
DMA_REQUEST_CAU_IN	请求DMAMUX CAU_IN

<i>AU_IN</i>	
<i>DMA_REQUEST_C</i> <i>AU_OUT</i>	请求DMAMUX CAU_OUT
<i>DMA_REQUEST_H</i> <i>RTIMER_MASTER</i>	请求DMAMUX HRTIMER_MASTER
<i>DMA_REQUEST_H</i> <i>RTIMER_TIMER0</i>	请求DMAMUX HRTIMER_TIMER0
<i>DMA_REQUEST_H</i> <i>RTIMER_TIMER1</i>	请求DMAMUX HRTIMER_TIMER1
<i>DMA_REQUEST_H</i> <i>RTIMER_TIMER2</i>	请求DMAMUX HRTIMER_TIMER2
<i>DMA_REQUEST_H</i> <i>RTIMER_TIMER3</i>	请求DMAMUX HRTIMER_TIMER3
<i>DMA_REQUEST_H</i> <i>RTIMER_TIMER4</i>	请求DMAMUX HRTIMER_TIMER4
<i>DMA_REQUEST_H</i> <i>RTIMER_TIMER5</i>	请求DMAMUX HRTIMER_TIMER5
<i>DMA_REQUEST_H</i> <i>RTIMER_TIMER6</i>	请求DMAMUX HRTIMER_TIMER6
<i>DMA_REQUEST_H</i> <i>RTIMER_TIMER7</i>	请求DMAMUX HRTIMER_TIMER7
<i>DMA_REQUEST_D</i> <i>AC2_CH0</i>	请求DMAMUX DAC2_CH0
<i>DMA_REQUEST_D</i> <i>AC2_CH1</i>	请求DMAMUX DAC2_CH1
<i>DMA_REQUEST_D</i> <i>AC3_CH0</i>	请求DMAMUX DAC3_CH0
<i>DMA_REQUEST_D</i> <i>AC3_CH1</i>	请求DMAMUX DAC3_CH1
<i>DMA_REQUEST_H</i> <i>PDF_FLT0</i>	请求DMAMUX HPDF_FLT0
<i>DMA_REQUEST_H</i> <i>PDF_FLT1</i>	请求DMAMUX HPDF_FLT1
<i>DMA_REQUEST_H</i> <i>PDF_FLT2</i>	请求DMAMUX HPDF_FLT2
<i>DMA_REQUEST_H</i> <i>PDF_FLT3</i>	请求DMAMUX HPDF_FLT3
<i>DMA_REQUEST_F</i> <i>AC_READ</i>	请求DMAMUX FAC_READ
<i>DMA_REQUEST_F</i> <i>AC_WRITE</i>	请求DMAMUX FAC_WRITE
<i>DMA_REQUEST_T</i> <i>MU_INPUT</i>	请求DMAMUX TMU_INPUT

<code>DMA_REQUEST_TMU_OUTPUT</code>	请求DMAMUX TMU_OUTPUT
<code>DMA_REQUEST_CAN0</code>	请求DMAMUX CAN0
<code>DMA_REQUEST_CAN1</code>	请求DMAMUX CAN1
<code>DMA_REQUEST_CAN2</code>	请求DMAMUX CAN2
输出参数{out}	
-	-
返回值	
-	-

例如：

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

函数 `dmamux_trigger_polarity_config`

函数 `dmamux_trigger_polarity_config` 描述见下表：

表 3-339. 函数 `dma_interrupt_disable`

函数名称	<code>dmamux_trigger_polarity_config</code>
函数原型	<code>void dmamux_trigger_polarity_config(dmamux_generator_channel_enum channelx, uint32_t polarity);</code>
功能描述	配置DMAMUX触发输入的有效边沿
先决条件	无
被调用函数	无
输入参数{in}	
channelx	指定要初始化的DMAMUX请求生成通道
<code>DMAMUX_GENCHx</code>	DMAMUX请求生成通道选择，参考 表3-294. 枚举 <code>dmamux_generator_channel_enum</code>
输入参数{in}	
polarity	触发输入信号有效边沿
<code>DMAMUX_GEN_NO_EVENT</code>	不检测边沿
<code>DMAMUX_GEN_RISING</code>	上升沿
<code>DMAMUX_GEN_FALLING</code>	下降沿
<code>DMAMUX_GEN_RISING_FALLING</code>	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 3-340. 函数 dmamux_request_generate_number_config

函数名称	dmamux_request_generate_number_config
函数原型	void dmamux_request_generate_number_config(dmamux_generator_channel_enum channelx, uint32_t number);
功能描述	配置DMAMUX请求生成器生成请求的数量
先决条件	无
被调用函数	无
输入参数{in}	
channelx	指定要初始化的DMAMUX请求生成通道
DMAMUX_GENCHx	DMAMUX请求生成通道选择，参考 表3-294. 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 3-341. 函数 dmamux_trigger_id_config

函数名称	dmamux_trigger_id_config
函数原型	void dmamux_trigger_id_config(dmamux_generator_channel_enum channelx, uint32_t id);
功能描述	配置DMAMUX触发输入标识

先决条件	无
被调用函数	无
输入参数{in}	
channelx	指定要初始化的DMAMUX请求生成通道
DMAMUX_GENCHx	DMAMUX请求生成通道选择, 参考 表3-294. 枚举 dmamux_generator_channel_enum
输入参数{in}	
id	触发输入标识
DMAMUX_TRIGGE R_EXTI0	触发输入为EXTI0
DMAMUX_TRIGGE R_EXTI1	触发输入为EXTI1
DMAMUX_TRIGGE R_EXTI2	触发输入为EXTI2
DMAMUX_TRIGGE R_EXTI3	触发输入为EXTI3
DMAMUX_TRIGGE R_EXTI4	触发输入为EXTI4
DMAMUX_TRIGGE R_EXTI5	触发输入为EXTI5
DMAMUX_TRIGGE R_EXTI6	触发输入为EXTI6
DMAMUX_TRIGGE R_EXTI7	触发输入为EXTI7
DMAMUX_TRIGGE R_EXTI8	触发输入为EXTI8
DMAMUX_TRIGGE R_EXTI9	触发输入为EXTI9
DMAMUX_TRIGGE R_EXTI10	触发输入为EXTI10
DMAMUX_TRIGGE R_EXTI11	触发输入为EXTI11
DMAMUX_TRIGGE R_EXTI12	触发输入为EXTI12
DMAMUX_TRIGGE R_EXTI13	触发输入为EXTI13
DMAMUX_TRIGGE R_EXTI14	触发输入为EXTI14
DMAMUX_TRIGGE R_EXTI15	触发输入为EXTI15
DMAMUX_TRIGGE R_EVTX_OUT0	触发输入为Evt_out0

DMAMUX_TRIGGER_EVTX_OUT1	触发输入为Evt_out1
DMAMUX_TRIGGER_EVTX_OUT2	触发输入为Evt_out2
DMAMUX_TRIGGER_EVTX_OUT3	触发输入为Evt_out3
DMAMUX_TRIGGER_LPTIMER_OUT	触发输入为LPTIMER_OUT
输出参数{out}	
-	-
返回值	
-	-

例如：

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

函数 dmamux_flag_get

函数 dmamux_flag_get 描述见下表：

表 3-342. 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DMAMUX通道x标志位状态
先决条件	无
被调用函数	无
输入参数{in}	
flag	标志类型，参考 表3-296. 枚举dmamux_flag_enum
输出参数{out}	
-	-
返回值	
FlagStatus	SET或RESET

例如：

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

函数 dmamux_flag_clear

函数 dmamux_flag_clear 描述见下表：

表 3-343. 函数 dmamux_flag_clear

函数名称	dmamux_flag_clear
函数原型	void dmamux_flag_clear(dmamux_flag_enum flag);
功能描述	清除DMAMUX通道x标志位状态
先决条件	无
被调用函数	无
输入参数{in}	
flag	标志类型，参考 表3-296. 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 3-344. 函数 dmamux_interrupt_enable

函数名称	dmamux_interrupt_enable
函数原型	void dmamux_interrupt_enable(dmamux_interrupt_enum interrupt);
功能描述	使能DMAMUX通道x中断
先决条件	无
被调用函数	无
输入参数{in}	
interrupt	中断类型，参考 表3-295. 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 3-345. 函数 dmamux_interrupt_disable

函数名称	dmamux_interrupt_disable
------	--------------------------

函数原型	void dmamux_interrupt_disable(dmamux_interrupt_enum interrupt);
功能描述	禁能DMAMUX通道x中断
先决条件	无
被调用函数	无
输入参数{in}	
interrupt	中断类型，参考 表3-295. 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 3-346. 函数 dmamux_interrupt_flag_get

函数名称	dmamux_interrupt_flag_get
函数原型	FlagStatus dmamux_interrupt_flag_get(dmamux_interrupt_flag_enum int_flag);
功能描述	获取DMAMUX通道x中断标志位状态
先决条件	无
被调用函数	无
输入参数{in}	
int_flag	标志类型，参考 表3-297. 枚举dmamux_interrupt_flag_enum
输出参数{out}	
-	-
返回值	
FlagStatus	SET或RESET

例如：

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

函数 dmamux_interrupt_flag_clear

函数 dmamux_interrupt_flag_clear 描述见下表：

表 3-347. 函数 dmamux_interrupt_flag_clear

函数名称	dmamux_interrupt_flag_clear
------	-----------------------------

函数原型	FlagStatus dmamux_interrupt_flag_get(dmamux_interrupt_flag_enum int_flag);
功能描述	清除DMAMUX通道x中断标志位状态
先决条件	无
被调用函数	无
输入参数{in}	
int_flag	标志类型，参考 表3-297. 枚举dmamux_interrupt_flag_enum
输出参数{out}	
-	-
返回值	
-	-

例如：

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

3.11. CLA

可配置逻辑阵列(CLA)为外部引脚、ADC和定时器提供256个可编程数字逻辑操作，而无需CPU干预。CLA模块共有4个单元，每个CLA单元支持GPIO引脚的可配置异步和同步输出。章节[3.11.1](#)描述了CLA的寄存器列表，章节[3.11.2](#)对CLA库函数进行说明。

3.11.1. 外设寄存器说明

CLA寄存器列表如下表所示：

表 3-348. CLA 寄存器

寄存器名称	寄存器描述
CLA_GCTL	全局控制寄存器
CLA_INTE	中断使能寄存器
CLA_INTF	中断标志寄存器0
CLA_STAT	状态寄存器
CLAx_SIGS(x=0..3)	信号选择寄存器
CLAx_LCUCTL(x=0..3)	LCU控制寄存器
CLAx_CTL(x=0..3)	控制寄存器

3.11.2. 外设库函数说明

CLA库函数列表如下表所示：

表 3-349. CLA 库函数

库函数名称	库函数描述
cla_deinit	复位CLA
cla_enable	使能CLA
cla_disable	除能CLA
cla_output_state_get	获取CLA输出状态
cla_sigs_input_config	配置信号选择器选择输入
cla_lcu_control_config	配置LCU控制寄存器
cla_output_config	配置CLA的输出
cla_output_enable	使能CLA输出
cla_output_disable	除能CLA输出
cla_flip_flop_output_reset	复位触发器输出异步
cla_flip_flop_clockpolarity_config	配置触发器时钟极性
cla_flip_flop_clocksource_config	配置触发器时钟源
cla_flag_get	查询CLA标志位
cla_flag_clear	清除CLA标志位
cla_negedge_interrupt_enable	使能CLA单元下降沿中断
cla_negedge_interrupt_disable	禁止CLA单元下降沿中断
cla_posedge_interrupt_enable	使能CLA单元上升沿中断
cla_posedge_interrupt_disable	禁止CLA单元上升沿中断
cla_interrupt_flag_get	查询CLA中断标志位
cla_interrupt_flag_clear	清除CLA中断标志位

函数 cla_deinit

函数cla_deinit描述见下表：

表 3-350. 函数 cla_deinit

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

例如：

```
/* CLA deinitialize */
```

```
cla_deinit ();
```

函数 cla_enable

函数cla_enable描述见下表：

表 3-351. 函数 cla_enable

函数名称	cla_enable
函数原型	void cla_enable(cla_enum cla_periph);
功能描述	使能CLA
先决条件	-
被调用函数	-
输入参数{in}	
cla_periph	CLA外设单元
CLAx	(x=0,1,2,3)
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* enable CLA0 */
cla_enable (CLA0);
```

函数 cla_disable

函数cla_disable描述见下表：

表 3-352. 函数 cla_disable

函数名称	cla_disable
函数原型	cla_disable(cla_enum cla_periph);
功能描述	除能CLA
先决条件	-
被调用函数	-
输入参数{in}	
cla_periph	CLA外设单元
CLAx	(x=0,1,2,3)
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* disable CLA0 */
cla_disable (CAL0);
```

函数 cla_output_state_get

函数cla_output_state_get描述见下表:

表 3-353. 函数 cla_output_state_get

函数名称	cla_output_state_get
函数原型	cla_outputstatus_enum cla_output_state_get(cla_enum cla_periph);
功能描述	获取CLA输出状态
先决条件	-
被调用函数	-
输入参数{in}	
cla_periph	CLA外设
CLAx	(x=0,1,2,3)
输出参数{out}	
-	-
返回值	
cla_outputstatus_enum	CLA_OUTPUT_HIGH / CLA_OUTPUT_LOW

例如:

```
/* get CLA0 output state */
```

```
cla_output_state_get (CLA0);
```

函数 cla_sigs_input_config

函数cla_sigs_input_config描述见下表:

表 3-354. 函数 cla_sigs_input_config

函数名称	cla_sigs_input_config
函数原型	void cla_sigs_input_config(cla_enum cla_periph, cla_sigs_enum sigs, uint32_t input);
功能描述	配置信号选择器选择输入
先决条件	-
被调用函数	-
输入参数{in}	
cla_periph	CLA外设单元
CLAx	(x=0,1,2,3)
输入参数{in}	
sigs	信号选择器
SIGS0	信号选择器1
SIGS1	信号选择器2
输入参数{in}	
input	信号选择器选择输入

CLA0SIGS1_CLA0_ASYNC_OUT	CLA0信号选择器0选择CLA0异步输出作为输入
CLA1SIGS1_CLA0_ASYNC_OUT	CLA1信号选择器1选择CLA0异步输出作为输入
CLA2SIGS1_CLA0_ASYNC_OUT	CLA2信号选择器1选择CLA0异步输出作为输入
CLA3SIGS1_CLA0_ASYNC_OUT	CLA3信号选择器1选择CLA0异步输出作为输入
CLA0SIGS1_CLA1_ASYNC_OUT	CLA0信号选择器1选择CLA1异步输出作为输入
CLA1SIGS1_CLA1_ASYNC_OUT	CLA1信号选择器1选择CLA1异步输出作为输入
CLA2SIGS1_CLA1_ASYNC_OUT	CLA2信号选择器1选择CLA1异步输出作为输入
CLA3SIGS1_CLA1_ASYNC_OUT	CLA3信号选择器1选择CLA1异步输出作为输入
CLA0SIGS1_CLA2_ASYNC_OUT	CLA0信号选择器1选择CLA2异步输出作为输入
CLA1SIGS1_CLA2_ASYNC_OUT	CLA1信号选择器1选择CLA2异步输出作为输入
.....
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* configure CLA0SIGS0_CLA0_ASYNC_OUT for CLA2 SIGS0 input */
cla_sigs_input_config (CLA2, SIGS0, CLA0SIGS0_CLA0_ASYNC_OUT);
```

函数 cla_lcu_control_config

函数cla_lcu_control_config描述见下表：

表 3-355. 函数 cla_lcu_control_config

函数名称	cla_lcu_control_config
函数原型	void cla_lcu_control_config(cla_enum cla_periph, uint8_t lcuctl_value);
功能描述	配置LCU控制寄存器
先决条件	-
被调用函数	-
输入参数{in}	
cla_periph	CLA外设单元
CLAx	(x=0,1,2,3)

输入参数{in}	
lcuctl_value	LCU控制寄存器值
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* configure CLA LCU control register value */
```

```
cla_lcu_control_config (CLA0,0xC0);
```

函数 cla_output_config

函数cla_output_config描述见下表：

表 3-356. 函数 cla_output_config

函数名称	cla_output_config
函数原型	void cla_output_config(cla_enum cla_periph, uint32_t output);
功能描述	配置CLA的输出
先决条件	-
被调用函数	-
输入参数{in}	
cla_periph	CLA外设单元
CLAx	(x=0,1,2,3)
输入参数{in}	
output	CLA的输出
FLIP_FLOP_OUTPUT	触发器的输出作为CLAx的输出
LCU_RESULT	LCU的结果作为CLAx的输出
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* configure LCU_RESULT for CLA0 output */
```

```
cla_output_config (CLA0, LCU_RESULT);
```

函数 cla_output_enable

函数cla_output_enable描述见下表：

表 3-357. 函数 cla_output_enable

函数名称	cla_output_enable
------	-------------------

函数原型	void cla_output_enable (cla_enum cla_periph);
功能描述	使能CLA输出
先决条件	-
被调用函数	-
输入参数{in}	
cla_periph	CLA外设单元
CLAx	(x=0,1,2,3)
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* enable CLA0 output */
void cla_output_enable (CLA0);
```

函数 cla_output_disable

函数cla_output_disable描述见下表：

表 3-358. 函数 cla_output_disable

函数名称	cla_output_disable
函数原型	void cla_output_disable (cla_enum cla_periph);
功能描述	除能CLA输出
先决条件	-
被调用函数	-
输入参数{in}	
cla_periph	CLA外设单元
CLAx	(x=0,1,2,3)
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* disable CLA0 output */
void cla_output_disable (CLA0);
```

函数 cla_flip_flop_output_reset

函数cla_flip_flop_output_reset描述见下表：

表 3-359. 函数 cla_flip_flop_output_reset

函数名称	cla_flip_flop_output_reset
函数原型	void cla_flip_flop_output_reset(cla_enum cla_periph);
功能描述	复位触发器输出异步
先决条件	-
被调用函数	-
输入参数{in}	
cla_periph	CLA外设单元
CLAx	(x=0,1,2,3)
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* reset the CLA0 flip-flop output asynchronously */
```

```
cla_flip_flop_output_reset (CLA0);
```

函数 cla_flip_flop_clockpolarity_config

函数cla_flip_flop_clockpolarity_config描述见下表：

表 3-360. 函数 cla_flip_flop_clockpolarity_config

函数名称	cla_flip_flop_clockpolarity_config
函数原型	void cla_flip_flop_clockpolarity_config(cla_enum cla_periph, uint32_t polarity);
功能描述	配置触发器时钟极性
先决条件	-
被调用函数	-
输入参数{in}	
cla_periph	CLA外设单元
CLAx	(x=0,1,2,3)
输入参数{in}	
polarity	触发器时钟极性
CLA_CLOCKPOLARITY_POSEDGE	时钟上升沿有效
CLA_CLOCKPOLARITY_NEGEDGE	时钟下降沿有效
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* configure clock posedge is valid for CLA0 flip-flop */
```

```
cla_flip_flop_clockpolarity_config (CLA0, CLA_CLOCKPOLARITY_POSEDGE);
```

函数 cla_flip_flop_clocksource_config

函数cla_flip_flop_clocksource_config描述见下表：

表 3-361. 函数 cla_flip_flop_clocksource_config

函数名称	cla_flip_flop_clocksource_config
函数原型	void cla_flip_flop_clocksource_config(cla_enum cla_periph, uint32_t clock_source);
功能描述	配置触发器时钟源
先决条件	-
被调用函数	-
输入参数{in}	
cla_periph	CLA外设单元
CLAx	(x=0,1,2,3)
输入参数{in}	
clock_source	触发器时钟极性
PRE_CLA_LCU_RE SULT	CLA[x-1] LCU的结果作为时钟源
SIGS0_OUTPUT	SIGS0输出作为时钟源
HCLK	HCLK作为时钟源
TIMER_TRGO	TIMER_TRGO作为时钟源
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* configure the SIGS0_OUTPUT for CAL0 flip-flop's clock polarity */
```

```
cla_flip_flop_clocksource_config (CLA0, SIGS0_OUTPUT);
```

函数 cla_flag_get

函数cla_flag_get描述见下表：

表 3-362. 函数 cla_flag_get

函数名称	cla_flag_get
函数原型	FlagStatus cla_flag_get(cla_flag_enum flag);
功能描述	查询CLA标志位
先决条件	-
被调用函数	-

输入参数{in}	
flag	CLA标志位
CLA_FLAG_CLA0NF	CLA0单元下降沿标志
CLA_FLAG_CLA0PF	CLA0单元上升沿标志
CLA_FLAG_CLA1NF	CLA1单元下降沿标志
CLA_FLAG_CLA1PF	CLA1单元上升沿标志
CLA_FLAG_CLA2NF	CLA2单元下降沿标志
CLA_FLAG_CLA2PF	CLA2单元上升沿标志
CLA_FLAG_CLA3NF	CLA3单元下降沿标志
CLA_FLAG_CLA3PF	CLA3单元上升沿标志
输出参数{out}	
-	-
返回值	
FlagStatus	SET / RESET

例如:

```
/* check CLA0 unit negedge flag */
```

```
cla_flag_get (CLA_FLAG_CLA0NF);
```

函数 cla_flag_clear

函数cla_flag_clear描述见下表:

表 3-363. 函数 cla_flag_clear

函数名称	cla_flag_clear
函数原型	void cla_flag_clear(cla_flag_enum flag);
功能描述	清除CLA标志位
先决条件	-
被调用函数	-
输入参数{in}	
flag	CLA标志位
CLA_FLAG_CLA0NF	CLA0单元下降沿标志
CLA_FLAG_CLA0PF	CLA0单元上升沿标志
CLA_FLAG_CLA1NF	CLA1单元下降沿标志
CLA_FLAG_CLA1PF	CLA1单元上升沿标志
CLA_FLAG_CLA2NF	CLA2单元下降沿标志
CLA_FLAG_CLA2PF	CLA2单元上升沿标志
CLA_FLAG_CLA3NF	CLA3单元下降沿标志
CLA_FLAG_CLA3PF	CLA3单元上升沿标志
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* clear CLA0 unit negedge flag */
cla_flag_clear (CLA_FLAG_CLA0NF);
```

函数 `cla_negedge_interrupt_enable`

函数 `cla_negedge_interrupt_enable` 描述见下表：

表 3-364. 函数 `cla_negedge_interrupt_enable`

函数名称	<code>cla_negedge_interrupt_enable</code>
函数原型	<code>void cla_negedge_interrupt_enable(uint32_t clanie);</code>
功能描述	使能CLA下降沿中断
先决条件	-
被调用函数	-
输入参数{in}	
clanie	CLA下降沿中断
<i>CLA0NIE</i>	CLA0单元下降沿中断使能
<i>CLA1NIE</i>	CLA1单元下降沿中断使能
<i>CLA2NIE</i>	CLA2单元下降沿中断使能
<i>CLA3NIE</i>	CLA3单元下降沿中断使能
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* enable CLA0 negedge interrupt */
cla_negedge_interrupt_enable (CLA0NIE);
```

函数 `cla_negedge_interrupt_disable`

函数 `cla_negedge_interrupt_disable` 描述见下表：

表 3-365. 函数 `cla_negedge_interrupt_disable`

函数名称	<code>cla_negedge_interrupt_disable</code>
函数原型	<code>void cla_negedge_interrupt_disable(uint32_t clanidis);</code>
功能描述	禁止CLA下降沿中断
先决条件	-
被调用函数	-
输入参数{in}	
clanidis	CLA下降沿中断
<i>CLA0NI_DISABLE</i>	CLA0单元下降沿中断禁止
<i>CLA1NI_DISABLE</i>	CLA1单元下降沿中断禁止

CLA2NI_DISABLE	CLA2单元下降沿中断禁止
CLA3NI_DISABLE	CLA3单元下降沿中断禁止
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* disable CLA0 negedge interrupt */
```

```
cla_negedge_interrupt_disable (CLA0NI_DISABLE);
```

函数 cla_posedge_interrupt_enable

函数cla_posedge_interrupt_enable描述见下表：

表 3-366. 函数 cla_posedge_interrupt_enable

函数名称	cla_posedge_interrupt_enable
函数原型	void cla_posedge_interrupt_enable(uint32_t clapie);
功能描述	使能CLA上升沿中断
先决条件	-
被调用函数	-
输入参数{in}	
clapie	CLA上升沿中断
CLA0PIE	CLA0单元上升沿中断使能
CLA1PIE	CLA1单元上升沿中断使能
CLA2PIE	CLA2单元上升沿中断使能
CLA3NIE	CLA3单元上升沿中断使能
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* enable CLA0 posedge interrupt */
```

```
cla_posedge_interrupt_enable (CLA0PIE);
```

函数 cla_posedge_interrupt_disable

函数cla_posedge_interrupt_disable描述见下表：

表 3-367. 函数 cla_posedge_interrupt_disable

函数名称	cla_posedge_interrupt_disable
函数原型	void cla_posedge_interrupt_disable(uint32_t clapidis);

功能描述	禁止CLA上升沿中断
先决条件	-
被调用函数	-
输入参数{in}	
clapidis	CLA上升沿中断
<i>CLA0PI_DISABLE</i>	CLA0单元上升沿中断禁止
<i>CLA1PI_DISABLE</i>	CLA1单元上升沿中断禁止
<i>CLA2PI_DISABLE</i>	CLA2单元上升沿中断禁止
<i>CLA3PI_DISABLE</i>	CLA3单元上升沿中断禁止
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* disable CLA0 posedge interrupt */
```

```
cla_posedge_interrupt_disable (CLA0PI_DISABLE);
```

函数 **cla_interrupt_flag_get**

函数cla_interrupt_flag_get描述见下表：

表 3-368. 函数 cla_interrupt_flag_get

函数名称	cla_interrupt_flag_get
函数原型	FlagStatus cla_interrupt_flag_get(cla_interrupt_flag_enum int_flag);
功能描述	查询CLA中断标志位
先决条件	-
被调用函数	-
输入参数{in}	
int_flag	CLA中断标志
<i>CLA_INT_FLAG_CLA0NF</i>	CLA0单元下降沿中断标志
<i>CLA_INT_FLAG_CLA0PF</i>	CLA0单元上升沿中断标志
<i>CLA_INT_FLAG_CLA1NF</i>	CLA1单元下降沿中断标志
<i>CLA_INT_FLAG_CLA1PF</i>	CLA1单元上升沿中断标志
<i>CLA_INTFLAG_CLA2NF</i>	CLA2单元下降沿中断标志
<i>CLA_INTFLAG_CLA2PF</i>	CLA2单元上升沿中断标志
<i>CLA_INTFLAG_CLA3NF</i>	CLA3单元下降沿中断标志
<i>CLA_INTFLAG_CLA3PF</i>	CLA3单元上升沿中断标志
输出参数{out}	
-	-
返回值	
FlagStatus	SET / RESET

例如：

```
/* check CLA0 unit negedge interrupt flag */
```

```
cla_interrupt_flag_get (CLA_INT_FLAG_CLA0NF);
```

函数 cla_interrupt_flag_clear

函数cla_interrupt_flag_clear描述见下表：

表 3-369. 函数 cla_interrupt_flag_clear

函数名称	cla_interrupt_flag_clear
函数原型	void cla_interrupt_flag_clear(cla_interrupt_flag_enum int_flag);
功能描述	清除CLA中断标志位
先决条件	-
被调用函数	-
输入参数{in}	
int_flag	CLA中断标志
CLA_INT_FLAG_CLA0NF	CLA0单元下降沿中断标志
CLA_INT_FLAG_CLA0PF	CLA0单元上升沿中断标志
CLA_INT_FLAG_CLA1NF	CLA1单元下降沿中断标志
CLA_INT_FLAG_CLA1PF	CLA1单元上升沿中断标志
CLA_INTFLAG_CLA2NF	CLA2单元下降沿中断标志
CLA_INTFLAG_CLA2PF	CLA2单元上升沿中断标志
CLA_INTFLAG_CLA3NF	CLA3单元下降沿中断标志
CLA_INTFLAG_CLA3PF	CLA3单元上升沿中断标志
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* clear CLA0 unit negedge interrupt flag */
```

```
cla_interrupt_flag_clear (CLA_INT_FLAG_CLA0NF);
```

3.12. EXMC

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

3.12.1. 外设寄存器描述

EXMC寄存器列表如下表所示:

表 3-370. EXMC 寄存器

寄存器名称	寄存器描述
EXMC_SNCTL	SRAM/NOR flash控制寄存器
EXMC_SNTCFG	SRAM/NOR flash时序寄存器
EXMC_SNWTCFG	SRAM/NOR flash写时序寄存器
EXMC_SNSTAT	SRAM/NOR flash状态寄存器
EXMC_SNLATDEC	SRAM/NOR Flash数据延迟减少寄存器

3.12.2. 外设库函数说明

EXMC库函数列表如下表所示:

表 3-371. 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_norsram_page_size_config	配置CRAM页大小
exmc_norsram_consecutive_clock_configuration	配置连续时钟产生条件
exmc_norsram_write_fifo_config	配置EXMC NOR/SRAM (bank0 region0) 写FIFO
exmc_norsram_status_get	获取EXMC NOR/SRAM (bank0 region0) region FIFO状态

结构体 exmc_norsram_timing_parameter_struct

表 3-372. 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_data_latency_dec	数据延迟减少值
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	地址建立时间

成员名称	功能描述
ptime	

结构体 `exmc_norsram_parameter_struct`

表 3-373. 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
databus_width	指定外部存储器数据总线宽度
memory_type	指定外部存储器的类型
address_data_mux	数据线/地址线复用是否复用
read_write_timing	未用扩展模式时，读时序参数和写时序参数；或采用扩展模式时，读时序参数
write_timing	未用扩展模式时，写时序参数

函数 `exmc_norsram_deinit`

函数`exmc_norsram_deinit`描述见下表：

表 3-374. 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 3-375. 函数 `exmc_norsram_struct_para_init`

函数名称	<code>exmc_norsram_struct_para_init</code>
函数原型	<code>void exmc_norsram_struct_para_init(exmc_norsram_parameter_struct* exmc_norsram_init_struct);</code>
功能描述	初始化结构体 <code>exmc_norsram_parameter_struct</code>
先决条件	-
被调用函数	-
输入参数{in}	
-	-
输出参数{out}	
<code>exmc_norsram_init_struct</code>	初始化结构体，结构体成员参考 表3-373. 结构体 <code>exmc_norsram_parameter_struct</code>
返回值	
-	-

例如：

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

函数 `exmc_norsram_init`

函数 `exmc_norsram_init` 描述见下表：

表 3-376. 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3-373. 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_data_latency_dec = 0;

lcd_timing_init_struct.syn_clk_division = EXMC_SYN_CLOCK_RATIO_DISABLE;

lcd_timing_init_struct.bus_latency = 2;

lcd_timing_init_struct.asyn_data_setup_time = 18;

lcd_timing_init_struct.asyn_address_hold_time = 3;

lcd_timing_init_struct.asyn_address_setup_time = 8;

lcd_timing_init_struct.byte_lane_setup_time = 0;

/* configure EXMC bus parameters */

lcd_init_struct.norsram_region = EXMC_BANK0_NORSRAM_REGION0;

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.wrap_burst_mode = DISABLE;

lcd_init_struct.nwait_polarity = EXMC_NWAIT_POLARITY_LOW;

lcd_init_struct.burst_mode = DISABLE;

lcd_init_struct.databus_width = EXMC_NOR_DATABUS_WIDTH_16B;

lcd_init_struct.memory_type = EXMC_MEMORY_TYPE_SRAM;

```

```

lcd_init_struct.address_data_mux = DISABLE;

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 3-377. 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 3-378. 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_NO RSRAM_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_norsram_page_size_config

函数exmc_norsram_page_size_config描述见下表：

表 3-379. 函数 exmc_norsram_page_size_config

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

例如：


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

```
exmc_norsram_page_size_config (EXMC_CRAM_PAGE_SIZE_128_BYTES);
```

函数 `exmc_norsram_consecutive_clock_config`

函数 `exmc_norsram_consecutive_clock_config` 描述见下表：

表 3-380. 函数 `exmc_norsram_consecutive_clock_config`

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

例如：

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

```
exmc_norsram_consecutive_clock_config(EXMC_CLOCK_SYN_MODE);
```

函数 `exmc_norsram_write_fifo_config`

函数 `exmc_norsram_write_fifo_config` 描述见下表：

表 3-381. 函数 `exmc_norsram_write_fifo_config`

函数名称	<code>exmc_norsram_write_fifo_config</code>
函数原型	<code>void exmc_norsram_write_fifo_config(uint32_t wr_fifo);</code>
功能描述	配置EXMC NOR/SRAM (bank0 region0) 写FIFO
先决条件	-
被调用函数	-
输入参数{in}	
<code>wr_fifo</code>	写FIFO配置
<code>EXMC_WRITE_FIFO_ENABLE</code>	使能写FIFO
<code>EXMC_WRITE_FIFO_DISABLE</code>	禁能写FIFO

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

例如：

```
/* enable the write FIFO function */
```

```
exmc_norsram_write_fifo_config(EXMC_WRITE_FIFO_ENABLE);
```

函数 **exmc_norsram_status_get**

函数exmc_norsram_status_get描述见下表：

表 3-382. 函数 exmc_norsram_status_get

函数名称	exmc_norsram_status_get
函数原型	uint32_t exmc_norsram_status_get(void);
功能描述	获取EXMC NOR/SRAM (bank0 region0) region FIFO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	NOR/SRAM写FIFO状态
<i>EXMC_FIFO_NOT_EMPTY</i>	NOR/SRAM写FIFO非空
<i>EXMC_FIFO_EMPTY</i>	NOR/SRAM写FIFO空

例如：

```
/* get the EXMC NOR/SRAM write FIFO status */
```

```
uint32_t status;
```

```
status = exmc_norsram_status_get();
```

3.13. EXTI

EXTI是MCU中的中断/事件控制器，包括39个相互独立的边沿检测电路并且能够向处理器内核产生中断请求或唤醒事件。章节[3.13.1](#)描述了EXTI的寄存器列表，章节[3.13.2](#)对EXTI库函数进行说明。

3.13.1. 外设寄存器说明

EXTI寄存器列表如下表所示：

表 3-383. EXTI 寄存器

寄存器名称	寄存器描述
EXTI_INTEN0	中断使能寄存器0
EXTI_EVEN0	事件使能寄存器0
EXTI_RTEN0	上升沿触发使能寄存器0
EXTI_FTEN0	下降沿触发使能寄存器0
EXTI_SWIEV0	软件中断事件寄存器0
EXTI_PD0	挂起寄存器0
EXTI_INTEN1	中断使能寄存器1
EXTI_EVEN1	事件使能寄存器1
EXTI_RTEN1	上升沿触发使能寄存器1
EXTI_FTEN1	下降沿触发使能寄存器1
EXTI_SWIEV1	软件中断事件寄存器1
EXTI_PD1	挂起寄存器1

3.13.2. 外设库函数说明

EXTI库函数列表如下表所示：

表 3-384. 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 3-385. 枚举类型 exti_line_enum

枚举名称	枚举描述
EXTI_0	EXTI线0

枚举名称	枚举描述
EXTI_1	EXTI线1
EXTI_2	EXTI线2
EXTI_3	EXTI线3
EXTI_4	EXTI线4
EXTI_5	EXTI线5
EXTI_6	EXTI线6
EXTI_7	EXTI线7
EXTI_8	EXTI线8
EXTI_9	EXTI线9
EXTI_10	EXTI线10
EXTI_11	EXTI线11
EXTI_12	EXTI线12
EXTI_13	EXTI线13
EXTI_14	EXTI线14
EXTI_15	EXTI线15
EXTI_16	EXTI线16
EXTI_17	EXTI线17
EXTI_18	EXTI线18
EXTI_19	EXTI线19
EXTI_20	EXTI线20
EXTI_21	EXTI线21
EXTI_22	EXTI线22
EXTI_23	EXTI线23
EXTI_24	EXTI线24
EXTI_25	EXTI线25
EXTI_26	EXTI线26
EXTI_27	EXTI线27
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
EXTI_38	EXTI线38

枚举类型 `exti_mode_enum`

表 3-386. 枚举类型 `exti_mode_enum`

枚举名称	枚举描述
<code>EXTI_INTERRUPT</code>	EXTI中断模式
<code>EXTI_EVENT</code>	EXTI事件模式

枚举类型 `exti_trig_type_enum`

表 3-387. 枚举类型 `exti_trig_type_enum`

枚举名称	枚举描述
<code>EXTI_TRIG_RISING</code>	EXTI上升沿触发
<code>EXTI_TRIG_FALLING</code>	EXTI下降沿触发
<code>EXTI_TRIG_BOTH</code>	EXTI双边沿触发
<code>EXTI_TRIG_NONE</code>	EXTI双边沿均不触发

函数 `exti_deinit`

函数`exti_deinit`描述见下表：

表 3-388. 函数 `exti_deinit`

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

例如：

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

```
exti_deinit();
```

函数 `exti_init`

函数`exti_init`描述见下表：

表 3-389. 函数 `exti_init`

函数名称	<code>exti_init</code>
函数原型	<code>void exti_init(exti_line_enum linex, exti_mode_enum mode,</code>

	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3-385. 枚举类型exti_line_enum
输入参数{in}	
mode	EXTI模式, 参考 表3-386. 枚举类型exti_mode_enum
输入参数{in}	
trig_type	触发类型, 参考 表3-387. 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 3-390. 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3-385. 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 3-391. 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3-385. 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 3-392. 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3-385. 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 3-393. 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3-385. 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 3-394. 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3-385. 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 3-395. 函数 `exti_software_interrupt_disable`

函数名称	<code>exti_software_interrupt_disable</code>
函数原型	<code>void exti_software_interrupt_disable(exti_line_enum linex);</code>
功能描述	禁用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3-385. 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 3-396. 函数 `exti_flag_get`

函数名称	<code>exti_flag_get</code>
函数原型	<code>FlagStatus exti_flag_get(exti_line_enum linex);</code>
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3-385. 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 3-397. 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3-385. 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 3-398. 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3-385. 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 3-399. 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3-385. 枚举类型exti_line_enum
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* clear EXTI line 0 interrupt flag status */
exti_interrupt_flag_clear(EXTI_0);
```

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

3.14.1. 外设寄存器说明

FAC寄存器列表如下表所示:

表 3-400. 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

3.14.2. 外设库函数说明

FAC库函数列表如下表所示:

表 3-401. FAC 库函数

库函数名称	库函数描述
fac_deinit	FAC外设复位
fac_struct_para_init	将FAC初始化结构体中所有参数初始化为默认值
fac_fixed_data_preload_init	将FAC定点预装载结构体中所有参数初始化为默认值
fac_float_data_preload_init	将FAC浮点预装载结构体中所有参数初始化为默认值
fac_init	初始化外设FAC
fac_fixed_buffer_preload	FAC预装载X0 X1 Y定点缓冲区
fac_float_buffer_preload	FAC预装载X0 X1 Y浮点缓冲区
fac_fixed_data_preload	FAC预装载定点数据
fac_float_data_preload	FAC预装载浮点数据
fac_reset	复位FAC
fac_clip_config	FAC限幅配置
fac_float_enable	使能FAC浮点数据类型
fac_float_disable	禁能FAC浮点数据类型
fac_dma_enable	使能FAC DMA功能
fac_dma_disable	禁能FAC DMA功能
fac_x0_config	配置FAC输入缓冲区
fac_x1_config	配置FAC滤波参数缓冲区
fac_y_config	配置FAC输出缓冲区
fac_function_config	配置FAC执行函数
fac_start	启动FAC
fac_stop	停止FAC
fac_finish_calculate	FAC完成滤波计算
fac_interrupt_enable	使能FAC中断
fac_interrupt_disable	禁能FAC中断
fac_interrupt_flag_get	获取FAC中断标志
fac_flag_get	获取FAC的状态标志
fac_fixed_data_write	FAC写定点数据
fac_fixed_data_read	FAC读定点数据
fac_float_data_write	FAC写浮点数据
fac_float_data_read	FAC读浮点数据

结构体 fac_parameter_struct

表 3-402. 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

成员名称	功能描述
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 3-403. 结构体 fac_fixed_data_preload_struct

成员名称	功能描述
coeffa_size	滤波参数向量A的大小
*coeffa_ctx	滤波参数向量A的内容（int16_t类型）
coeffb_size	滤波参数向量B的大小
*coeffb_ctx	滤波参数向量B的内容（int16_t类型）
input_size	输入数据向量长度
*input_ctx	输入数据向量的内容（int16_t类型）
output_size	输出数据向量长度
*output_ctx	输出数据向量的内容（int16_t类型）

结构体 fac_float_data_preload_struct

表 3-404. 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 3-405. 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 3-406. 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 3-407. 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 3-408. 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 3-409. 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 3-410. 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 3-411. 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 3-412. 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 3-413. 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 3-414. 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 3-415. 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
<i>FAC_CP_ENABLE</i>	使能限幅
<i>FAC_CP_DISABLE</i>	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 3-416. 函数 fac_float_enable

函数名称	fac_float_enable
函数原型	void fac_float_enable(void);
功能描述	浮点数据格式使能
先决条件	-
被调用函数	-
输入参数{in}	
-	-
输出参数{out}	

-	-
返回值	
-	-

例如:

```
/* enable FAC float point format */
```

```
fac_float_enable ();
```

函数 fac_float_disable

函数fac_float_disable描述见下表:

表 3-417. 函数 fac_float_disable

函数名称	fac_float_disable
函数原型	void fac_float_disable(void);
功能描述	浮点数据格式禁能
先决条件	-
被调用函数	-
输入参数{in}	
-	-
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* disable FAC float point format */
```

```
fac_float_disable ();
```

函数 fac_dma_enable

函数fac_dma_enable描述见下表:

表 3-418. 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 3-419. 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 3-420. 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
<i>FAC_THRESHOLD</i> _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 3-421. 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 3-422. 函数 `fac_y_config`

函数名称	<code>fac_y_config</code>
函数原型	<code>void fac_y_config(uint32_t watermark, uint8_t baseaddr, uint8_t bufsize);</code>
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}	
<code>watermark</code>	输出缓冲区阈值
<code>FAC_THRESHOLD_x</code>	Y缓冲区空阈值 (x =1, 2, 4, 8)
输入参数{in}	
<code>baseaddr</code>	输出缓冲区基地址, 0..255
输入参数{in}	
<code>bufsize</code>	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 3-423. 函数 `fac_function_config`

函数名称	<code>fac_function_config</code>
函数原型	<code>void fac_function_config(fac_parameter_struct* fac_parameter);</code>
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}	
<code>fac_parameter</code>	FAC初始化参数结构体, 结构体成员参考 结构体<code>fac_parameter_struct</code>
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 3-424. 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 3-425. 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 3-426. 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 3-427. 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 3-428. 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 3-429. 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 3-430. 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 3-431. 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
<i>FAC_CTL_RIE</i>	FAC读缓冲区中断
<i>FAC_CTL_WIE</i>	FAC写缓冲区中断
<i>FAC_CTL_OFEIE</i>	FAC上溢错误中断
<i>FAC_CTL_UFEIE</i>	FAC下溢错误中断
<i>FAC_CTL_STEIE</i>	FAC饱和错误中断
<i>FAC_CTL_GSTEIE</i>	FAC增益饱和错误中断
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* enable the FAC read buffer interrupt*/
```

```
fac_interrupt_enable(FAC_CTL_RIE);
```

函数 **fac_interrupt_disable**

函数fac_interrupt_disable描述见下表:

表 3-432. 函数 fac_interrupt_disable

函数名称	fac_interrupt_disable
函数原型	void fac_interrupt_disable(uint32_t interrupt);
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
<i>FAC_CTL_RIE</i>	FAC读缓冲区中断
<i>FAC_CTL_WIE</i>	FAC写缓冲区中断
<i>FAC_CTL_OFEIE</i>	FAC上溢错误中断
<i>FAC_CTL_UFEIE</i>	FAC下溢错误中断
<i>FAC_CTL_STEIE</i>	FAC饱和错误中断
<i>FAC_CTL_GSTEIE</i>	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 3-433. 函数 **fac_interrupt_flag_get**

函数名称	fac_interrupt_disable
函数原型	FlagStatus fac_interrupt_flag_get(uint8_t interrupt);
功能描述	获取FAC中断标志
先决条件	-
被调用函数	-
输入参数{in}	
interrupt	FAC中断
FAC_INT_FLAG_YBEF	Y缓冲区空中断标志
FAC_INT_FLAG_X0BFF	X0缓冲区满中断标志
FAC_INT_FLAG_OFEF	上溢错误中断标志
FAC_INT_FLAG_UFEF	下溢错误中断标志
FAC_INT_FLAG_STEF	饱和错误中断标志
FAC_INT_FLAG_GSTEF	增益饱和和错误中断标志
输出参数{out}	
-	-
返回值	
FlagStatus	SET/RESET

例如：

```
/* get Y buffer empty flag status */
```

```
fac_interrupt_flag_get(FAC_INT_FLAG_YBEF);
```

函数 **fac_flag_get**

函数 **fac_flag_get** 描述见下表：

表 3-434. 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);
```

3.15. FFT

快速傅里叶变换（FFT）是离散傅里叶变换（DFT）的高效计算方法。该模块可以进行FFT运算，减轻了CPU负担。与软件实现相比，该模块可以加速了FFT的计算时间。该模块支持6个可配置的FFT点数，最多1024，输入和输出数据应为IEEE-754单精度浮点复数。章节[3.15.1](#)描述了FFT的寄存器列表，章节[3.15.2](#)对FFT库函数进行说明。

3.15.1. 外设寄存器说明

FFT寄存器列表如下表所示：

表 3-435. FFT 寄存器

寄存器名称	寄存器描述
FFT_CSRT	控制和状态寄存器
FFT_RESADDR	实部基地址寄存器
FFT_IMSADDR	虚部基地址寄存器
FFT_WSADDR	窗函数基地址寄存器
FFT_OSADDR	输出基地址寄存器
FFT_LOOPLEN	循环长度寄存器

3.15.2. 外设库函数说明

FFT库函数列表如下表所示：

表 3-436. FFT 库函数

库函数名称	库函数描述
fft_deinit	FFT外设复位
fft_struct_para_init	将FFT初始化结构体中所有参数初始化为默认值
fft_init	FFT初始化
fft_calculation_start	启动FFT计算
fft_calculation_stop	停止FFT计算
fft_point_number_config	配置FFT点数
fft_mode_config	配置FFT模式
fft_window_enable	使能FFT窗函数
fft_window_disable	禁能FFT窗函数
fft_downsample_config	配置FFT下采样点数
fft_image_source_config	配置FFT虚部数据源
fft_real_addr_config	配置FFT实部基地址
fft_image_addr_config	配置FFT虚部基地址
fft_window_addr_config	配置FFT窗函数基地址
fft_output_addr_config	配置FFT输出基地址
fft_loop_buffer_length_config	配置FFT循环缓冲长度
fft_loop_buffer_index_config	配置FFT循环缓冲索引
fft_flag_get	获取FFT标志
fft_flag_clear	清除FFT标志
fft_interrupt_enable	使能FFT中断
fft_interrupt_disable	禁能FFT中断
fft_interrupt_flag_get	获取FFT中断标志位
fft_interrupt_flag_clear	清除FFT中断标志位

结构体 fft_parameter_struct

表 3-437. 结构体 fft_parameter_struct

成员名称	功能描述
mode_sel	FFT模式配置（FFT_MODE, IFFT_MODE）
point_num	FFT点数选择（FFT_POINT_x, x = 32, 64, 128, 256, 512, 1024）
window_enable	FFT窗函数使能（FFT_WINDOW_ENABLE, FFT_WINDOW_DISABLE）
downsamp_sel	FFT输入数据下采样选择（FFT_DOWNSAMPLE_x, x = 1..16）
image_source	FFT虚部输入源选择（FFT_IM_MEMORY, FFT_IM_ZERO, FFT_IM_MEMORY_OPPOSITE）
loopbuf_len	FFT循环缓冲长度
loopbuf_index	FFT循环缓冲索引
real_addr	FFT实部基地址
image_addr	FFT虚部基地址
window_addr	FFT窗函数基地址

函数 `fft_deinit`

函数`fft_deinit`描述见下表:

表 3-438. 函数 `fft_deinit`

函数名称	<code>fft_deinit</code>
函数原型	<code>void fft_deinit(void);</code>
功能描述	FFT外设复位
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 FFT */
```

```
fft_deinit();
```

函数 `fft_struct_para_init`

函数`fft_struct_para_init`描述见下表:

表 3-439. 函数 `fft_struct_para_init`

函数名称	<code>fft_struct_para_init</code>
函数原型	<code>void fft_struct_para_init(fft_parameter_struct* fft_parameter);</code>
功能描述	将FFT初始化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}	
<code>fft_parameter</code>	FFT初始化结构体指针, 结构体成员可以参考 结构体<code>fft_parameter_struct</code> 。
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* initialize the FFT filter parameter struct */
```

```
fft_parameter_struct fftconfig;
```

```
fft_struct_para_init(&fftconfig);
```


函数 `fft_init`

函数`fft_init`描述见下表:

表 3-440. 函数 `fft_init`

函数名称	<code>fft_init</code>
函数原型	<code>void fft_init(fft_parameter_struct* fft_parameter);</code>
功能描述	FFT初始化
先决条件	-
被调用函数	-
输入参数{in}	
<code>fft_parameter</code>	FFT初始化结构体指针, 结构体成员可以参考 结构体<code>fft_parameter_struct</code> 。
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* config FFT parameter */
fft_parameter_struct fftconfig;

float fft_real_buf[32] = {0};

float fft_output_buf[32] = {0};

fft_struct_para_init(&fftconfig);

fftconfig.mode_sel      = FFT_MODE;

fftconfig.point_num     = FFT_POINT_1024;

fftconfig.window_enable = FFT_WINDOW_DISABLE;

fftconfig.downsamp_sel  = FFT_DOWNSAMPLE_8;

fftconfig.image_source   = FFT_IM_ZERO;

fftconfig.loopbuf_len   = 32U;

fftconfig.loopbuf_index = 0U;

fftconfig.real_addr      = (uint32_t)fft_real_buf;

fftconfig.image_addr     = NULL;

fftconfig.window_addr    = NULL;

fftconfig.output_addr    = (uint32_t)fft_output_buf;

fft_init(&fftconfig);
```

函数 `fft_calculation_start`

函数 `fft_calculation_start` 描述见下表：

表 3-441. 函数 `fft_calculation_start`

函数名称	<code>fft_calculation_start</code>
函数原型	<code>void fft_calculation_start(void);</code>
功能描述	启动FFT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：

```
/* start FFT calculation */
fft_calculation_start();
```

函数 `fft_calculation_stop`

函数 `fft_calculation_stop` 描述见下表：

表 3-442. 函数 `fft_calculation_stop`

函数名称	<code>fft_calculation_stop</code>
函数原型	<code>void fft_calculation_stop(void);</code>
功能描述	停止FFT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：

```
/* stop FFT calculation */
fft_calculation_stop();
```

函数 **fft_point_number_config**

函数fft_point_number_config描述见下表:

表 3-443. 函数 **fft_point_number_config**

函数名称	fft_point_number_config
函数原型	void fft_point_number_config(uint8_t point_num);
功能描述	配置FFT点数
先决条件	-
被调用函数	-
输入参数{in}	
point_num	点数
<i>FFT_POINT_32</i>	FFT点数是32
<i>FFT_POINT_64</i>	FFT点数是64
<i>FFT_POINT_128</i>	FFT点数是128
<i>FFT_POINT_256</i>	FFT点数是256
<i>FFT_POINT_512</i>	FFT点数是512
<i>FFT_POINT_1024</i>	FFT点数是1024
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* configure FFT calculation point number */
fft_point_number_config (FFT_POINT_1024);
```

函数 **fft_mode_config**

函数fft_mode_config描述见下表:

表 3-444. 函数 **fft_mode_config**

函数名称	fft_mode_config
函数原型	void fft_mode_config(uint32_t mode);
功能描述	配置FFT模式
先决条件	-
被调用函数	-
输入参数{in}	
mode	calculation mode
<i>FFT_MODE</i>	excute FFT operation
<i>IFFT_MODE</i>	excute FFT inverse operation
输出参数{out}	
-	-

返回值	
-	-

例如:

```
/* excute FFT operation mode */
```

```
fft_mode_config (FFT_MODE);
```

函数 **fft_window_enable**

函数fft_window_enable描述见下表:

表 3-445. 函数 fft_window_enable

函数名称	fft_window_enable
函数原型	void fft_window_enable(void);
功能描述	使能FFT窗函数
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 FFT window function */
```

```
fft_window_enable();
```

函数 **fft_window_disable**

函数fft_window_disable描述见下表:

表 3-446. 函数 fft_window_disable

函数名称	fft_window_disable
函数原型	void fft_window_disable(void);
功能描述	禁能FFT窗函数
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 FFT window function */
```

```
fft_window_disable();
```

函数 `fft_downsample_config`

函数`fft_downsample_config`描述见下表:

表 3-447. 函数 `fft_downsample_config`

函数名称	<code>fft_downsample_config</code>
函数原型	<code>void fft_downsample_config(uint8_t sample_value);</code>
功能描述	配置FFT下采样点数
先决条件	-
被调用函数	-
输入参数{in}	
<code>sample_value</code>	下采样值
<code>FFT_DOWNSAMPLE</code> <code>E_x</code> ($x=1..16$)	下采样值, $x=1..16$
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* configure FFT down sample value */
```

```
fft_downsample_config (FFT_DOWNSAMPLE_16);
```

函数 `fft_image_source_config`

函数`fft_image_source_config`描述见下表:

表 3-448. 函数 `fft_image_source_config`

函数名称	<code>fft_image_source_config</code>
函数原型	<code>void fft_image_source_config(uint8_t im_src);</code>
功能描述	配置FFT虚部数据源
先决条件	-
被调用函数	-
输入参数{in}	
<code>im_src</code>	虚部数据源类型
<code>FFT_IM_MEMORY</code>	虚部输入来源于FFT_IMSADDR

<i>FFT_IM_ZERO</i>	虚部输入为0
<i>FFT_IM_MEMORY_OPPOSITE</i>	虚部输入是FFT_IMSADDR的相反数
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* configure FFT image part source */
fft_image_source_config (FFT_IM_MEMORY);
```

函数 `fft_real_addr_config`

函数`fft_real_addr_config`描述见下表:

表 3-449. 函数 `fft_real_addr_config`

函数名称	<code>fft_real_addr_config</code>
函数原型	<code>void fft_real_addr_config(uint32_t addr);</code>
功能描述	配置FFT实部基地址
先决条件	-
被调用函数	-
输入参数{in}	
addr	实部基地址
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* configure real part start address */
float fft_real_buf[32] = {0};
fft_real_addr_config ((uint32_t)fft_real_buf);
```

函数 `fft_image_addr_config`

函数`fft_image_addr_config`描述见下表:

表 3-450. 函数 `fft_image_addr_config`

函数名称	<code>fft_image_addr_config</code>
函数原型	<code>void fft_image_addr_config(uint32_t addr);</code>
功能描述	配置FFT虚部基地址
先决条件	-

被调用函数	-
输入参数{in}	
addr	虚部基地址
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* configure image part start address */
float fft_image_buf[32] = {0};
fft_image_addr_config ((uint32_t)fft_image_buf);
```

函数 fft_window_addr_config

函数fft_window_addr_config描述见下表:

表 3-451. 函数 fft_window_addr_config

函数名称	fft_window_addr_config
函数原型	void fft_window_addr_config(uint32_t addr);
功能描述	配置FFT窗函数基地址
先决条件	-
被调用函数	-
输入参数{in}	
addr	窗函数基地址
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* configure window part start address */
float fft_window_buf[32] = {0};
fft_window_addr_config ((uint32_t)fft_window_buf);
```

函数 fft_output_addr_config

函数fft_output_addr_config描述见下表:

表 3-452. 函数 fft_output_addr_config

函数名称	fft_output_addr_config
函数原型	void fft_output_addr_config(uint32_t addr);

功能描述	configure output start address
先决条件	-
被调用函数	-
输入参数{in}	
addr	输出基地址
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* configure output start address */
```

```
float fft_output_buf[32] = {0};
```

```
fft_output_addr_config ((uint32_t)fft_output_buf);
```

函数 fft_loop_buffer_length_config

函数fft_loop_buffer_length_config描述见下表:

表 3-453. 函数 fft_loop_buffer_length_config

函数名称	fft_loop_buffer_length_config
函数原型	void fft_loop_buffer_length_config(uint16_t length);
功能描述	配置FFT循环缓冲长度
先决条件	-
被调用函数	-
输入参数{in}	
length	循环缓冲区长度
0x0000~0xFFFF	循环缓冲区长度值
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* configure loop buffer length */
```

```
fft_loop_buffer_length_config (10U);
```

函数 fft_loop_buffer_index_config

函数fft_loop_buffer_index_config描述见下表:

表 3-454. 函数 fft_loop_buffer_index_config

函数名称	fft_loop_buffer_index_config
------	------------------------------

函数原型	void fft_loop_buffer_index_config(uint16_t index);
功能描述	配置FFT循环缓冲索引
先决条件	-
被调用函数	-
输入参数{in}	
index	索引值
0x0000~LENGTH[15:0]	不能超过LENGTH[15:0].
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* configure loop buffer index value*/
```

```
fft_loop_buffer_index_config (0U);
```

函数 fft_flag_get

函数fft_flag_get描述见下表:

表 3-455. 函数 fft_flag_get

函数名称	fft_flag_get
函数原型	FlagStatus fft_flag_get(uint32_t flag);
功能描述	获取FFT标志
先决条件	-
被调用函数	-
输入参数{in}	
flag	标志位
FFT_FLAG_DMABSY	DMA忙标志
FFT_FLAG_CCF	FFT计算完成标志
FFT_FLAG_TAEIF	FFT传输错误标志
输出参数{out}	
-	-
返回值	
FlagStatus	SET/RESET

例如:

```
/* get FFT flag status */
```

```
FlagStatus flag_value;
```

```
flag_value = fft_flag_get(FFT_FLAG_DMABSY);
```

函数 **fft_flag_clear**

函数fft_flag_clear描述见下表:

表 3-456. 函数 **fft_flag_clear**

函数名称	fft_flag_clear
函数原型	void fft_flag_clear(uint32_t flag);
功能描述	清除FFT标志
先决条件	-
被调用函数	-
输入参数{in}	
flag	标志位
FFT_FLAG_CCF	FFT计算完成标志
FFT_FLAG_TAEIF	FFT传输错误标志
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* clear FFT flag status */
fft_flag_clear (FFT_FLAG_CCF);
```

函数 **fft_interrupt_enable**

函数fft_interrupt_enable描述见下表:

表 3-457. 函数 **fft_interrupt_enable**

函数名称	fft_interrupt_enable
函数原型	void fft_interrupt_enable(uint32_t fft_interrupt);
功能描述	使能FFT中断
先决条件	-
被调用函数	-
输入参数{in}	
fft_interrupt	FFT中断
FFT_INT_CCF	FFT计算完成中断
FFT_INT_TAEIF	FFT传输错误中断
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* enable FFT calculation completion interrupt */
```

fft_interrupt_enable (FFT_INT_CCF);

函数 fft_interrupt_disable

函数fft_interrupt_disable描述见下表：

表 3-458. 函数 fft_interrupt_disable

函数名称	fft_interrupt_disable
函数原型	void fft_interrupt_disable (uint32_t fft_interrupt);
功能描述	禁能FFT中断
先决条件	-
被调用函数	-
输入参数{in}	
fft_interrupt	FFT中断
FFT_INT_CCF	FFT计算完成中断
FFT_INT_TAEIF	FFT传输错误中断
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* disable FFT calculation completion interrupt */
```

```
fft_interrupt_disable (FFT_INT_CCF);
```

函数 fft_interrupt_flag_get

函数fft_interrupt_flag_get描述见下表：

表 3-459. 函数 fft_interrupt_flag_get

函数名称	fft_interrupt_flag_get
函数原型	FlagStatus fft_interrupt_flag_get(uint32_t int_flag);
功能描述	获取FFT中断标志
先决条件	-
被调用函数	-
输入参数{in}	
int_flag	FFT中断标志
FFT_INT_FLAG_CCF	FFT计算完成中断标志
FFT_INT_FLAG_TAEIF	FFT传输完成中断标志
输出参数{out}	
-	-
返回值	

FlagStatus	SET/RESET
------------	-----------

例如:

```
/* get FFT interrupt flag status */
```

```
FlagStatus flag_value;
```

```
flag_value = fft_interrupt_flag_get(FFT_INT_FLAG_CCF);
```

函数 `fft_interrupt_flag_clear`

函数`fft_interrupt_flag_clear`描述见下表:

表 3-460. 函数 `fft_interrupt_flag_clear`

函数名称	<code>fft_interrupt_flag_clear</code>
函数原型	<code>void fft_interrupt_flag_clear(uint32_t int_flag);</code>
功能描述	清除FFT中断标志位
先决条件	-
被调用函数	-
输入参数{in}	
<code>int_flag</code>	FFT中断标志
<code>FFT_INT_FLAG_CCF</code>	FFT计算完成中断标志
<code>FFT_INT_FLAG_TAEIF</code>	FFT传输完成中断标志
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* clear FFT interrupt flag bit */
```

```
fft_interrupt_flag_clear(FFT_INT_FLAG_CCF);
```

3.16. FMC

FMC是MCU中的Flash控制器，其中包括存储数据的主编程块和选项字节。章节[3.16.1](#)描述了FMC的寄存器列表，章节[3.16.2](#)对FMC库函数进行说明。

3.16.1. 外设寄存器说明

FMC寄存器列表如下表所示:

表 3-461. FMC 寄存器

寄存器名称	寄存器描述
FMC_WS	等待状态寄存器
FMC_RUNKEY	系统运行时闪存的模式解锁寄存器
FMC_KEY	解锁寄存器
FMC_OBKEY	选项字节操作解锁寄存器
FMC_STAT	状态寄存器
FMC_CTL	控制寄存器
FMC_ECCCS	ECC控制与状态寄存器
FMC_OBCTL	选项字节控制寄存器
FMC_DCRP_SADD R0	DCRP0起始地址寄存器
FMC_DCRP_EADD R0	DCRP0结束地址寄存器
FMC_BK0WP0	Bank0擦除/编程保护区域0寄存器
FMC_BK0WP1	Bank0擦除/编程保护区域1寄存器
FMC_DCRP_SADD R1	DCRP1起始地址寄存器
FMC_DCRP_EADD R1	DCRP1结束地址寄存器
FMC_BK1WP0	Bank1擦除/编程保护区域0寄存器
FMC_BK1WP1	Bank1擦除/编程保护区域1寄存器
FMC_BK0SCR	Bank0安全用户区域寄存器
FMC_BK1SCR	Bank1安全用户区域寄存器
FMC_PID	产品ID寄存器

3.16.2. 外设库函数说明

FMC库函数列表如下表所示：

表 3-462. 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_wscnt_set	设置FMC等待状态
fmc_prefetch_enable	使能预取功能
fmc_prefetch_disable	失能预取功能
fmc_icache_enable	使能指令缓存区
fmc_icache_disable	失能指令缓存区
fmc_icache_reset	复位指令缓存区
fmc_dcache_enable	使能数据缓存区
fmc_dcache_disable	失能数据缓存区
fmc_dcache_reset	复位指令缓存区

库函数名称	库函数描述
fmc_page_erase	页擦除
fmc_bank0_erase	Bank0擦除
fmc_bank1_erase	Bank1擦除
fmc_mass_erase	整片擦除
fmc_doubleword_program	双字编程
fmc_bank0_base_address_get	获取bank0基地址
fmc_bank1_base_address_get	获取bank1基地址
fmc_page_size_get	获取页大小
fmc_debugger_enable	使能调试
fmc_debugger_disable	禁用调试
fmc_slp_pd_mode_enable	配置当系统进入睡眠模式时，闪存仍为空闲模式
fmc_slp_pd_mode_disable	配置当系统进入睡眠模式时，闪存进入掉电模式
fmc_scr_area_enable	安全用户区域使能
ob_unlock	解锁FMC_OBCTL寄存器
ob_lock	锁定FMC_OBCTL寄存器
ob_reload	选项字节重加载
ob_user_write	用户选项字节编程
ob_security_protection_config	配置安全保护等级
ob_dcrp_config	配置DCRP区域
ob_write_protection_config	配置写保护区域
ob_scr_area_config	配置安全用户区域
ob_boot_lock_config	配置选项字节boot锁定位
ob_bank_memory_swap_config	FMC存储器映射切换
ob_user_get	获取用户选项字节值
ob_bor_threshold_get	获取BOR阈值
ob_security_protection_level_get	获取安全保护等级
ob_dcrp_area_get	获取DCRP区域
ob_write_protection_get	获取写保护区域
ob_scr_area_size_get	获取安全用户区域
ob_boot_config_get	读取boot锁定配置
fmc_ecc_flag_get	获取ECC标志位
fmc_ecc_flag_clear	清除ECC标志位
fmc_eccor_interrupt_enable	使能ECC中断
fmc_eccor_interrupt_disable	失能ECC中断
fmc_eccor_interrupt_flag_get	获取ECC中断标志位
fmc_eccor_interrupt_flag_clear	清除ECC中断标志位
fmc_flag_get	获取FMC标志位
fmc_flag_clear	清除FMC标志位
fmc_interrupt_enable	使能FMC中断
fmc_interrupt_disable	失能FMC中断
fmc_interrupt_flag_get	获取FMC中断标志位

库函数名称	库函数描述
fmc_interrupt_flag_clear	清除FMC中断标志位
fmc_state_get	获取FMC状态
fmc_ready_wait	检测FMC等待状态
fmc_pd_mode_enter	配置当系统在运行模式时，闪存仍为空闲模式
fmc_pd_mode_exit	配置当系统在运行模式时，闪存进入掉电模式
ob_bank_mode_config	配置选项字节单双bank模式

枚举类型 fmc_state_enum

表 3-463. 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_OBERR	选项字节读错误
FMC_RPERR	读保护错误
FMC_PGSERR	编程顺序错误
FMC_PGMERR	编程大小错误
FMC_PGAERR	编程对齐错误
FMC_WPERR	擦除/编程保护错误
FMC_PGERR	编程错误标志
FMC_OPRERR	操作失败错误
FMC_TOERR	超时错误
FMC_OB_HSPC	高保护等级标志
FMC_OB_LSPC	低保护等级标志

函数 fmc_unlock

函数fmc_unlock描述见下表：

表 3-464. 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 3-465. 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_wscnt_set

函数fmc_wscnt_set描述见下表

表 3-466. 函数 fmc_wscnt_set

函数名称	fmc_wscnt_set
函数原型	void fmc_wscnt_set(uint32_t wscnt);
功能描述	设置等待状态时间
先决条件	-
被调用函数	-
输入参数{in}	
wscnt	等待状态时间
FMC_WAIT_STATE_0	增加0个等待状态
FMC_WAIT_STATE_1	增加1个等待状态
FMC_WAIT_STATE_2	增加2个等待状态
FMC_WAIT_STATE_3	增加3个等待状态
FMC_WAIT_STATE_4	增加4个等待状态
FMC_WAIT_STATE_5	增加5个等待状态
FMC_WAIT_STATE_6	增加6个等待状态

FMC_WAIT_STATE_7	增加7个等待状态
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* set the wait state 0 */
```

```
fmc_wscnt_set(FMC_WAIT_STATE_0);
```

函数 fmc_prefetch_enable

函数fmc_prefetch_enable描述见下表：

表 3-467. 函数 fmc_prefetch_enable

函数名称	fmc_prefetch_enable
函数原型	void fmc_prefetch_enable(void);
功能描述	使能预取
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 pre-fetch */
```

```
fmc_prefetch_enable( );
```

函数 fmc_prefetch_disable

函数fmc_prefetch_disable描述见下表：

表 3-468. 函数 fmc_prefetch_disable

函数名称	fmc_prefetch_disable
函数原型	void fmc_prefetch_disable (void);
功能描述	失能预取
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 pre-fetch */
fmc_prefetch_disable( );
```

函数 fmc_icache_enable

函数fmc_icache_enable描述见下表:

表 3-469. 函数 fmc_icache_enable

函数名称	fmc_icache_enable
函数原型	void fmc_icache_enable(void);
功能描述	使能指令cache
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 icache */
fmc_icache_enable( );
```

函数 fmc_icache_disable

函数fmc_icache_disable描述见下表:

表 3-470. 函数 fmc_icache_disable

函数名称	fmc_icache_disable
函数原型	void fmc_icache_disable (void);
功能描述	失能指令cache
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 icache */
```

```
fmc_icache_disable( );
```

函数 fmc_icache_reset

函数fmc_icache_reset_enable描述见下表:

表 3-471. 函数 fmc_icache_reset_enable

函数名称	fmc_icache_reset
函数原型	void fmc_icache_reset (void);
功能描述	复位指令cache
先决条件	-
被调用函数	-
输入参数{in}	
-	-
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* icache reset */
```

```
fmc_icache_reset( );
```

函数 fmc_dcache_enable

函数fmc_dcache_enable描述见下表:

表 3-472. 函数 fmc_dcache_enable

函数名称	fmc_dcache_enable
函数原型	void fmc_dcache_enable(void);
功能描述	使能数据cache
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 dcache */
fmc_dcache_enable( );
```

函数 fmc_dcache_disable

函数fmc_dcache_disable描述见下表：

表 3-473. 函数 fmc_dcache_disable

函数名称	fmc_dcache_disable
函数原型	void fmc_dcache_disable (void);
功能描述	失能数据cache
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 dcache */
fmc_dcache_disable( );
```

函数 fmc_dcache_reset

函数fmc_dcache_reset_enable描述见下表：

表 3-474. 函数 fmc_dcache_reset_enable

函数名称	fmc_dcache_reset
函数原型	void fmc_dcache_reset (void);
功能描述	复位数据cache
先决条件	-
被调用函数	-
输入参数{in}	
-	-
输出参数{out}	
-	-
返回值	

例如:

```
/* dcache reset */
```

```
fmc_dcache_reset( );
```

函数 fmc_page_erase

函数fmc_page_erase描述见下表

表 3-475. 函数 fmc_page_erase

函数名称	fmc_page_erase
函数原型	fmc_page_erase(uint32_t bank, uint32_t page_number_in_bank)
功能描述	FMC页擦除
先决条件	fmc_unlock
被调用函数	-
输入参数{in}	
bank	擦除页所在的bank号
FMC_BANK0	页码处于Bank0
FMC_BANK1	页码处于Bank1
输入参数{in}	
page_number_in_bank	擦除的页码
输出参数{out}	
-	-
返回值	
fmc_state_enum	FMC状态, 参考 表3-463. 枚举类型fmc_state_enum

例如:

```
fmc_unlock();
```

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

```
fmc_state_enum state = fmc_page_erase(FMC_BANK0, 0);
```

函数 fmc_bank0_erase

函数fmc_bank0_erase描述见下表

表 3-476. 函数 fmc_bank0_erase

函数名称	fmc_bank0_erase
函数原型	fmc_state_enum fmc_bank0_erase(void);
功能描述	FMC bank0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3-463. 枚举类型fmc_state_enum

例如：

```
fmc_unlock();
```

```
/* erase bank0 */
```

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

函数 fmc_bank1_erase

函数fmc_bank1_erase描述见下表

表 3-477. 函数 fmc_bank1_erase

函数名称	fmc_bank1_erase
函数原型	fmc_state_enum fmc_bank1_erase(void);
功能描述	FMC bank1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3-463. 枚举类型fmc_state_enum

例如：

```
fmc_unlock();
```

```
/* erase bank1 */
```

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

函数 fmc_mass_erase

函数fmc_mass_erase描述见下表

表 3-478. 函数 fmc_mass_erase

函数名称	fmc_mass_erase
函数原型	fmc_state_enum fmc_mass_erase(void);
功能描述	FMC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3-463. 枚举类型fmc_state_enum

例如:

```
fmc_unlock();

/* erase whole chip */

fmc_state_enum state = fmc_mass_erase( );
```

函数 fmc_doubleword_program

函数fmc_doubleword_program描述见下表

表 3-479. 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3-463. 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_bank0_base_address_get

函数fmc_bank0_base_address_get描述见下表

表 3-480. 函数 fmc_bank0_base_address_get

函数名称	fmc_bank0_base_address_get
函数原型	uint32_t fmc_bank0_base_address_get(void);
功能描述	获取bank0基地址
先决条件	-
被调用函数	-
输入参数{in}	
-	-
输出参数{out}	
-	-
返回值	
uint32_t	Bank0基地址

例如:

```
/* get base address of bank0 */  
  
uint32_t base_addr;  
  
base_addr = fmc_bank0_base_address_get( );
```

函数 fmc_bank1_base_address_get

函数fmc_bank1_base_address_get描述见下表

表 3-481. 函数 fmc_bank1_base_address_get

函数名称	fmc_bank1_base_address_get
函数原型	uint32_t fmc_bank1_base_address_get(void);
功能描述	获取bank1基地址
先决条件	-
被调用函数	-
输入参数{in}	
-	-
输出参数{out}	
-	-
返回值	
uint32_t	Bank1基地址

例如:

```
/* get base address of bank1 */  
  
uint32_t base_addr;
```



```
base_addr = fmc_bank1_base_address_get( );
```

函数 fmc_page_size_get

函数fmc_page_size_get描述见下表

表 3-482. 函数 fmc_page_size_get

函数名称	fmc_page_size_get
函数原型	uint32_t fmc_page_size_get(void);
功能描述	获取页大小
先决条件	-
被调用函数	-
输入参数{in}	
-	-
输出参数{out}	
-	-
返回值	
uint32_t	页大小

例如:

```
/* get page size */
uint32_t page_size;
page_size = fmc_page_size_get( );
```

函数 fmc_debugger_enable

函数fmc_debugger_enable描述见下表

表 3-483. 函数 fmc_debugger_enable

函数名称	fmc_debugger_enable
函数原型	void fmc_debugger_enable(void);
功能描述	使能debug
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 debugger */
```

fmc_debugger_enable();

函数 fmc_debugger_disable

函数fmc_debugger_disable描述见下表

表 3-484. 函数 fmc_debugger_disable

函数名称	fmc_debugger_disable
函数原型	void fmc_debugger_disable(void);
功能描述	失能debug
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 debugger */
```

```
fmc_debugger_disable( );
```

函数 fmc_slp_pd_mode_enable

函数fmc_slp_pd_mode_enable描述见下表

表 3-485. 函数 fmc_slp_pd_mode_enable

函数名称	fmc_slp_pd_mode_enable
函数原型	void fmc_slp_pd_mode_enable(void);
功能描述	当系统进入睡眠模式时，flash进入掉电模式
先决条件	-
被调用函数	-
输入参数{in}	
-	-
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* flash enter power down mode when MCU enter sleep mode */
```

```
fmc_slp_pd_mode_enable( );
```

函数 `fmc_slp_pd_mode_disable`

函数 `fmc_slp_pd_mode_disable` 描述见下表

表 3-486. 函数 `fmc_slp_pd_mode_disable`

函数名称	<code>fmc_slp_pd_mode_disable</code>
函数原型	<code>void fmc_slp_pd_mode_disable (void);</code>
功能描述	当系统进入睡眠模式时，flash 仍为空闲模式
先决条件	-
被调用函数	-
输入参数{in}	
-	-
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* flash enter idle mode when MCU enter sleep mode */
```

```
fmc_slp_pd_mode_disable( );
```

函数 `fmc_scr_area_enable`

函数 `fmc_scr_area_enable` 描述见下表

表 3-487. 函数 `fmc_scr_area_enable`

函数名称	<code>fmc_scr_area_enable</code>
函数原型	<code>void fmc_scr_area_enable(uint32_t scr_area);</code>
功能描述	使能SCR区域
先决条件	<code>fmc_unlock</code>
被调用函数	-
输入参数{in}	
<code>scr_area</code>	安全用户区域
<code>SCR_AREA0</code>	Bank0安全用户区域
<code>SCR_AREA1</code>	Bank1安全用户区域
输出参数{out}	
-	-
返回值	
-	-

例如：

```
fmc_unlock();
```

```
/* enable secure area protection */
```

```
fmc_scr_area_enable(SCR_AREA0);
```

函数 ob_unlock

函数ob_unlock描述见下表

表 3-488. 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 3-489. 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：

```
fmc_unlock();
```

```
/*lock the option bytes operation */
```

```
ob_lock();
```

函数 ob_reload

函数ob_reload描述见下表

表 3-490. 函数 ob_reload

函数名称	ob_reload
函数原型	void ob_reload(void);
功能描述	重载入选项字节
先决条件	ob_unlock
被调用函数	-
输入参数{in}	
-	-
输出参数{out}	
-	-
返回值	
-	-

例如:

```
fmc_unlock();
```

```
ob_unlock();
```

```
/* reload the option bytes operation */
```

```
ob_reload( );
```

函数 ob_user_write

函数ob_user_write描述见下表

表 3-491. 函数 ob_user_write

函数名称	ob_user_write
函数原型	fmc_state_enum ob_user_write(uint32_t ob_user, uint32_t ob_user_mask);
功能描述	修改用户选项字节值
先决条件	ob_unlock
被调用函数	fmc_ready_wait
输入参数{in}	
ob_user	用户选项字节
OB_NRST_PIN_INPUT_MODE	复位引脚输入复位信号模式
OB_NRST_PIN_NORMAL_GPIO	复位引脚仅为普通GPIO口

OB_NRST_PIN_INPUT_OUTPUT_MODE	复位引脚既输入复位信号也输出复位脉冲模式
OB_NBOOT0_VALUE_0	Boot0为1
OB_NBOOT0_VALUE_1	Boot0为0
OB_NSWBOOT0_FROM_OB_BOOT0	由选项字节决定boot
OB_NSWBOOT0_FROM_PIN	由PB8/Boot0决定复位
OB_TCMRAM_ERASE_ENABLE	系统复位时擦除TCM SRAM
OB_TCMRAM_ERASE_DISABLE	系统复位时不擦除TCM SRAM
OB_SRAM_ECC_ENABLE	使能SRAM与TCM SRAM ECC
OB_SRAM_ECC_DISABLE	失能SRAM与TCM SRAM ECC
OB_NBOOT1_VALUE_0	BOOT1为1
OB_NBOOT1_VALUE_1	BOOT1为0
OB_SINGLE_BANK_MODE	选择单bank模式，128位读取位宽
OB_DUAL_BANK_MODE	选择双bank模式，64位读取位宽
OB_BB_DISABLE	当配置为从主闪存启动时，从bank0启动
OB_BB_ENABLE	当配置为从主内存启动时，从bank1启动，若bank1为空则从bank0启动。(或者当bank0，bank1均空且SPC等级不为高时从bootloader启动)
OB_STDBY_FWDGT_SUSPEND	当系统处于待机模式时独立看门狗暂停运行
OB_STDBY_FWDGT_RUN	当系统处于待机模式时独立看门狗继续运行
OB_DPSLP_FWDGT_SUSPEND	当系统处于深度睡眠模式时独立看门狗暂停运行
OB_DPSLP_FWDGT_RUN	当系统处于深度睡眠模式时独立看门狗继续运行
OB_FWDGT_HW	硬件独立看门狗
OB_FWDGT_SW	软件独立看门狗
OB_STDBY_RST	当系统进入待机模式时复位
OB_STDBY_NRST	当系统进入待机模式时不复位
OB_DEEPSLEEP_RST	当系统进入深度睡眠模式时复位

<i>OB_DEEPSLEEP_NRS</i> <i>T</i>	当系统进入深度睡眠模式时不复位
<i>OB_BOR_TH_VALUE0</i>	BOR阈值0, 阈值约1.7 V
<i>OB_BOR_TH_VALUE1</i>	BOR阈值1, 阈值约2.0 V
<i>OB_BOR_TH_VALUE2</i>	BOR阈值2, 阈值约2.2 V
<i>OB_BOR_TH_VALUE3</i>	BOR阈值3, 阈值约2.5 V
输入参数{in}	
ob_user_mask	用户选项字节掩码
<i>FMC_OBCTL_NRST_</i> <i>MDSEL</i>	复位引脚模式掩码
<i>FMC_OBCTL_NBOOT</i> <i>0</i>	选项字节NBOOT0掩码
<i>FMC_OBCTL_NSWBT</i> <i>0</i>	软件boot掩码
<i>FMC_OBCTL_TCMSTR</i> <i>AM_ERS</i>	系统复位时TCM_SRAM复位掩码
<i>FMC_OBCTL_SRAM_</i> <i>ECCEN</i>	SRAM_ECC使能掩码
<i>FMC_OBCTL_NBOOT</i> <i>1</i>	选项字节NBOOT1掩码
<i>FMC_OBCTL_DBS</i>	Bank模式掩码
<i>FMC_OBCTL_BB</i>	双bank boot掩码
<i>FMC_OBCTL_FWDGS</i> <i>PD_STDBY</i>	系统待机模式时独立看门狗运行模式掩码
<i>FMC_OBCTL_FWDGS</i> <i>PD_DPSLP</i>	系统深度睡眠模式时独立看门狗运行模式掩码
<i>FMC_OBCTL_NFWDG</i> <i>_HW</i>	软件独立看门狗配置位掩码
<i>FMC_OBCTL_NRST_S</i> <i>TDBY</i>	系统待机模式时复位掩码
<i>FMC_OBCTL_NRST_D</i> <i>PSLP</i>	系统深度睡眠模式复位掩码
<i>FMC_OBCTL_BOR_TH</i>	选项字节BOR阈值掩码
输出参数{out}	
-	-
返回值	
fmc_state_enum	FMC状态, 参考 表3-463. 枚举类型fmc_state_enum

例如:

```
fmc_state_enum fmc_state;
```

```
fmc_unlock();
```

```
ob_unlock();
```

```
/* modify option byte */
```

```
fmc_state = ob_user_write (OB_BB_ENABLE, FMC_OBCTL_BB);
```

函数 ob_security_protection_config

函数ob_security_protection_config描述见下表

表 3-492. 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
先决条件	ob_unlock
被调用函数	fmc_ready_wait
输入参数{in}	
ob_spc	安全保护等级
FMC_NSPC	无安全保护
FMC_LSPC	低安全保护
FMC_HSPC	高安全保护
输出参数{out}	
-	-
返回值	
fmc_state_enum	FMC状态，参考 表3-463. 枚举类型fmc_state_enum

例如：

```
fmc_state_enum fmc_state;
```

```
fmc_unlock();
```

```
ob_unlock();
```

```
/* disable security protection */
```

```
fmc_state = ob_security_protection_config(FMC_NSPC);
```

函数 ob_dcrp_config

函数ob_dcrp_config描述见下表

表 3-493. 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_area, uint32_t dcrp_eren, uint32_t dcrp_start_addr, uint32_t dcrp_end_addr);
功能描述	配置DCRP区域
先决条件	ob_unlock
被调用函数	fmc_ready_wait
输入参数{in}	

dcrp_area	DCRP区域
<i>DCRP_AREA0</i>	双bank模式时，表示在bank0的DCRP区域；单bank模式时，表示第一个DCRP区域
<i>DCRP_AREA1</i>	双bank模式时，表示在bank1的DCRP区域；单bank模式时，表示第二个DCRP区域
输入参数{in}	
dcrp_eren	DCRP区域擦除选项
<i>OB_DCRP_AREA_ERASE_DISABLE</i>	当安全保护等级从低降为无保护时DCRP区域擦除失能
<i>OB_DCRP_AREA_ERASE_ENABLE</i>	当安全保护等级从低降为无保护时DCRP区域擦除使能
输入参数{in}	
dcrp_start	DCRP区域起始地址
输入参数{in}	
dcrp_end	DCRP区域结束地址
输出参数{out}	
-	-
返回值	
fmc_state_enum	FMC状态，参考 表3-463. 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(DCRP_AREA0, OB_DCRP_AREA_ERASE_ENABLE, 0x08002000, 0x08002400)
```

函数 ob_write_protection_config

函数ob_write_protection_config描述见下表

表 3-494. 函数 ob_write_protection_config

函数名称	ob_write_protection_config
函数原型	fmc_state_enum ob_write_protection_config(uint32_t wp_area, uint32_t wp_start, uint32_t wp_end);
功能描述	配置安全访问区域
先决条件	ob_unlock
被调用函数	fmc_ready_wait
输入参数{in}	
wp_area	写保护区域

<i>BK0WP_AREA0</i>	双bank模式时，表示在bank0的第一个写保护区域；单bank模式时，表示第一个写保护区域
<i>BK0WP_AREA1</i>	双bank模式时，表示在bank0的第二个写保护区域；单bank模式时，表示第二个写保护区域
<i>BK1WP_AREA0</i>	双bank模式时，表示在bank1的第二个写保护区域；单bank模式时，表示第三个写保护区域
<i>BK1WP_AREA1</i>	双bank模式时，表示在bank1的第二个写保护区域；单bank模式时，表示第三个写保护区域
输入参数{in}	
wp_start	写保护区域的第一页
输入参数{in}	
wp_end	写保护区域的最后一页
输出参数{out}	
-	-
返回值	
fmc_state_enum	FMC状态，参考 表3-463. 枚举类型fmc_state_enum

例如：

```
fmc_state_enum fmc_state;

fmc_unlock();

ob_unlock();

/* configure write protection area */

fmc_state = fmc_state_enum ob_write_protection_config(BK0WP_AREA0, 0x08, 0x09);
```

函数 ob_scr_area_config

函数ob_scr_area_config描述见下表

表 3-495. 函数 ob_scr_area_config

函数名称	ob_scr_area_config
函数原型	fmc_state_enum ob_scr_area_config(uint32_t scr_area, uint32_t secure_size);
功能描述	配置安全用户区域
先决条件	ob_unlock
被调用函数	fmc_ready_wait
输入参数{in}	
scr_area	安全用户区域
<i>SCR_AREA0</i>	Bank0的安全用户区域
<i>SCR_AREA1</i>	Bank1的安全用户区域
输入参数{in}	
secure_size	以页为单位的安全用户区域大小

输出参数{out}	
-	-
返回值	
fmc_state_enum	FMC状态, 参考 表3-463. 枚举类型fmc_state_enum

例如:

```
fmc_unlock();
```

```
ob_unlock();
```

```
/* configure the option byte secure area */
```

```
fmc_state_enum fmc_state = ob_scr_area_config (SCR_AREA0, 0x02);
```

函数 ob_boot_lock_config

函数ob_boot_lock_config描述见下表

表 3-496. 函数 ob_boot_lock_config

函数名称	ob_boot_lock_config
函数原型	fmc_state_enum ob_boot_lock_config(uint32_t boot_config);
功能描述	配置boot锁定
先决条件	ob_unlock
被调用函数	fmc_ready_wait
输入参数{in}	
boot_config	Boot配置位
OB_BOOT_LOCK_FR OM_MAIN_FLASH	强制从主闪存启动
OB_BOOT_UNLOCK	解除boot锁定
输出参数{out}	
-	-
返回值	
fmc_state_enum	FMC状态, 参考 表3-463. 枚举类型fmc_state_enum

例如:

```
fmc_unlock();
```

```
ob_unlock();
```

```
/* unlock boot */
```

```
fmc_state_enum fmc_state = ob_boot_lock_config(OB_BOOT_UNLOCK);
```

函数 ob_bank_memory_swap_config

函数ob_bank_memory_swap_config描述见下表

表 3-497. 函数 ob_bank_memory_swap_config

函数名称	ob_bank_memory_swap_config
函数原型	fmc_state_enum ob_bank_memory_swap_config(uint32_t swp_config);
功能描述	配置bank切换映射地址
先决条件	ob_unlock
被调用函数	fmc_ready_wait
输入参数{in}	
swp_config	bank切换映射地址
OB_BANK_MAPPING_SWAP	切换bank映射地址
OB_BANK_MAPPING_NOT_SWAP	不切换bank映射地址
输出参数{out}	
-	-
返回值	
fmc_state_enum	FMC状态，参考 表3-463. 枚举类型fmc_state_enum

例如：

