

**GigaDevice Semiconductor Inc.**

**GD32M53x**

**Arm® Cortex®-M33 32-bit MCU**

**固件库  
使用指南**

1.0 版本

(2026 年 2 月)

# 目录

目录 .....	2
图索引 .....	5
表索引 .....	6
<b>1. 介绍 .....</b>	<b>29</b>
<b>1.1. 文档和固件库规则 .....</b>	<b>29</b>
1.1.1. 外设缩写 .....	29
1.1.2. 命名规则 .....	30
<b>2. 固件库概述 .....</b>	<b>31</b>
<b>2.1. 文件组织结构 .....</b>	<b>31</b>
2.1.1. Examples 文件夹 .....	32
2.1.2. Firmware 文件夹 .....	32
2.1.3. Template 文件夹 .....	32
2.1.4. Utilities 文件夹 .....	35
<b>2.2. 固件库文件描述 .....</b>	<b>35</b>
<b>3. 外设固件库 .....</b>	<b>36</b>
<b>3.1. 外设固件库概述 .....</b>	<b>36</b>
<b>3.2. ADC .....</b>	<b>36</b>
3.2.1. 外设寄存器描述 .....	36
3.2.2. 外设库函数说明 .....	38
<b>3.3. CAN .....</b>	<b>97</b>
3.3.1. 外设寄存器说明 .....	97
3.3.2. 外设库函数说明 .....	97
<b>3.4. CFMU .....</b>	<b>115</b>
3.4.1. 外设寄存器说明 .....	115
3.4.2. 外设库函数说明 .....	115
<b>3.5. CMP .....</b>	<b>125</b>
3.5.1. 外设寄存器说明 .....	125
3.5.2. 外设库函数说明 .....	126
<b>3.6. CPTIMER .....</b>	<b>138</b>
3.6.1. 外设寄存器说明 .....	138
3.6.2. 外设库函数说明 .....	139
<b>3.7. CPTIMERW .....</b>	<b>149</b>
3.7.1. 外设寄存器说明 .....	149

3.7.2.	外设库函数说明 .....	150
<b>3.8.</b>	<b>CRC.....</b>	<b>166</b>
3.8.1.	外设寄存器说明 .....	166
3.8.2.	外设库函数说明 .....	166
<b>3.9.</b>	<b>DAC .....</b>	<b>175</b>
3.9.1.	外设寄存器说明 .....	175
3.9.2.	外设库函数说明 .....	175
<b>3.10.</b>	<b>DBG .....</b>	<b>190</b>
3.10.1.	外设寄存器说明 .....	190
3.10.2.	外设库函数说明 .....	190
<b>3.11.</b>	<b>DMA&amp;DMAMUX.....</b>	<b>196</b>
3.11.1.	外设寄存器说明 .....	197
3.11.2.	外设库函数说明 .....	197
<b>3.12.</b>	<b>EVIC .....</b>	<b>246</b>
3.12.1.	外设寄存器说明 .....	247
3.12.2.	外设库函数说明 .....	248
<b>3.13.</b>	<b>EXTI.....</b>	<b>268</b>
3.13.1.	外设寄存器说明 .....	268
3.13.2.	外设库函数说明 .....	269
<b>3.14.</b>	<b>FMC .....</b>	<b>277</b>
3.14.1.	外设寄存器说明 .....	278
3.14.2.	外设库函数说明 .....	278
<b>3.15.</b>	<b>FWDGT.....</b>	<b>310</b>
3.15.1.	外设寄存器说明 .....	310
3.15.2.	外设库函数说明 .....	310
<b>3.16.</b>	<b>GPIO .....</b>	<b>317</b>
3.16.1.	外设寄存器说明 .....	317
3.16.2.	外设库函数说明 .....	317
<b>3.17.</b>	<b>GPTIMER .....</b>	<b>332</b>
3.17.1.	外设寄存器说明 .....	332
3.17.2.	外设库函数说明 .....	333
<b>3.18.</b>	<b>GTOC.....</b>	<b>416</b>
3.18.1.	外设寄存器说明 .....	416
3.18.2.	外设库函数说明 .....	416
<b>3.19.</b>	<b>I2C .....</b>	<b>432</b>
3.19.1.	外设寄存器说明 .....	432
3.19.2.	外设库函数说明 .....	432
<b>3.20.</b>	<b>MISC .....</b>	<b>469</b>

3.20.1.	外设寄存器说明 .....	469
3.20.2.	外设库函数说明 .....	470
<b>3.21.</b>	<b>PMU.....</b>	<b>477</b>
3.21.1.	外设寄存器说明 .....	477
3.21.2.	外设库函数说明 .....	477
<b>3.22.</b>	<b>POC.....</b>	<b>491</b>
3.22.1.	外设寄存器说明 .....	491
3.22.2.	外设库函数说明 .....	492
<b>3.23.</b>	<b>RCU .....</b>	<b>521</b>
3.23.1.	外设寄存器说明 .....	521
3.23.2.	外设库函数说明 .....	521
<b>3.24.</b>	<b>SPI .....</b>	<b>551</b>
3.24.1.	外设寄存器说明 .....	551
3.24.2.	外设库函数说明 .....	551
<b>3.25.</b>	<b>SVPWM .....</b>	<b>571</b>
3.25.1.	外设寄存器说明 .....	572
3.25.2.	外设库函数说明 .....	572
<b>3.26.</b>	<b>SYSCFG .....</b>	<b>577</b>
3.26.1.	外设寄存器说明 .....	577
3.26.2.	外设库函数说明 .....	578
<b>3.27.</b>	<b>TIMER.....</b>	<b>595</b>
3.27.1.	外设寄存器说明 .....	595
3.27.2.	外设库函数说明 .....	596
<b>3.28.</b>	<b>TMU.....</b>	<b>695</b>
3.28.1.	外设寄存器说明 .....	695
3.28.2.	外设库函数说明 .....	695
<b>3.29.</b>	<b>UART.....</b>	<b>709</b>
3.29.1.	外设寄存器说明 .....	709
3.29.2.	外设库函数说明 .....	710
<b>3.30.</b>	<b>WWDGT .....</b>	<b>734</b>
3.30.1.	外设寄存器说明 .....	734
3.30.2.	外设库函数说明 .....	734
<b>4.</b>	<b>版本历史.....</b>	<b>739</b>



## 图索引

图 2-1. GD32M53x 固件库文件组织结构 .....	31
图 2-2. 选择外设例程文件 .....	33
图 2-3. 拷贝外设例程文件 .....	33
图 2-4. 打开工程文件.....	34
图 2-5. 配置工程文件.....	34
图 2-6. 编译调试下载.....	34

# 表索引

表 1-1. 外设缩写 .....	29
表 2-1. 固件函数库文件描述 .....	35
表 3-1. 外设固件库函数描述格式 .....	36
表 3-2. ADC 寄存器 .....	36
表 3-3. ADC 库函数 .....	38
表 3-4. 枚举类型 adc_group_select_enum .....	40
表 3-5. 枚举类型 adc_channel_select_enum .....	40
表 3-6. 枚举类型 adc_bifurcate_data_enum .....	41
表 3-7. 枚举类型 adc_disc_detect_mode_enum .....	41
表 3-8. 枚举类型 adc_self_diag_mode_enum .....	41
表 3-9. 枚举类型 adc_self_diag_fixed_voltage_enum .....	41
表 3-10. 枚举类型 adc_restart_channel_enum .....	41
表 3-11. 枚举类型 adc_watchdog_select_enum .....	42
表 3-12. 枚举类型 adc_watchdog_compare_condition_enum .....	42
表 3-13. 枚举类型 adc_oversample_mode_enum .....	42
表 3-14. 枚举类型 adc_interrupt_enum .....	42
表 3-15. 枚举类型 adc_interrupt_flag_enum .....	43
表 3-16. 枚举类型 adc_event_flag_enum .....	43
表 3-17. 枚举类型 adc_evic_link_source_enum .....	44
表 3-18. 函数 adc_deinit .....	44
表 3-19. 函数 adc_enable .....	44
表 3-20. 函数 adc_disable .....	45
表 3-21. 函数 adc_data_alignment_config .....	45
表 3-22. 函数 adc_resolution_config .....	46
表 3-23. 函数 adc_self_diagnosis_enable .....	47
表 3-24. 函数 adc_self_diagnosis_disable .....	47
表 3-25. 函数 adc_self_diagnosis_mode_config .....	48
表 3-26. 函数 adc_disconnect_detect_mode_config .....	49
表 3-27. 函数 adc_disconnect_detect_period_config .....	49
表 3-28. 函数 adc_data_auto_clear_enable .....	50
表 3-29. 函数 adc_data_auto_clear_disable .....	50
表 3-30. 函数 adc_data_auto_set_enable .....	51
表 3-31. 函数 adc_data_auto_set_disable .....	51
表 3-32. 函数 adc_sample_hold_channel_config .....	52
表 3-33. 函数 adc_sh_sample_time_config .....	53
表 3-34. 函数 adc_sh_hold_time_config .....	53
表 3-35. 函数 adc_sh_constant_sampling_mode_enable .....	54
表 3-36. 函数 adc_sh_constant_sampling_mode_disable .....	54
表 3-37. 函数 adc_sh_constant_sampling_start .....	55

表 3-38. 函数 adc_sh_constant_sampling_stop .....	55
表 3-39. 函数 adc_watchdog_enable.....	56
表 3-40. 函数 adc_watchdog_disable.....	56
表 3-41. 函数 adc_watchdog_a_channel_config.....	57
表 3-42. 函数 adc_watchdog_a_channel_deselect.....	57
表 3-43. 函数 adc_watchdog_b_channel_config .....	58
表 3-44. 函数 adc_watchdog_a_threshold_config.....	59
表 3-45. 函数 adc_watchdog_b_threshold_config .....	59
表 3-46. 函数 adc_watchdog_window_mode_enable .....	60
表 3-47. 函数 adc_watchdog_window_mode_disable .....	61
表 3-48. 函数 adc_watchdog_complex_condition_config.....	61
表 3-49. 函数 adc_oversample_channel_config .....	62
表 3-50. 函数 adc_oversample_mode_config.....	63
表 3-51. 函数 adc_oversample_mode_enable .....	64
表 3-52. 函数 adc_oversample_mode_disable .....	64
表 3-53. 函数 adc_group_scan_mode_config .....	65
表 3-54. 函数 adc_group_priority_control_enable .....	65
表 3-55. 函数 adc_group_priority_control_disable .....	66
表 3-56. 函数 adc_group_pri3_enable.....	66
表 3-57. 函数 adc_group_pri3_disable .....	67
表 3-58. 函数 adc_group_pri4_enable.....	67
表 3-59. 函数 adc_group_pri4_disable .....	68
表 3-60. 函数 adc_group_lowest_priority_continuous_enable .....	68
表 3-61. 函数 adc_group_lowest_priority_continuous_disable .....	69
表 3-62. 函数 adc_group_restart_enable.....	70
表 3-63. 函数 adc_group_restart_disable.....	70
表 3-64. 函数 adc_group_channel_config.....	71
表 3-65. 函数 adc_group_channel_deselect.....	71
表 3-66. 函数 adc_group_end_flag_round_config.....	72
表 3-67. 函数 adc_group_external_trigger_enable.....	73
表 3-68. 函数 adc_group_extern_trigger_edge_config.....	73
表 3-69. 函数 adc_group_external_trigger_disable.....	74
表 3-70. 函数 adc_group_synchronous_trigger_enable.....	75
表 3-71. 函数 adc_group_synchronous_trigger_source_config.....	75
表 3-72. 函数 adc_group_asynchronous_trigger_enable .....	78
表 3-73. 函数 adc_group_software_trigger_enable.....	78
表 3-74. 函数 adc_group_software_end_conversion.....	79
表 3-75. 函数 adc_group_bifurcate_mode_enable .....	80
表 3-76. 函数 adc_group_bifurcate_mode_disable .....	80
表 3-77. 函数 adc_group_bifurcate_channel_select .....	81
表 3-78. 函数 adc_group_bifurcate_extend_trigger_select .....	81
表 3-79. 函数 adc_group_bifurcate_trigger_restart_enable .....	82
表 3-80. 函数 adc_group_bifurcate_trigger_restart_disable .....	83

表 3-81. 函数 adc_group_dma_mode_enable .....	84
表 3-82. 函数 adc_group_dma_mode_disable .....	84
表 3-83. 函数 adc_group_dma_request_after_last_enable .....	85
表 3-84. 函数 adc_group_dma_request_after_last_disable .....	85
表 3-85. 函数 adc_group_dma_overrun_detect_enable .....	86
表 3-86. 函数 adc_group_dma_overrun_detect_disable .....	87
表 3-87. 函数 adc_channel_priority_config .....	87
表 3-88. 函数 adc_channel_sample_time_config.....	88
表 3-89. 函数 adc_evic_link_signal_event_config .....	89
表 3-90. 函数 adc_group_evic_link_signal_source .....	89
表 3-91. 函数 adc_channel_data_read .....	90
表 3-92. 函数 adc_self_diagnosis_data_read .....	91
表 3-93. 函数 adc_self_diagnosis_status_read .....	91
表 3-94. 函数 adc_bifurcate_data_read .....	92
表 3-95. 函数 adc_flag_get.....	92
表 3-96. 函数 adc_flag_clear.....	93
表 3-97. 函数 adc_interrupt_enable.....	94
表 3-98. 函数 adc_interrupt_disable.....	95
表 3-99. 函数 adc_interrupt_flag_get .....	95
表 3-100. 函数 adc_interrupt_flag_clear.....	96
表 3-101. CAN 寄存器.....	97
表 3-102. CAN 库函数.....	98
表 3-103. 结构体 can_parameter_struct.....	98
表 3-104. 结构体 can_transmit_message_struct.....	99
表 3-105. 结构体 can_receive_message_struct.....	99
表 3-106. 结构体 can_filter_parameter_struct .....	99
表 3-107. 函数 can_deinit .....	100
表 3-108. 函数 can_struct_para_init.....	100
表 3-109. 函数 can_init.....	101
表 3-110. 函数 can_filter_init.....	101
表 3-111. 函数 can_debug_freeze_enable .....	102
表 3-112. 函数 can_debug_freeze_disable .....	102
表 3-113. 函数 can_time_trigger_mode_enable .....	103
表 3-114. 函数 can_time_trigger_mode_disable .....	103
表 3-115. 函数 can_message_transmit .....	104
表 3-116. 函数 can_transmit_states.....	104
表 3-117. 函数 can_transmission_stop.....	105
表 3-118. 函数 can_message_receive .....	106
表 3-119. 函数 can_fifo_release .....	106
表 3-120. 函数 can_receive_message_length_get.....	107
表 3-121. 函数 can_working_mode_set.....	107
表 3-122. 函数 can_wakeup .....	108
表 3-123. 函数 can_error_get.....	108

表 3-124. 函数 can_receive_error_number_get.....	109
表 3-125. 函数 can_transmit_error_number_get.....	109
表 3-126. 函数 can_flag_get.....	110
表 3-127. 函数 can_flag_clear.....	111
表 3-128. 函数 can_interrupt_enable.....	111
表 3-129. 函数 can_interrupt_disable.....	112
表 3-130. 函数 can_interrupt_flag_get .....	113
表 3-131. 函数 can_interrupt_flag_clear .....	114
表 3-132. CFMU 寄存器 .....	115
表 3-133. CFMU 库函数 .....	115
表 3-134. 枚举类型 cfmu_int_enum .....	116
表 3-135. 枚举类型 cfmu_int_flag_enum.....	116
表 3-136. 枚举类型 cfmu_int_flag_clear_enum.....	116
表 3-137. 函数 cfmu_deinit.....	117
表 3-138. 函数 cfmu_enable.....	117
表 3-139. 函数 cfmu_disable.....	118
表 3-140. 函数 cfmu_cfmuref_enable.....	118
表 3-141. 函数 cfmu_cfmuref_disable .....	119
表 3-142. 函数 cfmu_reference_signal_config .....	119
表 3-143. 函数 cfmu_digital_filter_config .....	120
表 3-144. 函数 cfmu_reference_clock_config .....	120
表 3-145. 函数 cfmu_measurement_clock_config.....	121
表 3-146. 函数 cfmu_limit_value_config.....	122
表 3-147. 函数 cfmu_interrupt_enable .....	123
表 3-148. 函数 cfmu_interrupt_disable .....	123
表 3-149. 函数 cfmu_interrupt_flag_get.....	124
表 3-150. 函数 cfmu_interrupt_flag_clear .....	124
表 3-151. CMP 寄存器.....	125
表 3-152. CMP 库函数.....	126
表 3-153. 枚举类型 cmp_enum .....	126
表 3-154. 函数 cmp_deinit.....	126
表 3-155. 函数 cmp_mode_init .....	127
表 3-156. 函数 cmp_noninverting_input_select.....	128
表 3-157. 函数 cmp_output_init.....	129
表 3-158. 函数 cmp_digital_filter_init.....	130
表 3-159. 函数 cmp_enable .....	130
表 3-160. 函数 cmp_disable .....	131
表 3-161. 函数 cmp_lock_enable .....	131
表 3-162. 函数 cmp_output_to_pin_enable.....	132
表 3-163. 函数 cmp_output_to_pin_disable.....	132
表 3-164. 函数 cmp_output_enable .....	133
表 3-165. 函数 cmp_output_disable .....	133
表 3-166. 函数 cmp_output_level_get.....	134

表 3-167. 函数 cmp_flag_get.....	135
表 3-168. 函数 cmp_flag_clear .....	135
表 3-169. 函数 cmp_interrupt_enable .....	136
表 3-170. 函数 cmp_interrupt_disable .....	137
表 3-171. 函数 cmp_interrupt_flag_get.....	137
表 3-172. 函数 cmp_interrupt_flag_clear.....	138
表 3-173. CPTIMER 寄存器 .....	138
表 3-174. CPTIMER 库函数 .....	139
表 3-175. 函数 cptimer_deinit .....	139
表 3-176. 函数 cptimer_enable .....	140
表 3-177. 函数 cptimer_disable .....	140
表 3-178. 函数 cptimer_prescaler_config .....	141
表 3-179. 函数 cptimer_autoreload_value_config.....	142
表 3-180. 函数 cptimer_counter_value_config .....	142
表 3-181. 函数 cptimer_counter_read .....	143
表 3-182. 函数 cptimer_prescaler_read .....	144
表 3-183. 函数 cptimer_event_software_generate.....	144
表 3-184. 函数 cptimer_flag_get.....	145
表 3-185. 函数 cptimer_flag_clear .....	146
表 3-186. 函数 cptimer_interrupt_enable .....	146
表 3-187. 函数 cptimer_interrupt_disable .....	147
表 3-188. 函数 cptimer_interrupt_flag_get .....	147
表 3-189. 函数 cptimer_interrupt_flag_clear.....	148
表 3-190. CPTIMERW 寄存器 .....	149
表 3-191. CPTIMERW 库函数 .....	150
表 3-192. 结构体 cptimerw_init_parameter_struct.....	151
表 3-193. 结构体 cptimerw_ic_parameter_struct .....	151
表 3-194. 函数 cptimerw_deinit .....	151
表 3-195. 函数 cptimerw_struct_para_init.....	152
表 3-196. 函数 cptimerw_init.....	152
表 3-197. 函数 cptimerw_autoreload_value_config.....	153
表 3-198. 函数 cptimerw_prescaler_config .....	153
表 3-199. 函数 cptimerw_counter_value_config .....	154
表 3-200. 函数 cptimerw_counter_clear_source_config .....	154
表 3-201. 函数 cptimerw_counter_read .....	155
表 3-202. 函数 cptimerw_prescaler_read .....	155
表 3-203. 函数 cptimerw_enable .....	156
表 3-204. 函数 cptimerw_disable .....	156
表 3-205. 函数 cptimerw_channel_output_mode_select .....	157
表 3-206. 函数 cptimerw_channel_output_compare_value_config .....	158
表 3-207. 函数 cptimerw_channel_output_state_config.....	158
表 3-208. 函数 cptimerw_channel_input_struct_para_init .....	159
表 3-209. 函数 cptimerw_input_capture_config.....	159

表 3-210. 函数 <code>cptimerw_channel_capture_value_register_read</code> .....	160
表 3-211. 函数 <code>cptimerw_event_software_generate</code> .....	161
表 3-212. 函数 <code>cptimerw_flag_get</code> .....	161
表 3-213. 函数 <code>cptimerw_flag_clear</code> .....	162
表 3-214. 函数 <code>cptimerw_interrupt_enable</code> .....	163
表 3-215. 函数 <code>cptimerw_interrupt_disable</code> .....	163
表 3-216. 函数 <code>cptimerw_interrupt_flag_get</code> .....	164
表 3-217. 函数 <code>cptimerw_interrupt_flag_clear</code> .....	165
表 3-218. CRC 寄存器 .....	166
表 3-219. CRC 库函数 .....	166
表 3-220. 函数 <code>crc_deinit</code> .....	167
表 3-221. 函数 <code>crc_data_register_reset</code> .....	167
表 3-222. 函数 <code>crc_reverse_output_data_enable</code> .....	168
表 3-223. 函数 <code>crc_reverse_output_data_disable</code> .....	168
表 3-224. 函数 <code>crc_input_data_reverse_config</code> .....	169
表 3-225. 函数 <code>crc_data_register_read</code> .....	169
表 3-226. 函数 <code>crc_free_data_register_read</code> .....	170
表 3-227. 函数 <code>crc_free_data_register_write</code> .....	170
表 3-228. 函数 <code>crc_init_data_register_write</code> .....	171
表 3-229. 函数 <code>crc_polynomial_size_set</code> .....	171
表 3-230. 函数 <code>crc_polynomial_set</code> .....	172
表 3-231. 函数 <code>crc_single_data_calculate</code> .....	172
表 3-232. 函数 <code>crc_block_data_calculate</code> .....	173
表 3-233. DAC 寄存器 .....	175
表 3-234. DAC 库函数 .....	175
表 3-235. 函数 <code>dac_deinit</code> .....	176
表 3-236. 函数 <code>dac_enable</code> .....	176
表 3-237. 函数 <code>dac_disable</code> .....	177
表 3-238. 函数 <code>dac_sync_enable</code> .....	178
表 3-239. 函数 <code>dac_sync_disable</code> .....	178
表 3-240. 函数 <code>dac_connect_to_pin_enable</code> .....	179
表 3-241. 函数 <code>dac_connect_to_pin_disable</code> .....	179
表 3-242. 函数 <code>dac_connect_to_cmp_enable</code> .....	180
表 3-243. 函数 <code>dac_connect_to_cmp_disable</code> .....	181
表 3-244. 函数 <code>dac_output_value_get</code> .....	181
表 3-245. 函数 <code>dac_data_set</code> .....	182
表 3-246. 函数 <code>dac_trigger_enable</code> .....	183
表 3-247. 函数 <code>dac_trigger_disable</code> .....	183
表 3-248. 函数 <code>dac_trigger_source_config</code> .....	184
表 3-249. 函数 <code>dac_software_trigger_enable</code> .....	184
表 3-250. 函数 <code>dac_wave_mode_config</code> .....	185
表 3-251. 函数 <code>dac_lfsr_noise_config</code> .....	186
表 3-252. 函数 <code>dac_triangle_noise_config</code> .....	187



表 3-253. 函数 dac_concurrent_enable .....	187
表 3-254. 函数 dac_concurrent_disable .....	188
表 3-255. 函数 dac_concurrent_software_trigger_enable .....	188
表 3-256. 函数 dac_concurrent_data_set .....	189
表 3-257. DBG 寄存器 .....	190
表 3-258. DBG 库函数 .....	190
表 3-259. 枚举类型 dbg_periph_enum .....	190
表 3-260. 函数 dbg_deinit .....	191
表 3-261. 函数 dbg_id_get .....	192
表 3-262. 函数 dbg_low_power_enable .....	192
表 3-263. 函数 dbg_low_power_disable .....	193
表 3-264. 函数 dbg_periph_enable .....	194
表 3-265. 函数 dbg_periph_disable .....	194
表 3-266. 函数 dbg_trace_pin_enable .....	195
表 3-267. 函数 dbg_trace_pin_disable .....	196
表 3-268. DMA 寄存器 .....	197
表 3-269. DMAMUX 寄存器 .....	197
表 3-270. DMA 库函数 .....	197
表 3-271. DMAMUX 库函数 .....	198
表 3-272. 结构体 dma_parameter_struct .....	199
表 3-273. 结构体 dmamux_sync_parameter_struct .....	199
表 3-274. 结构体 dmamux_gen_parameter_struct .....	199
表 3-275. 枚举 dmamux_interrupt_enum .....	200
表 3-276. 枚举 dmamux_flag_enum .....	201
表 3-277. 枚举 dmamux_interrupt_flag_enum .....	201
表 3-278. 枚举 dma_channel_enum .....	202
表 3-279. 枚举 dmamux_multiplexer_channel_enum .....	203
表 3-280. 枚举 dmamux_generator_channel_enum .....	203
表 3-281. 函数 dma_deinit .....	203
表 3-282. 函数 dma_struct_para_init .....	204
表 3-283. 函数 dma_init .....	205
表 3-284. 函数 dma_circulation_enable .....	206
表 3-285. 函数 dma_circulation_disable .....	206
表 3-286. 函数 dma_memory_to_memory_enable .....	207
表 3-287. 函数 dma_memory_to_memory_disable .....	207
表 3-288. 函数 dma_channel_enable .....	208
表 3-289. 函数 dma_channel_disable .....	209
表 3-290. 函数 dma_periph_address_config .....	209
表 3-291. 函数 dma_memory_address_config .....	210
表 3-292. 函数 dma_transfer_number_config .....	211
表 3-293. 函数 dma_transfer_number_get .....	211
表 3-294. 函数 dma_priority_config .....	212
表 3-295. 函数 dma_memory_width_config .....	213



表 3-296. 函数 dma_periph_width_config .....	214
表 3-297. 函数 dma_memory_increase_enable .....	214
表 3-298. 函数 dma_memory_increase_disable .....	215
表 3-299. 函数 dma_periph_increase_enable .....	216
表 3-300. 函数 dma_periph_increase_disable .....	216
表 3-301. 函数 dma_transfer_direction_config .....	217
表 3-302. 函数 dma_flag_get .....	218
表 3-303. 函数 dma_flag_clear .....	218
表 3-304. 函数 dma_interrupt_enable .....	219
表 3-305. 函数 dma_interrupt_disable .....	220
表 3-306. 函数 dma_interrupt_flag_get .....	221
表 3-307. 函数 dma_interrupt_flag_clear .....	221
表 3-308. 函数 dmamux_sync_struct_para_init .....	222
表 3-309. 函数 dmamux_synchronization_init .....	223
表 3-310. 函数 dmamux_synchronization_enable .....	223
表 3-311. 函数 dmamux_synchronization_disable .....	224
表 3-312. 函数 dmamux_event_generation_enable .....	225
表 3-313. 函数 dmamux_event_generation_disable .....	225
表 3-314. 函数 dmamux_gen_struct_para_init .....	226
表 3-315. 函数 dmamux_request_generator_init .....	226
表 3-316. 函数 dmamux_request_generator_channel_enable .....	227
表 3-317. 函数 dmamux_request_generator_channel_disable .....	228
表 3-318. 函数 dmamux_synchronization_polarity_config .....	228
表 3-319. 函数 dmamux_request_forward_number_config .....	229
表 3-320. 函数 dmamux_sync_id_config .....	230
表 3-321. 函数 dmamux_request_id_config .....	231
表 3-322. 函数 dma_interrupt_disable .....	236
表 3-323. 函数 dmamux_request_generate_number_config .....	237
表 3-324. 函数 dmamux_trigger_id_config .....	237
表 3-325. 函数 dmamux_flag_get .....	239
表 3-326. 函数 dmamux_flag_clear .....	240
表 3-327. 函数 dmamux_interrupt_enable .....	241
表 3-328. 函数 dmamux_interrupt_disable .....	243
表 3-329. 函数 dmamux_interrupt_flag_get .....	244
表 3-330. 函数 dmamux_interrupt_flag_clear .....	245
表 3-331. EVIC 寄存器 .....	247
表 3-332. EVIC 库函数 .....	248
表 3-333. 枚举类型 event_source_enum .....	248
表 3-334. 枚举类型 evic_periph_enum .....	253
表 3-335. 枚举类型 evic_cptimer_enum .....	255
表 3-336. 枚举类型 evic_single_io_enum .....	255
表 3-337. 函数 evic_init .....	255
表 3-338. 函数 evic_event_source_get .....	256

表 3-339. 函数 evic_register_lock_set .....	256
表 3-340. 函数 evic_register_lock_get .....	257
表 3-341. 函数 evic_cptimer_slave_mode_select .....	257
表 3-342. 函数 evic_group_member_config .....	258
表 3-343. 函数 evic_group_edge_detection_config .....	259
表 3-344. 函数 evic_group_output_level_config .....	259
表 3-345. 函数 evic_group_overwrite_enable .....	260
表 3-346. 函数 evic_group_overwrite_disable .....	261
表 3-347. 函数 evic_data_set .....	261
表 3-348. 函数 evic_data_get .....	262
表 3-349. 函数 evic_single_io_config .....	263
表 3-350. 函数 evic_single_io_config .....	263
表 3-351. 函数 evic_single_io_output_level_config .....	264
表 3-352. 函数 evic_register_write_enable .....	265
表 3-353. 函数 evic_register_write_disable .....	265
表 3-354. 函数 evic_register_write_enable_get .....	266
表 3-355. 函数 evic_bit_write_enable .....	266
表 3-356. 函数 evic_bit_write_disable .....	267
表 3-357. 函数 evic_register_write_enable_get .....	267
表 3-358. 函数 evic_software_event_generation .....	268
表 3-359. EXTI 寄存器 .....	268
表 3-360. EXTI 库函数 .....	269
表 3-361. 枚举类型 exti_line_enum .....	269
表 3-362. 枚举类型 exti_mode_enum .....	270
表 3-363. 枚举类型 exti_trig_type_enum .....	270
表 3-364. 函数 exti_deinit .....	270
表 3-365. 函数 exti_init .....	271
表 3-366. 函数 exti_interrupt_enable .....	271
表 3-367. 函数 exti_interrupt_disable .....	272
表 3-368. 函数 exti_event_enable .....	272
表 3-369. 函数 exti_event_disable .....	273
表 3-370. 函数 exti_software_interrupt_enable .....	273
表 3-371. 函数 exti_software_interrupt_disable .....	274
表 3-372. 函数 exti_digital_filter_enable .....	274
表 3-373. 函数 exti_digital_filter_disable .....	275
表 3-374. 函数 exti_flag_get .....	276
表 3-375. 函数 exti_flag_clear .....	276
表 3-376. 函数 exti_interrupt_flag_get .....	277
表 3-377. 函数 exti_interrupt_flag_clear .....	277
表 3-378. FMC 寄存器 .....	278
表 3-379. FMC 固件库函数 .....	278
表 3-380. 结构体 optionbyte_user_struct .....	279
表 3-381. 结构体 optionbyte_wwdgt0_struct .....	280

表 3-382. 枚举类型 <code>fmc_state_enum</code> .....	280
表 3-383. 枚举类型 <code>fmc_flag_enum</code> .....	280
表 3-384. 枚举类型 <code>fmc_interrupt_flag_enum</code> .....	281
表 3-385. 枚举类型 <code>fmc_interrupt_enum</code> .....	282
表 3-386. 枚举类型 <code>fmc_ob_wp_enum</code> .....	282
表 3-387. 函数 <code>fmc_unlock</code> .....	283
表 3-388. 函数 <code>fmc_lock</code> .....	284
表 3-389. 函数 <code>fmc_wscnt_set</code> .....	284
表 3-390. 函数 <code>fmc_prefetch_enable</code> .....	285
表 3-391. 函数 <code>fmc_prefetch_disable</code> .....	285
表 3-392. 函数 <code>fmc_ibus_enable</code> .....	286
表 3-393. 函数 <code>fmc_ibus_disable</code> .....	286
表 3-394. 函数 <code>fmc_ibus_reset_enable</code> .....	287
表 3-395. 函数 <code>fmc_ibus_reset_disable</code> .....	287
表 3-396. 函数 <code>fmc_dbus_enable</code> .....	288
表 3-397. 函数 <code>fmc_dbus_disable</code> .....	288
表 3-398. 函数 <code>fmc_dbus_reset_enable</code> .....	289
表 3-399. 函数 <code>fmc_dbus_reset_disable</code> .....	289
表 3-400. 函数 <code>fmc_blank_check</code> .....	290
表 3-401. 函数 <code>fmc_page_erase</code> .....	291
表 3-402. 函数 <code>fmc_mflash_mass_erase</code> .....	291
表 3-403. 函数 <code>fmc_dflash_mass_erase</code> .....	292
表 3-404. 函数 <code>fmc_mass_erase</code> .....	293
表 3-405. 函数 <code>fmc_fourword_program</code> .....	293
表 3-406. 函数 <code>fmc_fast_program</code> .....	294
表 3-407. 函数 <code>fmc_otf_fourword_program</code> .....	296
表 3-408. 函数 <code>ob_unlock</code> .....	297
表 3-409. 函数 <code>ob_lock</code> .....	297
表 3-410. 函数 <code>ob_erase</code> .....	298
表 3-411. 函数 <code>ob_write_protection_enable</code> .....	298
表 3-412. 函数 <code>ob_security_protection_config</code> .....	299
表 3-413. 函数 <code>ob_user_write</code> .....	299
表 3-414. 函数 <code>ob_data_program</code> .....	300
表 3-415. 函数 <code>ob_user1_write</code> .....	301
表 3-416. 函数 <code>ob_wwdg0_write</code> .....	302
表 3-417. 函数 <code>ob_wwdgt1_wwdgt2_write</code> .....	303
表 3-418. 函数 <code>ob_fwdgt_write</code> .....	303
表 3-419. 函数 <code>ob_user_get</code> .....	304
表 3-420. 函数 <code>ob_data_get</code> .....	305
表 3-421. 函数 <code>ob_write_protection_get</code> .....	305
表 3-422. 函数 <code>ob_security_protection_flag_get</code> .....	306
表 3-423. 函数 <code>fmc_flag_get</code> .....	307
表 3-424. 函数 <code>fmc_flag_clear</code> .....	307

表 3-425. 函数 fmc_interrupt_enable .....	308
表 3-426. 函数 fmc_interrupt_disable .....	308
表 3-427. 函数 fmc_interrupt_flag_get .....	309
表 3-428. 函数 fmc_interrupt_flag_clear .....	309
表 3-429. FWDGT 寄存器 .....	310
表 3-430. FWDGT 库函数 .....	310
表 3-431. 函数 fwdgt_write_enable .....	310
表 3-432. 函数 fwdgt_write_disable .....	311
表 3-433. 函数 fwdgt_enable .....	311
表 3-434. 函数 fwdgt_prescaler_value_config .....	312
表 3-435. 函数 fwdgt_reload_value_config .....	313
表 3-436. 函数 fwdgt_window_value_config .....	313
表 3-437. 函数 fwdgt_counter_reload .....	314
表 3-438. 函数 fwdgt_config .....	314
表 3-439. 函数 fwdgt_reset_output .....	315
表 3-440. 函数 fwdgt_interrupt_output .....	315
表 3-441. 函数 fwdgt_flag_get .....	316
表 3-442. GPIO 寄存器 .....	317
表 3-443. GPIO 库函数 .....	317
表 3-444. 函数 gpio_deinit .....	318
表 3-445. 函数 gpio_mode_set .....	319
表 3-446. 函数 gpio_output_options_set .....	320
表 3-447. 函数 gpio_bit_set .....	321
表 3-448. 函数 gpio_bit_reset .....	321
表 3-449. 函数 gpio_bit_write .....	322
表 3-450. 函数 gpio_port_write .....	322
表 3-451. 函数 gpio_input_bit_get .....	323
表 3-452. 函数 gpio_input_port_get .....	324
表 3-453. 函数 gpio_output_bit_get .....	324
表 3-454. 函数 gpio_output_port_get .....	325
表 3-455. 函数 gpio_af_set .....	325
表 3-456. 函数 gpio_pin_lock .....	326
表 3-457. 函数 gpio_bit_toggle .....	327
表 3-458. 函数 gpio_port_toggle .....	328
表 3-459. 函数 void gpio_bit_hold_enable .....	328
表 3-460. 函数 void gpio_bit_hold_enable .....	329
表 3-461. 函数 void gpio_tsel_function_enable .....	329
表 3-462. 函数 void gpio_tsel_function_disable .....	330
表 3-463. 函数 gpio_one_line_function_enable .....	330
表 3-464. 函数 gpio_one_line_function_disable .....	331
表 3-465. 函数 void gpio_smple_hold_enable .....	331
表 3-466. GPTIMER 寄存器 .....	332
表 3-467. GPTIMER 库函数 .....	333

表 3-468. 结构体类型 gptimer_parameter_struct.....	338
表 3-469. 结构体类型 gptimer_counter_enable_source_parameter_struct .....	338
表 3-470. 结构体类型 gptimer_counter_disable_source_parameter_struct .....	339
表 3-471. 结构体类型 gptimer_counter_reset_source_parameter_struct .....	339
表 3-472. 结构体类型 gptimer_capture_source_parameter_struct .....	340
表 3-473. 结构体类型 gptimer_counter_up_source_parameter_struct .....	340
表 3-474. 结构体类型 gptimer_counter_down_source_parameter_struct .....	341
表 3-475. 结构体类型 gptimer_oc_parameter_struct .....	342
表 3-476. 结构体类型 gptimer_com_oc_parameter_struct .....	342
表 3-477. GTOC 寄存器 .....	416
表 3-478. GTOC 库函数 .....	416
表 3-479. 函数 gtoc_deinit.....	417
表 3-480. 函数 gtoc_output_closing_request_enable .....	417
表 3-481. 函数 gtoc_gptimer_trigger_status_get.....	418
表 3-482. 函数 gtoc_software_request_generate.....	419
表 3-483. 函数 gtoc_software_request_stop .....	420
表 3-484. 函数 gtoc_input_detection_mode_select.....	420
表 3-485. 函数 gtoc_input_polarity_config .....	421
表 3-486. 函数 gtoc_digital_filter_enable .....	421
表 3-487. 函数 gtoc_digital_filter_disable .....	422
表 3-488. 函数 gtoc_digital_filter_config.....	423
表 3-489. 函数 gtoc_output_closing_request_mask .....	424
表 3-490. 函数 gtoc_register_write_enable .....	425
表 3-491. 函数 gtoc_register_write_disable .....	425
表 3-492. 函数 gtoc_extended_closing_request_enable .....	426
表 3-493. 函数 gtoc_extended_closing_request_disable .....	427
表 3-494. 函数 gtoc_extended_closing_request_config.....	427
表 3-495. 函数 gtoc_extended_closing_request_mask .....	428
表 3-496. 函数 gtoc_flag_get .....	429
表 3-497. 函数 gtoc_flag_clear .....	430
表 3-498. 函数 gtoc_interrupt_flag_get.....	430
表 3-499. 函数 gtoc_interrupt_flag_clear.....	431
表 3-500. I2C 寄存器 .....	432
表 3-501. I2C 库函数 .....	432
表 3-502. 枚举类型 i2c_interrupt_flag_enum .....	434
表 3-503. 函数 i2c_deinit .....	434
表 3-504. 函数 i2c_timing_config.....	435
表 3-505. 函数 i2c_digital_noise_filter_config.....	435
表 3-506. 函数 i2c_master_clock_config.....	436
表 3-507. 函数 i2c_master_addressing.....	437
表 3-508. 函数 i2c_address10_header_enable .....	438
表 3-509. 函数 i2c_address10_header_disable .....	438
表 3-510. 函数 i2c_address10_enable.....	439

表 3-511. 函数 i2c_address10_disable .....	439
表 3-512. 函数 i2c_automatic_end_enable.....	440
表 3-513. 函数 i2c_automatic_end_disable.....	440
表 3-514. 函数 i2c_slave_response_to_gcall_enable.....	441
表 3-515. 函数 i2c_slave_response_to_gcall_disable.....	441
表 3-516. 函数 i2c_stretch_scl_low_enable .....	442
表 3-517. 函数 i2c_stretch_scl_low_disable .....	442
表 3-518. 函数 i2c_address_config.....	443
表 3-519. 函数 i2c_address_bit_compare_config .....	443
表 3-520. 函数 i2c_address_disable .....	444
表 3-521. 函数 i2c_second_address_config .....	445
表 3-522. 函数 i2c_second_address_disable.....	446
表 3-523. 函数 i2c_receved_address_get.....	446
表 3-524. 函数 i2c_slave_byte_control_enable .....	447
表 3-525. 函数 i2c_slave_byte_control_disable .....	447
表 3-526. 函数 i2c_nack_enable.....	448
表 3-527. 函数 i2c_wakeup_from_deepsleep_enable .....	448
表 3-528. 函数 i2c_wakeup_from_deepsleep_disable .....	449
表 3-529. 函数 i2c_enable.....	449
表 3-530. 函数 i2c_disable.....	450
表 3-531. 函数 i2c_start_on_bus.....	450
表 3-532. 函数 i2c_stop_on_bus .....	451
表 3-533. 函数 i2c_data_transmit.....	451
表 3-534. 函数 i2c_data_receive.....	452
表 3-535. 函数 i2c_reload_enable .....	452
表 3-536. 函数 i2c_reload_disable .....	453
表 3-537. 函数 i2c_transfer_byte_number_config .....	453
表 3-538. 函数 i2c_dma_enable.....	454
表 3-539. 函数 i2c_dma_disable.....	454
表 3-540. 函数 i2c_pec_transfer.....	455
表 3-541. 函数 i2c_pec_enable .....	455
表 3-542. 函数 i2c_pec_disable .....	456
表 3-543. 函数 i2c_pec_value_get .....	456
表 3-544. 函数 i2c_smbus_mode_enable .....	457
表 3-545. 函数 i2c_smbus_mode_disable .....	457
表 3-546. 函数 i2c_smbus_alert_enable .....	458
表 3-547. 函数 i2c_smbus_alert_disable .....	458
表 3-548. 函数 i2c_smbus_default_addr_enable .....	459
表 3-549. 函数 i2c_smbus_default_addr_disable .....	459
表 3-550. 函数 i2c_smbus_host_addr_enable .....	460
表 3-551. 函数 i2c_smbus_host_addr_disable.....	460
表 3-552. 函数 i2c_extented_clock_timeout_enable .....	461
表 3-553. 函数 i2c_extented_clock_timeout_disable .....	461



表 3-554. 函数 i2c_clock_timeout_enable .....	462
表 3-555. 函数 i2c_clock_timeout_disable .....	462
表 3-556. 函数 i2c_bus_timeout_b_config .....	463
表 3-557. 函数 i2c_bus_timeout_a_config .....	463
表 3-558. 函数 i2c_idle_clock_timeout_config .....	464
表 3-559. 函数 i2c_flag_get .....	464
表 3-560. 函数 i2c_flag_clear .....	465
表 3-561. 函数 i2c_interrupt_enable .....	466
表 3-562. 函数 i2c_interrupt_disable .....	466
表 3-563. 函数 i2c_interrupt_flag_get .....	467
表 3-564. 函数 i2c_interrupt_flag_clear .....	468
表 3-565. NVIC 寄存器 .....	469
表 3-566. SysTick 寄存器 .....	470
表 3-567. MISC 库函数 .....	470
表 3-568. 枚举类型 IRQn_Type .....	470
表 3-569. 函数 nvic_priority_group_set .....	472
表 3-570. 函数 nvic_irq_enable .....	473
表 3-571. 函数 nvic_irq_disable .....	473
表 3-572. 函数 nvic_system_reset .....	474
表 3-573. 函数 nvic_vector_table_set .....	474
表 3-574. 函数 system_lowpower_set .....	475
表 3-575. 函数 system_lowpower_reset .....	476
表 3-576. 函数 systick_clksource_set .....	476
表 3-577. PMU 寄存器 .....	477
表 3-578. PMU 库函数 .....	477
表 3-579. 函数 pmu_deinit .....	478
表 3-580. 函数 pmu_to_sleepmode .....	478
表 3-581. 函数 pmu_to_deepsleepmode .....	479
表 3-582. 函数 pmu_to_standbymode .....	479
表 3-583. 函数 pmu_deepsleepmode_vcore_output .....	480
表 3-584. 函数 pmu_wakeup_pin_enable .....	480
表 3-585. 函数 pmu_wakeup_pin_disable .....	481
表 3-586. 函数 pmu_lvd_enable .....	481
表 3-587. 函数 pmu_lvd_disable .....	482
表 3-588. 函数 pmu_lvd_interrupt_select .....	483
表 3-589. 函数 pmu_lvd_interrupt_type .....	483
表 3-590. 函数 pmu_lvd_interrupt_reset_enable .....	484
表 3-591. 函数 pmu_lvd_interrupt_reset_disable .....	484
表 3-592. 函数 pmu_lvd_output_enable .....	485
表 3-593. 函数 pmu_lvd_output_disable .....	486
表 3-594. 函数 pmu_lvd_mode_select .....	486
表 3-595. 函数 pmu_lvd_reset_select .....	487
表 3-596. 函数 pmu_lvd_level_select .....	487

表 3-597. 函数 pmu_lvd_digital_filter_enable.....	488
表 3-598. 函数 pmu_lvd_digital_filter_disable.....	489
表 3-599. 函数 pmu_lvd_sample_clock_select.....	489
表 3-600. 函数 pmu_flag_get .....	490
表 3-601. 函数 pmu_flag_clear .....	491
表 3-602. POC 寄存器.....	491
表 3-603. GTOC 库函数 .....	492
表 3-604. 结构体类型 poc_request_struct .....	493
表 3-605. 结构体类型 poc_complementary_detection_struct .....	493
表 3-606. 枚举类型 poc_pin_enum.....	493
表 3-607. 枚举类型 cmp_output_enum .....	494
表 3-608. 枚举类型 target_timer_enum .....	494
表 3-609. 枚举类型 poc_interrupt_enum.....	494
表 3-610. 函数 poc_deinit .....	494
表 3-611. 函数 poc_input_detection_config.....	495
表 3-612. 函数 poc_input_dreq_status_config.....	497
表 3-613. 函数 poc_input_polarity_config.....	497
表 3-614. 函数 poc_sys_fault_dreq_status_config.....	498
表 3-615. 函数 poc_software_request_generate .....	499
表 3-616. 函数 poc_software_request_stop.....	499
表 3-617. 函数 poc_complementary_detection_struct_para_init.....	500
表 3-618. 函数 poc_timer0_complementary_detection_config .....	500
表 3-619. 函数 poc_timer7_complementary_detection_config .....	501
表 3-620. 函数 poc_timer0_output_disable_mode_select.....	501
表 3-621. 函数 poc_timer7_output_disable_mode_select.....	503
表 3-622. 函数 poc_timer1_output_disable_mode_select.....	504
表 3-623. 函数 poc_timer2_output_disable_mode_select.....	505
表 3-624. 函数 poc_gptimer0_output_disable_mode_select .....	506
表 3-625. 函数 poc_gptimer1_output_disable_mode_select .....	507
表 3-626. 函数 poc_request_struct_para_init.....	508
表 3-627. 函数 poc_request_select.....	509
表 3-628. 函数 poc_cmp_dreq_status_config .....	509
表 3-629. 函数 poc_cmp_dreq_status_extended_config.....	510
表 3-630. 函数 poc_input_detection_mask .....	511
表 3-631. 函数 poc_cmp_output_detection_mask.....	514
表 3-632. 函数 poc_flag_get.....	516
表 3-633. 函数 poc_flag_clear .....	517
表 3-634. 函数 poc_interrupt_enable .....	518
表 3-635. 函数 poc_interrupt_disable .....	518
表 3-636. 函数 poc_interrupt_flag_get.....	519
表 3-637. 函数 poc_interrupt_flag_clear.....	520
表 3-638. RCU 寄存器.....	521
表 3-639. RCU 库函数.....	521



