

GigaDevice Semiconductor Inc.

GD32L23x

Arm® Cortex®-M23 32-bit MCU

**固件库
使用指南**

2.5 版本

(2026 年 2 月)

目录

目录	1
图索引	4
表索引	5
1. 介绍	27
1.1. 文档和固件库规则	27
1.1.1. 外设缩写	27
1.1.2. 命名规则	28
2. 固件库概述	29
2.1. 文件组织结构	29
2.1.1. Examples 文件夹	30
2.1.2. Firmware 文件夹	30
2.1.3. Template 文件夹	30
2.1.4. Utilities 文件夹	33
2.2. 固件库文件描述	33
3. 外设固件库	34
3.1. 外设固件库概述	34
3.2. ADC	34
3.2.1. 外设寄存器描述	34
3.2.2. 外设库函数说明	35
3.3. CAN（仅适用于 GD32L235xx 系列）	60
3.3.1. 外设寄存器说明	61
3.3.2. 外设库函数说明	61
3.4. CAU	80
3.4.1. 外设寄存器说明	80
3.4.2. 外设库函数说明	80
3.5. CMP	108
3.5.1. 外设寄存器说明	108
3.5.2. 外设库函数说明	109
3.6. CRC	119
3.6.1. 外设寄存器说明	119
3.6.2. 外设库函数说明	119
3.7. CTC	127
3.7.1. 外设寄存器说明	127

3.7.2.	外设库函数说明	127
3.8.	DBG	140
3.8.1.	外设寄存器说明	141
3.8.2.	外设库函数说明	141
3.9.	DAC	146
3.9.1.	外设寄存器说明	146
3.9.2.	外设库函数说明	146
3.10.	DMA/DMAMUX.....	161
3.10.1.	外设寄存器说明	162
3.10.2.	外设库函数说明	162
3.11.	EXTI.....	202
3.11.1.	外设寄存器说明	202
3.11.2.	外设库函数说明	202
3.12.	FMC	211
3.12.1.	外设寄存器说明	211
3.12.2.	外设库函数说明	212
3.13.	FWDGT.....	237
3.13.1.	外设寄存器说明	237
3.13.2.	外设库函数说明	237
3.14.	GPIO	243
3.14.1.	外设寄存器说明	243
3.14.2.	外设库函数说明	243
3.15.	I2C	254
3.15.1.	外设寄存器说明	254
3.15.2.	外设库函数说明	254
3.16.	LPTIMER.....	293
3.16.1.	外设寄存器说明	293
3.16.2.	外设库函数说明	293
3.17.	LPUART	316
3.17.1.	外设寄存器说明	316
3.17.2.	外设库函数说明	316
3.18.	MISC	346
3.18.1.	外设寄存器说明	346
3.18.2.	外设库函数说明	346
3.19.	PMU.....	354
3.19.1.	外设寄存器说明	354
3.19.2.	外设库函数说明	354
3.20.	RCU.....	374
3.20.1.	外设寄存器说明	375

3.20.2.	外设库函数说明	375
3.21.	RTC	409
3.21.1.	外设寄存器描述	409
3.21.2.	外设库函数描述	410
3.22.	SLCD.....	438
3.22.1.	外设寄存器说明	438
3.22.2.	外设库函数说明	438
3.23.	SPI.....	456
3.23.1.	外设寄存器说明	456
3.23.2.	外设库函数说明	457
3.24.	SYSCFG.....	485
3.24.1.	外设寄存器说明	485
3.24.2.	外设库函数说明	486
3.25.	TIMER	494
3.25.1.	外设寄存器说明	494
3.25.2.	外设库函数说明	495
3.26.	TRNG.....	555
3.26.1.	外设寄存器说明	555
3.26.2.	外设库函数说明	555
3.27.	USART	560
3.27.1.	外设寄存器说明	560
3.27.2.	外设库函数说明	561
3.28.	VREF	610
3.28.1.	外设寄存器说明	610
3.28.2.	外设库函数说明	610
3.29.	WWDGT	615
3.29.1.	外设寄存器说明	615
3.29.2.	外设库函数说明	615
4.	版本历史.....	621

图索引

图 2-1. GD32L23x 固件库文件组织结构.....	29
图 2-2. 选择外设例程文件	31
图 2-3. 拷贝外设例程文件	31
图 2-4. 打开工程文件.....	32
图 2-5. 配置工程文件.....	32
图 2-6. 编译调试下载.....	33

表索引

表 1-1. 外设缩写	27
表 2-1. 固件函数库文件描述	33
表 3-1. 外设固件库函数描述格式	34
表 3-2. ADC 寄存器	34
表 3-3. ADC 库函数	35
表 3-4. 函数 adc_deinit	36
表 3-5. 函数 adc_enable	36
表 3-6. 函数 adc_disable	37
表 3-7. 函数 adc_calibration_number	37
表 3-8. 函数 adc_calibration_enable	38
表 3-9. 函数 adc_dma_mode_enable	39
表 3-10. 函数 adc_dma_mode_disable	39
表 3-11. 函数 adc_discontinuous_mode_config	40
表 3-12. 函数 adc_special_function_config	40
表 3-13. 函数 adc_channel_16_to_19	41
表 3-14. 函数 adc_data_alignment_config	42
表 3-15. 函数 adc_channel_length_config	42
表 3-16. 函数 adc_routine_channel_config	43
表 3-17. 函数 adc_inserted_channel_config	44
表 3-18. 函数 adc_inserted_channel_offset_config	45
表 3-19. 函数 adc_channel_differential_mode_config	46
表 3-20. 函数 adc_external_trigger_config	46
表 3-21. 函数 adc_external_trigger_source_config	47
表 3-22. 函数 adc_software_trigger_enable	48
表 3-23. 函数 adc_routine_data_read	49
表 3-24. 函数 adc_inserted_data_read	49
表 3-25. 函数 adc_watchdog_single_channel_enable	50
表 3-26. 函数 adc_watchdog_sequence_channel_enable	51
表 3-27. 函数 adc_watchdog_disable	51
表 3-28. 函数 adc_watchdog_threshold_config	52
表 3-29. 函数 adc_resolution_config	52
表 3-30. 函数 adc_oversample_mode_config	53
表 3-31. 函数 adc_oversample_mode_enable	55
表 3-32. 函数 adc_oversample_mode_disable	55
表 3-33. 函数 adc_charge_pulse_width_counter	56
表 3-34. 函数 adc_charge_flag_get	56
表 3-35. 函数 adc_flag_get	57
表 3-36. 函数 adc_flag_clear	57
表 3-37. 函数 adc_interrupt_enable	58
表 3-38. 函数 adc_interrupt_disable	59

表 3-39. 函数 <code>adc_interrupt_flag_get</code>	59
表 3-40. 函数 <code>adc_interrupt_flag_clear</code>	60
表 3-41. CAN 寄存器	61
表 3-42. CAN 库函数	61
表 3-43. 结构体 <code>can_parameter_struct</code>	62
表 3-44. 结构体 <code>can_transmit_message_struct</code>	62
表 3-45. 结构体 <code>can_receive_message_struct</code>	63
表 3-46. 结构体 <code>can_filter_parameter_struct</code>	63
表 3-47. 函数 <code>can_deinit</code>	63
表 3-48. 函数 <code>can_struct_para_init</code>	64
表 3-49. 函数 <code>can_init</code>	65
表 3-50. 函数 <code>can_filter_init</code>	65
表 3-51. 函数 <code>can_debug_freeze_enable</code>	66
表 3-52. 函数 <code>can_debug_freeze_disable</code>	66
表 3-53. 函数 <code>can_time_trigger_mode_enable</code>	67
表 3-54. 函数 <code>can_time_trigger_mode_disable</code>	67
表 3-55. 函数 <code>can_message_transmit</code>	68
表 3-56. 函数 <code>can_transmit_states</code>	68
表 3-57. 函数 <code>can_transmission_stop</code>	69
表 3-58. 函数 <code>can_message_receive</code>	69
表 3-59. 函数 <code>can_fifo_release</code>	70
表 3-60. 函数 <code>can_receive_message_length_get</code>	70
表 3-61. 函数 <code>can_working_mode_set</code>	71
表 3-62. 函数 <code>can_wakeup</code>	72
表 3-63. 函数 <code>can_error_get</code>	72
表 3-64. 函数 <code>can_receive_error_number_get</code>	73
表 3-65. 函数 <code>can_transmit_error_number_get</code>	73
表 3-66. 函数 <code>can_flag_get</code>	74
表 3-67. 函数 <code>can_flag_clear</code>	74
表 3-68. 函数 <code>can_interrupt_enable</code>	75
表 3-69. 函数 <code>can_interrupt_disable</code>	76
表 3-70. 函数 <code>can_interrupt_flag_get</code>	77
表 3-71. 函数 <code>can_interrupt_flag_clear</code>	78
表 3-72. CAU 寄存器	80
表 3-73. CAU 库函数	80
表 3-74. 结构体 <code>cau_key_parameter_struct</code>	81
表 3-75. 结构体 <code>cau_iv_parameter_struct</code>	82
表 3-76. 结构体 <code>cau_context_parameter_struct</code>	82
表 3-77. 结构体 <code>cau_parameter_struct</code>	82
表 3-78. 函数 <code>cau_deinit</code>	83
表 3-79. 函数 <code>cau_struct_para_init</code>	83
表 3-80. 函数 <code>cau_key_struct_para_init</code>	84
表 3-81. 函数 <code>cau_iv_struct_para_init</code>	84
表 3-82. 函数 <code>cau_context_struct_para_init</code>	85

表 3-83. 函数 cau_enable	85
表 3-84. 函数 cau_disable	86
表 3-85. 函数 cau_dma_enable	86
表 3-86. 函数 cau_dma_disable	87
表 3-87. 函数 cau_init.....	87
表 3-88. 函数 cau_aes_keysize_config	89
表 3-89. 函数 cau_key_init	89
表 3-90. 函数 cau_iv_init	90
表 3-91. 函数 cau_phase_config	91
表 3-92. 函数 cau_fifo_flush	91
表 3-93. 函数 cau_enable_state_get.....	92
表 3-94. 函数 cau_data_write.....	92
表 3-95. 函数 cau_data_read.....	93
表 3-96. 函数 cau_context_save	93
表 3-97. 函数 cau_context_restore.....	94
表 3-98. 函数 cau_aes_ecb	95
表 3-99. 函数 cau_aes_cbc	96
表 3-100. 函数 cau_aes_ctr.....	97
表 3-101. 函数 cau_aes_cfb	98
表 3-102. 函数 cau_aes_ofb	99
表 3-103. 函数 cau_aes_gcm	100
表 3-104. 函数 cau_aes_ccm	101
表 3-105. 函数 cau_tdes_ecb.....	102
表 3-106. 函数 cau_tdes_cbc.....	103
表 3-107. 函数 cau_des_ecb	103
表 3-108. 函数 cau_des_cbc	104
表 3-109. 函数 cau_flag_get.....	105
表 3-110. 函数 cau_interrupt_enable.....	107
表 3-111. 函数 cau_interrupt_disable	107
表 3-112. 函数 cau_interrupt_flag_get	108
表 3-113. CMP 寄存器.....	108
表 3-114. CMP 库函数.....	109
表 3-115. 枚举类型 cmp_enum.....	109
表 3-116. 函数 cmp_deinit	109
表 3-117. 函数 cmp_mode_init	110
表 3-118. 函数 cmp_noninverting_input_select	111
表 3-119. 函数 cmp_output_init.....	112
表 3-120. 函数 cmp_blanking_init.....	113
表 3-121. 函数 cmp_enable	114
表 3-122. 函数 cmp_disable	114
表 3-123. 函数 cmp_window_enable	115
表 3-124. 函数 cmp_window_disable	115
表 3-125. 函数 cmp_lock_enable	116
表 3-126. 函数 cmp_voltage_scaler_enable.....	116

表 3-127. 函数 cmp_voltage_scaler_disable.....	117
表 3-128. 函数 cmp_scaler_bridge_enable	117
表 3-129. 函数 cmp_scaler_bridge_disable	118
表 3-130. 函数 cmp_output_level_get.....	118
表 3-131. CRC 寄存器.....	119
表 3-132. CRC 库函数.....	119
表 3-133. 函数 crc_deinit.....	119
表 3-134. 函数 crc_reverse_output_data_enable.....	120
表 3-135. 函数 crc_reverse_output_data_disable.....	120
表 3-136. 函数 crc_data_register_reset	121
表 3-137. 函数 crc_data_register_read	121
表 3-138. 函数 crc_free_data_register_read	122
表 3-139. 函数 crc_free_data_register_write	122
表 3-140. 函数 crc_init_data_register_write.....	123
表 3-141. 函数 crc_input_data_reverse_config.....	123
表 3-142. 函数 crc_polynomial_size_set	124
表 3-143. 函数 crc_polynomial_set.....	125
表 3-144. 函数 crc_single_data_calculate	125
表 3-145. 函数 crc_block_data_calculate	126
表 3-146. CTC 寄存器	127
表 3-147. CTC 库函数	127
表 3-148. 函数 ctc_deinit	128
表 3-149. 函数 ctc_counter_enable	128
表 3-150. 函数 ctc_counter_disable	129
表 3-151. 函数 ctc_irc48m_trim_value_config.....	129
表 3-152. 函数 ctc_software_refsource_pulse_generate	130
表 3-153. 函数 ctc_hardware_trim_mode_config.....	130
表 3-154. 函数 ctc_refsource_polarity_config.....	131
表 3-155. 函数 ctc_refsource_signal_select	132
表 3-156. 函数 ctc_refsource_prescaler_config.....	132
表 3-157. 函数 ctc_clock_limit_value_config.....	133
表 3-158. 函数 ctc_counter_reload_value_config.....	134
表 3-159. 函数 ctc_counter_capture_value_read.....	134
表 3-160. 函数 ctc_counter_direction_read	135
表 3-161. 函数 ctc_counter_reload_value_read	135
表 3-162. 函数 ctc_irc48m_trim_value_read	136
表 3-163. 函数 ctc_interrupt_enable.....	136
表 3-164. 函数 ctc_interrupt_disable.....	137
表 3-165. 函数 ctc_interrupt_flag_get	137
表 3-166. 函数 ctc_interrupt_flag_clear	138
表 3-167. 函数 ctc_flag_get.....	139
表 3-168. 函数 ctc_flag_clear.....	140
表 3-169. DBG 寄存器.....	141
表 3-170. DBG 库函数.....	141

表 3-171. 枚举类型 dbg_periph_enum.....	141
表 3-172. 函数 dbg_deinit.....	142
表 3-173. 函数 dbg_id_get.....	142
表 3-174. 函数 dbg_low_power_enable	143
表 3-175. 函数 dbg_low_power_disable	144
表 3-176. 函数 dbg_periph_enable	144
表 3-177. 函数 dbg_periph_disable	145
表 3-178. DAC 寄存器.....	146
表 3-179. DAC 库函数.....	146
表 3-180. 函数 dac_deinit	147
表 3-181. 函数 dac_enable	147
表 3-182. 函数 dac_disable	148
表 3-183. 函数 dac_dma_enable	148
表 3-184. 函数 dac_dma_disable	149
表 3-185. 函数 dac_gpio_connect_config	150
表 3-186. 函数 dac_output_buffer_enable.....	150
表 3-187. 函数 dac_output_buffer_disable.....	151
表 3-188. 函数 dac_output_value_get.....	151
表 3-189. 函数 dac_data_set.....	152
表 3-190. 函数 dac_trigger_enable	153
表 3-191. 函数 dac_trigger_disable	153
表 3-192. 函数 dac_trigger_source_config	154
表 3-193. 函数 dac_software_trigger_enable	155
表 3-194. 函数 dac_wave_mode_config	156
表 3-195. 函数 dac_lfsr_noise_config	156
表 3-196. 函数 dac_triangle_noise_config	157
表 3-197. 函数 dac_flag_get.....	158
表 3-198. 函数 dac_flag_clear.....	158
表 3-199. 函数 dac_interrupt_enable.....	159
表 3-200. 函数 dac_interrupt_disable.....	160
表 3-201. 函数 dac_interrupt_flag_get	160
表 3-202. 函数 dac_interrupt_flag_clear.....	161
表 3-203. DMA 寄存器	162
表 3-204. DMAMUX 寄存器.....	162
表 3-205. DMA 库函数	162
表 3-206. DMAMUX 库函数.....	163
表 3-207. 结构体 dma_parameter_struct	164
表 3-208. 结构体 dmamux_sync_parameter_struct.....	164
表 3-209. 结构体 dmamux_gen_parameter_struct.....	165
表 3-210. 枚举 dmamux_interrupt_enum	165
表 3-211. 枚举 dmamux_flag_enum.....	165
表 3-212. 枚举 dmamux_interrupt_flag_enum.....	166
表 3-213. 枚举 dma_channel_enum.....	167
表 3-214. 枚举 dmamux_multiplexer_channel_enum	167

表 3-215. 枚举 dmamux_generator_channel_enum	167
表 3-216. 函数 dma_deinit	168
表 3-217. 函数 dma_struct_para_init	168
表 3-218. 函数 dma_init	169
表 3-219. 函数 dma_circulation_enable	169
表 3-220. 函数 dma_circulation_disable	170
表 3-221. 函数 dma_memory_to_memory_enable	170
表 3-222. 函数 dma_memory_to_memory_disable	171
表 3-223. 函数 dma_channel_enable	171
表 3-224. 函数 dma_channel_disable	172
表 3-225. 函数 dma_periph_address_config	172
表 3-226. 函数 dma_memory_address_config	173
表 3-227. 函数 dma_transfer_number_config	174
表 3-228. 函数 dma_transfer_number_get	174
表 3-229. 函数 dma_priority_config	175
表 3-230. 函数 dma_memory_width_config	176
表 3-231. 函数 dma_periph_width_config	176
表 3-232. 函数 dma_memory_increase_enable	177
表 3-233. 函数 dma_memory_increase_disable	177
表 3-234. 函数 dma_periph_increase_enable	178
表 3-235. 函数 dma_periph_increase_disable	178
表 3-236. 函数 dma_transfer_direction_config	179
表 3-237. 函数 dma_flag_get	180
表 3-238. 函数 dma_flag_clear	180
表 3-239. 函数 dma_interrupt_enable	181
表 3-240. 函数 dma_interrupt_disable	182
表 3-241. 函数 dma_interrupt_flag_get	182
表 3-242. 函数 dma_interrupt_flag_clear	183
表 3-243. 函数 dmamux_sync_struct_para_init	184
表 3-244. 函数 dmamux_synchronization_init	184
表 3-245. 函数 dmamux_synchronization_enable	185
表 3-246. 函数 dmamux_synchronization_disable	185
表 3-247. 函数 dmamux_event_generation_enable	186
表 3-248. 函数 dmamux_event_generation_disable	187
表 3-249. 函数 dmamux_gen_struct_para_init	187
表 3-250. 函数 dmamux_request_generator_init	188
表 3-251. 函数 dmamux_request_generator_channel_enable	188
表 3-252. 函数 dmamux_request_generator_channel_disable	189
表 3-253. 函数 dmamux_synchronization_polarity_config	189
表 3-254. 函数 dmamux_request_forward_number_config	190
表 3-255. 函数 dmamux_sync_id_config	191
表 3-256. 函数 dmamux_request_id_config	192
表 3-257. 函数 dma_interrupt_disable	196
表 3-258. 函数 dmamux_request_generate_number_config	197

表 3-259. 函数 dmamux_trigger_id_config	197
表 3-260. 函数 dmamux_flag_get.....	199
表 3-261. 函数 dmamux_flag_clear.....	199
表 3-262. 函数 dmamux_interrupt_enable.....	200
表 3-263. 函数 dmamux_interrupt_disable.....	200
表 3-264. 函数 dmamux_interrupt_flag_get	201
表 3-265. 函数 dmamux_interrupt_flag_clear	201
表 3-266. EXTI 寄存器.....	202
表 3-267. EXTI 库函数.....	202
表 3-268. GD32L233xx 的枚举类型 exti_line_enum	203
表 3-269. GD32L235xx 的枚举类型 exti_line_enum	204
表 3-270. 枚举类型 exti_mode_enum	205
表 3-271. 枚举类型 exti_trig_type_enum	205
表 3-272. 函数 exti_deinit	205
表 3-273. 函数 exti_init.....	205
表 3-274. 函数 exti_interrupt_enable.....	206
表 3-275. 函数 exti_interrupt_disable.....	207
表 3-276. 函数 exti_event_enable	207
表 3-277. 函数 exti_event_disable	208
表 3-278. 函数 exti_software_interrupt_enable	208
表 3-279. 函数 exti_software_interrupt_disable	209
表 3-280. 函数 exti_flag_get.....	209
表 3-281. 函数 exti_flag_clear.....	210
表 3-282. 函数 exti_interrupt_flag_get	210
表 3-283. 函数 exti_interrupt_flag_clear.....	211
表 3-284. FMC 寄存器.....	212
表 3-285. FMC 固件库函数	212
表 3-286. 枚举类型 fmc_state_enum	213
表 3-287. 枚举类型 fmc_flag_enum.....	214
表 3-288. 枚举类型 fmc_interrupt_flag_enum.....	214
表 3-289. 枚举类型 fmc_interrupt_enum.....	215
表 3-290. 函数 fmc_unlock.....	215
表 3-291. 函数 Function fmc_lock	216
表 3-292. 函数 fmc_wscnt_set.....	216
表 3-293. 函数 fmc_prefetch_enable	217
表 3-294. 函数 fmc_prefetch_disable	217
表 3-295. 函数 fmc_low_power_enable	218
表 3-296. 函数 fmc_low_power_disable	218
表 3-297. 函数 fmc_page_erase	219
表 3-298. 函数 fmc_mass_erase	219
表 3-299. 函数 fmc_word_program.....	220
表 3-300. 函数 fmc_fast_program.....	220
表 3-301. 函数 fmc_doubleword_program	221
表 3-302. 函数 ob_unlock.....	222

表 3-303. 函数 ob_lock.....	223
表 3-304. 函数 ob_erase	223
表 3-305. 函数 ob_write_protection_enable.....	224
表 3-306. 函数 ob_security_protection_config.....	224
表 3-307. 函数 ob_user_write	225
表 3-308. 函数 ob_data_program	226
表 3-309. 函数 ob_user_get	227
表 3-310. 函数 ob_data_get.....	227
表 3-311. 函数 ob_write_protection_get	228
表 3-312. 函数 ob_security_protection_flag_get	228
表 3-313. 函数 fmc_ecc_address_get	229
表 3-314. 函数 fmc_slp_unlock	229
表 3-315. 函数 fmc_sleep_slp_enable.....	230
表 3-316. 函数 fmc_sleep_slp_disable.....	230
表 3-317. 函数 fmc_run_slp_enable	231
表 3-318. 函数 fmc_run_slp_disable	231
表 3-319. 函数 fmc_run_slp_disable fmc_sleep_mode_enter	232
表 3-320. 函数 fmc_pd_mode_enter.....	232
表 3-321. 函数 fmc_slp_mode_exit.....	233
表 3-322. 函数 fmc_flag_get.....	234
表 3-323. 函数 fmc_flag_clear	234
表 3-324. 函数 fmc_interrupt_enable	235
表 3-325. 函数 fmc_interrupt_disable	235
表 3-326. 函数 fmc_interrupt_flag_get.....	236
表 3-327. 函数 fmc_interrupt_flag_clear.....	236
表 3-328. FWDGT 寄存器	237
表 3-329. FWDGT 库函数	237
表 3-330. 函数 fwdgt_write_enable.....	238
表 3-331. 函数 fwdgt_write_disable.....	238
表 3-332. 函数 fwdgt_enable.....	239
表 3-333. 函数 fwdgt_prescaler_value_config.....	239
表 3-334. 函数 fwdgt_reload_value_config	240
表 3-335. 函数 fwdgt_window_value_config.....	241
表 3-336. 函数 fwdgt_counter_reload	241
表 3-337. 函数 fwdgt_config	242
表 3-338. 函数 fwdgt_flag_get	242
表 3-339. GPIO 寄存器.....	243
表 3-340. GPIO 库函数.....	244
表 3-341. 函数 gpio_deinit.....	244
表 3-342. 函数 gpio_mode_set	245
表 3-343. 函数 gpio_output_options_set.....	246
表 3-344. 函数 gpio_bit_set.....	246
表 3-345. 函数 gpio_bit_reset.....	247
表 3-346. 函数 gpio_bit_write	248

表 3-347. 函数 gpio_port_write	248
表 3-348. 函数 gpio_input_bit_get	249
表 3-349. 函数 gpio_input_port_get	250
表 3-350. 函数 gpio_output_bit_get.....	250
表 3-351. 函数 gpio_output_port_get.....	251
表 3-352. 函数 gpio_af_set.....	251
表 3-353. 函数 gpio_pin_lock	252
表 3-354. 函数 gpio_bit_toggle.....	253
表 3-355. 函数 gpio_port_toggle	253
表 3-356. I2C 寄存器	254
表 3-357. I2C 库函数	255
表 3-358. 枚举类型 i2c_interrupt_flag_enum	256
表 3-359. 函数 i2c_deinit	257
表 3-360. 函数 i2c_timing_config.....	257
表 3-361. 函数 i2c_digital_noise_filter_config.....	258
表 3-362. 函数 i2c_analog_noise_filter_enable	259
表 3-363. 函数 i2c_analog_noise_filter_disable	259
表 3-364. 函数 i2c_master_clock_config.....	260
表 3-365. 函数 i2c_master_addressing	260
表 3-366. 函数 i2c_address10_header_enable	261
表 3-367. 函数 i2c_address10_header_disable	262
表 3-368. 函数 i2c_address10_enable.....	262
表 3-369. 函数 i2c_address10_disable.....	263
表 3-370. 函数 i2c_automatic_end_enable.....	263
表 3-371. 函数 i2c_automatic_end_disable.....	264
表 3-372. 函数 i2c_slave_response_to_gcall_enable.....	264
表 3-373. 函数 i2c_slave_response_to_gcall_disable.....	265
表 3-374. 函数 i2c_stretch_scl_low_enable	265
表 3-375. 函数 i2c_stretch_scl_low_disable	266
表 3-376. 函数 i2c_address_config.....	266
表 3-377. 函数 i2c_address_bit_compare_config	267
表 3-378. 函数 i2c_address_disable	268
表 3-379. 函数 i2c_second_address_config	268
表 3-380. 函数 i2c_second_address_disable.....	269
表 3-381. 函数 i2c_receved_address_get.....	270
表 3-382. 函数 i2c_slave_byte_control_enable	270
表 3-383. 函数 i2c_slave_byte_control_disable	271
表 3-384. 函数 i2c_nack_enable.....	271
表 3-386. 函数 i2c_wakeup_from_deepsleep_enable	272
表 3-387. 函数 i2c_wakeup_from_deepsleep_disable	272
表 3-388. 函数 i2c_enable.....	273
表 3-389. 函数 i2c_disable.....	273
表 3-390. 函数 i2c_start_on_bus.....	274
表 3-391. 函数 i2c_stop_on_bus	274

表 3-392. 函数 i2c_data_transmit.....	275
表 3-393. 函数 i2c_data_receive.....	275
表 3-394. 函数 i2c_reload_enable	276
表 3-395. 函数 i2c_reload_disable	276
表 3-396. 函数 i2c_transfer_byte_number_config	277
表 3-397. 函数 i2c_dma_enable.....	278
表 3-398. 函数 i2c_dma_disable.....	278
表 3-399. 函数 i2c_pec_transfer.....	279
表 3-400. 函数 i2c_pec_enable	279
表 3-401. 函数 i2c_pec_disable	280
表 3-402. 函数 i2c_pec_value_get	280
表 3-403. 函数 i2c_smbus_alert_enable	281
表 3-404. 函数 i2c_smbus_alert_disable	281
表 3-405. 函数 i2c_smbus_default_addr_enable	282
表 3-406. 函数 i2c_smbus_default_addr_disable	282
表 3-407. 函数 i2c_smbus_host_addr_enable	283
表 3-408. 函数 i2c_smbus_host_addr_disable.....	283
表 3-409. 函数 i2c_extented_clock_timeout_enable	284
表 3-410. 函数 i2c_extented_clock_timeout_disable	284
表 3-411. 函数 i2c_clock_timeout_enable	285
表 3-412. 函数 i2c_clock_timeout_disable	285
表 3-413. 函数 i2c_bus_timeout_b_config	286
表 3-414. 函数 i2c_bus_timeout_a_config	287
表 3-415. 函数 i2c_idle_clock_timeout_config	287
表 3-416. 函数 i2c_flag_get.....	288
表 3-417. 函数 i2c_flag_clear.....	289
表 3-418. 函数 i2c_interrupt_enable	289
表 3-419. 函数 i2c_interrupt_disable.....	290
表 3-420. 函数 i2c_interrupt_flag_get	291
表 3-421. 函数 i2c_interrupt_flag_clear	292
表 3-422. LPTIMER 寄存器	293
表 3-423. TIMER 库函数	294
表 3-424. 结构体 lptimer_parameter_struct	294
表 3-425. 函数 lptimer_deinit	295
表 3-426. 函数 lptimer_struct_para_init.....	296
表 3-427. 函数 lptimer_init.....	296
表 3-428. 函数 lptimer_inputremap.....	297
表 3-429. 函数 lptimer_register_shadow_enable	298
表 3-430. 函数 lptimer_register_shadow_disable	299
表 3-431. 函数 lptimer_timeout_enable	299
表 3-432. 函数 lptimer_register_shadow_disable	300
表 3-433. 函数 lptimer_countinue_start.....	300
表 3-434. 函数 lptimer_countinue_start.....	301
表 3-435. 函数 lptimer_stop	302

表 3-436. 函数 lptimer_counter_read	302
表 3-437. 函数 lptimer_autoreload_read	303
表 3-438. 函数 lptimer_compare_read	304
表 3-439. 函数 lptimer_autoreload_value_config	304
表 3-440. 函数 lptimer_compare_value_config	305
表 3-441. 函数 lptimer_decodemode0_enable	306
表 3-442. 函数 lptimer_decodemode1_enable	306
表 3-443. 函数 lptimer_decodemode_disable	307
表 3-444. 函数 lptimer_highlevelcounter_enable	307
表 3-445. 函数 lptimer_highlevelcounter_disable	308
表 3-446. 函数 lptimer_flag_get	308
表 3-447. 函数 timer_flag_clear	310
表 3-448. 函数 lptimer_interrupt_enable	311
表 3-449. 函数 lptimer_interrupt_enable	312
表 3-450. 函数 lptimer_interrupt_flag_get	313
表 3-451. 函数 lptimer_interrupt_flag_clear	314
表 3-452. LPUART 寄存器	316
表 3-453. LPUART 库函数	316
表 3-454. 枚举类型 lpuart_flag_enum	317
表 3-455. 枚举类型 lpuart_interrupt_flag_enum	318
表 3-456. 枚举类型 lpuart_interrupt_enum	318
表 3-457. 枚举类型 lpuart_invert_enum	319
表 3-458. 函数 lpuart_deinit	319
表 3-459. 函数 lpuart_baudrate_set	320
表 3-460. 函数 lpuart_parity_config	320
表 3-461. 函数 lpuart_word_length_set	321
表 3-462. 函数 lpuart_stop_bit_set	321
表 3-463. 函数 lpuart_enable	322
表 3-464. 函数 lpuart_disable	323
表 3-465. 函数 lpuart_transmit_config	323
表 3-466. 函数 lpuart_receive_config	324
表 3-467. 函数 lpuart_data_first_config	324
表 3-468. 函数 lpuart_invert_config	325
表 3-469. 函数 lpuart_overrun_enable	326
表 3-470. 函数 lpuart_overrun_disable	326
表 3-471. 函数 lpuart_data_transmit	327
表 3-472. 函数 lpuart_data_receive	327
表 3-473. 函数 lpuart_command_enable	328
表 3-474. 函数 lpuart_address_config	328
表 3-475. 函数 lpuart_address_detection_mode_config	329
表 3-476. 函数 lpuart_mute_mode_enable	330
表 3-477. 函数 lpuart_mute_mode_disable	330
表 3-478. 函数 lpuart_mute_mode_wakeup_config	331
表 3-479. 函数 lpuart_halfduplex_enable	331

表 3-480. 函数 lpuart_halfduplex_disable	332
表 3-481. 函数 lpuart_hardware_flow_rts_config	332
表 3-482. 函数 lpuart_hardware_flow_cts_config	333
表 3-483. 函数 lpuart_hardware_flow_coherence_config	334
表 3-484. 函数 lpuart_rs485_driver_enable	334
表 3-485. 函数 lpuart_rs485_driver_disable	335
表 3-486. 函数 lpuart_driver_asserttime_config	335
表 3-487. 函数 lpuart_driver_deasserttime_config	336
表 3-488. 函数 lpuart_depolarity_config	336
表 3-489. 函数 lpuart_dma_receive_config	337
表 3-490. 函数 lpuart_dma_transmit_config	338
表 3-491. 函数 lpuart_reception_error_dma_disable	338
表 3-492. 函数 lpuart_reception_error_dma_enable	339
表 3-493. 函数 lpuart_wakeup_enable	339
表 3-494. 函数 lpuart_wakeup_disable	340
表 3-495. 函数 lpuart_wakeup_mode_config	340
表 3-496. 函数 lpuart_flag_get	341
表 3-497. 函数 lpuart_flag_clear	342
表 3-498. 函数 lpuart_interrupt_enable	343
表 3-499. 函数 lpuart_interrupt_disable	343
表 3-500. 函数 lpuart_interrupt_flag_get	344
表 3-501. 函数 lpuart_interrupt_flag_clear	345
表 3-502. NVIC 寄存器	346
表 3-503. SysTick 寄存器	346
表 3-504. MISC 库函数	346
表 3-505. GD32L233xx 的枚举类型 IRQn_Type	347
表 3-506. GD32L235xx 的枚举类型 IRQn_Type	348
表 3-507. 函数 nvic_irq_enable	350
表 3-508. 函数 nvic_irq_disable	350
表 3-509. 函数 nvic_system_reset	351
表 3-510. 函数 nvic_vector_table_set	351
表 3-511. 函数 system_lowpower_set	352
表 3-512. 函数 system_lowpower_reset	353
表 3-513. 函数 systick_clksource_set	353
表 3-514. PMU 寄存器	354
表 3-515. PMU 库函数	354
表 3-516. 函数 pmu_deinit	355
表 3-517. 函数 pmu_lvd_select	356
表 3-518. 函数 pmu_lvd_disable	357
表 3-519. 函数 pmu_lio_output_select	357
表 3-520. 函数 pmu_vc_enable	358
表 3-521. 函数 pmu_vc_disable	358
表 3-522. 函数 pmu_vcr_select	359
表 3-523. 函数 pmu_low_power_enable	359

表 3-524. 函数 pmu_low_power_disable	360
表 3-525. 函数 pmu_to_sleepmode	360
表 3-526. 函数 pmu_to_deepsleepmode	361
表 3-527. 函数 pmu_to_standbymode	362
表 3-528. 函数 pmu_wakeup_pin_enable	362
表 3-529. 函数 pmu_wakeup_pin_disable	363
表 3-530. 函数 pmu_backup_write_enable	364
表 3-531. 函数 pmu_backup_write_disable	364
表 3-532. 函数 pmu_sram_power_config	365
表 3-533. 函数 pmu_core1_power_config	365
表 3-534. 函数 pmu_deepsleep2_retention_enable	366
表 3-535. 函数 pmu_deepsleep2_retention_disable	366
表 3-536. 函数 pmu_eflash_sleep_power_config	367
表 3-537. 函数 pmu_eflash_deepsleep_power_config	367
表 3-538. 函数 pmu_deepsleep2_sram_power_config	368
表 3-539. 函数 pmu_deepsleep_wait_time_config	368
表 3-540. 函数 pmu_wakeuptime_core1_software_enable	369
表 3-541. 函数 pmu_wakeuptime_core1_software_disable	369
表 3-542. 函数 pmu_wakeuptime_eflash_config	370
表 3-543. 函数 Function pmu_wakeuptime_sram_config	370
表 3-544. 函数 pmu_wakeuptime_sram_software_enable	371
表 3-545. 函数 pmu_wakeuptime_sram_software_disable	371
表 3-546. 函数 pmu_wakeuptime_deepsleep2_software_enable	372
表 3-547. 函数 pmu_wakeuptime_deepsleep2_software_disable	372
表 3-548. 函数 pmu_flag_get	373
表 3-549. 函数 pmu_flag_clear	374
表 3-550. RCU 寄存器	375
表 3-551. RCU 库函数	375
表 3-552. 枚举类型 rcu_periph_enum	376
表 3-553. 枚举类型 rcu_periph_sleep_enum	378
表 3-554. 枚举类型 rcu_periph_reset_enum	378
表 3-555. 枚举类型 rcu_flag_enum	379
表 3-556. 枚举类型 rcu_int_flag_enum	379
表 3-557. 枚举类型 rcu_int_flag_clear_enum	380
表 3-558. 枚举类型 rcu_int_enum	380
表 3-559. 枚举类型 rcu_osci_type_enum	380
表 3-560. 枚举类型 rcu_clock_freq_enum	381
表 3-561. 枚举类型 usart_idx_enum	381
表 3-562. 枚举类型 lptimer_idx_enum	381
表 3-563. 枚举类型 lpuart_idx_enum	381
表 3-564. 枚举类型 i2c_idx_enum	382
表 3-565. 函数 rcu_deinit	382
表 3-566. 函数 rcu_periph_clock_enable	382
表 3-567. 函数 rcu_periph_clock_disable	383

表 3-568. 函数 rcu_periph_clock_sleep_enable.....	383
表 3-569. 函数 rcu_periph_clock_sleep_disable.....	384
表 3-570. 函数 rcu_periph_reset_enable.....	384
表 3-571. 函数 rcu_periph_reset_disable.....	385
表 3-572. 函数 rcu_bkp_reset_enable.....	385
表 3-573. 函数 rcu_bkp_reset_disable.....	386
表 3-574. 函数 rcu_system_clock_source_config.....	386
表 3-575. 函数 rcu_system_clock_source_get.....	387
表 3-576. 函数 rcu_ahb_clock_config.....	387
表 3-577. 函数 rcu_apb1_clock_config.....	388
表 3-578. 函数 rcu_apb2_clock_config.....	388
表 3-579. 函数 rcu_adc_clock_config.....	389
表 3-580. 函数 rcu_ckout_config.....	390
表 3-581. 函数 rcu_pll_config.....	391
表 3-582. 函数 rcu_usart_clock_config.....	392
表 3-583. 函数 rcu_i2c_clock_config.....	393
表 3-584. 函数 rcu_lptimer_clock_config.....	393
表 3-585. 函数 rcu_lptimer_clock_config.....	394
表 3-586. 函数 rcu_lpuart_clock_config.....	395
表 3-587. 函数 rcu_lpuart_clock_config.....	395
表 3-588. 函数 rcu_irc16mdiv_clock_config.....	396
表 3-589. 函数 rcu_irc16mdiv_clock_config.....	397
表 3-590. 函数 rcu_rtc_clock_config.....	397
表 3-591. 函数 rcu_pll_source_ck_prediv_config.....	398
表 3-592. 函数 rcu_lxtal_drive_capability_config.....	399
表 3-593. 函数 rcu_lp_bandgap_config.....	399
表 3-594. 函数 rcu_osci_stab_wait.....	400
表 3-595. 函数 rcu_osci_on.....	400
表 3-596. 函数 rcu_osci_off.....	401
表 3-597. 函数 rcu_osci_bypass_mode_enable.....	401
表 3-598. 函数 rcu_osci_bypass_mode_disable.....	402
表 3-599. 函数 rcu_irc16m_adjust_value_set.....	403
表 3-600. 函数 rcu_hxtal_clock_monitor_enable.....	403
表 3-601. 函数 rcu_hxtal_clock_monitor_disable.....	404
表 3-602. 函数 rcu_lxtal_clock_monitor_enable.....	404
表 3-603. 函数 rcu_lxtal_clock_monitor_disable.....	405
表 3-604. 函数 rcu_voltage_key_unlock.....	405
表 3-605. 函数 rcu_clock_freq_get.....	406
表 3-606. 函数 rcu_flag_get.....	406
表 3-607. 函数 rcu_all_reset_flag_clear.....	407
表 3-608. 函数 rcu_interrupt_flag_get.....	407
表 3-609. 函数 rcu_interrupt_flag_clear.....	408
表 3-610. 函数 rcu_interrupt_enable.....	408
表 3-611. 函数 rcu_interrupt_disable.....	409

表 3-612. RTC 寄存器	409
表 3-613. RTC 库函数	410
表 3-614. 结构体 rtc_parameter_struct	411
表 3-615. 结构体 rtc_alarm_struct	412
表 3-616. 结构体 rtc_timestamp_struct	412
表 3-617. 结构体 rtc_tamper_struct	412
表 3-618. 函数 rtc_deinit	413
表 3-619. 函数 rtc_init	413
表 3-620. 函数 rtc_init_mode_enter	414
表 3-621. 函数 rtc_init_mode_exit	414
表 3-622. 函数 rtc_register_sync_wait	415
表 3-623. 函数 rtc_current_time_get	415
表 3-624. 函数 rtc_subsecond_get	416
表 3-625. 函数 rtc_alarm_config	416
表 3-626. 函数 rtc_alarm_subsecond_config	417
表 3-627. 函数 rtc_alarm_enable	418
表 3-628. 函数 rtc_alarm_disable	418
表 3-629. 函数 rtc_alarm_get	419
表 3-630. 函数 rtc_alarm_subsecond_get	419
表 3-631. 函数 rtc_timestamp_enable	420
表 3-632. 函数 rtc_timestamp_disable	421
表 3-633. 函数 rtc_timestamp_internalevent_config	421
表 3-634. 函数 rtc_timestamp_get	422
表 3-635. 函数 rtc_timestamp_subsecond_get	422
表 3-636. 函数 rtc_tamper_enable	423
表 3-637. 函数 rtc_tamper_disable	423
表 3-638. 函数 rtc_tamper_mask	424
表 3-639. 函数 rtc_tamper_without_bkp_reset	424
表 3-640. 函数 rtc_output_pin_select	425
表 3-641. 函数 rtc_alarm_output_config	426
表 3-642. 函数 rtc_calibration_output_config	426
表 3-643. 函数 rtc_hour_adjust	427
表 3-644. 函数 rtc_second_adjust	427
表 3-645. 函数 rtc_bypass_shadow_enable	428
表 3-646. 函数 rtc_bypass_shadow_disable	429
表 3-647. 函数 rtc_refclock_detection_enable	429
表 3-648. 函数 rtc_refclock_detection_disable	430
表 3-649. 函数 rtc_wakeup_enable	430
表 3-650. 函数 rtc_wakeup_disable	431
表 3-651. 函数 rtc_wakeup_clock_set	431
表 3-652. 函数 rtc_wakeup_timer_set	432
表 3-653. 函数 rtc_wakeup_timer_get	432
表 3-654. 函数 rtc_smooth_calibration_config	433
表 3-655. 函数 rtc_interrupt_enable	434

表 3-656. 函数 rtc_interrupt_disable	434
表 3-657. 函数 rtc_flag_get	435
表 3-658. 函数 rtc_flag_clear	436
表 3-645. 函数 rtc_lxtal_stab_reset_enable	437
表 3-646. 函数 rtc_lxtal_stab_reset_disable	437
表 3-659. SLCD 寄存器.....	438
表 3-660. SLCD 寄存器.....	438
表 3-661. 枚举类型 slcd_data_register_enum.....	439
表 3-662. 函数 slcd_deinit	439
表 3-663. 函数 slcd_enable	440
表 3-664. 函数 slcd_disable	440
表 3-665. 函数 slcd_init.....	441
表 3-666. 函数 slcd_enhance_mode_enable.....	443
表 3-667. 函数 slcd_enhance_mode_disable	443
表 3-668. 函数 slcd_weak_driving_resistance_select.....	444
表 3-669. 函数 slcd_bias_voltage_select	444
表 3-670. 函数 slcd_duty_select	445
表 3-671. 函数 slcd_clock_config	445
表 3-672. 函数 slcd_blink_mode_config.....	447
表 3-673. 函数 slcd_contrast_ratio_config.....	448
表 3-674. 函数 slcd_dead_time_config	449
表 3-675. 函数 slcd_pulse_on_duration_config.....	450
表 3-676. 函数 slcd_com_seg_remap	450
表 3-677. 函数 slcd_voltage_source_select.....	451
表 3-678. 函数 slcd_high_drive_config.....	452
表 3-679. 函数 slcd_data_register_write	452
表 3-680. 函数 slcd_data_update_request	453
表 3-681. 函数 slcd_flag_get.....	453
表 3-682. 函数 slcd_flag_clear.....	454
表 3-683. 函数 slcd_interrupt_enable.....	454
表 3-684. 函数 slcd_interrupt_disable.....	455
表 3-685. 函数 slcd_interrupt_flag_get	455
表 3-686. 函数 slcd_interrupt_flag_clear	456
表 3-687. SPI/I2S 寄存器	456
表 3-688. SPI/I2S 库函数	457
表 3-689. 结构体 spi_parameter_struct.....	458
表 3-690. 函数 spi_i2s_deinit.....	458
表 3-691. 函数 spi_struct_para_init.....	459
表 3-692. 函数 spi_init.....	460
表 3-693. 函数 spi_enable	460
表 3-694. 函数 spi_disable	461
表 3-695. 函数 i2s_init.....	461
表 3-696. 函数 i2s_psc_config.....	462
表 3-697. 函数 i2s_enable.....	464

表 3-698. 函数 i2s_disable.....	464
表 3-699. 函数 spi_nss_output_enable.....	465
表 3-700. 函数 spi_nss_output_disable.....	465
表 3-701. 函数 spi_nss_internal_high.....	466
表 3-702. 函数 spi_nss_internal_low.....	466
表 3-703. 函数 spi_dma_enable	467
表 3-704. 函数 spi_dma_disable	468
表 3-705. 函数 spi_transmit_odd_config.....	468
表 3-706. 函数 spi_receive_odd_config.....	469
表 3-707. 函数 spi_i2s_data_frame_format_config	469
表 3-708. 函数 spi_fifo_access_size_config	470
表 3-709. 函数 spi_bidirectional_transfer_config	471
表 3-710. 函数 spi_i2s_data_transmit.....	471
表 3-711. 函数 spi_i2s_data_receive	472
表 3-712. 函数 spi_crc_polynomial_set.....	473
表 3-713. 函数 spi_crc_polynomial_get.....	473
表 3-714. 函数 spi_crc_length_set.....	474
表 3-715. 函数 spi_crc_on.....	474
表 3-716. 函数 spi_crc_off.....	475
表 3-717. 函数 spi_crc_next.....	475
表 3-718. 函数 spi_crc_get.....	476
表 3-719. 函数 spi_crc_error_clear.....	476
表 3-720. 函数 spi_ti_mode_enable.....	477
表 3-721. 函数 spi_ti_mode_disable.....	477
表 3-722. 函数 spi_nssp_mode_enable	478
表 3-723. 函数 spi_nssp_mode_disable	478
表 3-724. 函数 spi_quad_enable	479
表 3-725. 函数 spi_quad_disable	479
表 3-726. 函数 spi_quad_write_enable	480
表 3-727. 函数 spi_quad_read_enable	480
表 3-730. 函数 spi_i2s_format_error_clear	481
表 3-731. 函数 spi_i2s_flag_get	482
表 3-732. 函数 spi_i2s_interrupt_enable	483
表 3-733. 函数 spi_i2s_interrupt_disable	484
表 3-734. 函数 spi_i2s_interrupt_flag_get.....	484
表 3-735. SYSCFG 寄存器	485
表 3-736. SYSCFG 库函数	486
表 3-737. 函数 syscfg_deinit.....	486
表 3-738. 函数 syscfg_exti_line_config	487
表 3-739. 函数 syscfg_pin_remap_enable.....	487
表 3-740. 函数 syscfg_pin_remap_disable.....	488
表 3-741. 函数 syscfg_high_current_enable.....	489
表 3-742. 函数 syscfg_high_current_disable.....	490
表 3-743. 函数 irq_latency_set	490

表 3-744. 函数 syscfg_bootmode_get.....	491
表 3-745. 函数 syscfg_sram_waitstate_insert.....	491
表 3-746. 函数 syscfg_sram_waitstate_cancel.....	492
表 3-747. 函数 syscfg_lock_config.....	492
表 3-748. 函数 syscfg_flag_get	493
表 3-749. 函数 syscfg_flag_clear	494
表 3-750. TIMER 寄存器	494
表 3-751. TIMER 库函数	495
表 3-752. 结构体 timer_parameter_struct.....	498
表 3-753. 结构体 timer_break_parameter_struct.....	498
表 3-754. 结构体 timer_oc_parameter_struct	498
表 3-755. 结构体 timer_ic_parameter_struct.....	499
表 3-756. 函数 timer_deinit	499
表 3-757. 函数 timer_struct_para_init	500
表 3-758. 函数 timer_init.....	500
表 3-759. 函数 timer_enable.....	501
表 3-760. 函数 timer_disable.....	501
表 3-761. 函数 timer_auto_reload_shadow_enable.....	502
表 3-762. 函数 timer_auto_reload_shadow_disable.....	503
表 3-763. 函数 timer_update_event_enable	503
表 3-764. 函数 timer_update_event_disable	504
表 3-765. 函数 timer_counter_alignment.....	504
表 3-766. 函数 timer_counter_up_direction	505
表 3-767. 函数 timer_counter_down_direction.....	505
表 3-768. 函数 timer_prescaler_config	506
表 3-769. 函数 timer_repetition_value_config	507
表 3-770. 函数 timer_autoreload_value_config.....	507
表 3-771. 函数 timer_counter_value_config	508
表 3-772. 函数 timer_counter_read.....	509
表 3-773. 函数 timer_prescaler_read.....	509
表 3-774. 函数 timer_single_pulse_mode_config.....	510
表 3-775. 函数 timer_update_source_config	510
表 3-776. 函数 timer_dma_enable.....	511
表 3-777. 函数 timer_dma_disable.....	512
表 3-778. 函数 timer_channel_dma_request_source_select	513
表 3-779. 函数 timer_dma_transfer_config	514
表 3-780. 函数 timer_event_software_generate	515
表 3-781. 函数 timer_break_struct_para_init.....	516
表 3-782. 函数 timer_break_config.....	517
表 3-783. 函数 timer_break_enable	518
表 3-784. 函数 timer_break_disable	518
表 3-785. 函数 timer_automatic_output_enable.....	519
表 3-786. 函数 timer_automatic_output_disable.....	519
表 3-787. 函数 timer_primary_output_config	520

表 3-788. 函数 timer_channel_control_shadow_config.....	520
表 3-789. 函数 timer_channel_control_shadow_update_config	521
表 3-790. 函数 timer_channel_output_struct_para_init.....	522
表 3-791. 函数 timer_channel_output_config.....	522
表 3-792. 函数 timer_channel_output_mode_config.....	523
表 3-793. 函数 timer_channel_output_pulse_value_config	524
表 3-794. 函数 timer_channel_output_shadow_config.....	525
表 3-795. 函数 timer_channel_output_fast_config	526
表 3-796. 函数 timer_channel_output_clear_config	527
表 3-797. 函数 timer_channel_output_polarity_config	528
表 3-798. 函数 timer_channel_complementary_output_polarity_config	529
表 3-799. 函数 timer_channel_output_state_config	530
表 3-800. 函数 timer_channel_complementary_output_state_config	531
表 3-801. 函数 timer_channel_input_struct_para_init	531
表 3-802. 函数 timer_input_capture_config	532
表 3-803. 函数 timer_channel_input_capture_prescaler_config	533
表 3-804. 函数 timer_channel_capture_value_register_read.....	534
表 3-805. 函数 timer_input_pwm_capture_config	535
表 3-806. 函数 timer_hall_mode_config	536
表 3-807. 函数 timer_input_trigger_source_select.....	536
表 3-808. 函数 timer_master_output_trigger_source_select.....	537
表 3-809. 函数 timer_slave_mode_select.....	538
表 3-810. 函数 timer_master_slave_mode_config	539
表 3-811. 函数 timer_external_trigger_config.....	540
表 3-812. 函数 timer_quadrature_decoder_mode_config	541
表 3-813. 函数 timer_internal_clock_config.....	542
表 3-814. 函数 timer_internal_trigger_as_external_clock_config.....	543
表 3-815. 函数 timer_external_trigger_as_external_clock_config.....	543
表 3-816. 函数 timer_external_clock_mode0_config	544
表 3-817. 函数 timer_external_clock_mode1_config	545
表 3-818. 函数 timer_external_clock_mode1_disable	546
表 3-819. 函数 timer_channel_remap_config	547
表 3-820. 函数 timer_write_chxval_register_config	548
表 3-821. 函数 timer_flag_get	549
表 3-822. 函数 timer_flag_clear	550
表 3-823. 函数 timer_interrupt_enable	551
表 3-824. 函数 timer_interrupt_disable	552
表 3-825. 函数 timer_interrupt_flag_get	553
表 3-826. 函数 timer_interrupt_flag_clear	554
表 3-827. TRNG 寄存器	555
表 3-828. TRNG 库函数	555
表 3-829. 枚举 trng_flag_enum.....	555
表 3-830. 枚举 trng_int_flag_enum.....	555
表 3-831. 函数 trng_deinit	556

表 3-832. 函数 trng_enable	556
表 3-833. 函数 trng_disable	557
表 3-834. 函数 trng_get_true_random_data	557
表 3-835. 函数 trng_flag_get	558
表 3-836. 函数 trng_interrupt_enable	558
表 3-837. 函数 trng_interrupt_disable	559
表 3-838. 函数 trng_interrupt_flag_get	559
表 3-839. 函数 trng_interrupt_flag_clear	560
表 3-840. USART 寄存器	560
表 3-841. USART 库函数	561
表 3-842. 枚举类型 usart_flag_enum	563
表 3-843. 枚举类型 usart_interrupt_flag_enum	563
表 3-844. 枚举类型 usart_interrupt_enum	564
表 3-845. 枚举类型 usart_invert_enum	564
表 3-846. 函数 usart_deinit	565
表 3-847. 函数 usart_baudrate_set	565
表 3-848. 函数 usart_parity_config	566
表 3-849. 函数 usart_word_length_set	566
表 3-850. 函数 usart_stop_bit_set	567
表 3-851. 函数 usart_enable	568
表 3-852. 函数 usart_disable	568
表 3-853. 函数 usart_transmit_config	569
表 3-854. 函数 usart_receive_config	570
表 3-855. 函数 usart_data_first_config	570
表 3-856. 函数 usart_invert_config	571
表 3-857. 函数 usart_overrun_enable	572
表 3-858. 函数 usart_overrun_disable	572
表 3-859. 函数 usart_oversample_config	573
表 3-860. 函数 usart_sample_bit_config	574
表 3-861. 函数 usart_receiver_timeout_enable	574
表 3-862. 函数 usart_receiver_timeout_disable	575
表 3-863. 函数 usart_receiver_timeout_threshold_config	575
表 3-864. 函数 usart_data_transmit	576
表 3-865. 函数 usart_data_receive	576
表 3-866. 函数 usart_command_enable	577
表 3-867. 函数 usart_address_config	578
表 3-868. 函数 usart_address_detection_mode_config	578
表 3-869. 函数 usart_mute_mode_enable	579
表 3-870. 函数 usart_mute_mode_disable	580
表 3-871. 函数 usart_mute_mode_wakeup_config	580
表 3-872. 函数 usart_lin_mode_enable	581
表 3-873. 函数 usart_lin_mode_disable	581
表 3-874. 函数 usart_lin_break_detection_length_config	582
表 3-875. 函数 usart_halfduplex_enable	582

表 3-876. 函数 usart_halfduplex_disable	583
表 3-877. 函数 usart_clock_enable.....	583
表 3-878. 函数 usart_clock_disable.....	584
表 3-879. 函数 usart_synchronous_clock_config.....	584
表 3-880. 函数 usart_guard_time_config.....	585
表 3-881. 函数 usart_smartcard_mode_enable	586
表 3-882. 函数 usart_smartcard_mode_disable	586
表 3-883. 函数 usart_smartcard_mode_nack_enable	587
表 3-884. 函数 usart_smartcard_mode_nack_disable	587
表 3-885. 函数 usart_smartcard_mode_early_nack_enable	588
表 3-886. 函数 usart_smartcard_mode_early_nack_disable	588
表 3-887. 函数 usart_smartcard_autoretry_config	589
表 3-888. 函数 usart_block_length_config.....	590
表 3-889. 函数 usart_irda_mode_enable	590
表 3-890. 函数 usart_irda_mode_disable	591
表 3-891. 函数 usart_prescaler_config	591
表 3-892. 函数 usart_irda_lowpower_config.....	592
表 3-893. 函数 usart_hardware_flow_rts_config.....	592
表 3-894. 函数 usart_hardware_flow_cts_config	593
表 3-895. 函数 usart_hardware_flow_coherence_config	594
表 3-896. 函数 usart_rs485_driver_enable	594
表 3-897. 函数 usart_rs485_driver_disable	595
表 3-898. 函数 usart_driver_asserttime_config.....	595
表 3-899. 函数 usart_driver_deasserttime_config	596
表 3-900. 函数 usart_depolarity_config	596
表 3-901. 函数 usart_dma_receive_config	597
表 3-902. 函数 usart_dma_transmit_config	598
表 3-903. 函数 usart_reception_error_dma_disable	598
表 3-904. 函数 usart_reception_error_dma_enable.....	599
表 3-905. 函数 usart_wakeup_enable.....	599
表 3-906. 函数 usart_wakeup_disable	600
表 3-907. 函数 usart_wakeup_mode_config	600
表 3-908. 函数 usart_receive_fifo_enable	601
表 3-909. 函数 usart_receive_fifo_disable	602
表 3-910. 函数 usart_receive_fifo_counter_number	602
表 3-911. 函数 usart_flag_get	603
表 3-912. 函数 usart_flag_clear	604
表 3-913. 函数 usart_interrupt_enable	605
表 3-914. 函数 usart_interrupt_disable	606
表 3-915. 函数 usart_interrupt_flag_get	607
表 3-916. 函数 usart_interrupt_flag_clear	608
表 3-917. VREF 寄存器	610
表 3-918. VREF 库函数	610
表 3-919. 函数 vref_deinit.....	610

表 3-920. 函数 <code>vref_enable</code>	611
表 3-921. 函数 <code>vref_disable</code>	611
表 3-922. 函数 <code>vref_high_impedance_mode_enable</code>	612
表 3-923. 函数 <code>vref_high_impedance_mode_disable</code>	612
表 3-924. 函数 <code>vref_voltage_select</code>	613
表 3-925. 函数 <code>vref_status_get</code>	613
表 3-926. 函数 <code>vref_calib_value_set</code>	614
表 3-927. 函数 <code>vref_calib_value_get</code>	614
表 3-928. WWDGT 寄存器.....	615
表 3-929. WWDGT 库函数.....	615
表 3-930. 函数 <code>wwdgt_deinit</code>	616
表 3-931. 函数 <code>wwdgt_enable</code>	616
表 3-932. 函数 <code>wwdgt_counter_update</code>	617
表 3-933. 函数 <code>wwdgt_config</code>	617
表 3-934. 函数 <code>wwdgt_flag_get</code>	618
表 3-935. 函数 <code>wwdgt_flag_clear</code>	619
表 3-936. 函数 <code>wwdgt_interrupt_enable</code>	619
表 4-1. 版本历史	621
表 4-2. 函数 <code>rcu_rtc_clock_config</code> 中函数功能和入参的描述增加 SLCD	621

1. 介绍

本手册介绍了32位基于ARM微控制器GD32L23x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GD32L23x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	局域网控制器模块
CAU	加密处理器
CMP	比较器
CRC	循环冗余校验计算单元
CTC	时钟校准控制器
DBG	调试模块
DAC	数模转换器
DMA	直接存储器访问控制器
DMAMUX	DMA请求多路复用器

外设缩写	说明
EXTI	外部中断事件控制器
FMC	闪存控制器
FWDDGT	独立看门狗
GPIO/AFIO	通用和备用输入/输出接口
I2C	内部集成电路总线接口
LPTIMER	低功耗定时器
LPUART	低功耗通用异步收发器
MISC	嵌套中断向量列表控制器
PMU	电源管理单元
RCU	复位和时钟单元
RTC	实时时钟
SLCD	段码LCD控制器
SPI/I2S	串行外设接口/片上音频接口
SYSCFG	系统配置
TIMER	定时器
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：

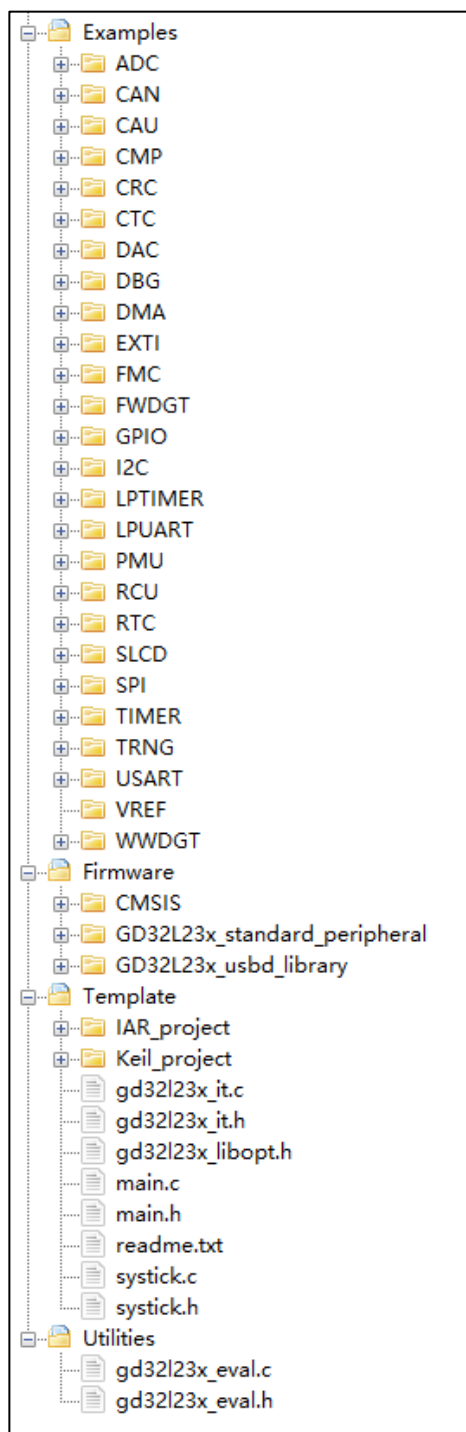
- XXX表示任一外设缩写，例如：ADC。更多缩写相关信息参阅[外设缩写](#)；
- 源文件和头文件命名都以“gd32l23x_”作为开头，例如：gd32l23x_adc.h；
- 常量仅被应用于一个文件的，定义于该文件中；被应用于多个文件的，在对应头文件中定义。所有常量都由英文字母大写书写；
- 寄存器作为常量处理。他们的命名都由英文字母大写书写。在大多数情况下，寄存器缩写规范与本用户手册一致；
- 变量名采用全部小写，有多个单词组成的，在单词之间以下划线分隔；
- 外设函数的命名以该外设的缩写加下划线为开头，有多个单词组成的，在单词之间以下划线分隔，所有外设函数都由英文字母小写书写。

2. 固件库概述

2.1. 文件组织结构

GD32L23x_Firmware_Library，文件组织结构见下图：

图 2-1. GD32L23x 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；
- **gd32l23x_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）；
- **gd32l23x_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）；
- **gd32l23x_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 M23**内核的支持文件、基于**Cortex M23**内核处理器的启动代码和库引导文件以及基于**GD32L23x**的全局头文件和系统配置文件；
- **GD32L23x_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：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。

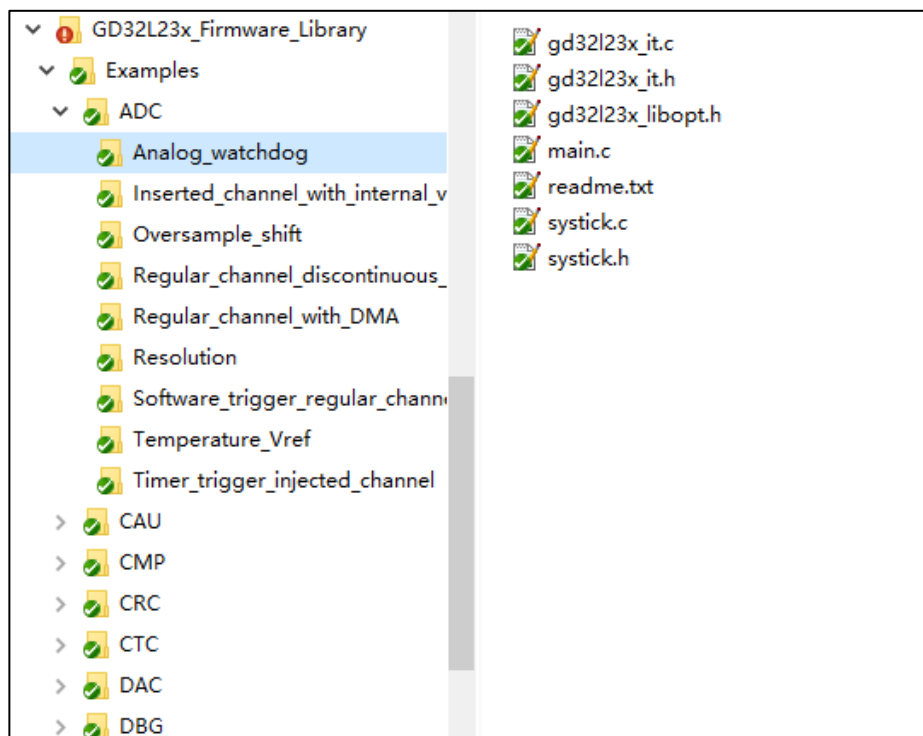
2.1.3. Template 文件夹

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“**Analog_watchdog**”，如下图所示：

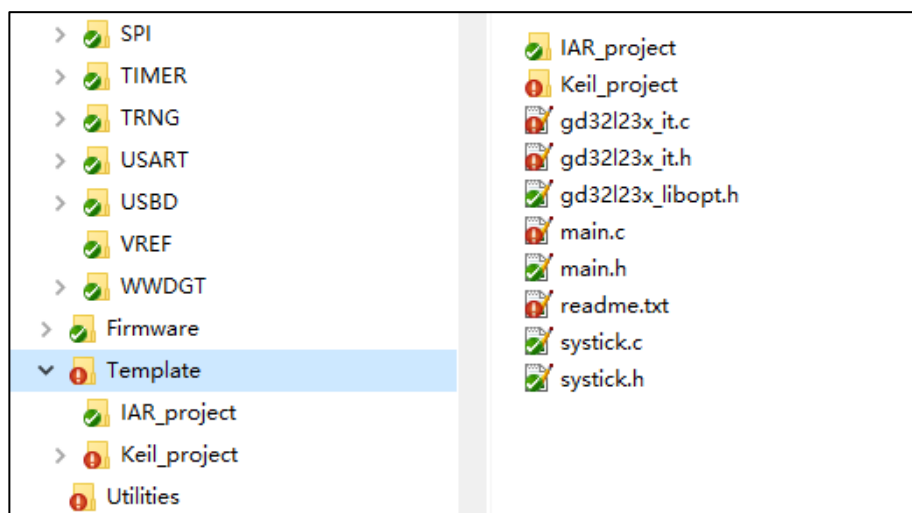
图 2-2. 选择外设例程文件



拷贝文件

打开“Template”文件夹，将“IAR_project”和“Keil_project”两个文件夹保留，其他文件都删除，然后将“Analog_watchdog”文件夹中的所有文件拷到“Template”文件夹子目录下，如下图所示：

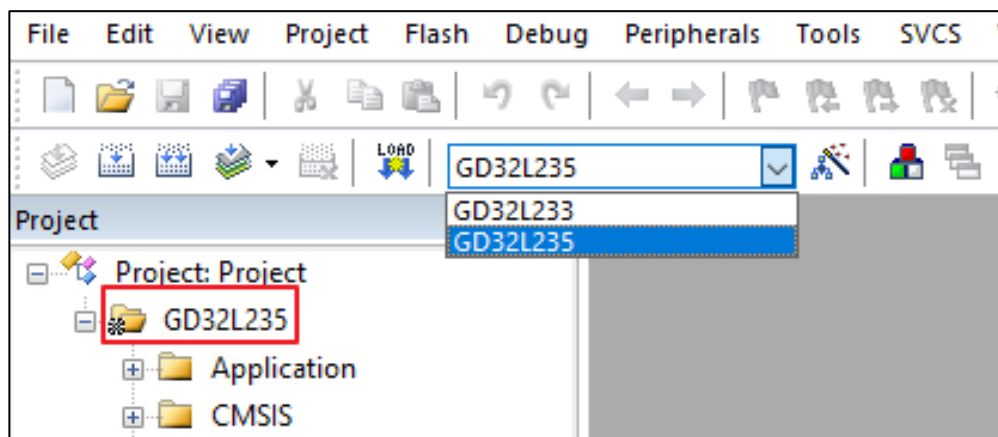
图 2-3. 拷贝外设例程文件



打开工程

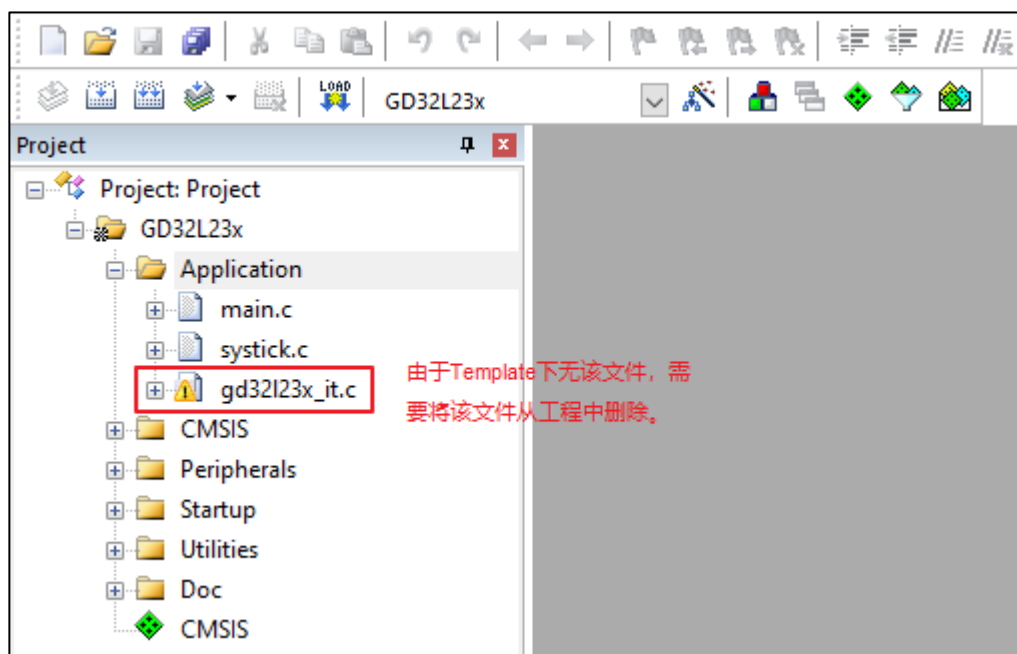
GD提供Keil和IAR两种版本的工程，根据客户所安装的软件，打开不同的project，如“Keil_project”，打开\Template\Keil_project\Project.uvprojx，可选择L233或L235工程，如下图所示：

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：

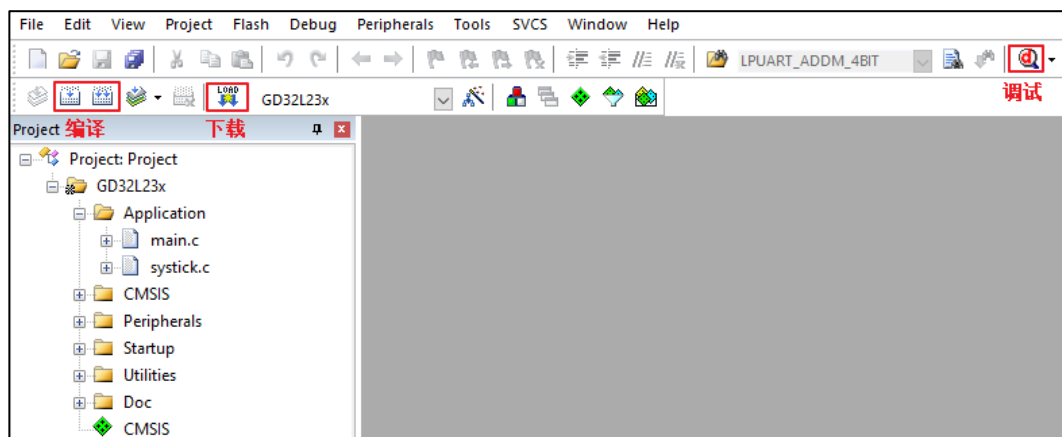
图 2-5. 配置工程文件



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：

- gd32l23x_eval.h文件是运行固件库例程所需关于评估板的头文件；
- gd32l23x_eval.c文件是运行固件库例程所需关于评估板的源文件。

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

2.2. 固件库文件描述

下表列举和描述了固件库使用的主要文件。

表 2-1. 固件函数库文件描述

文件名	描述
gd32l23x_libopt.h	包含了所有外设的头文件的头文件。它是唯一一个用户需要包括在自己应用中的文件，起到应用和库之间界面的作用。
main.c	主函数体示例。
gd32l23x_it.h	头文件，包含所有中断处理函数原形。
gd32l23x_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。
gd32l23x_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。
gd32l23x_xxx.c	由C语言编写的外设xxx的驱动源程序文件。
systick.h	systick.c的头文件。包含systick配置函数的定义，以及外部用延时函数的定义。
systick.c	systick配置与延时函数源文件。
readme.txt	固件库例程使用及配置说明文档。

3. 外设固件库

3.1. 外设固件库概述

外设固件库函数的描述格式如下表：

表 3-1. 外设固件库函数描述格式

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

3.2. ADC

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

3.2.1. 外设寄存器描述

ADC寄存器列表如下表所示：

表 3-2. ADC 寄存器

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

寄存器名称	寄存器描述
ADC_RSQ2	常规序列寄存器2
ADC_ISQ	注入序列寄存器
ADC_IDATAx	注入数据寄存器x (x=0..3)
ADC_RDATA	常规数据寄存器
ADC_OVSAMPCTL	过采样控制寄存器
ADC_CCTL	充电控制寄存器
ADC_DIFCTL	差分模式控制寄存器

3.2.2. 外设库函数说明

ADC库函数列表如下表所示：

表 3-3. ADC 库函数

库函数名称	库函数描述
adc_deinit	复位ADC
adc_enable	使能ADC
adc_disable	禁能ADC
adc_calibration_number	配置ADC校准次数
adc_calibration_enable	ADC校准复位
adc_dma_mode_enable	使能DMA请求
adc_dma_mode_disable	禁能DMA请求
adc_discontinuous_mode_config	配置ADC间断模式
adc_special_function_config	配置ADC特殊功能
adc_channel_16_to_19	配置温度传感器、内部参考电压通道、VBAT通道或VSLCD通道
adc_data_alignment_config	配置ADC数据对齐方式
adc_channel_length_config	配置常规序列或注入序列的通道长度
adc_routine_channel_config	配置ADC常规序列通道
adc_inserted_channel_config	配置ADC注入序列通道
adc_inserted_channel_offset_config	配置ADC注入序列通道数据偏移值
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_external_trigger_source_config	配置ADC外部触发源
adc_software_trigger_enable	使能ADC软件触发
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_watchdog_single_channel_enable	使能ADC模拟看门狗单通道功能
adc_watchdog_sequence_channel_enable	使能ADC模拟看门狗序列通道功能
adc_watchdog_disable	禁能ADC模拟看门狗
adc_watchdog_threshold_config	配置ADC模拟看门狗阈值

库函数名称	库函数描述
adc_resolution_config	配置ADC分辨率
adc_oversample_mode_config	配置ADC过采样模式
adc_oversample_mode_enable	使能ADC过采样
adc_oversample_mode_disable	禁能ADC过采样
adc_charge_pulse_width_counter	配置ADC充电脉冲宽度计数器
adc_charge_flag_get	获取ADC充电状态标志位
adc_flag_get	获取ADC标志位
adc_flag_clear	清除ADC标志位
adc_interrupt_enable	ADC中断使能
adc_interrupt_disable	ADC中断禁能
adc_interrupt_flag_get	获取ADC中断标志位
adc_interrupt_flag_clear	清除ADC中断标志位

函数 adc_deinit

函数adc_deinit描述见下表：

表 3-4. 函数 adc_deinit

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

例如：

```
/* reset ADC*/
```

```
adc_deinit();
```

函数 adc_enable

函数adc_enable描述见下表：

表 3-5. 函数 adc_enable

函数名称	adc_enable
函数原形	void adc_enable(void);
功能描述	使能ADC外设
先决条件	-

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

例如：

```
/* enable ADC */
```

```
adc_enable();
```

函数 adc_disable

函数adc_disable描述见下表：

表 3-6. 函数 adc_disable

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

```
adc_disable();
```

函数 adc_calibration_number（只对 GD32L235 有效）

函数 adc_calibration_number 描述见下表：

表 3-7. 函数 adc_calibration_number

函数名称	adc_calibration_number
函数原形	void adc_calibration_number(uint32_t clb_num);
功能描述	配置ADC校准次数
先决条件	-
被调用函数	-

输入参数{in}	
clb_num	校准次数
ADC_CALIBRATION_NUM1	校准1次
ADC_CALIBRATION_NUM2	校准2次
ADC_CALIBRATION_NUM4	校准4次
ADC_CALIBRATION_NUM8	校准8次
ADC_CALIBRATION_NUM16	校准16次
ADC_CALIBRATION_NUM32	校准32次
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* configure ADC calibration number */
```

```
adc_calibration_number(ADC_CALIBRATION_NUM1);
```

函数 adc_calibration_enable

函数adc_calibration_enable描述见下表：

表 3-8. 函数 adc_calibration_enable

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

例如：