```
fmc_state_enum fmc_state;

/* swap bank memory mapping */

fmc_state = ob_bank_memory_swap_config (OB_BANK_MAPPING_SWAP);
```

函数 ob_user_get

函数ob_user_get描述见下表

表 3-498. 函数 ob_user_get

函数名称	ob_user_get
函数原型	uint32_t ob_user_get(void);
功能描述	获取用户选项字节值
先决条件	-
被调用函数	-
输入参数{in}	
-	-
输出参数{out}	
-	-
返回值	
uint32_t	用户选项字节值

例如：

```
/* get value of option bytes USER */

uint32_t user_value;
```

```
user_value = ob_user_get();
```

函数 ob_security_protection_level_get

函数ob_security_protection_level_get描述见下表

表 3-499. 函数 ob_security_protection_level_get

函数名称	ob_security_protection_level_get
函数原型	uint8_t ob_security_protection_level_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	
uint8_t	安全保护等级
FMC_NSPC	无安全保护
FMC_LSPC	低安全保护
FMC_HSPC	高安全保护

例如：

```
/* get the value of FMC option bytes security protection level */
```

```
uint8_t user = ob_security_protection_level_get();
```

函数 ob_dcrp_area_get

函数ob_dcrp_area_get描述见下表

表 3-500. 函数 ob_dcrp_area_get

函数名称	ob_dcrp_area_get
函数原型	uint32_t ob_dcrp_area_get(uint32_t dcrp_area, uint32_t *dcrp_erase_option, uint32_t *dcrp_area_start_addr, uint32_t *dcrp_area_end_addr);
功能描述	获取DCRP相关配置
先决条件	-
被调用函数	-
输入参数{in}	
dcrp_area	DCRP区域
DCRP_AREA0	双bank模式时，表示在bank0的DCRP区域；单bank模式时，表示第一个DCRP区域
DCRP_AREA1	双bank模式时，表示在bank1的DCRP区域；单bank模式时，表示第二个DCRP区域

输出参数{out}	
dcrp_erase_option	DCRP擦除选项
输出参数{out}	
dcrp_start_addr	DCRP起始地址
输出参数{out}	
dcrp_end_addr	DCRP终止地址
返回值	
uint32_t	地址状态
INVLD_AREA_ADDRES	没有有效地址
VLD_AREA_ADDRESS	有效地址

例如:

```
/* get configuration of DCRP area */
```

```
uint32_t start, end, erase ,flag;
```

```
flag = ob_dcrp_area_get(DCRP_AREA0, &erase, &start, &end)
```

函数 ob_write_protection_get

函数ob_write_protection_get描述见下表

表 3-501. 函数 ob_write_protection_get

函数名称	ob_write_protection_get
函数原型	uint32_t ob_write_protection_get(uint32_t wp_area, uint32_t *wp_area_start_addr, uint32_t *wp_area_end_addr);
功能描述	获取写保护地址
先决条件	-
被调用函数	-
输入参数{in}	
wp_area	写保护区域
BK0WP_AREA0	双bank模式时，表示在bank0的第一个写保护区域；单bank模式时，表示第一个写保护区域
BK0WP_AREA1	双bank模式时，表示在bank0的第二个写保护区域；单bank模式时，表示第二个写保护区域
BK1WP_AREA0	双bank模式时，表示在bank1的第二个写保护区域；单bank模式时，表示第三个写保护区域
BK1WP_AREA1	双bank模式时，表示在bank1的第二个写保护区域；单bank模式时，表示第三个写保护区域
输出参数{out}	
wp_area_start_addr	写保护起始地址
输出参数{out}	
wp_area_end_addr	写保护终止地址

返回值	
uint32_t	地址状态
<i>INVLD_AREA_ADDRES</i> SS	没有有效地址
<i>VLD_AREA_ADDRESS</i>	有效地址

例如：

```
/* get configuration of WP area */
```

```
uint32_t start, end, flag;
```

```
flag = ob_write_protection_get(BK0WP_AREA0, &start, &end);
```

函数 **ob_scr_area_size_get**

函数ob_scr_area_size_get描述见下表

表 3-502. 函数 ob_scr_area_size_get

函数名称	ob_scr_area_size_get
函数原型	uint32_t ob_scr_area_size_get(uint32_t scr_area, uint32_t *scr_area_byte_cnt);
功能描述	获取安全用户区域大小
先决条件	-
被调用函数	-
输入参数{in}	
scr_area	安全用户区域
<i>SCR_AREA0</i>	Bank0的安全用户区域
<i>SCR_AREA1</i>	Bank1的安全用户区域
输出参数{out}	
secure_size	以字节为单位的安全用户区域大小
返回值	
uint32_t	地址状态
<i>INVLD_AREA_ADDRES</i> SS	没有有效地址
<i>VLD_AREA_ADDRESS</i>	有效地址

例如：

```
/* get size of secure area */
```

```
uint32_t size, flag;
```

```
flag = ob_scr_area_size_get(SCR_AREA0, &size);
```

函数 **ob_boot_config_get**

函数ob_boot_config_get描述见下表

表 3-503. 函数 ob_boot_config_get

函数名称	ob_boot_config_get
函数原型	uint32_t ob_boot_config_get(void);
功能描述	获取boot锁定配置
先决条件	-
被调用函数	-
输入参数{in}	
-	-
输出参数{out}	
-	-
返回值	
uint32_t	Boot配置

例如:

```
/* get boot value */
uint32_t boot_value;

boot_value = ob_boot_config_get();
```

函数 fmc_flag_get

函数fmc_flag_get描述见下表

表 3-504. 函数 fmc_flag_get

函数名称	fmc_flag_get
函数原型	FlagStatus fmc_flag_get(uint32_t flag);
功能描述	获取标志位状态
先决条件	-
被调用函数	-
输入参数{in}	
flag	FMC标志位
FMC_FLAG_BUSY	操作进行中标志
FMC_FLAG_OBERR	选项字节读错误标志
FMC_FLAG_WPERR	擦除/编程保护错误标志
FMC_FLAG_PGSERR	编程顺序错误标志
FMC_FLAG_PGMERR	编程大小错误标志
FMC_FLAG_PGAERR	编程对齐错误标志
FMC_FLAG_RPERR	读保护错误标志
FMC_FLAG_PGERR	编程错误标志
FMC_FLAG_OPRERR	操作失败错误标志
FMC_FLAG_ENDF	操作结束标志
输出参数{out}	
-	-

返回值	
FlagStatus	标志状态
<i>RESET</i>	复位
<i>SET</i>	置位

例如:

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

```
FlagStatus flag = fmc_flag_get(FMC_FLAG_ENDF);
```

函数 fmc_flag_clear

函数fmc_flag_clear描述见下表

表 3-505. 函数 fmc_flag_clear

函数名称	fmc_flag_clear
函数原型	void fmc_flag_clear(uint32_t flag);
功能描述	清除FMC标志
先决条件	-
被调用函数	-
输入参数{in}	
flag	FMC标志位
<i>FMC_FLAG_OBERR</i>	选项字节读错误标志
<i>FMC_FLAG_WPERR</i>	擦除/编程保护错误标志
<i>FMC_FLAG_PGSERR</i>	编程顺序错误标志
<i>FMC_FLAG_PGMERR</i>	编程大小错误标志
<i>FMC_FLAG_PGAERR</i>	编程对齐错误标志
<i>FMC_FLAG_RPERR</i>	读保护错误标志
<i>FMC_FLAG_PGERR</i>	编程错误标志
<i>FMC_FLAG_OPRERR</i>	操作失败错误标志
<i>FMC_FLAG_ENDF</i>	操作结束标志
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* clear FMC ENDF flag */
```

```
fmc_flag_clear(FMC_FLAG_ENDF);
```

函数 fmc_interrupt_enable

函数fmc_interrupt_enable描述见下表

表 3-506. 函数 fmc_interrupt_enable

函数名称	fmc_interrupt_enable
函数原型	void fmc_interrupt_enable(uint32_t interrupt);
功能描述	使能FMC中断
先决条件	-
被调用函数	-
输入参数{in}	
interrupt	FMC中断使能选项
FMC_INT_END	操作结束中断使能
FMC_INT_ERR	操作失败中断使能
FMC_INT_RPERR	读保护错误中断使能
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 3-507. 函数 fmc_interrupt_disable

函数名称	fmc_interrupt_disable
函数原型	void fmc_interrupt_disable(uint32_t interrupt);
功能描述	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
FMC_INT_END	操作结束中断失能
FMC_INT_ERR	操作失败中断失能
FMC_INT_RPERR	读保护错误中断失能
输出参数{out}	
-	-
返回值	
-	-

例如：

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

```
fmc_interrupt_disable(FMC_INT_END);
```

函数 fmc_ecc_flag_get

函数fmc_ecc_flag_get描述见下表

表 3-508. 函数 fmc_ecc_flag_get

函数名称	fmc_ecc_flag_get
函数原型	FlagStatus fmc_ecc_flag_get(uint32_t flag);
功能描述	获取ECC标志
先决条件	-
被调用函数	-
输入参数{in}	
flag	ECC标志
FMC_FLAG_ECCDET0	发现两个错误位（当单bank时，位于LSB的64位）
FMC_FLAG_ECCCOR0	纠正一个错误位（当单bank时，位于LSB的64位）
FMC_FLAG_ECCDET1	发现两个错误位，位于MSB的64位
FMC_FLAG_ECCCOR1	纠正一个错误位，位于MSB的64位
输出参数{out}	
-	-
返回值	
FlagStatus	标志状态
RESET	复位
SET	置位

例如：

```
/* get ECCDET0 flag status */
```

```
FlagStatus stat = fmc_ecc_flag_get(FMC_FLAG_ECCDET0);
```

函数 fmc_ecc_flag_clear

函数fmc_ecc_flag_clear描述见下表

表 3-509. 函数 fmc_ecc_flag_clear

函数名称	fmc_ecc_flag_clear
函数原型	void fmc_ecc_flag_clear (uint32_t flag);
功能描述	清除ECC标志
先决条件	-
被调用函数	-
输入参数{in}	
flag	ECC标志
FMC_FLAG_ECCDET0	发现两个错误位（当单bank时，位于LSB的64位）

<i>FMC_FLAG_ECCCOR</i> 0	纠正一个错误位（当单bank时，位于LSB的64位）
<i>FMC_FLAG_ECCDET1</i>	发现两个错误位，位于MSB的64位
<i>FMC_FLAG_ECCCOR</i> 1	纠正一个错误位，位于MSB的64位
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* clear ECCDET0 flag status */
```

```
fmc_ecc_flag_clear(FMC_FLAG_ECCDET0);
```

函数 **fmc_ecccor_interrupt_enable**

函数fmc_ecccor_interrupt_enable描述见下表

表 3-510. 函数 fmc_ecccor_interrupt_enable

函数名称	fmc_ecccor_interrupt_enable
函数原型	void fmc_ecccor_interrupt_enable(void);
功能描述	使能ECC纠错中断
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 ECCCOR interrupt */
```

```
fmc_ecccor_interrupt_enable();
```

函数 **fmc_ecccor_interrupt_disable**

函数fmc_ecccor_interrupt_disable描述见下表

表 3-511. 函数 fmc_ecccor_interrupt_disable

函数名称	fmc_ecccor_interrupt_disable
函数原型	void fmc_ecccor_interrupt_disable (void);
功能描述	失能ECC纠错中断

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 ECCCOR interrupt */
fmc_eccor_interrupt_disable();
```

函数 fmc_eccor_interrupt_flag_get

函数fmc_eccor_interrupt_flag_get描述见下表

表 3-512. 函数 fmc_eccor_interrupt_flag_get

函数名称	fmc_eccor_interrupt_flag_get
函数原型	FlagStatus fmc_eccor_interrupt_flag_get (uint32_t flag);
功能描述	获取ECC中断标志
先决条件	-
被调用函数	-
输入参数{in}	
flag	ECC标志
FMC_INT_FLAG_ECCCOR0	纠正一个错误位（当单bank时，位于LSB的64位）
FMC_INT_FLAG_ECCCOR1	纠正一个错误位，位于MSB的64位
输出参数{out}	
-	-
返回值	
FlagStatus	标志状态
RESET	复位
SET	置位

例如：

```
/* get ECCDET0 interrupt flag status */
FlagStatus stat = fmc_eccor_interrupt_flag_get(FMC_INT_FLAG_ECCCOR0);
```

函数 fmc_eccor_interrupt_flag_clear

函数fmc_eccor_interrupt_flag_clear描述见下表

表 3-513. 函数 `fmc_eccor_interrupt_flag_clear`

函数名称	<code>fmc_eccor_interrupt_flag_clear</code>
函数原型	<code>void fmc_eccor_interrupt_flag_clear(uint32_t flag);</code>
功能描述	清除ECC中断标志
先决条件	-
被调用函数	-
输入参数{in}	
flag	ECC标志
<code>FMC_INT_FLAG_ECCCOR0</code>	纠正一个错误位（当单bank时，位于LSB的64位）
<code>FMC_INT_FLAG_ECCCOR1</code>	纠正一个错误位，位于MSB的64位
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* clear ECCDET0 interrupt flag status */
```

```
fmc_eccor_interrupt_flag_clear(FMC_INT_FLAG_ECCCOR0);
```

函数 `fmc_interrupt_flag_get`

函数`fmc_interrupt_flag_get`描述见下表

表 3-514. 函数 `fmc_interrupt_flag_get`

函数名称	<code>fmc_interrupt_flag_get</code>
函数原型	<code>FlagStatus fmc_interrupt_flag_get(fmc_interrupt_flag_enum flag);</code>
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}	
interrupt	FMC中断标志
<code>FMC_INT_END</code>	操作结束中断标志
<code>FMC_INT_OPRERR</code>	操作失败中断标志
<code>FMC_INT_RPERR</code>	读保护错误中断标志
输出参数{out}	
-	-
返回值	
FlagStatus	标志状态
<code>RESET</code>	复位
<code>SET</code>	置位

例如：

```
/* get FMC RPERR error flag */
```

```
FlagStatus flag = fmc_interrupt_flag_get(FMC_INT_FLAG_RPERR);
```

函数 fmc_interrupt_flag_clear

函数fmc_interrupt_flag_clear描述见下表

表 3-515. 函数 fmc_interrupt_flag_clear

函数名称	fmc_interrupt_flag_clear
函数原型	void fmc_interrupt_flag_clear(uint32_t 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}	
interrupt	FMC中断标志
FMC_INT_END	操作结束中断标志
FMC_INT_OPRERR	操作失败中断标志
FMC_INT_RPERR	读保护错误中断标志
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* clear FMC RPERR error flag */
```

```
fmc_interrupt_flag_clear(FMC_INT_FLAG_RPERR);
```

函数 fmc_pd_mode_enter

函数fmc_pd_mode_enter描述见下表

表 3-516. 函数 fmc_pd_mode_enter

函数名称	fmc_pd_mode_enter
函数原型	__attribute__((section("RAMCODE"))) void fmc_pd_mode_enter(void);
功能描述	当系统运行时，flash进入掉电模式
先决条件	-
被调用函数	-
输入参数{in}	
-	-
输出参数{out}	
-	-
返回值	

-	-
---	---

例如:

```
/* flash enter power down mode when MCU run mode */
```

```
fmc_pd_mode_enter();
```

函数 fmc_pd_mode_exit

函数fmc_pd_mode_exit描述见下表

表 3-517. 函数 fmc_pd_mode_exit

函数名称	fmc_pd_mode_exit
函数原型	__attribute__((section("RAMCODE"))) void fmc_pd_mode_exit(void);
功能描述	当系统运行时，flash进入空闲模式
先决条件	-
被调用函数	-
输入参数{in}	
-	-
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* flash exit power down mode when MCU run mode */
```

```
fmc_pd_mode_exit();
```

函数 ob_bank_mode_config

函数ob_bank_mode_config描述见下表

表 3-518. 函数 ob_bank_mode_config

函数名称	ob_bank_mode_config
函数原型	__attribute__((section("RAMCODE"))) ErrStatus ob_bank_mode_config(uint32_t bank_mode);
功能描述	配置选项字节bank模式
先决条件	-
被调用函数	-
输入参数{in}	
bank_mode	bank模式
OB_SINGLE_BANK_MODE	单bank模式
OB_DUAL_BANK_MODE	双bank模式

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

例如:

```
/* configure the option byte to single bank mode */
```

ErrStatus flag;

```
flag = ob_bank_mode_config(OB_SINGLE_BANK_MODE);
```

3.17. FWDGT

独立看门狗定时器（FWDGT）是一个硬件计时电路，用来监测由软件故障导致的系统故障。适合于需要独立环境且对计时精度要求不高的场合。章节[3.17.1](#)描述了FWDGT的寄存器列表，章节[3.17.2](#)对FWDGT库函数进行说明。

3.17.1. 外设寄存器说明

FWDGT寄存器列表如下表所示:

表 3-519. 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

3.17.2. 外设库函数说明

FWDGT库函数列表如下表所示:

表 3-520. 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寄存器的值重载FWDGT计数器

库函数名称	库函数描述
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 3-521. 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 3-522. 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 3-523. 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 3-524. 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 3-525. 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 3-526. 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 3-527. 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 3-528. 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

<i>FWDGT_PSC_DIV3</i> 2	FWDGT预分频值设为32
<i>FWDGT_PSC_DIV6</i> 4	FWDGT预分频值设为64
<i>FWDGT_PSC_DIV1</i> 28	FWDGT预分频值设为128
<i>FWDGT_PSC_DIV2</i> 56	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 3-529. 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);
```

3.18. GPIO

GPIO用来实现各片上设备的逻辑输入/输出功能。章节[3.18.1](#)描述了GPIO的寄存器列表，章节[3.18.2](#)对GPIO库函数进行说明。

3.18.1. 外设寄存器说明

GPIO寄存器列表如下表所示：

表 3-530. GPIO 寄存器

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

3.18.2. 外设库函数说明

GPIO库函数列表如下表所示：

表 3-531. 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

库函数名称	库函数描述
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 3-532. 函数 gpio_deinit

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

例如：

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

```
gpio_deinit(GPIOA);
```

函数 gpio_mode_set

函数gpio_mode_set描述见下表：

表 3-533. 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

<i>GPIOx</i>	端口选择 (x = A,B,C,D,E,F,G)
输入参数{in}	
mode	GPIO引脚模式
<i>GPIO_MODE_INPUT</i>	输入模式
<i>GPIO_MODE_OUTPUT</i>	输出模式
<i>GPIO_MODE_AF</i>	备用功能模式
<i>GPIO_MODE_ANALOG</i>	模拟模式
输入参数{in}	
pull_up_down	GPIO引脚上拉下拉电阻设置
<i>GPIO_PUPD_NONE</i>	悬空模式，无上拉和下拉
<i>GPIO_PUPD_PULLUP</i>	带上拉电阻
<i>GPIO_PUPD_PULLDOWN</i>	带下拉电阻
输入参数{in}	
pin	GPIO pin
<i>GPIO_PIN_x</i>	引脚选择 (x = 0..15)
<i>GPIO_PIN_ALL</i>	所有引脚
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* config PA0 as 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 3-534. 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
<i>GPIOx</i>	端口选择 (x = A,B,C,D,E,F,G)
输入参数{in}	
otype	GPIO引脚输出模式
<i>GPIO_OTYPE_PP</i>	推挽输出模式
<i>GPIO_OTYPE_OD</i>	开漏输出模式
输入参数{in}	
speed	GPIO引脚输出最大速度
<i>GPIO_OSPEED_12MHZ</i>	最大输出速度为12MHz
<i>GPIO_OSPEED_60MHZ</i>	最大输出速度为60MHz
<i>GPIO_OSPEED_85MHZ</i>	最大输出速度为85MHz
<i>GPIO_OSPEED_100_220MHZ</i>	最大输出速度为100/220MHz
输入参数{in}	
pin	GPIO引脚
<i>GPIO_PIN_x</i>	引脚选择 (x = 0..15)
<i>GPIO_PIN_ALL</i>	所有引脚
输出参数{out}	
-	-
返回值	
-	-

例如:

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

```
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 3-535. 函数 gpio_bit_set

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

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

例如:

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

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

函数 gpio_bit_reset

函数gpio_bit_reset描述见下表:

表 3-536. 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
<i>GPIOx</i>	端口选择 (x = A,B,C,D,E,F,G)
输入参数{in}	
pin	GPIO引脚
<i>GPIO_PIN_x</i>	引脚选择 (x = 0..15)
<i>GPIO_PIN_ALL</i>	所有引脚
输出参数{out}	
-	-
返回值	
-	-

例如:

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

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

函数 gpio_bit_write

函数gpio_bit_write描述见下表:

表 3-537. 函数 `gpio_bit_write`

函数名称	<code>gpio_bit_write</code>
函数原型	<code>void gpio_bit_write(uint32_t gpio_periph, uint32_t pin, bit_status bit_value);</code>
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}	
<code>gpio_periph</code>	GPIO端口
<code>GPIOx</code>	端口选择 (x = A,B,C,D,E,F,G)
输入参数{in}	
<code>pin</code>	GPIO引脚
<code>GPIO_PIN_x</code>	引脚选择 (x = 0..15)
<code>GPIO_PIN_ALL</code>	所有引脚
输入参数{in}	
<code>bit_value</code>	设置或清除
<code>RESET</code>	清除引脚值
<code>SET</code>	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 3-538. 函数 `gpio_port_write`

函数名称	<code>gpio_port_write</code>
函数原型	<code>void gpio_port_write(uint32_t gpio_periph, uint16_t data);</code>
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}	
<code>gpio_periph</code>	GPIO端口
<code>GPIOx</code>	端口选择 (x = A,B,C,D,E,F,G)
输入参数{in}	
<code>data</code>	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 3-539. 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)
输入参数{in}	
speriod	过滤采样周期
GPIO_ISPERIOD(x)	周期值 (x = 0 ~ 255)
输入参数{in}	
iftype	过滤输入类型
GPIO_IFTYPE_SYNC	同步类型
GPIO_IFTYPE_3_SAMPLES	过滤 (3个采样点)
GPIO_IFTYPE_6_SAMPLES	过滤 (6个采样点)
GPIO_IFTYPE_ASYNC	异步类型
输入参数{in}	
pin	GPIO引脚
GPIO_PIN_x	引脚选择 (x = 0..15)
GPIO_PIN_ALL	所有引脚
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* set GPIO input filter */
```

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

函数 gpio_input_bit_get

函数gpio_input_bit_get描述见下表:

表 3-540. 函数 gpio_input_bit_get

函数名称	gpio_input_bit_get
函数原型	FlagStatus gpio_input_bit_get(uint32_t gpio_periph, uint32_t pin);
功能描述	获取引脚的输入值
先决条件	-
被调用函数	-
输入参数{in}	
gpio_periph	GPIO端口
GPIOx	端口选择 (x = A,B,C,D,E,F,G)
输入参数{in}	
pin	GPIO引脚
GPIO_PIN_x	引脚选择 (x = 0..15)
GPIO_PIN_ALL	所有引脚
输出参数{out}	
-	-
返回值	
FlagStatus	SET / RESET

例如:

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

```
FlagStatus bit_state;
```

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

函数 gpio_input_port_get

函数gpio_input_port_get描述见下表:

表 3-541. 函数 gpio_input_port_get

函数名称	gpio_input_port_get
函数原型	uint16_t gpio_input_port_get(uint32_t gpio_periph);
功能描述	获取端口的输入值
先决条件	-
被调用函数	-
输入参数{in}	

gpio_periph	GPIO端口
<i>GPIOx</i>	端口选择 (x = A,B,C,D,E,F,G)
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 3-542. 函数 gpio_output_bit_get

函数名称	gpio_output_bit_get
函数原型	FlagStatus gpio_output_bit_get(uint32_t gpio_periph, uint32_t pin);
功能描述	获取引脚的输出值
先决条件	-
被调用函数	-
输入参数{in}	
gpio_periph	GPIO端口
<i>GPIOx</i>	端口选择 (x = A,B,C,D,E,F,G)
输入参数{in}	
pin	GPIO引脚
<i>GPIO_PIN_x</i>	引脚选择 (x = 0..15)
<i>GPIO_PIN_ALL</i>	所有引脚
输出参数{out}	
-	-
返回值	
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 3-543. 函数 `gpio_output_port_get`

函数名称	<code>gpio_output_port_get</code>
函数原型	<code>uint16_t gpio_output_port_get(uint32_t gpio_periph);</code>
功能描述	获取端口的输出值
先决条件	-
被调用函数	-
输入参数{in}	
<code>gpio_periph</code>	GPIO端口
<code>GPIOx</code>	端口选择 (x = A,B,C,D,E,F,G)
输出参数{out}	
-	-
返回值	
<code>uint16_t</code>	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 3-544. 函数 `gpio_af_set`

函数名称	<code>gpio_af_set</code>
函数原型	<code>void gpio_af_set(uint32_t gpio_periph, uint32_t alt_func_num, uint32_t pin);</code>
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}	
<code>gpio_periph</code>	GPIO端口
<code>GPIOx</code>	端口选择 (x = A,B,C,D,E,F,G)
输入参数{in}	
<code>alt_func_num</code>	GPIO引脚备用功能, 请参见特定设备的数据手册
<code>GPIO_AF_0</code>	<i>RTC, HPDF</i>
<code>GPIO_AF_1</code>	<i>TIMER0, TIMER1, LPTIMER, TIMER4, TIMER14, TIMER15, TIMER16</i>
<code>GPIO_AF_2</code>	<i>TIMER0, TIMER1, TIMER2, TIMER3, TIMER4, TIMER7, TIMER14, TIMER19, I2C2</i>
<code>GPIO_AF_3</code>	<i>CLA0, CLA1, CLA2, CLA3, I2C0, I2C3, TIMER7, TIMER14, TIMER19, HPDF, CMP2, HRTIMER</i>
<code>GPIO_AF_4</code>	<i>TIMER0, TIMER7, TIMER15, TIMER16, I2C0, I2C1, I2C2, I2C3, EXMC, HPDF</i>

GPIO_AF_5	SPI0, SPI1, IFRP, TIMER7, HPDF, CLA3, UART3, UART4
GPIO_AF_6	UART3, SPI1, SPI2, TIMER0, TIMER4, TIMER7, TIMER19, HRTIMER, IFRP, HPDF
GPIO_AF_7	USART0, USART1, USART2, TIMER19, CMP4, CMP5, CMP6
GPIO_AF_8	CMP0, CMP1, CMP2, CMP3, CMP4, CMP5, CMP6, CMP7, UART3, UART4, I2C2, I2C3, HPDF,
GPIO_AF_9	TIMER0, TIMER7, TIMER14, EXMC, CAN0, CAN1, HRTIMER, QSPI
GPIO_AF_10	TIMER1, TIMER2, TIMER3, TIMER7, TIMER16, QSPI, HPDF
GPIO_AF_11	EXMC, TIMER0, TIMER7, CAN2, LPTIMER
GPIO_AF_12	SHTIMER, SPI1, TIMER0, TRIGSEL, CLA2, EXMC
GPIO_AF_13	TRIGSEL, HRTIMER
GPIO_AF_14	TIMER1, TIMER14, UART3, UART4, TRIGSEL, EXMC, TLI
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 3-545. 函数 gpio_pin_lock

函数名称	gpio_pin_lock
函数原型	void gpio_pin_lock(uint32_t gpio_periph, uint32_t pin);
功能描述	相应的引脚配置被锁定
先决条件	-
被调用函数	-
输入参数{in}	
gpio_periph	GPIO端口
GPIOx	端口选择 (x = A,B,C,D,E,F,G)
输入参数{in}	
pin	GPIO引脚
GPIO_PIN_x	引脚选择 (x = 0..15)
GPIO_PIN_ALL	所有引脚

输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* lock PA0 */
```

```
gpio_pin_lock(GPIOA, GPIO_PIN_0);
```

函数 gpio_bit_toggle

函数gpio_bit_toggle描述见下表：

表 3-546. 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）
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 3-547. 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)
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* toggle GPIOA */

gpio_port_toggle(GPIOA);
```

3.19. HPDF

GD32G5x3 内部集成了一种专门用于外部 Σ - Δ 调制器的高性能数字滤波器模块 (HPDF)。章节 [3.19.1](#) 描述了 HPDF 的寄存器列表, 章节 [3.19.2](#) 对 HPDF 库函数进行说明。

3.19.1. 外设寄存器说明

HPDF 寄存器列表如下表所示:

表 3-548. 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0
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

寄存器名称	寄存器描述
HPDF_FLTyTMFC	滤波器y阈值监视器标志清除寄存器
HPDF_FLTyEMMAX	滤波器y极值监视器最大值寄存器
HPDF_FLTyEMMIN	滤波器y极值监视器最小值寄存器
HPDF_FLTyCT	滤波器y转换定时器寄存器

3.19.2. 外设库函数说明

HPDF库函数列表如下表所示：

表 3-549. HPDF 库函数

库函数名称	库函数描述
hpdf_deinit	复位HPDF外设
hpdf_channel_struct_para_init	初始化HPDF通道结构体参数
hpdf_filter_struct_para_init	初始化HPDF滤波器结构体参数
hpdf_rc_struct_para_init	初始化规则转换结构体参数
hpdf_ic_struct_para_init	初始化注入转换结构体参数
hpdf_enable	使能HPDF模块
hpdf_disable	禁止HPDF模块
hpdf_channel_init	初始化HPDF通道
hpdf_filter_init	初始化HPDF滤波器
hpdf_rc_init	初始化规则转换
hpdf_ic_init	初始化注入转换
hpdf_clock_output_config	配置串行输出时钟
hpdf_clock_output_source_config	配置串行输出时钟源
hpdf_clock_output_duty_mode_disable	禁止串行输出时钟占空比模式
hpdf_clock_output_duty_mode_enable	使能串行输出时钟占空比模式
hpdf_clock_output_divider_config	配置串行输出时钟分频
hpdf_channel_enable	使能HPDF通道
hpdf_channel_disable	禁止HPDF通道
hpdf_spi_clock_source_config	配置SPI接口时钟源
hpdf_serial_interface_type_config	配置串行接口类型
hpdf_malfunction_monitor_disable	禁止故障监视器
hpdf_malfunction_monitor_enable	使能故障监视器
hpdf_clock_loss_disable	禁止时钟丢失检测
hpdf_clock_loss_enable	使能时钟丢失检测
hpdf_channel_pin_redirection_disable	禁止通道输入引脚重定向
hpdf_channel_pin_redirection_enable	使能通道输入引脚重定向
hpdf_channel_mux_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

库函数名称	库函数描述
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
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

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

成员名称	功能描述
data_packing_mode	并行数据寄存器的数据封装模式
channel_muxlexer	复用通道输入数据源
channel_pin_select	通道输入引脚选择
ck_loss_detector	时钟丢失检测
malfunction_monitor	故障监视器
spi_ck_source	SPI接口时钟源
serial_interface	串行接口类型
calibration_offset	24位校准偏移
right_bit_shift	右移位数
tm_filter	阈值监视器滤波器阶数选择
tm_filter_oversample	阈值监视器滤波器过采样率
mm_break_signal	分配故障监视器断路信号
mm_counter_threshold	故障监视器计数阈值
plsk_value	跳频脉冲数

结构体 hpdf_filter_parameter_struct

表 3-551. 结构体 hpdf_filter_parameter_struct

成员名称	功能描述
tm_fast_mode	阈值监视器快速模式
tm_channel	阈值监视器通道
tm_high_threshold	阈值监视器上限阈值
tm_low_threshold	阈值监视器下限阈值
extreme_monitor_channel	极值监视器通道
sinc_filter	Sinc滤波器阶数

成员名称	功能描述
sinc_oversample	Sinc滤波器过采样率
integrator_oversample	积分器过采样率
ht_break_signal	分配上限阈值断路信号
lt_break_signal	分配下限阈值断路信号

结构体 `hpdf_rc_parameter_struct`

表 3-552. 结构体 `hpdf_rc_parameter_struct`

成员名称	功能描述
fast_mode	规则转换快速转换模式
rsc_channel	规则转换通道
rcdmaen	使能读取规则转换数据的DMA通道
rccsyn	规则转换同步
continuous_mode	规则转换连续模式

结构体 `hpdf_ic_parameter_struct`

表 3-553. 结构体 `hpdf_ic_parameter_struct`

成员名称	功能描述
trigger_edge	注入转换触发边沿
trigger_signal	注入转换触发信号
icdmaen	使能读取注入转换数据的DMA通道
scmod	注入转换扫描模式
icsyn	注入转换同步
ic_channel_group	选择注入转换通道组

枚举类型 `hpdf_channel_enum`

表 3-554. 枚举类型 `hpdf_channel_enum`

成员名称	功能描述
CHANNEL0	HPDF通道0
CHANNEL1	HPDF通道1
CHANNEL2	HPDF通道2
CHANNEL3	HPDF通道3
CHANNEL4	HPDF通道4
CHANNEL5	HPDF通道5
CHANNEL6	HPDF通道6
CHANNEL7	HPDF通道7

枚举类型 `hpdf_filter_enum`

表3-555. 枚举类型 `hpdf_filter_enum`

成员名称	功能描述
FLT0	HPDF滤波器0
FLT1	HPDF滤波器1
FLT2	HPDF滤波器2
FLT3	HPDF滤波器3

枚举类型 `hpdf_flag_enum`

表3-556. 枚举类型 `hpdf_flag_enum`

成员名称	功能描述
HPDF_FLAG_FLTY_ICEF	注入转换结束标志
HPDF_FLAG_FLTY_RCEF	规则转换结束标志
HPDF_FLAG_FLTY_ICDOF	注入转换溢出标志
HPDF_FLAG_FLTY_RCDOF	规则转换溢出标志
HPDF_FLAG_FLTY_TMEOF	阈值监视器事件标志
HPDF_FLAG_FLTY_ICPF	注入转换正在进行标志
HPDF_FLAG_FLTY_RCPF	规则转换正在进行标志
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

成员名称	功能描述
HPDF_FLAG_FLYT_HTF0	通道0阈值监视器上限阈值标志
HPDF_FLAG_FLYT_HTF1	通道1阈值监视器上限阈值标志
HPDF_FLAG_FLYT_HTF2	通道2阈值监视器上限阈值标志
HPDF_FLAG_FLYT_HTF3	通道3阈值监视器上限阈值标志
HPDF_FLAG_FLYT_HTF4	通道4阈值监视器上限阈值标志
HPDF_FLAG_FLYT_HTF5	通道5阈值监视器上限阈值标志
HPDF_FLAG_FLYT_HTF6	通道6阈值监视器上限阈值标志
HPDF_FLAG_FLYT_HTF7	通道7阈值监视器上限阈值标志

枚举类型 `hpdf_interrupt_flag_enum`

表3-557. 枚举类型 `hpdf_interrupt_flag_enum`

成员名称	功能描述
HPDF_INT_FLAG_FLYT_ICEF	注入转换结束中断标志
HPDF_INT_FLAG_FLYT_RCEF	规则转换结束中断标志
HPDF_INT_FLAG_FLYT_ICDOF	注入转换溢出中断标志
HPDF_INT_FLAG_FLYT_RCDOF	规则转换溢出中断标志
HPDF_INT_FLAG_FLYT_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3-558. 枚举类型 `hpdf_interrupt_enum`

成员名称	功能描述
HPDF_INT_FLYT_ICEIE	使能注入转换结束中断
HPDF_INT_FLYT_RCEIE	使能规则转换结束中断
HPDF_INT_FLYT_ICDOIE	使能注入转换溢出中断

成员名称	功能描述
HPDF_INT_FLT_Y_RCDOIE	使能规则转换溢出中断
HPDF_INT_FLT_Y_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 3-559. 函数 hpdf_deinit

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

例如：

```
/* reset HPDF */
```

```
hpdf_deinit();
```

函数 hpdf_channel_struct_para_init

函数hpdf_channel_struct_para_init描述见下表：

表 3-560. 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3-554. 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 3-561. 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3-551. 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 3-562. 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3-552. 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 3-563. 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3-553. 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 3-564. 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 3-565. 函数 hpdf_disable

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

例如：

```
/* disable the HPDF module */
```

```
hpdf_disable();
```

函数 hpdf_channel_init

函数hpdf_channel_init描述见下表：

表 3-566. 函数 hpdf_channel_init

函数名称	hpdf_channel_init
函数原型	void hpdf_channel_init(hpdf_channel_enum channelx, hpdf_channel_parameter_struct* init_struct);
功能描述	初始化HPDF通道
先决条件	-
被调用函数	-

输入参数{in}	
channelx	HPDF模块通道
CHANNELx(x=0...7)	HPDF通道选择, 参考 表3-554. 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3-550. 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 3-567. 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
<i>FLTy(y=0..3)</i>	HPDF滤波器选择, 参考 表3-555. 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3-551. 结构体hpdf_filter_parameter_struct
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* initialize the HPDF filter0 */
```

```
hpdf_filter_parameter_struct hpdf_filter_init_struct;
```

```
hpdf_filter_init_struct.tm_fast_mode = TMFM_DISABLE;
```

```
hpdf_filter_init_struct.tm_channel = TMCHEN_CHANNEL0;
```

```
hpdf_filter_init_struct.tm_high_threshold = tm_high_val;
```

```
hpdf_filter_init_struct.tm_low_threshold = tm_low_val;
```

```
hpdf_filter_init_struct.extreme_monitor_channel = EM_CHANNEL0;
```

```
hpdf_filter_init_struct.sinc_filter = FLT_SINC3;
```

```
hpdf_filter_init_struct.sinc_oversample = FLT_OVER_SAMPLE_32;
```

```
hpdf_filter_init_struct.integrator_oversample = INTEGRATOR_BYPASS;
```

```
hpdf_filter_init(FLT0, &hpdf_filter_init_struct);
```

函数 hpdf_rc_init

函数hpdf_rc_init描述见下表:

表 3-568. 函数 hpdf_rc_init

函数名称	hpdf_rc_init
函数原型	void hpdf_rc_init(hpdf_filter_enum filtery, hpdf_rc_parameter_struct* init_struct);
功能描述	初始化规则转换
先决条件	-
被调用函数	-
输入参数{in}	
filtery	HPDF模块滤波器
<i>FLTy(y=0...3)</i>	HPDF滤波器选择, 参考 表3-555. 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3-552. 结构体

	<u>hpdf_rc_parameter_struct</u>
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 3-569. 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 <u>枚举类型hpdf_filter_enum</u>
输入参数{in}	
*init_struct	初始化注入转换参数，结构体成员参考 <u>表3-553. 结构体 hpdf_ic_parameter_struct</u>
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* initialize inserted conversion of the HPDF fliter0 */

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 3-570. 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_AUDIO_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 3-571. 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_AUDIO_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 3-572. 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 3-573. 函数 `hpdf_clock_output_duty_mode_enable`

函数名称	<code>hpdf_clock_output_duty_mode_enable</code>
函数原型	<code>void hpdf_clock_output_duty_mode_enable(void);</code>
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 3-574. 函数 `hpdf_clock_output_divider_config`

函数名称	<code>hpdf_clock_output_divider_config</code>
函数原型	<code>void hpdf_clock_output_divider_config(uint8_t divider);</code>
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 3-575. 函数 `hpdf_channel_enable`

函数名称	<code>hpdf_channel_enable</code>
函数原型	<code>void hpdf_channel_enable(hpdf_channel_enum channelx);</code>
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
<code>CHANNELx(x=0...7)</code>	HPDF通道选择, 参考 表3-554. 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 3-576. 函数 `hpdf_channel_disable`

函数名称	<code>hpdf_channel_disable</code>
函数原型	<code>void hpdf_channel_disable(hpdf_channel_enum channelx);</code>
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
<code>CHANNELx(x=0...7)</code>	HPDF通道选择, 参考 表3-554. 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 3-577. 函数 `hpdf_spi_clock_source_config`

函数名称	<code>hpdf_spi_clock_source_config</code>
函数原型	<code>void hpdf_spi_clock_source_config(hpdf_channel_enum channelx, uint32_t clock_source);</code>
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
<code>CHANNELx(x=0..7)</code>	HPDF通道选择, 参考 表3-554. 枚举类型hpdf_channel_enum
输入参数{in}	
clock_source	SPI接口时钟源
<code>EXTERNAL_CKIN</code>	外部输入时钟
<code>INTERNAL_CKOUT</code>	内部CKOUT时钟
<code>HALF_CKOUT_FALLIN G_EDGE</code>	内部每第二个CKOUT时钟的下降沿
<code>HALF_CKOUT_RISING _EDGE</code>	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 3-578. 函数 `hpdf_serial_interface_type_config`

函数名称	<code>hpdf_serial_interface_type_config</code>
函数原型	<code>void hpdf_serial_interface_type_config(hpdf_channel_enum channelx, uint32_t type);</code>
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
<code>CHANNELx(x=0...7)</code>	HPDF通道选择, 参考 表3-554. 枚举类型hpdf_channel_enum
输入参数{in}	
type	串行接口类型

<code>SPI_RISING_EDGE</code>	SPI接口上升沿采样
<code>SPI_FALLING_EDGE</code>	SPI接口下降沿采样
<code>MANCHESTER_CODE</code> 0	曼切斯特编码输入：上升沿=逻辑0，下降沿=逻辑1
<code>MANCHESTER_CODE</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 3-579. 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
<code>CHANNELx(x=0...7)</code>	HPDF通道选择，参考 表3-554. 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 3-580. 函数 `hpdf_malfunction_monitor_enable`

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

函数原型	void hpdf_malfunction_monitor_enable(hpdf_channel_enum channelx);
功能描述	使能故障检测器
先决条件	-
被调用函数	-
输入参数{in}	
channelx	HPDF模块通道
CHANNELx(x=0...7)	HPDF通道选择, 参考 表3-554. 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 3-581. 函数 hpdf_clock_loss_disable

函数名称	hpdf_clock_loss_disable
函数原型	void hpdf_clock_loss_disable(hpdf_channel_enum channelx);
功能描述	禁止时钟检测
先决条件	CHANNELx(x=0...7)禁止
被调用函数	-
输入参数{in}	
channelx	HPDF模块通道
CHANNELx(x=0...7)	HPDF通道选择, 参考 表3-554. 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 3-582. 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3-554. 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 3-583. 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3-554. 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 3-584. 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3-554. 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 3-585. 函数 `hpdf_channel_multiplexer_config`

函数名称	<code>hpdf_channel_multiplexer_config</code>
函数原型	<code>void hpdf_channel_multiplexer_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3-554. 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 3-586. 函数 hpdf_data_pack_mode_config

函数名称	hpdf_data_pack_mode_config
函数原型	void hpdf_data_pack_mode_config(hpdf_channel_enum channelx, uint32_t mode);
功能描述	配置数据封装模式
先决条件	CHANNELx(x=0...7)禁止
被调用函数	-
输入参数{in}	
channelx	HPDF模块通道
CHANNELx(x=0...7)	HPDF通道选择，参考 表3-554. 枚举类型hpdf_channel_enum
输入参数{in}	
mode	数据封装模式
DPM_STANDARD_MODE	标准模式
DPM_INTERLEAVED_MODE	交错模式
DPM_DUAL_MODE	双通道模式
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* configure data packing mode */
```

```
hpdf_data_pack_mode_config(CHANNEL0, DPM_STANDARD_MODE);
```

函数 hpdf_data_right_bit_shift_config

函数hpdf_data_right_bit_shift_config描述见下表：

表 3-587. 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3-554. 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 3-588. 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3-554. 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 3-589. 函数 `hpdf_malfunction_break_signal_config`

函数名称	<code>hpdf_malfunction_break_signal_config</code>
函数原型	<code>void hpdf_malfunction_break_signal_config(hpdf_channel_enum channelx, uint32_t break_signal);</code>
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
<code>CHANNELx(x=0...7)</code>	HPDF通道选择, 参考 表3-554. 枚举类型hpdf_channel_enum
输入参数{in}	
break_signal	数据封装模式
<code>NO_MM_BREAK</code>	无断路信号被分配
<code>MM_BREAK0</code>	断路信号0被分配至所监视的通道
<code>MM_BREAK1</code>	断路信号1被分配至所监视的通道
<code>MM_BREAK2</code>	断路信号2被分配至所监视的通道
<code>MM_BREAK3</code>	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 3-590. 函数 `hpdf_malfunction_counter_config`

函数名称	<code>hpdf_malfunction_counter_config</code>
函数原型	<code>void hpdf_malfunction_counter_config(hpdf_channel_enum channelx, uint8_t threshold);</code>
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
<code>CHANNELx(x=0...7)</code>	HPDF通道选择, 参考 表3-554. 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 3-591. 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3-554. 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 3-592. 函数 hpdf_write_parallel_data_interleaved_mode

函数名称	hpdf_write_parallel_data_interleaved_mode
函数原型	void hpdf_write_parallel_data_interleaved_mode(hpdf_channel_enum channelx, int32_t data);
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
CHANNELx(x=0...7)	HPDF通道选择, 参考 表3-554. 枚举类型hpdf_channel_enum
输入参数{in}	
data	并行数据
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* write the parallel data on interleaved mode of data packing */
```

```
hpdf_write_parallel_data_interleaved_mode(CHANNEL0, 0xEFFFFFFF);
```

函数 hpdf_write_parallel_data_dual_mode

函数hpdf_write_parallel_data_dual_mode描述见下表:

表 3-593. 函数 hpdf_write_parallel_data_dual_mode

函数名称	hpdf_write_parallel_data_dual_mode
函数原型	void hpdf_write_parallel_data_dual_mode(hpdf_channel_enum channelx, int32_t data);
功能描述	写入数据封装双通道模式下的并行数据
先决条件	-
被调用函数	-
输入参数{in}	
channelx	HPDF模块通道
CHANNELx(x=0...7)	HPDF通道选择, 参考 表3-554. 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 3-594. 函数 `hpdf_pulse_skip_update`

函数名称	<code>hpdf_pulse_skip_update</code>
函数原型	<code>void hpdf_pulse_skip_update(hpdf_channel_enum channelx, uint8_t number);</code>
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
<code>CHANNELx(x=0..7)</code>	HPDF通道选择, 参考 表3-554. 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 */
hpdf_pulse_skip_update(CHANNEL0, 63);
```

函数 `hpdf_pulse_skip_read`

函数 `hpdf_pulse_skip_read` 描述见下表:

表 3-595. 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
<code>CHANNELx(x=0..7)</code>	HPDF通道选择, 参考 表3-554. 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 3-596. 函数 hpdf_filter_enable

函数名称	hpdf_filter_enable
函数原型	void hpdf_filter_enable(hpdf_filter_enum filtery);
功能描述	使能滤波器
先决条件	-
被调用函数	-
输入参数{in}	
filtery	HPDF模块滤波器
FLTy(y=0...3)	HPDF滤波器选择，参考 表3-555. 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 3-597. 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3-555. 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 3-598. 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3-555. 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
FLT_SINC4	Sinc4型滤波器
FLT_SINC5	Sinc5型滤波器
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* configure sinc filter order and oversample */
```

```
hpdf_filter_config(FLT0, FLT_SINC2, 64);
```

函数 hpdf_integrator_oversample

函数hpdf_integrator_oversample描述见下表:

表 3-599. 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> (<i>y</i> =0...3)	HPDF滤波器选择, 参考 表3-555. 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 3-600. 函数 hpdf_threshold_monitor_filter_config

函数名称	hpdf_threshold_monitor_filter_config
函数原型	void hpdf_threshold_monitor_filter_config(hpdf_channel_enum channelx, uint32_t order, uint8_t oversample);
功能描述	配置阈值监视器的滤波器
先决条件	
被调用函数	-
输入参数{in}	
channelx	HPDF模块通道
<i>CHANNEL_x</i> (<i>x</i> =0...7)	HPDF通道选择, 参考 表3-554. 枚举类型hpdf_channel_enum
输入参数{in}	
order	滤波器阶数
<i>TM_FASTSINC</i>	FastSinc型滤波器
<i>TM_SINC1</i>	Sinc1型滤波器
<i>TM_SINC2</i>	Sinc2型滤波器
<i>TM_SINC3</i>	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 3-601. 函数 hpdf_threshold_monitor_filter_read_data

函数名称	hpdf_threshold_monitor_filter_read_data
函数原型	int16_t hpdf_threshold_monitor_filter_read_data(hpdf_channel_enum channelx);
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
CHANNELx(x=0...7)	HPDF通道选择，参考 表3-554. 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 3-602. 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
FLTy(y=0...3)	HPDF滤波器选择，参考 表3-555. 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 3-603. 函数 hpdf_threshold_monitor_fast_mode_enable

函数名称	hpdf_threshold_monitor_fast_mode_enable
函数原型	void hpdf_threshold_monitor_fast_mode_enable(hpdf_filter_enum filtery);
功能描述	使能阈值监视器快速模式
先决条件	-
被调用函数	-
输入参数{in}	
filtery	HPDF模块滤波器
FLTy(y=0...3)	HPDF滤波器选择，参考 表3-555. 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 3-604. 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

<i>FLTy</i> (<i>y</i> =0...3)	HPDF滤波器选择, 参考 表3-555. 枚举类型hpdf_filter_enum
输入参数{in}	
channel	阈值监视器所监视的通道
<i>TMCHEN_DISABLE</i>	禁止所有阈值监视器监视通道
<i>TMCHEN_CHANNEL0</i>	阈值监视器监视通道0
<i>TMCHEN_CHANNEL1</i>	阈值监视器监视通道1
<i>TMCHEN_CHANNEL2</i>	阈值监视器监视通道2
<i>TMCHEN_CHANNEL3</i>	阈值监视器监视通道3
<i>TMCHEN_CHANNEL4</i>	阈值监视器监视通道4
<i>TMCHEN_CHANNEL5</i>	阈值监视器监视通道5
<i>TMCHEN_CHANNEL6</i>	阈值监视器监视通道6
<i>TMCHEN_CHANNEL7</i>	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 3-605. 函数 hpdf_threshold_monitor_high_threshold

函数名称	hpdf_threshold_monitor_high_threshold
函数原型	void hpdf_threshold_monitor_high_threshold(hpdf_filter_enum filtery, int32_t value);
功能描述	配置阈值监视器上限阈值
先决条件	-
被调用函数	-
输入参数{in}	
filtery	HPDF模块滤波器
<i>FLTy</i> (<i>y</i> =0...3)	HPDF滤波器选择, 参考 表3-555. 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 3-606. 函数 hpdf_threshold_monitor_low_threshold

函数名称	hpdf_threshold_monitor_low_threshold
函数原型	void hpdf_threshold_monitor_low_threshold(hpdf_filter_enum filtery, int32_t value);
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
FLTy(y=0...3)	HPDF滤波器选择, 参考 表3-555. 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 3-607. 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3-555. 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 3-608. 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3-555. 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 3-609. 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3-555. 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
EM_CHANNEL1	极值监视y接收通道1的数据
EM_CHANNEL2	极值监视y接收通道2的数据
EM_CHANNEL3	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);
```

函数 hpdf_extremes_monitor_maximum_get

函数hpdf_extremes_monitor_maximum_get描述见下表：

表 3-610. 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(y=0...3)</i>	HPDF滤波器选择, 参考 表3-555. 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 3-611. 函数 hpdf_extremes_monitor_minimum_get

函数名称	hpdf_extremes_monitor_minimum_get
函数原型	int32_t hpdf_extremes_monitor_minimum_get(hpdf_filter_enum filtery);
功能描述	获取极值监视器最小极值
先决条件	-
被调用函数	-
输入参数{in}	
filtery	HPDF模块滤波器
<i>FLTy(y=0...3)</i>	HPDF滤波器选择, 参考 表3-555. 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 3-612. 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3-555. 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 3-613. 函数 hpdf_rc_continuous_disable

函数名称	hpdf_rc_continuous_disable
函数原型	void hpdf_rc_continuous_disable(hpdf_filter_enum filtery);
功能描述	禁止规则转换连续模式
先决条件	-
被调用函数	-
输入参数{in}	
filtery	HPDF模块滤波器
FLTy(y=0...3)	HPDF滤波器选择, 参考 表3-555. 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 3-614. 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3-555. 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 3-615. 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3-555. 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 3-616. 函数 `hpdf_rc_syn_disable`

函数名称	<code>hpdf_rc_syn_disable</code>
函数原型	<code>void hpdf_rc_syn_disable(hpdf_filter_enum filtery);</code>
功能描述	禁止规则转换同步
先决条件	禁止FLTy(y=0...3)
被调用函数	-
输入参数{in}	
filtery	HPDF模块滤波器
<i>FLTy(y=0...3)</i>	HPDF滤波器选择, 参考 表3-555. 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 3-617. 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3-555. 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 3-618. 函数 `hpdf_rc_dma_disable`

函数名称	hpdf_rc_dma_disable
函数原型	void hpdf_rc_dma_disable(hpdf_filter_enum filtery);
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
FLTy(y=0...3)	HPDF滤波器选择, 参考 表3-555. 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 3-619. 函数 `hpdf_rc_dma_enable`

函数名称	hpdf_rc_dma_enable
函数原型	void hpdf_rc_dma_enable(hpdf_filter_enum filtery);
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
FLTy(y=0...3)	HPDF滤波器选择, 参考 表3-555. 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 3-620. 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3-555. 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3-554. 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 3-621. 函数 `hpdf_rc_fast_mode_disable`

函数名称	<code>hpdf_rc_fast_mode_disable</code>
函数原型	<code>void hpdf_rc_fast_mode_disable(hpdf_filter_enum filtery);</code>
功能描述	禁止规则转换快速模式
先决条件	禁止 <code>FLTy(y=0...3)</code>
被调用函数	-
输入参数{in}	
filtery	HPDF模块滤波器
<i>FLTy(y=0...3)</i>	HPDF滤波器选择, 参考 表3-555. 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 3-622. 函数 hpdf_rc_fast_mode_enable

函数名称	hpdf_rc_fast_mode_enable
函数原型	void hpdf_rc_fast_mode_enable(hpdf_filter_enum filtery);
功能描述	使能规则转换快速模式
先决条件	禁止FLTy(y=0...3)
被调用函数	-
输入参数{in}	
filtery	HPDF模块滤波器
FLTy(y=0...3)	HPDF滤波器选择，参考 表3-555. 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 3-623. 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3-555. 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 3-624. 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3-555. 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 3-625. 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3-555. 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 3-626. 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3-555. 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 3-627. 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3-555. 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 3-628. 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3-555. 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 3-629. 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3-555. 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 3-630. 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3-555. 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 3-631. 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3-555. 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 3-632. 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3-555. 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 3-633. 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3-555. 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_TRG00

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_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 3-634. 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3-555. 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 3-635. 函数 hpdf_ic_data_get

函数名称	hpdf_ic_data_get
函数原型	int32_t hpdf_ic_data_get(hpdf_filter_enum filtery);
功能描述	获取注入转换数据
先决条件	-
被调用函数	-
输入参数{in}	
filtery	HPDF模块滤波器
FLTy(y=0...3)	HPDF滤波器选择，参考 表3-555. 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 3-636. 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3-555. 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 3-637. 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3-555. 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

HPDF_FLAG_FLTy_TM EOF	阈值监视器事件标志
HPDF_FLAG_FLTy_IC PF	注入转换正在进行标志
HPDF_FLAG_FLTy_RC PF	规则转换正在进行标志
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_RC HPDT	规则通道等待处理数据
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	
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 3-638. 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3-555. 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	规则转换结束标志

<i>EF</i>	
<i>HPDF_FLAG_FLTy_ICDOF</i>	注入转换溢出标志
<i>HPDF_FLAG_FLTy_RCDOF</i>	规则转换溢出标志
<i>HPDF_FLAG_FLTy_TMEOF</i>	阈值监视器事件标志
<i>HPDF_FLAG_FLT0_CKLFX</i>	通道(x = 0..7)时钟丢失标志
<i>HPDF_FLAG_FLT0_MMFx</i>	通道(x = 0..7)故障事件标志
<i>HPDF_FLAG_FLTy_LTFx</i>	通道(x = 0..7)阈值监视器下限阈值标志
<i>HPDF_FLAG_FLTy_HTFx</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	
-	-

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 3-639. 函数 `hpdf_interrupt_enable`

函数名称	<code>hpdf_interrupt_enable</code>
函数原型	<code>void hpdf_interrupt_enable(hpdf_filter_enum filtery, hpdf_interrput_enum interrupt);</code>
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
<i>FLTy(y=0...3)</i>	HPDF滤波器选择, 参考 表3-555. 枚举类型hpdf_filter_enum
输入参数{in}	
interrupt	HPDF模块中断
<i>HPDF_INT_FLTy_ICEI</i> <i>E</i>	使能注入转换结束中断
<i>HPDF_INT_FLTy_RCEI</i>	使能规则转换结束中断

E	
HPDF_INT_FLTy_ICD OIE	使能注入转换溢出中断
HPDF_INT_FLTy_RCD OIE	使能规则转换溢出中断
HPDF_INT_FLTy_TMIE	使能阈值监视器事件中断
HPDF_INT_FLT0_MMI E	使能故障监视中断
HPDF_INT_FLT0_CKLI E	使能时钟丢失中断
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* enable FLT0 threshold monitor interrupt */
```

```
hpdf_interrupt_enable(FTL0, HPDF_INT_FLTy_TMEIOIE);
```

函数 hpdf_interrupt_disable

函数hpdf_interrupt_disable描述见下表：

表 3-640. 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3-555. 枚举类型hpdf_filter_enum
输入参数{in}	
interrupt	HPDF模块中断
HPDF_INT_FLTy_ICEI E	使能注入转换结束中断
HPDF_INT_FLTy_RCEI E	使能规则转换结束中断
HPDF_INT_FLTy_ICD OIE	使能注入转换溢出中断
HPDF_INT_FLTy_RCD OIE	使能规则转换溢出中断

HPDF_INT_FLTy_TMIE	使能阈值监视器事件中断
HPDF_INT_FLT0_MMIE	使能故障监视中断
HPDF_INT_FLT0_CKLI	使能时钟丢失中断
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* disable FLT0 threshold monitor interrupt */
```

```
hpdf_interrupt_disable(FLT0, HPDF_INT_FLT0_CKLI);
```

函数 hpdf_interrupt_flag_get

函数hpdf_interrupt_flag_get描述见下表：

表 3-641. 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3-555. 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_FLT	通道(x = 0..7)故障事件中断标志

0_MMFX	
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 3-642. 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3-555. 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);
```

3.20. I2C

I2C（内部集成电路总线）模块提供了符合工业标准的两线串行制接口，可用于MCU和外部I2C设备的通讯。章节[3.20.1](#)描述了I2C的寄存器列表，章节[3.20.2](#)对I2C库函数进行说明。

3.20.1. 外设寄存器说明

I2C寄存器列表如下表所示：

表 3-643. I2C 寄存器

寄存器名称	寄存器描述
I2C_CTL0	控制寄存器 0
I2C_CTL1	控制寄存器 1
I2C_SADDR0	从机地址寄存器 0
I2C_SADDR1	从机地址寄存器 1
I2C_TIMING	时序寄存器
I2C_TIMEOUT	超时寄存器
I2C_STAT	状态寄存器
I2C_STATC	状态清除寄存器
I2C_PEC	PEC 寄存器
I2C_RDATA	接收数据寄存器
I2C_TDATA	发送数据寄存器
I2C_CTL2	控制寄存器 2

3.20.2. 外设库函数说明

I2C库函数列表如下表所示：

表 3-644. 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

库函数名称	库函数描述
i2c_master_addressing	配置 I2C 从机地址以及数据传输方向
i2c_address10_header_enable	主机接收模式下，10 位地址头只执行读操作
i2c_address10_header_disable	主机接收模式下，10 位地址头执行完整的读操作序列
i2c_address10_enable	使能主机模式下 10 位地址寻址模式
i2c_address10_disable	禁能主机模式下 10 位地址寻址模式
i2c_automatic_end_enable	使能主机模式下 I2C 自动结束模式
i2c_automatic_end_disable	禁能主机模式下 I2C 自动结束模式
i2c_slave_response_to_gcall_enable	使能从机响应广播呼叫
i2c_slave_response_to_gcall_disable	禁能从机响应广播呼叫
i2c_stretch_scl_low_enable	当从机数据没有准备好时拉低 SCL
i2c_stretch_scl_low_disable	当从机数据没有准备好时不拉低 SCL
i2c_address_config	配置 I2C 从机地址
i2c_address_bit_compare_config	定义 ADDRESS[7:1]的哪些位和接收到的地址进行比较
i2c_address_disable	禁能从机模式下 I2C 地址
i2c_second_address_config	配置 I2C 从机第二个地址
i2c_second_address_disable	禁能 I2C 从机第二个地址
i2c_receved_address_get	获取从机模式下匹配成功的地址
i2c_slave_byte_control_enable	使能从机字节控制
i2c_slave_byte_control_disable	禁能从机字节控制
i2c_nack_enable	从机模式下产生 NACK
i2c_nack_disable	从机模式下产生 ACK
i2c_wakeup_from_deepsleep_enable	使能从 Deep-sleep 模式中唤醒
i2c_wakeup_from_deepsleep_disable	禁止从 Deep-sleep 模式中唤醒
i2c_enable	使能 I2C
i2c_disable	禁能 I2C
i2c_start_on_bus	在 I2C 总线上生成起始位
i2c_stop_on_bus	在 I2C 总线上生成停止位
i2c_data_transmit	发送数据
i2c_data_receive	接收数据
i2c_reload_enable	使能 I2C 重载模式
i2c_reload_disable	禁能 I2C 重载模式
i2c_transfer_byte_number_config	配置待发送字节数
i2c_dma_enable	使能发送/接收模式下 DMA
i2c_dma_disable	禁能发送/接收模式下 DMA
i2c_pec_transfer	I2C 传输 PEC 值
i2c_pec_enable	使能报文错误校验
i2c_pec_disable	禁能报文错误校验
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

库函数名称	库函数描述
i2c_smbus_default_addr_disable	禁能 SMBus 设备默认地址
i2c_smbus_host_addr_enable	使能 SMBus 主机地址
i2c_smbus_host_addr_disable	禁能 SMBus 主机地址
i2c_extented_clock_timeout_enable	使能时钟信号延展超时检测
i2c_extented_clock_timeout_disable	禁能时钟信号延展超时检测
i2c_clock_timeout_enable	使能时钟超时检测
i2c_clock_timeout_disable	禁能时钟超时检测
i2c_bus_timeout_b_config	配置总线超时 B
i2c_bus_timeout_a_config	配置总线超时 A
i2c_idle_clock_timeout_config	配置空闲时钟超时检测
i2c_flag_get	获取 I2C 标志位
i2c_flag_clear	清除 I2C 标志位
i2c_interrupt_enable	中断使能
i2c_interrupt_disable	中断除能
i2c_interrupt_flag_get	获取中断标志位
i2c_interrupt_flag_clear	清除中断标志位

枚举类型 i2c_interrupt_flag_enum

表 3-645. 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 3-646. 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 3-647. 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 3-648. 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 3-649. 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 3-650. 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 3-651. 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 3-652. 函数 i2c_master_addressing

函数名称	i2c_master_addressing
函数原型	void i2c_master_addressing(uint32_t i2c_periph, uint32_t address, uint32_t trans_direction);
功能描述	配置 I2C 从机地址以及数据传输方向
先决条件	-
被调用函数	-
输入参数{in}	
i2c_periph	I2C 外设
I2Cx	(x=0,1,2,3)
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 3-653. 函数 i2c_address10_header_enable

函数名称	i2c_address10_header_enable
函数原型	void i2c_address10_header_enable(uint32_t i2c_periph);
功能描述	主机接收模式下，10 位地址头只执行读操作
先决条件	-
被调用函数	-
输入参数{in}	
i2c_periph	I2C 外设
<i>I2Cx</i>	(x=0,1,2,3)
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* 10-bit address header executes read direction only in master receive mode */
```

```
i2c_address10_header_enable(I2C0);
```

函数 i2c_address10_header_disable

函数i2c_address10_header_disable描述见下表：

表 3-654. 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 3-655. 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 3-656. 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 3-657. 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 3-658. 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 3-659. 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 3-660. 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 3-661. 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 3-662. 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 3-663. 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 3-664. 函数 i2c_address_bit_compare_config

函数名称	i2c_address_bit_compare_config
函数原型	void i2c_address_bit_compare_config(uint32_t i2c_periph, uint32_t compare_bits);
功能描述	定义 ADDRESS[7:1]的哪些位和接收到的地址进行比较
先决条件	-
被调用函数	-
输入参数{in}	
i2c_periph	I2C peripheral
I2Cx	(x=0,1,2,3)
输入参数{in}	
compare_bits	需要进行比较的位
ADDRESS_BIT1_COMPARE	地址的第 1 位需要进行比较
ADDRESS_BIT2_COMPARE	地址的第 2 位需要进行比较
ADDRESS_BIT3_COMPARE	地址的第 3 位需要进行比较
ADDRESS_BIT4_COMPARE	地址的第 4 位需要进行比较
ADDRESS_BIT5_COMPARE	地址的第 5 位需要进行比较
ADDRESS_BIT6_COMPARE	地址的第 6 位需要进行比较
ADDRESS_BIT7_COMPARE	地址的第 7 位需要进行比较
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* define which bits of ADDRESS[7:1] need to compare with the incoming address byte */
```

```
i2c_address_bit_compare_config(I2C0, ADDRESS_BIT1_COMPARE);
```

函数 i2c_address_disable

函数i2c_address_disable描述见下表：

表 3-665. 函数 i2c_address_disable

函数名称	i2c_address_disable
函数原型	void i2c_address_disable(uint32_t i2c_periph);
功能描述	禁能从机模式下 I2C 地址
先决条件	-
被调用函数	-
输入参数{in}	
i2c_periph	I2C 外设
I2Cx	(x=0,1,2,3)
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* disable i2c address in slave mode */
```

```
i2c_address_disable(I2C0);
```

函数 i2c_second_address_config

函数i2c_second_address_config描述见下表：

表 3-666. 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 3-667. 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 3-668. 函数 i2c_receved_address_get

函数名称	i2c_receved_address_get
函数原型	uint32_t i2c_receved_address_get(uint32_t i2c_periph);
功能描述	获取从机模式下匹配成功的地址
先决条件	-
被调用函数	-
输入参数{in}	
i2c_periph	I2C 外设
I2Cx	(x=0,1,2,3)
输出参数{out}	
-	-
返回值	
uint32_t	0x00..0x7F

例如：

```
/* get received match address in slave mode */
```

```
uint32_t address;
```

```
address = i2c_receved_address_get(I2C0);
```

函数 i2c_slave_byte_control_enable

函数i2c_slave_byte_control_enable描述见下表：

表 3-669. 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 3-670. 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 3-671. 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 3-672. 函数 i2c_nack_disable

函数名称	i2c_nack_disable
函数原型	void i2c_nack_disable(uint32_t i2c_periph);
功能描述	从机模式下产生 ACK
先决条件	-
被调用函数	-
输入参数{in}	
i2c_periph	I2C 外设
I2Cx	(x=0,1,2,3)
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* generate a ACK in slave mode */
```

```
i2c_nack_disable(I2C0);
```

函数 i2c_wakeup_from_deepsleep_enable

函数i2c_wakeup_from_deepsleep_enable描述见下表：

表 3-673. 函数 i2c_wakeup_from_deepsleep_enable

函数名称	i2c_wakeup_from_deepsleep_enable
函数原型	void i2c_wakeup_from_deepsleep_enable(uint32_t i2c_periph);
功能描述	使能从Deep-sleep模式中唤醒
先决条件	-
被调用函数	-
输入参数{in}	
i2c_periph	I2C外设
I2Cx	(x=0,1,2,3)
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* enable wakeup from Deep-sleep mode */
```

```
i2c_wakeup_from_deepsleep_enable(I2C0);
```

函数 i2c_wakeup_from_deepsleep_disable

函数i2c_wakeup_from_deepsleep_disable描述见下表：

表 3-674. 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 3-675. 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 3-676. 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 3-677. 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 3-678. 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 3-679. 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 3-680. 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	received data (0x00-0xFF)

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 3-681. 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 3-682. 函数 i2c_reload_disable

函数名称	i2c_reload_disable
函数原型	void i2c_reload_disable(uint32_t i2c_periph);
功能描述	禁用 I2C 重载模式
先决条件	-
被调用函数	-
输入参数{in}	
i2c_periph	I2C 外设
I2Cx	(x=0,1,2,3)
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* disable I2C reload mode */
```

```
i2c_reload_disable(I2C0);
```

函数 i2c_transfer_byte_number_config

函数i2c_transfer_byte_number_config描述见下表：

表 3-683. 函数 i2c_transfer_byte_number_config

函数名称	i2c_transfer_byte_number_config
函数原型	void i2c_transfer_byte_number_config(uint32_t i2c_periph, uint32_t byte_number);
功能描述	配置待发送字节数
先决条件	-
被调用函数	-
输入参数{in}	
i2c_periph	I2C 外设

<i>I2Cx</i>	(x=0,1,2,3)
输入参数{in}	
byte_number	0x0-0xFF, 待传输的字节数
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* configure number of bytes to be transferred */
```

```
i2c_transfer_byte_number_config(I2C0, 0xFF);
```

函数 i2c_dma_enable

函数i2c_dma_enable描述见下表:

表 3-684. 函数 i2c_dma_enable

函数名称	i2c_dma_enable
函数原型	void i2c_dma_enable(uint32_t i2c_periph, uint8_t dma);
功能描述	使能发送/接收模式下 DMA
先决条件	-
被调用函数	-
输入参数{in}	
i2c_periph	I2C 外设
<i>I2Cx</i>	(x=0,1,2,3)
输入参数{in}	
dma	I2C DMA
<i>I2C_DMA_TRANSMIT</i>	采用 DMA 方式发送数据
<i>I2C_DMA_RECEIVE</i>	采用 DMA 方式接收数据
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* enable I2C DMA for transmission or reception */
```

```
i2c_dma_enable(I2C0, I2C_DMA_RECEIVE);
```

函数 i2c_dma_disable

函数i2c_dma_disable描述见下表:

表 3-685. 函数 i2c_dma_disable

函数名称	i2c_dma_disable
函数原型	void i2c_dma_disable(uint32_t i2c_periph, uint8_t dma);
功能描述	禁能发送/接收模式下 DMA
先决条件	-
被调用函数	-
输入参数{in}	
i2c_periph	I2C 外设
I2Cx	(x=0,1,2,3)
输入参数{in}	
dma	I2C DMA
I2C_DMA_TRANSMIT	采用 DMA 方式发送数据
I2C_DMA_RECEIVE	采用 DMA 方式接收数据
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* disable I2C DMA for transmission or reception */
```

```
i2c_dma_disable(I2C0, I2C_DMA_RECEIVE);
```

函数 i2c_pec_transfer

函数i2c_pec_transfer描述见下表:

表 3-686. 函数 i2c_pec_transfer

函数名称	i2c_pec_transfer
函数原型	void i2c_pec_transfer(uint32_t i2c_periph);
功能描述	I2C 传输 PEC 值
先决条件	-
被调用函数	-
输入参数{in}	
i2c_periph	I2C 外设
I2Cx	(x=0,1,2,3)
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* I2C transfers PEC value */
```

```
i2c_pec_transfer(I2C0);
```

函数 i2c_pec_enable

函数i2c_pec_enable描述见下表:

表 3-687. 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 3-688. 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 3-689. 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 3-690. 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 3-691. 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 3-692. 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 3-693. 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 3-694. 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 3-695. 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 3-696. 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 3-697. 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 3-698. 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 3-699. 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 3-700. 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 3-701. 函数 i2c_bus_timeout_a_config

函数名称	i2c_bus_timeout_a_config
函数原型	void i2c_bus_timeout_a_config(uint32_t i2c_periph, uint32_t timeout);
功能描述	配置总线超时 A
先决条件	-
被调用函数	-
输入参数{in}	
i2c_periph	I2C 外设
I2Cx	(x=0,1,2,3)
输入参数{in}	
timeout	0-0xffff, 总线超时 A
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* configure bus timeout A */
```

```
i2c_bus_timeout_a_config(I2C0, 0xff);
```

函数 i2c_idle_clock_timeout_config

函数i2c_idle_clock_timeout_config描述见下表：

表 3-702. 函数 i2c_idle_clock_timeout_config

函数名称	i2c_idle_clock_timeout_config
函数原型	void i2c_idle_clock_timeout_config(uint32_t i2c_periph, uint32_t timeout);
功能描述	配置空闲时钟超时检测
先决条件	-
被调用函数	-
输入参数{in}	
i2c_periph	I2C 外设
I2Cx	(x=0,1,2,3)

输入参数{in}	
timeout	总线超时 A
<i>BUSTOA_DETECT_SCL_LOW</i>	BUSTOA 用于检测 SCL 低电平超时
<i>BUSTOA_DETECT_IDLE</i>	BUSTOA 用于检测总线空闲情况下 SCL 和 SDA 高电平超时
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* configure idle clock timeout detection */
```

```
i2c_idle_clock_timeout_config(I2C0, BUSTOA_DETECT_SCL_LOW);
```

函数 i2c_flag_get

函数 i2c_flag_get 描述见下表：

表 3-703. 函数 i2c_flag_get

函数名称	i2c_flag_get
函数原型	FlagStatus i2c_flag_get(uint32_t i2c_periph, uint32_t flag);
功能描述	获取 I2C 标志位
先决条件	-
被调用函数	-
输入参数{in}	
i2c_periph	I2C 外设
<i>I2Cx</i>	(x=0,1,2,3)
输入参数{in}	
flag	I2C 标志位
<i>I2C_FLAG_TBE</i>	发送期间 I2C_TDATA 寄存器空标志
<i>I2C_FLAG_TI</i>	发送中断标志
<i>I2C_FLAG_RBNE</i>	接收期间 I2C_RDATA 非空标志
<i>I2C_FLAG_ADDSEND</i>	从机模式下，接收到的地址与自身地址匹配
<i>I2C_FLAG_NACK</i>	NACK 标志
<i>I2C_FLAG_STPDET</i>	从机模式下检测到 STOP 信号
<i>I2C_FLAG_TC</i>	主机模式下传输完成标志
<i>I2C_FLAG_TCR</i>	传输完成重载标志
<i>I2C_FLAG_BERR</i>	总线错误标志
<i>I2C_FLAG_LOSTARB</i>	仲裁丢失标志
<i>I2C_FLAG_OUERR</i>	从机模式下，过载/欠载错误标志
<i>I2C_FLAG_PECERR</i>	PEC 错误标志
<i>I2C_FLAG_TIMEOUT</i>	超时标志

<i>I2C_FLAG_SMBALT</i>	SMBus 报警标志
<i>I2C_FLAG_I2CBSY</i>	忙标志
<i>I2C_FLAG_TR</i>	从机模式下, I2C 作为发送器还是接收器标志
输出参数{out}	
-	-
返回值	
FlagStatus	SET / RESET

例如:

```
/* get I2C flag status */
```

```
FlagStatus flag_state = RESET;
```

```
flag_state = i2c_flag_get(I2C0, I2C_FLAG_TBE);
```

函数 i2c_flag_clear

函数i2c_flag_clear描述见下表:

表 3-704. 函数 i2c_flag_clear

函数名称	i2c_flag_clear
函数原型	void i2c_flag_clear(uint32_t i2c_periph, uint32_t flag);
功能描述	清除 I2C 标志位
先决条件	-
被调用函数	-
输入参数{in}	
i2c_periph	I2C 外设
<i>I2Cx</i>	(x=0,1,2,3)
输入参数{in}	
flag	I2C 标志位
<i>I2C_FLAG_ADDSEND</i>	从机模式下, 接收到的地址与自身地址匹配
<i>I2C_FLAG_NACK</i>	NACK 标志
<i>I2C_FLAG_STPDET</i>	从机模式下检测到 STOP 信号
<i>I2C_FLAG_BERR</i>	总线错误标志
<i>I2C_FLAG_LOSTARB</i>	仲裁丢失标志
<i>I2C_FLAG_OUERR</i>	从机模式下, 过载/欠载错误标志
<i>I2C_FLAG_PECERR</i>	PEC 错误标志
<i>I2C_FLAG_TIMEOUT</i>	超时标志
<i>I2C_FLAG_SMBALT</i>	SMBus 报警标志
输出参数{out}	
-	-
返回值	
-	-

例如:


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

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

函数 i2c_interrupt_enable

函数i2c_interrupt_enable描述见下表：

表 3-705. 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 3-706. 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 3-707. 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 3-645. 枚举类型 i2c_interrupt_flag_enum 。
I2C_INT_FLAG_TI	发送中断标志
I2C_INT_FLAG_RBNE	接收期间 I2C_RDATA 非空中断标志
I2C_INT_FLAG_ADDS END	从机模式下，接收到的地址与自身地址匹配中断标志
I2C_INT_FLAG_NACK	NACK 中断标志

I2C_INT_FLAG_STPDET	从机模式下检测到 STOP 信号中断标志
I2C_INT_FLAG_TC	主机模式下传输完成中断标志
I2C_INT_FLAG_TCR	传输完成重载中断标志
I2C_INT_FLAG_BERR	总线错误中断标志
I2C_INT_FLAG_LOSTARB	仲裁丢失中断标志
I2C_INT_FLAG_OUERR	从机模式下，过载/欠载错误中断标志
I2C_INT_FLAG_PECERR	PEC 错误中断标志
I2C_INT_FLAG_TIMEOUTUT	超时中断标志
I2C_INT_FLAG_SMBALTL	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 3-708. 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 3-645. 枚举类型 i2c_interrupt_flag_enum 。
I2C_INT_FLAG_ADDS	从机模式下，接收到的地址与自身地址匹配中断标志

<i>END</i>	
<i>I2C_INT_FLAG_NACK</i>	NACK 中断标志
<i>I2C_INT_FLAG_STPD ET</i>	从机模式下检测到 STOP 信号中断标志
<i>I2C_INT_FLAG_BERR</i>	总线错误中断标志
<i>I2C_INT_FLAG_LOSTA RB</i>	仲裁丢失中断标志
<i>I2C_INT_FLAG_OUER R</i>	从机模式下，过载/欠载错误中断标志
<i>I2C_INT_FLAG_PEC RR</i>	PEC 错误中断标志
<i>I2C_INT_FLAG_TIMEO UT</i>	超时中断标志
<i>I2C_INT_FLAG_SMBA LT</i>	SMBus 报警中断标志
输出参数{out}	
-	-
返回值	
-	-

例如：

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

```
i2c_interrupt_flag_clear(I2C0, I2C_INT_FLAG_BERR);
```

3.21. LPTIMER

LPTIMER 是一个 16 位 / 32 位的定时器，它能够在除待机模式以外的所有功耗模式下运行。LPTIMER 提供了灵活的时钟机制，在将功耗降至最低的同时，还可以实现所需的功能和性能。章节 [3.21.1](#) 描述了 LPTIMER 的寄存器列表，章节 [3.21.2](#) 对 LPTIMER 库函数进行说明。

3.21.1. 外设寄存器说明

LPTIMER 寄存器列表如下表所示：

表 3-709. LPTIMER 寄存器

寄存器名称	寄存器描述
LPTIMER_INTF	中断标志寄存器
LPTIMER_INTC	中断标志清除寄存器
LPTIMER_INTEN	中断使能寄存器
LPTIMER_CTL0	控制寄存器0
LPTIMER_CTL1	控制寄存器1
LPTIMER_CMPV	比较寄存器

寄存器名称	寄存器描述
LPTIMER_CAR	计数器自动重载寄存器
LPTIMER_CNT	计数器寄存器
LPTIMER_EIRMP	外部输入重映射寄存器
LPTIMER_INHLCMV	输入高电平计数最大值寄存器

3.21.2. 外设库函数说明

LPTIMER库函数列表如下表所示：

表 3-710. LPTIMER 库函数

库函数名称	库函数描述
lptimer_deinit	复位LPTIMER
lptimer_struct_para_init	将LPTIMER初始化结构体中所有参数初始化为默认值
lptimer_init	初始化LPTIMER
lptimer_inputremap	配置外部输入重映射
lptimer_register_shadow_enable	使能LPTIMER_CAR和LPTIMER_CMPV寄存器的影子寄存器功能
lptimer_register_shadow_disable	禁能LPTIMER_CAR和LPTIMER_CMPV寄存器的影子寄存器功能
lptimer_timeout_enable	使能LPTIMER超时功能
lptimer_timeout_disable	禁能LPTIMER超时功能
lptimer_counter_sync_reset	LPTIMER计时器同步复位
lptimer_counter_read_async_reset_enable	LPTIMER计时器读操作异步复位使能
lptimer_counter_read_async_reset_disable	LPTIMER计时器读操作异步复位禁能
lptimer_continue_start	LPTIMER连续计数模式启动
lptimer_single_start	LPTIMER单次计数模式启动
lptimer_stop	LPTIMER停止计数
lptimer_counter_read	读取LPTIMER的计数器值
lptimer_autoreload_read	读取LPTIMER的自动重载寄存器值
lptimer_compare_read	读取LPTIMER的比较寄存器值
lptimer_autoreload_value_config	配置LPTIMER的自动重载寄存器值
lptimer_compare_value_config	配置LPTIMER的比较寄存器值
lptimer_decodemode0_enable	使能编码器模式0功能
lptimer_decodemode1_enable	使能编码器模式1功能
lptimer_decodemode_disable	禁能编码器模式功能
lptimer_highlevelcounter_enable	使能外部输入高电平计数器
lptimer_highlevelcounter_disable	禁能外部输入高电平计数器
lptimer_flag_get	获取LPTIMER的标志位
lptimer_flag_clear	清除LPTIMER的标志位
lptimer_interrupt_enable	使能LPTIMER中断

库函数名称	库函数描述
lptimer_interrupt_disable	禁能LPTIMER中断
lptimer_interrupt_flag_get	获取LPTIMER的中断标志位
lptimer_interrupt_flag_clear	清除LPTIMER的中断标志位

结构体 lptimer_parameter_struct

表 3-711. 结构体 lptimer_parameter_struct

成员名称	功能描述
clocksource	时钟源 (LPTIMER_INTERNALCLK, LPTIMER_EXTERNALCLK)
prescaler	计数器时钟预分频 (LPTIMER_PSC_x, x=1,2,4,8..128)
extclockpolarity	计数器的外部时钟极性 (LPTIMER_EXTERNALCLK_RISING, LPTIMER_EXTERNALCLK_FALLING, LPTIMER_EXTERNALCLK_BOTH)
extclockfilter	外部时钟滤波 (LPTIMER_EXTERNALCLK_FILTEROFF, LPTIMER_EXTERNALCLK_FILTER_2, LPTIMER_EXTERNALCLK_FILTER_4, LPTIMER_EXTERNALCLK_FILTER_8)
triggermode	触发模式 (LPTIMER_TRIGGER_SOFTWARE, LPTIMER_TRIGGER_EXTERNALRISING, LPTIMER_TRIGGER_EXTERNALFALLING, LPTIMER_TRIGGER_EXTERNALBOTH)
extriggersource	外部触发源 (LPTIMER_EXTRIGGER_GPIO, LPTIMER_EXTRIGGER_RTCALARM0, LPTIMER_EXTRIGGER_RTCALARM1, LPTIMER_EXTRIGGER_RTCTAMP0, LPTIMER_EXTRIGGER_RTCTAMP1, LPTIMER_EXTRIGGER_RTCTAMP2, LPTIMER_EXTRIGGER_CMP0_OUT, LPTIMER_EXTRIGGER_CMP1_OUT, LPTIMER_EXTRIGGER_CMP2_OUT, LPTIMER_EXTRIGGER_CMP3_OUT, LPTIMER_EXTRIGGER_CMP4_OUT, LPTIMER_EXTRIGGER_CMP5_OUT, LPTIMER_EXTRIGGER_CMP6_OUT, LPTIMER_EXTRIGGER_CMP7_OUT)
extriggerfilter	外部触发滤波 (LPTIMER_TRIGGER_FILTEROFF, LPTIMER_TRIGGER_FILTER_2, LPTIMER_TRIGGER_FILTER_4, LPTIMER_TRIGGER_FILTER_8)
outputpolarity	输出极性 (LPTIMER_OUTPUT_NOTINVERTED, LPTIMER_OUTPUT_INVERTED)
outputmode	输出模式 (LPTIMER_OUTPUT_PWMORSINGLE, LPTIMER_OUTPUT_SET)
countersource	计数器时钟源 (LPTIMER_COUNTER_INTERNAL,

成员名称	功能描述
	LPTIMER_COUNTER_EXTERNAL)

函数 lptimer_deinit

函数lptimer_deinit描述见下表:

表 3-712. 函数 lptimer_deinit

函数名称	lptimer_deinit
函数原型	void lptimer_deinit(void);
功能描述	复位LPTIMER
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 LPTIMER */
```

```
lptimer_deinit();
```

函数 lptimer_struct_para_init

函数lptimer_struct_para_init描述见下表:

表 3-713. 函数 lptimer_struct_para_init

函数名称	lptimer_struct_para_init
函数原型	void lptimer_struct_para_init(lptimer_parameter_struct *initpara);
功能描述	将LPTIMER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	LPTIMER初始化结构体, 结构体成员参考 表3-711. 结构体 lptimer_parameter_struct
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* initialize LPTIMER init parameter struct with a default value */
```

```
lptimer_parameter_struct lptimer_initpara;

lptimer_struct_para_init(&lptimer_initpara);
```

函数 lptimer_init

函数lptimer_init描述见下表：

表 3-714. 函数 lptimer_init

函数名称	lptimer_init
函数原型	void lptimer_init(lptimer_parameter_struct *initpara);
功能描述	初始化LPTIMER
先决条件	-
被调用函数	-
输入参数{in}	
initpara	LPTIMER初始化结构体，结构体成员参考 表3-711. 结构体 lptimer_parameter_struct
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* initialize LPTIMER */

lptimer_parameter_struct lptimer_structure;

lptimer_structure.clocksource      = LPTIMER_INTERNALCLK;

lptimer_structure.prescaler       = LPTIMER_PSC_16;

lptimer_structure.extclockpolarity = LPTIMER_EXTERNALCLK_RISING;

lptimer_structure.extclockfilter  = LPTIMER_EXTERNALCLK_FILTEROFF;

lptimer_structure.triggermode     = LPTIMER_TRIGGER_SOFTWARE;

lptimer_structure.extriggersource = LPTIMER_EXTRIGGER_GPIO;

lptimer_structure.extriggerfilter = LPTIMER_TRIGGER_FILTEROFF;

lptimer_structure.outputpolarity  = LPTIMER_OUTPUT_NOTINVERTED;

lptimer_structure.outputmode      = LPTIMER_OUTPUT_PWMORSINGLE;

lptimer_structure.countersource   = LPTIMER_COUNTER_INTERNAL;

lptimer_init(&lptimer_structure);
```


函数 lptimer_inputremap

函数lptimer_inputremap描述见下表：

表 3-715. 函数 lptimer_inputremap

函数名称	lptimer_inputremap
函数原型	void lptimer_inputremap(uint32_t input0remap, uint32_t input1remap);
功能描述	配置外部输入重映射
先决条件	-
被调用函数	-
输入参数{in}	
input0remap	外部输入0重映射
LPTIMER_INPUT0_GPIO	外部输入0重映射到GPIO
LPTIMER_INPUT0_CMP0_OUT	外部输入0重映射到CMP0_OUT
LPTIMER_INPUT0_CMP2_OUT	外部输入0重映射到CMP2_OUT
LPTIMER_INPUT0_CMP4_OUT	外部输入0重映射到CMP4_OUT
LPTIMER_INPUT0_CMP6_OUT	外部输入0重映射到CMP6_OUT
输入参数{in}	
Input1remap	外部输入1重映射
LPTIMER_INPUT1_GPIO	外部输入1重映射到GPIO
LPTIMER_INPUT1_CMP1_OUT	外部输入1重映射到CMP1_OUT
LPTIMER_INPUT1_CMP3_OUT	外部输入1重映射到CMP3_OUT
LPTIMER_INPUT1_CMP5_OUT	外部输入1重映射到CMP5_OUT
LPTIMER_INPUT1_CMP7_OUT	外部输入1重映射到CMP7_OUT
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* configure LPTIMER inputs is connected to GPIO */
```

```
lptimer_inputremap(LPTIMER_INPUT0_GPIO, LPTIMER_INPUT1_GPIO);
```