表 3-640. 枚举类型 rcu_periph_enum.....	522
表 3-641. 枚举类型 rcu_periph_reset_enum .....	523
表 3-642. 枚举类型 rcu_periph_sleep_enum.....	525
表 3-643. 枚举类型 rcu_flag_enum .....	525
表 3-644. 枚举类型 rcu_int_flag_enum.....	525
表 3-645. 枚举类型 rcu_int_flag_clear_enum.....	526
表 3-646. 枚举类型 rcu_int_enum .....	526
表 3-647. 枚举类型 rcu_osci_type_enum.....	526
表 3-648. 枚举类型 rcu_clock_freq_enum .....	526
表 3-649. 函数 rcu_deinit.....	527
表 3-650. 函数 rcu_periph_clock_enable .....	527
表 3-651. 函数 rcu_periph_clock_disable .....	528
表 3-652. 函数 rcu_periph_clock_sleep_enable.....	530
表 3-653. 函数 rcu_periph_clock_sleep_disable.....	530
表 3-654. 函数 rcu_periph_reset_enable .....	531
表 3-655. 函数 rcu_periph_reset_disable .....	532
表 3-656. 函数 rcu_system_clock_source_config .....	533
表 3-657. 函数 rcu_system_clock_source_get.....	534
表 3-658. 函数 rcu_ahb_clock_config .....	534
表 3-659. 函数 rcu_apb1_clock_config .....	535
表 3-660. 函数 rcu_apb2_clock_config .....	535
表 3-661. 函数 rcu_ckout_config .....	536
表 3-662. 函数 rcu_pll_config .....	537
表 3-663. 函数 rcu_system_reset_enable.....	538
表 3-664. 函数 rcu_system_reset_disable.....	538
表 3-665. 函数 rcu_adc_clock_config .....	539
表 3-666. 函数 rcu_i2c_clock_config .....	540
表 3-667. 函数 rcu_osci_stab_wait.....	540
表 3-668. 函数 rcu_osci_on.....	541
表 3-669. 函数 rcu_osci_off .....	542
表 3-670. 函数 rcu_osci_bypass_mode_enable .....	542
表 3-671. 函数 rcu_osci_bypass_mode_disable .....	543
表 3-672. 函数 rcu_hxtal_frequency_scale_select .....	543
表 3-673. 函数 rcu_irc32m_adjust_value_set .....	544
表 3-674. 函数 rcu_hxtal_clock_monitor_enable .....	544
表 3-675. 函数 rcu_hxtal_clock_monitor_disable .....	545
表 3-676. 函数 rcu_pll_clock_monitor_disable .....	545
表 3-677. 函数 rcu_clock_freq_get .....	546
表 3-678. 函数 rcu_flag_get .....	546
表 3-679. 函数 rcu_all_reset_flag_clear.....	547
表 3-680. 函数 rcu_interrupt_flag_get.....	548
表 3-681. 函数 rcu_interrupt_flag_clear.....	549
表 3-682. 函数 rcu_interrupt_enable .....	549

表 3-683. 函数 rcu_interrupt_disable .....	550
表 3-684. SPI/I2S 寄存器 .....	551
表 3-685. SPI/I2S 库函数 .....	551
表 3-686. 结构体 spi_parameter_struct .....	552
表 3-687. 函数 spi_deinit .....	553
表 3-688. 函数 spi_struct_para_init .....	553
表 3-689. 函数 spi_init .....	554
表 3-690. 函数 spi_enable .....	554
表 3-691. 函数 spi_disable .....	555
表 3-692. 函数 spi_nss_output_enable .....	555
表 3-693. 函数 spi_nss_output_disable .....	556
表 3-694. 函数 spi_nss_internal_high .....	556
表 3-695. 函数 spi_nss_internal_low .....	557
表 3-696. 函数 spi_dma_enable .....	557
表 3-697. 函数 spi_dma_disable .....	558
表 3-698. 函数 spi_data_frame_format_config .....	559
表 3-699. 函数 spi_data_transmit .....	559
表 3-700. 函数 spi_data_receive .....	560
表 3-701. 函数 spi_bidirectional_transfer_config .....	560
表 3-702. 函数 spi_format_error_clear .....	561
表 3-703. 函数 spi_crc_polynomial_set .....	561
表 3-704. 函数 spi_crc_polynomial_get .....	562
表 3-705. 函数 spi_crc_on .....	562
表 3-706. 函数 spi_crc_off .....	563
表 3-707. 函数 spi_crc_next .....	563
表 3-708. 函数 spi_crc_get .....	564
表 3-709. 函数 spi_crc_error_clear .....	564
表 3-710. 函数 spi_ti_mode_enable .....	565
表 3-711. 函数 spi_ti_mode_disable .....	565
表 3-712. 函数 spi_nssp_mode_enable .....	566
表 3-713. 函数 spi_nssp_mode_disable .....	566
表 3-714. 函数 spi_quad_enable .....	567
表 3-715. 函数 spi_quad_disable .....	567
表 3-716. 函数 spi_quad_write_enable .....	568
表 3-717. 函数 spi_quad_read_enable .....	568
表 3-718. 函数 spi_flag_get .....	569
表 3-719. 函数 spi_interrupt_enable .....	569
表 3-720. 函数 spi_interrupt_disable .....	570
表 3-721. 函数 spi_interrupt_flag_get .....	571
表 3-722. SYSCFG 寄存器 .....	572
表 3-723. SVPWM 库函数 .....	572
表 3-724. 结构体 svpwm_parameter_struct .....	572
表 3-725. 函数 svpwm_deinit .....	573

表 3-726. 函数 svpwm_init .....	573
表 3-727. 函数 svpwm_enable .....	574
表 3-728. 函数 svpwm_flag_get .....	574
表 3-729. 函数 svpwm_alpha_beta_write .....	575
表 3-730. 函数 svpwm_ta_tb_tc_read .....	575
表 3-731. 函数 svpwm_sector_read .....	576
表 3-732. SYSCFG 寄存器 .....	577
表 3-733. SYSCFG 库函数 .....	578
表 3-734. 枚举类型 exti_gpio_enum .....	578
表 3-735. 枚举类型 syscfg_interrupt_enum .....	580
表 3-736. 枚举类型 syscfg_adcsn_enum .....	580
表 3-737. 枚举类型 timer_channel_input_enum .....	581
表 3-738. 枚举类型 ck_cmp_sel_enum .....	582
表 3-739. 函数 syscfg_deinit .....	582
表 3-740. 函数 syscfg_i2c_fast_mode_plus_enable .....	583
表 3-741. 函数 syscfg_i2c_fast_mode_plus_disable .....	583
表 3-742. 函数 syscfg_exti_line_config .....	584
表 3-743. 函数 syscfg_timer_input_source_select .....	584
表 3-744. 函数 syscfg_lockup_enable .....	585
表 3-745. 函数 syscfg_sram_remap_set .....	585
表 3-746. 函数 syscfg_sram_remap_size_get .....	586
表 3-747. 函数 syscfg_ck_cmp_sel_set .....	587
表 3-748. 函数 syscfg_ck_cmp_sel_get .....	587
表 3-749. 函数 syscfg_register_write_enable .....	588
表 3-750. 函数 syscfg_register_write_disable .....	588
表 3-751. 函数 syscfg_adc_signal_monitor_select .....	589
表 3-752. 函数 syscfg_adc_signal_monitor_output_enable .....	590
表 3-753. 函数 syscfg_adc_signal_monitor_output_disable .....	590
表 3-754. 函数 syscfg_boot_mode_get .....	591
表 3-755. 函数 syscfg_sram_ecc_error_address_get .....	591
表 3-756. 函数 syscfg_sram_ecc_error_bit_get .....	592
表 3-757. 函数 syscfg_interrupt_enable .....	592
表 3-758. 函数 syscfg_interrupt_disable .....	593
表 3-759. 函数 syscfg_interrupt_flag_get .....	593
表 3-760. 函数 syscfg_interrupt_flag_clear .....	594
表 3-761. TIMER 寄存器 .....	595
表 3-762. TIMER 库函数 .....	596
表 3-763. 结构体 timer_parameter_struct .....	600
表 3-764. 结构体 timer_break_parameter_struct .....	600
表 3-765. 结构体 timer_oc_parameter_struct .....	601
表 3-766. 结构体 timer_omc_parameter_struct .....	601
表 3-767. 结构体 timer_ic_parameter_struct .....	601
表 3-768. 函数 timer_deinit .....	602

表 3-769. 函数 timer_struct_para_init .....	602
表 3-770. 函数 timer_init.....	603
表 3-771. 函数 timer_enable.....	604
表 3-772. 函数 timer_disable.....	604
表 3-773. 函数 timer_auto_reload_shadow_enable .....	605
表 3-774. 函数 timer_auto_reload_shadow_disable.....	605
表 3-775. 函数 timer_update_event_enable .....	606
表 3-776. 函数 timer_update_event_disable .....	606
表 3-777. 函数 timer_counter_alignment.....	607
表 3-778. 函数 timer_counter_up_direction .....	608
表 3-779. 函数 timer_counter_down_direction.....	608
表 3-780. 函数 timer_upif_backup_enable .....	609
表 3-781. 函数 timer_upif_backup_disable .....	609
表 3-782. 函数 timer_flow_interrupt_source_select.....	610
表 3-783. 函数 timer_update_source_select.....	610
表 3-784. 函数 timer_external_input_source_select .....	611
表 3-785. 函数 timer_prescaler_config .....	612
表 3-786. 函数 timer_update_repetition_value_config.....	613
表 3-787. 函数 timer_flow_interrupt_repetition_value_config.....	613
表 3-788. 函数 timer_flow_interrupt_repetition_value_reload.....	614
表 3-789. 函数 timer_autoreload_value_config .....	614
表 3-790. 函数 timer_autoreload_value_read .....	615
表 3-791. 函数 timer_counter_value_config .....	615
表 3-792. 函数 timer_counter_read.....	616
表 3-793. 函数 timer_prescaler_read.....	617
表 3-794. 函数 timer_single_pulse_mode_config.....	617
表 3-795. 函数 timer_update_source_config .....	618
表 3-796. 函数 timer_channel_control_shadow_config.....	619
表 3-797. 函数 timer_channel_control_shadow_update_config .....	619
表 3-798. 函数 timer_ocpre_clr_source_select .....	620
表 3-799. 函数 timer_ocpre_clr_int_source_select .....	621
表 3-800. 函数 timer_channel_composite_asymmetric_pwm_level_config .....	621
表 3-801. 函数 timer_dma_enable.....	622
表 3-802. 函数 timer_dma_disable.....	623
表 3-803. 函数 timer_channel_dma_request_source_select .....	624
表 3-804. 函数 timer_dma_transfer_config .....	625
表 3-805. 函数 timer_event_software_generate .....	628
表 3-806. 函数 timer_break_struct_para_init .....	629
表 3-807. 函数 timer_break_config .....	630
表 3-808. 函数 timer_break_enable .....	631
表 3-809. 函数 timer_break_disable .....	631
表 3-810. 函数 timer_automatic_output_enable.....	632
表 3-811. 函数 timer_automatic_output_disable .....	632

表 3-812. 函数 timer_primary_output_config .....	633
表 3-813. 函数 timer_channel_output_struct_para_init.....	633
表 3-814. 函数 timer_channel_output_config.....	634
表 3-815. 函数 timer_channel_output_mode_config.....	635
表 3-816. 函数 timer_channel_output_pulse_value_config .....	636
表 3-817. 函数 timer_channel_output_shadow_config.....	637
表 3-818. 函数 timer_channel_output_compare_fast_config .....	638
表 3-819. 函数 timer_channel_output_clear_config .....	639
表 3-820. 函数 timer_channel_output_polarity_config .....	640
表 3-821. 函数 timer_channel_complementary_output_polarity_config .....	641
表 3-822. 函数 timer_channel_output_state_config .....	642
表 3-823. 函数 timer_channel_complementary_output_state_config .....	643
表 3-824. 函数 timer_channel_input_struct_para_init .....	644
表 3-825. 函数 timer_input_capture_config .....	645
表 3-826. 函数 timer_channel_input_capture_prescaler_config .....	646
表 3-827. 函数 timer_channel_capture_value_register_read.....	647
表 3-828. 函数 timer_input_pwm_capture_config .....	648
表 3-829. 函数 timer_hall_mode_config .....	648
表 3-830. 函数 timer_multi_mode_channel_output_parameter_struct_init .....	649
表 3-831. 函数 timer_multi_mode_channel_output_config.....	650
表 3-832. 函数 timer_multi_mode_channel_mode_config .....	651
表 3-833. 函数 timer_input_trigger_source_select.....	651
表 3-834. 函数 timer_master_output_trigger_source_select.....	653
表 3-835. 函数 timer_slave_mode_select.....	654
表 3-836. 函数 timer_master_slave_mode_config.....	655
表 3-837. 函数 timer_external_trigger_config .....	656
表 3-838. 函数 timer_quadrature_decoder_mode_config .....	657
表 3-839. 函数 timer_decoder_mode_config .....	658
表 3-840. 函数 timer_internal_clock_config.....	659
表 3-841. 函数 timer_internal_trigger_as_external_clock_config.....	659
表 3-842. 函数 timer_external_trigger_as_external_clock_config.....	660
表 3-843. 函数 timer_slave_mode_input_config .....	661
表 3-844. 函数 timer_external_clock_mode0_config .....	662
表 3-845. 函数 timer_external_clock_mode1_config .....	663
表 3-846. 函数 timer_external_clock_mode1_disable .....	664
表 3-847. 函数 timer_write_chxval_register_config .....	665
表 3-848. 函数 timer_output_value_selection_config .....	666
表 3-849. 函数 timer_output_match_pulse_select.....	666
表 3-850. 函数 timer_channel_composite_pwm_pulse_config .....	667
表 3-851. 函数 timer_channel_asymmetric_pwm_pulse_config .....	668
表 3-852. 函数 timer_channel_additional_compare_value_config .....	669
表 3-853. 函数 timer_channel_additional_compare_value_read .....	670
表 3-854. 函数 timer_channel_additional_output_shadow_config.....	670

表 3-855. 函数 timer_break_external_input_enable .....	671
表 3-856. 函数 timer_break_cmp_enable .....	672
表 3-857. 函数 timer_break_external_input_disable .....	672
表 3-858. 函数 timer_break_cmp_disable .....	673
表 3-859. 函数 timer_break_external_input_polarity_config .....	673
表 3-860. 函数 timer_break_cmp_polarity_config .....	674
表 3-861. 函数 timer_break_auto_recover_event_select.....	675
表 3-862. 函数 timer_trigger_adc_compare_enable.....	676
表 3-863. 函数 timer_trigger_adc_compare_disable.....	676
表 3-864. 函数 timer_trigger_adc_flow_enable .....	677
表 3-865. 函数 timer_trigger_adc_flow_disable .....	678
表 3-866. 函数 timer_trigger_adc_compare_value_config.....	679
表 3-867. 函数 timer_trigger_adc_repetition_value_config .....	679
表 3-868. 函数 timer_trigger_adc_repetition_decrement_select .....	680
表 3-869. 函数 timer_trigger_adc_repetition_value_reload .....	681
表 3-870. 函数 timer_trigger_adc_compare_value_shadow_enable .....	681
表 3-871. 函数 timer_trigger_adc_compare_value_shadow_disable .....	682
表 3-872. 函数 timer_trigger_adc_monitor_config .....	683
表 3-873. 函数 timer_register_update_event_select .....	685
表 3-874. 函数 timer_flag_get.....	686
表 3-875. 函数 timer_flag_clear.....	688
表 3-876. 函数 timer_interrupt_enable .....	690
表 3-877. 函数 timer_interrupt_disable.....	691
表 3-878. 函数 timer_interrupt_flag_get .....	692
表 3-879. 函数 timer_interrupt_flag_clear .....	693
表 3-880. TMU 寄存器.....	695
表 3-881. TMU 库函数.....	695
表 3-882. 结构体 tmu_parameter_struct .....	696
表 3-883. 函数 tmu_deinit.....	696
表 3-884. 函数 tmu_struct_para_init .....	697
表 3-885. 函数 tmu_init .....	697
表 3-886. 函数 tmu_dma_read_enable .....	698
表 3-887. 函数 tmu_dma_read_disable.....	699
表 3-888. 函数 tmu_dma_write_enable .....	699
表 3-889. 函数 tmu_dma_write_disable .....	700
表 3-890. 函数 tmu_one_q31_write.....	700
表 3-891. 函数 tmu_two_q31_write.....	701
表 3-892. 函数 tmu_two_q15_write.....	701
表 3-893. 函数 tmu_one_f32_write.....	702
表 3-894. 函数 tmu_two_f32_write.....	702
表 3-895. 函数 tmu_one_q31_read .....	703
表 3-896. 函数 tmu_two_q31_read.....	704
表 3-897. 函数 tmu_two_q15_read.....	704



表 3-898. 函数 tmu_one_f32_read .....	705
表 3-899. 函数 tmu_two_f32_read.....	705
表 3-900. 函数 tmu_flag_get .....	706
表 3-901. 函数 tmu_flag_clear .....	706
表 3-902. 函数 tmu_interrupt_enable .....	707
表 3-903. 函数 tmu_interrupt_disable .....	708
表 3-904. 函数 tmu_interrupt_flag_get.....	708
表 3-905. 函数 tmu_interrupt_flag_clear.....	709
表 3-906. UART 寄存器 .....	709
表 3-907. UART 库函数 .....	710
表 3-908. 枚举类型 uart_flag_enum .....	711
表 3-909. 枚举类型 uart_interrupt_flag_enum .....	711
表 3-910. 枚举类型 uart_interrupt_enum .....	712
表 3-911. 枚举类型 uart_invert_enum.....	712
表 3-912. 函数 uart_deinit.....	712
表 3-913. 函数 uart_baudrate_set .....	713
表 3-914. 函数 uart_parity_config .....	713
表 3-915. 函数 uart_word_length_set.....	714
表 3-916. 函数 uart_stop_bit_set .....	715
表 3-917. 函数 uart_enable.....	715
表 3-918. 函数 uart_disable.....	716
表 3-919. 函数 uart_transmit_config .....	716
表 3-920. 函数 uart_receive_config .....	717
表 3-921. 函数 uart_data_first_config .....	717
表 3-922. 函数 uart_invert_config.....	718
表 3-923. 函数 uart_oversample_config .....	719
表 3-924. 函数 uart_sample_bit_config .....	719
表 3-925. 函数 uart_data_transmit.....	720
表 3-926. 函数 uart_data_receive.....	720
表 3-927. 函数 uart_address_config.....	721
表 3-928. 函数 uart_mute_mode_enable .....	722
表 3-929. 函数 uart_mute_mode_disable .....	722
表 3-930. 函数 uart_mute_mode_wakeup_config.....	723
表 3-931. 函数 uart_lin_mode_enable .....	723
表 3-932. 函数 uart_lin_mode_disable .....	724
表 3-933. 函数 uart_lin_break_dection_length_config.....	724
表 3-934. 函数 uart_halfduplex_enable.....	725
表 3-935. 函数 uart_halfduplex_disable.....	725
表 3-936. 函数 uart_halfduplex_disable.....	726
表 3-937. 函数 uart_irda_mode_enable .....	726
表 3-938. 函数 uart_irda_mode_disable .....	727
表 3-939. 函数 uart_prescaler_config .....	727
表 3-940. 函数 uart_irda_lowpower_config.....	728

表 3-941. 函数 <code>uart_parity_check_coherence_config</code> .....	729
表 3-942. 函数 <code>uart_dma_receive_config</code> .....	729
表 3-943. 函数 <code>uart_dma_transmit_config</code> .....	730
表 3-944. 函数 <code>uart_flag_get</code> .....	731
表 3-945. 函数 <code>uart_flag_clear</code> .....	731
表 3-946. 函数 <code>uart_interrupt_enable</code> .....	732
表 3-947. 函数 <code>uart_interrupt_disable</code> .....	732
表 3-948. 函数 <code>uart_interrupt_flag_get</code> .....	733
表 3-949. 函数 <code>uart_interrupt_flag_clear</code> .....	734
表 3-950. WWDGT 寄存器 .....	734
表 3-951. WWDGT 库函数 .....	735
表 3-952. 结构体 <code>wwdgt_cfg_parameter_struct</code> .....	735
表 3-953. 函数 <code>wwdgt_deinit</code> .....	735
表 3-954. 函数 <code>wwdgt_counter_update</code> .....	736
表 3-955. 函数 <code>wwdgt_struct_para_init</code> .....	736
表 3-956. 函数 <code>wwdgt_cfg_init</code> .....	737
表 3-957. 函数 <code>wwdgt_flag_get</code> .....	737
表 3-958. 函数 <code>wwdgt_flag_clear</code> .....	738
表 4-1. 版本历史 .....	739



## 1. 介绍

本手册介绍了32位基于ARM微控制器GD32M53x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GD32M53x所有外设的性能特征。该固件库还包括每一个外设的驱动描述和基于评估板的固件库使用例程。通过使用本固件库，用户无需深入掌握细节，也可以轻松应用每一个外设。使用本固件库可以大大减少用户的编程时间，从而降低开发成本。

每个外设驱动都由一组函数组成，这组函数覆盖了该外设所有功能。可以通过调用一组通用API(application programming interface应用编程界面)来实现对外设的驱动，这些API的结构、函数名称和参数名称都进行了标准化规范。

所有的驱动源代码都符合“MISRA-C:2004”标准（例程文件符合扩充ANSI-C标准），不会受到来自开发环境差异带来的影响。仅有启动文件取决于开发环境。

因为该固件库是通用的，并且包括了所有外设的功能，所以应用程序代码的大小和执行速度可能不是最优的。对大多数应用程序来说，用户可以直接使用之，对于那些在代码大小和执行速度方面有严格要求的应用程序，该固件库可以作为如何设置外设的一份参考资料，可以根据实际需求对其进行调整。

此份固件库使用手册的整体架构如下：

- 文档和固件库规则；
- 固件库概述；
- 外设固件库具体描述，外设固件库例程使用说明。

### 1.1. 文档和固件库规则

#### 1.1.1. 外设缩写

表 1-1. 外设缩写

外设缩写	说明
ADC	模数转换器
CAN	控制器局域网络
CFMU	时钟频率测量单元
CMP	比较器
CPTIMER	比较定时器
CPTIMERW	比较定时器W
CRC	循环冗余校验计算单元
DAC	数模转换器
DBG	调试模块
DMA	直接存储器访问控制器
EVIC	事件互连单元

外设缩写	说明
EXTI	外部中断事件控制器
FMC	闪存控制器
FWDGT	独立看门狗
GPIO	通用和备用输入
GPTIMER	通用定时器
GTOC	通用定时器输出控制器
I2C	内部集成电路总线接口
MISC	嵌套中断向量列表控制器
PMU	电源管理单元
POC	端口输出控制器
RCU	复位和时钟单元
SPI/I2S	串行外设接口/片上音频接口
SVPWM	空间矢量脉宽调制
SYSCFG	系统配置
TIMER	定时器
TMU	三角函数加速器
UART	通用异步收发器
WWDGT	窗口看门狗

### 1.1.2. 命名规则

固件库遵从以下命名规则：

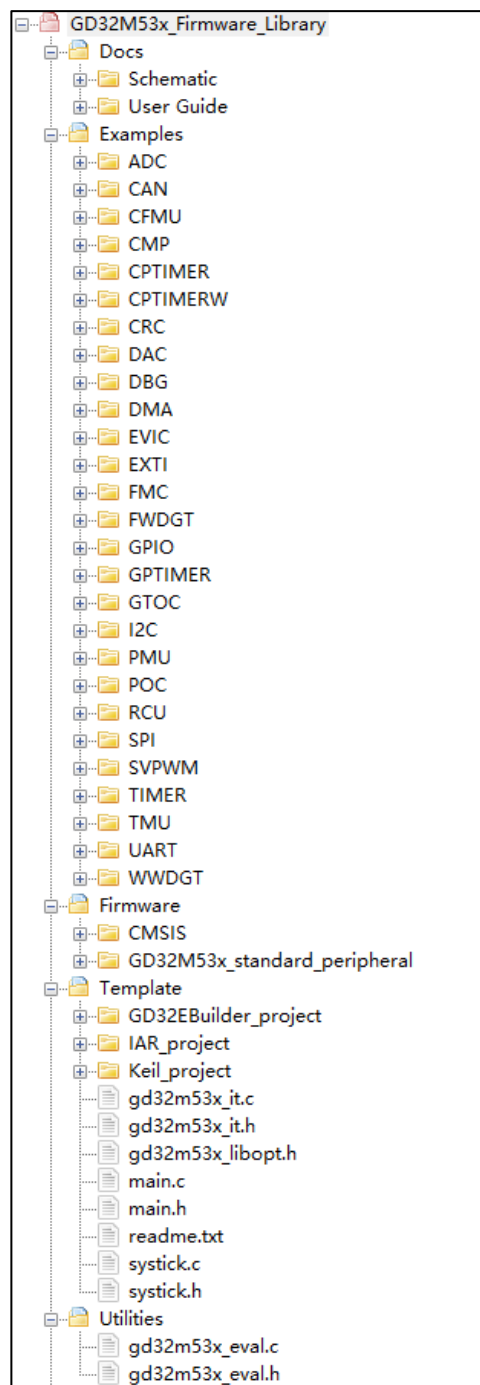
- XXX表示任一外设缩写，例如：ADC。更多缩写相关信息参阅[外设缩写](#)；
- 源文件和头文件命名都以“gd32m53x\_”作为开头，例如：gd32m53x\_adc.h；
- 常量仅被应用于一个文件的，定义于该文件中；被应用于多个文件的，在对应头文件中定义。所有常量都由英文字母大写书写；
- 寄存器作为常量处理。他们的命名都由英文字母大写书写。在大多数情况下，寄存器缩写规范与本用户手册一致；
- 变量名采用全部小写，有多个单词组成的，在单词之间以下划线分隔；
- 外设函数的命名以该外设的缩写加下划线为开头，有多个单词组成的，在单词之间以下划线分隔，所有外设函数都由英文字母小写书写。

## 2. 固件库概述

### 2.1. 文件组织结构

GD32M53x\_Firmware\_Library，文件组织结构见下图：

图 2-1. GD32M53x 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；
- **gd32m53x\_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）；
- **gd32m53x\_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）；
- **gd32m53x\_it.h**: 该头文件包含了所有的中断处理程序的原形；
- **systick.c**: 该源文件包含了使用**systick**的精准延时程序；
- **systick.h**: 该头文件包含了使用**systick**的精准延时程序的原形；
- **main.c**: 例程代码注：所有的例程的使用，都不受不同软件开发环境的影响。

### 2.1.2. Firmware 文件夹

**Firmware**文件夹包含组成固件库核心的所有子文件夹和文件：

- **CMSIS**子文件夹包含有**Cortex® M33**内核的支持文件、基于**Cortex® M33**内核处理器的启动代码和库引导文件以及基于**GD32M53x**的全局头文件和系统配置文件；
- **GD32M53x\_standard\_peripheral**子文件夹：
  - **Include**子文件夹包含了固件函数库所需的头文件，用户无需修改该文件夹；
  - **Source**子文件夹包含了固件函数库所需的源文件，用户无需修改该文件夹；

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

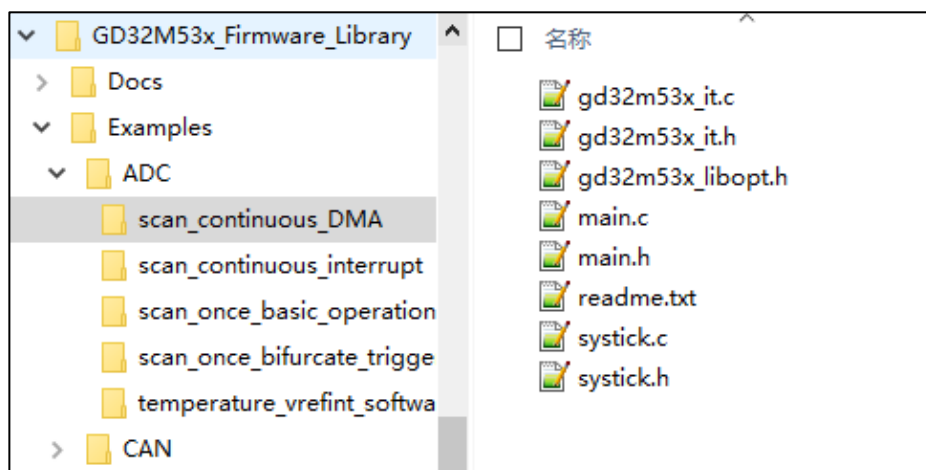
### 2.1.3. Template 文件夹

**Template**文件夹包含一个关于使用**LED**、**USART**打印、按键控制的简单例程，（**IAR\_project**用于**IAR**编译环境，**Keil\_project**用于**Keil5**编译环境，**GD32EBuilder\_project**用于**GD32EmbeddedBuilder**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“**scan\_continuous\_DMA**”，如下图所示：

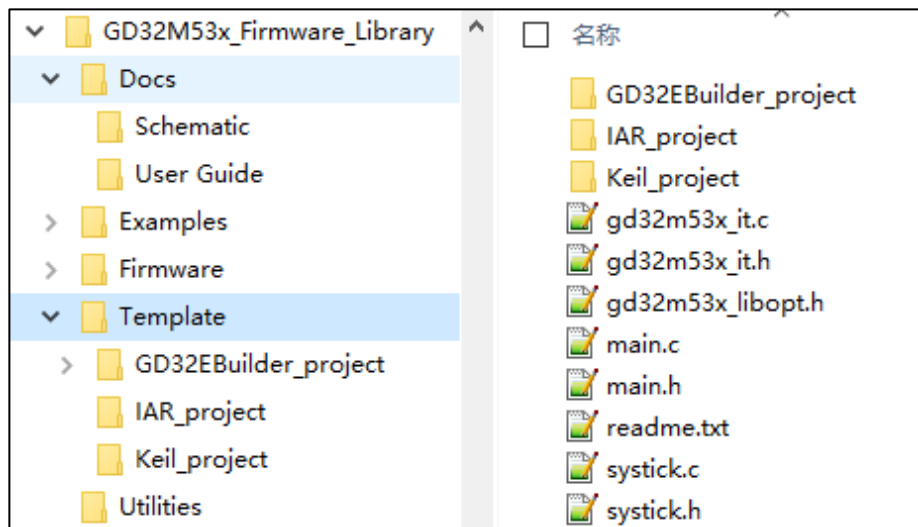
图 2-2. 选择外设例程文件



### 拷贝文件

打开“Template”文件夹，将“IAR\_project”，“Keil\_project”和“GD32EBuilder\_project”三个文件夹保留，其他文件都删除，然后将“scan\_continuous\_DMA”文件夹中的所有文件拷到“Template”文件夹子目录下，如下图所示：

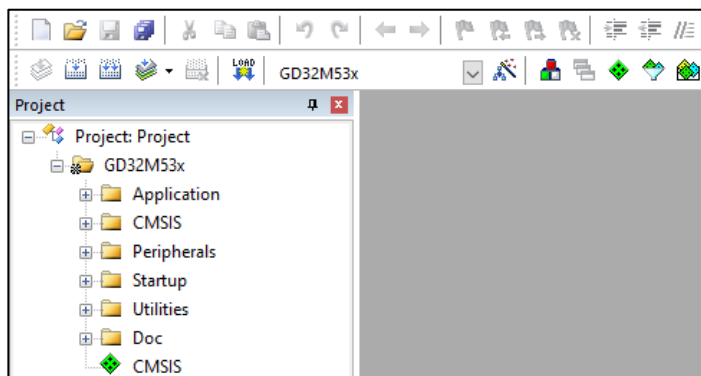
图 2-3. 拷贝外设例程文件



### 打开工程

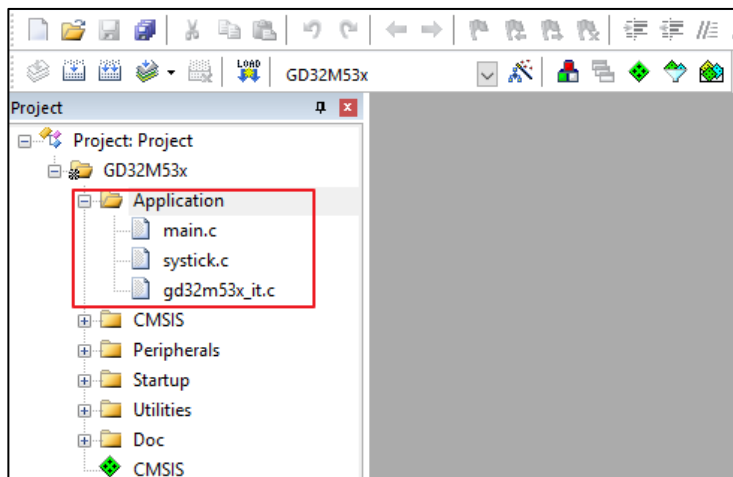
GD提供Keil, IAR和GD32EmbeddedBuilder三种版本的工程，根据客户所安装的软件，打开不同的project，如“Keil\_project”，打开\Template\Keil\_project\Project.uvprojx，如下图所示：

图 2-4. 打开工程文件



由于不同的模块、不同的功能，会使用到不同的文件，需要根据客户选择拷贝的文件，对工程里的文件进行增加或删除，如下图所示：

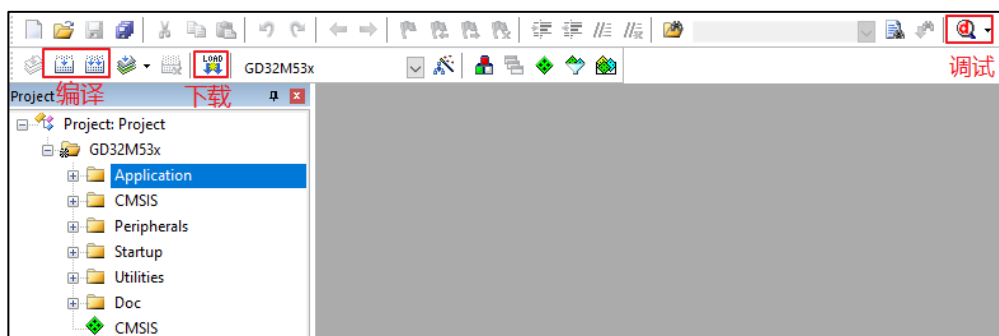
图 2-5. 配置工程文件



## 编译调试下载

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

图 2-6. 编译调试下载



## 2.1.4. Utilities 文件夹

Utilities文件夹包含运行固件库例程评估板的文件：

- gd32m53x\_eval.h文件是运行固件库例程所需关于评估板的头文件；
- gd32m53x\_eval.c文件是运行固件库例程所需关于评估板的源文件。

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

## 2.2. 固件库文件描述

下表列举和描述了固件库使用的主要文件。

**表 2-1. 固件函数库文件描述**

文件名	描述
gd32m53x_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。
gd32m53x_it.h	头文件，包含所有中断处理函数原形。
gd32m53x_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。
gd32m53x_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。
gd32m53x_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_EOCCTL	软件EOC控制寄存器
ADC_CTL0	控制寄存器0
ADC_GPCGF0	组配置寄存器0
ADC_GPCGF1	组配置寄存器1
ADC_CHSEL0	通道选择寄存器0
ADC_CHSEL1	通道选择寄存器1
ADC_SAMPR0	采样时间寄存器0
ADC_SAMPR1	采样时间寄存器1
ADC_SAMPR2	采样时间寄存器2
ADC_CHPRI0	通道优先级寄存器0



寄存器名称	寄存器描述
ADC_CHPRI1	通道优先级寄存器1
ADC_ADDT0	通道加法次数寄存器0
ADC_ADDT1	通道加法次数寄存器1
ADC_SDDATA	自诊断数据寄存器
ADC_GP1BIDATA	Group_pri 1组分叉数据寄存器
ADC_GP1BIDATA1	Group_pri 1组分叉数据寄存器1
ADC_GP1BIDATA2	Group_pri 1组分叉数据寄存器2
ADC_GP2BIDATA	Group_pri 2组分叉数据寄存器
ADC_GP2BIDATA1	Group_pri 2组分叉数据寄存器1
ADC_GP2BIDATA2	Group_pri 2组分叉数据寄存器2
ADC_GP3BIDATA	Group_pri 3组分叉数据寄存器
ADC_GP3BIDATA1	Group_pri 3组分叉数据寄存器1
ADC_GP3BIDATA2	Group_pri 3组分叉数据寄存器2
ADC_GP4BIDATA	Group_pri 4组分叉数据寄存器
ADC_GP4BIDATA1	Group_pri 4组分叉数据寄存器1
ADC_GP4BIDATA2	Group_pri 4组分叉数据寄存器2
ADC_GP1DMAR	Group_pri 1组DMA数据寄存器
ADC_GP2DMAR	Group_pri 2组DMA数据寄存器
ADC_GP3DMAR	Group_pri 3组DMA数据寄存器
ADC_GP4DMAR	Group_pri 4组DMA数据寄存器
ADC_CH0DATA	通道0数据寄存器
ADC_CH1DATA	通道1数据寄存器
ADC_CH2DATA	通道2数据寄存器
ADC_CH3DATA	通道3数据寄存器
ADC_CH4DATA	通道4数据寄存器
ADC_CH5DATA	通道5数据寄存器
ADC_CH6DATA	通道6数据寄存器
ADC_CH7DATA	通道7数据寄存器，仅用于ADC2
ADC_CH8DATA	通道8数据寄存器，仅用于ADC2
ADC_CH9DATA	通道9数据寄存器，仅用于ADC2
ADC_TEMPDATA	温度传感器通道数据寄存器，仅用于ADC2
ADC_VINTDATA	内部参考电压通道数据寄存器，仅用于ADC2
ADC_WDCTL	看门狗控制寄存器
ADC_WDATHOLD	模拟看门狗A阈值寄存器
ADC_WDBTHOLD	模拟看门狗B阈值寄存器
ADC_WDACHCFG	模拟看门狗A通道配置寄存器
ADC_WDACHSTAT	模拟看门狗A通道状态寄存器
ADC_CTL1	控制寄存器1
ADC_SHCTL	采样保持控制寄存器
ADC_DMACTL	DMA控制寄存器

寄存器名称	寄存器描述
ADC_BITRGCTL	分叉触发控制寄存器

### 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 ADCx 外设
adc_enable	使能 ADCx 外设
adc_disable	禁能 ADCx 外设
adc_data_alignment_config	配置 ADCx 数据对齐方式
adc_resolution_config	配置 ADCx 分辨率
adc_self_diagnosis_enable	使能自诊断
adc_self_diagnosis_disable	禁能自诊断
adc_self_diagnosis_mode_config	自诊断配置
adc_disconnect_detect_mode_config	配置断开检测辅助模式
adc_disconnect_detect_period_config	配置断开检测时长
adc_data_auto_clear_enable	使能数据寄存器自动清零
adc_data_auto_clear_disable	禁能数据寄存器自动清零
adc_data_auto_set_enable	使能数据寄存器自动置位
adc_data_auto_set_disable	禁能数据寄存器自动置位
adc_sample_hold_channel_config	配置 ADC 采保通道
adc_sh_sample_time_config	配置 ADC 采保电路的采样时间
adc_sh_hold_time_config	配置 ADC 采保电路的保持时间
adc_sh_constant_sampling_mode_enable	使能 ADC 采保电路的恒采样模式
adc_sh_constant_sampling_mode_disable	禁能 ADC 采保电路的恒采样模式
adc_sh_constant_sampling_start	软件启动采保电路的恒采样
adc_sh_constant_sampling_stop	软件结束采保电路的恒采样
adc_watchdog_enable	使能 ADC 模拟看门狗
adc_watchdog_disable	禁能 ADC 模拟看门狗
adc_watchdog_a_channel_config	配置 ADC 模拟看门狗 A 的通道
adc_watchdog_a_channel_deselect	取消 ADC 模拟看门狗 A 的通道选择
adc_watchdog_b_channel_config	配置 ADC 模拟看门狗 B 的通道
adc_watchdog_a_threshold_config	配置 ADC 模拟看门狗 A 的阈值
adc_watchdog_b_threshold_config	配置 ADC 模拟看门狗 B 的阈值
adc_watchdog_window_mode_enable	使能 ADC 模拟看门狗窗口功能
adc_watchdog_window_mode_disable	禁能 ADC 模拟看门狗窗口功能
adc_watchdog_complex_condition_config	配置模拟看门狗复合比较监视标志
adc_oversample_channel_config	配置 ADC 过采样通道
adc_oversample_mode_config	配置 ADC 过采样模式

adc_oversample_mode_enable	使能 ADC 过采样模式
adc_oversample_mode_disable	禁能 ADC 过采样模式
adc_group_scan_mode_config	配置 ADC 扫描模式
adc_group_priority_control_enable	使能组优先级控制
adc_group_priority_control_disable	禁能组优先级控制
adc_group_pri3_enable	使能 Group_pri3
adc_group_pri3_disable	禁能 Group_pri3
adc_group_pri4_enable	使能 Group_pri4
adc_group_pri4_disable	禁能 Group_pri4
adc_group_lowest_priority_continuous_enable	使能最低优先级组连续扫描
adc_group_lowest_priority_continuous_disable	禁能最低优先级组连续扫描
adc_group_restart_enable	使能低优先级组重启
adc_group_restart_disable	禁能低优先级组重启
adc_group_channel_config	选择 Group_prix 组通道
adc_group_channel_deselect	取消 ADC 组通道的选择
adc_group_end_flag_round_config	配置 Group_prix 组转换结束标志轮次
adc_group_external_trigger_enable	使能 ADC 外部触发
adc_group_extern_trigger_edge_config	配置 ADC 外部触发边沿
adc_group_external_trigger_disable	禁能 ADC 外部触发
adc_group_synchronous_trigger_enable	使能 ADC 同步触发
adc_group_synchronous_trigger_source_config	配置 ADC 同步触发源
adc_group_asynchronous_trigger_enable	使能 ADC 异步触发
adc_group_software_trigger_enable	使能 ADC 软件触发
adc_group_software_end_conversion	软件结束所有组的转换
adc_group_bifurcate_mode_enable	使能 Group_pri x 组的分叉触发模式
adc_group_bifurcate_mode_disable	禁能 Group_pri x 组的分叉触发模式
adc_group_bifurcate_channel_select	ADC 分叉触发通道选择
adc_group_bifurcate_extend_trigger_select	扩展分叉模式的触发源选择
adc_group_bifurcate_trigger_restart_enable	使能触发存储功能（当前转换周期内的触发会被存储起来，当前转换结束后响应存储起来的触发）
adc_group_bifurcate_trigger_restart_disable	禁能触发存储功能（当前转换周期内的触发会被丢弃）
adc_group_dma_mode_enable	使能 DMA 请求
adc_group_dma_mode_disable	禁能 DMA 请求
adc_group_dma_request_after_last_enable	当 DMAENx 位为 1 时，在每次 Group_pri x 组转换结束后产生 DMA 请求。
adc_group_dma_request_after_last_disable	在检测到 DMA 控制器的传输结束信号之后，禁能 DMA 传输
adc_group_dma_overshoot_detect_enable	使能 DMA 溢出检测
adc_group_dma_overshoot_detect_disable	禁能 DMA 溢出检测
adc_channel_priority_config	通道优先级配置

adc_channel_sample_time_config	配置通道采样时间
adc_evic_link_signal_event_config	选择 EVIC 连接信号
adc_group_evic_link_signal_source	选择 Group_pri x 的 EVIC 连接信号源
adc_channel_data_read	读 ADC 通道数据寄存器
adc_self_diagnosis_data_read	读 ADC 自诊断数据寄存器
adc_self_diagnosis_status_read	读 ADC 自诊断状态
adc_bifurcate_data_read	读 ADC 分叉数据寄存器
adc_flag_get	获取 ADC 标志
adc_flag_clear	清除 ADC 标志
adc_interrupt_enable	使能 ADC 中断
adc_interrupt_disable	禁能 ADC 中断
adc_interrupt_flag_get	获取 ADC 中断标志
adc_interrupt_flag_clear	清除 ADC 中断标志

### 枚举类型 adc\_group\_select\_enum

表 3-4. 枚举类型 adc\_group\_select\_enum

成员名称	功能描述
ADC_GROUP_PRI4	选择 Group_pri4 组
ADC_GROUP_PRI3	选择 Group_pri3 组
ADC_GROUP_PRI2	选择 Group_pri2 组
ADC_GROUP_PRI1	选择 Group_pri1 组

### 枚举类型 adc\_channel\_select\_enum

表 3-5. 枚举类型 adc\_channel\_select\_enum

成员名称	功能描述
ADC_CHANNEL_IN00	选择通道 0 (ADCx_IN00)
ADC_CHANNEL_IN01	选择通道 1 (ADCx_IN01)
ADC_CHANNEL_IN02	选择通道 2 (ADCx_IN02)
ADC_CHANNEL_IN03	选择通道 3 (ADCx_IN03)
ADC_CHANNEL_IN04	选择通道 4 (ADCx_IN04)
ADC_CHANNEL_IN05	选择通道 5 (ADCx_IN05)
ADC_CHANNEL_IN06	选择通道 6 (ADCx_IN06)
ADC_CHANNEL_IN07	选择通道 7 (ADCx_IN07), 仅适用 ADC2
ADC_CHANNEL_IN08	选择通道 8 (ADCx_IN08), 仅适用 ADC2
ADC_CHANNEL_IN09	选择通道 9 (ADCx_IN09), 仅适用 ADC2
ADC_CHANNEL_TEMPERATURE	选择温度传感器通道, 仅适用 ADC2
ADC_CHANNEL_VINT	选择内部参考电压通道, 仅适用 ADC2
ADC_CHANNEL_ALL	选择所有通道

### 枚举类型 `adc_bifurcate_data_enum`

表 3-6. 枚举类型 `adc_bifurcate_data_enum`

成员名称	功能描述
<code>ADC_BIFURCATE_DATA</code>	选择 Group_pri x 组分叉数据寄存器 (ADC_GPxBIDATA)
<code>ADC_EXTENDED_BIFURCATE_DATA_1</code>	选择 Group_pri x 组分叉数据寄存器 1 (ADC_GPxBIDATA1)
<code>ADC_EXTENDED_BIFURCATE_DATA_2</code>	选择 Group_pri x 组分叉数据寄存器 2 (ADC_GPxBIDATA2)

### 枚举类型 `adc_disc_detect_mode_enum`

表 3-7. 枚举类型 `adc_disc_detect_mode_enum`

成员名称	功能描述
<code>ADC_DISCHARGE_MODE</code>	选择放电模式
<code>ADC_PRECHARGE_MODE</code>	选择预充电模式
<code>ADC_PRECHARGE_DISCHARGE_MODE_DISABLE</code>	禁能预充电和放电模式

### 枚举类型 `adc_self_diag_mode_enum`

表 3-8. 枚举类型 `adc_self_diag_mode_enum`

成员名称	功能描述
<code>ADC_SELF_DIAGNOSIS_MODE_ROTATION</code>	自诊断电压选择轮转模式
<code>ADC_SELF_DIAGNOSIS_MODE_FIXED</code>	自诊断电压选择固定模式

### 枚举类型 `adc_self_diag_fixed_voltage_enum`

表 3-9. 枚举类型 `adc_self_diag_fixed_voltage_enum`

成员名称	功能描述
<code>ADC_SELF_DIAG_0_V</code>	自诊断电压为 0V
<code>ADC_SELF_DIAG_1_2_VREF</code>	自诊断电压为 $V_{REFP}/2$
<code>ADC_SELF_DIAG_VREF</code>	自诊断电压为 $V_{REFP}$

### 枚举类型 `adc_restart_channel_enum`

表 3-10. 枚举类型 `adc_restart_channel_enum`

成员名称	功能描述
<code>ADC_RESTART_ON_BEGINNING</code>	从初始通道开始重新扫描低优先级组
<code>ADC_RESTART_ON_BREAKING</code>	从上次被中止的通道开始扫描低优先级组

### 枚举类型 `adc_watchdog_select_enum`

表 3-11. 枚举类型 `adc_watchdog_select_enum`

成员名称	功能描述
<code>ADC_WATCHDOG_A</code>	选择模拟看门狗 A
<code>ADC_WATCHDOG_B</code>	选择模拟看门狗 B
<code>ADC_WATCHDOG_A_B</code>	选择模拟看门狗 A 和模拟看门狗 B

### 枚举类型 `adc_watchdog_compare_condition_enum`

表 3-12. 枚举类型 `adc_watchdog_compare_condition_enum`

成员名称	功能描述
<code>ADC_OUT_WINDOW</code>	当 <code>WINEN</code> 位为 0，比较比较条件为: <code>WDBLT[15:0] &gt; A/D 转换值</code> 。 当 <code>WINEN</code> 位为 1，比较比较条件为: ( <code>A/D 转换值 &lt; WDBLT[15:0]</code> ) 或者 ( <code>WDBHT[15:0] &lt; A/D 转换值</code> )
<code>ADC_IN_WINDOW</code>	当 <code>WINEN</code> 位为 0，比较比较条件为: <code>WDBLT[15:0] &lt; A/D 转换值</code> 。 当 <code>WINEN</code> 位为 1，比较比较条件为: <code>WDBLT[15:0] &lt; A/D 转换值 &lt; WDBHT[15:0]</code>

### 枚举类型 `adc_oversample_mode_enum`

表 3-13. 枚举类型 `adc_oversample_mode_enum`

成员名称	功能描述
<code>ADC_ACCUMULATION_MODE</code>	数据累加模式
<code>ADC_AVERAGE_MODE</code>	数据平均模式