```
/* ADC calibration and reset calibration */
```

```
adc_calibration_enable();
```

函数 adc_dma_mode_enable

函数 adc_dma_mode_enable 描述见下表：

表 3-9. 函数 adc_dma_mode_enable

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

例如：

```
/* enable ADC DMA request */
```

```
adc_dma_mode_enable();
```

函数 adc_dma_mode_disable

函数 adc_dma_mode_disable 描述见下表：

表 3-10. 函数 adc_dma_mode_disable

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

例如：

```
/* disable ADC DMA request */
```

```
adc_dma_mode_disable();
```

函数 adc_discontinuous_mode_config

函数 adc_discontinuous_mode_config 描述见下表:

表 3-11. 函数 adc_discontinuous_mode_config

函数名称	adc_discontinuous_mode_config
函数原形	void adc_discontinuous_mode_config(uint8_t adc_sequence, uint8_t length);
功能描述	配置ADC间断模式
先决条件	-
被调用函数	-
输入参数{in}	
adc_sequence	序列选择
ADC_ROUTINE_CHANNEL	常规序列
ADC_INSERTED_CHANNEL	注入序列
ADC_CHANNEL_DISCON_DISABLE	常规序列和注入序列间断模式禁能
输入参数{in}	
length	间断模式下的转换数目，常规序列取值为1..16，注入序列取值无意义
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* configure ADC routine sequence discontinuous mode */
```

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

函数 adc_special_function_config

函数 adc_special_function_config 描述见下表:

表 3-12. 函数 adc_special_function_config

函数名称	adc_special_function_config
函数原形	void adc_special_function_config(uint32_t function, ControlStatus newvalue);
功能描述	配置ADC特殊功能
先决条件	-
被调用函数	-
输入参数{in}	
function	功能配置
ADC_SCAN_MODE	扫描模式选择
ADC_INSERTED_CHANNEL_AUTO	注入序列自动转换

ADC_CONTINUOUS_MODE	连续模式选择
输入参数{in}	
newvalue	控制参数
ENABLE	使能
DISABLE	禁能
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* enable ADC scan mode */
```

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

函数 adc_channel_16_to_19

函数 adc_channel_16_to_19 描述见下表:

表 3-13. 函数 adc_channel_16_to_19

函数名称	adc_channel_16_to_19
函数原形	void adc_channel_16_to_19(uint32_t function, ControlStatus newvalue);
功能描述	配置温度传感器、内部参考电压通道、VBAT通道或VSLCD通道
先决条件	-
被调用函数	-
输入参数{in}	
function	温度传感器、内部参考电压通道、VBAT通道或VSLCD通道功能
ADC_TEMP_CHANNEL_SWITCH	ADC的通道16（温度传感器）开关
ADC_INTERNAL_CHANNEL_SWITCH	ADC的通道17（内部参考电压）开关
ADC_VBAT_CHANNEL_SWITCH	ADC通道18（外部电池的1/3电压）开关
ADC_VSLCD_CHANNEL_SWITCH	ADC的通道19（VSLCD的1/3电压）开关
输入参数{in}	
newvalue	控制参数
ENABLE	使能
DISABLE	禁能
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* enable ADC temperature sensor */  
  
adc_channel_16_to_19(ADC_TEMP_CHANNEL_SWITCH, ENABLE);
```

函数 `adc_data_alignment_config`

函数 `adc_alignment_config` 描述见下表:

表 3-14. 函数 `adc_data_alignment_config`

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

例如:

```
/* configure ADC data alignment */  
  
adc_data_alignment_config(ADC_DATAALIGN_RIGHT);
```

函数 `adc_channel_length_config`

函数 `adc_channel_length_config` 描述见下表:

表 3-15. 函数 `adc_channel_length_config`

函数名称	<code>adc_channel_length_config</code>
函数原形	<code>void adc_channel_length_config(uint8_t adc_sequence, uint32_t length);</code>
功能描述	配置常规序列或注入序列的通道长度
先决条件	-
被调用函数	-
输入参数{in}	
<code>adc_sequence</code>	序列选择
<code>ADC_ROUTINE_CHANNEL</code>	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 channel length of ADC routine sequence */
```

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

函数 adc_routine_channel_config

函数 adc_routine_channel_config 描述见下表：

表 3-16. 函数 adc_routine_channel_config

函数名称	adc_routine_channel_config
函数原形	void adc_routine_channel_config(uint8_t rank, uint8_t adc_channel, uint32_t sample_time);
功能描述	配置ADC常规序列通道
先决条件	-
被调用函数	-
输入参数{in}	
rank	常规序列通道顺序，取值范围为0~15
输入参数{in}	
adc_channel	ADC通道选择
ADC_CHANNEL_x	ADC通道x（x=0..19）
输入参数{in}	
sample_time	采样时间
ADC_SAMPLETIME_2POINT5	2.5周期
ADC_SAMPLETIME_7POINT5	7.5周期
ADC_SAMPLETIME_13POINT5	13.5周期
ADC_SAMPLETIME_28POINT5	28.5周期
ADC_SAMPLETIME_41POINT5	41.5周期
ADC_SAMPLETIME_55POINT5	55.5周期

ADC_SAMPLETIME_71POINT5	71.5周期
ADC_SAMPLETIME_239POINT5	239.5周期
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* configure ADC routine channel */
```

```
adc_routine_channel_config(1, ADC_CHANNEL_0, ADC_SAMPLETIME_7POINT5);
```

函数 adc_inserted_channel_config

函数 adc_inserted_channel_config 描述见下表：

表 3-17. 函数 adc_inserted_channel_config

函数名称	adc_inserted_channel_config
函数原形	void adc_inserted_channel_config(uint8_t rank, uint8_t adc_channel, uint32_t sample_time);
功能描述	配置ADC注入序列通道
先决条件	-
被调用函数	-
输入参数{in}	
rank	注入序列通道顺序，取值范围为0~3
输入参数{in}	
channel	ADC通道选择
ADC_CHANNEL_x	ADC通道x（x=0..19）
输入参数{in}	
sample_time	采样时间
ADC_SAMPLETIME_2POINT5	2.5周期
ADC_SAMPLETIME_7POINT5	7.5周期
ADC_SAMPLETIME_13POINT5	13.5周期
ADC_SAMPLETIME_28POINT5	28.5周期
ADC_SAMPLETIME_41POINT5	41.5周期
ADC_SAMPLETIME_55POINT5	55.5周期

ADC_SAMPLETIME_71POINT5	71.5周期
ADC_SAMPLETIME_239POINT5	239.5周期
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* configure ADC inserted channel */
adc_inserted_channel_config(1, ADC_CHANNEL_0, ADC_SAMPLETIME_7POINT5);
```

函数 adc_inserted_channel_offset_config

函数 adc_inserted_channel_offset_config 描述见下表：

表 3-18. 函数 adc_inserted_channel_offset_config

函数名称	adc_inserted_channel_offset_config
函数原形	void adc_inserted_channel_offset_config(uint8_t inserted_channel, uint16_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}	
inserted_channel	注入通道选择
ADC_INSERTED_CHANNEL_x	注入通道，x=0,1,2,3
输入参数{in}	
offset	数据偏移值
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* configure ADC inserted channel offset */
adc_inserted_channel_offset_config(ADC_INSERTED_CHANNEL_0, 100);
```

函数 adc_channel_differential_mode_config（只对 GD32L235 有效）

函数 adc_channel_differential_mode_config 描述见下表：

表 3-19. 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_channel, ControlStatus newvalue);</code>
功能描述	配置ADC通道差分模式
先决条件	-
被调用函数	-
输入参数{in}	
adc_channel	使用差分模式的通道
<code>ADC_DIFFERENTIAL_MODE_CHANNEL_x</code>	ADC通道x (x=0..14)
<code>ADC_DIFFERENTIAL_MODE_CHANNEL_ALL</code>	ADC通道0~14
输入参数{in}	
newvalue	控制参数
<code>ENABLE</code>	使能
<code>DISABLE</code>	禁能
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* enable ADC channel0 differential mode */
```

```
adc_channel_differential_mode_config(ADC_DIFFERENTIAL_MODE_CHANNEL_0, ENABLE);
```

函数 `adc_external_trigger_config`

函数 `adc_external_trigger_config` 描述见下表：

表 3-20. 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(uint8_t adc_sequence, ControlStatus newvalue);</code>
功能描述	配置ADC外部触发
先决条件	-
被调用函数	-
输入参数{in}	
adc_sequence	序列选择
<code>ADC_ROUTINE_CHANNEL</code>	常规序列
<code>ADC_INSERTED_CHANNEL</code>	注入序列

ANNE	
输入参数{in}	
newvalue	控制参数
ENABLE	使能
DISABLE	禁能
输出参数{out}	
-	-
返回值	
-	-

例如：

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

```
adc_external_trigger_config(ADC_INSERTED_CHANNEL, ENABLE);
```

函数 adc_external_trigger_source_config

函数 adc_external_trigger_source_config 描述见下表：

表 3-21. 函数 adc_external_trigger_source_config

函数名称	adc_external_trigger_source_config
函数原形	void adc_external_trigger_source_config(uint8_t adc_sequence, uint32_t external_trigger_source);
功能描述	配置ADC外部触发源
先决条件	-
被调用函数	-
输入参数{in}	
adc_sequence	序列选择
ADC_ROUTINE_CHANNEL	常规序列
ADC_INSERTED_CHANNEL	注入序列
输入参数{in}	
external_trigger_source	常规序列或注入序列触发源
ADC_EXTTRIG_ROUTINE_T8_CH0	TIMER8 CH0事件（常规序列）
ADC_EXTTRIG_ROUTINE_T8_CH1	TIMER8 CH1事件（常规序列）
ADC_EXTTRIG_ROUTINE_T0_CH2	TIMER0 CH2事件(对于 GD32L235xx 产品的常规序列)
ADC_EXTTRIG_ROUTINE_T1_CH1	TIMER1 CH1事件（常规序列）
ADC_EXTTRIG_ROUTINE_T2_TRGO	TIMER2 TRGO事件（常规序列）

TINE_T2_TRGO	
ADC_EXTTRIG_ROU TINE_T11_CH0	TIMER11 CH0事件（常规序列）
ADC_EXTTRIG_ROU TINE_EXTI_11	外部中断线11（常规序列）
ADC_EXTTRIG_ROU TINE_NONE	软件触发（常规序列）
ADC_EXTTRIG_INSE RTED_T0_TRGO	TIMER0 TRGO事件(对于GD32L235xx 产品的注入序列)
ADC_EXTTRIG_INSE RTED_T0_CH3	TIMER0 CH3事件(对于GD32L235xx 产品的注入序列)
ADC_EXTTRIG_INSE RTED_T14_TRGO	TIMER14 TRGO事件(对于GD32L235xx 产品的注入序列)
ADC_EXTTRIG_INSE RTED_T1_TRGO	TIMER1 TRGO事件（注入序列）
ADC_EXTTRIG_INSE RTED_T1_CH0	TIMER1 CH0事件（注入序列）
ADC_EXTTRIG_INSE RTED_T2_CH3	TIMER2 CH3事件（注入序列）
ADC_EXTTRIG_INSE RTED_EXTI_15	外部中断线15（注入序列）
ADC_EXTTRIG_INSE RTED_NONE	软件触发（注入序列）
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* configure ADC routine sequence external trigger source */
```

```
adc_external_trigger_source_config(ADC_ROUTINE_CHANNEL,  
ADC_EXTTRIG_ROUTINE_T8_CH0);
```

函数 adc_software_trigger_enable

函数 adc_software_trigger_enable 描述见下表：

表 3-22. 函数 adc_software_trigger_enable

函数名称	adc_software_trigger_enable
函数原形	void adc_software_trigger_enable(uint8_t adc_sequence);
功能描述	使能ADC软件触发
先决条件	-
被调用函数	-

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

例如:

```
/* enable ADC routine sequence software trigger */
adc_software_trigger_enable(ADC_ROUTINE_CHANNEL);
```

函数 adc_routine_data_read

函数 adc_routine_data_read 描述见下表:

表 3-23. 函数 adc_routine_data_read

函数名称	adc_routine_data_read
函数原形	uint16_t adc_routine_data_read(void);
功能描述	读ADC常规序列数据寄存器
先决条件	-
被调用函数	-
输出参数{out}	
-	-
返回值	
uint16_t	ADC转换值: 0~0xFFFF

例如:

```
/* read ADC routine sequence data register */
uint16_t adc_value = 0;
adc_value = adc_routine_data_read();
```

函数 adc_inserted_data_read

函数 adc_inserted_data_read 描述见下表:

表 3-24. 函数 adc_inserted_data_read

函数名称	adc_inserted_data_read
函数原形	uint16_t adc_inserted_data_read(uint8_t inserted_channel);
功能描述	读ADC注入序列数据寄存器

先决条件	-
被调用函数	-
输入参数{in}	
inserted_channel	注入通道选择
ADC_INSERTED_CHANNEL_x	注入通道x, x=0,1,2,3
输出参数{out}	
-	-
返回值	
uint16_t	ADC转换值: 0~0xFFFF

例如:

```
/* read ADC inserted sequence data register */
```

```
uint16_t adc_value = 0;
```

```
adc_value = adc_inserted_data_read(ADC_INSERTED_CHANNEL_0);
```

函数 adc_watchdog_single_channel_enable

函数 adc_watchdog_single_channel_enable 描述见下表:

表 3-25. 函数 adc_watchdog_single_channel_enable

函数名称	adc_watchdog_single_channel_enable
函数原形	void adc_watchdog_single_channel_enable(uint8_t adc_channel);
功能描述	使能ADC模拟看门狗单通道功能
先决条件	-
被调用函数	-
输入参数{in}	
adc_channel	选择ADC通道
ADC_CHANNEL_x	ADC Channelx(x=0..19)
输出参数{out}	
-	-
返回值	
-	-

例如:

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

```
adc_watchdog_single_channel_enable(ADC_CHANNEL_1);
```

函数 adc_watchdog_sequence_channel_enable

函数 adc_watchdog_sequence_channel_enable 描述见下表:

表 3-26. 函数 `adc_watchdog_sequence_channel_enable`

函数名称	<code>adc_watchdog_sequence_channel_enable</code>
函数原形	<code>void adc_watchdog_sequence_channel_enable(uint8_t adc_sequence);</code>
功能描述	使能ADC模拟看门狗序列通道功能
先决条件	-
被调用函数	-
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_ROUTINE_INSERTED_CHANNEL</code>	常规和注入序列
输出参数{out}	
-	-
返回值	
-	-

例如：

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

```
adc_watchdog_sequence_channel_enable(ADC_ROUTINE_CHANNEL);
```

函数 `adc_watchdog_disable`

函数 `adc_watchdog_disable` 描述见下表：

表 3-27. 函数 `adc_watchdog_disable`

函数名称	<code>adc_watchdog_disable</code>
函数原形	<code>void adc_watchdog_disable(void);</code>
功能描述	禁能ADC模拟看门狗
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 ADC analog watchdog */
```

```
adc_watchdog_disable();
```


函数 adc_watchdog_threshold_config

函数 adc_watchdog_threshold_config 描述见下表:

表 3-28. 函数 adc_watchdog_threshold_config

函数名称	adc_watchdog_threshold_config
函数原形	void adc_watchdog_threshold_config(uint16_t low_threshold, uint16_t high_threshold);
功能描述	配置ADC模拟看门狗阈值
先决条件	-
被调用函数	-
输入参数{in}	
low_threshold	模拟看门狗低阈值, 0..4095
输入参数{in}	
high_threshold	模拟看门狗高阈值, 0..4095
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* configure ADC analog watchdog threshold */
```

```
adc_watchdog_threshold_config(0x0400, 0x0A00);
```

函数 adc_resolution_config

函数 adc_resolution_config 描述见下表:

表 3-29. 函数 adc_resolution_config

函数名称	adc_resolution_config
函数原形	void adc_resolution_config(uint32_t resolution);
功能描述	配置ADC分辨率
先决条件	-
被调用函数	-
输入参数{in}	
resolution	ADC分辨率
ADC_RESOLUTION_12B	12位分辨率
ADC_RESOLUTION_10B	10位分辨率
ADC_RESOLUTION_8B	8位分辨率
ADC_RESOLUTION_6B	6位分辨率

_6B	
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* configure ADC resolution */
```

```
adc_resolution_config(ADC_RESOLUTION_12B);
```

函数 adc_oversample_mode_config

函数 adc_oversample_mode_config 描述见下表：

表 3-30. 函数 adc_oversample_mode_config

函数名称	adc_oversample_mode_config
函数原形	void adc_oversample_mode_config(uint32_t mode, uint16_t shift, uint8_t ratio);
功能描述	配置ADC过采样模式
先决条件	-
被调用函数	-
输入参数{in}	
mode	ADC过采样触发模式
ADC_OVERSAMPLING_ALL_CONVERT	在一个触发之后，对一个通道连续进行过采样转换
ADC_OVERSAMPLING_ONE_CONVERT	在一个触发之后，对一个通道只进行一次过采样转换
输入参数{in}	
shift	ADC过滤采样移位
ADC_OVERSAMPLING_SHIFT_NONE	不移位
ADC_OVERSAMPLING_SHIFT_1B	移1位
ADC_OVERSAMPLING_SHIFT_2B	移2位
ADC_OVERSAMPLING_SHIFT_3B	移3位
ADC_OVERSAMPLING_SHIFT_4B	移4位

MPLING_SHIFT _4B	
ADC_OVERSA MPLING_SHIFT _5B	移5位
ADC_OVERSA MPLING_SHIFT _6B	移6位
ADC_OVERSA MPLING_SHIFT _7B	移7位
ADC_OVERSA MPLING_SHIFT _8B	移8位
输入参数{in}	
ratio	ADC过采样率
ADC_OVERSA MPLING_RATIO _MUL2	2x
ADC_OVERSA MPLING_RATIO _MUL4	4x
ADC_OVERSA MPLING_RATIO _MUL8	8x
ADC_OVERSA MPLING_RATIO _MUL16	16x
ADC_OVERSA MPLING_RATIO _MUL32	32x
ADC_OVERSA MPLING_RATIO _MUL64	64x
ADC_OVERSA MPLING_RATIO _MUL128	128x
ADC_OVERSA MPLING_RATIO _MUL256	256x
输出参数{out}	
-	-
返回值	

-	-
---	---

例如：

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

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

函数 `adc_oversample_mode_enable`

函数 `adc_oversample_mode_enable` 描述见下表：

表 3-31. 函数 `adc_oversample_mode_enable`

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

例如：

```
/* enable ADC oversample mode */
```

```
adc_oversample_mode_enable();
```

函数 `adc_oversample_mode_disable`

函数 `adc_oversample_mode_disable` 描述见下表：

表 3-32. 函数 `adc_oversample_mode_disable`

函数名称	<code>adc_oversample_mode_disable</code>
函数原形	<code>void adc_oversample_mode_disable(void);</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}	
-	-
输出参数{out}	
-	-
返回值	

-	-
---	---

例如：

```
/* disable ADC oversample mode */
```

```
adc_oversample_mode_disable();
```

函数 `adc_charge_pulse_width_counter`

函数 `adc_charge_pulse_width_counter` 描述见下表：

表 3-33. 函数 `adc_charge_pulse_width_counter`

函数名称	<code>adc_charge_pulse_width_counter</code>
函数原形	<code>void adc_charge_pulse_width_counter(uint32_t value);</code>
功能描述	配置ADC充电脉冲宽度计数器
先决条件	-
被调用函数	-
输入参数{in}	
value	ADC充电脉冲宽度计数器值
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* configure ADC charge pulse width counter */
```

```
adc_charge_pulse_width_counter(100);
```

函数 `adc_charge_flag_get`

函数 `adc_charge_flag_get` 描述见下表：

表 3-34. 函数 `adc_charge_flag_get`

函数名称	<code>adc_charge_flag_get</code>
函数原形	<code>FlagStatus adc_charge_flag_get(uint32_t flag);</code>
功能描述	获取ADC充电状态标志位
先决条件	-
被调用函数	-
输入参数{in}	
value	ADC标志位
<code>ADC_FLAG_CHAR GE</code>	ADC充电状态标志位
输出参数{out}	
-	-

返回值	
FlagStatus	SET或RESET

例如：

```
/* get the ADC charge flag */
```

```
FlagStatus flag_value;
```

```
flag_value = adc_charge_flag_get(ADC_FLAG_CHARGE);
```

函数 adc_flag_get

函数 adc_flag_get 描述见下表：

表 3-35. 函数 adc_flag_get

函数名称	adc_flag_get
函数原形	FlagStatus adc_flag_get(uint32_t flag);
功能描述	获取ADC标志位
先决条件	-
被调用函数	-
输入参数{in}	
flag	ADC标志位
ADC_FLAG_WDE	模拟看门狗事件标志位
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
输出参数{out}	
-	-
返回值	
FlagStatus	SET或RESET

例如：

```
/* get the ADC analog watchdog flag */
```

```
FlagStatus flag_value;
```

```
flag_value = adc_flag_get(ADC_FLAG_WDE);
```

函数 adc_flag_clear

函数 adc_flag_clear 描述见下表：

表 3-36. 函数 adc_flag_clear

函数名称	adc_flag_clear
函数原形	void adc_flag_clear(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}	
flag	ADC标志位
ADC_FLAG_WDE	模拟看门狗事件标志位
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
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* clear the ADC analog watchdog flag */
```

```
adc_flag_clear(ADC_FLAG_WDE);
```

函数 adc_interrupt_enable

函数 adc_interrupt_enable 描述见下表：

表 3-37. 函数 adc_interrupt_enable

函数名称	adc_interrupt_enable
函数原形	void adc_interrupt_enable(uint32_t interrupt);
功能描述	使能ADC中断
先决条件	-
被调用函数	-
输入参数{in}	
interrupt	ADC中断
ADC_INT_WDE	模拟看门狗中断
ADC_INT_EOC	序列转换结束中断
ADC_INT_EOIC	注入序列转换结束中断
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* enable ADC analog watchdog interrupt */
```

```
adc_interrupt_enable(ADC_INT_WDE);
```

函数 adc_interrupt_disable

函数 adc_interrupt_disable 描述见下表:

表 3-38. 函数 adc_interrupt_disable

函数名称	adc_interrupt_disable
函数原形	void adc_interrupt_disable(uint32_t interrupt);
功能描述	禁用ADC中断
先决条件	-
被调用函数	-
输入参数{in}	
interrupt	ADC中断
ADC_INT_WDE	模拟看门狗中断
ADC_INT_EOC	序列转换结束中断
ADC_INT_EOIC	注入序列转换结束中断
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* disable ADC analog watchdog interrupt */
```

```
adc_interrupt_disable(ADC_INT_WDE);
```

函数 adc_interrupt_flag_get

函数 adc_interrupt_flag_get 描述见下表:

表 3-39. 函数 adc_interrupt_flag_get

函数名称	adc_interrupt_flag_get
函数原形	FlagStatus adc_interrupt_flag_get(uint32_t int_flag);
功能描述	获取ADC中断标志位
先决条件	-
被调用函数	-
输入参数{in}	
int_flag	ADC中断标志位
ADC_INT_FLAG_WDE	模拟看门狗中断标志位
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
输出参数{out}	
-	-

返回值	
FlagStatus	SET或RESET

例如：

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

```
FlagStatus flag_value;
```

```
flag_value = adc_interrupt_flag_get(ADC_INT_FLAG_WDE);
```

函数 `adc_interrupt_flag_clear`

函数 `adc_interrupt_flag_clear` 描述见下表：

表 3-40. 函数 `adc_interrupt_flag_clear`

函数名称	<code>adc_interrupt_flag_clear</code>
函数原形	<code>void adc_interrupt_flag_clear(uint32_t int_flag);</code>
功能描述	清除ADC中断标志位
先决条件	-
被调用函数	-
输入参数{in}	
<code>int_flag</code>	ADC中断标志位
<code>ADC_INT_FLAG_WDE</code>	模拟看门狗中断标志位
<code>ADC_INT_FLAG_EOC</code>	序列转换结束中断标志位
<code>ADC_INT_FLAG_EOIC</code>	注入序列转换结束中断标志位
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* clear the ADC analog watchdog interrupt flag */
```

```
adc_interrupt_flag_clear(ADC_INT_FLAG_WDE);
```

3.3. CAN（仅适用于 GD32L235xx 系列）

CAN（Controller Area Network）总线是一种可以在无主机情况下实现微处理器或者设备之间相互通信的总线标准。章节 [3.3.1](#) 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 3-41. CAN 寄存器

寄存器名称	寄存器描述
CAN_CTL	控制寄存器
CAN_STAT	状态寄存器
CAN_TSTAT	发送状态寄存器
CAN_RFIFO0	接收FIFO0寄存器
CAN_RFIFO1	接收FIFO1寄存器
CAN_INTEN	中断使能寄存器
CAN_ERR	错误寄存器
CAN_BT	位时序寄存器
CAN_TMIx	发送邮箱标识符寄存器
CAN_TMPx	发送邮箱属性寄存器
CAN_TMDATA0x	发送邮箱data0寄存器
CAN_TMDATA1x	发送邮箱data1寄存器
CAN_RFIFOMIx	接收FIFO邮箱标识符寄存器
CAN_RFIFOMPx	接收FIFO邮箱属性寄存器
CAN_RFIFOMDAT A0x	接收FIFO邮箱data0寄存器
CAN_RFIFOMDAT A1x	接收FIFO邮箱data1寄存器
CAN_FCTL	过滤器控制寄存器
CAN_FMCFG	过滤器模式配置寄存器
CAN_FSCFG	过滤器位宽配置寄存器
CAN_FAFIFO	过滤器关联FIFO寄存器
CAN_FW	过滤器激活寄存器
CAN_FxDATAy	过滤器(x)数据(y)寄存器

3.3.2. 外设库函数说明

CAN库函数列表如下表所示：

表 3-42. CAN 库函数

库函数名称	库函数描述
can_deinit	复位外设CAN
can_struct_para_init	初始化结构体
can_init	初始化外设CAN
can_filter_init	CAN过滤器初始化
can_debug_freeze_enable	CAN调试冻结使能
can_debug_freeze_disable	CAN调试冻结关闭

库函数名称	库函数描述
can_time_trigger_mode_enable	CAN时间触发模式使能
can_time_trigger_mode_disable	CAN时间触发模式关闭
can_message_transmit	CAN传输报文
can_transmit_states	获取CAN传输状态
can_transmission_stop	CAN邮箱停止发送
can_message_receive	CAN接收报文
can_fifo_release	CAN释放FIFO
can_receive_message_length_get	获取CAN接收帧的数量
can_working_mode_set	CAN工作模式设置
can_wakeup	从睡眠模式中唤醒CAN
can_error_get	获取CAN总线错误
can_receive_error_number_get	获取CAN接收错误
can_transmit_error_number_get	获取CAN发送错误
can_flag_get	获取CAN标志位状态
can_flag_clear	清除CAN标志位状态
can_interrupt_enable	CAN中断使能
can_interrupt_disable	CAN中断关闭
can_interrupt_flag_get	获取CAN中断标志位状态
can_interrupt_flag_clear	清除CAN中断标志位状态

结构体 can_parameter_struct

表 3-43. 结构体 can_parameter_struct

成员名称	功能描述
working_mode	工作模式
resync_jump_width	再同步补偿宽度
time_segment_1	位段1
time_segment_2	位段2
time_triggered	时间触发通信模式
auto_bus_off_recovery	自动离线恢复
auto_wake_up	自动唤醒
auto_retrans	自动重传
rec_fifo_overwrite	接收FIFO满时覆盖
trans_fifo_order	发送FIFO顺序
prescaler	波特率分频系数

结构体 can_transmit_message_struct

表 3-44. 结构体 can_transmit_message_struct

成员名称	功能描述
tx_sfid	标准格式帧标识符

成员名称	功能描述
tx_efid	扩展格式帧标识符
tx_ff	帧格式：标准格式/扩展格式
tx_ft	帧类型：数据帧/远程帧
tx_dlen	数据长度
reserved	为了对齐而保留的字节
tx_data[8]	数据值

结构体 can_receive_message_struct

表 3-45. 结构体 can_receive_message_struct

成员名称	功能描述
rx_sfid	标准格式帧标识符
rx_efid	扩展格式帧标识符
rx_ff	帧格式：标准格式/扩展格式
rx_ft	帧类型：数据帧/远程帧
rx_dlen	数据长度
rx_fi	过滤器索引
rx_data[8]	数据值

结构体 can_filter_parameter_struct

表 3-46. 结构体 can_filter_parameter_struct

成员名称	功能描述
filter_list_high	过滤器列表数高位
filter_list_low	过滤器列表数低位
filter_mask_high	过滤器掩码数高位
filter_mask_low	过滤器掩码数低位
filter_fifo_number	接收FIFO编号
filter_number	过滤器索引号
filter_mode	过滤模式：列表模式/掩码模式
filter_bits	过滤器位宽
filter_enable	过滤器是否工作

函数 can_deinit

函数can_deinit描述见下表：

表 3-47. 函数 can_deinit

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

输入参数{in}	
-	-
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* CAN deinitialize*/
```

```
can_deinit ();
```

函数 can_struct_para_init

函数can_struct_para_init描述见下表:

表 3-48. 函数 can_struct_para_init

函数名称	can_struct_para_init
函数原型	void can_struct_para_init(can_struct_type_enum type, void* p_struct)
功能描述	CAN外设库使用到的各类结构体初始化
先决条件	-
被调用函数	-
输入参数{in}	
type	需要初始化的结构体类型，仅可选择唯一参数
CAN_INIT_STRUC T	初始化结构体
CAN_FILTER_STR UCT	过滤器初始化结构体
CAN_TX_MESSAG E_STRUCT	存储发送帧结构体
CAN_RX_MESSAG E_STRUCT	接收帧结构体
输出参数{out}	
p_struct	对应的需要初始化的结构体指针
返回值	
-	-

例如:

```
can_parameter_struct can_init;
```

```
can_struct_para_init (CAN_INIT_STRUCT, &can_init);
```

函数 can_init

函数can_init描述见下表:

表 3-49. 函数 can_init

函数名称	can_init
函数原型	ErrStatus can_init(can_parameter_struct* can_parameter_init);
功能描述	初始化外设CAN
先决条件	can_struct_para_init()
被调用函数	-
输入参数{in}	
can_parameter_init	初始化结构体，结构体成员参考 表 3-43. 结构体can_parameter_struct
输出参数{out}	
-	-
返回值	
ErrStatus	SUCCESS / ERROR

例如：

```
can_parameter_struct can_parameter;

/* CAN initialize*/

can_init (&can_parameter);
```

函数 can_filter_init

函数can_filter_init描述见下表：

表 3-50. 函数 can_filter_init

函数名称	can_filter_init
函数原型	void can_filter_init(can_filter_parameter_struct* can_filter_parameter_init);
功能描述	CAN过滤器初始化
先决条件	can_struct_para_init()
被调用函数	-
输入参数{in}	
can_filter_parameter_init	过滤器初始化结构体，结构体成员参考 表 3-46. 结构体can_filter_parameter_struct
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* initialize CAN filter */

can_filter_init(&can_filter);
```

函数 can_debug_freeze_enable

函数can_debug_freeze_enable描述见下表：

表 3-51. 函数 can_debug_freeze_enable

函数名称	can_debug_freeze_enable
函数原型	void can_debug_freeze_enable(void);
功能描述	CAN调试冻结使能
先决条件	-
被调用函数	dbg_periph_enable
输入参数{in}	
-	-
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* enable CAN debug freeze */
```

```
can_debug_freeze_enable ();
```

函数 can_debug_freeze_disable

函数can_debug_freeze_disable描述见下表：

表 3-52. 函数 can_debug_freeze_disable

函数名称	can_debug_freeze_disable
函数原型	void can_debug_freeze_disable(void);
功能描述	CAN调试冻结关闭
先决条件	-
被调用函数	dbg_periph_disable
输入参数{in}	
-	-
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* disable CAN debug freeze */
```

```
can_debug_freeze_disable ();
```

函数 can_time_trigger_mode_enable

函数can_time_trigger_mode_enable描述见下表：

表 3-53. 函数 can_time_trigger_mode_enable

函数名称	can_time_trigger_mode_enable
函数原型	void can_time_trigger_mode_enable(void);
功能描述	CAN时间触发模式使能
先决条件	-
被调用函数	-
输入参数{in}	
-	-
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* enable CAN time trigger mode */
can_time_trigger_mode_enable ();
```

函数 can_time_trigger_mode_disable

函数can_time_trigger_mode_disable描述见下表：

表 3-54. 函数 can_time_trigger_mode_disable

函数名称	can_time_trigger_mode_disable
函数原型	void can_time_trigger_mode_disable(void);
功能描述	CAN时间触发模式关闭
先决条件	-
被调用函数	-
输入参数{in}	
-	-
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* disable CAN time trigger mode */
can_time_trigger_mode_disable ();
```


函数 `can_message_transmit`

函数 `can_message_transmit` 描述见下表：

表 3-55. 函数 `can_message_transmit`

函数名称	<code>can_message_transmit</code>
函数原型	<code>uint8_t can_message_transmit(can_transmit_message_struct* transmit_message);</code>
功能描述	CAN 传输报文
先决条件	<code>can_struct_para_init()</code>
被调用函数	-
输入参数{in}	
<code>transmit_message</code>	报文发送结构体，结构体成员参考 表 3-44. 结构体 <code>can_transmit_message_struct</code>
输出参数{out}	
-	-
返回值	
<code>uint8_t</code>	0x00-0x03

例如：

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

函数 `can_transmit_states`

函数 `can_transmit_states` 描述见下表：

表 3-56. 函数 `can_transmit_states`

函数名称	<code>can_transmit_states</code>
函数原型	<code>can_transmit_state_enum can_transmit_states(uint8_t mailbox_number);</code>
功能描述	获取 CAN 传输状态
先决条件	-
被调用函数	-
输入参数{in}	
<code>mailbox_number</code>	邮箱标号
<code>CAN_MAILBOXx</code>	<code>CAN_MAILBOXx(x=0,1,2)</code>
输出参数{out}	
-	-
返回值	
<code>can_transmit_state_enum</code>	0..4

例如：

```
/* CAN mailbox0 transmit state */
```

```
uint8_t transmit_state = 0;
```

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

函数 can_transmission_stop

函数can_transmission_stop描述见下表:

表 3-57. 函数 can_transmission_stop

函数名称	can_transmission_stop
函数原型	void can_transmission_stop(uint8_t mailbox_number);
功能描述	CAN邮箱停止发送
先决条件	-
被调用函数	-
输入参数{in}	
mailbox_number	邮箱标号
CAN_MAILBOXx	CAN_MAILBOXx(x=0,1,2)
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* stop CAN mailbox0 transmission */
```

```
can_transmission_stop (CAN_MAILBOX0);
```

函数 can_message_receive

函数can_message_receive描述见下表:

表 3-58. 函数 can_message_receive

函数名称	can_message_receive
函数原型	void can_message_receive(uint8_t fifo_number, can_receive_message_struct* receive_message);
功能描述	CAN接收报文
先决条件	can_struct_para_init()
被调用函数	-
输入参数{in}	
fifo_number	FIFO编号
CAN_FIFOx	CAN_FIFOx(x=0,1)
输入参数{in}	
receive_message	接收报文结构体, 结构体成员参考 表 3-45. 结构体 can_receive_message_struct

输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* CAN FIFO0 receive message */
```

```
can_message_receive(CAN_FIFO0, &receive_message);
```

函数 can_fifo_release

函数can_fifo_release描述见下表：

表 3-59. 函数 can_fifo_release

函数名称	can_fifo_release
函数原型	void can_fifo_release(uint8_t fifo_number);
功能描述	CAN释放FIFO
先决条件	-
被调用函数	-
输入参数{in}	
fifo_number	FIFO编号
CAN_FIFOx	CAN_FIFOx(x=0,1)
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* CAN release FIFO0 */
```

```
can_fifo_release (CAN_FIFO0);
```

函数 can_receive_message_length_get

函数can_receive_message_length_get描述见下表：

表 3-60. 函数 can_receive_message_length_get

函数名称	can_receive_message_length_get
函数原型	uint8_t can_receive_message_length_get(uint8_t fifo_number);
功能描述	获取CAN接收帧的数量
先决条件	-
被调用函数	-
输入参数{in}	
fifo_number	FIFO编号

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

例如:

```
/* CAN FIFO0 receive message length */
uint8_t frame_number = 0;
frame_number = can_receive_message_length_get (CAN_FIFO0);
```

函数 can_working_mode_set

函数can_working_mode_set描述见下表:

表 3-61. 函数 can_working_mode_set

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

例如:

```
/* set CAN working at initialize mode */
can_working_mode_set (CAN_MODE_INITIALIZE);
```

函数 can_wakeup

函数can_wakeup描述见下表:

表 3-62. 函数 can_wakeup

函数名称	can_wakeup
函数原型	ErrStatus can_wakeup(void);
功能描述	从睡眠模式中唤醒CAN
先决条件	-
被调用函数	-
输入参数{in}	
-	-
输出参数{out}	
-	-
返回值	
ErrStatus	SUCCESS / ERROR

例如：

```
/* wake up CAN */
can_wakeup ();
```

函数 can_error_get

函数can_error_get描述见下表：

表 3-63. 函数 can_error_get

函数名称	can_error_get
函数原型	can_error_enum can_error_get(void);
功能描述	获取CAN总线错误
先决条件	-
被调用函数	-
输入参数{in}	
-	-
输出参数{out}	
-	-
返回值	
can_error_enum	0..7

例如：

```
/* get CAN error type */
can_error_enum err_type;
err_type = can_error_get ();
```

函数 can_receive_error_number_get

函数can_receive_error_number_get描述见下表：

表 3-64. 函数 can_receive_error_number_get

函数名称	can_receive_error_number_get
函数原型	uint8_t can_receive_error_number_get(void);
功能描述	获取CAN接收错误
先决条件	-
被调用函数	-
输入参数{in}	
-	-
输出参数{out}	
-	-
返回值	
uint8_t	0..255

例如:

```
/* get CAN receive error number */
```

```
uint8_t error_num;
```

```
error_num = can_receive_error_number_get ();
```

函数 can_transmit_error_number_get

函数can_transmit_error_number_get描述见下表:

表 3-65. 函数 can_transmit_error_number_get

函数名称	can_transmit_error_number_get
函数原型	uint8_t can_transmit_error_number_get(void);
功能描述	获取CAN发送错误
先决条件	-
被调用函数	-
输入参数{in}	
-	-
输出参数{out}	
-	-
返回值	
uint8_t	0..255

例如:

```
/* get CAN transmit error number */
```

```
uint8_t error_num;
```

```
error_num = can_transmit_error_number_get ();
```

函数 can_flag_get

函数can_flag_get描述见下表:

表 3-66. 函数 can_flag_get

函数名称	can_flag_get
函数原型	FlagStatus can_flag_get(can_flag_enum flag);
功能描述	获取CAN标志位状态
先决条件	-
被调用函数	-
输入参数{in}	
flag	CAN 标志位
CAN_FLAG_MTE2	邮箱2发送错误
CAN_FLAG_MTE1	邮箱1发送错误
CAN_FLAG_MTE0	邮箱0发送错误
CAN_FLAG_MTF2	邮箱2发送完成
CAN_FLAG_MTF1	邮箱1发送完成
CAN_FLAG_MTF0	邮箱0发送完成
CAN_FLAG_RFO0	接收FIFO0溢出
CAN_FLAG_RFF0	接收FIFO0满
CAN_FLAG_RFO1	接收FIFO1溢出
CAN_FLAG_RFF1	接收FIFO1满
CAN_FLAG_BOERR	离线错误
CAN_FLAG_PERR	被动错误
CAN_FLAG_WERR	警告错误
输出参数{out}	
-	-
返回值	
FlagStatus	SET / RESET

例如:

```
/* get CAN mailbox 0 transmit finished flag */
```

```
can_flag_get (CAN_FLAG_MTF0);
```

函数 can_flag_clear

函数can_flag_clear描述见下表:

表 3-67. 函数 can_flag_clear

函数名称	can_flag_clear
函数原型	void can_flag_clear(can_flag_enum flag);
功能描述	清除CAN标志位状态
先决条件	-

被调用函数	-
输入参数{in}	
flag	CAN 标志位
CAN_FLAG_MTE2	邮箱2发送错误
CAN_FLAG_MTE1	邮箱1发送错误
CAN_FLAG_MTE0	邮箱0发送错误
CAN_FLAG_MTF2	邮箱2发送完成
CAN_FLAG_MTF1	邮箱1发送完成
CAN_FLAG_MTF0	邮箱0发送完成
CAN_FLAG_RFO0	接收FIFO0溢出
CAN_FLAG_RFF0	接收FIFO0满
CAN_FLAG_RFO1	接收FIFO1溢出
CAN_FLAG_RFF1	接收FIFO1满
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* clear CAN mailbox 0 transmit error flag */
```

```
can_flag_clear (CAN_FLAG_MTE0);
```

函数 can_interrupt_enable

函数can_interrupt_enable描述见下表:

表 3-68. 函数 can_interrupt_enable

函数名称	can_interrupt_enable
函数原型	void can_interrupt_enable(uint32_t interrupt);
功能描述	CAN中断使能
先决条件	-
被调用函数	-
输入参数{in}	
interrupt	中断类型
CAN_INT_TME	发送邮箱空中断使能
CAN_INT_RFNE0	接收FIFO0非空中断使能
CAN_INT_RFF0	接收FIFO0满中断使能
CAN_INT_RFO0	接收FIFO0溢出中断使能
CAN_INT_RFNE1	接收FIFO1非空中断使能
CAN_INT_RFF1	接收FIFO1满中断使能
CAN_INT_RFO1	接收FIFO1溢出中断使能
CAN_INT_WERR	警告错误中断使能
CAN_INT_PERR	被动错误中断使能

CAN_INT_BO	离线中断使能
CAN_INT_ERRN	错误种类中断使能
CAN_INT_ERR	错误中断使能
CAN_INT_WU	唤醒中断使能
CAN_INT_SLPW	睡眠中断使能
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* CAN transmit mailbox empty interrupt enable */
```

```
can_interrupt_enable (CAN_INT_TME);
```

函数 can_interrupt_disable

函数can_interrupt_disable描述见下表：

表 3-69. 函数 can_interrupt_disable

函数名称	can_interrupt_disable
函数原型	void can_interrupt_disable(uint32_t interrupt);
功能描述	CAN中断关闭
先决条件	-
被调用函数	-
输入参数{in}	
interrupt	中断类型
CAN_INT_TME	发送邮箱空中断使能
CAN_INT_RFNE0	接收FIFO0非空中断使能
CAN_INT_RFF0	接收FIFO0满中断使能
CAN_INT_RFO0	接收FIFO0溢出中断使能
CAN_INT_RFNE1	接收FIFO1非空中断使能
CAN_INT_RFF1	接收FIFO1满中断使能
CAN_INT_RFO1	接收FIFO1溢出中断使能
CAN_INT_WERR	警告错误中断使能
CAN_INT_PERR	被动错误中断使能
CAN_INT_BO	离线中断使能
CAN_INT_ERRN	错误种类中断使能
CAN_INT_ERR	错误中断使能
CAN_INT_WU	唤醒中断使能
CAN_INT_SLPW	睡眠中断使能
输出参数{out}	
-	-
返回值	

-	-
---	---

例如:

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

```
can_interrupt_disable (CAN_INT_TME);
```

函数 can_interrupt_flag_get

函数can_interrupt_flag_get描述见下表:

表 3-70. 函数 can_interrupt_flag_get

函数名称	can_interrupt_flag_get
函数原型	FlagStatus can_interrupt_flag_get(can_interrupt_flag_enum flag);
功能描述	获取CAN中断标志位状态
先决条件	-
被调用函数	-
输入参数{in}	
flag	CAN中断标志位
CAN_INT_FLAG_SLP LPIF	进入睡眠工作模式的状态改变中断标志
CAN_INT_FLAG_WU UIF	从睡眠工作模式唤醒的状态改变中断标志
CAN_INT_FLAG_ER RRIF	错误中断标志
CAN_INT_FLAG_MT TF2	邮箱2发送完成中断标志
CAN_INT_FLAG_MT TF1	邮箱1发送完成中断标志
CAN_INT_FLAG_MT TF0	邮箱0发送完成中断标志
CAN_INT_FLAG_R FO0	接收FIFO0溢出中断标志
CAN_INT_FLAG_R FF0	接收FIFO0满中断标志
CAN_INT_FLAG_R FO1	接收FIFO1溢出中断标志
CAN_INT_FLAG_R FF1	接收FIFO1满中断标志
输出参数{out}	
-	-
返回值	
FlagStatus	SET / RESET

例如:

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

```
can_interrupt_flag_get (CAN_INT_FLAG_MTF0);
```

函数 can_interrupt_flag_clear

函数can_interrupt_flag_clear描述见下表：

表 3-71. 函数 can_interrupt_flag_clear

函数名称	can_interrupt_flag_clear
函数原型	void can_interrupt_flag_clear(can_interrupt_flag_enum flag);
功能描述	清除CAN中断标志位状态
先决条件	-
被调用函数	-
输入参数{in}	
flag	CAN中断标志位
CAN_INT_FLAG_SLP LPIF	进入睡眠工作模式的状态改变中断标志
CAN_INT_FLAG_WU UIF	从睡眠工作模式唤醒的状态改变中断标志
CAN_INT_FLAG_ER RRIF	错误中断标志
CAN_INT_FLAG_MT TF2	邮箱2发送完成中断标志
CAN_INT_FLAG_MT TF1	邮箱1发送完成中断标志
CAN_INT_FLAG_MT TF0	邮箱0发送完成中断标志
CAN_INT_FLAG_R FO0	接收FIFO0溢出中断标志
CAN_INT_FLAG_R FF0	接收FIFO0满中断标志
CAN_INT_FLAG_R FO1	接收FIFO1溢出中断标志
CAN_INT_FLAG_R FF1	接收FIFO1满中断标志
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* clear CAN mailbox 0 transmit finished interrupt flag */
```

```
can_interrupt_flag_clear (CAN_INT_FLAG_MTF0);
```


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 3-72. 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 3-73. CAU 库函数

库函数名称	库函数描述
cau_deinit	复位CAU外设

库函数名称	库函数描述
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 3-74. 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 3-75. 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 3-76. 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 3-77. 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 3-78. 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/ rcu_periph_clock_enable
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 3-79. 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3-77. 结构体cau_parameter_struct
返回值	
-	-

例如：

```
cau_parameter_struct text;

/* initialize CAU encrypt and decrypt parameter struct */

cau_struct_para_init(&text);
```

函数 cau_key_struct_para_init

The description of cau_key_struct_para_init描述见下表：

表 3-80. 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3-74. 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 3-81. 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3-75. 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 3-82. 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3-76. 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 3-83. 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 3-84. 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 3-85. 函数 cau_dma_enable

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

输出参数{out}	
-	-
返回值	
-	-

例如：

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

```
cau_dma_enable(CAU_DMA_INFIFO);
```

函数 cau_dma_disable

函数cau_dma_disable描述见下表：

表 3-86. 函数 cau_dma_disable

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

例如：

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

```
cau_dma_disable(CAU_DMA_INFIFO);
```

函数 cau_init

函数cau_init描述见下表：

表 3-87. 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 3-88. 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 3-89. 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3-74. 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 3-90. 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3-75. 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 3-91. 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 3-92. 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 3-93. 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 3-94. 函数 cau_data_write

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

-	-
返回值	
-	-

例如：

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

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

函数 cau_data_read

函数cau_data_read描述见下表：

表 3-95. 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 3-96. 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3-74. 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3-76. 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 3-97. 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3-76. 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 3-98. 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3-77. 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 3-99. 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3-77. 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 3-100. 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3-77. 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 3-101. 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3-77. 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 3-102. 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3-77. 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 3-103. 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3-77. 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 3-104. 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3-77. 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 3-105. 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3-77. 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 3-106. 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3-77. 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 3-107. 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3-77. 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 3-108. 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3-77. 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 3-109. 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 3-110. 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 3-111. 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 3-112. 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

通用比较器可独立工作，其输出端可用于 I / O 口，也可和定时器结合使用。比较器可通过模拟信号将 MCU 从低功耗模式中唤醒，在一定的条件下，可将模拟信号作为 TIMER 的触发源。

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 3-113. CMP 寄存器

寄存器名称	寄存器描述
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 3-114. 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_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

枚举类型 cmp_enum

表 3-115. 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 3-116. 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	-
输入参数{in}	
cmp_periph	参考枚举 表3-115. 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 3-117. 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3-115. 枚举类型cmp_enum
输入参数{in}	
operating_mode	速度和功耗运行模式
CMP_MODE_HIGHSPEED	高速/全功耗
CMP_MODE_MIDDL SPEED	中速/中功耗
CMP_MODE_LOWSP EED	低速/低功耗
输入参数{in}	
inverting_input	反向输入源选择
CMP_INVERTING_IN PUT_1_4VREFINT	VREFINT *1/4作为输入源
CMP_INVERTING_IN PUT_1_2VREFINT	VREFINT *1/2作为输入源
CMP_INVERTING_IN PUT_3_4VREFINT	VREFINT *3/4作为输入源
CMP_INVERTING_IN PUT_VREFINT	VREFINT作为输入源
CMP_INVERTING_IN PUT_PA0_PA2	PA0作为CMP0输入源或者PA2作为CMP1输入源
CMP_INVERTING_IN PUT_DAC0_OUT0	PA4 (DAC) 作为输入源
CMP_INVERTING_IN PUT_PB3	PB3作为CMP1输入源

输入参数{in}	
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_INVERTING_INPUT_1_4VREF1NT, CMP_HYSTERESIS_NO);
```

函数 cmp_noninverting_input_select

函数cmp_noninverting_input_select描述见下表：

表 3-118. 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正相输入选择
先决条件	-
被调用函数	-
输入参数{in}	
cmp_periph	参考枚举 表3-115. 枚举类型cmp_enum
输入参数{in}	
noninverting_input	正相输入源
CMP_NONINVERTING_INPUT_PA3	PA3作为CMP1输入源
CMP_NONINVERTING_INPUT_PB4	PB4作为CMP1输入源
CMP_NONINVERTING_INPUT_PB5	PB5作为CMP1输入源
CMP_NONINVERTING_INPUT_PB6	PB6作为CMP1输入源

<i>CMP_NONINVERTING_INPUT_PB7</i>	PB7作为CMP1输入源
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* selecte the noninverting input for CMP1*/
```

```
cmp_noninverting_input_select (CMP1, CMP_NONINVERTING_INPUT_PA3);
```

函数 `cmp_output_init`

函数`cmp_output_init`描述见下表:

表 3-119. 函数 `cmp_output_init`

函数名称	<code>cmp_output_init</code>
函数原型	<code>void cmp_output_init(cmp_enum cmp_periph, uint32_t output_selection, uint32_t output_polarity)</code>
功能描述	CMP输出初始化
先决条件	-
被调用函数	-
输入参数{in}	
cmp_periph	参考枚举 表3-115. 枚举类型<code>cmp_enum</code>
输入参数{in}	
output_selction	CMP输出选择
<i>CMP_OUTPUT_NONE</i>	无选择
<i>CMP_OUTPUT_TIMER1_IC3</i>	CMP输出到TIMER1_CH3输入捕获
<i>CMP_OUTPUT_TIMER2_IC0</i>	CMP输出到TIMER2_CH0输入捕获
输入参数{in}	
output_polarity	CMP输出极性
<i>CMP_OUTPUT_POLARITY_INVERTED</i>	输出反相
<i>CMP_OUTPUT_POLARITY_NONINVERTED</i>	输出正相
输出参数{out}	
-	-
返回值	
-	-

例如:

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

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

函数 cmp_blanking_init

函数cmp_blanking_init描述见下表:

表 3-120. 函数 cmp_blanking_init

函数名称	cmp_blanking_init
函数原型	void cmp_blanking_init(cmp_enum cmp_periph, uint32_t blanking_source_selection);
功能描述	CMP消隐功能初始化
先决条件	-
被调用函数	-
输入参数{in}	
cmp_periph	参考枚举 表3-115. 枚举类型cmp_enum
输入参数{in}	
blanking_source_selection	消隐源选择
CMP_BLANKING_NONE	无消隐
CMP_BLANKING_TIMER1_OC1	TIMER1_CH1输出比较信号作为消隐源
CMP_BLANKING_TIMER2_OC1	TIMER2_CH1输出比较信号作为消隐源
CMP_BLANKING_TIMER8_OC1	TIMER8_CH1输出比较信号作为消隐源
CMP_BLANKING_TIMER11_OC1	TIMER11_CH1输出比较信号作为消隐源
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 3-121. 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3-115. 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 3-122. 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3-115. 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 3-123. 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 3-124. 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 3-125. 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3-115. 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 3-126. 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3-115. 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 3-127. 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3-115. 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_voltage_scaler_bridge_enable描述见下表:

表 3-128. 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3-115. 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_voltage_scaler_bridge_disable描述见下表:

表 3-129. 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3-115. 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 3-130. 函数 cmp_output_level_get

函数名称	cmp_output_level_get
函数原型	uint32_t cmp_output_level_get(cmp_enum 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3-115. 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);
```

3.6. CRC

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

3.6.1. 外设寄存器说明

CRC寄存器列表如下表所示：

表 3-131. CRC 寄存器

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

3.6.2. 外设库函数说明

CRC库函数列表如下表所示：

表 3-132. 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	失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计算单个数据
crc_block_data_calculate	CRC计算数组

函数 crc_deinit

函数crc_deinit描述见下表：

表 3-133. 函数 crc_deinit

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

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

例如：

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

```
crc_deinit();
```

函数 `crc_reverse_output_data_enable`

函数 `crc_reverse_output_data_enable` 描述见下表：

表 3-134. 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 3-135. 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	失能输出数据翻转功能

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

例如：

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

```
crc_reverse_output_data_disable();
```

函数 `crc_data_register_reset`

函数 `crc_data_register_reset` 描述见下表：

表 3-136. 函数 `crc_data_register_reset`

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

例如：

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

```
crc_data_register_reset();
```

函数 `crc_data_register_read`

函数 `crc_data_register_read` 描述见下表：

表 3-137. 函数 `crc_data_register_read`

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

被调用函数	-
输入参数{in}	
-	-
输出参数{out}	
-	-
返回值	
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 3-138. 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 3-139. 函数 crc_free_data_register_write

函数名称	crc_free_data_register_write
函数原形	void crc_free_data_register_write(uint8_t free_data);

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

例如：

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

函数 crc_init_data_register_write

函数crc_init_data_register_write描述见下表：

表 3-140. 函数 crc_init_data_register_write

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

例如：

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

函数 crc_input_data_reverse_config

函数crc_input_data_reverse_config描述见下表：

表 3-141. 函数 crc_input_data_reverse_config

函数名称	crc_input_data_reverse_config
函数原形	void crc_input_data_reverse_config(uint32_t data_reverse)
功能描述	配置输入数据翻转功能

先决条件	-
被调用函数	-
输入参数{in}	
data_reverse	设定的输入数据翻转功能
<i>CRC_INPUT_DATA_NOT</i>	输入数据不翻转
<i>CRC_INPUT_DATA_BYTE</i>	输入数据按字节翻转
<i>CRC_INPUT_DATA_HALFWORD</i>	输入数据按半字翻转
<i>CRC_INPUT_DATA_WORD</i>	输入数据按字翻转
输出参数{out}	
-	-
返回值	
-	-

例如：

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

```
crc_input_data_reverse_config(CRC_INPUT_DATA_WORD);
```

函数 **crc_polynomial_size_set**

函数 **crc_polynomial_size_set** 描述见下表：

表 3-142. 函数 **crc_polynomial_size_set**

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

例如：

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

```
crc_polynomial_size_set(CRC_CTL_PS_7);
```

函数 `crc_polynomial_set`

函数 `crc_polynomial_set` 描述见下表：

表 3-143. 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 3-144. 函数 `crc_single_data_calculate`

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

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

例如：

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

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

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

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

函数 crc_block_data_calculate

函数crc_block_data_calculate描述见下表：

表 3-145. 函数 crc_block_data_calculate

函数名称	crc_block_data_calculate
函数原形	uint32_t crc_block_data_calculate(void *array, uint32_t size, uint8_t data_format);
功能描述	CRC计算数组
先决条件	-
被调用函数	-
输入参数{in}	
array	输入数组指针
输入参数{in}	
size	数组大小
输入参数{in}	
data_format	数据格式
INPUT_FORMAT_WORD	输入数据格式为字
INPUT_FORMAT_HALFWORD	输入数据格式为半字
INPUT_FORMAT_BYTE	输入数据格式为字节
输出参数{out}	
-	-
返回值	
uint32_t	CRC计算结果（0-0xFFFFFFFF）

例如：

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

```
#define BUFFER_SIZE    6

uint32_t valcrc = 0;

static const uint32_t data_buffer[BUFFER_SIZE] = {

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

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

3.7. CTC

CTC模块基于外部高精度的参考信号源来校准IRC48M的时钟频率，通过自动的或手动的调整校准值，以得到一个精准的IRC48M时钟。章节[3.7.1](#)描述了CTC的寄存器列表，章节[3.7.2](#)对CTC库函数进行说明。

3.7.1. 外设寄存器说明

CTC寄存器列表如下表所示：

表 3-146. CTC 寄存器

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

3.7.2. 外设库函数说明

CTC库函数列表如下表所示：

表 3-147. CTC 库函数

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

库函数名称	库函数描述
ctc_counter_capture_value_read	读取CTC计数器捕获值
ctc_counter_direction_read	读取CTC校准时钟计数方向
ctc_counter_reload_value_read	读取CTC计数器重载值
ctc_irc48m_trim_value_read	读取IRC48M校准值
ctc_interrupt_enable	CTC中断使能
ctc_interrupt_disable	CTC中断禁能
ctc_interrupt_flag_get	CTC中断标志获取
ctc_interrupt_flag_clear	CTC中断标志清除
ctc_flag_get	CTC状态标志获取
ctc_flag_clear	CTC状态标志清除

函数 ctc_deinit

函数ctc_deinit描述见下表：

表 3-148. 函数 ctc_deinit

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

例如：

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

```
ctc_deinit();
```

函数 ctc_counter_enable

函数ctc_counter_enable描述见下表：

表 3-149. 函数 ctc_counter_enable

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

-	-
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* enable CTC trim counter*/
ctc_counter_enable();
```

函数 ctc_counter_disable

函数ctc_counter_disable描述见下表:

表 3-150. 函数 ctc_counter_disable

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

例如:

```
/* disable CTC trim counter */
ctc_counter_disable();
```

函数 ctc_irc48m_trim_value_config

函数ctc_irc48m_trim_value_config描述见下表:

表 3-151. 函数 ctc_irc48m_trim_value_config

函数名称	ctc_irc48m_trim_value_config
函数原形	void ctc_irc48m_trim_value_config(uint8_t trim_value);
功能描述	配置IRC48M时钟校准值
先决条件	-
被调用函数	-
输入参数{in}	
trim_value	0~127

输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* IRC48M trim value configuration */
ctc_irc48m_trim_value_config(0x01);
```

函数 ctc_software_refsource_pulse_generate

函数ctc_software_refsource_pulse_generate描述见下表：

表 3-152. 函数 ctc_software_refsource_pulse_generate

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

例如：

```
/* generate reference source sync pulse */
ctc_software_refsource_pulse_generate();
```

函数 ctc_hardware_trim_mode_config

函数ctc_hardware_trim_mode_config描述见下表：

表 3-153. 函数 ctc_hardware_trim_mode_config

函数名称	ctc_hardware_trim_mode_config
函数原形	void ctc_hardware_trim_mode_config(uint32_t hardmode);
功能描述	CTC硬件自动校准模式配置
先决条件	-
被调用函数	-
输入参数{in}	
hardmode	硬件校准开启还是关闭
CTC_HARDWARE_	硬件校准开启

TRIM_MODE_ENABLE	
CTC_HARDWARE_TRIM_MODE_DISABLE	硬件校准关闭
输出参数{out}	
-	-
返回值	
-	-

例如：

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

```
ctc_hardware_trim_mode_config(CTC_HARDWARE_TRIM_MODE_ENABLE);
```

函数 ctc_refsource_polarity_config

函数ctc_refsource_polarity_config描述见下表：

表 3-154. 函数 ctc_refsource_polarity_config

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

例如：

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

```
ctc_refsource_polarity_config(CTC_REFSOURCE_POLARITY_RISING);
```


函数 ctc_refsource_signal_select

函数ctc_refsource_signal_select描述见下表:

表 3-155. 函数 ctc_refsource_signal_select

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

例如:

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

```
ctc_refsource_signal_select(CTC_REFSOURCE_LXTAL);
```

函数 ctc_refsource_prescaler_config

函数ctc_refsource_prescaler_config描述见下表:

表 3-156. 函数 ctc_refsource_prescaler_config

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

<code>_PSC_DIV8</code>	
<code>CTC_REFSOURCE</code> <code>_PSC_DIV16</code>	参考信号16分频
<code>CTC_REFSOURCE</code> <code>_PSC_DIV32</code>	参考信号32分频
<code>CTC_REFSOURCE</code> <code>_PSC_DIV64</code>	参考信号64分频
<code>CTC_REFSOURCE</code> <code>_PSC_DIV128</code>	参考信号128分频
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* configure reference signal source prescaler */
```

```
ctc_refsource_prescaler_config(CTC_REFSOURCE_PSC_DIV2);
```

函数 `ctc_clock_limit_value_config`

函数 `ctc_clock_limit_value_config` 描述见下表：

表 3-157. 函数 `ctc_clock_limit_value_config`

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

例如：

```
/* configure clock trim base limit value */
```

```
ctc_clock_limit_value_config(0x1F);
```

函数 `ctc_counter_reload_value_config`

函数 `ctc_counter_reload_value_config` 描述见下表：

表 3-158. 函数 `ctc_counter_reload_value_config`

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

例如：

```
/* configure CTC counter reload value */
ctc_counter_reload_value_config(0x00FF);
```

函数 `ctc_counter_capture_value_read`

函数`ctc_counter_capture_value_read`描述见下表：

表 3-159. 函数 `ctc_counter_capture_value_read`

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

例如：

```
/* read CTC counter capture value */
uint16_t ctc_value = 0;
ctc_value = ctc_counter_capture_value_read();
```

函数 `ctc_counter_direction_read`

函数`ctc_counter_direction_read`描述见下表：

表 3-160. 函数 `ctc_counter_direction_read`

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

例如：

```
/* read ctc counter direction */
```

```
FlagStatus ctc_direction = SET;
```

```
ctc_direction = ctc_counter_direction_read();
```

函数 `ctc_counter_reload_value_read`

函数`ctc_counter_reload_value_read`描述见下表：

表 3-161. 函数 `ctc_counter_reload_value_read`

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

例如：

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

```
uint16_t ctc_reload_value = 0;
```

```
ctc_reload_value = ctc_counter_reload_value_read();
```

函数 ctc_irc48m_trim_value_read

函数ctc_irc48m_trim_value_read描述见下表：

表 3-162. 函数 ctc_irc48m_trim_value_read

函数名称	ctc_irc48m_trim_value_read
函数原形	uint8_t ctc_irc48m_trim_value_read(void);
功能描述	读取IRC48M校准值
先决条件	-
被调用函数	-
输入参数{in}	
-	-
输出参数{out}	
-	-
返回值	
uint8_t	7位IRC48M校准值（0-127）

例如：

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

```
uint8_t ctc_trim_value = 0;
```

```
ctc_trim_value = ctc_irc48m_trim_value_read();
```

函数 ctc_interrupt_enable

函数ctc_interrupt_enable描述见下表：

表 3-163. 函数 ctc_interrupt_enable

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

例如：

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

函数 ctc_interrupt_disable

函数ctc_interrupt_disable描述见下表：

表 3-164. 函数 ctc_interrupt_disable

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

例如：

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

函数 ctc_interrupt_flag_get

函数ctc_interrupt_flag_get描述见下表：

表 3-165. 函数 ctc_interrupt_flag_get

函数名称	ctc_interrupt_flag_get
函数原形	FlagStatus ctc_interrupt_flag_get(uint32_t int_flag);
功能描述	CTC中断标志获取
先决条件	-
被调用函数	-
输入参数{in}	
int_flag	CTC中断标志
CTC_INT_FLAG_C	时钟校准完成中断标志位

<i>KOK</i>	
<i>CTC_INT_FLAG_C KWARN</i>	时钟校准警告中断标志位
<i>CTC_INT_FLAG_E RR</i>	错误中断标志位
<i>CTC_INT_FLAG_E REF</i>	期望参考信号中断标志位
<i>CTC_INT_FLAG_C KERR</i>	时钟校准错误位
<i>CTC_INT_FLAG_R EFMISS</i>	参考同步脉冲信号丢失
<i>CTC_INT_FLAG_T RIMERR</i>	校准值错误位
输出参数{out}	
-	-
返回值	
FlagStatus	SET或RESET

例如：

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

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

函数 **ctc_interrupt_flag_clear**

函数ctc_interrupt_flag_clear描述见下表：

表 3-166. 函数 ctc_interrupt_flag_clear

函数名称	ctc_interrupt_flag_clear
函数原形	void ctc_interrupt_flag_clear(uint32_t int_flag);
功能描述	CTC中断标志清除
先决条件	-
被调用函数	-
输入参数{in}	
int_flag	CTC中断标志
<i>CTC_INT_FLAG_C KOK</i>	时钟校准完成中断标志位
<i>CTC_INT_FLAG_C KWARN</i>	时钟校准警告中断标志位
<i>CTC_INT_FLAG_E RR</i>	错误中断标志位
<i>CTC_INT_FLAG_E REF</i>	期望参考信号中断标志位
<i>CTC_INT_FLAG_C</i>	时钟校准错误位

<i>KERR</i>	
<i>CTC_INT_FLAG_R</i> <i>EFMISS</i>	参考同步脉冲信号丢失
<i>CTC_INT_FLAG_T</i> <i>RIMERR</i>	校准值错误位
输出参数{out}	
-	-
返回值	
-	-

例如：

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

```
ctc_interrupt_flag_clear(CTC_INT_FLAG_CKOK);
```

函数 ctc_flag_get

函数ctc_flag_get描述见下表：

表 3-167. 函数 ctc_flag_get

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

例如：


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

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

函数 ctc_flag_clear

函数ctc_flag_clear描述见下表：

表 3-168. 函数 ctc_flag_clear

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

例如：

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

```
ctc_flag_clear(CTC_FLAG_CKOK);
```

3.8. DBG

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

3.8.1. 外设寄存器说明

DBG寄存器列表如下表所示：

表 3-169. 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

3.8.2. 外设库函数说明

DBG库函数列表如下表所示：

表 3-170. DBG 库函数

库函数名称	库函数描述
dbg_deinit	复位DBG寄存器
dbg_id_get	读DBG_ID寄存器
dbg_low_power_enable	使能低功耗模式的MCU调试保持功能
dbg_low_power_disable	禁能低功耗模式的MCU调试保持功能
dbg_periph_enable	使能外设的MCU调试保持功能
dbg_periph_disable	禁能外设的MCU调试保持功能

枚举类型 dbg_periph_enum

表 3-171. 枚举类型 dbg_periph_enum

成员名称	功能描述
DBG_FWDGT_HOLD	当内核停止时，保持FWDGT计数器时钟
DBG_WWDGT_HOLD	当内核停止时，保持WWDGT计数器时钟
DBG_TIMER0_HOLD	当内核停止时，保持TIMER0计数器计数值不变（只适用于GD32L235xx）
DBG_TIMER1_HOLD	当内核停止时，保持TIMER1计数器计数值不变
DBG_TIMER2_HOLD	当内核停止时，保持TIMER2计数器计数值不变
DBG_CAN_HOLD	当内核停止时，保持CAN功能不变（只适用于 GD32L235xx）
DBG_I2C0_HOLD	当内核停止时，保持I2C0的SMBUS状态不变，用于调试
DBG_I2C1_HOLD	当内核停止时，保持I2C1的SMBUS状态不变，用于调试
DBG_TIMER5_HOLD	当内核停止时，保持TIMER5计数器计数值不变
DBG_TIMER6_HOLD	当内核停止时，保持TIMER6计数器计数值不变
DBG_TIMER8_HOLD	当内核停止时，保持TIMER8计数器计数值不变
DBG_TIMER11_HOLD	当内核停止时，保持TIMER11计数器计数值不变
DBG_TIMER14_HOLD	当内核停止时，保持TIMER14计数器计数值不变（只适用于GD32L235xx）
DBG_TIMER40_HOLD	当内核停止时，保持TIMER40计数器计数值不变（只适用于

成员名称	功能描述
	GD32L235xx)
DBG_RTC_HOLD	当内核停止时，保持RTC计数器，用于调试
DBG_LPTIMER_HOLD	当内核停止时，保持LPTIMER计数器计数值不变（只适用于GD32L233xx)
DBG_LPTIMER0_HOLD	当内核停止时，保持LPTIMER0计数器计数值不变（只适用于GD32L235xx)
DBG_I2C2_HOLD	当内核停止时，保持I2C2的SMBUS状态不变，用于调试
DBG_LPTIMER1_HOLD	当内核停止时，保持LPTIMER1计数器计数值不变（只适用于GD32L235xx)

函数 dbg_deinit

函数dbg_deinit描述见下表：

表 3-172. 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 3-173. 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 3-174. 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 3-175. 函数 `dbg_low_power_disable`

函数名称	<code>dbg_low_power_disable</code>
函数原形	<code>void dbg_low_power_disable(uint32_t dbg_low_power);</code>
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}	
<code>dbg_low_power</code>	低功耗模式调试保持
<code>DBG_LOW_POWER_SLEEP</code>	在睡眠模式下，保持调试器连接，可进行调试
<code>DBG_LOW_POWER_DEEPSLEEP</code>	在深度睡眠模式下，保持调试器连接，可进行调试
<code>DBG_LOW_POWER_STANDBY</code>	在待机模式下，保持调试器连接，可进行调试
输出参数{out}	
-	-
返回值	
-	-

例如：

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

```
dbg_low_power_disable(DBG_LOW_POWER_SLEEP);
```

函数 `dbg_periph_enable`

函数`dbg_periph_enable`描述见下表：

表 3-176. 函数 `dbg_periph_enable`

函数名称	<code>dbg_periph_enable</code>
函数原形	<code>void dbg_periph_enable(dbg_periph_enum dbg_periph);</code>
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}	
<code>dbg_periph</code>	参考枚举变量 表3-171. 枚举类型<code>dbg_periph_enum</code>
<code>DBG_FWDGT_HOLD</code>	当内核停止时，保持FWDGT计数器时钟
<code>DBG_WWDGT_HOLD</code>	当内核停止时，保持WWDGT计数器时钟
<code>DBG_TIMERx_HOLD</code>	当内核停止时，保持TIMERx计数器计数值不变（x=1, 2, 5, 6, 8, 11）
<code>DBG_I2Cx_HOLD</code>	当内核停止时，保持I2Cx（x=0, 1, 2）的SMBUS状态不变，用于调试

<i>DBG_RTC_HOLD</i>	当内核停止时，保持RTC计数器，用于调试
<i>DBG_LPTIMER_HOLD</i>	当内核停止时，保持LPTIMER计数器计数值不变
输出参数{out}	
-	-
返回值	
-	-

例如：

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

```
dbg_periph_enable(DBG_TIMER1_HOLD);
```

函数 `dbg_periph_disable`

函数 `dbg_periph_disable` 描述见下表：

表 3-177. 函数 `dbg_periph_disable`

函数名称	<code>dbg_periph_disable</code>
函数原形	<code>void dbg_periph_disable(dbg_periph_enum dbg_periph);</code>
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3-171. 枚举类型 <code>dbg_periph_enum</code>
<i>DBG_FWDGT_HOLD</i>	当内核停止时，保持FWDGT计数器时钟
<i>DBG_WWDGT_HOLD</i>	当内核停止时，保持WWDGT计数器时钟
<i>DBG_TIMERx_HOLD</i>	当内核停止时，保持TIMERx计数器计数值不变（x=1, 2, 5, 6, 8, 11）
<i>DBG_I2Cx_HOLD</i>	当内核停止时，保持I2Cx（x=0, 1, 2）的SMBUS状态不变，用于调试
<i>DBG_RTC_HOLD</i>	当内核停止时，保持RTC计数器，用于调试
<i>DBG_LPTIMER_HOLD</i>	当内核停止时，保持LPTIMER计数器计数值不变
输出参数{out}	
-	-
返回值	
-	-

例如：

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

```
dbg_periph_disable(DBG_TIMER1_HOLD);
```

3.9. DAC

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

3.9.1. 外设寄存器说明

DAC寄存器列表如下表所示：

表 3-178. DAC 寄存器

寄存器名称	寄存器描述
DAC_CTL0	DACx控制寄存器0
DAC_SWT	DACx软件触发寄存器
DAC_OUT0_R12DH	DACx_OUT0 12位右对齐数据保持寄存器
DAC_OUT0_L12DH	DACx_OUT0 12位左对齐数据保持寄存器
DAC_OUT0_R8DH	DACx_OUT0 8位右对齐数据保持寄存器
DAC_OUT0_DO	DACx_OUT0数据输出寄存器
DAC_STAT0	DACx状态寄存器0

3.9.2. 外设库函数说明

DAC库函数列表如下表所示：

表 3-179. 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_gpio_connect_config	DAC的GPIO连接模式配置
dac_output_buffer_enable	DAC输出缓冲区使能
dac_output_buffer_disable	DAC输出缓冲区禁能
dac_output_value_get	DAC输出数据获取
dac_data_set	DAC输出数据设置
dac_trigger_enable	DAC触发使能
dac_trigger_disable	DAC触发禁能
dac_trigger_source_config	DAC触发源选择
dac_software_trigger_enable	DAC软件触发使能
dac_wave_mode_config	DAC噪声波模式配置
dac_lfsr_noise_config	DAC LFSR模式配置
dac_triangle_noise_config	DAC三角波模式配置
dac_flag_get	DAC标志位获取

库函数名称	库函数描述
<code>dac_flag_clear</code>	DAC标志位清除
<code>dac_interrupt_enable</code>	DAC中断使能
<code>dac_interrupt_disable</code>	DAC中断禁能
<code>dac_interrupt_flag_get</code>	DAC中断标志位获取
<code>dac_interrupt_flag_clear</code>	DAC中断标志位清除

函数 `dac_deinit`

函数`dac_deinit`描述见下表：

表 3-180. 函数 `dac_deinit`

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

例如：

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

```
dac_deinit(DAC0);
```

函数 `dac_enable`

函数`dac_enable`描述见下表：

表 3-181. 函数 `dac_enable`

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

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

例如:

```
/* enable DAC0_OUT0 */
dac_enable(DAC0, DAC_OUT0);
```

函数 **dac_disable**

函数dac_disable描述见下表:

表 3-182. 函数 dac_disable

函数名称	dac_disable
函数原型	void dac_disable(uint32_t dac_periph, uint8_t dac_out);
功能描述	DAC禁能
先决条件	-
被调用函数	-
输入参数{in}	
dac_periph	DAC外设
<i>DACx</i>	DAC外设选择 (x = 0)
输入参数{in}	
dac_out	DAC输出
<i>DAC_OUTx</i>	DAC输出通道选择 (x = 0)
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 3-183. 函数 dac_dma_enable

函数名称	dac_dma_enable
函数原型	void dac_dma_enable(uint32_t dac_periph, uint8_t dac_out);
功能描述	DAC的DMA功能使能

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

例如:

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

函数 **dac_dma_disable**

函数dac_dma_disable描述见下表:

表 3-184. 函数 **dac_dma_disable**

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

例如:

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

函数 dac_gpio_connect_config

函数dac_gpio_connect_config描述见下表:

表 3-185. 函数 dac_gpio_connect_config

函数名称	dac_gpio_connect_config
函数原型	void dac_gpio_connect_config(uint32_t dac_periph, uint8_t dac_out, uint32_t gpio_connect);
功能描述	DAC的GPIO连接配置
先决条件	-
被调用函数	-
输入参数{in}	
dac_periph	DAC外设
DACx	DAC外设选择 (x = 0)
输入参数{in}	
dac_out	DAC输出
DAC_OUTx	DAC输出通道选择 (x = 0)
输入参数{in}	
gpio_connect	DAC_OUTx连接GPIO模式
PIN_PERIPHERAL	DAC_OUTx输出连接外部管脚以及片上CMP
PIN_PERIPHERAL_BUFFER	根据输出buffer的开关, 决定DAC_OUTx连接外部管脚以及片上CMP的模式
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* configure DAC0_OUT0 GPIO connection working in PIN_PERIPHERAL */
```

```
dac_gpio_connect_config (DAC0, DAC_OUT0, PIN_PERIPHERAL);
```

函数 dac_output_buffer_enable

函数dac_output_buffer_enable描述见下表:

表 3-186. 函数 dac_output_buffer_enable

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

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

例如:

```
/* enable DAC0_OUT0 output buffer */
```

```
dac_output_buffer_enable(DAC0, DAC_OUT0);
```

函数 **dac_output_buffer_disable**

函数dac_output_buffer_disable描述见下表:

表 3-187. 函数 **dac_output_buffer_disable**

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

例如:

```
/* disable DAC0_OUT0 output buffer */
```

```
dac_output_buffer_disable(DAC0, DAC_OUT0);
```

函数 **dac_output_value_get**

函数dac_output_value_get描述见下表:

表 3-188. 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)
输入参数{in}	
dac_out	DAC输出
DAC_OUTx	DAC输出通道选择 (x = 0)
输出参数{out}	
-	-
返回值	
uint16_t	外设DACx数据保持寄存器值 (0~4095)

例如:

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

```
uint16_t data = 0;
```

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

函数 dac_data_set

函数dac_data_set描述见下表:

表 3-189. 函数 dac_data_set

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

<i>DAC_ALIGN_8B_R</i>	8位数据右对齐
输入参数{in}	
data	写入DAC_OUTx的数据（0~4095）
输出参数{out}	
-	-
返回值	
-	-

例如:

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

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

函数 **dac_trigger_enable**

函数dac_trigger_enable描述见下表:

表 3-190. 函数 **dac_trigger_enable**

函数名称	dac_trigger_enable
函数原型	void dac_trigger_enable(uint32_t dac_periph, uint8_t dac_out);
功能描述	DAC触发使能
先决条件	-
被调用函数	-
输入参数{in}	
dac_periph	DAC外设
<i>DACx</i>	DAC外设选择（x = 0）
输入参数{in}	
dac_out	DAC输出
<i>DAC_OUTx</i>	DAC输出通道选择（x = 0）
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 3-191. 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
DACx	DAC外设选择 (x = 0)
输入参数{in}	
dac_out	DAC输出
DAC_OUTx	DAC输出通道选择 (x = 0)
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* disable DAC0_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 3-192. 函数 dac_trigger_source_config

函数名称	dac_trigger_source_config
函数原型	void dac_trigger_source_config(uint32_t dac_periph, uint8_t dac_out, uint32_t triggersource);
功能描述	DAC触发源配置
先决条件	-
被调用函数	-
输入参数{in}	
dac_periph	DAC外设
DACx	DAC外设选择 (x = 0)
输入参数{in}	
dac_out	DAC输出
DAC_OUTx	DAC输出通道选择 (x = 0)
输入参数{in}	
triggersource	DAC触发源
DAC_TRIGGER_T1_TRGO	TIMER1 TRGO
DAC_TRIGGER_T2_TRGO	TIMER2 TRGO
DAC_TRIGGER_T6	TIMER6 TRGO

<code>_TRGO</code>	
<code>DAC_TRIGGER_T5</code> <code>_TRGO</code>	TIMER5 TRGO
<code>DAC_TRIGGER_EXTI_9</code>	EXTI线9中断
<code>DAC_TRIGGER_SOFTWARE</code>	软件触发
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* configure DAC0_OUT0 trigger source */
dac_trigger_source_config(DAC0, DAC_OUT0, DAC_TRIGGER_T1_TRGO);
```

函数 `dac_software_trigger_enable`

函数 `dac_software_trigger_enable` 描述见下表:

表 3-193. 函数 `dac_software_trigger_enable`

函数名称	<code>dac_software_trigger_enable</code>
函数原型	<code>void dac_software_trigger_enable(uint32_t dac_periph, uint8_t dac_out);</code>
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}	
<code>dac_periph</code>	DAC外设
<code>DACx</code>	DAC外设选择 ($x = 0$)
输入参数{in}	
<code>dac_out</code>	DAC输出
<code>DAC_OUTx</code>	DAC输出通道选择 ($x = 0$)
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 3-194. 函数 dac_wave_mode_config

函数名称	dac_wave_mode_config
函数原型	void dac_wave_mode_config(uint32_t dac_periph, uint8_t dac_out, uint32_t wave_mode);
功能描述	DAC噪声波模式配置
先决条件	-
被调用函数	-
输入参数{in}	
dac_periph	DAC外设
DACx	DAC外设选择 (x = 0)
输入参数{in}	
dac_out	DAC输出
DAC_OUTx	DAC输出通道选择 (x = 0)
输入参数{in}	
wave_mode	噪声波模式选择
DAC_WAVE_DISABLE	噪声波模式禁能
DAC_WAVE_MODE_LFSR	LFSR噪声波模式
DAC_WAVE_MODE_TRIANGLE	三角波噪声波模式
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* configure DAC0_OUT0 wave mode */
```

```
dac_wave_mode_config(DAC0, DAC_OUT0, DAC_WAVE_DISABLE);
```

函数 dac_lfsr_noise_config

函数dac_lfsr_noise_config描述见下表:

表 3-195. 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模式配置
先决条件	-

被调用函数	-
输入参数{in}	
dac_periph	DAC外设
<i>DACx</i>	DAC外设选择 (x = 0)
输入参数{in}	
dac_out	DAC输出
<i>DAC_OUTx</i>	DAC输出通道选择 (x = 0)
输入参数{in}	
unmask_bits	噪声波的非屏蔽位宽
<i>DAC_LFSR_BIT0</i>	LFSR模式位0非屏蔽
<i>DAC_LFSR_BITSx_0</i>	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 3-196. 函数 **dac_triangle_noise_config**

函数名称	dac_triangle_noise_config
函数原型	void dac_triangle_noise_config(uint32_t dac_periph, uint8_t dac_out, uint32_t amplitude);
功能描述	DAC三角波模式配置
先决条件	-
被调用函数	-
输入参数{in}	
dac_periph	DAC外设
<i>DACx</i>	DAC外设选择 (x = 0)
输入参数{in}	
dac_out	DAC输出
<i>DAC_OUTx</i>	DAC输出通道选择 (x = 0)
输入参数{in}	
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_flag_get**

函数dac_flag_get描述见下表:

表 3-197. 函数 dac_flag_get

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

例如:

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

```
FlagStatus flag;
```

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

函数 **dac_flag_clear**

函数dac_flag_clear描述见下表:

表 3-198. 函数 dac_flag_clear

函数名称	dac_flag_clear
函数原型	void dac_flag_clear(uint32_t dac_periph, uint32_t flag);
功能描述	DAC标志位清除

先决条件	-
被调用函数	-
输入参数{in}	
dac_periph	DAC外设
<i>DACx</i>	DAC外设选择 (x = 0)
输入参数{in}	
flag	DAC状态标志位
<i>DAC_FLAG_DDUD</i> <i>R0</i>	DACx_OUT0 DMA欠载标志位
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 3-199. 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
<i>DACx</i>	DAC外设选择 (x = 0)
输入参数{in}	
interrupt	DAC中断
<i>DAC_INT_DDUDR0</i>	DACx_OUT0 DMA欠载中断
输出参数{out}	
-	-
返回值	
-	-

例如:

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

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

函数 **dac_interrupt_disable**

函数dac_interrupt_disable描述见下表:

表 3-200. 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)
输入参数{in}	
interrupt	DAC中断
DAC_INT_DDUDR0	DACx_OUT0 DMA欠载中断
输出参数{out}	
-	-
返回值	
-	-

例如:

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

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

函数 **dac_interrupt_flag_get**

函数dac_interrupt_flag_get描述见下表:

表 3-201. 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)
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
输出参数{out}	
-	-

返回值	
FlagStatus	DAC中断状态（SET或RESET）

例如:

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

```
FlagStatus flag;
```

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

函数 dac_interrupt_flag_clear

函数dac_interrupt_flag_clear描述见下表:

表 3-202. 函数 dac_interrupt_flag_clear

函数名称	dac_interrupt_flag_clear
函数原型	void dac_interrupt_flag_clear(uint32_t dac_periph, uint32_t int_flag);
功能描述	DAC中断标志位清除
先决条件	-
被调用函数	-
输入参数{in}	
dac_periph	DAC外设
DACx	DAC外设选择 (x = 0)
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
输出参数{out}	
-	-
返回值	
-	-

例如:

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

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

3.10. 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 3-203. 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 3-204. DMAMUX 寄存器

寄存器名称	寄存器描述
DMAMUX_RM_CHx CFG (x=0..6)	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..6)	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 3-205. 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

库函数名称	库函数描述
<code>dma_struct_para_init</code>	将DMA结构体中所有参数初始化为默认值
<code>dma_init</code>	初始化外设DMA的通道x
<code>dma_circulation_enable</code>	使能DMA循环模式
<code>dma_circulation_disable</code>	禁能DMA循环模式
<code>dma_memory_to_memory_enable</code>	使能存储器到存储器DMA传输
<code>dma_memory_to_memory_disable</code>	禁能存储器到存储器DMA传输
<code>dma_channel_enable</code>	使能DMA通道x传输
<code>dma_channel_disable</code>	禁能DMA通道x传输
<code>dma_periph_address_config</code>	配置DMA通道x传输的外设基地址
<code>dma_memory_address_config</code>	配置DMA通道x传输的存储器基地址
<code>dma_transfer_number_config</code>	配置DMA通道x还有多少数据要传输
<code>dma_transfer_number_get</code>	获取DMA通道x还有多少数据要传输
<code>dma_priority_config</code>	配置DMA通道x的传输软件优先级
<code>dma_memory_width_config</code>	配置DMA通道x传输的存储器数据
<code>dma_periph_width_config</code>	配置DMA通道x传输的外设数据宽度
<code>dma_memory_increase_enable</code>	使能DMA通道x传输的存储器地址生成算法增量模式
<code>dma_memory_increase_disable</code>	禁能DMA通道x传输的存储器地址生成算法增量模式
<code>dma_periph_increase_enable</code>	使能DMA通道x传输的外设地址生成算法增量模式
<code>dma_periph_increase_disable</code>	禁能DMA通道x传输的外设地址生成算法增量模式
<code>dma_transfer_direction_config</code>	配置DMA通道x的传输方向
<code>dma_flag_get</code>	获取DMA通道x标志位状态
<code>dma_flag_clear</code>	清除DMA通道x标志位状态
<code>dma_interrupt_enable</code>	使能DMA通道x中断
<code>dma_interrupt_disable</code>	禁能DMA通道x中断
<code>dma_interrupt_flag_get</code>	获取DMA通道x中断标志位状态
<code>dma_interrupt_flag_clear</code>	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 3-206. DMAMUX 库函数

库函数名称	库函数描述
<code>dmamux_sync_struct_para_init</code>	将DMAMUX同步结构体中所有参数初始化为默认值
<code>dmamux_synchronization_init</code>	初始化DMAMUX同步结构体通道x
<code>dmamux_synchronization_enable</code>	使能DMAMUX同步模式
<code>dmamux_synchronization_disable</code>	禁能DMAMUX同步模式
<code>dmamux_event_generation_enable</code>	使能DMAMUX事件输出
<code>dmamux_event_generation_disable</code>	禁能DMAMUX事件输出
<code>dmamux_gen_struct_para_init</code>	将DMAMUX请求生成结构体中所有参数初始化为默认值
<code>dmamux_request_generator_init</code>	初始化DMAMUX请求生成结构体通道x
<code>dmamux_request_generator_channel_enable</code>	使能DMAMUX请求生成通道x
<code>dmamux_request_generator_channel_disable</code>	禁能DMAMUX请求生成通道x

库函数名称	库函数描述
disable	
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 3-207. 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 3-208. 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 3-209. 结构体 `dmamux_gen_parameter_struct`

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

枚举 `dmamux_interrupt_enum`表 3-210. 枚举 `dmamux_interrupt_enum`

成员名称	功能描述
<code>DMAMUX_INT_MU_XCH0_SO</code>	DMAMUX请求路由通道0同步溢出中断
<code>DMAMUX_INT_MU_XCH1_SO</code>	DMAMUX请求路由通道1同步溢出中断
<code>DMAMUX_INT_MU_XCH2_SO</code>	DMAMUX请求路由通道2同步溢出中断
<code>DMAMUX_INT_MU_XCH3_SO</code>	DMAMUX请求路由通道3同步溢出中断
<code>DMAMUX_INT_MU_XCH4_SO</code>	DMAMUX请求路由通道4同步溢出中断
<code>DMAMUX_INT_MU_XCH5_SO</code>	DMAMUX请求路由通道5同步溢出中断
<code>DMAMUX_INT_MU_XCH6_SO</code>	DMAMUX请求路由通道6同步溢出中断
<code>DMAMUX_INT_GE_NCH0_TO</code>	DMAMUX请求生成通道0触发溢出中断
<code>DMAMUX_INT_GE_NCH1_TO</code>	DMAMUX请求生成通道1触发溢出中断
<code>DMAMUX_INT_GE_NCH2_TO</code>	DMAMUX请求生成通道2触发溢出中断
<code>DMAMUX_INT_GE_NCH3_TO</code>	DMAMUX请求生成通道3触发溢出中断

枚举 `dmamux_flag_enum`表 3-211. 枚举 `dmamux_flag_enum`

成员名称	功能描述
<code>DMAMUX_FLAG_M_UXCH0_SO</code>	DMAMUX请求路由通道0同步溢出标志
<code>DMAMUX_FLAG_M_UXCH1_SO</code>	DMAMUX请求路由通道1同步溢出标志
<code>DMAMUX_FLAG_M_UXCH2_SO</code>	DMAMUX请求路由通道2同步溢出标志

UXCH2_SO	
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_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 3-212. 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_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

G_GENCH3_TO	
-------------	--

枚举 dma_channel_enum

表 3-213. 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 3-214. 枚举 dmamux_multiplexer_channel_enum

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

枚举 dmamux_generator_channel_enum

表 3-215. 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

函数 dma_deinit

函数 dma_deinit 描述见下表：

表 3-216. 函数 dma_deinit

函数名称	dma_deinit
函数原型	void dma_deinit(dma_channel_enum channelx);
功能描述	复位DMA通道x的所有寄存器
先决条件	无
被调用函数	无
输入参数{in}	
channelx	DMA通道
DMA_CHx(x=0..6)	DMA通道选择，参考 表3-213. 枚举dma_channel_enum
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* deinitialize DMA channel0 registers */
dma_deinit(DMA_CH0);
```

函数 dma_struct_para_init

函数 dma_struct_para_init 描述见下表：

表 3-217. 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}	
-	-
输出参数{out}	
init_struct	一个已经定义的dma_parameter_struct结构体变量地址，参考 表3-207. 结构体 dma_parameter_struct
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 3-218. 函数 dma_init

函数名称	dma_init
函数原型	void dma_init(dma_channel_enum channelx, dma_parameter_struct* init_struct);
功能描述	初始化DMA通道x
先决条件	无
被调用函数	无
输入参数{in}	
channelx	DMA通道
DMA_CHx(x=0..6)	DMA通道选择, 参考 表3-213. 枚举dma_channel_enum
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3-207. 结构体 dma_parameter_struct
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* DMA channel0 initialize */
dma_parameter_struct dma_init_struct;

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_8BIT;
dma_init_struct.number = TRANSFER_NUM;
dma_init_struct.periph_addr = (uint32_t)BANK0_WRITE_START_ADDR;
dma_init_struct.periph_inc = DMA_PERIPH_INCREASE_ENABLE;
dma_init_struct.periph_width = DMA_PERIPHERAL_WIDTH_8BIT;
dma_init_struct.priority = DMA_PRIORITY_ULTRA_HIGH;
dma_init(DMA_CH0, &dma_init_struct);
```

函数 dma_circulation_enable

函数 dma_circulation_enable 描述见下表:

表 3-219. 函数 dma_circulation_enable

函数名称	dma_circulation_enable
------	------------------------

函数原型	void dma_circulation_enable(dma_channel_enum channelx);
功能描述	DMA循环模式使能
先决条件	相应通道使能位CHEN需为0
被调用函数	无
输入参数{in}	
channelx	DMA通道
DMA_CHx(x=0..6)	DMA通道选择, 参考 表3-213. 枚举dma_channel_enum
输出参数{out}	
-	-
返回值	
-	-

例如:

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

函数 dma_circulation_disable

函数 dma_circulation_disable 描述见下表:

表 3-220. 函数 dma_circulation_disable

函数名称	dma_circulation_disable
函数原型	void dma_circulation_disable(dma_channel_enum channelx);
功能描述	DMA循环模式禁能
先决条件	相应通道使能位CHEN需为0
被调用函数	无
输入参数{in}	
channelx	DMA通道
DMA_CHx(x=0..6)	DMA通道选择, 参考 表3-213. 枚举dma_channel_enum
输出参数{out}	
-	-
返回值	
-	-

例如:

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

函数 dma_memory_to_memory_enable

函数 dma_memory_to_memory_enable 描述见下表:

表 3-221. 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(dma_channel_enum channelx);
功能描述	存储器到存储器DMA传输使能
先决条件	相应通道使能位CHEN需为0
被调用函数	无
输入参数{in}	
channelx	DMA通道
DMA_CHx(x=0..6)	DMA通道选择, 参考 表3-213. 枚举dma_channel_enum
输出参数{out}	
-	-
返回值	
-	-

例如:

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

函数 dma_memory_to_memory_disable

函数 dma_memory_to_memory_disable 描述见下表:

表 3-222. 函数 dma_memory_to_memory_disable

函数名称	dma_memory_to_memory_disable
函数原形	void dma_memory_to_memory_disable(dma_channel_enum channelx);
功能描述	存储器到存储器DMA传输禁能
先决条件	相应通道使能位CHEN需为0
被调用函数	无
输入参数{in}	
channelx	DMA通道
DMA_CHx(x=0..6)	DMA通道选择, 参考 表3-213. 枚举dma_channel_enum
输出参数{out}	
-	-
返回值	
-	-

例如:

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

函数 dma_channel_enable

函数 dma_channel_enable 描述见下表:

表 3-223. 函数 dma_channel_enable

函数名称	dma_channel_enable
------	--------------------

函数原型	void dma_channel_enable(dma_channel_enum channelx);
功能描述	DMA通道x传输使能
先决条件	无
被调用函数	无
输入参数{in}	
channelx	DMA通道
DMA_CHx(x=0..6)	DMA通道选择, 参考 表3-213. 枚举dma_channel_enum
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* enable DMA channel0 */
dma_channel_enable(DMA_CH0)
```

函数 dma_channel_disable

函数 dma_channel_disable 描述见下表:

表 3-224. 函数 dma_channel_disable

函数名称	dma_channel_disable
函数原型	void dma_channel_disable(dma_channel_enum channelx);
功能描述	DMA通道x传输禁能
先决条件	无
被调用函数	无
输入参数{in}	
channelx	DMA通道
DMA_CHx(x=0..6)	DMA通道选择, 参考 表3-213. 枚举dma_channel_enum
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* disable DMA channel0 */
dma_channel_disable(DMA_CH0);
```

函数 dma_periph_address_config

函数 dma_periph_address_config 描述见下表:

表 3-225. 函数 dma_periph_address_config

函数名称	dma_periph_address_config
------	---------------------------

函数原型	void dma_periph_address_config(dma_channel_enum channelx, uint32_t address);
功能描述	DMA通道x传输的外设基地址配置
先决条件	相应通道使能位CHEN需为0
被调用函数	无
输入参数{in}	
channelx	DMA通道
DMA_CHx(x=0..6)	DMA通道选择, 参考 表3-213. 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:

```
/* configure DMA channel0 periph address */
#define BANK0_WRITE_START_ADDR ((uint32_t)0x08004000)
dma_periph_address_config(DMA_CH0, BANK0_WRITE_START_ADDR);
```

函数 dma_memory_address_config

函数 dma_memory_address_config 描述见下表:

表 3-226. 函数 dma_memory_address_config

函数名称	dma_memory_address_config
函数原型	void dma_memory_address_config(dma_channel_enum channelx, uint32_t address);
功能描述	DMA通道x传输的存储器基地址配置
先决条件	相应通道使能位CHEN需为0
被调用函数	无
输入参数{in}	
channelx	DMA通道
DMA_CHx(x=0..6)	DMA通道选择, 参考 表3-213. 枚举dma_channel_enum
输入参数{in}	
address	存储器基地址, 0 – 0xFFFFFFFF
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* configure DMA channel0 memory address */
uint8_t g_destbuf[TRANSFER_NUM];
```

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

函数 dma_transfer_number_config

函数 dma_transfer_number_config 描述见下表：

表 3-227. 函数 dma_transfer_number_config

函数名称	dma_transfer_number_config
函数原型	void dma_transfer_number_config(dma_channel_enum channelx, uint32_t number);
功能描述	配置DMA通道x还有多少数据要传输
先决条件	相应通道使能位CHEN需为0
被调用函数	无
输入参数{in}	
channelx	DMA通道
DMA_CHx(x=0..6)	DMA通道选择, 参考 表3-213. 枚举dma_channel_enum
输入参数{in}	
number	数据传输数量 (0x0 – 0xFFFF)
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* configure DMA channel0 transfer number */

#define TRANSFER_NUM                0x400
dma_transfer_number_config(DMA_CH0, TRANSFER_NUM);
```

函数 dma_transfer_number_get

函数 dma_transfer_number_get 描述见下表：

表 3-228. 函数 dma_transfer_number_get

函数名称	dma_transfer_number_get
函数原型	uint32_t dma_transfer_number_get(dma_channel_enum channelx);
功能描述	获取DMA通道x还有多少数据要传输
先决条件	相应通道使能位CHEN需为0
被调用函数	无
输入参数{in}	
channelx	DMA通道
DMA_CHx(x=0..6)	DMA通道选择, 参考 表3-213. 枚举dma_channel_enum
输出参数{out}	
-	-
返回值	

uint32_t	DMA数据传输剩余数量（0x0 – 0xFFFF）
-----------------	---------------------------

例如：

```
/* get DMA channel0 transfer number */
uint32_t number = 0;

number = dma_transfer_number_get(DMA_CH0);
```

函数 dma_priority_config

函数 dma_priority_config 描述见下表：

表 3-229. 函数 dma_priority_config

函数名称	dma_priority_config
函数原型	void dma_priority_config(dma_channel_enum channelx, uint32_t priority);
功能描述	DMA通道x的传输软件优先级配置
先决条件	相应通道使能位CHEN需为0
被调用函数	无
输入参数{in}	
channelx	DMA通道
DMA_CHx(x=0..6)	DMA通道选择，参考 表3-213. 枚举dma_channel_enum
输入参数{in}	
priority	DMA通道软件优先级
DMA_PRIORITY_LOW	低优先级
DMA_PRIORITY_MEDIUM	中优先级
DMA_PRIORITY_HIGH	高优先级
DMA_PRIORITY_ULTRA_HIGH	极高优先级
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* configure DMA channel0 priority */
dma_priority_config(DMA_CH0, DMA_PRIORITY_ULTRA_HIGH);
```

函数 dma_memory_width_config

函数 dma_memory_width_config 描述见下表：

表 3-230. 函数 dma_memory_width_config

函数名称	dma_memory_width_config
函数原型	void dma_memory_width_config(dma_channel_enum channelx, uint32_t mwidth);
功能描述	DMA通道x传输的存储器数据宽度配置
先决条件	相应通道使能位CHEN需为0
被调用函数	无
输入参数{in}	
channelx	DMA通道
DMA_CHx(x=0..6)	DMA通道选择, 参考 表3-213. 枚举dma_channel_enum
输入参数{in}	
mwidth	存储器数据传输宽度
DMA_MEMORY_WIDTH_8BIT	8位数据传输宽度
DMA_MEMORY_WIDTH_16BIT	16位数据传输宽度
DMA_MEMORY_WIDTH_32BIT	32位数据传输宽度
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* configure DMA channel0 memory width */
dma_memory_width_config(DMA_CH0, DMA_MEMORY_WIDTH_8BIT);
```

函数 dma_periph_width_config

函数 dma_periph_width_config 描述见下表:

表 3-231. 函数 dma_periph_width_config

函数名称	dma_periph_width_config
函数原型	void dma_periph_width_config(dma_channel_enum channelx, uint32_t pwidth);
功能描述	DMA通道x传输的外设数据宽度配置
先决条件	相应通道使能位CHEN需为0
被调用函数	无
输入参数{in}	
channelx	DMA通道
DMA_CHx(x=0..6)	DMA通道选择, 参考 表3-213. 枚举dma_channel_enum
输入参数{in}	
pwidth	外设数据传输宽度
DMA_PERIPHERAL	8位数据传输宽度

<code>_WIDTH_8BIT</code>	
<code>DMA_PERIPHERAL_WIDTH_16BIT</code>	16位数据传输宽度
<code>DMA_PERIPHERAL_WIDTH_32BIT</code>	32位数据传输宽度
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* configure DMA channel0 periph width */
dma_periph_width_config(DMA_CH0, DMA_PERIPHERAL_WIDTH_8BIT);
```

函数 `dma_memory_increase_enable`

函数 `dma_memory_increase_enable` 描述见下表：

表 3-232. 函数 `dma_memory_increase_enable`

函数名称	<code>dma_memory_increase_enable</code>
函数原型	<code>void dma_memory_increase_enable(dma_channel_enum channelx);</code>
功能描述	DMA通道x传输的存储器地址生成算法增量模式使能
先决条件	相应通道使能位CHEN需为0
被调用函数	无
输入参数{in}	
<code>channelx</code>	DMA通道
<code>DMA_CHx(x=0..6)</code>	DMA通道选择，参考 表3-213. 枚举dma_channel_enum
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* enable DMA channel0 memory increase */
dma_memory_increase_enable(DMA_CH0);
```

函数 `dma_memory_increase_disable`

函数 `dma_memory_increase_disable` 描述见下表：

表 3-233. 函数 `dma_memory_increase_disable`

函数名称	<code>dma_memory_increase_disable</code>
函数原型	<code>void dma_memory_increase_disable(dma_channel_enum channelx);</code>
功能描述	DMA通道x传输的存储器地址生成算法增量模式禁能

先决条件	相应通道使能位CHEN需为0
被调用函数	无
输入参数{in}	
channelx	DMA通道
DMA_CHx(x=0..6)	DMA通道选择, 参考 表3-213. 枚举dma_channel_enum
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* disable DMA channel0 memory increase */
dma_memory_increase_disable(DMA_CH0);
```

函数 dma_periph_increase_enable

函数 dma_periph_increase_enable 描述见下表:

表 3-234. 函数 dma_periph_increase_enable

函数名称	dma_periph_increase_enable
函数原型	void dma_periph_increase_enable(dma_channel_enum channelx);
功能描述	DMA通道x传输的外设地址生成算法增量模式使能
先决条件	相应通道使能位CHEN需为0
被调用函数	无
输入参数{in}	
channelx	DMA通道
DMA_CHx(x=0..6)	DMA通道选择, 参考 表3-213. 枚举dma_channel_enum
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* enable DMA channel0 periph increase*/
dma_periph_increase_enable(DMA_CH0);
```

函数 dma_periph_increase_disable

函数 dma_periph_increase_disable 描述见下表:

表 3-235. 函数 dma_periph_increase_disable

函数名称	dma_periph_increase_disable
函数原型	void dma_periph_increase_disable(dma_channel_enum channelx);
功能描述	DMA通道x传输的外设地址生成算法增量模式禁能

先决条件	相应通道使能位CHEN需为0
被调用函数	无
输入参数{in}	
channelx	DMA通道
DMA_CHx(x=0..6)	DMA通道选择, 参考 表3-213. 枚举dma_channel_enum
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* disable DMA channel0 periph increase*/
dma_periph_increase_disable(DMA_CH0);
```

函数 dma_transfer_direction_config

函数 dma_transfer_direction_config 描述见下表:

表 3-236. 函数 dma_transfer_direction_config

函数名称	dma_transfer_direction_config
函数原型	void dma_transfer_direction_config(dma_channel_enum channelx, uint32_t direction);
功能描述	DMA通道x的传输方向配置
先决条件	相应通道使能位CHEN需为0
被调用函数	无
输入参数{in}	
channelx	DMA通道
DMA_CHx(x=0..6)	DMA通道选择, 参考 表3-213. 枚举dma_channel_enum
输入参数{in}	
direction	数据传输方向
DMA_PERIPHERAL_TO_MEMORY	读取外设中数据, 写入存储器
DMA_MEMORY_TO_PERIPHERAL	读取存储器中数据, 写入外设
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* configure DMA channel0 transfer direction*/
dma_transfer_direction_config(DMA_CH0, DMA_PERIPHERAL_TO_MEMORY);
```


函数 dma_flag_get

函数 dma_flag_get 描述见下表:

表 3-237. 函数 dma_flag_get

函数名称	dma_flag_get
函数原型	FlagStatus dma_flag_get(dma_channel_enum channelx, uint32_t flag);
功能描述	获取DMA通道x标志位状态
先决条件	无
被调用函数	无
输入参数{in}	
channelx	DMA通道
DMA_CHx(x=0..6)	DMA通道选择, 参考 表3-213. 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:

```
/* get DMA channel0 flag*/
FlagStatus flag = RESET;
flag = dma_flag_get(DMA_CH0, DMA_FLAG_FTF);
```

函数 dma_flag_clear

函数 dma_flag_clear 描述见下表:

表 3-238. 函数 dma_flag_clear

函数名称	dma_flag_clear
函数原型	void dma_flag_clear(dma_channel_enum channelx, uint32_t flag);
功能描述	清除DMA通道x标志位状态
先决条件	无
被调用函数	无
输入参数{in}	
channelx	DMA通道
DMA_CHx(x=0..6)	DMA通道选择, 参考 表3-213. 枚举dma_channel_enum
输入参数{in}	
flag	DMA标志

<i>DMA_FLAG_G</i>	DMA通道全局中断标志
<i>DMA_FLAG_FTF</i>	DMA通道传输完成标志
<i>DMA_FLAG_HTF</i>	DMA通道半传输完成标志
<i>DMA_FLAG_ERR</i>	DMA通道错误标志
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* clear DMA channel0 flag*/
dma_flag_clear(DMA_CH0, DMA_FLAG_FTF);
```

函数 dma_interrupt_enable

函数 dma_interrupt_enable 描述见下表：

表 3-239. 函数 dma_interrupt_enable

函数名称	dma_interrupt_enable
函数原型	void dma_interrupt_enable(dma_channel_enum channelx, uint32_t source);
功能描述	DMA通道x中断使能
先决条件	无
被调用函数	无
输入参数{in}	
channelx	DMA通道
<i>DMA_CHx</i> (x=0..6)	DMA通道选择，参考 表3-213. 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：

```
/* enable DMA channel0 interrupt */
dma_interrupt_enable(DMA_CH0, DMA_INT_FTF);
```

函数 dma_interrupt_disable

函数 dma_interrupt_disable 描述见下表：

表 3-240. 函数 dma_interrupt_disable

函数名称	dma_interrupt_disable
函数原型	void dma_interrupt_disable(dma_channel_enum channelx, uint32_t source);
功能描述	DMA通道x中断禁能
先决条件	无
被调用函数	无
输入参数{in}	
channelx	DMA通道
DMA_CHx(x=0..6)	DMA通道选择, 参考 表3-213. 枚举dma_channel_enum
输入参数{in}	
source	DMA中断源
DMA_INT_FTF	DMA通道传输完成中断
DMA_INT_HTF	DMA通道半传输完成中断
DMA_INT_ERR	DMA通道错误中断
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* disable DMA channel0 interrupt */
dma_interrupt_disable(DMA_CH0, DMA_INT_FTF);
```

函数 dma_interrupt_flag_get

函数 dma_interrupt_flag_get 描述见下表:

表 3-241. 函数 dma_interrupt_flag_get

函数名称	dma_interrupt_flag_get
函数原型	FlagStatus dma_interrupt_flag_get(dma_channel_enum channelx, uint32_t flag);
功能描述	获取DMA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}	
channelx	DMA通道
DMA_CHx(x=0..6)	DMA通道选择, 参考 表3-213. 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

<i>DMA_INT_FLAG_ER</i> <i>RR</i>	DMA通道错误中断标志
输出参数{out}	
-	-
返回值	
FlagStatus	SET或RESET

例如：

```
/* get DMA interrupt flag*/
if(dma_interrupt_flag_get(DMA_CH3, DMA_INT_FLAG_FTF)){
    dma_interrupt_flag_clear(DMA_CH3, DMA_INT_FLAG_FTF);
}
```

函数 dma_interrupt_flag_clear

函数 dma_interrupt_flag_clear 描述见下表：

表 3-242. 函数 dma_interrupt_flag_clear

函数名称	dma_interrupt_flag_clear
函数原型	void dma_interrupt_flag_clear(dma_channel_enum channelx, uint32_t flag);
功能描述	清除DMA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}	
channelx	DMA通道
<i>DMA_CHx</i> (x=0..6)	DMA通道选择，参考 表3-213. 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_ER</i> <i>R</i>	DMA通道错误中断标志
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* clear DMA interrupt flag*/
if(dma_interrupt_flag_get(DMA_CH3, DMA_INT_FLAG_FTF)){
    dma_interrupt_flag_clear(DMA_CH3, DMA_INT_FLAG_FTF);
}
```

函数 dmamux_sync_struct_para_init

函数 dmamux_sync_struct_para_init 描述见下表：

表 3-243. 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 3-208. 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 3-244. 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(x=0..6)	DMAMUX通道选择，参考 表3-214. 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 3-208. 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 3-245. 函数 dmamux_synchronization_enable

函数名称	dmamux_synchronization_enable
函数原型	void dmamux_synchronization_enable(dmamux_multiplexer_channel_enum channelx);
功能描述	使能DMAMUX同步模式
先决条件	无
被调用函数	无
输入参数{in}	
channelx	指定要初始化的DMAMUX请求路由通道
DMAMUX_MUXCHx(x=0..6)	DMAMUX通道选择，参考 表3-214. 枚举 dmamux_multiplexer_channel_enum
输出参数{out}	
-	-
返回值	
-	-

例如：

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

函数 dmamux_synchronization_disable

函数 dmamux_synchronization_disable 描述见下表：

表 3-246. 函数 dmamux_synchronization_disable

函数名称	dmamux_synchronization_disable
函数原型	void dmamux_synchronization_disable(dmamux_multiplexer_channel_enum channelx);

功能描述	禁能DMAMUX同步模式
先决条件	无
被调用函数	无
输入参数{in}	
channelx	指定要初始化的DMAMUX请求路由通道
DMAMUX_MUXCH x(x=0..6)	DMAMUX通道选择, 参考 表3-214. 枚举 dmamux_mux_channel_enum
输出参数{out}	
-	-
返回值	
-	-

例如:

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

函数 dmamux_event_generation_enable

函数 dmamux_event_generation_enable 描述见下表:

表 3-247. 函数 dmamux_event_generation_enable

函数名称	dmamux_event_generation_enable
函数原型	void dmamux_event_generation_enable(dmamux_mux_channel_enum channelx);
功能描述	使能DMAMUX事件输出
先决条件	无
被调用函数	无
输入参数{in}	
channelx	指定要初始化的DMAMUX请求路由通道
DMAMUX_MUXCH x(x=0..6)	DMAMUX通道选择, 参考 表3-214. 枚举 dmamux_mux_channel_enum
输出参数{out}	
-	-
返回值	
-	-

例如:

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

函数 dmamux_event_generation_disable

函数 dmamux_event_generation_disable 描述见下表:

表 3-248. 函数 `dmamux_event_generation_disable`

函数名称	<code>dmamux_event_generation_disable</code>
函数原型	<code>void dmamux_event_generation_disable(dmamux_multiplexer_channel_enum channelx);</code>
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_MUXCHx(x=0..6)	DMAMUX通道选择, 参考 表3-214. 枚举 <code>dmamux_multiplexer_channel_enum</code>
输出参数{out}	
-	-
返回值	
-	-

例如:

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

函数 `dmamux_gen_struct_para_init`

函数 `dmamux_gen_struct_para_init` 描述见下表:

表 3-249. 函数 `dmamux_gen_struct_para_init`

函数名称	<code>dmamux_gen_struct_para_init</code>
函数原型	<code>void dmamux_gen_struct_para_init(dmamux_gen_parameter_struct *init_struct);</code>
功能描述	将DMAMUX请求生成结构体中所有参数初始化为默认值
先决条件	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3-209. 结构体 <code>dmamux_gen_parameter_struct</code>
返回值	
-	-

例如:

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


函数 dmamux_request_generator_init

函数 dmamux_request_generator_init 描述见下表：

表 3-250. 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 (x=0..3)	DMAMUX请求生成通道选择，参考 表3-215. 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3-209. 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 3-251. 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
<i>DMAMUX_GENCHx</i> (x=0..3)	DMAMUX请求生成通道选择, 参考 表3-215. 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 3-252. 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
<i>DMAMUX_GENCHx</i> (x=0..3)	DMAMUX请求生成通道选择, 参考 表3-215. 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 3-253. 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3-214. 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 3-254. 函数 dmamux_request_forward_number_config

函数名称	dmamux_request_forward_number_config
函数原型	void dmamux_request_forward_number_config(dmamux_multiplexer_channel_enum channelx, uint32_t number);
功能描述	配置DMAMUX通道x要传输多少个DMA请求
先决条件	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(x=0..6)	DMAMUX通道选择, 参考 表3-214. 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 3-255. 函数 dmamux_sync_id_config

函数名称	dmamux_sync_id_config
函数原型	void dmamux_sync_id_config(dmamux_mux_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(x=0..6)	DMAMUX通道选择, 参考 表3-214. 枚举 dmamux_mux_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

XTI6	
DMAMUX_SYNC_E XTI7	同步输入信号为EXTI7
DMAMUX_SYNC_E XTI8	同步输入信号为EXTI8
DMAMUX_SYNC_E XTI9	同步输入信号为EXTI9
DMAMUX_SYNC_E XTI10	同步输入信号为EXTI10
DMAMUX_SYNC_E XTI11	同步输入信号为EXTI11
DMAMUX_SYNC_E XTI12	同步输入信号为EXTI12
DMAMUX_SYNC_E XTI13	同步输入信号为EXTI13
DMAMUX_SYNC_E XTI14	同步输入信号为EXTI14
DMAMUX_SYNC_E XTI15	同步输入信号为EXTI15
DMAMUX_SYNC_E VT0_OUT	同步输入信号为Evt0_out
DMAMUX_SYNC_E VT1_OUT	同步输入信号为Evt1_out
DMAMUX_SYNC_E VT2_OUT	同步输入信号为Evt2_out
DMAMUX_SYNC_E VT3_OUT	同步输入信号为Evt3_out
DMAMUX_SYNC_T TIMER11_CH0_O	同步输入信号为TIMER11_CH0_O
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 3-256. 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_MUXCHx(x=0..6)	DMAMUX通道选择, 参考 表3-214. 枚举 dmamux_multiplexer_channel_enum
输入参数{in}	
id	DMA请求输入标识
DMA_REQUEST_M2M	内存到内存传输
DMA_REQUEST_GENERATOR0	DMAMUX请求生成通道0请求
DMA_REQUEST_GENERATOR1	DMAMUX请求生成通道1请求
DMA_REQUEST_GENERATOR2	DMAMUX请求生成通道2请求
DMA_REQUEST_GENERATOR3	DMAMUX请求生成通道3请求
DMA_REQUEST_ADC	DMAMUX ADC请求
DMA_REQUEST_DAC	DMAMUX DAC请求
DMA_REQUEST_I2C0_RX	DMAMUX I2C0 RX请求
DMA_REQUEST_I2C0_TX	DMAMUX I2C0 TX请求
DMA_REQUEST_I2C1_RX	DMAMUX I2C1 RX请求
DMA_REQUEST_I2C1_TX	DMAMUX I2C1 TX请求
DMA_REQUEST_I2C2_RX	DMAMUX I2C2 RX请求
DMA_REQUEST_I2C2_TX	DMAMUX I2C2 TX请求
DMA_REQUEST_SPI0_RX	DMAMUX SPI0 RX请求
DMA_REQUEST_SPI0_TX	DMAMUX SPI0 TX请求
DMA_REQUEST_SPI1_RX	DMAMUX SPI1 RX请求

DMA_REQUEST_SPI1_TX	DMAMUX SPI1 TX请求
DMA_REQUEST_TIMER1_CH0	DMAMUX TIMER1 CH0请求
DMA_REQUEST_TIMER1_CH1	DMAMUX TIMER1 CH1请求
DMA_REQUEST_TIMER1_CH2	DMAMUX TIMER1 CH2请求
DMA_REQUEST_TIMER1_CH3	DMAMUX TIMER1 CH3请求
DMA_REQUEST_TIMER1_UP	DMAMUX TIMER1 UP请求
DMA_REQUEST_TIMER2_CH0	DMAMUX TIMER2 CH0请求
DMA_REQUEST_TIMER2_CH1	DMAMUX TIMER2 CH1请求
DMA_REQUEST_TIMER2_CH2	DMAMUX TIMER2 CH2请求
DMA_REQUEST_TIMER2_CH3	DMAMUX TIMER2 CH3请求
DMA_REQUEST_TIMER2_TRIG	DMAMUX TIMER2 TRIG请求
DMA_REQUEST_TIMER2_UP	DMAMUX TIMER2 UP请求
DMA_REQUEST_TIMER5_UP	DMAMUX TIMER5 UP请求
DMA_REQUEST_TIMER6_UP	DMAMUX TIMER6 UP请求
DMA_REQUEST_CAU_IN	DMAMUX CAU IN请求
DMA_REQUEST_CAU_OUT	DMAMUX CAU OUT请求
DMA_REQUEST_USART0_RX	DMAMUX USART0 RX请求
DMA_REQUEST_USART0_TX	DMAMUX USART0 TX请求
DMA_REQUEST_USART1_RX	DMAMUX USART1 RX请求
DMA_REQUEST_USART1_TX	DMAMUX USART1 TX请求
DMA_REQUEST_UART3_RX	DMAMUX UART3 RX请求
DMA_REQUEST_UART3_TX	DMAMUX UART3 TX请求

ART3_TX	
DMA_REQUEST_U ART4_RX	DMAMUX UART4 RX请求
DMA_REQUEST_U ART4_TX	DMAMUX UART4 TX请求
DMA_REQUEST_L PUART_RX	DMAMUX LPUART RX请求，仅适用于GD32L233
DMA_REQUEST_L PUART_TX	DMAMUX LPUART TX请求，仅适用于GD32L233
DMA_REQUEST_L PUART0_RX	DMAMUX LPUART0 RX请求，仅适用于GD32L235
DMA_REQUEST_L PUART0_TX	DMAMUX LPUART0 TX请求，仅适用于GD32L235
DMA_REQUEST_L PUART1_RX	DMAMUX LPUART1 RX请求，仅适用于GD32L235
DMA_REQUEST_L PUART1_TX	DMAMUX LPUART1 TX请求，仅适用于GD32L235
DMA_REQUEST_TI MER0_CH0	DMAMUX TIMER0 CH0请求，仅适用于GD32L235
DMA_REQUEST_TI MER0_CH1	DMAMUX TIMER0 CH1请求，仅适用于GD32L235
DMA_REQUEST_TI MER0_CH2	DMAMUX TIMER0 CH2请求，仅适用于GD32L235
DMA_REQUEST_TI MER0_CH3	DMAMUX TIMER0 CH3请求，仅适用于GD32L235
DMA_REQUEST_TI MER0_UP	DMAMUX TIMER0 UP请求，仅适用于GD32L235
DMA_REQUEST_TI MER0_COM	DMAMUX TIMER0 COM请求，仅适用于GD32L235
DMA_REQUEST_TI MER14_CH0	DMAMUX TIMER14 CH0请求，仅适用于GD32L235
DMA_REQUEST_TI MER14_CH1	DMAMUX TIMER14 CH1请求，仅适用于GD32L235
DMA_REQUEST_TI MER14_TRIG	DMAMUX TIMER14 TRIG请求，仅适用于GD32L235
DMA_REQUEST_TI MER14_UP	DMAMUX TIMER14 UP请求，仅适用于GD32L235
DMA_REQUEST_TI MER14_COM	DMAMUX TIMER14 COM请求，仅适用于GD32L235
DMA_REQUEST_TI MER40_CH0	DMAMUX TIMER40 CH0请求，仅适用于GD32L235
DMA_REQUEST_TI MER40_CH1	DMAMUX TIMER40 CH1请求，仅适用于GD32L235

<code>DMA_REQUEST_TIMER40_TRIG</code>	DMAMUX TIMER40 TRIG请求，仅适用于GD32L235
<code>DMA_REQUEST_TIMER40_UP</code>	DMAMUX TIMER40 UP请求，仅适用于GD32L235
<code>DMA_REQUEST_TIMER40_COM</code>	DMAMUX TIMER40 COM请求，仅适用于GD32L235
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 3-257. 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> ($x=0..3$)	DMAMUX请求生成通道选择，参考 表3-215. 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 3-258. 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 (x=0..3)	DMAMUX请求生成通道选择, 参考 表3-215. 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 3-259. 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 (x=0..3)	DMAMUX请求生成通道选择, 参考 表3-215. 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_EVT0_OUT	触发输入为Evt0_out
DMAMUX_TRIGGE R_EVT1_OUT	触发输入为Evt1_out
DMAMUX_TRIGGE	触发输入为Evt2_out

R_EVT2_OUT	
DMAMUX_TRIGGER R_EVT3_OUT	触发输入为Evt3_out
DMAMUX_TRIGGER R_TIMER11_CH0_0	触发输入为TIMER11_CH0_O
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 3-260. 函数 dmamux_flag_get

函数名称	dmamux_flag_get
函数原型	FlagStatus dmamux_flag_get(dmamux_flag_enum flag);
功能描述	获取DMAMUXA通道x标志位状态
先决条件	无
被调用函数	无
输入参数{in}	
flag	标志类型，参考 表3-211. 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 3-261. 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3-211. 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 3-262. 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3-210. 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 3-263. 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3-210. 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 3-264. 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3-212. 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 3-265. 函数 dmamux_interrupt_flag_clear

函数名称	dmamux_interrupt_flag_clear
函数原型	FlagStatus dmamux_interrupt_flag_clear(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3-212. 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. EXTI

EXTI是MCU中的中断/事件控制器，包括30个相互独立的边沿检测电路（GD32L233xx产品）或32个相互独立的边沿检测电路（GD32L235xx产品）并且能够向处理器内核产生中断请求或唤醒事件。章节[3.11.1](#)描述了EXTI的寄存器列表，章节[3.11.2](#)对EXTI库函数进行说明。

3.11.1. 外设寄存器说明

EXTI寄存器列表如下表所示：

表 3-266. EXTI 寄存器

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

3.11.2. 外设库函数说明

EXTI库函数列表如下表所示：

表 3-267. 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

库函数名称	库函数描述
exti_interrupt_disable	EXTI线x中断禁能
exti_event_enable	EXTI线x事件使能
exti_event_disable	EXTI线x事件禁能
exti_software_interrupt_enable	EXTI线x软件中断事件使能
exti_software_interrupt_disable	EXTI线x软件中断事件禁能
exti_flag_get	获取EXTI线x中断标志位
exti_flag_clear	清除EXTI线x中断标志位
exti_interrupt_flag_get	获取EXTI线x中断标志位
exti_interrupt_flag_clear	清除EXTI线x中断标志位

枚举类型 `exti_line_enum`

表 3-268. GD32L233xx 的枚举类型 `exti_line_enum`

枚举名称	枚举描述
EXTI_0	EXTI线0
EXTI_1	EXTI线1
EXTI_2	EXTI线2
EXTI_3	EXTI线3
EXTI_4	EXTI线4
EXTI_5	EXTI线5
EXTI_6	EXTI线6
EXTI_7	EXTI线7
EXTI_8	EXTI线8
EXTI_9	EXTI线9
EXTI_10	EXTI线10
EXTI_11	EXTI线11
EXTI_12	EXTI线12
EXTI_13	EXTI线13
EXTI_14	EXTI线14
EXTI_15	EXTI线15
EXTI_16	EXTI线16
EXTI_17	EXTI线17
EXTI_18	EXTI线18
EXTI_19	EXTI线19
EXTI_20	EXTI线20
EXTI_21	EXTI线21
EXTI_22	EXTI线22
EXTI_23	EXTI线23
EXTI_24	EXTI线24
EXTI_25	EXTI线25
EXTI_26	EXTI线26
EXTI_27	EXTI线27

枚举名称	枚举描述
EXTI_28	EXTI线28
EXTI_29	EXTI线29

表 3-269. GD32L235xx 的枚举类型 exti_line_enum

枚举名称	枚举描述
EXTI_0	EXTI线0
EXTI_1	EXTI线1
EXTI_2	EXTI线2
EXTI_3	EXTI线3
EXTI_4	EXTI线4
EXTI_5	EXTI线5
EXTI_6	EXTI线6
EXTI_7	EXTI线7
EXTI_8	EXTI线8
EXTI_9	EXTI线9
EXTI_10	EXTI线10
EXTI_11	EXTI线11
EXTI_12	EXTI线12
EXTI_13	EXTI线13
EXTI_14	EXTI线14
EXTI_15	EXTI线15
EXTI_16	EXTI线16
EXTI_17	EXTI线17
EXTI_18	EXTI线18
EXTI_19	EXTI线19
EXTI_20	EXTI线20
EXTI_21	EXTI线21
EXTI_22	EXTI线22
EXTI_23	EXTI线23
EXTI_24	EXTI线24
EXTI_25	EXTI线25
EXTI_26	EXTI线26
EXTI_27	EXTI线27
EXTI_28	EXTI线28
EXTI_29	EXTI线29
EXTI_30	EXTI线30
EXTI_31	EXTI线31

枚举类型 `exti_mode_enum`

表 3-270. 枚举类型 `exti_mode_enum`

枚举名称	枚举描述
EXTI_INTERRUPT	EXTI中断模式
EXTI_EVENT	EXTI事件模式

枚举类型 `exti_trig_type_enum`

表 3-271. 枚举类型 `exti_trig_type_enum`

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

函数 `exti_deinit`

函数`exti_deinit`描述见下表：

表 3-272. 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 3-273. 函数 `exti_init`

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

	exti_trig_type_enum trig_type);
功能描述	初始化EXTI线x
先决条件	-
被调用函数	-
输入参数{in}	
linex	EXTI线x, GD32L233xx产品请参考 表3-268. GD32L233xx的枚举类型exti_line_enum , GD32L235xx产品请参考 表3-269. GD32L235xx的枚举类型exti_line_enum
输入参数{in}	
mode	EXTI模式, 参考 表3-270. 枚举类型exti_mode_enum
输入参数{in}	
trig_type	触发类型, 参考 表3-271. 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 3-274. 函数 exti_interrupt_enable

函数名称	exti_interrupt_enable
函数原形	void exti_interrupt_enable(exti_line_enum linex);
功能描述	EXTI线x中断使能
先决条件	-
被调用函数	-
输入参数{in}	
linex	EXTI线x, GD32L233xx产品请参考 表3-268. GD32L233xx的枚举类型exti_line_enum , GD32L235xx产品请参考 表3-269. GD32L235xx的枚举类型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 3-275. 函数 **exti_interrupt_disable**

函数名称	exti_interrupt_disable
函数原形	void exti_interrupt_disable(exti_line_enum linex);
功能描述	EXTI线x中断禁能
先决条件	-
被调用函数	-
输入参数{in}	
linex	EXTI线x，GD32L233xx产品请参考 表3-268. GD32L233xx的枚举类型 exti_line_enum ，GD32L235xx产品请参考 表3-269. GD32L235xx的枚举类型 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 3-276. 函数 **exti_event_enable**

函数名称	exti_event_enable
函数原形	void exti_event_enable(exti_line_enum linex);
功能描述	EXTI线x事件使能
先决条件	-
被调用函数	-
输入参数{in}	
linex	EXTI线x，GD32L233xx产品请参考 表3-268. GD32L233xx的枚举类型 exti_line_enum ，GD32L235xx产品请参考 表3-269. GD32L235xx的枚举类型 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 3-277. 函数 exti_event_disable

函数名称	exti_event_disable
函数原形	void exti_event_disable(exti_line_enum linex);
功能描述	EXTI线x事件禁能
先决条件	-
被调用函数	-
输入参数{in}	
linex	EXTI线x，GD32L233xx产品请参考 表3-268. GD32L233xx的枚举类型 exti_line_enum ，GD32L235xx产品请参考 表3-269. GD32L235xx的枚举类型 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 3-278. 函数 exti_software_interrupt_enable

函数名称	exti_software_interrupt_enable
函数原形	void exti_software_interrupt_enable(exti_line_enum linex);
功能描述	使能EXTI线x软件事件中断
先决条件	-
被调用函数	-
输入参数{in}	
linex	EXTI线x，GD32L233xx产品请参考 表3-268. GD32L233xx的枚举类型 exti_line_enum ，GD32L235xx产品请参考 表3-269. GD32L235xx的枚举类型 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 3-279. 函数 exti_software_interrupt_disable

函数名称	exti_software_interrupt_disable
函数原形	void exti_software_interrupt_disable(exti_line_enum linex);
功能描述	禁能EXTI线x软件事件中断
先决条件	-
被调用函数	-
输入参数{in}	
linex	EXTI线x，GD32L233xx产品请参考 表3-268. GD32L233xx的枚举类型 exti_line_enum ，GD32L235xx产品请参考 表3-269. GD32L235xx的枚举类型 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 3-280. 函数 exti_flag_get

函数名称	exti_flag_get
函数原形	FlagStatus exti_flag_get(exti_line_enum linex);
功能描述	获取EXTI线x中断标志位
先决条件	-
被调用函数	-
输入参数{in}	
linex	EXTI线x，GD32L233xx产品请参考 表3-268. GD32L233xx的枚举类型 exti_line_enum ，GD32L235xx产品请参考 表3-269. GD32L235xx的枚举类型 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 3-281. 函数 exti_flag_clear

函数名称	exti_flag_clear
函数原形	void exti_flag_clear(exti_line_enum linex);
功能描述	清除EXTI线x中断标志位
先决条件	-
被调用函数	-
输入参数{in}	
linex	EXTI线x, GD32L233xx产品请参考 表3-268. GD32L233xx的枚举类型 exti_line_enum , GD32L235xx产品请参考 表3-269. GD32L235xx的枚举类型 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 3-282. 函数 exti_interrupt_flag_get

函数名称	exti_interrupt_flag_get
函数原形	FlagStatus exti_interrupt_flag_get(exti_line_enum linex);
功能描述	获取EXTI线x中断标志位
先决条件	-
被调用函数	-
输入参数{in}	
linex	EXTI线x, GD32L233xx产品请参考 表3-268. GD32L233xx的枚举类型 exti_line_enum , GD32L235xx产品请参考 表3-269. GD32L235xx的枚举类型 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 3-283. 函数 exti_interrupt_flag_clear

函数名称	exti_interrupt_flag_clear
函数原形	void exti_interrupt_flag_clear(exti_line_enum linex);
功能描述	清除EXTI线x中断标志位
先决条件	-
被调用函数	-
输入参数{in}	
linex	EXTI线x, GD32L233xx产品请参考 表3-268. GD32L233xx的枚举类型 exti_line_enum , GD32L235xx产品请参考 表3-269. GD32L235xx的枚举类型 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);
```

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

3.12.1. 外设寄存器说明

FMC寄存器列表如下:

表 3-284. FMC 寄存器

寄存器	描述
FMC_WS	等待状态寄存器
FMC_KEY	解锁寄存器
FMC_OBKEY	选项字节解锁寄存器
FMC_STAT	状态寄存器
FMC_CTL	控制寄存器
FMC_ADDR	地址寄存器
FMC_ECCCS	ECC控制及状态寄存器（仅适用于GD32L235xx）
FMC_OBSTAT	选项字节状态寄存器
FMC_WP	擦写保护寄存器
FMC_SLPKEY	睡眠或掉电模式解锁寄存器
FMC_PID	产品ID寄存器

3.12.2. 外设库函数说明

FMC固件库函数列举如下表：

表 3-285. FMC 固件库函数

函数名称	函数描述
fmc_unlock	解锁FMC主编程块操作
fmc_lock	锁定FMC主编程块操作
fmc_wsnt_set	设置FMC等待状态
fmc_prefetch_enable	使能pre-fetch
fmc_prefetch_disable	禁能pre-fetch
fmc_low_power_enable	使能低电压模式（仅适用于GD32L233xx）
fmc_low_power_disable	禁能低电压模式（仅适用于GD32L233xx）
fmc_page_erase	FMC页擦除
fmc_mass_erase	FMC全片擦除
fmc_word_program	在相应地址全字编程（仅适用于GD32L233xx）
fmc_fast_program	在相应地址快速编程一行数据（32个双字）（仅适用于GD32L233xx）
fmc_doubleword_program	在相应地址双字编程（仅适用于GD32L235xx）
ob_unlock	解锁选项字节操作
ob_lock	锁定选项字节操作
ob_erase	擦除选项字节
ob_write_protection_enable	使能写保护
ob_security_protection_config	配置安全保护
ob_user_write	编程用户选项字节
ob_data_program	编程数据选项字节
ob_user_get	获取用户选项字节
ob_data_get	获取数据选项字节
ob_write_protection_get	获取擦写保护选项字节

函数名称	函数描述
ob_security_protection_flag_get	获取选项字节安全保护状态
fmc_ecc_address_get	获取发生ECC错误的地址
fmc_slp_unlock	解锁对FMC_WS寄存器RUN_SLP位的操作
fmc_sleep_slp_enable	Flash进入睡眠模式当MCU进入深度睡眠模式或RUN_SLP位置位
fmc_sleep_slp_disable	Flash进入掉电模式（GD32L233）\空闲模式（GD32L235）当MCU进入深度睡眠模式或RUN_SLP位置位
fmc_run_slp_enable	Flash进入空闲模式当MCU正常运行（GD32L235，和SLEEP_SLP位共同作用）\Flash进入空闲模式当MCU正常运行或低功耗运行（GD32L233，和SLEEP_SLP位共同作用）
fmc_run_slp_disable	Flash进入睡眠模式当MCU正常运行（GD32L235，和SLEEP_SLP位共同作用）\Flash进入睡眠模式或掉电模式当MCU正常运行或低功耗运行（GD32L233，和SLEEP_SLP位共同作用）
fmc_sleep_mode_enter	Flash进入睡眠模式当MCU正常运行。请注意该函数需要在SRAM中运行。
fmc_pd_mode_enter	Flash进入掉电模式当MCU正常运行或低功耗运行（仅适用于GD32L233xx）。请注意该函数需要在SRAM中运行。
fmc_slp_mode_exit	Flash退出睡眠模式或掉电模式当MCU正常运行或低功耗运行（GD32L233）\ Flash退出睡眠模式当MCU正常运行（GD32L235）。请注意该函数需要在SRAM中运行。
fmc_flag_get	检查FMC标志位是否置位
fmc_flag_clear	清除FMC标志
fmc_interrupt_enable	使能FMC中断
fmc_interrupt_disable	禁能FMC中断
fmc_interrupt_flag_get	获取FMC中断标志状态
fmc_interrupt_flag_clear	清除FMC中断标志状态

枚举类型 fmc_state_enum

表 3-286. 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_PGERR	编程错误
FMC_PGAERR	编程对齐错误
FMC_WPERR	擦写保护错误
FMC_TOERR	超时错误
FMC_OB_HSPC	高安全保护级别
FMC_SLP	睡眠或掉电状态

枚举类型 `fmc_flag_enum`**表 3-287. 枚举类型 `fmc_flag_enum`**

枚举名称	枚举描述
FMC_FLAG_BUSY	Flash忙状态
FMC_FLAG_PGER R	Flash编程错误标志
FMC_FLAG_PGAE RR	Flash编程对齐错误标志
FMC_FLAG_WPER R	Flash擦写保护错误标志
FMC_FLAG_SLP	Flash睡眠或掉电状态（仅适用于GD32L233xx）
FMC_FLAG_POWE RDOWN	Flash掉电状态（仅适用于GD32L235xx）
FMC_FLAG_SLEEP	Flash睡眠状态（仅适用于GD32L235xx）
FMC_FLAG_ECCC OR	单个位检测并纠正错误标志（仅适用于GD32L235xx）
FMC_FLAG_ECCD ET	双位检测错误标志（仅适用于GD32L235xx）
FMC_FLAG_SYSE CC	系统存储ECC错误标志（仅适用于GD32L235xx）
FMC_FLAG_MFEC C	主闪存ECC错误标志（仅适用于GD32L235xx）
FMC_FLAG_OTPE CC	OTP ECC错误标志（仅适用于GD32L235xx）
FMC_FLAG_OBEC C	选项字节ECC错误标志（仅适用于GD32L235xx）
FMC_FLAG_OBEC CDET	加载选项字节到寄存器时双位检测ECC错误标志（仅适用于GD32L235xx）
FMC_FLAG_OBER R	选项字节错误标志

枚举类型 `fmc_interrupt_flag_enum`**表 3-288. 枚举类型 `fmc_interrupt_flag_enum`**

枚举名称	枚举描述
FMC_INT_FLAG_P GERR	Flash编程错误中断标志
FMC_INT_FLAG_P GAERR	Flash编程对齐错误中断标志
FMC_INT_FLAG_W PERR	Flash擦写保护错误中断标志

枚举名称	枚举描述
FMC_INT_FLAG_END	Flash操作结束中断标志
FMC_INT_FLAG_ECCOR	单个位检测并纠正错误中断标志（仅适用于GD32L235xx）
FMC_INT_FLAG_ECCDET	双位检测错误中断标志（仅适用于GD32L235xx）
FMC_INT_FLAG_OBECCDET	加载选项字节到寄存器时双位检测ECC错误中断标志（仅适用于GD32L235xx）

枚举类型 `fmc_interrupt_enum`

表 3-289. 枚举类型 `fmc_interrupt_enum`

枚举名称	枚举描述
FMC_INT_ERR	FMC错误中断
FMC_INT_END	FMC操作结束中断
FMC_INT_ECCCORR	单个位检测并纠正错误中断（仅适用于GD32L235xx）
FMC_INT_ECCDET	双位检测错误中断（仅适用于GD32L235xx）

函数 `fmc_unlock`

函数`fmc_unlock`描述见下表：

表 3-290. 函数 `fmc_unlock`

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

例如：

```
/* unlock the main FMC operation */
fmc_unlock();
```

函数 **fmc_lock**

函数fmc_lock描述见下表：

表 3-291. 函数 **Function fmc_lock**

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

例如：

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

```
fmc_lock();
```

函数 **fmc_wscnt_set**

函数fmc_wscnt_set描述见下表：

表 3-292. 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	设置等待状态计数值
先决条件	-
被调用函数	-
输入参数{in}	
wscnt	等待状态计数值
<i>FMC_WAIT_STATE_0</i>	0个等待状态
<i>FMC_WAIT_STATE_1</i>	1个等待状态
<i>FMC_WAIT_STATE_2</i>	2个等待状态
<i>FMC_WAIT_STATE_3</i>	3个等待状态（仅适用于GD32L233xx）
输出参数{out}	
-	-
返回值	

-	-
---	---

例如:

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

```
fmc_wsctl_set(FMC_WAIT_STATE_1);
```

函数 fmc_prefetch_enable

函数fmc_prefetch_enable描述见下表:

表 3-293. 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pre-fetch
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 3-294. 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	禁能pre-fetch
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_low_power_enable

函数fmc_low_power_enable描述见下表：

表 3-295. 函数 fmc_low_power_enable

函数名称	fmc_low_power_enable
函数原型	void fmc_low_power_enable(void);
功能描述	使能低电压模式
先决条件	-
被调用函数	-
输入参数{in}	
-	-
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* enable low power */  
  
fmc_low_power_enable();
```

函数 fmc_low_power_disable

函数fmc_low_power_disable描述见下表：

表 3-296. 函数 fmc_low_power_disable

函数名称	fmc_low_power_disable
函数原型	void fmc_low_power_disable(void);
功能描述	禁能低电压模式
先决条件	-
被调用函数	-
输入参数{in}	
-	-
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* disable low power */
```

```
fmc_low_power_disable();
```

函数 fmc_page_erase

函数fmc_page_erase描述见下表:

表 3-297. 函数 fmc_page_erase

函数名称	fmc_page_erase
函数原型	fmc_state_enum fmc_page_erase(uint32_t page_address);
功能描述	页擦除
先决条件	fmc_unlock
被调用函数	-
输入参数{in}	
page_address	页擦除地址
输出参数{out}	
-	-
返回值	
fmc_state_enum	FMC状态值, 详情参考枚举变量 表3-287. 枚举类型 。

例如:

```
fmc_unlock();
```

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

```
fmc_state_enum state = fmc_page_erase(0x08004000);
```

函数 fmc_mass_erase

函数fmc_mass_erase描述见下表:

表 3-298. 函数 fmc_mass_erase

函数名称	fmc_mass_erase
函数原型	fmc_state_enum fmc_mass_erase(void);
功能描述	全片擦除
先决条件	fmc_unlock
被调用函数	-
输入参数{in}	
-	-
输出参数{out}	
-	-
返回值	
fmc_state_enum	FMC状态值, 详情参考枚举变量 表3-287. 枚举类型 。

例如:


```
fmc_unlock();

/* erase whole chip */

fmc_state_enum state = fmc_mass_erase();
```

函数 fmc_word_program

函数fmc_word_program描述见下表：

表 3-299. 函数 fmc_word_program

函数名称	fmc_word_program
函数原型	fmc_state_enum fmc_word_program(uint32_t address, uint32_t data);
功能描述	在相应地址全字编程
先决条件	fmc_unlock
被调用函数	-
输入参数{in}	
address	编程地址
输入参数{in}	
data	编程数据
输出参数{out}	
-	-
返回值	
fmc_state_enum	FMC状态值，详情参考枚举变量 表3-287. 枚举类型 。

例如：

```
fmc_unlock();

fmc_page_erase(0x08004000);

/* program a word at the corresponding address */

fmc_state_enum fmc_state = fmc_word_program( 0x08004000,0xaabbccdd);
```

函数 fmc_fast_program

函数fmc_fast_program描述见下表：

表 3-300. 函数 fmc_fast_program

函数名称	fmc_fast_program
函数原型	fmc_state_enum fmc_fast_program(uint32_t address, uint64_t data[]);
功能描述	在相应地址快速编程一行数据（32个双字）
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	编程数据
输出参数{out}	
-	-
返回值	
fmc_state_enum	FMC状态值，详情参考枚举变量 表3-287. 枚举类型 。

例如：

```
/* data buffer for fast programming */
```

```
static uint64_t data_buffer[32] = {
```

```
    0x0000000000000000U,    0x1111111111111111U,    0x2222222222222222U,
    0x3333333333333333U,
```

```
    0x4444444444444444U,    0x5555555555555555U,    0x6666666666666666U,
    0x7777777777777777U,
```

```
    0x8888888888888888U,    0x9999999999999999U,    0xAAAAAAAAAAAAAAAAAU,
    0BBBBBBBBBBBBBBBBBU,
```

```
    0CCCCCCCCCCCCCCCCCU,    0DDDDDDDDDDDDDDDDDU,
    0EEEEEEEEEEEEEEEEEU, 0FFFFFFFFFFFFFFFFFU,
```

```
    0x0011001100110011U,    0x2233223322332233U,    0x4455445544554455U,
    0x6677667766776677U,
```

```
    0x8899889988998899U, 0xAABBAABBAABBAABBU, 0xCCDDCCDDCCDDCCDDU,
    0xEEFFEEFFEEFFEEFFU,
```

```
    0x2200220022002200U,    0x3311331133113311U,    0x6644664466446644U,
    0x7755775577557755U,
```

```
    0xAA88AA88AA88AA88U, 0xBB99BB99BB99BB99U, 0xEECCEECCEECCECCU,
    0xFFDDFFDDFFDDFFDDU
```

```
};
```

```
fmc_unlock();
```

```
fmc_page_erase(0x08004000);
```

```
/* program flash */
```

```
fmc_state_enum fmc_state = fmc_fast_program(0x08004000, data_buffer);
```

函数 fmc_doubleword_program

函数fmc_doubleword_program描述见下表：

表 3-301. 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	编程数据
输出参数{out}	
-	-
返回值	
fmc_state_enum	FMC状态值，详情参考枚举变量 表3-287. 枚举类型 。

例如：

```
fmc_unlock();

fmc_page_erase(0x08004000);

/* program a double-word at the corresponding address */

fmc_state_enum          fmc_state          =
fmc_doubleword_program( 0x08004000,0xaabbccddaabbccdd);
```

函数 ob_unlock

函数ob_unlock描述见下表：

表 3-302. 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
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 byte operation */

ob_unlock();
```

函数 ob_lock

函数ob_lock描述见下表:

表 3-303. 函数 ob_lock

函数名称	ob_lock
函数原型	void ob_lock(void);
功能描述	锁定选项字节操作
先决条件	fmc_lock
被调用函数	-
输入参数{in}	
-	-
输出参数{out}	
-	-
返回值	
-	-

例如:

```
fmc_lock();
```

```
/* lock the option byte operation */
```

```
ob_lock();
```

函数 ob_erase

函数ob_erase描述见下表:

表 3-304. 函数 ob_erase

函数名称	ob_erase
函数原型	void ob_erase(void);
功能描述	擦除选项字节
先决条件	ob_unlock
被调用函数	-
输入参数{in}	
-	-
输出参数{out}	
-	-
返回值	
fmc_state_enum	FMC状态值, 详情参考枚举变量 表3-287. 枚举类型 。

例如:

```
fmc_unlock();
```

```
ob_unlock();
```

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

```
fmc_state_enum fmc_state = ob_erase();
```

函数 ob_write_protection_enable

函数ob_write_protection_enable描述见下表：

表 3-305. 函数 ob_write_protection_enable

函数名称	ob_write_protection_enable
函数原型	fmc_state_enum ob_write_protection_enable(uint32_t ob_wp);
功能描述	使能擦写保护
先决条件	ob_unlock
被调用函数	-
输入参数{in}	
ob_wp	待擦写保护的单元
输出参数{out}	
-	-
返回值	
fmc_state_enum	FMC状态值，详情参考枚举变量 表3-287. 枚举类型 。

例如：

```
fmc_unlock();
```

```
ob_unlock();
```

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

```
fmc_state_enum fmc_state = ob_write_protection_enable(OB_WP_1);
```

函数 ob_security_protection_config

函数ob_security_protection_config描述见下表：

表 3-306. 函数 ob_security_protection_config

函数名称	ob_security_protection_config
函数原型	fmc_state_enum ob_security_protection_config(uint8_t ob_spc);
功能描述	配置安全保护
先决条件	ob_unlock
被调用函数	-
输入参数{in}	
ob_spc	安全保护
FMC_NSPC	无安全保护
FMC_LSPC	低保护级别
FMC_HSPC	高保护级别
输出参数{out}	

-	-
返回值	
fmc_state_enum	FMC状态值，详情参考枚举变量 表3-287. 枚举类型 。

例如：

```
fmc_state_enum fmc_state;

fmc_unlock();

ob_unlock();

/* enable security protection */

fmc_state = ob_security_protection_config(FMC_USPC);
```

函数 ob_user_write

函数ob_user_write描述见下表：

表 3-307. 函数 ob_user_write

函数名称	ob_user_write
函数原型	fmc_state_enum ob_user_write(uint8_t ob_fwdgt, uint8_t ob_deepsleep, uint8_t ob_stdby, uint8_t ob_bor_th); (仅适用于GD32L233xx) / mc_state_enum ob_user_write(uint8_t ob_fwdgt, uint8_t ob_deepsleep, uint8_t ob_stdby, uint8_t ob_bor_th, uint8_t ob_sram_parity_check); (仅适用于GD32L235xx)
功能描述	编程用户选项字节
先决条件	ob_unlock
被调用函数	-
输入参数{in}	
ob_fwdgt	选项字节看门狗值
OB_FWDGT_SW	软件看门狗
OB_FWDGT_HW	硬件看门狗
输入参数{in}	
ob_deepsleep	选项字节深度睡眠模式复位使能
OB_DEEPSLEEP_N_RST	深度睡眠模式时进入深度睡眠模式而不产生复位
OB_DEEPSLEEP_R_RST	深度睡眠模式时产生复位而不进入深度睡眠模式
输入参数{in}	
ob_stdby	选项字节待机模式模式复位使能
OB_STDBY_N_RST	待机模式时进入待机模式而不产生复位
OB_STDBY_R_RST	待机模式时产生复位而不进入待机模式
输入参数{in}	
ob_bor_th	选项字节BOR复位阈值
OB_BOR_TH_VAL	BOR复位阈值0

UE0	
OB_BOR_TH_VAL UE1	BOR复位阈值1
OB_BOR_TH_VAL UE2	BOR复位阈值2
OB_BOR_TH_VAL UE3	BOR复位阈值3
OB_BOR_TH_VAL UE4	BOR复位阈值4
输入参数{in}	
ob_sram_parity_check	选项字节SRAM校验（仅适用于GD32L235xx）
OB_SRAM_PARITY_CHECK_ENABLE	使能SRAM校验
OB_SRAM_PARITY_CHECK_DISABLE	禁能SRAM校验
输出参数{out}	
-	-
返回值	
fmc_state_enum	FMC状态值，详情参考枚举变量 表3-287. 枚举类型 。

例如：

```
fmc_unlock();
```

```
ob_unlock();
```

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

```
fmc_state_enum fmc_state = ob_user_write(OB_FWDGT_HW, OB_DEEPSLEEP_Nrst,
OB_STDBY_Nrst, OB_BOR_TH_VALUE0);
```

函数 ob_data_program

函数ob_data_program描述见下表：

表 3-308. 函数 ob_data_program

函数名称	ob_data_program
函数原型	fmc_state_enum ob_data_program(uint16_t data);
功能描述	编程数据选项字节
先决条件	ob_unlock
被调用函数	-
输入参数{in}	
data	待编程的数据
输出参数{out}	
-	-

返回值	
fmc_state_enum	FMC状态值，详情参考枚举变量 表3-287. 枚举类型 。

例如：

```
fmc_unlock();

ob_unlock();

/* program option bytes data */

fmc_state_enum fmc_state = ob_data_program(0xdd22);
```

函数 ob_user_get

函数ob_user_get描述见下表：

表 3-309. 函数 ob_user_get

函数名称	ob_user_get
函数原型	uint8_t ob_user_get(void);
功能描述	获取用户选项字节
先决条件	-
被调用函数	-
输入参数{in}	
-	-
输出参数{out}	
-	-
返回值	
uint8_t	选项字节用户数值（0x00 – 0xFF）

例如：

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

uint8_t user = ob_user_get();
```

函数 ob_data_get

函数ob_data_get描述见下表：

表 3-310. 函数 ob_data_get

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

输出参数{out}	
-	-
返回值	
uint16_t	选项字节数据值（0x0 – 0xFFFF）

例如：

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

```
uint16_t data = ob_data_get();
```

函数 ob_write_protection_get

函数ob_write_protection_get描述见下表：

表 3-311. 函数 ob_write_protection_get

函数名称	ob_write_protection_get
函数原型	uint32_t ob_write_protection_get(void);
功能描述	获取选项字节的擦/写保护位的值
先决条件	-
被调用函数	-
输入参数{in}	
-	-
输出参数{out}	
-	-
返回值	
uint16_t	选项字节写保护数值（0x0 – 0xFFFF FFFF）

例如：

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

```
uint32_t wp = ob_write_protection_get();
```

函数 ob_security_protection_flag_get

函数ob_security_protection_flag_get描述见下表：

表 3-312. 函数 ob_security_protection_flag_get

函数名称	ob_security_protection_flag_get
函数原型	FlagStatus ob_security_protection_flag_get(void);
功能描述	获取选项字节安全保护状态
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 FMC option byte security protection level */
```

```
FlagStatus spc_flag = ob_security_protection_flag_get();
```

函数 fmc_ecc_address_get

函数fmc_ecc_address_get描述见下表：

表 3-313. 函数 fmc_ecc_address_get

函数名称	fmc_ecc_address_get
函数原型	uint16_t fmc_ecc_address_get(void);
功能描述	获取发生ECC错误的地址
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 ~ 0x3FFF

例如：

```
/* get the address where ECC error occur on */
```

```
uint16_t ecc_addr = fmc_ecc_address_get();
```

函数 fmc_slp_unlock

函数fmc_slp_unlock描述见下表：

表 3-314. 函数 fmc_slp_unlock

函数名称	fmc_slp_unlock
函数原型	void fmc_slp_unlock(void);
功能描述	解锁对FMC_WS寄存器RUN_SLP位的操作
先决条件	-
被调用函数	-
输入参数{in}	
-	-
输出参数{out}	
-	-

返回值	
-	-

例如：

```
/* unlock RUN_SLP bit in FMC_WS register */
```

```
fmc_slp_unlock();
```

函数 fmc_sleep_slp_enable

函数fmc_sleep_slp_enable描述见下表：

表 3-315. 函数 fmc_sleep_slp_enable

函数名称	fmc_sleep_slp_enable
函数原型	void fmc_sleep_slp_enable(void);
功能描述	Flash进入睡眠模式当MCU进入深度睡眠模式或RUN_SLP位置位
先决条件	-
被调用函数	-
输入参数{in}	
-	-
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* flash enter sleep/power-down mode during MCU run/low-power run mode */
```

```
fmc_sleep_slp_enable();
```

函数 fmc_sleep_slp_disable

函数fmc_sleep_slp_disable描述见下表：

表 3-316. 函数 fmc_sleep_slp_disable

函数名称	fmc_sleep_slp_disable
函数原型	void fmc_enter_slp_disable(void);
功能描述	Flash进入掉电模式（GD32L233）\空闲模式（GD32L235）当MCU进入深度睡眠模式或RUN_SLP位置位
先决条件	-
被调用函数	-
输入参数{in}	
-	-
输出参数{out}	
-	-

返回值	
-	-

例如：

```
/* for GD32L233, flash enter power-down mode when MCU enter deep-sleep mode or
RUN_SLP bit is set */
```

```
fmc_sleep_slp_disable();
```

函数 fmc_run_slp_enable

函数fmc_run_slp_enable描述见下表：

表 3-317. 函数 fmc_run_slp_enable

函数名称	fmc_run_slp_enable
函数原型	void fmc_run_slp_enable(void);
功能描述	Flash进入空闲模式当MCU正常运行（GD32L235，和SLEEP_SLP位共同作用）\Flash进入空闲模式当MCU正常运行或低功耗运行（GD32L233，和SLEEP_SLP位共同作用）
先决条件	-
被调用函数	-
输入参数{in}	
-	-
输出参数{out}	
-	-
返回值	
-	-

例如：

```
fmc_slp_unlock();
```

```
/* for GD32L235, flash enter idle mode when MCU run mode (together with the SLEEP_SLP
bit) */
```

```
fmc_run_slp_enable();
```

函数 fmc_run_slp_disable

函数fmc_run_slp_disable描述见下表：

表 3-318. 函数 fmc_run_slp_disable

函数名称	fmc_run_slp_disable
函数原型	void fmc_run_slp_disable(void);
功能描述	Flash进入睡眠模式当MCU正常运行（GD32L235，和SLEEP_SLP位共同作用）\Flash进入睡眠模式或掉电模式当MCU正常运行或低功耗运行（GD32L233，和SLEEP_SLP位共同作用）

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

例如：

```
fmc_slp_unlock();
```

```
/* for GD32L235, flash enter sleep mode when MCU run mode (together with the SLEEP_SLP bit) */
```

```
fmc_run_slp_disable();
```

函数 fmc_sleep_mode_enter

函数fmc_sleep_mode_enter描述见下表：

表 3-319. 函数 fmc_run_slp_disable fmc_sleep_mode_enter

函数名称	fmc_run_slp_disable fmc_sleep_mode_enter
函数原型	void fmc_sleep_mode_enter(void);
功能描述	Flash进入睡眠模式当MCU正常运行。请注意该函数需要在SRAM中运行。
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 sleep mode when MCU run mode */
```