函数 lptimer_register_shadow_enable

函数lptimer_register_shadow_enable描述见下表：

表 3-716. 函数 lptimer_register_shadow_enable

函数名称	lptimer_register_shadow_enable
函数原型	void lptimer_register_shadow_enable(void);
功能描述	使能LPTIMER_CAR和LPTIMER_CMPV寄存器的影子寄存器功能
先决条件	-
被调用函数	-
输入参数{in}	
-	-
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* enable the LPTIMER_CAR and LPTIMER_CMPV registers shadow function */  
lptimer_register_shadow_enable();
```

函数 lptimer_register_shadow_disable

函数lptimer_register_shadow_disable描述见下表：

表 3-717. 函数 lptimer_register_shadow_disable

函数名称	lptimer_register_shadow_disable
函数原型	void lptimer_register_shadow_disable(void);
功能描述	禁能LPTIMER_CAR和LPTIMER_CMPV寄存器的影子寄存器功能
先决条件	-
被调用函数	-
输入参数{in}	
-	-
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* disable the LPTIMER_CAR and LPTIMER_CMPV registers shadow function */  
lptimer_register_shadow_disable();
```

函数 lptimer_timeout_enable

函数lptimer_timeout_enable描述见下表：

表 3-718. 函数 lptimer_timeout_enable

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

例如：

```
/* enable the LPTIMER TIMEOUT function */
```

```
lptimer_timeout_enable();
```

函数 lptimer_timeout_disable

函数lptimer_timeout_disable描述见下表：

表 3-719. 函数 lptimer_register_shadow_disable

函数名称	lptimer_timeout_disable
函数原型	void lptimer_timeout_disable(void);
功能描述	禁能LPTIMER超时功能
先决条件	-
被调用函数	-
输入参数{in}	
-	-
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* disable the LPTIMER TIMEOUT function */
```

```
lptimer_timeout_disable();
```

函数 lptimer_counter_sync_reset

函数lptimer_counter_sync_reset描述见下表：

表 3-720. 函数 lptimer_counter_sync_reset

函数名称	lptimer_counter_sync_reset
函数原型	void lptimer_counter_sync_reset(void);
功能描述	LPTIMER计时器同步复位
先决条件	-
被调用函数	-
输入参数{in}	
-	-
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* LPTIMER counter reset synchronously */
```

```
lptimer_counter_sync_reset();
```

函数 lptimer_counter_read_async_reset_enable

函数lptimer_counter_read_async_reset_enable描述见下表：

表 3-721. 函数 lptimer_counter_read_async_reset_enable

函数名称	lptimer_counter_read_async_reset_enable
函数原型	void lptimer_counter_read_async_reset_enable(void);
功能描述	使能LPTIMER计时器读异步复位
先决条件	-
被调用函数	-
输入参数{in}	
-	-
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* read cause counter asynchronously reset enable */
```

```
lptimer_counter_read_async_reset_enable();
```

函数 lptimer_counter_read_async_reset_disable

函数lptimer_counter_read_async_reset_disable描述见下表：

表 3-722. 函数 lptimer_counter_read_async_reset_disable

函数名称	lptimer_counter_read_async_reset_disable
函数原型	void lptimer_counter_read_async_reset_disable(void);
功能描述	禁止LPTIMER计时器读异步复位
先决条件	-
被调用函数	-
输入参数{in}	
-	-
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* read cause counter asynchronously reset disable */
```

```
lptimer_counter_read_async_reset_disable();
```

函数 lptimer_countinue_start

函数lptimer_countinue_start描述见下表：

表 3-723. 函数 lptimer_countinue_start

函数名称	lptimer_countinue_start
函数原型	void lptimer_countinue_start(uint32_t autoreload, uint32_t compare);
功能描述	LPTIMER连续计数模式启动
先决条件	-
被调用函数	-
输入参数{in}	
-	-
输入参数{in}	
autoreload	自动重载寄存器值，0~0xFFFF
输入参数{in}	
compare	比较寄存器值，0~0xFFFF
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* LPTIMER countinue start */
```

```
lptimer_countinue_start(0xFFFF, 0x7FFF);
```

函数 lptimer_single_start

函数lptimer_single_start描述见下表：

表 3-724. 函数 lptimer_countinue_start

函数名称	lptimer_single_start
函数原型	void lptimer_single_start(uint32_t autoreload, uint32_t compare);
功能描述	LPTIMER单次计数模式启动
先决条件	-
被调用函数	-
输入参数{in}	
autoreload	自动重载寄存器值，0~0xFFFF
输入参数{in}	
compare	比较寄存器值，0~0xFFFF
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* LPTIMER single start */
```

```
lptimer_single_start(0xFFFF, 0x7FFF);
```

函数 lptimer_stop

函数lptimer_stop描述见下表：

表 3-725. 函数 lptimer_stop

函数名称	lptimer_stop
函数原型	void lptimer_stop(void);
功能描述	LPTIMER停止计数
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 LPTIMER */
```

lptimer_stop();

函数 lptimer_counter_read

函数lptimer_counter_read描述见下表:

表 3-726. 函数 lptimer_counter_read

函数名称	lptimer_counter_read
函数原型	uint32_t lptimer_counter_read(void);
功能描述	读取LPTIMER的计数器值
先决条件	-
被调用函数	-
输入参数{in}	
-	-
输出参数{out}	
-	-
返回值	
uint32_t	32位计数器值（0~0xFFFF）

例如:

```
/* read LPTIMER current counter value */
```

```
uint32_t i = 0;
```

```
i = lptimer_counter_read();
```

函数 lptimer_autoreload_read

函数lptimer_autoreload_read描述见下表:

表 3-727. 函数 lptimer_autoreload_read

函数名称	lptimer_autoreload_read
函数原型	uint32_t lptimer_autoreload_read(void);
功能描述	读取LPTIMER的自动重载寄存器值
先决条件	-
被调用函数	-
输入参数{in}	
-	-
输出参数{out}	
-	-
返回值	
uint32_t	32位自动重载寄存器值（0~0xFFFF）

例如:

```
/* read LPTIMER auto reload value */
```

```
uint32_t i = 0;  
  
i = lptimer_autoreload_read();
```

函数 lptimer_compare_read

函数lptimer_compare_read描述见下表:

表 3-728. 函数 lptimer_compare_read

函数名称	lptimer_compare_read
函数原型	uint32_t lptimer_compare_read(void);
功能描述	读取LPTIMER的比较寄存器值
先决条件	-
被调用函数	-
输入参数{in}	
-	-
输出参数{out}	
-	-
返回值	
uint32_t	32位比较寄存器值（0~0xFFFF）

例如:

```
/* read LPTIMER compare value */  
  
uint32_t i = 0;  
  
i = lptimer_compare_read();
```

函数 lptimer_autoreload_value_config

函数lptimer_autoreload_value_config描述见下表:

表 3-729. 函数 lptimer_autoreload_value_config

函数名称	lptimer_autoreload_value_config
函数原型	void lptimer_autoreload_value_config(uint32_t autoreload);
功能描述	配置LPTIMER的自动重载寄存器值
先决条件	-
被调用函数	-
输入参数{in}	
autoreload	自动重载寄存器值（0~0xFFFF）
输出参数{out}	
-	-
返回值	
-	-

例如:


```
/* configure LPTIMER autoreload register value */
```

```
lptimer_autoreload_value_config(0x00FF);
```

函数 lptimer_compare_value_config

函数lptimer_compare_value_config描述见下表：

表 3-730. 函数 lptimer_compare_value_config

函数名称	lptimer_compare_value_config
函数原型	void lptimer_compare_value_config(uint32_t compare);
功能描述	配置LPTIMER的比较寄存器值
先决条件	-
被调用函数	-
输入参数{in}	
compare	比较寄存器值（0~0xFFFF）
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* configure LPTIMER compare value */
```

```
lptimer_compare_value_config(0x00FF);
```

函数 lptimer_decodemode0_enable

函数lptimer_decodemode0_enable描述见下表：

表 3-731. 函数 lptimer_decodemode0_enable

函数名称	lptimer_decodemode0_enable
函数原型	void lptimer_decodemode0_enable();
功能描述	使能编码器模式0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 decode mode 0 */
```

```
lptimer_decodemode0_enable();
```

函数 lptimer_decodemode1_enable

函数lptimer_decodemode1_enable描述见下表：

表 3-732. 函数 lptimer_decodemode1_enable

函数名称	lptimer_decodemode1_enable
函数原型	void lptimer_decodemode1_enable(void);
功能描述	使能编码器模式1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 decode mode 1 */
```

```
lptimer_decodemode1_enable();
```

函数 lptimer_decodemode_disable

函数lptimer_decodemode_disable描述见下表：

表 3-733. 函数 lptimer_decodemode_disable

函数名称	lptimer_decodemode_disable
函数原型	void lptimer_decodemode_disable(void);
功能描述	禁能编码器模式0/1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 decode mode 0/1 */
```

```
lptimer_decodemode_disable();
```

函数 lptimer_highlevelcounter_enable

函数lptimer_highlevelcounter_enable描述见下表：

表 3-734. 函数 lptimer_highlevelcounter_enable

函数名称	lptimer_highlevelcounter_enable
函数原型	void lptimer_highlevelcounter_enable(uint32_t maxvalue);
功能描述	使能外部输入高电平计数器
先决条件	-
被调用函数	-
输入参数{in}	
maxvalue	外部输入高电平计数值最大值（0~0x03FFFFFF）
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* enable external input high level counter */  
lptimer_highlevelcounter_enable(0x00007FFF);
```

函数 lptimer_highlevelcounter_disable

函数lptimer_highlevelcounter_disable描述见下表：

表 3-735. 函数 lptimer_highlevelcounter_disable

函数名称	lptimer_highlevelcounter_disable
函数原型	void lptimer_highlevelcounter_disable(void);
功能描述	禁能外部输入高电平计数器
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 external input high level counter */  
lptimer_highlevelcounter_disable();
```

函数 `lptimer_flag_get`

函数 `lptimer_flag_get` 描述见下表:

表 3-736. 函数 `lptimer_flag_get`

函数名称	<code>lptimer_flag_get</code>
函数原型	<code>FlagStatus lptimer_flag_get(uint32_t flag);</code>
功能描述	获取LPTIMER的标志位
先决条件	-
被调用函数	-
输入参数{in}	
flag	LPTIMER 的标志位
<code>LPTIMER_FLAG_CMPVM</code>	比较寄存器匹配标志位
<code>LPTIMER_FLAG_CARM</code>	计数器自动重载寄存器匹配标志位
<code>LPTIMER_FLAG_ETEDEV</code>	外部触发边沿事件标志位
<code>LPTIMER_FLAG_CMPVUP</code>	比较寄存器更新标志位
<code>LPTIMER_FLAG_CARUP</code>	计数器自动重载寄存器更新标志位
<code>LPTIMER_FLAG_UP</code>	LPTIMER计数器由向下计数改为向上计数标志位
<code>LPTIMER_FLAG_DOWN</code>	LPTIMER计数器由向上计数改为向下计数标志位
<code>LPTIMER_FLAG_HLCMVUP</code>	输入高电平计数最大值寄存器更新标志位
<code>LPTIMER_FLAG_IN0_HLCO</code>	LPTIMER_INx (x=0,1) 高电平计数器溢出标志位
<code>LPTIMER_FLAG_IN0_HLOE</code>	LPTIMER_IN0 和 LPTIMER_IN1 高电平重叠错误标志位
<code>LPTIMER_FLAG_IN0_RFOE</code>	LPTIMER_IN0 和 LPTIMER_IN1 下降沿和上升沿重叠错误标志位
<code>LPTIMER_FLAG_IN0_OE</code>	LPTIMER_IN0 错误标志位
<code>LPTIMER_FLAG_IN1_OE</code>	LPTIMER_IN1 错误标志位
输出参数{out}	
-	-
返回值	
FlagStatus	SET或者RESET

例如:

```
/* get LPTIMER flag */
```

```
FlagStatus Flag_status = RESET;
```

```
Flag_status = lptimer_flag_get(LPTIMER_FLAG_CMPVM);
```

函数 lptimer_flag_clear

函数lptimer_flag_clear描述见下表：

表 3-737. 函数 timer_flag_clear

函数名称	lptimer_flag_clear
函数原型	void lptimer_flag_clear(uint32_t flag);
功能描述	清除LPTIMER标志位
先决条件	-
被调用函数	-
输入参数{in}	
flag	LPTIMER的标志位
LPTIMER_FLAG_C MPVM	比较寄存器匹配标志位
LPTIMER_FLAG_C ARM	计数器自动重载寄存器匹配标志位
LPTIMER_FLAG_E TEDEV	外部触发边沿事件标志位
LPTIMER_FLAG_C MPVUP	比较寄存器更新标志位
LPTIMER_FLAG_C ARUP	计数器自动重载寄存器更新标志位
LPTIMER_FLAG_U P	LPTIMER计数器由向下计数改为向上计数标志位
LPTIMER_FLAG_D OWN	LPTIMER计数器由向上计数改为向下计数标志位
LPTIMER_FLAG_H LCMVUP	输入高电平计数最大值寄存器更新标志位
LPTIMER_FLAG_IN HLCO	LPTIMER_INx (x=0,1) 高电平计数器溢出标志位
LPTIMER_FLAG_IN HLOE	LPTIMER_IN0和LPTIMER_IN1高电平重叠错误标志位
LPTIMER_FLAG_IN RFOE	LPTIMER_IN0和LPTIMER_IN1下降沿和上升沿重叠错误标志位
LPTIMER_FLAG_IN 0E	LPTIMER_IN0错误标志位
LPTIMER_FLAG_IN 1E	LPTIMER_IN1错误标志位
输出参数{out}	

-	-
返回值	
-	-

例如:

```
/* clear LPTIMER flag */
```

```
lptimer_flag_clear(LPTIMER_FLAG_CMPVM);
```

函数 lptimer_interrupt_enable

函数lptimer_interrupt_enable描述见下表:

表 3-738. 函数 lptimer_interrupt_enable

函数名称	lptimer_interrupt_enable
函数原型	void lptimer_interrupt_enable(uint32_t interrupt);
功能描述	使能LPTIMER中断
先决条件	-
被调用函数	-
输入参数{in}	
interrupt	LPTIMER中断源
LPTIMER_INT_CM PVM	比较寄存器匹配中断
LPTIMER_INT_C ARM	计数器自动重载寄存器匹配中断
LPTIMER_INT_ET DEV	外部触发边沿事件中断
LPTIMER_INT_CM PVUP	比较寄存器更新中断
LPTIMER_INT_C RUP	计数器自动重载寄存器更新中断
LPTIMER_INT_UP	LPTIMER计数器由向下计数改为向上计数中断
LPTIMER_INT_D WN	LPTIMER计数器由向上计数改为向下计数中断
LPTIMER_INT_H MVUP	输入高电平计数最大值寄存器更新中断
LPTIMER_INT_I NHLCO	LPTIMER_INx (x=0,1) 高电平计数器溢出中断
LPTIMER_INT_I NHLLOE	LPTIMER_IN0和LPTIMER_IN1高电平重叠错误中断
LPTIMER_INT_I NHLFOE	LPTIMER_IN0和LPTIMER_IN1下降沿和上升沿重叠错误中断
LPTIMER_INT_I NHLFE	LPTIMER_IN0错误中断

LPTIMER_INT_IN1 E	LPTIMER_IN1错误中断
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* enable the LPTIMER interrupt */
```

```
lptimer_interrupt_enable(LPTIMER_INT_CMPVM);
```

函数 lptimer_interrupt_disable

函数lptimer_interrupt_disable描述见下表:

表 3-739. 函数 lptimer_interrupt_enable

函数名称	lptimer_interrupt_disable
函数原型	void lptimer_interrupt_disable(uint32_t interrupt);
功能描述	禁能LPTIMER中断
先决条件	-
被调用函数	-
输入参数{in}	
interrupt	LPTIMER中断源
LPTIMER_INT_CM PVM	比较寄存器匹配中断
LPTIMER_INT_CA RM	计数器自动重载寄存器匹配中断
LPTIMER_INT_ETE DEV	外部触发边沿事件中断
LPTIMER_INT_CM PVUP	比较寄存器更新中断
LPTIMER_INT_CA RUP	计数器自动重载寄存器更新中断
LPTIMER_INT_UP	LPTIMER计数器由向下计数改为向上计数中断
LPTIMER_INT_DO WN	LPTIMER计数器由向上计数改为向下计数中断
LPTIMER_INT_HLC MVUP	输入高电平计数最大值寄存器更新中断
LPTIMER_INT_INH LCO	LPTIMER_INx (x=0,1) 高电平计数器溢出中断
LPTIMER_INT_INH LOE	LPTIMER_IN0和LPTIMER_IN1高电平重叠错误中断
LPTIMER_INT_INR	LPTIMER_IN0和LPTIMER_IN1下降沿和上升沿重叠错误中断

FOE	
LPTIMER_INT_IN0E	LPTIMER_IN0错误中断
LPTIMER_INT_IN1E	LPTIMER_IN1错误中断
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* disable the LPTIMER interrupt */
```

```
lptimer_interrupt_disable(LPTIMER_INT_CMPVM);
```

函数 lptimer_interrupt_flag_get

函数lptimer_interrupt_flag_get描述见下表:

表 3-740. 函数 lptimer_interrupt_flag_get

函数名称	lptimer_interrupt_flag_get
函数原型	FlagStatus lptimer_interrupt_flag_get(uint32_t int_flag);
功能描述	获取LPTIMER中断标志位
先决条件	-
被调用函数	-
输入参数{in}	
int_flag	LPTIMER中断标志位
LPTIMER_INT_FLAG_CMPVM	比较寄存器匹配中断标志位
LPTIMER_INT_FLAG_CARM	计数器自动重载寄存器匹配中断标志位
LPTIMER_INT_FLAG_ETEDEV	外部触发边沿事件中断标志位
LPTIMER_INT_FLAG_CMPVUP	比较寄存器更新中断标志位
LPTIMER_INT_FLAG_CARUP	计数器自动重载寄存器更新中断标志位
LPTIMER_INT_FLAG_UP	LPTIMER计数器由向下计数改为向上计数中断标志位
LPTIMER_INT_FLAG_DOWN	LPTIMER计数器由向上计数改为向下计数中断标志位
LPTIMER_INT_FLAG_HLCMVUP	输入高电平计数最大值寄存器更新中断标志位
LPTIMER_INT_FLAG	LPTIMER_INx (x=0,1) 高电平计数器溢出中断标志位

<code>G_INHLC0</code>	
<code>LPTIMER_INT_FLAG_INHLOE</code>	LPTIMER_IN0和LPTIMER_IN1高电平重叠错误中断标志位
<code>LPTIMER_INT_FLAG_INRFOE</code>	LPTIMER_IN0和LPTIMER_IN1下降沿和上升沿重叠错误中断标志位
<code>LPTIMER_INT_FLAG_IN0E</code>	LPTIMER_IN0错误中断标志位
<code>LPTIMER_INT_FLAG_IN1E</code>	LPTIMER_IN1错误中断标志位
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* get LPTIMER interrupt flag */
```

```
FlagStatus Flag_interrupt = RESET;
```

```
Flag_interrupt = lptimer_interrupt_flag_get(LPTIMER_INT_FLAG_CMPVM);
```

函数 `lptimer_interrupt_flag_clear`

函数 `lptimer_interrupt_flag_clear` 描述见下表：

表 3-741. 函数 `lptimer_interrupt_flag_clear`

函数名称	<code>lptimer_interrupt_flag_clear</code>
函数原型	<code>void lptimer_interrupt_flag_clear(uint32_t int_flag);</code>
功能描述	清除LPTIMER中断标志位
先决条件	-
被调用函数	-
输入参数{in}	
<code>int_flag</code>	LPTIMER中断标志位
<code>LPTIMER_INT_FLAG_CMPVM</code>	比较寄存器匹配中断标志位
<code>LPTIMER_INT_FLAG_CARM</code>	计数器自动重载寄存器匹配中断标志位
<code>LPTIMER_INT_FLAG_ETEDEV</code>	外部触发边沿事件中断标志位
<code>LPTIMER_INT_FLAG_CMPVUP</code>	比较寄存器更新中断标志位
<code>LPTIMER_INT_FLAG_CARUP</code>	计数器自动重载寄存器更新中断标志位
<code>LPTIMER_INT_FLAG</code>	LPTIMER计数器由向下计数改为向上计数中断标志位

G_UP	
LPTIMER_INT_FLAG_DOWN	LPTIMER计数器由向上计数改为向下计数中断标志位
LPTIMER_INT_FLAG_HLCMVUP	输入高电平计数最大值寄存器更新中断标志位
LPTIMER_INT_FLAG_INHLCO	LPTIMER_INx (x=0,1) 高电平计数器溢出中断标志位
LPTIMER_INT_FLAG_INHLOE	LPTIMER_IN0和LPTIMER_IN1高电平重叠错误中断标志位
LPTIMER_INT_FLAG_INRFOE	LPTIMER_IN0和LPTIMER_IN1下降沿和上升沿重叠错误中断标志位
LPTIMER_INT_FLAG_IN0E	LPTIMER_IN0错误中断标志位
LPTIMER_INT_FLAG_IN1E	LPTIMER_IN1错误中断标志位
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* clear LPTIMER interrupt flag */
```

```
lptimer_interrupt_flag_clear(LPTIMER_INT_FLAG_CMPVM);
```

3.22. 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 [3.22.1](#) 描述了 NVIC 和 SysTick 的寄存器列表, 章节 [3.22.2](#) 对 MISC 库函数进行说明。

3.22.1. 外设寄存器说明

表 3-742. NVIC 寄存器

寄存器名称	寄存器描述
ISER ⁽¹⁾	中断使能寄存器
ICER ⁽¹⁾	中断禁能寄存器
ISPR ⁽¹⁾	中断挂起寄存器
ICPR ⁽¹⁾	中断清除寄存器
IABR ⁽¹⁾	中断活动状态寄存器
ITNS ⁽¹⁾	非安全状态中断寄存器
IPR ⁽¹⁾	中断优先级寄存器
STIR ⁽¹⁾	软触发中断寄存器

寄存器名称	寄存器描述
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
NSACR ⁽²⁾	非安全访问控制寄存器
SFSR ⁽²⁾	安全故障状态寄存器
SFAR ⁽²⁾	安全故障地址寄存器
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
BPIALL ⁽²⁾	分支预测器使所有无效

1. 参考 core_cm33.h 文件中定义的结构体类型 NVIC_Type
2. 参考 core_cm33.h 文件中定义的结构体类型 SCB_Type

表 3-743. 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_cm33.h 文件中定义的结构体类型 SysTick_Type

3.22.2. 外设库函数说明

MISC库函数列表如下表所示：

表 3-744. 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_system_reset	发起一个系统复位请求来复位MCU
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_enable	使能MPU
mpu_disable	失能MPU
mpu_region_struct_para_init	将MPU结构体中所有参数初始化为默认值
mpu_attribute_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_attribute_config	配置MPU属性
mpu_region_enable	使能MPU区域

结构体 mpu_region_init_struct

表 3-745. 结构体 mpu_region_init_struct

成员名称	功能描述
region_number	区域编号
region_base_address	区域基地址
region_limit_address	区域限定地址
access_permission	访问权限段
instruction_exec	永远执行
shareability	定义普通内存的可共享性
attribute_index	属性指数

结构体 mpu_attribute_init_struct

表 3-746. 结构体 mpu_attribute_init_struct

Member name	Function description
attribute_number	属性编号
memory_type	内存类型
outer_attributes	外部内存属性
inner_attributes	内部内存属性

枚举类型 IRQn_Type

表 3-747. 枚举类型 IRQn_Type

成员名称	功能描述
WWDGT_IRQn	窗口看门狗中断
LVD_VAVD_VOVD_VUVD_IRQn	连接到EXTI线16的LVD / VAVD / VOVD / VUVD中断
TAMPER_IRQn	连接到 EXTI 线 18 的 RTC 侵入和时间戳中断 LXTAL时钟阻塞中断
RTC_IRQn	连接到EXTI线19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_IDX_IRQn	TIMER0触发, 换相和索引中断
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_WKUP_IRQn	连接到EXTI线31的I2C0事件和唤醒中断
I2C0_ER_IRQn	I2C0错误中断

成员名称	功能描述
I2C1_EV_WKUP_IRQn	连接到EXTI线32的I2C1事件和唤醒中断
I2C1_ER_IRQn	I2C1错误中断
SPI0_IRQn	SPI0全局中断
SPI1_IRQn	SPI1全局中断
USART0_IRQn	连接到EXTI线28的USART0全局和唤醒中断
USART1_IRQn	连接到EXTI线29的USART1全局和唤醒中断
USART2_IRQn	连接到EXTI线30的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17的RTC闹钟中断
TIMER7_BRK_TRS_IDX_IRQHandler	TIMER7中止，传输和索引错误中断
TIMER7_UP_IRQn	TIMER7更新中断
TIMER7_TRG_CMT_IDX_IRQHandler	TIMER7触发，换相和索引中断
TIMER7_Channel_IRQn	TIMER7捕获比较中断
ADC2_IRQHandler	ADC2全局中断
SYSCFG_IRQHandler	SYSCFG全局中断
LPTIMER_IRQHandler	连接到EXTI线35的LPTIMER全局和唤醒中断
TIMER4_IRQHandler	TIMER4全局中断
SPI2_IRQHandler	SPI2全局中断
UART3_IRQHandler	UART3全局中断
UART4_IRQHandler	UART4全局中断
TIMER5_DAC0_2_UDR_IRQHandler	TIMER5 全局中断 DAC2，DAC0下溢错误中断
TIMER6_DAC1_3_UDR_IRQHandler	TIMER6 全局中断 DAC3，DAC1下溢错误中断
DMA1_Channel0_IRQHandler	DMA1通道0全局中断
DMA1_Channel1_IRQHandler	DMA1通道1全局中断
DMA1_Channel2_IRQHandler	DMA1通道2全局中断
DMA1_Channel3_IRQHandler	DMA1通道3全局中断
DMA1_Channel4_IRQHandler	DMA1通道4全局中断
ADC3_IRQHandler	ADC3全局中断
VUVD1_VOVD1_IRQHandler	VUVD1，VOVD1中断
CMP0_3_IRQHandler	连接到EXTI线20 / 21 / 22 / 23的CMP0 / CMP1 / CMP2 / CMP3中断
CMP4_7_IRQHandler	连接到EXTI线24 / 36 / 37 / 38的CMP4 / CMP5 / CMP6 / CMP7中断
CMP_IRQHandler	CMP全局中断
HRTIMER_IRQ0_IRQHandler	HRTIMER中断0
HRTIMER_IRQ1_IRQHandler	HRTIMER中断1
HRTIMER_IRQ2_IRQHandler	HRTIMER中断2
HRTIMER_IRQ3_IRQHandler	HRTIMER中断3
HRTIMER_IRQ4_IRQHandler	HRTIMER中断4

成员名称	功能描述
HRTIMER_IRQ5_IRQHandler	HRTIMER中断5
HRTIMER_IRQ6_IRQHandler	HRTIMER中断6
HRTIMER_IRQ7_IRQHandler	HRTIMER中断7
HRTIMER_IRQ8_IRQHandler	HRTIMER中断8
HRTIMER_IRQ9_IRQHandler	HRTIMER中断9
TIMER19_BRK_TRS_IDX_IRQHandler	TIMER19中止，传输和索引错误中断
TIMER19_UP_IRQHandler	TIMER19更新中断
TIMER19_TRG_CMT_IDX_IRQHandler	TIMER19触发，换相和索引中断
TIMER19_Channel_IRQHandler	TIMER19捕获比较中断
FPU_IRQHandler	FPU全局中断
I2C2_EV_WKUP_IRQHandler	连接到EXTI线33的I2C2事件和唤醒中断
I2C2_ER_IRQHandler	I2C2错误中断
CAU_IRQHandler	CAU全局中断
TRNG_IRQHandler	TRNG全局中断
I2C3_EV_WKUP_IRQHandler	连接到EXTI线34的I2C3事件和唤醒中断
I2C3_ER_IRQHandler	I2C3错误中断
DMAMUX_OVR_IRQHandler	DMAMUX上溢中断
QSPI_IRQHandler	QSPI全局中断
FFT_IRQHandler	FFT全局中断
DMA1_Channel5_IRQHandler	DMA1通道5全局中断
DMA1_Channel6_IRQHandler	DMA1通道6全局中断
CLA_IRQHandler	CLA全局中断
TMU_IRQHandler	TMU全局中断
FAC_IRQHandler	FAC全局中断
HPDF_INT0_IRQHandler	HPDF全局中断0
HPDF_INT1_IRQHandler	HPDF全局中断1
HPDF_INT2_IRQHandler	HPDF全局中断2
HPDF_INT3_IRQHandler	HPDF全局中断3
TIMER14_IRQHandler	TIMER14全局中断
TIMER15_IRQHandler	TIMER15全局中断
TIMER16_IRQHandler	TIMER16全局中断
CAN0_WKUP_IRQHandler	连接到EXTI线25的CAN唤醒中断
CAN0_Message_IRQHandler	CAN0消息缓冲区中断
CAN0_Busoff_IRQHandler	CAN0总线关闭/总线关闭完成中断
CAN0_Error_IRQHandler	CAN0错误中断
CAN0_FastError_IRQHandler	CAN0快速传输错误中断
CAN0_TEC_IRQHandler	CAN0发送警告中断
CAN0_REC_IRQHandler	CAN0接收警告中断
CAN1_WKUP_IRQHandler	连接到EXTI线的CAN1唤醒中断
CAN1_Message_IRQHandler	CAN1消息缓冲区中断
CAN1_Busoff_IRQHandler	CAN1总线关闭/总线关闭完成中断

成员名称	功能描述
CAN1_Error_IRQHandler	CAN1错误中断
CAN1_FastError_IRQHandler	CAN1快速传输错误中断
CAN1_TEC_IRQHandler	CAN1发送警告中断
CAN1_REC_IRQHandler	CAN1接收警告中断
CAN2_WKUP_IRQHandler	连接到EXTI线27的CAN2唤醒中断
CAN2_Message_IRQHandler	CAN2消息缓冲区中断
CAN2_Busoff_IRQHandler	CAN2总线关闭/总线关闭完成中断
CAN2_Error_IRQHandler	CAN2错误中断
CAN2_FastError_IRQHandler	CAN2快速传输错误中断
CAN2_TEC_IRQHandler	CAN2发送警告中断
CAN2_REC_IRQHandler	CAN2接收警告中断
TIMER0_DEC_IRQHandler	TIMER0 DEC中断
TIMER1_DEC_IRQHandler	TIMER1 DEC中断
TIMER2_DEC_IRQHandler	TIMER2 DEC中断
TIMER3_DEC_IRQHandler	TIMER3 DEC中断
TIMER4_DEC_IRQHandler	TIMER4 DEC中断
TIMER7_DEC_IRQHandler	TIMER7 DEC中断
TIMER19_DEC_IRQHandler	TIMER19 DEC中断

函数 nvic_priority_group_set

函数nvic_priority_group_set描述见下表：

表 3-748. 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 3-749. 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3-747. 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 3-750. 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3-747. 枚举类型IRQn_Type
输出参数{out}	
-	-
返回值	
-	-

例如:

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

```
nvic_irq_disable(WWDGT_IRQn);
```

函数 nvic_system_reset

函数nvic_system_reset描述见下表:

表 3-751. 函数 nvic_system_reset

函数名称	nvic_system_reset
函数原型	void nvic_system_reset(void);
功能描述	发起一个系统复位请求来复位MCU
先决条件	-
被调用函数	-
输入参数{in}	
-	-
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* initiates a system reset request to reset the MCU */
```

```
nvic_system_reset();
```

函数 nvic_vector_table_set

函数nvic_vector_table_set描述见下表:

表 3-752. 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 3-753. 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_WAKEUP_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 3-754. 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_WAKEUP_BY_ALL_INT	该位为0时，系统只能被使能的中断唤醒
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* the system will exit low power mode by exiting from ISR */
```

```
system_lowpower_reset(SCB_LPM_SLEEP_EXIT_ISR);
```

函数 systick_clksource_set

函数systick_clksource_set描述见下表：

表 3-755. 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_RCE_CKSYS	systick时钟源为CK_SYS时钟
SYSTICK_CLKSOURCE_RCE_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_enable

函数mpu_enable描述见下表:

表 3-756. 函数 mpu_enable

函数名称	mpu_enable
函数原型	void mpu_enable(uint32_t MPU_Control);
功能描述	使能MPU
先决条件	-
被调用函数	-
输入参数{in}	
MPU_Control	选择不同的MPU模式
MPU_MODE_HFNMI_ PRIVDEF_NONE	HFNMIENA和PRIVDEFENA值是0
MPU_MODE_HARDFA ULT_NMI	仅通过HardFault和NMI处理程序使用MPU进行内存访问
MPU_MODE_PRIV_DE FAULT	使默认内存映射为仅供特权访问的后台区域
MPU_MODE_HFNMI_ PRIVDEF	HFNMIENA和PRIVDEFENA值是1
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* enable the MPU */
```

```
mpu_enable(MPU_MODE_PRIV_DEFAULT);
```

函数 mpu_disable

函数mpu_disable描述见下表:

表 3-757. 函数 mpu_disable

函数名称	mpu_disable
函数原型	void mpu_disable(void);

功能描述	失能MPU
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 MPU */
mpu_disable();
```

函数 mpu_region_struct_para_init

函数mpu_region_struct_para_init描述见下表：

表 3-758. 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3-745. 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_attribute_struct_para_init

函数mpu_attribute_struct_para_init描述见下表：

表 3-759. 函数 mpu_attribute_struct_para_init

函数名称	mpu_attribute_struct_para_init
函数原型	void mpu_attribute_struct_para_init(mpu_attribute_init_struct

	<code>*attribute_init_struct);</code>
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}	
<code>mpu_attribute_init_struct</code>	MPU初始化结构体, 参考 表3-746. 结构体mpu_attribute_init_struct .
输出参数{out}	
-	-
返回值	
-	-

例如:

```
mpu_attribute_init_struct mpu_attribute_init_para;

/* initialize mpu_attribute_init_struct with the default values */

mpu_attribute_struct_para_init(&mpu_attribute_init_para);
```

函数 `mpu_region_config`

函数`mpu_region_config`描述见下表:

表 3-760. 函数 `mpu_region_config`

函数名称	<code>mpu_region_config</code>
函数原型	<code>void mpu_region_config(mpu_region_init_struct *mpu_init_struct);</code>
功能描述	配置MPU区域
先决条件	-
被调用函数	-
输入参数{in}	
<code>mpu_init_struct</code>	MPU初始化结构体, 参考 表3-745. 结构体mpu_region_init_struct .
输出参数{out}	
-	-
返回值	
-	-

例如:

```
mpu_region_init_struct region_init_struct;

mpu_region_struct_para_init(&region_init_struct);

region_init_struct.region_number = MPU_REGION_NUMBER0;

region_init_struct.region_base_address = 0x20002000U;

region_init_struct.region_limit_address = 0x20002000U;
```

```
region_init_struct.access_permission = MPU_REGION_PRIVILEGED_RO;  
  
region_init_struct.shareability = MPU_ACCESS_NOT_SHAREABLE;  
  
region_init_struct.instruction_exec = MPU_INSTRUCTION_EXEC_PERMIT;  
  
region_init_struct.attribute_index = MPU_ATTRIBUTE_NUMBER4;  
  
/* configure the MPU region */  
  
mpu_region_config(&region_init_struct);
```

函数 mpu_attribute_config

函数mpu_attribute_config描述见下表:

表 3-761. 函数 mpu_attribute_config

函数名称	mpu_attribute_config
函数原型	void mpu_attribute_config(mpu_attribute_init_struct *attribute_init_struct);
功能描述	配置MPU属性
先决条件	-
被调用函数	-
输入参数{in}	
mpu_attribute_init_struct	MPU初始化结构体, 参考 表3-746. 结构体mpu_attribute_init_struct .
输出参数{out}	
-	-
返回值	
-	-

例如:

```
mpu_attribute_init_struct attribute_init_struct;  
  
mpu_attribute_struct_para_init(&attribute_init_struct);  
  
attribute_init_struct.attribute_number = MPU_ATTRIBUTE_NUMBER4;  
  
attribute_init_struct.memory_type = MPU_MEMORY_NORMAL;  
  
attribute_init_struct.outer_attributes = MPU_NORMAL_OUTER_WT_TRAN_RW_ALLOC;  
  
attribute_init_struct.inner_attributes = MPU_NORMAL_INNER_WT_TRAN_RW_ALLOC;  
  
/* configure the MPU attribute */  
  
mpu_attribute_config(&attribute_init_struct);
```

函数 mpu_region_enable

函数mpu_region_enable描述见下表:

表 3-762. 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();
```

3.23. 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 [3.23.1](#) 描述了 PMU 的寄存器列表，章节 [3.23.2](#) 对 PMU 库函数进行说明。

3.23.1. 外设寄存器说明

PMU 寄存器列表如下表所示:

表 3-763. 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

3.23.2. 外设库函数说明

PMU 库函数列表如下表所示:

表 3-764. 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

库函数名称	库函数描述
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_ovd_select	选择LDO V _{1.1V} 供电电压
pmu_ovd_enable	打开V _{1.1V} 内核电压检测器
pmu_ovd_disable	关闭V _{1.1V} 内核电压检测器
pmu_uvd_select	选择低压检测器阈值电压
pmu_uvd_enable	使能低压阈值检测
pmu_uvd_disable	关闭低压阈值检测
pmu_ovd_filter	过压数字噪声滤波器
pmu_uvd_filter	低压数字噪声滤波器
pmu_deepsleep_voltage	深度睡眠模式电压选择
pmu_vbat_charging_select	VBAT电池充电电阻的选择
pmu_vbat_charging_enable	使能VBAT电池充电功能
pmu_vbat_charging_disable	关闭VBAT电池充电功能
pmu_vbat_temp_monitor_enable	启用VBAT和温度监控
pmu_vbat_temp_monitor_disable	关闭VBAT和温度监控
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_flag_get	获取状态标志位
pmu_flag_clear	清状态标志位

函数 pmu_deinit

函数pmu_deinit描述见下表：

表 3-765. 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 3-766. 函数 pmu_lvd_select

函数名称	pmu_lvd_select
函数原型	void pmu_lvd_select(uint32_t lvd_t_n);
功能描述	选择低压检测阈值
先决条件	-
被调用函数	-
输入参数{in}	
lvd_t_n	电压阈值
PMU_LVDT_0	voltage threshold is 2.15V
PMU_LVDT_1	voltage threshold is 2.3V
PMU_LVDT_2	voltage threshold is 2.45V
PMU_LVDT_3	voltage threshold is 2.6V
PMU_LVDT_4	voltage threshold is 2.75V
PMU_LVDT_5	voltage threshold is 2.9V
PMU_LVDT_6	voltage threshold is 3.0V
PMU_LVDT_7	input analog voltage on PA10 (compared with 1.2V)
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 3-767. 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 3-768. 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 3-769. 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_AVDT_0	选择模拟电压检测器阈值为1.8V
PMU_AVDT_1	选择模拟电压检测器阈值为2.2V
PMU_AVDT_2	选择模拟电压检测器阈值为2.6V
PMU_AVDT_3	选择模拟电压检测器阈值为2.9V
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* select analog voltage detector threshold 2.9V */
```

```
pmu_avd_select(PMU_VAVDVC_3);
```

函数 pmu_avd_enable

函数pmu_avd_enable描述见下表：

表 3-770. 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 3-771. 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_ovd_select

函数pmu_ovd_select描述见下表:

表 3-772. 函数 pmu_ovd_select

Function name	pmu_ovd_select
Function prototype	void pmu_ovd_select (uint32_t ovd_t_n);
Function descriptions	选择V _{CORE} 内核电压检测器
Precondition	-
The called functions	-
Input parameter{in}	
ovdt_n	电压阈值
PMU_OVDT_0	电压阈值为1.25V
PMU_OVDT_1	电压阈值为1.35V
PMU_OVDT_2	电压阈值为1.45V
PMU_OVDT_3	电压阈值为1.55V
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* 选择过压阈值为1.25V */
```

```
pmu_ovd_select (PMU_OVDT_0);
```

函数 pmu_ovd_enable

函数pmu_ovd_enable描述见下表:

表 3-773. 函数 pmu_ovd_enable

函数名称	pmu_ovd_enable
函数原型	void pmu_ovd_enable(void);
功能描述	使能V _{CORE} 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:

```
/* 使能VCORE电压检测 */
```

```
pmu_ovd_enable();
```

函数 pmu_ovd_disable

函数pmu_ovd_disable描述见下表:

表 3-774. 函数 pmu_ovd_disable

函数名称	pmu_ovd_disable
函数原型	void pmu_ovd_disable(void);
功能描述	关闭V _{CORE} 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:

```
/* 关闭VCORE电压检测 */
```

```
pmu_ovd_disable();
```

函数 pmu_uvd_select

函数pmu_uvd_select描述如下:

表 3-775. 函数 pmu_uvd_select

Function name	pmu_uvd_select
Function prototype	void pmu_uvd_select (uint32_t ldo_n);
Function descriptions	电压检测阈值选择
Precondition	-
The called functions	-
Input parameter{in}	
ldo_n	选择电压阈值
<i>PMU_UVDT_0</i>	电压阈值为0.65V
<i>PMU_UVDT_1</i>	电压阈值为0.75V
<i>PMU_UVDT_2</i>	电压阈值为0.85V
<i>PMU_UVDT_3</i>	电压阈值为0.95V
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* select LDO output voltage 0.95V */
pmu_uvd_select(PMU_LDOVS_3);
```

函数 pmu_uvd_enable

函数pmu_uvd_enable描述如下:

表 3-776. 函数 pmu_uvd_enable

Function name	pmu_uvd_enable
Function prototype	void pmu_uvd_enable (void);
Function descriptions	使能低压检测
Precondition	-
The called functions	-
Input parameter{in}	
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* 使能低压检测 */
```



```
pmu_uvd_enable();
```

函数 pmu_uvd_disable

函数pmu_uvd_disable描述如下:

表 3-777. 函数 pmu_uvd_disable

Function name	pmu_uvd_disable
Function prototype	void pmu_uvd_disable (void);
Function descriptions	关闭低压检测
Precondition	-
The called functions	-
Input parameter{in}	
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* 关闭低压检测 */
```

```
pmu_uvd_disable();
```

函数 pmu_ovd_filter

函数pmu_ovd_filter描述如下:

表 3-778. 函数 pmu_ovd_filter

Function name	pmu_ovd_filter
Function prototype	void pmu_ovd_filter (uint32_t dnf);
Function descriptions	过压数字噪声滤波器
Precondition	-
The called functions	-
Input parameter{in}	
dnf	滤波峰值长度
Uint32_t	0~255
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* 开启数字滤波器，滤波峰值长度高达255 * 1024 * TPCLK */
```

```
pmu_ovd_filter(255);
```

函数 pmu_uvd_filter

函数pmu_uvd_filter描述如下:

表 3-779. 函数 pmu_uvd_filter

Function name	pmu_uvd_filter
Function prototype	void pmu_uvd_filter (uint32_t dnf);
Function descriptions	低压数字噪声滤波器
Precondition	-
The called functions	-
Input parameter{in}	
dnf	digital noise filter
uint32_t	0~255
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* close under voltage digital noise filter */
```

```
pmu_uvd_filter(0);
```

函数 pmu_deepsleep_voltage

函数pmu_deepsleep_voltage描述如下:

表 3-780. 函数 pmu_deepsleep_voltage

Function name	pmu_deepsleep_voltage
Function prototype	void pmu_deepsleep_voltage (uint32_t vol);
Function descriptions	深度睡眠模式电压选择
Precondition	-
The called functions	-
Input parameter{in}	
lvdt_n	电压阈值
PMU_DSLPVS_0	电压阈值为0.8V
PMU_DSLPVS_1	电压阈值为0.9V
PMU_DSLPVS_2	电压阈值为1.0V (默认)
PMU_DSLPVS_3	电压阈值为1.1V
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* select deepsleep mode voltage 0.8V */
```

```
pmu_deepsleep_voltage (PMU_DSLPVS_0);
```

函数 pmu_vbat_charging_select

函数pmu_vbat_charging_select描述见下表:

表 3-781. 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 3-782. 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 3-783. 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 3-784. 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 3-785. 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_to_sleepmode

函数pmu_to_sleepmode描述见下表：

表 3-786. 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 3-787. 函数 pmu_to_deepsleepmode

函数名称	pmu_to_deepsleepmode
函数原型	void pmu_to_deepsleepmode(uint32_t lowpower, 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}	
lowpower	LDO低电压模式
PMU_LDO_NORMAL	LDO在深度睡眠模式下工作正常
PMU_LDO_LOWPOWER	LDO在深度睡眠模式下进入低功耗模式
输入参数{in}	
deepsleepmodecmd	进入深度睡眠模式命令
WFI_CMD	WFI命令
WFE_CMD	WFE命令
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* PMU work at deepsleep mode */
pmu_to_deepsleepmode(WFI_CMD);
```

函数 pmu_to_standbymode

函数pmu_to_standbymode描述见下表：

表 3-788. 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 3-789. 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（PC13）使能
PMU_WAKEUP_PIN2	唤醒引脚1（PE6）使能
PMU_WAKEUP_PIN3	唤醒引脚3（PA2）使能
PMU_WAKEUP_PIN4	唤醒引脚5（PC5）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 3-790. 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 3-791. 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 3-792. 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_flag_get

函数pmu_flag_get描述见下表:

表 3-793. 函数 pmu_flag_get

函数名称	pmu_flag_get
函数原型	FlagStatus pmu_flag_get(uint32_t flag);
功能描述	获取标志位
先决条件	-
被调用函数	-
输入参数{in}	
flag	标志位
PMU_FLAG_WAKEUP	唤醒标志
PMU_FLAG_STANDBY	待机标志
PMU_FLAG_LVDF	低电压标志
PMU_FLAG_VAVDF	V _{DDA} 模拟电压检测器标志
PMU_FLAG_VOVDFO	V _{DDA} 外设电压检测器标志
PMU_FLAG_VUVDFO	V _{BAT} 电压检测器低电压标志
PMU_FLAG_VOVDFO1	数字滤波后V _{1.1v} 过电压检测器标志
PMU_FLAG_VUVDFO1	数字滤波后V _{1.1v} 低电压检测器标志
PMU_FLAG_BKPVSRF	备份域电压稳压器就绪标志位
PMU_FLAG_VBATLF	V _{BAT} 监测低阈值标志位
PMU_FLAG_VBATHF	V _{BAT} 监测高阈值标志位
PMU_FLAG_TEMPLF	温度检测器低电压标志
PMU_FLAG_TEMPHEF	温度检测器高电压标志
输出参数{out}	

-	-
返回值	
FlagStatus	SET或RESET

例如:

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

```
FlagStatus status;
```

```
status = pmu_flag_get (PMU_FLAG_WAKEUP);
```

函数 pmu_flag_clear

函数pmu_flag_clear描述见下表:

表 3-794. 函数 pmu_flag_clear

函数名称	pmu_flag_clear
函数原型	void pmu_flag_clear(uint32_t flag_reset);
功能描述	清除标志位
先决条件	-
被调用函数	-
输入参数{in}	
flag_reset	标志位
PMU_FLAG_WAKEUP	清除唤醒标志
PMU_FLAG_STANDBY	清除待机标志
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* clear flag bit */
```

```
pmu_flag_clear (PMU_FLAG_WAKEUP);
```

3.24. QSPI

QSPI是一种专用于和Flash存储器通信的接口, 可以支持单线, 双线, 四线SPI FLASH。章节[3.24.1](#)描述了QSPI的寄存器列表, 章节[3.24.2](#)对QSPI库函数进行说明。

3.24.1. 外设寄存器说明

QSPI寄存器列表如下表所示:

表 3-795. QSPI 寄存器

寄存器名称	寄存器描述
QSPI_CTL	QSPI控制寄存器
QSPI_DCFG	QSPI设备配置寄存器
QSPI_STAT	QSPI状态寄存器
QSPI_STATC	QSPI状态清除寄存器
QSPI_DTLEN	QSPI数据长度寄存器
QSPI_TCFG	QSPI传输配置寄存器
QSPI_ADDR	QSPI地址寄存器
QSPI_ALTE	QSPI交替字节寄存器
QSPI_DATA	QSPI数据寄存器
QSPI_STATMK	QSPI状态屏蔽寄存器
QSPI_STATMATCH	QSPI状态匹配寄存器
QSPI_INTERVAL	QSPI间隔寄存器
QSPI_TMOUT	QSPI超时寄存器
QSPI_FLUSH	QSPI FIFO刷新寄存器

3.24.2. 外设库函数说明

QSPI库函数列表如下表所示：

表 3-796. QSPI 库函数

库函数名称	库函数描述
qspi_deinit	复位外设QSPI
qspi_struct_para_init	将QSPI初始化结构体中所有参数初始化为默认值
qspi_cmd_struct_para_init	将QSPI命令结构体中所有参数初始化为默认值
qspi_polling_struct_para_init	将QSPI读轮询结构体中所有参数初始化为默认值
qspi_init	初始化QSPI
qspi_enable	使能QSPI
qspi_disable	禁能QSPI
qspi_dma_enable	使能QSPI DMA
qspi_dma_disable	禁能QSPI DMA
qspi_command_config	配置QSPI命令参数
qspi_polling_config	配置QSPI读轮询模式
qspi_memorymapped_config	配置QSPI存储器映射模式
qspi_data_transmit	QSPI发送数据
qspi_data_receive	QSPI接收数据
qspi_transmission_abort	终止当前传输
qspi_output_clock_delay_enable	使能输出时钟延迟
qspi_output_clock_delay_disable	禁能输出时钟延迟
qspi_output_clock_delay_config	配置输出时钟延迟
qspi_sample_shift_config	配置QSPI采样延迟
qspi_receive_clock_sel	选择QSPI接收时钟

库函数名称	库函数描述
qspi_delay_scan_enable	使能延迟扫描链
qspi_delay_scan_disable	禁能延迟扫描链
qspi_csn_edge_cycle	选择CSN延迟1个或2个sck拉高或拉低
qspi_flag_get	获取QSPI标志状态
qspi_flag_clear	清除QSPI标志状态
qspi_interrupt_enable	使能QSPI中断
qspi_interrupt_disable	禁能QSPI中断
qspi_interrupt_flag_get	获取QSPI中断标志状态
qspi_interrupt_flag_clear	清除QSPI中断标志状态

结构体 qspi_init_struct

表 3-797. 结构体 qspi_init_struct

成员名称	功能描述
prescaler	QSPI分频
fifo_threshold	QSPI FIFO阈值
sample_shift1	QSPI采样移位参数1
sample_shift2	QSPI采样移位参数2
flash_size	外部flash大小
cs_high_time	两个命令序列之间cs信号保持高电平的周期数
clock_mode	QSPI时钟模式

结构体 qspi_command_struct

表 3-798. 结构体 qspi_command_struct

成员名称	功能描述
instruction_mode	命令模式
instruction	8位命令
addr_mode	地址模式
addr_size	地址大小
addr	外部存储地址
altebytes_mode	交替字节模式
altebytes_size	交替字节大小
altebytes	交替字节信息
dummycycles	空闲周期
data_mode	数据模式
data_length	32位数据长度
sioo_mode	只发送一次指令模式
trans_rate	传输模式为SDR或DDR
trans_delay	传输延迟为0或1/4个时钟周期

结构体 `qspi_polling_struct`表 3-799. 结构体 `qspi_polling_struct`

成员名称	功能描述
match	匹配值
mask	屏蔽值
interval	两次状态轮询之间的时钟周期数
statusbytes_size	接收状态字节大小
match_mode	匹配方法
polling_stop	匹配后是否自动终止状态轮询

函数 `qspi_deinit`

函数 `qspi_deinit` 描述见下表：

表 3-800. 函数 `qspi_deinit`

函数名称	<code>qspi_deinit</code>
函数原形	<code>void qspi_deinit(void);</code>
功能描述	复位外设 QSPI
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 QSPI */
```

```
qspi_deinit();
```

函数 `qspi_struct_para_init`

函数 `qspi_struct_para_init` 描述见下表：

表 3-801. 函数 `qspi_struct_para_init`

函数名称	<code>qspi_struct_para_init</code>
函数原形	<code>void qspi_struct_para_init(qspi_init_struct *init_para);</code>
功能描述	将 QSPI 初始化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_para	QSPI初始化参数结构体，结构体成员可参考 表3-797. 结构体 qspi_init_struct 。
返回值	
-	-

例如：

```
/* initialize the parameters of QSPI */
```

```
qspi_init_struct qspi_init_para;
```

```
qspi_struct_para_init(&qspi_init_para);
```

函数 qspi_cmd_struct_para_init

函数qspi_cmd_struct_para_init描述见下表：

表 3-802. 函数 qspi_cmd_struct_para_init

函数名称	qspi_cmd_struct_para_init
函数原形	void qspi_cmd_struct_para_init(qspi_command_struct *init_para);
功能描述	将QSPI命令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_para	QSPI命令结构体，结构体成员可参考 表3-798. 结构体 qspi_command_struct 。
返回值	
-	-

例如：

```
/* initialize the parameters of QSPI command structrue */
```

```
qspi_command_struct qspi_cmd_para;
```

```
qspi_cmd_struct_para_init(&qspi_cmd_para);
```

函数 qspi_polling_struct_para_init

函数qspi_polling_struct_para_init描述见下表：

表 3-803. 函数 qspi_polling_struct_para_init

函数名称	qspi_polling_struct_para_init
函数原形	void qspi_polling_struct_para_init(qspi_polling_struct *init_para);
功能描述	将QSPI读轮询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_para	QSPI读轮询结构体，结构体成员可参考 表3-799. 结构体 <i>qspi_polling_struct</i>
返回值	
-	-

例如：

```
/* initialize the parameters of QSPI read polling structrue */
```

```
qspi_polling_struct qspi_polling_para;
```

```
qspi_polling_struct_para_init(&qspi_polling_para);
```

函数 qspi_init

函数qspi_init描述见下表：

表 3-804. 函数 qspi_init

函数名称	qspi_init
函数原形	void qspi_init(qspi_init_struct* qspi_struct);
功能描述	初始化QSPI
先决条件	-
被调用函数	-
输入参数{in}	
qspi_struct	QSPI初始化参数结构体，结构体成员可参考 表3-797. 结构体 <i>qspi_init_struct</i>
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* initialize the init parameter structure */
```

```
qspi_struct_para_init(&qspi_init_para);
```

```
qspi_init_para.clock_mode = QSPI_CLOCK_MODE_0;
```

```
qspi_init_para.fifo_threshold = 4;
```

```
qspi_init_para.sample_shift = QSPI_SAMPLE_SHIFTING_HALFCYCLE;
```

```
qspi_init_para.cs_high_time = QSPI_CS_HIGH_TIME_2_CYCLE;
```

```
qspi_init_para.flash_size = 27;
```

```
qspi_init_para.prescaler = 4;
```

```
qspi_init(&qspi_init_para);
```

函数 qspi_enable

函数qspi_enable描述见下表：

表 3-805. 函数 qspi_enable

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

```
qspi_enable();
```

函数 qspi_disable

函数qspi_disable描述见下表：

表 3-806. 函数 qspi_disable

函数名称	qspi_disable
函数原形	void qspi_disable(void);
功能描述	禁能QSPI
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 QSPI */
```

```
qspi_disable();
```

函数 qspi_dma_enable

函数qspi_dma_enable描述见下表：

表 3-807. 函数 qspi_dma_enable

函数名称	qspi_dma_enable
函数原形	void qspi_dma_enable(void);
功能描述	使能QSPI 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 QSPI DMA */
```

```
qspi_dma_enable();
```

函数 qspi_dma_disable

函数qspi_dma_disable描述见下表：

表 3-808. 函数 qspi_dma_disable

函数名称	qspi_dma_disable
函数原形	void qspi_dma_disable(void);
功能描述	禁能QSPI 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 QSPI DMA */
```

```
qspi_dma_disable();
```

函数 qspi_command_config

函数qspi_command_config描述见下表:

表 3-809. 函数 qspi_command_config

函数名称	qspi_command_config
函数原形	void qspi_command_config(qspi_command_struct *cmd);
功能描述	配置QSPI命令参数
先决条件	-
被调用函数	-
输入参数{in}	
cmd	QSPI命令参数结构体，结构体成员可参考 表3-798. 结构体 qspi_command_struct 。
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* initialize the QSPI command parameter structure */
qspi_cmd_struct_para_init(&qspi_cmd);

/* write enable */
qspi_cmd.instruction_mode = QSPI_INSTRUCTION_1_LINE;
qspi_cmd.instruction = WRITE_ENABLE_CMD;
qspi_cmd.addr_mode = QSPI_ADDR_NONE;
qspi_cmd.altebytes_mode = QSPI_ALTE_BYTES_NONE;
qspi_cmd.data_mode = QSPI_DATA_NONE;
qspi_cmd.dummyscycles = 0;
qspi_cmd.sioo_mode = QSPI_SIOO_INST_EVERY_CMD;
qspi_command_config(&qspi_cmd);
```

函数 qspi_polling_config

函数qspi_polling_config描述见下表:

表 3-810. 函数 qspi_polling_config

函数名称	qspi_polling_config
函数原形	void qspi_polling_config(qspi_command_struct *cmd, qspi_polling_struct

	*cfg);
功能描述	配置QSPI读轮询模式
先决条件	-
被调用函数	-
输入参数{in}	
cmd	QSPI命令参数结构体，结构体成员可参考 表3-798. 结构体 qspi_command_struct 。
输入参数{in}	
cfg	QSPI读轮询参数结构体，结构体成员可参考 表3-799. 结构体 qspi_polling_struct 。
输出参数{out}	
-	-
返回值	
-	-

例如：

```

/* initialize the QSPI command parameter structure */
qspi_cmd_struct_para_init(&qspi_cmd);

/* initialize the QSPI read polling parameter structure */
qspi_polling_struct_para_init(&polling_cmd);

/* read enable */
polling_cmd.match = 0x02;
polling_cmd.mask = 0x02;
polling_cmd.match_mode = QSPI_MATCH_MODE_AND;
polling_cmd.statusbytes_size = 1;
polling_cmd.interval = 0x10;
polling_cmd.polling_stop = QSPI_POLLING_STOP_ENABLE;
qspi_cmd.instruction = READ_STATUS_REG1_CMD;
qspi_cmd.data_mode = QSPI_DATA_1_LINE;
qspi_polling_config(&qspi_cmd, &polling_cmd);

```

函数 qspi_memorymapped_config

函数qspi_memorymapped_config描述见下表：

表 3-811. 函数 qspi_memorymapped_config

函数名称	qspi_memorymapped_config
------	--------------------------

函数原形	void qspi_memorymapped_config(qspi_command_struct *cmd, uint16_t timeout, uint32_t toen);
功能描述	配置QSPI存储器映射模式
先决条件	-
被调用函数	-
输入参数{in}	
cmd	QSPI命令参数结构体，结构体成员可参考 表3-798. 结构体 qspi_command_struct 。
输入参数{in}	
timeout	0-0xFFFF
输入参数{in}	
toen	超时计数
QSPI_TMOUT_DISABLE	禁能超时计数
QSPI_TMOUT_ENABLE	使能超时计数
输出参数{out}	
-	-
返回值	
-	-

例如：

```

/* initialize the QSPI command parameter structure */

qspi_cmd_struct_para_init(&qspi_cmd);

qspi_cmd.instruction_mode = QSPI_INSTRUCTION_1_LINE;

qspi_cmd.instruction = RDID;

qspi_cmd.addr_mode = QSPI_ADDR_NONE;

qspi_cmd.addr_size = QSPI_ADDR_8_BITS;

qspi_cmd.addr = 0;

qspi_cmd.altebytes_mode = QSPI_ALTE_BYTES_1_LINE;

qspi_cmd.altebytes_size = QSPI_ALTE_BYTES_24_BITS;

qspi_cmd.altebytes = 0;

qspi_cmd.dummycycles = 0;

qspi_cmd.data_mode = QSPI_DATA_1_LINE;

qspi_cmd.data_length = 3;

qspi_cmd.sioo_mode = QSPI_SIOO_INST_EVERY_CMD;

```

```
qspi_memorymapped_config(&qspi_cmd, 0xf, QSPI_TMOOUT_ENABLE);
```

函数 qspi_data_transmit

函数qspi_data_transmit描述见下表:

表 3-812. 函数 qspi_data_transmit

函数名称	qspi_data_transmit
函数原形	void qspi_data_transmit(uint8_t *tdata);
功能描述	QSPI发送数据
先决条件	-
被调用函数	-
输入参数{in}	
tdata	指向待发送数据的指针
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* QSPI transmit data */  
  
uint8_t data[32];  
  
qspi_data_transmit(&data);
```

函数 qspi_data_receive

函数qspi_data_receive描述见下表:

表 3-813. 函数 qspi_data_receive

函数名称	qspi_data_receive
函数原形	void qspi_data_receive(uint8_t *rdata);
功能描述	QSPI接收数据
先决条件	-
被调用函数	-
输入参数{in}	
rdata	指向待接收数据的指针
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* QSPI receive data */
```

```
uint8_t data[32];  
  
qspi_data_receive(&data);
```

函数 qspi_transmission_abort

函数qspi_transmission_abort描述见下表：

表 3-814. 函数 qspi_transmission_abort

函数名称	qspi_transmission_abort
函数原形	void qspi_transmission_abort(void);
功能描述	终止当前传输
先决条件	-
被调用函数	-
输入参数{in}	
-	-
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* abort QSPI transmission */  
  
qspi_transmission_abort();
```

函数 qspi_output_clock_delay_enable

函数qspi_output_clock_delay_enable描述见下表：

表 3-815. 函数 qspi_output_clock_delay_enable

函数名称	qspi_output_clock_delay_enable
函数原形	qspi_output_clock_delay_enable(void);
功能描述	使能输出时钟延迟
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 output clock delay */
```

```
qspi_output_clock_delay_enable();
```

函数 qspi_output_clock_delay_disable

函数qspi_output_clock_delay_disable描述见下表：

表 3-816. 函数 qspi_output_clock_delay_disable

函数名称	qspi_output_clock_delay_disable
函数原形	qspi_output_clock_delay_disable(void);
功能描述	禁能输出时钟延迟
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 output clock delay */  
  
qspi_output_clock_delay_disable();
```

函数 qspi_output_clock_delay_config

函数qspi_output_clock_delay_config描述见下表：

表 3-817. 函数 qspi_output_clock_delay_config

函数名称	qspi_output_clock_delay_config
函数原形	void qspi_output_clock_delay_config(uint32_t ck_delay);
功能描述	配置输出时钟延迟
先决条件	-
被调用函数	-
输入参数{in}	
ck_delay	时钟延迟值，0~15
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* configure output clock delay */  
  
qspi_output_clock_delay_config(1);
```

函数 `qspi_sample_shift_config`

函数 `qspi_sample_shift_config` 描述见下表:

表 3-818. 函数 `qspi_sample_shift_config`

函数名称	<code>qspi_sample_shift_config</code>
函数原形	<code>void qspi_sample_shift_config(uint32_t sample_shift1, uint32_t sample_shift2);</code>
功能描述	配置QSPI采样延迟
先决条件	-
被调用函数	-
输入参数{in}	
<code>sample_shift1</code>	QSPI采样延迟参数1
<code>QSPI_SAMPLE_SHIFTING_NONE</code>	不延迟
<code>QSPI_SAMPLE_SHIFTING_HALFCYCLE</code>	1/2 sck延迟
输入参数{in}	
<code>sample_shift2</code>	QSPI采样延迟参数2
<code>QSPI_SHIFTING_NONE</code>	不延迟
<code>QSPI_SHIFTING_1_CYCLE</code>	1 sck延迟
<code>QSPI_SHIFTING_2_CYCLE</code>	2 sck延迟
<code>QSPI_SHIFTING_3_CYCLE</code>	3 sck延迟
<code>QSPI_SHIFTING_4_CYCLE</code>	4 sck延迟
<code>QSPI_SHIFTING_5_CYCLE</code>	5 sck延迟
<code>QSPI_SHIFTING_6_CYCLE</code>	6 sck延迟
<code>QSPI_SHIFTING_7_CYCLE</code>	7 sck延迟
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* configure QSPI sample shift */
```

```
qspi_sample_shift_config(QSPI_SAMPLE_SHIFTING_NONE, QSPI_SHIFTING_1_CYCLE);
```


函数 qspi_receive_clock_sel

函数qspi_receive_clock_sel描述见下表:

表 3-819. 函数 qspi_receive_clock_sel

函数名称	qspi_receive_clock_sel
函数原形	void qspi_receive_clock_sel(uint32_t rcksel);
功能描述	选择QSPI接收时钟
先决条件	-
被调用函数	-
输入参数{in}	
rcksel	接收时钟选择
QSPI_RECEIVE_CLOCK_SCK	QSPI接收时钟选择SCK
QSPI_RECEIVE_CLOCK_DQS	QSPI接收时钟选择DQS
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* select QSPI receive clock */
```

```
qspi_receive_clock_sel(QSPI_RECEIVE_CLOCK_DQS);
```

函数 qspi_delay_scan_enable

函数qspi_delay_scan_enable描述见下表:

表 3-820. 函数 qspi_delay_scan_enable

函数名称	qspi_delay_scan_enable
函数原形	void qspi_delay_scan_enable(void);
功能描述	使能延迟扫描链
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 delay scan */
```

```
void qspi_delay_scan_enable();
```

函数 qspi_delay_scan_disable

函数qspi_delay_scan_disable描述见下表：

表 3-821. 函数 qspi_delay_scan_disable

函数名称	qspi_delay_scan_disable
函数原形	void qspi_delay_scan_disable(void);
功能描述	禁能延迟扫描链
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 delay scan */
```

```
void qspi_delay_scan_disable();
```

函数 qspi_csn_edge_cycle

函数qspi_csn_edge_cycle描述见下表：

表 3-822. 函数 qspi_csn_edge_cycle

函数名称	qspi_csn_edge_cycle
函数原形	void qspi_csn_edge_cycle(uint32_t csn_cycle);
功能描述	选择CSN延迟1个或2个sck拉高或拉低
先决条件	-
被调用函数	-
输入参数{in}	
csn_cycle	Csn延迟时钟
QSPI_CSN_1_CYCLE	CSN延迟1个sck拉高或拉低
QSPI_CSN_2_CYCLE	CSN延迟2个sck拉高或拉低
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* select csn falls and rises 2 sck cycles */
```

qspi_csn_edge_cycle(QSPI_CSN_2_CYCLE);

函数 qspi_flag_get

函数qspi_flag_get描述见下表:

表 3-823. 函数 qspi_flag_get

函数名称	qspi_flag_get
函数原形	FlagStatus qspi_flag_get(uint32_t flag);
功能描述	获取QSPI标志状态
先决条件	-
被调用函数	-
输入参数{in}	
flag	QSPI标志
QSPI_FLAG_BUSY	忙标志
QSPI_FLAG_TERR	传输错误标志
QSPI_FLAG_TC	传输完成标志
QSPI_FLAG_FT	FIFO阈值标志
QSPI_FLAG_RPMF	读轮询匹配标志
QSPI_FLAG_TMOUT	超时标志
输出参数{out}	
-	-
返回值	
FlagStatus	SET或RESET

例如:

```
/* get QSPI transfer complete flag */
FlagStatus status;

status = qspi_flag_get(QSPI_FLAG_TC);
```

函数 qspi_flag_clear

函数qspi_flag_clear描述见下表:

表 3-824. 函数 qspi_flag_clear

函数名称	qspi_flag_clear
函数原形	void qspi_flag_clear(uint32_t flag);
功能描述	清除QSPI标志状态
先决条件	-
被调用函数	-
输入参数{in}	
flag	QSPI标志
QSPI_FLAG_TERR	传输错误标志

QSPI_FLAG_TC	传输完成标志
QSPI_FLAG_RPMF	读轮询匹配标志
QSPI_FLAG_TMOUT	超时标志
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* clear QSPI transfer complete flag status */
```

```
qspi_flag_clear(QSPI_FLAG_TC);
```

函数 qspi_interrupt_enable

函数qspi_interrupt_enable描述见下表：

表 3-825. 函数 qspi_interrupt_enable

函数名称	qspi_interrupt_enable
函数原形	void qspi_interrupt_enable(uint32_t interrupt);
功能描述	使能QSPI中断
先决条件	-
被调用函数	-
输入参数{in}	
interrupt	QSPI中断
QSPI_INT_TC	传输完成中断
QSPI_INT_FT	FIFO阈值中断
QSPI_INT_TERR	传输错误中断
QSPI_INT_RPMF	读轮询匹配中断
QSPI_INT_TMOUT	超时中断
输出参数{out}	
-	-
返回值	

例如：

```
/* enable QSPI transfer complete interrupt */
```

```
qspi_interrupt_enable(QSPI_INT_TC);
```

函数 qspi_interrupt_disable

函数qspi_interrupt_disable描述见下表：

表 3-826. 函数 `qspi_interrupt_disable`

函数名称	<code>qspi_interrupt_disable</code>
函数原形	<code>void qspi_interrupt_disable(uint8_t interrupt);</code>
功能描述	禁能QSPI中断
先决条件	-
被调用函数	-
输入参数{in}	
interrupt	QSPI中断
<code>QSPI_INT_TC</code>	传输完成中断
<code>QSPI_INT_FT</code>	FIFO阈值中断
<code>QSPI_INT_TERR</code>	传输错误中断
<code>QSPI_INT_RPMF</code>	读轮询匹配中断
<code>QSPI_INT_TMOUT</code>	超时中断
输出参数{out}	
-	-
返回值	

例如：

```
/* disable QSPI transfer complete interrupt */
```

```
qspi_interrupt_disable(QSPI_INT_TC);
```

函数 `qspi_interrupt_flag_get`

函数`qspi_interrupt_flag_get`描述见下表：

表 3-827. 函数 `qspi_interrupt_flag_get`

函数名称	<code>qspi_interrupt_flag_get</code>
函数原形	<code>FlagStatus qspi_interrupt_flag_get(uint32_t int_flag);</code>
功能描述	获取QSPI中断标志状态
先决条件	-
被调用函数	-
输入参数{in}	
int_flag	QSPI中断标志
<code>QSPI_INT_FLAG_TERR</code>	传输错误中断标志
<code>QSPI_INT_FLAG_TC</code>	传输完成中断标志
<code>QSPI_INT_FLAG_FT</code>	FIFO阈值中断标志
<code>QSPI_INT_FLAG_RPMF</code>	读轮询匹配中断标志
<code>QSPI_INT_FLAG_TMO</code>	超时中断标志
输出参数{out}	

-	-
返回值	
FlagStatus	SET或RESET

例如：

```
/* get QSPI transfer complete flag */
```

```
FlagStatus status;
```

```
status = qspi_flag_get(QSPI_FLAG_TC);
```

函数 qspi_interrupt_flag_clear

函数qspi_interrupt_flag_clear描述见下表：

表 3-828. 函数 qspi_interrupt_flag_clear

函数名称	qspi_interrupt_flag_clear
函数原形	void qspi_interrupt_flag_clear(uint32_t int_flag);
功能描述	清除QSPI中断标志状态
先决条件	-
被调用函数	-
输入参数{in}	
int_flag	QSPI中断标志
QSPI_INT_FLAG_TERR	传输错误中断标志
QSPI_INT_FLAG_TCF	传输完成中断标志
QSPI_INT_FLAG_RPMF	读轮询匹配中断标志
QSPI_INT_FLAG_TMOUF	超时中断标志
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* clear QSPI transfer complete flag status */
```

```
qspi_flag_clear(QSPI_FLAG_TC);
```

3.25. 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 [3.25.1](#) 描述了 RCU 的寄存器列表，章节 [3.25.2](#) 对 RCU 库函数进行说明。

3.25.1. 外设寄存器说明

RCU寄存器列表如下表所示：

表 3-829. RCU 寄存器

寄存器名称	寄存器描述
RCU_CTL	控制寄存器
RCU_PLL	PLL 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_APB1RST	APB1复位寄存器
RCU_APB2RST	APB2复位寄存器
RCU_APB3RST	APB3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_APB1EN	APB1使能寄存器
RCU_APB2EN	APB2使能寄存器
RCU_APB3EN	APB3使能寄存器
RCU_AHB1SPDPE N	AHB1睡眠和深度睡眠模式使能寄存器
RCU_AHB2SPDPE N	AHB2睡眠和深度睡眠模式使能寄存器
RCU_AHB3SPDPE N	AHB3睡眠和深度睡眠模式使能寄存器
RCU_APB1SPDPE N	APB1睡眠和深度睡眠模式使能寄存器
RCU_APB2 SPDPEN	APB2睡眠和深度睡眠模式使能寄存器
RCU_APB3 SPDPEN	APB3睡眠和深度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_CFG1	配置寄存器1
RCU_CFG2	配置寄存器2

3.25.2. 外设库函数说明

RCU库函数列表如下表所示：

表 3-830. 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_ckout_config	配置CKOUT时钟源选择及分频系数
rcu_lsckout_enable	低速时钟输出使能
rcu_lsckout_disable	低速时钟输出失能
rcu_lsckout_config	配置LSCKOUT是时钟源输出
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_rtc_clock_config	配置RTC时钟
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_hpdc_clock_config	配置HPDF时钟源时钟
rcu_hpdcfaudio_clock_config	配置HPDF AUDIO时钟源时钟
rcu_trng_clock_config	配置TRNG时钟源时钟
rcu_lptimer_clock_config	配置LPTIMER时钟源时钟
rcu_qspi_clock_config	配置QSPI时钟源时钟
rcu_hrtimer_clock_config	配置HRTIMER时钟源时钟
rcu_lxtal_drive_capability_config	配置LXTAL的驱动力
rcu_osc_i_stab_wait	等待振荡器稳定标志位置位或振荡器起振超时
rcu_osc_i_on	打开振荡器
rcu_osc_i_off	关闭振荡器
rcu_osc_bypass_mode_enable	使能时钟旁路模式
rcu_osc_bypass_mode_disable	禁能时钟旁路模式
rcu_irc8m_adjust_value_set	设置内部8MHz RC振荡器时钟调整值
rcu_hxtal_clock_monitor_enable	HXTAL时钟监视器使能

库函数名称	库函数描述
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 3-831. 枚举类型 rcu_periph_enum

成员名称	功能描述
RCU_CRC	CRC时钟
RCU_CLA	CLA时钟
RCU_DMA0	DMA0 RX时钟
RCU_DMA1	DMA1时钟
RCU_DMAMUX	DMAMUX时钟
RCU_FFT	FFT时钟
RCU_FAC	FAC时钟
RCU_CAU	CAU时钟
RCU_TRNG	TRNG时钟
RCU_TMU	ENET0时钟
RCU_GPIOA	GPIOA时钟
RCU_GPIOB	GPIOB时钟
RCU_GPIOC	GPIOC时钟
RCU_GPIOD	GPIOD时钟
RCU_GPIOE	GPIOE时钟
RCU_GPIOF	GPIOF时钟
RCU_GPIOG	GPIOG时钟
RCU_EXMC	EXMC时钟
RCU_QSPI	QSPI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_LPTIMER	LPTIMER时钟

成员名称	功能描述
RCU_WWDGT	WWDGT时钟
RCU_SPI1	SPI1时钟
RCU_SPI2	SPI2时钟
RCU_USART1	USART1时钟
RCU_USART2	USART2时钟
RCU_UART3	UART3时钟
RCU_UART4	UART4时钟
RCU_I2C0	I2C0时钟
RCU_I2C1	I2C1时钟
RCU_I2C2	I2C2时钟
RCU_I2C3	I2C3时钟
RCU_PMU	PMU时钟
RCU_TIMER0	TIMER0时钟
RCU_TIMER7	TIMER7时钟
RCU_VREF	VREF时钟
RCU_CMP	CMP时钟
RCU_USART0	USART0时钟
RCU_CAN0	CAN0时钟
RCU_CAN1	CAN1时钟
RCU_CAN2	CAN2时钟
RCU_SPI0	SPI0时钟
RCU_SYSCFG	SYSCFG时钟
RCU_TIMER19	TIMER19时钟
RCU_TIMER14	TIMER14时钟
RCU_TIMER15	TIMER15时钟
RCU_TIMER16	TIMER16时钟
RCU_HPDF	HPDF时钟
RCU_HRTIMER	HRTIMER时钟
RCU_TRIGSEL	TRIGSEL时钟
RCU_ADC0	ADC0时钟
RCU_ADC1	ADC1时钟
RCU_ADC2	ADC2时钟
RCU_ADC3	ADC3时钟
RCU_DACHOLD	DACHOLD时钟
RCU_DAC0	DAC0时钟
RCU_DAC1	DAC1时钟
RCU_DAC2	DAC2时钟
RCU_DAC3	DAC3时钟
RCU_RTC	RTC时钟

枚举类型 rcu_periph_sleep_enum

表 3-832. 枚举类型 rcu_periph_sleep_enum

成员名称	功能描述
RCU_FMC_SLP	FMC时钟
RCU_CRC_SLP	CRC时钟
RCU_CLA_SLP	CLA时钟
RCU_SRAM0_SLP	SRAM0时钟
RCU_SRAM1_SLP	SRAM1时钟
RCU_TCM_SRAM_SLP	TCMSRAM I时钟
RCU_DMA0_SLP	DMA0时钟
RCU_DMA1_SLP	DMA1时钟
RCU_DMAMUX_SLP	DMAMUX时钟
RCU_FFT_SLP	FFT时钟
RCU_FAC_SLP	FAC时钟
RCU_CAU_SLP	CAU时钟
RCU_TRNG_SLP	TRNG时钟
RCU_TMU_SLP	TMU时钟
RCU_GPIOA_SLP	GPIOA时钟
RCU_GPIOB_SLP	GPIOB时钟
RCU_GPIOC_SLP	GPIOC时钟
RCU_GPIOD_SLP	GPIOD时钟
RCU_GPIOE_SLP	GPIOE时钟
RCU_GPIOF_SLP	GPIOF时钟
RCU_GPIOG_SLP	GPIOG时钟
RCU_EXMC_SLP	EXMC时钟
RCU_QSPI_SLP	QSPI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_LPTIMER_SLP	LPTIMER时钟
RCU_WWDGT_SLP	WWDGT时钟
RCU_SPI1_SLP	SPI1时钟
RCU_SPI2_SLP	SPI2时钟
RCU_USART1_SLP	USART1时钟
RCU_USART2_SLP	USART2时钟
RCU_UART3_SLP	UART3时钟
RCU_UART4_SLP	UART4时钟
RCU_I2C0_SLP	I2C0时钟

成员名称	功能描述
RCU_I2C1_SLP	I2C1时钟
RCU_I2C2_SLP	I2C2时钟
RCU_I2C3_SLP	I2C3时钟
RCU_PMU_SLP	PMU时钟
RCU_TIMER0_SLP	TIMER0时钟
RCU_TIMER7_SLP	TIMER7时钟
RCU_VREF_SLP	VREF时钟
RCU_CMP_SLP	CMP时钟
RCU_USART0_SLP	USART0时钟
RCU_CAN0_SLP	CAN0时钟
RCU_CAN1_SLP	CAN1时钟
RCU_CAN2_SLP	CAN2时钟
RCU_SPI0_SLP	SPI0时钟
RCU_SYSCFG_SLP	SYSCFG时钟
RCU_TIMER19_SLP	TIMER19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_HRTIMER_SLP	HRTIMER时钟
RCU_TRIGSEL_SLP	TRIGSEL时钟
RCU_ADC0_SLP	ADC0时钟
RCU_ADC1_SLP	ADC1时钟
RCU_ADC2_SLP	ADC2时钟
RCU_ADC3_SLP	ADC3时钟
RCU_DACHOLD_SLP	DACHOLD时钟
RCU_DAC0_SLP	DAC0时钟
RCU_DAC1_SLP	DAC1时钟
RCU_DAC2_SLP	DAC2时钟
RCU_DAC3_SLP	DAC3时钟

枚举类型 `rcu_periph_reset_enum`

表 3-833. 枚举类型 `rcu_periph_reset_enum`

成员名称	功能描述
RCU_CRCRST	复位CRC
RCU_CLARST	复位CLA
RCU_DMA0RST	复位DMA0 RX
RCU_DMA1RST	复位DMA1
RCU_DMAMUXRST	复位DMAMUX
RCU_FFTRST	复位FFT

成员名称	功能描述
RCU_FACRST	复位FAC
RCU_CAURST	复位CAU
RCU_TRNGRST	复位TRNG
RCU_TMURST	复位ENET0
RCU_GPIOARST	复位GPIOA
RCU_GPIOBRST	复位GPIOB
RCU_GPIOCRST	复位GPIOC
RCU_GPIODRST	复位GPIOD
RCU_GPIOERST	复位GPIOE
RCU_GPIOFRST	复位GPIOF
RCU_GPIOGRST	复位GPIOG
RCU_EXMCRST	复位EXMC
RCU_QSPIRST	复位QSPI
RCU_TIMER1RST	复位TIMER1
RCU_TIMER2RST	复位TIMER2
RCU_TIMER3RST	复位TIMER3
RCU_TIMER4RST	复位TIMER4
RCU_TIMER5RST	复位TIMER5
RCU_TIMER6RST	复位TIMER6
RCU_LPTIMERRST	复位LPTIMER
RCU_WWDGTRST	复位WWDGT
RCU_SPI1RST	复位SPI1
RCU_SPI2RST	复位SPI2
RCU_USART1RST	复位USART1
RCU_USART2RST	复位USART2
RCU_UART3RST	复位UART3
RCU_UART4RST	复位UART4
RCU_I2C0RST	复位I2C0
RCU_I2C1RST	复位I2C1
RCU_I2C2RST	复位I2C2
RCU_I2C3RST	复位I2C3
RCU_PMURST	复位PMU
RCU_TIMER0RST	复位TIMER0
RCU_TIMER7RST	复位TIMER7
RCU_VREFRST	复位VREF
RCU_CMPRST	复位CMP
RCU_USART0RST	复位USART0
RCU_CAN0RST	复位CAN0
RCU_CAN1RST	复位CAN1
RCU_CAN2RST	复位CAN2
RCU_SPI0RST	复位SPI0

成员名称	功能描述
RCU_SYSCFG_RST	复位SYSCFG
RCU_TIMER19_RST	复位TIMER19
RCU_TIMER14_RST	复位TIMER14
RCU_TIMER15_RST	复位TIMER15
RCU_TIMER16_RST	复位TIMER16
RCU_HPDRF_RST	复位HPDRF
RCU_HRTIMER_RST	复位HRTIMER
RCU_TRIGSEL_RST	复位TRIGSEL
RCU_ADC0_RST	复位ADC0
RCU_ADC1_RST	复位ADC1
RCU_ADC2_RST	复位ADC2
RCU_ADC3_RST	复位ADC3
RCU_DACHOLD_RST	复位DACHOLD
RCU_DAC0_RST	复位DAC0
RCU_DAC1_RST	复位DAC1
RCU_DAC2_RST	复位DAC2
RCU_DAC3_RST	复位DAC3

枚举类型 `rcu_flag_enum`

表 3-834. 枚举类型 `rcu_flag_enum`

成员名称	功能描述
RCU_FLAG_IRC8MSTB	IRC8M稳定标志
RCU_FLAG_HXTALSTB	HXTAL稳定标志
RCU_FLAG_PLLSTB	PLL稳定标志
RCU_FLAG_LXTALSTB	LXTAL稳定标志
RCU_FLAG_IRC32KSTB	IRC32K稳定标志
RCU_FLAG_OBLRST	选项字节重载复位标志
RCU_FLAG BORRST	选项字节重载
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	自由看门狗复位标志
RCU_FLAG_WWDGTRST	窗口看门狗复位标志
RCU_FLAG_LPRST	低电压电源复位标志

枚举类型 `rcu_int_flag_enum`

表 3-835. 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

成员名称	功能描述
RCU_INT_FLAG_IRC8MSTB	IRC8M时钟稳定中断标志
RCU_INT_FLAG_HXTALSTB	HXTAL时钟稳定中断标志
RCU_INT_FLAG_PLLSTB	PLL时钟稳定中断标志
RCU_INT_FLAG_CKM	HXTAL时钟故障中断标志
RCU_INT_FLAG_LCKM	LXTAL时钟故障中断标志

枚举类型 `rcu_int_flag_clear_enum`

表 3-836. 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_IRC8MSTB_CLR	IRC8M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_PLLSTB_CLR	PLL时钟稳定中断清除标志
RCU_INT_FLAG_CKM_CLR	HXTAL时钟故障中断清除标志
RCU_INT_FLAG_LCKM_CLR	LXTAL时钟故障中断清除标志

枚举类型 `rcu_int_enum`

表 3-837. 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_IRC8MSTB	IRC8M时钟稳定中断
RCU_INT_HXTALSTB	外部高速晶振时钟稳定中断
RCU_INT_PLLSTB	PLL时钟稳定中断

枚举类型 `rcu_osci_type_enum`

表 3-838. 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_IRC8M	IRC8M振荡器
RCU_IRC32K	IRC32K振荡器
RCU_PLL_CK	PLL振荡器

枚举类型 `rcu_clock_freq_enum`

表 3-839. 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_PLLP	PLLP时钟
CK_PLLR	PLLR时钟
CK_PLLQ	PLLQ时钟
CK_USART0	USART0时钟
CK_USART1	USART1时钟
CK_USART2	USART2时钟

枚举类型 `usart_idx_enum`

表 3-840. 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

枚举类型 `i2c_idx_enum`

表 3-841. 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 3-842. 枚举类型 `can_idx_enum`

成员名称	功能描述
IDX_CAN0	CAN0索引
IDX_CAN1	CAN1索引
IDX_CAN2	CAN2索引

枚举类型 `adc_idx_enum`表 3-843. 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
IDX_ADC3	ADC3索引

函数 `rcu_deinit`

函数`rcu_deinit`描述见下表：

表 3-844. 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 3-845. 函数 `rcu_periph_clock_enable`

函数名称	<code>rcu_periph_clock_enable</code>
函数原形	<code>void rcu_periph_clock_enable(rcu_periph_enum periph);</code>
功能描述	使能外设时钟
先决条件	-
被调用函数	-
输入参数{in}	
<code>periph</code>	RCU外设，具体参考 表3-831. 枚举类型<code>rcu_periph_enum</code>
输出参数{out}	
-	-

返回值	
-	-

例如：