### 枚举类型 `adc_interrupt_enum`

表 3-14. 枚举类型 `adc_interrupt_enum`

成员名称	功能描述
<code>ADC_INT_EOC1RF</code>	EOC1RF 中断
<code>ADC_INT_EOC2RF</code>	EOC2RF 中断
<code>ADC_INT_EOC3RF</code>	EOC3RF 中断
<code>ADC_INT_EOC4RF</code>	EOC4RF 中断
<code>ADC_INT_WDA_CHSTAT</code>	模拟看门狗 A 中断
<code>ADC_INT_WDBEF</code>	模拟看门狗 B 中断
<code>ADC_INT_GP1OVRF</code>	Group_pri1 组 DMA 溢出检测中断
<code>ADC_INT_GP2OVRF</code>	Group_pri2 组 DMA 溢出检测中断
<code>ADC_INT_GP3OVRF</code>	Group_pri3 组 DMA 溢出检测中断
<code>ADC_INT_GP4OVRF</code>	Group_pri4 组 DMA 溢出检测中断

## 枚举类型 `adc_interrupt_flag_enum`

表 3-15. 枚举类型 `adc_interrupt_flag_enum`

成员名称	功能描述
<code>ADC_INT_FLAG_EOC1RF</code>	EOC1RF 中断
<code>ADC_INT_FLAG_EOC2RF</code>	EOC2RF 中断
<code>ADC_INT_FLAG_EOC3RF</code>	EOC3RF 中断
<code>ADC_INT_FLAG_EOC4RF</code>	EOC4RF 中断
<code>ADC_INT_FLAG_WDA_CHSTAT</code>	模拟看门狗 A 中断
<code>ADC_INT_FLAG_WDBEF</code>	模拟看门狗 B 中断
<code>ADC_INT_FLAG_GP1OVRF</code>	Group_pri1 组 DMA 溢出检测中断
<code>ADC_INT_FLAG_GP2OVRF</code>	Group_pri2 组 DMA 溢出检测中断
<code>ADC_INT_FLAG_GP3OVRF</code>	Group_pri3 组 DMA 溢出检测中断
<code>ADC_INT_FLAG_GP4OVRF</code>	Group_pri4 组 DMA 溢出检测中断

## 枚举类型 `adc_event_flag_enum`

表 3-16. 枚举类型 `adc_event_flag_enum`

成员名称	功能描述
<code>ADC_FLAG_EOC1F</code>	Group_pri1 转换结束标志
<code>ADC_FLAG_EOC1RF</code>	Group_pri1 转换轮次结束标志
<code>ADC_FLAG_EOC2F</code>	Group_pri2 转换结束标志
<code>ADC_FLAG_EOC2RF</code>	Group_pri2 转换轮次结束标志
<code>ADC_FLAG_EOC3F</code>	Group_pri3 转换结束标志
<code>ADC_FLAG_EOC3RF</code>	Group_pri3 转换轮次结束标志
<code>ADC_FLAG_EOC4F</code>	Group_pri4 转换结束标志
<code>ADC_FLAG_EOC4RF</code>	Group_pri4 转换轮次结束标志
<code>ADC_FLAG_PROC</code>	A/D 转换正在进行标识
<code>ADC_FLAG_WDAMF</code>	模拟看门狗 A 比较监视标志
<code>ADC_FLAG_WDBMF</code>	模拟看门狗 B 比较监视标志
<code>ADC_FLAG_WDABMF</code>	模拟看门狗 A/B 复合比较监视标志
<code>ADC_FLAG_GP1OVRF</code>	Group_pri1 组 DMA 溢出标志
<code>ADC_FLAG_GP2OVRF</code>	Group_pri2 组 DMA 溢出标志
<code>ADC_FLAG_GP3OVRF</code>	Group_pri3 组 DMA 溢出标志
<code>ADC_FLAG_GP4OVRF</code>	Group_pri4 组 DMA 溢出标志
<code>ADC_FLAG_WDBEF</code>	模拟看门狗 B 事件标志
<code>ADC_FLAG_WDA_CH0CMPF</code>	模拟看门狗 A 中的 ADCx_IN00 通道比较状态
<code>ADC_FLAG_WDA_CH1CMPF</code>	模拟看门狗 A 中的 ADCx_IN01 通道比较状态
<code>ADC_FLAG_WDA_CH2CMPF</code>	模拟看门狗 A 中的 ADCx_IN02 通道比较状态
<code>ADC_FLAG_WDA_CH3CMPF</code>	模拟看门狗 A 中的 ADCx_IN03 通道比较状态
<code>ADC_FLAG_WDA_CH4CMPF</code>	模拟看门狗 A 中的 ADCx_IN04 通道比较状态
<code>ADC_FLAG_WDA_CH5CMPF</code>	模拟看门狗 A 中的 ADCx_IN05 通道比较状态



ADC_FLAG_WDA_CH6CMPF	模拟看门狗 A 中的 ADCx_IN06 通道比较状态
ADC_FLAG_WDA_CH7CMPF	模拟看门狗 A 中的 ADCx_IN07 通道比较状态
ADC_FLAG_WDA_CH8CMPF	模拟看门狗 A 中的 ADCx_IN08 通道比较状态
ADC_FLAG_WDA_CH9CMPF	模拟看门狗 A 中的 ADCx_IN09 通道比较状态
ADC_FLAG_WDA_TEMPCMPF	模拟看门狗 A 中的温度传感器通道比较状态
ADC_FLAG_WDA_VINTCMPF	模拟看门狗 A 中的内部参考电压通道比较状态
ADC_FLAG_WDA_ALL_CHCMPF	模拟看门狗 A 所有通道比较状态

### 枚举类型 `adc_evic_link_source_enum`

表 3-17. 枚举类型 `adc_evic_link_source_enum`

成员名称	功能描述
ADC_EVIC_LINK_SOURCE_EOCR_F_FLAG	EVIC 触发信号选择 EOCR_F (x=1,2,3,4)
ADC_EVIC_LINK_SOURCE_EOC_F_FLAG	EVIC 触发信号选择 EOC_F (x=1,2,3,4)

### 函数 `adc_deinit`

函数 `adc_deinit` 描述见下表

表 3-18. 函数 `adc_deinit`

函数名称	<code>adc_deinit</code>
函数原型	<code>void adc_deinit(uint32_t adc_periph)</code>
功能描述	复位 ADCx 外设
先决条件	-
被调用函数	-
输入参数{in}	
<code>adc_periph</code>	ADC 外设
<code>ADCx</code>	x=0,2
输出参数{out}	
-	-
返回值	
-	-

Example:

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

### 函数 `adc_enable`

函数 `adc_enable` 描述见下表

表 3-19. 函数 `adc_enable`

函数名称	<code>adc_enable</code>
函数原型	<code>void adc_enable(uint32_t adc_periph)</code>

功能描述	使能 ADCx 外设
先决条件	-
被调用函数	-
输入参数{in}	
adc_periph	ADC 外设
ADCx	x=0,2
输出参数{out}	
-	-
返回值	
-	-

Example:

```
/* enable ADC0 */
adc_enable(ADC0);
```

### 函数 adc\_disable

函数adc\_disable描述见下表

表 3-20. 函数 adc\_disable

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

Example:

```
/* disable ADC0 */
adc_disable(ADC0);
```

### 函数 adc\_data\_alignment\_config

函数adc\_data\_alignment\_config描述见下表

表 3-21. 函数 adc\_data\_alignment\_config

函数名称	adc_data_alignment_config
------	---------------------------

函数原型	void adc_data_alignment_config(uint32_t adc_periph, uint32_t data_alignment)
功能描述	配置 ADCx 数据对齐方式
先决条件	-
被调用函数	-
输入参数{in}	
adc_periph	ADC 外设
ADCx	x=0,2
输入参数{in}	
data_alignment	数据对齐选择
ADC_DATAALIGN_RIGHT	右对齐(LSB)
ADC_DATAALIGN_LEFT	左对齐(MSB)
输出参数{out}	
-	-
返回值	
-	-

Example:

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

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

### 函数 adc\_resolution\_config

函数adc\_resolution\_config描述见下表

表 3-22. 函数 adc\_resolution\_config

函数名称	adc_resolution_config
函数原型	void adc_resolution_config(uint32_t adc_periph, uint32_t resolution)
功能描述	配置 ADCx 分辨率
先决条件	-
被调用函数	-
输入参数{in}	
adc_periph	ADC 外设
ADCx	x=0,2
输入参数{in}	
resolution	ADC 分辨率
ADC_RESOLUTION_12B	12 位分辨率
ADC_RESOLUTION_10B	10 位分辨率

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

Example:

```
/* configure ADC resolution */
adc_resolution_config(ADC0, ADC_RESOLUTION_10B);
```

### 函数 adc\_self\_diagnosis\_enable

函数adc\_self\_diagnosis\_enable描述见下表

表 3-23. 函数 adc\_self\_diagnosis\_enable

函数名称	adc_self_diagnosis_enable
函数原型	void adc_self_diagnosis_enable(uint32_t adc_periph)
功能描述	使能自诊断
先决条件	-
被调用函数	-
输入参数{in}	
adc_periph	ADC 外设
ADCx	x=0,2
输出参数{out}	
-	-
返回值	
-	-

Example:

```
/* enable self-diagnosis */
adc_self_diagnosis_enable(ADC0);
```

### 函数 adc\_self\_diagnosis\_disable

函数adc\_self\_diagnosis\_disable描述见下表

表 3-24. 函数 adc\_self\_diagnosis\_disable

函数名称	adc_self_diagnosis_disable
函数原型	void adc_self_diagnosis_disable(uint32_t adc_periph)
功能描述	禁能自诊断

先决条件	-
被调用函数	-
输入参数{in}	
adc_periph	ADC 外设
ADCx	x=0,2
输出参数{out}	
-	-
返回值	
-	-

Example:

```
/* enable self-diagnosis */
```

```
adc_self_diagnosis_disable(ADC0);
```

### 函数 adc\_self\_diagnosis\_mode\_config

函数adc\_self\_diagnosis\_mode\_config描述见下表

表 3-25. 函数 adc\_self\_diagnosis\_mode\_config

函数名称	adc_self_diagnosis_mode_config
函数原型	void adc_self_diagnosis_mode_config(uint32_t adc_periph, adc_self_diag_mode_enum mode, adc_self_diag_fixed_voltage_enum voltage_select)
功能描述	自诊断配置
先决条件	-
被调用函数	-
输入参数{in}	
adc_periph	ADC 外设
ADCx	x=0,2
输入参数{in}	
mode	自诊断模式选择, 请参考 <a href="#">枚举类型 adc_self_diag_mode_enum</a>
输入参数{in}	
voltage_select	配置自诊断电压, 请参考 <a href="#">枚举类型 adc_self_diag_fixed_voltage_enum</a> 。
输出参数{out}	
-	-
返回值	
-	-

Example:

```
/* config self-diagnosis */
```

```
adc_self_diagnosis_mode_config (ADC0, ADC_SELF_DIAGNOSIS_MODE_FIXED,
```

```
ADC_SELF_DIAG_1_2_VREF);
```

### 函数 `adc_disconnect_detect_mode_config`

函数 `adc_disconnect_detect_mode_config` 描述见下表

表 3-26. 函数 `adc_disconnect_detect_mode_config`

函数名称	<code>adc_disconnect_detect_mode_config</code>
函数原型	<code>void adc_disconnect_detect_mode_config(uint32_t adc_periph, adc_disc_detect_mode_enum mode)</code>
功能描述	配置断开检测辅助模式
先决条件	-
被调用函数	-
输入参数{in}	
<code>adc_periph</code>	ADC 外设
<code>ADCx</code>	x=0,2
输入参数{in}	
<code>mode</code>	辅助断开检测模式选择，请参考 <a href="#">枚举类型 <code>adc_disc_detect_mode_enum</code></a> 。
输出参数{out}	
-	-
返回值	
-	-

Example:

```
/* configure disconnection detection assist mode */
```

```
adc_disconnect_detect_mode_config (ADC0, ADC_DISCHARGE_MODE);
```

### 函数 `adc_disconnect_detect_period_config`

函数 `adc_disconnect_detect_period_config` 描述见下表

表 3-27. 函数 `adc_disconnect_detect_period_config`

函数名称	<code>adc_disconnect_detect_period_config</code>
函数原型	<code>void adc_disconnect_detect_period_config(uint32_t adc_periph, uint32_t period)</code>
功能描述	配置断开检测时长
先决条件	-
被调用函数	-
输入参数{in}	
<code>adc_periph</code>	ADC 外设
<code>ADCx</code>	x=0,2
输入参数{in}	

period	断开检测时长, (0x0~0F)
输出参数{out}	
-	-
返回值	
-	-

Example:

```
/* configure disconnect detect period */
adc_disconnect_detect_period_config (ADC0, 0x3);
```

### 函数 adc\_data\_auto\_clear\_enable

函数adc\_data\_auto\_clear\_enable描述见下表

表 3-28. 函数 adc\_data\_auto\_clear\_enable

函数名称	adc_data_auto_clear_enable
函数原型	void adc_data_auto_clear_enable(uint32_t adc_periph)
功能描述	使能数据寄存器自动清零
先决条件	-
被调用函数	-
输入参数{in}	
adc_periph	ADC 外设
ADCx	x=0,2
输出参数{out}	
-	-
返回值	
-	-

Example:

```
/* enable automatic clearing data registers */
adc_data_auto_clear_enable (ADC0);
```

### 函数 adc\_data\_auto\_clear\_disable

函数adc\_data\_auto\_clear\_disable描述见下表

表 3-29. 函数 adc\_data\_auto\_clear\_disable

函数名称	adc_data_auto_clear_disable
函数原型	void adc_data_auto_clear_disable(uint32_t adc_periph)
功能描述	禁能数据寄存器自动清零
先决条件	-
被调用函数	-
输入参数{in}	



adc_periph	ADC 外设
ADCx	x=0,2
输出参数{out}	
-	-
返回值	
-	-

Example:

```
/* disable automatic clearing data registers */
```

```
adc_data_auto_clear_disable (ADC0);
```

### 函数 adc\_data\_auto\_set\_enable

函数adc\_data\_auto\_set\_enable描述见下表

表 3-30. 函数 adc\_data\_auto\_set\_enable

函数名称	adc_data_auto_set_enable
函数原型	void adc_data_auto_set_enable(uint32_t adc_periph)
功能描述	使能数据寄存器自动置位
先决条件	-
被调用函数	-
输入参数{in}	
adc_periph	ADC 外设
ADCx	x=0,2
输出参数{out}	
-	-
返回值	
-	-

Example:

```
/* enable automatic setting data registers */
```

```
adc_data_auto_set_enable (ADC0);
```

### 函数 adc\_data\_auto\_set\_disable

函数adc\_data\_auto\_set\_disable描述见下表

表 3-31. 函数 adc\_data\_auto\_set\_disable

函数名称	adc_data_auto_set_disable
函数原型	void adc_data_auto_set_disable(uint32_t adc_periph)
功能描述	禁能数据寄存器自动置位
先决条件	-
被调用函数	-

输入参数{in}	
<b>adc_periph</b>	ADC 外设
<i>ADCx</i>	x=0,2
输出参数{out}	
-	-
返回值	
-	-

Example:

```
/* disable automatic setting data registers */
```

```
adc_data_auto_set_disable (ADC0);
```

### 函数 **adc\_sample\_hold\_channel\_config**

函数 **adc\_sample\_hold\_channel\_config** 描述见下表

表 3-32. 函数 **adc\_sample\_hold\_channel\_config**

函数名称	adc_sample_hold_channel_config
函数原型	void adc_sample_hold_channel_config(uint32_t channel)
功能描述	配置 ADC 采保通道
先决条件	-
被调用函数	-
输入参数{in}	
<b>channel</b>	选择采保通道
<i>ADC_SH_CHANNEL_IN00</i>	使用 ADC0_IN00 通道采保电路
<i>ADC_SH_CHANNEL_IN01</i>	使用 ADC0_IN01 通道采保电路
<i>ADC_SH_CHANNEL_IN02</i>	使用 ADC0_IN02 通道采保电路
输入参数{in}	
<b>ctl</b>	控制状态
<i>ENABLE</i>	使能通道采保电路
<i>DISABLE</i>	禁能通道采保电路
输出参数{out}	
-	-
返回值	
-	-

Example:

```
/* configure ADC sample-and-hold channel */
```

```
adc_sample_hold_channel_config (ADC_SH_CHANNEL_IN00);
```

## 函数 adc\_sh\_sample\_time\_config

函数adc\_sh\_sample\_time\_config描述见下表

表 3-33. 函数 adc\_sh\_sample\_time\_config

函数名称	adc_sh_sample_time_config
函数原型	void adc_sh_sample_time_config(uint8_t sample_time)
功能描述	配置 ADC 采保电路的采样时间
先决条件	-
被调用函数	-
输入参数{in}	
sample_time	0x08~0xFF。推荐配置不小于 8。
输出参数{out}	
-	-
返回值	
-	-

Example:

```
/* configure ADC sample time for sample-and-hold circuits */
```

```
adc_sh_sample_time_config (12);
```

## 函数 adc\_sh\_hold\_time\_config

函数adc\_sh\_hold\_time\_config描述见下表

表 3-34. 函数 adc\_sh\_hold\_time\_config

函数名称	adc_sh_hold_time_config
函数原型	void adc_sh_hold_time_config(uint8_t hold_time)
功能描述	配置 ADC 采保电路的保持时间
先决条件	-
被调用函数	-
输入参数{in}	
sample_time	0x8~0xF。推荐配置不小于 8。
输出参数{out}	
-	-
返回值	
-	-

Example:

```
/* configure ADC hold time for sample-and-hold circuits */
```

```
adc_sh_hold_time_config (12);
```

### 函数 `adc_sh_constant_sampling_mode_enable`

函数 `adc_sh_constant_sampling_mode_enable` 描述见下表

**表 3-35. 函数 `adc_sh_constant_sampling_mode_enable`**

函数名称	<code>adc_sh_constant_sampling_mode_enable</code>
函数原型	<code>void adc_sh_constant_sampling_mode_enable(void)</code>
功能描述	使能 ADC 采保电路的恒采样模式
先决条件	-
被调用函数	-
输入参数{in}	
-	-
输出参数{out}	
-	-
返回值	
-	-

Example:

```
/* enable ADC constant sampling mode for sample-and-hold circuits */
adc_sh_constant_sampling_mode_enable ();
```

### 函数 `adc_sh_constant_sampling_mode_disable`

函数 `adc_sh_constant_sampling_mode_disable` 描述见下表

**表 3-36. 函数 `adc_sh_constant_sampling_mode_disable`**

函数名称	<code>adc_sh_constant_sampling_mode_disable</code>
函数原型	<code>void adc_sh_constant_sampling_mode_disable(void)</code>
功能描述	禁能 ADC 采保电路的恒采样模式
先决条件	-
被调用函数	-
输入参数{in}	
-	-
输出参数{out}	
-	-
返回值	
-	-

Example:

```
/* disable ADC constant sampling mode for sample-and-hold circuits */
adc_sh_constant_sampling_mode_disable ();
```

## 函数 adc\_sh\_constant\_sampling\_start

函数adc\_sh\_constant\_sampling\_start描述见下表

**表 3-37. 函数 adc\_sh\_constant\_sampling\_start**

函数名称	adc_sh_constant_sampling_start
函数原型	void adc_sh_constant_sampling_start(void)
功能描述	软件启动采保电路的恒采样
先决条件	-
被调用函数	-
输入参数{in}	
-	-
输出参数{out}	
-	-
返回值	
-	-

Example:

```
/* software start sample-and-hold circuit in constant sampling mode */
```

```
adc_sh_constant_sampling_start ();
```

## 函数 adc\_sh\_constant\_sampling\_stop

函数adc\_sh\_constant\_sampling\_stop描述见下表

**表 3-38. 函数 adc\_sh\_constant\_sampling\_stop**

函数名称	adc_sh_constant_sampling_stop
函数原型	void adc_sh_constant_sampling_stop(void)
功能描述	软件结束采保电路的恒采样
先决条件	-
被调用函数	-
输入参数{in}	
-	-
输出参数{out}	
-	-
返回值	
-	-

Example:

```
/* software start sample-and-hold circuit in constant sampling mode */
```

```
adc_sh_constant_sampling_stop ();
```

## 函数 adc\_watchdog\_enable

函数adc\_watchdog\_enable描述见下表

表 3-39. 函数 adc\_watchdog\_enable

函数名称	adc_watchdog_enable
函数原型	void adc_watchdog_enable(uint32_t adc_periph, adc_watchdog_select_enum window)
功能描述	使能 ADC 模拟看门狗
先决条件	-
被调用函数	-
输入参数{in}	
adc_periph	ADC 外设
ADCx	x=0,2
输入参数{in}	
window	选择模拟看门狗，请参考 <a href="#">枚举类型 adc_watchdog_select_enum</a> 。
输出参数{out}	
-	-
返回值	
-	-

Example:

```
/* enable ADC analog watchdog */
adc_watchdog_enable (ADC0, ADC_WATCHDOG_A);
```

## 函数 adc\_watchdog\_disable

函数adc\_watchdog\_disable描述见下表

表 3-40. 函数 adc\_watchdog\_disable

函数名称	adc_watchdog_disable
函数原型	void adc_watchdog_disable(uint32_t adc_periph, adc_watchdog_select_enum window)
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}	
adc_periph	ADC 外设
ADCx	x=0,2
输入参数{in}	
window	选择模拟看门狗，请参考 <a href="#">枚举类型 adc_watchdog_select_enum</a> 。
输出参数{out}	
-	-

返回值	
-	-

Example:

```
/* disableADC analog watchdog */
```

```
adc_watchdog_disable (ADC0, ADC_WATCHDOG_A);
```

### 函数 `adc_watchdog_a_channel_config`

函数`adc_watchdog_a_channel_config`描述见下表

**表 3-41. 函数 `adc_watchdog_a_channel_config`**

函数名称	<code>adc_watchdog_a_channel_config</code>
函数原型	<code>void adc_watchdog_a_channel_config(uint32_t adc_periph, adc_channel_select_enum channel, adc_watchdog_compare_condition_enum compare_mode)</code>
功能描述	配置 ADC 模拟看门狗 A 的通道
先决条件	-
被调用函数	-
输入参数{in}	
<b>adc_periph</b>	ADC 外设
<b>ADCx</b>	x=0,2
输入参数{in}	
<b>channel</b>	选择 ADC 通道, 请参考 <a href="#">枚举类型 <code>adc_channel_select_enum</code></a> 。
输入参数{in}	
<b>compare_mode</b>	模拟看门狗比较条件, 请参考 <a href="#">枚举类型 <code>adc_watchdog_compare_condition_enum</code></a> 。
输出参数{out}	
-	-
返回值	
-	-

Example:

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

```
adc_watchdog_a_channel_config (ADC0, ADC_CHANNEL_IN00, ADC_IN_WINDOW);
```

### 函数 `adc_watchdog_a_channel_deselect`

函数`adc_watchdog_a_channel_deselect`描述见下表

**表 3-42. 函数 `adc_watchdog_a_channel_deselect`**

函数名称	<code>adc_watchdog_a_channel_deselect</code>
函数原型	<code>void adc_watchdog_a_channel_deselect(uint32_t adc_periph,</code>



	adc_channel_select_enum channel)
功能描述	取消 ADC 模拟看门狗 A 的通道选择
先决条件	-
被调用函数	-
输入参数{in}	
adc_periph	ADC 外设
ADCx	x=0,2
输入参数{in}	
channel	选择 ADC 通道, 请参考 <a href="#">枚举类型 adc_channel_select_enum</a> 。
输出参数{out}	
-	-
返回值	
-	-

Example:

```
/* deselect ADC analog watchdog A channel */
```

```
adc_watchdog_a_channel_deselect (ADC0, ADC_CHANNEL_IN00);
```

### 函数 adc\_watchdog\_b\_channel\_config

函数adc\_watchdog\_b\_channel\_config描述见下表

表 3-43. 函数 adc\_watchdog\_b\_channel\_config

函数名称	adc_watchdog_b_channel_config
函数原型	void adc_watchdog_b_channel_config(uint32_t adc_periph, adc_channel_select_enum channel, adc_watchdog_compare_condition_enum compare_mode)
功能描述	配置 ADC 模拟看门狗 B 的通道
先决条件	-
被调用函数	-
输入参数{in}	
adc_periph	ADC 外设
ADCx	x=0,2
输入参数{in}	
channel	选择 ADC 通道, 请参考 <a href="#">枚举类型 adc_channel_select_enum</a> 注意: ADC_CHANNEL_ALL 除外
输入参数{in}	
compare_mode	模拟看门狗比较条件, 请参考 <a href="#">枚举类型 adc_watchdog_compare_condition_enum</a> 。
输出参数{out}	
-	-
返回值	

-	-
---	---

Example:

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

```
adc_watchdog_b_channel_config (ADC0, ADC_CHANNEL_IN00, ADC_IN_WINDOW);
```

### 函数 `adc_watchdog_a_threshold_config`

函数 `adc_watchdog_a_threshold_config` 描述见下表

表 3-44. 函数 `adc_watchdog_a_threshold_config`

函数名称	<code>adc_watchdog_a_threshold_config</code>
函数原型	<code>void adc_watchdog_a_threshold_config(uint32_t adc_periph, uint16_t low_threshold, uint16_t high_threshold)</code>
功能描述	配置 ADC 模拟看门狗 A 的阈值
先决条件	-
被调用函数	-
输入参数{in}	
<code>adc_periph</code>	ADC 外设
<code>ADCx</code>	x=0,2
输入参数{in}	
<code>low_threshold</code>	模拟看门狗低阈值, 0x0000..0xFFFF
输入参数{in}	
<code>high_threshold</code>	模拟看门狗高阈值, 0x0000..0xFFFF
输出参数{out}	
-	-
返回值	
-	-

Example:

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

```
adc_watchdog_a_threshold_config (ADC0, 0x1F3F, 0x2FFF);
```

### 函数 `adc_watchdog_b_threshold_config`

函数 `adc_watchdog_b_threshold_config` 描述见下表

表 3-45. 函数 `adc_watchdog_b_threshold_config`

函数名称	<code>adc_watchdog_b_threshold_config</code>
函数原型	<code>void adc_watchdog_b_threshold_config(uint32_t adc_periph, uint16_t low_threshold, uint16_t high_threshold)</code>
功能描述	配置 ADC 模拟看门狗 B 的阈值
先决条件	-

被调用函数	-
输入参数{in}	
adc_periph	ADC 外设
ADCx	x=0,2
输入参数{in}	
low_threshold	模拟看门狗低阈值, 0x0000..0xFFFF
输入参数{in}	
high_threshold	模拟看门狗高阈值, 0x0000..0xFFFF
输出参数{out}	
-	-
返回值	
-	-

Example:

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

```
adc_watchdog_b_threshold_config (ADC0, 0x1F3F, 0x2FFF);
```

### 函数 adc\_watchdog\_window\_mode\_enable

函数adc\_watchdog\_window\_mode\_enable描述见下表

表 3-46. 函数 adc\_watchdog\_window\_mode\_enable

函数名称	adc_watchdog_window_mode_enable
函数原型	void adc_watchdog_window_mode_enable(uint32_t adc_periph)
功能描述	使能 ADC 模拟看门狗窗口功能
先决条件	-
被调用函数	-
输入参数{in}	
adc_periph	ADC 外设
ADCx	x=0,2
输出参数{out}	
-	-
返回值	
-	-

Example:

```
/* enable ADC analog watchdog window function */
```

```
adc_watchdog_window_mode_enable (ADC0);
```

### 函数 adc\_watchdog\_window\_mode\_disable

函数adc\_watchdog\_window\_mode\_disable描述见下表

表 3-47. 函数 `adc_watchdog_window_mode_disable`

函数名称	<code>adc_watchdog_window_mode_disable</code>
函数原型	<code>void adc_watchdog_window_mode_disable(uint32_t adc_periph)</code>
功能描述	禁能 ADC 模拟看门狗窗口功能
先决条件	-
被调用函数	-
输入参数{in}	
<code>adc_periph</code>	ADC 外设
<code>ADCx</code>	x=0,2
输出参数{out}	
-	-
返回值	
-	-

Example:

```
/* disable ADC analog watchdog window function */
```

```
adc_watchdog_window_mode_disable (ADC0);
```

### 函数 `adc_watchdog_complex_condition_config`

函数`adc_watchdog_complex_condition_config`描述见下表

表 3-48. 函数 `adc_watchdog_complex_condition_config`

函数名称	<code>adc_watchdog_complex_condition_config</code>
函数原型	<code>void adc_watchdog_complex_condition_config(uint32_t adc_periph, uint32_t mode)</code>
功能描述	配置模拟看门狗复合比较监视标志
先决条件	-
被调用函数	-
输入参数{in}	
<code>adc_periph</code>	ADC 外设
<code>ADCx</code>	x=0,2
输入参数{in}	
<code>mode</code>	模拟看门狗 A/B 复合条件配置
<code>ADC_WATCHDOG_MATCH_A_OR_B</code>	(模拟看门狗 A 条件匹配) OR (模拟看门狗 B 条件匹配)
<code>ADC_WATCHDOG_MATCH_A_XOR_B</code>	(模拟看门狗 A 条件匹配) XOR (模拟看门狗 B 条件匹配)
<code>ADC_WATCHDOG_MATCH_A_AND_B</code>	(模拟看门狗 A 条件匹配) AND (模拟看门狗 B 条件匹配)

输出参数{out}	
-	-
返回值	
-	-

Example:

```
/* configure ADC analog watchdog complex conditions */
```

```
adc_watchdog_complex_condition_config (ADC0, ADC_WATCHDOG_MATCH_A_XOR_B);
```

### 函数 `adc_oversample_channel_config`

函数 `adc_oversample_channel_config` 描述见下表

表 3-49. 函数 `adc_oversample_channel_config`

函数名称	<code>adc_oversample_channel_config</code>
函数原型	<code>void adc_oversample_channel_config(uint32_t adc_periph, adc_channel_select_enum channel, uint32_t ratio)</code>
功能描述	配置 ADC 过采样通道
先决条件	-
被调用函数	-
输入参数{in}	
<b>adc_periph</b>	ADC 外设
<i>ADCx</i>	x=0,2
输入参数{in}	
<b>channel</b>	所选择的通道，请参考 <a href="#">枚举类型 <code>adc_channel_select_enum</code></a> 。 注意：ADC_CHANNEL_ALL 除外
输入参数{in}	
<b>ratio</b>	ADC 过采样率
<code>ADC_OVERSAMPLING_RATIO_MUL_1</code>	1x
<code>ADC_OVERSAMPLING_RATIO_MUL_2</code>	2x
<code>ADC_OVERSAMPLING_RATIO_MUL_3</code>	3x
<code>ADC_OVERSAMPLING_RATIO_MUL_4</code>	4x
<code>ADC_OVERSAMPLING_RATIO_MUL_8</code>	8x

ADC_OVERSAMPLING_RATIO_MUL_16	16x
ADC_OVERSAMPLING_RATIO_MUL_32	32x
ADC_OVERSAMPLING_RATIO_MUL_64	64x
输出参数{out}	
-	-
返回值	
-	-

Example:

```
/* configure ADC oversample channel */
```

```
adc_oversample_channel_config(ADC0, ADC_CHANNEL_IN00,
ADC_OVERSAMPLING_RATIO_MUL_2);
```

### 函数 adc\_oversample\_mode\_config

函数adc\_oversample\_mode\_config描述见下表

表 3-50. 函数 adc\_oversample\_mode\_config

函数名称	adc_oversample_mode_config
函数原型	void adc_oversample_mode_config(uint32_t adc_periph, adc_oversample_mode_enum mode)
功能描述	配置 ADC 过采样模式
先决条件	-
被调用函数	-
输入参数{in}	
adc_periph	ADC 外设
ADCx	x=0,2
输入参数{in}	
mode	A/D 转换数据加法模式选择, 请参考 <a href="#">枚举类型 adc_oversample_mode_enum</a> 。
输出参数{out}	
-	-
返回值	
-	-

Example:

```
/* configure ADC oversample mode */
```

adc\_oversample\_mode\_config (ADC0, ADC\_AVERAGE\_MODE);

### 函数 adc\_oversample\_mode\_enable

函数adc\_oversample\_mode\_enable描述见下表

表 3-51. 函数 adc\_oversample\_mode\_enable

函数名称	adc_oversample_mode_enable
函数原型	void adc_oversample_mode_enable(uint32_t adc_periph)
功能描述	使能 ADC 过采样模式
先决条件	-
被调用函数	-
输入参数{in}	
adc_periph	ADC 外设
ADCx	x=0,2
输出参数{out}	
-	-
返回值	
-	-

Example:

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

```
adc_oversample_mode_enable (ADC0);
```

### 函数 adc\_oversample\_mode\_disable

函数adc\_oversample\_mode\_disable描述见下表

表 3-52. 函数 adc\_oversample\_mode\_disable

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

Example:

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

```
adc_oversample_mode_disable (ADC0);
```

### 函数 `adc_group_scan_mode_config`

函数 `adc_group_scan_mode_config` 描述见下表

**表 3-53. 函数 `adc_group_scan_mode_config`**

函数名称	<code>adc_group_scan_mode_config</code>
函数原型	<code>void adc_group_scan_mode_config(uint32_t adc_periph, uint32_t scan_mode)</code>
功能描述	配置 ADC 扫描模式
先决条件	-
被调用函数	-
输入参数{in}	
<code>adc_periph</code>	ADC 外设
<code>ADCx</code>	<code>x=0,2</code>
输入参数{in}	
<code>scan_mode</code>	ADC 扫描模式
<code>ADC_GROUP_PRI1_SCAN_ONCE</code>	Group_pri1 单次扫描模式
<code>ADC_GROUPS_SCAN</code>	多组扫描模式
<code>ADC_GROUP_PRI1_SCAN_CONTINUOUS</code>	Group_pri1 连续扫描模式
输出参数{out}	
-	-
返回值	
-	-

Example:

```
/* configure the ADC scan mode */
```

```
adc_oversample_mode_disable (ADC0, ADC_GROUP_PRI1_SCAN_ONCE);
```

### 函数 `adc_group_priority_control_enable`

函数 `adc_group_priority_control_enable` 描述见下表

**表 3-54. 函数 `adc_group_priority_control_enable`**

函数名称	<code>adc_group_priority_control_enable</code>
函数原型	<code>void adc_group_priority_control_enable(uint32_t adc_periph)</code>
功能描述	使能组优先级控制
先决条件	-



被调用函数	-
输入参数{in}	
adc_periph	ADC 外设
ADCx	x=0,2
输出参数{out}	
-	-
返回值	
-	-

Example:

```
/* enable group priority control */
adc_group_priority_control_enable (ADC0);
```

### 函数 adc\_group\_priority\_control\_disable

函数adc\_group\_priority\_control\_disable描述见下表

表 3-55. 函数 adc\_group\_priority\_control\_disable

函数名称	adc_group_priority_control_disable
函数原型	void adc_group_priority_control_disable(uint32_t adc_periph)
功能描述	禁能组优先级控制
先决条件	-
被调用函数	-
输入参数{in}	
adc_periph	ADC 外设
ADCx	x=0,2
输出参数{out}	
-	-
返回值	
-	-

Example:

```
/* disable group priority control */
adc_group_priority_control_disable (ADC0);
```

### 函数 adc\_group\_pri3\_enable

函数adc\_group\_pri3\_enable描述见下表

表 3-56. 函数 adc\_group\_pri3\_enable

函数名称	adc_group_pri3_enable
函数原型	void adc_group_pri3_enable(uint32_t adc_periph)
功能描述	使能 Group_pri3

先决条件	-
被调用函数	-
输入参数{in}	
adc_periph	ADC 外设
ADCx	x=0,2
输出参数{out}	
-	-
返回值	
-	-

Example:

```
/* enable A/D conversion operation for Group_pri3 */
```

```
adc_group_pri3_enable (ADC0);
```

### 函数 adc\_group\_pri3\_disable

函数adc\_group\_pri3\_disable描述见下表

表 3-57. 函数 adc\_group\_pri3\_disable

函数名称	adc_group_pri3_disable
函数原型	void adc_group_pri3_disable(uint32_t adc_periph)
功能描述	禁能 Group_pri3
先决条件	-
被调用函数	-
输入参数{in}	
adc_periph	ADC 外设
ADCx	x=0,2
输出参数{out}	
-	-
返回值	
-	-

Example:

```
/* disable A/D conversion operation for Group_pri3 */
```

```
adc_group_pri3_disable (ADC0);
```

### 函数 adc\_group\_pri4\_enable

函数adc\_group\_pri4\_enable描述见下表

表 3-58. 函数 adc\_group\_pri4\_enable

函数名称	adc_group_pri4_enable
函数原型	void adc_group_pri4_enable(uint32_t adc_periph)

功能描述	使能 Group_pri4
先决条件	-
被调用函数	-
输入参数{in}	
adc_periph	ADC 外设
ADCx	x=0,2
输出参数{out}	
-	-
返回值	
-	-

Example:

```
/* enable A/D conversion operation for Group_pri4 */
```

```
adc_group_pri4_enable (ADC0);
```

### 函数 adc\_group\_pri4\_disable

函数adc\_group\_pri4\_disable描述见下表

表 3-59. 函数 adc\_group\_pri4\_disable

函数名称	adc_group_pri4_disable
函数原型	void adc_group_pri4_disable(uint32_t adc_periph)
功能描述	禁能 Group_pri4
先决条件	-
被调用函数	-
输入参数{in}	
adc_periph	ADC 外设
ADCx	x=0,2
输出参数{out}	
-	-
返回值	
-	-

Example:

```
/* disable A/D conversion operation for Group_pri4 */
```

```
adc_group_pri4_disable (ADC0);
```

### 函数 adc\_group\_lowest\_priority\_continuous\_enable

函数adc\_group\_lowest\_priority\_continuous\_enable描述见下表

表 3-60. 函数 adc\_group\_lowest\_priority\_continuous\_enable

函数名称	adc_group_lowest_priority_continuous_enable
------	---

函数原型	void adc_group_lowest_priority_continuous_enable(uint32_t adc_periph)
功能描述	使能最低优先级组连续扫描
先决条件	-
被调用函数	-
输入参数{in}	
adc_periph	ADC 外设
ADCx	x=0,2
输出参数{out}	
-	-
返回值	
-	-

Example:

```
/* enable lowest-priority group scan continuous */
```

```
adc_group_lowest_priority_continuous_enable (ADC0);
```

### 函数 adc\_group\_lowest\_priority\_continuous\_disable

函数adc\_group\_lowest\_priority\_continuous\_disable描述见下表

表 3-61. 函数 adc\_group\_lowest\_priority\_continuous\_disable

函数名称	adc_group_lowest_priority_continuous_disable
函数原型	void adc_group_lowest_priority_continuous_disable(uint32_t adc_periph)
功能描述	禁能最低优先级组连续扫描
先决条件	-
被调用函数	-
输入参数{in}	
adc_periph	ADC 外设
ADCx	x=0,2
输出参数{out}	
-	-
返回值	
-	-

Example:

```
/* disable lowest-priority group scan continuous */
```

```
adc_group_lowest_priority_continuous_disable (ADC0);
```

### 函数 adc\_group\_restart\_enable

函数adc\_group\_restart\_enable描述见下表

表 3-62. 函数 `adc_group_restart_enable`

函数名称	<code>adc_group_restart_enable</code>
函数原型	<code>void adc_group_restart_enable(uint32_t adc_periph, adc_group_select_enum group, adc_restart_channel_enum restart_ch)</code>
功能描述	使能低优先组重启
先决条件	-
被调用函数	-
输入参数{in}	
<code>adc_periph</code>	ADC 外设
<code>ADCx</code>	<code>x=0,2</code>
输入参数{in}	
<code>group</code>	选择组，请参考 <a href="#">枚举类型 <code>adc_group_select_enum</code></a> 。 注意：不包含 <code>Group_pri1</code> 组
输入参数{in}	
<code>restart_ch</code>	重启通道选择，参考 <a href="#">枚举类型 <code>adc_restart_channel_enum</code></a> 。
输出参数{out}	
-	-
返回值	
-	-

Example:

```
/* enable low-priority group restart */
```

```
adc_group_restart_enable (ADC0, ADC_GROUP_PRI2, ADC_RESTART_ON_BEGINNING);
```

### 函数 `adc_group_restart_disable`

函数`adc_group_restart_disable`描述见下表

表 3-63. 函数 `adc_group_restart_disable`

函数名称	<code>adc_group_restart_disable</code>
函数原型	<code>void adc_group_restart_disable(uint32_t adc_periph, adc_group_select_enum group)</code>
功能描述	禁用低优先组重启
先决条件	-
被调用函数	-
输入参数{in}	
<code>adc_periph</code>	ADC 外设
<code>ADCx</code>	<code>x=0,2</code>
输入参数{in}	
<code>group</code>	选择组，请参考 <a href="#">枚举类型 <code>adc_group_select_enum</code></a> 。 注意：不包含 <code>Group_pri1</code> 组
输出参数{out}	

-	-
返回值	
-	-

Example:

```
/* disable low-priority group restart */
```

```
adc_group_restart_disable (ADC0, ADC_GROUP_PRI2);
```

### 函数 adc\_group\_channel\_config

函数adc\_group\_channel\_config描述见下表

表 3-64. 函数 adc\_group\_channel\_config

函数名称	adc_group_channel_config
函数原型	void adc_group_channel_config(uint32_t adc_periph, adc_group_select_enum group, adc_channel_select_enum channel, uint32_t sample_time)
功能描述	选择 Group_prix 组通道
先决条件	-
被调用函数	-
输入参数{in}	
adc_periph	ADC 外设
ADCx	x=0,2
输入参数{in}	
group	所选择的组，请参考 <a href="#">枚举类型 adc_group_select_enum</a> 。
输入参数{in}	
channel	所选择的通道，请参考 <a href="#">枚举类型 adc_channel_select_enum</a> 。
输入参数{in}	
sample_time	0x02~0xFF。最小采样时间是 0x02。
输出参数{out}	
-	-
返回值	
-	-

Example:

```
/* select ADC channel for Group_pri x */
```

```
adc_group_channel_config (ADC0, ADC_GROUP_PRI2, ADC_CHANNEL_IN00, 0x04);
```

### 函数 adc\_group\_channel\_deselect

函数adc\_group\_channel\_deselect描述见下表

表 3-65. 函数 adc\_group\_channel\_deselect

函数名称	adc_group_channel_deselect
------	----------------------------

函数原型	void adc_group_channel_deselect(uint32_t adc_periph, adc_group_select_enum group, adc_channel_select_enum channel)	
功能描述	取消 ADC 组通道的选择	
先决条件	-	
被调用函数	-	
输入参数{in}		
adc_periph	ADC 外设	
ADCx	x=0,2	
输入参数{in}		
group	所选择的组，请参考 <a href="#">枚举类型 adc_group_select_enum</a> 。	
输入参数{in}		
channel	所选择的通道，请参考 <a href="#">枚举类型 adc_channel_select_enum</a> 。	
输出参数{out}		
-	-	
返回值		
-	-	

Example:

```
/* deselect ADC group channel */
```

```
adc_group_channel_deselect (ADC0, ADC_GROUP_PRI2, ADC_CHANNEL_IN00);
```

### 函数 adc\_group\_end\_flag\_round\_config

函数adc\_group\_end\_flag\_round\_config描述见下表

表 3-66. 函数 adc\_group\_end\_flag\_round\_config

函数名称	adc_group_end_flag_round_config	
函数原型	void adc_group_end_flag_round_config(uint32_t adc_periph, adc_group_select_enum group, uint8_t num)	
功能描述	配置 Group_prix 组转换结束标志轮次	
先决条件	-	
被调用函数	-	
输入参数{in}		
adc_periph	ADC 外设	
ADCx	x=0,2	
输入参数{in}		
group	所选择的组，请参考 <a href="#">枚举类型 adc_group_select_enum</a> 。	
输入参数{in}		
num	Group_pri x 转换结束轮次, 0~7	
输出参数{out}		
-	-	
返回值		

-	-
---	---

Example:

```
/* config end of Group_prix conversion round counts */
```

```
adc_group_end_flag_round_config (ADC0, ADC_GROUP_PRI2, 3);
```

### 函数 `adc_group_external_trigger_enable`

函数 `adc_group_external_trigger_enable` 描述见下表

表 3-67. 函数 `adc_group_external_trigger_enable`

函数名称	<code>adc_group_external_trigger_enable</code>
函数原型	<code>void adc_group_external_trigger_enable(uint32_t adc_periph, adc_group_select_enum group)</code>
功能描述	使能 ADC 外部触发
先决条件	-
被调用函数	-
输入参数{in}	
<code>adc_periph</code>	ADC 外设
<code>ADCx</code>	x=0,2
输入参数{in}	
<code>group</code>	所选择的组, 请参考 <a href="#">枚举类型 <code>adc_group_select_enum</code></a> 。
输出参数{out}	
-	-
返回值	
-	-

Example:

```
/* enable ADC external trigger */
```

```
adc_group_external_trigger_enable (ADC0, ADC_GROUP_PRI2, 3);
```

### 函数 `adc_group_extern_trigger_edge_config`

函数 `adc_group_extern_trigger_edge_config` 描述见下表

表 3-68. 函数 `adc_group_extern_trigger_edge_config`

函数名称	<code>adc_group_extern_trigger_edge_config</code>
函数原型	<code>void adc_group_extern_trigger_edge_config(uint32_t adc_periph, adc_group_select_enum group, uint32_t edge_sel)</code>
功能描述	配置 ADC 外部触发边沿
先决条件	-
被调用函数	-
输入参数{in}	



<b>adc_periph</b>	ADC 外设
<i>ADCx</i>	x=0,2
输入参数{in}	
<b>group</b>	所选择的组，请参考 <a href="#">枚举类型 <i>adc_group_select_enum</i></a> 。
输入参数{in}	
<b>edge_sel</b>	选择触发边沿
<i>ADC_EXTERNAL_TRIG_RISING_EDGE</i>	上升沿触发
<i>ADC_EXTERNAL_TRIG_FALLING_EDGE</i>	下降沿触发
<i>ADC_EXTERNAL_TRIG_BOTH_EDGES</i>	双边沿触发
输出参数{out}	
-	-
返回值	
-	-

Example:

```
/* enable ADC external trigger */
```

```
adc_group_external_trigger_enable(ADC0, ADC_GROUP_PRI2,
ADC_EXTERNAL_TRIG_RISING_EDGE);
```

### 函数 **adc\_group\_external\_trigger\_disable**

函数 `adc_group_external_trigger_disable` 描述见下表

表 3-69. 函数 **adc\_group\_external\_trigger\_disable**

函数名称	<code>adc_group_external_trigger_disable</code>
函数原型	<code>void adc_group_external_trigger_disable(uint32_t adc_periph, adc_group_select_enum group)</code>
功能描述	禁能 ADC 外部触发
先决条件	-
被调用函数	-
输入参数{in}	
<b>adc_periph</b>	ADC 外设
<i>ADCx</i>	x=0,2
输入参数{in}	
<b>group</b>	所选择的组，请参考 <a href="#">枚举类型 <i>adc_group_select_enum</i></a> 。
输出参数{out}	

-	-
返回值	
-	-

Example:

```
/* disable ADC external trigger */
```

```
adc_group_external_trigger_disable (ADC0, ADC_GROUP_PRI2);
```

### 函数 `adc_group_synchronous_trigger_enable`

函数`adc_group_synchronous_trigger_enable`描述见下表

表 3-70. 函数 `adc_group_synchronous_trigger_enable`

函数名称	<code>adc_group_synchronous_trigger_enable</code>
函数原型	<code>void adc_group_synchronous_trigger_enable(uint32_t adc_periph, adc_group_select_enum group)</code>
功能描述	使能 ADC 同步触发
先决条件	-
被调用函数	-
输入参数{in}	
<code>adc_periph</code>	ADC 外设
<code>ADCx</code>	x=0,2
输入参数{in}	
<code>group</code>	所选择的组, 请参考 <a href="#">枚举类型 <code>adc_group_select_enum</code></a> 。
输出参数{out}	
-	-
返回值	
-	-

Example:

```
/* enable ADC synchronous trigger */
```

```
adc_group_synchronous_trigger_enable (ADC0, ADC_GROUP_PRI2);
```

### 函数 `adc_group_synchronous_trigger_source_config`

函数`adc_group_synchronous_trigger_source_config`描述见下表

表 3-71. 函数 `adc_group_synchronous_trigger_source_config`

函数名称	<code>adc_group_synchronous_trigger_source_config</code>
函数原型	<code>void adc_group_synchronous_trigger_source_config(uint32_t adc_periph, adc_group_select_enum group, uint32_t external_trigger_source)</code>
功能描述	配置 ADC 同步触发源
先决条件	-