```
fmc_sleep_mode_enter ();
```

函数 fmc_pd_mode_enter

函数fmc_pd_mode_enter描述见下表：

表 3-320. 函数 fmc_pd_mode_enter

函数名称	fmc_pd_mode_enter
函数原型	void fmc_pd_mode_enter(void);

功能描述	Flash进入掉电模式当MCU正常运行或低功耗运行（仅适用于GD32L233xx）。请注意该函数需要在SRAM中运行。
先决条件	-
被调用函数	-
输入参数{in}	
-	-
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* for GD32L235, flash enter sleep mode when MCU run mode (together with the SLEEP_SLP bit) */
```

```
fmc_pd_mode_enter ();
```

函数 fmc_slp_mode_exit

函数fmc_slp_mode_exit描述见下表：

表 3-321. 函数 fmc_slp_mode_exit

函数名称	fmc_slp_mode_exit
函数原型	void fmc_slp_mode_exit(void);
功能描述	Flash退出睡眠模式或掉电模式当MCU正常运行或低功耗运行（GD32L233）\Flash退出睡眠模式当MCU正常运行（GD32L235）。请注意该函数需要在SRAM中运行。
先决条件	-
被调用函数	-
输入参数{in}	
-	-
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* for GD32L235, flash exit sleep mode when MCU run mode mode */
```

```
fmc_slp_mode_exit ();
```

函数 fmc_flag_get

函数fmc_flag_get描述见下表：

表 3-322. 函数 `fmc_flag_get`

函数名称	<code>fmc_flag_get</code>
函数原型	<code>FlagStatus fmc_flag_get(fmc_flag_enum flag);</code>
功能描述	检查FMC标志是否置位
先决条件	-
被调用函数	-
输入参数{in}	
flag	FMC标志，详情参考枚举变量 表3-287. 枚举类型fmc_flag_enum 。
输出参数{out}	
-	-
返回值	
FlagStatus	SET或RESET

例如：

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

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

函数 `fmc_flag_clear`

函数`fmc_flag_clear`描述见下表：

表 3-323. 函数 `fmc_flag_clear`

函数名称	<code>fmc_flag_clear</code>
函数原型	<code>void fmc_flag_clear(fmc_flag_enum flag);</code>
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
<code>FMC_FLAG_PGER</code> <code>R</code>	Flash编程错误标志
<code>FMC_FLAG_PGAE</code> <code>RR</code>	Flash编程对齐错误标志
<code>FMC_FLAG_WPER</code> <code>R</code>	Flash擦写保护错误标志
<code>FMC_FLAG_ECCC</code> <code>OR</code>	单个位检测并纠正错误标志（仅适用于GD32L235xx）
<code>FMC_FLAG_ECCD</code> <code>ET</code>	双位检测错误标志（仅适用于GD32L235xx）
<code>FMC_FLAG_SYSE</code> <code>CC</code>	系统存储ECC错误标志（仅适用于GD32L235xx）
<code>FMC_FLAG_MFEC</code> <code>C</code>	主闪存ECC错误标志（仅适用于GD32L235xx）

<i>FMC_FLAG_OTPE</i> CC	OTP ECC错误标志（仅适用于GD32L235xx）
<i>FMC_FLAG_OBEC</i> C	选项字节ECC错误标志（仅适用于GD32L235xx）
<i>FMC_FLAG_OBEC</i> CDET	加载选项字节到寄存器时双位检测ECC错误标志（仅适用于GD32L235xx）
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* clear FMC end flag */
fmc_flag_clear(FMC_FLAG_END);
```

函数 fmc_interrupt_enable

函数fmc_interrupt_enable描述见下表：

表 3-324. 函数 fmc_interrupt_enable

函数名称	fmc_interrupt_enable
函数原型	void fmc_interrupt_enable(fmc_interrupt_enum interrupt);
功能描述	使能FMC中断
先决条件	-
被调用函数	-
输入参数{in}	
interrupt	FMC中断，详情参考枚举变量 表3-289. 枚举类型fmc_interrupt_enum 。
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* enable FMC end interrupt */
fmc_interrupt_enable(FMC_INT_END);
```

函数 fmc_interrupt_disable

函数fmc_interrupt_disable描述见下表：

表 3-325. 函数 fmc_interrupt_disable

函数名称	fmc_interrupt_disable
函数原型	void fmc_interrupt_disable(fmc_interrupt_enum interrupt);

功能描述	禁能FMC中断
先决条件	-
被调用函数	-
输入参数{in}	
interrupt	FMC中断，详情参考枚举变量 表3-289. 枚举类型fmc_interrupt_enum 。
输出参数{out}	
-	-
返回值	
-	-

例如：

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

```
fmc_interrupt_disable(FMC_INT_END);
```

函数 fmc_interrupt_flag_get

函数fmc_interrupt_flag_get描述见下表：

表 3-326. 函数 fmc_interrupt_flag_get

函数名称	fmc_interrupt_flag_get
函数原型	FlagStatus fmc_interrupt_flag_get(fmc_interrupt_flag_enum int_flag);
功能描述	获取FMC中断标志状态
先决条件	-
被调用函数	-
输入参数{in}	
int_flag	中断标志，详情参考枚举变量 表3-288. 枚举类型fmc_interrupt_flag_enum 。
输出参数{out}	
-	-
返回值	
FlagStatus	SET或RESET

例如：

```
/* check FMC program operation error flag is set or not */
```

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

函数 fmc_interrupt_flag_clear

函数fmc_interrupt_flag_clear描述见下表：

表 3-327. 函数 fmc_interrupt_flag_clear

函数名称	fmc_interrupt_flag_clear
函数原型	void fmc_interrupt_flag_clear(fmc_interrupt_flag_enum int_flag);
功能描述	清除FMC中断标志

先决条件	-
被调用函数	-
输入参数{in}	
int_flag	中断标志，详情参考枚举变量 表3-288. 枚举类型fmc_interrupt_flag_enum 。
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* clear FMC program operation error flag */
```

```
fmc_interrupt_flag_get(FMC_INT_FLAG_PGERR);
```

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

3.13.1. 外设寄存器说明

FWDGT寄存器列表如下表所示：

表 3-328. FWDGT 寄存器

寄存器名称	寄存器描述
FWDGT_CTL	控制寄存器
FWDGT_PSC	预分频寄存器
FWDGT_RLD	重装载寄存器
FWDGT_STAT	状态寄存器
FWDGT_WND	窗口寄存器

3.13.2. 外设库函数说明

FWDGT库函数列表如下表所示：

表 3-329. 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	失能对寄存器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

库函数名称	库函数描述
fwdgt_reload_value_config	配置独立看门狗定时器计数器重装载值
fwdgt_window_value_config	配置独立看门狗定时器计数窗口值
fwdgt_counter_reload	按照FWDGT_RLD寄存器的值重装载FWDGT计数器
fwdgt_config	设置FWDGT重装载值、预分频值
fwdgt_flag_get	获取FWDGT标志位状态

函数 fwddgt_write_enable

函数fwddgt_write_enable描述见下表：

表 3-330. 函数 fwddgt_write_enable

函数名称	fwddgt_write_enable
函数原型	void fwddgt_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 and FWDGT_RLD and FWDGT_WND */
```

```
fwddgt_write_enable();
```

函数 fwddgt_write_disable

函数fwddgt_write_disable描述见下表：

表 3-331. 函数 fwddgt_write_disable

函数名称	fwddgt_write_disable
函数原型	void fwddgt_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 3-332. 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 3-333. 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_DIV4	FWDGT预分频值设为4
FWDGT_PSC_DIV8	FWDGT预分频值设为8
FWDGT_PSC_DIV16	FWDGT预分频值设为16
FWDGT_PSC_DIV32	FWDGT预分频值设为32

<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 / 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 3-334. 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 - 0xFFFF
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 3-335. 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 3-336. 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 3-337. 函数 fwdgt_config

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

例如:

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

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

函数 fwdgt_flag_get

函数fwdgt_flag_get描述见下表:

表 3-338. 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);
```

3.14. GPIO

GPIO用来实现各片上设备的逻辑输入/输出功能。章节[3.14.1](#)描述了GPIO的寄存器列表，章节[3.14.2](#)对GPIO库函数进行说明。

3.14.1. 外设寄存器说明

GPIO寄存器列表如下表所示:

表 3-339. 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

3.14.2. 外设库函数说明

GPIO库函数列表如下表所示:

表 3-340. 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_bit_get	获取引脚的输入值
gpio_input_port_get	获取一组端口的输入值
gpio_output_bit_get	获取引脚的输出值
gpio_output_port_get	获取一组端口的输出值
gpio_af_set	设置GPIO复用功能
gpio_pin_lock	相应的引脚配置被锁定
gpio_bit_toggle	翻转GPIO引脚状态
gpio_port_toggle	翻转一组GPIO状态

函数 gpio_deinit

函数gpio_deinit描述见下表：

表 3-341. 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,F)
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 3-342. 函数 gpio_mode_set

函数名称	gpio_mode_set
函数原型	void gpio_mode_set(uint32_t gpio_periph, uint32_t mode, uint32_t pull_up_down, uint32_t pin);
功能描述	设置GPIO模式
先决条件	-
被调用函数	-
输入参数{in}	
gpio_periph	GPIO端口
GPIOx	GPIOx(x = A,B,C,D,F)
输入参数{in}	
mode	GPIO引脚模式
GPIO_MODE_INPUT	输入模式
GPIO_MODE_OUTPUT	输出模式
GPIO_MODE_AF	备用功能模式
GPIO_MODE_ANALOG	模拟模式
输入参数{in}	
pull_up_down	GPIO引脚上拉下拉电阻设置
GPIO_PUPD_NONE	悬空模式，无上拉和下拉
GPIO_PUPD_PULLUP	带上拉电阻
GPIO_PUPD_PULLDOWN	带下拉电阻
输入参数{in}	
pin	GPIO pin
GPIO_PIN_x	引脚选择(x=0..15)
GPIO_PIN_ALL	所有引脚
输出参数{out}	
-	-
返回值	
-	-

例如：

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

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

函数 gpio_output_options_set

函数gpio_output_options_set描述见下表：

表 3-343. 函数 `gpio_output_options_set`

函数名称	<code>gpio_output_options_set</code>
函数原型	<code>void gpio_output_options_set(uint32_t gpio_periph, uint8_t otype, uint32_t speed, uint32_t pin);</code>
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}	
<code>gpio_periph</code>	GPIO端口
<code>GPIOx</code>	端口选择(x = A,B,C,D,F)
输入参数{in}	
<code>otype</code>	GPIO引脚输出模式
<code>GPIO_OTYPE_PP</code>	推挽输出模式
<code>GPIO_OTYPE_OD</code>	开漏输出模式
输入参数{in}	
<code>speed</code>	GPIO引脚输出最大速度
<code>GPIO_OSPEED_2MHZ</code>	最大输出速度为2MHz
<code>GPIO_OSPEED_10MHZ</code>	最大输出速度为10MHz
<code>GPIO_OSPEED_50MHZ</code>	最大输出速度为50MHz
输入参数{in}	
<code>pin</code>	GPIO引脚
<code>GPIO_PIN_x</code>	引脚选择(x=0..15)
<code>GPIO_PIN_ALL</code>	所有引脚
输出参数{out}	
-	-
返回值	
-	-

例如:

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

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

函数 `gpio_bit_set`

函数`gpio_bit_set`描述见下表:

表 3-344. 函数 `gpio_bit_set`

函数名称	<code>gpio_bit_set</code>
函数原型	<code>void gpio_bit_set(uint32_t gpio_periph, uint32_t pin);</code>

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

例如:

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

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

函数 gpio_bit_reset

函数gpio_bit_reset描述见下表:

表 3-345. 函数 gpio_bit_reset

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

例如:

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

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

函数 gpio_bit_write

函数gpio_bit_write描述见下表:

表 3-346. 函数 gpio_bit_write

函数名称	gpio_bit_write
函数原型	void gpio_bit_write(uint32_t gpio_periph, uint32_t pin, bit_status bit_value);
功能描述	将特定的值写入引脚
先决条件	-
被调用函数	-
输入参数{in}	
gpio_periph	GPIO端口
GPIOx	端口选择(x = A,B,C,D,F)
输入参数{in}	
pin	GPIO引脚
GPIO_PIN_x	引脚选择(x=0..15)
GPIO_PIN_ALL	所有引脚
输入参数{in}	
bit_value	设置或清除
RESET	清除引脚值
SET	设置引脚值
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* write 1 to PA0 */
```

```
gpio_bit_write(GPIOA, GPIO_PIN_0, SET);
```

函数 gpio_port_write

函数gpio_port_write描述见下表:

表 3-347. 函数 gpio_port_write

函数名称	gpio_port_write
函数原型	void gpio_port_write(uint32_t gpio_periph, uint16_t data);
功能描述	将特定的值写入端口
先决条件	-
被调用函数	-
输入参数{in}	
gpio_periph	GPIO端口

<i>GPIOx</i>	端口选择(x = A,B,C,D,F)
输入参数{in}	
data	将要写入的具体值
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* write 1010 0101 1010 0101 to Port A */
```

```
gpio_port_write(GPIOA, 0xA5A5);
```

函数 gpio_input_bit_get

函数gpio_input_bit_get描述见下表:

表 3-348. 函数 gpio_input_bit_get

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

例如:

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

```
FlagStatus bit_state;
```

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

函数 gpio_input_port_get

函数gpio_input_port_get描述见下表:

表 3-349. 函数 `gpio_input_port_get`

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

例如:

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

```
uint16_t port_state;
```

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

函数 `gpio_output_bit_get`

函数`gpio_output_bit_get`描述见下表:

表 3-350. 函数 `gpio_output_bit_get`

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

例如:

```
/* get 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 3-351. 函数 gpio_output_port_get

函数名称	gpio_output_port_get
函数原型	uint16_t gpio_output_port_get(uint32_t gpio_periph);
功能描述	获取引脚的输出值
先决条件	-
被调用函数	-
输入参数{in}	
gpio_periph	GPIO端口
GPIOx	端口选择(x = A,B,C,D,F)
输出参数{out}	
-	-
返回值	
uint16_t	0x0000-0xFFFF

例如:

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

```
uint16_t port_state;
```

```
port_state = gpio_output_port_get(GPIOA);
```

函数 gpio_af_set

函数gpio_af_set描述见下表:

表 3-352. 函数 gpio_af_set

函数名称	gpio_af_set
函数原型	void gpio_af_set(uint32_t gpio_periph, uint32_t alt_func_num, uint32_t pin);
功能描述	设置GPIO的备用功能
先决条件	-
被调用函数	-
输入参数{in}	
gpio_periph	GPIO端口
GPIOx	GPIOx(x = A,B,C,D,F)
输入参数{in}	
alt_func_num	GPIO 引脚备用功能, 请参见特定设备的数据手册
GPIO_AF_0	SPI0, SPI1, TIMER0, RTC_OUT, CK_OUT, SWDIO, SWCLK, LPTIMER1, TIMER14

<i>GPIO_AF_1</i>	<i>TIMER1, TIMER40, TIMER2, TIMER0, LPTIMER0, TIMER11, TIMER8, LPTIMER1, TIMER14</i>
<i>GPIO_AF_2</i>	<i>LPTIMER0, TIMER8, LPUART0, TIMER14, TIMER11, TIMER0</i>
<i>GPIO_AF_3</i>	<i>SLCD</i>
<i>GPIO_AF_4</i>	<i>I2C0, I2C1, I2C2, UART4, CAN</i>
<i>GPIO_AF_5</i>	<i>SPI0, SPI1, I2S1, I2C2</i>
<i>GPIO_AF_6</i>	<i>SPI1, CMP0, CMP1, I2S1, LPTIMER1, TIMER14</i>
<i>GPIO_AF_7</i>	<i>USART1, TIMER11, USART0, LPUART1, CAN, UART3, UART4, LPUART0</i>
<i>GPIO_AF_8</i>	<i>CTC, LPUART0, UART4, I2C1, CMP1, UART3, TIMER11, CAN, USART1</i>
<i>GPIO_AF_9</i>	<i>EVENTOUT</i>
<i>GPIO_AF_10</i>	<i>TIMER1, TIMER8, TIMER11, TIMER14, TIMER40, I2C1, LPUART1</i>
<i>GPIO_AF_11</i>	<i>TIMER0, TIMER40, I2C1, I2C2, CAN, TIMER11</i>
输入参数{in}	
pin	GPIO引脚
<i>GPIO_PIN_x</i>	引脚选择(x=0..15)
<i>GPIO_PIN_ALL</i>	所有引脚
输出参数{out}	
-	-
返回值	
-	-

例如:

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

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

函数 gpio_pin_lock

函数gpio_pin_lock描述见下表:

表 3-353. 函数 gpio_pin_lock

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

-	-
返回值	
-	-

例如:

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

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

函数 gpio_bit_toggle

函数gpio_bit_toggle描述见下表:

表 3-354. 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	GPIOx(x = A,B,C,D,F)
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 3-355. 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	GPIOx(x = A,B,C,D,F)
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* toggle GPIOA */
gpio_port_toggle(GPIOA);
```

3.15. 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[3.15.1](#)描述了I2C的寄存器列表，章节[3.15.2](#)对I2C库函数进行说明。

3.15.1. 外设寄存器说明

I2C寄存器列表如下表所示:

表 3-356. 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

3.15.2. 外设库函数说明

I2C库函数列表如下表所示:

表 3-357. I2C 库函数

库函数名称	库函数描述
i2c_deinit	复位外设 I2C
i2c_timing_config	配置时序参数
i2c_digital_noise_filter_config	配置数字噪声过滤器
i2c_analog_noise_filter_enable	使能数字噪声过滤器
i2c_analog_noise_filter_disable	禁能数字噪声过滤器
i2c_master_clock_config	配置主机模式下 SCL 高低电平时钟周期
i2c_master_addressing	配置 I2C 从机地址以及数据传输方向
i2c_address10_header_enable	主机接收模式下，10 位地址头只执行读操作
i2c_address10_header_disable	主机接收模式下，10 位地址头执行完整的读操作序列
i2c_address10_enable	使能主机模式下 10 位地址寻址模式
i2c_address10_disable	禁能主机模式下 10 位地址寻址模式
i2c_automatic_end_enable	使能主机模式下 I2C 自动结束模式
i2c_automatic_end_disable	禁能主机模式下 I2C 自动结束模式
i2c_slave_response_to_gcall_enable	使能从机响应广播呼叫
i2c_slave_response_to_gcall_disable	禁能从机响应广播呼叫
i2c_stretch_scl_low_enable	当从机数据没有准备好时拉低 SCL
i2c_stretch_scl_low_disable	当从机数据没有准备好时不拉低 SCL
i2c_address_config	配置 I2C 从机地址
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_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

库函数名称	库函数描述
i2c_pec_enable	使能报文错误校验
i2c_pec_disable	禁能报文错误校验
i2c_pec_value_get	获取报文错误校验值
i2c_smbus_alert_enable	使能 SMBus 报警
i2c_smbus_alert_disable	禁能 SMBus 报警
i2c_smbus_default_addr_enable	使能 SMBus 设备默认地址
i2c_smbus_default_addr_disable	禁能 SMBus 设备默认地址
i2c_smbus_host_addr_enable	使能 SMBus 主机地址
i2c_smbus_host_addr_disable	禁能 SMBus 主机地址
i2c_extented_clock_timeout_enable	使能时钟信号延展超时检测
i2c_extented_clock_timeout_disable	禁能时钟信号延展超时检测
i2c_clock_timeout_enable	使能时钟超时检测
i2c_clock_timeout_disable	禁能时钟超时检测
i2c_bus_timeout_b_config	配置总线超时 B
i2c_bus_timeout_a_config	配置总线超时 A
i2c_idle_clock_timeout_config	配置空闲时钟超时检测
i2c_flag_get	获取 I2C 标志位
i2c_flag_clear	清除 I2C 标志位
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 3-358. 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 3-359. 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)
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 3-360. 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)
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 3-361. 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)
输入参数{in}	
filter_length	过滤长度
FILTER_DISABLE	数字噪声过滤器禁能
FILTER_LENGTH_1	数字噪声滤波使能并且可以滤除脉宽宽度不大于 1 t _{I2CCLK} 的尖峰
FILTER_LENGTH_2	数字噪声滤波使能并且可以滤除脉宽宽度不大于 2 t _{I2CCLK} 的尖峰
FILTER_LENGTH_3	数字噪声滤波使能并且可以滤除脉宽宽度不大于 3 t _{I2CCLK} 的尖峰
FILTER_LENGTH_4	数字噪声滤波使能并且可以滤除脉宽宽度不大于 4 t _{I2CCLK} 的尖峰
FILTER_LENGTH_5	数字噪声滤波使能并且可以滤除脉宽宽度不大于 5 t _{I2CCLK} 的尖峰
FILTER_LENGTH_6	数字噪声滤波使能并且可以滤除脉宽宽度不大于 6 t _{I2CCLK} 的尖峰
FILTER_LENGTH_7	数字噪声滤波使能并且可以滤除脉宽宽度不大于 7 t _{I2CCLK} 的尖峰
FILTER_LENGTH_8	数字噪声滤波使能并且可以滤除脉宽宽度不大于 8 t _{I2CCLK} 的尖峰
FILTER_LENGTH_9	数字噪声滤波使能并且可以滤除脉宽宽度不大于 9 t _{I2CCLK} 的尖峰
FILTER_LENGTH_10	数字噪声滤波使能并且可以滤除脉宽宽度不大于 10 t _{I2CCLK} 的尖峰
FILTER_LENGTH_11	数字噪声滤波使能并且可以滤除脉宽宽度不大于 11 t _{I2CCLK} 的尖峰
FILTER_LENGTH_12	数字噪声滤波使能并且可以滤除脉宽宽度不大于 12 t _{I2CCLK} 的尖峰
FILTER_LENGTH_13	数字噪声滤波使能并且可以滤除脉宽宽度不大于 13 t _{I2CCLK} 的尖峰
FILTER_LENGTH_14	数字噪声滤波使能并且可以滤除脉宽宽度不大于 14 t _{I2CCLK} 的尖峰
FILTER_LENGTH_15	数字噪声滤波使能并且可以滤除脉宽宽度不大于 15 t _{I2CCLK} 的尖峰
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* I2C0 digital filter filters spikes with a length of up to 1 tI2CCLK */
```

```
i2c_digital_noise_filter_config(I2C0, FILTER_LENGTH_1);
```

函数 i2c_analog_noise_filter_enable

函数i2c_analog_noise_filter_enable描述见下表：

表 3-362. 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)
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 3-363. 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)
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 3-364. 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)
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 3-365. 函数 i2c_master_addressing

函数名称	i2c_master_addressing
函数原型	void i2c_master_addressing(uint32_t i2c_periph, uint32_t address, uint32_t trans_direction);
功能描述	配置 I2C 从机地址以及数据传输方向
先决条件	-
被调用函数	-
输入参数{in}	

i2c_periph	I2C 外设
<i>I2Cx</i>	(x=0,1,2)
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_TRANSMIT</i>	主机发送
<i>I2C_MASTER_RECEIVE</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 3-366. 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)
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 3-367. 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)
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 3-368. 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)
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 3-369. 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)
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 3-370. 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)
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 3-371. 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)
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 3-372. 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)
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 3-373. 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)
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 3-374. 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)
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 3-375. 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)
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 3-376. 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)
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 3-377. 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)
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 3-378. 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)
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 3-379. 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)
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_BIT_1	ADDRESS2[1]屏蔽， ADDRESS2[7:2]进行比较
ADDRESS2_MASK_BIT_1_2	ADDRESS2[2:1]屏蔽， ADDRESS2[7:3]进行比较
ADDRESS2_MASK_BIT_1_3	ADDRESS2[3:1]屏蔽， ADDRESS2[7:4]进行比较
ADDRESS2_MASK_BIT_1_4	ADDRESS2[4:1]屏蔽， ADDRESS2[7:5]进行比较
ADDRESS2_MASK_BIT_1_5	ADDRESS2[5:1]屏蔽， ADDRESS2[7:6]进行比较
ADDRESS2_MASK_BIT_1_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_BIT_1_2);
```

函数 i2c_second_address_disable

函数i2c_second_address_disable描述见下表：

表 3-380. 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)
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 3-381. 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)
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 3-382. 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)
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 3-383. 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)
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 3-384. 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)
输出参数{out}	

-	-
返回值	
-	-

例如：

```
/* generate a NACK in slave mode */
```

```
i2c_nack_enable(I2C0);
```

函数 i2c_wakeup_from_deepsleep_enable

函数i2c_wakeup_from_deepsleep_enable描述见下表：

表 3-385. 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)
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 3-386. 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)

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 3-387. 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)
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 3-388. 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)
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 3-389. 函数 i2c_start_on_bus

函数名称	i2c_start_on_bus
函数原型	void i2c_start_on_bus(uint32_t i2c_periph);
功能描述	在 I2C 总线上生成起始位
先决条件	-
被调用函数	-
输入参数{in}	
i2c_periph	I2C 外设
I2Cx	(x=0,1,2)
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* I2C0 send a start condition to I2C bus */
```

```
i2c_start_on_bus(I2C0);
```

函数 i2c_stop_on_bus

函数i2c_stop_on_bus描述见下表：

表 3-390. 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
<i>I2Cx</i>	(x=0,1,2)
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* I2C0 generate a STOP condition to I2C bus */
```

```
i2c_stop_on_bus(I2C0);
```

函数 i2c_data_transmit

函数i2c_data_transmit描述见下表：

表 3-391. 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
<i>I2Cx</i>	(x=0,1,2)
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 3-392. 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)
输出参数{out}	
-	-
返回值	
uint32_t	0x0000..0x00FF

例如：

```
/* I2C0 receive data */
```

```
uint32_t i2c_receiver;
```

```
i2c_receiver = i2c_data_receive(I2C0);
```

函数 i2c_reload_enable

函数i2c_reload_enable描述见下表：

表 3-393. 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)
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 3-394. 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)
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 3-395. 函数 i2c_transfer_byte_number_config

函数名称	i2c_transfer_byte_number_config
函数原型	void i2c_transfer_byte_number_config(uint32_t i2c_periph, uint32_t byte_number);
功能描述	配置待发送字节数
先决条件	-
被调用函数	-
输入参数{in}	
i2c_periph	I2C 外设
I2Cx	(x=0,1,2)
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 3-396. 函数 i2c_dma_enable

函数名称	i2c_dma_enable
函数原型	void i2c_dma_enable(uint32_t i2c_periph, uint8_t dma);
功能描述	使能发送/接收模式下 DMA
先决条件	-
被调用函数	-
输入参数{in}	
i2c_periph	I2C 外设
I2Cx	(x=0,1,2)
输入参数{in}	
dma	I2C DMA
I2C_DMA_TRANSMIT	采用 DMA 方式发送数据
I2C_DMA_RECEIVE	采用 DMA 方式接收数据
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* enable I2C DMA for transmission or reception */
```

```
i2c_dma_enable(I2C0, I2C_DMA_RECEIVE);
```

函数 i2c_dma_disable

函数i2c_dma_disable描述见下表:

表 3-397. 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)
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 3-398. 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)
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 3-399. 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)

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 3-400. 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)
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 3-401. 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)
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 3-402. 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)
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 3-403. 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)
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 3-404. 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)
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 3-405. 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	禁用 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)
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 3-406. 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)
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 3-407. 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)
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 3-408. 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)
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 3-409. 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)
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 3-410. 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)
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 3-411. 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)
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 3-412. 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)
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 3-413. 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)
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 3-414. 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)
输入参数{in}	
timeout	总线超时 A
BUSTOA_DETECT_SCL_LOW	BUSTOA 用于检测 SCL 低电平超时
BUSTOA_DETECT_IDLE	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 3-415. 函数 i2c_flag_get

函数名称	i2c_flag_get
函数原型	FlagStatus i2c_flag_get(uint32_t i2c_periph, uint32_t flag);
功能描述	获取 I2C 标志位
先决条件	-
被调用函数	-
输入参数{in}	
i2c_periph	I2C 外设
I2Cx	(x=0,1,2)
输入参数{in}	
flag	I2C 标志位
I2C_FLAG_TBE	发送期间 I2C_TDATA 寄存器空标志
I2C_FLAG_TI	发送中断标志
I2C_FLAG_RBNE	接收期间 I2C_RDATA 非空标志
I2C_FLAG_ADDSEND	从机模式下，接收到的地址与自身地址匹配
I2C_FLAG_NACK	NACK 标志
I2C_FLAG_STPDET	从机模式下检测到 STOP 信号
I2C_FLAG_TC	主机模式下传输完成标志
I2C_FLAG_TCR	传输完成重载标志
I2C_FLAG_BERR	总线错误标志
I2C_FLAG_LOSTARB	仲裁丢失标志
I2C_FLAG_OUERR	从机模式下，过载/欠载错误标志
I2C_FLAG_PECERR	PEC 错误标志
I2C_FLAG_TIMEOUT	超时标志
I2C_FLAG_SMBALT	SMBus 报警标志
I2C_FLAG_I2CBSY	忙标志
I2C_FLAG_TR	从机模式下，I2C 作为发送器还是接收器标志
输出参数{out}	
-	-
返回值	
FlagStatus	SET / RESET

例如：

```
/* get I2C flag status */
```

```
FlagStatus flag_state = RESET;
```

```
flag_state = i2c_flag_get(I2C0, I2C_FLAG_TBE);
```

函数 i2c_flag_clear

函数i2c_flag_clear描述见下表：

表 3-416. 函数 i2c_flag_clear

函数名称	i2c_flag_clear
函数原型	void i2c_flag_clear(uint32_t i2c_periph, uint32_t flag);
功能描述	清除 I2C 标志位
先决条件	-
被调用函数	-
输入参数{in}	
i2c_periph	I2C 外设
I2Cx	(x=0,1,2)
输入参数{in}	
flag	I2C 标志位
I2C_FLAG_ADDSEND	从机模式下，接收到的地址与自身地址匹配
I2C_FLAG_NACK	NACK 标志
I2C_FLAG_STPDET	从机模式下检测到 STOP 信号
I2C_FLAG_BERR	总线错误标志
I2C_FLAG_LOSTARB	仲裁丢失标志
I2C_FLAG_OUERR	从机模式下，过载/欠载错误标志
I2C_FLAG_PECERR	PEC 错误标志
I2C_FLAG_TIMEOUT	超时标志
I2C_FLAG_SMBALT	SMBus 报警标志
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* clear a bus error flag*/
```

```
i2c_flag_clear(I2C0, I2C_FLAG_BERR);
```

函数 i2c_interrupt_enable

函数i2c_interrupt_enable描述见下表：

表 3-417. 函数 i2c_interrupt_enable

函数名称	i2c_interrupt_enable
函数原型	void i2c_interrupt_enable(uint32_t i2c_periph, uint32_t interrupt);

功能描述	中断使能
先决条件	-
被调用函数	-
输入参数{in}	
i2c_periph	I2C 外设
<i>I2Cx</i>	(x=0,1,2)
输入参数{in}	
interrupt	I2C 中断
<i>I2C_INT_ERR</i>	错误中断
<i>I2C_INT_TC</i>	发送完成中断
<i>I2C_INT_STPDET</i>	检测到 STOP 中断
<i>I2C_INT_NACK</i>	接收到 NACK 中断
<i>I2C_INT_ADDM</i>	地址匹配中断
<i>I2C_INT_RBNE</i>	接收中断
<i>I2C_INT_TI</i>	发送中断
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* enable I2C0 transmit interrupt */
```

```
i2c_interrupt_enable(I2C0, I2C_INT_TI);
```

函数 i2c_interrupt_disable

函数i2c_interrupt_disable描述见下表：

表 3-418. 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
<i>I2Cx</i>	(x=0,1,2)
输入参数{in}	
interrupt	I2C 中断
<i>I2C_INT_ERR</i>	错误中断
<i>I2C_INT_TC</i>	发送完成中断
<i>I2C_INT_STPDET</i>	检测到 STOP 中断
<i>I2C_INT_NACK</i>	接收到 NACK 中断

<i>I2C_INT_ADDM</i>	地址匹配中断
<i>I2C_INT_RBNE</i>	接收中断
<i>I2C_INT_TI</i>	发送中断
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* 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 3-419. 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)
输入参数{in}	
int_flag	I2C 中断标志位，参考 表 3-358. 枚举类型 i2c_interrupt_flag_enum 。
<i>I2C_INT_FLAG_TI</i>	发送中断标志
<i>I2C_INT_FLAG_RBNE</i>	接收期间 I2C_RDATA 非空中断标志
<i>I2C_INT_FLAG_ADDS END</i>	从机模式下，接收到的地址与自身地址匹配中断标志
<i>I2C_INT_FLAG_NACK</i>	NACK 中断标志
<i>I2C_INT_FLAG_STPD ET</i>	从机模式下检测到 STOP 信号中断标志
<i>I2C_INT_FLAG_TC</i>	主机模式下传输完成中断标志
<i>I2C_INT_FLAG_TCR</i>	传输完成重载中断标志
<i>I2C_INT_FLAG_BERR</i>	总线错误中断标志
<i>I2C_INT_FLAG_LOSTA RB</i>	仲裁丢失中断标志
<i>I2C_INT_FLAG_OUER R</i>	从机模式下，过载/欠载错误中断标志
<i>I2C_INT_FLAG_PEC</i>	PEC 错误中断标志

<i>RR</i>	
<i>I2C_INT_FLAG_TTIMEO</i> <i>UT</i>	超时中断标志
<i>I2C_INT_FLAG_SMBA</i> <i>LT</i>	SMBus 报警中断标志
输出参数{out}	
-	-
返回值	
FlagStatus	SET / RESET

例如：

```
/* get I2C interrupt flag status */
```

```
FlagStatus flag_state = RESET;
```

```
flag_state = i2c_interrupt_flag_get(I2C0, I2C_INT_FLAG_TI);
```

函数 i2c_interrupt_flag_clear

函数i2c_interrupt_flag_clear描述见下表：

表 3-420. 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
<i>I2Cx</i>	(x=0,1,2)
输入参数{in}	
int_flag	I2C 中断标志位，参考 表 3-358. 枚举类型 i2c_interrupt_flag_enum 。
<i>I2C_INT_FLAG_ADDS</i> <i>END</i>	从机模式下，接收到的地址与自身地址匹配中断标志
<i>I2C_INT_FLAG_NACK</i>	NACK 中断标志
<i>I2C_INT_FLAG_STPD</i> <i>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</i> <i>RB</i>	仲裁丢失中断标志
<i>I2C_INT_FLAG_OUER</i> <i>R</i>	从机模式下，过载/欠载错误中断标志
<i>I2C_INT_FLAG_PECE</i>	PEC 错误中断标志

<i>RR</i>	
<i>I2C_INT_FLAG_TIMEO</i> <i>UT</i>	超时中断标志
<i>I2C_INT_FLAG_SMBA</i> <i>LT</i>	SMBus 报警中断标志
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* clear a bus error flag */
```

```
i2c_interrupt_flag_clear(I2C0, I2C_INT_FLAG_BERR);
```

3.16. 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 [3.16.1](#) 描述了 LPTIMER 的寄存器列表，章节 [3.16.2](#) 对 LPTIMER 库函数进行说明。

3.16.1. 外设寄存器说明

LPTIMER 寄存器列表如下表所示:

表 3-421. 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
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

3.16.2. 外设库函数说明

TIMER 库函数列表如下表所示:

表 3-422. TIMER 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_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
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 3-423. 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,

成员名称	功能描述
	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)
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, LPTIMER_COUNTER_EXTERNAL)

函数 lptimer_deinit

函数lptimer_deinit描述见下表:

表 3-424. 函数 lptimer_deinit

函数名称	lptimer_deinit
函数原型	void lptimer_deinit(uint32_t lptimer_periph);
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}	
uint32_t lptimer_periph	LPTIMER外设
LPTIMER	GD32L233xx系列
LPTIMER0 / 1	GD32L235xx系列
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* reset LPTIMER */
```

```
lptimer_deinit(LPTIMER);
```

函数 lptimer_struct_para_init

函数lptimer_struct_para_init描述见下表：

表 3-425. 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3-423. 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 3-426. 函数 lptimer_init

函数名称	lptimer_init
函数原型	void lptimer_init(uint32_t lptimer_periph, 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}	
uint32_t lptimer_periph	LPTIMER外设
LPTIMER	GD32L233xx系列
LPTIMER0 / 1	GD32L235xx系列

输入参数{in}	
initpara	LPTIMER初始化结构体，结构体成员参考 表3-423. 结构体 <i>lptimer_parameter_struct</i>
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, &lptimer_structure);
```

函数 `lptimer_inputremap`

函数 `lptimer_inputremap` 描述见下表：

表 3-427. 函数 `lptimer_inputremap`

函数名称	<code>lptimer_inputremap</code>
函数原型	<code>void lptimer_inputremap(uint32_t lptimer_periph, uint32_t input0remap, uint32_t input1remap);</code>
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}	
uint32_t lptimer_periph	LPTIMER 外设
LPTIMER	GD32L233xx 系列

LPTIMER0 / 1	GD32L235xx系列
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
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
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* configure LPTIMER inputs is connected to GPIO */
```

```
lptimer_inputremap(LPTIMER, LPTIMER_INPUT0_GPIO, LPTIMER_INPUT1_GPIO);
```

函数 lptimer_register_shadow_enable

函数lptimer_register_shadow_enable描述见下表：

表 3-428. 函数 lptimer_register_shadow_enable

函数名称	lptimer_register_shadow_enable
函数原型	void lptimer_register_shadow_enable(uint32_t lptimer_periph, uint32_t lptimer_periph);
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}	
uint32_t lptimer_periph	LPTIMER外设
LPTIMER	GD32L233xx系列
LPTIMER0 / 1	GD32L235xx系列
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* enable the LPTIMER_CAR and LPTIMER_CMPV registers shadow function */
```

```
lptimer_register_shadow_enable(LPTIMER);
```

函数 lptimer_register_shadow_disable

函数lptimer_register_shadow_disable描述见下表：

表 3-429. 函数 lptimer_register_shadow_disable

函数名称	lptimer_register_shadow_disable
函数原型	void lptimer_register_shadow_disable(uint32_t lptimer_periph, uint32_t lptimer_periph);
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}	
uint32_t lptimer_periph	LPTIMER外设
LPTIMER	GD32L233xx系列
LPTIMER0 / 1	GD32L235xx系列
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* disable the LPTIMER_CAR and LPTIMER_CMPV registers shadow function */
```

```
lptimer_register_shadow_disable(LPTIMER);
```

函数 lptimer_timeout_enable

函数lptimer_timeout_enable描述见下表：

表 3-430. 函数 lptimer_timeout_enable

函数名称	lptimer_timeout_enable
函数原型	void lptimer_timeout_enable(uint32_t lptimer_periph, uint32_t lptimer_periph);
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}	
uint32_t lptimer_periph	LPTIMER外设
LPTIMER	GD32L233xx系列
LPTIMER0 / 1	GD32L235xx系列

输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* enable the LPTIMER TIMEOUT function */
```

```
lptimer_timeout_enable(LPTIMER);
```

函数 lptimer_timeout_disable

函数lptimer_timeout_disable描述见下表：

表 3-431. 函数 lptimer_register_shadow_disable

函数名称	lptimer_timeout_disable
函数原型	void lptimer_timeout_disable(uint32_t lptimer_periph, uint32_t lptimer_periph);
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}	
uint32_t lptimer_periph	LPTIMER外设
LPTIMER	GD32L233xx系列
LPTIMER0 / 1	GD32L235xx系列
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* disable the LPTIMER TIMEOUT function */
```

```
lptimer_timeout_disable(LPTIMER);
```

函数 lptimer_countinue_start

函数lptimer_countinue_start描述见下表：

表 3-432. 函数 lptimer_countinue_start

函数名称	lptimer_countinue_start
函数原型	void lptimer_countinue_start(uint32_t lptimer_periph, 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}	
uint32_t lptimer_periph	LPTIMER 外设
LPTIMER	GD32L233xx 系列
LPTIMER0 / 1	GD32L235xx 系列
输入参数{in}	
autoreload	自动重载寄存器值，0~0xFFFF 用于 GD32L235xx，0~0xFFFFFFFF 用于 GD32L233xx
输入参数{in}	
compare	比较寄存器值，0~0xFFFF 用于 GD32L235xx，0~0xFFFFFFFF 用于 GD32L233xx
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* LPTIMER countinue start */
```

```
lptimer_countinue_start(LPTIMER, 0x0000FFFF, 0x00007FFF);
```

函数 lptimer_single_start

函数 lptimer_single_start 描述见下表：

表 3-433. 函数 lptimer_countinue_start

函数名称	lptimer_single_start
函数原型	void lptimer_single_start(uint32_t lptimer_periph, 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}	
uint32_t lptimer_periph	LPTIMER 外设
LPTIMER	GD32L233xx 系列
LPTIMER0 / 1	GD32L235xx 系列
输入参数{in}	
autoreload	自动重载寄存器值，0~0xFFFF 用于 GD32L235xx，0~0xFFFFFFFF 用于 GD32L233xx
输入参数{in}	
compare	比较寄存器值，0~0xFFFF 用于 GD32L235xx，0~0xFFFFFFFF 用于 GD32L233xx

输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* LPTIMER single start */
lptimer_single_start(LPTIMER, 0x0000FFFF, 0x00007FFF);
```

函数 lptimer_stop

函数lptimer_stop描述见下表:

表 3-434. 函数 lptimer_stop

函数名称	lptimer_stop
函数原型	void lptimer_stop(uint32_t lptimer_periph);
功能描述	LPTIMER停止计数
先决条件	-
被调用函数	-
输入参数{in}	
uint32_t lptimer_periph	LPTIMER外设
LPTIMER	GD32L233xx系列
LPTIMER0 / 1	GD32L235xx系列
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* stop LPTIMER */
lptimer_stop(LPTIMER);
```

函数 lptimer_counter_read

函数lptimer_counter_read描述见下表:

表 3-435. 函数 lptimer_counter_read

函数名称	lptimer_counter_read
函数原型	uint32_t lptimer_counter_read(uint32_t lptimer_periph);
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}	
uint32_t lptimer_periph	LPTIMER外设
<i>LPTIMER</i>	GD32L233xx系列
<i>LPTIMER0 / 1</i>	GD32L235xx系列
输出参数{out}	
-	-
返回值	
uint32_t	计数器值（0~0xFFFF用于GD32L235xx，0~0xFFFFFFFF用于GD32L233xx）

例如：

```
/* read LPTIMER current counter value */
```

```
uint32_t i = 0;
```

```
i = lptimer_counter_read(LPTIMER);
```

函数 lptimer_autoreload_read

函数lptimer_autoreload_read描述见下表：

表 3-436. 函数 lptimer_autoreload_read

函数名称	lptimer_autoreload_read
函数原型	uint32_t lptimer_autoreload_read(uint32_t lptimer_periph);
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}	
uint32_t lptimer_periph	LPTIMER外设
<i>LPTIMER</i>	GD32L233xx系列
<i>LPTIMER0 / 1</i>	GD32L235xx系列
输出参数{out}	
-	-
返回值	
uint32_t	自动重载寄存器值（0~0xFFFF用于GD32L235xx，0~0xFFFFFFFF用于GD32L233xx）

例如：

```
/* read LPTIMER auto reload value */
```

```
uint32_t i = 0;
```

```
i = lptimer_autoreload_read(LPTIMER);
```

函数 lptimer_compare_read

函数lptimer_compare_read描述见下表:

表 3-437. 函数 lptimer_compare_read

函数名称	lptimer_compare_read
函数原型	uint32_t lptimer_compare_read(uint32_t lptimer_periph);
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}	
uint32_t lptimer_periph	LPTIMER外设
LPTIMER	GD32L233xx系列
LPTIMER0 / 1	GD32L235xx系列
输出参数{out}	
-	-
返回值	
uint32_t	比较寄存器值（0~0xFFFF用于GD32L235xx，0~0xFFFFFFFF用于GD32L233xx）

例如:

```
/* read LPTIMER compare value */
uint32_t i = 0;
i = lptimer_compare_read(LPTIMER);
```

函数 lptimer_autoreload_value_config

函数lptimer_autoreload_value_config描述见下表:

表 3-438. 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 lptimer_periph, 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}	
uint32_t lptimer_periph	LPTIMER外设
LPTIMER	GD32L233xx系列
LPTIMER0 / 1	GD32L235xx系列
输入参数{in}	

autoreload	自动重载寄存器值（0~0xFFFF用于GD32L235xx，0~0xFFFFFFFF用于GD32L233xx）
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* configure LPTIMER autoreload register value */
```

```
lptimer_autoreload_value_config(LPTIMER, 0x000000FF);
```

函数 lptimer_compare_value_config

函数lptimer_compare_value_config描述见下表：

表 3-439. 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 lptimer_periph, 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}	
uint32_t lptimer_periph	LPTIMER外设
<i>LPTIMER</i>	GD32L233xx系列
<i>LPTIMER0 / 1</i>	GD32L235xx系列
输入参数{in}	
autoreload	比较寄存器值（0~0xFFFF用于GD32L235xx，0~0xFFFFFFFF用于GD32L233xx）
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* configure LPTIMER compare value */
```

```
lptimer_compare_value_config(LPTIMER, 0x000000FF);
```

函数 lptimer_decodemode0_enable

函数lptimer_decodemode0_enable描述见下表：

表 3-440. 函数 `lptimer_decodemode0_enable`

函数名称	<code>lptimer_decodemode0_enable</code>
函数原型	<code>void lptimer_decodemode0_enable(uint32_t lptimer_periph);</code>
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}	
uint32_t lptimer_periph	LPTIMER外设
<i>LPTIMER</i>	GD32L233xx系列
<i>LPTIMER0 / 1</i>	GD32L235xx系列
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* enable decode mode 0 */
```

```
lptimer_decodemode0_enable(LPTIMER);
```

函数 `lptimer_decodemode1_enable`

函数`lptimer_decodemode1_enable`描述见下表：

表 3-441. 函数 `lptimer_decodemode1_enable`

函数名称	<code>lptimer_decodemode1_enable</code>
函数原型	<code>void lptimer_decodemode1_enable(uint32_t lptimer_periph);</code>
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}	
uint32_t lptimer_periph	LPTIMER外设
<i>LPTIMER</i>	GD32L233xx系列
<i>LPTIMER0 / 1</i>	GD32L235xx系列
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* enable decode mode 1 */
```

```
lptimer_decodemode1_enable(LPTIMER);
```

函数 lptimer_decodemode_disable

函数lptimer_decodemode_disable描述见下表：

表 3-442. 函数 lptimer_decodemode_disable

函数名称	lptimer_decodemode_disable
函数原型	void lptimer_decodemode_disable(uint32_t lptimer_periph);
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}	
uint32_t lptimer_periph	LPTIMER外设
LPTIMER	GD32L233xx系列
LPTIMER0 / 1	GD32L235xx系列
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* disable decode mode 0/1 */
```

```
lptimer_decodemode_disable(LPTIMER);
```

函数 lptimer_highlevelcounter_enable

函数lptimer_highlevelcounter_enable描述见下表：

表 3-443. 函数 lptimer_highlevelcounter_enable

函数名称	lptimer_highlevelcounter_enable
函数原型	void lptimer_highlevelcounter_enable(uint32_t lptimer_periph, 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}	
uint32_t lptimer_periph	LPTIMER外设
LPTIMER	GD32L233xx系列
LPTIMER0 / 1	GD32L235xx系列
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(LPTIMER, 0x00007FFF);
```

函数 lptimer_highlevelcounter_disable

函数lptimer_highlevelcounter_disable描述见下表：

表 3-444. 函数 lptimer_highlevelcounter_disable

函数名称	lptimer_highlevelcounter_disable
函数原型	void lptimer_highlevelcounter_disable(uint32_t lptimer_periph);
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}	
uint32_t lptimer_periph	LPTIMER外设
LPTIMER	GD32L233xx系列
LPTIMER0 / 1	GD32L235xx系列
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* disable external input high level counter */
```

```
lptimer_highlevelcounter_disable(LPTIMER);
```

函数 lptimer_flag_get

函数lptimer_flag_get描述见下表：

表 3-445. 函数 lptimer_flag_get

函数名称	lptimer_flag_get
函数原型	FlagStatus lptimer_flag_get(uint32_t lptimer_periph, uint32_t flag);
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}	
uint32_t	LPTIMER外设

lptimer_periph	
<i>LPTIMER</i>	GD32L233xx 系列
<i>LPTIMER0 / 1</i>	GD32L235xx 系列
输入参数{in}	
flag	LPTIMER 的标志位
<i>LPTIMER_FLAG_CMPVM</i>	比较寄存器匹配标志位
<i>LPTIMER_FLAG_CARM</i>	计数器自动重载寄存器匹配标志位
<i>LPTIMER_FLAG_ETEDEV</i>	外部触发边沿事件标志位
<i>LPTIMER_FLAG_CMPVUP</i>	比较寄存器更新标志位
<i>LPTIMER_FLAG_CARUP</i>	计数器自动重载寄存器更新标志位
<i>LPTIMER_FLAG_UP</i>	LPTIMER计数器由向下计数改为向上计数标志位
<i>LPTIMER_FLAG_DOWN</i>	LPTIMER计数器由向上计数改为向下计数标志位
<i>LPTIMER_FLAG_LCMVUP</i>	输入高电平计数最大值寄存器更新标志位
<i>LPTIMER_FLAG_INHLCO</i>	LPTIMER_INx (x=0,1) 高电平计数器溢出标志位
<i>LPTIMER_FLAG_INHLOE</i>	LPTIMER_IN0 和 LPTIMER_IN1 高电平重叠错误标志位
<i>LPTIMER_FLAG_INRFOE</i>	LPTIMER_IN0 和 LPTIMER_IN1 下降沿和上升沿重叠错误标志位
<i>LPTIMER_FLAG_IN0E</i>	LPTIMER_IN0错误标志位
<i>LPTIMER_FLAG_IN1E</i>	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, LPTIMER_FLAG_CMPVM);
```

函数 `lptimer_flag_clear`

函数 `lptimer_flag_clear` 描述见下表：

表 3-446. 函数 `timer_flag_clear`

函数名称	<code>lptimer_flag_clear</code>
函数原型	<code>void lptimer_flag_clear(uint32_t lptimer_periph, uint32_t flag);</code>
功能描述	清除LPTIMER标志位
先决条件	-
被调用函数	-
输入参数{in}	
<code>uint32_t lptimer_periph</code>	LPTIMER外设
LPTIMER	GD32L233xx系列
LPTIMER0 / 1	GD32L235xx系列
输入参数{in}	
flag	LPTIMER的标志位
LPTIMER_FLAG_CMPVM	比较寄存器匹配标志位
LPTIMER_FLAG_CARM	计数器自动重载寄存器匹配标志位
LPTIMER_FLAG_ETEDEV	外部触发边沿事件标志位
LPTIMER_FLAG_CMPVUP	比较寄存器更新标志位
LPTIMER_FLAG_ARUP	计数器自动重载寄存器更新标志位
LPTIMER_FLAG_UP	LPTIMER计数器由向下计数改为向上计数标志位
LPTIMER_FLAG_DOWN	LPTIMER计数器由向上计数改为向下计数标志位
LPTIMER_FLAG_HLCMVUP	输入高电平计数最大值寄存器更新标志位
LPTIMER_FLAG_INHOCO	LPTIMER_INx (x=0,1) 高电平计数器溢出标志位
LPTIMER_FLAG_INHLOE	LPTIMER_IN0和LPTIMER_IN1高电平重叠错误标志位
LPTIMER_FLAG_INRFOE	LPTIMER_IN0和LPTIMER_IN1下降沿和上升沿重叠错误标志位
LPTIMER_FLAG_IN0E	LPTIMER_IN0错误标志位
LPTIMER_FLAG_IN1E	LPTIMER_IN1错误标志位

输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* clear LPTIMER flag */
```

```
lptimer_flag_clear(LPTIMER, LPTIMER_FLAG_CMPVM);
```

函数 lptimer_interrupt_enable

函数lptimer_interrupt_enable描述见下表：

表 3-447. 函数 lptimer_interrupt_enable

函数名称	lptimer_interrupt_enable
函数原型	void lptimer_interrupt_enable(uint32_t lptimer_periph, 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}	
uint32_t lptimer_periph	LPTIMER外设
LPTIMER	GD32L233xx系列
LPTIMER0 / 1	GD32L235xx系列
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_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_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_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

<i>LPTIMER_INT_INH LOE</i>	LPTIMER_IN0和LPTIMER_IN1高电平重叠错误中断
<i>LPTIMER_INT_INR FOE</i>	LPTIMER_IN0和LPTIMER_IN1下降沿和上升沿重叠错误中断
<i>LPTIMER_INT_IN0 E</i>	LPTIMER_IN0错误中断
<i>LPTIMER_INT_IN1 E</i>	LPTIMER_IN1错误中断
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* enable the LPTIMER interrupt */
```

```
lptimer_interrupt_enable(LPTIMER, LPTIMER_INT_CMPVM);
```

函数 lptimer_interrupt_disable

函数lptimer_interrupt_disable描述见下表:

表 3-448. 函数 lptimer_interrupt_enable

函数名称	lptimer_interrupt_disable
函数原型	void lptimer_interrupt_disable(uint32_t lptimer_periph, 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}	
uint32_t lptimer_periph	LPTIMER外设
LPTIMER	GD32L233xx系列
LPTIMER0 / 1	GD32L235xx系列
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	计数器自动重载寄存器更新中断

<i>RUP</i>	
<i>LPTIMER_INT_UP</i>	LPTIMER计数器由向下计数改为向上计数中断
<i>LPTIMER_INT_DOWN</i>	LPTIMER计数器由向上计数改为向下计数中断
<i>LPTIMER_INT_HLCMVUP</i>	输入高电平计数最大值寄存器更新中断
<i>LPTIMER_INT_INH_LCO</i>	LPTIMER_INx (x=0,1) 高电平计数器溢出中断
<i>LPTIMER_INT_INH_LOE</i>	LPTIMER_IN0和LPTIMER_IN1高电平重叠错误中断
<i>LPTIMER_INT_INR_FOE</i>	LPTIMER_IN0和LPTIMER_IN1下降沿和上升沿重叠错误中断
<i>LPTIMER_INT_IN0_E</i>	LPTIMER_IN0错误中断
<i>LPTIMER_INT_IN1_E</i>	LPTIMER_IN1错误中断
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* disable the LPTIMER interrupt */
```

```
lptimer_interrupt_disable(LPTIMER, LPTIMER_INT_CMPVM);
```

函数 lptimer_interrupt_flag_get

函数lptimer_interrupt_flag_get描述见下表:

表 3-449. 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 lptimer_periph, 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}	
uint32_t lptimer_periph	LPTIMER外设
LPTIMER	GD32L233xx系列
LPTIMER0 / 1	GD32L235xx系列
输入参数{in}	
int_flag	LPTIMER中断标志位
LPTIMER_INT_FLx	比较寄存器匹配中断标志位

<i>G_CMPVM</i>	
<i>LPTIMER_INT_FLAG_CARM</i>	计数器自动重载寄存器匹配中断标志位
<i>LPTIMER_INT_FLAG_ETEDEV</i>	外部触发边沿事件中断标志位
<i>LPTIMER_INT_FLAG_CMPVUP</i>	比较寄存器更新中断标志位
<i>LPTIMER_INT_FLAG_CARUP</i>	计数器自动重载寄存器更新中断标志位
<i>LPTIMER_INT_FLAG_UP</i>	LPTIMER计数器由向下计数改为向上计数中断标志位
<i>LPTIMER_INT_FLAG_DOWN</i>	LPTIMER计数器由向上计数改为向下计数中断标志位
<i>LPTIMER_INT_FLAG_HLCMVUP</i>	输入高电平计数最大值寄存器更新中断标志位
<i>LPTIMER_INT_FLAG_INHLCO</i>	LPTIMER_INx (x=0,1) 高电平计数器溢出中断标志位
<i>LPTIMER_INT_FLAG_INHLOE</i>	LPTIMER_IN0和LPTIMER_IN1高电平重叠错误中断标志位
<i>LPTIMER_INT_FLAG_INRFOE</i>	LPTIMER_IN0和LPTIMER_IN1下降沿和上升沿重叠错误中断标志位
<i>LPTIMER_INT_FLAG_IN0E</i>	LPTIMER_IN0错误中断标志位
<i>LPTIMER_INT_FLAG_IN1E</i>	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, LPTIMER_INT_FLAG_CMPVM);
```

函数 `lptimer_interrupt_flag_clear`

函数 `lptimer_interrupt_flag_clear` 描述见下表:

表 3-450. 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 lptimer_periph, 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}	
uint32_t lptimer_periph	LPTIMER外设
LPTIMER	GD32L233xx系列
LPTIMER0 / 1	GD32L235xx系列
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_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, LPTIMER_INT_FLAG_CMPVM);
```

3.17. LPUART

低功耗通用异步收发器(LPUART)提供了一个灵活方便的串行数据交换接口，章节[3.17.1](#)描述了LPUART的寄存器列表，章节[3.17.2](#)对LPUART库函数进行说明。

3.17.1. 外设寄存器说明

LPUART寄存器列表如下表所示：

表 3-451. LPUART 寄存器

寄存器名称	寄存器描述
LPUART_CTL0	控制寄存器0
LPUART_CTL1	控制寄存器1
LPUART_CTL2	控制寄存器2
LPUART_BAUD	波特率寄存器
LPUART_CMD	请求寄存器
LPUART_STAT	状态寄存器
LPUART_INTC	中断标志清除寄存器
LPUART_RDATA	数据接收寄存器
LPUART_TDATA	数据发送寄存器
LPUART_CHC	兼容性控制寄存器

3.17.2. 外设库函数说明

LPUART库函数列表如下表所示：

表 3-452. LPUART 库函数

库函数名称	库函数描述
lpuart_deinit	复位外设LPUART
lpuart_baudrate_set	配置LPUART波特率
lpuart_parity_config	配置LPUART奇偶校验
lpuart_word_length_set	配置LPUART字长
lpuart_stop_bit_set	配置LPUART停止位
lpuart_enable	使能LPUART
lpuart_disable	失能LPUART
lpuart_transmit_config	LPUART发送配置
lpuart_receive_config	LPUART接收配置
lpuart_data_first_config	配置数据传输时低位在前或高位在前
lpuart_invert_config	配置LPUART反转功能
lpuart_overrun_enable	使能LPUART溢出禁止功能
lpuart_overrun_disable	失能LPUART溢出禁止功能

库函数名称	库函数描述
lpuart_data_transmit	LPUART发送数据功能
lpuart_data_receive	LPUART接收数据功能
lpuart_command_enable	使能LPUART请求
lpuart_address_config	在地址掩码唤醒模式下配置LPUART地址
lpuart_address_detection_mode_config	配置LPUART地址检测模式
lpuart_mute_mode_enable	使能LPUART静默模式
lpuart_mute_mode_disable	失能LPUART静默模式
lpuart_mute_mode_wakeup_config	配置LPUART静默模式唤醒方式
lpuart_halfduplex_enable	使能LPUART半双工模式
lpuart_halfduplex_disable	失能LPUART半双工模式
lpuart_hardware_flow_rts_config	配置LPUART RTS硬件控制流
lpuart_hardware_flow_cts_config	配置LPUART CTS硬件控制流
lpuart_hardware_flow_coherence_config	配置硬件流控兼容模式
lpuart_rs485_driver_enable	使能LPUART RS485驱动
lpuart_rs485_driver_disable	失能LPUART RS485驱动
lpuart_driver_asserttime_config	配置LPUART驱动使能置位时间
lpuart_driver_deasserttime_config	配置LPUART驱动使能置低时间
lpuart_depolarity_config	配置LPUART驱动使能极性模式
lpuart_dma_receive_config	配置LPUART DMA接收
lpuart_dma_transmit_config	配置LPUART DMA发送
lpuart_reception_error_dma_disable	LPUART接收错误时失能DMA
lpuart_reception_error_dma_enable	LPUART接收错误时使能DMA
lpuart_wakeup_enable	使能LPUART唤醒
lpuart_wakeup_disable	失能LPUART唤醒
lpuart_wakeup_mode_config	配置LPUART唤醒模式
lpuart_flag_get	获取STAT/CHC寄存器中的标志
lpuart_flag_clear	清除LPUART状态
lpuart_interrupt_enable	使能LPUART中断
lpuart_interrupt_disable	失能LPUART中断
lpuart_interrupt_flag_get	获取LPUART中断和标志状态
lpuart_interrupt_flag_clear	清除LPUART中断标志位

枚举类型 lpuart_flag_enum

表 3-453. 枚举类型 lpuart_flag_enum

成员名称	功能描述
LPUART_FLAG_REA	接收使能通知标志
LPUART_FLAG_TEA	发送使能通知标志
LPUART_FLAG_WU	从深度睡眠模式唤醒标志
LPUART_FLAG_RWU	接收器从静默模式唤醒

成员名称	功能描述
LPUART_FLAG_AM	地址匹配标志
LPUART_FLAG_BSY	忙标志
LPUART_FLAG_CTS	CTS电平
LPUART_FLAG_CTSF	CTS变化标志
LPUART_FLAG_TBE	发送数据寄存器空
LPUART_FLAG_TC	发送完成
LPUART_FLAG_RBNE	读数据缓冲区非空
LPUART_FLAG_IDLE	空闲线检测标志
LPUART_FLAG_ORERR	溢出错误
LPUART_FLAG_NERR	噪声错误标志
LPUART_FLAG_FERR	帧错误
LPUART_FLAG_PERR	校验错误
LPUART_FLAG_EPERR	校验错误超前检测标志

枚举类型 `lpuart_interrupt_flag_enum`

表 3-454. 枚举类型 `lpuart_interrupt_flag_enum`

成员名称	功能描述
LPUART_INT_FLAG_AM	地址匹配中断标志
LPUART_INT_FLAG_PERR	奇偶校验错误中断标志
LPUART_INT_FLAG_TBE	发送寄存器空中断标志
LPUART_INT_FLAG_TC	发送完成中断标志
LPUART_INT_FLAG_RBNE	读缓冲区非空中断标志
LPUART_INT_FLAG_RBNE_ORERR	读缓冲区非空和溢出中断标志
LPUART_INT_FLAG_IDLE	空闲线检测中断标志
LPUART_INT_FLAG_WU	从深度睡眠模式唤醒中断标志
LPUART_INT_FLAG_CTS	CTS中断标志
LPUART_INT_FLAG_ERR_NERR	噪声错误中断标志
LPUART_INT_FLAG_ERR_ORERR	溢出错误中断标志
LPUART_INT_FLAG_ERR_FERR	帧错误中断标志

枚举类型 `lpuart_interrupt_enum`

表 3-455. 枚举类型 `lpuart_interrupt_enum`

成员名称	功能描述
LPUART_INT_AM	地址匹配中断使能
LPUART_INT_PERR	奇偶校验错误中断使能
LPUART_INT_TBE	发送寄存器空中断使能
LPUART_INT_TC	发送完成中断使能
LPUART_INT_RBNE	读缓冲区非空中断和溢出错误中断使能

成员名称	功能描述
LPUART_INT_IDLE	空闲线检测中断使能
LPUART_INT_WU	从深度睡眠模式唤醒中断使能
LPUART_INT_CTS	CTS中断使能
LPUART_INT_ERR	错误中断使能

枚举类型 `lpuart_invert_enum`

表 3-456. 枚举类型 `lpuart_invert_enum`

成员名称	功能描述
LPUART_DINV_ENABLE	数据位反转
LPUART_DINV_DISABLE	数据位不反转
LPUART_TXPIN_ENABLE	TX管脚电平反转
LPUART_TXPIN_DISABLE	TX管脚电平不反转
LPUART_RXPIN_ENABLE	RX管脚电平反转
LPUART_RXPIN_DISABLE	RX管脚电平不反转
LPUART_SWAP_ENABLE	交换TX/RX管脚
LPUART_SWAP_DISABLE	不交换TX/RX管脚

函数 `lpuart_deinit`

函数 `lpuart_deinit` 描述见下表：

表 3-457. 函数 `lpuart_deinit`

函数名称	<code>lpuart_deinit</code>
函数原型	<code>void lpuart_deinit(uint32_t lpuart_periph);</code>
功能描述	复位外设LPUART
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>lpuart_periph</code>	外设LPUARTx
<code>LPUARTx</code>	(x=0,1) (LPUART1仅适用于GD32L235xx系列)
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* reset LPUART0 */
```

```
lpuart_deinit(LPUART0);
```

函数 `lpuart_baudrate_set`

函数 `lpuart_baudrate_set` 描述见下表：

表 3-458. 函数 lpuart_baudrate_set

函数名称	lpuart_baudrate_set
函数原型	void lpuart_baudrate_set(uint32_t lpuart_periph, uint32_t baudval);
功能描述	配置LPUART波特率
先决条件	-
被调用函数	rcu_clock_freq_get
输入参数{in}	
lpuart_periph	外设LPUARTx
LPUARTx	(x=0,1) (LPUART1仅适用于GD32L235xx系列)
输入参数{in}	
baudval	波特率值
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* configure LPUART0 baud rate value */
lpuart_baudrate_set(LPUART0, 115200);
```

函数 lpuart_parity_config

函数lpuart_parity_config描述见下表:

表 3-459. 函数 lpuart_parity_config

函数名称	lpuart_parity_config
函数原型	void lpuart_parity_config(uint32_t lpuart_periph, uint32_t paritycfg);
功能描述	配置LPUART奇偶校验
先决条件	-
被调用函数	-
输入参数{in}	
lpuart_periph	外设LPUARTx
LPUARTx	(x=0,1) (LPUART1仅适用于GD32L235xx系列)
输入参数{in}	
paritycfg	配置LPUART奇偶校验
LPUART_PM_NON E	无校验
LPUART_PM_ODD	奇校验
LPUART_PM_EVE N	偶校验
输出参数{out}	
-	-
返回值	

-	-
---	---

例如:

```
/* configure LPUART0 parity */
```

```
lpuart_parity_config(LPUART0, LPUART_PM_EVEN);
```

函数 lpuart_word_length_set

函数lpuart_word_length_set描述见下表:

表 3-460. 函数 lpuart_word_length_set

函数名称	lpuart_word_length_set
函数原型	void lpuart_word_length_set(uint32_t lpuart_periph, uint32_t wlen);
功能描述	配置LPUART字长
先决条件	-
被调用函数	-
输入参数{in}	
lpuart_periph	外设LPUARTx
LPUARTx	(x=0,1) (LPUART1仅适用于GD32L235xx系列)
输入参数{in}	
wlen	配置LPUART字长
LPUART_WL_7BIT	7位
LPUART_WL_8BIT	8位
LPUART_WL_9BIT	9位
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* configure LPUART0 word length */
```

```
lpuart_word_length_set(LPUART0, LPUART_WL_9BIT);
```

函数 lpuart_stop_bit_set

函数lpuart_stop_bit_set描述见下表:

表 3-461. 函数 lpuart_stop_bit_set

函数名称	lpuart_stop_bit_set
函数原型	void lpuart_stop_bit_set(uint32_t lpuart_periph, uint32_t stblen);
功能描述	配置LPUART停止位
先决条件	-
被调用函数	-

输入参数{in}	
lpuart_periph	外设LPUARTx
<i>LPUARTx</i>	(x=0,1) (LPUART1仅适用于GD32L235xx系列)
输入参数{in}	
stblen	配置LPUART停止位
<i>LPUART_STB_1BIT</i>	1位
<i>LPUART_STB_2BIT</i>	2位
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* configure LPUART0 stop bit length */
```

```
lpuart_stop_bit_set(LPUART0, LPUART_STB_2BIT);
```

函数 lpuart_enable

函数lpuart_enable描述见下表：

表 3-462. 函数 lpuart_enable

函数名称	lpuart_enable
函数原型	void lpuart_enable(uint32_t lpuart_periph);
功能描述	使能LPUART
先决条件	-
被调用函数	-
输入参数{in}	
lpuart_periph	外设LPUARTx
<i>LPUARTx</i>	(x=0,1) (LPUART1仅适用于GD32L235xx系列)
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* enable LPUART0 */
```

```
lpuart_enable(LPUART0);
```

函数 lpuart_disable

函数lpuart_disable描述见下表：

表 3-463. 函数 lpuart_disable

函数名称	lpuart_disable
函数原型	void lpuart_disable(uint32_t lpuart_periph);
功能描述	失能LPUART
先决条件	-
被调用函数	-
输入参数{in}	
lpuart_periph	外设LPUARTx
LPUARTx	(x=0,1) (LPUART1仅适用于GD32L235xx系列)
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* disable LPUART0 */
```

```
lpuart_disable(LPUART0);
```

函数 lpuart_transmit_config

函数lpuart_transmit_config描述见下表:

表 3-464. 函数 lpuart_transmit_config

函数名称	lpuart_transmit_config
函数原型	void lpuart_transmit_config(uint32_t lpuart_periph, uint32_t txconfig);
功能描述	LPUART发送配置
先决条件	-
被调用函数	-
输入参数{in}	
lpuart_periph	外设LPUARTx
LPUARTx	(x=0,1) (LPUART1仅适用于GD32L235xx系列)
输入参数{in}	
txconfig	使能/失能LPUART发送器
LPUART_TRANSMIT_ENABLE	使能LPUART发送
LPUART_TRANSMIT_DISABLE	失能LPUART发送
输出参数{out}	
-	-
返回值	
-	-

例如:


```
/* configure LPUART0 transmitter */
```

```
lpuart_transmit_config(LPUART0, LPUART_TRANSMIT_ENABLE);
```

函数 lpuart_receive_config

函数lpuart_receive_config描述见下表：

表 3-465. 函数 lpuart_receive_config

函数名称	lpuart_receive_config
函数原型	void lpuart_receive_config(uint32_t lpuart_periph, uint32_t rxconfig);
功能描述	LPUART接收配置
先决条件	-
被调用函数	-
输入参数{in}	
lpuart_periph	外设LPUARTx
LPUARTx	(x=0,1) (LPUART1仅适用于GD32L235xx系列)
输入参数{in}	
rxconfig	使能/失能LPUART接收器
LPUART_RECEIVE_ENABLE	使能LPUART接收
LPUART_RECEIVE_DISABLE	失能LPUART接收
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* configure LPUART0 receiver */
```

```
lpuart_receive_config(LPUART0, LPUART_RECEIVE_ENABLE);
```

函数 lpuart_data_first_config

函数lpuart_data_first_config描述见下表：

表 3-466. 函数 lpuart_data_first_config

函数名称	lpuart_data_first_config
函数原型	void lpuart_data_first_config(uint32_t lpuart_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}	
lpuart_periph	外设LPUARTx

<i>LPUARTx</i>	(x=0,1) (LPUART1仅适用于GD32L235xx系列)
输入参数{in}	
msbf	数据传输时低位在前或高位在前
<i>LPUART_MSBF_LSB</i>	数据传输时低位在前
<i>LPUART_MSBF_MSB</i>	数据传输时高位在前
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* configure LSB of data first */
```

```
lpuart_data_first_config(LPUART0, LPUART_MSBF_LSB);
```

函数 lpuart_invert_config

函数lpuart_invert_config描述见下表:

表 3-467. 函数 lpuart_invert_config

函数名称	lpuart_invert_config
函数原型	void lpuart_invert_config(uint32_t lpuart_periph, lpuart_invert_enum invertpara);
功能描述	配置LPUART反转功能
先决条件	-
被调用函数	-
输入参数{in}	
lpuart_periph	外设LPUARTx
<i>LPUARTx</i>	(x=0,1) (LPUART1仅适用于GD32L235xx系列)
输入参数{in}	
invertpara	参考 表3-456. 枚举类型lpuart_invert_enum
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* configure LPUART0 inversion */
```

```
lpuart_invert_config(LPUART0, LPUART_DINV_ENABLE);
```

函数 lpuart_oversrun_enable

函数lpuart_oversrun_enable描述见下表:

表 3-468. 函数 lpuart_oversrun_enable

函数名称	lpuart_oversrun_enable
函数原型	void lpuart_oversrun_enable(uint32_t lpuart_periph);
功能描述	使能LPUART溢出禁止功能
先决条件	-
被调用函数	-
输入参数{in}	
lpuart_periph	外设LPUARTx
LPUARTx	(x=0,1) (LPUART1仅适用于GD32L235xx系列)
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* enable LPUART0 oversrun */
lpuart_oversrun_enable(LPUART0);
```

函数 lpuart_oversrun_disable

函数lpuart_oversrun_disable描述见下表:

表 3-469. 函数 lpuart_oversrun_disable

函数名称	lpuart_oversrun_disable
函数原型	void lpuart_oversrun_disable(uint32_t lpuart_periph);
功能描述	失能LPUART溢出禁止功能
先决条件	-
被调用函数	-
输入参数{in}	
lpuart_periph	外设LPUARTx
LPUARTx	(x=0,1) (LPUART1仅适用于GD32L235xx系列)
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* disable LPUART0 oversrun */
lpuart_oversrun_disable(LPUART0);
```

函数 lpuart_data_transmit

函数lpuart_data_transmit描述见下表：

表 3-470. 函数 lpuart_data_transmit

函数名称	lpuart_data_transmit
函数原型	void lpuart_data_transmit(uint32_t lpuart_periph, uint32_t data);
功能描述	LPUART发送数据功能
先决条件	-
被调用函数	-
输入参数{in}	
lpuart_periph	外设LPUARTx
LPUARTx	(x=0,1) (LPUART1仅适用于GD32L235xx系列)
输入参数{in}	
data	发送的数据（0-0xFF）
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* LPUART0 transmit data */
lpuart_data_transmit(LPUART0, 0xAA);
```

函数 lpuart_data_receive

函数lpuart_data_receive描述见下表：

表 3-471. 函数 lpuart_data_receive

函数名称	lpuart_data_receive
函数原型	uint16_t lpuart_data_receive(uint32_t lpuart_periph);
功能描述	LPUART接收数据功能
先决条件	-
被调用函数	-
输入参数{in}	
lpuart_periph	外设LPUARTx
LPUARTx	(x=0,1) (LPUART1仅适用于GD32L235xx系列)
输出参数{out}	
-	-
返回值	
uint16_t	接收到的数据（0x0000-0x01FF）

例如：

```
/* LPUART0 receive data */
```

```
uint16_t temp;

temp = lpuart_data_receive(LPUART0);
```

函数 lpuart_command_enable

函数lpuart_command_enable描述见下表：

表 3-472. 函数 lpuart_command_enable

函数名称	lpuart_command_enable
函数原型	void lpuart_command_enable(uint32_t lpuart_periph, uint32_t cmdtype);
功能描述	使能LPUART请求
先决条件	-
被调用函数	-
输入参数{in}	
lpuart_periph	外设LPUARTx
LPUARTx	(x=0,1) (LPUART1仅适用于GD32L235xx系列)
输入参数{in}	
cmdtype	请求类型
LPUART_CMD_MM CMD	静模式请求
LPUART_CMD_RX FCMD	接收数据清空请求
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* enable LPUART0 command */

lpuart_command_enable(LPUART0, LPUART_CMD_MMCMD);
```

函数 lpuart_address_config

函数lpuart_address_config描述见下表：

表 3-473. 函数 lpuart_address_config

函数名称	lpuart_address_config
函数原型	void lpuart_address_config(uint32_t lpuart_periph, uint8_t addr);
功能描述	在地址掩码唤醒模式下配置LPUART地址
先决条件	-
被调用函数	-
输入参数{in}	
lpuart_periph	外设LPUARTx

<i>LPUARTx</i>	(x=0,1) (LPUART1仅适用于GD32L235xx系列)
输入参数{in}	
addr	LPUART地址 (0-0xFF)
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* configure address of the LPUART0 */
```

```
lpuart_address_config(LPUART0, 0x00);
```

函数 lpuart_address_detection_mode_config

函数lpuart_address_detection_mode_config描述见下表:

表 3-474. 函数 lpuart_address_detection_mode_config

函数名称	lpuart_address_detection_mode_config
函数原型	void lpuart_address_detection_mode_config(uint32_t lpuart_periph, uint32_t addmod);
功能描述	配置LPUART地址检测模式
先决条件	-
被调用函数	-
输入参数{in}	
lpuart_periph	外设LPUARTx
<i>LPUARTx</i>	(x=0,1) (LPUART1仅适用于GD32L235xx系列)
输入参数{in}	
addmod	地址检测模式
<i>LPUART_ADDM_4BIT</i>	4位地址检测
<i>LPUART_ADDM_FULLBIT</i>	全位地址检测
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/*configure address detection mode */
```

```
lpuart_address_config(LPUART0, LPUART_ADDM_4BIT);
```

函数 lpuart_mute_mode_enable

函数lpuart_mute_mode_enable描述见下表：

表 3-475. 函数 lpuart_mute_mode_enable

函数名称	lpuart_mute_mode_enable
函数原型	void lpuart_mute_mode_enable(uint32_t lpuart_periph);
功能描述	使能LPUART静默模式
先决条件	-
被调用函数	-
输入参数{in}	
lpuart_periph	外设LPUARTx
LPUARTx	(x=0,1) (LPUART1仅适用于GD32L235xx系列)
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* enable LPUART0 receiver in mute mode */
```

```
lpuart_mute_mode_enable(LPUART0);
```

函数 lpuart_mute_mode_disable

函数lpuart_mute_mode_disable描述见下表：

表 3-476. 函数 lpuart_mute_mode_disable

函数名称	lpuart_mute_mode_disable
函数原型	void lpuart_mute_mode_disable(uint32_t lpuart_periph);
功能描述	失能LPUART静默模式
先决条件	-
被调用函数	-
输入参数{in}	
lpuart_periph	外设LPUARTx
LPUARTx	(x=0,1) (LPUART1仅适用于GD32L235xx系列)
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* disable LPUART0 receiver in mute mode */
```

```
lpuart_mute_mode_disable(LPUART0);
```

函数 `lpuart_mute_mode_wakeup_config`

函数 `lpuart_mute_mode_wakeup_config` 描述见下表:

表 3-477. 函数 `lpuart_mute_mode_wakeup_config`

函数名称	<code>lpuart_mute_mode_wakeup_config</code>
函数原型	<code>void lpuart_mute_mode_wakeup_config(uint32_t lpuart_periph, uint32_t wmethod);</code>
功能描述	配置LPUART静默模式唤醒方式
先决条件	-
被调用函数	-
输入参数{in}	
<code>lpuart_periph</code>	外设LPUARTx
<code>LPUARTx</code>	(x=0,1) (LPUART1仅适用于GD32L235xx系列)
输入参数{in}	
<code>wmethod</code>	两种方法用于进入或退出静默模式
<code>LPUART_WM_IDLE</code>	空闲线唤醒
<code>LPUART_WM_ADD R</code>	地址匹配唤醒
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* configure LPUART0 wakeup method in mute mode */
```

```
lpuart_mute_mode_wakeup_config(LPUART0, LPUART_WM_IDLE);
```

函数 `lpuart_halfduplex_enable`

函数 `lpuart_halfduplex_enable` 描述见下表:

表 3-478. 函数 `lpuart_halfduplex_enable`

函数名称	<code>lpuart_halfduplex_enable</code>
函数原型	<code>void lpuart_halfduplex_enable(uint32_t lpuart_periph);</code>
功能描述	使能LPUART半双工模式
先决条件	-
被调用函数	-
输入参数{in}	
<code>lpuart_periph</code>	外设LPUARTx
<code>LPUARTx</code>	(x=0,1) (LPUART1仅适用于GD32L235xx系列)
输出参数{out}	
-	-
返回值	

-	-
---	---

例如:

```
/* enable LPUART0 half duplex mode */
```

```
lpuart_halfduplex_enable(LPUART0);
```

函数 lpuart_halfduplex_disable

函数lpuart_halfduplex_disable描述见下表:

表 3-479. 函数 lpuart_halfduplex_disable

函数名称	lpuart_halfduplex_disable
函数原型	void lpuart_halfduplex_disable(uint32_t lpuart_periph);
功能描述	失能LPUART半双工模式
先决条件	-
被调用函数	-
输入参数{in}	
lpuart_periph	外设LPUARTx
LPUARTx	(x=0,1) (LPUART1仅适用于GD32L235xx系列)
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* disable LPUART0 half duplex mode */
```

```
lpuart_halfduplex_disable(LPUART);
```

函数 lpuart_hardware_flow_rts_config

函数lpuart_hardware_flow_rts_config描述见下表:

表 3-480. 函数 lpuart_hardware_flow_rts_config

函数名称	lpuart_hardware_flow_rts_config
函数原型	void lpuart_hardware_flow_rts_config(uint32_t lpuart_periph, uint32_t rtsconfig);
功能描述	配置LPUART RTS硬件控制流
先决条件	-
被调用函数	-
输入参数{in}	
lpuart_periph	外设LPUARTx
LPUARTx	(x=0,1) (LPUART1仅适用于GD32L235xx系列)
输入参数{in}	

rtsconfig	使能/失能RTS
<i>LPUART_RTS_ENA BLE</i>	使能RTS
<i>LPUART_RTS_DIS ABLE</i>	失能RTS
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* configure LPUART0 hardware flow control RTS */
```

```
lpuart_hardware_flow_rts_config(LPUART0, LPUART_RTS_ENABLE);
```

函数 lpuart_hardware_flow_cts_config

函数lpuart_hardware_flow_cts_config描述见下表:

表 3-481. 函数 lpuart_hardware_flow_cts_config

函数名称	lpuart_hardware_flow_cts_config
函数原型	void lpuart_hardware_flow_cts_config(uint32_t lpuart_periph, uint32_t ctsconfig);
功能描述	配置LPUART CTS硬件控制流
先决条件	-
被调用函数	-
输入参数{in}	
lpuart_periph	外设LPUARTx
<i>LPUARTx</i>	(x=0,1) (LPUART1仅适用于GD32L235xx系列)
输入参数{in}	
ctsconfig	使能/失能CTS
<i>LPUART_CTS_ENA BLE</i>	使能CTS
<i>LPUART_CTS_DIS ABLE</i>	失能CTS
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* configure LPUART0 hardware flow control CTS */
```

```
lpuart_hardware_flow_cts_config(LPUART0, LPUART_CTS_ENABLE);
```

函数 `lpuart_hardware_flow_coherence_config`

函数 `lpuart_hardware_flow_coherence_config` 描述见下表:

表 3-482. 函数 `lpuart_hardware_flow_coherence_config`

函数名称	<code>lpuart_hardware_flow_coherence_config</code>
函数原型	<code>void lpuart_hardware_flow_coherence_config(uint32_t lpuart_periph, uint32_t hcm);</code>
功能描述	配置硬件流控兼容模式
先决条件	-
被调用函数	-
输入参数{in}	
<code>lpuart_periph</code>	外设LPUARTx
<code>LPUARTx</code>	(x=0,1) (LPUART1仅适用于GD32L235xx系列)
输入参数{in}	
<code>hcm</code>	硬件流控制兼容模式
<code>LPUART_HCM_NONE</code>	nRTS信号与LPUART_STAT寄存器中RBNE位相同
<code>LPUART_HCM_EN</code>	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 */
```

```
lpuart_hardware_flow_coherence_config(LPUART0, LPUART_HCM_NONE);
```

函数 `lpuart_rs485_driver_enable`

函数 `lpuart_rs485_driver_enable` 描述见下表:

表 3-483. 函数 `lpuart_rs485_driver_enable`

函数名称	<code>lpuart_rs485_driver_enable</code>
函数原型	<code>void lpuart_rs485_driver_enable(uint32_t lpuart_periph);</code>
功能描述	使能LPUART RS485驱动
先决条件	-
被调用函数	-
输入参数{in}	
<code>lpuart_periph</code>	外设LPUARTx
<code>LPUARTx</code>	(x=0,1) (LPUART1仅适用于GD32L235xx系列)
输出参数{out}	
-	-
返回值	

-	-
---	---

例如:

```
/* enable LPUART0 RS485 driver */
```

```
lpuart_rs485_driver_enable(LPUART0);
```

函数 lpuart_rs485_driver_disable

函数lpuart_rs485_driver_disable描述见下表:

表 3-484. 函数 lpuart_rs485_driver_disable

函数名称	lpuart_rs485_driver_disable
函数原型	void lpuart_rs485_driver_disable(uint32_t lpuart_periph);
功能描述	失能LPUART RS485驱动
先决条件	-
被调用函数	-
输入参数{in}	
lpuart_periph	外设LPUARTx
LPUARTx	(x=0,1) (LPUART1仅适用于GD32L235xx系列)
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* disable LPUART0 RS485 driver */
```

```
lpuart_rs485_driver_disable(LPUART0);
```

函数 lpuart_driver_assertime_config

函数lpuart_driver_assertime_config描述见下表:

表 3-485. 函数 lpuart_driver_assertime_config

函数名称	lpuart_driver_assertime_config
函数原型	void lpuart_driver_assertime_config(uint32_t lpuart_periph, uint32_t deatime);
功能描述	配置LPUART驱动使能置位时间
先决条件	-
被调用函数	-
输入参数{in}	
lpuart_periph	外设LPUARTx
LPUARTx	(x=0,1) (LPUART1仅适用于GD32L235xx系列)
输入参数{in}	
deatime	驱动使能置位时间 (0-0x0000001F)

输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* set LPUART0 driver assertime */
```

```
lpuart_driver_assertime_config(LPUART0, 0x0000001F);
```

函数 lpuart_driver_deassertime_config

函数lpuart_driver_deassertime_config描述见下表:

表 3-486. 函数 lpuart_driver_deassertime_config

函数名称	lpuart_driver_deassertime_config
函数原型	void lpuart_driver_deassertime_config(uint32_t lpuart_periph, uint32_t dedtime);
功能描述	配置LPUART驱动使能置低时间
先决条件	-
被调用函数	-
输入参数{in}	
lpuart_periph	外设LPUARTx
LPUARTx	(x=0,1) (LPUART1仅适用于GD32L235xx系列)
输入参数{in}	
dedtime	驱动使能置低时间 (0-0x0000001F)
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* set LPUART0 driver deassertime */
```

```
lpuart_driver_deassertime_config(LPUART0, 0x0000001F);
```

函数 lpuart_depolarity_config

函数lpuart_depolarity_config描述见下表:

表 3-487. 函数 lpuart_depolarity_config

函数名称	lpuart_depolarity_config
函数原型	void lpuart_depolarity_config(uint32_t lpuart_periph, uint32_t dep);
功能描述	配置LPUART驱动使能极性模式
先决条件	-

被调用函数	-
输入参数{in}	
lpuart_periph	外设LPUARTx
LPUARTx	(x=0,1) (LPUART1仅适用于GD32L235xx系列)
输入参数{in}	
dep	驱动使能的极性选择模式
LPUART_DEP_HIGH	DE信号高有效
LPUART_DEP_LOW	DE信号低有效
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* configure driver enable polarity mode */
```

```
lpuart_driver_depolarity_config(LPUART0, LPUART_DEP_HIGH);
```

函数 lpuart_dma_receive_config

函数lpuart_dma_receive_config描述见下表:

表 3-488. 函数 lpuart_dma_receive_config

函数名称	lpuart_dma_receive_config
函数原型	void lpuart_dma_receive_config(uint32_t lpuart_periph, uint32_t dmacmd);
功能描述	配置LPUART DMA接收
先决条件	-
被调用函数	-
输入参数{in}	
lpuart_periph	外设LPUARTx
LPUARTx	(x=0,1) (LPUART1仅适用于GD32L235xx系列)
输入参数{in}	
dmacmd	DMA使能/失能DMA接收功能
LPUART_DENR_ENABLE	使能DMA接收功能
LPUART_DENR_DISABLE	失能DMA接收功能
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* LPUART DMA enable for reception */
```

```
lpuart_dma_receive_config(LPUART0, LPUART_DENR_ENABLE);
```

函数 lpuart_dma_transmit_config

函数lpuart_dma_transmit_config描述见下表:

表 3-489. 函数 lpuart_dma_transmit_config

函数名称	lpuart_dma_transmit_config
函数原型	void lpuart_dma_transmit_config(uint32_t lpuart_periph, uint32_t dmacmd);
功能描述	配置LPUART DMA发送
先决条件	-
被调用函数	-
输入参数{in}	
lpuart_periph	外设LPUARTx
LPUARTx	(x=0,1) (LPUART1仅适用于GD32L235xx系列)
输入参数{in}	
dmacmd	使能/失能DMA发送功能
LPUART_DENT_ENABLE	使能DMA发送功能
LPUART_DENT_DISABLE	失能DMA发送功能
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* LPUART DMA enable for transmission */
```

```
lpuart_dma_transmit_config(LPUART0, LPUART_DENT_ENABLE);
```

函数 lpuart_reception_error_dma_disable

函数lpuart_reception_error_dma_disable描述见下表:

表 3-490. 函数 lpuart_reception_error_dma_disable

函数名称	lpuart_reception_error_dma_disable
函数原型	void lpuart_reception_error_dma_disable(uint32_t lpuart_periph);
功能描述	LPUART接收错误时失能DMA
先决条件	-
被调用函数	-
输入参数{in}	

lpuart_periph	外设LPUARTx
<i>LPUARTx</i>	(x=0,1) (LPUART1仅适用于GD32L235xx系列)
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* disable DMA on reception error */
```

```
lpuart_reception_error_dma_disable(LPUART0);
```

函数 lpuart_reception_error_dma_enable

函数lpuart_reception_error_dma_enable描述见下表：

表 3-491. 函数 lpuart_reception_error_dma_enable

函数名称	lpuart_reception_error_dma_enable
函数原型	void lpuart_reception_error_dma_enable(uint32_t lpuart_periph);
功能描述	LPUART接收错误时使能DMA
先决条件	-
被调用函数	-
输入参数{in}	
lpuart_periph	外设LPUARTx
<i>LPUARTx</i>	(x=0,1) (LPUART1仅适用于GD32L235xx系列)
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* enable DMA on reception error */
```

```
lpuart_reception_error_dma_enable(LPUART0);
```

函数 lpuart_wakeup_enable

函数lpuart_wakeup_enable描述见下表：

表 3-492. 函数 lpuart_wakeup_enable

函数名称	lpuart_wakeup_enable
函数原型	void lpuart_wakeup_enable(uint32_t lpuart_periph);
功能描述	使能LPUART唤醒
先决条件	-
被调用函数	-

输入参数{in}	
lpuart_periph	外设LPUARTx
LPUARTx	(x=0,1) (LPUART1仅适用于GD32L235xx系列)
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* LPUART0 wake up enable */
```

```
lpuart_wakeup_enable(LPUART0);
```

函数 lpuart_wakeup_disable

函数lpuart_wakeup_disable描述见下表:

表 3-493. 函数 lpuart_wakeup_disable

函数名称	lpuart_wakeup_disable
函数原型	void lpuart_wakeup_disable(uint32_t lpuart_periph);
功能描述	失能LPUART唤醒
先决条件	-
被调用函数	-
输入参数{in}	
lpuart_periph	外设LPUARTx
LPUARTx	(x=0,1) (LPUART1仅适用于GD32L235xx系列)
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* LPUART0 wake up disable */
```

```
lpuart_wakeup_disable(LPUART0);
```

函数 lpuart_wakeup_mode_config

函数lpuart_wakeup_mode_config描述见下表:

表 3-494. 函数 lpuart_wakeup_mode_config

函数名称	lpuart_wakeup_mode_config
函数原型	void lpuart_wakeup_mode_config(uint32_t lpuart_periph, uint32_t wum);
功能描述	配置LPUART唤醒模式
先决条件	-

被调用函数	-
输入参数{in}	
lpuart_periph	外设LPUARTx
LPUARTx	(x=0,1) (LPUART1仅适用于GD32L235xx系列)
输入参数{in}	
wum	唤醒模式
LPUART_WUM_AD DR	WUF在地址匹配时置位
LPUART_WUM_ST ARTB	WUF在检测到起始位时置位
LPUART_WUM_RB NE	WUF在检测到RBNE时置位
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* configure LPUART0 wake up mode */
```

```
lpuart_wakeup_mode_config(LPUART0, LPUART_WUM_ADDR);
```

函数 lpuart_flag_get

函数lpuart_flag_get描述见下表：

表 3-495. 函数 lpuart_flag_get

函数名称	lpuart_flag_get
函数原型	FlagStatus lpuart_flag_get(uint32_t lpuart_periph, lpuart_flag_enum flag);
功能描述	获取LPUART STAT/CHC寄存器标志位
先决条件	-
被调用函数	-
输入参数{in}	
lpuart_periph	外设LPUARTx
LPUARTx	(x=0,1) (LPUART1仅适用于GD32L235xx系列)
输入参数{in}	
flag	LPUART标志位，参考 表3-453. 枚举类型lpuart_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 LPUART0 state */
```

```
FlagStatus status;
```

```
status = lpuart_flag_get(LPUART0, LPUART_FLAG_TBE);
```

函数 lpuart_flag_clear

函数lpuart_flag_clear描述见下表：

表 3-496. 函数 lpuart_flag_clear

函数名称	lpuart_flag_clear
函数原型	void lpuart_flag_clear(uint32_t lpuart_periph, lpuart_flag_enum flag);
功能描述	清除LPUART状态
先决条件	-
被调用函数	-
输入参数{in}	
lpuart_periph	外设LPUARTx
LPUARTx	(x=0,1) (LPUART1仅适用于GD32L235xx系列)
输入参数{in}	
flag	LPUART标志位，参考 表3-453. 枚举类型lpuart_flag_enum 只能选择一个参数
LPUART_FLAG_PERR	校验错误标志
LPUART_FLAG_FERR	帧错误标志
LPUART_FLAG_NERR	噪声错误标志
LPUART_FLAG_ORERR	溢出错误标志
LPUART_FLAG_IDLE	空闲线检测标志
LPUART_FLAG_TC	发送完成标志
LPUART_FLAG_CTSF	CTS变化标志
LPUART_FLAG_ADDRM	ADDR匹配标志
LPUART_FLAG_WU	从深度睡眠模式唤醒标志
LPUART_FLAG_EPERR	校验错误超前检测标志
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* clear LPUART0 flag */
```

```
lpuart_flag_clear(LPUART0, LPUART_FLAG_TC);
```

函数 lpuart_interrupt_enable

函数lpuart_interrupt_enable描述见下表：

表 3-497. 函数 lpuart_interrupt_enable

函数名称	lpuart_interrupt_enable
函数原型	void lpuart_interrupt_enable(uint32_t lpuart_periph, lpuart_interrupt_enum interrupt);
功能描述	使能LPUART中断
先决条件	-
被调用函数	-
输入参数{in}	
lpuart_periph	外设LPUARTx
LPUARTx	(x=0,1) (LPUART1仅适用于GD32L235xx系列)
输入参数{in}	
interrupt	LPUART中断LPUART标志位，参考 表3-455. 枚举类型lpuart_interrupt_enum 只能选择一个参数
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* enable LPUART0 TBE interrupt */
```

```
lpuart_interrupt_enable(LPUART0, LPUART_INT_TBE);
```

函数 lpuart_interrupt_disable

函数lpuart_interrupt_disable描述见下表：

表 3-498. 函数 lpuart_interrupt_disable

函数名称	lpuart_interrupt_disable
函数原型	void lpuart_interrupt_disable(uint32_t lpuart_periph, lpuart_interrupt_enum interrupt);
功能描述	失能LPUART中断
先决条件	-
被调用函数	-
输入参数{in}	
lpuart_periph	外设LPUARTx

<i>LPUARTx</i>	(x=0,1) (LPUART1仅适用于GD32L235xx系列)
输入参数{in}	
interrupt	LPUART中断LPUART标志位, 参考 表3-455. 枚举类型lpuart_interrupt_enum 只能选择一个参数
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* disable LPUART0 TBE interrupt */
```

```
lpuart_interrupt_disable(LPUART0, LPUART_INT_TBE);
```

函数 lpuart_interrupt_flag_get

函数lpuart_interrupt_flag_get描述见下表:

表 3-499. 函数 lpuart_interrupt_flag_get

函数名称	lpuart_interrupt_flag_get
函数原型	FlagStatus lpuart_interrupt_flag_get(uint32_t lpuart_periph, lpuart_interrupt_flag_enum int_flag);
功能描述	获取LPUART中断和标志状态
先决条件	-
被调用函数	-
输入参数{in}	
lpuart_periph	外设LPUARTx
<i>LPUARTx</i>	(x=0,1) (LPUART1仅适用于GD32L235xx系列)
输入参数{in}	
int_flag	LPUART中断标志, 参考 表3-454. 枚举类型lpuart_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 LPUART0 interrupt flag status */
```

```
FlagStatus status;
```

```
status = lpuart_interrupt_flag_get(LPUART0, LPUART_INT_FLAG_RBNE);
```

函数 lpuart_interrupt_flag_clear

函数lpuart_interrupt_flag_clear描述见下表:

表 3-500. 函数 lpuart_interrupt_flag_clear

函数名称	lpuart_interrupt_flag_clear
函数原型	void lpuart_interrupt_flag_clear(uint32_t lpuart_periph, lpuart_interrupt_flag_enum int_flag);
功能描述	清除LPUART中断标志位
先决条件	-
被调用函数	-
输入参数{in}	
lpuart_periph	外设LPUARTx
LPUARTx	(x=0,1) (LPUART1仅适用于GD32L235xx系列)
输入参数{in}	
int_flag	LPUART中断标志，参考 表3-454. 枚举类型lpuart_interrupt_flag_enum 只能选择一个参数
LPUART_INT_FLAG_PERR	校验错误中断标志
LPUART_INT_FLAG_ERR_FERR	帧错误中断标志
LPUART_INT_FLAG_ERR_NERR	噪声错误中断标志
LPUART_INT_FLAG_RBNE_ORERR	读数据缓冲区非空中断和溢出错误中断标志
LPUART_INT_FLAG_ERR_ORERR	过载错误中断标志
LPUART_INT_FLAG_IDLE	IDLE线检测中断标志
LPUART_INT_FLAG_TC	发送完成中断标志
LPUART_INT_FLAG_CTS	CTS变化中断标志
LPUART_INT_FLAG_AM	ADDR匹配中断标志
LPUART_INT_FLAG_WU	从深度睡眠模式唤醒中断标志
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* clear the LPUART0 interrupt flag */
```

```
lpuart_interrupt_flag_clear(LPUART0, LPUART_INT_FLAG_TC);
```

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

3.18.1. 外设寄存器说明

表 3-501. 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⁾	中断不安全状态寄存器
IPR ⁽¹⁾	中断优先级寄存器
CPUID ⁽²⁾	CPUID寄存器
ICSR ⁽²⁾	中断控制及状态寄存器
VTOR ⁽²⁾	向量表偏移量寄存器
AIRCR ⁽²⁾	应用程序中断及复位控制寄存器
SCR ⁽²⁾	系统控制寄存器
CCR ⁽²⁾	配置与控制寄存器
SHPR ⁽²⁾	系统异常优先级寄存器
SHCSR ⁽²⁾	系统异常控制及状态寄存器

1. 参考 core_cm23.h 文件中定义的结构体类型 NVIC_Type
2. 参考 core_cm23.h 文件中定义的结构体类型 SCB_Type

表 3-502. 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_cm23.h 文件中定义的结构体类型 SysTick_Type

3.18.2. 外设库函数说明

MISC库函数列表如下表所示：

表 3-503. MISC 库函数

库函数名称	库函数描述
nvic_irq_enable	使能NVIC的中断
nvic_irq_disable	禁能NVIC的中断

库函数名称	库函数描述
nvic_system_reset	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	设置系统定时器时钟源

枚举类型 IRQn_Type

表 3-504. GD32L233xx 的枚举类型 IRQn_Type

成员名称	功能描述
WWDGT_IRQn	窗口看门狗中断
LVD_IRQn	连接到 EXTI 线的 LVD 中断
TAMPER_STAMP_IRQn	连接到 EXTI 线的 RTC 侵入和时间戳中断
RTC_WKUP_IRQn	RTC 唤醒中断
FMC_IRQn	FMC 全局中断
RCU_CTC_IRQn	RCU 和 CTC 全局中断
EXTI0_IRQn	EXTI 线 0 中断
EXTI1_IRQn	EXTI 线 1 中断
EXTI2_IRQn	EXTI 线 2 中断
EXTI3_IRQn	EXTI 线 3 中断
EXTI4_IRQn	EXTI 线 4 中断
DMA_Channel0_IRQn	DMA0 通道 0 全局中断
DMA_Channel1_IRQn	DMA0 通道 1 全局中断
DMA_Channel2_IRQn	DMA0 通道 2 全局中断
DMA_Channel3_IRQn	DMA0 通道 3 全局中断
DMA_Channel4_IRQn	DMA0 通道 4 全局中断
DMA_Channel5_IRQn	DMA0 通道 5 全局中断
DMA_Channel6_IRQn	DMA0 通道 6 全局中断
ADC_IRQn	ADC 全局中断
USB_HP_IRQn	USB 高优先级中断
USB_LP_IRQn	USB 低优先级中断
TIMER1_IRQn	TIMER1 全局中断
TIMER2_IRQn	TIMER2 全局中断
TIMER8_IRQn	TIMER8 全局中断
TIMER11_IRQn	TIMER11 全局中断
TIMER5_IRQn	TIMER5 全局中断
TIMER6_IRQn	TIMER6 全局中断
USART0_IRQn	USART0 全局中断
USART1_IRQn	USART1 全局中断
UART3_IRQn	UART3 全局中断
UART4_IRQn	UART4 全局中断
I2C0_EV_IRQn	I2C0 事件中断

I2C0_ER_IRQn	I2C0 错误中断
I2C1_EV_IRQn	I2C1 事件中断
I2C1_ER_IRQn	I2C1 错误中断
SPI0_IRQn	SPI0 全局中断
SPI1_IRQn	SPI1 全局中断
DAC_IRQn	DAC 全局中断
I2C2_EV_IRQn	I2C2 事件中断
I2C2_ER_IRQn	I2C2 错误中断
RTC_Alarm_IRQn	RTC 闹钟中断
USBD_WKUP_IRQn	USBD 唤醒中断
EXTI5_9_IRQn	EXTI 线[9:5]中断
EXTI10_15_IRQn	EXTI 线[15:10]中断
DMAMUX_IRQn	DMAMUX 全局中断
CMP0_IRQn	CMP0 全局中断
CMP1_IRQn	CMP1 全局中断
I2C0_WKUP_IRQn	I2C0 唤醒中断
I2C2_WKUP_IRQn	I2C2 唤醒中断
USART0_WKUP_IRQn	USART0 唤醒中断
LPUART_IRQn	LPUART 全局中断
CAU_IRQn	CAU 全局中断
TRNG_IRQn	TRNG 全局中断
SLCD_IRQn	SLCD 全局中断
USART1_WKUP_IRQn	USART1 唤醒中断
I2C1_WKUP_IRQn	I2C1 唤醒中断
LPUART_WKUP_IRQn	LPUART 唤醒中断
LPTIMER_IRQn	LPTIMER 全局中断

表 3-505. GD32L235xx 的枚举类型 IRQn_Type

成员名称	功能描述
WWDGT_IRQn	窗口看门狗中断
LVD_IRQn	连接到 EXTI 线的 LVD 中断
TAMPER_STAMP_IRQn	连接到 EXTI 线的 RTC 侵入和时间戳中断
RTC_WKUP_IRQn	RTC 唤醒中断
FMC_IRQn	FMC 全局中断
RCU_CTC_IRQn	RCU 和 CTC 全局中断
EXTI0_IRQn	EXTI 线 0 中断
EXTI1_IRQn	EXTI 线 1 中断
EXTI2_IRQn	EXTI 线 2 中断
EXTI3_IRQn	EXTI 线 3 中断
EXTI4_IRQn	EXTI 线 4 中断
DMA_Channel0_IRQn	DMA0 通道 0 全局中断
DMA_Channel1_IRQn	DMA0 通道 1 全局中断
DMA_Channel2_IRQn	DMA0 通道 2 全局中断

DMA_Channel3_IRQn	DMA0 通道 3 全局中断
DMA_Channel4_IRQn	DMA0 通道 4 全局中断
DMA_Channel5_IRQn	DMA0 通道 5 全局中断
DMA_Channel6_IRQn	DMA0 通道 6 全局中断
ADC_IRQn	ADC 全局中断
USBD_HP_CAN_TX_IRQn	USBD 高优先级或 CAN TX 中断
USBD_LP_CAN_RX0_IRQn	USBD 低优先级或 CAN RX0 中断
TIMER1_IRQn	TIMER1 全局中断
TIMER2_IRQn	TIMER2 全局中断
TIMER8_IRQn	TIMER8 全局中断
TIMER11_IRQn	TIMER11 全局中断
TIMER5_IRQn	TIMER5 全局中断
TIMER6_IRQn	TIMER6 全局中断
USART0_IRQn	USART0 全局中断
USART1_IRQn	USART1 全局中断
UART3_IRQn	UART3 全局中断
UART4_IRQn	UART4 全局中断
I2C0_EV_IRQn	I2C0 事件中断
I2C0_ER_IRQn	I2C0 错误中断
I2C1_EV_IRQn	I2C1 事件中断
I2C1_ER_IRQn	I2C1 错误中断
SPI0_IRQn	SPI0 全局中断
SPI1_IRQn	SPI1 全局中断
DAC_IRQn	DAC 全局中断
I2C2_EV_IRQn	I2C2 事件中断
I2C2_ER_IRQn	I2C2 错误中断
RTC_Alarm_IRQn	RTC 闹钟中断
USBD_WKUP_IRQn	USBD 唤醒中断
EXTI5_9_IRQn	EXTI 线[9:5]中断
TIMER0_TRG_CMT_UP_BRK_IRQn	TIMER0 触发与通道换相中断或 TIMER0 更新中断或 TIMER0 中止中断
TIMER0_Channel_IRQn	TIMER0 捕获比较中断
TIMER14_IRQn	TIMER14 全局中断
EXTI10_15_IRQn	EXTI 线[15:10]中断
TIMER40_IRQn	TIMER40 全局中断
CAN_RX1_IRQn	CAN RX1 中断
CAN_EWMC_IRQn	CAN EWMC 中断
DMAMUX_IRQn	DMAMUX 全局中断
CMP0_IRQn	CMP0 全局中断
CMP1_IRQn	CMP1 全局中断
I2C0_WKUP_IRQn	I2C0 唤醒中断
I2C2_WKUP_IRQn	I2C2 唤醒中断

USART0_WKUP_IRQn	USART0 唤醒中断
LPUART0_IRQn	LPUART0 全局中断
CAU_IRQn	CAU 全局中断
TRNG_IRQn	TRNG 全局中断
SLCD_IRQn	SLCD 全局中断
USART1_WKUP_IRQn	USART1 唤醒中断
I2C1_WKUP_IRQn	I2C1 唤醒中断
LPUART0_WKUP_IRQn	LPUART0 唤醒中断
LPTIMER0_IRQn	LPTIMER0 全局中断
LPUART1_WKUP_IRQn	LPUART1 唤醒中断
LPTIMER1_IRQn	LPTIMER1 全局中断
LPUART1_IRQn	LPUART1 全局中断

函数 nvic_irq_enable

函数nvic_irq_enable描述见下表:

表 3-506. 函数 nvic_irq_enable

函数名称	nvic_irq_enable
函数原形	void nvic_irq_enable(uint8_t nvic_irq, uint8_t nvic_irq_priority);
功能描述	使能NVIC中断
先决条件	-
被调用函数	NVIC_SetPriority、NVIC_EnableIRQ
输入参数{in}	
nvic_irq	NVIC中断，GD32L233xx产品请参考枚举类型 表3-504. GD32L233xx的枚举类型IRQn_Type ，GD32L235xx产品请参考枚举类型 表3-505. GD32L235xx的枚举类型IRQn_Type
输入参数{in}	
nvic_irq_priority	优先级（0~3）
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* enable window watchDog timer interrupt, priority is 1 */
```

```
nvic_irq_enable(WWDGT_IRQn, 1);
```

函数 nvic_irq_disable

函数nvic_irq_disable描述见下表:

表 3-507. 函数 nvic_irq_disable

函数名称	nvic_irq_disable
------	------------------

函数原形	void nvic_irq_disable(uint8_t nvic_irq);
功能描述	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，GD32L233xx产品请参考枚举类型 表3-504. GD32L233xx的枚举类型IRQn_Type ，GD32L235xx产品请参考枚举类型 表3-505. GD32L235xx的枚举类型IRQn_Type
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* disable window watchDog timer interrupt */
```

```
nvic_irq_disable(WWDGT_IRQn);
```

表 3-508. 函数 nvic_system_reset

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

例如：

```
/* reset the MCU */
```

```
nvic_system_reset();
```

函数 nvic_vector_table_set

函数nvic_vector_table_set描述见下表：

表 3-509. 函数 nvic_vector_table_set

函数名称	nvic_vector_table_set
函数原形	void nvic_vector_table_set(uint32_t nvic_vect_tab, uint32_t offset);
功能描述	设置向量表基地址
先决条件	-

被调用函数	-
输入参数{in}	
nvic_vect_tab	RAM或者FLASH基地址
<i>NVIC_VECTTAB_RAM</i>	RAM基地址
<i>NVIC_VECTTAB_FLASH</i>	FLASH基地址
输入参数{in}	
offset	向量表偏移量（向量表地址=基地址+偏移量）
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* set vector table address = NVIC_VECTTAB_FLASH + 0x200 */
nvic_vector_table_set(NVIC_VECTTAB_FLASH, 0x200);
```

函数 **system_lowpower_set**

函数system_lowpower_set描述见下表：

表 3-510. 函数 **system_lowpower_set**

函数名称	system_lowpower_set
函数原形	void system_lowpower_set(uint8_t lowpower_mode);
功能描述	设置系统低功耗模式状态
先决条件	-
被调用函数	-
输入参数{in}	
lowpower_mode	系统低功耗模式的状态
<i>SCB_LPM_SLEEP_EXIT_ISR</i>	该位为1时，退出ISR时一直处于低功耗模式
<i>SCB_LPM_DEEPSLEEP</i>	该位为1时，系统处于deep sleep模式
<i>SCB_LPM_WAKEUP_BY_ALL_INT</i>	该位为1时，低功耗模式可以被所有中断唤醒（无论中断是否被使能）
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* the system always enter low power mode by exiting from ISR */
```

```
system_lowpower_set(SCB_LPM_SLEEP_EXIT_ISR);
```

函数 `system_lowpower_reset`

函数 `system_lowpower_reset` 描述见下表：

表 3-511. 函数 `system_lowpower_reset`

函数名称	<code>system_lowpower_reset</code>
函数原形	<code>void system_lowpower_reset(uint8_t lowpower_mode);</code>
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}	
<code>lowpower_mode</code>	系统低功耗模式的状态
<code>SCB_LPM_SLEEP_EXIT_ISR</code>	该位为0时，系统将通过退出ISR退出低功耗模式
<code>SCB_LPM_DEEPSLEEP</code>	该位为0时，系统进入sleep模式
<code>SCB_LPM_WAKEUP_BY_ALL_INT</code>	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 3-512. 函数 `systick_clksource_set`

函数名称	<code>systick_clksource_set</code>
函数原形	<code>void systick_clksource_set(uint32_t systick_clksource);</code>
功能描述	设置SysTick时钟源
先决条件	-
被调用函数	-
输入参数{in}	
<code>systick_clksource</code>	SysTick时钟源
<code>SYSTICK_CLKSOURCE_HCLK</code>	SysTick时钟源为AHB时钟
<code>SYSTICK_CLKSOURCE_D2CLK</code>	SysTick时钟源为AHB时钟的8分频

<code>RCE_HCLK_DIV8</code>	
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* systick clock source is HCLK/8 */
```

```
systick_clksource_set(SYSTICK_CLKSOURCE_HCLK_DIV8);
```

3.19. PMU

电源管理单元提供了十种省电模式，包括运行模式，运行模式 1，运行模式 2，睡眠模式，睡眠模式 1，睡眠模式 2，深度睡眠模式，深度睡眠模式 1，深度睡眠模式 2 和待机模式。章节 [3.19.1](#) 描述了 PMU 的寄存器列表，章节 [3.19.2](#) 对 PMU 库函数进行说明。

3.19.1. 外设寄存器说明

PMU 寄存器列表如下表所示：

表 3-513. 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控制和状态寄存器
PMU_CTL1	PMU控制寄存器1
PMU_STAT	PMU状态寄存器
PMU_PAR	PMU参数寄存器

3.19.2. 外设库函数说明

PMU 库函数列表如下表所示：

表 3-514. PMU 库函数

库函数名称	库函数描述
<code>pmu_deinit</code>	复位外设PMU
<code>pmu_lvd_select</code>	选择低压检测阈值
<code>pmu_lvd_disable</code>	关闭低压检测器
<code>pmu_ldo_output_select</code>	LDO输出电压选择
<code>pmu_vc_enable</code>	使能VBAT电池充电
<code>pmu_vc_disable</code>	禁能VBAT电池充电
<code>pmu_vcr_select</code>	选择PMU VBAT电池充电电阻
<code>pmu_low_power_enable</code>	在Run/Sleep模式下使能低驱
<code>pmu_low_power_disable</code>	在Run/Sleep模式下禁能低驱

库函数名称	库函数描述
pmu_to_sleepmode	进入睡眠模式
pmu_to_deepsleepmode	进入深度睡眠/深度睡眠1/深度睡眠2模式
pmu_to_standbymode	进入待机模式
pmu_wakeup_pin_enable	使能PMU WKUP引脚唤醒
pmu_wakeup_pin_disable	禁能PMU WKUP引脚唤醒
pmu_backup_write_enable	备份域写使能
pmu_backup_write_disable	备份域写禁能
pmu_sram_power_config	配置SRAM1电源状态
pmu_core1_power_config	配置COREOFF1域电源状态（仅适用于GD32L233xx系列）
pmu_eflash_sleep_power_config	配置在运行 / 运行1 / 运行2 / 运行3模式下eflash域电源状态（仅适用于GD32L235xx系列）
pmu_eflash_deepsleep_power_config	配置在深度睡眠 / 深度睡眠1 / 深度睡眠2模式下eflash域电源状态（仅适用于GD32L235xx系列）
pmu_deepsleep2_retention_enable	深度睡眠2模式下，CPU有保留寄存器
pmu_deepsleep2_retention_disable	深度睡眠2模式下，CPU没有保留寄存器
pmu_deepsleep2_sram_power_config	配置进入深度睡眠2后SRAM1电源状态
pmu_deepsleep_wait_time_config	配置在进入深度睡眠之前IRC16M计数
pmu_wakeuptime_core1_software_enable	采用软件设定值唤醒COREOFF1域（仅适用于GD32L233xx系列）
pmu_wakeuptime_core1_software_disable	采用硬件应答信号唤醒COREOFF1域（仅适用于GD32L233xx系列）
pmu_wakeuptime_eflash_config	配置eflash唤醒计数（仅适用于GD32L235xx系列）
pmu_wakeuptime_sram_config	配置SRAM1唤醒时间
pmu_wakeuptime_sram_software_enable	采用软件设定值唤醒SRAM1（仅适用于GD32L233xx系列）
pmu_wakeuptime_sram_software_disable	采用硬件应答信号唤醒SRAM1（仅适用于GD32L233xx系列）
pmu_wakeuptime_deepsleep2_software_enable	采用软件设定值唤醒深度睡眠2
pmu_wakeuptime_deepsleep2_software_disable	采用硬件应答信号唤醒深度睡眠2
pmu_flag_get	获取PMU标志位状态
pmu_flag_clear	清除PMU标志位状态

函数 pmu_deinit

函数 pmu_deinit 描述见下表：

表 3-515. 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 3-516. 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	电压阈值为2.1V
PMU_LVDT_1	电压阈值为2.3V
PMU_LVDT_2	电压阈值为2.4V
PMU_LVDT_3	电压阈值为2.6V
PMU_LVDT_4	电压阈值为2.7V
PMU_LVDT_5	电压阈值为2.9V
PMU_LVDT_6	电压阈值为3.0V
PMU_LVDT_7	PB7上的输入模拟电压（与0.8V相比）
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_disable

函数 pmu_lvd_disable 描述见下表：

表 3-517. 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_ldo_output_select

函数 pmu_ldo_output_select 描述见下表：

表 3-518. 函数 pmu_ldo_output_select

函数名称	pmu_ldo_output_select
函数原型	void pmu_ldo_output_select(uint32_t ldo_output);
功能描述	内部电压调节器（LDO）输出电压选择
先决条件	-
被调用函数	-
输入参数{in}	
ldo_output	输出电压模式
PMU_LDOVS_LOW	输出低电压模式，仅适用于GD32L233xx系列
PMU_LDOVS_HIGH	输出高电压模式
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* select output low voltage mode */
```

```
pmu_ldo_output_select(PMU_LDOVS_LOW);
```

函数 pmu_vc_enable

函数pmu_vc_enable描述见下表:

表 3-519. 函数 pmu_vc_enable

函数名称	pmu_vc_enable
函数原型	void pmu_vc_enable(void);
功能描述	使能VBAT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_vc_enable();
```

函数 pmu_vc_disable

函数pmu_vc_disable描述见下表:

表 3-520. 函数 pmu_vc_disable

函数名称	pmu_vc_disable
函数原型	void pmu_vc_disable(void);
功能描述	禁能VBAT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_vc_disable();
```

函数 pmu_vcr_select

函数pmu_vcr_select描述见下表:

表 3-521. 函数 pmu_vcr_select

函数名称	pmu_vcr_select
函数原型	void pmu_vcr_select(uint32_t resistor);
功能描述	选择PMU VBAT电池充电电阻
先决条件	-
被调用函数	-
输入参数{in}	
resistor	VBAT电池充电电阻
PMU_VCRSEL_5K	选择5k欧姆电阻作为VBAT电池充电电阻
PMU_VCRSEL_1P5K	选择1.5k欧姆电阻作为VBAT电池充电电阻
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* select PMU VBAT battery charging resistor */
```

```
pmu_vcr_select(PMU_VCRSEL_5K);
```

函数 pmu_low_power_enable

函数pmu_low_power_enable描述见下表:

表 3-522. 函数 pmu_low_power_enable

函数名称	pmu_low_power_enable
函数原型	void pmu_low_power_enable(void);
功能描述	在Run/Sleep模式下使能低驱
先决条件	-
被调用函数	-
输入参数{in}	
-	-
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* enable low power in Run/Sleep mode */
```

```
pmu_low_power_enable();
```

函数 **pmu_low_power_disable**

函数 **pmu_low_power_disable** 描述见下表：

表 3-523. 函数 **pmu_low_power_disable**

函数名称	pmu_low_power_disable
函数原型	void pmu_low_power_disable(void);
功能描述	在Run/Sleep模式下禁能低驱
先决条件	-
被调用函数	-
输入参数{in}	
-	-
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* disable low power in Run/Sleep mode */
```

```
pmu_low_power_disable();
```

函数 **pmu_to_sleepmode**

函数 **pmu_to_sleepmode** 描述见下表：

表 3-524. 函数 **pmu_to_sleepmode**

函数名称	pmu_to_sleepmode
函数原型	void pmu_to_sleepmode(uint32_t lowdrive, 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}	
lowdrive	NPLDO驱动模式
PMU_LDNP_NORM ALDRIVE	禁能低驱模式
PMU_LDNP_LOWD RIVE	使能低驱模式
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(PMU_LDNP_LOWDRIVE, WFI_CMD);
```

函数 pmu_to_deepsleepmode

函数 pmu_to_deepsleepmode 描述见下表:

表 3-525. 函数 pmu_to_deepsleepmode

函数名称	pmu_to_deepsleepmode
函数原型	void pmu_to_deepsleepmode(uint32_t lowdrive, uint8_t deepsleepmodecmd, uint8_t deepsleepmode);
功能描述	进入深度睡眠/深度睡眠1/深度睡眠2模式
先决条件	-
被调用函数	-
输入参数{in}	
lowdrive	NPLDO工作模式
PMU_LDNPDSP_NOR MALDRIVE	禁能低驱模式
PMU_LDNPDSP_LOW DRIVE	使能低驱模式
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
输入参数{in}	
deepsleepmode	深度睡眠模式
PMU_DEEPSLEEP	深度睡眠模式
PMU_DEEPSLEEP1	深度睡眠模式1
PMU_DEEPSLEEP2	深度睡眠模式2
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* PMU work at Deep-sleep mode */
```

```
pmu_to_deepsleepmode(PMU_LDNPDSP_NORMALDRIVE, WFI_CMD, PMU_DEEPSLEEP);
```

函数 pmu_to_standbymode

函数 pmu_to_standbymode 描述见下表:

表 3-526. 函数 pmu_to_standbymode

函数名称	pmu_to_standbymode
函数原型	void pmu_to_standbymode();
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(WFI_CMD);
```

函数 pmu_wakeup_pin_enable

函数 pmu_wakeup_pin_enable 描述见下表:

表 3-527. 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	使能PMU WKUP引脚唤醒
先决条件	-
被调用函数	-
输入参数{in}	
wakeup_pin	唤醒引脚
PMU_WAKEUP_PIN0	WKUP Pin 0 (PA0)
PMU_WAKEUP_PIN1	WKUP Pin 1 (PC13)
PMU_WAKEUP_PIN2	WKUP Pin 2 (PA2)
PMU_WAKEUP_PIN3	WKUP Pin 3 (PB2)
PMU_WAKEUP_PIN4	WKUP Pin 4 (PC6)
输出参数{out}	

-	-
返回值	
-	-

例如：

```
/* enable wakeup pin0 */
```

```
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 3-528. 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	禁能PMU WKUP引脚唤醒
先决条件	-
被调用函数	-
输入参数{in}	
wakeup_pin	唤醒引脚
PMU_WAKEUP_PIN0	WKUP Pin 0 (PA0)
PMU_WAKEUP_PIN1	WKUP Pin 1 (PC13)
PMU_WAKEUP_PIN2	WKUP Pin 2 (PA2)
PMU_WAKEUP_PIN3	WKUP Pin 3 (PB2)
PMU_WAKEUP_PIN4	WKUP Pin 4 (PC6)
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* disable wakeup pin0 */
```

```
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 3-529. 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 3-530. 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_sram_power_config

函数 pmu_sram_power_config 描述见下表：

表 3-531. 函数 pmu_sram_power_config

函数名称	pmu_sram_power_config
函数原型	void pmu_sram_power_config(uint32_t state);
功能描述	配置SRAM1电源状态
先决条件	-
被调用函数	-
输入参数{in}	
state	SRAM1电源状态
PMU_SRAM1_SLEEP	SRAM1断电
PMU_SRAM1_WAKE	SRAM1唤醒
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* configure power state of SRAM1 */
pmu_sram_power_config(PMU_SRAM1_WAKE);
```

函数 pmu_core1_power_config（仅适用于 GD32L233xx 系列）

函数pmu_core1_power_config描述见下表：

表 3-532. 函数 pmu_core1_power_config

函数名称	pmu_core1_power_config
函数原型	void pmu_core1_power_config(uint32_t state);
功能描述	配置COREOFF1域电源状态
先决条件	-
被调用函数	-
输入参数{in}	
state	COREOFF1域电源状态
PMU_CORE1_SLEEP	COREOFF1域断电
PMU_CORE1_WAKE	COREOFF1域唤醒
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* configure power state of COREOFF1 domain */
pmu_core1_power_config(PMU_SRAM1_WAKE);
```

函数 pmu_deepsleep2_retention_enable（仅适用于 GD32L233xx 系列）

函数pmu_deepsleep2_retention_enable描述见下表：

表 3-533. 函数 pmu_deepsleep2_retention_enable

函数名称	pmu_deepsleep2_retention_enable
函数原型	void pmu_deepsleep2_retention_enable(void);
功能描述	深度睡眠2模式下，CPU有保留寄存器
先决条件	-
被调用函数	-
输入参数{in}	
-	-
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* have retention register in Deep-sleep 2 */
```

```
pmu_deepsleep2_retention_enable();
```

函数 pmu_deepsleep2_retention_disable（仅适用于 GD32L233xx 系列）

函数pmu_deepsleep2_retention_disable描述见下表：

表 3-534. 函数 pmu_deepsleep2_retention_disable

函数名称	pmu_deepsleep2_retention_disable
函数原型	void pmu_deepsleep2_retention_disable(void);
功能描述	深度睡眠2模式下，CPU没有保留寄存器
先决条件	-
被调用函数	-
输入参数{in}	
-	-
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* no retention register in Deep-sleep 2 */
```

```
pmu_deepsleep2_retention_disable();
```

函数 **pmu_eflash_sleep_power_config**（仅适用于 GD32L235xx 系列）

函数 **pmu_eflash_sleep_power_config** 描述见下表：

表 3-535. 函数 **pmu_eflash_sleep_power_config**

函数名称	pmu_eflash_sleep_power_config
函数原型	void pmu_eflash_sleep_power_config(uint32_t state);
功能描述	配置在运行模式下eflash域电源状态
先决条件	-
被调用函数	-
输入参数{in}	
state	eflash域电源状态
PMU_EFLASH_SLEEP_ON	在运行模式下eflash域电源打开
PMU_EFLASH_SLEEP_OFF	在运行模式下eflash域电源关闭
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* configure power state of eflash domain */
```

```
pmu_eflash_sleep_power_config(PMU_EFLASH_SLEEP_ON);
```

函数 **pmu_eflash_deepsleep_power_config**（仅适用于 GD32L235xx 系列）

函数 **pmu_eflash_deepsleep_power_config** 描述见下表：

表 3-536. 函数 **pmu_eflash_deepsleep_power_config**

函数名称	pmu_eflash_deepsleep_power_config
函数原型	void pmu_eflash_deepsleep_power_config(uint32_t state);
功能描述	配置在深度睡眠 / 深度睡眠1 / 深度睡眠2模式下eflash域电源状态
先决条件	-
被调用函数	-
输入参数{in}	
state	eflash域电源状态
PMU_EFLASH_DSPSL_EEP_ON	在深度睡眠 / 深度睡眠1 / 深度睡眠2模式下eflash域电源打开
PMU_EFLASH_DSPSL_EEP_OFF	在深度睡眠 / 深度睡眠1 / 深度睡眠2模式下eflash域电源关闭
输出参数{out}	
-	-
返回值	

-	-
---	---

例如:

```
/* configure power state of eflash domain when in Deep-sleep / Deep-sleep 1 / Deep-sleep 2 mode */
```

```
pmu_eflash_deepsleep_power_config(PMU_EFLASH_DSPSLEEP_ON);
```

函数 pmu_deepsleep2_sram_power_config

函数pmu_deepsleep2_sram_power_config描述见下表:

表 3-537. 函数 pmu_deepsleep2_sram_power_config

函数名称	pmu_deepsleep2_sram_power_config
函数原型	void pmu_deepsleep2_sram_power_config(uint32_t state);
功能描述	配置进入深度睡眠2后SRAM1电源状态
先决条件	-
被调用函数	-
输入参数{in}	
state	SRAM1电源状态
PMU_SRAM1_POWER_OFF	SRAM1断电
PMU_SRAM1_POWER_REMAIN	SRAM1电源状态与GD32L233xx系列Run/Run1/Run2或GD32L235xx系列Run相同
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* configure SRAM1 power state when enter Deep-sleep 2 */
```

```
pmu_deepsleep2_sram_power_config(PMU_SRAM1_POWER_OFF);
```

函数 pmu_deepsleep_wait_time_config

函数pmu_deepsleep_wait_time_config描述见下表:

表 3-538. 函数 pmu_deepsleep_wait_time_config

函数名称	pmu_deepsleep_wait_time_config
函数原型	void pmu_deepsleep_wait_time_config(uint32_t wait_time);
功能描述	配置在进入深度睡眠之前IRC16M计数
先决条件	-
被调用函数	-
输入参数{in}	

wait_time	进入深度睡眠之前IRC16M计数
<i>uint32_t</i>	0x0~0x1F
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* configure IRC16M counter before enter Deep-sleep mode */
```

```
pmu_deepsleep_wait_time_config(1);
```

函数 pmu_wakeuptime_core1_software_enable（仅适用于 GD32L233xx 系列）

函数pmu_wakeuptime_core1_software_enable描述见下表：

表 3-539. 函数 pmu_wakeuptime_core1_software_enable

函数名称	pmu_wakeuptime_core1_software_enable
函数原型	void pmu_wakeuptime_core1_software_enable(uint32_t wakeup_time);
功能描述	采用软件设定值唤醒COREOFF1域
先决条件	-
被调用函数	-
输入参数{in}	
wakeup_time	COREOFF1域电源开关唤醒时间
<i>uint32_t</i>	0x0~0xFF
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* use software value signal when wake up COREOFF1 domain */
```

```
pmu_wakeuptime_core1_software_enable(1);
```

函数 pmu_wakeuptime_core1_software_disable（仅适用于 GD32L233xx 系列）

函数pmu_wakeuptime_core1_software_disable描述见下表：

表 3-540. 函数 pmu_wakeuptime_core1_software_disable

函数名称	pmu_wakeuptime_core1_software_disable
函数原型	void pmu_wakeuptime_core1_software_disable(void);
功能描述	采用硬件应答信号唤醒COREOFF1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：

```
/* use hardware ack signal when wake up COREOFF1 domain */
```

```
pmu_wakeuptime_core1_software_disable();
```

函数 pmu_wakeuptime_eflash_config（仅适用于 GD32L235xx 系列）

函数pmu_wakeuptime_eflash_config描述见下表：

表 3-541. 函数 pmu_wakeuptime_eflash_config

函数名称	pmu_wakeuptime_eflash_config
函数原型	void pmu_wakeuptime_eflash_config(uint32_t wait_time);
功能描述	配置eflash唤醒计数
先决条件	-
被调用函数	-
输入参数{in}	
wait_time	IRC16M counter, 0x0~0xFF
<i>uint32_t</i>	0x0~0xFF
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* configure eflash wake up counter */
```

```
pmu_wakeuptime_eflash_config(2);
```

函数 pmu_wakeuptime_sram_config

函数pmu_wakeuptime_sram_config描述见下表：

表 3-542. 函数 Function pmu_wakeuptime_sram_config

函数名称	pmu_wakeuptime_sram_config
函数原型	void pmu_wakeuptime_sram_config(uint32_t wakeup_time);
功能描述	配置SRAM1唤醒时间
先决条件	-
被调用函数	-

输入参数{in}	
wakeup_time	wakeup time of power switch of SRAM1
<i>uint32_t</i>	0x0~0xFF
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* configure SRAM1 wakeup time */
pmu_wakeuptime_sram_config(2);
```

函数 pmu_wakeuptime_sram_software_enable（仅适用于 GD32L233xx 系列）

函数pmu_wakeuptime_sram_software_enable描述见下表：

表 3-543. 函数 pmu_wakeuptime_sram_software_enable

函数名称	pmu_wakeuptime_sram_software_enable
函数原型	void pmu_wakeuptime_sram_software_enable(void);
功能描述	采用软件设定值唤醒SRAM1
先决条件	-
被调用函数	-
输入参数{in}	
-	-
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* use software value signal when wake up SRAM1 */
pmu_wakeuptime_sram_software_enable();
```

函数 pmu_wakeuptime_sram_software_disable（仅适用于 GD32L233xx 系列）

函数pmu_wakeuptime_sram_software_disable描述见下表：

表 3-544. 函数 pmu_wakeuptime_sram_software_disable

函数名称	pmu_wakeuptime_sram_software_disable
函数原型	void pmu_wakeuptime_sram_software_disable(void);
功能描述	采用硬件应答信号唤醒SRAM1
先决条件	-
被调用函数	-

输入参数{in}	
-	-
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* use hardware ack signal when wake up SRAM1 */
```

```
pmu_wakeuptime_sram_software_disable();
```

函数 pmu_wakeuptime_deepsleep2_software_enable

函数pmu_wakeuptime_deepsleep2_software_enable描述见下表：

表 3-545. 函数 pmu_wakeuptime_deepsleep2_software_enable

函数名称	pmu_wakeuptime_deepsleep2_software_enable
函数原型	void pmu_wakeuptime_deepsleep2_software_enable(uint32_t wakeup_time);
功能描述	采用软件设定值唤醒深度睡眠2
先决条件	-
被调用函数	-
输入参数{in}	
wakeup_time	COREOFF0域电源开关唤醒时间
uint32_t	0x0~0xFF
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* use software value signal when wake up Deep-sleep 2 */
```

```
pmu_wakeuptime_deepsleep2_software_enable(1);
```

函数 pmu_wakeuptime_deepsleep2_software_disable

函数pmu_wakeuptime_deepsleep2_software_disable描述见下表：

表 3-546. 函数 pmu_wakeuptime_deepsleep2_software_disable

函数名称	pmu_wakeuptime_deepsleep2_software_disable
函数原型	void pmu_wakeuptime_deepsleep2_software_disable(void);
功能描述	采用硬件应答信号唤醒深度睡眠2
先决条件	-

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

例如：

```
/* use hardware ack signal when wake up Deep-sleep 2 */
```

```
pmu_wakeuptime_deepsleep2_software_disable();
```

函数 pmu_flag_get

函数pmu_flag_get描述见下表：

表 3-547. 函数 pmu_flag_get

函数名称	pmu_flag_get
函数原型	FlagStatus pmu_flag_get(uint32_t flag);
功能描述	获取PMU标志位状态
先决条件	-
被调用函数	-
输入参数{in}	
flag	标志位
PMU_FLAG_WAKEUP	唤醒标志
PMU_FLAG_STANDBY	待机标志
PMU_FLAG_LVD	Lvd标志
PMU_FLAG_LDOVSRF	LDO电压选择准备就绪标志
PMU_FLAG_NPRDY	正常功耗LDO准备就绪标志
PMU_FLAG_LPRDY	低功耗LDO准备就绪标志
PMU_FLAG_SRAM1_SLEEP	SRAM1处于睡眠状态标志
PMU_FLAG_SRAM1_ACTIVE	SRAM1处于激活状态标志
PMU_FLAG_EFLASH_SLEEP	eflash域处于睡眠状态标志（仅适用于GD32L235xx系列）
PMU_FLAG_EFLASH_ACTIVE	eflash域处于激活状态标志（仅适用于GD32L235xx系列）
PMU_FLAG_CORE1_SLEEP	COREOFF1域处于睡眠状态标志（仅适用于GD32L233xx系列）
PMU_FLAG_CORE1_ACTIVE	COREOFF1域处于激活状态标志（仅适用于GD32L233xx系列）
PMU_FLAG_DEEPSLEEP	深度睡眠2模式状态标志

EP_2	
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 3-548. 函数 pmu_flag_clear

函数名称	pmu_flag_clear
函数原型	void pmu_flag_clear(uint32_t flag);
功能描述	清除PMU标志位状态
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_DEEPSLEEP_2	深度睡眠2模式状态标志
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* clear flag bit */
```

```
pmu_flag_clear(PMU_FLAG_STANDBY);
```

3.20. 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 [3.20.1](#) 描述了 RCU 的寄存器列表，章节 [0](#) 对 RCU 库函数进行说明。

3.20.1. 外设寄存器说明

RCU寄存器列表如下表所示：

表 3-549. RCU 寄存器

寄存器名称	寄存器描述
RCU_CTL	控制寄存器0
RCU_CFG0	配置寄存器0
RCU_INT	中断寄存器
RCU_APB2RST	APB2复位寄存器
RCU_APB1RST	APB1复位寄存器
RCU_AHBEN	AHB使能寄存器
RCU_APB2EN	APB2使能寄存器
RCU_APB1EN	APB1使能寄存器
RCU_BDCTL	备份域控制寄存器
RCU_RSTSCK	复位源/时钟寄存器
RCU_AHBRST	AHB复位寄存器
RCU_CFG1	配置寄存器1
RCU_CFG2	配置寄存器2
RCU_ANB2EN	AHB2使能寄存器
RCU_AHB2RST	AHB2复位寄存器
RCU_VKEY	电源解锁寄存器
RCU_LPB	低功耗模式寄存器

3.20.2. 外设库函数说明

RCU库函数列表如下表所示：

表 3-550. RCU 库函数

库函数名称	库函数描述
rcu_deinit	复位RCU
rcu_periph_clock_enable	使能外设时钟
rcu_periph_clock_disable	禁能外设时钟
rcu_periph_clock_sleep_enable	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	外设时钟复位除能
rcu_bkp_reset_enable	备份域时钟复位使能
rcu_bkp_reset_disable	备份域时钟复位除能
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

库函数名称	库函数描述
rcu_apb2_clock_config	配置APB2时钟预分频选择
rcu_adc_clock_config	配置ADC时钟预分频选择
rcu_ckout_config	配置CKOUT时钟源选择及分频系数
rcu_pll_config	配置主PLL时钟
rcu_usart_clock_config	配置串口时钟
rcu_i2c_clock_config	配置I2Cx(x=0,1,2)时钟
rcu_lptimer_clock_config	配置LPTIMER时钟
rcu_lpuart_clock_config	配置LPUART时钟
rcu_irc16mdiv_clock_config	配置IRC16M时钟
rcu_usbd_clock_config	配置USBDM时钟
rcu_rtc_clock_config	配置RTC/SLCD时钟
rcu_pll_source_ck_prediv_config	配置PLL输入源分频因子
rcu_lxtal_drive_capability_config	配置LXTAL的驱动力
rcu_lp_bandgap_config	配置低功耗模式带隙
rcu_osci_stab_wait	等待振荡器稳定标志位置位或振荡器起振超时
rcu_osci_on	打开振荡器
rcu_osci_off	关闭振荡器
rcu_osci_bypass_mode_enable	使能时钟旁路模式
rcu_osci_bypass_mode_disable	除能时钟旁路模式
rcu_irc16m_adjust_value_set	设置内部16MHz RC振荡器时钟调整值
rcu_hxtal_clock_monitor_enable	使能HXTAL时钟监视器
rcu_hxtal_clock_monitor_disable	禁能HXTAL时钟监视器
rcu_lxtal_clock_monitor_enable	使能LXTAL时钟监视器
rcu_lxtal_clock_monitor_disable	禁能LXTAL时钟监视器
rcu_voltage_key_unlock	解锁电压锁定
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	时钟稳定中断除能
rcu_interrupt_flag_get	获取时钟中断和CKM中断标志
rcu_interrupt_flag_clear	清除中断标志

枚举类型 rcu_periph_enum

表 3-551. 枚举类型 rcu_periph_enum

成员名称	功能描述
RCU_DMA	DMA时钟
RCU_CAU	CAU时钟
RCU_TRNG	TRNG时钟
RCU_CRC	CRC时钟
RCU_GPIOA	GPIOA时钟

成员名称	功能描述
RCU_GPIOB	GPIOB时钟
RCU_GPIOC	GPIOC时钟
RCU_GPIOD	GPIOD时钟
RCU_GPIOF	GPIOF时钟
RCU_SYSCFG	SYSCFG时钟
RCU_CMP	CMP时钟
RCU_ADC	ADC时钟
RCU_TIMER8	TIMER8时钟
RCU_SPI0	SPI0时钟
RCU_USART0	USART0时钟
RCU_DBGMCU	DBGMCU时钟
RCU_TIMER1	TIMER1时钟
RCU_TIMER2	TIMER2时钟
RCU_TIMER5	TIMER5时钟
RCU_TIMER6	TIMER6时钟
RCU_TIMER11	TIMER11时钟
RCU_LPTIMER	LPTIMER时钟(仅适用于GD32L233xx系列)
RCU_LPTIMER0	LPTIMER0时钟(仅适用于GD32L235xx系列)
RCU_LPTIMER1	LPTIMER1时钟(仅适用于GD32L235xx系列)
RCU_SLCD	SLCD时钟
RCU_WWDGT	WWDGT时钟
RCU_SPI1	SPI1时钟
RCU_USART1	USART1时钟
RCU_LPUART	LPUART时钟 (仅适用于GD32L233xx系列)
RCU_LPUART0	LPUART0时钟 (仅适用于GD32L235xx系列)
RCU_LPUART1	LPUART1时钟 (仅适用于GD32L235xx系列)
RCU_CAN	CAN时钟 (仅适用于GD32L235xx系列)
RCU_UART3	UART3时钟
RCU_UART4	UART4时钟
RCU_I2C0	I2C0时钟
RCU_I2C1	I2C1时钟
RCU_USBD	USBD时钟
RCU_I2C2	I2C2时钟
RCU_PMU	PMU时钟
RCU_DAC	DAC时钟
RCU_CTC	CTC时钟
RCU_BKP	BKP时钟
RCU_RTC	RTC时钟

枚举类型 rcu_periph_sleep_enum

表 3-552. 枚举类型 rcu_periph_sleep_enum

成员名称	功能描述
RCU_DMA	DMA时钟
RCU_CAU	CAU时钟

枚举类型 rcu_periph_reset_enum

表 3-553. 枚举类型 rcu_periph_reset_enum

成员名称	功能描述
RCU_CAURST	复位SRAM0时钟
RCU_TRNGRST	复位TRNG时钟
RCU_CRCRST	复位CRC时钟
RCU_GPIOARST	复位GPIOA时钟
RCU_GPIOBRST	复位GPIOB时钟
RCU_GPIOCRST	复位GPIOC时钟
RCU_GPIODRST	复位GPIOD时钟
RCU_GPIOFRST	复位GPIOF时钟
RCU_SYSCFGRST	复位SYSCFG时钟
RCU_CMPRST	复位CMP时钟
RCU_ADCRST	复位ADC时钟
RCU_TIMER8RST	复位TIMER8时钟
RCU_SPI0RST	复位SPI0时钟
RCU_USART0RST	复位USART0时钟
RCU_TIMER1RST	复位TIMER1时钟
RCU_TIMER2RST	复位TIMER2时钟
RCU_TIMER5RST	复位TIMER5时钟
RCU_TIMER6RST	复位TIMER6时钟
RCU_TIMER11RST	复位TIMER11时钟
RCU_LPTIMERRST	复位LPTIMER时钟 (仅适用于GD32L233xx系列)
RCU_LPTIMER0RST	复位LPTIMER0时钟 (仅适用于GD32L235xx系列)
RCU_LPTIMER1RST	复位LPTIMER1时钟 (仅适用于GD32L235xx系列)
RCU_SLCDRST	复位SLCD时钟
RCU_WWDGTRST	复位WWDGT时钟
RCU_SPI1RST	复位SPI1时钟
RCU_USART1RST	复位USART1时钟
RCU_LPUARTRST	复位LPUART时钟 (仅适用于GD32L233xx系列)
RCU_LPUART0RST	复位LPUART0时钟 (仅适用于GD32L235xx系列)
RCU_LPUART1RST	复位LPUART1时钟 (仅适用于GD32L235xx系列)
RCU_CANRST	复位CAN时钟 (仅适用于GD32L235xx系列)
RCU_UART3RST	复位UART3时钟
RCU_UART4RST	复位UART4时钟

成员名称	功能描述
RCU_I2C0RST	复位I2C0时钟
RCU_I2C1RST	复位I2C1时钟
RCU_USBDNST	复位USBDS时钟
RCU_I2C2RST	复位I2C2时钟
RCU_PMRST	复位PMU时钟
RCU_DACRST	复位DAC时钟
RCU_CTCRST	复位CTC时钟

枚举类型 rcu_flag_enum

表 3-554. 枚举类型 rcu_flag_enum

成员名称	功能描述
RCU_FLAG_IRC32KSTB	IRC32K稳定标志
RCU_FLAG_LXTALSTB	LXTAL稳定标志
RCU_FLAG_IRC16MSTB	IRC16M稳定标志
RCU_FLAG_HXTALSTB	HXTAL稳定标志
RCU_FLAG_IRC48MSTB	IRC48M稳定标志
RCU_FLAG_PLLSTB	PLL稳定标志
RCU_FLAG_V12RST	1.2V电压域复位标志
RCU_FLAG_EPRST	外部引脚复位标志
RCU_FLAG_PORRST	电源复位标志
RCU_FLAG_SWRST	软件复位标志
RCU_FLAG_FWDGTRST	独立看门狗复位标志
RCU_FLAG_WWDGTRST	窗口看门狗复位标志
RCU_FLAG_LPRST	低功耗复位标志

枚举类型 rcu_int_flag_enum

表 3-555. 枚举类型 rcu_int_flag_enum

成员名称	功能描述
RCU_INT_FLAG_IRC32KSTB	IRC32K时钟稳定中断标志
RCU_INT_FLAG_LXTALSTB	LXTAL时钟稳定中断标志
RCU_INT_FLAG_IRC16MSTB	IRC16M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_IRC48MSTB	IRC48M时钟稳定中断标志
RCU_INT_FLAG_LXTALCKM	LXTAL时钟稳定中断标志
RCU_INT_FLAG_CKM	外部高速晶振时钟阻塞中断标志
RCU_INT_FLAG_IRC32KSTB	IRC32K时钟稳定中断标志

枚举类型 rcu_int_flag_clear_enum

表 3-556. 枚举类型 rcu_int_flag_clear_enum

成员名称	功能描述
RCU_INT_FLAG_IRC32KSTB_CLR R	IRC32K时钟稳定中断清除标志
RCU_INT_FLAG_LXTALSTB_CLR R	LXTAL时钟稳定中断清除标志
RCU_INT_FLAG_IRC16MSTB_CLR LR	IRC16M时钟稳定中断清除标志
RCU_INT_FLAG_HXTALSTB_CLR R	HXTAL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_IRC48MSTB_CLR LR	IRC48M时钟稳定中断清除标志
RCU_INT_FLAG_LXTALCKM_CLR R	LXTAL时钟稳定中断清除标志
RCU_INT_FLAG_CKM_CLR	外部高速晶振时钟阻塞中断清除标志
RCU_INT_FLAG_IRC32KSTB_CLR R	IRC32K时钟稳定中断清除标志

枚举类型 rcu_int_enum

表 3-557. 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	LXTAL时钟稳定中断
RCU_INT_IRC16MSTB	IRC16M时钟稳定中断
RCU_INT_HXTALSTB	HXTAL时钟稳定中断
RCU_INT_PLLSTB	PLL时钟稳定中断
RCU_INT_IRC48MSTB	IRC48M时钟稳定中断
RCU_INT_LXTALCKM	LXTAL时钟稳定中断
RCU_INT_IRC32KSTB	IRC32K时钟稳定中断
RCU_INT_LXTALSTB	LXTAL时钟稳定中断

枚举类型 rcu_osci_type_enum

表 3-558. 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_IRC16M	IRC16M振荡器
RCU_IRC48M	IRC48M振荡器

成员名称	功能描述
RCU_IRC32K	IRC32K振荡器
RCU_PLL_CK	锁相环时钟

枚举类型 `rcu_clock_freq_enum`

表 3-559. 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_ADC	ADC时钟
CK_USART0	USART0时钟
CK_I2C0	I2C0时钟
CK_I2C1	I2C1时钟
CK_I2C2	I2C2时钟
CK_LPUART	LPUART时钟
CK_USART1	USART1时钟
CK_LPTIMER	LPTIMER时钟

枚举类型 `usart_idx_enum`

表 3-560. 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

枚举类型 `lptimer_idx_enum`

表 3-561. 枚举类型 `lptimer_idx_enum`

成员名称	功能描述
IDX_LPTIMER0	LPTIMER0索引
IDX_LPTIMER1	LPTIMER1索引

枚举类型 `lpuart_idx_enum`

表 3-562. 枚举类型 `lpuart_idx_enum`

成员名称	功能描述
IDX_LPUART0	LPUART0索引
IDX_LPUART1	LPUART1索引

枚举类型 `i2c_idx_enum`表 3-563. 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

函数 `rcu_deinit`

函数`rcu_deinit`描述见下表:

表 3-564. 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 3-565. 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3-551. 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 3-566. 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3-551. 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 3-567. 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3-552. 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 3-568. 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3-552. 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 3-569. 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3-569. 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 3-570. 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3-569. 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 3-571. 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 3-572. 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 3-573. 函数 rcu_system_clock_source_config

函数名称	rcu_system_clock_source_config
函数原形	void rcu_system_clock_source_config(uint32_t ck_sys);
功能描述	配置选择系统时钟源
先决条件	-
被调用函数	-
输入参数{in}	
ck_sys	系统时钟源选择
<i>RCU_CKSYSSRC_I RC16M</i>	选择CK_IRC16M时钟作为CK_SYS时钟源
<i>RCU_CKSYSSRC_ HXTAL</i>	选择CK_HXTAL时钟作为CK_SYS时钟源
<i>RCU_CKSYSSRC_ PLL</i>	选择CK_PLL时钟作为CK_SYS时钟源
<i>RCU_CKSYSSRC_I RC48M</i>	选择CK_IRC48M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 3-574. 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_IRC16M/RCU_SCSS_HXTAL/RCU_SCSS_PLL/RCU_SCSS_IRC48M

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 3-575. 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

<i>RCU_AHB_CKSYS</i> <i>_DIVx</i>	选择CK_SYS时钟x分频 (x=1, 2, 4, 8, 16, 64, 128, 256, 512)
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* configure CK_SYS/128 */
```

```
rcu_ahb_clock_config(RCU_AHB_CKSYS_DIV128);
```

函数 **rcu_apb1_clock_config**

函数rcu_apb1_clock_config描述见下表:

表 3-576. 函数 rcu_apb1_clock_config

函数名称	rcu_apb1_clock_config
函数原形	void rcu_apb1_clock_config(uint32_t ck_apb1);
功能描述	配置APB1时钟预分频选择
先决条件	-
被调用函数	-
输入参数{in}	
ck_apb1	APB1预分频选择
<i>RCU_APB1_CKAH</i> <i>B_DIVx</i>	选择CK_AHB时钟x分频作为CK_APB1时钟 (x=1,2,4,8,16)
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* configure CK_AHB/16 as CK_APB1 */
```

```
rcu_apb1_clock_config(RCU_APB1_CKAHB_DIV16);
```

函数 **rcu_apb2_clock_config**

函数rcu_apb2_clock_config描述见下表:

表 3-577. 函数 rcu_apb2_clock_config

函数名称	rcu_apb2_clock_config
函数原形	void rcu_apb2_clock_config(uint32_t ck_apb2);
功能描述	配置APB2时钟预分频选择
先决条件	-

被调用函数	-
输入参数{in}	
ck_apb2	APB2预分频选择
RCU_APB2_CK_AHB_DIVx	选择CK_AHB时钟x分频作为CK_APB2时钟 (x=1,2,4,8,16)
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* configure CK_AHB/8 as CK_APB2 */
```

```
rcu_apb2_clock_config(RCU_APB2_CK_AHB_DIV8);
```

函数 rcu_adc_clock_config

函数rcu_adc_clock_config描述见下表:

表 3-578. 函数 rcu_adc_clock_config

函数名称	rcu_adc_clock_config
函数原形	void rcu_adc_clock_config(rcu_adc_clock_enum ck_adc);
功能描述	配置adc时钟预分频选择
先决条件	-
被调用函数	-
输入参数{in}	
ck_adc	ADC预分频选择
RCU_ADCCCK_IRC16M	选择IRC16M作为CK_ADC时钟
RCU_ADCCCK_APB2_DIV2	选择APB2时钟的2分频作为CK_ADC时钟
RCU_ADCCCK_APB2_DIV4	选择APB2时钟的4分频作为CK_ADC时钟
RCU_ADCCCK_APB2_DIV6	选择APB2时钟的6分频作为CK_ADC时钟
RCU_ADCCCK_APB2_DIV8	选择APB2时钟的8分频作为CK_ADC时钟
RCU_ADCCCK_APB2_DIV10	选择APB2时钟的10分频作为CK_ADC时钟
RCU_ADCCCK_APB2_DIV12	选择APB2时钟的12分频作为CK_ADC时钟
RCU_ADCCCK_APB2_DIV14	选择APB2时钟的14分频作为CK_ADC时钟
RCU_ADCCCK_APB2_DIV16	选择APB2时钟的16分频作为CK_ADC时钟

2_DIV16	
RCU_ADCCCK_AHB _DIV3	选择AHB时钟的分频3作为CK_ADC时钟
RCU_ADCCCK_AHB _DIV5	选择AHB时钟的分频5作为CK_ADC时钟
RCU_ADCCCK_AHB _DIV7	选择AHB时钟的分频7作为CK_ADC时钟
RCU_ADCCCK_AHB _DIV9	选择AHB时钟的分频9作为CK_ADC时钟
RCU_ADCCCK_AHB _DIV11	选择AHB时钟的分频11作为CK_ADC时钟
RCU_ADCCCK_AHB _DIV13	选择AHB时钟的分频13作为CK_ADC时钟
RCU_ADCCCK_AHB _DIV15	选择AHB时钟的分频15作为CK_ADC时钟
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* configure the ADC prescaler factor */
rcu_adc_clock_config(RCU_ADCCCK_APB2_DIV2);
```

函数 rcu_ckout_config

函数rcu_ckout_config描述见下表：

表 3-579. 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_ NONE	无时钟输出
RCU_CKOUTSRC_I RC48M	选择内部48M RC振荡器时钟
RCU_CKOUTSRC_I RC32K	选择内部32K RC振荡器时钟
RCU_CKOUTSRC_	选择外部低速晶体振荡器时钟（LXTAL）

LXTAL	
RCU_CKOUTSRC_CKSYS	选择系统时钟CK_SYS
RCU_CKOUTSRC_IRC16M	选择内部16M RC振荡器时钟
RCU_CKOUTSRC_HXTAL	选择外部高速晶体振荡器时钟（HXTAL）
RCU_CKOUTSRC_CKPLL_DIV1	选择CK_PLL时钟
RCU_CKOUTSRC_CKPLL_DIV2	选择（CK_PLL / 2）时钟
输入参数{in}	
ckout_div	CKOUT分频系数
RCU_CKOUT_DIVx	将CKOUT所选时钟x分频（x=1,2,4,8,16,32,64,128）
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_pll_config

函数rcu_pll_config描述见下表：

表 3-580. 函数 rcu_pll_config

函数名称	rcu_pll_config
函数原形	void rcu_pll_config(uint32_t pll_src, uint32_t pll_mul);
功能描述	配置主PLL时钟
先决条件	-
被调用函数	-
输入参数{in}	
pll_src	PLL时钟源选择
RCU_PLLSRC_IRC16M	IRC16M被选择为PLL时钟的时钟源
RCU_PLLSRC_HXTAL	HXTAL时钟被选择为PLL时钟的时钟源
RCU_PLLSRC_IRC48M	IRC48M被选择为PLL时钟的时钟源
输入参数{in}	
pll_mul	PLL时钟倍频因子

<i>RCU_PLL_MULx</i>	PLL源时钟 * x (x =4..127)
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* configure the PLL */
```

```
rcu_pll_config(RCU_PLLSRC_HXTAL, RCU_PLL_MUL10);
```

函数 `rcu_usart_clock_config`

函数`rcu_usart_clock_config`描述见下表:

表 3-581. 函数 `rcu_usart_clock_config`

函数名称	<code>rcu_usart_clock_config</code>
函数原形	<code>void rcu_usart_clock_config(usart_idx_enum usart_idx, uint32_t ck_usart);</code>
功能描述	配置串口时钟
先决条件	-
被调用函数	-
输入参数{in}	
usart_idx	USART索引, 参考 表3-560. 枚举类型usart_idx_enum
<i>IDX_USARTx</i>	USARTx的索引值(x=0,1)
ck_usart	I2Cx时钟源
<i>RCU_USARTSRC_</i> <i>CKAPB</i>	选择CK_APB1/CK_APB2时钟作为CK_USART时钟
<i>RCU_USARTSRC_</i> <i>CKSYS</i>	选择CK_SYS时钟作为CK_USART时钟
<i>RCU_USARTSRC_</i> <i>LXTAL</i>	选择CK_LXTAL时钟作为CK_USART时钟
<i>RCU_USARTSRC_</i> <i>RC16M</i>	选择CK_IRC16M时钟作为CK_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_USART0SRC_LXTAL);
```

函数 `rcu_i2c_clock_config`

函数`rcu_i2c_clock_config`描述见下表:

表 3-582. 函数 `rcu_i2c_clock_config`

函数名称	<code>rcu_i2c_clock_config</code>
函数原形	<code>void rcu_i2c_clock_config(i2c_idx_enum i2c_idx, uint32_t ck_i2c);</code>
功能描述	配置I2C时钟
先决条件	-
被调用函数	-
输入参数{in}	
<code>i2c_idx</code>	I2C索引, 参考 表3-563. 枚举类型<i>i2c_idx_enum</i>
<code>IDX_I2Cx</code>	I2Cx的索引值(x=0,1,2)
<code>ck_i2c</code>	I2Cx输入时钟源
<code>RCU_I2CSRC_CKAPB1</code>	选择CK_APB1时钟作为CK_I2C时钟
<code>RCU_I2CSRC_CKSYS</code>	选择CK_SYS时钟作为CK_I2C时钟
<code>RCU_I2CSRC_CKIRC16MDIV</code>	选择CK_IRC16MDIV时钟作为CK_I2C时钟
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* configure the CKAPB1 as I2C0 clock */
```

```
rcu_usart_clock_config(IDX_I2C0, RCU_I2CSRC_CKAPB1);
```