```
/* enable the USART0 clock */
```

```
rcu_periph_clock_enable(RCU_USART0);
```

函数 rcu_periph_clock_disable

函数rcu_periph_clock_disable描述见下表：

表 3-846. 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3-831. 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 3-847. 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3-832. 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 3-848. 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3-832. 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 3-849. 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3-849. 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 3-850. 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3-849. 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 3-851. 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 3-852. 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 3-853. 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 RC8M	选择CK_IRC8MDIV时钟作为CK_SYS时钟源
RCU_CKSYSSRC_ HXTAL	选择CK_HXTAL时钟作为CK_SYS时钟源
RCU_CKSYSSRC_ PLL	选择CK_PLLP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 3-854. 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_IRC8M / RCU_SCSS_HXTAL / 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 3-855. 函数 rcu_ahb_clock_config

函数名称	rcu_ahb_clock_config
函数原形	void rcu_ahb_clock_config(uint32_t ck_ahb);
功能描述	配置AHB时钟预分频选择
先决条件	-
被调用函数	-
输入参数{in}	
ck_ahb	AHB预分频选择
RCU_AHB_CKSYS	选择CK_SYS时钟x分频（x = 1, 2, 4, 8, 16, 64, 128, 256, 512）

<code>_DIVx</code>	
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 3-856. 函数 `rcu_apb1_clock_config`

函数名称	<code>rcu_apb1_clock_config</code>
函数原形	<code>void rcu_apb1_clock_config(uint32_t ck_apb1);</code>
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}	
<code>ck_apb1</code>	APB1预分频选择
<code>RCU_APB1_CKAHB_DIVx</code>	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 3-857. 函数 `rcu_apb2_clock_config`

函数名称	<code>rcu_apb2_clock_config</code>
函数原形	<code>void rcu_apb2_clock_config(uint32_t ck_apb2);</code>
功能描述	配置APB2时钟预分频选择
先决条件	-
被调用函数	-

输入参数{in}	
ck_apb2	APB2预分频选择
<i>RCU_APB2_CKAHB_DIVx</i>	选择CK_AHB时钟x分频作为CK_APB2时钟 (x = 1, 2, 4, 8, 16)
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* configure CK_AHB/8 as CK_APB2 */
rcu_apb2_clock_config(RCU_APB2_CKAHB_DIV8);
```

函数 rcu_apb3_clock_config

函数rcu_apb3_clock_config描述见下表:

表 3-858. 函数 rcu_apb3_clock_config

函数名称	rcu_apb3_clock_config
函数原形	void rcu_apb3_clock_config(uint32_t ck_apb3);
功能描述	配置APB3时钟预分频选择
先决条件	-
被调用函数	-
输入参数{in}	
ck_apb3	APB3预分频选择
<i>RCU_APB3_CKAHB_DIVx</i>	选择CK_AHB时钟x分频作为CK_APB2时钟 (x = 1, 2, 4, 8, 16)
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* configure CK_AHB/8 as CK_APB3 */
rcu_apb3_clock_config(RCU_APB3_CKAHB_DIV8);
```

函数 rcu_ckout_config

函数rcu_ckout_config描述见下表:

表 3-859. 函数 rcu_ckout_config

函数名称	rcu_ckout_config
函数原形	void rcu_ckout_config(uint32_t ckout_src, uint32_t ckout_div);

功能描述	配置CKOUT时钟源选择及分频系数
先决条件	-
被调用函数	-
输入参数{in}	
ckout_src	CKOUT时钟源选择
RCU_CKOUTSRC_I RC32K	选择内部IRC32K时钟
RCU_CKOUT0SRC _LXTAL	选择外部低速晶体振荡器时钟（LXTAL）
RCU_CKOUTSRC_ CKSYS	选择系统时钟
RCU_CKOUTSRC_I RC8M	选择内部IRC8M时钟
RCU_CKOUTSRC_ HXTAL	选择外部高速晶体振荡器时钟（HXTAL）
RCU_CKOUTSRC_ PLL	选择PLL时钟
输入参数{in}	
ckout_div	CKOUT分频系数
RCU_CKOUT0_DIV x	将CKOUT所选时钟x分频（x = 1, 2, 4, 8, 16）
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* configure the HXTAL as CK_OUT clock source */
```

```
rcu_ckout_config(RCU_CKOUTSRC_HXTAL, RCU_CKOUT_DIV1);
```

函数 rcu_1sckout_enable

函数rcu_1sckout_enable描述见下表：

表 3-860. 函数 rcu_1sckout_enable

函数名称	rcu_1sckout_enable
函数原形	void rcu_1sckout_enable(void);
功能描述	低速时钟输出使能
先决条件	-
被调用函数	-
输入参数{in}	
-	-
输出参数{out}	

-	-
返回值	
-	-

例如：

```
/* enable the low speed clock output */
```

```
rcu_lsckout_enable ();
```

函数 rcu_lsckout_disable

函数rcu_lsckout_disable描述见下表：

表 3-861. 函数 rcu_lsckout_disable

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

例如：

```
/* disable the low speed clock output */
```

```
rcu_lsckout_disable ();
```

函数 rcu_lsckout_config

函数rcu_lsckout_config描述见下表：

表 3-862. 函数 rcu_lsckout_config

函数名称	rcu_lsckout_config
函数原形	void rcu_lsckout_config(uint32_t ls_ckout_src);
功能描述	配置LSCKOUT输出
先决条件	-
被调用函数	-
输入参数{in}	
ls_ckout_src	LSCKOUT时钟源选择
RCU_LSCKOUTSR C_IRC32K	选择IRC32K时钟

<i>RCU_LSCKOUTSR</i> <i>C_LXTAL</i>	选择LXTAL时钟
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* configure the IRC32K as LSCK_OUT clock source */
```

```
rcu_lsckout_config(RCU_LSCKOUTSRC_IRC32K);
```

函数 rcu_pll_source_config

函数rcu_pll_source_config描述见下表:

表 3-863. 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
<i>RCU_PLLSRC_IRC</i> <i>8M</i>	IRC8M时钟被选择为PLL时钟的时钟源
<i>RCU_PLLSRC_HXT</i> <i>AL</i>	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_IRC8M as the PLL clock source */
```

```
rcu_pll_source_config(RCU_PLLSRC_IRC8M);
```

函数 rcu_pll_config

函数rcu_pll_config描述见下表:

表 3-864. 函数 rcu_pll_config

函数名称	rcu_pll_config
函数原形	ErrStatus rcu_pll_config(uint32_t pll_psc, uint32_t pll_n, uint32_t pll_p, uint32_t

	pll_q,uint32_t pll_r);
功能描述	配置PLL时钟
先决条件	-
被调用函数	-
输入参数{in}	
pll_psc	PLL VCO源时钟分频
uint32_t	1-16
输入参数{in}	
pll_n	PLL时钟倍频因子
uint32_t	8-180
输入参数{in}	
pll_p	PLL P时钟分频因子
uint32_t	2,4,6,8
输入参数{in}	
pll_q	PLL Q时钟分频因子
uint32_t	2-15
输入参数{in}	
pll_r	PLL R时钟分频因子
uint32_t	2-31
输出参数{out}	
-	-
返回值	
ErrStatus	SUCCESS or ERROR-

例如:

```
/* configure the PLL */
```

```
rcu_pll_config(4, 90, 1, 2, 2);
```

函数 rcu_pll_clock_output_enable

函数rcu_pll_clock_output_enable描述见下表:

表 3-865. 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 pll_x);
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}	
pll_x	选择PLL P/Q/R分频输出使能
RCU_PLLP	选择PLL P分频输出使能
RCU_PLLQ	选择PLL Q分频输出使能
RCU_PLLR	选择PLL R分频输出使能

输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* enable PLLP divider output */
```

```
rcu_pll_clock_output_enable(RCU_PLLP);
```

函数 rcu_pll_clock_output_disable

函数rcu_pll_clock_output_disable描述见下表：

表 3-866. 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 pll);
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}	
pll	选择PLL P/Q/R分频输出使能
RCU_PLLP	选择PLLP分频输出使能
RCU_PLLQ	选择PLLQ分频输出使能
RCU_PLLR	选择PLLR分频输出使能
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* disable PLLP divider output */
```

```
rcu_pll_clock_output_disable(RCU_PLLP);
```

函数 rcu_rtc_clock_config

函数rcu_rtc_clock_config描述见下表：

表 3-867. 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
<i>RCU_RTCSRC_NO NE</i>	未选择时钟
<i>RCU_RTCSRC_LX TAL</i>	选择CK_LXTAL作为RTC时钟源
<i>RCU_RTCSRC_IRC 32K</i>	选择内部32K RC振荡器时钟作为RTC时钟源
<i>RCU_RTCSRC_HX TAL_DIV32</i>	选择外部高速晶振的32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_usart_clock_config

函数rcu_usart_clock_config描述见下表：

表 3-868. 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3-840. 枚举类型usart_idx_enum
输入参数{in}	
ck_usart	USARTx时钟源
<i>RCU_USARTSRC_ APB</i>	选择CK_APB时钟作为USART时钟
<i>RCU_USARTSRC_ CK_SYS</i>	选择CK_SYS时钟作为USART时钟
<i>RCU_USARTSRC_ LXTAL</i>	选择CK_LXTAL时钟作为USART时钟
<i>RCU_USARTSRC_ RC8M</i>	选择CK_IRC8M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 3-869. 函数 rcu_i2c_clock_config

函数名称	rcu_i2c_clock_config
函数原形	void rcu_i2c_clock_config(i2c_idx_enum i2c_idx, uint32_t ck_i2c);
功能描述	配置I2C时钟
先决条件	-
被调用函数	-
输入参数{in}	
i2c_idx	I2C索引，参考 表3-841. 枚举类型i2c_idx_enum
输入参数{in}	
ck_i2c	I2Cx时钟源
RCU_I2CSRC_APB1	选择CK_APB1时钟作为I2C时钟
RCU_I2CSRC_CKSYS	选择CK_SYS时钟作为I2C时钟
RCU_I2CSRC_IRC8M	选择CK_IRC8M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 3-870. 函数 rcu_can_clock_config

函数名称	rcu_can_clock_config
函数原形	void rcu_can_clock_config(can_idx_enum can_idx, uint32_t ck_can);

功能描述	配置CAN时钟
先决条件	-
被调用函数	-
输入参数{in}	
can_idx_enum	CAN索引, 参考 表3-842. 枚举类型can_idx_enum
输入参数{in}	
ck_can	CANx时钟源
RCU_CANSRC_IRC8M	选择IRC8M时钟作为CAN时钟
RCU_CANSRC_APB2	选择CK_APB2时钟作为CAN时钟
RCU_CANSRC_PLLQ	选择CK_PLLQ时钟作为CAN时钟
RCU_CANSRC_HXTAL	选择HXTAL时钟作为CAN时钟
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* configure the CK_IRC8M as the CAN0 clock source */
```

```
rcu_can_clock_config(IDX_CAN0, RCU_CANSRC_IRC8M);
```

函数 rcu_adc_clock_config

函数rcu_adc_clock_config描述见下表:

表 3-871. 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3-843. 枚举类型adc_idx_enum
输入参数{in}	
ck_adc	ADCx时钟源
RCU_ADCSRC_PLLR	选择CK_PLLR时钟作为ADC时钟
RCU_ADCSRC_CKSYS	选择CK_SYS时钟作为ADC时钟
输出参数{out}	

-	-
返回值	
-	-

例如：

```
/* configure the CK_PLLR as the ADC0 clock source */
```

```
rcu_adc_clock_config(IDX_ADC0, CK_PLLR);
```

函数 rcu_hpdf_clock_config

函数rcu_hpdf_clock_config描述见下表：

表 3-872. 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_hpdf);
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_hpdf	HPDF时钟源
RCU_HPDFSRC_A PB2	选择CK_APB2时钟作为HPDF时钟
RCU_HPDFSRC_A HB	选择CK_AHB时钟作为HPDF时钟
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* configure the CK_AHB as the HPDF clock source */
```

```
rcu_hpdf_clock_config(CK_AHB);
```

函数 rcu_hpdfaudio_clock_config

函数rcu_hpdfaudio_clock_config描述见下表：

表 3-873. 函数 rcu_hpdfaudio_clock_config

函数名称	rcu_hpdfaudio_clock_config
函数原形	void rcu_hpdfaudio_clock_config(uint32_t ck_hpdfaudio);
功能描述	配置HPDF AUDIO时钟源时钟
先决条件	-
被调用函数	-

输入参数{in}	
ck_hpdfaudio	PER时钟源
<i>RCU_HPDAUDIO_PLLQ</i>	选择CK_PLLQ时钟作为HPDF AUDIO时钟
<i>RCU_HPDAUDIO_EHPDAUDIOPIN</i>	选择CK_EHPDAUDIOPIN时钟作为HPDF AUDIO时钟
<i>RCU_HPDAUDIO_IRC8M</i>	选择CK_IRC8M时钟作为HPDF AUDIO时钟
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* configure the PLLQ as the HPDAUDIO clock source */
rcu_hpdfaudio_clock_config (RCU_HPDAUDIO_PLLQ);
```

函数 rcu_trng_clock_config

函数rcu_trng_clock_config描述见下表：

表 3-874. 函数 rcu_trng_clock_config

函数名称	rcu_trng_clock_config
函数原形	void rcu_trng_clock_config(uint32_t ck_trng);
功能描述	配置TRNG时钟源时钟
先决条件	-
被调用函数	-
输入参数{in}	
ck_trng	TRNG时钟源
<i>RCU_TRNG_CKPL LQ_DIVx (x = 2, 3, 4,... , 15)</i>	选择PLLQ的x (x = 2, 3, 4,... , 15) 分频作为TRNG时钟源
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* configure the TRNG prescaler selection */
rcu_trng_clock_config (RCU_TRNG_CKPLLQ_DIV3);
```

函数 rcu_lptimer_clock_config

函数rcu_lptimer_clock_config描述见下表：

表 3-875. 函数 rcu_lptimer_clock_config

函数名称	rcu_lptimer_clock_config
函数原形	void rcu_lptimer_clock_config(uint32_t ck_lptimer);
功能描述	配置LPTIMER时钟源时钟
先决条件	-
被调用函数	-
输入参数{in}	
ck_lptimer	LPTIMER时钟源
RCU_LPTIMERSRC_CKAPB1	选择CK_APB1时钟作为LPTIMER时钟
RCU_LPTIMERSRC_IRC32K	选择CK_IRC32K时钟作为LPTIMER时钟
RCU_LPTIMERSRC_LXTAL	选择CK_LXTAL时钟作为LPTIMER时钟
RCU_LPTIMERSRC_IRC8M	选择CK_IRC8M时钟作为LPTIMER时钟
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* configure the CK_APB1 as the LPTIMER clock source */
```

```
rcu_lptimer_clock_config(RCU_LPTIMERSRC_CKAPB1);
```

函数 rcu_qspi_clock_config

函数rcu_qspi_clock_config描述见下表：

表 3-876. 函数 rcu_qspi_clock_config

函数名称	rcu_qspi_clock_config
函数原形	void rcu_qspi_clock_config(uint32_t ck_qspi);
功能描述	配置QSPI时钟源时钟
先决条件	-
被调用函数	-
输入参数{in}	
ck_qspi	QSPI时钟源
RCU_QSPISRC_CKSYS	选择CK_SYS时钟作为QSPI时钟
RCU_QSPISRC_IRC8M	选择CK_IRC8M时钟作为QSPI时钟
RCU_QSPISRC_PLLR	选择CK_PLLR时钟作为QSPI时钟

输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* configure the CK_SYS as the QSPI clock source */
```

```
rcu_qspi_clock_config (RCU_QSPISRC_CKSYS);
```

函数 rcu_hrtimer_clock_config

函数rcu_hrtimer_clock_config描述见下表:

表 3-877. 函数 rcu_hrtimer_clock_config

函数名称	rcu_hrtimer_clock_config
函数原形	void rcu_hrtimer_clock_config(uint32_t ck_hrtimer);
功能描述	配置HRTIMER时钟源时钟
先决条件	-
被调用函数	-
输入参数{in}	
ck_hrtimer	HRTIMER时钟源
RCU_HRTIMERSR C_CKAPB2	选择CK_APB2时钟作为HRTIMER时钟
RCU_HRTIMERSR C_CKSYS	选择CK_SYS时钟作为HRTIMER时钟
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* configure the APB2 as the HRTIMER clock source */
```

```
rcu_hrtimer_clock_config (RCU_HRTIMERSRC_CKAPB2);
```

函数 rcu_lxtal_drive_capability_config

函数rcu_lxtal_drive_capability_config描述见下表:

表 3-878. 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
<i>RCU_LXTAL_LOW_DRI</i>	低驱动力
<i>RCU_LXTAL_MED_LOWDRI</i>	中低驱动力
<i>RCU_LXTAL_MED_HIGHDRI</i>	中高驱动力
<i>RCU_LXTAL_HIGH_DRI</i>	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 3-879. 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3-838. 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_osc_i_on

函数rcu_osc_i_on描述见下表:

表 3-880. 函数 rcu_osc_i_on

函数名称	rcu_osc_i_on
函数原形	void rcu_osc_i_on(rcu_osc_i_type_enum osci);
功能描述	打开振荡器
先决条件	-
被调用函数	-
输入参数{in}	
osci	振荡器类型, 参考 表3-838. 枚举类型rcu_osc_i_type_enum
输出参数{out}	
-	-
返回值	
-	-

例如:

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

```
rcu_osc_i_on(RCU_HXTAL);
```

函数 rcu_osc_i_off

函数rcu_osc_i_off描述见下表:

表 3-881. 函数 rcu_osc_i_off

函数名称	rcu_osc_i_off
函数原形	void rcu_osc_i_off(rcu_osc_i_type_enum osci);
功能描述	关闭振荡器
先决条件	-
被调用函数	-
输入参数{in}	
osci	振荡器类型, 参考 表3-838. 枚举类型rcu_osc_i_type_enum
输出参数{out}	
-	-
返回值	
-	-

例如:

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

```
rcu_osc_i_off(RCU_HXTAL);
```

函数 `rcu_osc_bypass_mode_enable`

函数 `rcu_osc_bypass_mode_enable` 描述见下表：

表 3-882. 函数 `rcu_osc_bypass_mode_enable`

函数名称	<code>rcu_osc_bypass_mode_enable</code>
函数原形	<code>void rcu_osc_bypass_mode_enable(rcu_osc_type_enum osci);</code>
功能描述	使能振荡器时钟旁路模式
先决条件	HXTALEN或LXTALEN应在使能振荡器时钟旁路模式前先复位
被调用函数	-
输入参数{in}	
osci	振荡器类型，参考 表3-838. 枚举类型rcu_osc_type_enum
输出参数{out}	
-	-
返回值	
-	-

例如：

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

```
rcu_osc_bypass_mode_enable(RCU_HXTAL);
```

函数 `rcu_osc_bypass_mode_disable`

函数 `rcu_osc_bypass_mode_disable` 描述见下表：

表 3-883. 函数 `rcu_osc_bypass_mode_disable`

函数名称	<code>rcu_osc_bypass_mode_disable</code>
函数原形	<code>void rcu_osc_bypass_mode_disable(rcu_osc_type_enum osci);</code>
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3-838. 枚举类型rcu_osc_type_enum
输出参数{out}	
-	-
返回值	
-	-

例如：

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

```
rcu_hxtal_clock_monitor_disable();
```

函数 rcu_irc8m_adjust_value_set

函数rcu_irc8m_adjust_value_set描述见下表：

表 3-884. 函数 rcu_irc8m_adjust_value_set

函数名称	rcu_irc8m_adjust_value_set
函数原形	void rcu_irc8m_adjust_value_set(uint32_t irc8m_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}	
irc8m_adjval	IRC8M调整值（0到0x1F之间）
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* set the IRC8M adjust value */
rcu_irc8m_adjust_value_set(0x20);
```

函数 rcu_hxtal_clock_monitor_enable

函数rcu_hxtal_clock_monitor_enable描述见下表：

表 3-885. 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 3-886. 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 3-887. 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 3-888. 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 3-889. 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3-839. 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 3-890. 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3-834. 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 3-891. 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 3-892. 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3-837. 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 3-893. 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3-837. 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 3-894. 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3-837. 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 3-895. 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3-836. 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);
```

3.26. 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[3.26.1](#)描述了RTC的寄存器列表，章节[3.26.2](#)对RTC库函数进行说明。

3.26.1. 外设寄存器描述

RTC寄存器列表如下表所示：

表 3-896. RTC 寄存器

寄存器名称	寄存器描述
RTC_TIME	RTC时间寄存器
RTC_DATE	RTC日期寄存器
RTC_CTL	RTC控制寄存器
RTC_STAT	RTC状态寄存器
RTC_PSC	RTC预分频寄存器
RTC_WUT	RTC唤醒定时器寄存器
RTC_ALRM0TD	RTC闹钟0时间日期寄存器
RTC_ALRM1TD	RTC闹钟1时间日期寄存器
RTC_WPK	RTC写保护钥匙寄存器
RTC_SS	RTC亚秒寄存器
RTC_SHIFTCTL	RTC移位控制寄存器
RTC_TTS	RTC时间戳时间寄存器
RTC_DTS	RTC时间戳日期寄存器
RTC_SSTS	RTC时间戳亚秒寄存器
RTC_HRFC	RTC高精度频率补偿寄存器
RTC_TAMP	RTC侵入寄存器
RTC_ALRM0SS	RTC闹钟0亚秒寄存器
RTC_ALRM1SS	RTC闹钟1亚秒寄存器
RTC_BKPx (x=0..31)	RTC备份域寄存器x (x=0..31)

3.26.2. 外设库函数描述

RTC库函数列表如下表所示：

表 3-897. 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

库函数名称	库函数描述
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
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 3-898. 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 3-899. 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 3-900. 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 3-901. 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 3-902. 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 3-903. 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3-898. 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 3-904. 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 3-905. 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 3-906. 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 3-907. 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3-898. 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 3-908. 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 3-909. 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3-899. 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 3-910. 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
<i>RTC_MASKSSC_0_14</i>	屏蔽闹钟亚秒设置
<i>RTC_MASKSSC_1_14</i>	屏蔽RTC_ALARM0SS_SSC[14:1], SSC[0]位用于时间匹配
<i>RTC_MASKSSC_2_14</i>	屏蔽RTC_ALARM0SS_SSC[14:2], SSC[1:0]位用于时间匹配
<i>RTC_MASKSSC_3_14</i>	屏蔽RTC_ALARM0SS_SSC[14:3], SSC[2:0]位用于时间匹配
<i>RTC_MASKSSC_4_14</i>	屏蔽RTC_ALARM0SS_SSC[14:4], SSC[3:0]位用于时间匹配
<i>RTC_MASKSSC_5_14</i>	屏蔽RTC_ALARM0SS_SSC[14:5], SSC[4:0]位用于时间匹配
<i>RTC_MASKSSC_6_14</i>	屏蔽RTC_ALARM0SS_SSC[14:6], SSC[5:0]位用于时间匹配
<i>RTC_MASKSSC_7_14</i>	屏蔽RTC_ALARM0SS_SSC[14:7], SSC[6:0]位用于时间匹配
<i>RTC_MASKSSC_8_14</i>	屏蔽RTC_ALARM0SS_SSC[14:8], SSC[7:0]位用于时间匹配
<i>RTC_MASKSSC_9_14</i>	屏蔽RTC_ALARM0SS_SSC[14:9], SSC[8:0]位用于时间匹配
<i>RTC_MASKSSC_10_14</i>	屏蔽RTC_ALARM0SS_SSC[14:10], SSC[9:0]位用于时间匹配
<i>RTC_MASKSSC_11_14</i>	屏蔽RTC_ALARM0SS_SSC[14:11], SSC[10:0]位用于时间匹配
<i>RTC_MASKSSC_12_14</i>	屏蔽RTC_ALARM0SS_SSC[14:12], SSC[11:0]位用于时间匹配
<i>RTC_MASKSSC_13_14</i>	屏蔽RTC_ALARM0SS_SSC[14:13], SSC[12:0]位用于时间匹配
<i>RTC_MASKSSC_14</i>	屏蔽RTC_ALARM0SS_SSC[14], SSC[13:0]位用于时间匹配
<i>RTC_MASKSSC_NONE</i>	无屏蔽, SSC[14:0]位用于时间匹配
输入参数{in}	
subsecond	闹钟亚秒值(0x000 - 0x7FFF)
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/*configure subsecond of RTC alarm*/
```

```
rtc_subsecond_config(RTC_ALARM0,RTC_MASKSSC_9_14, 0x7FFF);
```

函数 rtc_alarm_enable

函数rtc_alarm_enable描述见下表:

表 3-911. 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 3-912. 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 3-913. 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3-899. 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 3-914. 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或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 3-915. 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
<i>RTC_TIMESTAMP_RISING_EDGE</i>	上升沿是时间戳事件有效检测沿
<i>RTC_TIMESTAMP_FALLING_EDGE</i>	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 3-916. 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 3-917. 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 3-918. 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3-900. 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 3-919. 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 3-920. 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3-901. 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 3-921. 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 3-922. 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 3-923. 函数 `rtc_alarm_output_config`

函数名称	<code>rtc_alarm_output_config</code>
函数原型	<code>void rtc_alarm_output_config(uint32_t source, uint32_t mode);</code>
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
<code>RTC_ALARM0_HIGH</code>	当alarm0标志置位, 输出引脚为高电平
<code>RTC_ALARM0_LOW</code>	当alarm0标志置位, 输出引脚为低电平
<code>RTC_ALARM1_HIGH</code>	当alarm1标志置位, 输出引脚为高电平
<code>RTC_ALARM1_LOW</code>	当alarm1标志置位, 输出引脚为低电平
<code>RTC_WAKEUP_HIGH</code>	当唤醒标志置位, 输出引脚为高电平
<code>RTC_WAKEUP_LOW</code>	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
<code>RTC_ALARM_OUTPUT_T_OD</code>	开漏输出
<code>RTC_ALARM_OUTPUT_T_PP</code>	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 3-924. 函数 `rtc_calibration_output_config`

函数名称	<code>rtc_calibration_output_config</code>
函数原型	<code>void rtc_calibration_output_config(uint32_t source);</code>
功能描述	RTC校准输出配置
先决条件	-
被调用函数	-

输入参数{in}	
source	配置输出信号
<i>RTC_CALIBRATION_5</i> <i>12HZ</i>	当LSE频率为32768Hz并且RTC_PSC为默认值，输出512Hz信号
<i>RTC_CALIBRATION_1</i> <i>HZ</i>	当LSE频率为32768Hz并且RTC_PSC为默认值，输出1Hz信号
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* when the LSE frequency is 32768Hz and the RTC_PSC
```

```
is the default value, output 1Hz signal */
```

```
rtc_calibration_output_config(RTC_CALIBRATION_1HZ);
```

函数 rtc_hour_adjust

函数rtc_hour_adjust描述见下表：

表 3-925. 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 3-926. 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
RTC_SHIFT_ADD1S_RESET	无影响
RTC_SHIFT_ADD1S_SET	在当前时间增加1秒
输入参数{in}	
minus	在当前是时间上减少的亚秒值(0x0 - 0x7FFF)
输出参数{out}	
-	-
返回值	
ErrStatus	ERROR或SUCCESS

例如:

```
/* adjust RTC second or subsecond value of current time */
```

```
ErrStatus error_status = rtc_second_adjust(RTC_SHIFT_ADD1S_SET, 0);
```

函数 rtc_bypass_shadow_enable

函数rtc_bypass_shadow_enable描述见下表:

表 3-927. 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 3-928. 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 3-929. 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 3-930. 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或SUCCESS

例如:

```
/* disableRTC reference clock detection function*/
```

```
ErrStatus error_status = rtc_refclock_detection_disable();
```

函数 rtc_wakeup_enable

函数rtc_wakeup_enable描述见下表:

表 3-931. 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 3-932. 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 3-933. 函数 rtc_wakeup_clock_set

函数名称	rtc_wakeup_clock_set
函数原型	ErrStatus rtc_wakeup_clock_set(uint8_t wakeup_clock);
功能描述	设置RTC自动唤醒定时器时钟
先决条件	-
被调用函数	-
输入参数{in}	
wakeup_clock	时钟选择
WAKEUP_RTCK_DIV 16	RTC时钟的16分频
WAKEUP_RTCK_DIV 8	RTC时钟的8分频
WAKEUP_RTCK_DIV 4	RTC时钟的4分频
WAKEUP_RTCK_DIV 2	RTC时钟的2分频
WAKEUP_CKSPRE	ck_spre（默认1Hz）时钟
WAKEUP_CKSPRE_2 EXP16	ck_spre（默认1Hz）时钟并且将唤醒计数器值增加 2^{16}

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

例如:

```
/* RTC auto wakeup timer clock is ckspre */
```

```
ErrStatus error_status = rtc_wakeup_clock_set(WAKEUP_CKSPRE);
```

函数 rtc_wakeup_timer_set

函数rtc_wakeup_timer_set描述见下表:

表 3-934. 函数 rtc_wakeup_timer_set

函数名称	rtc_wakeup_timer_set
函数原型	ErrStatus rtc_wakeup_timer_set(uint16_t wakeup_timer);
功能描述	设置RTC自动唤醒定时器值
先决条件	-
被调用函数	-
输入参数{in}	
wakeup_timer	定时器选择
uint16_t	0x0000-0xffff
输出参数{out}	
-	-
返回值	
ErrStatus	ERROR或SUCCESS

例如:

```
/* set wakeup timer value */
```

```
ErrStatus error_status = rtc_wakeup_timer_set(0XFFEE);
```

函数 rtc_wakeup_timer_get

函数rtc_wakeup_timer_get描述见下表:

表 3-935. 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 3-936. 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_WINDOW_32S	采用32S校准周期
RTC_CALIBRATION_WINDOW_16S	采用16S校准周期
RTC_CALIBRATION_WINDOW_8S	采用8S校准周期
输入参数{in}	
plus	增加脉冲
RTC_CALIBRATION_PLUS_SET	每2048个脉冲增加一个RTCCLK脉冲
RTC_CALIBRATION_PLUS_RESET	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 3-937. 函数 rtc_interrupt_enable

函数名称	rtc_interrupt_enable
函数原型	void rtc_interrupt_enable(uint32_t interrupt);
功能描述	使能RTC指定的中断
先决条件	-
被调用函数	-
输入参数{in}	
interrupt	选定被使能的中断源
RTC_INT_TIMESTAMP	时间戳中断
RTC_INT_ALARM0	闹钟0中断
RTC_INT_ALARM1	闹钟1中断
RTC_INT_TAMP_ALL	所有侵入检测中断
RTC_INT_TAMP0	侵入检测0中断
RTC_INT_TAMP1	侵入检测1中断
RTC_INT_TAMP2	侵入检测2中断
RTC_INT_WAKEUP	唤醒定时器中断
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* enable specified RTC interrupt*/
```

```
rtc_interrupt_enable(RTC_INT_TAMP0);
```

函数 rtc_interrupt_disable

函数rtc_interrupt_disable描述见下表：

表 3-938. 函数 rtc_interrupt_disable

函数名称	rtc_interrupt_disable
函数原型	void rtc_interrupt_disable(uint32_t interrupt);
功能描述	失能RTC指定中断
先决条件	-

被调用函数	-
输入参数{in}	
interrupt	选定被失能的RTC中断
RTC_INT_TIMESTAMP	时间戳中断
RTC_INT_ALARM0	闹钟0中断
RTC_INT_ALARM1	闹钟1中断
RTC_INT_TAMP_ALL	所有侵入检测中断
RTC_INT_TAMP0	侵入检测0中断
RTC_INT_TAMP1	侵入检测1中断
RTC_INT_TAMP2	侵入检测2中断
RTC_INT_WAKEUP	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 3-939. 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_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
RTC_FLAG_SOP	移位功能操作挂起标志
RTC_FLAG_YCM	年份配置标志
RTC_FLAG_RSYN	寄存器同步标志
RTC_FLAG_INIT	进入初始化模式
RTC_FLAG_SCP	平滑校准挂起标志
RTC_FLAG_ALARM0	Alarm0发生标志

<i>RTC_FLAG_ALARM1</i>	Alarm1发生标志
<i>RTC_FLAG_WT</i>	唤醒事件标志
<i>RTC_FLAG_TS</i>	时间戳事件标志
<i>RTC_FLAG_TSOVR</i>	时间戳事件溢出标志
<i>RTC_FLAG_TP0</i>	tamper 0事件标志
<i>RTC_FLAG_TP1</i>	tamper 1事件标志
<i>RTC_FLAG_TP2</i>	tamper 2事件标志
<i>RTC_FLAG_ITS</i>	内部时间戳标志
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 3-940. 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
<i>RTC_FLAG_ALARM0</i> <i>W</i>	Alarm0配置可写标志
<i>RTC_FLAG_ALARM1</i> <i>W</i>	Alarm1配置可写标志
<i>RTC_FLAG_WTW</i>	唤醒计数器可写标志
<i>RTC_FLAG_SOP</i>	移位功能操作挂起标志
<i>RTC_FLAG_YCM</i>	年份配置标志
<i>RTC_FLAG_RSYN</i>	寄存器同步标志
<i>RTC_FLAG_INIT</i>	进入初始化模式
<i>RTC_FLAG_SCP</i>	平滑校准挂起标志
<i>RTC_FLAG_ALARM0</i>	Alarm0发生标志
<i>RTC_FLAG_ALARM1</i>	Alarm1发生标志
<i>RTC_FLAG_WT</i>	唤醒事件标志
<i>RTC_FLAG_TS</i>	时间戳事件标志

<i>RTC_FLAG_TSOVR</i>	时间戳事件溢出标志
<i>RTC_FLAG_TP0</i>	tamper 0事件标志
<i>RTC_FLAG_TP1</i>	tamper 1事件标志
<i>RTC_FLAG_TP2</i>	tamper 2事件标志
<i>RTC_FLAG_ITS</i>	内部时间戳标志
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* cleartime-stamp event flag */
```

```
rtc_flag_clear(RTC_FLAG_TS);
```

3.27. HRTIMER

HRTIMER具有高精度计数时钟，可用于高精度定时。它可以产生16个高精度的数字信号来灵活地控制电动机或用于电源管理应用。这16个数字信号可以独立输出，也可以耦合成8对互补信号输出。章节[3.27.1](#)描述了HRTIMER的寄存器列表，章节[3.27.2](#)对HRTIMER库函数进行说明。

3.27.1. 外设寄存器说明

HRTIMER寄存器列表如下表所示:

表 3-941. HRTIMER 寄存器

寄存器名称	寄存器描述
Master Timer寄存器	
HRTIMER_MTCTL0	Master_TIMER控制寄存器0
HRTIMER_MTINTF	Master_TIMER中断标志寄存器
HRTIMER_MTINTC	Master_TIMER中断标志清除寄存器
HRTIMER_MTDMAINTEN	Master_TIMER DMA和中断使能寄存器
HRTIMER_MTCNT	Master_TIMER计数器寄存器
HRTIMER_MTCAR	Master_TIMER计数器自动重载寄存器
HRTIMER_MTCREP	Master_TIMER重复计数寄存器
HRTIMER_MTCMP0V	Master_TIMER比较0寄存器
HRTIMER_MTCMP1V	Master_TIMER比较1寄存器
HRTIMER_MTCMP2V	Master_TIMER比较2寄存器
HRTIMER_MTCMP3V	Master_TIMER比较3寄存器
Slave Time寄存器	
HRTIMER_STXCTL0	Slave_TIMERx控制寄存器0
HRTIMER_STXINTF	Slave_TIMERx中断标志寄存器

寄存器名称	寄存器描述
HRTIMER_STXINTC	Slave_TIMERx中断标志清除寄存器
HRTIMER_STXDMAINTEN	Slave_TIMERx DMA和中断使能寄存器
HRTIMER_STXCNT	Slave_TIMERx计数器寄存器
HRTIMER_STXCAR	Slave_TIMERx计数器自动重载寄存器
HRTIMER_STXCREP	Slave_TIMERx重复计数寄存器
HRTIMER_STXCMP0V	Slave_TIMERx比较0寄存器
HRTIMER_STXCMP0CP	Slave_TIMERx比较0复合寄存器
HRTIMER_STXCMP1V	Slave_TIMERx比较1寄存器
HRTIMER_STXCMP2V	Slave_TIMERx比较2寄存器
HRTIMER_STXCMP3V	Slave_TIMERx比较3寄存器
HRTIMER_STXCAP0V	Slave_TIMERx捕获0寄存器
HRTIMER_STXCAP1V	Slave_TIMERx捕获1寄存器
HRTIMER_STXDTCTL	Slave_TIMERx死区控制寄存器
HRTIMER_STXCH0SET	Slave_TIMERx通道0置位请求寄存器
HRTIMER_STXCH0RST	Slave_TIMERx通道0复位请求寄存器
HRTIMER_STXCH1SET	Slave_TIMERx通道1置位请求寄存器
HRTIMER_STXCH1RST	Slave_TIMERx通道1复位请求寄存器
HRTIMER_STXEXEVFCFG0	Slave_TIMERx外部事件滤波配置寄存器0
HRTIMER_STXEXEVFCFG1	Slave_TIMERx外部事件滤波配置寄存器1
HRTIMER_STXCNTRST	Slave_TIMERx计数器复位寄存器
HRTIMER_STXCSCCTL	Slave_TIMERx载波控制寄存器
HRTIMER_STXCAP0TRG	Slave_TIMERx捕获0触发寄存器
HRTIMER_STXCAP1TRG	Slave_TIMERx捕获1触发寄存器
HRTIMER_STXCHOCTL	Slave_TIMERx通道输出控制寄存器
HRTIMER_STXFLTCTL	Slave_TIMERx故障控制寄存器
HRTIMER_STXCTL1	Slave_TIMERx控制寄存器1
HRTIMER_STXEXEVFCFG2	Slave_TIMERx外部事件滤波配置寄存器2
HRTIMER_STXCAPTRGCOM	Slave_TIMERx公共捕获触发寄存器
HRTIMER_STXCNTRSTA	Slave_TIMERx计数器复位附加寄存器
HRTIMER_STXACTL	Slave_TIMERx附加控制寄存器
通用寄存器	
HRTIMER_CTL0	HRTIMER控制寄存器0
HRTIMER_CTL1	HRTIMER控制寄存器1
HRTIMER_INTF	HRTIMER中断标志寄存器
HRTIMER_INTC	HRTIMER中断标志清除寄存器
HRTIMER_INTEN	HRTIMER中断使能寄存器
HRTIMER_CHOUTEN	HRTIMER通道输出使能寄存器
HRTIMER_CHOUTDIS	HRTIMER通道输出禁能寄存器
HRTIMER_CHOUTDISF	HRTIMER通道输出禁能标志寄存器
HRTIMER_BMCTL	HRTIMER突发模式控制寄存器

寄存器名称	寄存器描述
HRTIMER_BMSTRG	HRTIMER突发模式启动触发寄存器
HRTIMER_BMCMPV	HRTIMER突发模式比较值寄存器
HRTIMER_BMCAR	HRTIMER突发模式计数器自动重载寄存器
HRTIMER_EXEVCFG0	HRTIMER外部事件配置寄存器0
HRTIMER_EXEVCFG1	HRTIMER外部事件配置寄存器1
HRTIMER_EXEVDFCTL	HRTIMER外部事件数字滤波控制寄存器
HRTIMER_ADCTRIGS0	HRTIMER ADC触发源0寄存器
HRTIMER_ADCTRIGS1	HRTIMER ADC触发源1寄存器
HRTIMER_ADCTRIGS2	HRTIMER ADC触发源2寄存器
HRTIMER_ADCTRIGS3	HRTIMER ADC触发源3寄存器
HRTIMER_DLLCCTL	HRTIMER DLL校准控制寄存器
HRTIMER_FLTINCFG0	HRTIMER故障输入配置寄存器0
HRTIMER_FLTINCFG1	HRTIMER故障输入配置寄存器1
HRTIMER_DMAUPMTR	HRTIMER DMA更新Master_TIMER寄存器
HRTIMER_DMAUPST0R	HRTIMER DMA更新Slave_TIMER0寄存器
HRTIMER_DMAUPST1R	HRTIMER DMA更新Slave_TIMER1寄存器
HRTIMER_DMAUPST2R	HRTIMER DMA更新Slave_TIMER2寄存器
HRTIMER_DMAUPST3R	HRTIMER DMA更新Slave_TIMER3寄存器
HRTIMER_DMAUPST4R	HRTIMER DMA更新Slave_TIMER4寄存器
HRTIMER_DMAUPST5R	HRTIMER DMA更新Slave_TIMER5寄存器
HRTIMER_DMAUPST6R	HRTIMER DMA更新Slave_TIMER6寄存器
HRTIMER_DMAUPST7R	HRTIMER DMA更新Slave_TIMER7寄存器
HRTIMER_DMATB	HRTIMER DMA传输缓冲寄存器
HRTIMER_ADCEXTTRG	HRTIMER ADC外部触发源寄存器
HRTIMER_ADCTRIGUPD	HRTIMER ADC触发更新寄存器
HRTIMER_ADCPSCR0	HRTIMER ADC预分频寄存器0
HRTIMER_ADCPSCR1	HRTIMER ADC预分频寄存器1
HRTIMER_FLTINCFG2	HRTIMER故障输入配置寄存器2
HRTIMER_FLTINCFG3	HRTIMER故障输入配置寄存器3
HRTIMER_BMSTRGA	HRTIMER突发模式启动触发附加寄存器
HRTIMER_FLTINCFG4	HRTIMER故障输入配置寄存器4
HRTIMER_ADCEXTTRGA	HRTIMER ADC外部触发源附加寄存器
HRTIMER_ADCTRIGS0A	HRTIMER ADC触发源0附加寄存器
HRTIMER_ADCTRIGS1A	HRTIMER ADC触发源1附加寄存器
HRTIMER_ADCTRIGS2A	HRTIMER ADC触发源2附加寄存器
HRTIMER_ADCTRIGS3A	HRTIMER ADC触发源3附加寄存器

3.27.2. 外设库函数说明

HRTIMER库函数列表如下表所示：

表 3-942. HRTIMER 库函数

库函数名称	库函数描述
hrtimer_deinit	复位HRTIMER
hrtimer_dll_calibration_start	配置和启动DLL校准
hrtimer_baseinit_struct_para_init	初始化hrtimer_baseinit_parameter_struct结构体变量
hrtimer_timers_base_init	初始化Master_TIMER/Slave_TIMER时基
hrtimer_timers_counter_enable	使能定时器的计数器
hrtimer_timers_counter_disable	禁能定时器的计数器
hrtimer_timers_update_event_enable	使能Master_TIMER/Slave_TIMER更新事件
hrtimer_timers_update_event_disable	禁能Master_TIMER/Slave_TIMER更新事件
hrtimer_software_update	软件触发Master_TIMER/Slave_TIMER更新事件
hrtimer_software_counter_reset	软件复位Master_TIMER/Slave_TIMER计数器
hrtimer_output_exchange	交换Slave_TIMER输出
hrtimer_timerinit_struct_para_init	初始化hrtimer_timerinit_parameter_struct结构体变量
hrtimer_timers_waveform_init	定时器波形初始化
hrtimer_timercfg_struct_para_init	初始化hrtimer_timercfg_parameter_struct结构体变量
hrtimer_slavetimer_waveform_config	配置Slave_TIMER的波形
hrtimer_comparecfg_struct_para_init	初始化hrtimer_comparecfg_parameter_struct结构体
hrtimer_slavetimer_waveform_compare_config	配置Slave_TIMER波形的比较功能
hrtimer_channel_outputcfg_struct_para_init	初始化hrtimer_channel_outputcfg_parameter_struct结构体变量
hrtimer_slavetimer_waveform_channel_config	Slave_TIMER波形的通道配置
hrtimer_slavetimer_waveform_channel_software_request	软件产生通道置位或复位请求
hrtimer_slavetimer_waveform_channel_output_level_get	获取通道的输出电平
hrtimer_slavetimer_waveform_channel_state_get	获取通道的运行状态
hrtimer_deadtimecfg_struct_para_init	初始化hrtimer_deadtimecfg_parameter_struct结构体变量
hrtimer_slavetimer_deadtime_config	配置Slave_TIMER的死区时间
hrtimer_carriersignalcfg_struct_para_init	初始化hrtimer_carriersignalcfg_parameter_struct结构体变量
hrtimer_slavetimer_carriersignal_config	配置Slave_TIMER的载波功能
hrtimer_output_channel_enable	使能通道输出
hrtimer_output_channel_disable	禁能通道输出
hrtimer_mastertimer_compare_value_config	配置主定时器的比较寄存器值
hrtimer_mastertimer_compare_value_get	获取主定时器的比较寄存器值

库函数名称	库函数描述
get	
hrtimer_slavetimer_compare_value_config	配置Slave_TIMER的比较寄存器值
hrtimer_slavetimer_compare_value_get	获取Slave_TIMER的比较寄存器值
hrtimer_timers_counter_value_config	配置Master_TIMER/Slave_TIMER的计数寄存器值
hrtimer_timers_counter_value_get	获取Master_TIMER/Slave_TIMER的计数寄存器值
hrtimer_timers_autoreload_value_config	配置Master_TIMER/Slave_TIMER的自动重载寄存器值
hrtimer_timers_autoreload_value_get	获取Master_TIMER/Slave_TIMER的自动重载寄存器值
hrtimer_timers_repetition_value_config	配置Master_TIMER/Slave_TIMER的重复计数寄存器值
hrtimer_timers_repetition_value_get	获取Master_TIMER/Slave_TIMER的重复计数寄存器值
hrtimer_exevcfilter_struct_para_init	初始化hrtimer_exevcfilter_parameter_struct结构体变量
hrtimer_slavetimer_exeevent_filtering_config	配置Slave_TIMER的外部事件滤波功能
hrtimer_exeeventcfg_struct_para_init	初始化hrtimer_exeeventcfg_parameter_struct结构体变量
hrtimer_exeevent_config	配置外部事件
hrtimer_exeevent_prescaler	配置外部事件的数字滤波分频
hrtimer_exeeventx_counter_struct_para_init	初始化hrtimer_exeeventx_counter_struct结构体变量
hrtimer_exeeventx_counter_config	配置hrtimer外部事件X计数值
hrtimer_software_reset_exeeventx_counter	软件初始化hrtimer外部事件X计数值
hrtimer_exeeventx_counter_enable	hrtimer外部事件X计数使能
hrtimer_exeeventx_counter_disable	hrtimer外部事件X计数禁能
hrtimer_exeeventx_counter_read	hrtimer外部事件X计数值读取
hrtimer_synccfg_struct_para_init	初始化hrtimer_synccfg_parameter_struct结构体变量
hrtimer_synchronization_config	配置HRTIMER的同步输入/输出功能
hrtimer_double_channel_struct_para_init	初始化hrtimer_double_trigger_parameter_struct结构体变量
hrtimer_double_trigger_config	配置双通道触发
hrtimer_roll_over_struct_para_init	初始化hrtimer_roll_over_struct结构体变量
hrtimer_roll_over_mode_config	配置hrtimer翻转模式
hrtimer_faultcfg_struct_para_init	初始化hrtimer_faultcfg_parameter_struct结构体变量
hrtimer_fault_config	配置故障输入功能
hrtimer_fault_prescaler_config	配置故障输入数字滤波分频
hrtimer_fault_input_enable	故障输入使能
hrtimer_fault_input_disable	故障输入禁能
hrtimer_fault_counter_reset	复位hrtimer故障计数值
hrtimer_fault_blank_enable	使能hrtimer故障消隐

库函数名称	库函数描述
hrtimer_fault_blank_disable	禁能hrtimer故障消隐
hrtimer_timers_dma_enable	使能Master_TIMER/Slave_TIMER DMA请求
hrtimer_timers_dma_disable	禁能Master_TIMER/Slave_TIMER DMA请求
hrtimer_dmamode_config	配置Master_TIMER/Slave_TIMER的DMA模式
hrtimer_bunchmode_struct_para_init	初始化hrtimer_bunchmode_parameter_struct结构体
hrtimer_bunchmode_config	配置HRTIMER的突发模式
hrtimer_bunchmode_enable	使能突发模式
hrtimer_bunchmode_disable	禁能突发模式
hrtimer_bunchmode_flag_get	获取突发模式的运行标志
hrtimer_bunchmode_software_start	软件启动突发模式
hrtimer_slavetimer_capture_config	配置Slave_TIMER捕获源
hrtimer_slavetimer_capture_software	软件触发Slave_TIMER捕获
hrtimer_slavetimer_capture_value_read	读取捕获值
hrtimer_adctrigcfg_struct_para_init	初始化hrtimer_adctrigcfg_parameter_struct结构体变量
hrtimer_adc_trigger0_3_config	配置ADC0_3触发源和更新源
hrtimer_adc_trigger4_9_config	配置ADC4_9触发源和更新源
hrtimer_adc_prescaler_config	配置ADC触发分频系数
hrtimer_slavetimer_counter_direction_get	获取Slave_TIMER的计数方向
hrtimer_timers_flag_get	获取Master_TIMER/Slave_TIMER的标志
hrtimer_timers_flag_clear	清除Master_TIMER/Slave_TIMER的标志
hrtimer_common_flag_get	获取HRTIMER通用标志
hrtimer_common_flag_clear	清除HRTIMER通用标志
hrtimer_timers_interrupt_enable	使能Master_TIMER/Slave_TIMER中断
hrtimer_timers_interrupt_disable	禁能Master_TIMER/Slave_TIMER中断
hrtimer_timers_interrupt_flag_get	获取Master_TIMER/Slave_TIMER中断标志
hrtimer_timers_interrupt_flag_clear	清除Master_TIMER/Slave_TIMER中断标志
hrtimer_common_interrupt_enable	使能HRTIMER通用中断
hrtimer_common_interrupt_disable	禁能HRTIMER通用中断
hrtimer_common_interrupt_flag_get	获取HRTIMER通用中断标志
hrtimer_common_interrupt_flag_clear	清除HRTIMER通用中断标志

结构体 hrtimer_baseinit_parameter_struct

表 3-943. 结构体 hrtimer_baseinit_parameter_struct

成员名称	功能描述
period	周期值，最小值：3个tHRTIMER_CK时钟，最大值：0xFFFF - (1个tHRTIMER_CK)
repetitioncounter	重复计数值，0x00~0xFF
prescaler	预分频值
counter_mode	计数器运行模式

counterdirection	计数方向
------------------	------

结构体 `hrtimer_timerinit_parameter_struct`

表 3-944. 结构体 `hrtimer_timerinit_parameter_struct`

成员名称	功能描述
half_mode	使能或禁能半波模式
alternate_mode	交错模式
resynchronized_update	重新同步更新
start_sync	同步输入启动计数器
reset_sync	同步输入复位计数器
dac_trigger	DAC触发源
shadow	使能或者禁能影子寄存器
update_selection	更新事件源选择
cnt_bunch	突发模式下计数器时钟停止或者正常运行
repetition_update	重复事件产生更新事件

结构体 `hrtimer_timercfg_parameter_struct`

表 3-945. 结构体 `hrtimer_timercfg_parameter_struct`

成员名称	功能描述
balanced_mode	使能或禁能均衡模式
fault_enable	使能或禁能Slave_TIMER故障输入通道
fault_protect	使能或禁能故障保护功能
fault_automatic_resume	故障自动恢复功能
deadtime_enable	使能或禁能死区功能
delayed_idle	延时空闲模式
balanced_idle_automatic_resume	均衡空闲自动恢复功能
update_source	更新事件由其他定时器更新事件产生
cnt_reset	计数器复位源
reset_update	计数器复位产生更新事件

结构体 `hrtimer_capture_value_struct`

表 3-946. 结构体 `hrtimer_capture_value_struct`

成员名称	功能描述
value	捕获值
dir	发生捕获时的计数方向

结构体 `hrtimer_comparecfg_parameter_struct`

表 3-947. 结构体 `hrtimer_comparecfg_parameter_struct`

成员名称	功能描述
<code>compare_value</code>	比较寄存器值，最小值：3个tHRTIMER_CK时钟，最大值：0xFFFF - (1个tHRTIMER_CK)
<code>delayed_mode</code>	配置比较延时模式（只有比较1和3寄存器有该功能）
<code>timeout_value</code>	超时值（在比较延时模式具有超时功能时该值有效）
<code>trigger_half</code>	触发半波模式
<code>immediately_update_cmp0</code>	比较0事件立即更新
<code>immediately_update_cmp2</code>	比较2事件立即更新

结构体 `hrtimer_exevfilter_parameter_struct`

表 3-948. 结构体 `hrtimer_exevfilter_parameter_struct`

成员名称	功能描述
<code>filter_mode</code>	外部事件的滤波模式
<code>memorized</code>	外部事件滤波存储功能

结构体 `hrtimer_deadtimecfg_parameter_struct`

表 3-949. 结构体 `hrtimer_deadtimecfg_parameter_struct`

成员名称	功能描述
<code>prescaler</code>	死区时间发生器预分频
<code>rising_value</code>	死区上升沿数值，0x0000~0xFFFF
<code>rising_sign</code>	死区上升沿数值符号
<code>rising_protect</code>	死区上升数值值和符号保护
<code>risingsign_protect</code>	死区上升沿符号保护
<code>falling_value</code>	死区下降沿数值，0x0000~0xFFFF
<code>falling_sign</code>	死区下降沿数值符号
<code>falling_protect</code>	死区下降数值值和符号保护
<code>fallingsign_protect</code>	死区下降沿符号保护

结构体 `hrtimer_carriersignalcfg_parameter_struct`

表 3-950. 结构体 `hrtimer_carriersignalcfg_parameter_struct`

成员名称	功能描述
<code>period</code>	载波信号的周期tCSPRD，0x0~0xF， $tCSPRD = (period + 1) * 16 * tHRTIMER_CK$
<code>duty_cycle</code>	载波信号的占空比，0x0~0x7， $duty_cycle = duty_cycle/8$
<code>first_pulse</code>	第一个载波信号的脉冲宽度tCSFSTPW，0x0~0xF， $tCSFSTPW = (first_pulse+1) * 16 * tHRTIMER_CK$

结构体 `hrtimer_synccfg_parameter_struct`

表 3-951. 结构体 `hrtimer_synccfg_parameter_struct`

成员名称	功能描述
<code>input_source</code>	同步输入信号源
<code>output_source</code>	同步输出信号源
<code>output_polarity</code>	同步输出信号极性（ <code>output_source</code> 为有效信号时，该配置生效）

结构体 `hrtimer_bunchmode_parameter_struct`

表 3-952. 结构体 `hrtimer_bunchmode_parameter_struct`

成员名称	功能描述
<code>mode</code>	突发模式运行模式（连续或者单脉冲）
<code>clock_source</code>	突发模式的计数时钟源
<code>prescaler</code>	突发模式的预分频（仅在 <code>clock_source</code> 为 <code>hrtimer_ck</code> 该配置生效）
<code>shadow</code>	<code>HRTIMER_BMCMPV</code> 和 <code>HRTIMER_BMCAR</code> 寄存器的影子寄存器使能
<code>Trigger[2]</code>	突发模式的启动触发事件
<code>idle_duration</code>	空闲时长，0x0000~0xFFFF
<code>period</code>	周期，0x0001~0xFFFF

结构体 `hrtimer_exeventcfg_parameter_struct`

表 3-953. 结构体 `hrtimer_exeventcfg_parameter_struct`

成员名称	功能描述
<code>source</code>	外部事件源
<code>polarity</code>	外部事件极性，（当 <code>edge</code> 为电平有效时该配置生效）
<code>edge</code>	外部事件的有效沿
<code>fast</code>	外部事件快速模式
<code>digital_filter</code>	外部事件的数字滤波，0x0~0xF

结构体 `hrtimer_exeventcnt_parameter_struct`

表 3-954. 结构体 `hrtimer_exeventcnt_parameter_struct`

成员名称	功能描述
<code>reset_mode</code>	外部事件 X 复位模式
<code>counter_threshold</code>	外部事件 X 阈值，0x00~0x3F
<code>event_source</code>	外部事件 X 触发源选择

结构体 `hrtimer_faultcfg_parameter_struct`

表 3-955. 结构体 `hrtimer_faultcfg_parameter_struct`

成员名称	功能描述
<code>source</code>	故障输入源
<code>polarity</code>	故障输入极性

filter	故障输入数字滤波, 0x0~0xF
control	使能或禁能故障输入
protect	保护故障输入配置
blanksource	故障消隐源
counter	故障消隐计数值
resetmode	故障消隐计数值复位模式

结构体 `hrtimer_adctrigcfg_parameter_struct`

表 3-956. 结构体 `hrtimer_adctrigcfg_parameter_struct`

成员名称	功能描述
update_source	HRTIMER_ADCTRIGSy (y=0..3) 寄存器的更新源
Trigger0_3[2]	ADC触发源0~3
Trigger4_9	ADC触发源4~9

结构体 `hrtimer_channel_outputcfg_parameter_struct`

表 3-957. 结构体 `hrtimer_channel_outputcfg_parameter_struct`

成员名称	功能描述
polarity	通道输出极性
set_request	产生通道输出置位请求的事件
reset_request	产生通道输出复位请求的事件
idle_bunch	通道是否受突发模式控制
idle_state	通道输出的空闲状态
fault_state	故障时通道输出状态
carrier_mode	使能或禁能载波模式
deadtime_bunch	在突发模式进入IDLE状态前是否插入死区时间

结构体 `hrtimer_roll_over_parameter_struct`

表 3-958. 结构体 `hrtimer_roll_over_parameter_struct`

成员名称	功能描述
roll_over_mode	翻转模式
output_roll_over_mode	输出翻转模式
adc_roll_over_mode	ADC触发翻转模式
bunch_mode_roll_over_mode	突发翻转模式
fault_event_roll_over_mode	故障和事件翻转模式

结构体 `hrtimer_double_trigger_parameter_struct`表 3-959. 结构体 `hrtimer_double_trigger_parameter_struct`

成员名称	功能描述
<code>trigger_enable</code>	配置双通道触发使能
<code>trigger0</code>	配置双通道触发源0
<code>Trigger1</code>	配置双通道触发源1

函数 `hrtimer_deinit`

函数 `hrtimer_deinit` 描述见下表：

表 3-960. 函数 `hrtimer_deinit`

函数名称	<code>hrtimer_deinit</code>
函数原型	<code>void hrtimer_deinit(uint32_t hrtimer_periph);</code>
功能描述	复位外设 HRTIMER
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>hrtimer_periph</code>	HRTIMER 外设
<code>HRTIMERx(x=0)</code>	选择一个 HRTIMER 外设
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* reset HRTIMER */
```

```
hrtimer_deinit(HRTIMER0);
```

函数 `hrtimer_dll_calibration_start`

函数 `hrtimer_dll_calibration_start` 描述见下表：

表 3-961. 函数 `hrtimer_dll_calibration_start`

函数名称	<code>hrtimer_dll_calibration_start</code>
函数原型	<code>void hrtimer_dll_calibration_start(uint32_t hrtimer_periph);</code>
功能描述	配置和启动 DLL 校准
先决条件	-
被调用函数	-
输入参数{in}	
<code>hrtimer_periph</code>	HRTIMER 外设
<code>HRTIMERx(x=0)</code>	选择一个 HRTIMER 外设
输入参数{in}	

calform	指定校准的方式
<i>HRTIMER_CALIBRATION_ONCE</i>	只校准一次
<i>HRTIMER_CALIBRATION_1048576_PERIOD</i>	周期性校准，校准周期为 $1048576 * t_{HRTIMER_CK}$
<i>HRTIMER_CALIBRATION_131072_PERIOD</i>	周期性校准，校准周期为 $131072 * t_{HRTIMER_CK}$
<i>HRTIMER_CALIBRATION_16384_PERIOD</i>	周期性校准，校准周期为 $16384 * t_{HRTIMER_CK}$
<i>HRTIMER_CALIBRATION_2048_PERIOD</i>	周期性校准，校准周期为 $2048 * t_{HRTIMER_CK}$
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* configure and start DLL calibration */
```

```
hrtimer_dll_calibration_start(HRTIMER0, HRTIMER_CALIBRATION_ONCE);
```

函数 `hrtimer_baseinit_struct_para_init`

函数 `hrtimer_baseinit_struct_para_init` 描述见下表：

表 3-962. 函数 `hrtimer_baseinit_struct_para_init`

函数名称	<code>hrtimer_baseinit_struct_para_init</code>
函数原型	<code>void hrtimer_baseinit_struct_para_init(hrtimer_baseinit_parameter_struct* baseinit);</code>
功能描述	初始化 <code>hrtimer_baseinit_parameter_struct</code> 结构体变量
先决条件	-
被调用函数	-
输入参数{in}	
baseinit	<code>hrtimer_baseinit_parameter_struct</code> 结构体指针，该结构体成员变量参考 表 3-943. 结构体 <code>hrtimer_baseinit_parameter_struct</code>
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* initialize HRTIMER time base parameters struct with the default value */
```

```
hrtimer_baseinit_parameter_struct baseinit;
```

```
hrtimer_baseinit_struct_para_init(&baseinit);
```

函数 hrtimer_timers_base_init

函数hrtimer_timers_base_init描述见下表：

表 3-963. 函数 hrtimer_timers_base_init

函数名称	hrtimer_timers_base_init
函数原型	void hrtimer_timers_base_init(uint32_t hrtimer_periph, uint32_t timer_id, hrtimer_baseinit_parameter_struct* baseinit);
功能描述	初始化Master_TIMER/Slave_TIMER时基
先决条件	-
被调用函数	-
输入参数{in}	
hrtimer_periph	HRTIMER外设
HRTIMERx(x=0)	选择一个HRTIMER外设
输入参数{in}	
timer_id	Master_TIMER和Slave_TIMER索引
HRTIMER_MASTER_TIMER	选择Master_TIMER
HRTIMER_SLAVE_TIMERx	选择Slave_TIMER (x=0..7)
输入参数{in}	
baseinit	hrtimer_baseinit_parameter_struct结构体指针，该结构体成员变量参考 表 3-943. 结构体hrtimer_baseinit_parameter_struct
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* initialize Master_TIMER and Slave_TIMER timerbase */

hrtimer_baseinit_parameter_struct baseinit_para;

hrtimer_baseinit_struct_para_init(&baseinit_para);

baseinit_para.period = 384;

baseinit_para.prescaler = HRTIMER_PRESCALER_MUL32;

baseinit_para.repetitioncounter = 0;

baseinit_para.Counterdirection = HRTIMER_COUNTER_UP;

baseinit_para.counter_mode = HRTIMER_COUNTER_MODE_CONTINUOUS;

hrtimer_timers_base_init(HRTIMER0, HRTIMER_SLAVE_TIMER0, &baseinit_para);
```

函数 `hrtimer_timers_counter_enable`

函数 `hrtimer_timers_counter_enable` 描述见下表：

表 3-964. 函数 `hrtimer_timers_counter_enable`

函数名称	<code>hrtimer_timers_counter_enable</code>
函数原型	<code>void hrtimer_timers_counter_enable(uint32_t hrtimer_periph, uint32_t cntid);</code>
功能描述	使能定时器的计数器
先决条件	-
被调用函数	-
输入参数{in}	
<code>hrtimer_periph</code>	HRTIMER外设
<code>HRTIMERx(x=0)</code>	选择一个HRTIMER外设
输入参数{in}	
<code>cntid</code>	指定计数器
<code>HRTIMER_MT_COUNTER</code>	Master_TIMER的计数器
<code>HRTIMER_STx_COUNTER(x=0..7)</code>	Slave_TIMERx(x=0..7)的计数器
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* enable a counter */
```

```
void hrtimer_timers_counter_enable(HRTIMER0, HRTIMER_MT_COUNTER);
```

函数 `hrtimer_timers_counter_disable`

函数 `hrtimer_timers_counter_disable` 描述见下表：

表 3-965. 函数 `hrtimer_timers_counter_disable`

函数名称	<code>hrtimer_timers_counter_disable</code>
函数原型	<code>void hrtimer_timers_counter_disable(uint32_t hrtimer_periph, uint32_t cntid);</code>
功能描述	禁能定时器的计数器
先决条件	-
被调用函数	-
输入参数{in}	
<code>hrtimer_periph</code>	HRTIMER外设
<code>HRTIMERx(x=0)</code>	选择一个HRTIMER外设
输入参数{in}	
<code>cntid</code>	指定计数器

HRTIMER_MT_COUNTER	Master_TIMER的计数器
HRTIMER_STx_COUNTER(x=0..7)	Slave_TIMERx(x=0..7)的计数器
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* disable a counter */
```

```
hrtimer_timers_counter_disable(HRTIMER0, HRTIMER_MT_COUNTER);
```

函数 hrtimer_timers_update_event_enable

函数hrtimer_timers_update_event_enable描述见下表:

表 3-966. 函数 hrtimer_timers_update_event_enable

函数名称	hrtimer_timers_update_event_enable
函数原型	void hrtimer_timers_update_event_enable(uint32_t hrtimer_periph, uint32_t timer_id);
功能描述	使能Master_TIMER/Slave_TIMER更新事件
先决条件	-
被调用函数	-
输入参数{in}	
hrtimer_periph	HRTIMER外设
HRTIMERx(x=0)	选择一个HRTIMER外设
输入参数{in}	
timer_id	选择HRTIMER中的一个定时器
HRTIMER_MASTER_TIMER	选择Master_TIMER
HRTIMER_SLAVE_TIMERx(x=0..7)	选择Slave_TIMERx(x=0..7)
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* enable the Master_TIMER or Slave_TIMER update event */
```

```
hrtimer_timers_update_event_enable(HRTIMER0, HRTIMER_MASTER_TIMER);
```

函数 `hrtimer_timers_update_event_disable`

函数 `hrtimer_timers_update_event_disable` 描述见下表：

表 3-967. 函数 `hrtimer_timers_update_event_enable`

函数名称	<code>hrtimer_timers_update_event_disable</code>
函数原型	<code>void hrtimer_timers_update_event_disable(uint32_t hrtimer_periph, uint32_t timer_id);</code>
功能描述	禁用Master_TIMER/Slave_TIMER更新事件
先决条件	-
被调用函数	-
输入参数{in}	
<code>hrtimer_periph</code>	HRTIMER外设
<code>HRTIMERx(x=0)</code>	选择一个HRTIMER外设
输入参数{in}	
<code>timer_id</code>	选择HRTIMER中的一个定时器
<code>HRTIMER_MASTER_TIMER</code>	选择Master_TIMER
<code>HRTIMER_SLAVE_TIMERx(x=0..7)</code>	选择Slave_TIMERx(x=0..7)
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* disable the Master_TIMER or Slave_TIMER update event */
```

```
hrtimer_timers_update_event_disable(HRTIMER0, HRTIMER_MASTER_TIMER);
```

函数 `hrtimer_software_update`

函数 `hrtimer_software_update` 描述见下表：

表 3-968. 函数 `hrtimer_software_update`

函数名称	<code>hrtimer_software_update</code>
函数原型	<code>void hrtimer_software_update(uint32_t hrtimer_periph, uint32_t timersrc);</code>
功能描述	软件触发Master_TIMER与Slave_TIMER更新事件
先决条件	-
被调用函数	-
输入参数{in}	
<code>hrtimer_periph</code>	HRTIMER外设
<code>HRTIMERx(x=0)</code>	选择一个HRTIMER外设
输入参数{in}	
<code>timersrc</code>	选择软件触发的定时器

<i>HRTIMER_UPDATE_SW_MT</i>	软件触发Master_TIMER
<i>HRTIMER_UPDATE_SW_STx(x=0..7)</i>	软件触发Slave_TIMERx (x=0..7)
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* update the Master_TIMER or Slave_TIMER by software */
```

```
void hrtimer_software_update(HRTIMER0, HRTIMER_UPDATE_SW_MT);
```

函数 **hrtimer_software_counter_reset**

函数hrtimer_software_counter_reset描述见下表:

表 3-969. 函数 hrtimer_software_counter_reset

函数名称	hrtimer_software_counter_reset
函数原型	void hrtimer_software_counter_reset(uint32_t hrtimer_periph, uint32_t timerrst);
功能描述	软件复位Master_TIMER/Slave_TIMER计数器
先决条件	-
被调用函数	-
输入参数{in}	
hrtimer_periph	HRTIMER外设
<i>HRTIMERx(x=0)</i>	选择一个HRTIMER外设
输入参数{in}	
timersrc	选择软件复位的定时器
<i>HRTIMER_COUNTER_RESET_SW_MT</i>	软件复位Master_TIMER
<i>HRTIMER_COUNTER_RESET_SW_STx(x=0..7)</i>	软件复位Slave_TIMERx (x=0..7)
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* triggers the Master_TIMER or Slave_TIMER registers update by software */
```

```
void hrtimer_software_counter_reset(HRTIMER0, HRTIMER_COUNTER_RESET_SW_MT);
```


函数 `hrtimer_output_exchange`

函数 `hrtimer_output_exchange` 描述见下表：

表 3-970. 函数 `hrtimer_output_exchange`

函数名称	<code>hrtimer_output_exchange</code>
函数原型	<code>void hrtimer_output_exchange(uint32_t hrtimer_periph, uint32_t timerexc);</code>
功能描述	交换Slave_TIMERx通道输出
先决条件	-
被调用函数	-
输入参数{in}	
<code>hrtimer_periph</code>	HRTIMER外设
<code>HRTIMERx(x=0)</code>	选择一个HRTIMER外设
输入参数{in}	
<code>timersrc</code>	选择交换通道输出的定时器
<code>HRTIMER_OUTPUT_EXCHANGE_STx</code> ($x=0..7$)	Slave_TIMERx ($x=0..7$)交换通道
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* exchange Slave_TIMER0 outputs */
```

```
void hrtimer_output_exchange(HRTIMER0,
HRTIMER_HRTIMER_OUTPUT_EXCHANGE_ST0);
```

函数 `hrtimer_timerinit_struct_para_init`

函数 `hrtimer_timerinit_struct_para_init` 描述见下表：

表 3-971. 函数 `hrtimer_timerinit_struct_para_init`

函数名称	<code>hrtimer_timerinit_struct_para_init</code>
函数原型	<code>void hrtimer_timerinit_struct_para_init(hrtimer_timerinit_parameter_struct* timerinit);</code>
功能描述	初始化 <code>hrtimer_timerinit_parameter_struct</code> 结构体变量
先决条件	-
被调用函数	-
输入参数{in}	
<code>timerinit</code>	<code>hrtimer_timerinit_parameter_struct</code> 结构体指针，结构体成员变量参考 表 3-944. 结构体 <code>hrtimer_timerinit_parameter_struct</code>
输出参数{out}	
-	-

返回值	
-	-

例如：

```
/* initialize waveform mode initialization parameters struct with a default value */
```

```
hrtimer_timerinit_parameter_struct timerinit_para;
```

```
hrtimer_timerinit_struct_para_init(&timerinit_para);
```

函数 hrtimer_timers_waveform_init

函数hrtimer_timers_waveform_init描述见下表：

表 3-972. 函数 hrtimer_timers_waveform_init

函数名称	hrtimer_timers_waveform_init
函数原型	void hrtimer_timers_waveform_init(uint32_t hrtimer_periph, uint32_t timer_id, hrtimer_timerinit_parameter_struct* timerinitpara);
功能描述	定时器波形初始化
先决条件	-
被调用函数	master_timer_waveform_config / slave_timer_waveform_config
输入参数{in}	
hrtimer_periph	HRTIMER外设
HRTIMERx(x=0)	选择一个HRTIMER外设
输入参数{in}	
timer_id	选择HRTIMER中的一个定时器
HRTIMER_MASTER_TIMER	选择Master_TIMER
HRTIMER_SLAVE_TIMERx(x=0..7)	选择Slave_TIMERx(x=0..7)
输入参数{in}	
timerinit	hrtimer_timerinit_parameter_struct结构体指针，结构体成员变量参考 表 3-944. 结构体hrtimer_timerinit_parameter_struct
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* initialize a timer to work in waveform mode */
```

```
hrtimer_timerinit_parameter_struct timerinit_para;
```

```
hrtimer_timerinit_struct_para_init(&timerinit_para);
```

```
timerinit_para.cnt_bunch = HRTIMER_TIMERBUNCHMODE_MAINTAINCLOCK;
```

```

timerinit_para.DAC_trigger = HRTIMER_DAC_TRIGGER_NONE;

timerinit_para.half_mode = HRTIMER_HALFMODE_DISABLED;

timerinit_para.alternate_mode = HRTIMER_ALTERNATE_MODE_DISABLED;

timerinit_para.repetition_update = HRTIMER_UPDATEONREPETITION_DISABLED;

timerinit_para.reset_sync = HRTIMER_SYNCRESET_DISABLED;

timerinit_para.shadow = HRTIMER_SHADOW_DISABLED;

timerinit_para.start_sync = HRTIMER_SYNISTART_DISABLED;

timerinit_para.update_selection                                =
HRTIMER_MT_ST_UPDATE_SELECTION_INDEPENDENT;

timerinit_para.resynchronized_update = HRTIMER_RSYUPD_DISABLED;

hrtimer_timers_waveform_init(HRTIMER0, HRTIMER_SLAVE_TIMER0, &timerinit_para);

```

函数 hrtimer_timercfg_struct_para_init

函数hrtimer_timercfg_struct_para_init描述见下表:

表 3-973. 函数 hrtimer_timercfg_struct_para_init

函数名称	hrtimer_timercfg_struct_para_init		
函数原型	void hrtimer_timercfg_struct_para_init(hrtimer_timercfg_parameter_struct* timercfg);		
功能描述	初始化hrtimer_timercfg_parameter_struct结构体变量		
先决条件	-		
被调用函数	-		
输入参数{in}			
timercfg	hrtimer_timercfg_parameter_struct结构体指针，结构体成员参考		表 3-945. 结构体hrtimer_timercfg_parameter_struct
输出参数{out}			
-	-		
返回值			
-	-		

例如:

```

/* initialize Slave_TIMER general behavior configuration struct with a default value */

hrtimer_timercfg_parameter_struct timercfg_para;

hrtimer_timercfg_struct_para_init(timercfg_para);

```

函数 hrtimer_slavetimer_waveform_config

函数hrtimer_slavetimer_waveform_config描述见下表:

表 3-974. 函数 `hrtimer_slavetimer_waveform_config`

函数名称	<code>hrtimer_slavetimer_waveform_config</code>
函数原型	<code>void hrtimer_slavetimer_waveform_config(uint32_t hrtimer_periph, uint32_t timer_id, hrtimer_timercfg_parameter_struct * timercfg);</code>
功能描述	配置Slave_TIMER的波形
先决条件	-
被调用函数	-
输入参数{in}	
hrtimer_periph	HRTIMER外设
<i>HRTIMERx(x=0)</i>	选择一个HRTIMER外设
输入参数{in}	
timer_id	Master_TIMER和Slave_TIMER索引
<i>HRTIMER_MASTER_TIMER</i>	选择Master_TIMER
<i>HRTIMER_SLAVE_TIMER(x=0..7)</i>	选择Slave_TIMER (x=0..7)
输入参数{in}	
timercfg	<code>hrtimer_timercfg_parameter_struct</code> 结构体指针, 结构体成员参考 表 3-945. 结构体 <code>hrtimer_timercfg_parameter_struct</code>
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* initialize Slave_TIMER general behavior configuration struct with a default value */
```

```
hrtimer_timercfg_parameter_struct timercfg_para;
```

```
hrtimer_timercfg_struct_para_init(&timercfg_para);
```

```
timercfg_para.balanced_mode = HRTIMER_STXBALANCEDMODE_DISABLED;
```

```
timercfg_para.balanced_idle_automatic_resumption =  
HRTIMER_BALANE_IDLE_AUTOMATIC_RESUME_DISABLE;
```

```
timercfg_para.cnt_reset = HRTIMER_STXCNT_RESET_NONE;
```

```
timercfg_para.deadtime_enable = HRTIMER_STXDEADTIME_DISABLED;
```

```
timercfg_para.delayed_idle = HRTIMER_STXDELAYED_IDLE_DISABLED;
```

```
timercfg_para.fault_enable = HRTIMER_STXFAULTENABLE_NONE;
```

```
timercfg_para.fault_protect = HRTIMER_STXFAULT_PROTECT_READWRITE;
```

```
timercfg_para.reset_update = HRTIMER_STXUPDATEONRESET_DISABLED;
```

```
timercfg_para.update_source = HRTIMER_STXUPDATETRIGGER_NONE;
```

```
hrtimer_slavetimer_waveform_config(HRTIMER0, HRTIMER_SLAVE_TIMER0,  
&timercfg_para);
```

函数 hrtimer_comparecfg_struct_para_init

函数hrtimer_comparecfg_struct_para_init描述见下表：

表 3-975. 函数 hrtimer_comparecfg_struct_para_init

函数名称	hrtimer_comparecfg_struct_para_init
函数原型	void hrtimer_comparecfg_struct_para_init(hrtimer_comparecfg_parameter_struct * comparecfg);
功能描述	初始化hrtimer_comparecfg_parameter_struct结构体
先决条件	-
被调用函数	-
输入参数{in}	
comparecfg	hrtimer_comparecfg_parameter_struct结构体指针，结构体成员参考 表 3-947. 结构体hrtimer_comparecfg_parameter_struct
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* initialize compare unit configuration struct with a default value */
```

```
hrtimer_comparecfg_parameter_struct comparecfg_para;
```

```
hrtimer_comparecfg_struct_para_init(&comparecfg_para);
```

函数 hrtimer_slavetimer_waveform_compare_config

函数hrtimer_slavetimer_waveform_compare_config描述见下表：

表 3-976. 函数 hrtimer_slavetimer_waveform_compare_config

函数名称	hrtimer_slavetimer_waveform_compare_config
函数原型	void hrtimer_slavetimer_waveform_compare_config(uint32_t hrtimer_periph, uint32_t timer_id, uint32_t comparex, hrtimer_comparecfg_parameter_struct* cmpcfg);
功能描述	配置Slave_TIMER波形的比较功能
先决条件	-
被调用函数	-
输入参数{in}	
hrtimer_periph	HRTIMER外设

<i>HRTIMERx(x=0)</i>	选择一个HRTIMER外设
输入参数{in}	
timer_id	选择HRTIMER中的一个定时器
<i>HRTIMER_SLAVE_TIMERx(x=0..7)</i>	选择Slave_TIMERx(x=0..7)
输入参数{in}	
comparex	选择一个比较单元
<i>HRTIMER_COMPAREy(y=0..3)</i>	比较单元y(y=0..3)
输入参数{in}	
comparecfg	hrtimer_comparecfg_parameter_struct结构体指针，结构体成员参考 表 3-947. 结构体hrtimer_comparecfg_parameter_struct
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* configure the compare unit of a Slave_TIMER which work in waveform mode */
hrtimer_comparecfg_parameter_struct comparecfg_para;
hrtimer_comparecfg_struct_para_init(&comparecfg_para);
comparecfg_para.compare_value = 192;

hrtimer_slavetimer_waveform_compare_config(HRTIMER0, HRTIMER_SLAVE_TIMER0,
HRTIMER_COMPARE0, &comparecfg_para);
```

函数 hrtimer_channel_outputcfg_struct_para_init

函数hrtimer_channel_outputcfg_struct_para_init描述见下表：

表 3-977. 函数 hrtimer_channel_outputcfg_struct_para_init

函数名称	hrtimer_channel_outputcfg_struct_para_init
函数原型	void hrtimer_channel_outputcfg_struct_para_init(hrtimer_channel_outputcfg_parameter_struct * channelcfg);
功能描述	初始化hrtimer_channel_outputcfg_parameter_struct结构体变量
先决条件	-
被调用函数	-
输入参数{in}	
channelcfg	hrtimer_channel_outputcfg_parameter_struct结构体指针，结构体成员参考 表3-957. 结构体hrtimer_channel_outputcfg_parameter_struct
输出参数{out}	

-	-
返回值	
-	-

例如：

```
/* initialize channel output configuration struct with a default value */
hrtimer_channel_outputcfg_parameter_struct outcfg_para;
hrtimer_channel_outputcfg_struct_para_init(&outcfg_para);
```

函数 hrtimer_slavetimer_waveform_channel_config

函数hrtimer_slavetimer_waveform_channel_config描述见下表：

表 3-978. 函数 hrtimer_slavetimer_waveform_channel_config

函数名称	hrtimer_slavetimer_waveform_channel_config
函数原型	void hrtimer_slavetimer_waveform_channel_config(uint32_t hrtimer_periph, uint32_t timer_id, uint32_t channel, hrtimer_channel_outputcfg_parameter_struct * channelcfg);
功能描述	Slave_TIMER波形的通道配置
先决条件	-
被调用函数	channel_output_config
输入参数{in}	
hrtimer_periph	HRTIMER外设
HRTIMERx(x=0)	选择一个HRTIMER外设
输入参数{in}	
timer_id	选择HRTIMER中的一个定时器
HRTIMER_SLAVE_TIMERx(x=0..7)	选择Slave_TIMERx(x=0..7)
输入参数{in}	
channel	HRTIMER通道选择
HRTIMER_STx_CHy(x=0..7,y=0,1)	选择一个通道
channelcfg	
channelcfg	hrtimer_channel_outputcfg_parameter_struct结构体指针，结构体成员参考 表3-957. 结构体hrtimer_channel_outputcfg_parameter_struct
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* configure the channel of a Slave_TIMER work in waveform mode */
```

```

hrtimer_channel_outputcfg_parameter_struct outcfg_para;

hrtimer_channel_outputcfg_struct_para_init(&outcfg_para);

outcfg_para.carrier_mode = HRTIMER_CHANNEL_CARRIER_DISABLED;

outcfg_para.deadtime_bunch = HRTIMER_CHANNEL_BUNCH_ENTRY_REGULAR;

outcfg_para.fault_state = HRTIMER_CHANNEL_FAULTSTATE_NONE;

outcfg_para.idle_bunch = HRTIMER_CHANNEL_BUNCH_IDLE_DISABLE;

outcfg_para.idle_state = HRTIMER_CHANNEL_IDLESTATE_INACTIVE;

outcfg_para.polarity = HRTIMER_CHANNEL_POLARITY_HIGH;

outcfg_para.reset_request = HRTIMER_CHANNEL_RESET_CMP1;

outcfg_para.set_request = HRTIMER_CHANNEL_SET_CMP0;

hrtimer_slavetimer_waveform_channel_config(HRTIMER0, HRTIMER_SLAVE_TIMER0,
HRTIMER_ST0_CH0, &outcfg_para);

```

函数 hrtimer_slavetimer_waveform_channel_software_request

函数hrtimer_slavetimer_waveform_channel_software_request描述见下表:

表 3-979. 函数 hrtimer_slavetimer_waveform_channel_software_request

函数名称	hrtimer_slavetimer_waveform_channel_software_request
函数原型	void hrtimer_slavetimer_waveform_channel_software_request(uint32_t hrtimer_periph, uint32_t timer_id, uint32_t channel, uint32_t request)
功能描述	软件产生通道置位或复位请求
先决条件	-
被调用函数	-
输入参数{in}	
hrtimer_periph	HRTIMER外设
HRTIMERx(x=0)	选择一个HRTIMER外设
输入参数{in}	
timer_id	选择HRTIMER中的一个定时器
HRTIMER_SLAVE_TIMERx(x=0..7)	选择Slave_TIMERx(x=0..7)
输入参数{in}	
channel	HRTIMER通道选择
HRTIMER_STx_CHy(x=0..7,y=0,1)	选择一个通道
输入参数{in}	
request	置位请求
HRTIMER_CHANNEL_SOFTWARE_SET	软件产生置位请求

HRTIMER_CHANNEL_SOFTWARE_RESET	软件产生复位请求
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* software generates channel "set request" or "reset request" */
```

```
hrtimer_slavetimer_waveform_channel_software_request (HRTIMER0,
HRTIMER_SLAVE_TIMER0, HRTIMER_ST0_CH0,
HRTIMER_CHANNEL_SOFTWARE_SET);
```

函数 hrtimer_slavetimer_waveform_channel_output_level_get

函数hrtimer_slavetimer_waveform_channel_output_level_get描述见下表：

表 3-980. 函数 hrtimer_slavetimer_waveform_channel_output_level_get

函数名称	hrtimer_slavetimer_waveform_channel_output_level_get
函数原型	uint32_t hrtimer_slavetimer_waveform_channel_output_level_get(uint32_t hrtimer_periph, uint32_t timer_id, uint32_t channel);
功能描述	获取通道的输出电平
先决条件	-
被调用函数	-
输入参数{in}	
hrtimer_periph	HRTIMER外设
HRTIMERx(x=0)	选择一个HRTIMER外设
输入参数{in}	
timer_id	选择HRTIMER中的一个定时器
HRTIMER_SLAVE_TIMERx(x=0..7)	选择Slave_TIMERx(x=0..7)
输入参数{in}	
channel	HRTIMER通道选择
HRTIMER_STx_CHy(x=0..7,y=0,1)	选择一个通道
输出参数{out}	
-	-
返回值	
uint32_t	通道输出电平

例如：

```
/* get Slave_TIMER channel output level */
```

```
uint32_t output_level;
```

```
output_level = hrtimer_slavetimer_waveform_channel_output_level_get(uint32_t
hrtimer_periph, uint32_t timer_id, uint32_t channel)
```

函数 hrtimer_slavetimer_waveform_channel_state_get

函数hrtimer_slavetimer_waveform_channel_state_get描述见下表:

表 3-981. 函数 hrtimer_slavetimer_waveform_channel_state_get

函数名称	hrtimer_slavetimer_waveform_channel_state_get
函数原型	uint32_t hrtimer_slavetimer_waveform_channel_state_get(uint32_t hrtimer_periph, uint32_t timer_id, uint32_t channel)
功能描述	获取通道的运行状态
先决条件	-
被调用函数	-
输入参数{in}	
hrtimer_periph	HRTIMER外设
HRTIMERx(x=0)	选择一个HRTIMER外设
输入参数{in}	
timer_id	选择HRTIMER中的一个定时器
HRTIMER_SLAVE_TIMERx(x=0..7)	选择Slave_TIMERx(x=0..7)
输入参数{in}	
channel	HRTIMER通道选择
HRTIMER_STx_CHy(x=0..7,y=0,1)	选择一个通道
输出参数{out}	
-	-
返回值	
uint32_t	HRTIMER_CHANNEL_STATE_IDLE, HRTIMER_CHANNEL_STATE_RUN, HRTIMER_CHANNEL_STATE_FAULT

例如:

```
/* get Slave_TIMER channel run state */
```

```
uint32_t output_state;
```

```
output_state = hrtimer_slavetimer_waveform_channel_state_get (HRTIMER0,  
HRTIMER_SLAVE_TIMER0, HRTIMER_ST0_CH0);
```

函数 hrtimer_deadtimercfg_struct_para_init

函数hrtimer_deadtimercfg_struct_para_init描述见下表:

表 3-982. 函数 `hrtimer_channel_outputcfg_struct_para_init`

函数名称	<code>hrtimer_deadtimercfg_struct_para_init</code>
函数原型	<code>void hrtimer_deadtimercfg_struct_para_init(hrtimer_deadtimercfg_parameter_struct * dtcfg);</code>
功能描述	初始化 <code>hrtimer_deadtimercfg_parameter_struct</code> 结构体变量
先决条件	-
被调用函数	-
输入参数{in}	
dtcfg	<code>hrtimer_deadtimercfg_parameter_struct</code> 结构体指针, 结构体成员参考 表 3-949. 结构体 <code>hrtimer_deadtimercfg_parameter_struct</code> 结构体 <code>hrtimer_deadtimercfg_parameter_struct</code>
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* initialize dead time configuration struct with a default value */
```