被调用函数	-
输入参数{in}	
adc_periph	ADC 外设
ADCx	x=0,2
输入参数{in}	
group	所选择的组，请参考 <a href="#">枚举类型 adc_group_select_enum</a> 。
输入参数{in}	
external_trigger_source	触发源
ADC_SYNCTRIG_SOURCE_TIMER0_TRGOF	选择 TIMER0_TRGOF 事件
ADC_SYNCTRIG_SOURCE_TIMER0_TRGUF	选择 TIMER0_TRGUF 事件
ADC_SYNCTRIG_SOURCE_TIMER0_TRGA	选择 TIMER0_TRGA 事件
ADC_SYNCTRIG_SOURCE_TIMER0_TRGB	选择 TIMER0_TRGB 事件
ADC_SYNCTRIG_SOURCE_TIMER0_TRGAB	选择 TIMER0_TRGAB 事件
ADC_SYNCTRIG_SOURCE_TIMER0_TRGAORB	选择 TIMER0_TRGAORB 事件(扩展分叉模式触发源)
ADC_SYNCTRIG_SOURCE_TIMER0_TRGO	选择 TIMER0_TRGO 事件
ADC_SYNCTRIG_SOURCE_TIMER7_TRGOF	选择 TIMER7_TRGOF 事件
ADC_SYNCTRIG_SOURCE_TIMER7_TRGUF	选择 TIMER7_TRGUF 事件
ADC_SYNCTRIG_SOURCE_TIMER7_TRGA	选择 TIMER7_TRGA 事件
ADC_SYNCTRIG_SOURCE_TIMER7_TRGB	选择 TIMER7_TRGB 事件

ADC_SYNCTRIG_ SOURCE_TIMER7_ _TRGAB	选择 TIMER7_TRGAB 事件
ADC_SYNCTRIG_ SOURCE_TIMER7_ _TRGAORB	选择 TIMER7_TRGAORB 事件(扩展分叉模式触发源)
ADC_SYNCTRIG_ SOURCE_TIMER7_ _TRGO	选择 TIMER7_TRGO 事件
ADC_SYNCTRIG_ SOURCE_TIMER1_ _TRGO	选择 TIMER1_TRGO 事件
ADC_SYNCTRIG_ SOURCE_TIMER2_ _TRGO	选择 TIMER2_TRGO 事件
ADC_SYNCTRIG_ SOURCE_GPTIME R0_TRGA	选择 GPTIMER0_TRGA 事件
ADC_SYNCTRIG_ SOURCE_GPTIME R0_TRGB	选择 GPTIMER0_TRGB 事件
ADC_SYNCTRIG_ SOURCE_GPTIME R0_TRGAB	选择 GPTIMER0_TRGAB 事件(扩展分叉模式触发源)
ADC_SYNCTRIG_ SOURCE_GPTIME R1_TRGA	选择 GPTIMER1_TRGA 事件
ADC_SYNCTRIG_ SOURCE_GPTIME R1_TRGB	选择 GPTIMER1_TRGB 事件
ADC_SYNCTRIG_ SOURCE_GPTIME R1_TRGAB	选择 GPTIMER1_TRGAB 事件(扩展分叉模式触发源)
ADC_SYNCTRIG_ SOURCE_EVIC_E VSEL0	选择 EVIC_EVSEL0 事件
ADC_SYNCTRIG_ SOURCE_EVIC_E VSEL1	选择 EVIC_EVSEL1 事件
ADC_SYNCTRIG_ SOURCE_EVIC_E VSEL0_OR_1	选择 EVIC_EVSEL0 或者 EVIC_EVSEL1 事件

输出参数{out}	
-	-
返回值	
-	-

Example:

```
/* configure ADC synchronous trigger source */
```

```
adc_group_synchronous_trigger_source_config (ADC0, ADC_GROUP_PRI2,
ADC_SYNCTRIG_SOURCE_TIMER0_TRGB);
```

### 函数 `adc_group_asynchronous_trigger_enable`

函数 `adc_group_asynchronous_trigger_enable` 描述见下表

表 3-72. 函数 `adc_group_asynchronous_trigger_enable`

函数名称	<code>adc_group_asynchronous_trigger_enable</code>
函数原型	<code>void adc_group_asynchronous_trigger_enable(uint32_t adc_periph, adc_group_select_enum group)</code>
功能描述	使能 ADC 异步触发
先决条件	-
被调用函数	-
输入参数{in}	
<b>adc_periph</b>	ADC 外设
<i>ADCx</i>	x=0,2
输入参数{in}	
<b>group</b>	所选择的组，请参考 <a href="#">枚举类型 <code>adc_group_select_enum</code></a> 。
输出参数{out}	
-	-
返回值	
-	-

Example:

```
/* enable ADC asynchronous trigger */
```

```
adc_group_asynchronous_trigger_enable (ADC0, ADC_GROUP_PRI2);
```

### 函数 `adc_group_software_trigger_enable`

函数 `adc_group_software_trigger_enable` 描述见下表

表 3-73. 函数 `adc_group_software_trigger_enable`

函数名称	<code>adc_group_software_trigger_enable</code>
函数原型	<code>void adc_group_software_trigger_enable(uint32_t adc_periph, adc_group_select_enum group)</code>

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_periph	ADC 外设
ADCx	x=0,2
输入参数{in}	
group	所选择的组，请参考 <a href="#">枚举类型 adc_group_select_enum</a> 。
输出参数{out}	
-	-
返回值	
-	-

Example:

```
/* enable ADC software trigger */
```

```
adc_group_software_trigger_enable (ADC0, ADC_GROUP_PRI2);
```

### 函数 adc\_group\_software\_end\_conversion

函数adc\_group\_software\_end\_conversion描述见下表

表 3-74. 函数 adc\_group\_software\_end\_conversion

函数名称	adc_group_software_end_conversion
函数原型	void adc_group_software_end_conversion(uint32_t adc_periph)
功能描述	软件结束所有组的转换
先决条件	-
被调用函数	-
输入参数{in}	
adc_periph	ADC 外设
ADCx	x=0,2
输出参数{out}	
-	-
返回值	
-	-

Example:

```
/* software end conversion of all groups */
```

```
adc_group_software_end_conversion (ADC0);
```

### 函数 adc\_group\_bifurcate\_mode\_enable

函数adc\_group\_bifurcate\_mode\_enable描述见下表

表 3-75. 函数 `adc_group_bifurcate_mode_enable`

函数名称	<code>adc_group_bifurcate_mode_enable</code>
函数原型	<code>void adc_group_bifurcate_mode_enable(uint32_t adc_periph, adc_group_select_enum group)</code>
功能描述	使能 Group_pri x 组的分叉触发模式
先决条件	-
被调用函数	-
输入参数{in}	
<code>adc_periph</code>	ADC 外设
<code>ADCx</code>	x=0,2
输入参数{in}	
<code>group</code>	所选择的组, 请参考 <a href="#">枚举类型 <code>adc_group_select_enum</code></a> 。
输出参数{out}	
-	-
返回值	
-	-

Example:

```
/* enable ADC bifurcate trigger mode for Group_pri x */
adc_group_bifurcate_mode_enable (ADC0, ADC_GROUP_PRI1);
```

### 函数 `adc_group_bifurcate_mode_disable`

函数 `adc_group_bifurcate_mode_disable` 描述见下表

表 3-76. 函数 `adc_group_bifurcate_mode_disable`

函数名称	<code>adc_group_bifurcate_mode_disable</code>
函数原型	<code>void adc_group_bifurcate_mode_disable(uint32_t adc_periph, adc_group_select_enum group)</code>
功能描述	禁能 Group_pri x 组的分叉触发模式
先决条件	-
被调用函数	-
输入参数{in}	
<code>adc_periph</code>	ADC 外设
<code>ADCx</code>	x=0,2
输入参数{in}	
<code>group</code>	所选择的组, 请参考 <a href="#">枚举类型 <code>adc_group_select_enum</code></a> 。
输出参数{out}	
-	-
返回值	
-	-

Example:

```
/* disable ADC bifurcate trigger mode for Group_pri x */
```

```
adc_group_bifurcate_mode_disable (ADC0, ADC_GROUP_PRI1);
```

### 函数 `adc_group_bifurcate_channel_select`

函数 `adc_group_bifurcate_channel_select` 描述见下表

表 3-77. 函数 `adc_group_bifurcate_channel_select`

函数名称	<code>adc_group_bifurcate_channel_select</code>
函数原型	<code>void adc_group_bifurcate_channel_select(uint32_t adc_periph, adc_group_select_enum group, adc_channel_select_enum channel)</code>
功能描述	ADC 分叉触发通道选择
先决条件	-
被调用函数	-
输入参数{in}	
<code>adc_periph</code>	ADC 外设
<code>ADCx</code>	x=0,2
输入参数{in}	
<code>group</code>	所选择的组，请参考 <a href="#">枚举类型 <code>adc_group_select_enum</code></a> 。
输入参数{in}	
<code>channel</code>	所选择的通道，请参考 <a href="#">枚举类型 <code>adc_channel_select_enum</code></a> 。 注意： <code>ADC_CHANNEL_ALL</code> 除外
输出参数{out}	
-	-
返回值	
-	-

Example:

```
/* ADC bifurcate trigger channel select */
```

```
adc_group_bifurcate_channel_select (ADC0, ADC_GROUP_PRI1, ADC_BICHSEL_CHANNEL_IN00);
```

### 函数 `adc_group_bifurcate_extend_trigger_select`

函数 `adc_group_bifurcate_extend_trigger_select` 描述见下表

表 3-78. 函数 `adc_group_bifurcate_extend_trigger_select`

函数名称	<code>adc_group_bifurcate_extend_trigger_select</code>
函数原型	<code>void adc_group_bifurcate_extend_trigger_select(uint32_t adc_periph, adc_group_select_enum group, uint32_t trigger_source)</code>
功能描述	扩展分叉模式的触发源选择
先决条件	-
被调用函数	-



输入参数{in}	
adc_periph	ADC 外设
ADCx	x=0,2
输入参数{in}	
group	所选择的组，请参考 <a href="#">枚举类型 adc_group_select_enum</a> 。
输入参数{in}	
trigger_source	扩展分叉模式触发源选择
ADC_SYNCTRIG_SOURCE_TIMER0_TRGAORB	选择 TIMER0_TRGAORB 作为触发源，适用于 ADCx(x=0,2)
ADC_SYNCTRIG_SOURCE_TIMER7_TRGAORB	选择 TIMER7_TRGAORB 作为触发源，适用于 ADCx(x=0,2)
ADC_SYNCTRIG_SOURCE_GPTIMER0_TRGAB	选择 GPTIMER0_TRGAB 作为触发源，适用于 ADCx(x=0,2)
ADC_SYNCTRIG_SOURCE_GPTIMER1_TRGAB	选择 GPTIMER1_TRGAB 作为触发源，适用于 ADCx(x=0,2)
ADC_SYNCTRIG_SOURCE_EVIC_EVSEL0_OR_1	选择 EVIC_EVSEL0 或者 EVIC_EVSEL1 作为触发源，适用于 ADCx(x=0,2)
输出参数{out}	
-	-
返回值	
-	-

Example:

```
/* extend bifurcate trigger source select */
```

```
adc_group_bifurcate_extend_trigger_select(ADC0, ADC_GROUP_PRI1,
ADC_SYNCTRIG_SOURCE_TIMER0_TRGAORB);
```

### 函数 adc\_group\_bifurcate\_trigger\_restart\_enable

函数adc\_group\_bifurcate\_trigger\_restart\_enable描述见下表

表 3-79. 函数 adc\_group\_bifurcate\_trigger\_restart\_enable

函数名称	adc_group_bifurcate_trigger_restart_enable
函数原型	void adc_group_bifurcate_trigger_restart_enable(uint32_t adc_periph, adc_group_select_enum group)
功能描述	使能触发存储功能（当前转换周期内的触发会被存储起来，当前转换结束后响应存储起来的触发）
先决条件	-

被调用函数	-
输入参数{in}	
adc_periph	ADC 外设
ADCx	x=0,2
输入参数{in}	
group	所选择的组，请参考 <a href="#">枚举类型 adc_group_select_enum</a> 。
输出参数{out}	
-	-
返回值	
-	-

Example:

```
/* enable restore of the next trigger during the current A/D conversion round */
```

```
adc_group_bifurcate_trigger_restart_enable (ADC0, ADC_GROUP_PRI1);
```

### 函数 adc\_group\_bifurcate\_trigger\_restart\_disable

函数adc\_group\_bifurcate\_trigger\_restart\_disable描述见下表

表 3-80. 函数 adc\_group\_bifurcate\_trigger\_restart\_disable

函数名称	adc_group_bifurcate_trigger_restart_disable
函数原型	void adc_group_bifurcate_trigger_restart_disable(uint32_t adc_periph, adc_group_select_enum group)
功能描述	禁能触发存储功能（当前转换周期内的触发会被丢弃）
先决条件	-
被调用函数	-
输入参数{in}	
adc_periph	ADC 外设
ADCx	x=0,2
输入参数{in}	
group	所选择的组，请参考 <a href="#">枚举类型 adc_group_select_enum</a> 。
输出参数{out}	
-	-
返回值	
-	-

Example:

```
/* disable restore of the next trigger during the current A/D conversion round */
```

```
adc_group_bifurcate_trigger_restart_disable (ADC0, ADC_GROUP_PRI1);
```

## 函数 adc\_group\_dma\_mode\_enable

函数adc\_group\_dma\_mode\_enable描述见下表

表 3-81. 函数 adc\_group\_dma\_mode\_enable

函数名称	adc_group_dma_mode_enable
函数原型	void adc_group_dma_mode_enable(uint32_t adc_periph, adc_group_select_enum group)
功能描述	使能 DMA 请求
先决条件	-
被调用函数	-
输入参数{in}	
adc_periph	ADC 外设
ADCx	x=0,2
输入参数{in}	
group	所选择的组，请参考 <a href="#">枚举类型 adc_group_select_enum</a> 。
输出参数{out}	
-	-
返回值	
-	-

Example:

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

```
adc_group_dma_mode_enable (ADC0, ADC_GROUP_PRI1);
```

## 函数 adc\_group\_dma\_mode\_disable

函数adc\_group\_dma\_mode\_disable描述见下表

表 3-82. 函数 adc\_group\_dma\_mode\_disable

函数名称	adc_group_dma_mode_disable
函数原型	void adc_group_dma_mode_disable(uint32_t adc_periph, adc_group_select_enum group)
功能描述	禁能 DMA 请求
先决条件	-
被调用函数	-
输入参数{in}	
adc_periph	ADC 外设
ADCx	x=0,2
输入参数{in}	
group	所选择的组，请参考 <a href="#">枚举类型 adc_group_select_enum</a> 。
输出参数{out}	
-	-

返回值	
-	-

Example:

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

```
adc_group_dma_mode_disable (ADC0, ADC_GROUP_PRI1);
```

### 函数 `adc_group_dma_request_after_last_enable`

函数 `adc_group_dma_request_after_last_enable` 描述见下表

表 3-83. 函数 `adc_group_dma_request_after_last_enable`

函数名称	<code>adc_group_dma_request_after_last_enable</code>
函数原型	<code>void adc_group_dma_request_after_last_enable(uint32_t adc_periph, adc_group_select_enum group)</code>
功能描述	当 DMAENx 位为 1 时，在每次 Group_pri x 组转换结束后产生 DMA 请求。
先决条件	-
被调用函数	-
输入参数{in}	
<code>adc_periph</code>	ADC 外设
<code>ADCx</code>	x=0,2
输入参数{in}	
<code>group</code>	所选择的组，请参考 <a href="#">枚举类型 <code>adc_group_select_enum</code></a> 。
输出参数{out}	
-	-
返回值	
-	-

Example:

```
/* when DMAENx=1, the DMA engine issues request at end of conversion of Group_prixt */
```

```
adc_group_dma_request_after_last_enable (ADC0, ADC_GROUP_PRI1);
```

### 函数 `adc_group_dma_request_after_last_disable`

函数 `adc_group_dma_request_after_last_disable` 描述见下表

表 3-84. 函数 `adc_group_dma_request_after_last_disable`

函数名称	<code>adc_group_dma_request_after_last_disable</code>
函数原型	<code>void adc_group_dma_request_after_last_disable(uint32_t adc_periph, adc_group_select_enum group)</code>
功能描述	在检测到 DMA 控制器的传输结束信号之后，禁能 DMA 传输
先决条件	-
被调用函数	-

输入参数{in}	
<b>adc_periph</b>	ADC 外设
<i>ADCx</i>	x=0,2
输入参数{in}	
<b>group</b>	所选择的组，请参考 <a href="#">枚举类型 <i>adc_group_select_enum</i></a> 。
输出参数{out}	
-	-
返回值	
-	-

Example:

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

```
adc_group_dma_request_after_last_disable (ADC0, ADC_GROUP_PRI1);
```

### 函数 `adc_group_dma_overshoot_detect_enable`

函数 `adc_group_dma_overshoot_detect_enable` 描述见下表

表 3-85. 函数 `adc_group_dma_overshoot_detect_enable`

函数名称	<code>adc_group_dma_overshoot_detect_enable</code>
函数原型	<code>void adc_group_dma_overshoot_detect_enable(uint32_t adc_periph, adc_group_select_enum group)</code>
功能描述	使能 DMA 溢出检测
先决条件	-
被调用函数	-
输入参数{in}	
<b>adc_periph</b>	ADC 外设
<i>ADCx</i>	x=0,2
输入参数{in}	
<b>group</b>	所选择的组，请参考 <a href="#">枚举类型 <i>adc_group_select_enum</i></a> 。
输出参数{out}	
-	-
返回值	
-	-

Example:

```
/* enable dma overshoot detect */
```

```
adc_group_dma_overshoot_detect_enable (ADC0, ADC_GROUP_PRI1);
```

## 函数 `adc_group_dma_overshoot_detect_disable`

函数`adc_group_dma_overshoot_detect_disable`描述见下表

**表 3-86. 函数 `adc_group_dma_overshoot_detect_disable`**

函数名称	<code>adc_group_dma_overshoot_detect_disable</code>
函数原型	<code>void adc_group_dma_overshoot_detect_disable(uint32_t adc_periph, adc_group_select_enum group)</code>
功能描述	禁能 DMA 溢出检测
先决条件	-
被调用函数	-
输入参数{in}	
<code>adc_periph</code>	ADC 外设
<code>ADCx</code>	x=0,2
输入参数{in}	
<code>group</code>	所选择的组，请参考 <a href="#">枚举类型 <code>adc_group_select_enum</code></a> 。
输出参数{out}	
-	-
返回值	
-	-

Example:

```
/* disable dma overshoot detect */
```

```
adc_group_dma_overshoot_detect_disable (ADC0, ADC_GROUP_PRI1);
```

## 函数 `adc_channel_priority_config`

函数`adc_channel_priority_config`描述见下表

**表 3-87. 函数 `adc_channel_priority_config`**

函数名称	<code>adc_channel_priority_config</code>
函数原型	<code>void adc_channel_priority_config(uint32_t adc_periph, adc_channel_select_enum adc_channel, uint8_t rank)</code>
功能描述	通道优先级配置
先决条件	-
被调用函数	-
输入参数{in}	
<code>adc_periph</code>	ADC 外设
<code>ADCx</code>	x=0,2
输入参数{in}	
<code>channel</code>	所选择的通道，请参考 <a href="#">枚举类型 <code>adc_channel_select_enum</code></a> 。 注意： <code>ADC_CHANNEL_ALL</code> 除外
输入参数{in}	

rank	the priority of the selected channel
输出参数{out}	
-	-
返回值	
-	-

Example:

```
/* channel priority config */
```

```
adc_channel_priority_config (ADC0, ADC_CHANNEL_IN01, 2);
```

```
adc_channel_priority_config (ADC0, ADC_CHANNEL_IN02, 1);
```

### 函数 adc\_channel\_sample\_time\_config

函数adc\_channel\_sample\_time\_config描述见下表

表 3-88. 函数 adc\_channel\_sample\_time\_config

函数名称	adc_channel_sample_time_config
函数原型	void adc_channel_sample_time_config(uint32_t adc_periph, adc_channel_select_enum channel, uint32_t sample_time)
功能描述	配置通道采样时间
先决条件	-
被调用函数	-
输入参数{in}	
adc_periph	ADC 外设
ADCx	x=0,2
输入参数{in}	
channel	所选择的通道，请参考 <a href="#">枚举类型 adc_channel_select_enum</a> 。
输入参数{in}	
sample_time	0x02~0xFF。最小采样时间为 0x02。
输出参数{out}	
-	-
返回值	
-	-

Example:

```
/* configure ADC channel sample time */
```

```
adc_channel_sample_time_config (ADC0, ADC_CHANNEL_IN01, 0x02);
```

```
adc_channel_sample_time_config (ADC0, ADC_CHANNEL_IN02, 0x02);
```

### 函数 adc\_evic\_link\_signal\_event\_config

函数adc\_evic\_link\_signal\_event\_config描述见下表

表 3-89. 函数 `adc_evic_link_signal_event_config`

函数名称	<code>adc_evic_link_signal_event_config</code>
函数原型	<code>void adc_evic_link_signal_event_config(uint32_t adc_periph, uint32_t linksignal)</code>
功能描述	选择 EVIC 连接信号
先决条件	-
被调用函数	-
输入参数{in}	
<code>adc_periph</code>	ADC 外设
<code>ADCx</code>	<code>x=0,2</code>
输入参数{in}	
<code>linksignal</code>	EVIC 连接信号
<code>ADC_EVIC_LINK_GROUP_PRI1</code>	在扫描 Group_pri1 完成时生成事件信号
<code>ADC_EVIC_LINK_GROUP_PRI2</code>	在扫描 Group_pri2 完成时生成事件信号
<code>ADC_EVIC_LINK_GROUP_PRI3</code>	在扫描 Group_pri3 完成时生成事件信号
<code>ADC_EVIC_LINK_GROUP_PRI4</code>	在扫描 Group_pri4 完成时生成事件信号
<code>ADC_EVIC_LINK_GROUP_ALL</code>	在扫描 Group_pri1、Group_pri2、Group_pri3 或 Group_pri4 完成时生成事件信号。
输出参数{out}	
-	-
返回值	
-	-

Example:

```
/* select which event as the EVIC link signal */
```

```
adc_evic_link_signal_event_config (ADC0, ADC_EVIC_LINK_GROUP_PRI2);
```

### 函数 `adc_group_evic_link_signal_source`

函数 `adc_group_evic_link_signal_source` 描述见下表

表 3-90. 函数 `adc_group_evic_link_signal_source`

函数名称	<code>adc_group_evic_link_signal_source</code>
函数原型	<code>void adc_group_evic_link_signal_source(uint32_t adc_periph, adc_group_select_enum group, adc_evic_link_source_enum signal_src)</code>
功能描述	选择 Group_pri x 的 EVIC 连接信号源
先决条件	-
被调用函数	-



输入参数{in}	
<b>adc_periph</b>	ADC 外设
<i>ADCx</i>	x=0,2
输入参数{in}	
<b>group</b>	所选择的组, 请参考 <a href="#">枚举类型 <i>adc_group_select_enum</i></a> 。
输入参数{in}	
<b>signal_src</b>	EVIC 触发信号源, 请参考 <a href="#">枚举类型 <i>adc_evic_link_source_enum</i></a> 。
输出参数{out}	
-	-
返回值	
-	-

Example:

```
/* select which event as the EVIC link signal */
```

```
adc_evic_link_signal_event_config      (ADC0,          ADC_GROUP_PRI1,
ADC_EVIC_LINK_SOURCE_EOCR_FFLAG);
```

### 函数 **adc\_channel\_data\_read**

函数 `adc_channel_data_read` 描述见下表

表 3-91. 函数 **adc\_channel\_data\_read**

函数名称	<code>adc_channel_data_read</code>
函数原型	<code>uint16_t adc_channel_data_read(uint32_t adc_periph, adc_channel_select_enum channel)</code>
功能描述	读 ADC 通道数据寄存器
先决条件	-
被调用函数	-
输入参数{in}	
<b>adc_periph</b>	ADC 外设
<i>ADCx</i>	x=0,2
输入参数{in}	
<b>channel</b>	所选择的通道, 请参考 <a href="#">枚举类型 <i>adc_channel_select_enum</i></a> 。 注意: <i>ADC_CHANNEL_ALL</i> 除外
输出参数{out}	
-	-
返回值	
转换值	0~0xFFFF

Example:

```
/* read ADC channel data register */
```

```
uint16_t channel1_data;
```

```
channel1_data = adc_channel_data_read (ADC0, ADC_CHANNEL_IN01);
```

### 函数 `adc_self_diagnosis_data_read`

函数`adc_self_diagnosis_data_read`描述见下表

**表 3-92. 函数 `adc_self_diagnosis_data_read`**

函数名称	<code>adc_self_diagnosis_data_read</code>
函数原型	<code>uint16_t adc_self_diagnosis_data_read(uint32_t adc_periph)</code>
功能描述	read ADC self-diagnosis data register
先决条件	-
被调用函数	-
输入参数{in}	
<code>adc_periph</code>	ADC 外设
<code>ADCx</code>	x=0,2
输出参数{out}	
-	-
返回值	
转换值	0x0000~0x0FFF (DAL=0, LSB 对齐), 0x0000~0xFFFF (DAL=1, MSB 对齐),

Example:

```
/* read ADC self-diagnosis data */
```

```
uint16_t data;
```

```
data = adc_self_diagnosis_data_read (ADC0);
```

### 函数 `adc_self_diagnosis_status_read`

函数`adc_self_diagnosis_status_read`描述见下表

**表 3-93. 函数 `adc_self_diagnosis_status_read`**

函数名称	<code>adc_self_diagnosis_status_read</code>
函数原型	<code>uint16_t adc_self_diagnosis_status_read(uint32_t adc_periph)</code>
功能描述	读 ADC 自诊断状态
先决条件	-
被调用函数	-
输入参数{in}	
<code>adc_periph</code>	ADC 外设
<code>ADCx</code>	x=0,2
输出参数{out}	
-	-
返回值	
自诊断转换电压状	0x0: 从上电开始自诊断未执行过

态	0x1: 自诊断转换0V 0x2: 自诊断转换 $V_{REFP}/2$ 0x3: 自诊断转换 $V_{REFP}$
---	--

Example:

```
/* read ADC self-diagnosis status */

uint16_t status;

status= adc_self_diagnosis_status_read (ADC0);
```

### 函数 adc\_bifurcate\_data\_read

函数adc\_bifurcate\_data\_read描述见下表

表 3-94. 函数 adc\_bifurcate\_data\_read

函数名称	adc_bifurcate_data_read
函数原型	uint16_t adc_bifurcate_data_read(uint32_t adc_periph, adc_bifurcate_data_enum channel)
功能描述	读 ADC 分叉数据寄存器
先决条件	-
被调用函数	-
输入参数{in}	
adc_periph	ADC 外设
ADCx	x=0,2
输入参数{in}	
channel	数据通道选择, 请参考 <a href="#">枚举类型 adc_bifurcate_data_enum</a> 。
输出参数{out}	
-	-
返回值	
转换值	0~0xFFFF

Example:

```
/* read ADC bifurcate data register */

uint16_t data;

data = adc_bifurcate_data_read (ADC0, ADC_BIFURCATE_DATA);
```

### 函数 adc\_flag\_get

函数adc\_flag\_get描述见下表

表 3-95. 函数 adc\_flag\_get

函数名称	adc_flag_get
函数原型	FlagStatus adc_flag_get(uint32_t adc_periph, adc_event_flag_enum flag)

功能描述	获取 ADC 标志	
先决条件	-	
被调用函数	-	
输入参数{in}		
adc_periph	ADC 外设	
ADCx	x=0,2	
输入参数{in}		
flag	ADC 标志，请参考 <a href="#">枚举类型adc interrupt flag enum</a>	
	表 3-15. 枚举类型 adc_interrupt_flag_enum	
	成员名称	
	ADC_INT_FLAG_EOC1RF	
	ADC_INT_FLAG_EOC2RF	
	ADC_INT_FLAG_EOC3RF	
	ADC_INT_FLAG_EOC4RF	
	ADC_INT_FLAG_WDA_CHSTAT	
	ADC_INT_FLAG_WDBEF	
	ADC_INT_FLAG_GP1OVRF	Group_
	ADC_INT_FLAG_GP2OVRF	Group_
	ADC_INT_FLAG_GP3OVRF	Group_
ADC_INT_FLAG_GP4OVRF	Group_	
枚举类型 adc_event_flag_enum。		
输出参数{out}		
-	-	
返回值		
FlagStatus	SET or RESET	

Example:

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

```
uint16_t flag;
```

```
flag = adc_flag_get (ADC0, ADC_FLAG_EOC1RF);
```

### 函数 adc\_flag\_clear

函数adc\_flag\_clear描述见下表

表 3-96. 函数 adc\_flag\_clear

函数名称	adc_flag_clear
函数原型	void adc_flag_clear(uint32_t adc_periph, adc_event_flag_enum flag)
功能描述	清除 ADC 标志
先决条件	-
被调用函数	-

输入参数{in}																								
adc_periph	ADC 外设																							
ADCx	x=0,2																							
输入参数{in}																								
flag	ADC 标志, 请参考 <a href="#">枚举类型 adc_interrupt_flag_enum</a>																							
	表 3-15. 枚举类型 adc_interrupt_flag_enum																							
	<table><tr><th>成员名称</th><th></th></tr><tr><td>ADC_INT_FLAG_EOC1RF</td><td></td></tr><tr><td>ADC_INT_FLAG_EOC2RF</td><td></td></tr><tr><td>ADC_INT_FLAG_EOC3RF</td><td></td></tr><tr><td>ADC_INT_FLAG_EOC4RF</td><td></td></tr><tr><td>ADC_INT_FLAG_WDA_CHSTAT</td><td></td></tr><tr><td>ADC_INT_FLAG_WDBEF</td><td></td></tr><tr><td>ADC_INT_FLAG_GP1OVRF</td><td>Group_</td></tr><tr><td>ADC_INT_FLAG_GP2OVRF</td><td>Group_</td></tr><tr><td>ADC_INT_FLAG_GP3OVRF</td><td>Group_</td></tr><tr><td>ADC_INT_FLAG_GP4OVRF</td><td>Group_</td></tr></table>		成员名称		ADC_INT_FLAG_EOC1RF		ADC_INT_FLAG_EOC2RF		ADC_INT_FLAG_EOC3RF		ADC_INT_FLAG_EOC4RF		ADC_INT_FLAG_WDA_CHSTAT		ADC_INT_FLAG_WDBEF		ADC_INT_FLAG_GP1OVRF	Group_	ADC_INT_FLAG_GP2OVRF	Group_	ADC_INT_FLAG_GP3OVRF	Group_	ADC_INT_FLAG_GP4OVRF	Group_
	成员名称																							
	ADC_INT_FLAG_EOC1RF																							
	ADC_INT_FLAG_EOC2RF																							
	ADC_INT_FLAG_EOC3RF																							
	ADC_INT_FLAG_EOC4RF																							
	ADC_INT_FLAG_WDA_CHSTAT																							
	ADC_INT_FLAG_WDBEF																							
	ADC_INT_FLAG_GP1OVRF	Group_																						
	ADC_INT_FLAG_GP2OVRF	Group_																						
ADC_INT_FLAG_GP3OVRF	Group_																							
ADC_INT_FLAG_GP4OVRF	Group_																							
枚举类型 adc_event_flag_enum。																								
输出参数{out}																								
-	-																							
返回值																								
-	-																							

Example:

```
/* clear the ADC flag */
```

```
adc_flag_clear (ADC0, ADC_FLAG_EOC1RF);
```

### 函数 adc\_interrupt\_enable

函数adc\_interrupt\_enable描述见下表

表 3-97. 函数 adc\_interrupt\_enable

函数名称	adc_interrupt_enable
函数原型	void adc_interrupt_enable(uint32_t adc_periph, adc_interrupt_enum interrupt)
功能描述	使能 ADC 中断
先决条件	-
被调用函数	-
输入参数{in}	
adc_periph	ADC 外设
ADCx	x=0,2
输入参数{in}	
flag	中断选择, 请参考 <a href="#">枚举类型 adc_interrupt_enum</a> 。

输出参数{out}	
-	-
返回值	
-	-

Example:

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

```
adc_interrupt_enable (ADC0, ADC_INT_EOC1RF);
```

### 函数 `adc_interrupt_disable`

函数`adc_interrupt_disable`描述见下表

表 3-98. 函数 `adc_interrupt_disable`

函数名称	<code>adc_interrupt_disable</code>
函数原型	<code>void adc_interrupt_disable(uint32_t adc_periph, adc_interrupt_enum interrupt)</code>
功能描述	禁能 ADC 中断
先决条件	-
被调用函数	-
输入参数{in}	
<b>adc_periph</b>	ADC 外设
<b>ADCx</b>	x=0,2
输入参数{in}	
<b>flag</b>	中断选择, 请参考 <a href="#">枚举类型 <code>adc_interrupt_enum</code></a> 。
输出参数{out}	
-	-
返回值	
-	-

Example:

```
/* disable ADC interrupt */
```

```
adc_interrupt_disable (ADC0, ADC_INT_EOC1RF);
```

### 函数 `adc_interrupt_flag_get`

函数`adc_interrupt_flag_get`描述见下表

表 3-99. 函数 `adc_interrupt_flag_get`

函数名称	<code>adc_interrupt_flag_get</code>
函数原型	<code>FlagStatus adc_interrupt_flag_get(uint32_t adc_periph, adc_interrupt_flag_enum int_flag)</code>
功能描述	获取 ADC 中断标志
先决条件	-

被调用函数	-
输入参数{in}	
adc_periph	ADC 外设
ADCx	x=0,2
输入参数{in}	
int_flag	ADC 中断标志，请参考 <a href="#">枚举类型 adc_interrupt_flag_enum</a> 。
输出参数{out}	
-	-
返回值	
FlagStatus: SET or RESET	

Example:

```
/* get ADC interrupt flag */
```

```
uint16_t flag;
```

```
flag = adc_interrupt_flag_get (ADC0, ADC_INT_FLAG_EOC1RF);
```

### 函数 adc\_interrupt\_flag\_clear

函数adc\_interrupt\_flag\_clear描述见下表

表 3-100. 函数 adc\_interrupt\_flag\_clear

函数名称	adc_interrupt_flag_clear
函数原型	void adc_interrupt_flag_clear(uint32_t adc_periph, adc_interrupt_flag_enum int_flag)
功能描述	清楚 ADC 中断标志
先决条件	-
被调用函数	-
输入参数{in}	
adc_periph	ADC 外设
ADCx	x=0,2
输入参数{in}	
int_flag	ADC 中断标志，请参考 <a href="#">枚举类型 adc_interrupt_flag_enum</a> 。
输出参数{out}	
-	-
返回值	
-	-

Example:

```
/* clear ADC interrupt flag */
```

```
adc_interrupt_flag_clear (ADC0, ADC_INT_FLAG_EOC1RF);
```

### 3.3. CAN

CAN（Controller Area Network）总线是一种可以在无主机情况下实现微处理器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 3-101. 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 3-102. 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
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 3-103. 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
bit_sample_mode	位采样模式
sample_bit_select	位采样同步选择
frame_inter_to_id	帧间隔到帧ID功能
time_triggered	时间触发通信模式
auto_bus_off_recovery	自动离线恢复

成员名称	功能描述
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\_trasnmit\_message\_struct

表 3-104. 结构体 can\_trasnmit\_message\_struct

成员名称	功能描述
tx_sfId	标准格式帧标识符
tx_efId	扩展格式帧标识符
tx_ff	帧格式：标准格式/扩展格式
tx_ft	帧类型：数据帧/远程帧
tx_dlen	数据长度
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 3-105. 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 3-106. 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

成员名称	功能描述
filter_enable	过滤器是否工作

### 函数 can\_deinit

函数can\_deinit描述见下表：

表 3-107. 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 3-108. 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	接收帧结构体

<i>E_STRUCT</i>	
输出参数{out}	
<b>p_struct</b>	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 3-109. 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}	
<b>can_parameter_init</b>	初始化结构体, 结构体成员参考 <a href="#">表 3-103. 结构体can_parameter_struct</a>
输出参数{out}	
-	-
返回值	
<b>ErrStatus</b>	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 3-110. 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 <a href="#">表 3-106. 结构体 can_filter_parameter_struct</a>
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 3-111. 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 3-112. 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 3-113. 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 3-114. 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 3-115. 函数 can\_message\_transmit

函数名称	can_message_transmit
函数原型	uint8_t can_message_transmit(can_transmit_message_struct* transmit_message);
功能描述	CAN传输报文
先决条件	can_struct_para_init()
被调用函数	-
输入参数{in}	
transmit_message	报文发送结构体，结构体成员参考 <a href="#">表 3-104. 结构体 can_transmit_message_struct</a>
输出参数{out}	
-	-
返回值	
uint8_t	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 3-116. 函数 can\_transmit\_states

函数名称	can_transmit_states
函数原型	can_transmit_state_enum can_transmit_states(uint8_t mailbox_number);

功能描述	获取CAN传输状态
先决条件	-
被调用函数	-
输入参数{in}	
mailbox_number	邮箱标号
CAN_MAILBOXx	CAN_MAILBOXx(x=0,1,2)
输出参数{out}	
-	-
返回值	
can_transmit_state_enum	0..4

例如:

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

```
can_transmit_state_enum transmit_state = CAN_TRANSMIT_FAILED;
```

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

### 函数 can\_transmission\_stop

函数can\_transmission\_stop描述见下表:

表 3-117. 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 3-118. 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 <a href="#">表 3-105. 结构体 can_receive_message_struct</a>
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 3-119. 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 3-120. 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
CAN_FIFOx	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 3-121. 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

<i>P</i>	
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 3-122. 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 3-123. 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 3-124. 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 3-125. 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 3-126. 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 */

FlagStatus can_flag = RESET;

can_flag = can_flag_get (CAN_FLAG_MTF0);
```

### 函数 can\_flag\_clear

函数can\_flag\_clear描述见下表:

表 3-127. 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 3-128. 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 3-129. 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 3-130. 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_SLPW	进入睡眠工作模式的状态改变中断标志
CAN_INT_FLAG_WU	从睡眠工作模式唤醒的状态改变中断标志
CAN_INT_FLAG_ERR	错误中断标志
CAN_INT_FLAG_MAILBOX2	邮箱2发送完成中断标志



<code>CAN_INT_FLAG_MTF1</code>	邮箱1发送完成中断标志
<code>CAN_INT_FLAG_MTF0</code>	邮箱0发送完成中断标志
<code>CAN_INT_FLAG_RF00</code>	接收FIFO0溢出中断标志
<code>CAN_INT_FLAG_RFF0</code>	接收FIFO0满中断标志
<code>CAN_INT_FLAG_RF01</code>	接收FIFO1溢出中断标志
<code>CAN_INT_FLAG_RFF1</code>	接收FIFO1满中断标志
输出参数{out}	
-	-
返回值	
<b>FlagStatus</b>	SET / RESET

例如：

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

```
FlagStatus can_int_flag = RESET;
```

```
can_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 3-131. 函数 `can_interrupt_flag_clear`

函数名称	<code>can_interrupt_flag_clear</code>
函数原型	<code>void can_interrupt_flag_clear(can_interrupt_flag_enum flag);</code>
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}	
<b>flag</b>	CAN中断标志位
<code>CAN_INT_FLAG_SLPIF</code>	进入睡眠工作模式的状态改变中断标志
<code>CAN_INT_FLAG_WUIF</code>	从睡眠工作模式唤醒的状态改变中断标志
<code>CAN_INT_FLAG_ERRIF</code>	错误中断标志
<code>CAN_INT_FLAG_MTF2</code>	邮箱2发送完成中断标志

CAN_INT_FLAG_M TF1	邮箱1发送完成中断标志
CAN_INT_FLAG_M TF0	邮箱0发送完成中断标志
CAN_INT_FLAG_R FO0	接收FIFO0溢出中断标志
CAN_INT_FLAG_R FF0	接收FIFO0满中断标志
CAN_INT_FLAG_R 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. CFMU

时钟频率测量单元提供时钟频率精度测量功能。章节[3.4.1](#)描述了CFMU的寄存器列表，章节[3.4.2](#)对CFMU库函数进行说明。

#### 3.4.1. 外设寄存器说明

表 3-132. CFMU 寄存器

寄存器名称	寄存器描述
CFMU_CTL	控制寄存器
CFMU_STAT	状态寄存器
CFMU_LVCFG	电流限制值配置寄存器
CFMU_CNT	计数器值寄存器

#### 3.4.2. 外设库函数说明

CFMU库函数列表如下表所示：

表 3-133. CFMU 库函数

库函数名称	库函数描述
cfmu_deinit	重新初始化 CFMU

cfmu_enable	使能时钟频率测量
cfmu_disable	禁能时钟频率测量
cfmu_cfmuref_enable	使能 CFMUREF 引脚输入
cfmu_cfmuref_disable	禁能 CFMUREF 引脚输入
cfmu_reference_signal_config	配置参考信号
cfmu_digital_filter_config	配置数字滤波器
cfmu_reference_clock_config	配置测量参考时钟源
cfmu_measurement_clock_config	配置测量目标时钟源
cfmu_limit_value_config	配置 CFMU 的上限值和下限值
cfmu_interrupt_enable	使能 CFMU 中断
cfmu_interrupt_disable	禁能 CFMU 中断
cfmu_interrupt_flag_get	获取 CFMU 中断标志
cfmu_interrupt_flag_clear	清除 CFMU 中断标志

### 枚举类型 cfmu\_int\_enum

表 3-134. 枚举类型 cfmu\_int\_enum

成员名称	功能描述
CFMU_INT_CFERR	时钟频率错误中断请求
CFMU_INT_CFMEND	时钟频率精度测量结束中断请求
CFMU_INT_OVF	溢出中断请求

### 枚举类型 cfmu\_int\_flag\_enum

表 3-135. 枚举类型 cfmu\_int\_flag\_enum

成员名称	功能描述
CFMU_INT_FLAG_OVF	溢出标志
CFMU_INT_FLAG_CFMEND	时钟频率精度测量结束标志
CFMU_INT_FLAG_CFERR	时钟频率错误标志

### 枚举类型 cfmu\_int\_flag\_clear\_enum

表 3-136. 枚举类型 cfmu\_int\_flag\_clear\_enum

成员名称	功能描述
CFMU_INT_FLAG_CFE	时钟频率错误标志清除

RR_CLR	
CFMU_INT_FLAG_CF MEND_CLR	时钟频率精度测量结束标志清除
CFMU_INT_FLAG_OVF_CLR	溢出标志清除

### 函数 cfmu\_deinit

函数cfmu\_deinit描述见下表：

**表 3-137. 函数 cfmu\_deinit**

函数名称	cfmu_deinit
函数原型	void cfmu_deinit(void)
功能描述	重新初始化 CFMU
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 CFMU */
cfmu_deinit();
```

### 函数 cfmu\_enable

函数cfmu\_enable描述见下表：

**表 3-138. 函数 cfmu\_enable**

函数名称	cfmu_enable
函数原型	void cfmu_enable(void)
功能描述	使能时钟频率测量
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 clock frequency measurement */

cfmu_enable();
```

### 函数 cfmu\_disable

函数cfmu\_disable描述见下表：

表 3-139. 函数 cfmu\_disable

函数名称	cfmu_disable
函数原型	void cfmu_disable(void)
功能描述	禁能时钟频率测量
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 clock frequency measurement */

cfmu_disable();
```

### 函数 cfmu\_cfmuref\_enable

函数cfmu\_cfmuref\_enable描述见下表：

表 3-140. 函数 cfmu\_cfmuref\_enable

函数名称	cfmu_cfmuref_enable
函数原型	void cfmu_cfmuref_enable(void)
功能描述	使能 CFMUREF 引脚输入
先决条件	-
被调用函数	-
输入参数{in}	
-	-
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* enable CFMUREF pin input */
```

```
cfmu_cfmuref_enable();
```

### 函数 cfmu\_cfmuref\_disable

函数cfmu\_cfmuref\_disable描述见下表：

表 3-141. 函数 cfmu\_cfmuref\_disable

函数名称	cfmu_cfmuref_disable
函数原型	void cfmu_cfmuref_disable(void)
功能描述	禁能 CFMUREF 引脚输入
先决条件	-
被调用函数	-
输入参数{in}	
-	-
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* disable CFMUREF pin input */
```

```
cfmu_cfmuref_disable();
```

### 函数 cfmu\_reference\_signal\_config

函数cfmu\_reference\_signal\_config描述见下表：

表 3-142. 函数 cfmu\_reference\_signal\_config

函数名称	cfmu_reference_signal_config
函数原型	void cfmu_reference_signal_config(uint32_t rssel)
功能描述	配置参考信号
先决条件	-
被调用函数	-
输入参数{in}	
rssel	参考信号选择
CFMU_RSSEL_CFMU_REF	参考信号选择 CFMUREF 引脚
CFMU_RSSEL_INTERNAL_CLOCK	参考信号选择内部时钟
输出参数{out}	
-	-
返回值	

-	-
---	---

例如:

```
/* configure the reference signal */
```

```
cfmu_reference_signal_config(CFMU_RSSEL_CFMUREF);
```

### 函数 cfmu\_digital\_filter\_config

函数cfmu\_digital\_filter\_config描述见下表:

表 3-143. 函数 cfmu\_digital\_filter\_config

函数名称	cfmu_digital_filter_config
函数原型	void cfmu_digital_filter_config(uint32_t dfssel)
功能描述	配置数字滤波器
先决条件	-
被调用函数	-
输入参数{in}	
dfssel	数字滤波器选择
CFMU_DFSEL_DISABLE	禁用数字滤波器
CFMU_DFSEL_DIV1	数字滤波器的采样时钟是频率测量时钟除以 1
CFMU_DFSEL_DIV4	数字滤波器的采样时钟是频率测量时钟除以 4
CFMU_DFSEL_DIV16	数字滤波器的采样时钟是频率测量时钟除以 16
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* configure the digital filter */
```

```
cfmu_digital_filter_config(CFMU_DFSEL_DISABLE);
```

### 函数 cfmu\_reference\_clock\_config

函数cfmu\_reference\_clock\_config描述见下表:

表 3-144. 函数 cfmu\_reference\_clock\_config

函数名称	cfmu_reference_clock_config
函数原型	void cfmu_reference_clock_config(uint32_t rck_src, uint32_t rck_div, uint32_t val_edge)
功能描述	配置测量参考时钟源
先决条件	-
被调用函数	-

输入参数{in}	
<b>rck_src</b>	参考时钟源
<i>CFMU_RCKSRC_HXTAL</i>	选择 HXTAL
<i>CFMU_RCKSRC_IRC32M</i>	选择 IRC32M
<i>CFMU_RCKSRC_IRC32K</i>	选择 IRC32K
<i>CFMU_RCKSRC_PCLK1</i>	选择 PCLK1
输入参数{in}	
<b>rck_div</b>	参考时钟源分频
<i>CFMU_RCK_DIVx(x=3, 2, 128, 1024, 8192)</i>	参考时钟除以 x
输入参数{in}	
<b>val_edge</b>	有效边沿
<i>CFMU_VAL_RISING</i>	上升沿有效
<i>CFMU_VAL_FALLING</i>	下降沿有效
<i>CFMU_VAL_BOTH</i>	上升沿和下降沿均有效
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* configure the measurement reference clock source */
```

```
cfmu_reference_clock_config();
```

### 函数 cfmu\_measurement\_clock\_config

函数cfmu\_measurement\_clock\_config描述见下表：

表 3-145. 函数 cfmu\_measurement\_clock\_config

函数名称	cfmu_measurement_clock_config
函数原型	void cfmu_measurement_clock_config(uint32_t mck_src, uint32_t mck_div)
功能描述	配置测量目标时钟源
先决条件	-
被调用函数	-
输入参数{in}	
<b>mck_src</b>	测量时钟
<i>CFMU_MCKSRC_HXTAL</i>	选择 HXTAL



AL	
CFMU_MCKSRC_IRC3 2M	选择 IRC32M
CFMU_MCKSRC_IRC3 2K	选择 IRC32K
CFMU_MCKSRC_PCL K1	选择 PCLK1
输入参数{in}	
mck_div	测量时钟分频器
CFMU_MCK_DIVx(x=1, 4,8,32)	MCK 除以 x
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* configure the measurement target clock source */
```

```
cfmu_reference_clock_config(CFMU_RCKSRC_HXTAL,CFMU_RCK_DIV3,  
CFMU_VAL_RISING);
```

### 函数 cfmu\_limit\_value\_config

函数cfmu\_limit\_value\_config描述见下表:

表 3-146. 函数 cfmu\_limit\_value\_config

函数名称	cfmu_limit_value_config
函数原型	void cfmu_limit_value_config(uint32_t cfmu_hlv, uint32_t cfmu_llv)
功能描述	配置 CFMU 的上限值和下限值
先决条件	-
被调用函数	-
输入参数{in}	
cfmu_hlv	上限值
输入参数{in}	
cfmu_llv	下限值
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* configure CFMU higher-limit value and lower-limit value */
```