函数 `rcu_lptimer_clock_config` (仅适用于 GD32L233xx 系列)

函数`rcu_lptimer_clock_config`描述见下表:

表 3-583. 函数 `rcu_lptimer_clock_config`

函数名称	<code>rcu_lptimer_clock_config</code>
函数原形	<code>void rcu_lptimer_clock_config(uint32_t ck_lptimer);</code>
功能描述	配置LPTIMER时钟
先决条件	-
被调用函数	-
输入参数{in}	
<code>ck_lptimer</code>	LPTIMER时钟源选择
<code>RCU_LPTIMERSRC_CKAPB2</code>	选择CK_APB2作为LPTIMER时钟源
<code>RCU_LPTIMERSRC_CKIRC32K</code>	选择CK_IRC32K作为LPTIMER时钟源

<code>_IRC32K</code>	
<code>RCU_LPTIMERSRC_LXTAL</code>	选择CK_LXTAL作为LPTIMER时钟源
<code>RCU_LPTIMERSRC_IRC16MDIV</code>	选择CK_IRC16MDIV作为LPTIMER时钟源
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* configure the LPTIMER clock source selection */
```

```
rcu_lptimer_clock_config(RCU_LPTIMERSRC_CKAPB2);
```

函数 `rcu_lptimer_clock_config` (仅适用于 GD32L235xx 系列)

函数`rcu_lptimer_clock_config`描述见下表:

表 3-584. 函数 `rcu_lptimer_clock_config`

函数名称	<code>rcu_lptimer_clock_config</code>
函数原形	<code>void rcu_lptimer_clock_config(lptimer_idx_enum lptimer_idx, uint32_t ck_lptimer);</code>
功能描述	配置LPTIMER时钟
先决条件	-
被调用函数	-
输入参数{in}	
<code>lptimer_idx</code>	LPTIMER索引, 参考 表3-561. 枚举类型 <code>lptimer_idx_enum</code>
<code>IDX_LPTIMERx</code>	LPTIMERx索引(x=0,1)
<code>ck_lptimer</code>	LPTIMER时钟源选择
<code>RCU_LPTIMERSRC_CKAPB1</code>	选择CK_APB1作为LPTIMER时钟源
<code>RCU_LPTIMERSRC_IRC32K</code>	选择CK_IRC32K作为LPTIMER时钟源
<code>RCU_LPTIMERSRC_LXTAL</code>	选择CK_LXTAL作为LPTIMER时钟源
<code>RCU_LPTIMERSRC_IRC16MDIV</code>	选择CK_IRC16MDIV作为LPTIMER时钟源
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* configure the LPTIMER clock source selection */
```

```
rcu_lptimer_clock_config(IDX_LPTIMER0, RCU_LPTIMERSRC_CKAPB1);
```

函数 rcu_lpuart_clock_config (仅适用于 GD32L233xx 系列)

函数rcu_lpuart_clock_config描述见下表:

表 3-585. 函数 rcu_lpuart_clock_config

函数名称	rcu_lpuart_clock_config
函数原形	void rcu_lpuart_clock_config(uint32_t ck_lpuart);
功能描述	配置LPUART时钟
先决条件	-
被调用函数	-
输入参数{in}	
ck_lpuart	LPUART时钟源选择
RCU_LPUARTSRC_CKAPB1	选择CK_APB1作为LPUART时钟源
RCU_LPUARTSRC_CKSYS	选择CK_SYS作为LPUART时钟源
RCU_LPUARTSRC_LXTAL	选择CK_LXTAL作为LPUART时钟源
RCU_LPUARTSRC_IRC16MDIV	选择CK_IRC16MDIV作为LPUART时钟源
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* configure the lpuart clock source selection */
```

```
rcu_lpuart_clock_config(RCU_LPUARTSRC_CKAPB1);
```

函数 rcu_lpuart_clock_config (仅适用于 GD32L235xx 系列)

函数rcu_lpuart_clock_config描述见下表:

表 3-586. 函数 rcu_lpuart_clock_config

函数名称	rcu_lpuart_clock_config
函数原形	void rcu_lpuart_clock_config(lpuart_idx_enum lpuart_idx, uint32_t ck_lpuart);
功能描述	配置LPUART时钟
先决条件	-
被调用函数	-
输入参数{in}	

lpuart_idx	lpuart索引, 参考 表3-562. 枚举类型lpuart_idx_enum
<i>IDX_LPUARTx</i>	LPUARTx的索引值(x=0,1)
ck_lpuart	LPUART时钟源选择
<i>RCU_LPUARTSRC</i> <i>_CKAPB1</i>	选择CK_APB1作为LPUART时钟源
<i>RCU_LPUARTSRC</i> <i>_CKSYS</i>	选择CK_SYS作为LPUART时钟源
<i>RCU_LPUARTSRC</i> <i>_LXTAL</i>	选择CK_LXTAL作为LPUART时钟源
<i>RCU_LPUARTSRC</i> <i>_IRC16MDIV</i>	选择CK_IRC16MDIV作为LPUART时钟源
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* configure the lpuart clock source selection */
```

```
rcu_lpuart_clock_config(IDX_LPUART0, RCU_LPUARTSRC_CKAPB1);
```

函数 rcu_irc16mdiv_clock_config

函数rcu_irc16mdiv_clock_config描述见下表:

表 3-587. 函数 rcu_irc16mdiv_clock_config

函数名称	rcu_irc16mdiv_clock_config
函数原形	void rcu_irc16mdiv_clock_config(uint32_t ck_irc16mdiv);
功能描述	配置IRC16MDIV时钟
先决条件	-
被调用函数	-
输入参数{in}	
ck_irc16mdiv	IRC16MDIV时钟源选择
<i>RCU_IRC16MDIV_</i> <i>NONE</i>	选择CK_IRC16M作为CK_IRC16MDIV时钟源
<i>RCU_IRC16MDIV_2</i>	选择CK_IRC16M/2作为CK_IRC16MDIV时钟源
<i>RCU_IRC16MDIV_4</i>	选择CK_IRC16M/4作为CK_IRC16MDIV时钟源
<i>RCU_IRC16MDIV_8</i>	选择CK_IRC16M/8作为CK_IRC16MDIV时钟源
<i>RCU_IRC16MDIV_16</i>	选择CK_IRC16M/16作为CK_IRC16MDIV时钟源
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* configure the IRC16MDIV clock selection */
rcu_irc16mdiv_clock_config(RCU_IRC16MDIV_2);
```

函数 rcu_usbd_clock_config

函数rcu_usbd_clock_config描述见下表：

表 3-588. 函数 rcu_irc16mdiv_clock_config

函数名称	rcu_usbd_clock_config
函数原形	void rcu_usbd_clock_config(uint32_t ck_usbd);
功能描述	配置USBID时钟
先决条件	-
被调用函数	-
输入参数{in}	
ck_usbd	USBID时钟源选择
RCU_USBDSRC_IRC48M	选择CK_IRC48M作为USBID时钟源
RCU_USBDSRC_PLL	选择CK_PLL作为USBID时钟源
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* configure the USBID clock source selection */
rcu_usbd_clock_config(RCU_USBDSRC_IRC48M);
```

函数 rcu_rtc_clock_config

函数rcu_rtc_clock_config描述见下表：

表 3-589. 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/SLCD时钟
先决条件	-
被调用函数	-
输入参数{in}	
rtc_clock_source	RTC/SLCD时钟源选择
RCU_RTC_SRC_NONE	未选择时钟

<i>RCU_RTCSRC_LX TAL</i>	选择CK_LXTAL作为RTC/SLCD时钟源
<i>RCU_RTCSRC_IRC 40K</i>	选择内部40K RC振荡器时钟作为RTC/SLCD时钟源
<i>RCU_RTCSRC_HX TAL_DIV32</i>	选择外部高速晶振32分频作为RTC/SLCD时钟源
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* configure the RTC clock source selection */
rcu_rtc_clock_config(RCU_RTCSRC_IRC40K);
```

函数 rcu_pll_source_ck_prediv_config

函数rcu_pll_source_ck_prediv_config描述见下表:

表 3-590. 函数 rcu_pll_source_ck_prediv_config

函数名称	rcu_pll_source_ck_prediv_config
函数原形	void rcu_pll_source_ck_prediv_config(uint32_t pllsource_ck_prediv);
功能描述	配置PLL输入源分频因子
先决条件	-
被调用函数	-
输入参数{in}	
pllsource_ck_prediv	PLL时钟源分频因子选择
<i>RCU_PLL_PREDIVx</i>	PLL时钟源的x分频作为PLL时钟 (x=1..16)
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* configure the PLL clock source selection */
rcu_hxtal_prediv_config(RCU_PLL_PREDIV2);
```

函数 rcu_lxtal_drive_capability_config

函数rcu_lxtal_drive_capability_config描述见下表:

表 3-591. 函数 rcu_lxtal_drive_capability_config

函数名称	rcu_lxtal_drive_capability_config
函数原形	void rcu_lxtal_drive_capability_config(uint32_t lxtal_dricap);
功能描述	配置LXTAL驱动能力
先决条件	-
被调用函数	-
输入参数{in}	
lxtal_dricap	LXTAL驱动能力
RCU_LXTAL_LOW DRI	低驱动力
RCU_LXTAL_MED_ LOWDRI	中低驱动力
RCU_LXTAL_MED_ HIGHDRI	中高驱动力
RCU_LXTAL_HIGH DRI	高驱动力
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* configure the LXTAL drive capability */
```

```
rcu_lxtal_drive_capability_config(RCU_LAXTAL_LOWDRI);
```

函数 rcu_lp_bandgap_config

函数rcu_lp_bandgap_config描述见下表：

表 3-592. 函数 rcu_lp_bandgap_config

函数名称	rcu_lp_bandgap_config
函数原形	void rcu_lp_bandgap_config(uint32_t lp_bandgap_clock);
功能描述	配置低功耗模式bandgap模式
先决条件	-
被调用函数	-
输入参数{in}	
lp_bandgap_clock	低功耗模式bandgap时钟
RCU_LPBM_32CLK	32个时钟周期
RCU_LPBM_64CLK	64个时钟周期
RCU_LPBM_128CL K	128个时钟周期
RCU_LPBM_256CL K	256个时钟周期

<i>RCU_LPBM_512CLK</i>	512个时钟周期
<i>RCU_LPBM_1024CLK</i>	1024个时钟周期
<i>RCU_LPBM_2048CLK</i>	2048个时钟周期
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* configure low power bandgap mode selection */
rcu_lp_bandgap_config(RCU_LPBM_64CLK);
```

函数 rcu_osci_stab_wait

函数rcu_osci_stab_wait描述见下表：

表 3-593. 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3-558. 枚举类型rcu_osci_type_enum
输出参数{out}	
-	-
返回值	
ErrStatus	SUCCESS 或 ERROR

例如：

```
/* wait for oscillator stabilization flag */
if(SUCCESS == rcu_osci_stab_wait(RCU_HXTAL)){
}
```

函数 rcu_osci_on

函数rcu_osci_on描述见下表：

表 3-594. 函数 rcu_osci_on

函数名称	rcu_osci_on
------	-------------

函数原形	void rcu_osc_on(rcu_osc_type_enum osci);
功能描述	打开振荡器
先决条件	-
被调用函数	-
输入参数{in}	
osci	振荡器类型，参考 表3-558. 枚举类型rcu_osc_type_enum
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* turn on the high speed crystal oscillator */
```

```
rcu_osc_on(RCU_HXTAL);
```

函数 rcu_osc_off

函数rcu_osc_off描述见下表：

表 3-595. 函数 rcu_osc_off

函数名称	rcu_osc_off
函数原形	void rcu_osc_off(rcu_osc_type_enum osci);
功能描述	关闭振荡器
先决条件	-
被调用函数	-
输入参数{in}	
osci	振荡器类型，参考 表3-558. 枚举类型rcu_osc_type_enum
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* turn off the high speed crystal oscillator */
```

```
rcu_osc_off(RCU_HXTAL);
```

函数 rcu_osc_bypass_mode_enable

函数rcu_osc_bypass_mode_enable描述见下表：

表 3-596. 函数 rcu_osc_bypass_mode_enable

函数名称	rcu_osc_bypass_mode_enable
函数原形	void rcu_osc_bypass_mode_enable(rcu_osc_type_enum osci);

功能描述	使能振荡器时钟旁路模式
先决条件	HXTALEN或LXTALEN应在使能振荡器时钟旁路模式前先复位
被调用函数	-
输入参数{in}	
osci	振荡器类型，参考 表3-558. 枚举类型rcu_osci_type_enum
RCU_HXTAL	高速晶体振荡器
RCU_LXTAL	低速晶体振荡器
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* enable the high speed crystal oscillator bypass mode */
```

```
rcu_osci_bypass_mode_enable(RCU_HXTAL);
```

函数 rcu_osci_bypass_mode_disable

函数rcu_osci_bypass_mode_disable描述见下表：

表 3-597. 函数 rcu_osci_bypass_mode_disable

函数名称	rcu_osci_bypass_mode_disable
函数原形	void rcu_osci_bypass_mode_disable(rcu_osci_type_enum osci);
功能描述	除能振荡器时钟旁路模式
先决条件	HXTALEN或LXTALEN应在使能振荡器时钟旁路模式前先复位
被调用函数	-
输入参数{in}	
osci	振荡器类型，参考 表3-558. 枚举类型rcu_osci_type_enum
RCU_HXTAL	高速晶体振荡器
RCU_LXTAL	低速晶体振荡器
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* disable the HXTAL clock monitor */
```

```
rcu_hxtal_clock_monitor_disable();
```

函数 rcu_irc16m_adjust_value_set

函数rcu_irc16m_adjust_value_set描述见下表：

表 3-598. 函数 rcu_irc16m_adjust_value_set

函数名称	rcu_irc16m_adjust_value_set
函数原形	void rcu_irc16m_adjust_value_set(uint32_t irc16m_adjval);
功能描述	设置内部16MHz RC振荡器时钟调整值
先决条件	-
被调用函数	-
输入参数{in}	
irc16m_adjval	IRC16M调整值（0到0x1F之间）
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* set the IRC16M adjust value */
rcu_irc16m_adjust_value_set(0x10);
```

函数 rcu_hxtal_clock_monitor_enable

函数rcu_hxtal_clock_monitor_enable描述见下表：

表 3-599. 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 3-600. 函数 rcu_hxtal_clock_monitor_disable

函数名称	rcu_hxtal_clock_monitor_disable
函数原形	void rcu_hxtal_clock_monitor_disable(void);
功能描述	除能HXTAL时钟监视器
先决条件	-
被调用函数	-
输入参数{in}	
-	-
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* disable the HXTAL clock monitor */
```

```
rcu_hxtal_clock_monitor_disable();
```

函数 rcu_lxtal_clock_monitor_enable

函数rcu_lxtal_clock_monitor_enable描述见下表:

表 3-601. 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 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 3-602. 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	除能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_voltage_key_unlock

函数rcu_voltage_key_unlock描述见下表:

表 3-603. 函数 rcu_voltage_key_unlock

函数名称	rcu_voltage_key_unlock
函数原形	void rcu_voltage_key_unlock(void);
功能描述	解锁电压寄存器
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 the voltage key*/
```

```
rcu_voltage_key_unlock();
```

函数 rcu_clock_freq_get

函数rcu_clock_freq_get描述见下表:

表 3-604. 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3-559. 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/ADC时钟 /USART0/1,LPUART/LPTIMER时钟频率,LPUART0/LPUART1/LPTIMER0/LPTIMER1时钟频率（仅适用于GD32L235xx系列）

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 3-605. 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3-554. 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 3-606. 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_flag_get

函数rcu_interrupt_flag_get描述见下表：

表 3-607. 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3-557. 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 3-608. 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3-556. 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);
```

函数 rcu_interrupt_enable

函数rcu_interrupt_enable描述见下表：

表 3-609. 函数 rcu_interrupt_enable

函数名称	rcu_interrupt_enable
函数原形	void rcu_interrupt_enable(rcu_int_enum stab_int);
功能描述	使能时钟稳定中断
先决条件	-
被调用函数	-
输入参数{in}	
stab_int	时钟稳定中断，具体参考 表3-557. 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 3-610. 函数 rcu_interrupt_disable

函数名称	rcu_interrupt_disable
函数原形	void rcu_interrupt_disable(rcu_int_enum stab_int);
功能描述	除能时钟稳定中断
先决条件	-
被调用函数	-
输入参数{in}	
stb_int	时钟稳定中断，具体参考 表3-557. 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);
```

3.21. 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[3.21.1](#)描述了RTC的寄存器列表，章节[3.21.2](#)对RTC库函数进行说明。

3.21.1. 外设寄存器描述

RTC寄存器列表如下表所示:

表 3-611. 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

寄存器名称	寄存器描述
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_BKP0	RTC备份域寄存器0
RTC_BKP1	RTC备份域寄存器1
RTC_BKP2	RTC备份域寄存器2
RTC_BKP3	RTC备份域寄存器3
RTC_BKP4	RTC备份域寄存器4

3.21.2. 外设库函数描述

RTC库函数列表如下表所示：

表 3-612. RTC 库函数

库函数名称	库函数描述
rtc_deinit	复位大多数RTC寄存器
rtc_init	初始化RTC寄存器
rtc_init_mode_enter	进入RTC初始化模式
rtc_init_mode_exit	退出RTC初始化模式
rtc_register_sync_wait	等待直到RTC_TIME和RTC_DATE寄存器与APB时钟同步，并且阴影寄存器被更新
rtc_current_time_get	获取当前的时间和日期
rtc_subsecond_get	获取当前的亚秒值
rtc_alarm_config	配置RTC闹钟
rtc_alarm_subsecond_config	配置RTC闹钟的亚秒值
rtc_alarm_get	获取RTC闹钟
rtc_alarm_subsecond_get	获取RTC闹钟亚秒值
rtc_alarm_enable	使能RTC 闹钟
rtc_alarm_disable	失能RTC 闹钟
rtc_timestamp_enable	使能RTC 时间戳
rtc_timestamp_disable	失能RTC时间戳
rtc_timestamp_get	获取RTC时间戳时间和日期
rtc_timestamp_internalevent_config	RTC时间戳内部事件配置
rtc_timestamp_subsecond_get	获取RTC时间戳亚秒值
rtc_tamper_enable	使能RTC侵入检测
rtc_tamper_disable	失能RTC侵入检测
rtc_tamper_mask	RTC侵入检测屏蔽配置

库函数名称	库函数描述
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
rtc_lxtal_stab_reset_enable	使能LXTAL时钟稳定复位
rtc_lxtal_stab_reset_disable	失能LXTAL时钟稳定复位

结构体 rtc_parameter_struct

表 3-613. 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 3-614. 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 3-615. 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 3-616. 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 3-617. 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 3-618. 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3-613. 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 3-619. 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 3-620. 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 3-621. 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 3-622. 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3-613. 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 3-623. 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 3-624. 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3-614. 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 3-625. 函数 rtc_alarm_subsecond_config

函数名称	rtc_alarm_subsecond_config
函数原型	void rtc_alarm_subsecond_config(uint8_t rtc_alarm, uint32_t mask_subsecond, uint32_t subsecond)
功能描述	配置RTC闹钟的亚秒值
先决条件	-
被调用函数	-
输入参数{in}	
rtc_alarm	RTC_ALARM0或者RTC_ALARM1
输入参数{in}	
mask_subsecond	闹钟亚秒屏蔽位
RTC_MASKSSC_0_14	屏蔽闹钟亚秒设置
RTC_MASKSSC_1_14	屏蔽RTC_ALRM0SS_SSC[14:1]，SSC[0]位用于时间匹配
RTC_MASKSSC_2_14	屏蔽RTC_ALRM0SS_SSC[14:2]，SSC[1:0]位用于时间匹配
RTC_MASKSSC_3_14	屏蔽RTC_ALRM0SS_SSC[14:3]，SSC[2:0]位用于时间匹配
RTC_MASKSSC_4_14	屏蔽RTC_ALRM0SS_SSC[14:4]，SSC[3:0]位用于时间匹配
RTC_MASKSSC_5_14	屏蔽RTC_ALRM0SS_SSC[14:5]，SSC[4:0]位用于时间匹配
RTC_MASKSSC_6_14	屏蔽RTC_ALRM0SS_SSC[14:6]，SSC[5:0]位用于时间匹配
RTC_MASKSSC_7_14	屏蔽RTC_ALRM0SS_SSC[14:7]，SSC[6:0]位用于时间匹配
RTC_MASKSSC_8_14	屏蔽RTC_ALRM0SS_SSC[14:8]，SSC[7:0]位用于时间匹配
RTC_MASKSSC_9_14	屏蔽RTC_ALRM0SS_SSC[14:9]，SSC[8:0]位用于时间匹配
RTC_MASKSSC_10_14	屏蔽RTC_ALRM0SS_SSC[14:10]，SSC[9:0]位用于时间匹配
RTC_MASKSSC_11_14	屏蔽RTC_ALRM0SS_SSC[14:11]，SSC[10:0]位用于时间匹配
RTC_MASKSSC_12_14	屏蔽RTC_ALRM0SS_SSC[14:12]，SSC[11:0]位用于时间匹配

<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 3-626. 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 3-627. 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 3-628. 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3-614. 结构体rtc_alarm_struct
返回值	
-	-

例如:

```
/*disable 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 3-629. 函数 rtc_alarm_subsecond_get

函数名称	rtc_alarm_subsecond_get
------	-------------------------

函数原型	uint32_t rtc_alarm_subsecond_get(uint8_t rtc_alarm);
功能描述	获取RTC闹钟亚秒值
先决条件	-
被调用函数	-
输入参数{in}	
rtc_alarm	RTC_ALARM0 or RTC_ALARM1
输出参数{out}	
-	-
返回值	
uint32_t	RTC 闹钟亚秒值(0x0-0x3FFF)

例如:

```
/*get RTC alarm subsecond*/
```

```
uint32_t subsecond = rtc_alarm_subsecond_get(RTC_ALARM0);
```

函数 rtc_timestamp_enable

函数rtc_timestamp_enable描述见下表:

表 3-630. 函数 rtc_timestamp_enable

函数名称	rtc_timestamp_enable
函数原型	void rtc_timestamp_enable(uint32_t edge);
功能描述	使能RTC时间戳
先决条件	-
被调用函数	-
输入参数{in}	
edge	选定哪种边沿触发时间戳检测
RTC_TIMESTAMP_RISING_EDGE	上升沿是时间戳事件有效检测沿
RTC_TIMESTAMP_FALLING_EDGE	下降沿是时间戳事件有效检测沿
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/*enable RTC time-stamp*/
```

```
rtc_timestamp_enable(RTC_TIMESTAMP_RISING_EDGE);
```

函数 rtc_timestamp_disable

函数rtc_timestamp_disable描述见下表:

表 3-631. 函数 `rtc_timestamp_disable`

函数名称	<code>rtc_timestamp_disable</code>
函数原型	<code>void rtc_timestamp_disable(void);</code>
功能描述	失能RTC时间戳
先决条件	-
被调用函数	-
输入参数{in}	
-	-
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/*disable RTC time-stamp*/
rtc_timestamp_disable();
```

函数 `rtc_timestamp_internalevent_config`

函数`rtc_timestamp_internalevent_config`描述见下表：

表 3-632. 函数 `rtc_timestamp_internalevent_config`

函数名称	<code>rtc_timestamp_internalevent_config</code>
函数原型	<code>void rtc_timestamp_internalevent_config(uint32_t mode)</code>
功能描述	配置RTC时间戳内部事件
先决条件	-
被调用函数	-
输入参数{in}	
mode	配置内部事件还是外部事件被检测
<code>RTC_ITSEN_DISABLE</code>	失能RTC时间戳内部事件
<code>RTC_ITSEN_ENABLE</code>	使能RTC时间戳内部事件
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* disable RTC time-stamp internal event */
rtc_timestamp_internalevent_config(RTC_ITSEN_DISABLE);
```

函数 `rtc_timestamp_get`

函数`rtc_timestamp_get`描述见下表：

表 3-633. 函数 `rtc_timestamp_get`

函数名称	<code>rtc_timestamp_get</code>
函数原型	<code>void rtc_timestamp_get(rtc_timestamp_struct* rtc_timestamp);</code>
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}	
<code>rtc_timestamp</code>	时间戳结构体，结构体成员参考 表3-615. 结构体 <code>rtc_timestamp_struct</code>
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 3-634. 函数 `rtc_timestamp_subsecond_get`

函数名称	<code>rtc_timestamp_subsecond_get</code>
函数原型	<code>uint32_t rtc_timestamp_subsecond_get(void);</code>
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	
<code>uint32_t</code>	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 3-635. 函数 `rtc_tamper_enable`

函数名称	<code>rtc_tamper_enable</code>
函数原型	<code>void rtc_tamper_enable(rtc_tamper_struct* rtc_tamper);</code>
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3-616. 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 3-636. 函数 `rtc_tamper_disable`

函数名称	<code>rtc_tamper_disable</code>
函数原型	<code>void rtc_tamper_disable(uint32_t source);</code>
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
<code>RTC_TAMPER0</code>	RTC 侵入检测0
<code>RTC_TAMPER1</code>	RTC 侵入检测1
<code>RTC_TAMPER2</code>	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_tamper_mask`

函数 `rtc_tamper_mask` 描述见下表：

表 3-637. 函数 `rtc_tamper_mask`

函数名称	<code>rtc_tamper_mask</code>
函数原型	<code>void rtc_tamper_mask(uint32_t source);</code>
功能描述	屏蔽RTC侵入检测
先决条件	-
被调用函数	-
输入参数{in}	
source	选定哪个侵入检测被屏蔽
<code>RTC_TAMPMASK_NONE</code>	侵入检测0、1、2都不被屏蔽
<code>RTC_TAMPMASK_TP0</code>	侵入检测0被屏蔽， <code>TP1IE</code> 和 <code>TP0IE</code> 被强制复位
<code>RTC_TAMPMASK_TP1</code>	侵入检测1被屏蔽， <code>TP1IE</code> 和 <code>TP1IE</code> 被强制复位
<code>RTC_TAMPMASK_TP2</code>	侵入检测2被屏蔽， <code>TP1IE</code> 和 <code>TP2IE</code> 被强制复位
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* both tamper 0~3 would not be masked */
```

```
rtc_tamper_mask(RTC_TAMPMASK_NONE);
```

函数 `rtc_tamper_without_bkp_reset`

函数 `rtc_tamper_without_bkp_reset` 描述见下表：

表 3-638. 函数 `rtc_tamper_without_bkp_reset`

函数名称	<code>rtc_tamper_without_bkp_reset</code>
函数原型	<code>void rtc_tamper_without_bkp_reset(uint32_t ne_source);</code>
功能描述	配置哪个侵入检测事件不擦除备份域寄存器
先决条件	-
被调用函数	-
输入参数{in}	
ne_source	配置哪个侵入检测事件不擦除备份域寄存器
<code>RTC_TAMPXNOERASE_NONE</code>	侵入检测0、侵入检测1和侵入检测2都擦除备份域寄存器
<code>RTC_TAMPXNOERASE_TP0</code>	侵入检测0不擦除备份域寄存器
<code>RTC_TAMPXNOERASE_TP1</code>	侵入检测1不擦除备份域寄存器

<i>RTC_TAMPXNOERAS E_TP2</i>	侵入检测2不擦除备份域寄存器
<i>RTC_TAMPXNOERAS E_TP0_TP1</i>	侵入检测0和侵入检测1都不擦除备份域寄存器
<i>RTC_TAMPXNOERAS E_TP_ALL</i>	侵入检测0、侵入检测1和侵入检测2都不擦除备份域寄存器
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* tamper 0~3 event all will not trigger RTC_BKP registers reset */
```

```
rtc_tamper_without_bkp_reset(RTC_TAMPXNOERASE_TP_ALL);
```

函数 rtc_output_pin_select

函数rtc_output_pin_select描述见下表:

表 3-639. 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_PB14</i>	RTC输出引脚为PB2 或者PB14
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 3-640. 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 3-641. 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
<code>RTC_CALIBRATION_512HZ</code>	当LSE频率为32768Hz并且RTC_PSC为默认值，输出512Hz信号

<i>RTC_CALIBRATION_1</i> HZ	当LSE频率为32768Hz并且RTC_PSC 为默认值，输出1Hz信号
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* when the LSE frequency is 32768Hz and the RTC_PSC
```

```
is the default value, output 1Hz signal */
```

```
rtc_calibration_output_config(RTC_CALIBRATION_1HZ);
```

函数 `rtc_hour_adjust`

函数`rtc_hour_adjust`描述见下表：

表 3-642. 函数 `rtc_hour_adjust`

函数名称	<code>rtc_hour_adjust</code>
函数原型	<code>void rtc_hour_adjust(uint32_t operation);</code>
功能描述	通过在当前时间上增加或者减少一个小时来适应夏令时和冬令时
先决条件	-
被调用函数	-
输入参数{in}	
operation	小时调整操作
<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 3-643. 函数 `rtc_second_adjust`

函数名称	<code>rtc_second_adjust</code>
函数原型	<code>ErrStatus rtc_second_adjust(uint32_t add, uint32_t minus);</code>
功能描述	调整RTC当前时间的秒或亚秒值

先决条件	-
被调用函数	-
输入参数{in}	
add	在当前时间上增加1S或者不增加
<i>RTC_SHIFT_ADD1S_R</i> <i>ESET</i>	无影响
<i>RTC_SHIFT_ADD1S_S</i> <i>ET</i>	在当前时间增加1秒
输入参数{in}	
minus	在当前是时间上减少的亚秒值(0x0 - 0x7FFF)
输出参数{out}	
-	-
返回值	
ErrStatus	ERROR或 SUCCESS

例如:

```
/* adjust RTC second or subsecond value of current time */
```

```
ErrStatus error_status = rtc_second_adjust(RTC_SHIFT_ADD1S_SET, 0);
```

函数 **rtc_bypass_shadow_enable**

函数rtc_bypass_shadow_enable描述见下表:

表 3-644. 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 3-645. 函数 `rtc_bypass_shadow_disable`

函数名称	<code>rtc_bypass_shadow_disable</code>
函数原型	<code>void rtc_bypass_shadow_disable(void);</code>
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 3-646. 函数 `rtc_refclock_detection_enable`

函数名称	<code>rtc_refclock_detection_enable</code>
函数原型	<code>ErrStatus rtc_refclock_detection_enable(void);</code>
功能描述	使能RTC参考时钟检测功能
先决条件	-
被调用函数	<code>rtc_init_mode_enter/rtc_init_mode_exit</code>
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 3-647. 函数 rtc_refclock_detection_disable

函数名称	rtc_refclock_detection_disable
函数原型	ErrStatus rtc_refclock_detection_disable(void);
功能描述	失能RTC参考时钟检测功能
先决条件	-
被调用函数	rtc_init_mode_enter/rtc_init_mode_exit
输入参数{in}	
-	-
输出参数{out}	
-	-
返回值	
ErrStatus	ERROR or SUCCESS

例如:

```
/* disable RTC reference clock detection function*/
```

```
ErrStatus error_status = rtc_refclock_detection_disable();
```

函数 rtc_wakeup_enable

函数rtc_wakeup_enable描述见下表:

表 3-648. 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 3-649. 函数 `rtc_wakeup_disable`

函数名称	<code>rtc_wakeup_disable</code>
函数原型	<code>ErrStatus rtc_wakeup_disable(void);</code>
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 3-650. 函数 `rtc_wakeup_clock_set`

函数名称	<code>rtc_wakeup_clock_set</code>
函数原型	<code>ErrStatus rtc_wakeup_clock_set(uint8_t wakeup_clock);</code>
功能描述	设置RTC自动唤醒定时器时钟
先决条件	-
被调用函数	-
输入参数{in}	
wakeup_clock	时钟选择
<code>WAKEUP_RTCK_DIV16</code>	RTC时钟的16分频
<code>WAKEUP_RTCK_DIV8</code>	RTC时钟的8分频
<code>WAKEUP_RTCK_DIV4</code>	RTC时钟的4分频
<code>WAKEUP_RTCK_DIV2</code>	RTC时钟的2分频
<code>WAKEUP_CKSPRE</code>	<code>ck_spre</code> (默认1Hz)时钟
<code>WAKEUP_CKSPRE_2EXP16</code>	<code>ck_spre</code> (默认1Hz)时钟并且将唤醒计数器值增加 2^{16}
输出参数{out}	
-	-
返回值	

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

例如:

```
/* RTC auto wakeup timer clock is ckspre */
```

```
ErrStatus error_status = rtc_wakeup_clock_set(WAKEUP_CKSPRE);
```

函数 rtc_wakeup_timer_set

函数rtc_wakeup_timer_set描述见下表:

表 3-651. 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 3-652. 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 3-653. 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 3-654. 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 3-655. 函数 rtc_interrupt_disable

函数名称	rtc_interrupt_disable
函数原型	void rtc_interrupt_disable(uint32_t interrupt);
功能描述	失能RTC指定中断
先决条件	-
被调用函数	-
输入参数{in}	
interrupt	选定被失能的RTC中断

<i>RTC_INT_TIMESTAMP</i>	时间戳中断
<i>RTC_INT_ALARM0</i>	闹钟0中断
<i>RTC_INT_ALARM1</i>	闹钟1中断
<i>RTC_INT_TAMP_ALL</i>	所有侵入检测中断
<i>RTC_INT_TAMP0</i>	侵入检测0中断
<i>RTC_INT_TAMP1</i>	侵入检测1中断
<i>RTC_INT_TAMP2</i>	侵入检测2中断
<i>RTC_INT_WAKEUP</i>	唤醒定时器中断
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* disble RTC ALARM interrupt */
```

```
rtc_interrupt_disable(RTC_INT_TAMP0);
```

函数 **rtc_flag_get**

函数rtc_flag_get描述见下表：

表 3-656. 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
<i>RTC_FLAG_SCP</i>	平滑校准挂起标志
<i>RTC_FLAG_TP2</i>	tamper 2事件标志
<i>RTC_FLAG_TP1</i>	tamper 1事件标志
<i>RTC_FLAG_TP0</i>	tamper 0事件标志
<i>RTC_FLAG_TSOVR</i>	时间戳事件溢出标志
<i>RTC_FLAG_TS</i>	时间戳事件标志
<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_INIT</i>	进入初始化模式
<i>RTC_FLAG_RSYN</i>	寄存器同步标志
<i>RTC_FLAG_YCM</i>	年份配置标志
<i>RTC_FLAG_SOP</i>	移位功能操作挂起标志
<i>RTC_FLAG_ALARM0</i>	Alarm0配置可写标志

<i>W</i>	
<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
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 3-657. 函数 `rtc_flag_clear`

函数名称	<code>rtc_flag_clear</code>
函数原型	<code>void rtc_flag_clear(uint32_t flag);</code>
功能描述	清除指定中断标志位
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_TP2</i>	tamper 2事件标志
<i>RTC_FLAG_TP1</i>	tamper 1事件标志
<i>RTC_FLAG_TP0</i>	tamper 0事件标志
<i>RTC_FLAG_TSOVR</i>	时间戳事件溢出标志
<i>RTC_FLAG_TS</i>	时间戳事件标志
<i>RTC_FLAG_WT</i>	唤醒标志
<i>RTC_FLAG_ALARM0</i>	Alarm0发生标志
<i>RTC_FLAG_ALARM1</i>	Alarm1发生标志
<i>RTC_FLAG_RSYN</i>	寄存器同步标志
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* cleartime-stamp event flag */
```

```
rtc_flag_clear(RTC_FLAG_TS);
```

函数 rtc_lxtal_stab_reset_enable

函数rtc_lxtal_stab_reset_enable描述见下表:

表 3-658. 函数 rtc_lxtal_stab_reset_enable

函数名称	rtc_lxtal_stab_reset_enable
函数原型	void rtc_lxtal_stab_reset_enable (void);
功能描述	使能RTC LXTAL时钟稳定复位
先决条件	-
被调用函数	-
输入参数{in}	
-	-
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* enable LXTAL stabilization reset*/
```

```
rtc_lxtal_stab_reset_enable ();
```

函数 rtc_lxtal_stab_reset_disable

函数rtc_lxtal_stab_reset_disable描述见下表:

表 3-659. 函数 rtc_lxtal_stab_reset_disable

函数名称	rtc_lxtal_stab_reset_disable
函数原型	void rtc_lxtal_stab_reset_disable (void);
功能描述	失能RTC LXTAL时钟稳定复位
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 LXTAL stabilization reset */
```

```
rtc_lxtal_stab_reset_disable( );
```

3.22. SLCD

SLCD驱动器通过自动产生SEG和COM交流电压信号来直接驱动LCD显示。章节[3.22.1](#)描述了SLCD的寄存器列表，章节[3.22.2](#)对SLCD库函数进行说明。

3.22.1. 外设寄存器说明

SLCD寄存器列表如下表所示：

表 3-660. SLCD 寄存器

寄存器名称	寄存器描述
SLCD_CTL	控制寄存器
SLCD_CFG	配置寄存器
SLCD_STAT	状态标志寄存器
SLCD_STATC	状态标志清除寄存器
SLCD_DATA0	显示数据寄存器0
SLCD_DATA1	显示数据寄存器1
SLCD_DATA2	显示数据寄存器2
SLCD_DATA3	显示数据寄存器3
SLCD_DATA4	显示数据寄存器4
SLCD_DATA5	显示数据寄存器5
SLCD_DATA6	显示数据寄存器6
SLCD_DATA7	显示数据寄存器7

3.22.2. 外设库函数说明

SLCD库函数列表如下表所示：

表 3-661. SLCD 寄存器

库函数名称	库函数描述
slcd_deinit	复位SLCD
slcd_enable	使能SLCD
slcd_disable	失能SLCD
slcd_init	初始化SLCD接口
slcd_enhance_mode_enable	使能SLCD增强模式（仅适用于GD32L233xx系列）
slcd_enhance_mode_disable	失能SLCD增强模式（仅适用于GD32L233xx系列）
slcd_weak_driving_resistance_select	选择SLCD弱驱动电阻（仅适用于GD32L235xx系列）
slcd_bias_voltage_select	选择SLCD偏置电压
slcd_duty_select	选择SLCD占空比
slcd_clock_config	配置SLCD时钟预分频器和时钟分频器
slcd_blink_mode_config	配置SLCD闪烁模式
slcd_contrast_ratio_config	配置SLCD对比度（仅适用于GD32L233xx系列）
slcd_dead_time_config	配置SLCD死区时间

库函数名称	库函数描述
slcd_pulse_on_duration_config	配置SLCD脉冲持续时间
slcd_com_seg_remap	SLCD COM/SEG引脚
slcd_voltage_source_select	选择SLCD电压源选择
slcd_high_drive_config	使能或失能SLCD高驱动
slcd_data_register_write	写SLCD显示数据寄存器
slcd_data_update_request	SLCD数据更新请求
slcd_flag_get	获取SLCD状态标志
slcd_flag_clear	清除SLCD状态标志
slcd_interrupt_enable	使能SLCD中断
slcd_interrupt_disable	失能SLCD中断
slcd_interrupt_flag_get	获取SLCD中断标志
slcd_interrupt_flag_clear	清除SLCD中断标志

枚举类型 slcd_data_register_enum

表 3-662. 枚举类型 slcd_data_register_enum

成员名称	功能描述
SLCD_DATA_REG0	SLCD显示数据寄存器0
SLCD_DATA_REG1	SLCD显示数据寄存器1
SLCD_DATA_REG2	SLCD显示数据寄存器2
SLCD_DATA_REG3	SLCD显示数据寄存器3
SLCD_DATA_REG4	SLCD显示数据寄存器4
SLCD_DATA_REG5	SLCD显示数据寄存器5
SLCD_DATA_REG6	SLCD显示数据寄存器6
SLCD_DATA_REG7	SLCD显示数据寄存器7

slcd_deinit

函数slcd_deinit描述见下表

表 3-663. 函数 slcd_deinit

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

例如:

```
/* reset the SLCD */
```

```
slcd_deinit();
```

slcd_enable

函数slcd_enable描述见下表

表 3-664. 函数 slcd_enable

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

例如：

```
/* enable the SLCD */
```

```
slcd_enable();
```

slcd_disable

函数slcd_disable描述见下表

表 3-665. 函数 slcd_disable

函数名称	slcd_disable
函数原形	void slcd_disable(void);
功能描述	失能SLCD
先决条件	-
输入参数{in}	
-	-
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* disable the SLCD */
```

```
slcd_disable();
```

slcd_init

函数slcd_init描述见下表:

表 3-666. 函数 slcd_init

函数名称	slcd_init
函数原形	void slcd_init(uint32_t prescaler, uint32_t divider, uint32_t duty, uint32_t bias);
功能描述	初始化SLCD接口
先决条件	-
输入参数{in}	
prescaler	SLCD时钟预分频器
SLCD_PRESCALER _1	$f_{PSC} = f_{in_clk}$
SLCD_PRESCALER _2	$f_{PSC} = f_{in_clk}/2$
SLCD_PRESCALER _4	$f_{PSC} = f_{in_clk}/4$
SLCD_PRESCALER _8	$f_{PSC} = f_{in_clk}/8$
SLCD_PRESCALER _16	$f_{PSC} = f_{in_clk}/16$
SLCD_PRESCALER _32	$f_{PSC} = f_{in_clk}/32$
SLCD_PRESCALER _64	$f_{PSC} = f_{in_clk}/64$
SLCD_PRESCALER _128	$f_{PSC} = f_{in_clk}/128$
SLCD_PRESCALER _256	$f_{PSC} = f_{in_clk}/256$
SLCD_PRESCALER _512	$f_{PSC} = f_{in_clk}/512$
SLCD_PRESCALER _1024	$f_{PSC} = f_{in_clk}/1024$
SLCD_PRESCALER _2048	$f_{PSC} = f_{in_clk}/2048$
SLCD_PRESCALER _4096	$f_{PSC} = f_{in_clk}/4096$
SLCD_PRESCALER _8192	$f_{PSC} = f_{in_clk}/8192$
SLCD_PRESCALER _16384	$f_{PSC} = f_{in_clk}/16384$
SLCD_PRESCALER _32768	$f_{PSC} = f_{in_clk}/32768$

输入参数{in}	
divider	分配
SLCD_DIVIDER_16	fSLCD = fPSC/16
SLCD_DIVIDER_17	fSLCD = fPSC/17
SLCD_DIVIDER_18	fSLCD = fPSC/18
SLCD_DIVIDER_19	fSLCD = fPSC/19
SLCD_DIVIDER_20	fSLCD = fPSC/20
SLCD_DIVIDER_21	fSLCD = fPSC/21
SLCD_DIVIDER_22	fSLCD = fPSC/22
SLCD_DIVIDER_23	fSLCD = fPSC/23
SLCD_DIVIDER_24	fSLCD = fPSC/24
SLCD_DIVIDER_25	fSLCD = fPSC/25
SLCD_DIVIDER_26	fSLCD = fPSC/26
SLCD_DIVIDER_27	fSLCD = fPSC/27
SLCD_DIVIDER_28	fSLCD = fPSC/28
SLCD_DIVIDER_29	fSLCD = fPSC/29
SLCD_DIVIDER_30	fSLCD = fPSC/30
SLCD_DIVIDER_31	fSLCD = fPSC/31
输入参数{in}	
duty	占空比选择
SLCD_DUTY_STATIC	占空比
SLCD_DUTY_1_2	1/2占空比
SLCD_DUTY_1_3	1/3占空比
SLCD_DUTY_1_4	1/4占空比
SLCD_DUTY_1_6	1/6占空比
SLCD_DUTY_1_8	1/8占空比
输入参数{in}	
bias	SLCD偏置电压
SLCD_BIAS_1_4	1/4 voltage bias
SLCD_BIAS_1_2	1/2 voltage bias
SLCD_BIAS_1_3	1/3 voltage bias
Output parameter{out}	
-	-
Return value	
-	-

例如:

```
/* initialize SLCD interface */
```

```
slcd_init(SLCD_PRESCALER_16,          SLCD_DIVIDER_20,          SLCD_DUTY_1_4,
SLCD_BIAS_1_4);
```

slcd_enhance_mode_enable（仅适用于 GD32L233xx 系列）

函数slcd_enhance_mode_enable描述见下表：

表 3-667. 函数 slcd_enhance_mode_enable

函数名称	slcd_enhance_mode_enable
函数原形	void slcd_enhance_mode_enable(void);
功能描述	使能SLCD增强模式
先决条件	-
输入参数{in}	
-	-
Output parameter{out}	
-	-
Return value	
-	-

例如：

```
/* enable SLCD enhance mode */
slcd_enhance_mode_enable();
```

slcd_enhance_mode_disable（仅适用于 GD32L233xx 系列）

函数slcd_enhance_mode_disable描述见下表：

表 3-668. 函数 slcd_enhance_mode_disable

函数名称	slcd_enhance_mode_disable
函数原形	void slcd_enhance_mode_disable(void);
功能描述	使能SLCD增强模式
先决条件	-
输入参数{in}	
-	-
Output parameter{out}	
-	-
Return value	
-	-

例如：

```
/* disable SLCD enhance mode */
slcd_enhance_mode_disable();
```

slcd_weak_driving_resistance_select（仅适用于 GD32L235xx 系列）

函数slcd_weak_driving_resistance_select描述见下表：

表 3-669. 函数 `slcd_weak_driving_resistance_select`

函数名称	<code>slcd_weak_driving_resistance_select</code>
函数原形	<code>void slcd_weak_driving_resistance_select(uint32_t resistance);</code>
功能描述	选择SLCD弱驱动电阻
先决条件	-
输入参数{in}	
resistance	弱驱动电阻
<code>SLCD_RSEL_6M</code>	弱驱动电阻6M
<code>SLCD_RSEL_4M</code>	弱驱动电阻4M
<code>SLCD_RSEL_2M</code>	弱驱动电阻2M
<code>SLCD_RSEL_1M</code>	弱驱动电阻1M
Output parameter{out}	
-	-
Return value	
-	-

例如:

```
/* select weak driving resistance */
```

```
slcd_weak_driving_resistance_select(SLCD_RSEL_1M);
```

`slcd_bias_voltage_select`

函数`slcd_bias_voltage_select`描述见下表:

表 3-670. 函数 `slcd_bias_voltage_select`

函数名称	<code>slcd_bias_voltage_select</code>
函数原形	<code>void slcd_bias_voltage_select(uint32_t bias_voltage);</code>
功能描述	选择SLCD偏置电压
先决条件	-
输入参数{in}	
bias_voltage	SLCD偏置电压
<code>SLCD_BIAS_1_4</code>	1/4 voltage bias
<code>SLCD_BIAS_1_2</code>	1/2 voltage bias
<code>SLCD_BIAS_1_3</code>	1/3 voltage bias
Output parameter{out}	
-	-
Return value	
-	-

例如:

```
/* set the SLCD 1/4 bias voltage */
```

```
slcd_bias_voltage_select(SLCD_BIAS_1_4);
```

slcd_duty_select

函数slcd_duty_select描述见下表:

表 3-671. 函数 slcd_duty_select

函数名称	slcd_duty_select
函数原形	void slcd_duty_select(uint32_t duty);
功能描述	选择SLCD占空比
先决条件	-
输入参数{in}	
duty	占空比选择
SLCD_DUTY_STAT1 C	占空比
SLCD_DUTY_1_2	1/2占空比
SLCD_DUTY_1_3	1/3占空比
SLCD_DUTY_1_4	1/4占空比
SLCD_DUTY_1_6	1/6占空比
SLCD_DUTY_1_8	1/8占空比
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* select SLCD duty cycle */
```

```
slcd_duty_select(SLCD_DUTY_1_2);
```

slcd_clock_config

函数slcd_clock_config描述见下表:

表 3-672. 函数 slcd_clock_config

函数名称	slcd_clock_config
函数原形	void slcd_clock_config(uint32_t prescaler, uint32_t divider);
功能描述	配置SLCD时钟预分频器和时钟分频器
先决条件	-
输入参数{in}	
prescaler	预分频
SLCD_PRESCALER _1	$f_{PSC} = f_{in_clk}$
SLCD_PRESCALER _2	$f_{PSC} = f_{in_clk}/2$
SLCD_PRESCALER _4	$f_{PSC} = f_{in_clk}/4$

SLCD_PRESCALER _8	$f_{PSC} = f_{in_clk}/8$
SLCD_PRESCALER _16	$f_{PSC} = f_{in_clk}/4$
SLCD_PRESCALER _32	$f_{PSC} = f_{in_clk}/32$
SLCD_PRESCALER _64	$f_{PSC} = f_{in_clk}/64$
SLCD_PRESCALER _128	$f_{PSC} = f_{in_clk}/128$
SLCD_PRESCALER _256	$f_{PSC} = f_{in_clk}/256$
SLCD_PRESCALER _512	$f_{PSC} = f_{in_clk}/512$
SLCD_PRESCALER _1024	$f_{PSC} = f_{in_clk}/1024$
SLCD_PRESCALER _2048	$f_{PSC} = f_{in_clk}/2048$
SLCD_PRESCALER _4096	$f_{PSC} = f_{in_clk}/4096$
SLCD_PRESCALER _8192	$f_{PSC} = f_{in_clk}/8192$
SLCD_PRESCALER _16384	$f_{PSC} = f_{in_clk}/16384$
SLCD_PRESCALER _32768	$f_{PSC} = f_{in_clk}/32768$
输入参数{in}	
divider	分频
SLCD_DIVIDER_16	$f_{SLCD} = f_{PSC}/16$
SLCD_DIVIDER_17	$f_{SLCD} = f_{PSC}/17$
SLCD_DIVIDER_18	$f_{SLCD} = f_{PSC}/18$
SLCD_DIVIDER_19	$f_{SLCD} = f_{PSC}/19$
SLCD_DIVIDER_20	$f_{SLCD} = f_{PSC}/20$
SLCD_DIVIDER_21	$f_{SLCD} = f_{PSC}/21$
SLCD_DIVIDER_22	$f_{SLCD} = f_{PSC}/22$
SLCD_DIVIDER_23	$f_{SLCD} = f_{PSC}/23$
SLCD_DIVIDER_24	$f_{SLCD} = f_{PSC}/24$
SLCD_DIVIDER_25	$f_{SLCD} = f_{PSC}/25$
SLCD_DIVIDER_26	$f_{SLCD} = f_{PSC}/26$
SLCD_DIVIDER_27	$f_{SLCD} = f_{PSC}/27$
SLCD_DIVIDER_28	$f_{SLCD} = f_{PSC}/28$
SLCD_DIVIDER_29	$f_{SLCD} = f_{PSC}/29$

SLCD_DIVIDER_30	$f_{SLCD} = f_{PSC}/30$
SLCD_DIVIDER_31	$f_{SLCD} = f_{PSC}/31$
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* configure the prescaler and the divider of SLCD clock */
```

```
slcd_clock_config(SLCD_PRESCALER_4, SLCD_DIVIDER_19);
```

slcd_blink_mode_config

函数slcd_blink_mode_config描述见下表：

表 3-673. 函数 slcd_blink_mode_config

函数名称	slcd_blink_mode_config
函数原形	void slcd_blink_mode_config(uint32_t mode,uint32_t blink_divider);
功能描述	配置SLCD闪烁模式
先决条件	-
输入参数{in}	
mode	闪烁模式
SLCD_BLINKMODE _OFF	不闪烁
SLCD_BLINKMODE _SEG0_COM0	闪烁SEG[0]、COM[0]
SLCD_BLINKMODE _SEG0_ALLCOM	闪烁SEG[0]和所有COM
SLCD_BLINKMODE _ALLSEG_ALLCOM	闪烁所有SEG和所有COM
输入参数{in}	
divider	闪烁分频器
SLCD_BLINK_FREQ UENCY_DIV8	$f_{BLINK} = f_{SLCD}/8$
SLCD_BLINK_FREQ UENCY_DIV16	$f_{BLINK} = f_{SLCD}/16$
SLCD_BLINK_FREQ UENCY_DIV32	$f_{BLINK} = f_{SLCD}/32$
SLCD_BLINK_FREQ UENCY_DIV64	$f_{BLINK} = f_{SLCD}/64$
SLCD_BLINK_FREQ UENCY_DIV128	$f_{BLINK} = f_{SLCD}/128$
SLCD_BLINK_FREQ	$f_{BLINK} = f_{SLCD}/256$

UENCY_DIV256	
SLCD_BLINK_FREQ UENCY_DIV512	$f_{BLINK} = f_{SLCD}/512$
SLCD_BLINK_FREQ UENCY_DIV1024	$f_{BLINK} = f_{SLCD}/1024$
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* configure SLCD blink mode */
```

```
slcd_blink_mode_config(SLCD_BLINKMODE_SEG0_COM0,  
SLCD_BLINK_FREQUENCY_DIV8);
```

slcd_contrast_ratio_config

函数slcd_contrast_ratio_config描述见下表

表 3-674. 函数 slcd_contrast_ratio_config

函数名称	slcd_contrast_ratio_config
函数原形	void slcd_contrast_ratio_config(uint32_t contrast_ratio);
功能描述	配置SLCD对比度
先决条件	-
输入参数{in}	
contrast_ratio	指定VSLCD电压
SLCD_CONTRAST_LEVEL_0	SLCD最大电压 = 2.65V
SLCD_CONTRAST_LEVEL_1	SLCD最大电压 = 2.80V
SLCD_CONTRAST_LEVEL_2	SLCD最大电压= 2.92V
SLCD_CONTRAST_LEVEL_3	SLCD最大电压= 3.08V
SLCD_CONTRAST_LEVEL_4	SLCD最大电压= 3.23V
SLCD_CONTRAST_LEVEL_5	SLCD最大电压= 3.37V
SLCD_CONTRAST_LEVEL_6	SLCD最大电压= 3.52V
SLCD_CONTRAST_LEVEL_7	SLCD最大电压= 3.67V
输出参数{out}	

-	-
返回值	
-	-

例如:

```
/* configure SLCD contrast ratio */
```

```
slcd_contrast_ratio_config(SLCD_CONTRAST_LEVEL_0);
```

slcd_dead_time_config

函数slcd_dead_time_config描述见下表

表 3-675. 函数 slcd_dead_time_config

函数名称	slcd_dead_time_config
函数原形	void slcd_dead_time_config(uint32_t dead_time);
功能描述	配置SLCD死区时间
先决条件	-
输入参数{in}	
dead_time	帧间死区时间长度
SLCD_DEADTIME_PERIOD_0	无死区时间
SLCD_DEADTIME_PERIOD_1	1相周期死区时间
SLCD_DEADTIME_PERIOD_2	2相周期死区时间
SLCD_DEADTIME_PERIOD_3	3相周期死区时间
SLCD_DEADTIME_PERIOD_4	4相周期死区时间
SLCD_DEADTIME_PERIOD_5	5相周期死区时间
SLCD_DEADTIME_PERIOD_6	6相周期死区时间
SLCD_DEADTIME_PERIOD_7	7相周期死区时间
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* configure SLCD dead time duration */
```

```
slcd_dead_time_config(SLCD_DEADTIME_PERIOD_1);
```

slcd_pulse_on_duration_config（仅适用于 GD32L233xx 系列）

函数slcd_pulse_on_duration_config描述见下表：

表 3-676. 函数 slcd_pulse_on_duration_config

函数名称	slcd_pulse_on_duration_config
函数原形	void slcd_pulse_on_duration_config(uint32_t duration);
功能描述	配置SLCD脉冲持续时间
先决条件	-
输入参数{in}	
duration	根据PSC脉冲定义脉冲持续时间
SLCD_PULSEON_DURATION_0	脉冲持续时间 = 0
SLCD_PULSEON_DURATION_1	脉冲持续时间 = $1/f_{psc}$
SLCD_PULSEON_DURATION_2	脉冲持续时间 = $2/f_{psc}$
SLCD_PULSEON_DURATION_3	脉冲持续时间 = $3/f_{psc}$
SLCD_PULSEON_DURATION_4	脉冲持续时间 = $4/f_{psc}$
SLCD_PULSEON_DURATION_5	脉冲持续时间 = $5/f_{psc}$
SLCD_PULSEON_DURATION_6	脉冲持续时间 = $6/f_{psc}$
SLCD_PULSEON_DURATION_7	脉冲持续时间 = $7/f_{psc}$
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* configure SLCD pulse on duration */
```

```
slcd_pulse_on_duration_config(SLCD_PULSEON_DURATION_7);
```

slcd_com_seg_remap

函数slcd_com_seg_remap描述见下表：

表 3-677. 函数 slcd_com_seg_remap

函数名称	slcd_com_seg_remap
函数原形	void slcd_com_seg_remap(ControlStatus newvalue);
功能描述	选择COM/SEG引脚

先决条件	-
输入参数{in}	
newvalue	使能或失能
DISABLE	SLCD_COM[7:4]引脚选择SLCD_COM[7:4]
ENABLE	SLCD_COM[7:4]引脚选择SLCD_SEG[31:28]
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* SLCD common/segment pad select */
```

```
slcd_com_seg_remap(ENABLE);
```

slcd_voltage_source_select

函数slcd_voltage_source_select见下表：

表 3-678. 函数 slcd_voltage_source_select

函数名称	slcd_voltage_source_select
函数原形	void slcd_voltage_source_select(uint8_t voltage_source);
功能描述	选择SLCD电压源
先决条件	-
输入参数{in}	
voltage_source	SLCD电压源
SLCD_VOLTAGE_INTERNAL	内部电压源
SLCD_VOLTAGE_EXTERNAL	外部电压源(VSLCD引脚)
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* select SLCD voltage source */
```

```
slcd_voltage_source_select(SLCD_VOLTAGE_EXTERNAL);
```

slcd_high_drive_config

函数slcd_high_drive_config描述见下表

表 3-679. 函数 `slcd_high_drive_config`

函数名称	<code>slcd_high_drive_config</code>
函数原形	<code>void slcd_high_drive_config(ControlStatus newvalue);</code>
功能描述	使能/失能高驱动
先决条件	-
输入参数{in}	
<code>newvalue</code>	使能/失能
<code>ENABLE</code>	使能高驱动
<code>DISABLE</code>	失能高驱动
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* enable permanent high drive */
```

```
slcd_high_drive_config(ENABLE);
```

`slcd_data_register_write`

函数`slcd_data_register_write`描述见下表：

表 3-680. 函数 `slcd_data_register_write`

函数名称	<code>slcd_data_register_write</code>
函数原形	<code>void slcd_data_register_write(slcd_data_register_enum register_number, uint32_t data);</code>
功能描述	写SLCD显示寄存器
先决条件	-
输入参数{in}	
<code>register_number</code>	参考 表3-662. 枚举类型slcd_data_register_enum
<code>SLCD_DATA_REGx(x=0, 1, ..., 7)</code>	<code>SLCD_DATAx</code>
输入参数{in}	
<code>data</code>	写入的数据值
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* write slcd display data registers */
```

```
slcd_data_register_write(SLCD_DATA_REG0, 0x0000FFFF);
```

slcd_data_update_request

函数slcd_data_update_request描述见下表

表 3-681. 函数 slcd_data_update_request

函数名称	slcd_data_update_request
函数原形	void slcd_data_update_request(void);
功能描述	SLCD数据更新请求
先决条件	-
输入参数{in}	
-	-
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* update SLCD data request */
slcd_data_update_request();
```

slcd_flag_get

函数slcd_flag_get描述见下表：

表 3-682. 函数 slcd_flag_get

函数名称	slcd_flag_get
函数原形	FlagStatus slcd_flag_get(uint8_t flag);
功能描述	获取SLCD状态标志
先决条件	-
输入参数{in}	
flag	状态标志
SLCD_FLAG_ON	SLCD控制器开始标志
SLCD_FLAG_SOF	帧开始标志
SLCD_FLAG_UPR	SLCD数据更新请求标志
SLCD_FLAG_UPD	更新SLCD数据完成标志
SLCD_FLAG_VRDY	SLCD电压就绪标志（仅适用于GD32L233xx系列）
SLCD_FLAG_SYN	SLCD CFG寄存器同步标志
输出参数{out}	
-	-
返回值	
FlagStatus	SET或RESET

例如：

```
/* get the SLCD status flag */
```

```
slcd_flag_get(SLCD_FLAG_ON);
```

slcd_flag_clear

函数slcd_flag_clear描述见下表：

表 3-683. 函数 slcd_flag_clear

函数名称	slcd_flag_clear
函数原形	void slcd_flag_clear(uint8_t flag);
功能描述	清除SLCD状态标志
先决条件	-
输入参数{in}	
flag	状态标志
SLCD_FLAG_SOF	帧起始标志
SLCD_FLAG_UPD	更新SLCD数据完成标志
输入参数{out}	
-	-
返回值	
-	-

例如：

```
/* clear the SLCD status flag */
slcd_flag_clear(SLCD_FLAG_SOF);
```

slcd_interrupt_enable

函数slcd_interrupt_enable描述见下表：

表 3-684. 函数 slcd_interrupt_enable

函数名称	slcd_interrupt_enable
函数原形	void slcd_interrupt_enable(uint32_t interrupt);
功能描述	使能SLCD中断
先决条件	-
输入参数{in}	
interrupt	中断源
SLCD_INT_SOF	帧开始中断
SLCD_INT_UPD	更新完成中断
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* enable the start of frame interrupt */
```

```
slcd_interrupt_enable(SLCD_INT_SOF);
```

slcd_interrupt_disable

函数slcd_interrupt_disable描述见下表：

表 3-685. 函数 slcd_interrupt_disable

函数名称	slcd_interrupt_disable
函数原形	void slcd_interrupt_disable(uint32_t interrupt);
功能描述	失能SLCD中断
先决条件	-
输入参数{in}	
interrupt	中断源
SLCD_INT_SOF	帧开始中断
SLCD_INT_UPD	更新完成中断
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* disable the start of frame interrupt */
```

```
slcd_interrupt_disable(SLCD_INT_SOF);
```

slcd_interrupt_flag_get

函数slcd_interrupt_flag_get描述见下表：

表 3-686. 函数 slcd_interrupt_flag_get

函数名称	slcd_interrupt_flag_get
函数原形	FlagStatus slcd_interrupt_flag_get(uint8_t interrupt);
功能描述	获取SLCD中断标志
先决条件	-
输入参数{in}	
interrupt	中断源
SLCD_INT_FLAG_SOF	帧起始中断
SLCD_INT_FLAG_UPD	SLCD更新完成中断
输出参数{out}	
-	-
返回值	
FlagStatus	SET or RESET

例如：

```
/* get the SLCD interrupt flag */
slcd_interrupt_flag_get(SLCD_INT_FLAG_SOF);
```

slcd_interrupt_flag_clear

函数slcd_interrupt_flag_clear描述见下表：

表 3-687. 函数 slcd_interrupt_flag_clear

函数名称	slcd_interrupt_flag_clear
函数原形	FlagStatus slcd_interrupt_flag_clear(uint8_t interrupt);
功能描述	清除SLCD中断标志
先决条件	-
输入参数{in}	
interrupt	中断源
SLCD_INT_FLAG_SOF	帧起始中断
SLCD_INT_FLAG_UPD	SLCD更新完成中断
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* clear the SLCD interrupt flag */
slcd_interrupt_flag_clear(SLCD_INT_FLAG_SOF);
```

3.23. SPI

SPI/I2S模块可以通过SPI协议或I2S音频协议与外部设备进行通信。章节[3.23.1](#)描述了SPI/I2S的寄存器列表，章节[3.23.2](#)对SPI/I2S库函数进行说明。

3.23.1. 外设寄存器说明

SPI/I2S寄存器列表如下表所示：

表 3-688. SPI/I2S 寄存器

寄存器名称	寄存器描述
SPI_CTL0	控制寄存器0
SPI_CTL1	控制寄存器1
SPI_STAT	状态寄存器

寄存器名称	寄存器描述
SPI_DATA	数据寄存器
SPI_CRCPOLY	CRC多项式寄存器
SPI_RCRC	接收CRC寄存器
SPI_TCRC	发送CRC寄存器
SPI_I2SCTL	I2S控制寄存器
SPI_I2SPSC	I2S时钟预分频寄存器
SPI_QCTL	四线SPI控制寄存器

3.23.2. 外设库函数说明

SPI/I2S库函数列表如下表所示：

表 3-689. SPI/I2S 库函数

库函数名称	库函数描述
spi_i2s_deinit	复位外设SPI/I2S
spi_struct_para_init	初始化SPI结构体中所有参数为默认值
spi_init	初始化外设SPI
spi_enable	使能外设SPI
spi_disable	失能外设SPI
i2s_init	初始化外设I2S
i2s_psc_config	配置I2S预分频器
i2s_enable	使能外设I2S
i2s_disable	失能外设I2S
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_i2s_data_frame_format_config	配置外设SPI/I2S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_i2s_data_transmit	发送数据
spi_i2s_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

库函数名称	库函数描述
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
spi_nssp_mode_disable	禁能SPI NSS脉冲模式
spi_quad_enable	使能四线SPI模式
spi_quad_disable	禁能四线SPI模式
spi_quad_write_enable	使能四线SPI写
spi_quad_read_enable	使能四线SPI读
spi_i2s_format_error_clear	清除SPI/I2S格式错误标志
spi_i2s_flag_get	获取外设SPI/I2S标志状态
spi_i2s_interrupt_enable	使能外设SPI/I2S中断
spi_i2s_interrupt_disable	失能外设SPI/I2S中断
spi_i2s_interrupt_flag_get	获取外设SPI/I2S中断状态

结构体 spi_parameter_struct

表 3-690. 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_i2s_deinit

函数spi_i2s_deinit描述见下表:

表 3-691. 函数 spi_i2s_deinit

函数名称	spi_i2s_deinit
------	----------------

函数原形	void spi_i2s_deinit(uint32_t spi_periph);
功能描述	复位外设SPI/I2S
先决条件	-
被调用函数	rcu_periph_reset_enable / rcu_periph_reset_disable
输入参数{in}	
spi_periph	外设SPIx
SPIx (x=0,1)	SPI外设选择
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* reset SPI0 */
```

```
spi_i2s_deinit(SPI0);
```

函数 spi_struct_para_init

函数spi_struct_para_init描述见下表：

表 3-692. 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}	
spi_struct	SPI初始化结构体，结构体成员参考 表3-690. 结构体spi_parameter_struct
输出参数{out}	
-	-
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 3-693. 函数 spi_init

函数名称	spi_init
函数原形	ErrStatus 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SPI
SPIx (x=0,1)	SPI外设选择
输入参数{in}	
spi_struct	初始化结构体，结构体成员参考 表3-690. 结构体spi_parameter_struct
输出参数{out}	
-	-
返回值	
ErrStatus	ERROR或者SUCCESS

例如：

```

/* initialize SPI0 */

spi_parameter_struct spi_init_struct;

ErrStatus errstatus = ERROR;

spi_init_struct.trans_mode      = SPI_TRANSMODE_BDTRANSMIT;
spi_init_struct.device_mode     = SPI_MASTER;
spi_init_struct.frame_size      = SPI_FRAME_SIZE_8BIT;
spi_init_struct.clock_polarity_phase = SPI_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;

errorstatus = spi_init(SPI0, &spi_init_struct);

```

函数 spi_enable

函数spi_enable描述见下表：

表 3-694. 函数 spi_enable

函数名称	spi_enable
函数原形	void spi_enable(uint32_t spi_periph);
功能描述	使能外设SPI
先决条件	-
被调用函数	-

输入参数{in}	
spi_periph	外设SPIx
<i>SPIx</i> (<i>x=0,1</i>)	SPI外设选择
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* enable SPI0 */
spi_enable(SPI0);
```

函数 spi_disable

函数spi_disable描述见下表:

表 3-695. 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	失能外设SPI
先决条件	-
被调用函数	-
输入参数{in}	
spi_periph	外设SPI
<i>SPIx</i> (<i>x=0,1</i>)	SPI外设选择
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* disable SPI0 */
spi_disable(SPI0);
```

函数 i2s_init

函数i2s_init描述见下表:

表 3-696. 函数 i2s_init

函数名称	i2s_init
函数原形	void i2s_init(uint32_t spi_periph, uint32_t mode, uint32_t standard, uint32_t ckpl);
功能描述	初始化外设I2S

先决条件	-
被调用函数	-
输入参数{in}	
spi_periph	外设I2S
SPIx (x=1)	I2S外设选择
输入参数{in}	
mode	I2S运行模式
I2S_MODE_SLAVE_TX	I2S从机发送模式
I2S_MODE_SLAVE_RX	I2S从机接收模式
I2S_MODE_MASTERTX	I2S主机发送模式
I2S_MODE_MASTERRX	I2S主机接收模式
输入参数{in}	
standard	I2S标准选择
I2S_STD_PHILLIPS	I2S飞利浦标准
I2S_STD_MSB	I2S MSB对齐标准
I2S_STD_LSB	I2S LSB对齐标准
I2S_STD_PCMSHORT	I2S PCM短帧标准
I2S_STD_PCMLONG	I2S PCM长帧标准
输入参数{in}	
ckpl	I2S空闲状态时钟极性
I2S_CKPL_LOW	I2S_CK空闲状态为低电平
I2S_CKPL_HIGH	I2S_CK空闲状态为高电平
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* initialize I2S1 */
```

```
i2s_init(SPI1, I2S_MODE_MASTERTX, I2S_STD_PHILLIPS, I2S_CKPL_LOW);
```

函数 i2s_psc_config

函数i2s_psc_config描述见下表：

表 3-697. 函数 i2s_psc_config

函数名称	i2s_psc_config
------	----------------

函数原形	void i2s_psc_config(uint32_t spi_periph, uint32_t audiosample, uint32_t frameformat, uint32_t mckout);
功能描述	配置I2S预分频器
先决条件	-
被调用函数	rcu_clock_freq_get
输入参数{in}	
spi_periph	外设I2S
SPIx (x=1)	I2S外设选择
输入参数{in}	
audiosample	I2S音频采样频率
I2S_AUDIOSAMPL E_8K	音频采样频率为8KHz
I2S_AUDIOSAMPL E_11K	音频采样频率为11KHz
I2S_AUDIOSAMPL E_16K	音频采样频率为16KHz
I2S_AUDIOSAMPL E_22K	音频采样频率为22KHz
I2S_AUDIOSAMPL E_32K	音频采样频率为32KHz
I2S_AUDIOSAMPL E_44K	音频采样频率为44KHz
I2S_AUDIOSAMPL E_48K	音频采样频率为48KHz
I2S_AUDIOSAMPL E_96K	音频采样频率为96KHz
I2S_AUDIOSAMPL E_192K	音频采样频率为192KHz
输入参数{in}	
frameformat	I2S数据长度和通道长度
I2S_FRAMEFORMA T_DT16B_CH16B	I2S数据长度为16位，通道长度为16位
I2S_FRAMEFORMA T_DT16B_CH32B	I2S数据长度为16位，通道长度为32位
I2S_FRAMEFORMA T_DT24B_CH32B	I2S数据长度为24位，通道长度为32位
I2S_FRAMEFORMA T_DT32B_CH32B	I2S数据长度为32位，通道长度为32位
输入参数{in}	
mckout	I2S_MCK输出
I2S_MCKOUT_ENA BLE	I2S_MCK输出使能

I2S_MCKOUT_DISABLE	I2S_MCK输出禁止
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* configure I2S1 prescaler */
```

```
i2s_psc_config(SPI1, I2S_AUDIOSAMPLE_44K, I2S_FRAMEFORMAT_DT16B_CH16B,
I2S_MCKOUT_DISABLE);
```

函数 i2s_enable

函数i2s_enable描述见下表：

表 3-698. 函数 i2s_enable

函数名称	i2s_enable
函数原形	void i2s_enable(uint32_t spi_periph);
功能描述	使能外设I2S
先决条件	-
被调用函数	-
输入参数{in}	
spi_periph	外设I2S1
SPIx (x=1)	I2S外设选择
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* enable I2S1 */
```

```
i2s_enable(SPI1);
```

函数 i2s_disable

函数i2s_disable描述见下表：

表 3-699. 函数 i2s_disable

函数名称	i2s_disable
函数原形	void i2s_disable(uint32_t spi_periph);
功能描述	失能外设I2S
先决条件	-

被调用函数	-
输入参数{in}	
spi_periph	外设I2S
<i>SPIx (x=1)</i>	I2S外设选择
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* disable I2S1 */
i2s_disable(SPI1);
```

函数 spi_nss_output_enable

函数spi_nss_output_enable描述见下表:

表 3-700. 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SPI
<i>SPIx (x=0,1)</i>	SPI外设选择
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* enable SPI0 NSS output */
spi_nss_output_enable(SPI0);
```

函数 spi_nss_output_disable

函数spi_nss_output_disable描述见下表:

表 3-701. 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	失能外设SPI NSS输出

先决条件	-
被调用函数	-
输入参数{in}	
spi_periph	外设SPI
<i>SPIx</i> (<i>x=0,1</i>)	SPI外设选择
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* disable SPI0 NSS output */
```

```
spi_nss_output_disable(SPI0);
```

函数 spi_nss_internal_high

函数spi_nss_internal_high描述见下表：

表 3-702. 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SPI
<i>SPIx</i> (<i>x=0,1</i>)	SPI外设选择
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* SPI0 NSS pin is pulled high level in software mode */
```

```
spi_nss_internal_high(SPI0);
```

函数 spi_nss_internal_low

函数spi_nss_internal_low描述见下表：

表 3-703. 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SPI
<i>SPIx</i> (<i>x</i> =0,1)	SPI外设选择
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* SPI0 NSS pin is pulled low level in software mode */
```

```
spi_nss_internal_low(SPI0);
```

函数 spi_dma_enable

函数spi_dma_enable描述见下表:

表 3-704. 函数 spi_dma_enable

函数名称	spi_dma_enable
函数原形	void spi_dma_enable(uint32_t spi_periph, uint8_t dma);
功能描述	使能外设SPI的DMA功能
先决条件	-
被调用函数	-
输入参数{in}	
spi_periph	外设SPI
<i>SPIx</i> (<i>x</i> =0,1)	SPI外设选择
输入参数{in}	
dma	SPI DMA模式
<i>SPI_DMA_TRANSMIT</i>	SPI发送缓冲区DMA使能
<i>SPI_DMA_RECEIVE</i>	SPI接收缓冲区DMA使能
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* enable SPI0 transmit data DMA function */
```

```
spi_dma_enable(SPI0, SPI_DMA_TRANSMIT);
```


函数 **spi_dma_disable**

函数spi_dma_disable描述见下表:

表 3-705. 函数 **spi_dma_disable**

函数名称	spi_dma_disable
函数原形	void spi_dma_disable(uint32_t spi_periph, uint8_t dma);
功能描述	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SPI
SPIx (x=0,1)	SPI外设选择
输入参数{in}	
dma	SPI DMA模式
SPI_DMA_TRANSMIT	SPI发送缓冲区DMA失能
SPI_DMA_RECEIVE	SPI接收缓冲区DMA失能
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* disable SPI0 transmit data DMA function */
```

```
spi_dma_disable(SPI0, SPI_DMA_TRANSMIT);
```

函数 **spi_transmit_odd_config**

函数spi_transmit_odd_config描述见下表:

表 3-706. 函数 **spi_transmit_odd_config**

函数名称	spi_transmit_odd_config
函数原形	void spi_transmit_odd_config(uint32_t spi_periph, uint16_t odd);
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}	
spi_periph	外设SPI
SPIx (x=0)	SPI外设选择
输入参数{in}	
odd	DMA通道发送的字节数是奇数还是偶数
SPI_TXDMA_EVEN	DMA发送的字节数是偶数

<i>SPI_TXDMA_ODD</i>	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 3-707. 函数 spi_receive_odd_config

函数名称	spi_receive_odd_config
函数原形	void spi_receive_odd_config(uint32_t spi_periph, uint16_t odd);
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}	
spi_periph	外设SPI
<i>SPIx (x=0)</i>	SPI外设选择
输入参数{in}	
<i>odd</i>	DMA通道接收的字节数时奇数还是偶数
<i>SPI_RXDMA_EVEN</i>	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_i2s_data_frame_format_config**

函数spi_i2s_data_frame_format_config描述见下表：

表 3-708. 函数 spi_i2s_data_frame_format_config

函数名称	spi_i2s_data_frame_format_config
函数原形	ErrStatus spi_i2s_data_frame_format_config(uint32_t spi_periph, uint16_t

	frame_format);
功能描述	配置外设SPI/I2S数据帧格式
先决条件	-
被调用函数	-
输入参数{in}	
spi_periph	外设SPI
SPIx (x=0,1)	SPI外设选择
输入参数{in}	
frame_format	SPI帧大小
SPI_FRAME_SIZE_x BIT	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 SPI1/I2S1 data frame format size is 16 bits */
```

```
spi_i2s_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 3-709. 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)	SPI外设选择
输入参数{in}	
fifo_access_size	fifo访问大小
SPI_HALFWORD_A CCESS	半字访问
SPI_BYTE_ACCES S	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 3-710. 函数 `spi_bidirectional_transfer_config`

函数名称	<code>spi_bidirectional_transfer_config</code>
函数原形	<code>void spi_bidirectional_transfer_config(uint32_t spi_periph, uint32_t transfer_direction);</code>
功能描述	配置外设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
<i>SPIx (x=0,1)</i>	SPI外设选择
输入参数{in}	
transfer_direction	SPI双向传输输出使能
<i>SPI_BIDIRECTIONAL_TRANSMIT</i>	SPI工作在只发送模式
<i>SPI_BIDIRECTIONAL_RECEIVE</i>	SPI工作在只接收模式
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* SPI0 works in transmit-only mode */
```

```
spi_bidirectional_transfer_config(SPI0, SPI_BIDIRECTIONAL_TRANSMIT);
```

函数 **spi_i2s_data_transmit**

函数 `spi_i2s_data_transmit` 描述见下表:

表 3-711. 函数 `spi_i2s_data_transmit`

函数名称	<code>spi_i2s_data_transmit</code>
函数原形	<code>void spi_i2s_data_transmit(uint32_t spi_periph, uint16_t data);</code>
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SPI
<i>SPIx</i> (<i>x=0,1</i>)	SPI外设选择
输入参数{in}	
data	16位数据
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* SPI0 transmit data */
```

```
spi_i2s_data_transmit(SPI0, spi0_send_array[send_n]);
```

函数 spi_i2s_data_receive

函数spi_i2s_data_receive描述见下表:

表 3-712. 函数 spi_i2s_data_receive

函数名称	spi_i2s_data_receive
函数原形	uint16_t spi_i2s_data_receive(uint32_t spi_periph);
功能描述	SPI接收数据
先决条件	-
被调用函数	-
输入参数{in}	
spi_periph	外设SPI
<i>SPIx</i> (<i>x=0,1</i>)	SPI外设选择
输出参数{out}	
-	-
返回值	
uint16_t	16位数据

例如:

```
/* SPI0 receive data */
```

```
spi0_receive_array[receive_n] = spi_i2s_data_receive(SPI0);
```

函数 spi_crc_polynomial_set

函数spi_crc_polynomial_set描述见下表:

表 3-713. 函数 spi_crc_polynomial_set

函数名称	spi_crc_polynomial_set
函数原形	void spi_crc_polynomial_set(uint32_t spi_periph, uint16_t crc_poly);
功能描述	设置外设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SPI
SPIx (x=0,1)	SPI外设选择
输入参数{in}	
crc_poly	CRC多项式值
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* set SPI0 CRC polynomial */
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 3-714. 函数 spi_crc_polynomial_get

函数名称	spi_crc_polynomial_get
函数原形	uint16_t spi_crc_polynomial_get(uint32_t spi_periph);
功能描述	获取外设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SPI
SPIx (x=0,1)	SPI外设选择
输出参数{out}	
-	-
返回值	
uint16_t	16位CRC多项式值（0-0xFFFF）

例如：

```
/* get SPI0 CRC polynomial */
uint16_t crc_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 3-715. 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, uint16_t crc_length);
功能描述	设置CRC长度
先决条件	-
被调用函数	-
输入参数{in}	
spi_periph	外设SPI
SPIx (x=0)	SPI外设选择
输入参数{in}	
crc_length	crc长度
SPI_CRC_8BIT	CRC长度为8位数据
SPI_CRC_16BIT	CRC长度为16位数据
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/*set SPI0 CRC length 16 bits */
```

```
spi_crc_length_set(SPI0,SPI_CRC_16BIT);
```

函数 **spi_crc_on**

函数spi_crc_on描述见下表：

表 3-716. 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SPI
SPIx (x=0,1)	SPI外设选择
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* turn on SPI0 CRC function */
```

```
spi_crc_on(SPI0);
```

函数 spi_crc_off

函数spi_crc_off描述见下表:

表 3-717. 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SPI
SPIx (x=0,1)	SPI外设选择
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* turn off SPI0 CRC function */
```

```
spi_crc_off(SPI0);
```

函数 spi_crc_next

函数spi_crc_next描述见下表:

表 3-718. 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)	SPI外设选择
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* SPI0 next data is CRC value */
```

```
spi_crc_next(SPI0);
```

函数 spi_crc_get

函数spi_crc_get描述见下表:

表 3-719. 函数 spi_crc_get

函数名称	spi_crc_get
函数原形	uint16_t spi_crc_get(uint32_t spi_periph, uint8_t crc);
功能描述	外设SPI获取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)	SPI外设选择
输入参数{in}	
crc	SPI crc值
SPI_CRC_TX	获取发送CRC寄存器值
SPI_CRC_RX	获取接收CRC寄存器值
输出参数{out}	
-	-
返回值	
uint16_t	16位CRC值 (0-0xFFFF)

例如:

```
/* get SPI0 CRC send value */
```

```
uint16_t crc_val;
```

```
crc_val = spi_crc_get(SPI0, SPI_CRC_TX);
```

函数 spi_crc_error_clear

函数spi_crc_error_clear描述见下表:

表 3-720. 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
<i>SPIx</i> (<i>x=0,1</i>)	SPI外设选择
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* clear SPI0 CRC error flag status */
```

```
spi_crc_error_clear(SPI0);
```

函数 spi_ti_mode_enable

函数spi_ti_mode_enable描述见下表：

表 3-721. 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SPI
<i>SPIx</i> (<i>x=0,1</i>)	SPI外设选择
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 3-722. 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	失能SPI TI模式
先决条件	-
被调用函数	-

输入参数{in}	
spi_periph	外设SPIx
<i>SPIx (x=0,1)</i>	SPI外设选择
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 3-723. 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
<i>SPIx (x=0,1)</i>	SPI外设选择
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 3-724. 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
<i>SPIx</i> (<i>x</i> =0,1)	SPI外设选择
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 3-725. 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SPI
<i>SPIx</i> (<i>x</i> =0)	SPI外设选择
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 3-726. 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	失能四线SPI模式

先决条件	-
被调用函数	-
输入参数{in}	
spi_periph	外设SPI
SPIx (x=0)	SPI外设选择
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 3-727. 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SPI
SPIx (x=0)	SPI外设选择
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 3-728. 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SPI
SPIx (x=0)	SPI外设选择
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* enable SPI0 quad wire read */
```

```
spi_quad_read_enable(SPI0);
```

函数 spi_i2s_format_error_clear

函数spi_i2s_format_error_clear描述见下表：

表 3-729. 函数 spi_i2s_format_error_clear

函数名称	spi_i2s_format_error_clear
函数原形	void spi_i2s_format_error_clear(uint32_t spi_periph, uint32_t flag);
功能描述	清除SPI/I2S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
SPIx (x=0,1)	SPI外设选择
输入参数{in}	
flag	SPI/I2S格式错误标志
SPI_FLAG_FERR	SPI格式错误标志
I2S_FLAG_FERR	I2S错误标志
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* clear SPI format error flag status */
```

```
spi_i2s_format_error_clear(SPI0, SPI_FLAG_FERR);
```

函数 spi_i2s_flag_get

函数spi_i2s_flag_get描述见下表:

表 3-730. 函数 spi_i2s_flag_get

函数名称	spi_i2s_flag_get
函数原形	FlagStatus spi_i2s_flag_get(uint32_t spi_periph, uint32_t flag);
功能描述	获取外设SPI/I2S标志状态
先决条件	-
被调用函数	-
输入参数{in}	
spi_periph	外设SPI
SPIx (x=0,1)	SPI外设选择
输入参数{in}	
flag	SPI/I2S标志状态
SPI_FLAG_TBE	SPI发送缓冲区空标志
SPI_FLAG_RBNE	SPI接收缓冲区非空标志
SPI_FLAG_TRANS	SPI通信进行中标志
SPI_FLAG_RXORERR	SPI接收过载错误标志
SPI_FLAG_CONFERR	SPI配置错误标志
SPI_FLAG_CRCERR	SPI CRC错误标志
SPI_FLAG_FERR	SPI格式错误标志
I2S_FLAG_TBE	I2S发送缓冲区空标志
I2S_FLAG_RBNE	I2S接收缓冲区非空标志
I2S_FLAG_TRANS	I2S通信进行中标志
I2S_FLAG_RXORERR	I2S接收过载错误标志
I2S_FLAG_TXURERR	I2S发送欠载错误标志
I2S_FLAG_CH	I2S通道标志
I2S_FLAG_FERR	I2S格式错误标志
以下参数只适用于SPI0	
SPI_TXLVL_EMPTY	SPI TXFIFO空
SPI_TXLVL_QUARTER_FULL	SPI TXFIFO四分之一满
SPI_TXLVL_HALF_FULL	SPI TXFIFO半满
SPI_TXLVL_FULL	TXFIFO全满
SPI_RXLVL_EMPTY	SPI RXFIFO空

Y	
SPI_RXLVL_QUARTER_FULL	SPI RXFIFO四分之一满
SPI_RXLVL_HALF_FULL	SPI RXFIFO半满
SPI_RXLVL_FULL	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_i2s_flag_get(SPI0, SPI_FLAG_TBE);
```

函数 spi_i2s_interrupt_enable

函数spi_i2s_interrupt_enable描述见下表：

表 3-731. 函数 spi_i2s_interrupt_enable

函数名称	spi_i2s_interrupt_enable
函数原形	void spi_i2s_interrupt_enable(uint32_t spi_periph, uint8_t interrupt);
功能描述	使能外设SPI/I2S中断
先决条件	-
被调用函数	-
输入参数{in}	
spi_periph	外设SPI
SPIx (x=0,1)	SPI外设选择
输入参数{in}	
interrupt	SPI/I2S中断
SPI_I2S_INT_TBE	发送缓冲区空中断
SPI_I2S_INT_RBNE	接收缓冲区非空中断
SPI_I2S_INT_ERR	错误中断使能
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* enable SPI0 transmit buffer empty interrupt */
```

```
spi_i2s_interrupt_enable(SPI0, SPI_I2S_INT_TBE);
```


函数 **spi_i2s_interrupt_disable**

函数spi_i2s_interrupt_disable描述见下表:

表 3-732. 函数 **spi_i2s_interrupt_disable**

函数名称	spi_i2s_interrupt_disable
函数原形	void spi_i2s_interrupt_disable(uint32_t spi_periph, uint8_t interrupt);
功能描述	失能外设SPI/I2S中断
先决条件	-
被调用函数	-
输入参数{in}	
spi_periph	外设SPI
SPIx (x=0,1)	SPI外设选择
输入参数{in}	
interrupt	SPI/I2S中断
SPI_I2S_INT_TBE	发送缓冲区空中断
SPI_I2S_INT_RBNE	接收缓冲区非空中断
SPI_I2S_INT_ERR	错误中断使能
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* disable SPI0 transmit buffer empty interrupt */
```

```
spi_i2s_interrupt_disable(SPI0, SPI_I2S_INT_TBE);
```

函数 **spi_i2s_interrupt_flag_get**

函数spi_i2s_interrupt_flag_get描述见下表:

表 3-733. 函数 **spi_i2s_interrupt_flag_get**

函数名称	spi_i2s_interrupt_flag_get
函数原形	FlagStatus spi_i2s_interrupt_flag_get(uint32_t spi_periph, uint8_t interrupt);
功能描述	获取外设SPI/I2S中断状态
先决条件	-
被调用函数	-
输入参数{in}	
spi_periph	外设SPI
SPIx	x=0,1
输入参数{in}	
interrupt	SPI/I2S中断状态
SPI_I2S_INT_FLAG_TBE	发送缓冲区空中断

<code>SPI_I2S_INT_FLAG_RBNE</code>	接收缓冲区非空中断
<code>SPI_I2S_INT_FLAG_RXOERR</code>	接收过载错误中断
<code>SPI_INT_FLAG_CONFERR</code>	配置错误中断
<code>SPI_INT_FLAG_CRCERR</code>	CRC错误中断
<code>I2S_INT_FLAG_TXURERR</code>	发送欠载错误中断
<code>SPI_I2S_INT_FLAG_FERR</code>	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_i2s_interrupt_flag_get(SPI0, SPI_I2S_INT_FLAG_TBE)){
    while(RESET == spi_i2s_flag_get(SPI0, SPI_FLAG_TBE));
    spi_i2s_data_transmit(SPI0, spi0_send_array[send_n++]);
}
```

3.24. SYSCFG

章节 [3.24.1](#) 描述了 SYSCFG 的寄存器列表，章节 [3.24.2](#) 对 SYSCFG 库函数进行说明。

3.24.1. 外设寄存器说明

SYSCFG 寄存器列表如下表所示：

表 3-734. SYSCFG 寄存器

寄存器名称	寄存器描述
SYSCFG_CFG0	配置寄存器0
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_CFG1	配置寄存器1（仅适用于GD32L235）
SYSCFG_CPU_IRQ	IRQ延迟寄存器

寄存器名称	寄存器描述
_LAT	
SYSCFG_TIMERxC FG	TIMERx配置寄存器，x=0,1,2,8,11,14,40（仅适用于GD32L235）

3.24.2. 外设库函数说明

SYSCFG库函数列表如下表所示：

表 3-735. SYSCFG 库函数

库函数名称	库函数描述
syscfg_deinit	复位SYSCFG寄存器
syscfg_exti_line_config	配置GPIO引脚作为EXTI
syscfg_pin_remap_enable	对于小封装芯片，使能引脚重映设功能
syscfg_pin_remap_disable	对于小封装芯片，失能引脚重映设功能
syscfg_high_current_enable	使能PBx(x=6,7,8,9)引脚大电流能力
syscfg_high_current_disable	失能PBx(x=6,7,8,9)引脚大电流能力
irq_latency_set	设置延迟值
syscfg_bootmode_get	获取BOOT启动方式
syscfg_sram_waitstate_insert	当LDO为1.1V时对SRAM0/1的读访问期间插入等待周期（仅适用于GD32L235）
syscfg_sram_waitstate_cancel	当LDO为1.1V时取消对SRAM0/1的读访问期间插入的等待周期（仅适用于GD32L235）
syscfg_lock_config	将TIMER0/14/40的break输入连接到指定参数（仅适用于GD32L235）
syscfg_flag_get	获取SYSCFG SYSCFG_CFG1寄存器里的标志（仅适用于GD32L235）
syscfg_flag_clear	清除SYSCFG SYSCFG_CFG1寄存器里的标志（仅适用于GD32L235）

函数 syscfg_deinit

函数syscfg_deinit描述见下表：

表 3-736. 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_exti_line_config

函数syscfg_exti_line_config描述见下表:

表 3-737. 函数 syscfg_exti_line_config

函数名称	syscfg_exti_line_config
函数原形	void syscfg_exti_line_config(uint8_t exti_port, uint8_t exti_pin);
功能描述	配置GPIO引脚作为EXTI
先决条件	-
被调用函数	-
输入参数{in}	
exti_port	指定EXTI使用的GPIO端口
EXTI_SOURCE_GPIOx	x=A,B,C,D,F
输入参数{in}	
exti_pin	EXTI引脚
EXTI_SOURCE_PINx	GPIOAx = 0..15, GPIOBx = 0..15, GPIOCx = 0..15, GPIODx = 0..6,8,9, GPIOFx = 0,1
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* configure the GPIO pin as EXTI Line */
```

```
syscfg_exti_line_config(EXTI_SOURCE_GPIOA, EXTI_SOURCE_PIN0);
```

函数 syscfg_pin_remap_enable

函数syscfg_pin_remap_enable描述见下表:

表 3-738. 函数 syscfg_pin_remap_enable

函数名称	syscfg_pin_remap_enable
函数原形	void syscfg_pin_remap_enable(uint32_t remap_pin);
功能描述	对于小封装芯片，使能引脚重映射功能
先决条件	-

被调用函数	-
输入参数{in}	
remap_pin	引脚重映设
SYSCFG_PA11_PA12_REMAP	重新映射PA11 PA12
SYSCFG_BOOT0_PD3_REMAP	重新映射BOOT0 PD3
SYSCFG_PA8_REMAP	重新映射PA8（仅适用于GD32L235）
SYSCFG_PD0_PD1_PD2_REMAP	重新映射PD0 PD1 PD2（仅适用于GD32L235）
SYSCFG_PA11_PA12_PB6_PB8_REMAP	重新映射PA11 PA12 PB6 PB8（仅适用于GD32L235）
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* enable BOOT0 remap to PD3 function */
```

```
syscfg_pin_remap_enable(SYSCFG_BOOT0_PD3_REMAP);
```

函数 syscfg_pin_remap_disable

函数syscfg_pin_remap_disable描述见下表：

表 3-739. 函数 syscfg_pin_remap_disable

函数名称	syscfg_pin_remap_disable
函数原形	void syscfg_pin_remap_disable(uint32_t remap_pin);
功能描述	对于小封装芯片，失能引脚重映设功能
先决条件	-
被调用函数	-
输入参数{in}	
remap_pin	引脚重映设
SYSCFG_PA11_PA12_REMAP	重新映射PA11 PA12
SYSCFG_BOOT0_PD3_REMAP	重新映射BOOT0 PD3
SYSCFG_PA8_REMAP	重新映射PA8（仅适用于GD32L235）
SYSCFG_PD0_PD1_PD2_REMAP	重新映射PD0 PD1 PD2（仅适用于GD32L235）

SYSCFG_PA11_PA12_PB6_PB8_REMAP	重新映射PA11 PA12 PB6 PB8（仅适用于GD32L235）
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* disable BOOT0 remap to PD3 function */
syscfg_pin_remap_disable(SYSCFG_BOOT0_PD3_REMAP);
```

函数 syscfg_high_current_enable

函数syscfg_high_current_enable描述见下表：

表 3-740. 函数 syscfg_high_current_enable

函数名称	syscfg_high_current_enable
函数原形	void syscfg_high_current_enable(uint32_t syscfg_gpio);
功能描述	使能PBx(x=6,7,8,9)引脚大电流能力
先决条件	-
被调用函数	-
输入参数{in}	
syscfg_gpio	SYSCFG GPIO
SYSCFG_PB6_HIGH_CURRENT	PB6引脚大电流能力
SYSCFG_PB7_HIGH_CURRENT	PB7引脚大电流能力
SYSCFG_PB8_HIGH_CURRENT	PB8引脚大电流能力
SYSCFG_PB9_HIGH_CURRENT	PB9引脚大电流能力
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* enable PB9 high current capability */
syscfg_high_current_enable(SYSCFG_PB9_HIGH_CURRENT);
```

函数 **syscfg_high_current_disable**

函数syscfg_high_current_disable描述见下表:

表 3-741. 函数 **syscfg_high_current_disable**

函数名称	syscfg_high_current_disable
函数原形	void syscfg_high_current_disable(uint32_t syscfg_gpio);
功能描述	失能PBx(x=6,7,8,9)引脚大电流能力
先决条件	-
被调用函数	-
输入参数{in}	
syscfg_gpio	SYSCFG GPIO
SYSCFG_PB6_HIGH_CURRENT	PB6引脚大电流能力
SYSCFG_PB7_HIGH_CURRENT	PB7引脚大电流能力
SYSCFG_PB8_HIGH_CURRENT	PB8引脚大电流能力
SYSCFG_PB9_HIGH_CURRENT	PB9引脚大电流能力
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* disable PB9 high current capability */
```

```
syscfg_high_current_disable(SYSCFG_PB9_HIGH_CURRENT);
```

函数 **irq_latency_set**

函数irq_latency_set描述见下表:

表 3-742. 函数 **irq_latency_set**

函数名称	irq_latency_set
函数原形	void irq_latency_set(uint8_t irq_latency);
功能描述	设置延迟时间值
先决条件	-
被调用函数	-
输入参数{in}	
irq_latency	延迟时间值
0x00-0xFF	延迟时间值
输出参数{out}	
-	-

返回值	
-	-

例如：

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

```
Irq_latency_set(0xFF);
```

函数 syscfg_bootmode_get

函数syscfg_bootmode_get描述见下表：

表 3-743. 函数 syscfg_bootmode_get

函数名称	syscfg_bootmode_get
函数原形	uint8_t syscfg_bootmode_get(void);
功能描述	获取BOOT启动方式
先决条件	-
被调用函数	-
输入参数{in}	
-	-
输出参数{out}	
-	-
返回值	
uint8_t	启动方式
SYSCFG_BOOTMODE_FLASH	从片上闪存的主存引导启动
SYSCFG_BOOTMODE_SYSTEM	从片上闪存的系统存储器引导启动
SYSCFG_BOOTMODE_SRAM	从片上SRAM引导启动

例如：

```
/* get the current boot mode */
```

```
uint8_t boot_mode;
```

```
boot_mode = syscfg_bootmode_get();
```

函数 syscfg_sram_waitstate_insert

函数syscfg_sram_waitstate_insert描述见下表：

表 3-744. 函数 syscfg_sram_waitstate_insert

函数名称	syscfg_sram_waitstate_insert
函数原形	void syscfg_sram_waitstate_insert(void);
功能描述	当LDO为1.1V时对SRAM0/1的读访问期间插入等待周期（仅适用于

	GD32L235)
先决条件	-
被调用函数	-
输入参数{in}	
-	-
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* insert wait state in read accesses of SRAM0/SRAM1 */
```

```
syscfg_sram_waitstate_insert();
```

函数 syscfg_sram_waitstate_cancel

函数syscfg_sram_waitstate_cancel描述见下表：

表 3-745. 函数 syscfg_sram_waitstate_cancel

函数名称	syscfg_sram_waitstate_cancel
函数原形	void syscfg_sram_waitstate_cancel(void);
功能描述	当LDO为1.1V时取消对SRAM0/1的读访问期间插入的等待周期（仅适用于GD32L235）
先决条件	-
被调用函数	-
输入参数{in}	
-	-
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* cancel insertion of wait state in read accesses of SRAM0/SRAM1 */
```

```
syscfg_sram_waitstate_cancel();
```

函数 syscfg_lock_config

函数syscfg_lock_config描述见下表：

表 3-746. 函数 syscfg_lock_config

函数名称	syscfg_lock_config
函数原形	void syscfg_lock_config(uint32_t syscfg_lock);

功能描述	将TIMER0/14/40的break输入连接到指定参数（仅适用于GD32L235）
先决条件	-
被调用函数	-
输入参数{in}	
syscfg_lock	SYSCFG锁定
SYSCFG_LOCK_LOCKUP	Cortex-M23 LOCKUP 输出与TIMER0/14/40的break输入端连接
SYSCFG_LOCK_SRAM_PARITY_ERROR	SRAM奇偶校验错误与TIMER0/14/40的break输入端连接
SYSCFG_LOCK_LVD	LVD中断与TIMER0/14/40的break输入端连接
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* connect TIMER0/14/40 break input to the SYSCFG_LOCK_LOCKUP */
syscfg_lock_config(SYSCFG_LOCK_LOCKUP);
```

函数 syscfg_flag_get

函数syscfg_flag_get描述见下表：

表 3-747. 函数 syscfg_flag_get

函数名称	syscfg_flag_get
函数原形	FlagStatus syscfg_flag_get(uint32_t syscfg_flag);
功能描述	获取SYSCFG SYSCFG_CFG1寄存器里的标志（仅适用于GD32L235）
先决条件	-
被调用函数	-
输入参数{in}	
syscfg_flag	SYSCFG_CFG1寄存器里标志
SYSCFG_FLAG_SRAM_PCEF	SRAM奇偶校验失败标志
输出参数{out}	
-	-
返回值	
FlagStatus	SET或RESET

例如：

```
FlagStatus tempflag;

/* get the SYSCFG flag */
```

```
tempflag = syscfg_flag_get(SYSCFG_FLAG_SRAM_PCEF);
```

函数 syscfg_flag_clear

函数syscfg_flag_clear描述见下表:

表 3-748. 函数 syscfg_flag_clear

函数名称	syscfg_flag_clear
函数原形	void syscfg_flag_clear(uint32_t syscfg_flag);
功能描述	清除SYSCFG SYSCFG_CFG1寄存器里的标志（仅适用于GD32L235）
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 the SYSCFG flag */
```

```
syscfg_flag_clear(SYSCFG_FLAG_SRAM_PCEF);
```

3.25. TIMER

定时器含有可编程的一个无符号计数器，支持输入捕获和输出比较，分为五种类型：高级定时器（TIMER0），通用定时器L0（TIMER1，TIMER2），通用定时器L1（TIMER8，TIMER11），通用定时器L3（TIMER14，TIMER40），基本定时器（TIMER5，TIMER6），不同类型的定时器具体功能有所差别。章节[3.25.1](#)描述了TIMER的寄存器列表，章节[3.25.2](#)对TIMER库函数进行说明。

3.25.1. 外设寄存器说明

TIMER寄存器列表如下表所示:

表 3-749. TIMER 寄存器

寄存器名称	寄存器描述
TIMER_CTL0（timerx, x=0, 1, 2, 5, 6, 8, 11, 14, 40）	控制寄存器0
TIMERx_CTL1（timerx, x=0,1, 2, 5, 6, 14, 40）	控制寄存器1
TIMERx_SMCFG（timerx, x=0, 1, 2, 8, 11, 14, 40）	从模式配置寄存器

寄存器名称	寄存器描述
TIMERx_DMAINTEN (timerx, x=0,1, 2, 5, 6, 8, 11, 14, 40)	DMA和中断使能寄存器
TIMERx_INTF (timerx, x=0,1, 2, 5, 6, 8, 11, 14, 40)	中断标志寄存器
TIMERx_SWEVG (timerx, x=0,1, 2, 5, 6, 8, 11, 14, 40)	软件事件产生寄存器
TIMERx_CHCTL0 (timerx, x=0, 1, 2, 8, 11 14, 40)	通道控制寄存器0
TIMERx_CHCTL1 (timerx, x=0, 1, 2)	通道控制寄存器1
TIMERx_CHCTL2 (timerx, x=0, 1, 2, 8, 11. 14, 40)	通道控制寄存器2
TIMERx_CNT (timerx, x=0, 1, 2, 5, 6, 8, 11 14, 40)	计数器寄存器
TIMERx_PSC (timerx, x=0, 1, 2, 5, 6, 8, 11 14, 40)	预分频寄存器
TIMERx_CAR (timerx, x=0, 1, 2, 5, 6, 8, 11 14, 40)	计数器自动重载寄存器
TIMERx_CREP (timerx, x=0, 14, 40)	重复计数寄存器
TIMERx_CH0CV (timerx, x=0, 1, 2, 8, 11 14, 40)	通道0捕获/比较寄存器
TIMERx_CH1CV (timerx, x=0, 1, 2, 8, 11 14, 40)	通道1捕获/比较寄存器
TIMERx_CH2CV (timerx, x=0, 1, 2)	通道2捕获/比较寄存器
TIMERx_CH3CV (timerx, x=0, 1, 2)	通道3捕获/比较寄存器
TIMERx_IRMP (timerx, x= 8, 11)	通道输入重映射寄存器
TIMERx_CCHP (timerx, x=0, 14, 40)	互补通道保护寄存器
TIMERx_DMACFG (timerx, x=0, 1, 2, 14 ,40)	DMA配置寄存器
TIMERx_DMATB (timerx, x=0, 1, 2, 14, 40)	DMA发送缓冲区寄存器
TIMERx_CFG (timerx, x=0, 1, 2, 8, 11, 14, 40)	配置寄存器

3.25.2. 外设库函数说明

TIMER库函数列表如下表所示：

表 3-750. TIMER 库函数

库函数名称	库函数描述
timer_deinit	复位外设TIMERx
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TIMERx
timer_enable	使能外设TIMERx
timer_disable	禁能外设TIMERx

库函数名称	库函数描述
timer_auto_reload_shadow_enable	TIMERx自动重载影子使能
timer_auto_reload_shadow_disable	TIMERx自动重载影子禁能
timer_update_event_enable	TIMERx更新事件使能
timer_update_event_disable	TIMERx更新事件禁能
timer_counter_alignment	设置外设TIMERx的对齐模式
timer_counter_up_direction	设置外设TIMERx向上计数
timer_counter_down_direction	设置外设TIMERx向下计数
timer_prescaler_config	配置外设TIMERx预分频器
timer_repetition_value_config	配置外设TIMERx的重复计数器
timer_autoreload_value_config	配置外设TIMERx的自动重载寄存器
timer_counter_value_config	配置外设TIMERx的计数器值
timer_counter_read	读取外设TIMERx的计数器值
timer_prescaler_read	读取外设TIMERx的预分频器值
timer_single_pulse_mode_config	配置外设TIMERx的单脉冲模式
timer_update_source_config	配置外设TIMERx的更新源
timer_dma_enable	使能TIMERx的DMA功能
timer_dma_disable	禁能TIMERx的DMA功能
timer_channel_dma_request_source_select	外设TIMERx的通道DMA请求源选择
timer_dma_transfer_config	配置外设TIMERx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TIMERx的中止功能
timer_break_disable	禁能TIMERx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	通道换相控制影子寄存器配置
timer_channel_control_shadow_update_config	通道换相控制影子寄存器更新控制
timer_channel_output_struct_para_init	将TIMER通道输出参数结构体中所有参数初始化为默认值
timer_channel_output_config	外设TIMERx的通道输出配置
timer_channel_output_mode_config	配置外设TIMERx通道输出比较模式
timer_channel_output_pulse_value_config	配置外设TIMERx的通道输出比较值
timer_channel_output_shadow_config	配置TIMERx通道输出比较影子寄存器功能
timer_channel_output_fast_config	配置TIMERx通道输出比较快速功能

库函数名称	库函数描述
timer_channel_output_clear_config	配置TIMERx的通道输出比较清0功能
timer_channel_output_polarity_config	通道输出极性配置
timer_channel_complementary_output_polarity_config	互补通道输出极性配置
timer_channel_output_state_config	配置通道状态
timer_channel_complementary_output_state_config	配置互补通道输出状态
timer_channel_input_struct_para_init	将TIMER通道输入参数结构体中所有参数初始化为默认值
timer_input_capture_config	配置TIMERx输入捕获参数
timer_channel_input_capture_prescaler_config	配置TIMERx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TIMERx捕获PWM输入参数
timer_hall_mode_config	配置TIMERx的HALL接口功能
timer_input_trigger_source_select	TIMERx的输入触发源选择
timer_master_output_trigger_source_select	选择TIMERx主模式输出触发源
timer_slave_mode_select	TIMERx从模式配置
timer_master_slave_mode_config	TIMERx主从模式配置
timer_external_trigger_config	配置TIMERx外部触发输入
timer_quadrature_decoder_mode_config	TIMERx配置为编码器模式
timer_internal_trigger_as_external_clock_config	配置TIMERx的内部触发为时钟源
timer_external_trigger_as_external_clock_config	配置TIMERx的外部触发作为时钟源
timer_external_clock_mode0_config	配置TIMERx外部时钟模式0，ETI作为时钟源
timer_external_clock_mode1_config	配置TIMERx外部时钟模式1
timer_external_clock_mode1_disable	禁能TIMERx外部时钟模式1
timer_channel_remap_config	配置TIMERx通道重映射功能
timer_write_chxval_register_config	配置TIMERx写CHxVAL选择位
timer_flag_get	外设TIMERx标志位获取
timer_flag_clear	外设TIMERx标志位清除
timer_interrupt_enable	外设TIMERx的中断使能
timer_interrupt_disable	外设TIMERx的中断禁能
timer_interrupt_flag_get	外设TIMERx中断标志获取
timer_interrupt_flag_clear	外设TIMERx中断标志清除

结构体 timer_parameter_struct

表 3-751. 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）
clockdivision	时钟分频因子（TIMER_CKDIV_DIV1, TIMER_CKDIV_DIV2, TIMER_CKDIV_DIV4）
period	周期（0~65535）
repetitioncounter	重复计数器值（0~255） 只对GD32L235有效

结构体 timer_break_parameter_struct （只对 GD32L235 有效）

表 3-752. 结构体 timer_break_parameter_struct

成员名称	功能描述
runoffstate	运行模式下“关闭状态”配置（TIMER_ROS_STATE_ENABLE, TIMER_ROS_STATE_DISABLE）
ideloffstate	空闲模式下“关闭状态”配置（TIMER_IOS_STATE_ENABLE, TIMER_IOS_STATE_DISABLE）
deadtime	死区时间（0~255）
breakpolarity	中止信号极性（TIMER_BREAK_POLARITY_LOW, TIMER_BREAK_POLARITY_HIGH）
outputautostate	自动输出使能（TIMER_OUTAUTO_ENABLE, TIMER_OUTAUTO_DISABLE）
protectmode	互补寄存器保护控制（TIMER_CCHP_PROT_OFF, TIMER_CCHP_PROT_0, TIMER_CCHP_PROT_1, TIMER_CCHP_PROT_2）
breakstate	中止使能（TIMER_BREAK_ENABLE, TIMER_BREAK_DISABLE）

结构体 timer_oc_parameter_struct

表 3-753. 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） 只对GD32L235有效
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） 只对GD32L235有效
ocidlestate	空闲状态下通道输出（TIMER_OC_IDLE_STATE_LOW, TIMER_OC_IDLE_STATE_HIGH）

成员名称	功能描述
	TIMER_OC_IDLE_STATE_HIGH) 只对GD32L235有效
ocnidlestate	空闲状态下互补通道输出 (TIMER_OCN_IDLE_STATE_LOW, TIMER_OCN_IDLE_STATE_HIGH) 只对GD32L235有效

结构体 timer_ic_parameter_struct

表 3-754. 结构体 timer_ic_parameter_struct

成员名称	功能描述
icpolarity	通道输入极性 (TIMER_IC_POLARITY_RISING, TIMER_IC_POLARITY_FALLING, TIMER_IC_POLARITY_BOTH_EDGE)
icselection	通道输入模式选择 (TIMER_IC_SELECTION_DIRECTTI, TIMER_IC_SELECTION_INDIRECTTI, TIMER_IC_SELECTION_ITS)
icprescaler	通道输入捕获预分频 (TIMER_IC_PSC_DIV1, TIMER_IC_PSC_DIV2, TIMER_IC_PSC_DIV4, TIMER_IC_PSC_DIV8)
icfilter	通道输入捕获滤波 (0~15)

函数 timer_deinit

函数timer_deinit描述见下表:

表 3-755. 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TIMERx
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	TIMER外设选择 x=1,2,5,6,8,11 for GD32L233, x=0,1,2,5,6,8,11,14,40 for GD32L235)
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* reset TIMER1 */
timer_deinit(TIMER1);
```


函数 timer_struct_para_init

函数timer_struct_para_init描述见下表:

表 3-756. 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 表3-751. 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 3-757. 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TIMERx
先决条件	-
被调用函数	-
输入参数{in}	
timer_periph	TIMER外设
TIMERx	TIMER外设选择 (x=1,2,5,6,8,11 for GD32L233, x=0,1,2,5,6,8,11,14,40 for GD32L235)
输入参数{in}	
initpara	TIMER初始化结构体, 结构体成员参考 表3-751. 结构体 timer_parameter_struct
输出参数{out}	
-	-
返回值	

-	-
---	---

例如:

```
/* initialize TIMER1 */

timer_parameter_struct timer_initpara;

timer_initpara.prescaler      = 63;

timer_initpara.alignedmode    = TIMER_COUNTER_EDGE;

timer_initpara.counterdirection = TIMER_COUNTER_UP;

timer_initpara.period         = 999;

timer_initpara.clockdivision   = TIMER_CKDIV_DIV1;

timer_init(TIMER1, &timer_initpara);
```

函数 timer_enable

函数timer_enable描述见下表:

表 3-758. 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TIMERx
先决条件	-
被调用函数	-
输入参数{in}	
timer_periph	TIMER外设
TIMERx	TIMER外设选择 (x=1,2,5,6,8,11 for GD32L233, x=0,1,2,5,6,8,11,14,40 for GD32L235)
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* enable TIMER1 */

timer_enable(TIMER1);
```

函数 timer_disable

函数timer_disable描述见下表:

表 3-759. 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TIMERx
先决条件	-
被调用函数	-
输入参数{in}	
timer_periph	TIMER外设
TIMERx	TIMER外设选择 (x=1,2,5,6,8,11 for GD32L233, x=0,1,2,5,6,8,11,14,40 for GD32L235)
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* disable TIMER1 */
```

```
timer_disable(TIMER1);
```

函数 timer_auto_reload_shadow_enable

函数timer_auto_reload_shadow_enable描述见下表:

表 3-760. 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	TIMERx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	TIMER外设选择 (x=1,2,5,6,8,11 for GD32L233, x=0,1,2,5,6,8,11,14,40 for GD32L235)
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* enable the TIMER1 auto reload shadow function */
```

```
timer_auto_reload_shadow_enable(TIMER1);
```

函数 timer_auto_reload_shadow_disable

函数timer_auto_reload_shadow_disable描述见下表:

表 3-761. 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	TIMERx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	TIMER外设选择 (x=1,2,5,6,8,11 for GD32L233, x=0,1,2,5,6,8,11,14,40 for GD32L235)
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* disable the TIMER1 auto reload shadow function */
```

```
timer_auto_reload_shadow_disable(TIMER1);
```

函数 timer_update_event_enable

函数timer_update_event_enable描述见下表:

表 3-762. 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	TIMERx更新事件使能
先决条件	-
被调用函数	-
输入参数{in}	
timer_periph	TIMER外设
TIMERx	TIMER外设选择 (x=1,2,5,6,8,11 for GD32L233, x=0,1,2,5,6,8,11,14,40 for GD32L235)
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* enable TIMER1 the update event */
```

```
timer_update_event_enable(TIMER1);
```

函数 timer_update_event_disable

函数timer_update_event_disable描述见下表：

表 3-763. 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	TIMERx更新事件禁能
先决条件	-
被调用函数	-
输入参数{in}	
timer_periph	TIMER外设
TIMERx	TIMER外设选择 (x=1,2,5,6,8,11 for GD32L233, x=0,1,2,5,6,8,11,14,40 for GD32L235)
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* disable TIMER1 the update event */
```

```
timer_update_event_disable(TIMER1);
```

函数 timer_counter_alignment

函数timer_counter_alignment描述见下表：

表 3-764. 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TIMERx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	TIMER外设选择 (x=1,2 for GD32L233, x=0,1,2 for GD32L235)
输入参数{in}	
aligned	对齐模式
TIMER_COUNTER_EDGE	无中央对齐计数模式（边沿对齐模式），DIR位指定了计数方向
TIMER_COUNTER_CENTER_DOWN	中央对齐向下计数置1模式。计数器在中央计数模式计数，通道被配置在输出模式（TIMERx_CHCTL0寄存器中CHxMS=00），只有在向下计数时，通道的比较中断标志置1

<i>TIMER_COUNTER_CENTER_UP</i>	中央对齐向上计数置1模式。计数器在中央计数模式计数，通道被配置在输出模式（ <i>TIMERx_CHCTL0</i> 寄存器中 <i>CHxMS=00</i> ），只有在向上计数时，通道的比较中断标志置1
<i>TIMER_COUNTER_CENTER_BOTH</i>	中央对齐上下计数置1模式。计数器在中央计数模式计数，通道被配置在输出模式（ <i>TIMERx_CHCTL0</i> 寄存器中 <i>CHxMS=00</i> ），在向上和向下计数时，通道的比较中断标志都会置1
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* set TIMER1 counter center-aligned and counting up assert mode */
timer_counter_alignment(TIMER1, TIMER_COUNTER_CENTER_UP);
```

函数 timer_counter_up_direction

函数timer_counter_up_direction描述见下表：

表 3-765. 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 <i>TIMERx</i> 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	<i>TIMER</i> 外设
<i>TIMERx</i>	<i>TIMER</i> 外设选择 (<i>x</i> =1,2 for GD32L233, <i>x</i> =0,1,2 for GD32L235)
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* set TIMER1 counter up direction */
timer_counter_up_direction(TIMER1);
```

函数 timer_counter_down_direction

函数timer_counter_down_direction描述见下表：

表 3-766. 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TIMERx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	TIMER外设选择 (x=1,2 for GD32L233, x=0,1,2 for GD32L235)
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* set TIMER1 counter down direction */
timer_counter_down_direction(TIMER1);
```

函数 timer_prescaler_config

函数timer_prescaler_config描述见下表：

表 3-767. 函数 timer_prescaler_config

函数名称	timer_prescaler_config
函数原型	void timer_prescaler_config(uint32_t timer_periph, uint16_t prescaler, uint8_t pscreload);
功能描述	配置外设TIMERx预分频器
先决条件	-
被调用函数	-
输入参数{in}	
timer_periph	TIMER外设
TIMERx	TIMER外设选择 (x=1,2,5,6,8,11 for GD32L233, x=0,1,2,5,6,8,11,14,40 for GD32L235)
输入参数{in}	
prescaler	预分频值，0~65535
输入参数{in}	
pscreload	预分频值加载模式
TIMER_PSC_RELO AD_NOW	预分频值立即加载
TIMER_PSC_RELO AD_UPDATE	预分频值在下次更新事件发生时加载
输出参数{out}	
-	-
返回值	

-	-
---	---

例如:

```
/* configure TIMER1 prescaler */
```

```
timer_prescaler_config(TIMER1, 3000, TIMER_PSC_RELOAD_NOW);
```

函数 timer_repetition_value_config

函数timer_repetition_value_config描述见下表:

表 3-768. 函数 timer_repetition_value_config

函数名称	timer_repetition_value_config
函数原型	void timer_repetition_value_config(uint32_t timer_periph, uint16_t repetition);
功能描述	配置外设TIMERx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	TIMER外设选择 (x=0,14,40 for GD32L235)
输入参数{in}	
repetition	重复计数器值, 取值范围0~255
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* configure TIMER0 repetition register value */
```

```
timer_repetition_value_config (TIMER0, 98);
```

函数 timer_autoreload_value_config

函数timer_autoreload_value_config描述见下表:

表 3-769. 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, uint16_t autoreload);
功能描述	配置外设TIMERx的自动重载寄存器
先决条件	-
被调用函数	-
输入参数{in}	
timer_periph	TIMER外设

<i>TIMERx</i>	TIMER外设选择 (x=1,2,5,6,8,11 for GD32L233, x=0,1,2,5,6,8,11,14,40 for GD32L235)
输入参数{in}	
autoreload	计数器自动重载值 (0-65535)
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* configure TIMER1 autoreload register value */
timer_autoreload_value_config(TIMER1, 3000);
```

函数 timer_counter_value_config

函数timer_counter_value_config描述见下表:

表 3-770. 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, uint16_t counter);
功能描述	配置外设TIMERx的计数器值
先决条件	-
被调用函数	-
输入参数{in}	
timer_periph	TIMER外设
<i>TIMERx</i>	TIMER外设选择 (x=1,2,5,6,8,11 for GD32L233, x=0,1,2,5,6,8,11,14,40 for GD32L235)
输入参数{in}	
counter	计数器值 (0-65535)
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* configure TIMER1 counter register value */
timer_counter_value_config(TIMER1, 3000);
```

函数 timer_counter_read

函数timer_counter_read描述见下表:

表 3-771. 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TIMERx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	TIMER外设选择 (x=1,2,5,6,8,11 for GD32L233, x=0,1,2,5,6,8,11,14,40 for GD32L235)
输出参数{out}	
-	-
返回值	
uint32_t	外设TIMERx的计数器值 (0~65535)

例如:

```
/* read TIMER1 counter value */
uint32_t i = 0;
i = timer_counter_read(TIMER1);
```

函数 timer_prescaler_read

函数timer_prescaler_read描述见下表:

表 3-772. 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TIMERx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	TIMER外设选择 (x=1,2,5,6,8,11 for GD32L233, x=0,1,2,5,6,8,11,14,40 for GD32L235)
输出参数{out}	
-	-
返回值	
uint16_t	外设TIMERx的预分频器值 (0~65535)

例如:

```
/* read TIMER1 prescaler value */
uint16_t i = 0;
```

```
i = timer_prescaler_read(TIMER1);
```

函数 timer_single_pulse_mode_config

函数timer_single_pulse_mode_config描述见下表：

表 3-773. 函数 timer_single_pulse_mode_config

函数名称	timer_single_pulse_mode_config
函数原型	void timer_single_pulse_mode_config(uint32_t timer_periph, uint8_t spmode);
功能描述	配置外设TIMERx的单脉冲模式
先决条件	-
被调用函数	-
输入参数{in}	
timer_periph	TIMER外设
TIMERx	TIMER外设选择 (x=1,2,8,11 for GD32L233, x=0,1,2,8,11,14,40 for GD32L235)
输入参数{in}	
spmode	脉冲模式
TIMER_SP_MODE_SINGLE	单脉冲模式计数
TIMER_SP_MODE_REPETITIVE	重复模式计数
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* configure TIMER1 single pulse mode */
timer_single_pulse_mode_config(TIMER1, TIMER_SP_MODE_SINGLE);
```

函数 timer_update_source_config

函数timer_update_source_config描述见下表：

表 3-774. 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TIMERx的更新源
先决条件	-
被调用函数	-
输入参数{in}	
timer_periph	TIMER外设
TIMERx	TIMER外设选择

	(x=1,2,5,6,8,11 for GD32L233, x=0,1,2,5,6,8,11,14,40 for GD32L235)
输入参数{in}	
update	更新源
TIMER_UPDATE_SRC_GLOBAL	下述任一事件产生更新中断或DMA请求: – UPG位被置1 – 计数器溢出/下溢 – 从模式控制器产生的更新
TIMER_UPDATE_SRC_REGULAR	只有计数器溢出/下溢才产生更新中断或DMA请求
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* configure TIMER1 update only by counter overflow/underflow */
timer_update_source_config(TIMER1, TIMER_UPDATE_SRC_REGULAR);
```

函数 timer_dma_enable

函数timer_dma_enable描述见下表:

表 3-775. 函数 timer_dma_enable

函数名称	timer_dma_enable
函数原型	void timer_dma_enable(uint32_t timer_periph, uint16_t dma);
功能描述	外设TIMERx的DMA使能
先决条件	-
被调用函数	-
输入参数{in}	
timer_periph	TIMER外设
TIMERx	参考具体参数
输入参数{in}	
dma	DMA源
TIMER_DMA_UPD	更新DMA请求 TIMERx (x=1,2,5,6,8,11 for GD32L233, x=0,1,2,5,6,8,11,14,40 for GD32L235)
TIMER_DMA_CH0_D	通道0比较/捕获DMA请求 TIMERx (x=1,2,8,11 for GD32L233, x=0,1,2,8,11,14,40 for GD32L235)
TIMER_DMA_CH1_D	通道1比较/捕获 DMA请求 TIMERx (x=1,2,8,11 for GD32L233, x=0,1,2,8,11,14,40 for GD32L235)
TIMER_DMA_CH2_D	通道2比较/捕获DMA请求 TIMERx (x=1,2 for GD32L233, x=0,1,2 for GD32L235)
TIMER_DMA_CH3	通道3比较/捕获DMA请求

<i>D</i>	TIMERx (x=1,2 for GD32L233, x=0,1,2 for GD32L235)
<i>TIMER_DMA_CMT</i> <i>D</i>	换相DMA更新请求 TIMERx (x=0,14,40 for GD32L235)
<i>TIMER_DMA_TRG</i> <i>D</i>	触发DMA请求使能 TIMERx (x=1,2,8,11 for GD32L233, x=0,1,2,8,11,14,40 for GD32L235)
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* enable the TIMER1 update DMA */
```

```
timer_dma_enable(TIMER1, TIMER_DMA_UPD);
```

函数 timer_dma_disable

函数timer_dma_disable描述见下表:

表 3-776. 函数 timer_dma_disable

函数名称	timer_dma_disable
函数原型	void timer_dma_disable(uint32_t timer_periph, uint16_t dma);
功能描述	外设TIMERx的DMA禁能
先决条件	-
被调用函数	-
输入参数{in}	
timer_periph	TIMER外设
<i>TIMERx</i>	参考具体参数
输入参数{in}	
dma	DMA源
<i>TIMER_DMA_UPD</i>	更新DMA请求 TIMERx (x=1,2,5,6,8,11 for GD32L233, x=0,1,2,5,6,8,11,14,40 for GD32L235)
<i>TIMER_DMA_CH0</i> <i>D</i>	通道0比较/捕获DMA请求 TIMERx (x=1,2,8,11 for GD32L233, x=0,1,2,8,11,14,40 for GD32L235)
<i>TIMER_DMA_CH1</i> <i>D</i>	通道1比较/捕获 DMA请求 TIMERx (x=1,2,8,11 for GD32L233, x=0,1,2,8,11,14,40 for GD32L235)
<i>TIMER_DMA_CH2</i> <i>D</i>	通道2比较/捕获DMA请求 TIMERx (x=1,2 for GD32L233, x=0,1,2 for GD32L235)
<i>TIMER_DMA_CH3</i> <i>D</i>	通道3比较/捕获DMA请求 TIMERx (x=1,2 for GD32L233, x=0,1,2 for GD32L235)
<i>TIMER_DMA_CMT</i> <i>D</i>	换相DMA更新请求 TIMERx (x=0,14,40 for GD32L235)
<i>TIMER_DMA_TRG</i>	触发DMA请求使能

<i>D</i>	TIMERx (x=1,2,8,11 for GD32L233, x=0,1,2,8,11,14,40 for GD32L235)
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* disable the TIMER1 update DMA */
```

```
timer_dma_disable(TIMER1, 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 3-777. 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TIMERx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	TIMER外设选择 (x=1,2 for GD32L233, x=0,1,2,14,40 for GD32L235)
输入参数{in}	
dma_request	通道的DMA请求源选择
TIMER_DMAREQUEST_CHANNELEVENT	当通道捕获/比较事件发生时, 发送通道n的DMA请求
TIMER_DMAREQUEST_UPDATEEVENT	当更新事件发生, 发送通道n的DMA请求
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* TIMER1 channel DMA request of channel n is sent when channel y event occurs */
```

```
timer_channel_dma_request_source_select(TIMER1,
TIMER_DMAREQUEST_CHANNELEVENT);
```

函数 timer_dma_transfer_config

函数timer_dma_transfer_config描述见下表：

表 3-778. 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TIMERx的DMA模式
先决条件	-
被调用函数	-
输入参数{in}	
timer_periph	TIMER外设
TIMERx	定时器外设选择 (x=1,2 for GD32L233, x=0,1,2,14,40 for GD32L235)
输入参数{in}	
dma_baseaddr	DMA传输起始地址
TIMER_DMACFG_DMATA_CTL0	DMA传输起始地址: TIMER_DMACFG_DMATA_CTL0
TIMER_DMACFG_DMATA_CTL1	DMA传输起始地址: TIMER_DMACFG_DMATA_CTL1
TIMER_DMACFG_DMATA_SMCFG	DMA传输起始地址: TIMER_DMACFG_DMATA_SMCFG
TIMER_DMACFG_DMATA_DMAINTEN	DMA传输起始地址: TIMER_DMACFG_DMATA_DMAINTEN
TIMER_DMACFG_DMATA_INTF	DMA传输起始地址: TIMER_DMACFG_DMATA_INTF
TIMER_DMACFG_DMATA_SWEVG	DMA传输起始地址: TIMER_DMACFG_DMATA_SWEVG
TIMER_DMACFG_DMATA_CHCTL0	DMA传输起始地址: TIMER_DMACFG_DMATA_CHCTL0
TIMER_DMACFG_DMATA_CHCTL1	DMA传输起始地址: TIMER_DMACFG_DMATA_CHCTL1
TIMER_DMACFG_DMATA_CHCTL2	DMA传输起始地址: TIMER_DMACFG_DMATA_CHCTL2
TIMER_DMACFG_DMATA_CNT	DMA传输起始地址: TIMER_DMACFG_DMATA_CNT
TIMER_DMACFG_DMATA_PSC	DMA传输起始地址: TIMER_DMACFG_DMATA_PSC
TIMER_DMACFG_DMATA_CAR	DMA传输起始地址: TIMER_DMACFG_DMATA_CAR
TIMER_DMACFG_DMATA_CH0CV	DMA传输起始地址: TIMER_DMACFG_DMATA_CH0CV

<i>DMATA_CH0CV</i>	
<i>TIMER_DMACFG_DMATA_CH1CV</i>	DMA传输起始地址: <i>TIMER_DMACFG_DMATA_CH1CV</i>
<i>TIMER_DMACFG_DMATA_CH2CV</i>	DMA传输起始地址: <i>TIMER_DMACFG_DMATA_CH2CV</i>
<i>TIMER_DMACFG_DMATA_CH3CV</i>	DMA传输起始地址: <i>TIMER_DMACFG_DMATA_CH3CV</i>
<i>TIMER_DMACFG_DMATA_CCHP</i>	DMA传输起始地址: <i>TIMER_DMACFG_DMATA_CCHP</i>
<i>TIMER_DMACFG_DMATA_DMACFG</i>	DMA传输起始地址: <i>TIMER_DMACFG_DMATA_DMACFG</i>
输入参数{in}	
dma_lenth	DMA传输长度
<i>TIMER_DMACFG_DMATC_xTRANSFER</i>	x=1..18, DMA传输 x 次
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* configure the TIMER1 DMA transfer */
```

```
timer_dma_transfer_config(TIMER1, 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 3-779. 函数 timer_event_software_generate

函数名称	timer_event_software_generate
函数原型	void timer_event_software_generate(uint32_t timer_periph, uint16_t event);
功能描述	软件产生事件
先决条件	-
被调用函数	-
输入参数{in}	
timer_periph	TIMER外设
<i>TIMERx</i>	参考具体参数
输入参数{in}	
event	事件源
<i>TIMER_EVENT_SRC_UPG</i>	更新事件产生 TIMERx (x=1,2,5,6,8,11 for GD32L233, x=0,1,2,5,6,8,11,14,40 for

	GD32L235)
TIMER_EVENT_SR C_CH0G	通道0捕获或比较事件发生, TIMERx (x=1,2,5,6,8,11 for GD32L233, x=0,1,2,5,6,8,11,14,40 for GD32L235)
TIMER_EVENT_SR C_CH1G	通道1捕获或比较事件发生, TIMERx (x=1,2,8,11 for GD32L233, x=0,1,2,8,11,14,40 for GD32L235)
TIMER_EVENT_SR C_CH2G	通道2捕获或比较事件发生 TIMERx (x=1,2 for GD32L233, x=0,1,2 for GD32L235)
TIMER_EVENT_SR C_CH3G	通道3捕获或比较事件发生 TIMERx (x=1,2 for GD32L233, x=0,1,2 for GD32L235)
TIMER_EVENT_SR C_CMTG	通道换相更新事件发生 TIMERx (x=0,14,40 for GD32L235)
TIMER_EVENT_SR C_TRGG	触发事件产生 TIMERx (x=1,2,8,11 for GD32L233, x=0,1,2,8,11,14,40 for GD32L235)
TIMER_EVENT_SR C_BRKG	产生中止事件 TIMERx (x=0,14,40 for GD32L235)
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* software generate update event*/
```

```
timer_event_software_generate(TIMER1, TIMER_EVENT_SRC_UPG);
```

函数 timer_break_struct_para_init (只对 GD32L235 有效)

函数timer_break_struct_para_init描述见下表:

表 3-780. 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（只对 GD32L235 有效）

函数timer_break_config描述见下表：

表 3-781. 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, 14, 40)	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 = 255;
```

```
timer_breakpara.breakpolarity = TIMER_BREAK_POLARITY_LOW;
```

```
timer_breakpara.outputautostate = TIMER_OUTAUTO_ENABLE;
```

```
timer_breakpara.protectmode = TIMER_CCHP_PROT_0;
```

```
timer_breakpara.breakstate = TIMER_BREAK_ENABLE;
```

```
timer_break_config(TIMER0,&timer_breakpara);
```

函数 **timer_break_enable**（只对 **GD32L235** 有效）

函数timer_break_enable描述见下表：

表 3-782. 函数 **timer_break_enable**

函数名称	timer_break_enable
函数原型	void timer_break_enable(uint32_t timer_periph);
功能描述	使能TIMERx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, 14, 40)	TIMER外设选择
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* enable TIMER0 break function*/
```

```
timer_break_enable (TIMER0);
```

函数 **timer_break_disable**（只对 **GD32L235** 有效）

函数timer_break_disable描述见下表：

表 3-783. 函数 **timer_break_disable**

函数名称	timer_break_disable
函数原型	void timer_break_disable (uint32_t timer_periph);
功能描述	禁能TIMERx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)	TIMER外设选择
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* disable TIMER0 break function*/
```

timer_break_disable (TIMER0);

函数 timer_automatic_output_enable（只对 GD32L235 有效）

函数timer_automatic_output_enable描述见下表：

表 3-784. 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, 14, 40)	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（只对 GD32L235 有效）

函数timer_automatic_output_disable描述见下表：

表 3-785. 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, 14, 40)	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 (只对 GD32L235 有效)

函数timer_primary_output_config描述见下表:

表 3-786. 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, 14, 40)	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 (只对 GD32L235 有效)

函数timer_channel_control_shadow_config描述见下表:

表 3-787. 函数 timer_channel_control_shadow_config

函数名称	timer_channel_control_shadow_config
函数原型	void timer_channel_control_shadow_config(uint32_t timer_periph, ControlStatus newvalue);
功能描述	通道换相控制影子配置
先决条件	-
被调用函数	-

输入参数{in}	
timer_periph	TIMER外设
<i>TIMERx</i> (<i>x</i> =0, 14, 40)	TIMER外设选择
输入参数{in}	
newvalue	控制状态
ENABLE	使能
DISABLE	禁能
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* channel capture/compare control shadow register enable */
```

```
timer_channel_control_shadow_config (TIMER0, ENABLE);
```

函数 timer_channel_control_shadow_update_config (只对 GD32L235 有效)

函数timer_channel_control_shadow_update_config描述见下表:

表 3-788. 函数 timer_channel_control_shadow_update_config

函数名称	timer_channel_control_shadow_update_config
函数原型	void timer_channel_control_shadow_update_config(uint32_t timer_periph, uint8_t ccuctl);
功能描述	通道换相控制影子寄存器更新控制
先决条件	-
被调用函数	-
输入参数{in}	
timer_periph	TIMER外设
<i>TIMERx</i> (<i>x</i> =0, 14, 40)	TIMER外设选择
输入参数{in}	
ccuctl	通道换相控制影子寄存器更新控制
<i>TIMER_UPDATECT_L_CCUC</i>	CMTG位被置1时更新影子寄存器
<i>TIMER_UPDATECT_L_CCUTRI</i>	当CMTG位被置1或检测到TRIGI上升沿时，影子寄存器更新
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* configure TIMER0 channel control shadow register update when CMTG bit is set */
timer_channel_control_shadow_update_config (TIMER0, TIMER_UPDATECTL_CCU);
```

函数 timer_channel_output_struct_para_init

函数timer_channel_output_struct_para_init描述见下表：

表 3-789. 函数 timer_channel_output_struct_para_init

函数名称	timer_channel_output_struct_para_init
函数原型	void timer_channel_output_struct_para_init(timer_oc_parameter_struct* ocpara);
功能描述	将TIMER通道输出参数结构体中所有参数初始化为默认值
先决条件	-
被调用函数	-
输入参数{in}	
ocpara	输出通道结构体，详见 结构体timer_oc_parameter_struct
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* initialize TIMER channel output parameter struct with a default value */
timer_oc_parameter_struct timer_ocinitpara;
timer_channel_output_struct_para_init(timer_ocinitpara);
```

函数 timer_channel_output_config

函数timer_channel_output_config描述见下表：

表 3-790. 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TIMERx的通道输出配置
先决条件	-
被调用函数	-
输入参数{in}	
timer_periph	TIMER外设
TIMERx	参考具体参数
输入参数{in}	

channel	待配置通道
<i>TIMER_CH_0</i>	通道0 TIMERx (x=1,2,8,11 for GD32L233, x=0,1,2,8,11,14,40 for GD32L235)
<i>TIMER_CH_1</i>	通道1 TIMERx (x=1,2,8,11 for GD32L233, x=0,1,2,8,11,14,40 for GD32L235)
<i>TIMER_CH_2</i>	通道2 TIMERx (x=1,2 for GD32L233, x=0,1,2 for GD32L235)
<i>TIMER_CH_3</i>	通道3 TIMERx (x=1,2 for GD32L233, x=0,1,2 for GD32L235)
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 TIMER1 channel 0 output function */

timer_oc_parameter_struct timer_ocintpara;

timer_ocintpara.outputstate = TIMER_CCX_ENABLE;

timer_ocintpara.ocpolarity = TIMER_OC_POLARITY_HIGH;

timer_channel_output_config(TIMER1, 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 3-791. 函数 timer_channel_output_mode_config

函数名称	timer_channel_output_mode_config
函数原型	void timer_channel_output_mode_config(uint32_t timer_periph, uint16_t channel, uint16_t ocmode);
功能描述	配置外设TIMERx通道输出比较模式
先决条件	-
被调用函数	-
输入参数{in}	
timer_periph	TIMER外设
<i>TIMERx</i>	参考具体参数
输入参数{in}	
channel	待配置通道
<i>TIMER_CH_0</i>	通道0 TIMERx (x=1,2,8,11 for GD32L233, x=0,1,2,8,11,14,40 for GD32L235)
<i>TIMER_CH_1</i>	通道1

	TIMERx (x=1,2,8,11 for GD32L233, x=0,1,2,8,11,14,40 for GD32L235)
<i>TIMER_CH_2</i>	通道2 TIMERx (x=1,2 for GD32L233, x=0,1,2 for GD32L235)
<i>TIMER_CH_3</i>	通道3 TIMERx (x=1,2 for GD32L233, x=0,1,2 for GD32L235)
输入参数{in}	
ocmode	通道输出比较模式
<i>TIMER_OC_MODE_TIMING</i>	冻结模式
<i>TIMER_OC_MODE_ACTIVE</i>	匹配时设置为高
<i>TIMER_OC_MODE_INACTIVE</i>	匹配时设置为低
<i>TIMER_OC_MODE_TOGGLE</i>	匹配时翻转
<i>TIMER_OC_MODE_LOW</i>	强制为低
<i>TIMER_OC_MODE_HIGH</i>	强制为高
<i>TIMER_OC_MODE_PWM0</i>	PWM模式0
<i>TIMER_OC_MODE_PWM1</i>	PWM模式1
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* configure TIMER1 channel PWM 0 mode */
```

```
timer_channel_output_mode_config(TIMER1, 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 3-792. 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TIMERx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>	参考具体参数
输入参数{in}	
channel	待配置通道
<i>TIMER_CH_0</i>	通道0 TIMERx (x=1,2,8,11 for GD32L233, x=0,1,2,8,11,14,40 for GD32L235)
<i>TIMER_CH_1</i>	通道1 TIMERx (x=1,2,8,11 for GD32L233, x=0,1,2,8,11,14,40 for GD32L235)
<i>TIMER_CH_2</i>	通道2 TIMERx (x=1,2 for GD32L233, x=0,1,2 for GD32L235)
<i>TIMER_CH_3</i>	通道3 TIMERx (x=1,2 for GD32L233, x=0,1,2 for GD32L235)
输入参数{in}	
pulse	通道输出比较值 (0~65535)
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* configure TIMER1 channel 0 output pulse value */
timer_channel_output_pulse_value_config(TIMER1, TIMER_CH_0, 399);
```

函数 timer_channel_output_shadow_config

函数timer_channel_output_shadow_config描述见下表:

表 3-793. 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TIMERx通道输出比较影子寄存器功能
先决条件	-
被调用函数	-
输入参数{in}	
timer_periph	TIMER外设
<i>TIMERx</i>	参考具体参数
输入参数{in}	
channel	待配置通道
<i>TIMER_CH_0</i>	通道0 TIMERx (x=1,2,8,11 for GD32L233, x=0,1,2,8,11,14,40 for GD32L235)
<i>TIMER_CH_1</i>	通道1

	TIMERx (x=1,2,8,11 for GD32L233, x=0,1,2,8,11,14,40 for GD32L235)
<i>TIMER_CH_2</i>	通道2 TIMERx (x=1,2 for GD32L233, x=0,1,2 for GD32L235)
<i>TIMER_CH_3</i>	通道3 TIMERx (x=1,2 for GD32L233, x=0,1,2 for GD32L235)
输入参数{in}	
ocshadow	输出比较影子寄存器功能状态
<i>TIMER_OC_SHADOW_ENABLE</i>	使能输出比较影子寄存器
<i>TIMER_OC_SHADOW_DISABLE</i>	禁能输出比较影子寄存器
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/*configure TIMER1 channel 0 output shadow function */
```

```
timer_channel_output_shadow_config(TIMER1, TIMER_CH_0,  
TIMER_OC_SHADOW_ENABLE);
```

函数 timer_channel_output_fast_config

函数timer_channel_output_fast_config描述见下表:

表 3-794. 函数 timer_channel_output_fast_config

函数名称	timer_channel_output_fast_config
函数原型	void timer_channel_output_fast_config(uint32_t timer_periph, uint16_t channel, uint16_t ocfast);
功能描述	配置TIMERx通道输出比较快速功能
先决条件	-
被调用函数	-
输入参数{in}	
timer_periph	TIMER外设
<i>TIMERx</i>	参考具体参数
输入参数{in}	
channel	待配置通道
<i>TIMER_CH_0</i>	通道0 TIMERx (x=1,2,8,11 for GD32L233, x=0,1,2,8,11,14,40 for GD32L235)
<i>TIMER_CH_1</i>	通道1 TIMERx (x=1,2,8,11 for GD32L233, x=0,1,2,8,11,14,40 for GD32L235)
<i>TIMER_CH_2</i>	通道2 TIMERx (x=1,2 for GD32L233, x=0,1,2 for GD32L235)

<i>TIMER_CH_3</i>	通道3 TIMERx (x=1,2 for GD32L233, x=0,1,2 for GD32L235)
输入参数{in}	
ocfast	通道输出比较快速功能状态
<i>TIMER_OC_FAST_ENABLE</i>	通道输出比较快速功能使能
<i>TIMER_OC_FAST_DISABLE</i>	通道输出比较快速功能禁能
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* configure TIMER1 channel 0 output fast function */
```

```
timer_channel_output_fast_config(TIMER1, TIMER_CH_0, TIMER_OC_FAST_ENABLE);
```

函数 timer_channel_output_clear_config

函数timer_channel_output_clear_config描述见下表:

表 3-795. 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TIMERx的通道输出比较清0功能
先决条件	-
被调用函数	-
输入参数{in}	
timer_periph	TIMER外设
<i>TIMERx</i>	参考具体参数
输入参数{in}	
channel	待配置通道
<i>TIMER_CH_0</i>	通道0 TIMERx (x=1,2 for GD32L233, x=0,1,2 for GD32L235)
<i>TIMER_CH_1</i>	通道1 TIMERx (x=1,2 for GD32L233, x=0,1,2 for GD32L235)
<i>TIMER_CH_2</i>	通道2 TIMERx (x=1,2 for GD32L233, x=0,1,2 for GD32L235)
<i>TIMER_CH_3</i>	通道3 TIMERx (x=1,2 for GD32L233, x=0,1,2 for GD32L235)
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
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* configure TIMER1 channel 0 output clear function */
```

```
timer_channel_output_clear_config(TIMER1, TIMER_CH_0,  
TIMER_OC_CLEAR_ENABLE);
```

函数 timer_channel_output_polarity_config

函数timer_channel_output_polarity_config描述见下表：

表 3-796. 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</i>	参考具体参数
输入参数{in}	
channel	待配置通道
<i>TIMER_CH_0</i>	通道0 TIMERx (x=1,2,8,11 for GD32L233, x=0,1,2,8,11,14,40 for GD32L235)
<i>TIMER_CH_1</i>	通道1 TIMERx (x=1,2,8,11 for GD32L233, x=0,1,2,8,11,14,40 for GD32L235)
<i>TIMER_CH_2</i>	通道2 TIMERx (x=1,2 for GD32L233, x=0,1,2 for GD32L235)
<i>TIMER_CH_3</i>	通道3 TIMERx (x=1,2 for GD32L233, x=0,1,2 for GD32L235)
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

ITY_LOW	
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* configure TIMER1 channel 0 output polarity */
```

```
timer_channel_output_polarity_config(TIMER1, TIMER_CH_0,  
TIMER_OC_POLARITY_HIGH);
```

函数 timer_channel_complementary_output_polarity_config（只对 GD32L235 有效）

函数timer_channel_complementary_output_polarity_config描述见下表：

表 3-797. 函数 timer_channel_complementary_output_polarity_config

函数名称	timer_channel_complementary_output_polarity_config
函数原型	void timer_channel_complementary_output_polarity_config(uint32_t timer_periph, uint16_t channel, uint16_t ocnpolarity);
功能描述	互补通道输出极性配置
先决条件	-
被调用函数	-
输入参数{in}	
timer_periph	TIMER外设
TIMERx	参考具体参数
输入参数{in}	
channel	待配置通道
TIMER_CH_0	通道0, TIMERx (x=0, 14, 40)
TIMER_CH_1	通道1, TIMERx (x=0)
TIMER_CH_2	通道2, TIMERx (x=0)
输入参数{in}	
ocnpolarity	互补通道输出极性
TIMER_OCN_POLARITY_HIGH	互补通道输出极性高电平有效
TIMER_OCN_POLARITY_LOW	互补通道输出极性低电平有效
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* configure TIMER0 channel 0 complementary output polarity */
```

```
timer_channel_complementary_output_polarity_config (TIMER0, TIMER_CH_0,  
TIMER_OCN_POLARITY_HIGH);
```

函数 timer_channel_output_state_config

函数timer_channel_output_state_config描述见下表:

表 3-798. 函数 timer_channel_output_state_config

函数名称	timer_channel_output_state_config
函数原型	void timer_channel_output_state_config(uint32_t timer_periph, uint16_t channel, uint32_t state);
功能描述	配置通道状态
先决条件	-
被调用函数	-
输入参数{in}	
timer_periph	TIMER外设
TIMERx	参考具体参数
输入参数{in}	
channel	待配置通道
TIMER_CH_0	通道0 TIMERx (x=1,2,8,11 for GD32L233, x=0,1,2,8,11,14,40 for GD32L235)
TIMER_CH_1	通道1 TIMERx (x=1,2,8,11 for GD32L233, x=0,1,2,8,11,14,40 for GD32L235)
TIMER_CH_2	通道2 TIMERx (x=1,2 for GD32L233, x=0,1,2 for GD32L235)
TIMER_CH_3	通道3 TIMERx (x=1,2 for GD32L233, x=0,1,2 for GD32L235)
输入参数{in}	
state	通道状态
TIMER_CCX_ENABLE	通道使能
TIMER_CCX_DISABLE	通道禁能
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* configure TIMER1 channel 0 enable state */
```

```
timer_channel_output_state_config(TIMER1, TIMER_CH_0, TIMER_CCX_ENABLE);
```

函数 **timer_channel_complementary_output_state_config** （只对 GD32L235 有效）

函数timer_channel_complementary_output_state_config描述见下表：

表 3-799. 函数 timer_channel_complementary_output_state_config

函数名称	timer_channel_complementary_output_state_config
函数原型	void timer_channel_complementary_output_state_config(uint32_t timer_periph, uint16_t channel, uint16_t ocnstate);
功能描述	配置互补通道输出状态
先决条件	-
被调用函数	-
输入参数{in}	
timer_periph	TIMER外设
TIMERx	参考具体参数
输入参数{in}	
channel	待配置通道
TIMER_CH_0	通道0, TIMERx (x=0, 14, 40)
TIMER_CH_1	通道1, TIMERx (x=0)
TIMER_CH_2	通道2, TIMERx (x=0)
输入参数{in}	
state	互补通道状态
TIMER_CCXN_ENABLE	互补通道使能
TIMER_CCXN_DISABLE	互补通道禁能
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* configure TIMER0 channel 0 complementary output enable state */
timer_channel_complementary_output_state_config (TIMER0, TIMER_CH_0,
TIMER_CCXN_ENABLE);
```

函数 **timer_channel_input_struct_para_init**

函数timer_channel_input_struct_para_init描述见下表：

表 3-800. 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 表3-754. 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 3-801. 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TIMERx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	参考具体参数
输入参数{in}	
channel	待配置通道
TIMER_CH_0	通道0 TIMERx (x=1,2,8,11 for GD32L233, x=0,1,2,8,11,14,40 for GD32L235)
TIMER_CH_1	通道1 TIMERx (x=1,2,8,11 for GD32L233, x=0,1,2,8,11,14,40 for GD32L235)
TIMER_CH_2	通道2 TIMERx (x=1,2 for GD32L233, x=0,1,2 for GD32L235)
TIMER_CH_3	通道3 TIMERx (x=1,2 for GD32L233, x=0,1,2 for GD32L235)
输入参数{in}	
icpara	输入捕获结构体, 详见 表3-754. 结构体timer_ic_parameter_struct
输出参数{out}	

-	-
返回值	
-	-

例如：

```
/* configure TIMER1 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(TIMER1, 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 3-802. 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TIMERx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	参考具体参数
输入参数{in}	
channel	待配置通道
TIMER_CH_0	通道0 TIMERx (x=1,2,8,11 for GD32L233, x=0,1,2,8,11,14,40 for GD32L235)
TIMER_CH_1	通道1 TIMERx (x=1,2,8,11 for GD32L233, x=0,1,2,8,11,14,40 for GD32L235)
TIMER_CH_2	通道2 TIMERx (x=1,2 for GD32L233, x=0,1,2 for GD32L235)
TIMER_CH_3	通道3 TIMERx (x=1,2 for GD32L233, x=0,1,2 for GD32L235)
输入参数{in}	
prescaler	通道输入捕获预分频值
TIMER_IC_PSC_DIV1	不分频

<i>TIMER_IC_PSC_DIV2</i>	2分频
<i>TIMER_IC_PSC_DIV4</i>	4分频
<i>TIMER_IC_PSC_DIV8</i>	8分频
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* configure TIMER1 channel 0 input capture prescaler value */
timer_channel_input_capture_prescaler_config(TIMER1, 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 3-803. 函数 timer_channel_capture_value_register_read

函数名称	timer_channel_capture_value_register_read
函数原型	uint32_t timer_channel_capture_value_register_read(uint32_t timer_periph, uint16_t channel);
功能描述	读取通道捕获值
先决条件	-
被调用函数	-
输入参数{in}	
timer_periph	TIMER外设
<i>TIMERx</i>	参考具体参数
输入参数{in}	
channel	待配置通道
<i>TIMER_CH_0</i>	通道0 TIMERx (x=1,2,8,11 for GD32L233, x=0,1,2,8,11,14,40 for GD32L235)
<i>TIMER_CH_1</i>	通道1 TIMERx (x=1,2,8,11 for GD32L233, x=0,1,2,8,11,14,40 for GD32L235)
<i>TIMER_CH_2</i>	通道2 TIMERx (x=1,2 for GD32L233, x=0,1,2 for GD32L235)
<i>TIMER_CH_3</i>	通道3 TIMERx (x=1,2 for GD32L233, x=0,1,2 for GD32L235)
输出参数{out}	
-	-
返回值	

uint32_t	通道输入捕获值，（0~65535）
----------	-------------------

例如：

```
/* read TIMER1 channel 0 capture compare register value */
```

```
uint32_t ch0_value = 0;
```

```
ch0_value = timer_channel_capture_value_register_read(TIMER1, TIMER_CH_0);
```

函数 timer_input_pwm_capture_config

函数timer_input_pwm_capture_config描述见下表：

表 3-804. 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TIMERx捕获PWM输入参数
先决条件	-
被调用函数	timer_channel_input_capture_prescaler_config
输入参数{in}	
timer_periph	TIMER外设
TIMERx	TIMER外设选择 x=1,2,8,11 for GD32L233, x=0,1,2,8,11,14,40 for GD32L235
输入参数{in}	
channel	待配置通道
TIMER_CH_0	通道0
TIMER_CH_1	通道1
输入参数{in}	
icpwm	输入捕获结构体，详见 表3-754. 结构体timer_ic_parameter_struct
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* configure TIMER1 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(TIMER1, TIMER_CH_0, &timer_icinitpara);
```

函数 timer_hall_mode_config

函数timer_hall_mode_config描述见下表:

表 3-805. 函数 timer_hall_mode_config

函数名称	timer_hall_mode_config
函数原型	void timer_hall_mode_config(uint32_t timer_periph, uint8_t hallmode);
功能描述	配置TIMERx的HALL接口功能
先决条件	-
被调用函数	-
输入参数{in}	
timer_periph	TIMER外设
TIMERx	TIMER外设选择 x=1,2 for GD32L233, x=0,1,2 for GD32L235
输入参数{in}	
hallmode	HALL接口功能状态
TIMER_HALLINTE RFACE_ENABLE	HALL接口使能
TIMER_HALLINTE RFACE_DISABLE	HALL接口禁能
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* configure TIMER1 hall sensor mode */
```

```
timer_hall_mode_config(TIMER1, TIMER_HALLINTERFACE_ENABLE);
```

函数 timer_input_trigger_source_select

函数timer_input_trigger_source_select描述见下表:

表 3-806. 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	TIMERx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

<i>TIMERx</i>	TIMER外设选择
输入参数{in}	
intrigger	待选择的触发源
<i>TIMER_SMCFG_TRGSEL_ITI0</i>	内部触发输入0 TIMERx (x=1,2,8,11 for GD32L233, x=0,1,2,8,11,14,40 for GD32L235)
<i>TIMER_SMCFG_TRGSEL_ITI1</i>	内部触发输入1 TIMERx (x=1,2,8,11 for GD32L233, x=0,1,2,8,11,14,40 for GD32L235)
<i>TIMER_SMCFG_TRGSEL_ITI2</i>	内部触发输入2 TIMERx (x=1,2,8,11 for GD32L233, x=0,1,2,8,11,14,40 for GD32L235)
<i>TIMER_SMCFG_TRGSEL_ITI3</i>	内部触发输入3 TIMERx (x=1,2,8,11 for GD32L233, x=0,1,2,8,11,14,40 for GD32L235)
<i>TIMER_SMCFG_TRGSEL_CIOF_ED</i>	CI0的边沿标志位 TIMERx (x=1,2,8,11 for GD32L233, x=0,1,2,8,11,14,40 for GD32L235)
<i>TIMER_SMCFG_TRGSEL_CIOFE0</i>	滤波后的通道0输入 TIMERx (x=1,2,8,11 for GD32L233, x=0,1,2,8,11,14,40 for GD32L235)
<i>TIMER_SMCFG_TRGSEL_CIOFE1</i>	滤波后的通道1输入 TIMERx (x=1,2,8,11 for GD32L233, x=0,1,2,8,11,14,40 for GD32L235)
<i>TIMER_SMCFG_TRGSEL_ETIFP</i>	滤波后的外部触发输入 TIMERx (x=1,2,8,11 for GD32L233, x=0,1,2,8,11,14,40 for GD32L235)
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* select TIMER1 input trigger source */
```

```
timer_input_trigger_source_select(TIMER1, TIMER_SMCFG_TRGSEL_ITI0);
```

函数 timer_master_output_trigger_source_select

函数timer_master_output_trigger_source_select描述见下表:

表 3-807. 函数 timer_master_output_trigger_source_select

函数名称	timer_master_output_trigger_source_select
函数原型	void timer_master_output_trigger_source_select(uint32_t timer_periph, uint32_t outtrigger);
功能描述	选择TIMERx主模式输出触发
先决条件	-
被调用函数	-
输入参数{in}	
timer_periph	TIMER外设
<i>TIMERx</i>	TIMER外设选择
输入参数{in}	

outrigger	主模式输出触发
<code>TIMER_TRI_OUT_SRC_RESET</code>	复位。TIMERx_SWEVG寄存器的UPG位被置1或从模式控制器产生复位触发一次TRGO脉冲，后一种情况下，TRGO上的信号相对实际的复位会有一个延迟。
<code>TIMER_TRI_OUT_SRC_ENABLE</code>	使能。此模式可用于同时启动多个定时器或控制在一段时间内使能从定时器。主模式控制器选择计数器使能信号作为触发输出TRGO。当CEN控制位被置1或者暂停模式下触发输入为高电平时，计数器使能信号被置1。在暂停模式下，计数器使能信号受控于触发输入，在触发输入和TRGO上会有一个延迟，除非选择了主/从模式。
<code>TIMER_TRI_OUT_SRC_UPDATE</code>	更新。主模式控制器选择更新事件作为TRGO。
<code>TIMER_TRI_OUT_SRC_CH0</code>	捕获/比较脉冲.通道0在发生一次捕获或一次比较成功时，主模式控制器产生一个TRGO脉冲
<code>TIMER_TRI_OUT_SRC_O0CPRE</code>	比较。在这种模式下主模式控制器选择O0CPRE信号被用于作为触发输出TRGO
<code>TIMER_TRI_OUT_SRC_O1CPRE</code>	比较。在这种模式下主模式控制器选择O1CPRE信号被用于作为触发输出TRGO
<code>TIMER_TRI_OUT_SRC_O2CPRE</code>	比较。在这种模式下主模式控制器选择O2CPRE信号被用于作为触发输出TRGO
<code>TIMER_TRI_OUT_SRC_O3CPRE</code>	比较。在这种模式下主模式控制器选择O3CPRE信号被用于作为触发输出TRGO
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* select TIMER1 master mode output trigger source */
```

```
timer_master_output_trigger_source_select(TIMER1, TIMER_TRI_OUT_SRC_RESET);
```

函数 timer_slave_mode_select

函数timer_slave_mode_select描述见下表：

表 3-808. 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	TIMERx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	TIMER外设选择
输入参数{in}	

slavemode	从模式
<i>TIMER_SLAVE_MODE_DISABLE</i>	关闭从模式
<i>TIMER_QUAD_DECODER_MODE0</i>	正交译码器模式0 TIMERx (x=1,2 for GD32L233, x=0,1,2 for GD32L235)
<i>TIMER_QUAD_DECODER_MODE1</i>	正交译码器模式1 TIMERx (x=1,2 for GD32L233, x=0,1,2 for GD32L235)
<i>TIMER_QUAD_DECODER_MODE2</i>	正交译码器模式2 TIMERx (x=1,2 for GD32L233, x=0,1,2 for GD32L235)
<i>TIMER_SLAVE_MODE_RESTART</i>	复位模式 TIMERx (x=1,2,8,11 for GD32L233, x=0,1,2,8,11,14,40 for GD32L235)
<i>TIMER_SLAVE_MODE_PAUSE</i>	暂停模式 TIMERx (x=1,2,8,11 for GD32L233, x=0,1,2,8,11,14,40 for GD32L235)
<i>TIMER_SLAVE_MODE_EVENT</i>	事件模式 TIMERx (x=1,2,8,11 for GD32L233, x=0,1,2,8,11,14,40 for GD32L235)
<i>TIMER_SLAVE_MODE_EXTERNAL0</i>	外部时钟模式0 TIMERx (x=1,2,8,11 for GD32L233, x=0,1,2,8,11,14,40 for GD32L235)
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* select TIMER1 slave mode */
```

```
timer_slave_mode_select(TIMER1, TIMER_QUAD_DECODER_MODE0);
```

函数 timer_master_slave_mode_config

函数timer_master_slave_mode_config描述见下表:

表 3-809. 函数 timer_master_slave_mode_config

函数名称	timer_master_slave_mode_config
函数原型	void timer_master_slave_mode_config(uint32_t timer_periph, uint8_t masterslave);
功能描述	TIMERx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	TIMER外设选择 x=1,2,8,11 for GD32L233, x=0,1,2,8,11,14,40 for GD32L235
输入参数{in}	
masterslave	主从模式使能状态

<i>TIMER_MASTER_SLAVE_MODE_ENABLE</i>	主从模式使能
<i>TIMER_MASTER_SLAVE_MODE_DISABLE</i>	主从模式禁能
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* configure TIMER1 master slave mode */
```

```
timer_master_slave_mode_config(TIMER1, TIMER_MASTER_SLAVE_MODE_ENABLE);
```

函数 timer_external_trigger_config

函数timer_external_trigger_config描述见下表：

表 3-810. 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TIMERx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	TIMER外设选择 x=1,2 for GD32L233, x=0,1,2 for GD32L235
输入参数{in}	
extprescaler	外部触发预分频
TIMER_EXT_TRIP_SC_OFF	不分频
TIMER_EXT_TRIP_SC_DIV2	2分频
TIMER_EXT_TRIP_SC_DIV4	4分频
TIMER_EXT_TRIP_SC_DIV8	8分频
输入参数{in}	
expolarity	外部触发输入极性
TIMER_ETP_FALLI	低电平或者下降沿有效

NG	
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 TIMER1 external trigger input */
```

```
timer_external_trigger_config(TIMER1, 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 3-811. 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	TIMERx配置为编码器模式
先决条件	-
被调用函数	-
输入参数{in}	
timer_periph	TIMER外设
TIMERx	TIMER外设选择 x=1,2 for GD32L233, x=0,1,2 for GD32L235
输入参数{in}	
decomode	编码器模式
TIMER_QUAD_DECODER_MODE0	根据CI0FE0的电平，计数器在CI1FE1的边沿向上/下计数
TIMER_QUAD_DECODER_MODE1	根据CI1FE1的电平，计数器在CI0FE0的边沿向上/下计数
TIMER_QUAD_DECODER_MODE2	根据另一个信号的输入电平，计数器在CI0FE0和CI1FE1的边沿向上/下计数
输入参数{in}	
ic0polarity	IC0极性
TIMER_IC_POLARITY_RISING	捕获上升边沿
TIMER_IC_POLARITY_FALLING	捕获下降边沿

TY_FALLING	
TIMER_IC_POLARITY_BOTH_EDGE	捕获双边沿
输入参数{in}	
ic1polarity	IC1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	捕获双边沿
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* configure TIMER1 quadrature decoder mode */
```

```
timer_quadrature_decoder_mode_config(TIMER1,    TIMER_QUAD_DECODER_MODE0,
timer_IC_POLARITY_RISING, TIMER_IC_POLARITY_RISING);
```

函数 timer_internal_clock_config

函数timer_internal_clock_config描述见下表：

表 3-812. 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	TIMERx配置为内部时钟模式
先决条件	-
被调用函数	-
输入参数{in}	
timer_periph	TIMER外设
TIMERx	TIMER外设选择 x=1,2,8,11 for GD32L233, x=0,1,2,8,11,14,40 for GD32L235
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 3-813. 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TIMERx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	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	选择内部触发0 (ITI0) TIMERx (x=1,2,8,11 for GD32L233, x=0,1,2,8,11,14,40 for GD32L235)
TIMER_SMCFG_T RGSEL_ITI1	选择内部触发1 (ITI1) TIMERx (x=1,2,8,11 for GD32L233, x=0,1,2,8,11,14,40 for GD32L235)
TIMER_SMCFG_T RGSEL_ITI2	选择内部触发2 (ITI2) TIMERx (x=1,2,8,11 for GD32L233, x=0,1,2,8,11,14,40 for GD32L235)
TIMER_SMCFG_T RGSEL_ITI3	选择内部触发3 (ITI3) TIMERx (x=1,2,8,11 for GD32L233, x=0,1,2,8,11,14,40 for GD32L235)
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* configure TIMER1 the internal trigger ITI0 as external clock input */
```

```
timer_internal_trigger_as_external_clock_config(TIMER1, TIMER_SMCFG_TRGSEL_ITI0);
```

函数 timer_external_trigger_as_external_clock_config

函数timer_external_trigger_as_external_clock_config描述见下表：

表 3-814. 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TIMERx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
<i>TIMERx</i>	TIMER外设选择 x=1,2,8,11 for GD32L233, x=0,1,2,8,11,14,40 for GD32L235
输入参数{in}	
extrigger	外部触发源
<i>TIMER_SMCFG_TRGSEL_CIOF_ED</i>	CI0的边沿标志 (CIOF_ED)
<i>TIMER_SMCFG_TRGSEL_CIOFE0</i>	滤波后的通道0输入 (CIOFE0)
<i>TIMER_SMCFG_TRGSEL_C1FE1</i>	滤波后的通道1输入 (C1FE1)
输入参数{in}	
expolarity	外部触发源极性
<i>TIMER_IC_POLARITY_RISING</i>	外部触发源高电平或者上升沿有效
<i>TIMER_IC_POLARITY_FALLING</i>	外部触发源低电平或者下降沿有效
<i>TIMER_IC_POLARITY_BOTH_EDGE</i>	下降沿或者上升沿有效
输入参数{in}	
extfilter	滤波参数 (0~15)
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* configure TIMER1 the external trigger CIOFE0 as external clock input */
```

```
timer_external_trigger_as_external_clock_config(TIMER1,
TIMER_SMCFG_TRGSEL_CIOFE0, TIMER_IC_POLARITY_RISING, 0);
```

函数 timer_external_clock_mode0_config

函数timer_external_clock_mode0_config描述见下表:

表 3-815. 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TIMERx外部时钟模式0, ETI作为时钟源
先决条件	-
被调用函数	timer_external_trigger_config

输入参数{in}	
timer_periph	TIMER外设
<i>TIMERx</i>	TIMER外设选择 x=1,2,8,11 for GD32L233, x=0,1,2,8,11,14,40 for GD32L235
输入参数{in}	
extprescaler	ETI触发源预分频值
<i>TIMER_EXT_TRI_P</i> <i>SC_OFF</i>	不分频
<i>TIMER_EXT_TRI_P</i> <i>SC_DIV2</i>	2分频
<i>TIMER_EXT_TRI_P</i> <i>SC_DIV4</i>	4分频
<i>TIMER_EXT_TRI_P</i> <i>SC_DIV8</i>	8分频
输入参数{in}	
expolarity	ETI触发源极性
<i>TIMER_ETP_FALLI</i> <i>NG</i>	下降沿或者低电平有效
<i>TIMER_ETP_RISIN</i> <i>G</i>	上升沿或者高电平有效
输入参数{in}	
extfilter	ETI触发源滤波参数（0~15）
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* configure TIMER1 the external clock mode0 */
```

```
timer_external_clock_mode0_config(TIMER1, 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 3-816. 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TIMERx外部时钟模式1
先决条件	-
被调用函数	timer_external_trigger_config

输入参数{in}	
timer_periph	TIMER外设
<i>TIMERx</i>	TIMER外设选择 x=1,2 for GD32L233, x=0,1,2 for GD32L235
输入参数{in}	
extprescaler	ETI触发源预分频值
<i>TIMER_EXT_TRI_PSC_OFF</i>	不分频
<i>TIMER_EXT_TRI_PSC_DIV2</i>	2分频
<i>TIMER_EXT_TRI_PSC_DIV4</i>	4分频
<i>TIMER_EXT_TRI_PSC_DIV8</i>	8分频
输入参数{in}	
expolarity	ETI触发源极性
<i>TIMER_ETP_FALLING</i>	下降沿或者低电平有效
<i>TIMER_ETP_RISING</i>	上升沿或者高电平有效
输入参数{in}	
extfilter	ETI触发源滤波参数（0~15）
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* configure TIMER1 the external clock mode1 */
```

```
timer_external_clock_mode1_config(TIMER1, 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 3-817. 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	TIMERx外部时钟模式1禁能
先决条件	-
被调用函数	-
输入参数{in}	

timer_periph	TIMER外设
<i>TIMERx</i>	TIMER外设选择 x=1,2 for GD32L233, x=0,1,2 for GD32L235
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* disable TIMER1 the external clock mode1 */
```

```
timer_external_clock_mode1_disable(TIMER1);
```

函数 timer_channel_remap_config

函数timer_channel_remap_config描述见下表:

表 3-818. 函数 timer_channel_remap_config

函数名称	timer_channel_remap_config
函数原型	void timer_channel_remap_config(uint32_t timer_periph, uint32_t remap);
功能描述	配置 TIMERx 通道重映射功能
先决条件	-
被调用函数	-
输入参数{in}	
timer_periph	TIMER 外设
<i>TIMERx (x=8, 11)</i>	TIMER 外设选择
输入参数{in}	
remap	重映射功能选择
<i>TIMER8_CIO_RMP_GPIO</i>	通道 0 连接到 GPIO TIMERx (x=8)
<i>TIMER8_CIO_RMP_LXTAL</i>	通道 0 连接到 LXTAL TIMERx (x=8)
<i>TIMER8_CIO_RMP_HXTAL_DIV32</i>	通道 0 连接到 HXTAL/32 TIMERx (x=8)
<i>TIMER8_CIO_RMP_CKOUTSEL</i>	通道 0 连接到 CKOUTSEL TIMERx (x=8)
<i>TIMER11_CIO_RMP_GPIO</i>	通道 0 连接到 GPIO TIMERx (x=11)
<i>TIMER11_CIO_RMP_IRC32K</i>	通道 0 连接到 IRC32K TIMERx (x=11)
<i>TIMER11_CIO_RMP_LXTAL</i>	通道 0 连接到 LXTAL TIMERx (x=11)
<i>TIMER11_CIO_RMP_RTC_OUT</i>	通道 0 连接到 RTC_OUT TIMERx (x=11)

输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* configure TIMER8 channel 0 input is connected to GPIO */
```

```
timer_channel_remap_config(TIMER8, TIMER8_C10_RMP_GPIO);
```

函数 timer_write_chxval_register_config

函数timer_write_chxval_register_config描述见下表:

表 3-819. 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TIMERx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	TIMER外设选择 x=1,2,8,11 for GD32L233, x=0,1,2,8,11,14,40 for GD32L235
输入参数{in}	
ccsel	写CHxVAL寄存器选择位
TIMER_CHVSEL_DISABLE	无影响
TIMER_CHVSEL_ENABLE	当写入捕获比较寄存器的值与寄存器当前值相等时，写入操作无效。
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* configure TIMER1 write CHxVAL register selection */
```

```
timer_write_chxval_register_config(TIMER1, TIMER_CHVSEL_ENABLE);
```

函数 timer_flag_get

函数timer_flag_get描述见下表:

表 3-820. 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	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=1,2,5,6,8,11 for GD32L233, x=0,1,2,5,6,8,11,14,40 for GD32L235)
TIMER_FLAG_CH0	通道0比较/捕获标志 TIMERx (x=1,2,8,11 for GD32L233, x=0,1,2,8,11,14,40 for GD32L235)
TIMER_FLAG_CH1	通道1比较/捕获标志 TIMERx (x=1,2,8,11 for GD32L233, x=0,1,2,8,11,14,40 for GD32L235)
TIMER_FLAG_CH2	通道2比较/捕获标志 TIMERx (x=1,2 for GD32L233, x=0,1,2 for GD32L235)
TIMER_FLAG_CH3	通道3比较/捕获标志 TIMERx (x=1,2 for GD32L233, x=0,1,2 for GD32L235)
TIMER_FLAG_CMT	通道换相更新标志 TIMERx (x=0,14,40 for GD32L235)
TIMER_FLAG_TRG	触发标志 TIMERx (x=1,2,8,11 for GD32L233, x=0,1,2,8,11,14,40 for GD32L235)
TIMER_FLAG_BRK	中止标志位 TIMERx (x=0,14,40 for GD32L235)
TIMER_FLAG_CH0 O	通道0捕获溢出标志 TIMERx (x=1,2,8,11 for GD32L233, x=0,1,2,8,11,14,40 for GD32L235)
TIMER_FLAG_CH1 O	通道1捕获溢出标志 TIMERx (x=1,2,8,11 for GD32L233, x=0,1,2,8,11,14,40 for GD32L235)
TIMER_FLAG_CH2 O	通道2捕获溢出标志 TIMERx (x=1,2 for GD32L233, x=0,1,2 for GD32L235)
TIMER_FLAG_CH3 O	通道3捕获溢出标志 TIMERx (x=1,2 for GD32L233, x=0,1,2 for GD32L235)
输出参数{out}	
-	-
返回值	
FlagStatus	SET或者RESET

例如:

```
/* get TIMER1 update flags */
```

```
FlagStatus Flag_status = RESET;
```

```
Flag_status = timer_flag_get(TIMER1, TIMER_FLAG_UP);
```

函数 timer_flag_clear

函数timer_flag_clear描述见下表：

表 3-821. 函数 timer_flag_clear

函数名称	timer_flag_clear
函数原型	void timer_flag_clear(uint32_t timer_periph, uint32_t flag);
功能描述	清除外设TIMERx状态标志
先决条件	-
被调用函数	-
输入参数{in}	
timer_periph	TIMER外设
TIMERx	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=1,2,5,6,8,11 for GD32L233, x=0,1,2,5,6,8,11,14,40 for GD32L235)
TIMER_FLAG_CH0	通道0比较/捕获标志 TIMERx (x=1,2,8,11 for GD32L233, x=0,1,2,8,11,14,40 for GD32L235)
TIMER_FLAG_CH1	通道1比较/捕获标志 TIMERx (x=1,2,8,11 for GD32L233, x=0,1,2,8,11,14,40 for GD32L235)
TIMER_FLAG_CH2	通道2比较/捕获标志 TIMERx (x=1,2 for GD32L233, x=0,1,2 for GD32L235)
TIMER_FLAG_CH3	通道3比较/捕获标志 TIMERx (x=1,2 for GD32L233, x=0,1,2 for GD32L235)
TIMER_FLAG_CMT	通道换相更新标志 TIMERx (x=0,14,40 for GD32L235)
TIMER_FLAG_TRG	触发标志 TIMERx (x=1,2,8,11 for GD32L233, x=0,1,2,8,11,14,40 for GD32L235)
TIMER_FLAG_BRK	中止标志位 TIMERx (x=0,14,40 for GD32L235)
TIMER_FLAG_CH0 O	通道0捕获溢出标志 TIMERx (x=1,2,8,11 for GD32L233, x=0,1,2,8,11,14,40 for GD32L235)
TIMER_FLAG_CH1 O	通道1捕获溢出标志 TIMERx (x=1,2,8,11 for GD32L233, x=0,1,2,8,11,14,40 for GD32L235)
TIMER_FLAG_CH2 O	通道2捕获溢出标志 TIMERx (x=1,2 for GD32L233, x=0,1,2 for GD32L235)
TIMER_FLAG_CH3 O	通道3捕获溢出标志 TIMERx (x=1,2 for GD32L233, x=0,1,2 for GD32L235)

输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* clear TIMER1 update flags */
```

```
timer_flag_clear(TIMER1, TIMER_FLAG_UP);
```

函数 timer_interrupt_enable

函数timer_interrupt_enable描述见下表:

表 3-822. 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	参考具体参数
输入参数{in}	
interrupt	中断源
TIMER_INT_UP	更新中断 TIMERx (x=1,2,5,6,8,11 for GD32L233, x=0,1,2,5,6,8,11,14,40 for GD32L235)
TIMER_INT_CH0	通道0比较/捕获中断 TIMERx (x=1,2,8,11 for GD32L233, x=0,1,2,8,11,14,40 for GD32L235)
TIMER_INT_CH1	通道1比较/捕获中断 TIMERx (x=1,2,8,11 for GD32L233, x=0,1,2,8,11,14,40 for GD32L235)
TIMER_INT_CH2	通道2比较/捕获中断 TIMERx (x=1,2 for GD32L233, x=0,1,2 for GD32L235)
TIMER_INT_CH3	通道3比较/捕获中断 TIMERx (x=1,2 for GD32L233, x=0,1,2 for GD32L235)
TIMER_INT_CMT	换相更新中断 TIMERx (x=0,14,40 for GD32L235)
TIMER_INT_TRG	触发中断 TIMERx (x=1,2,8,11 for GD32L233, x=0,1,2,8,11,14,40 for GD32L235)
TIMER_INT_BRK	中止中断 TIMERx (x=0,14,40 for GD32L235)
输出参数{out}	
-	-

返回值	
-	-

例如:

```
/* enable the TIMER1 update interrupt */
```

```
timer_interrupt_enable(TIMER1, TIMER_INT_UP);
```

函数 timer_interrupt_disable

函数timer_interrupt_disable描述见下表:

表 3-823. 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	参考具体参数
输入参数{in}	
interrupt	中断源
TIMER_INT_UP	更新中断 TIMERx (x=1,2,5,6,8,11 for GD32L233, x=0,1,2,5,6,8,11,14,40 for GD32L235)
TIMER_INT_CH0	通道0比较/捕获中断 TIMERx (x=1,2,8,11 for GD32L233, x=0,1,2,8,11,14,40 for GD32L235)
TIMER_INT_CH1	通道1比较/捕获中断 TIMERx (x=1,2,8,11 for GD32L233, x=0,1,2,8,11,14,40 for GD32L235)
TIMER_INT_CH2	通道2比较/捕获中断 TIMERx (x=1,2 for GD32L233, x=0,1,2 for GD32L235)
TIMER_INT_CH3	通道3比较/捕获中断 TIMERx (x=1,2 for GD32L233, x=0,1,2 for GD32L235)
TIMER_INT_CMT	换相更新中断 TIMERx (x=0,14,40 for GD32L235)
TIMER_INT_TRG	触发中断 TIMERx (x=1,2,8,11 for GD32L233, x=0,1,2,8,11,14,40 for GD32L235)
TIMER_INT_BRK	中止中断 TIMERx (x=0,14,40 for GD32L235)
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* disable the TIMER1 update interrupt */
```

```
timer_interrupt_disable(TIMER1, TIMER_INT_UP);
```

函数 timer_interrupt_flag_get

函数timer_interrupt_flag_get描述见下表:

表 3-824. 函数 timer_interrupt_flag_get

函数名称	timer_interrupt_flag_get
函数原型	FlagStatus timer_interrupt_flag_get(uint32_t timer_periph, uint32_t interrupt);
功能描述	获取外设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	参考具体参数
输入参数{in}	
interrupt	中断源
TIMER_INT_FLAG_UP	更新中断 TIMERx (x=1,2,5,6,8,11 for GD32L233, x=0,1,2,5,6,8,11,14,40 for GD32L235)
TIMER_INT_FLAG_CH0	通道0比较/捕获中断 TIMERx (x=1,2,8,11 for GD32L233, x=0,1,2,8,11,14,40 for GD32L235)
TIMER_INT_FLAG_CH1	通道1比较/捕获中断 TIMERx (x=1,2,8,11 for GD32L233, x=0,1,2,8,11,14,40 for GD32L235)
TIMER_INT_FLAG_CH2	通道2比较/捕获中断 TIMERx (x=1,2 for GD32L233, x=0,1,2 for GD32L235)
TIMER_INT_FLAG_CH3	通道3比较/捕获中断 TIMERx (x=1,2 for GD32L233, x=0,1,2 for GD32L235)
TIMER_INT_FLAG_CMT	换相更新中断 TIMERx (x=0,14,40 for GD32L235)
TIMER_INT_FLAG_TRG	触发中断 TIMERx (x=1,2,8,11 for GD32L233, x=0,1,2,8,11,14,40 for GD32L235)
TIMER_INT_FLAG_BRK	中止中断 TIMERx (x=0,14,40 for GD32L235)
输出参数{out}	
-	-
返回值	
FlagStatus	SET或者RESET

例如:

```
/* get TIMER1 update interrupt flag */
```

```
FlagStatus Flag_interrupt = RESET;
```

```
Flag_interrupt = timer_interrupt_flag_get(TIMER1, TIMER_INT_FLAG_UP);
```

函数 timer_interrupt_flag_clear

函数timer_interrupt_flag_clear描述见下表：

表 3-825. 函数 timer_interrupt_flag_clear

函数名称	timer_interrupt_flag_clear
函数原型	void timer_interrupt_flag_clear(uint32_t timer_periph, uint32_t interrupt);
功能描述	清除外设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	参考具体参数
输入参数{in}	
interrupt	中断源
TIMER_INT_FLAG_UP	更新中断 TIMERx (x=1,2,5,6,8,11 for GD32L233, x=0,1,2,5,6,8,11,14,40 for GD32L235)
TIMER_INT_FLAG_CH0	通道0比较/捕获中断 TIMERx (x=1,2,8,11 for GD32L233, x=0,1,2,8,11,14,40 for GD32L235)
TIMER_INT_FLAG_CH1	通道1比较/捕获中断 TIMERx (x=1,2,8,11 for GD32L233, x=0,1,2,8,11,14,40 for GD32L235)
TIMER_INT_FLAG_CH2	通道2比较/捕获中断 TIMERx (x=1,2 for GD32L233, x=0,1,2 for GD32L235)
TIMER_INT_FLAG_CH3	通道3比较/捕获中断 TIMERx (x=1,2 for GD32L233, x=0,1,2 for GD32L235)
TIMER_INT_FLAG_CMT	换相更新中断 TIMERx (x=0,14,40 for GD32L235)
TIMER_INT_FLAG_TRG	触发中断 TIMERx (x=1,2,8,11 for GD32L233, x=0,1,2,8,11,14,40 for GD32L235)
TIMER_INT_FLAG_BRK	中止中断 TIMERx (x=0,14,40 for GD32L235)
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* clear TIMER1 update interrupt flag */
```

```
timer_interrupt_flag_clear(TIMER1, TIMER_INT_FLAG_UP);
```

3.26. 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[3.26.1](#)，TRNG固件库函数介绍在章节[3.26.2](#)。

3.26.1. 外设寄存器说明

TRNG寄存器列表如下表所示：

表 3-826. TRNG 寄存器

寄存器名称	寄存器描述
TRNG_CTL	TRNG控制寄存器
TRNG_STAT	TRNG状态寄存器
TRNG_DATA	TRNG数据寄存器

3.26.2. 外设库函数说明

TRNG库函数列表如下表所示：

表 3-827. TRNG 库函数

库函数名称	库函数描述
trng_deinit	复位TRNG
trng_enable	使能TRNG
trng_disable	禁能TRNG
trng_get_true_random_data	获取真随机值
trng_flag_get	获取TRNG状态标志
trng_interrupt_enable	使能TRNG中断
trng_interrupt_disable	禁能TRNG中断
trng_interrupt_flag_get	获取TRNG中断标志
trng_interrupt_flag_clear	清除TRNG中断标志

枚举 trng_flag_enum

表 3-828. 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 3-829. 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

成员名称	功能描述
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 3-830. 函数 trng_deinit

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

例如:

```
/* reset TRNG */
```

```
trng_deinit();
```

函数 trng_enable

函数trng_enable描述见下表:

表 3-831. 函数 trng_enable

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

例如:

```
/* enable TRNG */
```

```
trng_enable();
```

函数 trng_disable

函数trng_disable描述见下表:

表 3-832. 函数 trng_disable

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

例如:

```
/* disable TRNG */  
  
trng_disable();
```

函数 trng_get_true_random_data

函数trng_get_true_random_data描述见下表:

表 3-833. 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 true random data */  
  
uint32_t data;  
  
data = trng_get_true_random_data();
```

函数 `trng_flag_get`

函数 `trng_flag_get` 描述见下表:

表 3-834. 函数 `trng_flag_get`

函数名称	<code>trng_flag_get</code>
函数原形	<code>FlagStatus trng_flag_get(trng_flag_enum flag);</code>
功能描述	获取TRNG状态标志
先决条件	-
被调用函数	-
输入参数{in}	
flag	TRNG状态标志, 参阅 表3-828. 枚举trng_flag_enum
输出参数{out}	
-	-
返回值	
FlagStatus	SET或RESET

例如:

```
/* get TRNG clock error current flag status */
FlagStatus flag_status = RESET;
flag_status == trng_flag_get(TRNG_FLAG_CECS);
```

函数 `trng_interrupt_enable`

函数 `trng_interrupt_enable` 描述见下表:

表 3-835. 函数 `trng_interrupt_enable`

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

例如:

```
/* enable TRNG interrupt */
trng_interrupt_enable();
```

函数 **trng_interrupt_disable**

函数trng_interrupt_disable描述见下表：

表 3-836. 函数 trng_interrupt_disable

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

例如：

```
/* disable TRNG interrupt */
```

```
trng_interrupt_disable();
```

函数 **trng_interrupt_flag_get**

函数trng_interrupt_flag_get描述见下表：

表 3-837. 函数 trng_interrupt_flag_get

函数名称	trng_interrupt_flag_get
函数原形	FlagStatus trng_interrupt_flag_get(trng_int_flag_enum int_flag);
功能描述	获取TRNG中断标志
先决条件	-
被调用函数	-
输入参数{in}	
int_flag	TRNG中断标志，参阅 表3-829. 枚举trng_int_flag_enum
输出参数{out}	
-	-
返回值	
FlagStatus	SET或RESET

例如：

```
/* get TRNG clock error interrupt flag */
```

```
FlagStatus interrupt_flag = RESET;
```

```
interrupt_flag = trng_interrupt_flag_get(TRNG_INT_FLAG_CEIF);
```

函数 `trng_interrupt_flag_clear`

函数 `trng_interrupt_flag_clear` 描述见下表：

表 3-838. 函数 `trng_interrupt_flag_clear`

函数名称	<code>trng_interrupt_flag_clear</code>
函数原形	<code>void trng_interrupt_flag_clear(trng_int_flag_enum int_flag);</code>
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}	
<code>int_flag</code>	TRNG中断标志，参阅 表3-829. 枚举 <code>trng_int_flag_enum</code>
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* clear TRNG clock error interrupt flag */
trng_interrupt_flag_clear(TRNG_INT_FLAG_CEIF);
```

3.27. 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 [3.27.1](#) 描述了USART的寄存器列表，章节 [3.27.2](#) 对USART库函数进行说明。

3.27.1. 外设寄存器说明

USART寄存器列表如下表所示：

表 3-839. 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

寄存器名称	寄存器描述
USART_CHC	兼容性控制寄存器
USART_RFCS	接收FIFO控制和状态寄存器

3.27.2. 外设库函数说明

USART库函数列表如下表所示：

表 3-840. USART 库函数

库函数名称	库函数描述
usart_deinit	复位外设USART
usart_baudrate_set	配置USART波特率
usart_parity_config	配置USART奇偶校验
usart_word_length_set	配置USART字长
usart_stop_bit_set	配置USART停止位
usart_enable	使能USART
usart_disable	失能USART
usart_transmit_config	USART发送配置
usart_receive_config	USART接收配置
usart_data_first_config	配置数据传输时低位在前或高位在前
usart_invert_config	配置USART反转功能
usart_overrun_enable	使能USART溢出禁止功能
usart_overrun_disable	失能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	失能USART接收超时
usart_receiver_timeout_threshold_config	设置USART接收超时阈值
usart_data_transmit	USART发送数据功能
usart_data_receive	USART接收数据功能
usart_command_enable	使能USART请求
usart_address_config	配置USART地址
usart_address_detection_mode_config	配置USART地址检测模式
usart_mute_mode_enable	使能USART静默模式
usart_mute_mode_disable	失能USART静默模式
usart_mute_mode_wakeup_config	配置USART静默模式唤醒方式
usart_lin_mode_enable	使能USART LIN模式
usart_lin_mode_disable	失能USART LIN模式
usart_lin_break_detection_length_config	配置USART LIN模式中断帧长度
usart_halfduplex_enable	使能USART半双工模式

库函数名称	库函数描述
usart_halfduplex_disable	失能USART半双工模式
usart_clock_enable	使能USART CK引脚
usart_clock_disable	失能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	失能USART智能卡模式
usart_smartcard_mode_nack_enable	在USART智能卡模式下使能NACK
usart_smartcard_mode_nack_disable	在USART智能卡模式下失能NACK
usart_smartcard_mode_early_nack_enable	使能USART智能卡模式提前NACK
usart_smartcard_mode_early_nack_disable	失能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	失能USART串行红外编解码功能模块
usart_prescaler_config	在USART IrDA低功耗模式下配置外设时钟分频系数
usart_irda_lowpower_config	配置USART IrDA低功耗模式
usart_hardware_flow_rts_config	配置USART RTS硬件控制流
usart_hardware_flow_cts_config	配置USART CTS硬件控制流
usart_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	失能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	失能USART唤醒
usart_wakeup_mode_config	配置USART唤醒模式
usart_receive_fifo_enable	使能接收FIFO
usart_receive_fifo_disable	失能接收FIFO
usart_receive_fifo_counter_number	读取接收FIFO计数器的值
usart_flag_get	得到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

库函数名称	库函数描述
usart_interrupt_disable	失能USART中断
usart_interrupt_flag_get	得到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 3-841. 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_AM	地址匹配标志
USART_FLAG_BSY	忙标志
USART_FLAG_EB	块结束标志
USART_FLAG_RT	接收超时标志
USART_FLAG_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_TC	发送完成
USART_FLAG_RBNE	读数据缓冲区非空
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_RFFINT	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

枚举类型 `usart_interrupt_flag_enum`

表 3-842. 枚举类型 `usart_interrupt_flag_enum`

成员名称	功能描述
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_AM	地址匹配中断标志
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

成员名称	功能描述
USART_INT_FLAG_TC	发送完成中断标志
USART_INT_FLAG_RBNE	读缓冲区非空中断标志
USART_INT_FLAG_RBNE_ORE RR	读缓冲区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_RFF	接收FIFO满中断标志

枚举类型 `usart_interrupt_enum`

表 3-843. 枚举类型 `usart_interrupt_enum`

成员名称	功能描述
USART_INT_EB	块结束中断使能
USART_INT_RT	接收超时中断使能
USART_INT_AM	地址匹配中断使能
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_TC	发送完成中断使能
USART_INT_RBNE	读缓冲区非空中断和溢出错误中断使能
USART_INT_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_RFF	接收FIFO满中断使能

枚举类型 `usart_invert_enum`

表 3-844. 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

成员名称	功能描述
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 3-845. 函数 usart_deinit

函数名称	usart_deinit
函数原型	void usart_deinit(uint32_t usart_periph);
功能描述	复位外设USARTx/UARTx
先决条件	-
被调用函数	rcu_periph_reset_enable / rcu_periph_reset_disable
输入参数{in}	
usart_periph	外设USARTx/UARTx
USARTx	x=0,1
UARTx	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 3-846. 函数 usart_baudrate_set

函数名称	usart_baudrate_set
函数原型	void usart_baudrate_set(uint32_t usart_periph, uint32_t baudval);
功能描述	配置USART波特率
先决条件	-
被调用函数	rcu_clock_freq_get
输入参数{in}	
usart_periph	外设USARTx/UARTx
USARTx	x=0,1
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 3-847. 函数 usart_parity_config

函数名称	usart_parity_config
函数原型	void usart_parity_config(uint32_t usart_periph, uint32_t paritycfg);
功能描述	配置USART奇偶校验
先决条件	-
被调用函数	-
输入参数{in}	
usart_periph	外设USARTx/UARTx
USARTx	x=0,1
UARTx	x=3,4
输入参数{in}	
paritycfg	配置USART奇偶校验
USART_PM_NONE	无校验
USART_PM_ODD	奇校验
USART_PM_EVEN	偶校验
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* configure USART0 parity */
```

```
usart_parity_config(USART0, USART_PM_EVEN);
```

函数 usart_word_length_set

函数usart_word_length_set描述见下表：

表 3-848. 函数 usart_word_length_set

函数名称	usart_word_length_set
函数原型	void usart_word_length_set(uint32_t usart_periph, uint32_t wlen);

功能描述	配置USART字长
先决条件	-
被调用函数	-
输入参数{in}	
usart_periph	外设USARTx/UARTx
<i>USARTx</i>	x=0,1
<i>UARTx</i>	x=3,4
输入参数{in}	
wlen	配置USART字长
<i>USART_WL_8BIT</i>	8 bits
<i>USART_WL_9BIT</i>	9 bits
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* configure USART0 word length */
```

```
usart_word_length_set(USART0, USART_WL_9BIT);
```

函数 usart_stop_bit_set

函数usart_stop_bit_set描述见下表：

表 3-849. 函数 usart_stop_bit_set

函数名称	usart_stop_bit_set
函数原型	void usart_stop_bit_set(uint32_t usart_periph, uint32_t stblen);
功能描述	配置USART停止位
先决条件	-
被调用函数	-
输入参数{in}	
usart_periph	外设USARTx/UARTx
<i>USARTx</i>	x=0,1
<i>UARTx</i>	x=3,4
输入参数{in}	
stblen	配置USART停止位
<i>USART_STB_1BIT</i>	1 bit
<i>USART_STB_0_5BIT</i>	0.5 bit
<i>T</i>	
<i>USART_STB_2BIT</i>	2 bit
<i>USART_STB_1_5BIT</i>	1.5 bit
<i>T</i>	
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 3-850. 函数 usart_enable

函数名称	usart_enable
函数原型	void usart_enable(uint32_t usart_periph);
功能描述	使能USART
先决条件	-
被调用函数	-
输入参数{in}	
usart_periph	外设USARTx/UARTx
USARTx	x=0,1
UARTx	x=3,4
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* enable USART0 */
```

```
usart_enable(USART0);
```

函数 usart_disable

函数usart_disable描述见下表：

表 3-851. 函数 usart_disable

函数名称	usart_disable
函数原型	void usart_disable(uint32_t usart_periph);
功能描述	失能USART
先决条件	-
被调用函数	-
输入参数{in}	
usart_periph	外设USARTx/UARTx

USARTx	x=0,1
UARTx	x=3,4
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* disable USART0 */
```

```
usart_disable(USART0);
```

函数 usart_transmit_config

函数usart_transmit_config描述见下表:

表 3-852. 函数 usart_transmit_config

函数名称	usart_transmit_config
函数原型	void usart_transmit_config(uint32_t usart_periph, uint32_t txconfig);
功能描述	USART发送器配置
先决条件	-
被调用函数	-
输入参数{in}	
usart_periph	外设USARTx/UARTx
USARTx	x=0,1
UARTx	x=3,4
输入参数{in}	
txconfig	使能/失能USART发送器
USART_TRANSMIT_ENABLE	使能USART发送
USART_TRANSMIT_DISABLE	失能USART发送
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* configure USART0 transmitter */
```

```
usart_transmit_config(USART0,USART_TRANSMIT_ENABLE);
```

函数 usart_receive_config

函数usart_receive_config描述见下表:

表 3-853. 函数 `usart_receive_config`

函数名称	<code>usart_receive_config</code>
函数原型	<code>void usart_receive_config(uint32_t usart_periph, uint32_t rxconfig);</code>
功能描述	USART接收器配置
先决条件	-
被调用函数	-
输入参数{in}	
<code>usart_periph</code>	外设USARTx/UARTx
<code>USARTx</code>	x=0,1
<code>UARTx</code>	x=3,4
输入参数{in}	
<code>rxconfig</code>	使能/失能USART接收器
<code>USART_RECEIVE_ENABLE</code>	使能USART接收
<code>USART_RECEIVE_DISABLE</code>	失能USART接收
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* configure USART0 receiver */
```

```
usart_receive_config(USART0, USART_RECEIVE_ENABLE);
```

函数 `usart_data_first_config`

函数`usart_data_first_config`描述见下表:

表 3-854. 函数 `usart_data_first_config`

函数名称	<code>usart_data_first_config</code>
函数原型	<code>void usart_data_first_config(uint32_t usart_periph, uint32_t msbf);</code>
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}	
<code>usart_periph</code>	外设USARTx/UARTx
<code>USARTx</code>	x=0,1
<code>UARTx</code>	x=3,4
输入参数{in}	
<code>msbf</code>	数据传输时低位在前/高位在前
<code>USART_MSBF_LSB</code>	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 3-855. 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/UARTx
USARTx	x=0,1
UARTx	x=3,4
输入参数{in}	
invertpara	参考 表3-844. 枚举类型usart_invert_enum
USART_DINV_ENABLE	数据位电平反转
USART_DINV_DISABLE	数据位电平不反转
USART_TXPIN_ENABLE	TX引脚电平反转
USART_TXPIN_DISABLE	TX引脚电平不反转
USART_RXPIN_ENABLE	RX引脚电平反转
USART_RXPIN_DISABLE	RX引脚电平不反转
USART_SWAP_ENABLE	TX和RX管脚功能被交换
USART_SWAP_DISABLE	TX和RX管脚功能不被交换

输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* configure USART0 inversion */
```

```
usart_invert_config(USART0, USART_DINV_ENABLE);
```

函数 usart_oversize_enable

函数usart_oversize_enable描述见下表：

表 3-856. 函数 usart_oversize_enable

函数名称	usart_oversize_enable
函数原型	void usart_oversize_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/UARTx
USARTx	x=0,1
UARTx	x=3,4
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* enable USART0 overrun */
```

```
usart_oversize_enable(USART0);
```

函数 usart_oversize_disable

函数usart_oversize_disable描述见下表：

表 3-857. 函数 usart_oversize_disable

函数名称	usart_oversize_disable
函数原型	void usart_oversize_disable(uint32_t usart_periph);
功能描述	失能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/UARTx
<i>USARTx</i>	x=0,1
<i>UARTx</i>	x=3,4
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* disable USART0 overrun */
usart_overrun_disable(USART0);
```

函数 usart_oversample_config

函数usart_oversample_config描述见下表:

表 3-858. 函数 usart_oversample_config

函数名称	usart_oversample_config
函数原型	void usart_oversample_config(uint32_t usart_periph, uint32_t oversamp);
功能描述	配置USART过采样模式
先决条件	-
被调用函数	-
输入参数{in}	
usart_periph	外设USARTx/UARTx
<i>USARTx</i>	x=0,1
<i>UARTx</i>	x=3,4
输入参数{in}	
oversamp	过采样值
<i>USART_OVSMOD_8</i>	8倍过采样
<i>USART_OVSMOD_16</i>	16倍过采样
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* config 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 3-859. 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/UARTx
USARTx	x=0,1
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 3-860. 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
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 3-861. 函数 usart_receiver_timeout_disable

函数名称	usart_receiver_timeout_disable
函数原型	void usart_receiver_timeout_disable(uint32_t usart_periph);
功能描述	失能USART接收超时
先决条件	-
被调用函数	-
输入参数{in}	
usart_periph	外设USARTx
USARTx	x=0,1
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 3-862. 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
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 3-863. 函数 usart_data_transmit

函数名称	usart_data_transmit
函数原型	void usart_data_transmit(uint32_t usart_periph, uint16_t data);
功能描述	USART发送数据功能
先决条件	-
被调用函数	-
输入参数{in}	
usart_periph	外设USARTx/UARTx
USARTx	x=0,1
UARTx	x=3,4
输入参数{in}	
data	发送的数据（0x00-0x1FF）
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 3-864. 函数 usart_data_receive

函数名称	usart_data_receive
函数原型	uint16_t usart_data_receive(uint32_t usart_periph);
功能描述	USART接收数据功能
先决条件	-
被调用函数	-

输入参数{in}	
usart_periph	外设USARTx/UARTx
<i>USARTx</i>	x=0,1
<i>UARTx</i>	x=3,4
输出参数{out}	
-	-
返回值	
uint16_t	接收到的数据（0x00-0x1FF）

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 3-865. 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/UARTx
<i>USARTx</i>	x=0,1
<i>UARTx</i>	x=3,4
输入参数{in}	
cmdtype	请求类型
<i>USART_CMD_SBK CMD</i>	发送断开帧请求
<i>USART_CMD_MM CMD</i>	静模式请求
<i>USART_CMD_RXF CMD</i>	接收数据清空请求
<i>USART_CMD_TXF 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_config

函数usart_address_config描述见下表:

表 3-866. 函数 usart_address_config

函数名称	usart_address_config
函数原型	void usart_address_config(uint32_t usart_periph, uint8_t addr);
功能描述	配置USART地址
先决条件	-
被调用函数	-
输入参数{in}	
usart_periph	外设USARTx/UARTx
USARTx	x=0,1
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 of the USART0 */
```

```
usart_address_config(USART0, 0x00);
```

函数 usart_address_detection_mode_config

函数usart_address_detection_mode_config描述见下表:

表 3-867. 函数 usart_address_detection_mode_config

函数名称	usart_address_detection_mode_config
函数原型	void usart_address_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/UARTx
USARTx	x=0,1

<i>USARTx</i>	<i>x=3,4</i>
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 detection mode */
usart_address_config(USART0, USART_ADDDM_4BIT);
```

函数 usart_mute_mode_enable

函数usart_mute_mode_enable描述见下表:

表 3-868. 函数 usart_mute_mode_enable

函数名称	usart_mute_mode_enable
函数原型	void usart_mute_mode_enable(uint32_t usart_periph);
功能描述	使能USART静默模式
先决条件	-
被调用函数	-
输入参数{in}	
usart_periph	外设USARTx/UARTx
<i>USARTx</i>	<i>x=0,1</i>
<i>UARTx</i>	<i>x=3,4</i>
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 3-869. 函数 usart_mute_mode_disable

函数名称	usart_mute_mode_disable
函数原型	void usart_mute_mode_disable(uint32_t usart_periph);
功能描述	失能USART静默模式
先决条件	-
被调用函数	-
输入参数{in}	
usart_periph	外设USARTx/UARTx
USARTx	x=0,1
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 3-870. 函数 usart_mute_mode_wakeup_config

函数名称	usart_mute_mode_wakeup_config
函数原型	void usart_mute_mode_wakeup_config(uint32_t usart_periph, uint32_t wmethod);
功能描述	配置USART静默模式唤醒方式
先决条件	-
被调用函数	-
输入参数{in}	
usart_periph	外设USARTx/UARTx
USARTx	x=0,1
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 3-871. 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
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* USART0 LIN mode enable */  
  
usart_lin_mode_enable(USART0);
```

函数 usart_lin_mode_disable

函数usart_lin_mode_disable描述见下表：

表 3-872. 函数 usart_lin_mode_disable

函数名称	usart_lin_mode_disable
函数原型	void usart_lin_mode_disable(uint32_t usart_periph);
功能描述	失能USART LIN模式
先决条件	-
被调用函数	-
输入参数{in}	
usart_periph	外设USARTx
USARTx	x=0,1
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* USART0 LIN mode disable */
usart_lin_mode_disable(USART0);
```

函数 usart_lin_break_dection_length_config

函数usart_lin_break_dection_length_config描述见下表：

表 3-873. 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
输入参数{in}	
lblen	LIN模式中断帧长度
USART_LBLEN_10 B	断开帧长度为10 bits
USART_LBLEN_11 B	断开帧长度为11 bits
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* configure LIN break frame length */
usart_lin_break_dection_length_config(USART0, USART_LBLEN_10B);
```

函数 usart_halfduplex_enable

函数usart_halfduplex_enable描述见下表：

表 3-874. 函数 usart_halfduplex_enable

函数名称	usart_halfduplex_enable
函数原型	void usart_halfduplex_enable(uint32_t usart_periph);
功能描述	使能USART半双工模式
先决条件	-
被调用函数	-

输入参数{in}	
usart_periph	外设USARTx/UARTx
<i>USARTx</i>	x=0,1
<i>UARTx</i>	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 3-875. 函数 usart_halfduplex_disable

函数名称	usart_halfduplex_disable
函数原型	void usart_halfduplex_disable(uint32_t usart_periph);
功能描述	失能USART半双工模式
先决条件	-
被调用函数	-
输入参数{in}	
usart_periph	外设USARTx/UARTx
<i>USARTx</i>	x=0,1
<i>UARTx</i>	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 3-876. 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
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 3-877. 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	失能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
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 3-878. 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
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 3-879. 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
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 3-880. 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
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* USART0 smartcard mode enable */
```

```
usart_smartcard_mode_enable(USART0);
```

函数 usart_smartcard_mode_disable

函数usart_smartcard_mode_disable描述见下表:

表 3-881. 函数 usart_smartcard_mode_disable

函数名称	usart_smartcard_mode_disable
函数原型	void usart_smartcard_mode_disable(uint32_t usart_periph);
功能描述	失能USART智能卡模式
先决条件	-

被调用函数	-
输入参数{in}	
usart_periph	外设USARTx
USARTx	x=0,1
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* USART0 smartcard mode disable */
usart_smartcard_mode_disable(USART0);
```

函数 usart_smartcard_mode_nack_enable

函数usart_smartcard_mode_nack_enable描述见下表：

表 3-882. 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
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 3-883. 函数 usart_smartcard_mode_nack_disable

函数名称	usart_smartcard_mode_nack_disable
函数原型	void usart_smartcard_mode_nack_disable(uint32_t usart_periph);
功能描述	在USART智能卡模式下失能NACK

先决条件	-
被调用函数	-
输入参数{in}	
usart_periph	外设USARTx
USARTx	x=0,1
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 3-884. 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
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 3-885. 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	失能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
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 3-886. 函数 usart_smartcard_autoretry_config

函数名称	usart_smartcard_autoretry_config
函数原型	void usart_smartcard_autoretry_config(uint32_t usart_periph, uint32_t scrtnum);
功能描述	配置智能卡自动重试次数
先决条件	-
被调用函数	-
输入参数{in}	
usart_periph	外设USARTx
USARTx	x=0,1
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 3-887. 函数 `usart_block_length_config`

函数名称	<code>usart_block_length_config</code>
函数原型	<code>void usart_block_length_config(uint32_t usart_periph, uint32_t bl);</code>
功能描述	配置智能卡T=1的接收时块的长度
先决条件	-
被调用函数	-
输入参数{in}	
<code>usart_periph</code>	外设USARTx
<code>USARTx</code>	x=0,1
输入参数{in}	
<code>bl</code>	块长度（0x00-0x000000FF）
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* configure block length in Smartcard T=1 reception */
```

```
usart_block_length_config(USART0, 0x000000FF);
```

函数 `usart_irda_mode_enable`

函数`usart_irda_mode_enable`描述见下表：

表 3-888. 函数 `usart_irda_mode_enable`

函数名称	<code>usart_irda_mode_enable</code>
函数原型	<code>void usart_irda_mode_enable(uint32_t usart_periph);</code>
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}	
<code>usart_periph</code>	外设USARTx
<code>USARTx</code>	x=0,1
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* enable USART0 IrDA mode */
```

```
usart_irda_mode_enable(USART0);
```

函数 usart_irda_mode_disable

函数usart_irda_mode_disable描述见下表：

表 3-889. 函数 usart_irda_mode_disable

函数名称	usart_irda_mode_disable
函数原型	void usart_irda_mode_disable(uint32_t usart_periph);
功能描述	失能USART串行红外编解码功能模块
先决条件	-
被调用函数	-
输入参数{in}	
usart_periph	外设USARTx
USARTx	x=0,1
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* disable USART0 IrDA mode */
usart_irda_mode_disable(USART0);
```

函数 usart_prescaler_config

函数usart_prescaler_config描述见下表：

表 3-890. 函数 usart_prescaler_config

函数名称	usart_prescaler_config
函数原型	void usart_prescaler_config(uint32_t usart_periph, 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
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 3-891. 函数 usart_irda_lowpower_config

函数名称	usart_irda_lowpower_config
函数原型	void usart_irda_lowpower_config(uint32_t usart_periph, uint32_t irlp);
功能描述	配置USART IrDA低功耗模式
先决条件	-
被调用函数	-
输入参数{in}	
usart_periph	外设USARTx
USARTx	x=0,1
输入参数{in}	
irlp	IrDA低功耗模式或正常模式
USART_IRLP_LOW	低功耗模式
USART_IRLP_NORMAL	正常模式
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* configure USART0 IrDA low-power */
```

```
usart_irda_lowpower_config(USART0, USART_IRLP_LOW);
```

函数 usart_hardware_flow_rts_config

函数usart_hardware_flow_rts_config描述见下表:

表 3-892. 函数 usart_hardware_flow_rts_config

函数名称	usart_hardware_flow_rts_config
函数原型	void usart_hardware_flow_rts_config(uint32_t usart_periph, uint32_t rtsconfig);
功能描述	配置USART RTS硬件控制流
先决条件	-
被调用函数	-
输入参数{in}	
usart_periph	外设USARTx
USARTx	x=0,1
输入参数{in}	
rtsconfig	使能/失能RTS

<i>USART_RTS_ENA</i> <i>BLE</i>	使能RTS
<i>USART_RTS_DISA</i> <i>BLE</i>	失能RTS
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* configure USART0 hardware flow control RTS */
```

```
usart_hardware_flow_rts_config(USART0, USART_RTS_ENABLE);
```

函数 usart_hardware_flow_cts_config

函数usart_hardware_flow_cts_config描述见下表：

表 3-893. 函数 usart_hardware_flow_cts_config

函数名称	usart_hardware_flow_cts_config
函数原型	void usart_hardware_flow_cts_config(uint32_t usart_periph, uint32_t ctsconfig);
功能描述	配置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
<i>USARTx</i>	x=0,1
输入参数{in}	
ctsconfig	使能/失能CTS
<i>USART_CTS_ENA</i> <i>BLE</i>	使能CTS
<i>USART_CTS_DISA</i> <i>BLE</i>	失能CTS
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* configure USART0 hardware flow control CTS */
```

```
usart_hardware_flow_cts_config(USART0, USART_CTS_ENABLE);
```

函数 **usart_hardware_flow_coherence_config**

函数usart_hardware_flow_coherence_config描述见下表:

表 3-894. 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
USARTx	x=0,1
输入参数{in}	
hcm	硬件流控制兼容模式
USART_HCM_NONE	nRTS信号与USART_STAT0寄存器中RBNE位相同
USART_HCM_EN	nRTS信号在最后一个数据位被采样后被置位
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* configure 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 3-895. 函数 **usart_rs485_driver_enable**

函数名称	usart_rs485_driver_enable
函数原型	void usart_rs485_driver_enable(uint32_t usart_periph);
功能描述	使能USART rs485驱动
先决条件	-
被调用函数	-
输入参数{in}	
usart_periph	外设USARTx
USARTx	x=0,1
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 3-896. 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	失能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
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 3-897. 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
输入参数{in}	
deatime	驱动使能置位时间 (0x00-0x0000001F)

输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* set USART0 driver assertime */
```

```
usart_driver_assertime_config(USART0, 0x0000001F);
```

函数 usart_driver_deassertime_config

函数usart_driver_deassertime_config描述见下表：

表 3-898. 函数 usart_driver_deassertime_config

函数名称	usart_driver_deassertime_config
函数原型	void usart_driver_deassertime_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
输入参数{in}	
dedtime	驱动使能置低时间（0x00-0x0000001F）
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* set USART0 driver deassertime */
```

```
usart_driver_deassertime_config(USART0, 0x0000001F);
```

函数 usart_depolarity_config

函数usart_depolarity_config描述见下表：

表 3-899. 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
<i>USARTx</i>	x=0,1
输入参数{in}	
dep	驱动使能的极性选择模式
<i>USART_DEP_HIGH</i>	DE信号高有效
<i>USART_DEP_LOW</i>	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 3-900. 函数 usart_dma_receive_config

函数名称	usart_dma_receive_config
函数原型	void usart_dma_receive_config(uint32_t usart_periph, uint32_t dmacmd);
功能描述	配置USART DMA接收功能
先决条件	-
被调用函数	-
输入参数{in}	
usart_periph	外设USARTx/UARTx
<i>USARTx</i>	x=0,1
<i>UARTx</i>	x=3,4
输入参数{in}	
dmacmd	DMA使能/失能DMA接收功能
<i>USART_RECEIVE_DMA_ENABLE</i>	使能DMA接收功能
<i>USART_RECEIVE_DMA_DISABLE</i>	失能DMA接收功能
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* USART0 DMA enable for reception */
```

```
usart_dma_receive_config(USART0, USART_RECEIVE_DMA_ENABLE);
```

函数 usart_dma_transmit_config

函数usart_dma_transmit_config描述见下表：

表 3-901. 函数 usart_dma_transmit_config

函数名称	usart_dma_transmit_config
函数原型	void usart_dma_transmit_config(uint32_t usart_periph, uint32_t dmacmd);
功能描述	配置USART DMA发送功能
先决条件	-
被调用函数	-
输入参数{in}	
usart_periph	外设USARTx/UARTx
USARTx	x=0,1
UARTx	x=3,4
输入参数{in}	
dmacmd	使能/失能DMA发送功能
USART_TRANSMIT_DMA_ENABLE	使能DMA发送功能
USART_TRANSMIT_DMA_DISABLE	失能DMA发送功能
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* USART0 DMA enable for transmission */
```

```
usart_dma_transmit_config(USART0, USART_TRANSMIT_DMA_ENABLE);
```

函数 usart_reception_error_dma_disable

函数usart_reception_error_dma_disable描述见下表：

表 3-902. 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接收错误时失能DMA
先决条件	-
被调用函数	-
输入参数{in}	
usart_periph	外设USARTx/UARTx
USARTx	x=0,1

<i>USARTx</i>	<i>x=3,4</i>
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 3-903. 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/UARTx
<i>USARTx</i>	<i>x=0,1</i>
<i>UARTx</i>	<i>x=3,4</i>
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 3-904. 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
<i>USARTx</i>	x=0,1
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* USART0 wake up enable */
```

```
usart_wakeup_enable(USART0);
```

函数 usart_wakeup_disable

函数usart_wakeup_disable描述见下表:

表 3-905. 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	失能USART唤醒
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
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* USART0 wake up disable */
```

```
usart_wakeup_disable(USART0);
```

函数 usart_wakeup_mode_config

函数usart_wakeup_mode_config描述见下表:

表 3-906. 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
输入参数{in}	
wum	唤醒模式
USART_WUM_ADD R	WUF在地址匹配时置位
USART_WUM_STA RTB	WUF在检测到起始位时置位
USART_WUM_RBN E	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_receive_fifo_enable

函数usart_receive_fifo_enable描述见下表:

表 3-907. 函数 usart_receive_fifo_enable

函数名称	usart_receive_fifo_enable
函数原型	void usart_receive_fifo_enable(uint32_t usart_periph);
功能描述	使能接收FIFO
先决条件	-
被调用函数	-
输入参数{in}	
usart_periph	外设USARTx/UARTx
USARTx	x=0,1
UARTx	x=3,4
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* enable receive FIFO */
usart_receive_fifo_enable(USART0);
```

函数 usart_receive_fifo_disable

函数usart_receive_fifo_disable描述见下表：

表 3-908. 函数 usart_receive_fifo_disable

函数名称	usart_receive_fifo_disable
函数原型	void usart_receive_fifo_disable(uint32_t usart_periph);
功能描述	失能接收FIFO
先决条件	-
被调用函数	-
输入参数{in}	
usart_periph	外设USARTx/UARTx
USARTx	x=0,1
UARTx	x=3,4
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* disable receive FIFO */
usart_receive_fifo_disable(USART0);
```

函数 usart_receive_fifo_counter_number

函数usart_receive_fifo_counter_number描述见下表：

表 3-909. 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/UARTx
USARTx	x=0,1
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 3-910. 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/RFCS寄存器标志位
先决条件	-
被调用函数	-
输入参数{in}	
usart_periph	外设USARTx/UARTx
USARTx	x=0,1
UARTx	x=3,4
输入参数{in}	
flag	USART标志位，参考 表3-841. 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_ORERR	溢出错误标志
USART_FLAG_IDLE	空闲线检测标志
USART_FLAG_RBNE	读数据缓冲区非空标志
USART_FLAG_TC	发送完成标志
USART_FLAG_TBE	发送数据缓冲区空标志
USART_FLAG_LBD	LIN断开检测标志
USART_FLAG_CTSF	CTS变化标志
USART_FLAG_CTS	CTS电平
USART_FLAG_RT	接收超时标志
USART_FLAG_EB	块结束标志
USART_FLAG_BSY	忙状态标志
USART_FLAG_AM	ADDR匹配标志
USART_FLAG_SB	断开信号发送标识

USART_FLAG_RWU	接收器从静默模式唤醒
USART_FLAG_WU	从深度睡眠模式唤醒标志
USART_FLAG_TE	发送使能通知标志
USART_FLAG_REA	接收使能通知标志
USART_FLAG_EPERR	校验错误超前检测标志
USART_FLAG_RFE	接收FIFO空标志
USART_FLAG_RFF	接收FIFO满标志
USART_FLAG_RFFINT	接收FIFO满中断标志
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 3-911. 函数 usart_flag_clear

函数名称	usart_flag_clear
函数原型	void usart_flag_clear(uint32_t usart_periph, usart_flag_enum flag);
功能描述	清除USART状态寄存器标志位
先决条件	-
被调用函数	-
输入参数{in}	
usart_periph	外设USARTx/UARTx
USARTx	x=0,1
UARTx	x=3,4
输入参数{in}	
flag	USART标志位，参考 表3-841. 枚举类型usart_flag_enum 只能选择一个参数
USART_FLAG_PERR	校验错误标志
USART_FLAG_FER	帧错误标志

<i>R</i>	
USART_FLAG_NE RR	噪声错误标志
USART_FLAG_OR ERR	溢出错误标志
USART_FLAG_IDL E	空闲线检测标志
USART_FLAG_TC	发送完成标志
USART_FLAG_LBD	LIN断开检测标志
USART_FLAG_CTS F	CTS变化标志
USART_FLAG_RT	接收超时标志
USART_FLAG_EB	块结束标志
USART_FLAG_AM	ADDR匹配标志
USART_FLAG_WU	从深度睡眠模式唤醒标志
USART_FLAG_EPE RR	校验错误超前检测标志
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 3-912. 函数 usart_interrupt_enable

函数名称	usart_interrupt_enable
函数原型	void usart_interrupt_enable(uint32_t usart_periph, usart_interrupt_enum interrupt);
功能描述	使能USART中断
先决条件	-
被调用函数	-
输入参数{in}	
usart_periph	外设USARTx/UARTx
USARTx	x=0,1
UARTx	x=3,4
输入参数{in}	
interrupt	USART中断USART标志位, 参考 表3-843. 枚举类型usart_interrupt_enum

	只能选择一个参数
USART_INT_IDLE	IDLE线检测中断
USART_INT_RBNE	读数据缓冲区非空中断和过载错误中断
USART_INT_TC	发送完成中断
USART_INT_TBE	发送缓冲区空中断
USART_INT_PERR	校验错误中断
USART_INT_AM	ADDR匹配中断
USART_INT_RT	接收超时事件中断
USART_INT_EB	块结束事件中断
USART_INT_LBD	LIN断开信号检测中断
USART_INT_ERR	错误中断
USART_INT_CTS	CTS中断
USART_INT_WU	从深度睡眠模式唤醒中断
USART_INT_RFF	接收FIFO满中断
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 3-913. 函数 usart_interrupt_disable

函数名称	usart_interrupt_disable
函数原型	void usart_interrupt_disable(uint32_t usart_periph, usart_interrupt_enum interrupt);
功能描述	失能USART中断
先决条件	-
被调用函数	-
输入参数{in}	
usart_periph	外设USARTx/UARTx
USARTx	x=0,1
UARTx	x=3,4
输入参数{in}	
interrupt	USART中断USART标志位，参考 表3-843. 枚举类型usart_interrupt_enum 只能选择一个参数
USART_INT_IDLE	IDLE线检测中断
USART_INT_RBNE	读数据缓冲区非空中断和过载错误中断

USART_INT_TC	发送完成中断
USART_INT_TBE	发送缓冲区空中断
USART_INT_PERR	校验错误中断
USART_INT_AM	ADDR匹配中断
USART_INT_RT	接收超时事件中断
USART_INT_EB	块结束事件中断
USART_INT_LBD	LIN断开信号检测中断
USART_INT_ERR	错误中断
USART_INT_CTS	CTS中断
USART_INT_WU	从深度睡眠模式唤醒中断
USART_INT_RFF	接收FIFO满中断
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 3-914. 函数 usart_interrupt_flag_get

函数名称	usart_interrupt_flag_get
函数原型	FlagStatus usart_interrupt_flag_get(uint32_t usart_periph, usart_interrupt_flag_enum int_flag);
功能描述	获取USART中断标志位状态
先决条件	-
被调用函数	-
输入参数{in}	
usart_periph	外设USARTx/UARTx
USARTx	x=0,1
UARTx	x=3,4
输入参数{in}	
int_flag	USART中断标志，参考 表3-842. 枚举类型usart_interrupt_flag_enum 只能选择一个参数
USART_INT_FLAG_EB	块结束事件中断标志
USART_INT_FLAG_RT	超时事件中断标志
USART_INT_FLAG	ADDR匹配中断标志

<code>_AM</code>	
<code>USART_INT_FLAG_PERR</code>	校验错误中断标志
<code>USART_INT_FLAG_TBE</code>	发送缓冲区空中断标志
<code>USART_INT_FLAG_TC</code>	发送完成中断标志
<code>USART_INT_FLAG_RBNE</code>	读数据缓冲区非空中断标志
<code>USART_INT_FLAG_RBNE_ORERR</code>	读数据缓冲区非空中断和溢出错误中断标志
<code>USART_INT_FLAG_IDLE</code>	IDLE线检测中断标志
<code>USART_INT_FLAG_LBD</code>	LIN断开检测中断标志
<code>USART_INT_FLAG_WU</code>	从深度睡眠模式唤醒中断标志
<code>USART_INT_FLAG_CTS</code>	CTS中断标志
<code>USART_INT_FLAG_ERR_NERR</code>	噪声错误中断标志
<code>USART_INT_FLAG_ERR_ORERR</code>	过载错误中断标志
<code>USART_INT_FLAG_ERR_FERR</code>	帧错误中断标志
<code>USART_INT_FLAG_RFF</code>	接收FIFO满中断标志
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 3-915. 函数 `usart_interrupt_flag_clear`

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

函数原型	void usart_interrupt_flag_clear(uint32_t usart_periph, usart_interrupt_flag_enum int_flag);
功能描述	清除USART中断标志位状态
先决条件	-
被调用函数	-
输入参数{in}	
usart_periph	外设USARTx/UARTx
USARTx	x=0,1
UARTx	x=3,4
输入参数{in}	
int_flag	USART中断标志，参考 表3-842. 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 _AM	ADDR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 _RFF	接收FIFO满中断标志
输出参数{out}	
-	-
返回值	

例如：

```
/* clear the USART0 interrupt flag */
```

```
usart_interrupt_flag_clear(USART0, USART_INT_FLAG_TC);
```

3.28. VREF

VREF用于为ADC/DAC提供基准电压，或由连接到VREF引脚的片外电路使用。章节[3.28.1](#)描述了VREF的寄存器列表，章节[3.28.2](#)对VREF库函数进行说明。

3.28.1. 外设寄存器说明

VREF寄存器列表如下表所示：

表 3-916. 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

3.28.2. 外设库函数说明

VREF库函数列表如下表所示：

表 3-917. VREF 库函数

库函数名称	库函数说明
vref_deinit	将VREF寄存器重设为缺省值
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_voltage_select	VREF参考电压选择（只适用于L235xx系列）
vref_status_get	获取VREF状态
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 3-918. 函数 vref_deinit

函数名称	vref_deinit
函数原型	void vref_deinit(void);
功能描述	将VREF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：

```
/* deinitialize the VREF */
```

```
vref_deinit();
```

函数 vref_enable

函数vref_enable描述见下表：

表 3-919. 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 3-920. 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_disablet();
```

函数 vref_high_impedance_mode_enable

函数vref_high_impedance_mode_enable描述见下表：

表 3-921. 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 3-922. 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_voltage_select（只适用于 L235xx 系列）

函数vref_voltage_select描述见下表：

表 3-923. 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	VREF参考电压选择
先决条件	-
被调用函数	-
输入参数{in}	
vref_voltage	参考电压
VREF_VOLTAGE_SEL_LEVEL_0	2.048V
VREF_VOLTAGE_SEL_LEVEL_1	2.5V
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* VREF voltage reference select 2.5V */
```

```
vref_voltage_select(VREF_VOLTAGE_SEL_LEVEL_1);
```

函数 vref_status_get

函数vref_status_get描述见下表：

表 3-924. 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_calib_value_set

函数vref_calib_value_set描述见下表：

表 3-925. 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}	
-	-
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 3-926. 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();
```

3.29. WWDGT

窗口看门狗定时器(WWDGT)用来监测由软件故障导致的系统故障。章节[3.29.1](#)描述了WWDGT的寄存器列表，章节[3.29.2](#)对WWDGT库函数进行说明。

3.29.1. 外设寄存器说明

WWDGT寄存器列表如下表所示：

表 3-927. WWDGT 寄存器

寄存器名称	寄存器描述
WWDGT_CTL	控制寄存器
WWDGT_CFG	配置寄存器
WWDGT_STAT	状态寄存器

3.29.2. 外设库函数说明

WWDGT库函数列表如下表所示：

表 3-928. WWDGT 库函数

库函数名称	库函数说明
wwdgt_deinit	将WWDGT寄存器重设为缺省值
wwdgt_enable	使能WWDGT
wwdgt_counter_update	设置WWDGT计数器更新值
wwdgt_config	设置WWDGT计数器值、窗口值和预分频值
wwdgt_flag_get	检查WWDGT提前唤醒中断标志位是否置位
wwdgt_flag_clear	清除WWDGT提前唤醒中断标志位状态

库函数名称	库函数说明
wwdgt_interrupt_enable	使能WWDGT提前唤醒中断

函数 wwdgt_deinit

函数wwdgt_deinit描述见下表:

表 3-929. 函数 wwdgt_deinit

函数名称	wwdgt_deinit
函数原型	void wwdgt_deinit(void);
功能描述	将WWDGT寄存器重设为缺省值
先决条件	-
被调用函数	-
输入参数{in}	
-	-
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* reset the WWDGT configuration */
wwdgt_deinit();
```

函数 wwdgt_enable

函数wwdgt_enable描述见下表:

表 3-930. 函数 wwdgt_enable

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

例如:

```
/* start the WWDGT counter */
wwdgt_enable();
```

函数 **wwdgt_counter_update**

函数 **wwdgt_counter_update** 描述见下表：

表 3-931. 函数 **wwdgt_counter_update**

函数名称	wwdgt_counter_update
函数原型	void wwdgt_counter_update(uint16_t counter_value);
功能描述	设置WWDGT计数器更新值
先决条件	-
被调用函数	-
输入参数{in}	
counter_value	计数器值，数值范围为0x00000000 - 0x0000007F
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 3-932. 函数 **wwdgt_config**

函数名称	wwdgt_config
函数原型	void wwdgt_config(uint16_t counter, uint16_t window, uint32_t prescaler);
功能描述	设置WWDGT计数器值、窗口值和预分频值
先决条件	-
被调用函数	-
输入参数{in}	
counter	定时器计数值，数值范围0x00000000 - 0x0000007F
输入参数{in}	
window	窗口值，数值范围0x00000000 - 0x0000007F
输入参数{in}	
prescaler	WWDGT预分频值
WWDGT_CFG_PSC_DIV1	WWDGT计数器时钟为 (PCLK/4096) /1
WWDGT_CFG_PSC_DIV2	WWDGT计数器时钟为 (PCLK/4096) /2
WWDGT_CFG_PSC_DIV4	WWDGT计数器时钟为 (PCLK/4096) /4
WWDGT_CFG_PSC	WWDGT计数器时钟为 (PCLK/4096) /8

<code>_DIV8</code>	
<code>WWDGT_CFG_PSC_DIV16</code>	WWDGT计数器时钟为 (PCLK/4096) /16
<code>WWDGT_CFG_PSC_DIV32</code>	WWDGT计数器时钟为 (PCLK/4096) /32
<code>WWDGT_CFG_PSC_DIV64</code>	WWDGT计数器时钟为 (PCLK/4096) /64
<code>WWDGT_CFG_PSC_DIV128</code>	WWDGT计数器时钟为 (PCLK/4096) /128
<code>WWDGT_CFG_PSC_DIV256</code>	WWDGT计数器时钟为 (PCLK/4096) /256
<code>WWDGT_CFG_PSC_DIV512</code>	WWDGT计数器时钟为 (PCLK/4096) /512
<code>WWDGT_CFG_PSC_DIV1024</code>	WWDGT计数器时钟为 (PCLK/4096) /1024
<code>WWDGT_CFG_PSC_DIV2048</code>	WWDGT计数器时钟为 (PCLK/4096) /2048
<code>WWDGT_CFG_PSC_DIV4096</code>	WWDGT计数器时钟为 (PCLK/4096) /4096
<code>WWDGT_CFG_PSC_DIV8192</code>	WWDGT计数器时钟为 (PCLK/4096) /8192
输出参数{out}	
-	-
Return value	
-	-

例如:

```
/* configure WWDGT counter value to 0x7F, window value to 0x50, prescaler divider value to 8 */
```

```
wwdgt_config(127, 80, WWDGT_CFG_PSC_DIV8);
```

函数 `wwdgt_flag_get`

函数 `wwdgt_flag_get` 描述见下表:

表 3-933. 函数 `wwdgt_flag_get`

函数名称	<code>wwdgt_flag_get</code>
函数原型	<code>FlagStatus wwdgt_flag_get(void);</code>
功能描述	检查WWDGT提前唤醒中断标志位是否置位
先决条件	-
被调用函数	-
输入参数{in}	
-	-

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

例如：

```
/* test if the counter value update has reached the 0x40 */
```

```
FlagStatus status;
```

```
status = wwdgt_flag_get();
```

函数 wwdgt_flag_clear

函数wwdgt_flag_clear描述见下表：

表 3-934. 函数 wwdgt_flag_clear

函数名称	wwdgt_flag_clear
函数原型	void wwdgt_flag_clear(void);
功能描述	清除WWDGT提前唤醒中断标志位状态
先决条件	-
被调用函数	-
输入参数{in}	
-	-
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* clear early wakeup interrupt state of WWDGT */
```

```
wwdgt_flag_clear();
```

函数 wwdgt_interrupt_enable

函数wwdgt_interrupt_enable描述见下表：

表 3-935. 函数 wwdgt_interrupt_enable

函数名称	wwdgt_interrupt_enable
函数原型	void wwdgt_interrupt_enable(void);
功能描述	使能WWDGT提前唤醒中断
先决条件	-
被调用函数	-
输入参数{in}	
-	-

输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* enable early wakeup interrupt of WWDGT */
```

```
wwdgt_interrupt_enable();
```

4. 版本历史

表 4-1. 版本历史

版本号.	说明	日期
1.0	初稿发布	2021 年 8 月 25 日
1.1	调整文档格式排版	2022 年 7 月 8 号
1.2	修改部分函数入参保持与固件库代码一致	2022 年 12 月 21 日
1.3	<u>表 4-2. 函数 <code>rcu_rtc_clock_config</code></u> 中函数功能和入参的描述增加 SLCD	2023 年 6 月 21 日
2.0	新增支持 GD32L235xx 系列	2023 年 7 月 17 日
2.1	修改部分函数描述	2023 年 11 月 2 日
2.2	删除 GD32L235 系列 <code>pmu_lowepower_ldo_enable</code> 函数和 <code>pmu_lowepower_ldo_disable</code> 函数	2024 年 1 月 29 日
2.3	更新 3.11 章节 EXTI Line 枚举类型表格	2024 年 7 月 30 日
2.4	1. 增加 <code>rtc_lxtal_stab_reset_disable</code> <code>rtc_lxtal_stab_reset_enable</code> 函数 2. 删除 <code>i2c_nack_disable(uint32_t i2c_periph)</code> 函数	2025 年 8 月 8 号
2.5	1.删除 <code>spi_quad_io23_output_enable</code> 和 <code>spi_quad_io23_output_disable</code> 函数 2. L233 LPTIEMR 时钟源由 APB1 更新为 APB2, <code>rcu_lptimer_clock_config</code> 函数入参修改 为 <code>RCU_LPTIMERSRC_CKAPB2</code>	2026 年 2 月 2 号

Important Notice

This document is the property of GigaDevice Semiconductor Inc. and its subsidiaries (the "Company"). This document, including any product of the Company described in this document (the "Product"), is owned by the Company according to the laws of the People's Republic of China and other applicable laws. The Company reserves all rights under such laws and no Intellectual Property Rights are transferred (either wholly or partially) or licensed by the Company (either expressly or impliedly) herein. The names and brands of third party referred thereto (if any) are the property of their respective owner and referred to for identification purposes only.

To the maximum extent permitted by applicable law, the Company makes no representations or warranties of any kind, express or implied, with regard to the merchantability and the fitness for a particular purpose of the Product, nor does the Company assume any liability arising out of the application or use of any Product. Any information provided in this document is provided only for reference purposes. It is the sole responsibility of the user of this document to determine whether the Product is suitable and fit for its applications and products planned, and properly design, program, and test the functionality and safety of its applications and products planned using the Product. The Product is designed, developed, and/or manufactured for ordinary business, industrial, personal, and/or household applications only, and the Product is not designed or intended for use in (i) safety critical applications such as weapons systems, nuclear facilities, atomic energy controller, combustion controller, aeronautic or aerospace applications, traffic signal instruments, pollution control or hazardous substance management; (ii) life-support systems, other medical equipment or systems (including life support equipment and surgical implants); (iii) automotive applications or environments, including but not limited to applications for active and passive safety of automobiles (regardless of front market or aftermarket), for example, EPS, braking, ADAS (camera/fusion), EMS, TCU, BMS, BSG, TPMS, Airbag, Suspension, DMS, ICMS, Domain, ESC, DCDC, e-clutch, advanced-lighting, etc.. Automobile herein means a vehicle propelled by a self-contained motor, engine or the like, such as, without limitation, cars, trucks, motorcycles, electric cars, and other transportation devices; and/or (iv) other uses where the failure of the device or the Product can reasonably be expected to result in personal injury, death, or severe property or environmental damage (collectively "Unintended Uses"). Customers shall take any and all actions to ensure the Product meets the applicable laws and regulations. The Company is not liable for, in whole or in part, and customers shall hereby release the Company as well as its suppliers and/or distributors from, any claim, damage, or other liability arising from or related to all Unintended Uses of the Product. Customers shall indemnify and hold the Company, and its officers, employees, subsidiaries, affiliates as well as its suppliers and/or distributors harmless from and against all claims, costs, damages, and other liabilities, including claims for personal injury or death, arising from or related to any Unintended Uses of the Product.

Information in this document is provided solely in connection with the Product. The Company reserves the right to make changes, corrections, modifications or improvements to this document and the Product described herein at any time without notice. The Company shall have no responsibility whatsoever for conflicts or incompatibilities arising from future changes to them. Information in this document supersedes and replaces information previously supplied in any prior versions of this document.