```
hrtimer_deadtimercfg_parameter_struct deadtimercfg_para;
```

```
hrtimer_deadtimercfg_struct_para_init(&deadtimercfg_para);
```

函数 `hrtimer_slavetimer_deadtime_config`

函数 `hrtimer_slavetimer_deadtime_config` 描述见下表:

表 3-983. 函数 `hrtimer_slavetimer_waveform_channel_software_request`

函数名称	<code>hrtimer_slavetimer_deadtime_config</code>
函数原型	<code>void hrtimer_slavetimer_deadtime_config(uint32_t hrtimer_periph, uint32_t timer_id, hrtimer_deadtimercfg_parameter_struct* dtcfg)</code>
功能描述	配置 Slave_TIMER 的死区时间
先决条件	-
被调用函数	-
输入参数{in}	
hrtimer_periph	HRTIMER 外设
<code>HRTIMERx(x=0)</code>	选择一个 HRTIMER 外设
输入参数{in}	
timer_id	选择 HRTIMER 中的一个定时器
<code>HRTIMER_SLAVE_TIMERx(x=0..7)</code>	选择 Slave_TIMERx(x=0..7)
输入参数{in}	
dtcfg	<code>hrtimer_deadtimercfg_parameter_struct</code> 结构体指针, 结构体成员参考 表 3-949. 结构体 <code>hrtimer_deadtimercfg_parameter_struct</code> 结构体

	3-949. 结构体hrtimer_deadtimecfg_parameter_struct 结构体 hrtimer_deadtimecfg_parameter_struct
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* configure the dead time for Slave_TIMER */

hrtimer_deadtimecfg_parameter_struct deadtimecfg_para;

hrtimer_deadtimecfg_struct_para_init(&deadtimecfg_para);

deadtimecfg_para.fallingsign_protect                                =
HRTIMER_DEADTIME_FALLINGSIGN_PROTECT_DISABLE;

deadtimecfg_para.falling_protect = HRTIMER_DEADTIME_FALLING_PROTECT_DISABLE;

deadtimecfg_para.falling_sign = HRTIMER_DEADTIME_FALLINGSIGN_POSITIVE;

deadtimecfg_para.falling_value = 0x0040;

deadtimecfg_para.prescaler = HRTIMER_DEADTIME_PRESCALER_MUL8;

deadtimecfg_para.risingsign_protect                                =
HRTIMER_DEADTIME_RISINGSIGN_PROTECT_DISABLE;

deadtimecfg_para.rising_protect = HRTIMER_DEADTIME_RISING_PROTECT_DISABLE;

deadtimecfg_para.rising_sign = HRTIMER_DEADTIME_RISINGSIGN_POSITIVE;

deadtimecfg_para.rising_value = 0x0040;

hrtimer_slavetimer_deadtime_config(HRTIMER0, HRTIMER_SLAVE_TIMER0,
&deadtimecfg_para);
```

函数 hrtimer_carriersignalcfg_struct_para_init

函数hrtimer_carriersignalcfg_struct_para_init描述见下表：

表 3-984. 函数 hrtimer_channel_outputcfg_struct_para_init

函数名称	hrtimer_carriersignalcfg_struct_para_init
函数原型	void hrtimer_carriersignalcfg_struct_para_init(hrtimer_carriersignalcfg_paramete r_struct* carriercfg);
功能描述	初始化hrtimer_carriersignalcfg_parameter_struct结构体变量
先决条件	-
被调用函数	-
输入参数{in}	

carriercfg	hrtimer_carriersignalcfg_parameter_struct结构体指针，结构体成员参考 表3-950. 结构体hrtimer_carriersignalcfg_parameter_struct 结构体 hrtimer_carriersignalcfg_parameter_struct
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* initialize carrier signal configuration struct with a default value */
```

```
hrtimer_carriersignalcfg_parameter_struct carriercfg;
```

```
hrtimer_carriersignalcfg_struct_para_init(&carriercfg);
```

函数 hrtimer_slavetimer_carriersignal_config

函数hrtimer_slavetimer_carriersignal_config描述见下表：

表 3-985. 函数 hrtimer_slavetimer_carriersignal_config

函数名称	hrtimer_slavetimer_carriersignal_config
函数原型	void hrtimer_slavetimer_carriersignal_config(uint32_t hrtimer_periph, uint32_t timer_id, hrtimer_carriersignalcfg_parameter_struct* carriercfg);
功能描述	配置Slave_TIMER的载波功能
先决条件	-
被调用函数	-
输入参数{in}	
hrtimer_periph	HRTIMER外设
<i>HRTIMERx(x=0)</i>	选择一个HRTIMER外设
输入参数{in}	
timer_id	选择HRTIMER中的一个定时器
<i>HRTIMER_SLAVE_TIMERx(x=0..7)</i>	选择Slave_TIMERx(x=0..7)
输入参数{in}	
carriercfg	hrtimer_carriersignalcfg_parameter_struct结构体指针，结构体成员参考 表3-950. 结构体hrtimer_carriersignalcfg_parameter_struct
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* configure the carrier signal mode for Slave_TIMER */
```

```
hrtimer_carriersignalcfg_parameter_struct carriercfg;
```

```
hrtimer_carriersignalcfg_struct_para_init(&carriercfg);
```

```
hrtimer_slavetimer_carriersignal_config(HRTIMER0, HRTIMER_SLAVE_TIMER0,  
&carriercfg);
```

函数 hrtimer_output_channel_enable

函数hrtimer_output_channel_enable描述见下表：

表 3-986. 函数 hrtimer_output_channel_enable

函数名称	hrtimer_output_channel_enable
函数原型	void hrtimer_output_channel_enable(uint32_t hrtimer_periph, uint32_t chid);
功能描述	使能通道输出
先决条件	-
被调用函数	-
输入参数{in}	
hrtimer_periph	HRTIMER外设
HRTIMERx(x=0)	选择一个HRTIMER外设
输入参数{in}	
ch_id	HRTIMER通道
HRTIMER_STx_CHy(x=0..7;y=0,1)	选择一个通道
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* enable a output channel */
```

```
void hrtimer_output_channel_enable(HRTIMER0, HRTIMER_ST0_CH0);
```

函数 hrtimer_output_channel_disable

函数hrtimer_output_channel_disable描述见下表：

表 3-987. 函数 hrtimer_output_channel_disable

函数名称	hrtimer_output_channel_disable
函数原型	void hrtimer_output_channel_disable(uint32_t hrtimer_periph, uint32_t chid);
功能描述	禁能一个输出通道
先决条件	-
被调用函数	-
输入参数{in}	
hrtimer_periph	HRTIMER外设

<i>HRTIMERx</i> (<i>x</i> =0)	选择一个HRTIMER外设
输入参数{in}	
ch_id	HRTIMER通道
<i>HRTIMER_STx_CHy</i> (<i>x</i> =0..7; <i>y</i> =0,1)	选择一个通道
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* disable a output channel */
```

```
void hrtimer_output_channel_disable(HRTIMER0, HRTIMER_ST0_CH0);
```

函数 `hrtimer_slavetimer_compare_value_config`

函数 `hrtimer_slavetimer_compare_value_config` 描述见下表:

表 3-988. 函数 `hrtimer_slavetimer_waveform_compare_config`

函数名称	<code>hrtimer_slavetimer_compare_value_config</code>
函数原型	<code>void hrtimer_slavetimer_compare_value_config(uint32_t hrtimer_periph, uint32_t timer_id, uint32_t comparex, uint32_t cmpvalue);</code>
功能描述	配置Slave_TIMER的比较寄存器值
先决条件	-
被调用函数	-
输入参数{in}	
hrtimer_periph	HRTIMER外设
<i>HRTIMERx</i> (<i>x</i> =0)	选择一个HRTIMER外设
输入参数{in}	
timer_id	选择HRTIMER中的一个定时器
<i>HRTIMER_SLAVE_TIMERx</i> (<i>x</i> =0..7)	选择Slave_TIMERx(<i>x</i> =0..7)
输入参数{in}	
comparex	比较单元选择
<i>HRTIMER_COMPAREy</i> (<i>y</i> =0..3)	选择比较单元y(<i>y</i> =0..3)
输入参数{in}	
cmpvalue	比较值, 最小值: 3个tHRTIMER_CK, 最大值: 0xFFFF - (1个tHRTIMER_CK)
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* configure the compare value in Slave_TIMER */
```

```
hrtimer_slavetimer_compare_value_config(HRTIMER0, HRTIMER_SLAVE_TIMER0,
HRTIMER_COMPARE0, 3);
```

函数 hrtimer_slavetimer_compare_value_get

函数hrtimer_slavetimer_compare_value_get描述见下表：

表 3-989. 函数 hrtimer_slavetimer_compare_value_get

函数名称	hrtimer_slavetimer_compare_value_get
函数原型	uint32_t hrtimer_slavetimer_compare_value_get(uint32_t hrtimer_periph, uint32_t timer_id, uint32_t comparex)
功能描述	获取Slave_TIMER的比较寄存器值
先决条件	-
被调用函数	-
输入参数{in}	
hrtimer_periph	HRTIMER外设
HRTIMERx(x=0)	选择一个HRTIMER外设
输入参数{in}	
timer_id	选择HRTIMER中的一个定时器
HRTIMER_SLAVE_TIMERx(x=0..7)	选择Slave_TIMERx(x=0..7)
输入参数{in}	
comparex	比较单元选择
HRTIMER_COMPAREy(y=0..3)	选择比较单元y(y=0..3)
输出参数{out}	
-	-
返回值	
uint32_t	比较值

例如：

```
/* get the compare value in Slave_TIMER */
```

```
uint32_t cmpvalue;
```

```
cmpvalue = hrtimer_slavetimer_compare_value_get (HRTIMER0,
HRTIMER_SLAVE_TIMER0, HRTIMER_COMPARE0);
```

函数 hrtimer_mastertimer_compare_value_config

函数hrtimer_mastertimer_compare_value_config描述见下表：

表 3-990. 函数 `hrtimer_mastertimer_compare_value_config`

函数名称	<code>hrtimer_mastertimer_compare_value_config</code>
函数原型	<code>void hrtimer_mastertimer_compare_value_config(uint32_t hrtimer_periph, uint32_t comparex, uint32_t cmpvalue);</code>
功能描述	配置主定时器的比较寄存器值
先决条件	-
被调用函数	-
输入参数{in}	
<code>hrtimer_periph</code>	HRTIMER外设
<code>HRTIMERx(x=0)</code>	选择一个HRTIMER外设
输入参数{in}	
<code>comparex</code>	比较单元选择
<code>HRTIMER_COMPAREy</code> ($y=0..3$)	选择比较单元 $y(y=0..3)$
输入参数{in}	
<code>cmpvalue</code>	比较值，最小值：3个tHRTIMER_CK，最大值：0xFFFF - (1个tHRTIMER_CK)
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* configure the compare value in Master_TIMER */
```

```
hrtimer_mastertimer_compare_value_config(HRTIMER0, HRTIMER_COMPARE0, 3);
```

函数 `hrtimer_mastertimer_compare_value_get`

函数`hrtimer_mastertimer_compare_value_get`描述见下表：

表 3-991. 函数 `hrtimer_slavetimer_compare_value_get`

函数名称	<code>hrtimer_mastertimer_compare_value_get</code>
函数原型	<code>uint32_t hrtimer_mastertimer_compare_value_get(uint32_t hrtimer_periph, uint32_t comparex);</code>
功能描述	获取主定时器的比较寄存器值
先决条件	-
被调用函数	-
输入参数{in}	
<code>hrtimer_periph</code>	HRTIMER外设
<code>HRTIMERx(x=0)</code>	选择一个HRTIMER外设
输入参数{in}	
<code>comparex</code>	比较单元选择
<code>HRTIMER_COMPAREy</code>	选择比较单元 $y(y=0..3)$

(y=0..3)	
输出参数{out}	
-	-
返回值	
uint32_t	比较值

例如:

```
/* get the compare value in Master_TIMER */
```

```
uint32_t cmpvalue;
```

```
cmpvalue = hrtimer_mastertimer_compare_value_get(HRTIMER0, HRTIMER_COMPARE0)
```

函数 hrtimer_timers_counter_value_config

函数hrtimer_timers_counter_value_config描述见下表:

表 3-992. 函数 hrtimer_timers_counter_value_config

函数名称	hrtimer_timers_counter_value_config
函数原型	void hrtimer_timers_counter_value_config(uint32_t hrtimer_periph, uint32_t timer_id, uint32_t cntvalue);
功能描述	配置Master_TIMER/Slave_TIMER的计数寄存器值
先决条件	-
被调用函数	-
输入参数{in}	
hrtimer_periph	HRTIMER外设
HRTIMERx(x=0)	选择一个HRTIMER外设
输入参数{in}	
timer_id	选择HRTIMER中的一个定时器
HRTIMER_MASTER_TIMER	选择Master_TIMER
HRTIMER_SLAVE_TIMERx(x=0..7)	选择Slave_TIMERx(x=0..7)
输入参数{in}	
cntvalue	计数器值, 范围从0到0xffff
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* configure the counter value in Master_TIMER and Slave_TIMER */
```

```
hrtimer_timers_counter_value_config(HRTIMER0, HRTIMER_MASTER_TIMER, 100);
```

函数 hrtimer_timers_counter_value_get

函数hrtimer_timers_counter_value_get描述见下表：

表 3-993. 函数 hrtimer_timers_counter_value_get

函数名称	hrtimer_timers_counter_value_get
函数原型	uint32_t hrtimer_timers_counter_value_get(uint32_t hrtimer_periph, uint32_t timer_id);
功能描述	获取Master_TIMER/Slave_TIMER的计数寄存器值
先决条件	-
被调用函数	-
输入参数{in}	
hrtimer_periph	HRTIMER外设
HRTIMERx(x=0)	选择一个HRTIMER外设
输入参数{in}	
timer_id	选择HRTIMER中的一个定时器
HRTIMER_MASTER_TIMER	选择Master_TIMER
HRTIMER_SLAVE_TIMERx(x=0..7)	选择Slave_TIMERx(x=0..7)
输出参数{out}	
-	-
返回值	
uint32_t	计数器值

例如：

```
/* get the counter value in Master_TIMER and Slave_TIMER */
```

```
uint32_t value;
```

```
value = hrtimer_timers_counter_value_get(HRTIMER0, HRTIMER_MASTER_TIMER);
```

函数 hrtimer_timers_autoreload_value_config

函数hrtimer_timers_autoreload_value_config描述见下表：

表 3-994. 函数 hrtimer_timers_autoreload_value_config

函数名称	hrtimer_timers_autoreload_value_config
函数原型	void hrtimer_timers_autoreload_value_config(uint32_t hrtimer_periph, uint32_t timer_id, uint32_t carlvalue);
功能描述	配置Master_TIMER/Slave_TIMER的自动重载寄存器值
先决条件	-
被调用函数	-
输入参数{in}	
hrtimer_periph	HRTIMER外设

<i>HRTIMERx(x=0)</i>	选择一个HRTIMER外设
输入参数{in}	
timer_id	选择HRTIMER中的一个定时器
<i>HRTIMER_MASTER_TIMER</i>	选择Master_TIMER
<i>HRTIMER_SLAVE_TIMERx(x=0..7)</i>	选择Slave_TIMERx(x=0..7)
输入参数{in}	
carlvalue	自动重载值，范围从0到0xffff
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* configure the counter auto reload value in Master_TIMER and Slave_TIMER */
hrtimer_timers_autoreload_value_config(HRTIMER0, HRTIMER_MASTER_TIMER, 100);
```

函数 `hrtimer_timers_autoreload_value_get`

函数 `hrtimer_timers_autoreload_value_get` 描述见下表：

表 3-995. 函数 `hrtimer_timers_autoreload_value_get`

函数名称	<code>hrtimer_timers_autoreload_value_get</code>
函数原型	<code>uint32_t hrtimer_timers_autoreload_value_get(uint32_t hrtimer_periph, uint32_t timer_id)</code>
功能描述	获取Master_TIMER/Slave_TIMER的自动重载寄存器值
先决条件	-
被调用函数	-
输入参数{in}	
hrtimer_periph	HRTIMER外设
<i>HRTIMERx(x=0)</i>	选择一个HRTIMER外设
输入参数{in}	
timer_id	选择HRTIMER中的一个定时器
<i>HRTIMER_MASTER_TIMER</i>	选择Master_TIMER
<i>HRTIMER_SLAVE_TIMERx(x=0..7)</i>	选择Slave_TIMERx(x=0..7)
输出参数{out}	
-	-
返回值	
uint32_t	自动重载值

例如:

```
/* get the counter auto reload value in Master_TIMER and Slave_TIMER */
```

```
uint32_t value;
```

```
value = hrtimer_timers_autoreload_value_get(HRTIMER0, HRTIMER_MASTER_TIMER);
```

函数 `hrtimer_timers_repetition_value_config`

函数 `hrtimer_timers_repetition_value_config` 描述见下表:

表 3-996. 函数 `hrtimer_timers_repetition_value_config`

函数名称	<code>hrtimer_timers_repetition_value_config</code>
函数原型	<code>void hrtimer_timers_repetition_value_config(uint32_t hrtimer_periph, uint32_t timer_id, uint32_t replvalue);</code>
功能描述	配置Master_TIMER/Slave_TIMER的重复计数寄存器值
先决条件	-
被调用函数	-
输入参数{in}	
hrtimer_periph	HRTIMER外设
<code>HRTIMERx(x=0)</code>	选择一个HRTIMER外设
输入参数{in}	
timer_id	选择HRTIMER中的一个定时器
<code>HRTIMER_MASTER_TIMER</code>	选择Master_TIMER
<code>HRTIMER_SLAVE_TIMERx(x=0..7)</code>	选择Slave_TIMERx(x=0..7)
输入参数{in}	
replvalue	重复计数值, 范围从0到0xff
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* configure the counter repetition value in Master_TIMER and Slave_TIMER */
```

```
hrtimer_timers_repetition_value_config(HRTIMER0, HRTIMER_MASTER_TIMER, 100);
```

函数 `hrtimer_timers_repetition_value_get`

函数 `hrtimer_timers_repetition_value_get` 描述见下表:

表 3-997. 函数 `hrtimer_timers_repetition_value_get`

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

函数原型	uint32_t hrtimer_timers_repetition_value_get(uint32_t hrtimer_periph, uint32_t timer_id);
功能描述	获取Master_TIMER/Slave_TIMER的重复计数寄存器值
先决条件	-
被调用函数	-
输入参数{in}	
hrtimer_periph	HRTIMER外设
HRTIMERx(x=0)	选择一个HRTIMER外设
输入参数{in}	
timer_id	选择HRTIMER中的一个定时器
HRTIMER_MASTER_TIMER	选择Master_TIMER
HRTIMER_SLAVE_TIMERx(x=0..7)	选择Slave_TIMERx(x=0..7)
输出参数{out}	
-	-
返回值	
uint32_t	重复计数值

例如：

```
/* get the counter repetition value in Master_TIMER and Slave_TIMER */
```

```
uint32_t value;
```

```
value = hrtimer_timers_repetition_value_get (HRTIMER0, HRTIMER_MASTER_TIMER);
```

函数 hrtimer_exeefilter_struct_para_init

函数hrtimer_exeefilter_struct_para_init描述见下表：

表 3-998. 函数 hrtimer_exeefilter_struct_para_init

函数名称	hrtimer_exeefilter_struct_para_init
函数原型	void hrtimer_exeefilter_struct_para_init(hrtimer_exeefilter_parameter_struct * exeefilter);
功能描述	初始化hrtimer_exeefilter_parameter_struct结构体变量
先决条件	-
被调用函数	-
输入参数{in}	
exeefilter	hrtimer_exeefilter_parameter_struct结构体指针，结构体成员为 表3-948. 结构体hrtimer_exeefilter_parameter_struct
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* initialize external event filtering for Slave_TIMER configuration struct with a default value */
```

```
hrtimer_exeefilter_parameter_struct exeefilter;
```

```
hrtimer_exeefilter_struct_para_init(&exeefilter);
```

函数 hrtimer_slavetimer_exeevent_filtering_config

函数hrtimer_slavetimer_exeevent_filtering_config描述见下表：

表 3-999. 函数 hrtimer_slavetimer_exeevent_filtering_config

函数名称	hrtimer_slavetimer_exeevent_filtering_config
函数原型	void hrtimer_slavetimer_exeevent_filtering_config(uint32_t hrtimer_periph, uint32_t timer_id, uint32_t event_id, hrtimer_exeefilter_parameter_struct *exeefilter);
功能描述	配置Slave_TIMER的外部事件滤波功能
先决条件	-
被调用函数	-
输入参数{in}	
hrtimer_periph	HRTIMER外设
HRTIMERx(x=0)	选择一个HRTIMER外设
输入参数{in}	
timer_id	选择HRTIMER中的一个定时器
HRTIMER_SLAVE_TIMERx(x=0..7)	选择Slave_TIMERx(x=0..7)
输入参数{in}	
event_id	HRTIMER外部事件索引
HRTIMER_EXEVENT_NONE	清除外部事件滤波配置
HRTIMER_EXEVENT_y(y=0..9)	外部事件y(y=0..9)
输入参数{in}	
exeefilter	hrtimer_exeefilter_parameter_struct结构体指针，结构体成员为 表3-948. 结构体hrtimer_exeefilter_parameter_struct
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* configure the external event filtering for Slave_TIMER (blanking, windowing) */
```

```
hrtimer_exeefilter_parameter_struct exeefilter;
```

```

hrtimer_exevertimer_struct_para_init(&exevertimer);

exevertimer.filter_mode = HRTIMER_EXEVERTIMER_BLANKINGCMP1;

exevertimer.memorized = HRTIMER_EXEVMEMORIZED_DISABLE;

hrtimer_slavetimer_exevertimer_filtering_config(HRTIMER0, HRTIMER_SLAVE_TIMER0,
HRTIMER_EXEVENT_5, &exevertimer);

```

函数 hrtimer_exevertimer_struct_para_init

函数hrtimer_exevertimer_struct_para_init描述见下表：

表 3-1000. 函数 hrtimer_exevertimer_struct_para_init

函数名称	hrtimer_exevertimer_struct_para_init
函数原型	void hrtimer_exevertimer_struct_para_init(hrtimer_exevertimer_parameter_struct * exevertimer);
功能描述	初始化hrtimer_exevertimer_parameter_struct结构体变量
先决条件	-
被调用函数	-
输入参数{in}	
exevertimer	hrtimer_exevertimer_parameter_struct结构体指针，结构体成员参考 表3-953. 结构体hrtimer_exevertimer_parameter_struct
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* initialize external event configuration struct with a default value */
```

```
hrtimer_exevertimer_parameter_struct exevertimer;
```

```
hrtimer_exevertimer_struct_para_init(&exevertimer);
```

函数 hrtimer_exevertimer_config

函数hrtimer_exevertimer_config描述见下表：

表 3-1001. 函数 hrtimer_exevertimer_config

函数名称	hrtimer_exevertimer_config
函数原型	void hrtimer_exevertimer_config(uint32_t hrtimer_periph, uint32_t event_id, hrtimer_exevertimer_parameter_struct* exevertimer);
功能描述	配置外部事件
先决条件	-
被调用函数	-

输入参数{in}	
hrtimer_periph	HRTIMER外设
<i>HRTIMERx(x=0)</i>	选择一个HRTIMER外设
输入参数{in}	
event_id	外部事件索引
<i>HRTIMER_EXEVENT_NONE</i>	清除外部事件配置
<i>HRTIMER_EXEVENT_y(y=0..9)</i>	外部事件y(y=0..9)
输入参数{in}	
exevcfg	hrtimer_exeventcfg_parameter_struct结构体指针，结构体成员参考 表3-953. 结构体hrtimer_exeventcfg_parameter_struct
输出参数{out}	
-	-
返回值	
-	-

例如：

```

/* configure the an external event */

hrtimer_exeventcfg_parameter_struct exevcfg;

hrtimer_exeventcfg_struct_para_init(&exevcfg);

exevcfg.digital_filter = 0x5;

exevcfg.edge = HRTIMER_EXEV_EDGE_RISING;

exevcfg.polarity = HRTIMER_EXEV_EDGE_LEVEL;

exevcfg.source = HRTIMER_EXEV_SRC0;

hrtimer_exevent_config(HRTIMER0, HRTIMER_EXEVENT_5, &exevcfg);

```

函数 hrtimer_exevent_prescaler

函数hrtimer_exevent_prescaler描述见下表：

表 3-1002. 函数 hrtimer_exevent_prescaler

函数名称	hrtimer_exevent_prescaler
函数原型	void hrtimer_exevent_prescaler(uint32_t hrtimer_periph, uint32_t prescaler);
功能描述	配置外部事件的数字滤波分频
先决条件	-
被调用函数	-
输入参数{in}	
hrtimer_periph	HRTIMER外设
<i>HRTIMERx(x=0)</i>	选择一个HRTIMER外设

输入参数{in}	
prescaler	外部事件数字滤波器时钟分频
<i>HRTIMER_EXEV_PRE</i> <i>SCALER_DIVx</i> (<i>x</i> =1,2,4,8)	$fHRTIMER_EXEVFCK = fHRTIMER_CK/x$ (<i>x</i> =1,2,4,8)
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* configure external event digital filter clock division */
```

```
hrtimer_exevent_prescaler(HRTIMER0, HRTIMER_EXEV_PRESCALER_DIV1);
```

函数 `hrtimer_exeventx_counter_struct_para_init`

函数 `hrtimer_exeventx_counter_struct_para_init` 描述见下表：

表 3-1003. 函数 `hrtimer_exeventx_counter_struct_para_init`

函数名称	<code>hrtimer_exeventx_counter_struct_para_init</code>
函数原型	void <code>hrtimer_exeventx_counter_struct_para_init(hrtimer_exeventcnt_parameter_struct *exevcnt);</code>
功能描述	初始化 <code>hrtimer_exeventx_counter_struct</code> 结构体变量
先决条件	-
被调用函数	-
输入参数{in}	
exevcnt	<code>hrtimer_exeventcnt_parameter_struct</code> 结构体指针，结构体成员参考 表3-954. 结构体 <code>hrtimer_exeventcnt_parameter_struct</code>
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* initialize the an external event struct with a default value */
```

```
hrtimer_exeventcnt_parameter_struct exevcnt;
```

```
hrtimer_exeventx_counter_struct_para_init(&exevcnt);
```

函数 `hrtimer_exeventx_counter_config`

函数 `hrtimer_exeventx_counter_config` 描述见下表：

表 3-1004. 函数 `hrtimer_exeventx_counter_config`

函数名称	<code>hrtimer_exeventx_counter_config</code>
函数原型	<code>void hrtimer_exeventx_counter_config(uint32_t hrtimer_periph, uint32_t timer_id, hrtimer_exeventcnt_parameter_struct *exevcnt);</code>
功能描述	配置外部事件X计数值
先决条件	-
被调用函数	-
输入参数{in}	
hrtimer_periph	HRTIMER外设
<i>HRTIMERx(x=0)</i>	选择一个HRTIMER外设
输入参数{in}	
timer_id	选择HRTIMER中的一个定时器
<i>HRTIMER_SLAVE_TIMERx(x=0..7)</i>	选择Slave_TIMERx(x=0..7)
输入参数{in}	
exevcnt	<code>hrtimer_exeventcnt_parameter_struct</code> 结构体指针，结构体成员参考 表3-954. 结构体hrtimer_exeventcnt_parameter_struct
输出参数{out}	
-	-
返回值	
-	-

例如：

```

/* configure the external event X counter */

hrtimer_exeventcnt_parameter_struct exevcnt;

hrtimer_exeventx_counter_struct_para_init (&exevcnt);

exevcnt.exevcnt = 0x5;

exevcnt.reset_mode = HRTIMER_EXEVX_RESET_MODE0;

exevcnt.event_source = HRTIMER_EXEVX_SOURCE_EXEV0;

hrtimer_exeventx_counter_config(HRTIMER0, HRTIMER_SLAVE_TIMER0, &exevcnt);

```

函数 `hrtimer_software_reset_exeventx_counter`

函数`hrtimer_software_reset_exeventx_counter`描述见下表：

表 3-1005. 函数 `hrtimer_software_reset_exeventx_counter`

函数名称	<code>hrtimer_software_reset_exeventx_counter</code>
函数原型	<code>void hrtimer_software_reset_exeventx_counter(uint32_t hrtimer_periph, uint32_t timer_id);</code>
功能描述	初始化外部事件X的计数值

先决条件	-
被调用函数	-
输入参数{in}	
hrtimer_periph	HRTIMER外设
<i>HRTIMERx(x=0)</i>	选择一个HRTIMER外设
输入参数{in}	
timer_id	选择HRTIMER中的一个定时器
<i>HRTIMER_SLAVE_TIMERx(x=0..7)</i>	选择Slave_TIMERx(x=0..7)
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* configure the external event X counter */
```

```
/* reset external event X counter */
```

```
hrtimer_software_reset_exeventx_counter(HRTIMER0, HRTIMER_SLAVE_TIMER0);
```

函数 **hrtimer_exeventx_counter_enable**

函数hrtimer_exeventx_counter_enable描述见下表：

表 3-1006. 函数 hrtimer_exeventx_counter_enable

函数名称	hrtimer_exeventx_counter_enable
函数原型	void hrtimer_exeventx_counter_enable(uint32_t hrtimer_periph, uint32_t timer_id);
功能描述	使能外部事件X的计数值
先决条件	-
被调用函数	-
输入参数{in}	
hrtimer_periph	HRTIMER外设
<i>HRTIMERx(x=0)</i>	选择一个HRTIMER外设
输入参数{in}	
timer_id	选择HRTIMER中的一个定时器
<i>HRTIMER_SLAVE_TIMERx(x=0..7)</i>	选择Slave_TIMERx(x=0..7)
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* enable external event X counter */
```

```
hrtimer_exeventx_counter_enable(HRTIMER0, HRTIMER_SLAVE_TIMER0);
```

函数 hrtimer_exeventx_counter_disable

函数hrtimer_exeventx_counter_disable描述见下表：

表 3-1007. 函数 hrtimer_exeventx_counter_disable

函数名称	hrtimer_exeventx_counter_disable
函数原型	void hrtimer_exeventx_counter_disable(uint32_t hrtimer_periph, uint32_t timer_id);
功能描述	禁用外部事件X的计数值
先决条件	-
被调用函数	-
输入参数{in}	
hrtimer_periph	HRTIMER外设
HRTIMERx(x=0)	选择一个HRTIMER外设
输入参数{in}	
timer_id	选择HRTIMER中的一个定时器
HRTIMER_SLAVE_TIMERx(x=0..7)	选择Slave_TIMERx(x=0..7)
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* disable external event X counter */
```

```
hrtimer_exeventx_counter_disable(HRTIMER0, HRTIMER_SLAVE_TIMER0);
```

函数 hrtimer_exeventx_counter_read

函数hrtimer_exeventx_counter_read描述见下表：

表 3-1008. 函数 hrtimer_exeventx_counter_read

函数名称	hrtimer_exeventx_counter_read
函数原型	uint32_t hrtimer_exeventx_counter_read(uint32_t hrtimer_periph, uint32_t timer_id)
功能描述	读取外部事件X的计数值
先决条件	-
被调用函数	-
输入参数{in}	
hrtimer_periph	HRTIMER外设

<i>HRTIMERx(x=0)</i>	选择一个HRTIMER外设
输入参数{in}	
timer_id	选择HRTIMER中的一个定时器
<i>HRTIMER_SLAVE_TIMERx(x=0..7)</i>	选择Slave_TIMERx(x=0..7)
输出参数{out}	
-	-
返回值	
uint32_t	外部事件X的计数值

例如:

```
/* disable external event X counter */
```

```
/* get the external event X counter value */
```

```
uint32_t value;
```

```
value = hrtimer_exeventx_counter_read (HRTIMER0, HRTIMER_SLAVE_TIMER0);
```

函数 `hrtimer_synccfg_struct_para_init`

函数 `hrtimer_synccfg_struct_para_init` 描述见下表:

表 3-1009. 函数 `hrtimer_synccfg_struct_para_init`

函数名称	<code>hrtimer_synccfg_struct_para_init</code>
函数原型	<code>void hrtimer_synccfg_struct_para_init(hrtimer_synccfg_parameter_struct* synccfg);</code>
功能描述	初始化 <code>hrtimer_synccfg_parameter_struct</code> 结构体变量
先决条件	-
被调用函数	-
输入参数{in}	
synccfg	<code>hrtimer_synccfg_parameter_struct</code> 结构体指针, 结构体成员参考 表3-951. 结构体 <code>hrtimer_synccfg_parameter_struct</code> 结构体 <code>hrtimer_synccfg_parameter_struct</code>
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* initialize synchronization configuration struct with a default value */
```

```
hrtimer_synccfg_parameter_struct synccfg;
```

```
hrtimer_synccfg_struct_para_init (&synccfg);
```

函数 hrtimer_synchronization_config

函数hrtimer_synchronization_config描述见下表:

表 3-1010. 函数 hrtimer_synchronization_config

函数名称	hrtimer_synchronization_config
函数原型	void hrtimer_synchronization_config(uint32_t hrtimer_periph, hrtimer_synccfg_parameter_struct* synccfg);
功能描述	配置HRTIMER的同步输入/输出功能
先决条件	-
被调用函数	-
输入参数{in}	
hrtimer_periph	HRTIMER外设
HRTIMERx(x=0)	选择一个HRTIMER外设
输入参数{in}	
synccfg	hrtimer_synccfg_parameter_struct结构体指针, 结构体成员参考 表3-951. 结构体hrtimer_synccfg_parameter_struct 结构体 hrtimer_synccfg_parameter_struct
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* configure the synchronization input/output of the HRTIMER */
hrtimer_synccfg_parameter_struct synccfg;
hrtimer_synccfg_struct_para_init (&synccfg);
synccfg.input_source = HRTIMER_SYNCINPUTSOURCE_EXTERNAL;
hrtimer_synchronization_config(HRTIMER0, &synccfg);
```

函数 hrtimer_double_channel_struct_para_init

函数hrtimer_double_channel_struct_para_init描述见下表:

表 3-1011. 函数 hrtimer_double_channel_struct_para_init

函数名称	hrtimer_double_channel_struct_para_init
函数原型	void hrtimer_double_channel_struct_para_init void hrtimer_double_channel_struct_para_init(hrtimer_double_trigger_parameter _struct *doubletrigger);
功能描述	初始化hrtimer_double_trigger_parameter_struct结构体变量
先决条件	-
被调用函数	-

输入参数{in}	
doubletrigger	hrtimer_double_trigger_parameter_struct, 结构体成员参考 表3-959. 结构体hrtimer_double_trigger_parameter_struct
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* initialize double trigger struct with a default value */
```

```
hrtimer_double_trigger_parameter_struct doubletrigger;
```

```
hrtimer_double_channel_struct_para_init(&doubletrigger);
```

函数 hrtimer_double_trigger_config

函数hrtimer_double_trigger_config描述见下表:

表 3-1012. 函数 hrtimer_double_trigger_config

函数名称	hrtimer_double_trigger_config
函数原型	void hrtimer_double_trigger_config(uint32_t hrtimer_periph, uint32_t timer_id, hrtimer_double_trigger_parameter_struct *doubletrigger);
功能描述	配置双触发模式
先决条件	-
被调用函数	-
输入参数{in}	
hrtimer_periph	HRTIMER外设
HRTIMERx(x=0)	选择一个HRTIMER外设
输入参数{in}	
timer_id	选择HRTIMER中的一个定时器
HRTIMER_SLAVE_TIMERx(x=0..7)	选择Slave_TIMERx(x=0..7)
输入参数{in}	
doubletrigger	hrtimer_double_trigger_parameter_struct结构体指针, 结构体成员参考 表3-959. 结构体hrtimer_double_trigger_parameter_struct
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* configure Slave_TIMER in double channel trigger */
```



```
hrtimer_double_trigger_parameter_struct doubletrigger;
```

```
hrtimer_double_trigger_config (HRTIMER0, HRTIMER_SLAVE_TIMER0, & doubletrigger);
```

函数 hrtimer_roll_over_struct_para_init

函数hrtimer_roll_over_struct_para_init描述见下表：

表 3-1013. 函数 hrtimer_roll_over_struct_para_init

函数名称	hrtimer_roll_over_struct_para_init
函数原型	void hrtimer_roll_over_struct_para_init(hrtimer_roll_over_parameter_struct *rollover);
功能描述	初始化hrtimer_roll_over_struct结构体变量
先决条件	-
被调用函数	-
输入参数{in}	
rollover	hrtimer_two_dac_trigger_parameter_struct, 结构体成员参考 表3-958. 结构体hrtimer_roll_over_parameter_struct
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* initialize roll over struct with a default value */
```

```
hrtimer_roll_over_parameter_struct rollover;
```

```
hrtimer_roll_over_struct_para_init (&rollover);
```

函数 hrtimer_roll_over_mode_config

函数hrtimer_roll_over_mode_config描述见下表：

表 3-1014. 函数 hrtimer_roll_over_mode_config

函数名称	hrtimer_roll_over_mode_config
函数原型	void hrtimer_roll_over_mode_config(uint32_t hrtimer_periph, uint32_t timer_id, hrtimer_roll_over_parameter_struct *rollover);
功能描述	配置Slave_TIMERx在翻转模式
先决条件	-
被调用函数	-
输入参数{in}	
hrtimer_periph	HRTIMER外设
HRTIMERx(x=0)	选择一个HRTIMER外设
输入参数{in}	
timer_id	选择HRTIMER中的一个定时器

HRTIMER_SLAVE_TIMERx(x=0..7)	选择Slave_TIMERx(x=0..7)
输入参数{in}	
rollover	hrtimer_two_dac_trigger_parameter_struct结构体指针，结构体成员参考 表3-958. 结构体hrtimer_roll_over_parameter_struct
输出参数{out}	
-	-
返回值	
-	-

例如：

```

/* configure Slave_TIMER in roll over mode */

hrtimer_roll_over_parameter_struct rollover;

hrtimer_roll_over_struct_para_init (&rollover);

rollover.roll_over_mode = HRTIMER_ROLLOVER_MODE1;

rollover.output_roll_over_mode = HRTIMER_OUTPUT_ROLLOVER_MODE0;

rollover.adc_roll_over_mode = HRTIMER_ADC_ROLLOVER_MODE0;

rollover.bunch_mode_roll_over_mode = HRTIMER_BUNCH_MODE_ROLLOVER_MODE0;

rollover.fault_event_roll_over_mode = HRTIMER_FAULT_EVENTROLLOVER_MODE0;

hrtimer_roll_over_mode_config (HRTIMER0 , HRTIMER_SLAVE_TIMER0, & rollover);

```

函数 hrtimer_faultcfg_struct_para_init

函数hrtimer_faultcfg_struct_para_init描述见下表：

表 3-1015. 函数 hrtimer_faultcfg_struct_para_init

函数名称	hrtimer_faultcfg_struct_para_init
函数原型	void hrtimer_faultcfg_struct_para_init(hrtimer_faultcfg_parameter_struct * faultcfg);
功能描述	初始化hrtimer_faultcfg_parameter_struct结构体变量
先决条件	-
被调用函数	-
输入参数{in}	
faultcfg	hrtimer_faultcfg_parameter_struct结构体指针，结构体成员参考 表3-955. 结构体hrtimer_faultcfg_parameter_struct
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* initialize fault input configuration struct with a default value */
```

```
hrtimer_faultcfg_parameter_struct faultcfg;
```

```
hrtimer_synccfg_struct_para_init (&faultcfg);
```

函数 hrtimer_fault_config

函数hrtimer_fault_config描述见下表:

表 3-1016. 函数 hrtimer_fault_config

函数名称	hrtimer_fault_config
函数原型	void hrtimer_fault_config(uint32_t hrtimer_periph, uint32_t fault_id, hrtimer_faultcfg_parameter_struct* faultcfg);
功能描述	配置故障输入功能
先决条件	-
被调用函数	-
输入参数{in}	
hrtimer_periph	HRTIMER外设
HRTIMERx(x=0)	选择一个HRTIMER外设
输入参数{in}	
fault_id	故障输入索引
HRTIMER_FAULT_y(y=0..7)	选择一个故障输入
输入参数{in}	
faultcfg	hrtimer_faultcfg_parameter_struct结构体指针, 结构体成员参考 表3-955. 结构体hrtimer_faultcfg_parameter_struct
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* configure the fault input */
```

```
hrtimer_faultcfg_parameter_struct faultcfg;
```

```
hrtimer_synccfg_struct_para_init (&faultcfg);
```

```
faultcfg_para.control = HRTIMER_FAULT_CHANNEL_ENABLE;
```

```
faultcfg_para.filter = 0x0;
```

```
faultcfg_para.polarity = HRTIMER_FAULT_POLARITY_HIGH;
```

```
faultcfg_para.protect = HRTIMER_FAULT_PROTECT_ENABLE;
```

```
faultcfg_para.source = HRTIMER_FAULT_SOURCE_PIN;
```

```
hrtimer_fault_config(HRTIMER0, HRTIMER_FAULT_0, &faultcfg);
```

函数 hrtimer_fault_prescaler_config

函数hrtimer_fault_prescaler_config描述见下表：

表 3-1017. 函数 hrtimer_fault_prescaler_config

函数名称	hrtimer_fault_prescaler_config
函数原型	void hrtimer_fault_prescaler_config(uint32_t hrtimer_periph, uint32_t prescaler);
功能描述	配置故障输入数字滤波分频
先决条件	-
被调用函数	-
输入参数{in}	
hrtimer_periph	HRTIMER外设
HRTIMERx(x=0)	选择一个HRTIMER外设
输入参数{in}	
prescaler	故障输入数字滤波器时钟分频
HRTIMER_FAULT_PRESCALER_DIVy(y=1,2,4,8)	$f_{HRTIMER_FLTFCCK} = f_{HRTIMER_CK}/y$ (y=1,2,4,8)
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* configure the fault input digital filter clock division */
```

```
void hrtimer_fault_prescaler_config(HRTIMER0, HRTIMER_FAULT_PRESCALER_DIV1);
```

函数 hrtimer_fault_input_enable

函数hrtimer_fault_input_enable描述见下表：

表 3-1018. 函数 hrtimer_fault_input_enable

函数名称	hrtimer_fault_input_enable
函数原型	void hrtimer_fault_input_enable(uint32_t hrtimer_periph, uint32_t fault_id);
功能描述	故障输入使能
先决条件	-
被调用函数	-
输入参数{in}	
hrtimer_periph	HRTIMER外设

<i>HRTIMERx</i> (<i>x</i> =0)	选择一个HRTIMER外设
输入参数{in}	
fault_id	故障输入索引
<i>HRTIMER_FAULT_y</i> (<i>y</i> =0..7)	选择一个故障输入
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* fault input enable */
```

```
hrtimer_fault_input_enable(HRTIMER0, HRTIMER_FAULT_0);
```

函数 `hrtimer_fault_input_disable`

函数 `hrtimer_fault_input_disable` 描述见下表:

表 3-1019. 函数 `hrtimer_fault_input_disable`

函数名称	<code>hrtimer_fault_input_disable</code>
函数原型	<code>void hrtimer_fault_input_disable(uint32_t hrtimer_periph, uint32_t fault_id);</code>
功能描述	故障输入禁能
先决条件	-
被调用函数	-
输入参数{in}	
hrtimer_periph	HRTIMER外设
<i>HRTIMERx</i> (<i>x</i> =0)	选择一个HRTIMER外设
输入参数{in}	
fault_id	故障输入索引
<i>HRTIMER_FAULT_y</i> (<i>y</i> =0..7)	选择一个故障输入
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* fault input disable */
```

```
hrtimer_fault_input_disable(HRTIMER0, HRTIMER_FAULT_0);
```

函数 `hrtimer_fault_counter_reset`

函数 `hrtimer_fault_counter_reset` 描述见下表:

表 3-1020. 函数 `hrtimer_fault_counter_reset`

函数名称	<code>hrtimer_fault_counter_reset</code>
函数原型	<code>void hrtimer_fault_counter_reset(uint32_t hrtimer_periph, uint32_t fault_id);</code>
功能描述	故障计数值复位
先决条件	-
被调用函数	-
输入参数{in}	
hrtimer_periph	HRTIMER外设
<i>HRTIMERx(x=0)</i>	选择一个HRTIMER外设
输入参数{in}	
fault_id	故障输入索引
<i>HRTIMER_FAULT_y(y=0..7)</i>	选择一个故障输入
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* fault input counter reset */
```

```
hrtimer_fault_counter_reset(HRTIMER0, HRTIMER_FAULT_0);
```

函数 `hrtimer_fault_blank_enable`

函数`hrtimer_fault_blank_enable`见下表:

表 3-1021. 函数 `hrtimer_fault_blank_enable`

函数名称	<code>hrtimer_fault_blank_enable</code>
函数原型	<code>void hrtimer_fault_blank_enable(uint32_t hrtimer_periph, uint32_t fault_id);</code>
功能描述	故障消隐使能
先决条件	-
被调用函数	-
输入参数{in}	
hrtimer_periph	HRTIMER外设
<i>HRTIMERx(x=0)</i>	选择一个HRTIMER外设
输入参数{in}	
fault_id	故障输入索引
<i>HRTIMER_FAULT_y(y=0..7)</i>	选择一个故障输入
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* fault input blank enable*/
```

```
hrtimer_fault_blank_enable(HRTIMER0, HRTIMER_FAULT_0);
```

函数 hrtimer_fault_blank_disable

函数hrtimer_fault_blank_disable见下表：

表 3-1022. 函数 hrtimer_fault_blank_disable

函数名称	hrtimer_fault_blank_disable
函数原型	void hrtimer_fault_blank_disable (uint32_t hrtimer_periph, uint32_t fault_id);
功能描述	故障消隐禁能
先决条件	-
被调用函数	-
输入参数{in}	
hrtimer_periph	HRTIMER外设
HRTIMERx(x=0)	选择一个HRTIMER外设
输入参数{in}	
fault_id	故障输入索引
HRTIMER_FAULT_y(y=0..7)	选择一个故障输入
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* fault input blank disable*/
```

```
hrtimer_fault_blank_disable(HRTIMER0, HRTIMER_FAULT_0);
```

函数 hrtimer_timers_dma_enable

函数hrtimer_timers_dma_enable描述见下表：

表 3-1023. 函数 hrtimer_timers_dma_enable

函数名称	hrtimer_timers_dma_enable
函数原型	void hrtimer_timers_dma_enable(uint32_t hrtimer_periph, uint32_t timer_id, uint32_t dmareq);
功能描述	使能Master_TIMER/Slave_TIMER DMA请求
先决条件	-
被调用函数	-
输入参数{in}	
hrtimer_periph	HRTIMER外设

<i>HRTIMERx</i> (<i>x</i> =0)	选择一个HRTIMER外设
输入参数{in}	
timer_id	选择HRTIMER中的一个定时器
<i>HRTIMER_MASTER_TIMER</i>	选择Master_TIEMR
<i>HRTIMER_SLAVE_TIMERx</i>	选择Slave_TIEMRx (<i>x</i> =0..7)
输入参数{in}	
dmareq	DMA请求源
<i>HRTIMER_MT_ST_DMA_CMPy</i> (<i>y</i> =0..3)	比较 <i>y</i> (<i>y</i> =0..3)DMA请求 (适用于Master_TIMER和Slave_TIMER)
<i>HRTIMER_MT_ST_DMA_A_REP</i>	重复DMA请求 (适用于Master_TIMER和Slave_TIMER)
<i>HRTIMER_MT_DMA_SYNCID</i>	同步输入DMA请求 (仅适用于Master_TIMER)
<i>HRTIMER_MT_ST_DMA_A_UPD</i>	更新DMA请求 (适用于Master_TIMER和Slave_TIMER)
<i>HRTIMER_ST_DMA_CAPy</i> (<i>y</i> =0,1)	捕获 <i>y</i> (<i>y</i> =0,1)DMA请求 (仅适用于Slave_TIMER)
<i>HRTIMER_ST_DMA_CHyOA</i> (<i>y</i> =0,1)	通道 <i>y</i> (<i>y</i> =0,1)输出置位DMA请求 (仅适用于Slave_TIMER)
<i>HRTIMER_ST_DMA_CHyONA</i> (<i>y</i> =0,1)	通道 <i>y</i> (<i>y</i> =0,1)输出置位DMA请求 (仅适用于Slave_TIMER)
<i>HRTIMER_ST_DMA_CNTRST</i>	计数器复位DMA请求 (仅适用于Slave_TIMER)
<i>HRTIMER_ST_DMA_DLIDLE</i>	延迟空闲模式进入DMA请求 (仅适用于Slave_TIMER)
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* enable the Master_TIMER and Slave_TIMER DMA request */
```

```
hrtimer_timers_dma_enable(HRTIMER0, HRTIMER_MASTER_TIMER,
HRTIMER_MT_ST_DMA_CMP0);
```

函数 hrtimer_timers_dma_disable

函数hrtimer_timers_dma_disable描述见下表:

表 3-1024. 函数 hrtimer_timers_dma_enable

函数名称	hrtimer_timers_dma_disable
------	----------------------------

函数原型	void hrtimer_timers_dma_disable(uint32_t hrtimer_periph, uint32_t timer_id, uint32_t dmareq);
功能描述	禁能Master_TIMER/Slave_TIMER DMA请求
先决条件	-
被调用函数	-
输入参数{in}	
hrtimer_periph	HRTIMER外设
HRTIMERx(x=0)	选择一个HRTIMER外设
输入参数{in}	
timer_id	选择HRTIMER中的一个定时器
HRTIMER_MASTER_TIMER	选择Master_TIEMR
HRTIMER_SLAVE_TIMERx	选择Slave_TIEMRx (x=0..7)
输入参数{in}	
dmareq	DMA请求源
HRTIMER_MT_ST_DMA_CMPy(y=0..3)	比较y(y=0..3)DMA请求（适用于Master_TIMER和Slave_TIMER）
HRTIMER_MT_ST_DMA_REP	重复DMA请求（适用于Master_TIMER和Slave_TIMER）
HRTIMER_MT_DMA_SYNCI	同步输入DMA请求（仅适用于Master_TIMER）
HRTIMER_MT_ST_DMA_UPD	更新DMA请求（适用于Master_TIMER和Slave_TIMER）
HRTIMER_ST_DMA_CAPy(y=0,1)	捕获y(y=0,1)DMA请求（仅适用于Slave_TIMER）
HRTIMER_ST_DMA_CHyOA(y=0,1)	通道y(y=0,1)输出置位DMA请求（仅适用于Slave_TIMER）
HRTIMER_ST_DMA_CHyONA(y=0,1)	通道y(y=0,1)输出置位DMA请求（仅适用于Slave_TIMER）
HRTIMER_ST_DMA_CNTRST	计数器复位DMA请求（仅适用于Slave_TIMER）
HRTIMER_ST_DMA_DLYIDLE	延迟空闲模式进入DMA请求（仅适用于Slave_TIMER）
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* disable the Master_TIMER and Slave_TIMER DMA request */
hrtimer_timers_dma_disable(HRTIMER0, HRTIMER_MASTER_TIMER,
```

HRTIMER_MT_ST_DMA_CMP0);

函数 hrtimer_dmamode_config

函数hrtimer_dmamode_config描述见下表:

表 3-1025. 函数 hrtimer_dmamode_config

函数名称	hrtimer_dmamode_config
函数原型	void hrtimer_dmamode_config(uint32_t hrtimer_periph, uint32_t timer_id, uint32_t regupdate);
功能描述	配置Master_TIMER/Slave_TIMER的DMA模式
先决条件	-
被调用函数	-
输入参数{in}	
hrtimer_periph	HRTIMER外设
HRTIMERx(x=0)	选择一个HRTIMER外设
输入参数{in}	
timer_id	选择HRTIMER中的一个定时器
HRTIMER_MASTER_TIMER	选择Master_TIEMR
HRTIMER_SLAVE_TIMERx	选择Slave_TIEMRx (x=0..7)
输入参数{in}	
regupdate	更新的寄存器
HRTIMER_DMAMODE_NONE	不通过DMA模式更新寄存器（适用于Master_TIMER和Slave_TIMER）
HRTIMER_DMAMODE_CTL0	DMA模式更新MTCTL0, STxCTL0寄存器（适用于Master_TIMER和Slave_TIMER）
HRTIMER_DMAMODE_INTC	DMA模式更新MT, STx寄存器（适用于Master_TIMER和Slave_TIMER）
HRTIMER_DMAMODE_DMAINTEN	DMA模式更新MTINTC, STxINTC寄存器（适用于Master_TIMER和Slave_TIMER）
HRTIMER_DMAMODE_CNT	DMA模式更新MTCNT, STxCNT寄存器（适用于Master_TIMER和Slave_TIMER）
HRTIMER_DMAMODE_CAR	DMA模式更新MTCAR, STxCAR寄存器（适用于Master_TIMER和Slave_TIMER）
HRTIMER_DMAMODE_CREP	DMA模式更新MTCREP, STxCREP寄存器（适用于Master_TIMER和Slave_TIMER）
HRTIMER_DMAMODE_CMPyV(y=0..3)	DMA模式更新MTCMPyV(y=0..3), STxCMPyV(y=0..3)寄存器（适用于Master_TIMER和Slave_TIMER）
HRTIMER_DMAMODE_DTCTL	DMA模式更新STxDTCTL寄存器（仅适用于Slave_TIMER）
HRTIMER_DMAMODE	DMA模式更新STxCHySET(y=0,1)寄存器（仅适用于Slave_TIMER）

<code>_CHySET(y=0,1)</code>	
<code>HRTIMER_DMAMODE_CHyRST(y=0,1)</code>	DMA模式更新STxCHyRST(y=0,1)寄存器（仅适用于Slave_TIMER）
<code>HRTIMER_DMAMODE_EXEVFCFGy(y=0,1)</code>	DMA模式更新STxEXEVFCFGy(y=0,1)寄存器（仅适用于Slave_TIMER）
<code>HRTIMER_DMAMODE_CNTRST</code>	DMA模式更新STxCNTRST寄存器（仅适用于Slave_TIMER）
<code>HRTIMER_DMAMODE_CSCTL</code>	DMA模式更新STxCSCTL寄存器（仅适用于Slave_TIMER）
<code>HRTIMER_DMAMODE_CHOCTL</code>	DMA模式更新STxCHOCTL寄存器（仅适用于Slave_TIMER）
<code>HRTIMER_DMAMODE_FLTCTL</code>	DMA模式更新STxFLTCTL寄存器（仅适用于Slave_TIMER）
<code>HRTIMER_DMAMODE_CTL1</code>	DMA模式更新HRTIMER_DMAMODE_CTL1寄存器（仅适用于Slave_TIMER）
<code>HRTIMER_DMAMODE_EXEVFCFG2</code>	DMA模式更新HRTIMER_DMAMODE_EXEVFCFG2寄存器（仅适用于Slave_TIMER）
<code>HRTIMER_DMAMODE_ACTL</code>	DMA模式更新STxACTL寄存器（仅适用于Slave_TIMER）
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* enable the Master_TIMER and Slave_TIMER DMA request */
```

```
hrtimer_damode_config(HRTIMER0, HRTIMER_MASTER_TIMER,
HRTIMER_DMAMODE_NONE);
```

函数 hrtimer_bunchmode_struct_para_init

函数hrtimer_bunchmode_struct_para_init描述见下表：

表 3-1026. 函数 hrtimer_bunchmode_struct_para_init

函数名称	hrtimer_bunchmode_struct_para_init
函数原型	void hrtimer_bunchmode_struct_para_init(hrtimer_bunchmode_parameter_struct * bmcfg);
功能描述	初始化hrtimer_bunchmode_parameter_struct结构体
先决条件	-
被调用函数	-
输入参数{in}	
bmcfg	hrtimer_bunchmode_parameter_struct结构体指针，结构体成员参考

	表3-952. 结构体hrtimer_bunchmode_parameter_struct
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* initialize bunch mode configuration struct with a default value */
```

```
hrtimer_bunchmode_parameter_struct bmcfg;
```

```
hrtimer_bunchmode_struct_para_init(&bmcfg);
```

函数 hrtimer_bunchmode_config

函数hrtimer_bunchmode_config描述见下表：

表 3-1027. 函数 hrtimer_bunchmode_config

函数名称	hrtimer_bunchmode_config
函数原型	void hrtimer_bunchmode_config(uint32_t hrtimer_periph, hrtimer_bunchmode_parameter_struct* bmcfg);
功能描述	配置HRTIMER的突发模式
先决条件	-
被调用函数	-
输入参数{in}	
hrtimer_periph	HRTIMER外设
HRTIMERx(x=0)	选择一个HRTIMER外设
输入参数{in}	
bmcfg	hrtimer_bunchmode_parameter_struct结构体指针，结构体成员参考 表3-952. 结构体hrtimer_bunchmode_parameter_struct
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* configure bunch mode for the HRTIMER */
```

```
hrtimer_bunchmode_parameter_struct bmcfg;
```

```
hrtimer_bunchmode_struct_para_init(&bmcfg);
```

```
bmcfg.clock_source = HRTIMER_BUNCHMODE_CLOCKSOURCE_ST0;
```

```
bmcfg.idle_duration = 3;
```

```
bmcfg.mode = HRTIMER_BUNCHMODE_CONTINUOUS;
```

```

bmcfg.period = 6;

bmcfg.prescaler = HRTIMER_BUNCHMODE_PRESCALER_DIV1;

bmcfg.shadow = HRTIMER_BUNCHMODEPRELOAD_DISABLED;

bmcfg.trigger = HRTIMER_BUNCHMODE_TRIGGER_SOFTWARE;

hrtimer_bunchmode_config(HRTIMER0, &bmcfg);

```

函数 hrtimer_bunchmode_enable

函数hrtimer_bunchmode_enable描述见下表:

表 3-1028. 函数 hrtimer_bunchmode_enable

函数名称	hrtimer_bunchmode_enable
函数原型	void hrtimer_bunchmode_enable(uint32_t hrtimer_periph);
功能描述	使能突发模式
先决条件	-
被调用函数	-
输入参数{in}	
hrtimer_periph	HRTIMER外设
HRTIMERx(x=0)	选择一个HRTIMER外设
输出参数{out}	
-	-
返回值	
-	-

例如:

```

/* enable bunch mode for the HRTIMER */

hrtimer_bunchmode_enable(HRTIMER0);

```

函数 hrtimer_bunchmode_disable

函数hrtimer_bunchmode_disable描述见下表:

表 3-1029. 函数 hrtimer_bunchmode_disable

函数名称	hrtimer_bunchmode_disable
函数原型	void hrtimer_bunchmode_disable(uint32_t hrtimer_periph);
功能描述	禁能突发模式
先决条件	-
被调用函数	-
输入参数{in}	
hrtimer_periph	HRTIMER外设
HRTIMERx(x=0)	选择一个HRTIMER外设
输出参数{out}	

-	-
返回值	
-	-

例如：

```
/* disable bunch mode for the HRTIMER */
```

```
hrtimer_bunchmode_disable(HRTIMER0);
```

函数 hrtimer_bunchmode_flag_get

函数hrtimer_bunchmode_flag_get描述见下表：

表 3-1030. 函数 hrtimer_bunchmode_flag_get

函数名称	hrtimer_bunchmode_flag_get
函数原型	uint32_t hrtimer_bunchmode_flag_get(uint32_t hrtimer_periph);
功能描述	获取突发模式的运行标志
先决条件	-
被调用函数	-
输入参数{in}	
hrtimer_periph	HRTIMER外设
HRTIMERx(x=0)	选择一个HRTIMER外设
输出参数{out}	
-	-
返回值	
uint32_t	HRTIMER_BUNCHMODE_OPERATION_OFF或 HRTIMER_BUNCHMODE_OPERATION_ON

例如：

```
/* get bunch mode operating flag */
```

```
uint32_t flag;
```

```
flag = hrtimer_bunchmode_flag_get(HRTIMER0);
```

函数 hrtimer_bunchmode_software_start

函数hrtimer_bunchmode_software_start描述见下表：

表 3-1031. 函数 hrtimer_bunchmode_software_start

函数名称	hrtimer_bunchmode_software_start
函数原型	void hrtimer_bunchmode_software_start(uint32_t hrtimer_periph);
功能描述	软件启动突发模式
先决条件	-
被调用函数	-
输入参数{in}	

hrtimer_periph	HRTIMER外设
<i>HRTIMERx(x=0)</i>	选择一个HRTIMER外设
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* bunch mode started by software */
```

```
hrtimer_bunchmode_software_start(HRTIMER0);
```

函数 **hrtimer_slavetimer_capture_config**

函数hrtimer_slavetimer_capture_config描述见下表：

表 3-1032. 函数 hrtimer_slavetimer_capture_config

函数名称	hrtimer_slavetimer_capture_config
函数原型	void hrtimer_slavetimer_capture_config(uint32_t hrtimer_periph, uint32_t timer_id, uint32_t capturex, uint64_t trgsource);
功能描述	配置Slave_TIMER捕获源
先决条件	-
被调用函数	-
输入参数{in}	
hrtimer_periph	HRTIMER外设
<i>HRTIMERx(x=0)</i>	选择一个HRTIMER外设
输入参数{in}	
timer_id	选择HRTIMER中的一个定时器
<i>HRTIMER_SLAVE_TIMERx</i>	选择Slave_TIMERx(x=0..7)
输入参数{in}	
capturex	捕获单元索引
<i>HRTIMER_CAPTURE_y(y=0,1)</i>	选择捕获单元y(y=0,1)
输入参数{in}	
trgsource	捕获触发源
<i>HRTIMER_CAPTURET_RIGGER_NONE</i>	清除所有捕获触发源
<i>HRTIMER_CAPTURET_RIGGER_UPDATE</i>	更新事件触发捕获
<i>HRTIMER_CAPTURET_RIGGER_EXEV_y(y=0..9)</i>	外部事件y(y=0..9)触发捕获
<i>HRTIMER_CAPTURET</i>	Slave_TIMERy(y=0..7)的通道0(STyCH0_O)的输出有效电平触发捕获

<i>RIGGER_STy_ACTIVE</i> (y=0..7)	
<i>HRTIMER_CAPTURET</i> <i>RIGGER_STy_INACTI</i> <i>VE</i> (y=0..7)	Slave_TIMERy(y=0..7)的通道0(STyCH0_O)的输出无效电平触发捕获
<i>HRTIMER_CAPTURET</i> <i>RIGGER_STy_CMP0</i> (y =0..7)	Slave_TIMERy(y=0..7)的比较0事件触发捕获
<i>HRTIMER_CAPTURET</i> <i>RIGGER_STy_CMP1</i> (y =0..7)	Slave_TIMERy(y=0..7)的比较1事件触发捕获
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* configure the capture source in Slave_TIMER */
```

```
hrtimer_slavetimer_capture_config(HRTIMER0, HRTIMER_SLAVE_TIMER0,  
HRTIMER_CAPTURE_0, HRTIMER_CAPTURETRIGGER_NONE);
```

函数 hrtimer_slavetimer_capture_software

函数hrtimer_slavetimer_capture_software描述见下表:

表 3-1033. 函数 hrtimer_slavetimer_capture_software

函数名称	hrtimer_slavetimer_capture_software
函数原型	void hrtimer_slavetimer_capture_software(uint32_t hrtimer_periph, uint32_t timer_id, uint32_t capturex);
功能描述	软件触发Slave_TIMER捕获
先决条件	-
被调用函数	-
输入参数{in}	
hrtimer_periph	HRTIMER外设
HRTIMERx(x=0)	选择一个HRTIMER外设
输入参数{in}	
timer_id	选择HRTIMER中的一个定时器
HRTIMER_SLAVE_TIMERx	选择Slave_TIMERx(x=0..7)
输入参数{in}	
capturex	捕获单元索引
HRTIMER_CAPTURE_y(y=0,1)	选择捕获单元y(y=0,1)

输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* configure the capture source in Slave_TIMER */
```

```
hrtimer_slavetimer_capture_software(HRTIMER0, HRTIMER_SLAVE
HRTIMER_CAPTURE_0);
```

函数 hrtimer_slavetimer_capture_value_read

函数hrtimer_slavetimer_capture_value_read描述见下表:

表 3-1034. 函数 hrtimer_slavetimer_capture_value_read

函数名称	hrtimer_slavetimer_capture_value_read
函数原型	hrtimer_capture_value_struct hrtimer_slavetimer_capture_value_read(uint32_t hrtimer_periph, uint32_t timer_id, uint32_t capturex);
功能描述	读取捕获值
先决条件	-
被调用函数	-
输入参数{in}	
hrtimer_periph	HRTIMER外设
HRTIMERx(x=0)	选择一个HRTIMER外设
输入参数{in}	
timer_id	选择HRTIMER中的一个定时器
HRTIMER_SLAVE_TIMERx	选择Slave_TIMERx(x=0..7)
输入参数{in}	
capturex	捕获单元索引
HRTIMER_CAPTURE_y(y=0,1)	选择捕获单元y(y=0,1)
输出参数{out}	
-	-
返回值	
uint32_t	捕获值和方向

例如:

```
/* read the capture value */
```

```
hrtimer_capture_value_struct captured;
```

```
captured = hrtimer_slavetimer_capture_value_read (HRTIMER0, HRTIMER_SLAVE
```

HRTIMER_CAPTURE_0);

函数 `hrtimer_adctrigcfg_struct_para_init`

函数 `hrtimer_adctrigcfg_struct_para_init` 描述见下表:

表 3-1035. 函数 `hrtimer_adctrigcfg_struct_para_init`

函数名称	<code>hrtimer_adctrigcfg_struct_para_init</code>
函数原型	void <code>hrtimer_adctrigcfg_struct_para_init(hrtimer_adctrigcfg_parameter_struct* triggercfg);</code>
功能描述	初始化 <code>hrtimer_adctrigcfg_parameter_struct</code> 结构体变量
先决条件	-
被调用函数	-
输入参数{in}	
<code>triggercfg</code>	<code>hrtimer_adctrigcfg_parameter_struct</code> 结构体指针, 结构体成员参考 表3-956. 结构体 <code>hrtimer_adctrigcfg_parameter_struct</code>
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* initialize ADC trigger configuration struct with a default value */
```

```
hrtimer_adctrigcfg_parameter_struct triggercfg;
```

```
hrtimer_adctrigcfg_struct_para_init(&triggercfg);
```

函数 `hrtimer_adc_trigger0_3_config`

函数 `hrtimer_adc_trigger0_3_config` 描述见下表:

表 3-1036. 函数 `hrtimer_adc_trigger0_3_config`

函数名称	<code>hrtimer_adc_trigger0_3_config</code>
函数原型	void <code>hrtimer_adc_trigger0_3_config(uint32_t hrtimer_periph, uint32_t trigger_id, hrtimer_adctrigcfg_parameter_struct* triggercfg);</code>
功能描述	配置ADC触发源和更新源(0..3)
先决条件	-
被调用函数	-
输入参数{in}	
<code>hrtimer_periph</code>	HRTIMER外设
<code>HRTIMERx(x=0)</code>	选择一个HRTIMER外设
输入参数{in}	
<code>trigger_id</code>	ADC触发源

HRTIMER_ADCTRIG_y (y=0..3)	选择一个ADC触发源
输入参数{in}	
triggercfg	hrtimer_adctrigcfg_parameter_struct结构体指针，结构体成员参考 表3-956. 结构体hrtimer_adctrigcfg_parameter_struct
输出参数{out}	
-	-
返回值	
-	-

例如：

```

/* configure the trigger source to ADC and the update source */

hrtimer_adctrigcfg_parameter_struct triggercfg;

hrtimer_adctrigcfg_struct_para_init(&triggercfg);

triggercfg.update_source = HRTIMER_ADCTRIG_UPDATE_MT;

triggercfg.Trigger[0] = HRTIMER_ADCTRIG02_EVENT_MTCMP0;

triggercfg.Trigger[1] = HRTIMER_ADCTRIG02_EVENT_NONE;

hrtimer_adc_trigger_config(HRTIMER0, HRTIMER_ADCTRIG_0, &triggercfg);

```

函数 hrtimer_adc_trigger4_9_config

函数hrtimer_adc_trigger4_9_config描述见下表：

表 3-1037. 函数 hrtimer_adc_trigger4_9_config

函数名称	hrtimer_adc_trigger4_9_config
函数原型	void hrtimer_adc_trigger4_9_config(uint32_t hrtimer_periph, uint32_t trigger_id, hrtimer_adctrigcfg_parameter_struct* triggercfg);
功能描述	配置ADC触发源和更新源(0..3)
先决条件	-
被调用函数	-
输入参数{in}	
hrtimer_periph	HRTIMER外设
HRTIMERx(x=0)	选择一个HRTIMER外设
输入参数{in}	
trigger_id	ADC触发源
HRTIMER_ADCTRIG_y (y=4..9)	选择一个ADC触发源
输入参数{in}	
triggercfg	hrtimer_adctrigcfg_parameter_struct结构体指针，结构体成员参考 表3-956. 结构体hrtimer_adctrigcfg_parameter_struct

输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* configure the trigger source to ADC and the update source */
hrtimer_adctrigcfg_parameter_struct triggercfg;
hrtimer_adctrigcfg_struct_para_init(&triggercfg);
triggercfg.update_source = HRTIMER_ADCTRIG_UPDATE_MT;
triggercfg.Trigger4_9 = HRTIMER_ADC_EXT_TRIG_579_BY_MASTER_PERIOD;
hrtimer_adc_trigger_config(HRTIMER0, HRTIMER_ADCTRIG_0, &triggercfg);
```

函数 hrtimer_adc_prescaler_config

函数hrtimer_adc_prescaler_config描述见下表：

表 3-1038. 函数 hrtimer_adc_prescaler_config

函数名称	hrtimer_adc_prescaler_config
函数原型	void hrtimer_adc_prescaler_config(uint32_t hrtimer_periph, uint32_t trigger_id, uint32_t psc);
功能描述	配置ADC触发源预分频（0..9）
先决条件	-
被调用函数	-
输入参数{in}	
hrtimer_periph	HRTIMER外设
HRTIMERx(x=0)	选择一个HRTIMER外设
输入参数{in}	
timer_id	选择HRTIMER中的一个定时器
HRTIMER_SLAVE_TIMERx(x=0..7)	选择Slave_TIMERx(x=0..7)
输入参数{in}	
psc	预分频值
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* configure the trigger source prescaler of ADC trigger 0 */
hrtimer_adc_prescaler_config (HRTIMER0, HRTIMER_ADCTRIG_0, 100);
```

函数 hrtimer_slavetimer_counter_direction_get

函数hrtimer_slavetimer_counter_direction_get描述见下表:

表 3-1039. 函数 hrtimer_slavetimer_counter_direction_get

函数名称	hrtimer_slavetimer_counter_direction_get
函数原型	DirectionStatus hrtimer_slavetimer_counter_direction_get(uint32_t hrtimer_periph, uint32_t timer_id);
功能描述	获取Slave_TIMER的计数方向
先决条件	-
被调用函数	-
输入参数{in}	
hrtimer_periph	HRTIMER外设
HRTIMERx(x=0)	选择一个HRTIMER外设
输入参数{in}	
timer_id	选择HRTIMER中的一个定时器
HRTIMER_SLAVE_TIMERx	选择Slave_TIMERx(x=0..7)
输出参数{out}	
-	-
返回值	
DirectionStatus	COUNTER_UP or COUNTER_DOWN

例如:

```
/* get the Slave_TIMER counter direction */
```

```
DirectionStatus dir;
```

```
dir = hrtimer_slavetimer_counter_direction_get(HRTIMER0, HRTIMER_SLAVE_TIMER0);
```

函数 hrtimer_timers_flag_get

函数hrtimer_timers_flag_get描述见下表:

表 3-1040. 函数 hrtimer_timers_flag_get

函数名称	hrtimer_timers_flag_get
函数原型	FlagStatus hrtimer_timers_flag_get(uint32_t hrtimer_periph, uint32_t timer_id, uint32_t flag);
功能描述	获取Master_TIMER/Slave_TIMER的标志
先决条件	-
被调用函数	-
输入参数{in}	
hrtimer_periph	HRTIMER外设
HRTIMERx(x=0)	选择一个HRTIMER外设
输入参数{in}	

timer_id	选择HRTIMER中的一个定时器
<i>HRTIMER_MASTER_TIMER</i>	选择Master_TIMER
<i>HRTIMER_SLAVE_TIMERx</i>	选择Slave_TIMERx(x=0..7)
输入参数{in}	
flag	选择一个标志源
<i>HRTIMER_MT_ST_FLAG_CMPy(y=0..3)</i>	比较y(y=0..3)中断标志（适用于Master_TIMER和Slave_TIMER）
<i>HRTIMER_MT_ST_FLAG_AG_REP</i>	重复中断标志（适用于Master_TIMER 和 Slave_TIMER）
<i>HRTIMER_MT_INT_FLAG_AG_SYNI</i>	同步输入中断标志（仅适用于Master_TIMER）
<i>HRTIMER_MT_ST_FLAG_AG_UPD</i>	更新中断标志（适用于Master_TIMER和Slave_TIMER）
<i>HRTIMER_ST_FLAG_CAPy(y=0,1)</i>	捕获y(y=0,1)中断标志（仅适用于Slave_TIMER）
<i>HRTIMER_ST_FLAG_CHyOA(y=0,1)</i>	通道y(y=0,1)输出置位请求中断标志（仅适用于Slave_TIMER）
<i>HRTIMER_ST_FLAG_CHyONA(y=0,1)</i>	通道y(y=0,1)输出复位请求中断标志（仅适用于Slave_TIMER）
<i>HRTIMER_ST_FLAG_CNTRST</i>	计数器复位中断标志（仅适用于Slave_TIMER）
<i>HRTIMER_ST_FLAG_DLYIDLE</i>	延迟空闲模式进入中断标志（仅适用于Slave_TIMER）
<i>HRTIMER_ST_FLAG_CBLN</i>	当前的均衡状态标志（仅适用于Slave_TIMER）
<i>HRTIMER_ST_FLAG_BLNIDLE</i>	均衡空闲标志（仅适用于Slave_TIMER）
<i>HRTIMER_ST_FLAG_CHyOUTS(y=0,1)</i>	通道 y(y=0,1)输出状态标志（仅适用于Slave_TIMER）
<i>HRTIMER_ST_FLAG_CHyOUT(y=0,1)</i>	通道 y(y=0,1)输出标志（仅适用于Slave_TIMER）
输出参数{out}	
-	-
返回值	
FlagStatus	SET or RESET

例如:

```
/* get the Master_TIMER and Slave_TIMER flag */
```

```
FlagStatus flag = RESET;
```

```
flag = hrtimer_timers_flag_get(HRTIMER0, HRTIMER_MASTER_TIMER,
```

HRTIMER_MT_ST_FLAG_CMP0);

函数 hrtimer_timers_flag_clear

函数hrtimer_timers_flag_clear描述见下表:

表 3-1041. 函数 hrtimer_timers_flag_clear

函数名称	hrtimer_timers_flag_clear
函数原型	void hrtimer_timers_flag_clear(uint32_t hrtimer_periph, uint32_t timer_id, uint32_t flag);
功能描述	clear the Master_TIMER and Slave_TIMER flag
先决条件	-
被调用函数	-
输入参数{in}	
hrtimer_periph	HRTIMER外设
HRTIMERx(x=0)	选择一个HRTIMER外设
输入参数{in}	
timer_id	选择HRTIMER中的一个定时器
HRTIMER_MASTER_TIMER	选择Master_TIMER
HRTIMER_SLAVE_TIMERx	选择Slave_TIMERx(x=0..7)
输入参数{in}	
flag	the flag source
HRTIMER_MT_ST_FLAG_CMPy(y=0..3)	比较y(y=0..3)中断标志（适用于Master_TIMER和Slave_TIMER）
HRTIMER_MT_ST_FLAG_REP	重复中断标志（适用于Master_TIMER和Slave_TIMER）
HRTIMER_MT_INT_FLAG_SYNC	同步输入中断标志（仅适用于Master_TIMER）
HRTIMER_MT_ST_FLAG_UPD	更新中断标志（适用于Master_TIMER和Slave_TIMER）
HRTIMER_ST_FLAG_CAPy(y=0,1)	捕获y(y=0,1)中断标志（仅适用于Slave_TIMER）
HRTIMER_ST_FLAG_CHyOA(y=0,1)	通道y(y=0,1)输出置位请求中断标志（仅适用于Slave_TIMER）
HRTIMER_ST_FLAG_CHyONA(y=0,1)	通道y(y=0,1)输出复位请求中断标志（仅适用于Slave_TIMER）
HRTIMER_ST_FLAG_CNTRST	计数器复位中断标志（仅适用于Slave_TIMER）
HRTIMER_ST_FLAG_DLYIDLE	延迟空闲模式进入中断标志（仅适用于Slave_TIMER）
HRTIMER_ST_FLAG_	当前的均衡状态标志（仅适用于Slave_TIMER）

<i>CBLN</i>	
<i>HRTIMER_ST_FLAG_BLNIDLE</i>	均衡空闲标志（仅适用于Slave_TIMER）
<i>HRTIMER_ST_FLAG_CHyOUTS(y=0,1)</i>	通道 y(y=0,1)输出状态标志（仅适用于Slave_TIMER）
<i>HRTIMER_ST_FLAG_CHyOUT(y=0,1)</i>	通道y(y=0,1)输出标志（仅适用于Slave_TIMER）
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* clear the Master_TIMER and Slave_TIMER flag */
```

```
hrtimer_timers_flag_clear(HRTIMER0, HRTIMER_MASTER_TIMER,  
HRTIMER_MT_ST_FLAG_CMP0);
```

函数 `hrtimer_common_flag_get`

函数 `hrtimer_common_flag_get` 描述见下表：

表 3-1042. 函数 `hrtimer_common_flag_get`

函数名称	<code>hrtimer_common_flag_get</code>
函数原型	<code>FlagStatus hrtimer_common_flag_get(uint32_t hrtimer_periph, uint32_t flag);</code>
功能描述	获取HRTIMER通用标志
先决条件	-
被调用函数	-
输入参数{in}	
hrtimer_periph	HRTIMER外设
<i>HRTIMERx(x=0)</i>	选择一个HRTIMER外设
输入参数{in}	
flag	标志源
<i>HRTIMER_FLAG_FLTy(y=0..7)</i>	故障y(y=0..7)中断标志
<i>HRTIMER_FLAG_SYSFLT</i>	系统故障中断标志
<i>HRTIMER_FLAG_DLLCAL</i>	DLL 校准完成中断标志
<i>HRTIMER_FLAG_BMPER</i>	突发模式周期中断标志
输出参数{out}	
-	-

返回值	
FlagStatus	SET or RESET

例如:

```
/* get the common flag */
```

```
FlagStatus flag = RESET;
```

```
flag = hrtimer_common_flag_get(HRTIMER0, HRTIMER_FLAG_FLT0);
```

函数 **hrtimer_common_flag_clear**

函数hrtimer_common_flag_clear描述见下表:

表 3-1043. 函数 hrtimer_common_flag_clear

函数名称	hrtimer_common_flag_clear
函数原型	void hrtimer_common_flag_clear(uint32_t hrtimer_periph, uint32_t flag);
功能描述	清除HRTIMER通用标志
先决条件	-
被调用函数	-
输入参数{in}	
hrtimer_periph	HRTIMER外设
<i>HRTIMERx(x=0)</i>	选择一个HRTIMER外设
输入参数{in}	
flag	标志源
<i>HRTIMER_FLAG_FLTy</i> (y=0..7)	故障y(y=0..7)中断标志
<i>HRTIMER_FLAG_SYSFLT</i>	系统故障中断标志
<i>HRTIMER_FLAG_DLLCAL</i>	DLL校准完成中断标志
<i>HRTIMER_FLAG_BMPER</i>	突发模式周期中断标志
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* clear the common flag */
```

```
hrtimer_common_flag_clear(HRTIMER0, HRTIMER_FLAG_FLT0);
```

函数 **hrtimer_timers_interrupt_enable**

函数hrtimer_timers_interrupt_enable描述见下表:

表 3-1044. 函数 `hrtimer_timers_interrupt_enable`

函数名称	<code>hrtimer_timers_interrupt_enable</code>
函数原型	<code>void hrtimer_timers_interrupt_enable(uint32_t hrtimer_periph, uint32_t timer_id, uint32_t interrupt);</code>
功能描述	使能Master_TIMER/Slave_TIMER中断
先决条件	-
被调用函数	-
输入参数{in}	
<code>hrtimer_periph</code>	HRTIMER外设
<code>HRTIMERx(x=0)</code>	选择一个HRTIMER外设
输入参数{in}	
<code>timer_id</code>	选择HRTIMER中的一个定时器
<code>HRTIMER_MASTER_TIMER</code>	选择Master_TIMER
<code>HRTIMER_SLAVE_TIMERx</code>	选择Slave_TIMERx(x=0..7)
输入参数{in}	
<code>interrupt</code>	中断源
<code>HRTIMER_MT_ST_INT_CMPy(y=0..3)</code>	比较y(y=0..3)中断（适用于Master_TIMER和Slave_TIMER）
<code>HRTIMER_MT_ST_INT_REP</code>	重复中断（适用于Master_TIMER和Slave_TIMER）
<code>HRTIMER_MT_INT_SYNC</code>	同步输入中断（仅适用于Master_TIMER）
<code>HRTIMER_MT_ST_INT_UPD</code>	更新中断标志（适用于Master_TIMER和Slave_TIMER）
<code>HRTIMER_ST_INT_CAPy(y=0,1)</code>	捕获y(y=0,1)中断（仅适用于Slave_TIMER）
<code>HRTIMER_ST_INT_CHyOA(y=0,1)</code>	通道y(y=0,1)输出置位请求中断（仅适用于Slave_TIMER）
<code>HRTIMER_ST_INT_CHyONA(y=0,1)</code>	通道y(y=0,1)输出复位请求中断（仅适用于Slave_TIMER）
<code>HRTIMER_ST_INT_CNTRST</code>	计数器复位中断（仅适用于Slave_TIMER）
<code>HRTIMER_ST_INT_DLYIDLE</code>	延迟空闲模式进入中断（仅适用于Slave_TIMER）
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* enable the Master_TIMER and Slave_TIMER interrupt */
```

```
hrtimer_timers_interrupt_enable(HRTIMER0, HRTIMER_MASTER_TIMER,
HRTIMER_MT_ST_INT_CMP0);
```

函数 hrtimer_timers_interrupt_disable

函数hrtimer_timers_interrupt_disable描述见下表：

表 3-1045. 函数 hrtimer_timers_interrupt_disable

函数名称	hrtimer_timers_interrupt_disable
函数原型	void hrtimer_timers_interrupt_disable(uint32_t hrtimer_periph, uint32_t timer_id, uint32_t interrupt);
功能描述	禁能Master_TIMER/Slave_TIMER中断
先决条件	-
被调用函数	-
输入参数{in}	
hrtimer_periph	HRTIMER外设
HRTIMERx(x=0)	选择一个HRTIMER外设
输入参数{in}	
timer_id	选择HRTIMER中的一个定时器
HRTIMER_MASTER_TIMER	选择Master_TIMER
HRTIMER_SLAVE_TIMERx	选择Slave_TIMERx(x=0..7)
输入参数{in}	
interrupt	中断源
HRTIMER_MT_ST_INT_CMPy(y=0..3)	比较y(y=0..3)中断（适用于Master_TIMER和Slave_TIMER）
HRTIMER_MT_ST_INT_REP	重复中断（适用于Master_TIMER和Slave_TIMER）
HRTIMER_MT_INT_SYNC	同步输入中断（仅适用于Master_TIMER）
HRTIMER_MT_ST_INT_UPD	更新中断标志（适用于Master_TIMER和Slave_TIMER）
HRTIMER_ST_INT_CAPy(y=0,1)	捕获y(y=0,1)中断（仅适用于Slave_TIMER）
HRTIMER_ST_INT_CHyOA(y=0,1)	通道y(y=0,1)输出置位请求中断（仅适用于Slave_TIMER）
HRTIMER_ST_INT_CHyONA(y=0,1)	通道y(y=0,1)输出复位请求中断（仅适用于Slave_TIMER）
HRTIMER_ST_INT_CNTRST	计数器复位中断（仅适用于Slave_TIMER）
HRTIMER_ST_INT_DLYIDLE	延迟空闲模式进入中断（仅适用于Slave_TIMER）

输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* disable the Master_TIMER and Slave_TIMER interrupt */
```

```
hrtimer_timers_interrupt_disable(HRTIMER0, HRTIMER_MASTER_TIMER,
HRTIMER_MT_ST_INT_CMP0);
```

函数 hrtimer_timers_interrupt_flag_get

函数hrtimer_timers_interrupt_flag_get描述见下表：

表 3-1046. 函数 hrtimer_timers_interrupt_flag_get

函数名称	hrtimer_timers_interrupt_flag_get
函数原型	FlagStatus hrtimer_timers_interrupt_flag_get(uint32_t hrtimer_periph, uint32_t timer_id, uint32_t interrupt);
功能描述	获取Master_TIMER/Slave_TIMER中断标志
先决条件	-
被调用函数	-
输入参数{in}	
hrtimer_periph	HRTIMER外设
HRTIMERx(x=0)	选择一个HRTIMER外设
输入参数{in}	
timer_id	选择HRTIMER中的一个定时器
HRTIMER_MASTER_TIMER	选择Master_TIMER
HRTIMER_SLAVE_TIMERx	选择Slave_TIMERx(x=0..7)
输入参数{in}	
interrupt	中断源
HRTIMER_MT_ST_INT_FLAG_CMPy(y=0..3)	比较y(y=0..3)中断标志（适用于Master_TIMER和Slave_TIMER）
HRTIMER_MT_ST_INT_FLAG_REP	重复中断标志（适用于Master_TIMER和Slave_TIMER）
HRTIMER_MT_INT_FLAG_SYNC	同步输入中断标志（仅适用于Master_TIMER）
HRTIMER_MT_ST_INT_FLAG_UPD	更新中断标志（适用于Master_TIMER和Slave_TIMER）
HRTIMER_ST_INT_FLAG_CAPy(y=0,1)	捕获y(y=0,1)中断标志（仅适用于Slave_TIMER）
HRTIMER_ST_INT_FLAG	通道y(y=0,1)输出置位请求中断标志（仅适用于Slave_TIMER）

AG_CHyOA(y=0,1)	
HRTIMER_ST_INT_FL AG_CHyONA(y=0,1)	通道y(y=0,1)输出复位请求中断标志（仅适用于Slave_TIMER）
HRTIMER_ST_INT_FL AG_CNTRST	计数器复位中断标志（仅适用于Slave_TIMER）
HRTIMER_ST_INT_FL AG_DLYIDLE	延迟空闲模式进入中断标志（仅适用于Slave_TIMER）
输出参数{out}	
-	-
返回值	
FlagStatus	SET or RESET

例如：