```
cfmu_limit_value_config(0xFF,0x10);
```

### 函数 cfmu\_interrupt\_enable

函数cfmu\_interrupt\_enable描述见下表:

表 3-147. 函数 cfmu\_interrupt\_enable

函数名称	cfmu_interrupt_enable
函数原型	void cfmu_interrupt_enable(cfmu_int_enum cfmu_interrupt)
功能描述	使能 CFMU 中断
先决条件	-
被调用函数	-
输入参数{in}	
cfmu_interrupt	参考 <a href="#">表 3-134. 枚举类型 cfmu_int_enum</a>
CFMU_INT_OVF	溢出中断请求
CFMU_INT_CFMEND	测量结束中断请求
CFMU_INT_CFERR	频率错误中断请求
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* enable the CFMU interrupt */
```

```
cfmu_interrupt_enable(CFMU_INT_OVF);
```

### 函数 cfmu\_interrupt\_disable

函数cfmu\_interrupt\_disable描述见下表:

表 3-148. 函数 cfmu\_interrupt\_disable

函数名称	cfmu_interrupt_disable
函数原型	void cfmu_interrupt_disable(cfmu_int_enum cfmu_interrupt)
功能描述	禁能 CFMU 中断
先决条件	-
被调用函数	-
输入参数{in}	
cfmu_interrupt	参考 <a href="#">表 3-134. 枚举类型 cfmu_int_enum</a>
CFMU_INT_OVF	溢出中断请求
CFMU_INT_CFMEND	测量结束中断请求
CFMU_INT_CFERR	频率错误中断请求
输出参数{out}	
-	-

返回值	
-	-

例如：

```
/* disable the CFMU interrupt */
```

```
cfmu_interrupt_disable(CFMU_INT_OVF);
```

### 函数 cfmu\_interrupt\_flag\_get

函数cfmu\_interrupt\_flag\_get描述见下表：

表 3-149. 函数 cfmu\_interrupt\_flag\_get

函数名称	cfmu_interrupt_flag_get
函数原型	FlagStatus cfmu_interrupt_flag_get(cfmu_int_flag_enum cfmu_int_flag)
功能描述	获取 CFMU 中断标志
先决条件	-
被调用函数	-
输入参数{in}	
cfmu_int_flag	中断标志，参考 <a href="#">表 3-135. 枚举类型 cfmu_int_flag_enum</a>
CFMU_INT_FLAG_OVF	溢出标志
CFMU_INT_FLAG_CFMEND	时钟频率精度测量结束标志
CFMU_INT_FLAG_CFEERR	时钟频率错误标志
输出参数{out}	
-	-
返回值	
FlagStatus	SET 或 RESET

例如：

```
/* get the CFMU interrupt flags */
```

```
cfmu_interrupt_flag_get(CFMU_INT_FLAG_OVF);
```

### 函数 cfmu\_interrupt\_flag\_clear

函数cfmu\_interrupt\_flag\_clear描述见下表：

表 3-150. 函数 cfmu\_interrupt\_flag\_clear

函数名称	cfmu_interrupt_flag_clear
函数原型	void cfmu_interrupt_flag_clear(cfmu_int_flag_clear_enum cfmu_int_flag_clear)
功能描述	清除 CFMU 中断标志

先决条件	-
被调用函数	-
输入参数{in}	
cfmu_int_flag	中断标志, 参考 <a href="#">表 3-136. 枚举类型 cfmu_int_flag_clear_enum</a>
CFMU_INT_FLAG_OVF_CLR	溢出标志清除
CFMU_INT_FLAG_CFMEND_CLR	时钟频率精度测量结束标志清除
CFMU_INT_FLAG_CFMERR_CLR	时钟频率错误标志清除
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* clear the CFMU interrupt flags */
```

```
cfmu_interrupt_flag_clear(CFMU_INT_FLAG_OVF_CLR);
```

## 3.5. CMP

CMP通用比较器可独立工作, 其输出端口可用于I/O口, 也可和定时器结合使用。在一定的条件下, 比较器可将模拟信号作为触发源, 结合定时器的PWM输出, 可以实现电流控制。章节[3.5.1](#)描述了CMP的寄存器列表, 章节[3.5.2](#)对CMP库函数进行说明。

### 3.5.1. 外设寄存器说明

CMP寄存器列表如下表所示:

表 3-151. CMP 寄存器

寄存器名称	寄存器描述
CMP_STAT	比较器状态控制器
CMP_IFC	比较器中断标志位清除寄存器
CMP0_CS	CMP0控制状态寄存器
CMP1_CS	CMP1控制状态寄存器
CMP2_CS	CMP2控制状态寄存器
CMP3_CS	CMP3控制状态寄存器

## 3.5.2. 外设库函数说明

表 3-152. 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_digital_filter_init	CMP数字滤波器初始化
cmp_enable	使能CMP
cmp_disable	禁能CMP
cmp_lock_enable	锁定CMP
cmp_output_to_pin_enable	使能CMP输出到引脚
cmp_output_to_pin_disable	禁能CMP输出到引脚
cmp_output_enable	使能CMP输出
cmp_output_disable	禁能CMP输出
cmp_output_level_get	获取CMP输出状态
cmp_flag_get	获取CMP标志位
cmp_flag_clear	清除CMP标志位
cmp_interrupt_enable	CMP中断使能
cmp_interrupt_disable	CMP中断禁能
cmp_interrupt_flag_get	获取CMP中断标志位
cmp_interrupt_flag_clear	清除CMP中断标志位

## 枚举类型 cmp\_enum

表 3-153. 枚举类型 cmp\_enum

成员名称	功能描述
CMP0	比较器0
CMP1	比较器1
CMP2	比较器2
CMP3	比较器3

## 函数 cmp\_deinit

函数cmp\_deinit描述见下表：

表 3-154. 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}	
<b>cmp_periph</b>	参考枚举 <a href="#">表 3-153. 枚举类型cmp_enum</a>
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 3-155. 函数 cmp\_mode\_init

函数名称	cmp_mode_init
函数原型	void cmp_mode_init(cmp_enum cmp_periph, uint32_t inverting_input, uint32_t output_hysteresis);
功能描述	CMP工作模式初始化
先决条件	-
被调用函数	-
输入参数{in}	
<b>cmp_periph</b>	参考枚举 <a href="#">表 3-153. 枚举类型cmp_enum</a>
输入参数{in}	
<b>inverting_input</b>	反相输入源
CMP_INVERTING_INPUT_NONE	无输入源
CMP_INVERTING_INPUT_PC11	选择PC11作为输入源
CMP_INVERTING_INPUT_PC12	选择PC12作为输入源
CMP_INVERTING_INPUT_DAC0_OUT0	选择DAC0_OUT0作为输入源
CMP_INVERTING_INPUT_DAC0_OUT1	选择DAC0_OUT1作为输入源
输入参数{in}	
<b>output_hysteresis</b>	迟滞水平
CMP_HYSTERESIS_NO	无迟滞
CMP_HYSTERESIS_LOW	低迟滞

<code>CMP_HYSTERESIS_MIDDLE</code>	中迟滞
<code>CMP_HYSTERESIS_HIGH</code>	高迟滞
输出参数{out}	
-	-
返回值	
-	-

例如：

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

```
cmp_mode_init(CMP0, CMP_INVERTING_INPUT_PC11, CMP_HYSTERESIS_NO);
```

### 函数 `cmp_noninverting_input_select`

函数 `cmp_noninverting_input_select` 描述见下表：

表 3-156. 函数 `cmp_noninverting_input_select`

函数名称	<code>cmp_noninverting_input_select</code>
函数原型	<code>void cmp_noninverting_input_select(cmp_enum cmp_periph, uint32_t noninverting_input);</code>
功能描述	CMP选择正相输入源
先决条件	-
被调用函数	-
输入参数{in}	
<code>cmp_periph</code>	参考枚举 <a href="#">表 3-153. 枚举类型 <code>cmp_enum</code></a>
输入参数{in}	
<code>noninverting_input</code>	正相输入源
<code>CMP_NONINVERTING_INPUT_NONE</code>	无输入源
<code>CMP_NONINVERTING_INPUT_PC0_PC1_PC2_PC3</code>	PC0作为CMP0输入源或者PC1作为CMP1输入源或者PC2作为CMP2输入源或者PC3作为CMP3输入源
<code>CMP_NONINVERTING_INPUT_PC4_PC5_PC6_PD2</code>	PC4作为CMP0输入源或者PC5作为CMP1输入源或者PC6作为CMP2输入源或者PD2作为CMP3输入源
<code>CMP_NONINVERTING_INPUT_PC10_PC11_PC12_PD4</code>	PC10作为CMP0输入源或者PC11作为CMP1输入源或者PC12作为CMP2输入源或者PD4作为CMP3输入源
<code>CMP_NONINVERTING_INPUT_PC7_PC0_PC3_PD5</code>	PC7作为CMP0输入源或者PC0作为CMP1输入源或者PC3作为CMP2输入源或者PD5作为CMP3输入源
<code>CMP_NONINVERTING_INPUT_PC10_PC9</code>	PC10作为CMP2输入源或者PC9作为CMP3输入源

NG_INPUT_PC10_ PC9	
输出参数{out}	
-	-
返回值	
-	-

例如：

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

```
cmp_noninverting_input_select(CMP0,  
CMP_NONINVERTING_INPUT_PC0_PC1_PC2_PC3);
```

### 函数 cmp\_output\_init

函数cmp\_output\_init描述见下表：

表 3-157. 函数 cmp\_output\_init

函数名称	cmp_output_init
函数原型	void cmp_output_init(cmp_enum cmp_periph, uint32_t output_polarity);
功能描述	CMP输出初始化
先决条件	-
被调用函数	-
输入参数{in}	
cmp_periph	参考枚举 <a href="#">表 3-153. 枚举类型cmp_enum</a>
输入参数{in}	
output_polarity	CMP输出极性
CMP_OUTPUT_PO LARITY_INVERTED	输出反相
CMP_OUTPUT_PO LARITY_NONINVE RTED	输出正相
输出参数{out}	
-	-
返回值	
-	-

例如：

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

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

### 函数 cmp\_digital\_filter\_init

函数cmp\_digital\_filter\_init描述见下表：



表 3-158. 函数 `cmp_digital_filter_init`

函数名称	<code>cmp_digital_filter_init</code>
函数原型	<code>void cmp_digital_filter_init(cmp_enum cmp_periph, uint32_t sampling_frequency, uint32_t sampling_number);</code>
功能描述	CMP数字滤波器初始化
先决条件	-
被调用函数	-
输入参数{in}	
<code>cmp_periph</code>	参考枚举 <a href="#">表 3-153. 枚举类型 <code>cmp_enum</code></a>
输入参数{in}	
<code>sampling_frequency</code>	采样频率
<code>CMP_DIGITAL_FILTER_NOT_USED</code>	不使用数字滤波器
<code>CMP_SAMPLING_FREQUENCY_DIV8</code>	采样频率为 $f_{CK\_CMP}/8$
<code>CMP_SAMPLING_FREQUENCY_DIV16</code>	采样频率为 $f_{CK\_CMP}/16$
<code>CMP_SAMPLING_FREQUENCY_DIV32</code>	采样频率为 $f_{CK\_CMP}/32$
输入参数{in}	
<code>sampling_number</code>	有效采样数
<code>CMP_SAMPLING_NUM_3_TIMES</code>	滤波器有效采样数为3次
<code>CMP_SAMPLING_NUM_4_TIMES</code>	滤波器有效采样数为4次
输出参数{out}	
-	-
返回值	
-	-

例如：

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

```
cmp_digital_filter_init(CMP0,                                CMP_SAMPLING_FREQUENCY_DIV8,
                        CMP_SAMPLING_NUM_3_TIMES);
```

### 函数 `cmp_enable`

函数`cmp_enable`描述见下表：

表 3-159. 函数 `cmp_enable`

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

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 <a href="#">表 3-153. 枚举类型cmp_enum</a>
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 3-160. 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 <a href="#">表 3-153. 枚举类型cmp_enum</a>
输出参数{out}	
-	-
返回值	
-	-

例如：

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

### 函数 cmp\_lock\_enable

函数cmp\_lock\_enable描述见下表：

表 3-161. 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 <a href="#">表 3-153. 枚举类型cmp_enum</a>
输出参数{out}	
-	-
返回值	
-	-

例如：

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

### 函数 cmp\_output\_to\_pin\_enable

函数cmp\_output\_to\_pin\_enable描述见下表：

表 3-162. 函数 cmp\_output\_to\_pin\_enable

函数名称	cmp_output_to_pin_enable
函数原型	void cmp_output_to_pin_enable(cmp_enum cmp_periph);
功能描述	使能CMP输出到引脚
先决条件	-
被调用函数	-
输入参数{in}	
cmp_periph	参考枚举 <a href="#">表 3-153. 枚举类型cmp_enum</a>
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* enable CMP0 output to pin */
cmp_output_to_pin_enable(CMP0);
```

### 函数 cmp\_output\_to\_pin\_disable

函数cmp\_output\_to\_pin\_disable描述见下表：

表 3-163. 函数 cmp\_output\_to\_pin\_disable

函数名称	cmp_output_to_pin_disable
函数原型	void cmp_output_to_pin_disable (cmp_enum cmp_periph);
功能描述	禁能CMP输出到引脚

先决条件	-
被调用函数	-
输入参数{in}	
cmp_periph	参考枚举 <a href="#">表 3-153. 枚举类型cmp_enum</a>
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* disable CMP0 output to pin */
```

```
cmp_output_to_pin_disable(CMP0);
```

### 函数 cmp\_output\_enable

函数cmp\_output\_enable描述见下表：

表 3-164. 函数 cmp\_output\_enable

函数名称	cmp_output_enable
函数原型	void cmp_output_enable(cmp_enum cmp_periph);
功能描述	使能CMP输出
先决条件	-
被调用函数	-
输入参数{in}	
cmp_periph	参考枚举 <a href="#">表 3-153. 枚举类型cmp_enum</a>
输出参数{out}	
-	-
返回值	
-	-

例如：

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

```
cmp_output_enable(CMP0);
```

### 函数 cmp\_output\_disable

函数cmp\_output\_disable描述见下表：

表 3-165. 函数 cmp\_output\_disable

函数名称	cmp_output_disable
函数原型	void cmp_output_disable(cmp_enum cmp_periph);
功能描述	禁能CMP输出
先决条件	-

被调用函数	-
输入参数{in}	
cmp_periph	参考枚举 <a href="#">表 3-153. 枚举类型cmp_enum</a>
输出参数{out}	
-	-
返回值	
-	-

例如：

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

```
cmp_output_disable(CMP0);
```

### 函数 cmp\_output\_level\_get

函数cmp\_output\_level\_get描述见下表：

表 3-166. 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 <a href="#">表 3-153. 枚举类型cmp_enum</a>
输出参数{out}	
-	-
返回值	
uint32_t	输出电平
CMP_OUTPUTLEV EL_HIGH	比较器输出高电平
CMP_OUTPUTLEV EL_LOW	比较器输出低电平

例如：

```
uint32_t level;
```

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

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

### 函数 cmp\_flag\_get

函数cmp\_flag\_get描述见下表：

表 3-167. 函数 cmp\_flag\_get

函数名称	cmp_flag_get
函数原形	FlagStatus cmp_flag_get(cmp_enum cmp_periph, uint32_t flag);
功能描述	获取CMP标志位
先决条件	-
被调用函数	-
输入参数{in}	
cmp_periph	参考枚举 <a href="#">表 3-153. 枚举类型cmp_enum</a>
输入参数{in}	
flag	CMP中断标志位
CMP_FLAG_COMPARE	CMP0比较中断标志位
输出参数{out}	
-	-
返回值	
FlagStatus	SET或RESET

例如：

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

```
FlagStatus flag_value;
```

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

### 函数 cmp\_flag\_clear

函数cmp\_flag\_clear描述见下表：

表 3-168. 函数 cmp\_flag\_clear

函数名称	cmp_flag_clear
函数原形	void cmp_flag_clear(cmp_enum cmp_periph, uint32_t flag);
功能描述	清除CMP标志位
先决条件	-
被调用函数	-
输入参数{in}	
cmp_periph	参考枚举 <a href="#">表 3-153. 枚举类型cmp_enum</a>
输入参数{in}	
flag	CMP中断标志位
CMP_FLAG_COMPARE	CMP0比较中断标志位
输出参数{out}	
-	-
返回值	
-	-

例如：

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

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

### 函数 cmp\_interrupt\_enable

函数cmp\_interrupt\_enable描述见下表：

表 3-169. 函数 cmp\_interrupt\_enable

函数名称	cmp_interrupt_enable
函数原形	void cmp_interrupt_enable(cmp_enum cmp_periph, uint32_t trigger_mode, uint32_t interrupt);
功能描述	CMP中断使能
先决条件	-
被调用函数	-
输入参数{in}	
cmp_periph	参考枚举 <a href="#">表 3-153. 枚举类型cmp_enum</a>
输入参数{in}	
trigger_mode	触发模式
CMP_INT_RISING_EDGE	上升沿产生中断
CMP_INT_FALLING_EDGE	下降沿产生中断
CMP_INT_BOTH_EDGE	双边沿产生中断
输入参数{in}	
interrupt	CMP中断
CMP_INT_COMPARE	CMP比较中断
输出参数{out}	
-	-
返回值	
-	-

例如：

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

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

### 函数 cmp\_interrupt\_disable

函数cmp\_interrupt\_disable描述见下表：

表 3-170. 函数 cmp\_interrupt\_disable

函数名称	cmp_interrupt_disable
函数原形	void cmp_interrupt_disable(cmp_enum cmp_periph, uint32_t interrupt);
功能描述	CMP中断禁能
先决条件	-
被调用函数	-
输入参数{in}	
cmp_periph	参考枚举 <a href="#">表 3-153. 枚举类型cmp_enum</a>
输入参数{in}	
interrupt	CMP中断
CMP_INT_COMPARE	CMP比较中断
输出参数{out}	
-	-
返回值	
-	-

例如：

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

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

### 函数 cmp\_interrupt\_flag\_get

函数cmp\_interrupt\_flag\_get描述见下表：

表 3-171. 函数 cmp\_interrupt\_flag\_get

函数名称	cmp_interrupt_flag_get
函数原形	FlagStatus cmp_interrupt_flag_get(cmp_enum cmp_periph, uint32_t flag);
功能描述	获取CMP中断标志位
先决条件	-
被调用函数	-
输入参数{in}	
cmp_periph	参考枚举 <a href="#">表 3-153. 枚举类型cmp_enum</a>
输入参数{in}	
flag	CMP中断标志位
CMP_INT_FLAG_COMPARE	CMP比较中断标志位
输出参数{out}	
-	-
返回值	
FlagStatus	SET或RESET

例如：



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

```
FlagStatus flag_value;
```

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

### 函数 cmp\_interrupt\_flag\_clear

函数cmp\_interrupt\_flag\_clear描述见下表：

表 3-172. 函数 cmp\_interrupt\_flag\_clear

函数名称	cmp_interrupt_flag_clear
函数原形	void cmp_interrupt_flag_clear(cmp_enum cmp_periph, uint32_t flag);
功能描述	清除CMP中断标志位
先决条件	-
被调用函数	-
输入参数{in}	
cmp_periph	参考枚举 <a href="#">表 3-153. 枚举类型cmp_enum</a>
输入参数{in}	
flag	CMP中断标志位
CMP_INT_FLAG_COMPARE	CMP比较中断标志位
输出参数{out}	
-	-
返回值	
-	-

例如：

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

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

## 3.6. CPTIMER

比较匹配定时器模块CPTIMER0/1 有两个独立的16位无符号计数器，两个模块共4个计数器。比较匹配定时器可以被配置成以设定周期产生中断或者EVIC事件请求。章节[3.6.1](#)描述了CPTIMER的寄存器列表，章节[3.6.2](#)对CPTIMER库函数进行说明。

### 3.6.1. 外设寄存器说明

CPTIMER寄存器列表如下表所示：

表 3-173. CPTIMER 寄存器

寄存器名称	寄存器描述
CPTIMER_CTL0	CPTIMER control register

寄存器名称	寄存器描述
CPTIMER_CNT0	CPTIMER counter 0 register
CPTIMER_CNT1	CPTIMER counter 1 register
CPTIMER_CNT0 PSC	CPTIMER counter 0 prescaler register
CPTIMER_CNT1 PSC	CPTIMER counter 1 prescaler register
CPTIMER_INTEN	CPTIMER interrupt enable register
CPTIMER_INTF	CPTIMER interrupt flag register
CPTIMER_SWEV G	CPTIMER software event generation register
CPTIMER_CNT0 CAR	CPTIMER counter 0 autoreload register
CPTIMER_CNT1 CAR	CPTIMER counter 1 autoreload register

### 3.6.2. 外设库函数说明

表 3-174. CPTIMER 库函数

库函数名称	库函数描述
cptimer_deinit	CPTIMER deinit
cptimer_enable	enable CPTIMER
cptimer_disable	disable CPTIMER
cptimer_prescaler_config	configure CPTIMER prescaler
cptimer_autoreload_value_config	configure CPTIMER autoreload value
cptimer_counter_value_config	configure CPTIMER counter value
cptimer_counter_read	read CPTIMER counter value
cptimer_prescaler_read	read CPTIMER prescaler value
cptimer_event_software_generate	software generate events
cptimer_flag_get	get CPTIMER flag
cptimer_flag_clear	clear CPTIMER flag
cptimer_interrupt_enable	enable CPTIMER interrupt
cptimer_interrupt_disable	disable CPTIMER interrupt
cptimer_interrupt_flag_get	get CPTIMER interrupt flag
cptimer_interrupt_flag_clear	clear CPTIMER interrupt flag

#### 函数 cptimer\_deinit

函数cptimer\_deinit描述见下表：

表 3-175. 函数 cptimer\_deinit

函数名称	cptimer_deinit
函数原型	void cptimer_deinit(uint32_t cptimer_periph);

功能描述	复位CPTIMER
先决条件	-
被调用函数	rcu_periph_reset_enable / rcu_periph_reset_disable
输入参数{in}	
cptimer_periph	cptimer外设
CPTIMERx	x=0,1
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* deinitialize CPTIMER0 */
cptimer_deinit(CPTIMER0);
```

### 函数 cptimer\_enable

函数cptimer\_enable描述见下表：

表 3-176. 函数 cptimer\_enable

函数名称	cptimer_enable
函数原型	void cptimer_enable(uint32_t cptimer_periph);
功能描述	使能CPTIMER
先决条件	-
被调用函数	-
输入参数{in}	
cptimer_periph	cptimer外设
CPTIMERx	x=0,1
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* enable CPTIMER0 */
cptimer_enable(CPTIMER0);
```

### 函数 cptimer\_disable

函数cptimer\_disable描述见下表：

表 3-177. 函数 cptimer\_disable

函数名称	cptimer_disable
函数原型	void cptimer_disable(uint32_t cptimer_periph);

功能描述	禁能CPTIMER
先决条件	-
被调用函数	-
输入参数{in}	
<b>cptimer_periph</b>	cptimer外设
CPTIMERx	x=0,1
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* disable CPTIMER0 */
cptimer_disable(CPTIMER0);
```

### 函数 **cptimer\_prescaler\_config**

函数cptimer\_prescaler\_config描述见下表:

表 3-178. 函数 **cptimer\_prescaler\_config**

函数名称	cptimer_prescaler_config
函数原型	void cptimer_prescaler_config(uint32_t cptimer_periph , uint32_t counter, uint16_t prescaler);
功能描述	配置CPTIMER时钟预分频
先决条件	-
被调用函数	-
输入参数{in}	
<b>cptimer_periph</b>	cptimer外设
CPTIMERx	x=0,1
输入参数{in}	
<b>counter</b>	cptimer计数器
CPTIMER_COUNT ERx	x=0,1
输入参数{in}	
<b>prescaler</b>	预分频值,0~65535
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* configure the CPTIMER0 counter 0 prescaler */
cptimer_prescaler_config(CPTIMER0, CPTIMER_COUNTER0, 10U);
```

**函数 cptimer\_autoreload\_value\_config**

函数cptimer\_autoreload\_value\_config描述见下表:

**表 3-179. 函数 cptimer\_autoreload\_value\_config**

函数名称	cptimer_autoreload_value_config
函数原型	void cptimer_autoreload_value_config(uint32_t cptimer_periph , uint32_t counter, uint16_t autoreload);
功能描述	配置CPTIMER重装载值
先决条件	-
被调用函数	-
输入参数{in}	
<b>cptimer_periph</b>	cptimer外设
CPTIMERx	x=0,1
输入参数{in}	
<b>counter</b>	cptimer counter
CPTIMER_COUNT ERx	x=0,1
输入参数{in}	
<b>autoreload</b>	重装载值,0~65535
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* configure the CPTIMER0 counter 0 autoreload value */
cptimer_autoreload_value_config (CPTIMER0, CPTIMER_COUNTER0, 10U);
```

**函数 cptimer\_counter\_value\_config**

函数cptimer\_counter\_value\_config描述见下表:

**表 3-180. 函数 cptimer\_counter\_value\_config**

函数名称	cptimer_counter_value_config
函数原型	void cptimer_counter_value_config(uint32_t cptimer_periph, uint32_t counter, uint16_t value);
功能描述	配置CPTIMER计数器寄存器值
先决条件	-
被调用函数	-
输入参数{in}	
<b>cptimer_periph</b>	cptimer外设
CPTIMERx	x=0,1
输入参数{in}	

<b>counter</b>	cptimer计数器
CPTIMER_COUNT ERx	x=0,1
输入参数{in}	
<b>value</b>	计数器值, 0~65535
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* configure CPTIMER counter register value */
cptimer_counter_value_config (CPTIMER0, CPTIMER_COUNTER0, 10U);
```

### 函数 cptimer\_counter\_read

函数cptimer\_counter\_read描述见下表:

表 3-181. 函数 cptimer\_counter\_read

<b>函数名称</b>	cptimer_counter_read
<b>函数原型</b>	uint16_t cptimer_counter_read(uint32_t cptimer_periph, uint32_t counter);
<b>功能描述</b>	读取CPTIMER计数器值
<b>先决条件</b>	-
<b>被调用函数</b>	-
输入参数{in}	
<b>cptimer_periph</b>	cptimer外设
CPTIMERx	x=0,1
输入参数{in}	
<b>counter</b>	cptimer计数器
CPTIMER_COUNT ERx	x=0,1
输出参数{out}	
-	-
返回值	
<b>uint16_t</b>	计数器值(0~65535)

例如:

```
/* read CPTIMER counter value */
uint16_t cnt = cptimer_counter_read(CPTIMER0, CPTIMER_COUNTER0);
```

**函数 cptimer\_prescaler\_read**

函数cptimer\_prescaler\_read描述见下表：

**表 3-182. 函数 cptimer\_prescaler\_read**

函数名称	cptimer_prescaler_read
函数原型	uint16_t cptimer_prescaler_read(uint32_t cptimer_periph, uint32_t counter);
功能描述	读取CPTIMER预分频值
先决条件	-
被调用函数	-
输入参数{in}	
cptimer_periph	cptimer外设
CPTIMERx	x=0,1
输入参数{in}	
counter	cptimer计数器
CPTIMER_COUNT ERx	x=0,1
输出参数{out}	
-	-
返回值	
uint16_t	预分频值(0~65535)

例如：

```
/* read CPTIMER prescaler value */
uint16_t pre = cptimer_prescaler_read(CPTIMER0, CPTIMER_COUNTER0);
```

**函数 cptimer\_event\_software\_generate**

函数cptimer\_event\_software\_generate描述见下表：

**表 3-183. 函数 cptimer\_event\_software\_generate**

函数名称	cptimer_event_software_generate
函数原型	void cptimer_event_software_generate(uint32_t cptimer_periph, uint32_t event);
功能描述	软件事件产生
先决条件	-
被调用函数	-
输入参数{in}	
cptimer_periph	cptimer外设
CPTIMERx	x=0,1
输入参数{in}	
event	软件事件
CPTIMER_EVENT_ SRC_UPG0	计数器0更新软件事件

CPTIMER_EVENT_SRC_UPG1	计数器1更新软件事件
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* generate CPTIMER0 counter 0 update event */
cptimer_event_software_generate (CPTIMER0, CPTIMER_EVENT_SRC_UPG0);
```

### 函数 cptimer\_flag\_get

函数cptimer\_flag\_get描述见下表:

表 3-184. 函数 cptimer\_flag\_get

函数名称	cptimer_flag_get
函数原形	FlagStatus cptimer_flag_get(uint32_t cptimer_periph, uint32_t flag);
功能描述	获取CPTIMER标志位
先决条件	-
被调用函数	-
输入参数{in}	
cptimer_periph	cptimer外设
CPTIMERx	x=0,1
输入参数{in}	
flag	CPTIMER标志位
CPTIMER_FLAG_CNT0UP	CPTIMER计数器0更新标志
CPTIMER_FLAG_CNT1UP	CPTIMER计数器1更新标志
输出参数{out}	
-	-
返回值	
FlagStatus	SET or RESET

例如:

```
/* get the CPTIMER0 flag bit */
FlagStatus flag_value;
flag_value = cptimer_flag_get(CPTIMER0, CPTIMER_FLAG_CNT1UP);
```

### 函数 cptimer\_flag\_clear

函数cptimer\_flag\_clear描述见下表:



表 3-185. 函数 `cptimer_flag_clear`

函数名称	<code>cptimer_flag_clear</code>
函数原形	<code>void cptimer_flag_clear(uint32_t cptimer_periph, uint32_t flag);</code>
功能描述	清除CPTIMER标志位
先决条件	-
被调用函数	-
输入参数{in}	
<code>cptimer_periph</code>	cptimer外设
CPTIMERx	x=0,1
输入参数{in}	
<code>flag</code>	CPTIMER标志位
<code>CPTIMER_FLAG_CNT0UP</code>	CPTIMER计数器0更新标志
<code>CPTIMER_FLAG_CNT1UP</code>	CPTIMER计数器1更新标志
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* clear the CPTIMER0 flag bit */
cptimer_flag_clear(CPTIMER0, CPTIMER_FLAG_CNT0UP);
```

### 函数 `cptimer_interrupt_enable`

函数`cptimer_interrupt_enable`描述见下表：

表 3-186. 函数 `cptimer_interrupt_enable`

函数名称	<code>cptimer_interrupt_enable</code>
函数原形	<code>void cptimer_interrupt_enable(uint32_t cptimer_periph, uint32_t interrupt);</code>
功能描述	CPTIMER中断使能
先决条件	-
被调用函数	-
输入参数{in}	
<code>cptimer_periph</code>	cptimer外设
CPTIMERx	x=0,1
输入参数{in}	
<code>interrupt</code>	CPTIMER中断
<code>CPTIMER_INT_CN_T0UP</code>	CPTIMER计数器0更新中断
<code>CPTIMER_INT_CN_T1UP</code>	CPTIMER计数器1更新中断

输出参数{out}	
-	-
输出参数{out}	
-	-

例如：

```
/* enable the CPTIMER0 interrupt */
cptimer_interrupt_enable(CPTIMER0, CPTIMER_INT_CNT1UP);
```

### 函数 `cptimer_interrupt_disable`

函数 `cptimer_interrupt_disable` 描述见下表：

**表 3-187. 函数 `cptimer_interrupt_disable`**

函数名称	<code>cptimer_interrupt_disable</code>
函数原形	<code>void cptimer_interrupt_disable(uint32_t cptimer_periph, uint32_t interrupt);</code>
功能描述	CPTIMER中断禁能
先决条件	-
被调用函数	-
输入参数{in}	
<b><code>cptimer_periph</code></b>	cptimer外设
CPTIMERx	x=0,1
输入参数{in}	
<b><code>interrupt</code></b>	CPTIMER中断
CPTIMER_INT_CN T0UP	CPTIMER计数器0更新中断
CPTIMER_INT_CN T1UP	CPTIMER计数器0更新中断
输出参数{out}	
-	-
输出参数{out}	
-	-

例如：

```
/* disable the CPTIMER0 interrupt */
cptimer_interrupt_disable(CPTIMER0, CPTIMER_INT_CNT1UP);
```

### 函数 `cptimer_interrupt_flag_get`

函数 `cptimer_interrupt_flag_get` 描述见下表：

**表 3-188. 函数 `cptimer_interrupt_flag_get`**

函数名称	<code>cptimer_interrupt_flag_get</code>
函数原形	<code>FlagStatus cptimer_interrupt_flag_get(uint32_t cptimer_periph, uint32_t</code>

	int_flag);
功能描述	获取CPTIMER中断标志位
先决条件	-
被调用函数	-
输入参数{in}	
cptimer_periph	cptimer外设
CPTIMERx	x=0,1
输入参数{in}	
int_flag	CPTIMER中断标志
CPTIMER_INT_FLAG_CNT0UP	CPTIMER计数器0更新中断标志
CPTIMER_INT_FLAG_CNT1UP	CPTIMER计数器0更新中断标志
输出参数{out}	
-	-
返回值	
FlagStatus	SET or RESET

例如:

```
/* get the CPTIMER0 interrupt bit */
FlagStatus flag_value;
flag_value = cptimer_interrupt_flag_get(CPTIMER0, CPTIMER_INT_FLAG_CNT0UP);
```

### 函数 cptimer\_interrupt\_flag\_clear

函数cptimer\_interrupt\_flag\_clear描述见下表:

表 3-189. 函数 cptimer\_interrupt\_flag\_clear

函数名称	cptimer_interrupt_flag_clear
函数原形	void cptimer_interrupt_flag_clear(uint32_t cptimer_periph, uint32_t int_flag);
功能描述	清除CPTIMER中断标志位
先决条件	-
被调用函数	-
输入参数{in}	
cptimer_periph	cptimer外设
CPTIMERx	x=0,1
输入参数{in}	
int_flag	CPTIMER中断标志
CPTIMER_INT_FLAG_CNT0UP	CPTIMER计数器0更新中断标志
CPTIMER_INT_FLAG_CNT1UP	CPTIMER计数器0更新中断标志
输出参数{out}	

-	-
返回值	
-	-

例如：

```
/* clear the CPTIMER0 interrupt bit */
cptimer_interrupt_flag_clear(CPTIMER0, CPTIMER_INT_FLAG_CNT0UP);
```

### 3.7. CPTIMERW

比较匹配定时器W模块包含一个32位无符号计数器，也可以被配置成16位。比较匹配定时器W有两个输入通道和四个比较匹配通道，其中两个比较匹配通道支持脉冲输出。所有通道以及计数器更新事件可以被配置以设定周期产生中断或者EVIC事件。章节[3.7.1](#)描述了CPTIMERW的寄存器列表，章节[3.7.2](#)对CPTIMERW库函数进行说明。

#### 3.7.1. 外设寄存器说明

CPTIMERW寄存器列表如下表所示：

表 3-190. CPTIMERW 寄存器

寄存器名称	寄存器描述
CPTIMERW_CTL0	CPTIMERW control 0 register
CPTIMERW_CTL1	CPTIMERW control 1 register
CPTIMERW_PSC	CPTIMERW prescaler register
CPTIMERW_CNT	CPTIMERW counter register
CPTIMERW_INTEN	CPTIMERW interrupt enable register
CPTIMERW_INTF	CPTIMERW interrupt flag register
CPTIMERW_SWEVG	CPTIMERW software event generation register
CPTIMERW_ICH0CV	CPTIMERW input channel 0 capture value register
CPTIMERW_ICH1CV	CPTIMERW input channel 1 capture value register
CPTIMERW_OCH0CV	CPTIMERW output compare channel 0 compare value register
CPTIMERW_OCH1CV	CPTIMERW output compare channel 1 compare value register
CPTIMERW_OCH2CV	CPTIMERW output compare channel 2 compare value register

寄存器名称	寄存器描述
CPTIMERW_OCH3CV	CPTIMERW output compare channel 3 compare value register
CPTIMERW_CAR	CPTIMERW autoreload register

### 3.7.2. 外设库函数说明

CPTIMERW库函数列表如下表所示：

**表 3-191. CPTIMERW 库函数**

库函数名称	库函数描述
cptimerw_deinit	CPTIMER deinit
cptimerw_struct_para_init	initialize CPTIMERW init parameter struct
cptimerw_enable	enable CPTIMER
cptimerw_disable	disable CPTIMER
cptimerw_prescaler_config	configure CPTIMER prescaler
cptimerw_autoreload_value_config	configure CPTIMER autoreload value
cptimerw_counter_value_config	configure CPTIMER counter value
cptimerw_counter_clear_source_config	configure CPTIMER counter clear source
cptimerw_counter_read	read CPTIMER counter value
cptimerw_prescaler_read	read CPTIMER prescaler value
cptimerw_channel_output_mode_select	select CPTIMERW channel output compare mode
cptimerw_channel_output_compare_value_config	configure CPTIMERW channel output compare value
cptimerw_channel_output_state_config	configure CPTIMERW output compare channel enable state
cptimerw_channel_input_struct_para_init	initialize CPTIMERW channel input capture parameter struct
cptimerw_input_capture_config	configure CPTIMERW input capture parameter
cptimerw_channel_capture_value_register_read	read CPTIMERW channel input capture value
cptimerw_event_software_generate	software generate events
cptimerw_flag_get	get CPTIMER flag
cptimerw_flag_clear	clear CPTIMER flag
cptimerw_interrupt_enable	enable CPTIMER interrupt
cptimerw_interrupt_disable	disable CPTIMER interrupt
cptimerw_interrupt_flag_get	get CPTIMER interrupt flag
cptimerw_interrupt_flag_clear	clear CPTIMER interrupt flag

## 结构体 `cptimerw_init_parameter_struct`

表 3-192. 结构体 `cptimerw_init_parameter_struct`

成员名称	功能描述
width	计数器位宽 (CPTIMERW_CNT_WIDTH_16BIT, CPTIMERW_CNT_WIDTH_32BIT)
prescaler	计数器时钟预分频系数 (0~0xFFFF)
period	计数器自动重装载值 (16bit:0~0xFFFF, 32-bit:0~0xFFFFFFFF)
clear_source	计数器清零源 (CPTIMERW_CNT_CLEAR_DISABLE, CPTIMERW_CNT_CLEAR_ICH0, CPTIMERW_CNT_CLEAR_ICH1)
clockdivision	输入捕获时钟分频值 (CPTIMERW_CKDIV_DIV1, CPTIMERW_CKDIV_DIV2, CPTIMERW_CKDIV_DIV4)

## 结构体 `cptimerw_ic_parameter_struct`

表 3-193. 结构体 `cptimerw_ic_parameter_struct`

成员名称	功能描述
icedge	通道输入捕获边沿 (CPTIMERW_IC_FALLING_EDGE, CPTIMERW_IC_RISING_EDGE, CPTIMERW_IC_BOTH_EDGE, CPTIMERW_IC_DISABLE)
icfilter	通道输入捕获滤波控制 (0~15)

## 函数 `cptimerw_deinit`

函数 `cptimerw_deinit` 描述见下表:

表 3-194. 函数 `cptimerw_deinit`

函数名称	<code>cptimerw_deinit</code>
函数原型	<code>void cptimerw_deinit(void);</code>
功能描述	CPTIMERW deinit
先决条件	-
被调用函数	<code>rcu_periph_reset_enable</code> / <code>rcu_periph_reset_disable</code>
输入参数{in}	
-	-
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* deinitialize CPTIMERW0 */
cptimerw_deinit(CPTIMERW0);
```

### 函数 cptimerw\_struct\_para\_init

函数 cptimerw\_struct\_para\_init 描述见下表：

**表 3-195. 函数 cptimerw\_struct\_para\_init**

函数名称	cptimerw_struct_para_init
函数原型	void cptimerw_struct_para_init(cptimerw_init_parameter_struct *initpara)
功能描述	初始化 CPTIMER 初始化参数结构体为默认值
先决条件	-
被调用函数	-
输入参数{in}	
initpara	参考 <a href="#">表 3-192. 结构体 cptimerw_init_parameter_struct</a>
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* init parameter struct with a default value */
cptimerw_parameter_struct para;
cptimerw_struct_para_init(&para);
```

### 函数 cptimerw\_init

函数 cptimerw\_init 描述见下表：

**表 3-196. 函数 cptimerw\_init**

函数名称	cptimerw_init
函数原型	void cptimerw_init(cptimerw_init_parameter_struct *initpara)
功能描述	初始化 CPTIMERW 计数器
先决条件	-
被调用函数	-
输入参数{in}	
initpara	参考 <a href="#">表 3-192. 结构体 cptimerw_init_parameter_struct</a>
输出参数{out}	
-	-
返回值	
-	-

例如：

```

/* initialize CPTIMERW counter */
cptimerw_parameter_struct para;
para.width = CPTIMERW_CNT_WIDTH_16BIT;
para.period = 10000U;
para.prescaler = 99;
para.clear_source = CPTIMERW_CNT_CLEAR_DISABLE;
para.clockdivision = CPTIMERW_CKDIV_DIV1;
cptimerw_init(&para);

```

### 函数 cptimerw\_autoreload\_value\_config

函数cptimerw\_autoreload\_value\_config描述见下表:

表 3-197. 函数 cptimerw\_autoreload\_value\_config

函数名称	cptimerw_autoreload_value_config
函数原型	void cptimerw_autoreload_value_config(uint32_t autoreload);
功能描述	配置CPTIMERW自动重装载值
先决条件	-
被调用函数	-
输入参数{in}	
autoreload	计数器自动重装载值，基于位宽设置16或32位值
输出参数{out}	
-	-
返回值	
-	-

例如:

```

/* configure the CPTIMERW autoreload value */
cptimerw_autoreload_value_config (10000U);

```

### 函数 cptimerw\_prescaler\_config

函数cptimerw\_prescaler\_config描述见下表:

表 3-198. 函数 cptimerw\_prescaler\_config

函数名称	cptimerw_prescaler_config
函数原型	void cptimerw_prescaler_config(uint16_t prescaler);
功能描述	配置the CPTIMERW时钟预分频系数
先决条件	-
被调用函数	-
输入参数{in}	
prescaler	时钟预分频值（0~65535）
输出参数{out}	
-	-



返回值	
-	-

例如：

```
/* configure the CPTIMERW prescaler */
cptimerw_prescaler_config(99U);
```

### 函数 cptimerw\_counter\_value\_config

函数cptimerw\_counter\_value\_config描述见下表：

表 3-199. 函数 cptimerw\_counter\_value\_config

函数名称	cptimerw_counter_value_config
函数原型	void cptimerw_counter_value_config(uint16_t value);
功能描述	配置CPTIMERW计数器寄存器值
先决条件	-
被调用函数	-
输入参数{in}	
value	计数器寄存器值，基于位宽设置16或32位值
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* configure CPTIMERW counter register value */
cptimerw_counter_value_config (10U);
```

### 函数 cptimerw\_counter\_clear\_source\_config

函数cptimerw\_counter\_clear\_source\_config描述见下表：

表 3-200. 函数 cptimerw\_counter\_clear\_source\_config

函数名称	cptimerw_counter_clear_source_config
函数原型	void cptimerw_counter_clear_source_config(uint32_t clear_source);
功能描述	配置CPTIMERW计数器清零源
先决条件	-
被调用函数	-
输入参数{in}	
clear_source	计数器清零源
CPTIMERW_CNT_CLEAR_DISABLE	计数器清零禁能
CPTIMERW_CNT_CLEAR_ICHO	计数器在通道0输入捕获事件时清零

CPTIMERW_CNT_CLEAR_ICH1	计数器在通道1输入捕获事件时清零
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* onfigure CPTIMERW counter clear source */
cptimerw_counter_clear_source_config(CPTIMERW_CNT_CLEAR_ICH0);
```

### 函数 cptimerw\_counter\_read

函数cptimerw\_counter\_read描述见下表：

表 3-201. 函数 cptimerw\_counter\_read

函数名称	cptimerw_counter_read
函数原型	uint32_t cptimerw_counter_read(void);
功能描述	读取CPTIMERW计数器值
先决条件	-
被调用函数	-
输入参数{in}	
-	-
输出参数{out}	
-	-
返回值	
uint32_t	计数器值

例如：

```
/* read CPTIMERW counter value */
uint32_t cnt = cptimerw_counter_read();
```

### 函数 cptimerw\_prescaler\_read

函数cptimerw\_prescaler\_read描述见下表：

表 3-202. 函数 cptimerw\_prescaler\_read

函数名称	cptimerw_prescaler_read
函数原型	uint16_t cptimerw_prescaler_read(void);
功能描述	读取CPTIMERW预分频值
先决条件	-
被调用函数	-
输入参数{in}	
-	-

输出参数{out}	
-	-
返回值	
uint16_t	预分频值(0~65535)

例如:

```
/* read CPTIMERW prescaler value */
uint16_t pre = cptimerw_prescaler_read(void);
```

### 函数 cptimerw\_enable

函数cptimerw\_enable描述见下表:

表 3-203. 函数 cptimerw\_enable

函数名称	cptimerw_enable
函数原型	void cptimerw_enable(void);
功能描述	CPTIMERW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 CPTIMERW */
cptimerw_enable(CPTIMERW);
```

### 函数 cptimerw\_disable

函数cptimerw\_disable描述见下表:

表 3-204. 函数 cptimerw\_disable

函数名称	cptimerw_disable
函数原型	void cptimerw_disable(void);
功能描述	CPTIMERW禁能
先决条件	-
被调用函数	-
输入参数{in}	
-	-
输出参数{out}	
-	-

返回值	
-	-

例如：

```
/* disable CPTIMERW */
cptimerw_disable(CPTIMERW);
```

### 函数 cptimerw\_channel\_output\_mode\_select

函数cptimerw\_channel\_output\_mode\_select描述见下表：

表 3-205. 函数 cptimerw\_channel\_output\_mode\_select

函数名称	cptimerw_channel_output_mode_select
函数原型	void cptimerw_channel_output_mode_select(uint32_t channel, uint32_t mode);
功能描述	选择CPTIMERW通道输出比较模式
先决条件	-
被调用函数	-
输入参数{in}	
channel	输出比较通道
CPTIMERW_OCHx	x=0,1
输入参数{in}	
mode	输出比较模式
CPTIMERW_OC_MOD E_DISABLE	输出禁能
CPTIMERW_OC_MOD E_MANTAIN	输出保持
CPTIMERW_OC_MOD E_LOW_TOGGLE	初始输出低电平，比较匹配时输出电平翻转
CPTIMERW_OC_MOD E_HIGH_TOGGLE	初始输出高电平，比较匹配时输出电平翻转
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* select CPTIMERW channel output compare mode */
cptimerw_channel_output_mode_select(CPTIMERW_OCH0,
CPTIMERW_OC_MODE_LOW_TOGGLE);
```

### 函数 cptimerw\_channel\_output\_compare\_value\_config

函数cptimerw\_channel\_output\_compare\_value\_config描述见下表：

表 3-206. 函数 `cptimerw_channel_output_compare_value_config`

函数名称	<code>cptimerw_channel_output_compare_value_config</code>
函数原型	<code>void cptimerw_channel_output_compare_value_config(uint32_t channel, uint32_t ochcv);</code>
功能描述	配置CPTIMERW通道输出比较值
先决条件	-
被调用函数	-
输入参数{in}	
<b>channel</b>	输出比较通道
CPTIMERW_OCHx	x= 0,1,2,3
输入参数{in}	
<b>ochcv</b>	通道输出比较值，基于位宽设置16或32位值
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* configure the CPTIMERW channel output compare value */
cptimerw_channel_output_compare_value_config(CPTIMERW_OCH0, 99U);
```

### 函数 `cptimerw_channel_output_state_config`

函数`cptimerw_channel_output_state_config`描述见下表：

表 3-207. 函数 `cptimerw_channel_output_state_config`

函数名称	<code>cptimerw_channel_output_state_config</code>
函数原型	<code>void cptimerw_channel_output_state_config(uint32_t channel, uint32_t state);</code>
功能描述	配置CPTIMERW输出比较通道使能状态
先决条件	-
被调用函数	-
输入参数{in}	
<b>channel</b>	输出比较通道
CPTIMERW_OCHx	x= 0,1,2,3
输入参数{in}	
<b>state</b>	使能状态
CPTIMERW_OCX_DISABLE	输出比较通道禁止
CPTIMERW_OCX_ENABLE	输出比较通道使能
输出参数{out}	
-	-

返回值	
-	-

例如：

```
/* configure CPTIMERW output compare channel enable state */
cptimerw_channel_output_state_config(CPTIMERW_OCH0, CPTIMERW_OCX_ENABLE);
```

### 函数 `cptimerw_channel_input_struct_para_init`

函数 `cptimerw_channel_input_struct_para_init` 描述见下表：

表 3-208. 函数 `cptimerw_channel_input_struct_para_init`

函数名称	<code>cptimerw_channel_input_struct_para_init</code>
函数原型	void <code>cptimerw_channel_input_struct_para_init(cptimerw_ic_parameter_struct *icpara)</code>
功能描述	初始化 CPTIMERW 通道输入捕获参数结构体为默认值
先决条件	-
被调用函数	-
输入参数{in}	
icpara	参见 <a href="#">表 3-193. 结构体 <code>cptimerw_ic_parameter_struct</code></a>
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* init parameter struct with a default value */
cptimerw_ic_parameter_struct para;
cptimerw_channel_input_struct_para_init(&para);
```

### 函数 `cptimerw_input_capture_config`

函数 `cptimerw_input_capture_config` 描述见下表：

表 3-209. 函数 `cptimerw_input_capture_config`

函数名称	<code>cptimerw_input_capture_config</code>
函数原型	void <code>cptimerw_input_capture_config(uint16_t channel, cptimerw_ic_parameter_struct *icpara)</code>
功能描述	配置 CPTIMERW 输入捕获参数
先决条件	-
被调用函数	-

输入参数{in}	
<b>channel</b>	输入捕获通道
CPTIMERW_ICHx	x= 0,1
输入参数{in}	
<b>initpara</b>	参见 <a href="#">表 3-193. 结构体 <code>cptimerw_ic_parameter_struct</code></a>
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* configure CPTIMERW input capture parameter */
cptimerw_ic_parameter_struct para;
para.icedge = CPTIMERW_IC_RISING_EDGE;
para.icfilter = 5U;
cptimerw_input_capture_config(CPTIMERW_ICH0, &para);
```