```
/* get the Master_TIMER and Slave_TIMER interrupt flag */
```

```
FlagStatus flag = RESET;
```

```
flag = hrtimer_timers_interrupt_flag_get(HRTIMER0, HRTIMER_MASTER_TIMER,  
HRTIMER_MT_ST_INT_FLAG_CMP0);
```

函数 hrtimer_timers_interrupt_flag_clear

函数hrtimer_timers_interrupt_flag_clear描述见下表：

表 3-1047. 函数 hrtimer_timers_interrupt_flag_clear

函数名称	hrtimer_timers_interrupt_flag_clear
函数原型	void hrtimer_timers_interrupt_flag_clear(uint32_t hrtimer_periph, uint32_t timer_id, uint32_t interrupt);
功能描述	清除Master_TIMER/Slave_TIMER中断标志
先决条件	-
被调用函数	-
输入参数{in}	
hrtimer_periph	HRTIMER外设
HRTIMERx(x=0)	选择一个HRTIMER外设
输入参数{in}	
timer_id	选择HRTIMER中的一个定时器
HRTIMER_MASTER_TIMER	选择Master_TIMER
HRTIMER_SLAVE_TIMERx	选择Slave_TIMERx(x=0..7)
输入参数{in}	
interrupt	中断源
HRTIMER_MT_ST_INT_FLAG_CMPy(y=0..3)	比较y(y=0..3)中断标志（适用于Master_TIMER和Slave_TIMER）

<code>HRTIMER_MT_ST_INT_FLAG_REP</code>	重复中断标志（适用于Master_TIMER和Slave_TIMER）
<code>HRTIMER_MT_INT_FLAG_SYNC</code>	同步输入中断标志（仅适用于Master_TIMER）
<code>HRTIMER_MT_ST_INT_FLAG_UPD</code>	更新中断标志（适用于Master_TIMER和Slave_TIMER）
<code>HRTIMER_ST_INT_FLAG_CAPTURE(y=0,1)</code>	捕获y(y=0,1)中断标志（仅适用于Slave_TIMER）
<code>HRTIMER_ST_INT_FLAG_CHyOA(y=0,1)</code>	通道y(y=0,1)输出置位请求中断标志（仅适用于Slave_TIMER）
<code>HRTIMER_ST_INT_FLAG_CHyONA(y=0,1)</code>	通道y(y=0,1)输出复位请求中断标志（仅适用于Slave_TIMER）
<code>HRTIMER_ST_INT_FLAG_CNTRST</code>	计数器复位中断标志（仅适用于Slave_TIMER）
<code>HRTIMER_ST_INT_FLAG_DLYIDLE</code>	延迟空闲模式进入中断标志（仅适用于Slave_TIMER）
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* clear the Master_TIMER and Slave_TIMER interrupt flag */
```

```
hrtimer_timers_interrupt_flag_clear(HRTIMER0, HRTIMER_MASTER_TIMER,
HRTIMER_MT_ST_INT_FLAG_CMP0);
```

函数 `hrtimer_common_interrupt_enable`

函数 `hrtimer_common_interrupt_enable` 描述见下表：

表 3-1048. 函数 `hrtimer_common_interrupt_enable`

函数名称	<code>hrtimer_common_interrupt_enable</code>
函数原型	<code>void hrtimer_common_interrupt_enable(uint32_t hrtimer_periph, uint32_t interrupt);</code>
功能描述	使能HRTIMER通用中断
先决条件	-
被调用函数	-
输入参数{in}	
<code>hrtimer_periph</code>	HRTIMER外设
<code>HRTIMERx(x=0)</code>	选择一个HRTIMER外设
输入参数{in}	
<code>interrupt</code>	中断源
<code>HRTIMER_INT_FLTy(y</code>	故障y(y=0..7)中断

$=0..7)$	
<i>HRTIMER_INT_SYSFLT</i> <i>T</i>	系统故障中断
<i>HRTIMER_INT_DLLCAL</i> <i>L</i>	DLL校准完成中断
<i>HRTIMER_INT_BMPE</i> <i>R</i>	突发模式周期中断
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* enable HRTIMER common interrupt */
```

```
hrtimer_common_interrupt_enable(HRTIMER0, HRTIMER_INT_FLT0);
```

函数 `hrtimer_common_interrupt_disable`

函数 `hrtimer_common_interrupt_disable` 描述见下表:

表 3-1049. 函数 `hrtimer_common_interrupt_disable`

函数名称	<code>hrtimer_common_interrupt_disable</code>
函数原型	<code>void hrtimer_common_interrupt_disable(uint32_t hrtimer_periph, uint32_t interrupt);</code>
功能描述	禁用HRTIMER通用中断
先决条件	-
被调用函数	-
输入参数{in}	
hrtimer_periph	HRTIMER外设
<i>HRTIMERx</i> ($x=0$)	选择一个HRTIMER外设
输入参数{in}	
interrupt	中断源
<i>HRTIMER_INT_FLTy</i> ($y=0..7)$	故障 y ($y=0..7$)中断
<i>HRTIMER_INT_SYSFLT</i> <i>T</i>	系统故障中断
<i>HRTIMER_INT_DLLCAL</i> <i>L</i>	DLL校准完成中断
<i>HRTIMER_INT_BMPE</i> <i>R</i>	突发模式周期中断
输出参数{out}	
-	-
返回值	

-	-
---	---

例如:

```
/* disable HRTIMER common interrupt */
```

```
hrtimer_common_interrupt_disable(HRTIMER0, HRTIMER_INT_FLAG0);
```

函数 `hrtimer_common_interrupt_flag_get`

函数 `hrtimer_common_interrupt_flag_get` 描述见下表:

表 3-1050. 函数 `hrtimer_common_interrupt_flag_get`

函数名称	<code>hrtimer_common_interrupt_flag_get</code>
函数原型	<code>FlagStatus hrtimer_common_interrupt_flag_get(uint32_t hrtimer_periph, uint32_t interrupt);</code>
功能描述	获取HRTIMER通用中断标志
先决条件	-
被调用函数	-
输入参数{in}	
hrtimer_periph	HRTIMER外设
<code>HRTIMERx(x=0)</code>	选择一个HRTIMER外设
输入参数{in}	
interrupt	中断标志源
<code>HRTIMER_INT_FLAG_FLTy(y=0..7)</code>	故障y(y=0..7)中断标志
<code>HRTIMER_INT_FLAG_SYSFLT</code>	系统故障中断标志
<code>HRTIMER_INT_FLAG_DLLCAL</code>	DLL校准完成中断标志
<code>HRTIMER_INT_FLAG_BMPER</code>	突发模式周期中断标志
输出参数{out}	
-	-
返回值	
FlagStatus	SET or RESET

例如:

```
/* get the common interrupt flag */
```

```
FlagStatus flag = RESET;
```

```
flag = hrtimer_common_interrupt_flag_get(HRTIMER0, HRTIMER_INT_FLAG0);
```

函数 `hrtimer_common_interrupt_flag_clear`

函数 `hrtimer_common_interrupt_flag_clear` 描述见下表:

表 3-1051. 函数 `hrtimer_common_interrupt_flag_clear`

函数名称	<code>hrtimer_common_interrupt_flag_clear</code>
函数原型	<code>void hrtimer_common_interrupt_flag_clear(uint32_t hrtimer_periph, uint32_t interrupt);</code>
功能描述	清除HRTIMER通用中断标志
先决条件	-
被调用函数	-
输入参数{in}	
<code>hrtimer_periph</code>	HRTIMER外设
<code>HRTIMERx(x=0)</code>	选择一个HRTIMER外设
输入参数{in}	
<code>interrupt</code>	中断标志源
<code>HRTIMER_INT_FLAG_FLTy(y=0..7)</code>	故障y(y=0..7)中断标志
<code>HRTIMER_INT_FLAG_SYSFLT</code>	系统故障中断标志
<code>HRTIMER_INT_FLAG_DLLCAL</code>	DLL校准完成中断标志
<code>HRTIMER_INT_FLAG_BMPER</code>	突发模式周期中断标志
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* clear the common interrupt flag */
```

```
hrtimer_common_interrupt_flag_clear(HRTIMER0, HRTIMER_INT_FLAG_FLT0);
```

3.28. SPI

SPI模块可以通过SPI协议与外部设备进行通信。章节[3.28.1](#)描述了SPI的寄存器列表，章节[3.28.2](#)对SPI库函数进行说明。

3.28.1. 外设寄存器说明

SPI寄存器列表如下表所示：

表 3-1052. SPI寄存器

寄存器名称	寄存器描述
<code>SPI_CTL0</code>	控制寄存器0
<code>SPI_CTL1</code>	控制寄存器1

寄存器名称	寄存器描述
SPI_STAT	状态寄存器
SPI_DATA	数据寄存器
SPI_CRCPOLY	CRC多项式寄存器
SPI_RCRC	接收CRC寄存器
SPI_TCRC	发送CRC寄存器
SPI_QCTL	四线SPI控制寄存器

3.28.2. 外设库函数说明

SPI库函数列表如下表所示：

表 3-1053. SPI 库函数

库函数名称	库函数描述
spi_deinit	复位外设SPI
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	失能外设SPI
spi_nss_output_enable	使能外设SPI NSS输出
spi_nss_output_disable	失能外设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	失能外设SPI的DMA功能
spi_transmit_odd_config	配置SPI通过DMA发送的数据总数是否为奇数
spi_receive_odd_config	配置SPI通过DMA接收的数据总数是否为奇数
spi_data_frame_format_config	配置外设SPI数据帧格式
spi_fifo_access_size_config	配置SPI访问FIFO的大小（8位或16位）
spi_bidirectional_transfer_config	配置外设SPI的数据传输方向
spi_data_transmit	发送数据
spi_data_receive	接收数据
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_set	设置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_next	设置外设SPI下一次传输数据为CRC值
spi_crc_get	外设SPI获取CRC值
spi_crc_error_clear	清除SPI CRC错误标志状态
spi_ti_mode_enable	使能SPI TI模式
spi_ti_mode_disable	禁能SPI TI模式
spi_nssp_mode_enable	使能SPI NSS脉冲模式

库函数名称	库函数描述
spi_nssp_mode_disable	禁能SPI NSS脉冲模式
spi_quad_enable	使能四线SPI模式
spi_quad_disable	禁能四线SPI模式
spi_quad_write_enable	使能四线SPI写
spi_quad_read_enable	使能四线SPI读
spi_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_format_error_clear	清除SPI格式错误标志
spi_flag_get	获取外设SPI标志状态
spi_interrupt_enable	使能外设SPI中断
spi_interrupt_disable	失能外设SPI中断
spi_interrupt_flag_get	获取外设SPI中断状态

结构体 spi_parameter_struct

表 3-1054. 结构体 spi_parameter_struct

成员名称	功能描述
device_mode	主机或设备模式配置 (SPI_MASTER, SPI_SLAVE)
trans_mode	传输模式 (SPI_TRANSMODE_FULLDUPLEX, SPI_TRANSMODE_RECEIVEONLY, SPI_TRANSMODE_BDRECEIVE, SPI_TRANSMODE_BDTRANSMIT)
frame_size	数据帧格式配置 (SPI_FRAME_SIZE_xBIT, x=4,5,..16)
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_deinit

函数spi_deinit描述见下表：

表 3-1055. 函数 spi_deinit

函数名称	spi_deinit
函数原形	void spi_deinit(uint32_t spi_periph);
功能描述	复位外设SPI
先决条件	-

被调用函数	rcu_periph_reset_enable / rcu_periph_reset_disable
输入参数{in}	
spi_periph	外设SPIx
SPIx	x=0,1,2
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* reset SPI0 */
```

```
spi_deinit(SPI0);
```

函数 spi_struct_para_init

函数spi_struct_para_init描述见下表：

表 3-1056. 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3-1054. 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 3-1057. 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=0,1,2
输入参数{in}	
spi_struct	初始化结构体，结构体成员参考 表3-1054. 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_FRAMESIZE_8BIT;
spi_init_struct.clock_polarity_phase = SPI_CK_PL_HIGH_PH_2EDGE;
spi_init_struct.nss             = SPI_NSS_SOFT;
spi_init_struct.prescale        = SPI_PSC_8;
spi_init_struct.endian          = SPI_ENDIAN_MSB;

spi_init(SPI0, &spi_init_struct);

```

函数 spi_enable

函数spi_enable描述见下表：

表 3-1058. 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=0,1,2
输出参数{out}	
-	-

返回值	
-	-

例如：

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

```
spi_enable(SPI0);
```

函数 spi_disable

函数spi_disable描述见下表：

表 3-1059. 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=0,1,2
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* disable SPI0 */
```

```
spi_disable(SPI0);
```

函数 spi_nss_output_enable

函数spi_nss_output_enable描述见下表：

表 3-1060. 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
SPIx	x=0,1,2
输出参数{out}	

-	-
返回值	
-	-

例如：

```
/* enable SPI0 NSS output */
```

```
spi_nss_output_enable(SPI0);
```

函数 spi_nss_output_disable

函数spi_nss_output_disable描述见下表：

表 3-1061. 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=0,1,2
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* disable SPI0 NSS output */
```

```
spi_nss_output_disable(SPI0);
```

函数 spi_nss_internal_high

函数spi_nss_internal_high描述见下表：

表 3-1062. 函数 spi_nss_internal_high

函数名称	spi_nss_internal_high
函数原形	void spi_nss_internal_high(uint32_t spi_periph);
功能描述	NSS软件模式下NSS引脚拉高
先决条件	-
被调用函数	-
输入参数{in}	
spi_periph	外设SPIx
SPIx	x=0,1,2

输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* SPI0 NSS pin is pulled high level in software mode */
```

```
spi_nss_internal_high(SPI0);
```

函数 spi_nss_internal_low

函数spi_nss_internal_low描述见下表：

表 3-1063. 函数 spi_nss_internal_low

函数名称	spi_nss_internal_low
函数原形	void spi_nss_internal_low(uint32_t spi_periph);
功能描述	NSS软件模式下NSS引脚拉低
先决条件	-
被调用函数	-
输入参数{in}	
spi_periph	外设SPIx
SPIx	x=0,1,2
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* SPI0 NSS pin is pulled low level in software mode */
```

```
spi_nss_internal_low(SPI0);
```

函数 spi_dma_enable

函数spi_dma_enable描述见下表：

表 3-1064. 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

<i>SPIx</i>	<i>x</i> =0,1,2
输入参数{in}	
spi_dma	SPI DMA模式
<i>SPI_DMA_TRANSMIT</i>	SPI发送缓冲区DMA使能
<i>SPI_DMA_RECEIVE</i>	SPI接收缓冲区DMA使能
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* enable SPI0 transmit data DMA function */
```

```
spi_dma_enable(SPI0, SPI_DMA_TRANSMIT);
```

函数 spi_dma_disable

函数spi_dma_disable描述见下表：

表 3-1065. 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
<i>SPIx</i>	<i>x</i> =0,1,2
输入参数{in}	
spi_dma	SPI DMA模式
<i>SPI_DMA_TRANSMIT</i>	SPI发送缓冲区DMA失能
<i>SPI_DMA_RECEIVE</i>	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_transmit_odd_config`

函数 `spi_transmit_odd_config` 描述见下表:

表 3-1066. 函数 `spi_transmit_odd_config`

函数名称	<code>spi_transmit_odd_config</code>
函数原形	<code>void spi_transmit_odd_config(uint32_t spi_periph, uint16_t odd);</code>
功能描述	配置SPI0通过DMA发送的数据总数是否为奇数
先决条件	-
被调用函数	-
输入参数{in}	
<code>spi_periph</code>	外设SPI
<code>SPIx</code>	<code>x=0,1,2</code>
输入参数{in}	
<code>odd</code>	DMA通道发送的字节数是奇数还是偶数
<code>SPI_TXDMA_EVEN</code>	DMA发送的字节数是偶数
<code>SPI_TXDMA_ODD</code>	DMA发送的字节数是奇数
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* configure SPI0 total number of data to transmit by DMA is odd */
```

```
spi_transmit_odd_config(SPI0, SPI_TXDMA_ODD);
```

函数 `spi_receive_odd_config`

函数 `spi_receive_odd_config` 描述见下表:

表 3-1067. 函数 `spi_receive_odd_config`

函数名称	<code>spi_receive_odd_config</code>
函数原形	<code>void spi_receive_odd_config(uint32_t spi_periph, uint16_t odd);</code>
功能描述	配置SPI0通过DMA接收到的数据总数是否为奇数
先决条件	-
被调用函数	-
输入参数{in}	
<code>spi_periph</code>	外设SPI
<code>SPIx</code>	<code>x=0,1,2</code>
输入参数{in}	
<code>odd</code>	DMA通道接收的字节数时奇数还是偶数
<code>SPI_RXDMA_EVEN</code>	DMA接收的字节数是偶数

<i>SPI_RXDMA_ODD</i>	DMA接收的字节数是奇数
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* configure SPI0 total number of data to receive by DMA is odd */
```

```
spi_receive_odd_config(SPI0, SPI_RXDMA_ODD);
```

函数 **spi_data_frame_format_config**

函数spi_data_frame_format_config描述见下表：

表 3-1068. 函数 spi_data_frame_format_config

函数名称	spi_data_frame_format_config
函数原形	ErrStatus spi_data_frame_format_config(uint32_t spi_periph, uint16_t frame_format);
功能描述	配置外设SPI数据帧格式
先决条件	-
被调用函数	-
输入参数{in}	
spi_periph	外设SPI
<i>SPIx</i>	x=0,1,2
输入参数{in}	
frame_format	SPI帧大小
<i>SPI_FRAME_SIZE_x</i> <i>BIT</i>	SPI x位数据帧格式，x=4,5,..16
输出参数{out}	
-	-
返回值	
ErrStatus	ERROR或者SUCCESS-

例如：

```
/* configure SPI0 data frame format size is 16 bits */
```

```
spi_data_frame_format_config(SPI1, SPI_FRAME_SIZE_16BIT);
```

函数 **spi_fifo_access_size_config**

函数spi_fifo_access_size_config描述见下表：

表 3-1069. 函数 spi_fifo_access_size_config

函数名称	spi_fifo_access_size_config
------	-----------------------------

函数原形	void spi_fifo_access_size_config(uint32_t spi_periph, uint16_t fifo_access_size);
功能描述	配置SPI0的FIFO访问大小
先决条件	-
被调用函数	-
输入参数{in}	
spi_periph	外设SPI
SPIx	x=0,1,2
输入参数{in}	
fifo_access_size	fifo访问大小
SPI_HALFWORD_ACCESS	半字访问
SPI_BYTE_ACCESS	字节访问
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* configure SPI0 access size half word */
```

```
spi_fifo_access_size_config(SPI0, SPI_HALFWORD_ACCESS);
```

函数 spi_bidirectional_transfer_config

函数spi_bidirectional_transfer_config描述见下表：

表 3-1070. 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SPI
SPIx	x=0,1,2
输入参数{in}	
transfer_direction	SPI双向传输输出使能
SPI_BIDIRECTIONAL_TRANSMIT	SPI工作在只发送模式
SPI_BIDIRECTIONAL_RECEIVE	SPI工作在只接收模式

输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* SPI0 works in transmit-only mode */
```

```
spi_bidirectional_transfer_config(SPI0, SPI_BIDIRECTIONAL_TRANSMIT);
```

函数 spi_data_transmit

函数spi_data_transmit描述见下表：

表 3-1071. 函数 spi_data_transmit

函数名称	spi_data_transmit
函数原形	void spi_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
SPIx	x=0,1,2
输入参数{in}	
data	32位数据
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* SPI0 transmit data */
```

```
spi_data_transmit(SPI0, spi0_send_array[send_n]);
```

函数 spi_data_receive

函数spi_data_receive描述见下表：

表 3-1072. 函数 spi_data_receive

函数名称	spi_data_receive
函数原形	uint32_t spi_data_receive(uint32_t spi_periph);
功能描述	SPI接收数据
先决条件	-
被调用函数	-

输入参数{in}	
spi_periph	外设SPIx
<i>SPIx</i>	x=0,1,2
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_data_receive(SPI0);
```

函数 spi_crc_polynomial_set

函数spi_crc_polynomial_set描述见下表:

表 3-1073. 函数 spi_crc_polynomial_set

函数名称	spi_crc_polynomial_set
函数原形	void spi_crc_polynomial_set(uint32_t spi_periph, uint32_t crc_poly);
功能描述	设置外设SPI的CRC多项式值
先决条件	-
被调用函数	-
输入参数{in}	
spi_periph	外设SPIx
<i>SPIx</i>	x=0,1,2
输入参数{in}	
crc_poly	CRC多项式值
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* set SPI0 CRC polynomial */
```

```
spi_crc_polynomial_set(SPI0,CRC_VALUE);
```

函数 spi_crc_polynomial_get

函数spi_crc_polynomial_get描述见下表:

表 3-1074. 函数 spi_crc_polynomial_get

函数名称	spi_crc_polynomial_get
函数原形	uint32_t spi_crc_polynomial_get(uint32_t spi_periph);

功能描述	获取外设SPI的CRC多项式值
先决条件	-
被调用函数	-
输入参数{in}	
spi_periph	外设SPIx
<i>SPIx</i>	x=0,1,2
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_set

函数spi_crc_length_set描述见下表：

表 3-1075. 函数 spi_crc_length_set

函数名称	spi_crc_length_set
函数原形	void spi_crc_length_set(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
<i>SPIx</i>	x=0,1,2
输入参数{in}	
crc_size	CRC长度
<i>SPI_CRC_8BIT</i>	CRC长度为8位数据
<i>SPI_CRC_16BIT</i>	CRC长度为16位数据
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* config SPI0 CRC 16-bit length */
spi_crc_length_set(SPI0, SPI_CRC_16BIT);
```

函数 spi_crc_on

函数spi_crc_on描述见下表：

表 3-1076. 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=0,1,2
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* turn on SPI0 CRC function */
```

```
spi_crc_on(SPI0);
```

函数 spi_crc_off

函数spi_crc_off描述见下表：

表 3-1077. 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=0,1,2
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* turn off SPI0 CRC function */
```

```
spi_crc_off(SPI0);
```


函数 spi_crc_next

函数spi_crc_next描述见下表:

表 3-1078. 函数 spi_crc_next

函数名称	spi_crc_next
函数原形	void spi_crc_next(uint32_t spi_periph);
功能描述	设置外设SPI下一次传输数据为CRC值
先决条件	-
被调用函数	-
输入参数{in}	
spi_periph	外设SPI
SPIx	x=0,1,2
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* SPI0 next data is CRC value */
```

```
spi_crc_next(SPI0);
```

函数 spi_crc_get

函数spi_crc_get描述见下表:

表 3-1079. 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=0,1,2
输入参数{in}	
crc	SPI crc值
SPI_CRC_TX	获取发送CRC寄存器值
SPI_CRC_RX	获取接收CRC寄存器值
输出参数{out}	
-	-
返回值	
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_error_clear

函数spi_crc_error_clear描述见下表：

表 3-1080. 函数 spi_crc_error_clear

函数名称	spi_crc_error_clear
函数原形	void spi_crc_error_clear(uint32_t spi_periph);
功能描述	清除SPI CRC错误标志状态
先决条件	-
被调用函数	-
输入参数{in}	
spi_periph	外设SPI
SPIx	x=0,1,2
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* clear SPI0 CRC error flag status */

spi_crc_error_clear(SPI0);
```

函数 spi_ti_mode_enable

函数spi_ti_mode_enable描述见下表：

表 3-1081. 函数 spi_ti_mode_enable

函数名称	spi_ti_mode_enable
函数原形	void spi_ti_mode_enable(uint32_t spi_periph);
功能描述	使能SPI TI模式
先决条件	-
被调用函数	-
输入参数{in}	
spi_periph	外设SPIx
SPIx	x=0,1,2
输出参数{out}	
-	-

返回值	
-	-

例如：

```
/* enable SPI0 TI mode */
spi_ti_mode_enable(SPI0);
```

函数 spi_ti_mode_disable

函数spi_ti_mode_disable描述见下表：

表 3-1082. 函数 spi_ti_mode_disable

函数名称	spi_ti_mode_disable
函数原形	void spi_ti_mode_disable(uint32_t spi_periph);
功能描述	禁能SPI TI模式
先决条件	-
被调用函数	-
输入参数{in}	
spi_periph	外设SPIx
SPIx	x=0,1,2
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* disable SPI0 TI mode */
spi_ti_mode_disable(SPI0);
```

函数 spi_nssp_mode_enable

函数spi_nssp_mode_enable描述见下表：

表 3-1083. 函数 spi_nssp_mode_enable

函数名称	spi_nssp_mode_enable
函数原形	void spi_nssp_mode_enable(uint32_t spi_periph);
功能描述	使能SPI NSS脉冲模式
先决条件	-
被调用函数	-
输入参数{in}	
spi_periph	外设SPI
SPIx	x=0,1,2
输出参数{out}	

-	-
返回值	
-	-

例如：

```
/* enable SPI0 NSS pulse mode */
```

```
spi_nssp_mode_enable(SPI0);
```

函数 spi_nssp_mode_disable

函数spi_nssp_mode_disable描述见下表：

表 3-1084. 函数 spi_nssp_mode_disable

函数名称	spi_nssp_mode_disable
函数原形	void spi_nssp_mode_disable(uint32_t spi_periph);
功能描述	失能SPI NSS脉冲模式
先决条件	-
被调用函数	-
输入参数{in}	
spi_periph	外设SPI
SPIx	x=0,1,2
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* disable SPI0 NSS pulse mode */
```

```
spi_nssp_mode_disable(SPI0);
```

函数 spi_quad_enable

函数spi_quad_enable描述见下表：

表 3-1085. 函数 spi_quad_enable

函数名称	spi_quad_enable
函数原形	void spi_quad_enable(uint32_t spi_periph);
功能描述	使能四线SPI模式
先决条件	-
被调用函数	-
输入参数{in}	
spi_periph	外设SPIx
SPIx	x=0

输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* enable SPI0 quad wire mode */
```

```
spi_quad_enable(SPI0);
```

函数 spi_quad_disable

函数spi_quad_disable描述见下表：

表 3-1086. 函数 spi_quad_disable

函数名称	spi_quad_disable
函数原形	void spi_quad_disable(uint32_t spi_periph);
功能描述	禁能四线SPI模式
先决条件	-
被调用函数	-
输入参数{in}	
spi_periph	外设SPIx
SPIx	x=0
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* disable SPI0 quad wire mode */
```

```
spi_quad_disable(SPI0);
```

函数 spi_quad_write_enable

函数spi_quad_write_enable描述见下表：

表 3-1087. 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

<i>SPIx</i>	<i>x=0</i>
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* enable SPI0 quad wire write */
```

```
spi_quad_write_enable(SPI0);
```

函数 spi_quad_read_enable

函数spi_quad_read_enable描述见下表：

表 3-1088. 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
<i>SPIx</i>	<i>x=0</i>
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* enable SPI0 quad wire read */
```

```
spi_quad_read_enable(SPI0);
```

函数 spi_quad_io23_output_enable

函数spi_quad_io23_output_enable描述见下表：

表 3-1089. 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
<i>SPIx</i>	x=0
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* enable SPI3 SPI_IO2 and SPI_IO3 pin output */
```

```
spi_quad_io23_output_enable(SPI0);
```

函数 spi_quad_io23_output_disable

函数spi_quad_io23_output_disable描述见下表：

表 3-1090. 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
<i>SPIx</i>	x=0
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* disable SPI3 SPI_IO2 and SPI_IO3 pin output */
```

```
spi_quad_io23_output_disable(SPI0);
```

函数 spi_format_error_clear

函数spi_format_error_clear描述见下表：

表 3-1091. 函数 spi_format_error_clear

函数名称	spi_format_error_clear
函数原形	void spi_format_error_clear(uint32_t spi_periph, uint32_t flag);
功能描述	清除SPI格式错误标志
先决条件	-
被调用函数	-

输入参数{in}	
spi_periph	外设SPI
<i>SPIx</i>	x=0,1,2
输入参数{in}	
flag	SPI/格式错误标志
<i>SPI_FLAG_FERR</i>	SPI格式错误标志
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* clear SPI format error flag status */
```

```
spi_format_error_clear(SPI0, SPI_FLAG_FERR);
```

函数 **spi_flag_get**

函数spi_flag_get描述见下表:

表 3-1092. 函数 spi_flag_get

函数名称	spi_flag_get
函数原形	FlagStatus spi_flag_get(uint32_t spi_periph, uint32_t flag);
功能描述	获取外设SPI标志状态
先决条件	-
被调用函数	-
输入参数{in}	
spi_periph	外设SPI
<i>SPIx</i>	x=0,1,2
输入参数{in}	
flag	SPI标志状态
<i>SPI_FLAG_TBE</i>	SPI发送缓冲区空标志
<i>SPI_FLAG_RBNE</i>	SPI接收缓冲区非空标志
<i>SPI_FLAG_TRANS</i>	SPI通信进行中标志
<i>SPI_FLAG_RXORERR</i>	SPI接收过载错误标志
<i>SPI_FLAG_CONFERR</i>	SPI配置错误标志
<i>SPI_FLAG_CRCERR</i>	SPI CRC错误标志
<i>SPI_FLAG_FERR</i>	SPI格式错误标志
以下参数只适用于SPI0	
<i>SPI_TXLVL_EMPTY</i>	SPI TXFIFO空

<i>SPI_TXLVL_QUARTER_FULL</i>	SPI TXFIFO四分之一满
<i>SPI_TXLVL_HALF_FULL</i>	SPI TXFIFO半满
<i>SPI_TXLVL_FULL</i>	TXFIFO全满
<i>SPI_RXLVL_EMPTY</i>	SPI RXFIFO空
<i>SPI_RXLVL_QUARTER_FULL</i>	SPI RXFIFO四分之一满
<i>SPI_RXLVL_HALF_FULL</i>	SPI RXFIFO半满
<i>SPI_RXLVL_FULL</i>	RXFIFO全满
输出参数{out}	
-	-
返回值	
FlagStatus	SET 或 RESET

例如：

```
/* get SPI0 transmit buffer empty flag status */
```

```
FlagStatus Flag = RESET;
```

```
Flag = spi_flag_get(SPI0, SPI_FLAG_TBE);
```

函数 spi_interrupt_enable

函数spi_interrupt_enable描述见下表：

表 3-1093. 函数 spi_interrupt_enable

函数名称	spi_interrupt_enable
函数原形	void spi_interrupt_enable(uint32_t spi_periph, uint8_t interrupt);
功能描述	使能外设SPI中断
先决条件	-
被调用函数	-
输入参数{in}	
spi_periph	外设SPI
<i>SPIx</i>	x=0,1,2
输入参数{in}	
interrupt	SPI中断
<i>SPI_INT_TBE</i>	发送缓冲区空中断
<i>SPI_INT_RBNE</i>	接收缓冲区非空中断
<i>SPI_INT_ERR</i>	错误中断使能
输出参数{out}	
-	-

返回值	
-	-

例如：

```
/* enable SPI0 transmit buffer empty interrupt */
```

```
spi_interrupt_enable(SPI0, SPI_INT_TBE);
```

函数 spi_interrupt_disable

函数spi_interrupt_disable描述见下表：

表 3-1094. 函数 spi_interrupt_disable

函数名称	spi_interrupt_disable
函数原形	void spi_interrupt_disable(uint32_t spi_periph, uint8_t interrupt);
功能描述	失能外设SPI中断
先决条件	-
被调用函数	-
输入参数{in}	
spi_periph	外设SPI
SPIx	x=0,1,2
输入参数{in}	
interrupt	SPI/中断
SPI_INT_TBE	发送缓冲区空中断
SPI_INT_RBNE	接收缓冲区非空中断
SPI_INT_ERR	错误中断使能
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* disable SPI0 transmit buffer empty interrupt */
```

```
spi_interrupt_disable(SPI0, SPI_INT_TBE);
```

函数 spi_interrupt_flag_get

函数spi_interrupt_flag_get描述见下表：

表 3-1095. 函数 spi_interrupt_flag_get

函数名称	spi_interrupt_flag_get
函数原形	FlagStatus spi_interrupt_flag_get(uint32_t spi_periph, uint8_t interrupt);
功能描述	获取外设SPI中断状态
先决条件	-

被调用函数	-
输入参数{in}	
spi_periph	外设SPI
SPIx	x=0,1,2
输入参数{in}	
interrupt	SPI中断状态
SPI_INT_FLAG_TB E	发送缓冲区空中断
SPI_INT_FLAG_RB NE	接收缓冲区非空中断
SPI_INT_FLAG_RX ORERR	接收过载错误中断
SPI_INT_FLAG_CO NFERR	配置错误中断
SPI_INT_FLAG_CR CERR	CRC错误中断
SPI_INT_FLAG_FE RR	格式错误中断
输出参数{out}	
-	-
返回值	
FlagStatus	SET 或 RESET

例如：

```

/* get SPI0 transmit buffer empty interrupt status */
if(RESET != spi_interrupt_flag_get(SPI0, SPI_INT_FLAG_TBE)){
    while(RESET == spi_flag_get(SPI0, SPI_FLAG_TBE));
    spi_data_transmit(SPI0, spi0_send_array[send_n++]);
}

```

3.29. SYSCFG

章节 [3.29.1](#) 描述了SYSCFG的寄存器列表，章节 [3.29.2](#) 对SYSCFG库函数进行说明。

3.29.1. 外设寄存器说明

SYSCFG寄存器列表如下表所示：

表 3-1096. SYSCFG 寄存器

寄存器名称	寄存器描述
SYSCFG_CFG0	配置寄存器0
SYSCFG_CFG1	配置寄存器1
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_CFG2	配置寄存器2
SYSCFG_STAT	状态寄存器
SYSCFG_CFG3	配置寄存器3
SYSCFG_CFG4	配置寄存器4
SYSCFG_CFG5	配置寄存器5
SYSCFG_TCMSRA MCS	TCMSRAM控制与状态寄存器
SYSCFG_TCMSRA MKEY	TCMSRAM密钥寄存器
SYSCFG_TCMSRA MWP	TCMSRAM写保护寄存器
SYSCFG_CPSCTL	I/O补偿控制寄存器
SYSCFG_TIMERCI SEL0	TIMER输入选择寄存器0
SYSCFG_TIMERCI SEL1	TIMER输入选择寄存器1
SYSCFG_TIMERCI SEL2	TIMER输入选择寄存器2
SYSCFG_TIMERCI SEL3	TIMER输入选择寄存器3
SYSCFG_TIMERCI SEL4	TIMER输入选择寄存器4
SYSCFG_TIMERxC FG0, x=0, 1, 2, 3, 4, 7, 19	TIMERx配置寄存器0, x=0, 1, 2, 3, 4, 7, 19
SYSCFG_TIMERxC FG1, x=0, 1, 2, 3, 4, 7, 19	TIMERx配置寄存器1, x=0, 1, 2, 3, 4, 7, 19
SYSCFG_TIMERxC FG2, x=0, 1, 2, 3, 4, 7, 19	TIMERx配置寄存器2, x=0, 1, 2, 3, 4, 7, 19
SYSCFG_TIMERxC FG0, x=14	TIMERx配置寄存器0, x=14
SYSCFG_TIMERxC	TIMERx配置寄存器1, x=14

寄存器名称	寄存器描述
FG1, x=14	
SYSCFG_TIMERxC FG2, x=14	TIMERx配置寄存器2, x=14

3.29.2. 外设库函数说明

SYSCFG库函数列表如下表所示：

表 3-1097. SYSCFG 库函数

库函数名称	库函数描述
syscfg_deinit	复位SYSCFG寄存器
syscfg_flash_bank_remap_set	切换Flash的Bank1和Bank0的地址
syscfg_fpu_interrupt_enable	使能FPU中断
syscfg_fpu_interrupt_disable	除能FPU中断
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_exti_line_config	配置GPIO引脚作为EXTI
syscfg_pin_reset_mode_config	配置GPIO复位模式
syscfg_trigsel_cla_reset_mode_config	配置触发选择控制器_可配置逻辑阵列复位模式
syscfg_lockup_enable	使能模块锁定
syscfg_timer_input_source_select	选择TIMER输入源
syscfg_bootmode_memmap_select	选择启动模式内存映射
syscfg_sram_ecc_single_correctable_bit_get	获取SRAM ECC单比特错误位域
syscfg_sram_ecc_error_address_get	获取SRAM ECC错误地址
syscfg_tcmsram_erase	擦除TCMSRAM
syscfg_tcmsram_erase_lock	锁定TCMSRAM
syscfg_tcmsram_erase_unlock	解锁TCMSRAM擦除
syscfg_tcmsram_page_wp_enable	使能TCMSRAM某页的写保护
syscfg_io_compensation_config	配置I/O补偿单元
syscfg_interrupt_flag_get	获取中断标志
syscfg_interrupt_flag_clear	清除中断标志
syscfg_interrupt_enable	使能SYSCFG外设中断
syscfg_interrupt_disable	除能SYSCFG外设中断
syscfg_tcmsram_busy_flag_get	获取TCMSRAM擦除忙碌标志
syscfg_compensation_cell_ready_flag_get	获取补偿单元准备就绪标志

枚举类型 **syscfg_interrupt_enum**表 3-1098. 枚举类型 **syscfg_interrupt_enum**

枚举名称	枚举描述
SYSCFG_INT_SRAM0ECCME	SRAM0 ECC多比特无纠错事件
SYSCFG_INT_SRAM0ECCSE	SRAM0 ECC单比特可纠错事件
SYSCFG_INT_FLASH ECC	FLASH ECC NMI中断
SYSCFG_INT_CKMNMI	HXTAL时钟监视NMI中断
SYSCFG_INT_NMIPIN	NMI引脚中断
SYSCFG_INT_SRAM1ECCME	SRAM1 ECC多比特无纠错事件
SYSCFG_INT_SRAM1ECCSE	SRAM1 ECC单比特可纠错事件
SYSCFG_INT_TCMSRAM ECC ME	TCMSRAM ECC多比特无纠错事件
SYSCFG_INT_TCMSRAM ECC E	TCMSRAM ECC单比特可纠错事件

枚举类型 **syscfg_flag_enum**表 3-1099. 枚举类型 **syscfg_flag_enum**

枚举名称	枚举描述
SYSCFG_INT_FLAG_SRAM0E CCME	SRAM0 ECC多比特无纠错事件标志
SYSCFG_INT_FLAG_SRAM0E CCSE	SRAM0 ECC单比特可纠错事件标志
SYSCFG_INT_FLAG_FLASH ECC C	FLASH ECC NMI中断标志
SYSCFG_INT_FLAG_CKMNMI	HXTAL时钟监视NMI中断标志
SYSCFG_INT_FLAG_NMIPIN	NMI引脚中断标志
SYSCFG_INT_FLAG_SRAM1E CCME	SRAM1 ECC多比特无纠错事件标志
SYSCFG_INT_FLAG_SRAM1E CCSE	SRAM1 ECC单比特可纠错事件标志
SYSCFG_INT_FLAG_TCMSRAM MECCME	TCMSRAM ECC多比特无纠错事件标志
SYSCFG_INT_FLAG_TCMSRAM MECCSE	TCMSRAM ECC单比特可纠错事件标志

枚举类型 **syscfg_sram_serrbits_enum**表 3-1100. 枚举类型 **syscfg_sram_serrbits_enum**

枚举名称	枚举描述
SYSCFG_SRAM0_SERRBITS	SRAM0 ECC单比特可纠错事件错误比特位
SYSCFG_SRAM1_SERRBITS	SRAM1 ECC单比特可纠错事件错误比特位

枚举名称	枚举描述
SYSCFG_TCMRAM_SERRBIT S	TCMRAM ECC单比特可纠错事件错误比特位

枚举类型 `syscfg_sram_erraddr_enum`

表 3-1101. 枚举类型 `syscfg_sram_erraddr_enum`

枚举名称	枚举描述
SYSCFG_SRAM0_ERR_ADDR	SRAM0 ECC单比特可纠错事件错误地址
SYSCFG_SRAM1_ERR_ADDR	SRAM1 ECC单比特可纠错事件错误地址
SYSCFG_TCMRAM_ERR_AD DR	TCMRAM ECC单比特可纠错事件错误地址

枚举类型 `timer_channel_input_enum`

表 3-1102. 枚举类型 `timer_channel_input_enum`

枚举名称	枚举描述
TIMER7_CIO_INPUT_TIMER7_ CH0	选择TIMER7 CH0作为TIMER7 CIO输入
TIMER7_CIO_INPUT_CMP0_O UT	选择CMP0输出作为TIMER7 CIO输入
TIMER7_CIO_INPUT_CMP1_O UT	选择CMP1输出作为TIMER7 CIO输入
TIMER7_CIO_INPUT_CMP2_O UT	选择CMP2输出作为TIMER7 CIO输入
TIMER7_CIO_INPUT_CMP3_O UT	选择CMP3输出作为TIMER7 CIO输入
TIMER7_CIO_INPUT_CLA0_OU T	选择CLA0输出作为TIMER7 CIO输入
TIMER7_CI1_INPUT_TIMER7_ CH1	选择TIMER7 CH1作为TIMER7 CI1输入
TIMER7_CI1_INPUT_CLA1_OU T	选择CLA1输出作为TIMER7 CI1输入
TIMER7_CI2_INPUT_TIMER7_ CH2	选择TIMER7 CH2作为TIMER7 CI2输入
TIMER7_CI2_INPUT_CLA2_OU T	选择CLA2输出作为TIMER7 CI2输入
TIMER7_CI3_INPUT_TIMER7_ CH3	选择TIMER7 CH3作为TIMER7 CI3输入
TIMER7_CI3_INPUT_CLA3_OU T	选择CLA3输出作为TIMER7 CI3输入
TIMER0_CIO_INPUT_TIMER0_	选择TIMER0 CH0作为TIMER0 CIO输入

枚举名称	枚举描述
CH0	
TIMER0_CIO_INPUT_CMP0_O UT	选择CMP0输出作为TIMER0 CIO输入
TIMER0_CIO_INPUT_CMP1_O UT	选择CMP1输出作为TIMER0 CIO输入
TIMER0_CIO_INPUT_CMP2_O UT	选择CMP2输出作为TIMER0 CIO输入
TIMER0_CIO_INPUT_CMP3_O UT	选择CMP3输出作为TIMER0 CIO输入
TIMER0_CIO_INPUT_CLA0_OU T	选择CLA0输出作为TIMER0 CIO输入
TIMER0_C11_INPUT_TIMER0_ CH1	选择TIMER0 CH1作为TIMER0 C11输入
TIMER0_C11_INPUT_CLA1_OU T	选择CLA1输出作为TIMER0 C11输入
TIMER0_C12_INPUT_TIMER0_ CH2	选择TIMER0 CH2作为TIMER0 C12输入
TIMER0_C12_INPUT_CLA2_OU T	选择CLA2输出作为TIMER0 C12输入
TIMER0_C13_INPUT_TIMER0_ CH3	选择TIMER0 CH3作为TIMER0 C13输入
TIMER0_C13_INPUT_CLA3_OU T	选择CLA3输出作为TIMER0 C13输入
TIMER19_CIO_INPUT_TIMER19_ _CH0	选择TIMER19CH0作为TIMER19 CIO输入
TIMER19_CIO_INPUT_CMP0_O UT	选择CMP0输出作为TIMER19 CIO输入
TIMER19_CIO_INPUT_CMP1_O UT	选择CMP1输出作为TIMER19 CIO输入
TIMER19_CIO_INPUT_CMP2_O UT	选择CMP2输出作为TIMER19 CIO输入
TIMER19_CIO_INPUT_CMP3_O UT	选择CMP3输出作为TIMER19 CIO输入
TIMER19_CIO_INPUT_CLA0_O UT	选择CLA0输出作为TIMER19 CIO输入
TIMER19_C11_INPUT_TIMER19_ _CH1	选择TIMER19CH1作为TIMER19 C11输入
TIMER19_C11_INPUT_CLA1_O UT	选择CLA1输出作为TIMER19 C11输入
TIMER19_C12_INPUT_TIMER19_ _CH2	选择TIMER19CH2作为TIMER19 C12输入
TIMER19_C12_INPUT_CLA2_O	选择CLA2输出作为TIMER19 C12输入

枚举名称	枚举描述
UT	
TIMER19_CI3_INPUT_TIMER19_CH3	选择TIMER19CH3作为TIMER19 CI3输入
TIMER19_CI3_INPUT_CLA3_OUT	选择CLA3输出作为TIMER19 CI3输入
TIMER2_CIO_INPUT_TIMER2_CH0	选择TIMER2 CH0作为TIMER2 CIO输入
TIMER2_CIO_INPUT_CMP0_OUT	选择CMP0作为TIMER2 CIO输入
TIMER2_CIO_INPUT_CMP1_OUT	选择CMP1作为TIMER2 CIO输入
TIMER2_CIO_INPUT_CMP2_OUT	选择CMP2作为TIMER2 CIO输入
TIMER2_CIO_INPUT_CMP3_OUT	选择CMP3作为TIMER2 CIO输入
TIMER2_CIO_INPUT_CMP4_OUT	选择CMP4作为TIMER2 CIO输入
TIMER2_CIO_INPUT_CMP5_OUT	选择CMP5作为TIMER2 CIO输入
TIMER2_CIO_INPUT_CMP6_OUT	选择CMP6作为TIMER2 CIO输入
TIMER2_CIO_INPUT_CLA0_OUT	选择CLA2输出作为TIMER2 CIO输入
TIMER2_CI1_INPUT_TIMER2_CH1	选择TIMER2 CH1作为TIMER2 CI1输入
TIMER2_CI1_INPUT_CMP0_OUT	选择CMP0作为TIMER2 CI1输入
TIMER2_CI1_INPUT_CMP1_OUT	选择CMP1作为TIMER2 CI1输入
TIMER2_CI1_INPUT_CMP2_OUT	选择CMP2作为TIMER2 CI1输入
TIMER2_CI1_INPUT_CMP3_OUT	选择CMP3作为TIMER2 CI1输入
TIMER2_CI1_INPUT_CMP4_OUT	选择CMP4作为TIMER2 CI1输入
TIMER2_CI1_INPUT_CMP5_OUT	选择CMP5作为TIMER2 CI1输入
TIMER2_CI1_INPUT_CMP6_OUT	选择CMP6作为TIMER2 CI1输入
TIMER2_CI1_INPUT_CLA1_OUT	选择CLA1输出作为TIMER2 CI1输入
TIMER2_CI2_INPUT_TIMER2_	选择TIMER2 CH2作为TIMER2 CI2输入

枚举名称	枚举描述
CH2	
TIMER2_C12_INPUT_CMP2_O UT	选择CMP2作为TIMER2 C12输入
TIMER2_C12_INPUT_CLA2_OU T	选择CLA2输出作为TIMER2 C12输入
TIMER2_C13_INPUT_TIMER2_ CH3	选择TIMER2 CH3作为TIMER2 C13输入
TIMER2_C13_INPUT_CLA3_OU T	选择CLA3作为TIMER2 C13输入
TIMER1_C10_INPUT_TIMER1_ CH0	选择TIMER1 CH0作为TIMER1 C10输入
TIMER1_C10_INPUT_CMP0_O UT	选择CMP0作为TIMER1 C10输入
TIMER1_C10_INPUT_CMP1_O UT	选择CMP1作为TIMER1 C10输入
TIMER1_C10_INPUT_CMP2_O UT	选择CMP2作为TIMER1 C10输入
TIMER1_C10_INPUT_CMP3_O UT	选择CMP3作为TIMER1 C10输入
TIMER1_C10_INPUT_CMP4_O UT	选择CMP4作为TIMER1 C10输入
TIMER1_C10_INPUT_CLA0_OU T	选择CLA0输出作为TIMER1 C10输入
TIMER1_C11_INPUT_TIMER1_ CH1	选择TIMER1 CH1作为TIMER1 C11输入
TIMER1_C11_INPUT_CMP0_O UT	选择CMP0作为TIMER1 C11输入
TIMER1_C11_INPUT_CMP1_O UT	选择CMP1作为TIMER1 C11输入
TIMER1_C11_INPUT_CMP2_O UT	选择CMP2作为TIMER1 C11输入
TIMER1_C11_INPUT_CMP3_O UT	选择CMP3作为TIMER1 C11输入
TIMER1_C11_INPUT_CMP5_O UT	选择CMP5作为TIMER1 C11输入
TIMER1_C11_INPUT_CLA1_OU T	选择CLA1输出作为TIMER1 C11输入
TIMER1_C12_INPUT_TIMER1_ CH2	选择TIMER1 CH2作为TIMER1 C12输入
TIMER1_C12_INPUT_CMP3_O UT	选择CMP3作为TIMER1 C12输入
TIMER1_C12_INPUT_CLA2_OU	选择CLA2输出作为TIMER1 C12输入

枚举名称	枚举描述
T	
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_CLA3_OUT	选择CLA3输出作为TIMER1 CI3输入
TIMER4_CIO_INPUT_TIMER4_CH0	选择TIMER4 CH0作为TIMER4 CIO输入
TIMER4_CIO_INPUT_IRC32K	选择IRC32K作为TIMER4 CIO输入
TIMER4_CIO_INPUT_LXTAL	选择LXTAL作为TIMER4 CIO输入
TIMER4_CIO_INPUT_RTC_WAKEUP	选择RTC_WAKEUP作为TIMER4 CIO输入
TIMER4_CIO_INPUT_CMP0_OUT	选择CMP0输出作为TIMER4 CIO输入
TIMER4_CIO_INPUT_CMP1_OUT	选择CMP1输出作为TIMER4 CIO输入
TIMER4_CIO_INPUT_CMP2_OUT	选择CMP2输出作为TIMER4 CIO输入
TIMER4_CIO_INPUT_CMP3_OUT	选择CMP3输出作为TIMER4 CIO输入
TIMER4_CIO_INPUT_CMP4_OUT	选择CMP4输出作为TIMER4 CIO输入
TIMER4_CIO_INPUT_CMP5_OUT	选择CMP5输出作为TIMER4 CIO输入
TIMER4_CIO_INPUT_CMP6_OUT	选择CMP6输出作为TIMER4 CIO输入
TIMER4_CIO_INPUT_CLA0_OUT	选择CLA0输出作为TIMER4 CIO输入
TIMER4_CI1_INPUT_TIMER4_CH1	选择TIMER4 CH1作为TIMER4 CI1输入
TIMER4_CI1_INPUT_CMP0_OUT	选择CMP0输出作为TIMER4 CI1输入
TIMER4_CI1_INPUT_CMP1_OUT	选择CMP1输出作为TIMER4 CI1输入
TIMER4_CI1_INPUT_CMP2_OUT	选择CMP2输出作为TIMER4 CI1输入
TIMER4_CI1_INPUT_CMP3_OUT	选择CMP3输出作为TIMER4 CI1输入
TIMER4_CI1_INPUT_CMP4_OUT	选择CMP4输出作为TIMER4 CI1输入

枚举名称	枚举描述
UT	
TIMER4_C11_INPUT_CMP5_O UT	选择CMP5输出作为TIMER4 CI1输入
TIMER4_C11_INPUT_CMP6_O UT	选择CMP6输出作为TIMER4 CI1输入
TIMER4_C11_INPUT_CLA1_OU T	选择CLA1输出作为TIMER4 CI1输入
TIMER4_C12_INPUT_TIMER4_ CH2	选择TIMER4 CH2作为TIMER4 CI2输入
TIMER4_C12_INPUT_CLA2_OU T	选择CLA2输出作为TIMER4 CI2输入
TIMER4_C13_INPUT_TIMER4_ CH3	选择TIMER4 CH3作为TIMER4 CI3输入
TIMER4_C13_INPUT_CLA3_OU T	选择CLA3输出作为TIMER4 CI3输入
TIMER3_C10_INPUT_TIMER3_ CH0	选择TIMER3 CH0作为TIMER3 CI0输入
TIMER3_C10_INPUT_CMP0_O UT	选择CMP0输出作为TIMER3 CI0输入
TIMER3_C10_INPUT_CMP1_O UT	选择CMP1输出作为TIMER3 CI0输入
TIMER3_C10_INPUT_CMP2_O UT	选择CMP2输出作为TIMER3 CI0输入
TIMER3_C10_INPUT_CMP3_O UT	选择CMP3输出作为TIMER3 CI0输入
TIMER3_C10_INPUT_CMP4_O UT	选择CMP4输出作为TIMER3 CI0输入
TIMER3_C10_INPUT_CMP5_O UT	选择CMP5输出作为TIMER3 CI0输入
TIMER3_C10_INPUT_CMP6_O UT	选择CMP6输出作为TIMER3 CI0输入
TIMER3_C10_INPUT_CLA0_OU T	选择CLA0输出作为TIMER3 CI0输入
TIMER3_C11_INPUT_TIMER3_ CH1	选择TIMER3 CH1作为TIMER3 CI1输入
TIMER3_C11_INPUT_CMP0_O UT	选择CMP0输出作为TIMER3 CI1输入
TIMER3_C11_INPUT_CMP1_O UT	选择CMP1输出作为TIMER3 CI1输入
TIMER3_C11_INPUT_CMP2_O UT	选择CMP2输出作为TIMER3 CI1输入
TIMER3_C11_INPUT_CMP3_O	选择CMP3输出作为TIMER3 CI1输入

枚举名称	枚举描述
UT	
TIMER3_C11_INPUT_CMP4_O UT	选择CMP4输出作为TIMER3 CI1输入
TIMER3_C11_INPUT_CMP5_O UT	选择CMP5输出作为TIMER3 CI1输入
TIMER3_C11_INPUT_CMP6_O UT	选择CMP6输出作为TIMER3 CI1输入
TIMER3_C11_INPUT_CLA1_OU T	选择CLA1输出作为TIMER3 CI1输入
TIMER3_C12_INPUT_TIMER3_ CH2	选择TIMER3 CH2作为TIMER3 CI2输入
TIMER3_C12_INPUT_CMP4_O UT	选择CMP4输出作为TIMER3 CI2输入
TIMER3_C12_INPUT_CLA2_OU T	选择CLA2输出作为TIMER3 CI2输入
TIMER3_C13_INPUT_TIMER3_ CH3	选择TIMER3 CH3作为TIMER3 CI3输入
TIMER3_C13_INPUT_CMP5_O UT	选择CMP5输出作为TIMER3 CI3输入
TIMER3_C13_INPUT_CLA3_OU T	选择CLA3输出作为TIMER3 CI3输入
TIMER14_C10_INPUT_TIMER14_ _CH0	选择TIMER14 CH0作为TIMER14 CI0输入
TIMER14_C10_INPUT_LXTAL	选择TIMER1 CH0作为TIMER14 CI0输入
TIMER14_C10_INPUT_CMP0_O UT	选择CMP0输出作为TIMER14 CI0输入
TIMER14_C10_INPUT_CMP1_O UT	选择CMP1输出作为TIMER14 CI0输入
TIMER14_C10_INPUT_CMP4_O UT	选择CMP4输出作为TIMER14 CI0输入
TIMER14_C10_INPUT_CMP6_O UT	选择CMP6输出作为TIMER14 CI0输入
TIMER14_C10_INPUT_CLA0_O UT	选择CLA0输出作为TIMER14 CI0输入
TIMER14_C10_INPUT_CLA1_O UT	选择CLA1输出作为TIMER14 CI0输入
TIMER14_C10_INPUT_CLA2_O UT	选择CLA2输出作为TIMER14 CI0输入
TIMER14_C10_INPUT_CLA3_O UT	选择CLA3输出作为TIMER14 CI0输入
TIMER14_C11_INPUT_TIMER14_ _CH1	选择TIMER14 CH1作为TIMER14 CI1输入

枚举名称	枚举描述
TIMER14_CI1_INPUT_CMP1_0 UT	选择CMP1输出作为TIMER14 CI1输入
TIMER14_CI1_INPUT_CMP2_0 UT	选择CMP2输出作为TIMER14 CI1输入
TIMER14_CI1_INPUT_CMP5_0 UT	选择CMP5输出作为TIMER14 CI1输入
TIMER14_CI1_INPUT_CMP6_0 UT	选择CMP6输出作为TIMER14 CI1输入
TIMER15_CI0_INPUT_TIMER15 _CH0	选择TIMER15 CH0作为TIMER15 CI0输入
TIMER15_CI0_INPUT_CMP5_0 UT	选择CMP5输出作为TIMER15 CI0输入
TIMER15_CI0_INPUT_CKOUT	选择CKOUT输出作为TIMER15 CI0输入
TIMER15_CI0_INPUT_HXTAL_ DIV32	选择HXTAL/32作为TIMER15 CI0输入
TIMER15_CI0_INPUT_RTC_CL OCK	选择RTC_CLOCK作为TIMER15 CI0输入
TIMER15_CI0_INPUT_LXTAL	选择LXTAL作为TIMER15 CI0输入
TIMER15_CI0_INPUT_IRC32K	选择IRC32K作为TIMER15 CI0输入
TIMER15_CI0_INPUT_CLA0_O UT	选择CLA0输出作为TIMER15 CI0输入
TIMER15_CI0_INPUT_CLA1_O UT	选择CLA1输出作为TIMER15 CI0输入
TIMER15_CI0_INPUT_CLA2_O UT	选择CLA2输出作为TIMER15 CI0输入
TIMER15_CI0_INPUT_CLA3_O UT	选择CLA3输出作为TIMER15 CI0输入
TIMER15_CI0_INPUT_HXTAL	选择HXTAL输出作为TIMER15 CI0输入
TIMER15_CI0_INPUT_WKUP_I T	选择WKUP IT作为TIMER15 CI0输入
TIMER16_CI0_INPUT_TIMER16 _CH0	选择TIMER16 CH0作为TIMER16 CI0输入
TIMER16_CI0_INPUT_CMP4_0 UT	选择CMP4输出作为TIMER16 CI0输入
TIMER16_CI0_INPUT_CKOUT	选择CKOUT输出作为TIMER16 CI0输入
TIMER16_CI0_INPUT_HXTAL_ DIV32	选择HXTAL/32作为TIMER16 CI0输入
TIMER16_CI0_INPUT_RTC_CL OCK	选择RTC_CLOCK作为TIMER16 CI0输入
TIMER16_CI0_INPUT_LXTAL	选择LXTAL作为TIMER16 CI0输入
TIMER16_CI0_INPUT_IRC32K	选择IRC32K作为TIMER16 CI0输入

枚举名称	枚举描述
TIMER16_CIO_INPUT_CLA0_OU T	选择CLA0输出TIMER16 CIO输入
TIMER16_CIO_INPUT_CLA1_OU T	选择CLA1输出TIMER16 CIO输入
TIMER16_CIO_INPUT_CLA2_OU T	选择CLA2输出TIMER16 CIO输入
TIMER16_CIO_INPUT_CLA3_OU T	选择CLA3输出TIMER16 CIO输入
TIMER16_CIO_INPUT_HXTAL	选择HXTAL作为TIMER16 CIO输入

函数 syscfg_deinit

函数syscfg_deinit描述见下表：

表 3-1103. 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	-
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 3-1104. 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_enable);
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_enable	I2C Fm+模式

<code>SYSCFG_I2C0_FMP</code> <i>P</i>	I2C0 Fm+模式
<code>SYSCFG_I2C1_FMP</code> <i>P</i>	I2C1 Fm+模式
<code>SYSCFG_I2C2_FMP</code> <i>P</i>	I2C2 Fm+模式
<code>SYSCFG_I2C3_FMP</code> <i>P</i>	I2C3 Fm+模式
<code>SYSCFG_I2C_FMP</code> <code>_PB6</code>	PB6引脚Fm+模式
<code>SYSCFG_I2C_FMP</code> <code>_PB7</code>	PB7引脚Fm+模式
<code>SYSCFG_I2C_FMP</code> <code>_PB8</code>	PB8引脚Fm+模式
<code>SYSCFG_I2C_FMP</code> <code>_PB9</code>	PB9引脚Fm+模式
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* enable I2C0 fust mode plus function */
```

```
syscfg_i2c_fast_mode_plus_enable(SYSCFG_I2C0_FMP);
```

函数 `syscfg_i2c_fast_mode_plus_disable`

函数 `syscfg_i2c_fast_mode_plus_disable` 描述见下表：

表 3-1105. 函数 `syscfg_i2c_fast_mode_plus_disable`

函数名称	<code>syscfg_i2c_fast_mode_plus_disable</code>
函数原型	<code>void syscfg_i2c_fast_mode_plus_disable(uint32_t i2c_fmp);</code>
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}	
<code>i2c_fmp</code>	I2C Fm+模式
<code>SYSCFG_I2C0_FMP</code> <i>P</i>	I2C0 Fm+模式
<code>SYSCFG_I2C1_FMP</code> <i>P</i>	I2C1 Fm+模式
<code>SYSCFG_I2C2_FMP</code> <i>P</i>	I2C2 Fm+模式

<code>SYSCFG_I2C3_FMP</code>	I2C3 Fm+模式
<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_exti_line_config`

函数`syscfg_exti_line_config`描述见下表:

表 3-1106. 函数 `syscfg_exti_line_config`

函数名称	<code>syscfg_exti_line_config</code>
函数原形	<code>void syscfg_exti_line_config(uint8_t exti_port, uint8_t exti_pin);</code>
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
<code>EXTI_SOURCE_GPIOx</code>	x = A,B,C,D,E,F,G
输入参数{in}	
exti_pin	EXTI引脚
<code>EXTI_SOURCE_PINx</code>	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
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_pin_reset_mode_config

函数syscfg_pin_reset_mode_config描述见下表：

表 3-1107. 函数 syscfg_pin_reset_mode_config

函数名称	syscfg_pin_reset_mode_config
函数原形	void syscfg_pin_reset_mode_config(uint32_t syscfg_pin_reset_mode);
功能描述	配置GPIO复位模式
先决条件	-
被调用函数	-
输入参数{in}	
syscfg_pin_reset_mode	配置GPIO复位模式
SYSCFG_PIN_NRS T	除了POR事件外，GPIO引脚配置将在任何重置事件中保持状态
SYSCFG_PIN_RST	当发生复位事件时，GPIO引脚配置将复位
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* GPIO remain state after reset */
```

```
syscfg_pin_reset_mode_config(SYSCFG_PIN_NRST);
```

函数 syscfg_trigsel_cla_reset_mode_config

函数syscfg_trigsel_cla_reset_mode_config描述见下表：

表 3-1108. 函数 syscfg_trigsel_cla_reset_mode_config

函数名称	syscfg_trigsel_cla_reset_mode_config
函数原形	void syscfg_trigsel_cla_reset_mode_config(uint32_t syscfg_trigsel_reset_mode);
功能描述	配置触发选择控制器_可配置逻辑阵列复位模式
先决条件	-
被调用函数	-
输入参数{in}	
syscfg_trigsel_reset_mode	配置触发选择控制器_可配置逻辑阵列复位模式
SYSCFG_TRGSEL	除了POR事件外，Trigsel CLA配置将在任何重置事件中保持状态

<code>_CLA_NRST</code>	
<code>SYSCFG_TRGSEL</code> <code>_CLA_RST</code>	当发生复位事件时，Trigsel CLA配置将复位
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* Trigsel CLA remain state after reset */
syscfg_trigsel_cla_reset_mode_config(SYSCFG_TRGSEL_CLA_NRST);
```

函数 `syscfg_lockup_enable`

函数`syscfg_lockup_enable`描述见下表：

表 3-1109. 函数 `syscfg_lockup_enable`

函数名称	<code>syscfg_lockup_enable</code>
函数原型	<code>void syscfg_lockup_enable(uint32_t lockup);</code>
功能描述	配置SYSCFG锁定功能
先决条件	-
被调用函数	-
输入参数{in}	
lockup	锁定功能
<code>SYSCFG_LVD_LO</code> <code>CKUP</code>	LVD锁定
<code>SYSCFG_LOCKUP</code> <code>_LOCKUP</code>	CPU锁定
<code>SYSCFG_FLASH_L</code> <code>OCKUP</code>	FLASH ECC双差错锁定
<code>SYSCFG_SRAM1_</code> <code>LOCKUP</code>	SRAM1 ECC双差错锁定
<code>SYSCFG_SRAM0_</code> <code>LOCKUP</code>	SRAM0 ECC双差错锁定
<code>SYSCFG_TCMSRA</code> <code>M_LOCKUP</code>	TCM 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_CFG2_LOCKUP_LOCK);
```

函数 syscfg_timer_input_source_select

函数syscfg_timer_input_source_select描述见下表：

表 3-1110. 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3-1102. 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_flash_bank_remap_set

函数syscfg_flash_bank_remap_set描述见下表：

表 3-1111. 函数 syscfg_flash_bank_remap_set

函数名称	syscfg_flash_bank_remap_set
函数原形	void syscfg_flash_bank_remap_set(uint32_t value);
功能描述	切换FLASH Bank0和Bank1地址
先决条件	-
被调用函数	-
输入参数{in}	
value	Flash地址映射
SYSCFG_FLASH_BANK0_MAPPED	Flash Bank 1映射到0x08000000, Flash Bank 0映射到0x08040000
SYSCFG_FLASH_BANK1_MAPPED	Flash Bank 0映射到0x08000000, Flash Bank 1映射到0x08040000
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* Flash Bank 1 mapped at 0x08000000, and Flash Bank 0 mapped at 0x08040000 */
```

```
syscfg_flash_bank_remap_set(SYS_CFG_FLASH_BANK0_MAPPED);
```

函数 syscfg_bootmode_memmap_select

函数syscfg_bootmode_memmap_select描述见下表:

表 3-1112. 函数 syscfg_bootmode_memmap_select

函数名称	syscfg_bootmode_memmap_select
函数原型	void syscfg_bootmode_memmap_select(uint32_t mem_select);
功能描述	选择启动模式内存映射
先决条件	-
被调用函数	-
输入参数{in}	
mem_select	内存映射
SYS_CFG_MAIN_FLASH_MEMORY	主FLASH存储器被映射到地址0x0000 0000
SYS_CFG_SYSTEM_FLASH_MEMORY	引导装载代码所在系统存储器被映射到地址0x0000 0000
SYS_CFG_EXMC_MEMORY	EXMC存储器地址被映射到地址0x0000 0000
SYS_CFG_SRAM0_MEMORY	片上SRAM的SRAM0被映射到地址0x0000 0000
SYS_CFG_QSPI_MEMORY	QSPI存储器地址被映射到地址0x0000 0000
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* Main Flash memory mapped at 0x00000000 */
```

```
syscfg_bootmode_memmap_select(SYS_CFG_MAIN_FLASH_MEMORY);
```

函数 syscfg_sram_ecc_single_correctable_bit_get

函数syscfg_sram_ecc_single_correctable_bit_get描述见下表:

表 3-1113. 函数 syscfg_sram_ecc_single_correctable_bit_get

函数名称	syscfg_sram_ecc_single_correctable_bit_get
函数原型	uint32_t syscfg_sram_ecc_single_correctable_bit_get(syscfg_sram_serrbits_enum

	sram);
功能描述	获取SRAM ECC错误比特位
先决条件	-
被调用函数	-
输入参数{in}	
sram	SRAM ECC错误比特位，参考 表3-1100. 枚举类型 syscfg_sram_serrbits_enum
输出参数{out}	
-	-
返回值	
error_bits	错误比特位

例如：

```
/* get SRAM0 SRAM0 ECC single-bit correctable error bits */
syscfg_sram_ecc_single_correctable_bit_get(SYSCFG_SRAM0_SERRBITS);
```

函数 syscfg_sram_ecc_error_address_get

函数syscfg_sram_ecc_error_address_get描述见下表：

表 3-1114. 函数 syscfg_sram_ecc_error_address_get

函数名称	syscfg_io_low_voltage_speed_optimization_enable
函数原型	uint32_t syscfg_sram_ecc_error_address_get(syscfg_sram_erraddr_enum sram);
功能描述	获取SRAM ECC错误地址
先决条件	-
被调用函数	-
输入参数{in}	
sram	SRAM ECC错误地址，参考 表3-1101. 枚举类型syscfg_sram_erraddr_enum
输出参数{out}	
-	-
返回值	
addr	错误地址

例如：

```
/* get SRAM0 ECC error address */
syscfg_sram_ecc_error_address_get(SYSCFG_SRAM0_ERR_ADDR);
```

函数 syscfg_tcmsram_erase

函数syscfg_tcmsram_erase描述见下表：

表 3-1115. 函数 syscfg_tcmsram_erase

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

例如：

```
/* erase tcmsram */
syscfg_tcmsram_erase ();
```

函数 syscfg_tcmsram_erase_lock

函数syscfg_tcmsram_erase_lock描述见下表：

表 3-1116. 函数 syscfg_tcmsram_erase_lock

函数名称	syscfg_tcmsram_erase_lock
函数原型	void syscfg_tcmsram_erase_lock(void);
功能描述	锁定TCM SRAM擦除
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 TCMSRAM erase */
syscfg_tcmsram_erase_lock();
```

函数 syscfg_tcmsram_erase_unlock

函数syscfg_tcmsram_erase_unlock描述见下表：

表 3-1117. 函数 `syscfg_tcmsram_erase_unlock`

函数名称	<code>syscfg_tcmsram_erase_unlock</code>
函数原型	<code>void syscfg_tcmsram_erase_unlock (void);</code>
功能描述	解锁TCM SRAM擦除
先决条件	-
被调用函数	-
输入参数{in}	
-	-
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* unlock the TCMSRAM erase */
syscfg_tcmsram_erase_unlock();
```

函数 `syscfg_tcmsram_page_wp_enable`

函数`syscfg_tcmsram_page_wp_enable`描述见下表:

表 3-1118. 函数 `syscfg_tcmsram_page_wp_enable`

函数名称	<code>syscfg_tcmsram_page_wp_enable</code>
函数原型	<code>void syscfg_tcmsram_page_wp_enable(uint32_t pagex);</code>
功能描述	使能TCMSRAM某页的写保护
先决条件	-
被调用函数	-
输入参数{in}	
pagex	输入TCMSRAM写保护页, SYSCFG_TCMSRAMWP_PxWPEN (x = 0..31)
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* enable TCMSRAM page 0 write protection */
syscfg_tcmsram_page_wp_enable(SYSCFG_TCMSRAMWP_P0WPEN);
```

函数 `syscfg_io_compensation_config`

函数`syscfg_io_compensation_config`描述见下表:

表 3-1119. 函数 syscfg_io_compensation_config

函数名称	syscfg_io_compensation_config
函数原型	void syscfg_io_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_cps	I/O补偿单元使能配置
SYSCFG_IO_COMPENSATION_ENABLE	I/O补偿单元使能
SYSCFG_IO_COMPENSATION_DISABLE	I/O补偿单元禁能
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* I/O compensation cell enable */
```

```
syscfg_io_compensation_config(SYSCFG_IO_COMPENSATION_ENABLE);
```

函数 syscfg_fpu_interrupt_enable

函数syscfg_fpu_interrupt_enable描述见下表：

表 3-1120. 函数 syscfg_fpu_interrupt_enable

函数名称	syscfg_fpu_interrupt_enable
函数原型	void syscfg_fpu_interrupt_enable(uint32_t interrupt);
功能描述	使能FPU中断
先决条件	-
被调用函数	-
输入参数{in}	
interrupt	FPU中断
SYSCFG_FPUINT_INVALID_OPERATION	无效操作中断
SYSCFG_FPUINT_DIV0	除0中断
SYSCFG_FPUINT_OVERFLOW	溢出中断
SYSCFG_FPUINT_	下溢中断

UNDERFLOW	
SYSCFG_FPUINT_INPUT_ABNORMAL	输入错误中断
SYSCFG_FPUINT_INEXACT	不精确中断
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 3-1121. 函数 syscfg_fpu_interrupt_disable

函数名称	syscfg_fpu_interrupt_disable
函数原型	void syscfg_fpu_interrupt_enable(uint32_t interrupt);
功能描述	禁能FPU中断
先决条件	-
被调用函数	-
输入参数{in}	
interrupt	FPU中断
SYSCFG_FPUINT_INVALID_OPERATION	无效操作中断
SYSCFG_FPUINT_DIV0	除0中断
SYSCFG_FPUINT_OVERFLOW	溢出中断
SYSCFG_FPUINT_UNDERFLOW	下溢中断
SYSCFG_FPUINT_INPUT_ABNORMAL	输入错误中断
SYSCFG_FPUINT_INEXACT	不精确中断
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* disable FPU inexact interrupt */
```

```
syscfg_fpu_interrupt_disable(SYSCFG_FPUINT_INEXACT);
```

函数 syscfg_interrupt_flag_get

函数syscfg_interrupt_flag_get描述见下表：

表 3-1122. 函数 syscfg_interrupt_flag_get

函数名称	syscfg_interrupt_flag_get
函数原型	FlagStatus syscfg_interrupt_flag_get(syscfg_flag_enum int_flag);
功能描述	获取中断标志
先决条件	-
被调用函数	-
输入参数{in}	
int_flag	中断标志获取，参考 表3-1099. 枚举类型syscfg_flag_enum
输出参数{out}	
-	-
返回值	
FlagStatus	中断标志状态
SET	获取到标志
RESET	未获取到标志

例如：

```
/* get the SRAM0 ECC multi-bits non-correction event flag */
```

```
FlagStatus Flag = RESET;
```

```
Flag = syscfg_interrupt_flag_get(SYSCFG_INT_FLAG_SRAM0ECCME);
```

函数 syscfg_interrupt_flag_clear

函数syscfg_interrupt_flag_clear描述见下表：

表 3-1123. 函数 syscfg_interrupt_flag_clear

函数名称	syscfg_interrupt_flag_clear
函数原型	void syscfg_interrupt_flag_clear(syscfg_flag_enum int_flag);
功能描述	清除中断标志
先决条件	-
被调用函数	-
输入参数{in}	
int_flag	清除中断标志，参考 表3-1099. 枚举类型syscfg_flag_enum
输出参数{out}	
-	-

返回值	
-	-

例如：

```
/* clear the SRAM0 ECC multi-bits non-correction event flag */
```

```
syscfg_interrupt_flag_clear (SYSCFG_INT_FLAG_SRAM0ECCME);
```

函数 syscfg_interrupt_enable

函数syscfg_interrupt_enable描述见下表：

表 3-1124. 函数 syscfg_interrupt_enable

函数名称	syscfg_interrupt_enable
函数原型	void syscfg_interrupt_enable(syscfg_interrupt_enum interrupt);
功能描述	使能中断
先决条件	-
被调用函数	-
输入参数{in}	
interrupt	使能中断输入，参考 表3-1098. 枚举类型syscfg_interrupt_enum
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* enable the SRAM0 ECC multi-bits non-correction event flag */
```

```
syscfg_interrupt_enable(SYSCFG_INT_SRAM0ECCME);
```

函数 syscfg_interrupt_disable

函数syscfg_interrupt_disable描述见下表：

表 3-1125. 函数 syscfg_interrupt_disable

函数名称	syscfg_interrupt_disable
函数原型	void syscfg_interrupt_disable(syscfg_interrupt_enum interrupt);
功能描述	除能中断
先决条件	-
被调用函数	-
输入参数{in}	
interrupt	除能中断输入，参考 表3-1098. 枚举类型syscfg_interrupt_enum
输出参数{out}	
-	-
返回值	

-	-
---	---

例如：

```
/* disable the SRAM0 ECC multi-bits non-correction event flag */
```

```
syscfg_interrupt_disable(SYSCFG_INT_SRAM0ECCME);
```

函数 syscfg_tcmsram_busy_flag_get

函数syscfg_tcmsram_busy_flag_get描述见下表：

表 3-1126. 函数 syscfg_tcmsram_busy_flag_get

函数名称	syscfg_tcmsram_busy_flag_get
函数原型	FlagStatus syscfg_tcmsram_busy_flag_get(void);
功能描述	获取TCMSRAM擦除忙碌标志
先决条件	-
被调用函数	-
输入参数{in}	
-	-
输出参数{out}	
-	-
返回值	
FlagStatus	是否获取到标志
SET	获取到标志
RESET	未获取到标志

例如：

```
/* get tcmsram erase busy flag */
```

```
FlagStatus Flag = RESET;
```

```
Flag = syscfg_tcmsram_busy_flag_get();
```

函数 syscfg_compensation_cell_ready_flag_get

函数syscfg_compensation_cell_ready_flag_get描述见下表：

表 3-1127. 函数 syscfg_compensation_cell_ready_flag_get

函数名称	syscfg_compensation_cell_ready_flag_get
函数原型	FlagStatus syscfg_compensation_cell_ready_flag_get(void);
功能描述	获取补偿单元就绪标志
先决条件	-
被调用函数	-
输入参数{in}	
-	-
输出参数{out}	

-	-
返回值	
FlagStatus	是否获取到标志
SET	获取到标志
RESET	未获取到标志

例如：

```
/* get compensation cell ready flag */
```

```
FlagStatus Flag = RESET;
```

```
Flag = syscfg_compensation_cell_ready_flag_get();
```

3.30. 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, 19），通用定时器L0（TIMERx, x=1~4），通用定时器L3（TIMERx, x=14），通用定时器L4（TIMERx, x=15, 16），基本定时器（TIMERx, x=5, 6），不同类型的定时器具体功能有所差别。章节[3.30.1](#)描述了TIMER的寄存器列表，章节[3.30.2](#)对TIMER库函数进行说明。

3.30.1. 外设寄存器说明

TIMER寄存器列表如下表所示：

表 3-1128. TIMER 寄存器

寄存器名称	寄存器描述
TIMER_CTL0	控制寄存器0
TIMER_CTL1	控制寄存器1
TIMER_SMCFG	从模式配置寄存器
TIMER_DMAINTEN	DMA和中断使能寄存器
TIMER_INTF	中断标志寄存器
TIMER_SWEVG	软件事件产生寄存器
TIMER_CHCTL0	通道控制寄存器0
TIMER_CHCTL1	通道控制寄存器1
TIMER_CHCTL2	通道控制寄存器2
TIMER_CNT	计数器寄存器
TIMER_PSC	预分频寄存器
TIMER_CAR	计数器自动重载寄存器
TIMER_CREP0	重复计数寄存器0
TIMER_CH0CV	通道0捕获/比较寄存器
TIMER_CH1CV	通道1捕获/比较寄存器
TIMER_CH2CV	通道2捕获/比较寄存器
TIMER_CH3CV	通道3捕获/比较寄存器

寄存器名称	寄存器描述
TIMER_CCHP0	互补通道保护寄存器0
TIMER_MCHCTL0	TIMER多模式通道控制寄存器0
TIMER_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_WDGPFR	看门狗计数器周期寄存器
TIMER_CREP1	重复计数寄存器1
TIMER_CCHP1	TIMER互补通道保护寄存器1
TIMER_DECCTL	TIMER译码器控制寄存器
TIMER_CINITCTL	TIMER计数器初始控制寄存器
TIMER_CINITV	TIMER计数器初始值寄存器
TIMER_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

3.30.2. 外设库函数说明

TIMER库函数列表如下表所示：

表 3-1129. 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

库函数名称	库函数描述
timer_update_event_enable	TIMER更新使能
timer_update_event_disable	TIMER更新禁能
timer_counter_alignment	设置外设TIMER的对齐模式
timer_counter_up_direction	设置外设TIMER向上计数
timer_counter_down_direction	设置外设TIMER向下计数
timer_adjustment_mode_config	配置外设TIMER的微调模式功能
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_auto_reload_fract_value_config	配置外设TIMER的自动重载寄存器的小数部分值
timer_autoreload_value_read	读取TIMER自动重载寄存器计数器值
timer_auto_reload_fract_value_read	读取TIMER自动重载寄存器计数器的小数部分值
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_ocpre_clear_source_config	配置外设TIMER的OCPRE_CLR_INT输入源
timer_ocpre_clear_input_config	配置外设TIMER的OCPRE_CLR输入
timer_pulse_on_compare_config	配置外设TIMER的比较脉冲输出功能
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	将TIMER通道输出参数结构体中所有参数初始化为默认值

库函数名称	库函数描述
_para_init	
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_pulse_fraction_value_config	配置外设TIMER的通道输出比较值的小数部分
timer_channel_output_shadow_config	配置TIMER通道输出比较影子寄存器功能
timer_channel_output_clear_config	配置TIMER的通道输出比较清0功能
timer_channel_output_compare_fast_config	配置TIMER的通道输出比较快速输出功能
timer_channel_output_polarity_config	通道输出极性配置
timer_channel_complementary_output_polarity_config	互补通道输出极性配置
timer_channel_output_state_config	配置通道状态
timer_channel_complementary_output_state_config	配置互补通道输出状态
timer_channel_input_struct_para_init	将TIMER通道输入参数结构体中所有参数初始化为默认值
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_pause_reset_polarity_config	配置外设TIMER的暂停+复位模式的极性

库函数名称	库函数描述
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配置为正交编码器模式
timer_decoder_mode_config	TIMER配置为译码器模式
timer_internal_clock_config	TIMER配置为内部时钟模式
timer_internal_trigger_as_external_clock_config	配置TIMER的内部触发为时钟源
timer_external_trigger_as_external_clock_config	配置TIMER的外部触发作为时钟源
timer_external_clock_mode0_config	配置TIMER外部时钟模式0，ETI作为时钟源
timer_external_clock_mode1_config	配置TIMER外部时钟模式1
timer_external_clock_mode1_disable	TIMER外部时钟模式1禁能
timer_write_chxval_register_config	配置TIMER写CHxVAL选择位
timer_output_value_selection_config	配置TIMER输出值选择位
timer_commutation_control_shadow_register_config	配置换相控制影子寄存器更新选择
timer_output_match_pulse_select	配置TIMER输出匹配脉冲选择
timer_channel_composite_pwm_mode_config	配置TIMER的复合PWM模式
timer_channel_composite_pwm_mode_output_pulse_value_config	配置TIMER的复合PWM模式输出脉冲值
timer_channel_additional_compare_value_config	配置TIMER通道附加比较寄存器值
timer_channel_additional_output_shadow_config	配置TIMER通道附加输出比较影子寄存器功能
timer_channel_additional_output_update_select	选择TIMER通道附加输出比较寄存器更新源
timer_channel_additional_compare_value_read	读取TIMER通道附加输出比较寄存器值
timer_break_external_source_config	配置TIMER中止功能外部输入源
timer_break_external_polarity_config	配置TIMER中止功能输入极性
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_dead_time_falling_edge_config	配置TIMER下降沿死区时间
timer_dead_time_different_config	配置TIMER的不同死区时间功能
timer_dead_time_modify_config	配置TIMER的死区时间在线修改功能
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

库函数名称	库函数描述
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	正交译码器信号断线检测看门狗计数器值配置
timer_watchdog_value_read	读取正交译码器信号断线检测看门狗计数器值
timer_decoder_disconnection_detection_config	正交译码器信号断线检测功能配置
timer_decoder_jump_detection_config	正交译码器信号信号跳变检测功能配置
timer_decoder_modify_config	配置译码器模式在线修改功能
timer_decoder_mode_update_source_config	配置译码器模式更新源
timer_index_reset_counter_config	配置索引信号复位计数器功能
timer_index_reset_direction_config	配置译码器的索引信号复位寄存器的方向
timer_decoder_index_position_config	配置索引信号的定位
timer_first_index_reset_counter_config	配置仅第一个索引信号复位计数器的功能
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_counter_initial_register_config	配置计数器初始值寄存器
timer_counter_initial_config	配置计数器的初始值和计数方向
timer_synchronization_event_generate	产生软件同步事件
timer_flag_get	获取外设TIMER的状态标志
timer_flag_clear	清除外设TIMER状态标志
timer_interrupt_enable	外设TIMER中断使能
timer_interrupt_disable	外设TIMER中断禁能
timer_interrupt_flag_get	获取外设TIMER中断标志
timer_interrupt_flag_clear	清除外设TIMER的中断标志

结构体 timer_parameter_struct

表 3-1130. 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	周期值的整数部分 0~0xFFFF (微调模式禁能, TIMERx(x=0,2,3,5~7,14~16,19)) 0~0xFFFFE(微调模式使能, TIMERx(x=0,2,3,5~7,14~16,19))

成员名称	功能描述
	0~0xFFFFFFFF(微调模式禁能, TIMERx(x=1,4)) 0~0xFFFFFFFFE(微调模式使能, TIMERx(x=1,4))
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 3-1131. 结构体 timer_break_parameter_struct

成员名称	功能描述
runoffstate	运行模式下“关闭状态”配置 (TIMER_ROS_STATE_ENABLE, TIMER_ROS_STATE_DISABLE)
ideloffstate	空闲模式下“关闭状态”配置 (TIMER_IOS_STATE_ENABLE, TIMER_IOS_STATE_DISABLE)
deadtime	死区时间 (0~255)
outputautostate	自动输出使能 (TIMER_OUTAUTO_ENABLE, TIMER_OUTAUTO_DISABLE)
protectmode	互补寄存器保护控制 (TIMER_CCHP0_PROT_OFF, TIMER_CCHP0_PROT_0, TIMER_CCHP0_PROT_1, TIMER_CCHP0_PROT_2)
break0state	BREAK0输入使能 (TIMER_BREAK0_ENABLE, TIMER_BREAK0_DISABLE)
break0filter	BREAK0输入滤波 (0~15)
break0polarity	BREAK0输入极性 (TIMER_BREAK0_POLARITY_LOW, TIMER_BREAK0_POLARITY_HIGH)
break0lock	BREAK0输入锁存功能 (TIMER_BREAK0_LK_ENABLE, TIMER_BREAK0_LK_DISABLE)
break0release	BREAK0输入锁存释放功能 (TIMER_BREAK0_RELEASE, TIMER_BREAK0_UNRELEASE)
break1state	BREAK1输入使能 (TIMER_BREAK1_ENABLE, TIMER_BREAK1_DISABLE)
break1filter	BREAK1输入滤波 (0~15)
break1polarity	BREAK1输入极性 (TIMER_BREAK1_POLARITY_LOW, TIMER_BREAK1_POLARITY_HIGH)
break1lock	BREAK1输入锁存功能 (TIMER_BREAK1_LK_ENABLE, TIMER_BREAK1_LK_DISABLE)
break1release	BREAK1输入锁存释放功能 (TIMER_BREAK1_RELEASE, TIMER_BREAK1_UNRELEASE)

结构体 timer_oc_parameter_struct

表 3-1132. 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 3-1133. 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 3-1134. 结构体 timer_ic_parameter_struct

成员名称	功能描述
icpolarity	通道输入极性 (TIMER_IC_POLARITY_RISING, TIMER_IC_POLARITY_FALLING, TIMER_IC_POLARITY_BOTH_EDGE)
icselection	通道输入模式选择 (TIMER_IC_SELECTION_DIRECTTI, TIMER_IC_SELECTION_INDIRECTTI, TIMER_IC_SELECTION_ITS, TIMER_IC_SELECTION_PAIR)
icprescaler	通道输入捕获预分频 (TIMER_IC_PSC_DIV1, TIMER_IC_PSC_DIV2, TIMER_IC_PSC_DIV4, TIMER_IC_PSC_DIV8)
icfilter	通道输入捕获滤波 (0~15)

结构体 timer_free_complementary_parameter_struct

表 3-1135. 结构体 timer_free_complementary_parameter_struct

成员名称	功能描述
freecomstate	独立互补通道保护使能 (TIMER_FCCHP_STATE_ENABLE, TIMER_FCCHP_STATE_DISABLE)

成员名称	功能描述
runoffstate	运行模式下“关闭状态”配置（TIMER_ROS_STATE_ENABLE, TIMER_ROS_STATE_DISABLE）
ideloffstate	空闲模式下“关闭状态”配置（TIMER_IOS_STATE_ENABLE, TIMER_IOS_STATE_DISABLE）
deadtime	死区时间（0~255）
fallingdeadtime	下降沿死区时间（0~255）

函数 timer_deinit

函数timer_deinit描述见下表：

表 3-1136. 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, 19)	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 3-1137. 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 3-1138. 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, 19)	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 3-1139. 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, 19)	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 3-1140. 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, 19)	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 3-1141. 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, 19)	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 3-1142. 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, 19)	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 3-1143. 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, 19)	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 3-1144. 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, 19)	TIMER外设选择

16,19)	
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 3-1145. 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=0~4,7,19)	TIMER外设选择
输入参数{in}	
aligned	对齐模式
TIMER_COUNTER_EDGE	边沿对齐模式
TIMER_COUNTER_CENTER_DOWN	中央对齐向下计数模式
TIMER_COUNTER_CENTER_UP	中央对齐向上计数模式
TIMER_COUNTER_CENTER_BOTH	中央对齐上下计数模式
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* set TIMER0 counter center-aligned and counting up assert mode */
timer_counter_alignment(TIMER0, TIMER_COUNTER_CENTER_UP);
```

函数 timer_counter_up_direction

函数timer_counter_up_direction描述见下表：

表 3-1146. 函数 timer_counter_up_direction

函数名称	timer_counter_up_direction
函数原型	void timer_counter_up_direction(uint32_t timer_periph);
功能描述	设置外设TIMER向上计数
先决条件	计数器设置为无中央对齐计数模式（边沿对齐模式）
被调用函数	-
输入参数{in}	
timer_periph	TIMER外设
TIMERx(x=0~4,7,19)	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 3-1147. 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=0~4,7,19)	TIMER外设选择
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* set TIMER0 counter down direction */
```