### 函数 `cptimerw_channel_capture_value_register_read`

函数 `cptimerw_channel_capture_value_register_read` 描述见下表：

**表 3-210. 函数 `cptimerw_channel_capture_value_register_read`**

<b>函数名称</b>	<code>cptimerw_channel_capture_value_register_read</code>
<b>函数原型</b>	<code>uint32_t cptimerw_channel_capture_value_register_read(uint16_t channel);</code>
<b>功能描述</b>	读取CPTIMERW通道输出捕获值
<b>先决条件</b>	-
<b>被调用函数</b>	-
输入参数{in}	
<b>channel</b>	输入捕获通道
CPTIMERW_ICHx	x= 0,1
输出参数{out}	
-	-
返回值	
<b>uint32_t</b>	捕获值

例如：

```
/* read CPTIMERW channel input capture value */
uint32_t val = cptimerw_channel_capture_value_register_read(CPTIMERW_ICH0);
```

### 函数 `cptimerw_event_software_generate`

函数 `cptimerw_event_software_generate` 描述见下表：

表 3-211. 函数 `cptimerw_event_software_generate`

函数名称	<code>cptimerw_event_software_generate</code>
函数原型	<code>void cptimerw_event_software_generate(uint32_t event);</code>
功能描述	软件产生事件
先决条件	-
被调用函数	-
输入参数{in}	
<b>event</b>	定时器软件事件源
<code>CPTIMERW_EVENT_SRC_UPG</code>	计数器更新时间产生
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* generate counter update event */
cptimerw_event_software_generate (CPTIMERW_EVENT_SRC_UPG);
```

### 函数 `cptimerw_flag_get`

函数`cptimerw_flag_get`描述见下表:

表 3-212. 函数 `cptimerw_flag_get`

函数名称	<code>cptimerw_flag_get</code>
函数原型	<code>FlagStatus cptimerw_flag_get(uint32_t flag);</code>
功能描述	获取CPTIMERW标志位
先决条件	-
被调用函数	-
输入参数{in}	
<b>flag</b>	CPTIMERW标志位
<code>CPTIMERW_FLAG_UP</code>	CPTIMERW更新事件标志位
<code>CPTIMERW_FLAG_IC_H0</code>	CPTIMERW通道0输入捕获标志位
<code>CPTIMERW_FLAG_IC_H1</code>	CPTIMERW通道1输入捕获标志位
<code>CPTIMERW_FLAG_OC_H0</code>	CPTIMERW通道0输出比较标志位
<code>CPTIMERW_FLAG_OC_H1</code>	CPTIMERW通道1输出比较标志位
<code>CPTIMERW_FLAG_OC_H2</code>	CPTIMERW通道2输出比较标志位
<code>CPTIMERW_FLAG_OC_H3</code>	CPTIMERW通道3输出比较标志位



H3	
输出参数{out}	
-	-
返回值	
FlagStatus	SET or RESET

例如：

```
/* get the CPTIMERW flag */
FlagStatus flag_value;
flag_value = cptimerw_flag_get(CPTIMERW_FLAG_OCH0);
```

### 函数 cptimerw\_flag\_clear

函数cptimerw\_flag\_clear描述见下表：

表 3-213. 函数 cptimerw\_flag\_clear

函数名称	cptimerw_flag_clear
函数原型	void cptimerw_flag_clear(uint32_t flag);
功能描述	清除CPTIMERW标志位
先决条件	-
被调用函数	-
输入参数{in}	
flag	CPTIMERW标志位
CPTIMERW_FLAG_UP	CPTIMERW更新事件标志位
CPTIMERW_FLAG_IC H0	CPTIMERW通道0输入捕获标志位
CPTIMERW_FLAG_IC H1	CPTIMERW通道1输入捕获标志位
CPTIMERW_FLAG_OC H0	CPTIMERW通道0输出比较标志位
CPTIMERW_FLAG_OC H1	CPTIMERW通道1输出比较标志位
CPTIMERW_FLAG_OC H2	CPTIMERW通道2输出比较标志位
CPTIMERW_FLAG_OC H3	CPTIMERW通道3输出比较标志位
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* clear the CPTIMERW flag */
```

```
cptimerw_flag_clear(CPTIMERW_FLAG_UP);
```

### 函数 `cptimerw_interrupt_enable`

函数 `cptimerw_interrupt_enable` 描述见下表:

表 3-214. 函数 `cptimerw_interrupt_enable`

函数名称	<code>cptimerw_interrupt_enable</code>
函数原型	<code>void cptimerw_interrupt_enable(uint32_t interrupt);</code>
功能描述	使能CPTIMERW中断
先决条件	-
被调用函数	-
输入参数{in}	
<b>interrupt</b>	CPTIMERW中断
<code>CPTIMERW_INT_UP</code>	CPTIMERW更新中断
<code>CPTIMERW_INT_ICH0</code>	CPTIMERW通道0输入捕获中断
<code>CPTIMERW_INT_ICH1</code>	CPTIMERW通道1输入捕获中断
<code>CPTIMERW_INT_OCH0</code>	CPTIMERW通道0输出比较中断
<code>CPTIMERW_INT_OCH1</code>	CPTIMERW通道1输出比较中断
<code>CPTIMERW_INT_OCH2</code>	CPTIMERW通道2输出比较中断
<code>CPTIMERW_INT_OCH3</code>	CPTIMERW通道3输出比较中断
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* enable the CPTIMERW interrupt */
cptimerw_interrupt_enable(CPTIMERW_INT_UP);
```

### 函数 `cptimerw_interrupt_disable`

函数 `cptimerw_interrupt_disable` 描述见下表:

表 3-215. 函数 `cptimerw_interrupt_disable`

函数名称	<code>cptimerw_interrupt_disable</code>
函数原型	<code>void cptimerw_interrupt_disable(uint32_t interrupt);</code>
功能描述	禁能CPTIMERW中断

先决条件	-
被调用函数	-
输入参数{in}	
interrupt	CPTIMERW中断
CPTIMERW_INT_UP	CPTIMERW更新中断
CPTIMERW_INT_ICH0	CPTIMERW通道0输入捕获中断
CPTIMERW_INT_ICH1	CPTIMERW通道1输入捕获中断
CPTIMERW_INT_OCH0	CPTIMERW通道0输出比较中断
CPTIMERW_INT_OCH1	CPTIMERW通道1输出比较中断
CPTIMERW_INT_OCH2	CPTIMERW通道2输出比较中断
CPTIMERW_INT_OCH3	CPTIMERW通道3输出比较中断
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* disable the CPTIMERW interrupt */
cptimerw_interrupt_disable(CPTIMERW_INT_UP);
```

### 函数 cptimerw\_interrupt\_flag\_get

函数cptimerw\_interrupt\_flag\_get描述见下表：

表 3-216. 函数 cptimerw\_interrupt\_flag\_get

函数名称	cptimerw_interrupt_flag_get
函数原型	FlagStatus cptimerw_interrupt_flag_get(uint32_t flag);
功能描述	获取CPTIMERW中断标志位
先决条件	-
被调用函数	-
输入参数{in}	
int_flag	CPTIMERW中断标志位
CPTIMERW_INT_FLAG_UP	CPTIMERW更新中断标志位
CPTIMERW_INT_FLAG_ICH0	CPTIMERW通道0输入捕获中断标志位
CPTIMERW_INT_FLAG_ICH1	CPTIMERW通道1输入捕获中断标志位

<i>G_ICH1</i>	
<i>CPTIMERW_INT_FLAG_OCH0</i>	CPTIMERW通道0输出比较中断标志位
<i>CPTIMERW_INT_FLAG_OCH1</i>	CPTIMERW通道1输出比较中断标志位
<i>CPTIMERW_INT_FLAG_OCH2</i>	CPTIMERW通道2输出比较中断标志位
<i>CPTIMERW_INT_FLAG_OCH3</i>	CPTIMERW通道3输出比较中断标志位
输出参数{out}	
-	-
返回值	
FlagStatus	SET or RESET

例如：

```
/* get the CPTIMERW interrupt bit */
FlagStatus flag_value;
flag_value = cptimerw_interrupt_flag_get(CPTIMERW_INT_FLAG_UP);
```

### 函数 `cptimerw_interrupt_flag_clear`

函数 `cptimerw_interrupt_flag_clear` 描述见下表：

表 3-217. 函数 `cptimerw_interrupt_flag_clear`

函数名称	<code>cptimerw_interrupt_flag_clear</code>
函数原型	<code>void cptimerw_interrupt_flag_clear(uint32_t flag);</code>
功能描述	清除CPTIMERW中断标志位
先决条件	-
被调用函数	-
输入参数{in}	
int_flag	CPTIMERW中断标志位
<i>CPTIMERW_INT_FLAG_UP</i>	CPTIMERW更新中断标志位
<i>CPTIMERW_INT_FLAG_ICH0</i>	CPTIMERW通道0输入捕获中断标志位
<i>CPTIMERW_INT_FLAG_ICH1</i>	CPTIMERW通道1输入捕获中断标志位
<i>CPTIMERW_INT_FLAG_OCH0</i>	CPTIMERW通道0输出比较中断标志位
<i>CPTIMERW_INT_FLAG_OCH1</i>	CPTIMERW通道1输出比较中断标志位
<i>CPTIMERW_INT_FLAG_OCH2</i>	CPTIMERW通道2输出比较中断标志位

CPTIMERW_INT_FLA G_OCH3	CPTIMERW通道3输出比较中断标志位
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* clear the CPTIMERW interrupt bit */
cptimerw_interrupt_flag_clear(CPTIMERW_INT_FLAG_UP);
```

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

### 3.8.1. 外设寄存器说明

CRC寄存器列表如下表所示：

表 3-218. CRC 寄存器

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

### 3.8.2. 外设库函数说明

CRC库函数列表如下表所示：

表 3-219. CRC 库函数

库函数名称	库函数描述
crc_deinit	复位CRC计算单元
crc_reverse_output_data_enable	使能输出数据翻转功能
crc_reverse_output_data_disable	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（0xFFFFFFFF）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

库函数名称	库函数描述
crc_polynomial_size_set	配置多项式长度
crc_polynomial_set	设置多项式寄存器数据
crc_single_data_calculate	CRC计算一个数据
crc_block_data_calculate	CRC计算一个数组

### 函数 crc\_deinit

函数crc\_deinit描述见下表：

表 3-220. 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\_data\_register\_reset

函数crc\_data\_register\_reset描述见下表：

表 3-221. 函数 crc\_data\_register\_reset

函数名称	crc_data_register_reset
函数原形	void crc_data_register_reset(void);
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_reverse_output_data_enable`

函数 `crc_reverse_output_data_enable` 描述见下表:

**表 3-222. 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 3-223. 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_input_data_reverse_config`

函数 `crc_input_data_reverse_config` 描述见下表：

**表 3-224. 函数 `crc_input_data_reverse_config`**

函数名称	<code>crc_input_data_reverse_config</code>
函数原形	<code>void crc_input_data_reverse_config(uint32_t data_reverse);</code>
功能描述	配置输入数据翻转功能
先决条件	-
被调用函数	-
输入参数{in}	
<code>data_reverse</code>	设定的输入数据翻转功能
<code>CRC_INPUT_DATA_NOT</code>	输入数据不翻转
<code>CRC_INPUT_DATA_BYTE</code>	输入数据按字节翻转
<code>CRC_INPUT_DATA_HALFWORD</code>	输入数据按半字翻转
<code>CRC_INPUT_DATA_WORD</code>	输入数据按字翻转
输出参数{out}	
-	-
返回值	
-	-

例如：

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

```
crc_input_data_reverse_config (CRC_INPUT_DATA_WORD);
```

### 函数 `crc_data_register_read`

函数 `crc_data_register_read` 描述见下表：

**表 3-225. 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 3-226. 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 3-227. 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 3-228. 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\_polynomial\_size\_set

函数crc\_polynomial\_size\_set描述见下表：

表 3-229. 函数 crc\_polynomial\_size\_set

函数名称	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}	
<b>poly_size</b>	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 3-230. 函数 crc\_polynomial\_set**

函数名称	crc_polynomial_set
函数原形	void crc_polynomial_set(uint32_t poly);
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}	
<b>poly</b>	设置多项式长度寄存器值
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* configure the CRC polynomial value */
crc_polynomial_set (0x11223344);
```

### 函数 **crc\_single\_data\_calculate**

函数crc\_single\_data\_calculate描述见下表：

**表 3-231. 函数 crc\_single\_data\_calculate**

函数名称	crc_single_data_calculate
函数原形	uint32_t crc_single_data_calculate(uint32_t sdata, uint8_t data_format);

功能描述	CRC计算一个数据
先决条件	-
被调用函数	-
输入参数{in}	
sdata	特定的一个输入数据
输入参数{in}	
data_format	输入数据格式
INPUT_FORMAT_WORD	输入数据格式为字
INPUT_FORMAT_HALFWORD	输入数据格式为半字
INPUT_FORMAT_BYTE	输入数据格式为字节
输出参数{out}	
-	-
返回值	
uint32_t	32位CRC计算结果 (0-0xFFFFFFFF)

例如：

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

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

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

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

### 函数 crc\_block\_data\_calculate

函数crc\_block\_data\_calculate描述见下表：

表 3-232. 函数 crc\_block\_data\_calculate

函数名称	crc_block_data_calculate
函数原形	uint32_t crc_block_data_calculate(void *array, uint32_t size, uint8_t data_format);
功能描述	CRC计算一个数组
先决条件	-
被调用函数	-
输入参数{in}	
array	输入数组的指针
输入参数{in}	
size	数据长度
输入参数{in}	
data_format	输入数据格式

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

例如：

```

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

#define BUFFER_SIZE    6

uint32_t valcrc = 0;

static const uint32_t data_buffer[BUFFER_SIZE] = {
    0x00001111, 0x00002222, 0x00003333, 0x00004444, 0x00005555, 0x00006666};

valcrc = crc_block_data_calculate(data_buffer, BUFFER_SIZE, INPUT_FORMAT_WORD);

```

### 3.9. 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 3-233. 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	DAC_OUT0 12位右对齐数据保持寄存器
DAC_OUT0_L12DH	DAC_OUT0 12位左对齐数据保持寄存器
DAC_OUT1_R12DH	DAC_OUT1 12位右对齐数据保持寄存器
DAC_OUT1_L12DH	DAC_OUT1 12位左对齐数据保持寄存器
DACC_R12DH	DAC并发模式12位右对齐数据保持寄存器
DACC_L12DH	DAC并发模式12位左对齐数据保持寄存器
DAC_OUT0_DO	DAC_OUT0数据输出寄存器
DAC_OUT1_DO	DAC_OUT1数据输出寄存器

#### 3.9.2. 外设库函数说明

DAC库函数列表如下表所示：

**表 3-234. DAC 库函数**

库函数名称	库函数描述
dac_deinit	DAC外设复位
dac_enable	DAC使能
dac_disable	DAC禁能
dac_sync_enable	DAC同步使能
dac_sync_disable	DAC同步禁能
dac_connect_to_pin_enable	DAC连接到引脚使能
dac_connect_to_pin_disable	DAC连接到引脚禁能
dac_connect_to_cmp_enable	DAC连接到比较器使能
dac_connect_to_cmp_disable	DAC连接到比较器禁能
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

库函数名称	库函数描述
<code>dac_software_trigger_enable</code>	DAC软件触发使能
<code>dac_wave_mode_config</code>	DAC噪声波模式选择
<code>dac_lfsr_noise_config</code>	DAC LFSR模式设置
<code>dac_triangle_noise_config</code>	DAC三角波模式设置
<code>dac_concurrent_enable</code>	并发DAC模式使能
<code>dac_concurrent_disable</code>	并发DAC模式禁能
<code>dac_concurrent_software_trigger_enable</code>	并发DAC模式软件触发使能
<code>dac_concurrent_data_set</code>	并发DAC模式输出数据设置

### 函数 `dac_deinit`

函数`dac_deinit`描述见下表：

**表 3-235. 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 3-236. 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}	
<b>dac_periph</b>	DAC外设
<i>DACx</i>	DAC外设选择 (x = 0)
输入参数{in}	
<b>dac_out</b>	DAC输出
<i>DAC_OUTx</i>	DAC输出通道选择 (x = 0,1)
输出参数{out}	
-	-
返回值	
-	-

例如:

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

### 函数 **dac\_disable**

函数dac\_disable描述见下表:

**表 3-237. 函数 **dac\_disable****

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

例如:

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



## 函数 dac\_sync\_enable

函数dac\_sync\_enable描述见下表:

表 3-238. 函数 dac\_sync\_enable

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

例如:

```
/* enable DAC0_OUT0 sync function */
dac_sync_enable (DAC0, DAC_OUT0);
```

## 函数 dac\_sync\_disable

函数dac\_sync\_disable描述见下表:

表 3-239. 函数 dac\_sync\_disable

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

返回值	
-	-

例如：

```
/* disable DAC0_OUT0 sync function */
```

```
dac_sync_disable (DAC0, DAC_OUT0);
```

### 函数 `dac_connect_to_pin_enable`

函数 `dac_connect_to_pin_enable` 描述见下表：

**表 3-240. 函数 `dac_connect_to_pin_enable`**

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

例如：

```
/* enable DAC0_OUT0 connect to pin */
```

```
dac_connect_to_pin_enable (DAC0, DAC_OUT0);
```

### 函数 `dac_connect_to_pin_disable`

函数 `dac_connect_to_pin_disable` 描述见下表：

**表 3-241. 函数 `dac_connect_to_pin_disable`**

函数名称	<code>dac_connect_to_pin_disable</code>
函数原型	<code>void dac_connect_to_pin_disable (uint32_t dac_periph, uint8_t dac_out);</code>
功能描述	DAC连接到引脚禁能
先决条件	-
被调用函数	-
输入参数{in}	

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

例如:

```
/* disable DAC0_OUT0 connect to pin */
dac_connect_to_pin_disable (DAC0, DAC_OUT0);
```

### 函数 **dac\_connect\_to\_cmp\_enable**

函数dac\_connect\_to\_cmp\_enable描述见下表:

**表 3-242. 函数 **dac\_connect\_to\_cmp\_enable****

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

例如:

```
/* enable DAC0_OUT0 connect to CMP */
dac_connect_to_cmp_enable (DAC0, DAC_OUT0);
```

### 函数 **dac\_connect\_to\_cmp\_disable**

函数dac\_connect\_to\_cmp\_disable描述见下表:

表 3-243. 函数 `dac_connect_to_cmp_disable`

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

例如：

```
/* disable DAC0_OUT0 connect to CMP */
```

```
dac_connect_to_cmp_disable (DAC0, DAC_OUT0);
```

### 函数 `dac_output_value_get`

函数`dac_output_value_get`描述见下表：

表 3-244. 函数 `dac_output_value_get`

函数名称	<code>dac_output_value_get</code>
函数原型	<code>uint16_t dac_output_value_get(uint32_t dac_periph, uint8_t dac_out);</code>
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}	
<code>dac_periph</code>	DAC外设
<code>DACx</code>	DAC外设选择 ( $x = 0$ )
输入参数{in}	
<code>dac_out</code>	DAC输出
<code>DAC_OUTx</code>	DAC输出通道选择 ( $x = 0, 1$ )
输出参数{out}	
-	-
返回值	
<code>uint16_t</code>	外设DACx数据保持寄存器值 (0~4095)

例如：

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

```
Uint16_t data;
```

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

### 函数 **dac\_data\_set**

函数dac\_data\_set描述见下表：

**表 3-245. 函数 dac\_data\_set**

函数名称	dac_data_set
函数原型	void dac_data_set(uint32_t dac_periph, uint8_t dac_out, uint32_t dac_align, uint16_t data);
功能描述	DAC输出数据设置
先决条件	-
被调用函数	-
输入参数{in}	
dac_periph	DAC外设
DACx	DAC外设选择 (x = 0)
输入参数{in}	
dac_out	DAC输出
DAC_OUTx	DAC输出通道选择 (x = 0,1)
输入参数{in}	
dac_align	DAC对齐模式
DAC_ALIGN_12B_R	12位数据右对齐
DAC_ALIGN_12B_L	12位数据左对齐
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_12B_R, 0xFFFF);
```

### 函数 **dac\_trigger\_enable**

函数dac\_trigger\_enable描述见下表：

表 3-246. 函数 `dac_trigger_enable`

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

例如：

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

### 函数 `dac_trigger_disable`

函数`dac_trigger_disable`描述见下表：

表 3-247. 函数 `dac_trigger_disable`

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

例如：

```
/* disable DAC0_OUT0 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 3-248. 函数 `dac_trigger_source_config`**

函数名称	<code>dac_trigger_source_config</code>
函数原型	<code>void dac_trigger_source_config(uint32_t dac_periph, uint8_t dac_out, uint32_t triggersource);</code>
功能描述	DAC触发源配置
先决条件	-
被调用函数	-
输入参数{in}	
<code>dac_periph</code>	DAC外设
<code>DACx</code>	DAC外设选择（ <code>x = 0</code> ）
输入参数{in}	
<code>dac_out</code>	DAC输出
<code>DAC_OUTx</code>	DAC输出通道选择（ <code>x = 0,1</code> ）
输入参数{in}	
<code>triggersource</code>	DAC触发源
<code>DAC_TRIGGER_EXTERNAL</code>	来自于EVIC的外部触发
<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_EXTERNAL);
```

### 函数 `dac_software_trigger_enable`

函数 `dac_software_trigger_enable` 描述见下表：

**表 3-249. 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}	
<b>dac_periph</b>	DAC外设
<i>DACx</i>	DAC外设选择 (x = 0)
输入参数{in}	
<b>dac_out</b>	DAC输出
<i>DAC_OUTx</i>	DAC输出通道选择 (x = 0,1)
输出参数{out}	
-	-
返回值	
-	-

例如:

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

### 函数 **dac\_wave\_mode\_config**

函数dac\_wave\_mode\_config描述见下表:

表 3-250. 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}	
<b>dac_periph</b>	DAC外设
<i>DACx</i>	DAC外设选择 (x = 0)
输入参数{in}	
<b>dac_out</b>	DAC输出
<i>DAC_OUTx</i>	DAC输出通道选择 (x = 0,1)
输入参数{in}	
<b>wave_mode</b>	噪声波模式选择
<i>DAC_WAVE_DISABLE</i>	噪声波模式禁能
<i>DAC_WAVE_MODE_LFSR</i>	LFSR噪声波模式
<i>DAC_WAVE_MODE_TRIANGLE</i>	三角波噪声波模式



输出参数{out}	
-	-
返回值	
-	-

例如：

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

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

### 函数 `dac_lfsr_noise_config`

函数 `dac_lfsr_noise_config` 描述见下表：

表 3-251. 函数 `dac_lfsr_noise_config`

函数名称	<code>dac_lfsr_noise_config</code>
函数原型	<code>void dac_lfsr_noise_config(uint32_t dac_periph, uint8_t dac_out, uint32_t unmask_bits);</code>
功能描述	DAC LFSR 模式配置
先决条件	-
被调用函数	-
输入参数{in}	
<b>dac_periph</b>	DAC 外设
<i>DACx</i>	DAC 外设选择 ( $x = 0$ )
输入参数{in}	
<b>dac_out</b>	DAC 输出
<i>DAC_OUTx</i>	DAC 输出通道选择 ( $x = 0, 1$ )
输入参数{in}	
<b>unmask_bits</b>	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 ( $x = 1, 2, 3 \dots 11$ )
输出参数{out}	
-	-
返回值	
-	-

例如：

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

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

**函数 dac\_triangle\_noise\_config**

函数dac\_triangle\_noise\_config描述见下表:

**表 3-252. 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
DACx	DAC外设选择 (x = 0)
输入参数{in}	
dac_out	DAC输出
DAC_OUTx	DAC输出通道选择 (x = 0,1)
输入参数{in}	
amplitude	三角波幅值
DAC_TRIANGLE_AMPLITUDE_x	$x = 2^n - 1$ (n = 1..12)
输出参数{out}	
-	-
返回值	
-	-

例如:

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

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

**函数 dac\_concurrent\_enable**

函数dac\_concurrent\_enable描述见下表:

**表 3-253. 函数 dac\_concurrent\_enable**

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

输出参数{out}	
-	-
返回值	
-	-

例如:

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

```
dac_concurrent_enable(DAC0);
```

### 函数 **dac\_concurrent\_disable**

函数dac\_concurrent\_disable描述见下表:

**表 3-254. 函数 dac\_concurrent\_disable**

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

例如:

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

```
dac_concurrent_disable(DAC0);
```

### 函数 **dac\_concurrent\_software\_trigger\_enable**

函数dac\_concurrent\_software\_trigger\_enable描述见下表:

**表 3-255. 函数 dac\_concurrent\_software\_trigger\_enable**

函数名称	dac_concurrent_software_trigger_enable
函数原型	void dac_concurrent_software_trigger_enable(uint32_t dac_periph);
功能描述	并发DAC模式软件触发使能
先决条件	-
被调用函数	-
输入参数{in}	
<b>dac_periph</b>	DAC外设

DACx	DAC外设选择 (x = 0)
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* enable DAC0 concurrent software trigger */
dac_concurrent_software_trigger_enable(DAC0);
```

### 函数 dac\_concurrent\_data\_set

函数dac\_concurrent\_data\_set描述见下表:

表 3-256. 函数 dac\_concurrent\_data\_set

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

例如:

```
/* set DAC0 concurrent mode data holding register value */
dac_concurrent_data_set(DAC0, DAC_ALIGN_12B_R, 0xFFFF, 0xFFFF);
```

## 3.10. DBG

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

### 3.10.1. 外设寄存器说明

DBG寄存器列表如下表所示：

表 3-257. DBG 寄存器

寄存器名称	寄存器描述
DBG_ID	DBG ID寄存器
DBG_CTL	DBG控制寄存器

### 3.10.2. 外设库函数说明

DBG库函数列表如下表所示：

表 3-258. DBG 库函数

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

#### 枚举类型 dbg\_periph\_enum

表 3-259. 枚举类型 dbg\_periph\_enum

成员名称	功能描述
DBG_FWDGT_HOLD	当内核停止时，保持FWDGT计数器时钟

DBG_WWDGT_HOLD	当内核停止时，保持WWDGT计数器时钟
DBG_TIMER0_HOLD	当内核停止时，保持TIMER0计数器计数值不变
DBG_TIMER1_HOLD	当内核停止时，保持TIMER1计数器计数值不变
DBG_TIMER2_HOLD	当内核停止时，保持TIMER2计数器计数值不变
DBG_GPTIMER0_HOLD	当内核停止时，保持GPTIMER0计数器计数值不变
DBG_CAN_HOLD	当内核停止时，CAN接收寄存器停止接收数据
DBG_I2C_HOLD	当内核停止时，保持I2C 的SMBUS状态不变，用于调试
DBG_GPTIMER1_HOLD	当内核停止时，保持GPTIMER1计数器计数值不变
DBG_CPTIMER0_HOLD	当内核停止时，保持CPTIMER0计数器计数值不变
DBG_CPTIMERW_HOLD	当内核停止时，保持CPTIMERW计数器计数值不变

### 函数 dbg\_deinit

函数dbg\_deinit描述见下表：

表 3-260. 函数 dbg\_deinit

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

例如：

```
/* deinit DBG*/
```

```
dbg_deinit();
```

## 函数 dbg\_id\_get

函数dbg\_id\_get描述见下表:

表 3-261. 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	Read DBG_ID code register
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 3-262. 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

<i>DBG_LOW_POWER_SLEEP</i>	在睡眠模式下，保持调试器连接，可进行调试
<i>DBG_LOW_POWER_DEEPSLEEP</i>	在深度睡眠模式下，保持调试器连接，可进行调试
<i>DBG_LOW_POWER_STANDBY</i>	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 3-263. 函数 **dbg\_low\_power\_disable****

函数名称	dbg_low_power_disable
函数原形	void dbg_low_power_disable(uint32_t dbg_low_power);
功能描述	禁能低功耗模式的MCU调试保持功能
先决条件	-
被调用函数	-
输入参数{in}	
<b>dbg_low_power</b>	低功耗模式调试保持
<i>DBG_LOW_POWER_SLEEP</i>	在睡眠模式下，保持调试器连接，可进行调试
<i>DBG_LOW_POWER_DEEPSLEEP</i>	在深度睡眠模式下，保持调试器连接，可进行调试
<i>DBG_LOW_POWER_STANDBY</i>	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 3-264. 函数 dbg\_periph\_enable

函数名称	dbg_periph_enable
函数原形	void dbg_periph_enable(dbg_periph_enum dbg_periph);
功能描述	使能外设的MCU调试保持功能
先决条件	-
被调用函数	-
输入参数{in}	
dbg_periph	Peripheral refer to <a href="#">表 3-259. 枚举类型dbg_periph_enum</a>
输出参数{out}	
-	-
返回值	
-	-

例如:

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

```
dbg_periph_enable(DBG_TIMER0_HOLD);
```

### 函数 dbg\_periph\_disable

函数dbg\_periph\_disable描述见下表:

表 3-265. 函数 dbg\_periph\_disable

函数名称	dbg_periph_disable
------	--------------------

函数原形	void dbg_periph_disable(dbg_periph_enum dbg_periph);
功能描述	禁能外设的MCU调试保持功能
先决条件	-
被调用函数	-
输入参数{in}	
dbg_periph	Peripheral refer to <a href="#">表 3-259. 枚举类型dbg_periph_enum</a>
输出参数{out}	
-	-
返回值	
-	-

例如：

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

```
dbg_periph_disable(DBG_TIMER0_HOLD);
```

### 函数 dbg\_trace\_pin\_enable

函数dbg\_trace\_pin\_enable描述见下表：

表 3-266. 函数 dbg\_trace\_pin\_enable

函数名称	dbg_trace_pin_enable
函数原形	void dbg_trace_pin_enable(void);
功能描述	使能跟踪引脚分配
先决条件	-
被调用函数	-
输入参数{in}	
-	-
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* enable trace pin assignment */
```

```
dbg_trace_pin_enable();
```

### 函数 dbg\_trace\_pin\_disable

函数dbg\_trace\_pin\_disable描述见下表：

表 3-267. 函数 dbg\_trace\_pin\_disable

函数名称	dbg_trace_pin_disable
函数原形	void dbg_trace_pin_disable(void);
功能描述	禁能跟踪引脚分配
先决条件	-
被调用函数	-
输入参数{in}	
-	-
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* disable trace pin assignment */
```

```
dbg_trace_pin_disable();
```

## 3.11. DMA&DMAMUX

DMA控制器提供了一种硬件的方式在外设和存储器之间或者存储器和存储器之间传输数据，而无需CPU的介入，从而使CPU可以专注在处理其他系统功能上。章节[3.11.1](#)描述了DMA的寄存器列表，章节[3.11.2](#)对DMA库函数进行说明。

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

### 3.11.1. 外设寄存器说明

DMA寄存器列表如下表所示：

**表 3-268. 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..5)	通道x控制寄存器
DMA_CHxCNT (x=0..5)	通道x计数寄存器
DMA_CHxPADDR (x=0..5)	通道x外设基地址寄存器
DMA_CHxMADDR (x=0..5)	通道x存储器基地址寄存器

DMAMUX寄存器列表如下表所示：

**表 3-269. DMAMUX 寄存器**

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

### 3.11.2. 外设库函数说明

DMA库函数列表如下表所示：

**表 3-270. DMA 库函数**

库函数名称	库函数描述
dma_deinit	复位外设DMA通道x的所有寄存器
dma_struct_para_init	将DMA结构体中所有参数初始化为默认值
dma_init	初始化外设DMA的通道x
dma_circulation_enable	使能DMA循环模式

库函数名称	库函数描述
<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 3-271. 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
<code>dmamux_synchronization_polarity_config</code>	配置DMAMUX同步输入的有效边沿

库函数名称	库函数描述
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 3-272. 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 3-273. 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 3-274. 结构体 dmamux\_gen\_parameter\_struct

成员名称	功能描述
------	------

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

### 枚举 dmamux\_interrupt\_enum

表 3-275. 枚举 dmamux\_interrupt\_enum

成员名称	功能描述
DMAMUX_INT_MU XCH0_SO	DMAMUX请求路由通道0同步溢出中断
DMAMUX_INT_MU XCH1_SO	DMAMUX请求路由通道1同步溢出中断
DMAMUX_INT_MU XCH2_SO	DMAMUX请求路由通道2同步溢出中断
DMAMUX_INT_MU XCH3_SO	DMAMUX请求路由通道3同步溢出中断
DMAMUX_INT_MU XCH4_SO	DMAMUX请求路由通道4同步溢出中断
DMAMUX_INT_MU XCH5_SO	DMAMUX请求路由通道5同步溢出中断
DMAMUX_INT_MU XCH6_SO	DMAMUX请求路由通道6同步溢出中断
DMAMUX_INT_MU XCH7_SO	DMAMUX请求路由通道7同步溢出中断
DMAMUX_INT_MU XCH8_SO	DMAMUX请求路由通道8同步溢出中断
DMAMUX_INT_MU XCH9_SO	DMAMUX请求路由通道9同步溢出中断
DMAMUX_INT_MU XCH10_SO	DMAMUX请求路由通道10同步溢出中断
DMAMUX_INT_MU XCH11_SO	DMAMUX请求路由通道11同步溢出中断
DMAMUX_INT_GE NCH0_TO	DMAMUX请求生成通道0触发溢出中断
DMAMUX_INT_GE NCH1_TO	DMAMUX请求生成通道1触发溢出中断
DMAMUX_INT_GE NCH2_TO	DMAMUX请求生成通道2触发溢出中断
DMAMUX_INT_GE NCH3_TO	DMAMUX请求生成通道3触发溢出中断

## 枚举 dmamux\_flag\_enum

表 3-276. 枚举 dmamux\_flag\_enum

成员名称	功能描述
DMAMUX_FLAG_M UXCH0_SO	DMAMUX请求路由通道0同步溢出标志
DMAMUX_FLAG_M UXCH1_SO	DMAMUX请求路由通道1同步溢出标志
DMAMUX_FLAG_M UXCH2_SO	DMAMUX请求路由通道2同步溢出标志
DMAMUX_FLAG_M UXCH3_SO	DMAMUX请求路由通道3同步溢出标志
DMAMUX_FLAG_M UXCH4_SO	DMAMUX请求路由通道4同步溢出标志
DMAMUX_FLAG_M UXCH5_SO	DMAMUX请求路由通道5同步溢出标志
DMAMUX_FLAG_M UXCH6_SO	DMAMUX请求路由通道6同步溢出标志
DMAMUX_FLAG_M UXCH7_SO	DMAMUX请求路由通道7同步溢出标志
DMAMUX_FLAG_M UXCH8_SO	DMAMUX请求路由通道8同步溢出标志
DMAMUX_FLAG_M UXCH9_SO	DMAMUX请求路由通道9同步溢出标志
DMAMUX_FLAG_M UXCH10_SO	DMAMUX请求路由通道10同步溢出标志
DMAMUX_FLAG_M UXCH11_SO	DMAMUX请求路由通道11同步溢出标志
DMAMUX_FLAG_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 3-277. 枚举 dmamux\_interrupt\_flag\_enum

成员名称	功能描述
DMAMUX_INT_FLG MUXCH0_SO	DMAMUX请求路由通道0同步溢出中断标志



DMAMUX_INT_FLAG_MUXCH1_SO	DMAMUX请求路由通道1同步溢出中断标志
DMAMUX_INT_FLAG_MUXCH2_SO	DMAMUX请求路由通道2同步溢出中断标志
DMAMUX_INT_FLAG_MUXCH3_SO	DMAMUX请求路由通道3同步溢出中断标志
DMAMUX_INT_FLAG_MUXCH4_SO	DMAMUX请求路由通道4同步溢出中断标志
DMAMUX_INT_FLAG_MUXCH5_SO	DMAMUX请求路由通道5同步溢出中断标志
DMAMUX_INT_FLAG_MUXCH6_SO	DMAMUX请求路由通道6同步溢出中断标志
DMAMUX_INT_FLAG_MUXCH7_SO	DMAMUX请求路由通道7同步溢出中断标志
DMAMUX_INT_FLAG_MUXCH8_SO	DMAMUX请求路由通道8同步溢出中断标志
DMAMUX_INT_FLAG_MUXCH9_SO	DMAMUX请求路由通道9同步溢出中断标志
DMAMUX_INT_FLAG_MUXCH10_SO	DMAMUX请求路由通道10同步溢出中断标志
DMAMUX_INT_FLAG_MUXCH11_SO	DMAMUX请求路由通道11同步溢出中断标志
DMAMUX_INT_FLAG_GENCH0_TO	DMAMUX请求生成通道0触发溢出中断标志
DMAMUX_INT_FLAG_GENCH1_TO	DMAMUX请求生成通道1触发溢出中断标志
DMAMUX_INT_FLAG_GENCH2_TO	DMAMUX请求生成通道2触发溢出中断标志
DMAMUX_INT_FLAG_GENCH3_TO	DMAMUX请求生成通道3触发溢出中断标志

## 枚举 dma\_channel\_enum

表 3-278. 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

枚举 `dmamux_multiplexer_channel_enum`表 3-279. 枚举 `dmamux_multiplexer_channel_enum`

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

枚举 `dmamux_generator_channel_enum`表 3-280. 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 3-281. 函数 `dma_deinit`

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

函数原型	void dma_deinit(uint32_t dma_periph, 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}	
<b>dma_periph</b>	DMA外设
<i>DMAx(x=0,1)</i>	DMA外设选择
输入参数{in}	
<b>channelx</b>	DMA通道
<i>DMA_CHx(x=0..5)</i>	DMA通道选择, 参考 <a href="#">表3-278. 枚举dma_channel_enum</a>
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* deinitialize DMA0 channel 0 registers */
dma_deinit(DMA0, DMA_CH0);
```

### 函数 dma\_struct\_para\_init

函数 dma\_struct\_para\_init 描述见下表:

表 3-282. 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}	
<b>init_struct</b>	用于初始化DMA通道的初始化数据的结构体地址, 参考 <a href="#">表3-272. 结构体 dma_parameter_struct</a>
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 3-283. 函数 dma\_init

函数名称	dma_init
函数原型	void dma_init(uint32_t dma_periph, dma_channel_enum channelx, dma_parameter_struct* init_struct);
功能描述	初始化DMA通道x
先决条件	无
被调用函数	无
输入参数{in}	
dma_periph	DMA外设
DMAx(x=0,1)	DMA外设选择
输入参数{in}	
channelx	DMA通道
DMA_CHx(x=0..5)	DMA通道选择, 参考 <a href="#">表3-278. 枚举dma_channel_enum</a>
输入参数{in}	
init_struct	用于初始化DMA通道的初始化数据的结构体地址, 参考 <a href="#">表3-272. 结构体 dma_parameter_struct</a>
输出参数{out}	
-	-
返回值	
-	-

例如：

```

/* DMA0 channel 0 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(DMA0, DMA_CH0, &dma_init_struct);

```

## 函数 dma\_circulation\_enable

函数 dma\_circulation\_enable 描述见下表：

表 3-284. 函数 dma\_circulation\_enable

函数名称	dma_circulation_enable
函数原型	void dma_circulation_enable(uint32_t dma_periph, dma_channel_enum channelx);
功能描述	DMA循环模式使能
先决条件	相应通道使能位CHEN需为0
被调用函数	无
输入参数{in}	
dma_periph	DMA外设
DMAx(x=0,1)	DMA外设选择
输入参数{in}	
channelx	DMA通道
DMA_CHx(x=0..5)	DMA通道选择, 参考 <a href="#">表3-278. 枚举dma_channel_enum</a>
输出参数{out}	
-	-
返回值	
-	-

例如：

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

## 函数 dma\_circulation\_disable

函数 dma\_circulation\_disable 描述见下表：

表 3-285. 函数 dma\_circulation\_disable

函数名称	dma_circulation_disable
函数原型	void dma_circulation_disable(uint32_t dma_periph, dma_channel_enum channelx);
功能描述	DMA循环模式禁能
先决条件	相应通道使能位CHEN需为0
被调用函数	无
输入参数{in}	
dma_periph	DMA外设
DMAx(x=0,1)	DMA外设选择
输入参数{in}	
channelx	DMA通道
DMA_CHx(x=0..5)	DMA通道选择, 参考 <a href="#">表3-278. 枚举dma_channel_enum</a>

输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* disable DMA0 channel 0 circulation mode */
dma_circulation_disable(DMA0, DMA_CH0);
```

### 函数 dma\_memory\_to\_memory\_enable

函数 dma\_memory\_to\_memory\_enable 描述见下表：

表 3-286. 函数 dma\_memory\_to\_memory\_enable

函数名称	dma_memory_to_memory_enable
函数原型	void dma_memory_to_memory_enable(uint32_t dma_periph, dma_channel_enum channelx);
功能描述	存储器到存储器DMA传输使能
先决条件	相应通道使能位CHEN需为0
被调用函数	无
输入参数{in}	
dma_periph	DMA外设
DMAx(x=0,1)	DMA外设选择
输入参数{in}	
channelx	DMA通道
DMA_CHx(x=0..5)	DMA通道选择，参考 <a href="#">表3-278. 枚举dma_channel_enum</a>
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* enable DMA0 channel 0 memory to memory mode */
dma_memory_to_memory_enable(DMA0, DMA_CH0);
```

### 函数 dma\_memory\_to\_memory\_disable

函数 dma\_memory\_to\_memory\_disable 描述见下表：

表 3-287. 函数 dma\_memory\_to\_memory\_disable

函数名称	dma_memory_to_memory_disable
函数原形	void dma_memory_to_memory_disable(uint32_t dma_periph, dma_channel_enum channelx);
功能描述	存储器到存储器DMA传输禁能

先决条件	相应通道使能位CHEN需为0
被调用函数	无
输入参数{in}	
<b>dma_periph</b>	DMA外设
<i>DMAx(x=0,1)</i>	DMA外设选择
输入参数{in}	
<b>channelx</b>	DMA通道
<i>DMA_CHx(x=0..5)</i>	DMA通道选择, 参考 <a href="#">表3-278. 枚举dma_channel_enum</a>
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* disable DMA0 channel 0 memory to memory mode */
dma_memory_to_memory_disable(DMA0, DMA_CH0);
```

### 函数 dma\_channel\_enable

函数 dma\_channel\_enable 描述见下表:

表 3-288. 函数 dma\_channel\_enable

函数名称	dma_channel_enable
函数原型	void dma_channel_enable(uint32_t dma_periph, dma_channel_enum channelx);
功能描述	DMA通道x传输使能
先决条件	无
被调用函数	无
输入参数{in}	
<b>dma_periph</b>	DMA外设
<i>DMAx(x=0,1)</i>	DMA外设选择
输入参数{in}	
<b>channelx</b>	DMA通道
<i>DMA_CHx(x=0..5)</i>	DMA通道选择, 参考 <a href="#">表3-278. 枚举dma_channel_enum</a>
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* enable DMA0 channel 0 */
dma_channel_enable(DMA0, DMA_CH0)
```

## 函数 dma\_channel\_disable

函数 dma\_channel\_disable 描述见下表：

表 3-289. 函数 dma\_channel\_disable

函数名称	dma_channel_disable
函数原型	void dma_channel_disable(uint32_t dma_periph, dma_channel_enum channelx);
功能描述	DMA通道x传输禁能
先决条件	无
被调用函数	无
输入参数{in}	
dma_periph	DMA外设
DMAx(x=0,1)	DMA外设选择
输入参数{in}	
channelx	DMA通道
DMA_CHx(x=0..5)	DMA通道选择, 参考 <a href="#">表3-278. 枚举dma_channel_enum</a>
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* disable DMA0 channel 0 */
dma_channel_disable(DMA0, DMA_CH0);
```

## 函数 dma\_periph\_address\_config

函数 dma\_periph\_address\_config 描述见下表：

表 3-290. 函数 dma\_periph\_address\_config

函数名称	dma_periph_address_config
函数原型	void dma_periph_address_config(uint32_t dma_periph, dma_channel_enum channelx, uint32_t address);
功能描述	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}	
dma_periph	DMA外设
DMAx(x=0,1)	DMA外设选择
输入参数{in}	
channelx	DMA通道
DMA_CHx(x=0..5)	DMA通道选择, 参考 <a href="#">表3-278. 枚举dma_channel_enum</a>



输入参数{in}	
address	外设基地址
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* configure DMA0 channel 0 periph address */
#define BANK0_WRITE_START_ADDR ((uint32_t)0x08004000)
dma_periph_address_config(DMA0, DMA_CH0, BANK0_WRITE_START_ADDR);
```

### 函数 dma\_memory\_address\_config

函数 dma\_memory\_address\_config 描述见下表：

表 3-291. 函数 dma\_memory\_address\_config

函数名称	dma_memory_address_config
函数原型	void dma_memory_address_config(uint32_t dma_periph, dma_channel_enum channelx, uint32_t address);
功能描述	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}	
dma_periph	DMA外设
DMAx(x=0,1)	DMA外设选择
输入参数{in}	
channelx	DMA通道
DMA_CHx(x=0..5)	DMA通道选择，参考 <a href="#">表3-278. 枚举dma_channel_enum</a>
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 DMA0 channel 0 memory address */
uint8_t g_destbuf[TRANSFER_NUM];
dma_memory_address_config(DMA0, DMA_CH0, (uint32_t) g_destbuf);
```

## 函数 dma\_transfer\_number\_config

函数 dma\_transfer\_number\_config 描述见下表：

表 3-292. 函数 dma\_transfer\_number\_config

函数名称	dma_transfer_number_config
函数原型	void dma_transfer_number_config(uint32_t dma_periph, dma_channel_enum channelx, uint32_t number);
功能描述	配置DMA通道x还有多少数据要传输
先决条件	相应通道使能位CHEN需为0
被调用函数	无
输入参数{in}	
dma_periph	DMA外设
DMAx(x=0,1)	DMA外设选择
输入参数{in}	
channelx	DMA通道
DMA_CHx(x=0..5)	DMA通道选择, 参考 <a href="#">表3-278. 枚举dma_channel_enum</a>
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 DMA0 channel 0 transfer number */

#define TRANSFER_NUM                0x400
dma_transfer_number_config(DMA0, DMA_CH0, TRANSFER_NUM);
```

## 函数 dma\_transfer\_number\_get

函数 dma\_transfer\_number\_get 描述见下表：

表 3-293. 函数 dma\_transfer\_number\_get

函数名称	dma_transfer_number_get
函数原型	uint32_t dma_transfer_number_get(uint32_t dma_periph, dma_channel_enum channelx);
功能描述	获取DMA通道x还有多少数据要传输
先决条件	相应通道使能位CHEN需为0
被调用函数	无
输入参数{in}	
dma_periph	DMA外设
DMAx(x=0,1)	DMA外设选择

输入参数{in}	
<b>channelx</b>	DMA通道
<i>DMA_CHx(x=0..5)</i>	DMA通道选择, 参考 <a href="#">表3-278. 枚举dma_channel_enum</a>
输出参数{out}	
-	-
返回值	
<b>uint32_t</b>	DMA数据传输剩余数量 (0x0 – 0xFFFF)

例如:

```
/* get DMA0 channel 0 transfer number */
```

```
uint32_t number = 0;
```

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

### 函数 dma\_priority\_config

函数 dma\_priority\_config 描述见下表:

**表 3-294. 函数 dma\_priority\_config**

<b>函数名称</b>	dma_priority_config
<b>函数原型</b>	void dma_priority_config(uint32_t dma_periph, dma_channel_enum channelx, uint32_t priority);
<b>功能描述</b>	DMA通道x的传输软件优先级配置
<b>先决条件</b>	相应通道使能位CHEN需为0
<b>被调用函数</b>	无
输入参数{in}	
<b>dma_periph</b>	DMA外设
<i>DMAx(x=0,1)</i>	DMA外设选择
输入参数{in}	
<b>channelx</b>	DMA通道
<i>DMA_CHx(x=0..5)</i>	DMA通道选择, 参考 <a href="#">表3-278. 枚举dma_channel_enum</a>
输入参数{in}	
<b>priority</b>	DMA通道软件优先级
<i>DMA_PRIORITY_LOW</i>	低优先级
<i>DMA_PRIORITY_MEDIUM</i>	中优先级
<i>DMA_PRIORITY_HIGH</i>	高优先级
<i>DMA_PRIORITY_ULTRA_HIGH</i>	极高优先级
输出参数{out}	
-	-

返回值	
-	-

例如：

```
/* configure DMA0 channel 0 priority */
dma_priority_config(DMA0, DMA_CH0, DMA_PRIORITY_ULTRA_HIGH);
```

### 函数 dma\_memory\_width\_config

函数 dma\_memory\_width\_config 描述见下表：

表 3-295. 函数 dma\_memory\_width\_config

函数名称	dma_memory_width_config
函数原型	void dma_memory_width_config(uint32_t dma_periph, dma_channel_enum channelx, uint32_t mwidth);
功能描述	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}	
<b>dma_periph</b>	DMA外设
<i>DMAx(x=0,1)</i>	DMA外设选择
输入参数{in}	
<b>channelx</b>	DMA通道
<i>DMA_CHx(x=0..5)</i>	DMA通道选择，参考 <a href="#">表3-278. 枚举dma_channel_enum</a>
输入参数{in}	
<b>mwidth</b>	存储器数据传输宽度
<i>DMA_MEMORY_WIDTH_8BIT</i>	8位数据传输宽度
<i>DMA_MEMORY_WIDTH_16BIT</i>	16位数据传输宽度
<i>DMA_MEMORY_WIDTH_32BIT</i>	32位数据传输宽度
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* configure DMA0 channel 0 memory width */
dma_memory_width_config(DMA0, DMA_CH0, DMA_MEMORY_WIDTH_8BIT);
```

**函数 dma\_periph\_width\_config**

函数 dma\_periph\_width\_config 描述见下表:

**表 3-296. 函数 dma\_periph\_width\_config**

函数名称	dma_periph_width_config
函数原型	void dma_periph_width_config(uint32_t dma_periph, dma_channel_enum channelx, uint32_t pwidth);
功能描述	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}	
dma_periph	DMA外设
DMAx(x=0,1)	DMA外设选择
输入参数{in}	
channelx	DMA通道
DMA_CHx(x=0..5)	DMA通道选择, 参考 <a href="#">表3-278. 枚举dma_channel_enum</a>
输入参数{in}	
pwidth	外设数据传输宽度
DMA_PERIPHERAL_WIDTH_8BIT	8位数据传输宽度
DMA_PERIPHERAL_WIDTH_16BIT	16位数据传输宽度
DMA_PERIPHERAL_WIDTH_32BIT	32位数据传输宽度
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* configure DMA0 channel 0 periph width */
dma_periph_width_config(DMA0, DMA_CH0, DMA_PERIPHERAL_WIDTH_8BIT);
```

**函数 dma\_memory\_increase\_enable**

函数 dma\_memory\_increase\_enable 描述见下表:

**表 3-297. 函数 dma\_memory\_increase\_enable**

函数名称	dma_memory_increase_enable
函数原型	void dma_memory_increase_enable(uint32_t dma_periph, dma_channel_enum channelx);
功能描述	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}	
<b>dma_periph</b>	DMA外设
<i>DMAx(x=0,1)</i>	DMA外设选择
输入参数{in}	
<b>channelx</b>	DMA通道
<i>DMA_CHx(x=0..5)</i>	DMA通道选择, 参考 <a href="#">表3-278. 枚举dma_channel_enum</a>
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* enable DMA0 channel 0 memory increase */
dma_memory_increase_enable(DMA0, DMA_CH0);
```

### 函数 dma\_memory\_increase\_disable

函数 dma\_memory\_increase\_disable 描述见下表:

表 3-298. 函数 dma\_memory\_increase\_disable

函数名称	dma_memory_increase_disable
函数原型	void dma_memory_increase_disable(uint32_t dma_periph, dma_channel_enum channelx);
功能描述	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}	
<b>dma_periph</b>	DMA外设
<i>DMAx(x=0,1)</i>	DMA外设选择
输入参数{in}	
<b>channelx</b>	DMA通道
<i>DMA_CHx(x=0..5)</i>	DMA通道选择, 参考 <a href="#">表3-278. 枚举dma_channel_enum</a>
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* disable DMA channel 0 memory increase */
dma_memory_increase_disable(DMA0, DMA_CH0);
```

## 函数 dma\_periph\_increase\_enable

函数 dma\_periph\_increase\_enable 描述见下表：

**表 3-299. 函数 dma\_periph\_increase\_enable**

函数名称	dma_periph_increase_enable
函数原型	void dma_periph_increase_enable(uint32_t dma_periph, dma_channel_enum channelx);
功能描述	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}	
dma_periph	DMA外设
DMAx(x=0,1)	DMA外设选择
输入参数{in}	
channelx	DMA通道
DMA_CHx(x=0..5)	DMA通道选择，参考 <a href="#">表3-278. 枚举dma_channel_enum</a>
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* enable next address increasement algorithm of DMA channel 0 */
dma_periph_increase_enable(DMA0, DMA_CH0);
```

## 函数 dma\_periph\_increase\_disable

函数 dma\_periph\_increase\_disable 描述见下表：

**表 3-300. 函数 dma\_periph\_increase\_disable**

函数名称	dma_periph_increase_disable
函数原型	void dma_periph_increase_disable(uint32_t dma_periph, dma_channel_enum channelx);
功能描述	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}	
dma_periph	DMA外设
DMAx(x=0,1)	DMA外设选择
输入参数{in}	
channelx	DMA通道
DMA_CHx(x=0..5)	DMA通道选择，参考 <a href="#">表3-278. 枚举dma_channel_enum</a>

输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* disable next address increasement algorithm of DMA0 channel 0*/
dma_periph_increase_disable(DMA0, DMA_CH0);
```

### 函数 dma\_transfer\_direction\_config

函数 dma\_transfer\_direction\_config 描述见下表：

表 3-301. 函数 dma\_transfer\_direction\_config

函数名称	dma_transfer_direction_config
函数原型	void dma_transfer_direction_config(uint32_t dma_periph, dma_channel_enum channelx, uint32_t direction);
功能描述	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}	
dma_periph	DMA外设
DMAx(x=0,1)	DMA外设选择
输入参数{in}	
channelx	DMA通道
DMA_CHx(x=0..5)	DMA通道选择，参考 <a href="#">表3-278. 枚举dma_channel_enum</a>
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 DMA0 channel 0 transfer direction*/
dma_transfer_direction_config(DMA0, DMA_CH0, DMA_PERIPHERAL_TO_MEMORY);
```



## 函数 dma\_flag\_get

函数 dma\_flag\_get 描述见下表：

表 3-302. 函数 dma\_flag\_get

函数名称	dma_flag_get
函数原型	FlagStatus dma_flag_get(uint32_t dma_periph, dma_channel_enum channelx, uint32_t flag);
功能描述	获取DMA通道x标志位状态
先决条件	无
被调用函数	无
输入参数{in}	
dma_periph	DMA外设
DMAx(x=0,1)	DMA外设选择
输入参数{in}	
channelx	DMA通道
DMA_CHx(x=0..5)	DMA通道选择，参考 <a href="#">表3-278. 枚举dma_channel_enum</a>
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 DMA0 channel 0 flag*/
FlagStatus flag = RESET;
flag = dma_flag_get(DMA0, DMA_CH0, DMA_FLAG_FTF);
```

## 函数 dma\_flag\_clear

函数 dma\_flag\_clear 描述见下表：

表 3-303. 函数 dma\_flag\_clear

函数名称	dma_flag_clear
函数原型	void dma_flag_clear(uint32_t dma_periph, dma_channel_enum channelx, uint32_t flag);
功能描述	清除DMA通道x标志位状态
先决条件	无
被调用函数	无

输入参数{in}	
<b>dma_periph</b>	DMA外设
<i>DMAx(x=0,1)</i>	DMA外设选择
输入参数{in}	
<b>channelx</b>	DMA通道
<i>DMA_CHx(x=0..5)</i>	DMA通道选择, 参考 <a href="#">表3-278. 枚举dma_channel_enum</a>
输入参数{in}	
<b>flag</b>	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 DMA0 channel 0 flag*/
dma_flag_clear(DMA0, DMA_CH0, DMA_FLAG_FTF);
```

### 函数 dma\_interrupt\_enable

函数 dma\_interrupt\_enable 描述见下表:

表 3-304. 函数 dma\_interrupt\_enable

函数名称	dma_interrupt_enable
函数原型	void dma_interrupt_enable(uint32_t dma_periph, dma_channel_enum channelx, uint32_t source);
功能描述	DMA通道x中断使能
先决条件	无
被调用函数	无
输入参数{in}	
<b>dma_periph</b>	DMA外设
<i>DMAx(x=0,1)</i>	DMA外设选择
输入参数{in}	
<b>channelx</b>	DMA通道
<i>DMA_CHx(x=0..5)</i>	DMA通道选择, 参考 <a href="#">表3-278. 枚举dma_channel_enum</a>
输入参数{in}	
<b>source</b>	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 DMA0 channel0 interrupt */
dma_interrupt_enable(DMA0, DMA_CH0, DMA_INT_FTF);
```

### 函数 dma\_interrupt\_disable

函数 dma\_interrupt\_disable 描述见下表：

**表 3-305. 函数 dma\_interrupt\_disable**

函数名称	dma_interrupt_disable
函数原型	void dma_interrupt_disable(uint32_t dma_periph, dma_channel_enum channelx, uint32_t source);
功能描述	DMA通道x中断禁能
先决条件	无
被调用函数	无
输入参数{in}	
dma_periph	DMA外设
DMAx(x=0,1)	DMA外设选择
输入参数{in}	
channelx	DMA通道
DMA_CHx(x=0..5)	DMA通道选择，参考 <a href="#">表3-278. 枚举dma_channel_enum</a>
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 DMA0 channel0 interrupt */
dma_interrupt_disable(DMA0, DMA_CH0, DMA_INT_FTF);
```

### 函数 dma\_interrupt\_flag\_get

函数 dma\_interrupt\_flag\_get 描述见下表：

表 3-306. 函数 dma\_interrupt\_flag\_get

函数名称	dma_interrupt_flag_get
函数原型	FlagStatus dma_interrupt_flag_get(uint32_t dma_periph, dma_channel_enum channelx, uint32_t flag);
功能描述	获取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}	
dma_periph	DMA外设
DMAx(x=0,1)	DMA外设选择
输入参数{in}	
channelx	DMA通道
DMA_CHx(x=0..5)	DMA通道选择, 参考 <a href="#">表3-278. 枚举dma_channel_enum</a>
输入参数{in}	
flag	DMA标志
DMA_INT_FLAG_FTF	DMA通道传输完成中断标志
DMA_INT_FLAG_HTF	DMA通道半传输完成中断标志
DMA_INT_FLAG_ERR	DMA通道错误中断标志
输出参数{out}	
-	-
返回值	
FlagStatus	SET或RESET

例如:

```
/* get DMA0 channel 3 interrupt flag*/
if(dma_interrupt_flag_get(DMA0, DMA_CH3, DMA_INT_FLAG_FTF)){
    dma_interrupt_flag_clear(DMA0, DMA_CH3, DMA_INT_FLAG_G);
}
```

### 函数 dma\_interrupt\_flag\_clear

函数 dma\_interrupt\_flag\_clear 描述见下表:

表 3-307. 函数 dma\_interrupt\_flag\_clear

函数名称	dma_interrupt_flag_clear
函数原型	void dma_interrupt_flag_clear(uint32_t dma_periph, dma_channel_enum channelx, uint32_t flag);
功能描述	清除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}	
<b>dma_periph</b>	DMA外设
<i>DMAx(x=0,1)</i>	DMA外设选择
输入参数{in}	
<b>channelx</b>	DMA通道
<i>DMA_CHx(x=0..5)</i>	DMA通道选择, 参考 <a href="#">表3-278. 枚举dma_channel_enum</a>
输入参数{in}	
<b>flag</b>	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>	DMA通道错误中断标志
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* clear DMA channel 3 interrupt flag*/
if(dma_interrupt_flag_get(DMA0, DMA_CH3, DMA_INT_FLAG_FTF)){
    dma_interrupt_flag_clear(DMA0, DMA_CH3, DMA_INT_FLAG_G);
}
```

### 函数 dmamux\_sync\_struct\_para\_init

函数 dmamux\_sync\_struct\_para\_init 描述见下表:

表 3-308. 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}	
<b>init_struct</b>	一个已经定义的dmamux_sync_parameter_struct结构体变量地址, 参考 <a href="#">表3-273. 结构体dmamux_sync_parameter_struct</a>
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 3-309. 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..11)	DMAMUX通道选择，参考 <a href="#">表3-279. 枚举 dmamux_multiplexer_channel_enum</a>
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 <a href="#">表 3-273. 结构体dmamux_sync_parameter_struct</a>
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 3-310. 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..11)	DMAMUX通道选择, 参考 <a href="#">表3-279. 枚举 dmamux_multiplexer_channel_enum</a>
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 3-311. 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_MUXCHx(x=0..11)	DMAMUX通道选择, 参考 <a href="#">表3-279. 枚举 dmamux_multiplexer_channel_enum</a>
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 3-312. 函数 dmamux\_event\_generation\_enable**

函数名称	dmamux_event_generation_enable
函数原型	void dmamux_event_generation_enable(dmamux_multiplexer_channel_enum channelx);
功能描述	使能DMAMUX事件输出
先决条件	无
被调用函数	无
输入参数{in}	
channelx	指定要初始化的DMAMUX请求路由通道
DMAMUX_MUXCH x(x=0..11)	DMAMUX通道选择，参考 <a href="#">表3-279. 枚举 dmamux_multiplexer_channel_enum</a>
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 3-313. 函数 dmamux\_event\_generation\_disable**

函数名称	dmamux_event_generation_disable
函数原型	void dmamux_event_generation_disable(dmamux_multiplexer_channel_enum channelx);
功能描述	禁能DMAMUX事件输出
先决条件	无
被调用函数	无
输入参数{in}	
channelx	指定要初始化的DMAMUX请求路由通道
DMAMUX_MUXCH x(x=0..11)	DMAMUX通道选择，参考 <a href="#">表3-279. 枚举 dmamux_multiplexer_channel_enum</a>
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 3-314. 函数 dmamux\_gen\_struct\_para\_init**

函数名称	dmamux_gen_struct_para_init
函数原型	void dmamux_gen_struct_para_init(dmamux_gen_parameter_struct *init_struct);
功能描述	将DMAMUX请求生成结构体中所有参数初始化为默认值
先决条件	无
被调用函数	无
输入参数{in}	
-	-
输出参数{out}	
init_struct	一个已经定义的dmamux_gen_parameter_struct结构体变量地址，参考 <a href="#">表3-274. 结构体dmamux_gen_parameter_struct</a>
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 3-315. 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 <a href="#">表3-280. 枚举 dmamux_generator_channel_enum</a>

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 <a href="#">表3-274. 结构体dmamux_gen_parameter_struct</a>
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 3-316. 函数 dmamux\_request\_generator\_channel\_enable**

函数名称	dmamux_request_generator_channel_enable
函数原型	void dmamux_request_generator_channel_enable(dmamux_generator_channel_enum channelx);
功能描述	使能DMAMUX请求生成通道x
先决条件	无
被调用函数	无
输入参数{in}	
channelx	指定要初始化的DMAMUX请求生成通道
DMAMUX_GENCHx (x=0..3)	DMAMUX请求生成通道选择，参考 <a href="#">表3-280. 枚举 dmamux_generator_channel_enum</a>
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 3-317. 函数 dmamux\_request\_generator\_channel\_disable**

函数名称	dmamux_request_generator_channel_disable
函数原型	void dmamux_request_generator_channel_disable(dmamux_generator_channel_enum channelx);
功能描述	禁能DMAMUX请求生成通道x
先决条件	无
被调用函数	无
输入参数{in}	
channelx	指定要初始化的DMAMUX请求生成通道
DMAMUX_GENCHx (x=0..3)	DMAMUX请求生成通道选择，参考 <a href="#">表3-280. 枚举 dmamux_generator_channel_enum</a>
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 3-318. 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..11)	DMAMUX通道选择，参考 <a href="#">表3-279. 枚举 dmamux_multiplexer_channel_enum</a>
输入参数{in}	
polarity	同步输入有效边沿
DMAMUX_SYNC_N	不检测边沿

<i>O_EVENT</i>	
<i>DMAMUX_SYNC_RISING</i>	上升沿
<i>DMAMUX_SYNC_FALLING</i>	下降沿
<i>DMAMUX_SYNC_RISING_FALLING</i>	上升和下降沿
输出参数{out}	
-	-
返回值	
-	-

例如:

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

### 函数 dmamux\_request\_forward\_number\_config

函数 dmamux\_request\_forward\_number\_config 描述见下表:

表 3-319. 函数 dmamux\_request\_forward\_number\_config

函数名称	dmamux_request_forward_number_config
函数原型	void dmamux_request_forward_number_config(dmamux_mux_channel_enum channelx, uint32_t number);
功能描述	配置DMAMUX通道x要传输多少个DMA请求
先决条件	无
被调用函数	无
输入参数{in}	
channelx	指定要初始化的DMAMUX请求路由通道
DMAMUX_MUXCHx(x=0..11)	DMAMUX通道选择, 参考 <a href="#">表3-279. 枚举 dmamux_mux_channel_enum</a>
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 3-320. 函数 dmamux\_sync\_id\_config**

函数名称	dmamux_sync_id_config
函数原型	void dmamux_sync_id_config(dmamux_multiplexer_channel_enum channelx, uint32_t id);
功能描述	配置DMAMUX同步输入标识
先决条件	无
被调用函数	无
输入参数{in}	
channelx	指定要初始化的DMAMUX请求路由通道
DMAMUX_MUXCHx(x=0..11)	DMAMUX通道选择，参考 <a href="#">表3-279. 枚举 dmamux_multiplexer_channel_enum</a>
输入参数{in}	
id	同步输入标识
DMAMUX_SYNC_EXTI0	同步输入信号为EXTI0
DMAMUX_SYNC_EXTI1	同步输入信号为EXTI1
DMAMUX_SYNC_EXTI2	同步输入信号为EXTI2
DMAMUX_SYNC_EXTI3	同步输入信号为EXTI3
DMAMUX_SYNC_EXTI4	同步输入信号为EXTI4
DMAMUX_SYNC_EXTI5	同步输入信号为EXTI5
DMAMUX_SYNC_EXTI6	同步输入信号为EXTI6
DMAMUX_SYNC_EXTI7	同步输入信号为EXTI7
DMAMUX_SYNC_EXTI8	同步输入信号为EXTI8
DMAMUX_SYNC_EXTI9	同步输入信号为EXTI9
DMAMUX_SYNC_EXTI10	同步输入信号为EXTI10
DMAMUX_SYNC_EXTI11	同步输入信号为EXTI11
DMAMUX_SYNC_EXTI12	同步输入信号为EXTI12

DMAMUX_SYNC_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_OUT0	同步输入信号为Evt_out0
DMAMUX_SYNC_E VT1_OUT1	同步输入信号为Evt_out1
DMAMUX_SYNC_E VT2_OUT2	同步输入信号为Evt_out2
DMAMUX_SYNC_E VT3_OUT3	同步输入信号为Evt_out3
DMAMUX_SYNC_L PTIMER_OUT	同步输入信号为LPTIMER_OUT
输出参数{out}	
-	-
返回值	
-	-

例如：

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

### 函数 dmamux\_request\_id\_config

函数 dmamux\_request\_id\_config 描述见下表：

表 3-321. 函数 dmamux\_request\_id\_config

函数名称	dmamux_request_id_config
函数原型	void dmamux_request_id_config(dmamux_mux_channel_enum channelx, uint32_t id);
功能描述	配置DMAMUX请求路由通道输入标识
先决条件	无
被调用函数	无
输入参数{in}	
channelx	指定要初始化的DMAMUX请求路由通道
DMAMUX_MUXCH x(x=0..11)	DMAMUX通道选择，参考 <a href="#">表3-279. 枚举 dmamux_mux_channel_enum</a>
输入参数{in}	
id	DMA请求输入标识
DMA_REQUEST_M	内存到内存传输

2M	
DMA_REQUEST_GENERATOR0	DMAMUX GENERATOR0 请求
DMA_REQUEST_GENERATOR1	DMAMUX GENERATOR1 请求
DMA_REQUEST_GENERATOR2	DMAMUX GENERATOR2 请求
DMA_REQUEST_GENERATOR3	DMAMUX GENERATOR3 请求
DMA_REQUEST_ADC0_A	DMAMUX ADC0_A 请求
DMA_REQUEST_ADC2_A	DMAMUX ADC2_A 请求
DMA_REQUEST_ADC0_B	DMAMUX ADC0_B 请求
DMA_REQUEST_ADC2_B	DMAMUX ADC2_B 请求
DMA_REQUEST_ADC0_C	DMAMUX ADC0_C 请求
DMA_REQUEST_ADC2_C	DMAMUX ADC2_C 请求
DMA_REQUEST_I2C_RX	DMAMUX I2C_RX 请求
DMA_REQUEST_I2C_TX	DMAMUX I2C_TX 请求
DMA_REQUEST_UART0_RX	DMAMUX UART0_RX 请求
DMA_REQUEST_UART0_TX	DMAMUX UART0_TX 请求
DMA_REQUEST_UART1_RX	DMAMUX UART1_RX 请求
DMA_REQUEST_UART1_TX	DMAMUX UART1_TX 请求
DMA_REQUEST_UART2_RX	DMAMUX UART2_RX 请求
DMA_REQUEST_UART2_TX	DMAMUX UART2_TX 请求
DMA_REQUEST_UART3_RX	DMAMUX UART3_RX 请求
DMA_REQUEST_UART3_TX	DMAMUX UART3_TX 请求
DMA_REQUEST_SPI0_RX	DMAMUX SPI0_RX 请求

PI0_RX	
DMA_REQUEST_S PI0_TX	DMAMUX SPI0_TX 请求
DMA_REQUEST_TI MER0_UP	DMAMUX TIMER0_UP 请求
DMA_REQUEST_TI MER0_CO	DMAMUX TIMER0_CO 请求
DMA_REQUEST_TI MER0_TI	DMAMUX TIMER0_TI 请求
DMA_REQUEST_TI MER0_CH0	DMAMUX TIMER0_CH0 请求
DMA_REQUEST_TI MER0_CH1	DMAMUX TIMER0_CH1 请求
DMA_REQUEST_TI MER0_CH2	DMAMUX TIMER0_CH2 请求
DMA_REQUEST_TI MER0_CH3	DMAMUX TIMER0_CH3 请求
DMA_REQUEST_TI MER0_MCH0	DMAMUX TIMER0_MCH0 请求
DMA_REQUEST_TI MER0_MCH1	DMAMUX TIMER0_MCH1 请求
DMA_REQUEST_TI MER0_MCH2	DMAMUX TIMER0_MCH2 请求
DMA_REQUEST_TI MER0_MCH3	DMAMUX TIMER0_MCH3 请求
DMA_REQUEST_TI MER7_UP	DMAMUX TIMER7_UP 请求
DMA_REQUEST_TI MER7_CO	DMAMUX TIMER7_CO 请求
DMA_REQUEST_TI MER7_TI	DMAMUX TIMER7_TI 请求
DMA_REQUEST_TI MER7_CH0	DMAMUX TIMER7_CH0 请求
DMA_REQUEST_TI MER7_CH1	DMAMUX TIMER7_CH1 请求
DMA_REQUEST_TI MER7_CH2	DMAMUX TIMER7_CH2 请求
DMA_REQUEST_TI MER7_CH3	DMAMUX TIMER7_CH3 请求
DMA_REQUEST_TI MER7_MCH0	DMAMUX TIMER7_MCH0 请求
DMA_REQUEST_TI	DMAMUX TIMER7_MCH1 请求



<i>MER7_MCH1</i>	
<i>DMA_REQUEST_TIMER7_MCH2</i>	DMAMUX TIMER7_MCH2 请求
<i>DMA_REQUEST_TIMER7_MCH3</i>	DMAMUX TIMER7_MCH3 请求
<i>DMA_REQUEST_TIMER1_UP</i>	DMAMUX TIMER1_UP 请求
<i>DMA_REQUEST_TIMER1_TI</i>	DMAMUX TIMER1_TI 请求
<i>DMA_REQUEST_TIMER1_CH0</i>	DMAMUX TIMER1_CH0 请求
<i>DMA_REQUEST_TIMER1_CH1</i>	DMAMUX TIMER1_CH1 请求
<i>DMA_REQUEST_TIMER1_CH2</i>	DMAMUX TIMER1_CH2 请求
<i>DMA_REQUEST_TIMER1_CH3</i>	DMAMUX TIMER1_CH3 请求
<i>DMA_REQUEST_TIMER2_UP</i>	DMAMUX TIMER2_UP 请求
<i>DMA_REQUEST_TIMER2_TI</i>	DMAMUX TIMER2_TI 请求
<i>DMA_REQUEST_TIMER2_CH0</i>	DMAMUX TIMER2_CH0 请求
<i>DMA_REQUEST_TIMER2_CH1</i>	DMAMUX TIMER2_CH1 请求
<i>DMA_REQUEST_TIMER2_CH2</i>	DMAMUX TIMER2_CH2 请求
<i>DMA_REQUEST_TIMER2_CH3</i>	DMAMUX TIMER2_CH3 请求
<i>DMA_REQUEST_TMU_TX</i>	DMAMUX TMU_TX 请求
<i>DMA_REQUEST_TMU_RX</i>	DMAMUX TMU_RX 请求
<i>DMA_REQUEST_GPTIMER0_UP</i>	DMAMUX GPTIMER0_UP 请求
<i>DMA_REQUEST_GPTIMER0_PRD</i>	DMAMUX GPTIMER0_PRD 请求
<i>DMA_REQUEST_GPTIMER0_CH0</i>	DMAMUX GPTIMER0_CH0 请求
<i>DMA_REQUEST_GPTIMER0_CH1</i>	DMAMUX GPTIMER0_CH1 请求
<i>DMA_REQUEST_GPTIMER1_UP</i>	DMAMUX GPTIMER1_UP 请求

<i>PTIMER1_UP</i>	
<i>DMA_REQUEST_G</i> <i>PTIMER1_PRD</i>	DMAMUX GPTIMER1_PRD 请求
<i>DMA_REQUEST_G</i> <i>PTIMER1_CH0</i>	DMAMUX GPTIMER1_CH0 请求
<i>DMA_REQUEST_G</i> <i>PTIMER1_CH1</i>	DMAMUX GPTIMER1_CH1 请求
<i>DMA_REQUEST_E</i> <i>VIC0</i>	DMAMUX EVIC0 请求
<i>DMA_REQUEST_E</i> <i>VIC1</i>	DMAMUX EVIC1 请求
<i>DMA_REQUEST_E</i> <i>VIC2</i>	DMAMUX EVIC2 请求
<i>DMA_REQUEST_E</i> <i>VIC3</i>	DMAMUX EVIC3 请求
<i>DMA_REQUEST_E</i> <i>VIC4</i>	DMAMUX EVIC4 请求
<i>DMA_REQUEST_E</i> <i>VIC5</i>	DMAMUX EVIC5 请求
<i>DMA_REQUEST_E</i> <i>VIC6</i>	DMAMUX EVIC6 请求
<i>DMA_REQUEST_E</i> <i>VIC7</i>	DMAMUX EVIC7 请求
<i>DMA_REQUEST_E</i> <i>VIC8</i>	DMAMUX EVIC8 请求
<i>DMA_REQUEST_E</i> <i>VIC9</i>	DMAMUX EVIC9 请求
<i>DMA_REQUEST_A</i> <i>DC0_D</i>	DMAMUX ADC0_D 请求
<i>DMA_REQUEST_A</i> <i>DC2_D</i>	DMAMUX ADC2_D 请求
<i>DMA_REQUEST_G</i> <i>PTIMER0_CH0CO</i> <i>MV_ADD</i>	DMAMUX GPTIMER0_CH0COMV_ADD 请求
<i>DMA_REQUEST_G</i> <i>PTIMER0_CH1CO</i> <i>MV_ADD</i>	DMAMUX GPTIMER0_CH1COMV_ADD 请求
<i>DMA_REQUEST_G</i> <i>PTIMER1_CH0CO</i> <i>MV_ADD</i>	DMAMUX GPTIMER1_CH0COMV_ADD 请求
<i>DMA_REQUEST_G</i> <i>PTIMER1_CH1CO</i>	DMAMUX GPTIMER1_CH1COMV_ADD 请求

<i>MV_ADD</i>	
<i>DMA_REQUEST_E</i> <i>VIC10</i>	DMAMUX EVIC10 请求
<i>DMA_REQUEST_E</i> <i>VIC11</i>	DMAMUX EVIC11 请求
输出参数{out}	
-	-
返回值	
-	-

例如:

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

### 函数 dmamux\_trigger\_polarity\_config

函数 dmamux\_trigger\_polarity\_config 描述见下表:

表 3-322. 函数 dma\_interrupt\_disable

函数名称	dmamux_trigger_polarity_config
函数原型	void dmamux_trigger_polarity_config(dmamux_generator_channel_enum channelx, uint32_t polarity);
功能描述	配置DMAMUX触发输入的有效边沿
先决条件	无
被调用函数	无
输入参数{in}	
channelx	指定要初始化的DMAMUX请求生成通道
DMAMUX_GENCHx (x=0..3)	DMAMUX请求生成通道选择, 参考 <a href="#">表3-280. 枚举 dmamux_generator_channel_enum</a>
输入参数{in}	
polarity	触发输入信号有效边沿
DMAMUX_GEN_NO_EVENT	不检测边沿
DMAMUX_GEN_RISING	上升沿
DMAMUX_GEN_FALLING	下降沿
DMAMUX_GEN_RISING_FALLING	上升和下降沿
输出参数{out}	
-	-
返回值	
-	-

例如:

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

### 函数 dmamux\_request\_generate\_number\_config

函数 dmamux\_request\_generate\_number\_config 描述见下表:

表 3-323. 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 <a href="#">表3-280. 枚举 dmamux_generator_channel_enum</a>
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 3-324. 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}	

<b>channelx</b>	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 <a href="#">表3-280. 枚举</a> <a href="#">dmamux_generator_channel_enum</a>
<b>输入参数{in}</b>	
<b>id</b>	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_EVT_OUT0	触发输入为Evt_out0
DMAMUX_TRIGGE R_EVT_OUT1	触发输入为Evt_out1
DMAMUX_TRIGGE	触发输入为Evt_out2

R_EVT_OUT2	
DMAMUX_TRIGGER R_EVT_OUT3	触发输入为Evt_out3
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 3-325. 函数 dmamux\_flag\_get

函数名称	dmamux_flag_get
函数原型	FlagStatus dmamux_flag_get(dmamux_flag_enum flag);
功能描述	获取DMAMUX通道x标志位状态
先决条件	无
被调用函数	无
输入参数{in}	
flag	标志类型，参考 <a href="#">表3-276. 枚举dmamux_flag_enum</a>
DMAMUX_FLAG_M UXCH0_SO	DMAMUX请求路由通道0同步溢出标志
DMAMUX_FLAG_M UXCH1_SO	DMAMUX请求路由通道1同步溢出标志
DMAMUX_FLAG_M UXCH2_SO	DMAMUX请求路由通道2同步溢出标志
DMAMUX_FLAG_M UXCH3_SO	DMAMUX请求路由通道3同步溢出标志
DMAMUX_FLAG_M UXCH4_SO	DMAMUX请求路由通道4同步溢出标志
DMAMUX_FLAG_M UXCH5_SO	DMAMUX请求路由通道5同步溢出标志
DMAMUX_FLAG_M UXCH6_SO	DMAMUX请求路由通道6同步溢出标志
DMAMUX_FLAG_M UXCH7_SO	DMAMUX请求路由通道7同步溢出标志
DMAMUX_FLAG_M UXCH8_SO	DMAMUX请求路由通道8同步溢出标志
DMAMUX_FLAG_M	DMAMUX请求路由通道9同步溢出标志

<i>UXCH9_SO</i>	
<i>DMAMUX_FLAG_M UXCH10_SO</i>	DMAMUX请求路由通道10同步溢出标志
<i>DMAMUX_FLAG_M UXCH11_SO</i>	DMAMUX请求路由通道11同步溢出标志
<i>DMAMUX_FLAG_G ENCH0_TO</i>	DMAMUX请求生成通道0触发溢出标志
<i>DMAMUX_FLAG_G ENCH1_TO</i>	DMAMUX请求生成通道1触发溢出标志
<i>DMAMUX_FLAG_G ENCH2_TO</i>	DMAMUX请求生成通道2触发溢出标志
<i>DMAMUX_FLAG_G ENCH3_TO</i>	DMAMUX请求生成通道3触发溢出标志
输出参数{out}	
-	-
返回值	
<b>FlagStatus</b>	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 3-326. 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}	
<b>flag</b>	标志类型，参考 <a href="#">表3-276. 枚举dmamux_flag_enum</a>
<i>DMAMUX_FLAG_M UXCH0_SO</i>	DMAMUX请求路由通道0同步溢出标志
<i>DMAMUX_FLAG_M UXCH1_SO</i>	DMAMUX请求路由通道1同步溢出标志
<i>DMAMUX_FLAG_M UXCH2_SO</i>	DMAMUX请求路由通道2同步溢出标志
<i>DMAMUX_FLAG_M UXCH3_SO</i>	DMAMUX请求路由通道3同步溢出标志

DMAMUX_FLAG_M UXCH4_SO	DMAMUX请求路由通道4同步溢出标志
DMAMUX_FLAG_M UXCH5_SO	DMAMUX请求路由通道5同步溢出标志
DMAMUX_FLAG_M UXCH6_SO	DMAMUX请求路由通道6同步溢出标志
DMAMUX_FLAG_M UXCH7_SO	DMAMUX请求路由通道7同步溢出标志
DMAMUX_FLAG_M UXCH8_SO	DMAMUX请求路由通道8同步溢出标志
DMAMUX_FLAG_M UXCH9_SO	DMAMUX请求路由通道9同步溢出标志
DMAMUX_FLAG_M UXCH10_SO	DMAMUX请求路由通道10同步溢出标志
DMAMUX_FLAG_M UXCH11_SO	DMAMUX请求路由通道11同步溢出标志
DMAMUX_FLAG_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
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 3-327. 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 <a href="#">表3-275. 枚举dmamux_interrupt_enum</a>
DMAMUX_INT_MU XCH0_SO	DMAMUX请求路由通道0同步溢出中断
DMAMUX_INT_MU XCH1_SO	DMAMUX请求路由通道1同步溢出中断
DMAMUX_INT_MU XCH2_SO	DMAMUX请求路由通道2同步溢出中断
DMAMUX_INT_MU XCH3_SO	DMAMUX请求路由通道3同步溢出中断
DMAMUX_INT_MU XCH4_SO	DMAMUX请求路由通道4同步溢出中断
DMAMUX_INT_MU XCH5_SO	DMAMUX请求路由通道5同步溢出中断
DMAMUX_INT_MU XCH6_SO	DMAMUX请求路由通道6同步溢出中断
DMAMUX_INT_MU XCH7_SO	DMAMUX请求路由通道7同步溢出中断
DMAMUX_INT_MU XCH8_SO	DMAMUX请求路由通道8同步溢出中断
DMAMUX_INT_MU XCH9_SO	DMAMUX请求路由通道9同步溢出中断
DMAMUX_INT_MU XCH10_SO	DMAMUX请求路由通道10同步溢出中断
DMAMUX_INT_MU XCH11_SO	DMAMUX请求路由通道11同步溢出中断
DMAMUX_INT_GE NCH0_TO	DMAMUX请求生成通道0触发溢出中断
DMAMUX_INT_GE NCH1_TO	DMAMUX请求生成通道1触发溢出中断
DMAMUX_INT_GE NCH2_TO	DMAMUX请求生成通道2触发溢出中断
DMAMUX_INT_GE NCH3_TO	DMAMUX请求生成通道3触发溢出中断
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 3-328. 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 <a href="#">表3-275. 枚举dmamux_interrupt_enum</a>
DMAMUX_INT_MU XCH0_SO	DMAMUX请求路由通道0同步溢出中断
DMAMUX_INT_MU XCH1_SO	DMAMUX请求路由通道1同步溢出中断
DMAMUX_INT_MU XCH2_SO	DMAMUX请求路由通道2同步溢出中断
DMAMUX_INT_MU XCH3_SO	DMAMUX请求路由通道3同步溢出中断
DMAMUX_INT_MU XCH4_SO	DMAMUX请求路由通道4同步溢出中断
DMAMUX_INT_MU XCH5_SO	DMAMUX请求路由通道5同步溢出中断
DMAMUX_INT_MU XCH6_SO	DMAMUX请求路由通道6同步溢出中断
DMAMUX_INT_MU XCH7_SO	DMAMUX请求路由通道7同步溢出中断
DMAMUX_INT_MU XCH8_SO	DMAMUX请求路由通道8同步溢出中断
DMAMUX_INT_MU XCH9_SO	DMAMUX请求路由通道9同步溢出中断
DMAMUX_INT_MU XCH10_SO	DMAMUX请求路由通道10同步溢出中断
DMAMUX_INT_MU XCH11_SO	DMAMUX请求路由通道11同步溢出中断
DMAMUX_INT_GE NCH0_TO	DMAMUX请求生成通道0触发溢出中断
DMAMUX_INT_GE NCH1_TO	DMAMUX请求生成通道1触发溢出中断
DMAMUX_INT_GE NCH2_TO	DMAMUX请求生成通道2触发溢出中断
DMAMUX_INT_GE	DMAMUX请求生成通道3触发溢出中断

NCH3_TO	
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 3-329. 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 <a href="#">表3-277. 枚举dmamux_interrupt_flag_enum</a>
DMAMUX_INT_FLAG_MUXCH0_SO	DMAMUX请求路由通道0同步溢出中断标志
DMAMUX_INT_FLAG_MUXCH1_SO	DMAMUX请求路由通道1同步溢出中断标志
DMAMUX_INT_FLAG_MUXCH2_SO	DMAMUX请求路由通道2同步溢出中断标志
DMAMUX_INT_FLAG_MUXCH3_SO	DMAMUX请求路由通道3同步溢出中断标志
DMAMUX_INT_FLAG_MUXCH4_SO	DMAMUX请求路由通道4同步溢出中断标志
DMAMUX_INT_FLAG_MUXCH5_SO	DMAMUX请求路由通道5同步溢出中断标志
DMAMUX_INT_FLAG_MUXCH6_SO	DMAMUX请求路由通道6同步溢出中断标志
DMAMUX_INT_FLAG_MUXCH7_SO	DMAMUX请求路由通道7同步溢出中断标志
DMAMUX_INT_FLAG_MUXCH8_SO	DMAMUX请求路由通道8同步溢出中断标志
DMAMUX_INT_FLAG_MUXCH9_SO	DMAMUX请求路由通道9同步溢出中断标志
DMAMUX_INT_FLAG_MUXCH10_SO	DMAMUX请求路由通道10同步溢出中断标志

<code>G_MUXCH10_SO</code>	
<code>DMAMUX_INT_FLAG_G_MUXCH11_SO</code>	DMAMUX请求路由通道11同步溢出中断标志
<code>DMAMUX_INT_FLAG_GENCH0_TO</code>	DMAMUX请求生成通道0触发溢出中断标志
<code>DMAMUX_INT_FLAG_GENCH1_TO</code>	DMAMUX请求生成通道1触发溢出中断标志
<code>DMAMUX_INT_FLAG_GENCH2_TO</code>	DMAMUX请求生成通道2触发溢出中断标志
<code>DMAMUX_INT_FLAG_GENCH3_TO</code>	DMAMUX请求生成通道3触发溢出中断标志
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 3-330. 函数 dmamux\_interrupt\_flag\_clear

函数名称	dmamux_interrupt_flag_clear
函数原型	FlagStatus dmamux_interrupt_flag_get(dmamux_interrupt_flag_enum int_flag);
功能描述	清除DMAMUX通道x中断标志位状态
先决条件	无
被调用函数	无
输入参数{in}	
int_flag	标志类型，参考 <a href="#">表3-277. 枚举dmamux_interrupt_flag_enum</a>
<code>DMAMUX_INT_FLAG_MUXCH0_SO</code>	DMAMUX请求路由通道0同步溢出中断标志
<code>DMAMUX_INT_FLAG_MUXCH1_SO</code>	DMAMUX请求路由通道1同步溢出中断标志
<code>DMAMUX_INT_FLAG_MUXCH2_SO</code>	DMAMUX请求路由通道2同步溢出中断标志
<code>DMAMUX_INT_FLAG_MUXCH3_SO</code>	DMAMUX请求路由通道3同步溢出中断标志
<code>DMAMUX_INT_FLAG_MUXCH4_SO</code>	DMAMUX请求路由通道4同步溢出中断标志

G_MUXCH4_SO	
DMAMUX_INT_FLG G_MUXCH5_SO	DMAMUX请求路由通道5同步溢出中断标志
DMAMUX_INT_FLG G_MUXCH6_SO	DMAMUX请求路由通道6同步溢出中断标志
DMAMUX_INT_FLG G_MUXCH7_SO	DMAMUX请求路由通道7同步溢出中断标志
DMAMUX_INT_FLG G_MUXCH8_SO	DMAMUX请求路由通道8同步溢出中断标志
DMAMUX_INT_FLG G_MUXCH9_SO	DMAMUX请求路由通道9同步溢出中断标志
DMAMUX_INT_FLG G_MUXCH10_SO	DMAMUX请求路由通道10同步溢出中断标志
DMAMUX_INT_FLG G_MUXCH11_SO	DMAMUX请求路由通道11同步溢出中断标志
DMAMUX_INT_FLG G_GENCH0_TO	DMAMUX请求生成通道0触发溢出中断标志
DMAMUX_INT_FLG G_GENCH1_TO	DMAMUX请求生成通道1触发溢出中断标志
DMAMUX_INT_FLG G_GENCH2_TO	DMAMUX请求生成通道2触发溢出中断标志
DMAMUX_INT_FLG G_GENCH3_TO	DMAMUX请求生成通道3触发溢出中断标志
输出参数{out}	
-	-
返回值	
-	-

例如：