```
timer_counter_down_direction(TIMERO);
```

函数 timer_adjustment_mode_config

函数timer_adjustment_mode_config描述见下表：

表 3-1148. 函数 timer_adjustment_mode_config

函数名称	timer_adjustment_mode_config
函数原型	void timer_adjustment_mode_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, 19)	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 TIMER adjustment mode function */
```

```
timer_adjustment_mode_config(TIMERO, ENABLE);
```

函数 timer_prescaler_config

函数timer_prescaler_config描述见下表：

表 3-1149. 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~	TIMER外设选择

16, 19)	
输入参数{in}	
prescaler	预分频值, 0~0xFFFF
输入参数{in}	
pscreload	预分频值加载模式
TIMER_PSC_RELOAD_NOW	预分频值立即加载
TIMER_PSC_RELOAD_UPDATE	预分频值在下次更新事件发生时加载
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* configure TIMER0 prescaler */
```

```
timer_prescaler_config(TIMER0, 3000, TIMER_PSC_RELOAD_NOW);
```

函数 timer_repetition_value_config

函数timer_repetition_value_config描述见下表:

表 3-1150. 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, 19)	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 3-1151. 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=0,7,14~16,19)	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 3-1152. 函数 timer_autoreload_value_config

函数名称	timer_autoreload_value_config
函数原型	void timer_autoreload_value_config(uint32_t timer_periph, uint64_t autoreload);
功能描述	配置外设TIMER的自动重载寄存器
先决条件	-
被调用函数	-
输入参数{in}	

timer_periph	TIMER外设
<i>TIMERx(x=0~7,14~16,19)</i>	TIMER外设选择
输入参数{in}	
autoreload	计数器自动重载值的整数部分， 0~0xFFFF（微调模式禁能，TIMERx(x=0,2,3,5~7,14~16,19)） 0~0xFFFE（微调模式使能，TIMERx(x=0,2,3,5~7,14~16,19)） 0~0xFFFFFFFF（微调模式禁能，TIMERx(x=1,4)） 0~0xFFFFFFFFE（微调模式使能，TIMERx(x=1,4)）
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* configure TIMER autoreload register value */
timer_autoreload_value_config(TIMER0, 3000);
```

函数 timer_auto_reload_fract_value_config

函数timer_auto_reload_fract_value_config描述见下表：

表 3-1153. 函数 timer_auto_reload_fract_value_config

函数名称	timer_auto_reload_fract_value_config
函数原型	void timer_auto_reload_fract_value_config(uint32_t timer_periph, uint32_t fract);
功能描述	配置外设TIMER的自动重载寄存器的小数部分值
先决条件	-
被调用函数	-
输入参数{in}	
timer_periph	TIMER外设
<i>TIMERx(x=0~7,14~16,19)</i>	TIMER外设选择
输入参数{in}	
autoreload	计数器自动重载值的小数部分，0~F
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* configure TIMER fractional part of auto reload register value */
timer_auto_reload_fract_value_config (TIMER0, 15);
```


函数 timer_autoreload_value_read

函数timer_autoreload_value_read描述见下表：

表 3-1154. 函数 timer_autoreload_value_read

函数名称	timer_autoreload_value_read
函数原型	uint32_t timer_autoreload_value_read(uint32_t timer_periph);
功能描述	读取外设TIMER自动重载寄存器值
先决条件	-
被调用函数	-
输入参数{in}	
timer_periph	TIMER外设
TIMERx(x=0~7,14~16,19)	TIMER外设选择
输出参数{out}	
-	-
返回值	
uint32_t	计数器自动重载值的整数部分， 0~0xFFFF（微调模式禁能，TIMERx(x=0,2,3,5~7,14~16,19)） 0~0xFFFFE（微调模式使能，TIMERx(x=0,2,3,5~7,14~16,19)） 0~0xFFFFFFFF（微调模式禁能，TIMERx(x=1,4)） 0~0xFFFFFFFFE（微调模式使能，TIMERx(x=1,4)）

例如：

```
/* get TIMER autoreload register value */
uint32_t i = 0;
i =(uint32_t) timer_autoreload_value_read (TIMER0);
```

函数 timer_auto_reload_fract_value_read

函数timer_auto_reload_fract_value_read描述见下表：

表 3-1155. 函数 timer_auto_reload_fract_value_read

函数名称	timer_auto_reload_fract_value_read
函数原型	uint32_t timer_auto_reload_fract_value_read(uint32_t timer_periph);
功能描述	读取TIMER自动重载寄存器计数器的小数部分值
先决条件	-
被调用函数	-
输入参数{in}	
timer_periph	TIMER外设
TIMERx(x=0~7,14~16,19)	TIMER外设选择
输出参数{out}	

-	-
返回值	
uint32_t	计数器自动重载值，0~F

例如：

```
/* get TIMER fractional part of auto reload register value */
```

```
uint32_t i = 0;
```

```
i =(uint32_t) timer_auto_reload_fract_value_read (TIMER0);
```

函数 timer_counter_value_config

函数timer_counter_value_config描述见下表：

表 3-1156. 函数 timer_counter_value_config

函数名称	timer_counter_value_config
函数原型	void timer_counter_value_config(uint32_t timer_periph, uint32_t counter);
功能描述	配置外设TIMER的计数器值
先决条件	-
被调用函数	-
输入参数{in}	
timer_periph	TIMER外设
TIMERx(x=0~7,14~16,19)	TIMER外设选择
输入参数{in}	
counter	计数器值 0~0xFFFF, TIMERx(x=0,2,3,5~7,14~16,19) 0~0xFFFFFFFF, TIMERx(x=1,4)
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 3-1157. 函数 timer_counter_read

函数名称	timer_counter_read
函数原型	uint32_t timer_counter_read(uint32_t timer_periph);

功能描述	读取外设TIMER的计数器值
先决条件	-
被调用函数	-
输入参数{in}	
timer_periph	TIMER外设
TIMERx(x=0~7, 14~16, 19)	TIMER外设选择
输出参数{out}	
-	-
返回值	
uint32_t	外设TIMER的计数器值 0~0xFFFF, TIMERx(x=0,2,3,5~7,14~16,19) 0~0xFFFFFFFF, TIMERx(x=1,4)

例如:

```
/* read TIMER0 counter value */
uint32_t i = 0;
i = timer_counter_read(TIMER0);
```

函数 timer_prescaler_read

函数timer_prescaler_read描述见下表:

表 3-1158. 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, 19)	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 3-1159. 函数 `timer_single_pulse_mode_config`

函数名称	<code>timer_single_pulse_mode_config</code>
函数原型	<code>void timer_single_pulse_mode_config(uint32_t timer_periph, uint32_t spmode);</code>
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}	
<code>timer_periph</code>	TIMER外设
<code>TIMERx(x=0~7,14~16,19)</code>	TIMER外设选择
输入参数{in}	
<code>spmode</code>	脉冲模式
<code>TIMER_SP_MODE_SINGLE</code>	单脉冲模式计数
<code>TIMER_SP_MODE_REPETITIVE</code>	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 3-1160. 函数 `timer_delayable_single_pulse_mode_config`

函数名称	<code>timer_delayable_single_pulse_mode_config</code>
函数原型	<code>void timer_delayable_single_pulse_mode_config(uint32_t timer_periph, uint16_t channel, uint32_t dspmode, uint16_t cnt_dir);</code>
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	<code>timer_input_trigger_source_select</code>
输入参数{in}	
<code>timer_periph</code>	TIMER外设
<code>TIMERx(x=0~4,7,14~19)</code>	TIMER外设选择

输入参数{in}	
channel	TIMER通道
<i>TIMER_CH_0</i>	通道0, TIMERx(x=0~4,7,14,19)
<i>TIMER_CH_1</i>	通道1, TIMERx(x=0~4,7,14,19)
<i>TIMER_CH_2</i>	通道2, TIMERx(x=0~4,7,19)
<i>TIMER_CH_3</i>	通道3, TIMERx(x=0~4,7,19)
<i>TIMER_MCH_0</i>	多模式通道0, TIMERx(x=0,7,14,19)
<i>TIMER_MCH_1</i>	多模式通道1, TIMERx(x=0,7,19)
<i>TIMER_MCH_2</i>	多模式通道2, TIMERx(x=0,7,19)
<i>TIMER_MCH_3</i>	多模式通道3, TIMERx(x=0,7,19)
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 3-1161. 函数 timer_update_source_config

函数名称	timer_update_source_config
函数原型	void timer_update_source_config(uint32_t timer_periph, uint32_t update);
功能描述	配置外设TIMER的更新源
先决条件	-
被调用函数	-
输入参数{in}	

timer_periph	TIMER外设
<i>TIMERx</i> (<i>x</i> =0~7, 14~16, 19)	TIMER外设选择
输入参数{in}	
update	更新源
<i>TIMER_UPDATE_S</i> <i>RC_GLOBAL</i>	下述任一事件产生更新中断或DMA请求： – UPG位被置1 – 计数器溢出/下溢 – 从模式控制器产生的更新
<i>TIMER_UPDATE_S</i> <i>RC_REGULAR</i>	只有计数器溢出/ 下溢才产生更新中断或DMA请求
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* configure TIMER update only by counter overflow/underflow */
```

```
timer_update_source_config(TIMER0, TIMER_UPDATE_SRC_REGULAR);
```

函数 timer_ocpre_clear_source_config

函数timer_ocpre_clear_source_config描述见下表：

表 3-1162. 函数 timer_ocpre_clear_source_config

函数名称	timer_ocpre_clear_source_config
函数原型	void timer_ocpre_clear_source_config(uint32_t timer_periph, uint32_t clear_source);
功能描述	配置外设TIMER的OCPRE_CLR_INT输入源
先决条件	-
被调用函数	-
输入参数{in}	
timer_periph	TIMER外设
<i>TIMERx</i> (<i>x</i> =0~4, 7, 19)	TIMER外设选择
输入参数{in}	
clear_source	OCPRE_CLR_INT源
<i>TIMER_OCPRE_CLR_INT_OCPRE_CLR</i> <i>R</i>	OCPRE_CLR_INT连接到OCPRE_CLR输入
<i>TIMER_OCPRE_CLR_INT_ETIF</i>	OCPRE_CLR_INT连接到ETIF
输出参数{out}	

-	-
返回值	
-	-

例如：

```
/* configure timer OCPRE_CLR_INT source */
```

```
timer_ocpre_clear_source_config (TIMER0, TIMER_OCPRE_CLR_INT_ETIF);
```

函数 timer_ocpre_clear_input_config

函数timer_ocpre_clear_input_config描述见下表：

表 3-1163. 函数 timer_ocpre_clear_input_config

函数名称	timer_ocpre_clear_input_config
函数原型	void timer_ocpre_clear_source_config(uint32_t timer_periph, uint32_t clear_source);
功能描述	配置外设TIMER的OCPRE_CLR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, 19)	TIMER外设选择
输入参数{in}	
clear_source	OCPRE_CLR源
TIMER_OCPRE_CLR_INPUT_x	OCPRE_CLR_INT连接到OCPRE_CLR x(x=0~7)
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* configure timer OCPRE_CLR input */
```

```
timer_ocpre_clear_input_config (TIMER0, TIMER_OCPRE_CLR_INPUT_7);
```

函数 timer_pulse_on_compare_config

函数timer_pulse_on_compare_config描述见下表：

表 3-1164. 函数 timer_pulse_on_compare_config

函数名称	timer_pulse_on_compare_config
函数原型	void timer_pulse_on_compare_config(uint32_t timer_periph, uint32_t

	pulse_width, uint32_t pulse_prescaler);
功能描述	配置外设TIMER的比较脉冲输出功能
先决条件	-
被调用函数	-
输入参数{in}	
timer_periph	TIMER外设
TIMERx(x=0,7,19)	TIMER外设选择
输入参数{in}	
pulse_width	输出脉冲宽度, 0~255
输入参数{in}	
pulse_prescaler	输出脉冲预分频, 0~7
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* configure the TIMER pulse on compare function */
```

```
timer_pulse_on_compare_config (TIMER0, 100, 7);
```

函数 timer_dma_enable

函数timer_dma_enable描述见下表:

表 3-1165. 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=0~7,14~16,19)	参考具体参数
输入参数{in}	
dma	DMA源
TIMER_DMA_UPD	更新DMA请求, TIMERx(x=0~7,14~16,19)
TIMER_DMA_CH0 D	通道0比较/捕获DMA请求, TIMERx (x=0~4,7,14~16,19)
TIMER_DMA_CH1 D	通道1比较/捕获DMA请求, TIMERx (x=0~4,7,14,19)
TIMER_DMA_CH2 D	通道2比较/捕获DMA请求, TIMERx (x=0~4,7,19)

<i>TIMER_DMA_CH3</i> <i>D</i>	通道3比较/捕获DMA请求, TIMERx (x=0~4,7,19)
<i>TIMER_DMA_CMT</i> <i>D</i>	换相DMA更新请求, TIMERx (x=0,7,14~16,19)
<i>TIMER_DMA_TRG</i> <i>D</i>	触发DMA请求, TIMERx (x=0~4,7,14,19)
<i>TIMER_DMA_MCH</i> <i>0D</i>	多模式通道0比较/捕获DMA请求, TIMERx (x=0,7,14~16,19)
<i>TIMER_DMA_MCH</i> <i>1D</i>	多模式通道1比较/捕获DMA请求, TIMERx(x=0,7,19)
<i>TIMER_DMA_MCH</i> <i>2D</i>	多模式通道2比较/捕获DMA请求, TIMERx(x=0,7,19)
<i>TIMER_DMA_MCH</i> <i>3D</i>	多模式通道3比较/捕获DMA请求, TIMERx (x=0,7,19)
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 3-1166. 函数 timer_dma_disable

函数名称	timer_dma_disable
函数原型	void timer_dma_disable (uint32_t timer_periph, uint32_t dma);
功能描述	外设TIMER的DMA禁能
先决条件	-
被调用函数	-
输入参数{in}	
timer_periph	TIMER外设
TIMERx(x=0~7,14~16,19)	参考具体参数
输入参数{in}	
dma	DMA源
TIMER_DMA_UPD	更新DMA请求, TIMERx(x=0~7,14~16,19)
TIMER_DMA_CH0 <i>D</i>	通道0比较/捕获DMA请求, TIMERx (x=0~4,7,14~16,19)
TIMER_DMA_CH1	通道1比较/捕获DMA请求, TIMERx (x=0~4,7,14,19)

<i>D</i>	
<i>TIMER_DMA_CH2</i> <i>D</i>	通道2比较/捕获DMA请求, TIMERx (x=0~4,7,19)
<i>TIMER_DMA_CH3</i> <i>D</i>	通道3比较/捕获DMA请求, TIMERx (x=0~4,7,19)
<i>TIMER_DMA_CMT</i> <i>D</i>	换相DMA更新请求, TIMERx (x=0,7,14~16,19)
<i>TIMER_DMA_TRG</i> <i>D</i>	触发DMA请求, TIMERx (x=0~4,7,14,19)
<i>TIMER_DMA_MCH</i> <i>0D</i>	多模式通道0比较/捕获DMA请求, TIMERx (x=0,7,14~16,19)
<i>TIMER_DMA_MCH</i> <i>1D</i>	多模式通道1比较/捕获DMA请求, TIMERx(x=0,7,19)
<i>TIMER_DMA_MCH</i> <i>2D</i>	多模式通道2比较/捕获DMA请求, TIMERx(x=0,7,19)
<i>TIMER_DMA_MCH</i> <i>3D</i>	多模式通道3比较/捕获DMA请求, TIMERx (x=0,7,19)
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 3-1167. 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=0~4,7,14~16,19)	TIMER外设选择
输入参数{in}	
dma_request	通道的DMA请求源选择

<code>TIMER_DMAREQUEST_CHANNELEVENT</code>	当通道捕获/比较事件发生时，发送通道n的DMA请求
<code>TIMER_DMAREQUEST_UPDATEEVENT</code>	当更新事件发生，发送通道n的DMA请求
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* TIMER0 channel DMA request of channel n is sent when channel event occurs */
```

```
timer_channel_dma_request_source_select (TIMER0,  
TIMER_DMAREQUEST_CHANNELEVENT);
```

函数 timer_dma_transfer_config

函数timer_dma_transfer_config描述见下表：

表 3-1168. 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
<code>TIMERx(x=0~4,7,14~16,19)</code>	参考具体参数
输入参数{in}	
dma_baseaddr	DMA传输起始地址
<code>TIMER_DMACFG_DMATA_CTL0</code>	DMA传输起始地址：TIMER_CTL0，TIMERx(x=0~4,7,14~16,19)
<code>TIMER_DMACFG_DMATA_CTL1</code>	DMA传输起始地址：TIMER_CTL1，TIMERx(x=0~4,7,14~16,19)
<code>TIMER_DMACFG_DMATA_SMCFG</code>	DMA传输起始地址：TIMER_SMCFG，TIMERx(x=0~4,7,14,19)
<code>TIMER_DMACFG_DMATA_DMAINTEN</code>	DMA传输起始地址：TIMER_DMAINTEN，TIMERx(x=0~4,7,14~16,19)
<code>TIMER_DMACFG_</code>	DMA传输起始地址：TIMER_INTF，TIMERx(x=0~4,7,14~16,19)

<i>DMATA_INTF</i>	
<i>TIMER_DMACFG_DMATA_SWEVG</i>	DMA传输起始地址: TIMER_SWEVG, TIMERx(x=0~4,7,14~16,19)
<i>TIMER_DMACFG_DMATA_CHCTL0</i>	DMA传输起始地址: TIMER_CHCTL0, TIMERx(x=0~4,7,14~16,19)
<i>TIMER_DMACFG_DMATA_CHCTL1</i>	DMA传输起始地址: TIMER_CHCTL1, TIMERx(x=0~4,7,19)
<i>TIMER_DMACFG_DMATA_CHCTL2</i>	DMA传输起始地址: TIMER_CHCTL2, TIMERx(x=0~4,7,14~16,19)
<i>TIMER_DMACFG_DMATA_CNT</i>	DMA传输起始地址: TIMER_CNT, TIMERx(x=0~4,7,14~16,19)
<i>TIMER_DMACFG_DMATA_PSC</i>	DMA传输起始地址: TIMER_PSC, TIMERx(x=0~4,7,14~16,19)
<i>TIMER_DMACFG_DMATA_CAR</i>	DMA传输起始地址: TIMER_CAR, TIMERx(x=0~4,7,14~16,19)
<i>TIMER_DMACFG_DMATA_CREP0</i>	DMA传输起始地址: TIMER_CREP0, TIMERx(x=0,7,14~16,19)
<i>TIMER_DMACFG_DMATA_CH0CV</i>	DMA传输起始地址: TIMER_CH0CV, TIMERx(x=0~4,7,14~16,19)
<i>TIMER_DMACFG_DMATA_CH1CV</i>	DMA传输起始地址: TIMER_CH1CV, TIMERx(x=0~4,7,14,19)
<i>TIMER_DMACFG_DMATA_CH2CV</i>	DMA传输起始地址: TIMER_CH2CV, TIMERx(x=0~4,7,19)
<i>TIMER_DMACFG_DMATA_CH3CV</i>	DMA传输起始地址: TIMER_CH3CV, TIMERx(x=0~4,7,19)
<i>TIMER_DMACFG_DMATA_CCHP0</i>	DMA传输起始地址: TIMER_CCHP0, TIMERx(x=0~4,7,14~16,19)
<i>TIMER_DMACFG_DMATA_MCHCTL0</i>	DMA传输起始地址: TIMER_MCHCTL0, TIMERx(x=0~4,7,14~16,19)
<i>TIMER_DMACFG_DMATA_MCHCTL1</i>	DMA传输起始地址: TIMER_MCHCTL1, TIMERx(x=0,7,19)
<i>TIMER_DMACFG_DMATA_MCHCTL2</i>	DMA传输起始地址: TIMER_MCHCTL2, TIMERx(x=0,7,14~16,19)
<i>TIMER_DMACFG_DMATA_MCH0CV</i>	DMA传输起始地址: TIMER_MCH0CV, TIMERx(x=0,7,14~16,19)
<i>TIMER_DMACFG_DMATA_MCH1CV</i>	DMA传输起始地址: TIMER_MCH1CV, TIMERx(x=0,7,19)
<i>TIMER_DMACFG_DMATA_MCH2CV</i>	DMA传输起始地址: TIMER_TIMER_MCH2CV, TIMERx(x=0,7,19)
<i>TIMER_DMACFG_DMATA_MCH3CV</i>	DMA传输起始地址: TIMER_TIMER_MCH3CV, TIMERx(x=0,7,19)
<i>TIMER_DMACFG_DMATA_CH0COMV</i>	DMA传输起始地址: TIMER_CH0COMV_ADD, TIMERx(x=0~4,7,14,19)

<code>TIMER_DMACFG_DMATA_CH1COMV_ADD</code>	DMA传输起始地址: <code>TIMER_CH1COMV_ADD</code> , <code>TIMERx</code> ($x=0\sim4,7,14,19$)
<code>TIMER_DMACFG_DMATA_CH2COMV_ADD</code>	DMA传输起始地址: <code>TIMER_CH2COMV_ADD</code> , <code>TIMERx</code> ($x=0\sim4,7,19$)
<code>TIMER_DMACFG_DMATA_CH3COMV_ADD</code>	DMA传输起始地址: <code>TIMER_CH3COMV_ADD</code> , <code>TIMERx</code> ($x=0\sim4,7,19$)
<code>TIMER_DMACFG_DMATA_CTL2</code>	DMA传输起始地址: <code>TIMER_CTL2</code> , <code>TIMERx</code> ($x=0\sim4,7,14\sim16,19$)
<code>TIMER_DMACFG_DMATA_FCCHP0</code>	DMA传输起始地址: <code>TIMER_FCCHP0</code> , <code>TIMERx</code> ($x=0,7,19$)
<code>TIMER_DMACFG_DMATA_FCCHP1</code>	DMA传输起始地址: <code>TIMER_FCCHP1</code> , <code>TIMERx</code> ($x=0,7,19$)
<code>TIMER_DMACFG_DMATA_FCCHP2</code>	DMA传输起始地址: <code>TIMER_FCCHP2</code> , <code>TIMERx</code> ($x=0,7,19$)
<code>TIMER_DMACFG_DMATA_FCCHP3</code>	DMA传输起始地址: <code>TIMER_FCCHP3</code> , <code>TIMERx</code> ($x=0,7,19$)
<code>TIMER_DMACFG_DMATA_AFCTL0</code>	DMA传输起始地址: <code>TIMER_AFCTL0</code> , <code>TIMERx</code> ($x=0,7,14\sim16,19$)
<code>TIMER_DMACFG_DMATA_AFCTL1</code>	DMA传输起始地址: <code>TIMER_AFCTL1</code> , <code>TIMERx</code> ($x=0,7,19$)
<code>TIMER_DMACFG_DMATA_WDGCNT</code>	DMA传输起始地址: <code>TIMER_WDGCNT</code> , <code>TIMERx</code> ($x=0\sim4,7,19$)
<code>TIMER_DMACFG_DMATA_CREP1</code>	DMA传输起始地址: <code>TIMER_CREP1</code> , <code>TIMERx</code> ($x=0,7,14\sim16,19$)
<code>TIMER_DMACFG_DMATA_CCHP1</code>	DMA传输起始地址: <code>TIMER_CCHP1</code> , <code>TIMERx</code> ($x=0,7,14\sim16,19$)
<code>TIMER_DMACFG_DMATA_DECCTL</code>	DMA传输起始地址: <code>TIMER_DECCTL</code> , <code>TIMERx</code> ($x=0\sim4,7,19$)
<code>TIMER_DMACFG_DMATA_CINITCTL</code>	DMA传输起始地址: <code>TIMER_CINITCTL</code> , <code>TIMERx</code> ($x=0,7,19$)
<code>TIMER_DMACFG_DMATA_CINITV</code>	DMA传输起始地址: <code>TIMER_CINITV</code> , <code>TIMERx</code> ($x=0,7,19$)
输入参数{in}	
<code>dma_lenth</code>	DMA传输长度
<code>TIMER_DMACFG_DMATC_xTRANSFER</code>	($x=1\sim42$), 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 3-1169. 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=0~7,14~16,19)	参考具体参数
输入参数{in}	
event	事件源
TIMER_EVENT_SRC_UPG	更新事件产生，TIMERx(x=0~7,14~16,19)
TIMER_EVENT_SRC_CH0G	通道0捕获或比较事件发生，TIMERx(x=0~4,7,14~16,19)
TIMER_EVENT_SRC_CH1G	通道1捕获或比较事件发生，TIMERx(x=0~4,7,14,19)
TIMER_EVENT_SRC_CH2G	通道2捕获或比较事件发生，TIMERx(x=0~4,7,19)
TIMER_EVENT_SRC_CH3G	通道3捕获或比较事件发生，TIMERx(x=0~4,7,19)
TIMER_EVENT_SRC_CMTG	通道换相更新事件发生，TIMERx(x=0,7,14~16,19)
TIMER_EVENT_SRC_TRGG	触发事件产生，TIMERx(x=0~4,7,14,19)
TIMER_EVENT_SRC_BRK0G	产生BREAK0事件，TIMERx(x=0,7,14~16,19)
TIMER_EVENT_SRC_BRK1G	产生BREAK1事件，TIMERx(x=0,7,19)
TIMER_EVENT_SRC	多模式通道0捕获或比较事件发生，TIMERx(x=0,7,14~16,19)

C_MCH0G	
TIMER_EVENT_SR C_MCH1G	多模式通道1捕获或比较事件发生, TIMERx(x=0,7,19)
TIMER_EVENT_SR C_MCH2G	多模式通道2捕获或比较事件发生, TIMERx(x=0,7,19)
TIMER_EVENT_SR C_MCH3G	多模式通道3捕获或比较事件发生, TIMERx(x=0,7,19)
TIMER_EVENT_SR C_CH0COMADDG	通道0附加比较事件发生, TIMERx(x=0~4,7,14,19)
TIMER_EVENT_SR C_CH1COMADDG	通道1附加比较事件发生, TIMERx(x=0~4,7,14,19)
TIMER_EVENT_SR C_CH2COMADDG	通道2附加比较事件发生, TIMERx(x=0~4,7,19)
TIMER_EVENT_SR C_CH3COMADDG	通道3附加比较事件发生, TIMERx(x=0~4,7,19)
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 3-1170. 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 3-1171. 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=0,7,14~16,19)	TIMER外设选择
输入参数{in}	
breakpara	中止功能配置结构体，详见 结构体timer_break_parameter_struct 。
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* configure TIMER0 break function */

timer_break_parameter_struct timer_breakpara;

timer_breakpara.runoffstate      = TIMER_ROS_STATE_DISABLE;

timer_breakpara.ideloffstate     = TIMER_IOS_STATE_DISABLE;

timer_breakpara.deadtime         = 0U;

timer_breakpara.outputautostate  = TIMER_OUTAUTO_ENABLE;

timer_breakpara.protectmode      = TIMER_CCHP0_PROT_OFF;

timer_breakpara.break0state      = TIMER_BREAK0_ENABLE;

timer_breakpara.break0filter     = 0U;

timer_breakpara.break0polarity   = TIMER_BREAK0_POLARITY_LOW;

timer_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 3-1172. 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=0,7,14~16,19)	TIMER外设选择
输入参数{in}	
break_num	TIMER BREAKx
TIMER_BREAK0	BREAK0输入信号，TIMERx(x=0,7,14~16,19)
TIMER_BREAK1	BREAK1输入信号，TIMERx(x=0,7,19)
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 3-1173. 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=0,7,14~16,19)	TIMER外设选择
输入参数{in}	
break_num	TIMER BREAKx
TIMER_BREAK0	BREAK0输入信号，TIMERx(x=0,7,14~16,19)
TIMER_BREAK1	BREAK1输入信号，TIMERx(x=0,7,19)
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 3-1174. 函数 timer_automatic_output_enable

函数名称	timer_automatic_output_enable
函数原型	void timer_automatic_output_enable(uint32_t timer_periph);
功能描述	自动输出使能
先决条件	只有在TIMERx_CCHP寄存器的PROT [1:0] =00 时，才可修改
被调用函数	-
输入参数{in}	
timer_periph	TIMER外设
TIMERx(x=0,7,14~16,19)	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 3-1175. 函数 timer_automatic_output_disable

函数名称	timer_automatic_output_disable
函数原型	void timer_automatic_output_disable (uint32_t timer_periph);
功能描述	自动输出禁能
先决条件	只有在TIMERx_CCHP寄存器的PROT [1:0] = 00时, 才可修改
被调用函数	-
输入参数{in}	
timer_periph	TIMER外设
TIMERx(x=0,7,14~16,19)	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 3-1176. 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=0,7,14~16,19)	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 3-1177. 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=0,7,14~16,19)	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 3-1178. 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=0,7,14~16,19)	TIMER外设选择
输入参数{in}	
ccuctl	通道换相控制影子寄存器更新控制
TIMER_UPDATECTL_CCUC	CMTG位被置1时更新影子寄存器
TIMER_UPDATECTL_CCUTRI	当CMTG位被置1或检测到TRIG1上升沿时，影子寄存器更新
TIMER_UPDATECTL_CCUCOVER	当计数器上溢事件发生时，影子寄存器更新
TIMER_UPDATECTL_CCUCUNDER	当计数器下溢事件发生时，影子寄存器更新
TIMER_UPDATECTL_CCUCOVERUNDER	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 3-1179. 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 3-1180. 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
<i>TIMERx(x=0~4,7,14~16,19)</i>	参考具体参数
输入参数{in}	
channel	TIMER通道
<i>TIMER_CH_0</i>	通道0, TIMERx(x=0~4,7,14~16,19)
<i>TIMER_CH_1</i>	通道1, TIMERx(x=0~4,7,14,19)
<i>TIMER_CH_2</i>	通道2, TIMERx(x=0~4,7,19)
<i>TIMER_CH_3</i>	通道3, TIMERx(x=0~4,7,19)
输入参数{in}	
ocpara	输出通道结构体，详见 结构体timer oc parameter struct 。
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* configure TIMER0 channel 0 output function */
timer_oc_parameter_struct timer_ocinitpara;
timer_ocintpara.outputstate = TIMER_CCX_ENABLE;
```

```

timer_ocintpara.outputnstate = TIMER_CCXN_ENABLE;

timer_ocintpara.ocpolarity   = TIMER_OC_POLARITY_HIGH;

timer_ocintpara.ocnpolarity  = TIMER_OCN_POLARITY_HIGH;

timer_ocintpara.ocidlestate  = TIMER_OC_IDLE_STATE_HIGH;

timer_ocintpara.ocnidlestate = TIMER_OCN_IDLE_STATE_LOW;

timer_channel_output_config(TIMER0, TIMER_CH_0, & timer_ocintpara);

```

函数 timer_channel_output_mode_config

函数timer_channel_output_mode_config描述见下表:

表 3-1181. 函数 timer_channel_output_mode_config

函数名称	timer_channel_output_mode_config
函数原型	void timer_channel_output_mode_config(uint32_t timer_periph, uint16_t channel, uint32_t ocmode);
功能描述	配置外设TIMER通道输出比较模式
先决条件	-
被调用函数	-
输入参数{in}	
timer_periph	TIMER外设
TIMERx(x=0~4,7,14~16,19)	参考具体参数
输入参数{in}	
channel	TIMER通道
TIMER_CH_0	通道0, TIMERx(x=0~4,7,14~16,19)
TIMER_CH_1	通道1, TIMERx(x=0~4,7,14,19)
TIMER_CH_2	通道2, TIMERx(x=0~4,7,19)
TIMER_CH_3	通道3, TIMERx(x=0~4,7,19)
TIMER_MCH_0	多模式通道0, TIMERx(x=0,7,14~16,19)
TIMER_MCH_1	多模式通道1, TIMERx(x=0,7,19)
TIMER_MCH_2	多模式通道2, TIMERx(x=0,7,19)
TIMER_MCH_3	多模式通道3, TIMERx(x=0,7,19)
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	匹配时翻转

<code>_TOGGLE</code>	
<code>TIMER_OC_MODE_LOW</code>	强制为低
<code>TIMER_OC_MODE_HIGH</code>	强制为高
<code>TIMER_OC_MODE_PWM0</code>	PWM模式0
<code>TIMER_OC_MODE_PWM1</code>	PWM模式1
<code>TIMER_OC_MODE_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
<code>TIMER_OC_MODE_PULSE</code>	比较脉冲输出, <code>TIMERx(x=0~4,7,19)</code> , <code>TIMER_CH_x(x=2~3)</code>
<code>TIMER_OC_MODE_DIR</code>	输出方向位, <code>TIMERx(x=0~4,7,19)</code> , <code>TIMER_CH_x(x=2~3)</code>
<code>TIMER_OC_MODE_APWM0</code>	非对称PWM模式0, <code>TIMERx(x=0~4,7,19)</code>
<code>TIMER_OC_MODE_APWM1</code>	非对称PWM模式1, <code>TIMERx(x=0~4,7,19)</code>
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 3-1182. 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

<i>TIMERx</i> (<i>x</i> =0~4,7,14~16,19)	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> =0~4,7,14~16,19)
<i>TIMER_CH_1</i>	通道1, <i>TIMERx</i> (<i>x</i> =0~4,7,14,19)
<i>TIMER_CH_2</i>	通道2, <i>TIMERx</i> (<i>x</i> =0~4,7,19)
<i>TIMER_CH_3</i>	通道3, <i>TIMERx</i> (<i>x</i> =0~4,7,19)
<i>TIMER_MCH_0</i>	多模式通道0, <i>TIMERx</i> (<i>x</i> =0,7,14~16,19)
<i>TIMER_MCH_1</i>	多模式通道1, <i>TIMERx</i> (<i>x</i> =0,7,19)
<i>TIMER_MCH_2</i>	多模式通道2, <i>TIMERx</i> (<i>x</i> =0,7,19)
<i>TIMER_MCH_3</i>	多模式通道3, <i>TIMERx</i> (<i>x</i> =0,7,19)
输入参数{in}	
pulse	通道输出比较值的整数部分 对于 <i>TIMER_CH_x</i> (<i>x</i> =0~3): 0~0xFFFF (微调模式禁能, <i>TIMERx</i> (<i>x</i> =0,2,3,5~7,14~16,19)) 0~0xFFFFE (微调模式使能, <i>TIMERx</i> (<i>x</i> =0,2,3,5~7,14~16,19)) 0~0xFFFFFFFF (微调模式禁能, <i>TIMERx</i> (<i>x</i> =1,4)) 0~0xFFFFFEE (微调模式使能, <i>TIMERx</i> (<i>x</i> =1,4)) 对于 <i>TIMER_MCH_x</i> (<i>x</i> =0~3): 0~0xFFFF (<i>TIMERx</i> (<i>x</i> =0,7,14~16,19))
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_pulse_fract_value_config

函数timer_channel_output_pulse_fract_value_config描述见下表:

表 3-1183. 函数 timer_channel_output_pulse_fract_value_config

函数名称	timer_channel_output_pulse_fract_value_config
函数原型	void timer_channel_output_pulse_fract_value_config(uint32_t timer_periph, uint16_t channel, uint32_t fract);
功能描述	配置外设TIMER的通道输出比较值的小数部分
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	参考具体参数

~16,19)	
输入参数{in}	
channel	TIMER通道
TIMER_CH_0	通道0, TIMERx(x=0~4,7,14~16,19)
TIMER_CH_1	通道1, TIMERx(x=0~4,7,14,19)
TIMER_CH_2	通道2, TIMERx(x=0~4,7,19)
TIMER_CH_3	通道3, TIMERx(x=0~4,7,19)
输入参数{in}	
pulse	
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* configure TIMER channel fractional part of output pulse value */
```

```
timer_channel_output_pulse_fract_value_config (TIMER0, TIMER_CH_0, 15);
```

函数 timer_channel_output_shadow_config

函数timer_channel_output_shadow_config描述见下表:

表 3-1184. 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=0~4,7,14~16,19)	参考具体参数
输入参数{in}	
channel	TIMER通道
TIMER_CH_0	通道0, TIMERx(x=0~4,7,14~16,19)
TIMER_CH_1	通道1, TIMERx(x=0~4,7,14,19)
TIMER_CH_2	通道2, TIMERx(x=0~4,7,19)
TIMER_CH_3	通道3, TIMERx(x=0~4,7,19)
TIMER_MCH_0	多模式通道0, TIMERx(x=0,7,14~16,19)
TIMER_MCH_1	多模式通道1, TIMERx(x=0,7,19)
TIMER_MCH_2	多模式通道2, TIMERx(x=0,7,19)
TIMER_MCH_3	多模式通道3, TIMERx(x=0,7,19)

输入参数{in}	
ocshadow	输出比较影子寄存器功能状态
TIMER_OC_SHADOW_ENABLE	通道输出比较影子寄存器使能
TIMER_OC_SHADOW_DISABLE	通道输出比较影子寄存器禁能
TIMER_OMC_SHADOW_ENABLE	多模式通道输出比较影子寄存器使能
TIMER_OMC_SHADOW_DISABLE	多模式通道输出比较影子寄存器禁能
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/*configure TIMER0 channel 0 output shadow function */
```

```
timer_channel_output_shadow_config(TIMER0,                                TIMER_CH_0,
timer_channel_output_shadow_config(TIMER0,                                TIMER_OC_SHADOW_ENABLE);
```

函数 timer_channel_output_clear_config

函数timer_channel_output_clear_config描述见下表：

表 3-1185. 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的通道输出比较清除功能
先决条件	-
被调用函数	-
输入参数{in}	
timer_periph	TIMER外设
TIMERx(x=0~4,7,14~16,19)	参考具体参数
输入参数{in}	
channel	TIMER通道
TIMER_CH_0	通道0, TIMERx(x=0~4,7,14~16,19)
TIMER_CH_1	通道1, TIMERx(x=0~4,7,14,19)
TIMER_CH_2	通道2, TIMERx(x=0~4,7,19)
TIMER_CH_3	通道3, TIMERx(x=0~4,7,19)
TIMER_MCH_0	多模式通道0, TIMERx(x=0,7,14~16,19)
TIMER_MCH_1	多模式通道1, TIMERx(x=0,7,19)

<i>TIMER_MCH_2</i>	多模式通道2, <i>TIMERx</i> (<i>x</i> =0,7,19)
<i>TIMER_MCH_3</i>	多模式通道3, <i>TIMERx</i> (<i>x</i> =0,7,19)
输入参数{in}	
occlear	通道比较输出清0功能状态
<i>TIMER_OC_CLEAR_ENABLE</i>	通道比较输出清0功能使能
<i>TIMER_OC_CLEAR_DISABLE</i>	通道比较输出清0功能禁能
<i>TIMER_OMC_CLEAR_ENABLE</i>	多模式通道比较输出清0功能使能
<i>TIMER_OMC_CLEAR_DISABLE</i>	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_compare_fast_config

函数timer_channel_output_compare_fast_config描述见下表:

表 3-1186. 函数 timer_channel_output_compare_fast_config

函数名称	timer_channel_output_compare_fast_config
函数原型	void timer_channel_output_compare_fast_config(uint32_t timer_periph, uint16_t channel, uint16_t ocfast);
功能描述	配置TIMER的通道输出比较快速输出功能
先决条件	-
被调用函数	-
输入参数{in}	
timer_periph	TIMER外设
<i>TIMERx</i> (<i>x</i> =0~4,7,14~16,19)	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> =0~4,7,14~16,19)
<i>TIMER_CH_1</i>	通道1, <i>TIMERx</i> (<i>x</i> =0~4,7,14,19)
<i>TIMER_CH_2</i>	通道2, <i>TIMERx</i> (<i>x</i> =0~4,7,19)
<i>TIMER_CH_3</i>	通道3, <i>TIMERx</i> (<i>x</i> =0~4,7,19)
<i>TIMER_MCH_0</i>	多模式通道0, <i>TIMERx</i> (<i>x</i> =0,7,14~16,19)

<i>TIMER_MCH_1</i>	多模式通道1, TIMERx(x=0,7,19)
<i>TIMER_MCH_2</i>	多模式通道2, TIMERx(x=0,7,19)
<i>TIMER_MCH_3</i>	多模式通道3, TIMERx(x=0,7,19)
输入参数{in}	
occlear	通道比较快速输出功能状态
<i>TIMER_OC_FAST_ENABLE</i>	通道比较快速输出功能使能
<i>TIMER_OC_FAST_DISABLE</i>	通道比较快速输出功能禁能
<i>TIMER_OMC_FAST_ENABLE</i>	多模式通道比较快速输出功能使能
<i>TIMER_OMC_FAST_DISABLE</i>	多模式通道比较快速输出功能禁能
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* configure TIMER0 channel0 output compare fast function */
```

```
timer_channel_output_compare_fast_config(TIMER0, TIMER_CH_0,
TIMER_OC_FAST_ENABLE);
```

函数 timer_channel_output_polarity_config

函数timer_channel_output_polarity_config描述见下表:

表 3-1187. 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
<i>TIMERx(x=0~4,7,14~16,19)</i>	参考具体参数
输入参数{in}	
channel	TIMER通道
<i>TIMER_CH_0</i>	通道0, TIMERx(x=0~4,7,14~16,19)
<i>TIMER_CH_1</i>	通道1, TIMERx(x=0~4,7,14,19)
<i>TIMER_CH_2</i>	通道2, TIMERx(x=0~4,7,19)

<i>TIMER_CH_3</i>	通道3, $TIMERx(x=0\sim4,7,19)$
<i>TIMER_MCH_0</i>	多模式通道0, $TIMERx(x=0,7,14\sim16,19)$
<i>TIMER_MCH_1</i>	多模式通道1, $TIMERx(x=0,7,19)$
<i>TIMER_MCH_2</i>	多模式通道2, $TIMERx(x=0,7,19)$
<i>TIMER_MCH_3</i>	多模式通道3, $TIMERx(x=0,7,19)$
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 3-1188. 函数 timer_channel_complementary_output_polarity_config

函数名称	timer_channel_complementary_output_polarity_config
函数原型	void timer_channel_complementary_output_polarity_config(uint32_t timer_periph, uint16_t channel, uint16_t ocpolarity);
功能描述	互补通道输出极性配置
先决条件	-
被调用函数	-
输入参数{in}	
timer_periph	TIMER外设
$TIMERx(x=0\sim4,7,14\sim16,19)$	TIMER外设选择
输入参数{in}	
channel	TIMER通道
<i>TIMER_CH_0</i>	通道0, $TIMERx(x=0\sim4,7,14\sim16,19)$

<i>TIMER_CH_1</i>	通道1, TIMERx(x=0~4,7,14,19)
<i>TIMER_CH_2</i>	通道2, TIMERx(x=0~4,7,19)
<i>TIMER_CH_3</i>	通道3, TIMERx(x=0~4,7,19)
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 3-1189. 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(x=0~4,7,14~16,19)</i>	参考具体参数
输入参数{in}	
channel	TIMER通道
<i>TIMER_CH_0</i>	通道0, TIMERx(x=0~4,7,14~16,19)
<i>TIMER_CH_1</i>	通道1, TIMERx(x=0~4,7,14,19)
<i>TIMER_CH_2</i>	通道2, TIMERx(x=0~4,7,19)
<i>TIMER_CH_3</i>	通道3, TIMERx(x=0~4,7,19)
<i>TIMER_MCH_0</i>	多模式通道0, TIMERx(x=0,7,14~16,19)
<i>TIMER_MCH_1</i>	多模式通道1, TIMERx(x=0,7,19)
<i>TIMER_MCH_2</i>	多模式通道2, TIMERx(x=0,7,19)

<i>TIMER_MCH_3</i>	多模式通道3, <i>TIMERx</i> (<i>x</i> =0,7,19)
输入参数{in}	
state	通道状态
<i>TIMER_CCX_ENABLE</i>	通道使能
<i>TIMER_CCX_DISABLE</i>	通道禁能
<i>TIMER_MCCX_ENABLE</i>	多模式通道使能
<i>TIMER_MCCX_DISABLE</i>	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 3-1190. 函数 timer_channel_complementary_output_state_config

函数名称	timer_channel_complementary_output_state_config
函数原型	void timer_channel_complementary_output_state_config(uint32_t timer_periph, uint16_t channel, uint16_t ocnstate);
功能描述	配置互补通道输出状态
先决条件	-
被调用函数	-
输入参数{in}	
timer_periph	TIMER外设
<i>TIMERx</i> (<i>x</i> =0,7,19)	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> =0~4,7,14~16,19)
<i>TIMER_CH_1</i>	通道1, <i>TIMERx</i> (<i>x</i> =0~4,7,14,19)
<i>TIMER_CH_2</i>	通道2, <i>TIMERx</i> (<i>x</i> =0~4,7,19)
<i>TIMER_CH_3</i>	通道3, <i>TIMERx</i> (<i>x</i> =0~4,7,19)
输入参数{in}	
state	互补通道状态
<i>TIMER_CCXN_ENA</i>	互补通道使能

<i>BLE</i>	
<i>TIMER_CCXN_DIS</i> <i>ABLE</i>	互补通道禁能
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* configure TIMER0 channel 0 complementary output enable state */
```

```
timer_channel_complementary_output_state_config(TIMER0,          TIMER_CH_0,
TIMER_CCXN_ENABLE);
```

函数 timer_channel_input_struct_para_init

函数timer_channel_input_struct_para_init描述见下表：

表 3-1191. 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 3-1192. 函数 timer_input_capture_config

函数名称	timer_input_capture_config
函数原型	void timer_input_capture_config(uint32_t timer_periph, uint16_t channel, timer_ic_parameter_struct* icpara);

功能描述	配置TIMER输入捕获参数
先决条件	-
被调用函数	timer_channel_input_capture_prescaler_config
输入参数{in}	
timer_periph	TIMER外设
TIMERx(x=0~4,7,14~16,19)	参考具体参数
输入参数{in}	
channel	TIMER通道
TIMER_CH_0	通道0, TIMERx(x=0~4,7,14~16,19)
TIMER_CH_1	通道1, TIMERx(x=0~4,7,14,19)
TIMER_CH_2	通道2, TIMERx(x=0~4,7,19)
TIMER_CH_3	通道3, TIMERx(x=0~4,7,19)
TIMER_MCH_0	多模式通道0, TIMERx(x=0,7,14~16,19)
TIMER_MCH_1	多模式通道1, TIMERx(x=0,7,19)
TIMER_MCH_2	多模式通道2, TIMERx(x=0,7,19)
TIMER_MCH_3	多模式通道3, TIMERx(x=0,7,19)
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 3-1193. 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=0~4,7,14~16,19)	参考具体参数
输入参数{in}	
channel	TIMER通道
TIMER_CH_0	通道0, TIMERx(x=0~4,7,14~16,19)
TIMER_CH_1	通道1, TIMERx(x=0~4,7,14,19)
TIMER_CH_2	通道2, TIMERx(x=0~4,7,19)
TIMER_CH_3	通道3, TIMERx(x=0~4,7,19)
TIMER_MCH_0	多模式通道0, TIMERx(x=0,7,14~16,19)
TIMER_MCH_1	多模式通道1, TIMERx(x=0,7,19)
TIMER_MCH_2	多模式通道2, TIMERx(x=0,7,19)
TIMER_MCH_3	多模式通道3, TIMERx(x=0,7,19)
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 3-1194. 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=0~4,7,14~16,19)	参考具体参数
输入参数{in}	
channel	TIMER通道
TIMER_CH_0	通道0, TIMERx(x=0~4,7,14~16,19)
TIMER_CH_1	通道1, TIMERx(x=0~4,7,14,19)
TIMER_CH_2	通道2, TIMERx(x=0~4,7,19)
TIMER_CH_3	通道3, TIMERx(x=0~4,7,19)
TIMER_MCH_0	多模式通道0, TIMERx(x=0,7,14~16,19)
TIMER_MCH_1	多模式通道1, TIMERx(x=0,7,19)
TIMER_MCH_2	多模式通道2, TIMERx(x=0,7,19)
TIMER_MCH_3	多模式通道3, TIMERx(x=0,7,19)
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 3-1195. 函数 timer_input_pwm_capture_config

函数名称	timer_input_pwm_capture_config
函数原型	void timer_input_pwm_capture_config(uint32_t timer_periph, uint16_t channel, timer_ic_parameter_struct* icpwm);
功能描述	配置TIMER捕获PWM输入参数
先决条件	-
被调用函数	timer_channel_input_capture_prescaler_config

输入参数{in}	
timer_periph	TIMER外设
<i>TIMERx</i> (<i>x</i> =0~4, 7, 14, 19)	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 3-1196. 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</i> (<i>x</i> =0~4, 7, 19)	TIMER外设选择
输入参数{in}	
hallmode	HALL接口功能状态

<code>TIMER_HALLINTERFACE_ENABLE</code>	HALL接口使能
<code>TIMER_HALLINTERFACE_DISABLE</code>	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 3-1197. 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 3-1198. 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=0,7,14~16,19)	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=0,7,14~16,19)
TIMER_MCH_1	多模式通道1, TIMERx(x=0,7,19)
TIMER_MCH_2	多模式通道2, TIMERx(x=0,7,19)
TIMER_MCH_3	多模式通道3, TIMERx(x=0,7,19)
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 3-1199. 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=0,7,14~16,19)	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=0,7,14~16,19)
TIMER_MCH_1	多模式通道1, TIMERx(x=0,7,19)
TIMER_MCH_2	多模式通道2, TIMERx(x=0,7,19)
TIMER_MCH_3	多模式通道3, TIMERx(x=0,7,19)
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 3-1200. 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)	TIMER外设选择

,19)	
输入参数{in}	
intrigger	输入触发源
<i>TIMER_SMCFG_T</i> <i>RGSEL_ITI0</i>	内部触发输入0 (ITI0, TIMERx(x=0~4,7,14,19))
<i>TIMER_SMCFG_T</i> <i>RGSEL_ITI1</i>	内部触发输入1 (ITI1, TIMERx(x=0~4,7,14,19))
<i>TIMER_SMCFG_T</i> <i>RGSEL_ITI2</i>	内部触发输入2 (ITI2, TIMERx(x=0~4,7,14,19))
<i>TIMER_SMCFG_T</i> <i>RGSEL_ITI3</i>	内部触发输入3 (ITI3, TIMERx(x=0~4,7,14,19))
<i>TIMER_SMCFG_T</i> <i>RGSEL_CIOF_ED</i>	TI0的边沿检测 (CIOF_ED, TIMERx(x=0~4,7,14,19))
<i>TIMER_SMCFG_T</i> <i>RGSEL_CIOFE0</i>	滤波后的通道0输入 (CIOFE0, TIMERx(x=0~4,7,14,19))
<i>TIMER_SMCFG_T</i> <i>RGSEL_CI1FE1</i>	滤波后的通道1输入 (CI1FE1, TIMERx(x=0~4,7,14,19))
<i>TIMER_SMCFG_T</i> <i>RGSEL_ETIFP</i>	滤波后的外部触发输入 (ETIFP, TIMERx(x=0~4,7,19))
<i>TIMER_SMCFG_T</i> <i>RGSEL_ITI4</i>	内部触发输入4 (ITI4, TIMERx(x=0~4,7,14,19))
<i>TIMER_SMCFG_T</i> <i>RGSEL_ITI5</i>	内部触发输入5 (ITI5, TIMERx(x=0~4,7,14,19))
<i>TIMER_SMCFG_T</i> <i>RGSEL_ITI6</i>	内部触发输入6 (ITI6, TIMERx(x=0~4,7,14,19))
<i>TIMER_SMCFG_T</i> <i>RGSEL_ITI7</i>	内部触发输入7 (ITI7, TIMERx(x=0~4,7,14,19))
<i>TIMER_SMCFG_T</i> <i>RGSEL_ITI8</i>	内部触发输入8 (ITI8, TIMERx(x=0~4,7,14,19))
<i>TIMER_SMCFG_T</i> <i>RGSEL_ITI9</i>	内部触发输入9 (ITI9, TIMERx(x=0~4,7,14,19))
<i>TIMER_SMCFG_T</i> <i>RGSEL_ITI10</i>	内部触发输入10 (ITI10, TIMERx(x=0~4,7,14,19))
<i>TIMER_SMCFG_T</i> <i>RGSEL_ITI14</i>	内部触发输入14 (ITI14, TIMERx(x=0~4,7,14,19))
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 3-1201. 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=0~7,14,19)	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=0~7,14,19))
TIMER_TRI_OUT0_SRC_ENABLE	TIMER使能信号作为TRGO0 (TIMERx(x=0~7,14,19))
TIMER_TRI_OUT0_SRC_UPDATE	更新事件作为TRGO0 (TIMERx(x=0~7,14,19))
TIMER_TRI_OUT0_SRC_CH0	通道0捕获/比较事件作为TRGO0 (TIMERx(x=0~4,7,14,19))
TIMER_TRI_OUT0_SRC_O0CPRE	O0CPRE作为触发输出TRGO0 (TIMERx(x=0~4,7,14,19))
TIMER_TRI_OUT0_SRC_O1CPRE	O1CPRE作为触发输出TRGO0 (TIMERx(x=0~4,7,14,19))
TIMER_TRI_OUT0_SRC_O2CPRE	O2CPRE作为触发输出TRGO0 (TIMERx(x=0~4,7,19))
TIMER_TRI_OUT0_SRC_O3CPRE	O3CPRE作为触发输出TRGO0 (TIMERx(x=0~4,7,19))
TIMER_TRI_OUT0_SRC_DECODER_CLOCK	译码器时钟作为触发输出TRGO0 (TIMERx(x=0~4,7,19))
TIMER_TRI_OUT0_SRC_SYNC	同步事件作为触发输出TRGO0 (TIMERx(x=0,7,19))
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 3-1202. 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=0,7,19)	TIMER外设选择
输入参数{in}	
outrigger	输出触发源
TIMER_TRI_OUT1_SRC_RESET	UPG位作为TRGO1
TIMER_TRI_OUT1_SRC_ENABLE	TIMER使能信号作为TRGO1
TIMER_TRI_OUT1_SRC_UPDATE	更新事件作为TRGO1
TIMER_TRI_OUT1_SRC_CH0	通道0捕获/比较事件作为TRGO1
TIMER_TRI_OUT1_SRC_O0CPRE	O0CPRE作为触发输出TRGO1
TIMER_TRI_OUT1_SRC_O1CPRE	O1CPRE作为触发输出TRGO1
TIMER_TRI_OUT1_SRC_O2CPRE	O2CPRE作为触发输出TRGO1
TIMER_TRI_OUT1_SRC_O3CPRE	O3CPRE作为触发输出TRGO1
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* select TIMER0 master mode output 1 trigger source */
```

```
timer_master_output1_trigger_source_select(TIMER0, TIMER_TRI_OUT1_SRC_RESET);
```

函数 timer_slave_mode_select

函数timer_slave_mode_select描述见下表:

表 3-1203. 函数 timer_slave_mode_select

函数名称	timer_slave_mode_select
函数原型	void timer_slave_mode_select(uint32_t timer_periph, uint32_t slavemode);
功能描述	TIMER从模式配置
先决条件	-
被调用函数	-
输入参数{in}	
timer_periph	TIMER外设
TIMERx(x=0~4,7,14,19)	TIMER外设选择
输入参数{in}	
slavemode	从模式
TIMER_SLAVE_MODE_DISABLE	关闭从模式, TIMERx(x=0~4,7,14,19)
TIMER_ENCODER_MODE0	正交译码器模式0, TIMERx(x=0~4,7,19)
TIMER_ENCODER_MODE1	正交译码器模式1, TIMERx(x=0~4,7,19)
TIMER_ENCODER_MODE2	正交译码器模式2, TIMERx(x=0~4,7,19)
TIMER_SLAVE_MODE_RESTART	复位模式, TIMERx(x=0~4,7,14,19)
TIMER_SLAVE_MODE_PAUSE	暂停模式, TIMERx(x=0~4,7,14,19)
TIMER_SLAVE_MODE_EVENT	事件模式, TIMERx(x=0~4,7,14,19)
TIMER_SLAVE_MODE_EXTERNAL0	外部时钟模式0, TIMERx(x=0~4,7,14,19)
TIMER_SLAVE_MODE_RESTART_EVENT	复位 + 事件模式, TIMERx(x=0~4,7,14,19)
TIMER_SLAVE_MODE_PAUSE_RESTART	暂停+ 复位模式, TIMERx(x=0~4,7,14,19)
TIMER_DECODER	译码器模式0, TIMERx(x=0~4,7,19)

<code>_MODE0</code>	
<code>TIMER_DECODER_MODE1</code>	译码器模式1, <code>TIMERx(x=0~4,7,19)</code>
<code>TIMER_DECODER_MODE2</code>	译码器模式2, <code>TIMERx(x=0~4,7,19)</code>
<code>TIMER_DECODER_MODE3</code>	译码器模式3, <code>TIMERx(x=0~4,7,19)</code>
<code>TIMER_QUAD_DECODER_MODE3</code>	正交译码器模式3, <code>TIMERx(x=0~4,7,19)</code>
<code>TIMER_QUAD_DECODER_MODE4</code>	正交译码器模式4, <code>TIMERx(x=0~4,7,19)</code>
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_pause_reset_polarity_config`

函数 `timer_pause_reset_polarity_config` 描述见下表:

表 3-1204. 函数 `timer_pause_reset_polarity_config`

函数名称	<code>timer_pause_reset_polarity_config</code>
函数原型	<code>void timer_master_slave_mode_config(uint32_t timer_periph, uint32_t masterslave);</code>
功能描述	配置外设 <code>TIMER</code> 的暂停+复位模式的极性
先决条件	-
被调用函数	-
输入参数{in}	
<code>timer_periph</code>	<code>TIMER</code> 外设
<code>TIMERx(x=0~4,7,14,19)</code>	<code>TIMER</code> 外设选择
输入参数{in}	
<code>rstpolarity</code>	复位极性
<code>TIMER_CNT_RESET_ON_FALLING_EDGE</code>	计数器在下降沿复位
<code>TIMER_CNT_RESET_ON_RISING_EDGE</code>	计数器在上升沿复位

输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* configure TIMER pause+reset mode reset polarity */
```

```
timer_pause_reset_polarity_config (TIMER0, TIMER_CNT_RESET_ON_FALLING_EDGE);
```

函数 timer_master_slave_mode_config

函数timer_master_slave_mode_config描述见下表：

表 3-1205. 函数 timer_master_slave_mode_config

函数名称	timer_master_slave_mode_config
函数原型	void timer_master_slave_mode_config(uint32_t timer_periph, uint32_t masterslave);
功能描述	TIMER主从模式配置
先决条件	-
被调用函数	-
输入参数{in}	
timer_periph	TIMER外设
TIMERx(x=0~4,7,14,19)	TIMER外设选择
输入参数{in}	
masterslave	主从模式使能状态
TIMER_MASTER_SLAVE_MODE_ENABLE	主从模式使能
TIMER_MASTER_SLAVE_MODE_DISABLE	主从模式禁能
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* configure TIMER0 master slave mode */
```

```
timer_master_slave_mode_config(TIMER0, TIMER_MASTER_SLAVE_MODE_ENABLE);
```

函数 timer_external_trigger_config

函数timer_external_trigger_config描述见下表:

表 3-1206. 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, 19)	TIMER外设选择
输入参数{in}	
extprescaler	外部触发预分频
TIMER_EXT_TRI_PSC_OFF	不分频
TIMER_EXT_TRI_PSC_DIV2	2分频
TIMER_EXT_TRI_PSC_DIV4	4分频
TIMER_EXT_TRI_PSC_DIV8	8分频
输入参数{in}	
expolarity	外部触发输入极性
TIMER_ETP_FALLING	低电平或者下降沿有效
TIMER_ETP_RISING	高电平或者上升沿有效
输入参数{in}	
extfilter	外部触发滤波控制 (0~15)
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* configure TIMER0 external trigger input */
```

```
timer_external_trigger_config(TIMER0, TIMER_EXT_TRI_PSC_DIV2,
TIMER_ETP_FALLING, 10);
```

函数 timer_quadrature_decoder_mode_config

函数timer_quadrature_decoder_mode_config描述见下表:

表 3-1207. 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配置为正交译码器模式
先决条件	-
被调用函数	-
输入参数{in}	
timer_periph	TIMER外设
TIMERx(x=0~4, 7, 19)	TIMER外设选择
输入参数{in}	
decomode	正交译码器模式
TIMER_QUAD_DECODER_MODE0	根据CI0FE0的电平, 计数器在CI1FE1的边沿向上/下计数
TIMER_QUAD_DECODER_MODE1	根据CI1FE1的电平, 计数器在CI0FE0的边沿向上/下计数
TIMER_QUAD_DECODER_MODE2	根据另一个信号的输入电平, 计数器在CI0FE0和CI1FE1的边沿向上/下计数
TIMER_QUAD_DECODER_MODE3	根据CI0FE0的电平, 计数器在CI0FE0的边沿向上/下计数
TIMER_QUAD_DECODER_MODE4	根据CI1FE1的电平, 计数器在CI1FE1的边沿向上/下计数
输入参数{in}	
ic0polarity	IC0输入极性
TIMER_IC_POLARITY_RISING	捕获上升边沿
TIMER_IC_POLARITY_FALLING	捕获下降边沿
TIMER_IC_POLARITY_BOTH_EDGE	双边沿有效
输入参数{in}	
ic1polarity	IC1输入极性
TIMER_IC_POLARITY_RISING	捕获上升边沿
TIMER_IC_POLARITY_FALLING	捕获下降边沿
TIMER_IC_POLARITY_BOTH_EDGE	双边沿有效

输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* configure TIMER0 quadrature decoder mode */
```

```
timer_quadrature_decoder_mode_config(TIMER0,          TIMER_ENCODER_MODE0,
TIMER_IC_POLARITY_RISING, TIMER_IC_POLARITY_RISING);
```

函数 timer_decoder_mode_config

函数timer_decoder_mode_config描述见下表:

表 3-1208. 函数 timer_decoder_mode_config

函数名称	timer_decoder_mode_config
函数原型	void timer_decoder_mode_config(uint32_t timer_periph, uint32_t decomode, uint16_t ic0polarity, uint16_t ic1polarity);
功能描述	TIMER配置为译码器模式
先决条件	-
被调用函数	-
输入参数{in}	
timer_periph	TIMER外设
TIMERx(x=0~4, 7, 19)	TIMER外设选择
输入参数{in}	
decomode	译码器模式
TIMER_DECODER_MODE0	译码器模式0
TIMER_DECODER_MODE1	译码器模式1
TIMER_DECODER_MODE2	译码器模式2
TIMER_DECODER_MODE3	译码器模式3
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 decoder mode */
```

```
timer_decoder_mode_config (TIMER0,          TIMER_DECODER_MODE2,
                           TIMER_IC_POLARITY_RISING, TIMER_IC_POLARITY_RISING);
```

函数 timer_internal_clock_config

函数timer_internal_clock_config描述见下表：

表 3-1209. 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, 19)</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 3-1210. 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, 19)	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)
TIMER_SMCFG_T RGSEL_ITI1	内部触发输入1 (ITI1)
TIMER_SMCFG_T RGSEL_ITI2	内部触发输入2 (ITI2)
TIMER_SMCFG_T RGSEL_ITI3	内部触发输入3 (ITI3)
TIMER_SMCFG_T RGSEL_ITI4	内部触发输入4 (ITI4)
TIMER_SMCFG_T RGSEL_ITI5	内部触发输入5 (ITI5)
TIMER_SMCFG_T RGSEL_ITI6	内部触发输入6 (ITI6)
TIMER_SMCFG_T RGSEL_ITI7	内部触发输入7 (ITI7)
TIMER_SMCFG_T RGSEL_ITI8	内部触发输入8 (ITI8)
TIMER_SMCFG_T RGSEL_ITI9	内部触发输入9 (ITI9)
TIMER_SMCFG_T RGSEL_ITI10	内部触发输入10 (ITI10)
TIMER_SMCFG_T RGSEL_ITI14	内部触发输入14 (ITI14)
输出参数{out}	
-	-

返回值	
-	-

例如：

```
/* configure TIMER0 the internal trigger ITIO as external clock input */
```

```
timer_internal_trigger_as_external_clock_config(TIMER0, TIMER_SMCFG_TRGSEL_ITIO);
```

函数 timer_external_trigger_as_external_clock_config

函数timer_external_trigger_as_external_clock_config描述见下表：

表 3-1211. 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=0~4,7,14,19)	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=0~4,7,14,19)）
TIMER_SMCFG_TRGSEL_CIOFE0	滤波后的通道0输入（CIOFE0，TIMERx(x=0~4,7,14,19)）
TIMER_SMCFG_TRGSEL_CIIFE1	滤波后的通道1输入（CIIFE1，TIMERx(x=0~4,7,14,19)）
输入参数{in}	
expolarity	外部触发源极性
TIMER_IC_POLARITY_RISING	外部触发源高电平或者上升沿有效
TIMER_IC_POLARITY_FALLING	外部触发源低电平或者下降沿有效
TIMER_IC_POLARITY_BOTH_EDGE	外部触发双边沿有效
输入参数{in}	
extfilter	滤波参数（0~15）
输出参数{out}	
-	-
返回值	

例如:

```
/* configure TIMER0 the external trigger CI0FE0 as external clock input */
```

```
timer_external_trigger_as_external_clock_config(TIMER0,  
TIMER_SMCFG_TRGSEL_CI0FE0, TIMER_IC_POLARITY_RISING, 0);
```

函数 timer_external_clock_mode0_config

函数timer_external_clock_mode0_config描述见下表:

表 3-1212. 函数 timer_external_clock_mode0_config

函数名称	timer_external_clock_mode0_config
函数原型	void timer_external_clock_mode0_config(uint32_t timer_periph, uint32_t extprescaler, uint32_t expolarity, uint32_t extfilter);
功能描述	配置TIMER外部时钟模式0, ETI作为时钟源
先决条件	-
被调用函数	timer_external_trigger_config
输入参数{in}	
timer_periph	TIMER外设
TIMERx(x=0~4, 7, 14, 19)	TIMER外设选择
输入参数{in}	
extprescaler	ETI触发源预分频值
TIMER_EXT_TRI_PSC_OFF	不分频
TIMER_EXT_TRI_PSC_DIV2	2分频
TIMER_EXT_TRI_PSC_DIV4	4分频
TIMER_EXT_TRI_PSC_DIV8	8分频
输入参数{in}	
expolarity	ETI触发源极性
TIMER_ETP_FALLING	下降沿或者低电平有效
TIMER_ETP_RISING	上升沿或者高电平有效
输入参数{in}	
extfilter	ETI触发源滤波参数 (0~15)
输出参数{out}	
-	-
返回值	

例如:

```
/* configure TIMER0 the external clock mode0 */
```

```
timer_external_clock_mode0_config(TIMER0, TIMER_EXT_TRI_PSC_DIV2,  
TIMER_ETP_FALLING, 0);
```

函数 timer_external_clock_mode1_config

函数timer_external_clock_mode1_config描述见下表:

表 3-1213. 函数 timer_external_clock_mode1_config

函数名称	timer_external_clock_mode1_config
函数原型	void timer_external_clock_mode1_config(uint32_t timer_periph, uint32_t extprescaler, uint32_t expolarity, uint32_t extfilter);
功能描述	配置TIMER外部时钟模式1
先决条件	-
被调用函数	timer_external_trigger_config
输入参数{in}	
timer_periph	TIMER外设
TIMERx(x=0~4, 7, 19)	TIMER外设选择
输入参数{in}	
extprescaler	ETI触发源预分频值
TIMER_EXT_TRI_PSC_OFF	不分频
TIMER_EXT_TRI_PSC_DIV2	2分频
TIMER_EXT_TRI_PSC_DIV4	4分频
TIMER_EXT_TRI_PSC_DIV8	8分频
输入参数{in}	
expolarity	ETI触发源极性
TIMER_ETP_FALLING	下降沿或者低电平有效
TIMER_ETP_RISING	上升沿或者高电平有效
输入参数{in}	
extfilter	ETI触发源滤波参数 (0~15)
输出参数{out}	
-	-
返回值	

-	-
---	---

例如:

```
/* configure TIMER0 the external clock mode1 */
```

```
timer_external_clock_mode1_config(TIMER0,                TIMER_EXT_TRI_PSC_DIV2,
TIMER_ETP_FALLING, 0);
```

函数 timer_external_clock_mode1_disable

函数timer_external_clock_mode1_disable描述见下表:

表 3-1214. 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=0~4,7,19)	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 3-1215. 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~	TIMER外设选择

16,19)	
输入参数{in}	
ccsel	写CHxVAL寄存器选择位
<i>TIMER_CHVSEL_DISABLE</i>	无影响
<i>TIMER_CHVSEL_ENABLE</i>	当写入捕获比较寄存器的值与寄存器当前值相等时，写入操作无效。
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* configure TIMER0 write CHxVAL register selection */
```

```
timer_write_chxval_register_config(TIMER0, TIMER_CHVSEL_ENABLE);
```

函数 timer_output_value_selection_config

函数timer_output_value_selection_config描述见下表：

表 3-1216. 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
<i>TIMERx(x=0,7,14~16,19)</i>	TIMER外设选择
输入参数{in}	
ccsel	输出值选择位
<i>TIMER_OUTSEL_DISABLE</i>	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 3-1217. 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=0,7,14~16,19)	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 3-1218. 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=0~4,7,14,19)	参考具体参数
输入参数{in}	
channel	TIMER通道
TIMER_CH_0	通道0, TIMERx(x=0~4,7,14~16,19)
TIMER_CH_1	通道1, TIMERx(x=0~4,7,14,19)
TIMER_CH_2	通道2, TIMERx(x=0~4,7,19)
TIMER_CH_3	通道3, TIMERx(x=0~4,7,19)
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 3-1219. 函数 timer_channel_composite_pwm_mode_config

函数名称	timer_channel_composite_pwm_mode_config
函数原型	void timer_channel_composite_pwm_mode_config(uint32_t timer_periph, uint32_t channel, ControlStatus newvalue);
功能描述	配置TIMER的复合PWM模式
先决条件	-
被调用函数	-
输入参数{in}	

timer_periph	TIMER外设
<i>TIMERx</i> (<i>x</i> =0~4,7,14,19))	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> =0~4,7,14~16,19)
<i>TIMER_CH_1</i>	通道1, <i>TIMERx</i> (<i>x</i> =0~4,7,14,19)
<i>TIMER_CH_2</i>	通道2, <i>TIMERx</i> (<i>x</i> =0~4,7,19)
<i>TIMER_CH_3</i>	通道3, <i>TIMERx</i> (<i>x</i> =0~4,7,19)
输入参数{in}	
newvalue	控制状态
<i>ENABLE</i>	使能
<i>DISABLE</i>	禁能
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* configure the TIMER composite PWM mode */
```

```
timer_channel_composite_pwm_mode_config (TIMER0, TIMER_CH_0, ENABLE);
```

函数 timer_channel_composite_pwm_mode_output_pulse_value_config

函数timer_channel_composite_pwm_mode_output_pulse_value_config描述见下表:

表 3-1220. 函数 timer_channel_composite_pwm_mode_output_pulse_value_config

函数名称	timer_channel_composite_pwm_mode_output_pulse_value_config
函数原型	void timer_channel_composite_pwm_mode_output_pulse_value_config(uint32_t timer_periph, uint32_t channel, uint32_t pulse, uint32_t add_pulse);
功能描述	配置TIMER的复合PWM模式输出脉冲值
先决条件	-
被调用函数	-
输入参数{in}	
timer_periph	TIMER外设
<i>TIMERx</i> (<i>x</i> =0~4,7,14,19))	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> =0~4,7,14~16,19)
<i>TIMER_CH_1</i>	通道1, <i>TIMERx</i> (<i>x</i> =0~4,7,14,19)
<i>TIMER_CH_2</i>	通道2, <i>TIMERx</i> (<i>x</i> =0~4,7,19)

<i>TIMER_CH_3</i>	通道3, <i>TIMERx</i> (<i>x</i> =0~4,7,19)
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 3-1221. 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
<i>TIMERx</i> (<i>x</i> =0~4,7,14,19)	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> =0~4,7,14~16,19)
<i>TIMER_CH_1</i>	通道1, <i>TIMERx</i> (<i>x</i> =0~4,7,14,19)
<i>TIMER_CH_2</i>	通道2, <i>TIMERx</i> (<i>x</i> =0~4,7,19)
<i>TIMER_CH_3</i>	通道3, <i>TIMERx</i> (<i>x</i> =0~4,7,19)
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 3-1222. 函数 timer_channel_additional_output_shadow_config

函数名称	timer_channel_additional_output_shadow_config
函数原型	void timer_channel_additional_output_shadow_config(uint32_t timer_periph, uint16_t channel, uint16_t aocshadow);
功能描述	配置TIMER通道附加输出比较影子寄存器功能
先决条件	-
被调用函数	-
输入参数{in}	
timer_periph	TIMER外设
TIMERx(x=0~4,7,14~16,19)	参考具体参数
输入参数{in}	
channel	TIMER通道
TIMER_CH_0	通道0, TIMERx(x=0~4,7,14~16,19)
TIMER_CH_1	通道1, TIMERx(x=0~4,7,14,19)
TIMER_CH_2	通道2, TIMERx(x=0~4,7,19)
TIMER_CH_3	通道3, TIMERx(x=0~4,7,19)
输入参数{in}	
aocshadow	通道附加输出比较影子寄存器状态
TIMER_ADD_SHADOW_ENABLE	通道附加输出比较影子寄存器使能
TIMER_ADD_SHADOW_DISABLE	通道附加输出比较影子寄存器禁能
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/*configure TIMER0 channel 0 additional output shadow function */
```

```
timer_channel_additional_output_shadow_config (TIMER0, TIMER_CH_0,
TIMER_OC_SHADOW_ENABLE);
```

函数 timer_channel_additional_output_update_select

函数timer_channel_additional_output_update_select描述见下表：

表 3-1223. 函数 timer_channel_additional_output_update_select

函数名称	timer_channel_additional_output_update_select
函数原型	void timer_channel_additional_output_update_select(uint32_t timer_periph, uint16_t channel, uint16_t update);
功能描述	选择TIMER通道附加输出比较寄存器更新源
先决条件	-
被调用函数	-
输入参数{in}	
timer_periph	TIMER外设
TIMERx(x=0~4,7,14~16,19)	参考具体参数
输入参数{in}	
channel	TIMER通道
TIMER_CH_0	通道0, TIMERx(x=0~4,7,14~16,19)
TIMER_CH_1	通道1, TIMERx(x=0~4,7,14,19)
TIMER_CH_2	通道2, TIMERx(x=0~4,7,19)
TIMER_CH_3	通道3, TIMERx(x=0~4,7,19)
输入参数{in}	
update	channel additional output register update
TIMER_ADD_UPDA TE_BY_UPDATE	channel additional compare value register update by update event
TIMER_ADD_UPDA TE_BY_COM_MAT CH	channel additional compare value register update by compare value match event
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* select TIMER channel additional output register update source */
```

```
timer_channel_additional_output_update_select(TIMER0, TIMER_CH_0,  
TIMER_ADD_UPDATE_BY_UPDATE);
```

函数 timer_channel_additional_compare_value_read

函数timer_channel_additional_compare_value_read描述见下表:

表 3-1224. 函数 timer_channel_additional_compare_value_read

函数名称	timer_channel_additional_compare_value_read
函数原型	uint32_t timer_channel_additional_compare_value_read(uint32_t timer_periph, uint16_t channel);
功能描述	读取TIMER通道附加输出比较寄存器值

先决条件	-
被调用函数	-
输入参数{in}	
timer_periph	TIMER外设
<i>TIMERx</i> (<i>x</i> =0~4, 7, 14, 19))	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, 14~16, 19)
<i>TIMER_CH_1</i>	通道1, <i>TIMERx</i> (<i>x</i> =0~4, 7, 14, 19)
<i>TIMER_CH_2</i>	通道2, <i>TIMERx</i> (<i>x</i> =0~4, 7, 19)
<i>TIMER_CH_3</i>	通道3, <i>TIMERx</i> (<i>x</i> =0~4, 7, 19)
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 3-1225. 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
<i>TIMERx</i> (<i>x</i> =0, 7, 14~16, 19)	参考具体参数
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, 19)
<i>TIMER_BREAK1</i>	BREAK1输入信号, <i>TIMERx</i> (<i>x</i> =0, 7, 19)
输入参数{in}	

break_src	中止源
TIMER_BRKIN0	BRKIN0备用功能输入使能, TIMERx(x=0,7,14~16,19)
TIMER_BRKIN1	BRKIN1备用功能输入使能, TIMERx(x=0,7,19)
TIMER_BRKIN2	BRKIN2备用功能输入使能, TIMERx(x=0,7,19)
TIMER_BRKCOMP0	CMP0输入使能, TIMERx(x=0,7,14~16,19)
TIMER_BRKCOMP1	CMP1输入使能, TIMERx(x=0,7,14~16,19)
TIMER_BRKCOMP2	CMP2输入使能, TIMERx(x=0,7,14~16,19)
TIMER_BRKCOMP3	CMP3输入使能, TIMERx(x=0,7,14~16,19)
TIMER_BRKCOMP4	CMP4输入使能, TIMERx(x=0,7,14~16,19)
TIMER_BRKCOMP5	CMP5输入使能, TIMERx(x=0,7,14~16,19)
TIMER_BRKCOMP6	CMP6输入使能, TIMERx(x=0,7,14~16,19)
TIMER_BRKCOMP7	CMP7输入使能, TIMERx(x=0,7,14~16,19)
TIMER_BRKHPDF	HPDF输入使能, TIMERx(x=0,7,14~16,19)
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 3-1226. 函数 timer_break_external_polarity_config

函数名称	timer_break_external_polarity_config
函数原型	void timer_break_external_polarity_config(uint32_t timer_periph, uint16_t break_num, uint32_t break_src, uint16_t bkinpolarity);
功能描述	配置TIMER中止功能输入极性
先决条件	-
被调用函数	-
输入参数{in}	
timer_periph	TIMER外设
TIMERx(x=0,7,14~16,19)	参考具体参数
输入参数{in}	

break_num	TIMER BREAKx
<i>TIMER_BREAK0</i>	BREAK0输入信号, TIMERx(x=0,7,14~16,19)
<i>TIMER_BREAK1</i>	BREAK1输入信号, TIMERx(x=0,7)
输入参数{in}	
break_src	中止源
<i>TIMER_BRKIN0</i>	BRKIN0备用功能输入使能, TIMERx(x=0,7,14~16,19)
<i>TIMER_BRKIN1</i>	BRKIN1备用功能输入使能, TIMERx(x=0,7,19)
<i>TIMER_BRKIN2</i>	BRKIN2备用功能输入使能, TIMERx(x=0,7,19)
<i>TIMER_BRKCOMP0</i>	CMP0输入使能, TIMERx(x=0,7,14~16,19)
<i>TIMER_BRKCOMP1</i>	CMP1输入使能, TIMERx(x=0,7,14~16,19)
<i>TIMER_BRKCOMP2</i>	CMP2输入使能, TIMERx(x=0,7,14~16,19)
<i>TIMER_BRKCOMP3</i>	CMP3输入使能, TIMERx(x=0,7,14~16,19)
输入参数{in}	
bkinpolarity	中止极性
<i>TIMER_BRKIN_POLARITY_LOW</i>	低电平有效
<i>TIMER_BRKIN_POLARITY_HIGH</i>	高电平有效
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* configure the TIMER break polarity */
```

```
timer_break_external_polarity_config(TIMER0,    TIMER_BREAK0,    TIMER_BRKIN0,
TIMER_BRKIN_POLARITY_HIGH);
```

函数 timer_break_lock_config

函数timer_break_lock_config描述见下表:

表 3-1227. 函数 timer_break_lock_config

函数名称	timer_break_lock_config
函数原型	void timer_break_lock_config(uint32_t timer_periph, uint16_t break_num, ControlStatus newvalue);
功能描述	配置TIMER锁存中止功能
先决条件	-
被调用函数	-
输入参数{in}	
timer_periph	TIMER外设
<i>TIMERx(x=0,7,14~16,19)</i>	参考具体参数

输入参数{in}	
break_num	TIMER BREAKx
<i>TIMER_BREAK0</i>	BREAK0输入信号, TIMERx(x=0,7,14~16,19)
<i>TIMER_BREAK1</i>	BREAK1输入信号, TIMERx(x=0,7,19)
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 3-1228. 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
<i>TIMERx(x=0,7,14~16,19)</i>	参考具体参数
输入参数{in}	
break_num	TIMER BREAKx
<i>TIMER_BREAK0</i>	BREAK0输入信号, TIMERx(x=0,7,14~16,19)
<i>TIMER_BREAK1</i>	BREAK1输入信号, TIMERx(x=0,7,19)
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_dead_time_falling_edge_config

函数timer_dead_time_falling_edge_config描述见下表:

表 3-1229. 函数 timer_dead_time_falling_edge_config

函数名称	timer_dead_time_falling_edge_config
函数原型	void timer_dead_time_falling_edge_config(uint32_t timer_periph, uint32_t deadtime);
功能描述	配置TIMER下降沿死区时间
先决条件	-
被调用函数	-
输入参数{in}	
timer_periph	TIMER外设
TIMEx(x=0,7,14~16,19)	参考具体参数
输入参数{in}	
deadtime	死区时间, 0~255
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* configure the TIMER falling edge dead time */
```

```
timer_dead_time_falling_edge_config (TIMER0, 255);
```

函数 timer_dead_time_different_config

函数timer_dead_time_different_config描述见下表:

表 3-1230. 函数 timer_dead_time_different_config

函数名称	timer_dead_time_different_config
函数原型	void timer_dead_time_different_config(uint32_t timer_periph, ControlStatus newvalue);
功能描述	配置TIMER的不同死区时间功能
先决条件	-
被调用函数	-

输入参数{in}	
timer_periph	TIMER外设
<i>TIMERx</i> (<i>x</i> =0,7,14~16,19)	参考具体参数
输入参数{in}	
newvalue	控制状态
ENABLE	使能
DISABLE	禁能
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* configure the TIMER dead time different function */
```

```
timer_dead_time_different_config (TIMER0, ENABLE);
```

函数 timer_dead_time_modify_config

函数timer_dead_time_modify_config描述见下表：

表 3-1231. 函数 timer_dead_time_modify_config

函数名称	timer_dead_time_modify_config
函数原型	void timer_dead_time_modify_config(uint32_t timer_periph, ControlStatus newvalue);
功能描述	配置TIMER的死区时间在线修改功能
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> =0,7,14~16,19)	参考具体参数
输入参数{in}	
newvalue	控制状态
ENABLE	使能
DISABLE	禁能
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* configure the TIMER dead time modified on-the-fly function */
```

timer_dead_time_modify_config (TIMER0, ENABLE);

函数 timer_channel_break_control_config

函数timer_channel_break_control_config描述见下表:

表 3-1232. 函数 timer_channel_break_control_config

函数名称	timer_channel_break_control_config
函数原型	void timer_channel_break_control_config(uint32_t timer_periph, uint32_t channel, ControlStatus newvalue);
功能描述	配置TIMER通道的中止功能
先决条件	-
被调用函数	-
输入参数{in}	
timer_periph	TIMER外设
TIMERx(x=0,7,19)	参考具体参数
输入参数{in}	
channel	TIMER通道
TIMER_CH_0	通道0
TIMER_CH_1	通道1
TIMER_CH_2	通道2
TIMER_CH_3	通道3
输入参数{in}	
newvalue	控制状态
ENABLE	使能
DISABLE	禁能
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* configure TIMER0 channel 0 break function */
```

```
timer_channel_break_control_config (TIMER0, TIMER_CH_0, ENABLE);
```

函数 timer_channel_dead_time_config

函数timer_channel_dead_time_config描述见下表:

表 3-1233. 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=0,7,19)	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 3-1234. 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 3-1235. 函数 timer_channel_free_complementary_config

函数名称	timer_channel_free_complementary_config
函数原型	void timer_channel_free_complementary_config(uint32_t timer_periph, uint16_t channel, timer_free_complementary_parameter_struct *fcpara);
功能描述	配置TIMER独立互补通道保护功能
先决条件	-
被调用函数	-
输入参数{in}	
timer_periph	TIMER外设
TIMERx(x=0,7,19)	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 3-1236. 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, 19)	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 3-1237. 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, 19)	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 3-1238. 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	正交译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=0~4,7,19)	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 3-1239. 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	正交译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, 19)	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_decoder_modify_config

函数timer_decoder_modify_config描述见下表：

表 3-1240. 函数 timer_decoder_jump_detection_config

函数名称	timer_decoder_modify_config
函数原型	void timer_decoder_modify_config(uint32_t timer_periph, ControlStatus newvalue);
功能描述	配置译码器模式在线修改功能
先决条件	-
被调用函数	-
输入参数{in}	
timer_periph	TIMER外设
TIMERx(x=0~4, 7, 19)	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 decoder mode modified on-the-fly function */
```

```
timer_decoder_modify_config (TIMER0, ENABLE);
```

函数 timer_decoder_mode_update_source_config

函数timer_decoder_mode_update_source_config描述见下表：

表 3-1241. 函数 timer_decoder_mode_update_source_config

函数名称	timer_decoder_mode_update_source_config
函数原型	void timer_decoder_mode_update_source_config(uint32_t timer_periph, uint32_t source);
功能描述	配置译码器模式更新源
先决条件	-
被调用函数	-
输入参数{in}	
timer_periph	TIMER外设
TIMEx(x=0~4,7,19)	参考具体参数
输入参数{in}	
source	更新源
TIMER_DEC_UPDA TE_BY_UPDATE	译码器模式在更新事件更新
TIMER_DEC_UPDA TE_BY_INDEX	译码器模式在索引事件更新
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* configure decoder mode update source */
```

```
timer_decoder_mode_update_source_config (TIMER0, TIMER_DEC_UPDATE_BY_UPDATE);
```

函数 timer_index_reset_counter_config

函数timer_index_reset_counter_config描述见下表：

表 3-1242. 函数 timer_index_reset_counter_config

函数名称	timer_index_reset_counter_config
函数原型	void timer_index_reset_counter_config(uint32_t timer_periph, ControlStatus newvalue);
功能描述	配置所有信号复位计数器功能

先决条件	-
被调用函数	-
输入参数{in}	
timer_periph	TIMER外设
TIMERx(x=0~4, 7, 19)	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 index signal reset counter function */
```

```
timer_index_reset_counter_config (TIMER0, ENABLE);
```

函数 timer_index_reset_direction_config

函数timer_index_reset_direction_config描述见下表：

表 3-1243. 函数 timer_index_reset_direction_config

函数名称	timer_index_reset_direction_config
函数原型	void timer_index_reset_direction_config(uint32_t timer_periph, uint32_t direction);
功能描述	配置译码器的索引信号复位寄存器的方向
先决条件	-
被调用函数	-
输入参数{in}	
timer_periph	TIMER外设
TIMERx(x=0~4, 7, 19)	参考具体参数
输入参数{in}	
direction	索引信号复位计数器方向
TIMER_INDEX_RESET_DIRECTION_BOTH	当计数器向上和向下计数时，索引信号复位计数器
TIMER_INDEX_RESET_DIRECTION_UP	仅在计数器向上计数时，索引信号复位计数器
TIMER_INDEX_RESET_DIRECTION_DOWN	仅在计数器向下计数时，索引信号复位计数器

SET_DIRECTION_DOWN	
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* configure index signal reset counter direction */
```

```
timer_index_reset_direction_config (TIMER0, TIMER_INDEX_RESET_DIRECTION_BOTH);
```

函数 timer_decoder_index_position_config

函数timer_decoder_index_position_config描述见下表:

表 3-1244. 函数 timer_decoder_index_position_config

函数名称	timer_decoder_index_position_config
函数原型	void timer_decoder_index_position_config(uint32_t timer_periph, uint32_t position);
功能描述	配置索引信号的定位
先决条件	-
被调用函数	-
输入参数{in}	
timer_periph	TIMER外设
TIMERx(x=0~4, 7, 19)	参考具体参数
输入参数{in}	
position	索引定位
TIMER_INDEX_POSITION_AB_00	当AB相输入为00时, 索引事件复位计数器 (正交译码器模式0~4)
TIMER_INDEX_POSITION_AB_01	当AB相输入为01时, 索引事件复位计数器 (正交译码器模式0~4)
TIMER_INDEX_POSITION_AB_10	当AB相输入为10时, 索引事件复位计数器 (正交译码器模式0~4)
TIMER_INDEX_POSITION_AB_11	当AB相输入为11时, 索引事件复位计数器 (正交译码器模式0~4)
TIMER_INDEX_POSITION_LOW	当时钟信号为低电平时, 索引事件复位计数器 (译码器模式0~3)
TIMER_INDEX_POSITION_HIGH	当时钟信号为高电平时, 索引事件复位计数器 (译码器模式0~3)
输出参数{out}	
-	-
返回值	

-	-
---	---

例如:

```
/* configure index position */
```

```
timer_decoder_index_position_config (TIMER0, TIMER_INDEX_POSITION_AB_01);
```

函数 timer_first_index_reset_counter_config

函数timer_first_index_reset_counter_config描述见下表:

表 3-1245. 函数 timer_first_index_reset_counter_config

函数名称	timer_first_index_reset_counter_config
函数原型	void timer_first_index_reset_counter_config(uint32_t timer_periph, ControlStatus newvalue);
功能描述	配置仅第一个索引信号复位计数器的功能
先决条件	-
被调用函数	-
输入参数{in}	
timer_periph	TIMER外设
TIMEx(x=0~4,7,19)	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 only the first index signal reset counter function */
```

```
timer_first_index_reset_counter_config (TIMER0, ENABLE);
```

函数 timer_upif_backup_config

函数timer_upif_backup_config描述见下表:

表 3-1246. 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=0~7,14~16,19)	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(TIMERO0, ENABLE);
```

函数 timer_upifbu_bit_get

函数timer_upifbu_bit_get描述见下表:

表 3-1247. 函数 timer_upifbu_bit_get

函数名称	timer_upifbu_bit_get
函数原型	UPIFBUSStatus timer_upifbu_bit_get(uint32_t timer_periph);
功能描述	获取TIMERx_CNT寄存器中的UPIFBU位
先决条件	-
被调用函数	-
输入参数{in}	
timer_periph	TIMER外设
TIMERx(x=0~7,14~16,19)	TIMER外设选择
输出参数{out}	
-	-
返回值	
UPIFBUSStatus	UPIFBU位状态
VALID_SET	UPIFBU位有效, 返回值为1
VALID_RESET	UPIFBU位有效, 返回值为0
INVALID	UPIFBU位无效

例如:

```
/* get the UPIFBU bit in the TIMERx_CNT register */
```

```
UPIFBUSStatus upstatus = INVALID;
```

```
upstatus = timer_upifbu_bit_get (TIMER0);
```

函数 timer_counter_initial_register_config

函数timer_counter_initial_register_config描述见下表：

表 3-1248. 函数 timer_counter_initial_register_config

函数名称	timer_counter_initial_register_config
函数原型	void timer_counter_initial_register_config(uint32_t timer_periph, ControlStatus newvalue);
功能描述	配置计数器初始值寄存器
先决条件	-
被调用函数	-
输入参数{in}	
timer_periph	TIMER外设
TIMERx(x=0,7,19)	TIMER外设选择
输入参数{in}	
newvalue	控制状态
ENABLE	使能
DISABLE	禁能
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* configure counter initial value register */
```

```
timer_counter_initial_register_config (TIMER0, ENABLE);
```

函数 timer_counter_initial_config

函数timer_counter_initial_config描述见下表：

表 3-1249. 函数 timer_counter_initial_config

函数名称	timer_counter_initial_config
函数原型	void timer_counter_initial_config(uint32_t timer_periph, uint32_t value, uint32_t direction);
功能描述	配置计数器初始值和计数方向
先决条件	-
被调用函数	-
输入参数{in}	
timer_periph	TIMER外设

<i>TIMERx(x=0,7,19)</i>	TIMER外设选择
输入参数{in}	
value	计数器初始值, 0~0xFFFF
输入参数{in}	
direction	计数器初始计数方向
<i>TIMER_INITIAL_DIRECTION_DOWN</i>	当同步事件发生时, 计数器向下计数
<i>TIMER_INITIAL_DIRECTION_UP</i>	当同步事件发生时, 计数器向上计数
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* configure counter initial value and direction */
```

```
timer_counter_initial_config (TIMER0, TIMER_INITIAL_DIRECTION_UP);
```

函数 timer_synchronization_event_generate

函数timer_synchronization_event_generate描述见下表:

表 3-1250. 函数 timer_synchronization_event_generate

函数名称	timer_synchronization_event_generate
函数原型	void timer_synchronization_event_generate(uint32_t timer_periph);
功能描述	产生软件同步事件
先决条件	-
被调用函数	-
输入参数{in}	
timer_periph	TIMER外设
<i>TIMERx(x=0,7,19)</i>	TIMER外设选择
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* generate soft synchronization event */
```

```
timer_synchronization_event_generate (TIMER0);
```

函数 timer_flag_get

函数timer_flag_get描述见下表:

表 3-1251. 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=0~7,14~16,19)	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=0~7,14~16,19)
TIMER_FLAG_CH0	通道0比较/捕获标志, TIMERx(x=0~4,7,14~16,19)
TIMER_FLAG_CH1	通道1比较/捕获标志, TIMERx(x=0~4,7,14,19)
TIMER_FLAG_CH2	通道2比较/捕获标志, TIMERx(x=0~4,7,19)
TIMER_FLAG_CH3	通道3比较/捕获标志, TIMERx(x=0~4,7,19)
TIMER_FLAG_CMT	通道换相更新标志, TIMERx(x=0,7,14~16,19)
TIMER_FLAG_TRG	触发标志, TIMERx(x=0~4,7,14,19)
TIMER_FLAG_BRK0	BREAK标志, TIMERx(x=0,7,14~16,19)
TIMER_FLAG_BRK1	TIMERx(x=0,7,19)
TIMER_FLAG_CH0O	通道0捕获溢出标志, TIMERx(x=0~4,7,14~16,19)
TIMER_FLAG_CH1O	通道1捕获溢出标志, TIMERx(x=0~4,7,14,19)
TIMER_FLAG_CH2O	通道2捕获溢出标志, TIMERx(x=0~4,7,19)
TIMER_FLAG_CH3O	通道3捕获溢出标志, TIMERx(x=0~4,7,19)
TIMER_FLAG_SYSB	系统源中止标志, TIMERx(x=0,7,19)
TIMER_FLAG_IND	索引标志位, TIMERx(x=0~4,7,19)
TIMER_FLAG_DECJ	正交译码器跳变标志, TIMERx(x=0~4,7,19)
TIMER_FLAG_DECDIS	正交译码器断线标志, TIMERx(x=0~4,7,19)
TIMER_FLAG_DIRTRAN	计数方向变化标志位, TIMERx(x=0~4,7,19)
TIMER_FLAG_INDErr	索引错误标志位, TIMERx(x=0~4,7,19)

<code>TIMER_FLAG_MCH0</code>	多模式通道0比较/捕获标志, <code>TIMERx(x=0,7,14~16,19)</code>
<code>TIMER_FLAG_MCH1</code>	多模式通道1比较/捕获标志, <code>TIMERx(x=0,7,19)</code>
<code>TIMER_FLAG_MCH2</code>	多模式通道2比较/捕获标志, <code>TIMERx(x=0,7,19)</code>
<code>TIMER_FLAG_MCH3</code>	多模式通道3比较/捕获标志, <code>TIMERx(x=0,7,19)</code>
<code>TIMER_FLAG_MCH0O</code>	多模式通道0捕获溢出标志, <code>TIMERx(x=0,7,14~16,19)</code>
<code>TIMER_FLAG_MCH1O</code>	多模式通道1捕获溢出标志, <code>TIMERx(x=0,7,19)</code>
<code>TIMER_FLAG_MCH2O</code>	多模式通道2捕获溢出标志, <code>TIMERx(x=0,7,19)</code>
<code>TIMER_FLAG_MCH3O</code>	多模式通道3捕获溢出标志, <code>TIMERx(x=0,7,19)</code>
<code>TIMER_FLAG_CH0COMADD</code>	通道0附加比较标志, <code>TIMERx(x=0~4,7,14,19)</code>
<code>TIMER_FLAG_CH1COMADD</code>	通道1附加比较标志, <code>TIMERx(x=0~4,7,14,19)</code>
<code>TIMER_FLAG_CH2COMADD</code>	通道2附加比较标志, <code>TIMERx(x=0~4,7,19)</code>
<code>TIMER_FLAG_CH3COMADD</code>	通道3附加比较标志, <code>TIMERx(x=0~4,7,19)</code>
输出参数{out}	
-	-
返回值	
FlagStatus	SET或者RESET

例如:

```
/* get TIMER0 update flags */
```

```
FlagStatus Flag_status = RESET;
```

```
Flag_status = timer_flag_get(TIMER0, TIMER_FLAG_UP);
```

函数 `timer_flag_clear`

函数`timer_flag_clear`描述见下表:

表 3-1252. 函数 `timer_flag_clear`

函数名称	<code>timer_flag_clear</code>
函数原型	<code>void timer_flag_clear(uint32_t timer_periph, uint32_t flag);</code>
功能描述	清除外设 <code>TIMERx</code> 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=0~7,14~16,19)	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=0~7,14~16,19)
TIMER_FLAG_CH0	通道0比较/捕获标志, TIMERx(x=0~4,7,14~16,19)
TIMER_FLAG_CH1	通道1比较/捕获标志, TIMERx(x=0~4,7,14,19)
TIMER_FLAG_CH2	通道2比较/捕获标志, TIMERx(x=0~4,7,19)
TIMER_FLAG_CH3	通道3比较/捕获标志, TIMERx(x=0~4,7,19)
TIMER_FLAG_CMT	通道换相更新标志, TIMERx(x=0,7,14~16,19)
TIMER_FLAG_TRG	触发标志, TIMERx(x=0~4,7,14,19)
TIMER_FLAG_BRK 0	BREAK标志, TIMERx(x=0,7,14~16,19)
TIMER_FLAG_BRK 1	TIMERx(x=0,7,19)
TIMER_FLAG_CH0 0	通道0捕获溢出标志, TIMERx(x=0~4,7,14~16,19)
TIMER_FLAG_CH1 0	通道1捕获溢出标志, TIMERx(x=0~4,7,14,19)
TIMER_FLAG_CH2 0	通道2捕获溢出标志, TIMERx(x=0~4,7,19)
TIMER_FLAG_CH3 0	通道3捕获溢出标志, TIMERx(x=0~4,7,19)
TIMER_FLAG_SYS B	系统源中止标志, TIMERx(x=0,7,19)
TIMER_FLAG_IND	索引标志位, TIMERx(x=0~4,7,19)
TIMER_FLAG_DEC J	正交译码器跳变标志, TIMERx(x=0~4,7,19)
TIMER_FLAG_DEC DIS	正交译码器断线标志, TIMERx(x=0~4,7,19)
TIMER_FLAG_DIR TRAN	计数方向变化标志位, TIMERx(x=0~4,7,19)
TIMER_FLAG_IND ERR	索引错误标志位, TIMERx(x=0~4,7,19)
TIMER_FLAG_MCH 0	多模式通道0比较/捕获标志, TIMERx(x=0,7,14~16,19)
TIMER_FLAG_MCH 1	多模式通道1比较/捕获标志, TIMERx(x=0,7,19)
TIMER_FLAG_MCH	多模式通道2比较/捕获标志, TIMERx(x=0,7,19)

2	
TIMER_FLAG_MCH3	多模式通道3比较/捕获标志, TIMERx(x=0,7,19)
TIMER_FLAG_MCH00	多模式通道0捕获溢出标志, TIMERx(x=0,7,14~16,19)
TIMER_FLAG_MCH10	多模式通道1捕获溢出标志, TIMERx(x=0,7,19)
TIMER_FLAG_MCH20	多模式通道2捕获溢出标志, TIMERx(x=0,7,19)
TIMER_FLAG_MCH30	多模式通道3捕获溢出标志, TIMERx(x=0,7,19)
TIMER_FLAG_CH0COMADD	通道0附加比较标志, TIMERx(x=0~4,7,14,19)
TIMER_FLAG_CH1COMADD	通道1附加比较标志, TIMERx(x=0~4,7,14,19)
TIMER_FLAG_CH2COMADD	通道2附加比较标志, TIMERx(x=0~4,7,19)
TIMER_FLAG_CH3COMADD	通道3附加比较标志, TIMERx(x=0~4,7,19)
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 3-1253. 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=0~7,14~16,19)	参考具体参数
输入参数{in}	

interrupt	中断源
TIMER_INT_UP	更新中断, TIMERx(x=0~7,14~16,19)
TIMER_INT_CH0	通道0比较/捕获中断, TIMERx(x=0~4,7,14~16,19)
TIMER_INT_CH1	通道1比较/捕获中断, TIMERx(x=0~4,7,14,19)
TIMER_INT_CH2	通道2比较/捕获中断, TIMERx(x=0~4,7,19)
TIMER_INT_CH3	通道3比较/捕获中断, TIMERx(x=0~4,7,19)
TIMER_INT_CMT	换相更新中断, TIMERx(x=0,7,14~16,19)
TIMER_INT_TRG	触发中断, TIMERx(x=0~4,7,14,19)
TIMER_INT_BRK	中止中断, TIMERx(x=0,7,14~16,19)
TIMER_INT_IND	索引中断, TIMERx(x=0~4,7,19)
TIMER_INT_DECJ	正交译码器跳变中断, TIMERx(x=0~4,7,22,23,30,31)
TIMER_INT_DECDIS	正交译码器断线中断, TIMERx(x=0~4,7,22,23,30,31)
TIMER_INT_DIRTRAN	计数方向变化中断, TIMERx(x=0~4,7,19)
TIMER_INT_INDERR	索引错误中断, TIMERx(x=0~4,7,19)
TIMER_INT_MCH0	多模式通道0比较/捕获中断, TIMERx(x=0,7,14~16,19)
TIMER_INT_MCH1	多模式通道1比较/捕获中断, TIMERx(x=0,7,19)
TIMER_INT_MCH2	多模式通道2比较/捕获中断, TIMERx(x=0,7,19)
TIMER_INT_MCH3	多模式通道3比较/捕获中断, TIMERx(x=0,7,19)
TIMER_INT_CH0COMADD	通道0附加比较中断, TIMERx(x=0~4,7,14,19)
TIMER_INT_CH1COMADD	通道1附加比较中断, TIMERx(x=0~4,7,14,19)
TIMER_INT_CH2COMADD	通道2附加比较中断, TIMERx(x=0~4,7,19)
TIMER_INT_CH3COMADD	通道3附加比较中断, TIMERx(x=0~4,7,19)
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 3-1254. 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=0~7,14~16,19)	参考具体参数
输入参数{in}	
interrupt	中断源
TIMER_INT_UP	更新中断, TIMERx(x=0~7,14~16,19)
TIMER_INT_CH0	通道0比较/捕获中断, TIMERx(x=0~4,7,14~16,19)
TIMER_INT_CH1	通道1比较/捕获中断, TIMERx(x=0~4,7,14,19)
TIMER_INT_CH2	通道2比较/捕获中断, TIMERx(x=0~4,7,19)
TIMER_INT_CH3	通道3比较/捕获中断, TIMERx(x=0~4,7,19)
TIMER_INT_CMT	换相更新中断, TIMERx(x=0,7,14~16,19)
TIMER_INT_TRG	触发中断, TIMERx(x=0~4,7,14,19)
TIMER_INT_BRK	中止中断, TIMERx(x=0,7,14~16,19)
TIMER_INT_IND	索引中断, TIMERx(x=0~4,7,19)
TIMER_INT_DECJ	正交译码器跳变中断, TIMERx(x=0~4,7,22,23,30,31)
TIMER_INT_DECDIS	正交译码器断线中断, TIMERx(x=0~4,7,22,23,30,31)
TIMER_INT_DIRTRAN	计数方向变化中断, TIMERx(x=0~4,7,19)
TIMER_INT_INDERR	索引错误中断, TIMERx(x=0~4,7,19)
TIMER_INT_MCH0	多模式通道0比较/捕获中断, TIMERx(x=0,7,14~16,19)
TIMER_INT_MCH1	多模式通道1比较/捕获中断, TIMERx(x=0,7,19)
TIMER_INT_MCH2	多模式通道2比较/捕获中断, TIMERx(x=0,7,19)
TIMER_INT_MCH3	多模式通道3比较/捕获中断, TIMERx(x=0,7,19)
TIMER_INT_CH0COMADD	通道0附加比较中断, TIMERx(x=0~4,7,14,19)
TIMER_INT_CH1COMADD	通道1附加比较中断, TIMERx(x=0~4,7,14,19)
TIMER_INT_CH2COMADD	通道2附加比较中断, TIMERx(x=0~4,7,19)
TIMER_INT_CH3COMADD	通道3附加比较中断, TIMERx(x=0~4,7,19)
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 3-1255. 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=0~7,14~16,19)	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=0~7,14~16,19)
TIMER_INT_FLAG_CH0	通道0比较/捕获中断标志, TIMERx(x=0~4,7,14~16,19)
TIMER_INT_FLAG_CH1	通道1比较/捕获中断标志, TIMERx(x=0~4,7,14,19)
TIMER_INT_FLAG_CH2	通道2比较/捕获中断标志, TIMERx(x=0~4,7,19)
TIMER_INT_FLAG_CH3	通道3比较/捕获中断标志, TIMERx(x=0~4,7,19)
TIMER_INT_FLAG_CMT	换相更新中断标志, TIMERx(x=0,7,14~16,19)
TIMER_INT_FLAG_TRG	触发中断标志, TIMERx(x=0~4,7,14,19)
TIMER_INT_FLAG_BRK0	BREAK0中断标志, TIMERx(x=0,7,14~16,19)
TIMER_INT_FLAG_BRK1	BREAK1中断标志, TIMERx(x=0,7,19)
TIMER_INT_FLAG_SYSB	系统源中止中断标志, TIMERx(x=0,7,19)

<i>TIMER_INT_FLAG_IND</i>	索引中断标志, <i>TIMERx</i> (<i>x</i> =0~4,7,19)
<i>TIMER_INT_FLAG_DECJ</i>	正交译码器跳变中断标志, <i>TIMERx</i> (<i>x</i> =0~4,7,19)
<i>TIMER_INT_FLAG_DECDIS</i>	正交译码器断线中断标志, <i>TIMERx</i> (<i>x</i> =0~4,7,19)
<i>TIMER_INT_FLAG_DIRTRAN</i>	计数方向变化中断标志, <i>TIMERx</i> (<i>x</i> =0~4,7,19)
<i>TIMER_INT_FLAG_INDERR</i>	索引错误中断标志, <i>TIMERx</i> (<i>x</i> =0~4,7,19)
<i>TIMER_INT_FLAG_MCH0</i>	多模式通道0比较/捕获中断标志, <i>TIMERx</i> (<i>x</i> =0,7,14~16,19)
<i>TIMER_INT_FLAG_MCH1</i>	多模式通道1比较/捕获中断标志, <i>TIMERx</i> (<i>x</i> =0,7,19)
<i>TIMER_INT_FLAG_MCH2</i>	多模式通道2比较/捕获中断标志, <i>TIMERx</i> (<i>x</i> =0,7,19)
<i>TIMER_INT_FLAG_MCH3</i>	多模式通道3比较/捕获中断标志, <i>TIMERx</i> (<i>x</i> =0,7,19)
<i>TIMER_INT_FLAG_CH0COMADD</i>	通道0附加比较中断标志, <i>TIMERx</i> (<i>x</i> =0~4,7,14,19)
<i>TIMER_INT_FLAG_CH1COMADD</i>	通道1附加比较中断标志, <i>TIMERx</i> (<i>x</i> =0~4,7,14,19)
<i>TIMER_INT_FLAG_CH2COMADD</i>	通道2附加比较中断标志, <i>TIMERx</i> (<i>x</i> =0~4,7,19)
<i>TIMER_INT_FLAG_CH3COMADD</i>	通道3附加比较中断标志, <i>TIMERx</i> (<i>x</i> =0~4,7,19)
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 3-1256. 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=0~7,14~16,19)	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=0~7,14~16,19)
TIMER_INT_FLAG_CH0	通道0比较/捕获中断标志, TIMERx(x=0~4,7,14~16,19)
TIMER_INT_FLAG_CH1	通道1比较/捕获中断标志, TIMERx(x=0~4,7,14,19)
TIMER_INT_FLAG_CH2	通道2比较/捕获中断标志, TIMERx(x=0~4,7,19)
TIMER_INT_FLAG_CH3	通道3比较/捕获中断标志, TIMERx(x=0~4,7,19)
TIMER_INT_FLAG_CMT	换相更新中断标志, TIMERx(x=0,7,14~16,19)
TIMER_INT_FLAG_TRG	触发中断标志, TIMERx(x=0~4,7,14,19)
TIMER_INT_FLAG_BRK0	BREAK0中断标志, TIMERx(x=0,7,14~16,19)
TIMER_INT_FLAG_BRK1	BREAK1中断标志, TIMERx(x=0,7,19)
TIMER_INT_FLAG_SYSB	系统源中止中断标志, TIMERx(x=0,7,19)
TIMER_INT_FLAG_IND	索引中断标志, TIMERx(x=0~4,7,19)
TIMER_INT_FLAG_DECJ	正交译码器跳变中断标志, TIMERx(x=0~4,7,19)
TIMER_INT_FLAG_DECDIS	正交译码器断线中断标志, TIMERx(x=0~4,7,19)
TIMER_INT_FLAG_DIRTRAN	计数方向变化中断标志, TIMERx(x=0~4,7,19)
TIMER_INT_FLAG_INDERR	索引错误中断标志, TIMERx(x=0~4,7,19)
TIMER_INT_FLAG_MCH0	多模式通道0比较/捕获中断标志, TIMERx(x=0,7,14~16,19)
TIMER_INT_FLAG_	多模式通道1比较/捕获中断标志, TIMERx(x=0,7,19)

<i>MCH1</i>	
<i>TIMER_INT_FLAG_MCH2</i>	多模式通道2比较/捕获中断标志，TIMERx(x=0,7,19)
<i>TIMER_INT_FLAG_MCH3</i>	多模式通道3比较/捕获中断标志，TIMERx(x=0,7,19)
<i>TIMER_INT_FLAG_CH0COMADD</i>	通道0附加比较中断标志，TIMERx(x=0~4,7,14,19)
<i>TIMER_INT_FLAG_CH1COMADD</i>	通道1附加比较中断标志，TIMERx(x=0~4,7,14,19)
<i>TIMER_INT_FLAG_CH2COMADD</i>	通道2附加比较中断标志，TIMERx(x=0~4,7,19)
<i>TIMER_INT_FLAG_CH3COMADD</i>	通道3附加比较中断标志，TIMERx(x=0~4,7,19)
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* clear TIMER0 update interrupt flag */
timer_interrupt_flag_clear(TIMER0, TIMER_INT_FLAG_UP);
```

3.31. 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[3.31.1](#)描述了TMU的寄存器列表，章节[3.31.2](#)对TMU库函数进行说明。

3.31.1. 外设寄存器说明

TMU寄存器列表如下表所示：

表 3-1257. 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

3.31.2. 外设库函数说明

TMU库函数列表如下表所示：

表 3-1258. 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_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_f32_write	写一个浮点格式数据
tmu_two_f32_write	写两个浮点格式数据
tmu_one_q31_read	读一个 q1.31 格式数据
tmu_two_q31_read	读两个 q1.31 格式数据
tmu_two_q15_read	读两个 q1.15 格式数据
tmu_one_f32_read	读一个浮点格式数据
tmu_two_f32_read	读两个浮点格式数据
tmu_flag_get	获取 TMU 标志位
tmu_flag_clear	清除 TMU 标志位
tmu_interrupt_enable	TMU 中断使能
tmu_interrupt_disable	TMU 中断禁能
tmu_interrupt_flag_get	获取 TMU 中断标志位
tmu_interrupt_flag_clear	清除 TMU 中断标志位

结构体 tmu_parameter_struct

表 3-1259. 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_MODALUS,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))
output_floating	输出浮点格式数据使能(TMU_OUTPUT_FLOAT_DISABLE, TMU_OUTPUT_FLOAT_ENABLE)
input_floating	输入浮点格式数据使能(TMU_INPUT_FLOAT_DISABLE, TMU_INPUT_FLOAT_ENABLE)
dma_read	读TMU_ODATA寄存器DMA请求(TMU_READ_DMA_DISABLE, TMU_READ_DMA_ENABLE)

成员名称	功能描述
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 3-1260. 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 3-1261. 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 3-1262. 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.output_floating = TMU_OUTPUT_FLOAT_DISABLE;
```

```
tmu_init_struct.input_floating = TMU_INPUT_FLOAT_DISABLE;
```

```
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_dma_read_enable

函数tmu_dma_read_enable的描述如下表：

表 3-1263. 函数 tmu_dma_read_enable

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

例如：

```
/* enable TMU DMA read request */
```

```
tmu_dma_read_enable();
```

函数 tmu_dma_read_disable

函数tmu_dma_read_disable的描述如下表：

表 3-1264. 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 3-1265. 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 3-1266. 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 3-1267. 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 3-1268. 函数 tmu_two_q31_write

函数名称	tmu_two_q31_write
函数原型	void tmu_two_q31_write(uint32_t data1, uint32_t data2);
功能描述	写两个q1.31格式数据
先决条件	-
被调用函数	-
输入参数{in}	
data1	q1.31格式的输入数据(0x00000000~0xFFFFFFFF)
输入参数{in}	
data2	q1.31格式的第二个输入数据(0x00000000~0xFFFFFFFF)
输出参数{out}	
-	-
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 3-1269. 函数 tmu_two_q15_write

函数名称	tmu_two_q15_write
函数原型	void tmu_two_q15_write(uint16_t data1, uint16_t data2);
功能描述	写两个q1.15格式数据
先决条件	-
被调用函数	-
输入参数{in}	
data1	q1.15格式的第二个输入数据(0x0000~0xFFFF)
输入参数{in}	
data2	q1.15格式的第二个输入数据(0x0000~0xFFFF)
输出参数{out}	
-	-
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_f32_write

函数tmu_one_f32_write的描述如下表：

表 3-1270. 函数 tmu_one_f32_write

函数名称	tmu_one_f32_write
函数原型	void tmu_one_f32_write(float data);
功能描述	写一个浮点格式数据
先决条件	-
被调用函数	-
输入参数{in}	

data	浮点格式的输入数据
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* write one data in floating point format */
```

```
float in = -2000.0f;
```

```
tmu_one_f32_write(in);
```

函数 tmu_two_f32_write

函数tmu_two_f32_write的描述如下表：

表 3-1271. 数 tmu_two_f32_write

函数名称	tmu_two_f32_write
函数原型	void tmu_two_f32_write(float data1, float data2);
功能描述	写两个浮点格式数据
先决条件	-
被调用函数	-
输入参数{in}	
data1	浮点格式的第一个输入数据
输入参数{in}	
data2	浮点格式的第二个输入数据
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* write two data in floating point format */
```

```
float in1 = -2000.0f;
```

```
float in2 = 3000.0f;
```

```
tmu_two_f32_write(in1, in2);
```

函数 tmu_one_q31_read

函数tmu_one_q31_read的描述如下表：

表 3-1272. 函数 tmu_one_q31_read

函数名称	tmu_one_q31_read
-------------	------------------

函数原型	void tmu_one_q31_read(uint32_t* p);
功能描述	读一个q1.31格式数据
先决条件	-
被调用函数	-
输入参数{in}	
p	指向输出数据的指针（q1.31格式）
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* read one data in q1.31 format */
```

```
uint32_t out = 0;
```

```
tmu_one_q31_read (&out);
```

函数 tmu_two_q31_read

函数tmu_two_q31_read的描述如下表：

表 3-1273. 函数 tmu_two_q31_read

函数名称	tmu_two_q31_read
函数原型	void tmu_two_q31_read(uint32_t* p1, uint32_t* p2);
功能描述	读两个q1.31格式数据
先决条件	-
被调用函数	-
输入参数{in}	
p1	指向第一个输出数据的指针（q1.31格式）
输入参数{in}	
p2	指向第二个输出数据的指针（q1.31格式）
输出参数{out}	
-	-
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 3-1274. 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）
输出参数{out}	
-	-
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);
```

函数 tmu_one_f32_read

函数tmu_one_f32_read的描述如下表：

表 3-1275. 函数 tmu_one_f32_read

函数名称	tmu_one_f32_read
函数原型	void tmu_one_f32_read(float* p);
功能描述	读一个浮点格式数据
先决条件	-
被调用函数	-
输入参数{in}	
p	指向输出数据的指针（浮点格式）
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* read one data in floating point format */
```

```
float out = 0.0f;
```

```
tmu_one_f32_read (&out);
```

函数 tmu_two_f32_read

函数tmu_two_f32_read的描述如下表：

表 3-1276. 函数 tmu_two_f32_read

函数名称	tmu_two_f32_read
函数原型	void tmu_two_f32_read(float* p1, float* p2)
功能描述	读两个浮点格式数据
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（浮点格式）
输入参数{in}	
p2	指向第二个输出数据的指针（浮点格式）
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* read two data in floating point format */
```

```
float out1 = 0.0f;
```

```
float out2 = 0.0f;
```

```
tmu_two_f32_read(&out1, &out2);
```

函数 tmu_flag_get

函数tmu_flag_get的描述如下表：

表 3-1277. 函数 tmu_flag_get

函数名称	tmu_flag_get
函数原型	FlagStatus tmu_flag_get(uint32_t flag);
功能描述	获取TMU标志位
先决条件	-
被调用函数	-
输入参数{in}	
flag	TMU标志位
TMU_FLAG_OVRF	TMU溢出错误标志位

TMU_FLAG_END	TMU运算结束标志位
输出参数{out}	
-	-
返回值	
FlagStatus	SET or RESET

例如：

```
/* get TMU flag bit */
```

```
FlagStatus flag_value;
```

```
flag_value = tmu_flag_get(TMU_FLAG_OVRF);
```

函数 tmu_flag_clear

函数tmu_flag_clear的描述如下表：

表 3-1278. 函数 tmu_flag_clear

函数名称	tmu_flag_clear
函数原型	void tmu_flag_clear(uint32_t flag);
功能描述	清除TMU标志位
先决条件	-
被调用函数	-
输入参数{in}	
flag	TMU标志位
TMU_FLAG_OVRF	TMU溢出错误标志位
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* clear TMU flag bit */
```

```
tmu_flag_clear(TMU_FLAG_OVRF);
```

函数 tmu_interrupt_enable

函数tmu_interrupt_enable的描述如下表：

表 3-1279. 函数 tmu_interrupt_enable

函数名称	tmu_interrupt_enable
函数原型	void tmu_interrupt_enable(uint32_t interrupt);
功能描述	TMU中断使能
先决条件	-
被调用函数	-

输入参数{in}	
interrupt	TMU中断
<i>TMU_INT_OVRF</i>	TMU溢出中断
<i>TMU_INT_END</i>	TMU读TMU_ODATA寄存器请求中断
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* enable TMU interrupt */
```

```
tmu_interrupt_enable(TMU_INT_OVRF);
```

函数 tmu_interrupt_disable

函数tmu_interrupt_disable的描述如下表：

表 3-1280. 函数 tmu_interrupt_disable

函数名称	tmu_interrupt_disable
函数原型	void tmu_interrupt_disable(uint32_t interrupt);
功能描述	TMU中断禁能
先决条件	-
被调用函数	-
输入参数{in}	
interrupt	TMU中断
<i>TMU_INT_OVRF</i>	TMU溢出中断
<i>TMU_INT_END</i>	TMU读TMU_ODATA寄存器请求中断
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* disable TMU interrupt */
```

```
tmu_interrupt_disable(TMU_INT_OVRF);
```

函数 tmu_interrupt_flag_get

函数tmu_interrupt_flag_get的描述如下表：

表 3-1281. 函数 tmu_interrupt_flag_get

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

功能描述	获取TMU中断标志位
先决条件	-
被调用函数	-
输入参数{in}	
int_flag	TMU中断标志位
TMU_INT_FLAG_OVR F	TMU溢出中断标志位
TMU_INT_FLAG_END	TMU读TMU_ODATA寄存器请求中断标志位
输出参数{out}	
-	-
返回值	
FlagStatus	SET或RESET

例如:

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

```
FlagStatus flag_value;
```

```
flag_value = tmu_interrupt_flag_get(TMU_INT_FLAG_OVRF);
```

函数 tmu_interrupt_flag_clear

函数tmu_interrupt_flag_clear的描述如下表:

表 3-1282. 函数 tmu_interrupt_flag_clear

函数名称	tmu_interrupt_flag_clear
函数原型	void tmu_interrupt_flag_clear(uint32_t int_flag);
功能描述	清除TMU中断标志位
先决条件	-
被调用函数	-
输入参数{in}	
int_flag	TMU中断标志位
TMU_INT_FLAG_OVR F	TMU溢出中断标志位
输出参数{out}	
-	-
返回值	
-	-

例如:

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

```
tmu_interrupt_flag_clear(TMU_INT_FLAG_OVRF);
```

3.32. TRIGSEL

TRIGSEL 是 MCU 中的触发选择控制器。可通过软件配置的方式，为各种外设选择触发输入信号。章节 [3.32.1](#) 描述了 TRIGSEL 的寄存器列表，章节 [3.32.2](#) 对 TRIGSEL 库函数进行说明。

3.32.1. 外设寄存器说明

TRIGSEL 寄存器列表如下表所示：

表 3-1283. TRIGSEL 寄存器

寄存器名称	寄存器描述
TRIGSEL_EXTOUT_0	EXTOUT触发选择寄存器0
TRIGSEL_EXTOUT_1	EXTOUT触发选择寄存器1
TRIGSEL_EXTOUT_2	EXTOUT触发选择寄存器2
TRIGSEL_EXTOUT_3	EXTOUT触发选择寄存器3
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_ADC3	ADC3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_TIMER19BRKIN	TIMER19_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_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_TIMER19ETI	TIMER19_ETI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

寄存器名称	寄存器描述
TRIGSEL_TIMER7ITI14	TIMER7_ITI14触发选择寄存器
TRIGSEL_TIMER14ITI14	TIMER14_ITI14触发选择寄存器
TRIGSEL_TIMER19ITI14	TIMER19_ITI14触发选择寄存器
TRIGSEL_DAC0	DAC0触发选择寄存器
TRIGSEL_DAC1	DAC1触发选择寄存器
TRIGSEL_DAC2	DAC2触发选择寄存器
TRIGSEL_DAC3	DAC3触发选择寄存器
TRIGSEL_EXTDAC0	DAC0触发选择扩展寄存器
TRIGSEL_EXTDAC1	DAC1触发选择扩展寄存器
TRIGSEL_EXTDAC2	DAC2触发选择扩展寄存器
TRIGSEL_EXTDAC3	DAC3触发选择扩展寄存器
TRIGSEL_CLA_0	CLA触发选择寄存器0
TRIGSEL_CLA_1	CLA触发选择寄存器1
TRIGSEL_CLA_2	CLA触发选择寄存器2
TRIGSEL_CLA_3	CLA触发选择寄存器3
TRIGSEL_CLA_4	CLA触发选择寄存器4

3.32.2. 外设库函数说明

TRIGSEL库函数列表如下表所示：

表 3-1284. 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 3-1285. 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

成员名称	功能描述
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_TIMER0_TRGO0	触发输入源为TIMER0 TRGO0
TRIGSEL_INPUT_TIMER0_TRGO1	触发输入源为TIMER0 TRGO1
TRIGSEL_INPUT_TIMER0_CH0	触发输入源为TIMER0 CH0
TRIGSEL_INPUT_TIMER0_CH1	触发输入源为TIMER0 CH1
TRIGSEL_INPUT_TIMER0_CH2	触发输入源为TIMER0 CH2
TRIGSEL_INPUT_TIMER0_CH3	触发输入源为TIMER0 CH3
TRIGSEL_INPUT_TIMER0_MCH0	触发输入源为TIMER0 MCH0
TRIGSEL_INPUT_TIMER0_MCH1	触发输入源为TIMER0 MCH1
TRIGSEL_INPUT_TIMER0_MCH2	触发输入源为TIMER0 MCH2
TRIGSEL_INPUT_TIMER0_MCH3	触发输入源为TIMER0 MCH3
TRIGSEL_INPUT_TIMER0_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

成员名称	功能描述
TRIGSEL_INPUT_TIMER4_ETI	触发输入源为TIMER4 ETI
TRIGSEL_INPUT_TIMER5_TRGO0	触发输入源为TIMER5 TRGO0
TRIGSEL_INPUT_TIMER6_TRGO0	触发输入源为TIMER6 TRGO0
TRIGSEL_INPUT_TIMER7_TRGO0	触发输入源为TIMER7 TRGO0
TRIGSEL_INPUT_TIMER7_TRGO1	触发输入源为TIMER7 TRGO1
TRIGSEL_INPUT_TIMER7_CH0	触发输入源为TIMER7 CH0
TRIGSEL_INPUT_TIMER7_CH1	触发输入源为TIMER7 CH1
TRIGSEL_INPUT_TIMER7_CH2	触发输入源为TIMER7 CH2
TRIGSEL_INPUT_TIMER7_CH3	触发输入源为TIMER7 CH3
TRIGSEL_INPUT_TIMER7_MCH0	触发输入源为TIMER7 MCH0
TRIGSEL_INPUT_TIMER7_MCH1	触发输入源为TIMER7 MCH1
TRIGSEL_INPUT_TIMER7_MCH2	触发输入源为TIMER7 MCH2
TRIGSEL_INPUT_TIMER7_MCH3	触发输入源为TIMER7 MCH3
TRIGSEL_INPUT_TIMER7_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_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_TIMER19_TRGO0	触发输入源为TIMER19 TRGO0
TRIGSEL_INPUT_TIMER19_TRGO1	触发输入源为TIMER19 TRGO1
TRIGSEL_INPUT_TIMER19_CH0	触发输入源为TIMER19 CH0
TRIGSEL_INPUT_TIMER19_CH1	触发输入源为TIMER19 CH1
TRIGSEL_INPUT_TIMER19_CH2	触发输入源为TIMER19 CH2
TRIGSEL_INPUT_TIMER19_CH3	触发输入源为TIMER19 CH3
TRIGSEL_INPUT_TIMER19_MCH0	触发输入源为TIMER19 MCH0
TRIGSEL_INPUT_TIMER19_MCH1	触发输入源为TIMER19 MCH1
TRIGSEL_INPUT_TIMER19_MCH2	触发输入源为TIMER19 MCH2
TRIGSEL_INPUT_TIMER19_MCH3	触发输入源为TIMER19 MCH3
TRIGSEL_INPUT_TIMER19_ETI	触发输入源为TIMER19 ETI
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_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

成员名称	功能描述
TRIGSEL_INPUT_TIMER14_BRKIN0	触发输入源为TIMER14 BRKIN0
TRIGSEL_INPUT_TIMER15_BRKIN0	触发输入源为TIMER15 BRKIN0
TRIGSEL_INPUT_TIMER16_BRKIN0	触发输入源为TIMER16 BRKIN0
TRIGSEL_INPUT_TIMER19_BRKIN0	触发输入源为TIMER19 BRKIN0
TRIGSEL_INPUT_TIMER19_BRKIN1	触发输入源为TIMER19 BRKIN1
TRIGSEL_INPUT_TIMER19_BRKIN2	触发输入源为TIMER19 BRKIN2
TRIGSEL_INPUT_LPTIMER_OUT	触发输入源为LPTIMER_OUT
TRIGSEL_INPUT_LPTIMER_ETI	触发输入源为LPTIMER_ETI
TRIGSEL_INPUT_HRTIMER_SCOUT	触发输入源为HRTIMER_SCOUT
TRIGSEL_INPUT_HRTIMER_SCIN	触发输入源为HRTIMER_SCIN
TRIGSEL_INPUT_HRTIMER_ADC_TRIG0	触发输入源为HRTIMER_ADC_TRIG0
TRIGSEL_INPUT_HRTIMER_ADC_TRIG1	触发输入源为HRTIMER_ADC_TRIG1
TRIGSEL_INPUT_HRTIMER_ADC_TRIG2	触发输入源为HRTIMER_ADC_TRIG2
TRIGSEL_INPUT_HRTIMER_ADC_TRIG3	触发输入源为HRTIMER_ADC_TRIG3
TRIGSEL_INPUT_HRTIMER_ADC_TRIG4	触发输入源为HRTIMER_ADC_TRIG4
TRIGSEL_INPUT_HRTIMER_ADC_TRIG5	触发输入源为HRTIMER_ADC_TRIG5
TRIGSEL_INPUT_HRTIMER_ADC_TRIG6	触发输入源为HRTIMER_ADC_TRIG6
TRIGSEL_INPUT_HRTIMER_ADC_TRIG7	触发输入源为HRTIMER_ADC_TRIG7
TRIGSEL_INPUT_HRTIMER_ADC_TRIG8	触发输入源为HRTIMER_ADC_TRIG8
TRIGSEL_INPUT_HRTIMER_ADC_TRIG9	触发输入源为HRTIMER_ADC_TRIG9
TRIGSEL_INPUT_HRTIMER_DAC_TRIG0	触发输入源为HRTIMER_DAC_TRIG0
TRIGSEL_INPUT_HRTIMER_DAC_TRIG1	触发输入源为HRTIMER_DAC_TRIG1
TRIGSEL_INPUT_HRTIMER_DAC_TRIG2	触发输入源为HRTIMER_DAC_TRIG2
TRIGSEL_INPUT_HRTIMER_ST0_TRIG1	触发输入源为HRTIMER_ST0_TRIG1
TRIGSEL_INPUT_HRTIMER_ST1_TRIG1	触发输入源为HRTIMER_ST1_TRIG1
TRIGSEL_INPUT_HRTIMER_ST2_TRIG1	触发输入源为HRTIMER_ST2_TRIG1
TRIGSEL_INPUT_HRTIMER_ST3_TRIG1	触发输入源为HRTIMER_ST3_TRIG1
TRIGSEL_INPUT_HRTIMER_ST4_TRIG1	触发输入源为HRTIMER_ST4_TRIG1
TRIGSEL_INPUT_HRTIMER_ST5_TRIG1	触发输入源为HRTIMER_ST5_TRIG1
TRIGSEL_INPUT_HRTIMER_ST6_TRIG1	触发输入源为HRTIMER_ST6_TRIG1
TRIGSEL_INPUT_HRTIMER_ST7_TRIG1	触发输入源为HRTIMER_ST7_TRIG1
TRIGSEL_INPUT_HRTIMER_SYSFLT	触发输入源为HRTIMER_SYSFLT
TRIGSEL_INPUT_ADC0_WD0_OUT	触发输入源为ADC0_WD0_OUT
TRIGSEL_INPUT_ADC0_WD1_OUT	触发输入源为ADC0_WD1_OUT
TRIGSEL_INPUT_ADC0_WD2_OUT	触发输入源为ADC0_WD2_OUT
TRIGSEL_INPUT_ADC1_WD0_OUT	触发输入源为ADC1_WD0_OUT
TRIGSEL_INPUT_ADC1_WD1_OUT	触发输入源为ADC1_WD1_OUT
TRIGSEL_INPUT_ADC1_WD2_OUT	触发输入源为ADC1_WD2_OUT
TRIGSEL_INPUT_ADC2_WD0_OUT	触发输入源为ADC2_WD0_OUT
TRIGSEL_INPUT_ADC2_WD1_OUT	触发输入源为ADC2_WD1_OUT
TRIGSEL_INPUT_ADC2_WD2_OUT	触发输入源为ADC2_WD2_OUT

成员名称	功能描述
TRIGSEL_INPUT_HXTAL_DIV32_TRIG	触发输入源为HXTAL_DIV32_TRIG
TRIGSEL_INPUT_IRC32K_TRIG	触发输入源为IRC32K_TRIG
TRIGSEL_INPUT_LXTAL_TRIG	触发输入源为LXTAL_TRIG
TRIGSEL_INPUT_CKOUT_TRIG	触发输入源为CKOUT_TRIG
TRIGSEL_INPUT_EXTI2_TRIG	触发输入源为EXTI2_TRIG
TRIGSEL_INPUT_EXTI3_TRIG	触发输入源为EXTI3_TRIG
TRIGSEL_INPUT_EXTI9_TRIG	触发输入源为EXTI9_TRIG
TRIGSEL_INPUT_EXTI10_TRIG	触发输入源为EXTI10_TRIG
TRIGSEL_INPUT_EXTI11_TRIG	触发输入源为EXTI11_TRIG
TRIGSEL_INPUT_EXTI15_TRIG	触发输入源为EXTI15_TRIG
TRIGSEL_INPUT_RTC_WAKEUP	触发输入源为RTC_WAKEUP
TRIGSEL_INPUT_RTC_ALARM0	触发输入源为RTC_ALARM0
TRIGSEL_INPUT_RTC_ALARM1	触发输入源为RTC_ALARM1
TRIGSEL_INPUT_RTC_TAMP0	触发输入源为RTC_TAMP0
TRIGSEL_INPUT_RTC_TAMP1	触发输入源为RTC_TAMP1
TRIGSEL_INPUT_RTC_TAMP2	触发输入源为RTC_TAMP2
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_CMP2_OUT	触发输入源为CMP2_OUT
TRIGSEL_INPUT_CMP3_OUT	触发输入源为CMP3_OUT
TRIGSEL_INPUT_CMP4_OUT	触发输入源为CMP4_OUT
TRIGSEL_INPUT_CMP5_OUT	触发输入源为CMP5_OUT
TRIGSEL_INPUT_CMP6_OUT	触发输入源为CMP6_OUT
TRIGSEL_INPUT_CMP7_OUT	触发输入源为CMP7_OUT
TRIGSEL_INPUT_CLA_OUT0	触发输入源为CLA_OUT0
TRIGSEL_INPUT_CLA_OUT1	触发输入源为CLA_OUT1
TRIGSEL_INPUT_CLA_OUT2	触发输入源为CLA_OUT2
TRIGSEL_INPUT_CLA_OUT3	触发输入源为CLA_OUT3
TRIGSEL_INPUT_HRTIMER_ST0_TRIG0	触发输入源为HRTIMER_ST0_TRIG0
TRIGSEL_INPUT_HRTIMER_ST1_TRIG0	触发输入源为HRTIMER_ST1_TRIG0
TRIGSEL_INPUT_HRTIMER_ST2_TRIG0	触发输入源为HRTIMER_ST2_TRIG0
TRIGSEL_INPUT_HRTIMER_ST3_TRIG0	触发输入源为HRTIMER_ST3_TRIG0
TRIGSEL_INPUT_HRTIMER_ST4_TRIG0	触发输入源为HRTIMER_ST4_TRIG0
TRIGSEL_INPUT_HRTIMER_ST5_TRIG0	触发输入源为HRTIMER_ST5_TRIG0
TRIGSEL_INPUT_HRTIMER_ST6_TRIG0	触发输入源为HRTIMER_ST6_TRIG0
TRIGSEL_INPUT_HRTIMER_ST7_TRIG0	触发输入源为HRTIMER_ST7_TRIG0
TRIGSEL_INPUT_HRTIMER_ST0_CH0	触发输入源为HRTIMER_ST0_CH0
TRIGSEL_INPUT_HRTIMER_ST0_CH1	触发输入源为HRTIMER_ST0_CH1
TRIGSEL_INPUT_HRTIMER_ST1_CH0	触发输入源为HRTIMER_ST1_CH0
TRIGSEL_INPUT_HRTIMER_ST1_CH1	触发输入源为HRTIMER_ST1_CH1
TRIGSEL_INPUT_HRTIMER_ST2_CH0	触发输入源为HRTIMER_ST2_CH0

成员名称	功能描述
TRIGSEL_INPUT_HRTIMER_ST2_CH1	触发输入源为HRTIMER_ST2_CH1
TRIGSEL_INPUT_HRTIMER_ST3_CH0	触发输入源为HRTIMER_ST3_CH0
TRIGSEL_INPUT_HRTIMER_ST3_CH1	触发输入源为HRTIMER_ST3_CH1
TRIGSEL_INPUT_HRTIMER_ST4_CH0	触发输入源为HRTIMER_ST4_CH0
TRIGSEL_INPUT_HRTIMER_ST4_CH1	触发输入源为HRTIMER_ST4_CH1
TRIGSEL_INPUT_HRTIMER_ST5_CH0	触发输入源为HRTIMER_ST5_CH0
TRIGSEL_INPUT_HRTIMER_ST5_CH1	触发输入源为HRTIMER_ST5_CH1
TRIGSEL_INPUT_HRTIMER_ST6_CH0	触发输入源为HRTIMER_ST6_CH0
TRIGSEL_INPUT_HRTIMER_ST6_CH1	触发输入源为HRTIMER_ST6_CH1
TRIGSEL_INPUT_HRTIMER_ST7_CH0	触发输入源为HRTIMER_ST7_CH0
TRIGSEL_INPUT_HRTIMER_ST7_CH1	触发输入源为HRTIMER_ST7_CH1
TRIGSEL_INPUT_ADC0_CONV	触发输入源为ADC0_CONV
TRIGSEL_INPUT_ADC1_CONV	触发输入源为ADC1_CONV
TRIGSEL_INPUT_ADC2_CONV	触发输入源为ADC2_CONV
TRIGSEL_INPUT_ADC3_CONV	触发输入源为ADC3_CONV
TRIGSEL_INPUT_LCKM_OUT	触发输入源为LCKM_OUT

枚举类型 `trigsel_periph_enum`

表 3-1286. 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_ROUTRG	输出到目标外设ADC0_ROUTRG
TRIGSEL_OUTPUT_ADC0_INSTRG	输出到目标外设ADC0_INSTRG
TRIGSEL_OUTPUT_ADC1_ROUTRG	输出到目标外设ADC1_ROUTRG
TRIGSEL_OUTPUT_ADC1_INSTRG	输出到目标外设ADC1_INSTRG
TRIGSEL_OUTPUT_ADC2_ROUTRG	输出到目标外设ADC2_ROUTRG
TRIGSEL_OUTPUT_ADC2_INSTRG	输出到目标外设ADC2_INSTRG
TRIGSEL_OUTPUT_ADC3_ROUTRG	输出到目标外设ADC3_ROUTRG
TRIGSEL_OUTPUT_ADC3_INSTRG	输出到目标外设ADC3_INSTRG
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

成员名称	功能描述
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_TIMER19_BRKIN0	输出到目标外设TIMER19_BRKIN0
TRIGSEL_OUTPUT_TIMER19_BRKIN0	输出到目标外设TIMER19_BRKIN0
TRIGSEL_OUTPUT_TIMER19_BRKIN0	输出到目标外设TIMER19_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_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_TIMER19_ETI	输出到目标外设TIMER19_ETI
TRIGSEL_OUTPUT_HPDI_ITRG	输出到目标外设HPDI_ITRG
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_TIMER19_ITI14	输出到目标外设TIMER19_ITI14
TRIGSEL_OUTPUT_DAC0_EXTRIG0	输出到目标外设DAC0_EXTRIG0
TRIGSEL_OUTPUT_DAC0_EXTRIG1	输出到目标外设DAC0_EXTRIG1
TRIGSEL_OUTPUT_DAC1_EXTRIG0	输出到目标外设DAC1_EXTRIG0
TRIGSEL_OUTPUT_DAC1_EXTRIG1	输出到目标外设DAC1_EXTRIG1
TRIGSEL_OUTPUT_DAC2_EXTRIG0	输出到目标外设DAC2_EXTRIG0
TRIGSEL_OUTPUT_DAC2_EXTRIG1	输出到目标外设DAC2_EXTRIG1
TRIGSEL_OUTPUT_DAC3_EXTRIG0	输出到目标外设DAC3_EXTRIG0
TRIGSEL_OUTPUT_DAC3_EXTRIG1	输出到目标外设DAC3_EXTRIG1
TRIGSEL_OUTPUT_DAC0_ST_EXTRIG0	输出到目标外设DAC0_ST_EXTRIG0
TRIGSEL_OUTPUT_DAC0_ST_EXTRIG1	输出到目标外设DAC0_ST_EXTRIG1
TRIGSEL_OUTPUT_DAC1_ST_EXTRIG0	输出到目标外设DAC1_ST_EXTRIG0
TRIGSEL_OUTPUT_DAC1_ST_EXTRIG1	输出到目标外设DAC1_ST_EXTRIG1
TRIGSEL_OUTPUT_DAC2_ST_EXTRIG0	输出到目标外设DAC2_ST_EXTRIG0
TRIGSEL_OUTPUT_DAC2_ST_EXTRIG1	输出到目标外设DAC2_ST_EXTRIG1

成员名称	功能描述
TRIGSEL_OUTPUT_DAC3_ST_EXTRIG0	输出到目标外设DAC3_ST_EXTRIG0
TRIGSEL_OUTPUT_DAC3_ST_EXTRIG1	输出到目标外设DAC3_ST_EXTRIG1
TRIGSEL_OUTPUT_CLA_IN0	输出到目标外设CLA_IN0
TRIGSEL_OUTPUT_CLA_IN1	输出到目标外设CLA_IN1
TRIGSEL_OUTPUT_CLA_IN2	输出到目标外设CLA_IN2
TRIGSEL_OUTPUT_CLA_IN3	输出到目标外设CLA_IN3
TRIGSEL_OUTPUT_CLA_IN4	输出到目标外设CLA_IN4
TRIGSEL_OUTPUT_CLA_IN5	输出到目标外设CLA_IN5
TRIGSEL_OUTPUT_CLA_IN6	输出到目标外设CLA_IN6
TRIGSEL_OUTPUT_CLA_IN7	输出到目标外设CLA_IN7
TRIGSEL_OUTPUT_CLA_IN8	输出到目标外设CLA_IN8
TRIGSEL_OUTPUT_CLA_IN9	输出到目标外设CLA_IN9
TRIGSEL_OUTPUT_CLA_IN10	输出到目标外设CLA_IN10
TRIGSEL_OUTPUT_CLA_IN11	输出到目标外设CLA_IN11

函数 trigsels_deinit

函数trigsels_init描述见下表：

表 3-1287. 函数 trigsels_init

函数名称	trigsels_deinit
函数原型	void trigsels_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 */
```

```
trigsels_deinit();
```

函数 trigsels_init

函数trigsels_init描述见下表：

表 3-1288. 函数 trigsels_init

函数名称	trigsels_init
函数原型	void trigsels_init(trigsels_periph_enum target_periph, trigsels_source_enum

	trigger_source);
功能描述	为外设选择触发输入源
先决条件	-
被调用函数	-
输入参数{in}	
target_periph	目标外设, 参考 表 3-1286. 枚举类型 trigsels_periph_enum
输入参数{in}	
trigger_source	触发源, 参考 表 3-1285. 枚举类型 trigsels_source_enum
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* select TIMER0_CH2 to trigger ADC0 */
```

```
trigsels_init(TRIGSELS_OUTPUT_ADC0_ROUTRG, TRIGSELS_INPUT_TIMER0_CH2);
```

函数 trigsels_trigger_source_get

函数trigsels_trigger_source_get描述见下表:

表 3-1289. 函数 trigsels_trigger_source_get

函数名称	trigsels_trigger_source_get
函数原型	trigsels_source_enum trigsels_trigger_source_get(trigsels_periph_enum target_periph);
功能描述	获取外设的触发输入源
先决条件	-
被调用函数	-
输入参数{in}	
target_periph	目标外设, 参考 表 3-1286. 枚举类型 trigsels_periph_enum
输出参数{out}	
-	-
返回值	
trigger_source	触发源, 参考 表 3-1285. 枚举类型 trigsels_source_enum

例如:

```
/* get the trigger input signal for ADC0 */
```

```
trigsels_source_enum input_signal;
```

```
input_signal = trigsels_trigger_source_get(TRIGSELS_OUTPUT_ADC0_ROUTRG);
```

函数 **trigsel_register_lock_set**

函数trigsel_register_lock_set描述见下表：

表 3-1290. 函数 **trigsel_trigger_source_set**

函数名称	trigsel_register_lock_set
函数原型	void trigsel_register_lock_set(trigsel_periph_enum target_periph);
功能描述	锁定触发寄存器
先决条件	-
被调用函数	-
输入参数{in}	
target_periph	目标外设，参考 表 3-1286. 枚举类型 trigsel_periph_enum
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* lock the trigger register for ADC0 */
```

```
trigsel_register_lock_set(TRIGSEL_OUTPUT_ADC0_ROUTRG);
```

函数 **trigsel_register_lock_get**

函数trigsel_register_lock_get描述见下表：

表 3-1291. 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 3-1286. 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_ROUTRG);
```

3.33. 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[3.33.1](#)，TRNG固件库函数介绍在章节[3.33.2](#)。

3.33.1. 外设寄存器说明

TRNG寄存器列表如下表所示：

表 3-1292. 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
TRNG_HTCFG	健康测试配置寄存器

3.33.2. 外设库函数说明

TRNG库函数列表如下表所示：

表 3-1293. 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_nist_seed_config	选择NIST工作模式下的种子模式
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

库函数名称	库函数描述
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 3-1294. 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 3-1295. 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 3-1296. 枚举 trng_modsel_enum

成员名称	功能描述
TRNG_MODSEL_LFSR	工作在LFSR模式
TRNG_MODSEL_NIST	工作在NIST模式

枚举 trng_nist_seed_enum

表 3-1297. 枚举 trng_nist_seed_enum

成员名称	功能描述
TRNG_NIST_SEED_ANALOG	在NIST模式下选择模拟源作为种子
TRNG_NIST_SEED_LFSR	在NIST模式下选择LFSR源作为种子

枚举 trng_flag_enum

表 3-1298. 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 3-1299. 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 3-1300. 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 3-1301. 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 3-1302. 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 3-1303. 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 3-1304. 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3-1296. 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_nist_seed_config

函数trng_nist_seed_config描述见下表:

表 3-1305. 函数 trng_nist_seed_config

函数名称	trng_nist_seed_config
函数原形	void trng_nist_seed_config(trng_nist_seed_enum seed_select);
功能描述	选择NIST工作模式下的种子模式
先决条件	trng_mode_config(TRNG_MODSEL_NIST)
被调用函数	-
输入参数{in}	
seed_select	TRNG NIST种子选择, 参考 表3-1297. 枚举trng_nist_seed_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_nist_seed_config(TRNG_NIST_SEED_ANALOG);  
  
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 3-1306. 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 3-1307. 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 3-1308. 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 3-1309. 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 3-1310. 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3-1294. 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 3-1311. 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3-1295. 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 3-1312. 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 3-1313. 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 3-1314. 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 3-1315. 函数 trng_hash_init_disable

函数名称	trng_hash_init_disable
函数原形	void trng_hash_init_disable(void);
功能描述	当训练单元使能时失能哈希算法初始化
先决条件	trng_conditioning_reset_enable / trng_conditioning_enable
被调用函数	-
输入参数{in}	
-	-
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* disable the TRNG interface */
```

```
trng_deinit();
```

```
trng_conditioning_reset_enable();
```

```
trng_conditioning_enable();

trng_hash_init_disable();

trng_enable();

trng_conditioning_reset_disable();
```

函数 trng_powermode_config

函数trng_powermode_config描述见下表：

表 3-1316. 函数 trng_powermode_config

函数名称	trng_powermode_config
函数原形	void trng_powermode_config(uint32_t powermode);
功能描述	配置功耗模式
先决条件	trng_conditioning_reset_enable
被调用函数	-
输入参数{in}	
powermode	功耗模式选项
TRNG_NR_ULATRLLOW	极低功耗模式
TRNG_NR_LOW	低功耗模式
TRNG_NR_MEDIUM	中等功耗模式
TRNG_NR_HIGH	高功耗模式
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* configure the TRNG analog power mode as high */

trng_deinit();

trng_conditioning_reset_enable();

trng_powermode_config(TRNG_NR_HIGH);

trng_enable();

trng_conditioning_reset_disable();
```

函数 trng_clockdiv_config

函数trng_clockdiv_config描述见下表：

表 3-1317. 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 3-1318. 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 3-1319. 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 3-1320. 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 3-1321. 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 3-1322. 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 3-1323. 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 3-1324. 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 3-1325. 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3-1298. 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 3-1326. 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 3-1327. 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 3-1328. 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3-1299. 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 3-1329. 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3-1299. 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);
```

3.34. 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[3.34.1](#)描述了USART的寄存器列表，章节[3.34.2](#)对USART库函数进行说明。

3.34.1. 外设寄存器说明

USART寄存器列表如下表所示：

表 3-1330. 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

3.34.2. 外设库函数说明

USART库函数列表如下表所示：

表 3-1331. 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 3-1332. 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_TFNF	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_RFNE	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_TFTIF	发送FIFO阈值中断标志
USART_FLAG_TFEIF	发送FIFO空中断标志
USART_FLAG_RFTIF	接收FIFO阈值中断标志
USART_FLAG_RFFIF	接收FIFO满中断标志
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 3-1333. 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0匹配中断标志
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 3-1334. 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

成员名称	功能描述
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
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 3-1335. 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 3-1336. 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}	
<code>usart_periph</code>	外设USARTx
<code>USARTx</code>	x = 0, 1, 2
<code>UARTx</code>	x = 3, 4
输出参数{out}	
-	-
返回值	

-	-
---	---

例如:

```
/* reset USART0 */
```

```
usart_deinit(USART0);
```

函数 usart_baudrate_set

函数usart_baudrate_set描述见下表:

表 3-1337. 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
UARTx	x = 3, 4
输入参数{in}	
baudval	波特率值
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* configure USART0 baud rate value */
```

```
usart_baudrate_set(USART0, 115200);
```

函数 usart_parity_config

函数usart_parity_config描述见下表:

表 3-1338. 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

<i>USARTx</i>	<i>x</i> = 0, 1, 2
<i>UARTx</i>	<i>x</i> = 3, 4
输入参数{in}	
paritycfg	配置USART奇偶校验
<i>USART_PM_NONE</i>	无校验
<i>USART_PM_ODD</i>	奇校验
<i>USART_PM_EVEN</i>	偶校验
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* configure 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 3-1339. 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
<i>USARTx</i>	<i>x</i> = 0, 1, 2
<i>UARTx</i>	<i>x</i> = 3, 4
输入参数{in}	
wlen	配置USART字长
<i>USART_WL_8BIT</i>	8位
<i>USART_WL_9BIT</i>	9位
<i>USART_WL_7BIT</i>	7位
<i>USART_WL_10BIT</i>	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 3-1340. 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
UARTx	x = 3, 4
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 3-1341. 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
<i>USARTx</i>	x = 0, 1, 2
<i>UARTx</i>	x = 3, 4
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* enable USART0 */
```

```
usart_enable(USART0);
```

函数 usart_disable

函数usart_disable描述见下表:

表 3-1342. 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
<i>USARTx</i>	x = 0, 1, 2
<i>UARTx</i>	x = 3, 4
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* disable USART0 */
```

```
usart_disable(USART0);
```

函数 usart_transmit_config

函数usart_transmit_config描述见下表:

表 3-1343. 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
USARTx	x = 0, 1, 2
UARTx	x = 3, 4
输入参数{in}	
txconfig	使能/禁能USART发送器
USART_TRANSMIT_ENABLE	使能USART发送
USART_TRANSMIT_DISABLE	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 3-1344. 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
USARTx	x = 0, 1, 2
UARTx	x = 3, 4
输入参数{in}	
rxconfig	使能/禁能USART接收器
USART_RECEIVE_ENABLE	使能USART接收
USART_RECEIVE_DISABLE	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 3-1345. 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
UARTx	x = 3, 4
输入参数{in}	
msbf	数据传输时低位在前/高位在前
USART_MSBF_LSB	数据传输时低位在前
USART_MSBF_MSB	数据传输时高位在前
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* configure LSB of data first */
```

```
usart_data_first_config(USART0, USART_MSBF_LSB);
```

函数 usart_invert_config

函数usart_invert_config描述见下表：

表 3-1346. 函数 usart_invert_config

函数名称	usart_invert_config
函数原型	void usart_invert_config(uint32_t usart_periph, usart_invert_enum invertpara);

功能描述	配置USART反转功能
先决条件	-
被调用函数	-
输入参数{in}	
usart_periph	外设USARTx
USARTx	x = 0, 1, 2
UARTx	x = 3, 4
输入参数{in}	
invertpara	参考 表3-1335. 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 3-1347. 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
USARTx	x = 0, 1, 2
UARTx	x = 3, 4
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* enable USART0 overrun */
```

```
usart_overrun_enable(USART0);
```


函数 usart_oversample_disable

函数usart_oversample_disable描述见下表:

表 3-1348. 函数 usart_oversample_disable

函数名称	usart_oversample_disable
函数原型	void usart_oversample_disable(uint32_t usart_periph);
功能描述	禁用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
UARTx	x = 3, 4
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 3-1349. 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
UARTx	x = 3, 4
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 3-1350. 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
UARTx	x = 3, 4
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 3-1351. 函数 usart_receiver_timeout_enable

函数名称	usart_receiver_timeout_enable
函数原型	void usart_receiver_timeout_enable(uint32_t usart_periph);

功能描述	使能USART接收超时
先决条件	-
被调用函数	-
输入参数{in}	
usart_periph	外设USARTx
USARTx	x = 0, 1, 2
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* enable 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 3-1352. 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	禁用USART接收超时
先决条件	-
被调用函数	-
输入参数{in}	
usart_periph	外设USARTx
USARTx	x = 0, 1, 2
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* disable 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 3-1353. 函数 usart_receiver_timeout_threshold_config

函数名称	usart_receiver_timeout_threshold_config
------	---

函数原型	void usart_receiver_timeout_threshold_config(uint32_t usart_periph, uint32_t rtimeout);
功能描述	设置USART接收超时阈值
先决条件	-
被调用函数	-
输入参数{in}	
usart_periph	外设USARTx
USARTx	x = 0, 1, 2
输入参数{in}	
rtimeout	超时时间（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 3-1354. 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
USARTx	x = 0, 1, 2
UARTx	x = 3, 4
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 3-1355. 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
UARTx	x = 3, 4
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 3-1356. 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
UARTx	x = 3, 4
输入参数{in}	
cmdtype	请求类型
USART_CMD_SBK	发送断开帧请求

<i>CMD</i>	
<i>USART_CMD_MM</i> <i>CMD</i>	静模式请求
<i>USART_CMD_RXF</i> <i>CMD</i>	接收数据清空请求
<i>USART_CMD_TXF</i> <i>CMD</i>	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 3-1357. 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
<i>USARTx</i>	x = 0, 1, 2
<i>UARTx</i>	x = 3, 4
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 3-1358. 函数 `usart_address_0_match_mode_disable`

函数名称	<code>usart_address_0_match_mode_disable</code>
函数原型	<code>void usart_address_0_match_mode_disable(uint32_t usart_periph);</code>
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}	
<code>usart_periph</code>	外设USARTx
<code>USARTx</code>	x = 0, 1, 2
<code>UARTx</code>	x = 3, 4
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 3-1359. 函数 `usart_address_1_match_mode_enable`

函数名称	<code>usart_address_1_match_mode_enable</code>
函数原型	<code>void usart_address_1_match_mode_enable(uint32_t usart_periph);</code>
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}	
<code>usart_periph</code>	外设USARTx
<code>USARTx</code>	x = 0, 1, 2
<code>UARTx</code>	x = 3, 4
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 3-1360. 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
UARTx	x = 3, 4
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 3-1361. 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
UARTx	x = 3, 4
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 3-1362. 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
UARTx	x = 3, 4
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 3-1363. 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

<i>UARTx</i>	<i>x</i> = 3, 4
输入参数{in}	
addmod	地址检测模式
<i>USART_ADDDM_4BIT</i>	4位地址检测
<i>USART_ADDDM_FULLBIT</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_ADDDM_4BIT);
```

函数 **usart_address_1_detection_mode_config**

函数usart_address_1_detection_mode_config描述见下表:

表 3-1364. 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>	<i>x</i> = 0, 1, 2
<i>UARTx</i>	<i>x</i> = 3, 4
输入参数{in}	
addmod	地址检测模式
<i>USART_ADDDM_4BIT</i>	4位地址检测
<i>USART_ADDDM_FULLBIT</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_ADDDM_4BIT);
```

函数 usart_mute_mode_enable

函数usart_mute_mode_enable描述见下表：

表 3-1365. 函数 usart_mute_mode_enable

函数名称	usart_mute_mode_enable
函数原型	void usart_mute_mode_enable(uint32_t usart_periph);
功能描述	使能USART静默模式
先决条件	-
被调用函数	-
输入参数{in}	
usart_periph	外设USARTx
USARTx	x = 0, 1, 2
UARTx	x = 3, 4
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* enable USART0 receiver in mute mode */
```

```
usart_mute_mode_enable(USART0);
```

函数 usart_mute_mode_disable

函数usart_mute_mode_disable描述见下表：

表 3-1366. 函数 usart_mute_mode_disable

函数名称	usart_mute_mode_disable
函数原型	void usart_mute_mode_disable(uint32_t usart_periph);
功能描述	禁能USART静默模式
先决条件	-
被调用函数	-
输入参数{in}	
usart_periph	外设USARTx
USARTx	x = 0, 1, 2
UARTx	x = 3, 4
输出参数{out}	
-	-
返回值	

-	-
---	---

例如:

```
/* disable USART0 receiver in mute mode */
```

```
usart_mute_mode_disable(USART0);
```

函数 usart_mute_mode_wakeup_config

函数usart_mute_mode_wakeup_config描述见下表:

表 3-1367. 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
UARTx	x = 3, 4
输入参数{in}	
wmethod	两种方法用于进入或退出静默模式
USART_WM_IDLE	空闲线唤醒
USART_WM_ADDR	地址匹配唤醒
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 3-1368. 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
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 3-1369. 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
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 3-1370. 函数 usart_lin_break_dection_length_config

函数名称	usart_lin_break_dection_length_config
函数原型	void usart_lin_break_dection_length_config(uint32_t usart_periph, uint32_t lblen);

功能描述	配置USART LIN模式中断帧长度
先决条件	-
被调用函数	-
输入参数{in}	
usart_periph	外设USARTx
USARTx	x = 0, 1, 2
输入参数{in}	
lblen	LIN模式中断帧长度
USART_LBLEN_10 B	断开帧长度为10位
USART_LBLEN_11 B	断开帧长度为11位
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* configure LIN break frame length */
```

```
usart_lin_break_dection_length_config(USART0, USART_LBLEN_10B);
```

函数 usart_halfduplex_enable

函数usart_halfduplex_enable描述见下表：

表 3-1371. 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
USARTx	x = 0, 1, 2
UARTx	x = 3, 4
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* enable USART0 half duplex mode */
```

```
usart_halfduplex_enable(USART0);
```

函数 `usart_halfduplex_disable`

函数`usart_halfduplex_disable`描述见下表：

表 3-1372. 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
UARTx	x = 3, 4
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* disable USART0 half duplex mode */
```

```
usart_halfduplex_disable(USART0);
```

函数 `usart_clock_enable`

函数`usart_clock_enable`描述见下表：

表 3-1373. 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
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 3-1374. 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
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 3-1375. 函数 usart_synchronous_clock_config

函数名称	usart_synchronous_clock_config
函数原型	void usart_synchronous_clock_config(uint32_t usart_periph, uint32_t clen, uint32_t cph, uint32_t cpl);
功能描述	配置USART同步通讯模式参数
先决条件	-
被调用函数	-
输入参数{in}	
usart_periph	外设USARTx
USARTx	x = 0, 1, 2
输入参数{in}	
clen	CK信号长度
USART_CLEN_NO NE	8位数据帧中有7个CK脉冲，9位数据帧中有8个CK脉冲

USART_CLEN_EN	8位数据帧中有8个CK脉冲，9位数据帧中有9个CK脉冲
输入参数{in}	
cph	时钟相位
USART_CPH_1CK	在首个时钟边沿采样第一个数据
USART_CPH_2CK	在第二个时钟边沿采样第一个数据
输入参数{in}	
cpl	时钟极性
USART_CPL_LOW	CK引脚不对外发送时保持为低电平
USART_CPL_HIGH	CK引脚不对外发送时保持为高电平
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* configure USART0 synchronous mode parameters */
```

```
usart_synchronous_clock_config(USART0, USART_CLEN_EN, USART_CPH_2CK,
USART_CPL_HIGH);
```

函数 usart_guard_time_config

函数usart_guard_time_config描述见下表：

表 3-1376. 函数 usart_guard_time_config

函数名称	usart_guard_time_config
函数原型	void usart_guard_time_config(uint32_t usart_periph,uint32_t guat);
功能描述	在USART智能卡模式下配置保护时间值
先决条件	-
被调用函数	-
输入参数{in}	
usart_periph	外设USARTx
USARTx	x = 0, 1, 2
输入参数{in}	
guat	保护时间值（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 3-1377. 函数 usart_smartcard_mode_enable

函数名称	usart_smartcard_mode_enable
函数原型	void usart_smartcard_mode_enable(uint32_t usart_periph);
功能描述	使能USART智能卡模式
先决条件	-
被调用函数	-
输入参数{in}	
usart_periph	外设USARTx
USARTx	x = 0, 1, 2
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* 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 3-1378. 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
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 3-1379. 函数 usart_smartcard_mode_nack_enable

函数名称	usart_smartcard_mode_nack_enable
函数原型	void usart_smartcard_mode_nack_enable(uint32_t usart_periph);
功能描述	在USART智能卡模式下使能NACK
先决条件	-
被调用函数	-
输入参数{in}	
usart_periph	外设USARTx
USARTx	x = 0, 1, 2
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* enable USART0 NACK in smartcard mode */
usart_smartcard_mode_nack_enable(USART0);
```

函数 usart_smartcard_mode_nack_disable

函数usart_smartcard_mode_nack_disable描述见下表：

表 3-1380. 函数 usart_smartcard_mode_nack_disable

函数名称	usart_smartcard_mode_nack_disable
函数原型	void usart_smartcard_mode_nack_disable(uint32_t usart_periph);
功能描述	在USART智能卡模式下禁能NACK
先决条件	-
被调用函数	-
输入参数{in}	
usart_periph	外设USARTx
USARTx	x = 0, 1, 2
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 3-1381. 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
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 3-1382. 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
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 3-1383. 函数 usart_smartcard_autoretry_config

函数名称	usart_smartcard_autoretry_config
函数原型	void usart_smartcard_autoretry_config(uint32_t usart_periph, uint32_t scrtnum);
功能描述	配置智能卡自动重试次数
先决条件	-
被调用函数	-
输入参数{in}	
usart_periph	外设USARTx
USARTx	x = 0, 1, 2
输入参数{in}	
scrtnum	智能卡自动重试次数（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 3-1384. 函数 usart_block_length_config

函数名称	usart_block_length_config
函数原型	void usart_block_length_config(uint32_t usart_periph, uint32_t bl);
功能描述	配置智能卡T=1的接收时块的长度
先决条件	-
被调用函数	-
输入参数{in}	
usart_periph	外设USARTx
USARTx	x = 0, 1, 2
输入参数{in}	
bl	块长度（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 3-1385. 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
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 3-1386. 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
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 3-1387. 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
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 3-1388. 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

<i>USARTx</i>	<i>x</i> = 0, 1, 2
输入参数{in}	
irlp	IrDA低功耗模式或正常模式
<i>USART_IRLP_LOW</i>	低功耗模式
<i>USART_IRLP_NOR</i> <i>MAL</i>	正常模式
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* configure USART0 IrDA low-power */
```

```
usart_irda_lowpower_config(USART0, USART_IRLP_LOW);
```

函数 usart_hardware_flow_rts_config

函数usart_hardware_flow_rts_config描述见下表:

表 3-1389. 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
<i>USARTx</i>	<i>x</i> = 0, 1, 2
<i>UARTx</i>	<i>x</i> = 3, 4
输入参数{in}	
rtsconfig	使能/禁能RTS
<i>USART_RTS_ENA</i> <i>BLE</i>	使能RTS
<i>USART_RTS_DISA</i> <i>BLE</i>	禁能RTS
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* configure USART0 hardware flow control RTS */
```



```
usart_hardware_flow_rts_config(USART0, USART_RTS_ENABLE);
```

函数 usart_hardware_flow_cts_config

函数usart_hardware_flow_cts_config描述见下表：

表 3-1390. 函数 usart_hardware_flow_cts_config

函数名称	usart_hardware_flow_cts_config
函数原型	void usart_hardware_flow_cts_config(uint32_t usart_periph, uint32_t ctsconfig);
功能描述	配置USART CTS硬件控制流
先决条件	-
被调用函数	-
输入参数{in}	
usart_periph	外设USARTx
USARTx	x = 0, 1, 2
UARTx	x = 3, 4
输入参数{in}	
ctsconfig	使能/禁能CTS
USART_CTS_ENA BLE	使能CTS
USART_CTS_DISA BLE	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 3-1391. 函数 usart_hardware_flow_coherence_config

函数名称	usart_hardware_flow_coherence_config
函数原型	void usart_hardware_flow_coherence_config(uint32_t usart_periph, uint32_t hcm);
功能描述	配置硬件流控兼容模式
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
<i>UARTx</i>	x = 3, 4
输入参数{in}	
hcm	硬件流控制兼容模式
<i>USART_HCM_NONE</i>	nRTS信号与USART_STAT0寄存器中RBNE位相同
<i>USART_HCM_EN</i>	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 3-1392. 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
<i>USARTx</i>	x = 0, 1, 2
<i>UARTx</i>	x = 3, 4
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 3-1393. 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
USARTx	x = 0, 1, 2
UARTx	x = 3, 4
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 3-1394. 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
USARTx	x = 0, 1, 2
UARTx	x = 3, 4
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_deasserttime_config(USART0, 0x0000001F);
```

函数 usart_driver_deasserttime_config

函数usart_driver_deasserttime_config描述见下表：

表 3-1395. 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
UARTx	x = 3, 4
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 3-1396. 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
UARTx	x = 3, 4
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 3-1397. 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
UARTx	x = 3, 4
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 3-1398. 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
USARTx	x = 0, 1, 2
UARTx	x = 3, 4
输入参数{in}	
dmacmd	USART DMA模式
USART_TRANSMIT_DMA_ENABLE	USART DMA发送使能
USART_TRANSMIT_DMA_DISABLE	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 3-1399. 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
USARTx	x = 0, 1, 2
UARTx	x = 3, 4
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 3-1400. 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
UARTx	x = 3, 4
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 3-1401. 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
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 3-1402. 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
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 3-1403. 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
输入参数{in}	
wum	唤醒模式

<i>USART_WUM_ADD R</i>	WUF在地址匹配时置位
<i>USART_WUM_STA RTB</i>	WUF在检测到起始位时置位
<i>USART_WUM_RBN E</i>	WUF在检测到RBNE时置位
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 3-1404. 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
<i>USARTx</i>	x = 0, 1, 2
<i>UARTx</i>	x = 3, 4
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 3-1405. 函数 `usart_fifo_disable`

函数名称	<code>usart_fifo_disable</code>
函数原型	<code>void usart_fifo_disable (uint32_t usart_periph);</code>
功能描述	禁能FIFO
先决条件	-
被调用函数	-
输入参数{in}	
<code>usart_periph</code>	外设USARTx
<code>USARTx</code>	<code>x = 0, 1, 2</code>
<code>UARTx</code>	<code>x = 3, 4</code>
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 3-1406. 函数 `usart_transmit_fifo_threshold_config`

函数名称	<code>usart_transmit_fifo_threshold_config</code>
函数原型	<code>void usart_transmit_fifo_threshold_config(uint32_t usart_periph, uint32_t txthreshold);</code>
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}	
<code>usart_periph</code>	外设USARTx
<code>USARTx</code>	<code>x = 0, 1, 2</code>
<code>UARTx</code>	<code>x = 3, 4</code>
输入参数{in}	
<code>txthreshold</code>	发送FIFO阈值
<code>USART_TFTCFG_T HRESHOLD_1_8</code>	发送FIFO到达其深度的1/8
<code>USART_TFTCFG_T HRESHOLD_1_4</code>	发送FIFO到达其深度的1/4
<code>USART_TFTCFG_T HRESHOLD_1_2</code>	发送FIFO到达其深度的1/2
<code>USART_TFTCFG_T</code>	发送FIFO到达其深度的3/4

HRESHOLD_3_4	
USART_TFTCFG_THRESHOLD_7_8	发送FIFO到达其深度的7/8
USART_TFTCFG_THRESHOLD_EMPTY	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 3-1407. 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
UARTx	x = 3, 4
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_	接收FIFO变为满

THRESHOLD_FULL	
L	
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 3-1408. 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
UARTx	x = 3, 4
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 3-1409. 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
UARTx	x = 3, 4
输入参数{in}	
flag	USART标志位, 参考 表3-1332. 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 3-1410. 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
UARTx	x = 3, 4
输入参数{in}	
flag	USART标志位, 参考 表3-1332. 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_FERR	帧错误标志
USART_FLAG_NERR	噪声错误标志

USART_FLAG_OR ERR	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_CTSF	CTS变化标志
USART_FLAG_RT	接收超时标志
USART_FLAG_EB	块结束标志
USART_FLAG_AM0	ADDR0匹配标志
USART_FLAG_AM1	ADDR1匹配标志
USART_FLAG_WU	从深度睡眠模式唤醒标志
USART_FLAG_EPERR	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 3-1411. 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
UARTx	x = 3, 4
输入参数{in}	

interrupt	USART中断USART标志位，参考 表3-1334. 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 3-1412. 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
UARTx	x = 3, 4
输入参数{in}	
interrupt	USART中断USART标志位，参考 表3-1334. 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 3-1413. 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
USARTx	x = 0, 1, 2
UARTx	x = 3, 4
输入参数{in}	
int_flag	USART中断标志, 参考 表3-1333. 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 3-1414. 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
USARTx	x = 0, 1, 2
UARTx	x = 3, 4
输入参数{in}	
int_flag	USART中断标志, 参考 表3-1333. 枚举类型usart_interrupt_flag_enum 只能选择一个参数

USART_INT_FLAG_PERR	校验错误中断标志
USART_INT_FLAG_ERR_FERR	帧错误中断标志
USART_INT_FLAG_ERR_NERR	噪声错误中断标志
USART_INT_FLAG_RBNE_ORERR	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);
```

3.35. 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[3.35.1](#)描述了VREF的寄存器列表，章节[3.35.2](#)对VREF库函数进行说明。

3.35.1. 外设寄存器说明

VREF寄存器列表如下表所示：

表 3-1415. 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

3.35.2. 外设库函数说明

VREF库函数列表如下表所示：

表 3-1416. 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 3-1417. 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 3-1418. 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 3-1419. 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 3-1420. 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 3-1421. 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 3-1422. 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 3-1423. 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_048V	VREF参考电压选择为2.048 V
VREF_VOLTAGE_S EL_2_5V	VREF参考电压选择为2.5 V

VREF_VOLTAGE_SEL_2_9V	VREF参考电压选择为2.9 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 3-1424. 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 3-1425. 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();
```

3.36. 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[3.36.1](#)描述了WWDGT的寄存器列表，章节[3.36.2](#)对WWDGT库函数进行说明。

3.36.1. 外设寄存器说明

WWDGT寄存器列表如下表所示:

表 3-1426. 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

3.36.2. 外设库函数说明

WWDGT库函数列表如下表所示:

表 3-1427. 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 3-1428. 函数 `wwdgt_deinit`

函数名称	<code>wwdgt_deinit</code>
函数原型	<code>void wwdgt_deinit(void);</code>
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 3-1429. 函数 `wwdgt_enable`

函数名称	<code>wwdgt_enable</code>
函数原型	<code>void wwdgt_enable(void);</code>
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 3-1430. 函数 `wwdgt_counter_update`

函数名称	<code>wwdgt_counter_update</code>
函数原型	<code>void wwdgt_counter_update(uint16_t counter_value);</code>
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 3-1431. 函数 `wwdgt_config`

函数名称	<code>wwdgt_config</code>
函数原型	<code>void wwdgt_config(uint16_t counter, uint16_t window, uint32_t prescaler);</code>
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
<code>WWDGT_CFG_PSC_DIV1</code>	WWDGT计数器时钟为 (PCLK3/4096) /1
<code>WWDGT_CFG_PSC_DIV2</code>	WWDGT计数器时钟为 (PCLK3/4096) /2
<code>WWDGT_CFG_PSC_DIV4</code>	WWDGT计数器时钟为 (PCLK3/4096) /4
<code>WWDGT_CFG_PSC_DIV8</code>	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 3-1432. 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 3-1433. 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 3-1434. 函数 wwdgt_flag_clear

函数名称	wwdgt_flag_clear
函数原型	void wwdgt_flag_clear(void);
功能描述	清除WWDGT提前唤醒中断标志位状态
先决条件	-
被调用函数	-
输入参数{in}	
-	-
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* clear early wakeup interrupt state of WWDGT */
```

```
wwdgt_flag_clear();
```

4. 版本历史

表 4-1. 版本历史

版本号.	说明	日期
1.0	初稿发布	2024 年 11 月 8 日
1.1	<p>1. 在表 3-711 中添加 LPTIMER_EXTRIGGER_RTCTAMP1 和 LPTIMER_EXTRIGGER_RTCTAMP2。</p> <p>2. 在表 3-1285 中添加 TRIGSEL_INPUT_RTC_TAMP1 和 TRIGSEL_INPUT_RTC_TAMP2。</p> <p>3. 3.27.2 小节中 hrtimer_comparecfg_struct_para_init(comparecfg_para)更新为 hrtimer_comparecfg_struct_para_init(&comparecfg_para)。</p> <p>4. 更新表 3-963 中函数 hrtimer_output_exchange 参数 timersrc 描述。</p> <p>5. 在表 3-965 中添加函数 hrtimer_timers_waveform_init 的被调用函数，在表 3-971 中添加被调用函数 channel_output_config。</p> <p>6. 删除表 3-969 中函数 hrtimer_slavetimer_waveform_compare_config、表 3-971 中函数 hrtimer_slavetimer_waveform_channel_config、表 3-972 中函数 hrtimer_slavetimer_waveform_channel_software_request、表 3-973 中函数 hrtimer_slavetimer_waveform_channel_output_level_get、表 3-974 中函数 function hrtimer_slavetimer_waveform_channel_state_get 的输入参数 timer_id。</p> <p>7. 表 3-1025 中函数接口 hrtimer_slavetimer_capture_config(uint32_t hrtimer_periph, uint32_t timer_id, uint32_t capturex, uint32_t trgsource)更新为 hrtimer_slavetimer_capture_config(uint32_t hrtimer_periph, uint32_t timer_id, uint32_t capturex, uint64_t trgsource)。</p> <p>8. 更新图 2-2 和图 2-5。</p> <p>9. 将表 3-1102 中</p>	2025 年 2 月 11 日

版本号.	说明	日期
	<p>TIMER4_CIO_INPUT_IRC40K / TIMER15_CIO_INPUT_IRC40K / TIMER16_CIO_INPUT_IRC40K 改为 TIMER4_CIO_INPUT_IRC32K / TIMER15_CIO_INPUT_IRC32K / TIMER16_CIO_INPUT_IRC32K。</p> <p>10. 增加函数 mpu_enable / mpu_disable / mpu_attribute_struct_para_init / mpu_attribute_config。</p>	

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.