```

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

```

### 3.12. EVIC

事件互连单元（EVIC）允许软件选择由各种外设产生的事件信号，用于各种应用。EVIC为外设提供了灵活的机制来选择不同的事件源。章节[3.12.1](#)描述了EVIC的寄存器列表，章节[3.12.2](#)对EVIC库函数进行说明。

### 3.12.1. 外设寄存器说明

EVIC寄存器列表如下表所示：

**表 3-331. EVIC 寄存器**

寄存器名称	寄存器描述
EVIC_SWEV	软件事件寄存器
EVIC_SGIOx	单个I/O事件互连寄存器x
EVIC INGRPE	输入组E事件互连寄存器
EVIC INGRPF	输入组F事件互连寄存器
EVIC_OUTGRPE	输出组E事件互连寄存器
EVIC_OUTGRPF	输出组F事件互连寄存器
EVIC_DAC0COV	DAC0转换事件互连寄存器
EVIC_ADC0COV	ADC0转换事件互连寄存器
EVIC_ADC2COV	ADC2转换事件互连寄存器
EVIC_RCU	RCU事件互连寄存器
EVIC_TIMER0	TIMER0事件互连寄存器
EVIC_TIMER7	TIMER7事件互连寄存器
EVIC_CPTIMER0	CPTIMER0事件互连寄存器
EVIC_CPTIMERW	CPTIMERW事件互连寄存器
EVIC_TIMER1	TIMER1事件互连寄存器
EVIC_TIMER2	TIMER2事件互连寄存器
EVIC_GPTIMER_0	GPTIMER事件互连寄存器0
EVIC_GPTIMER_1	GPTIMER事件互连寄存器1
EVIC_GPTIMER_2	GPTIMER事件互连寄存器2
EVIC_GPTIMER_3	GPTIMER事件互连寄存器3
EVIC_DAC0EN	DAC0使能事件互连寄存器
EVIC_DMAMUX_0	DMAMUX事件互连寄存器0
EVIC_DMAMUX_1	DMAMUX事件互连寄存器1
EVIC_DMAMUX_2	DMAMUX事件互连寄存器2
EVIC_DMAMUX_3	DMAMUX事件互连寄存器3
EVIC_DMAMUX_4	DMAMUX事件互连寄存器4
EVIC_DMAMUX_5	DMAMUX事件互连寄存器5
EVIC_SMCFG0	从模式配置寄存器0
EVIC_SMCFG1	从模式配置寄存器1
EVIC_SGIOCFGx	单个I/O配置寄存器x
EVIC_GRPFCFG	组E配置寄存器
EVIC_GRPEDH	组E数据保持寄存器
EVIC_GPRFCFG	组F配置寄存器
EVIC_GRPFDH	组F数据保持寄存器

### 3.12.2. 外设库函数说明

EVIC库函数列表如下表所示：

**表 3-332. EVIC 库函数**

库函数名称	库函数描述
evic_init	为目标外设设置事件源
evic_event_source_get	获得目标外设的触发信号源
evic_register_lock_set	锁定事件互连寄存器
evic_register_lock_get	获得事件互连寄存器锁定状态
evic_cptimer_slave_mode_select	比较匹配定时器从模式选择
evic_group_member_config	配置I/O组成员
evic_group_edge_detection_config	配置I/O组的边沿检测
evic_group_output_level_config	配置I/O组的输出电平
evic_group_overwrite_enable	使能I/O组覆写功能
evic_group_overwrite_disable	禁能I/O组覆写功能
evic_data_set	EVIC保持数据设置
evic_data_get	获取EVIC保持数据
evic_single_io_config	配置单I/O
evic_single_io_edge_detection_config	配置单I/O的边沿检测
evic_single_io_output_level_config	配置单I/O的输出电平
evic_register_write_enable	使能软件事件寄存器写
evic_register_write_disable	禁能软件事件寄存器写
evic_register_write_enable_get	获取软件事件寄存器写使能状态
evic_bit_write_enable	使能软件事件寄存器位写
evic_bit_write_disable	禁能软件事件寄存器位写
evic_bit_write_enable_get	获取软件事件寄存器位写使能状态
evic_software_event_generation	生成EVIC软件事件

#### 枚举类型 event\_source\_enum

**表 3-333. 枚举类型 event\_source\_enum**

成员名称	功能描述
EVIC_SOURCE_DISABLED	无事件信号被选择
EVIC_SOURCE_SOFTWARE	软件生成的事件信号
EVIC_SOURCE_SGIO_INPUT_DETECTION0	单个I/O输入检测0
EVIC_SOURCE_SGIO_INPUT_DETECTION1	单个I/O输入检测1
EVIC_SOURCE_SGIO_INPUT_DETECTION2	单个I/O输入检测2
EVIC_SOURCE_SGIO_INPUT_DETECTION3	单个I/O输入检测3

EVIC_SOURCE_GRP_E_INPUT_DETECTION	E组I/O输入检测
EVIC_SOURCE_GRP_F_INPUT_DETECTION	F组I/O输入检测
EVIC_SOURCE_TIMER0_OVERFLOW	TIMER0上溢事件
EVIC_SOURCE_TIMER0_UNDERFLOW	TIMER0下溢事件
EVIC_SOURCE_TIMER0_CH0_COMPARE	TIMER0_CH0比较事件
EVIC_SOURCE_TIMER0_CH1_COMPARE	TIMER0_CH1比较事件
EVIC_SOURCE_TIMER0_CH2_COMPARE	TIMER0_CH2比较事件
EVIC_SOURCE_TIMER0_CH3_COMPARE	TIMER0_CH3比较事件
EVIC_SOURCE_TIMER0_MCH0_COMPARE	TIMER0_MCH0比较事件
EVIC_SOURCE_TIMER0_MCH1_COMPARE	TIMER0_MCH1比较事件
EVIC_SOURCE_TIMER0_MCH2_COMPARE	TIMER0_MCH2比较事件
EVIC_SOURCE_TIMER0_MCH3_COMPARE	TIMER0_MCH3比较事件
EVIC_SOURCE_TIMER0_TRGO	TIMER0 TRGO信号
EVIC_SOURCE_TIMER7_OVERFLOW	TIMER7上溢事件
EVIC_SOURCE_TIMER7_UNDERFLOW	TIMER7下溢事件
EVIC_SOURCE_TIMER7_CH0_COMPARE	TIMER7_CH0比较事件
EVIC_SOURCE_TIMER7_CH1_COMPARE	TIMER7_CH1比较事件
EVIC_SOURCE_TIMER7_CH2_COMPARE	TIMER7_CH2比较事件
EVIC_SOURCE_TIMER7_CH3_COMPARE	TIMER7_CH3比较事件
EVIC_SOURCE_TIMER7_MCH0_COMPARE	TIMER7_MCH0比较事件
EVIC_SOURCE_TIMER7_MCH1_COMPARE	TIMER7_MCH1比较事件



EVIC_SOURCE_TIMER7_MCH2_COMPARE	TIMER7_MCH2比较事件
EVIC_SOURCE_TIMER7_MCH3_COMPARE	TIMER7_MCH3比较事件
EVIC_SOURCE_TIMER7_TRGO	TIMER7 TRGO信号
EVIC_SOURCE_TIMER1_CH0_COMPARE	TIMER1_CH0比较事件
EVIC_SOURCE_TIMER1_CH1_COMPARE	TIMER1_CH1比较事件
EVIC_SOURCE_TIMER1_CH2_COMPARE	TIMER1_CH2比较事件
EVIC_SOURCE_TIMER1_CH3_COMPARE	TIMER1_CH3比较事件
EVIC_SOURCE_TIMER1_OVERFLOW	TIMER1上溢事件
EVIC_SOURCE_TIMER1_UNDERFLOW	TIMER1下溢事件
EVIC_SOURCE_TIMER1_TRGO	TIMER1 TRGO信号
EVIC_SOURCE_TIMER2_CH0_COMPARE	TIMER2_CH0比较事件
EVIC_SOURCE_TIMER2_CH1_COMPARE	TIMER2_CH1比较事件
EVIC_SOURCE_TIMER2_CH2_COMPARE	TIMER2_CH2比较事件
EVIC_SOURCE_TIMER2_CH3_COMPARE	TIMER2_CH3比较事件
EVIC_SOURCE_TIMER2_OVERFLOW	TIMER2上溢事件
EVIC_SOURCE_TIMER2_UNDERFLOW	TIMER2下溢事件
EVIC_SOURCE_TIMER2_TRGO	TIMER2 TRGO信号
EVIC_SOURCE_CPTIMER0_COUNTER0_OVERFLOW	CPTIMER0计数器0上溢事件
EVIC_SOURCE_CPTIMERW_OCH0_COMPARE_MATCH	CPTIMERW_OCH0比较匹配事件
EVIC_SOURCE_CPTIMERW_OCH1_COMPARE_MATCH	CPTIMERW_OCH1比较匹配事件
EVIC_SOURCE_CPTIMERW_OCH2_COMPARE_MATCH	CPTIMERW_OCH2比较匹配事件

EVIC_SOURCE_CPTIMERW_O CH3_COMPARE_MATCH	CPTIMERW_OCH3比较匹配事件
EVIC_SOURCE_GPTIMER0_C H0_COMPARE_MATCH	GPTIMER0_CH0比较匹配事件
EVIC_SOURCE_GPTIMER0_C H1_COMPARE_MATCH	GPTIMER0_CH1比较匹配事件
EVIC_SOURCE_GPTIMER0_C H0_ADD_COMPARE_MATCH	GPTIMER0_CH0附加比较匹配事件
EVIC_SOURCE_GPTIMER0_C H1_ADD_COMPARE_MATCH	GPTIMER0_CH1附加比较匹配事件
EVIC_SOURCE_GPTIMER0_O VERFLOW	GPTIMER0上溢事件
EVIC_SOURCE_GPTIMER0_U NDERFLOW	GPTIMER0下溢事件
EVIC_SOURCE_GPTIMER0_A DTCV1_MATCH	GPTIMER0 ADTCV1匹配产生的ADC触发事件
EVIC_SOURCE_GPTIMER0_A DTCV2_MATCH	GPTIMER0 ADTCV2匹配产生的ADC触发事件
EVIC_SOURCE_GPTIMER0_R EPEAT_COUNT_END	GPTIMER0重复计数完成事件
EVIC_SOURCE_CMP0_OUT	CMP0输出
EVIC_SOURCE_CMP1_OUT	CMP1输出
EVIC_SOURCE_CMP2_OUT	CMP2输出
EVIC_SOURCE_CMP3_OUT	CMP3输出
EVIC_SOURCE_ADC0_GRP_S CAN_COMPLETE	ADC0组扫描完成事件
EVIC_SOURCE_ADC2_GRP_S CAN_COMPLETE	ADC2组扫描完成事件
EVIC_SOURCE_LVD1_VOLTA GE_DETECTION	LVD1电压检测
EVIC_SOURCE_LVD2_VOLTA GE_DETECTION	LVD2电压检测
EVIC_SOURCE_FWDGT_UND ERFLOW_OR_REFRESH	FWDGT上溢或者刷新
EVIC_SOURCE_HXTAL_STUC K	HXTAL卡住事件
EVIC_SOURCE_DMA0_CH0_T RANSFER_FINISH	DMA0通道0传输完成
EVIC_SOURCE_DMA0_CH1_T RANSFER_FINISH	DMA0通道1传输完成
EVIC_SOURCE_DMA0_CH2_T RANSFER_FINISH	DMA0通道2传输完成

EVIC_SOURCE_DMA0_CH3_TRANSFER_FINISH	DMA0通道3传输完成
EVIC_SOURCE_DMA0_CH4_TRANSFER_FINISH	DMA0通道4传输完成
EVIC_SOURCE_DMA0_CH5_TRANSFER_FINISH	DMA0通道5传输完成
EVIC_SOURCE_DMA1_CH0_TRANSFER_FINISH	DMA1通道0传输完成
EVIC_SOURCE_DMA1_CH1_TRANSFER_FINISH	DMA1通道1传输完成
EVIC_SOURCE_DMA1_CH2_TRANSFER_FINISH	DMA1通道2传输完成
EVIC_SOURCE_DMA1_CH3_TRANSFER_FINISH	DMA1通道3传输完成
EVIC_SOURCE_DMA1_CH4_TRANSFER_FINISH	DMA1通道4传输完成
EVIC_SOURCE_DMA1_CH5_TRANSFER_FINISH	DMA1通道5传输完成
EVIC_SOURCE_CAN_TRANSMIT_INT	CAN发送中断事件
EVIC_SOURCE_CAN_FIFO0_INTERRUPT	CAN FIFO0中断事件
EVIC_SOURCE_CAN_FIFO1_INTERRUPT	CAN FIFO1中断事件
EVIC_SOURCE_CAN_EWMC_INTERRUPT	CAN EMMC中断事件
EVIC_SOURCE_UART0_ERROR	USART0错误(PERR /FERR /NERR /ORERR)
EVIC_SOURCE_UART0_TRANSMIT_COMPLETE	USART0发送结束
EVIC_SOURCE_UART1_ERROR	USART1错误(PERR /FERR /NERR /ORERR)
EVIC_SOURCE_UART1_TRANSMIT_COMPLETE	USART1发送结束
EVIC_SOURCE_UART2_ERROR	USART2错误(PERR /FERR /NERR /ORERR)
EVIC_SOURCE_UART2_TRANSMIT_COMPLETE	USART2发送结束
EVIC_SOURCE_UART3_ERROR	USART3错误(PERR /FERR /NERR /ORERR)
EVIC_SOURCE_UART3_TRANSMIT_COMPLETE	USART3发送结束

EVIC_SOURCE_SPI_ERROR	SPI错误
EVIC_SOURCE_SPI_COMMUNICATION_COMPLETE	SPI传输完成
EVIC_SOURCE_I2C_COMMUNICATION_ERROR_OR_EVENT	I2C通信错误/通信事件
EVIC_SOURCE_I2C_RECEIVE_DATA_NOT_EMPTY	在接收期间I2C_RDATA非空
EVIC_SOURCE_I2C_TRANSMIT_INT	I2C发送中断
EVIC_SOURCE_I2C_TRANSFER_COMPLETE	I2C传输完成
EVIC_SOURCE_GPTIMER1_CH0_COMPARE_MATCH	GPTIMER1_CH0比较匹配事件
EVIC_SOURCE_GPTIMER0_UNDERFLOW	GPTIMER1_CH1比较匹配事件
EVIC_SOURCE_GPTIMER0_ADTCV1_MATCH	GPTIMER1_CH0附加比较匹配事件
EVIC_SOURCE_GPTIMER0_ADTCV2_MATCH	GPTIMER1_CH1附加比较匹配事件
EVIC_SOURCE_GPTIMER0_REPEAT_COUNT_END	GPTIMER1上溢事件
EVIC_SOURCE_CMP0_OUT	GPTIMER1下溢事件
EVIC_SOURCE_CMP1_OUT	GPTIMER1 ADTCV1匹配产生的ADC触发事件
EVIC_SOURCE_CMP2_OUT	GPTIMER1 ADTCV2匹配产生的ADC触发事件
EVIC_SOURCE_CMP3_OUT	GPTIMER1重复计数完成事件

### 枚举类型 **evic\_periph\_enum**

表 3-334. 枚举类型 **evic\_periph\_enum**

成员名称	功能描述
EVENT_INTERCONNECT_SGIO0	单I/O 0
EVENT_INTERCONNECT_SGIO1	单I/O 1
EVENT_INTERCONNECT_SGIO2	单I/O 2
EVENT_INTERCONNECT_SGIO3	单I/O 3
EVENT_INTERCONNECT INGRPE	输入组E
EVENT_INTERCONNECT INGRPF	输入组F
EVENT_INTERCONNECT_OUTGRPE	输出组E
EVENT_INTERCONNECT_OUTGRPF	输出组F
EVENT_INTERCONNECT_DAC0_OUT0_COV	DAC0_OUT0转换
EVENT_INTERCONNECT_DAC0_OUT1_COV	DAC0_OUT1转换

成员名称	功能描述
EVENT_INTERCONNECT_ADC0_GRP_EXTRIG0	ADC0组外部触发0
EVENT_INTERCONNECT_ADC0_GRP_EXTRIG1	ADC0组外部触发1
EVENT_INTERCONNECT_ADC2_GRP_EXTRIG0	ADC2组外部触发0
EVENT_INTERCONNECT_ADC2_GRP_EXTRIG1	ADC2组外部触发1
EVENT_INTERCONNECT_RCU	时钟切换至IRC32M
EVENT_INTERCONNECT_TIMER0	TIMER0
EVENT_INTERCONNECT_TIMER7	TIMER7
EVENT_INTERCONNECT_CPTIMER0	CPTIMER0
EVENT_INTERCONNECT_CPTIMERW	CPTIMERW
EVENT_INTERCONNECT_TIMER1	TIMER1
EVENT_INTERCONNECT_TIMER2	TIMER2
EVENT_INTERCONNECT_GPTIMER_T_RIGIN0	GPTIMER触发输入0
EVENT_INTERCONNECT_GPTIMER_T_RIGIN1	GPTIMER触发输入1
EVENT_INTERCONNECT_GPTIMER_T_RIGIN2	GPTIMER触发输入2
EVENT_INTERCONNECT_GPTIMER_T_RIGIN3	GPTIMER触发输入3
EVENT_INTERCONNECT_GPTIMER_T_RIGIN4	GPTIMER触发输入4
EVENT_INTERCONNECT_GPTIMER_T_RIGIN5	GPTIMER触发输入5
EVENT_INTERCONNECT_GPTIMER_T_RIGIN6	GPTIMER触发输入6
EVENT_INTERCONNECT_GPTIMER_T_RIGIN7	GPTIMER触发输入7
EVENT_INTERCONNECT_DAC0_OUT0_EN	DAC0_OUT0使能
EVENT_INTERCONNECT_DAC0_OUT1_EN	DAC0_OUT1使能
EVENT_INTERCONNECT_DMAMUX_0	DMAMUX输入0
EVENT_INTERCONNECT_DMAMUX_1	DMAMUX输入1
EVENT_INTERCONNECT_DMAMUX_2	DMAMUX输入2
EVENT_INTERCONNECT_DMAMUX_3	DMAMUX输入3
EVENT_INTERCONNECT_DMAMUX_4	DMAMUX输入4
EVENT_INTERCONNECT_DMAMUX_5	DMAMUX输入5

成员名称	功能描述
EVENT_INTERCONNECT_DMAMUX_6	DMAMUX输入6
EVENT_INTERCONNECT_DMAMUX_7	DMAMUX输入7
EVENT_INTERCONNECT_DMAMUX_8	DMAMUX输入8
EVENT_INTERCONNECT_DMAMUX_9	DMAMUX输入9
EVENT_INTERCONNECT_DMAMUX_10	DMAMUX输入10
EVENT_INTERCONNECT_DMAMUX_11	DMAMUX输入11

### 枚举类型 `evic_cptimer_enum`

表 3-335. 枚举类型 `evic_cptimer_enum`

成员名称	功能描述
TARGET_CPTIMERW	CPTIMERW
TARGET_CPTIMER0	CPTIMER0

### 枚举类型 `evic_single_io_enum`

表 3-336. 枚举类型 `evic_single_io_enum`

成员名称	功能描述
SGIO0	单I/O实例0
SGIO1	单I/O实例1
SGIO2	单I/O实例2
SGIO3	单I/O实例3

### 函数 `evic_init`

函数`evic_init`描述见下表：

表 3-337. 函数 `evic_init`

函数名称	<code>evic_init</code>
函数原型	<code>void evic_init(evic_periph_enum target_periph, event_source_enum event_source);</code>
功能描述	为目标外设设置事件源
先决条件	-
被调用函数	-
输入参数{in}	
<code>target_periph</code>	参考枚举类型 <a href="#">表 3-334. 枚举类型 <code>evic_periph_enum</code></a>
输入参数{in}	
<code>event_source</code>	参考枚举类型 <a href="#">表 3-333. 枚举类型 <code>event_source_enum</code></a>
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* set the software event for TIMER0 */  
  
evic_init(EVIC_SOURCE_SOFTWARE, EVENT_INTERCONNECT_TIMER0);
```

### 函数 `evic_event_source_get`

函数 `evic_event_source_get` 描述见下表：

表 3-338. 函数 `evic_event_source_get`

函数名称	<code>evic_event_source_get</code>
函数原型	<code>event_source_enum evic_event_source_get(evic_periph_enum target_periph);</code>
功能描述	获得目标外设的触发信号源
先决条件	-
被调用函数	-
输入参数{in}	
<code>target_periph</code>	参考枚举类型 <a href="#">表 3-334. 枚举类型 <code>evic_periph_enum</code></a>
输出参数{out}	
-	-
返回值	
<code>event_source</code>	参考枚举类型 <a href="#">表 3-333. 枚举类型 <code>event_source_enum</code></a>

例如：

```
/* get the trigger input signal for TIMER0 */  
  
event_source_enum event_source;  
  
event_source = evic_event_source_get(EVENT_INTERCONNECT_TIMER0);
```

### 函数 `evic_register_lock_set`

函数 `evic_register_lock_set` 描述见下表：

表 3-339. 函数 `evic_register_lock_set`

函数名称	<code>evic_register_lock_set</code>
函数原型	<code>void evic_register_lock_set(evic_periph_enum target_periph);</code>
功能描述	锁定事件互连寄存器
先决条件	-
被调用函数	-
输入参数{in}	
<code>target_periph</code>	参考枚举类型 <a href="#">表 3-334. 枚举类型 <code>evic_periph_enum</code></a>
输出参数{out}	
-	-
返回值	

-	-
---	---

例如:

```
/* lock the event interconnect for TIMER0 register */
evic_register_lock_set(EVENT_INTERCONNECT_TIMER0);
```

### 函数 evic\_register\_lock\_get

函数evic\_register\_lock\_get描述见下表:

表 3-340. 函数 evic\_register\_lock\_get

函数名称	evic_register_lock_get
函数原型	FlagStatus evic_register_lock_get(evic_periph_enum target_periph);
功能描述	获得事件互连寄存器锁定状态
先决条件	-
被调用函数	-
输入参数{in}	
target_periph	参考枚举类型 <a href="#">表 3-334. 枚举类型evic_periph_enum</a>
输出参数{out}	
-	-
返回值	
FlagStatus	SET 或者 RESET

例如:

```
/* get the event interconnect for TIMER0 register lock status */
FlagStatus status;
status = evic_register_lock_get(EVENT_INTERCONNECT_TIMER0);
```

### 函数 evic\_cptimer\_slave\_mode\_select

函数evic\_cptimer\_slave\_mode\_select描述见下表:

表 3-341. 函数 evic\_cptimer\_slave\_mode\_select

函数名称	evic_cptimer_slave_mode_select
函数原型	void evic_cptimer_slave_mode_select(evic_cptimer_enum cptimer_periph, uint32_t slavemode);
功能描述	比较匹配定时器从模式选择
先决条件	-
被调用函数	-
输入参数{in}	
cptimer_periph	参考枚举类型 <a href="#">表 3-335. 枚举类型evic_cptimer_enum</a>
输入参数{in}	



<b>slavemode</b>	从模式选择
<i>EVIC_COUNTER_ENABLE</i>	计数器使能
<i>EVIC_COUNTER_RESET</i>	计数器重新开始
<i>EVIC_EVENT_COUNT</i>	事件计数
<i>EVIC_EVENT_TRIGGER_DISABLE</i>	禁能事件触发
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* select event count as CPTIMERW slave mode */
```

```
evic_cptimer_slave_mode_select(TARGET_CPTIMERW, EVIC_EVENT_COUNT);
```

### 函数 **evic\_group\_member\_config**

函数 **evic\_group\_member\_config** 描述见下表:

**表 3-342. 函数 **evic\_group\_member\_config****

<b>函数名称</b>	<b>evic_group_member_config</b>
<b>函数原型</b>	<code>void evic_group_member_config(uint32_t group_pin, uint32_t group_port);</code>
<b>功能描述</b>	配置I/O组成员
<b>先决条件</b>	-
<b>被调用函数</b>	-
输入参数{in}	
<b>group_pin</b>	组I/O引脚
<i>EVIC_GRP_GPIO_PIN_x</i>	引脚 x (x=8..14)
输入参数{in}	
<b>group_port</b>	组端口
<i>EVIC_GRP_GPIO_PORT_E</i>	端口E
<i>EVIC_GRP_GPIO_PORT_F</i>	端口F
输入参数{in}	
<b>newvalue</b>	ENABLE 或者 DISABLE
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* configure evic group E member */
```

```
evic_group_member_config(EVIC_GRP_GPIO_PIN_8|EVIC_GRP_GPIO_PIN_9, EVIC_GRP_GPIO_PORT_E, ENABLE);
```

### 函数 `evic_group_edge_detection_config`

函数 `evic_group_edge_detection_config` 描述见下表：

表 3-343. 函数 `evic_group_edge_detection_config`

函数名称	<code>evic_group_edge_detection_config</code>
函数原型	<code>void evic_group_edge_detection_config(uint32_t group_edge, uint32_t group_port);</code>
功能描述	配置I/O组的边沿检测
先决条件	-
被调用函数	-
输入参数{in}	
<b>group_edge</b>	组I/O输入边沿检测
<code>EVIC_GRP_GPIO_DETECT_ION_RISING</code>	组I/O输入上升沿产生事件信号
<code>EVIC_GRP_GPIO_DETECT_ION_FALLING</code>	组I/O输入下降沿产生事件信号
<code>EVIC_GRP_GPIO_DETECT_ION_BOTH_EDGE</code>	组I/O输入双边沿产生事件信号
输入参数{in}	
<b>group_port</b>	组端口
<code>EVIC_GRP_GPIO_PORT_E</code>	端口E
<code>EVIC_GRP_GPIO_PORT_F</code>	端口F
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* configure evic group E input detection edge */
```

```
evic_group_member_config(EVIC_GRP_GPIO_DETECTION_RISING, EVIC_GRP_GPIO_PORT_E);
```

### 函数 `evic_group_output_level_config`

函数 `evic_group_output_level_config` 描述见下表：

表 3-344. 函数 `evic_group_output_level_config`

函数名称	<code>evic_group_output_level_config</code>
函数原型	<code>void evic_group_output_level_config(uint32_t group_level, uint32_t</code>

	group_port);
功能描述	配置I/O组的输出电平
先决条件	-
被调用函数	-
输入参数{in}	
group_level	组I/O输出电平
EVIC_GRP_GPIO_OUTPUTPU T_LOW	当事件信号到来, 组I/O输出低电平
EVIC_GRP_GPIO_OUTPUTPU T_HIGH	当事件信号到来, 组I/O输出高电平
EVIC_GRP_GPIO_OUTPUTPU T_INVERTED	当事件信号到来, 组I/O输出相反电平
EVIC_GRP_GPIO_OUTPUTPU T_DATA	当事件信号到来, 组I/O输出保持数据
EVIC_GRP_GPIO_OUTPUTPU T_CIRCULAR_DATA	当事件信号到来, 组I/O输出循环右移的保持数据
输入参数{in}	
group_port	组端口
EVIC_GRP_GPIO_PORT_E	端口E
EVIC_GRP_GPIO_PORT_F	端口F
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* configure evic group E output level */
```

```
evic_group_output_level_config(EVIC_GRP_GPIO_OUTPUT_LOW,EVIC_GRP_GPIO_PORT_E);
```

### 函数 evic\_group\_overwrite\_enable

函数evic\_group\_overwrite\_enable描述见下表:

表 3-345. 函数 evic\_group\_overwrite\_enable

函数名称	evic_group_overwrite_enable
函数原型	void evic_group_overwrite_enable(uint32_t group_port);
功能描述	使能I/O组覆写功能
先决条件	-
被调用函数	-
输入参数{in}	
group_port	组端口
EVIC_GRP_GPIO_PORT_E	端口E

EVIC_GRP_GPIO_PORT_F	端口F
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* enable evic group E overwrite function */
```

```
evic_group_overwrite_enable(EVIC_GRP_GPIO_PORT_E);
```

### 函数 **evic\_group\_overwrite\_disable**

函数 **evic\_group\_overwrite\_disable** 描述见下表：

**表 3-346. 函数 **evic\_group\_overwrite\_disable****

函数名称	evic_group_overwrite_disable
函数原型	void evic_group_overwrite_disable (uint32_t group_port);
功能描述	禁能I/O组覆写功能
先决条件	-
被调用函数	-
输入参数{in}	
group_port	组端口
EVIC_GRP_GPIO_PORT_E	端口E
EVIC_GRP_GPIO_PORT_F	端口F
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* disable evic group E overwrite function */
```

```
evic_group_overwrite_disable(EVIC_GRP_GPIO_PORT_E);
```

### 函数 **evic\_data\_set**

函数 **evic\_data\_set** 描述见下表：

**表 3-347. 函数 **evic\_data\_set****

函数名称	evic_data_set
函数原型	void evic_data_set(uint8_t data, uint32_t group_port);
功能描述	EVIC保持数据设置
先决条件	-
被调用函数	-

输入参数{in}	
<b>data</b>	加载数据 （0~127）
输入参数{in}	
<b>group_port</b>	组端口
<i>EVIC_GRPIO_PORT_E</i>	端口E
<i>EVIC_GRPIO_PORT_F</i>	端口F
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* set group E holding data value */
evic_data_set(0xF, EVIC_GRPIO_PORT_E);
```

### 函数 **evic\_data\_get**

函数 **evic\_data\_get** 描述见下表：

**表 3-348. 函数 **evic\_data\_get****

<b>函数名称</b>	<b>evic_data_get</b>
<b>函数原型</b>	uint8_t evic_data_get(uint32_t group_port);
<b>功能描述</b>	获得EVIC保持数据
<b>先决条件</b>	-
<b>被调用函数</b>	-
输入参数{in}	
<b>group_port</b>	组端口
<i>EVIC_GRPIO_PORT_E</i>	端口E
<i>EVIC_GRPIO_PORT_F</i>	端口F
输出参数{out}	
-	-
返回值	
<b>holding data</b>	0x00~0x7F

例如：

```
/* get group E holding data value */
uint8_t data;
data = evic_data_get(EVIC_GRPIO_PORT_E);
```

### 函数 **evic\_single\_io\_config**

函数 **evic\_single\_io\_config** 描述见下表：

表 3-349. 函数 `evic_single_io_config`

函数名称	<code>evic_single_io_config</code>
函数原型	<code>void evic_single_io_config(uint32_t single_pin, uint32_t single_port, evic_single_io_enum single_io);</code>
功能描述	配置单I/O
先决条件	-
被调用函数	-
输入参数{in}	
<code>single_pin</code>	单I/O引脚
<code>EVIC_SGIO_PIN_x</code>	引脚 x (x=8..14)
输入参数{in}	
<code>group_port</code>	组端口
<code>EVIC_SGIO_PORT_E</code>	端口E
<code>EVIC_SGIO_PORT_F</code>	端口F
输入参数{in}	
<code>single_io</code>	单I/O实例
<code>SGIOx</code>	单I/O实例x (x=0..3)
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* configure evic single I/O instance 0 */
```

```
evic_single_io_config(EVIC_SGIO_PIN_1, EVIC_SGIO_PORT_F, SGIO0);
```

### 函数 `evic_single_io_edge_detection_config`

函数`evic_single_io_edge_detection_config`描述见下表：

表 3-350. 函数 `evic_single_io_config`

函数名称	<code>evic_single_io_edge_detection_config</code>
函数原型	<code>void evic_single_io_edge_detection_config(uint32_t single_edge, evic_single_io_enum single_io);</code>
功能描述	配置单I/O的边沿检测
先决条件	-
被调用函数	-
输入参数{in}	
<code>single_edge</code>	单I/O的输入检测边沿
<code>EVIC_SGIO_DETECTI ON_RISING</code>	单I/O输入上升沿产生事件信号
<code>EVIC_SGIO_DETECTI</code>	单I/O输入下降沿产生事件信号

ON_FALLING	
EVIC_SGIO_DETECTI ON_BOTH_EDGE	单I/O输入双边沿产生事件信号
输入参数{in}	
single_io	单I/O实例
SGIOx	单I/O实例x (x=0..3)
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* configure evic single I/O instance 0 input detection edge */
```

```
evic_single_io_edge_detection_config(EVIC_SGIO_DETECTION_RISING, SGIO0);
```

### 函数 evic\_single\_io\_output\_level\_config

函数evic\_single\_io\_output\_level\_config描述见下表:

表 3-351. 函数 evic\_single\_io\_output\_level\_config

函数名称	evic_single_io_output_level_config
函数原型	void evic_single_io_output_level_config(uint32_t single_level, evic_single_io_enum single_io);
功能描述	配置单I/O的输出电平
先决条件	-
被调用函数	-
输入参数{in}	
single_level	单I/O输出电平
EVIC_SGIO_OUTPUT_ LOW	当事件信号到来, 单I/O输出低电平
EVIC_SGIO_OUTPUT_ HIGH	当事件信号到来, 单I/O输出高电平
EVIC_SGIO_OUTPUT_ INVERTED	当事件信号到来, 单I/O输出相反电平
输入参数{in}	
single_io	单I/O实例
SGIOx	单I/O实例x (x=0..3)
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* configure evic single I/O instance 0 output level */
```

```
evic_single_io_output_level_config(EVIC_SGIO_OUTPUT_LOW, SGIO0);
```

### 函数 `evic_register_write_enable`

函数 `evic_register_write_enable` 描述见下表：

表 3-352. 函数 `evic_register_write_enable`

函数名称	<code>evic_register_write_enable</code>
函数原型	<code>void evic_register_write_enable(void);</code>
功能描述	使能软件事件寄存器写
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 software event register write */
```

```
void evic_register_write_enable(void);
```

### 函数 `evic_register_write_disable`

函数 `evic_register_write_disable` 描述见下表：

表 3-353. 函数 `evic_register_write_disable`

函数名称	<code>evic_register_write_disable</code>
函数原型	<code>void evic_register_write_disable(void);</code>
功能描述	禁能软件事件寄存器写
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 software event register write */
```



```
void evic_register_write_disable(void);
```

### 函数 `evic_register_write_enable_get`

函数 `evic_register_write_enable_get` 描述见下表：

表 3-354. 函数 `evic_register_write_enable_get`

函数名称	<code>evic_register_write_enable_get</code>
函数原型	<code>FlagStatus evic_register_write_enable_get(void);</code>
功能描述	获取软件事件寄存器写使能状态
先决条件	-
被调用函数	-
输入参数{in}	
-	-
输出参数{out}	
-	-
返回值	
<b>FlagStatus</b>	SET 或者 RESET

例如：

```
/* get software event register write enable status */
```

```
FlagStatus status;
```

```
status = evic_register_write_enable_get(void);
```

### 函数 `evic_bit_write_enable`

函数 `evic_bit_write_enable` 描述见下表：

表 3-355. 函数 `evic_bit_write_enable`

函数名称	<code>evic_bit_write_enable</code>
函数原型	<code>void evic_bit_write_enable(void);</code>
功能描述	使能软件事件寄存器位写
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 software event generation bit write */
```

```
void evic_bit_write_enable(void);
```

### 函数 `evic_bit_write_disable`

函数 `evic_bit_write_disable` 描述见下表：

表 3-356. 函数 `evic_bit_write_disable`

函数名称	<code>evic_bit_write_disable</code>
函数原型	<code>void evic_bit_write_disable(void);</code>
功能描述	禁用软件事件寄存器位写
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 software event generation bit write */
```

```
void evic_bit_write_disable(void);
```

### 函数 `evic_bit_write_enable_get`

函数 `evic_bit_write_enable_get` 描述见下表：

表 3-357. 函数 `evic_bit_write_enable_get`

函数名称	<code>evic_bit_write_enable_get</code>
函数原型	<code>FlagStatus evic_bit_write_enable_get(void);</code>
功能描述	获取软件事件寄存器位写使能状态
先决条件	-
被调用函数	-
输入参数{in}	
-	-
输出参数{out}	
-	-
返回值	
<b>FlagStatus</b>	SET 或者 RESET

例如：

```
/* get software event generation bit write enable status */
```

```
FlagStatus status;
```

```
status = evic_bit_write_enable_get(void);
```

### 函数 `evic_software_event_generation`

函数 `evic_software_event_generation` 描述见下表：

**表 3-358. 函数 `evic_software_event_generation`**

函数名称	<code>evic_software_event_generation</code>
函数原型	<code>void evic_software_event_generation(void);</code>
功能描述	生成EVIC软件事件
先决条件	-
被调用函数	-
输入参数{in}	
-	-
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* evic software event generation */
void evic_software_event_generation(void);
```

## 3.13. EXTI

EXTI是MCU中的中断/事件控制器，包括25个相互独立的边沿检测电路并且能够向处理器内核产生中断请求或唤醒事件。章节[3.13.1](#)描述了EXTI的寄存器列表，章节[3.13.2](#)对EXTI库函数进行说明。

### 3.13.1. 外设寄存器说明

EXTI寄存器列表如下表所示：

**表 3-359. 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
EXTI_DFEN	数字滤波使能寄存器

寄存器名称	寄存器描述
EXTI_SCS	采样时钟选择寄存器

### 3.13.2. 外设库函数说明

EXTI库函数列表如下表所示：

表 3-360. EXTI 库函数

库函数名称	库函数描述
exti_deinit	复位EXTI
exti_init	初始化EXTI线x
exti_interrupt_enable	EXTI线x中断使能
exti_interrupt_disable	EXTI线x中断禁能
exti_event_enable	EXTI线x事件使能
exti_event_disable	EXTI线x事件禁能
exti_software_interrupt_enable	EXTI线x软件中断事件使能
exti_software_interrupt_disable	EXTI线x软件中断事件禁能
exti_digital_filter_enable	EXTI线x数字滤波使能
exti_digital_filter_disable	EXTI线x数字滤波禁能
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 3-361. 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

枚举名称	枚举描述
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

### 枚举类型 `exti_mode_enum`

表 3-362. 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 3-363. 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 3-364. 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 3-365. 函数 exti\_init**

函数名称	exti_init
函数原型	void exti_init(exti_line_enum linex, exti_mode_enum mode, exti_trig_type_enum trig_type);
功能描述	初始化EXTI线x
先决条件	-
被调用函数	-
输入参数{in}	
linex	EXTI线x, 参考 <a href="#">表3-361. 枚举类型exti_line_enum</a>
输入参数{in}	
mode	EXTI模式, 参考 <a href="#">表3-362. 枚举类型exti_mode_enum</a>
输入参数{in}	
trig_type	触发类型, 参考 <a href="#">表3-363. 枚举类型exti_trig_type_enum</a>
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 3-366. 函数 exti\_interrupt\_enable**

函数名称	exti_interrupt_enable
函数原型	void exti_interrupt_enable(exti_line_enum linex);
功能描述	EXTI线x中断使能
先决条件	-
被调用函数	-
输入参数{in}	

linex	EXTI线x, 参考 <a href="#">表3-361. 枚举类型exti_line_enum</a>
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 3-367. 函数 exti\_interrupt\_disable**

函数名称	exti_interrupt_disable
函数原型	void exti_interrupt_disable(exti_line_enum linex);
功能描述	EXTI线x中断禁能
先决条件	-
被调用函数	-
输入参数{in}	
linex	EXTI线x, 参考 <a href="#">表3-361. 枚举类型exti_line_enum</a>
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 3-368. 函数 exti\_event\_enable**

函数名称	exti_event_enable
函数原型	void exti_event_enable(exti_line_enum linex);
功能描述	EXTI线x事件使能
先决条件	-
被调用函数	-
输入参数{in}	
linex	EXTI线x, 参考 <a href="#">表3-361. 枚举类型exti_line_enum</a>

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 3-369. 函数 exti\_event\_disable

函数名称	exti_event_disable
函数原型	void exti_event_disable(exti_line_enum linex);
功能描述	EXTI线x事件禁能
先决条件	-
被调用函数	-
输入参数{in}	
linex	EXTI线x，参考 <a href="#">表3-361. 枚举类型exti_line_enum</a>
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 3-370. 函数 exti\_software\_interrupt\_enable

函数名称	exti_software_interrupt_enable
函数原型	void exti_software_interrupt_enable(exti_line_enum linex);
功能描述	EXTI线x软件中断事件使能
先决条件	-
被调用函数	-
输入参数{in}	
linex	EXTI线x，参考 <a href="#">表3-361. 枚举类型exti_line_enum</a>
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 3-371. 函数 exti\_software\_interrupt\_disable

函数名称	exti_software_interrupt_disable
函数原型	void exti_software_interrupt_disable(exti_line_enum linex);
功能描述	EXTI线x软件中断事件禁能
先决条件	-
被调用函数	-
输入参数{in}	
linex	EXTI线x，参考 <a href="#">表3-361. 枚举类型exti_line_enum</a>
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\_digital\_filter\_enable

函数exti\_digital\_filter\_enable描述见下表：

表 3-372. 函数 exti\_digital\_filter\_enable

函数名称	exti_digital_filter_enable
函数原型	void exti_digital_filter_enable(exti_line_enum linex, uint32_t pclk_div);
功能描述	EXTI数字滤波使能
先决条件	-
被调用函数	-
输入参数{in}	
linex	EXTI线x，参考 <a href="#">表3-361. 枚举类型exti_line_enum</a>
输入参数{in}	
pclk_div	选择采样时钟

EXTI_SAMPLING_P CLK_DIV1	采样时钟选择PCLK
EXTI_SAMPLING_P CLK_DIV8	采样时钟选择PCLK / 8
EXTI_SAMPLING_P CLK_DIV32	采样时钟选择PCLK / 32
EXTI_SAMPLING_P CLK_DIV64	采样时钟选择PCLK / 64
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* enable EXTI line 0 digital filter */
```

```
exti_digital_filter_enable(EXTI_0, EXTI_SAMPLING_PCLK_DIV1);
```

### 函数 exti\_digital\_filter\_disable

函数exti\_digital\_filter\_disable描述见下表：

表 3-373. 函数 exti\_digital\_filter\_disable

函数名称	exti_digital_filter_disable
函数原型	void exti_digital_filter_disable(exti_line_enum linex);
功能描述	EXTI线数字滤波禁能
先决条件	-
被调用函数	-
输入参数{in}	
linex	EXTI线x，参考 <a href="#">表3-361. 枚举类型exti_line_enum</a>
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* disable EXTI line 0 digital filter */
```

```
exti_digital_filter_disable(EXTI_0);
```

### 函数 exti\_flag\_get

函数exti\_flag\_get描述见下表：

表 3-374. 函数 exti\_flag\_get

函数名称	exti_flag_get
函数原型	FlagStatus exti_flag_get(exti_line_enum linex);
功能描述	获取EXTI线x中断标志位
先决条件	-
被调用函数	-
输入参数{in}	
linex	EXTI线x, 参考 <a href="#">表3-361. 枚举类型exti_line_enum</a>
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 3-375. 函数 exti\_flag\_clear

函数名称	exti_flag_clear
函数原型	void exti_flag_clear(exti_line_enum linex);
功能描述	清除EXTI线x中断标志位
先决条件	-
被调用函数	-
输入参数{in}	
linex	EXTI线x, 参考 <a href="#">表3-361. 枚举类型exti_line_enum</a>
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 3-376. 函数 exti\_interrupt\_flag\_get

函数名称	exti_interrupt_flag_get
函数原型	FlagStatus exti_interrupt_flag_get(exti_line_enum linex);
功能描述	获取EXTI线x中断标志位
先决条件	-
被调用函数	-
输入参数{in}	
linex	EXTI线x, 参考 <a href="#">表3-361. 枚举类型exti_line_enum</a>
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 3-377. 函数 exti\_interrupt\_flag\_clear

函数名称	exti_interrupt_flag_clear
函数原型	void exti_interrupt_flag_clear(exti_line_enum linex);
功能描述	清除EXTI线x中断标志位
先决条件	-
被调用函数	-
输入参数{in}	
linex	EXTI线x, 参考 <a href="#">表3-361. 枚举类型exti_line_enum</a>
输出参数{out}	
-	-
返回值	
-	-

例如:

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

exti_interrupt_flag_clear(EXTI_0);
```

## 3.14. FMC

闪存控制器 (FMC), 提供了片上闪存需要的所有功能。章节[3.14.1](#)描述了FMC的寄存器列表,

章节[3.14.2](#)对FMC库函数进行说明。

### 3.14.1. 外设寄存器说明

FMC寄存器列表如下：

**表 3-378. 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控制和状态寄存器
FMC_OBSTAT	选项字节状态寄存器
FMC_WP	擦写/编程保护寄存器
FMC_PID0	产品ID寄存器0
FMC_PID1	产品ID寄存器1

### 3.14.2. 外设库函数说明

FMC固件库函数列举如下表：

**表 3-379. FMC 固件库函数**

函数名称	函数描述
fmc_unlock	解锁FMC主闪存块及数据闪存块操作
fmc_lock	锁定FMC主闪存块及数据闪存块操作
fmc_wsnt_set	设置FMC等待状态计数值
fmc_prefetch_enable	使能pre-fetch
fmc_prefetch_disable	禁能pre-fetch
fmc_ibus_enable	使能IBUS缓存区
fmc_ibus_disable	禁能IBUS缓存区
fmc_dbus_enable	使能DBUS缓存区
fmc_dbus_disable	禁能DBUS缓存区
fmc_ibus_reset_enable	使能复位IBUS缓存区
fmc_ibus_reset_disable	禁能复位IBUS缓存区
fmc_dbus_reset_enable	使能复位DBUS缓存区
fmc_dbus_reset_disable	禁能复位DBUS缓存区
fmc_blank_check	通过查空命令检查flash页是否为空
fmc_page_erase	FMC页擦除
fmc_mflash_mass_erase	主闪存块擦除
fmc_dflash_mass_erase	数据闪存块擦除

函数名称	函数描述
fmc_mass_erase	FMC全片擦除
fmc_fourword_program	在相应地址4字编程
fmc_fast_program	在主编程块相应地址快速编程一行数据
fmc_otp_fourword_program	在OTP区域相应地址4字编程
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	写用户选项字节
ob_data_program	写数据选项字节
ob_user1_write	编程选项字节USER1
ob_wwdgt0_write	编程选项字节WWDGT0
ob_wwdgt1_wwdgt2_write	编程选项字节WWDGT1和WWDGT2
ob_fwdgt_write	编程选项字节FWDGT
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	获取写保护选项字节
ob_security_protection_flag_get	获取安全保护选项字节
fmc_ecc_error_address_get	获取FMC_ECCCS寄存器ECC错误地址
fmc_flag_get	检查标志位是否置位
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

### 结构体 optionbyte\_user\_struct

表 3-380. 结构体 optionbyte\_user\_struct

成员名称	功能描述
nfdwg_hw	选项字节 USER 段 nFWDG_HW (OB_FWDGT_HW, OB_FWDGT_SW)
nrst_dpslp	选项字节 USER 段 nRST_DPSLP (OB_DEEPSLEEP_RST, OB_DEEPSLEEP_NRST)
nrst_stdby	选项字节 USER 段 nRST_STDBY (OB_STDBY_RST, OB_STDBY_NRST)
lvd0en	选项字节 USER 段 LVD0EN (OB_LVDOEN_ENABLE, OB_LVDOEN_DISABLE)
fdgspd_dpslp	选项字节 USER 段 FWDGSPD_DPSLP (OB_FWDGDPSLP_ENABLE, OB_FWDGDPSLP_DISABLE)

成员名称	功能描述
fwdgspd_stdby	选项字节 USER 段 FWDGSPD_STDBY (OB_FWDGSTANDBY_ENABLE, OB_FWDGSTANDBY_DISABLE)
lvd0t	选项字节 USER 段 LVD0T (OB_LVD0T_VALUE0, OB_LVD0T_VALUE1, OB_LVD0T_VALUE2)

### 结构体 optionbyte\_wwdgt0\_struct

表 3-381. 结构体 optionbyte\_wwdgt0\_struct

成员名称	功能描述
psc	选项字节 WWDGT0 段 WWDGT 分频系数 (OB_WWDG_PSC8, OB_WWDG_PSC4, OB_WWDG_PSC2, OB_WWDG_PSC1)
wsp	选项字节 WWDGT0 段窗口启动位置 (OB_WSPS_VALUE25, OB_WSPS_VALUE50, OB_WSPS_VALUE75, OB_WSPS_VALUE100)
wep	选项字节 WWDGT0 段窗口结束位置 (OB_WEPS_VALUE75, OB_WEPS_VALUE50, OB_WEPS_VALUE25, OB_WEPS_VALUE0)
ewie	选项字节 WWDGT0 段早唤醒中断 (OB_EWIE_ENABLE, OB_EWIE_DISABLE)
wdga	选项字节 WWDGT0 段配置 WWDGT 使能/禁能产生系统复位 (OB_WWDGT_RST_DISABLE, OB_WWDGT_RST_ENABLE)

### 枚举类型 fmc\_state\_enum

表 3-382. 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_PGSERR	编程顺序错误
FMC_PGERR	编程错误
FMC_PGAERR	编程对齐错误
FMC_WPERR	写保护错误
FMC_TOERR	超时错误
FMC_CB CMDERR	被选中区域非空错误
FMC_OB_HSPC	高保护等级

### 枚举类型 fmc\_flag\_enum

表 3-383. 枚举类型 fmc\_flag\_enum

枚举名称	枚举描述
FMC_FLAG_BUSY	闪存忙标志

枚举名称	枚举描述
FMC_FLAG_PGSE RR	编程顺序错误标志
FMC_FLAG_PGER R	编程错误标志
FMC_FLAG_PGAE RR	编程对齐错误标志
FMC_FLAG_WPER R	擦写保护错误标志
FMC_FLAG_END	操作结束标志
FMC_FLAG_CBCM DERR	被查空命令选中的区域非空标志
FMC_FLAG_ECCC OR	单比特ECC错误标志
FMC_FLAG_ECCD ET	多比特ECC错误标志
FMC_FLAG_SYSE CC	系统内存ECC错误标志
FMC_FLAG_MFEC C	主闪存ECC错误标志
FMC_FLAG_OTPE CC	OTP ECC错误标志
FMC_FLAG_OBEC C	选项字节ECC错误标志
FMC_FLAG_DFEC C	数据闪存ECC错误标志
FMC_FLAG_SYSE CCBK	ECC错误发生在信息块的前3K或后3K区域
FMC_FLAG_OBER R	选项字节读取错误标志

#### 枚举类型 `fmc_interrupt_flag_enum`

表 3-384. 枚举类型 `fmc_interrupt_flag_enum`

枚举名称	枚举描述
FMC_INT_FLAG_P GSERR	闪存编程序列错误中断标志
FMC_INT_FLAG_P GERR	闪存编程错误中断标志
FMC_INT_FLAG_P GAERR	闪存编程对齐错误中断标志
FMC_INT_FLAG_W PERR	闪存擦写保护错误中断标志



枚举名称	枚举描述
FMC_INT_FLAG_END	闪存操作结束中断标志
FMC_INT_FLAG_CB_CMDERR	闪存被查空命令选中的区域非空中断标志
FMC_INT_FLAG_ECCOR	单比特ECC错误中断标志
FMC_INT_FLAG_ECCDET	多比特ECC错误中断标志

### 枚举类型 `fmc_interrupt_enum`

表 3-385. 枚举类型 `fmc_interrupt_enum`

枚举名称	枚举描述
FMC_INT_ERR	闪存错误中断
FMC_INT_END	闪存操作结束中断
FMC_INT_ECCDET	多比特ECC错误中断
FMC_INT_ECCCOR	单比特ECC错误中断

### 枚举类型 `fmc_ob_wp_enum`

表 3-386. 枚举类型 `fmc_ob_wp_enum`

枚举名称	枚举描述
OB_WP_NONE	禁用所有页擦写保护
OB_WP_0	主闪存0 ~ 7页擦写保护
OB_WP_1	主闪存8 ~ 15页擦写保护
OB_WP_2	主闪存16 ~ 23页擦写保护
OB_WP_3	主闪存24 ~ 31页擦写保护
OB_WP_4	主闪存32 ~ 39页擦写保护
OB_WP_5	主闪存40 ~ 47页擦写保护
OB_WP_6	主闪存48 ~ 55页擦写保护
OB_WP_7	主闪存56 ~ 63页擦写保护
OB_WP_8	主闪存64 ~ 71页擦写保护
OB_WP_9	主闪存72 ~ 79页擦写保护
OB_WP_10	主闪存80 ~ 87页擦写保护
OB_WP_11	主闪存88 ~ 95页擦写保护
OB_WP_12	主闪存96 ~ 103页擦写保护
OB_WP_13	主闪存104 ~ 111页擦写保护
OB_WP_14	主闪存112 ~ 119页擦写保护
OB_WP_15	主闪存120 ~ 127页擦写保护
OB_WP_16	主闪存128 ~ 135页擦写保护
OB_WP_17	主闪存136 ~ 143页擦写保护

枚举名称	枚举描述
OB_WP_18	主闪存144 ~ 151页擦写保护
OB_WP_19	主闪存152 ~ 159页擦写保护
OB_WP_20	主闪存160 ~ 167页擦写保护
OB_WP_21	主闪存168 ~ 175页擦写保护
OB_WP_22	主闪存176 ~ 183页擦写保护
OB_WP_23	主闪存184 ~ 256页擦写保护
OB_WP_24	数据闪存0 ~ 7页擦写保护
OB_WP_25	数据闪存8 ~ 15页擦写保护
OB_WP_26	数据闪存16 ~ 23页擦写保护
OB_WP_27	数据闪存24 ~ 31页擦写保护
OB_WP_28	数据闪存32 ~ 39页擦写保护
OB_WP_29	数据闪存40 ~ 47页擦写保护
OB_WP_30	数据闪存48 ~ 55页擦写保护
OB_WP_31	数据闪存56 ~ 63页擦写保护
OB_WP_ALL	使能所有页擦写保护

### 函数 fmc\_unlock

函数fmc\_unlock描述见下表：

表 3-387. 函数 fmc\_unlock

函数名称	fmc_unlock
函数原型	void fmc_unlock(void);
功能描述	解锁主闪存和数据闪存操作
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 3-388. 函数 `fmc_lock`

函数名称	<code>fmc_lock</code>
函数原型	<code>void fmc_lock(void);</code>
功能描述	锁定主闪存和数据闪存操作
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 3-389. 函数 `fmc_wscnt_set`

函数名称	<code>fmc_wscnt_set</code>
函数原型	<code>void fmc_wscnt_set(uint32_t wscnt);</code>
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}	
<b>wscnt</b>	等待状态计数值
<code>FMC_WAIT_STATE_0</code>	FMC 0个等待状态
<code>FMC_WAIT_STATE_1</code>	FMC 1个等待状态
<code>FMC_WAIT_STATE_2</code>	FMC 2个等待状态
<code>FMC_WAIT_STATE_3</code>	FMC 3个等待状态
<code>FMC_WAIT_STATE_4</code>	FMC 4个等待状态
<code>FMC_WAIT_STATE_5</code>	FMC 5个等待状态
<code>FMC_WAIT_STATE_6</code>	FMC 6个等待状态

_6	
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* set 1wait state */
```

```
fmc_wscnt_set(FMC_WAIT_STATE_1);
```

### 函数 fmc\_prefetch\_enable

函数fmc\_prefetch\_enable描述见下表：

表 3-390. 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 3-391. 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\_ibus\_enable

函数fmc\_ibus\_enable描述见下表：

表 3-392. 函数 fmc\_ibus\_enable

函数名称	fmc_ibus_enable
函数原型	void fmc_ibus_enable(void);
功能描述	使能IBUS缓存区
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 IBUS cache */
```

```
fmc_ibus_enable();
```

### 函数 fmc\_ibus\_disable

函数fmc\_ibus\_disable描述见下表：

表 3-393. 函数 fmc\_ibus\_disable

函数名称	fmc_ibus_disable
函数原型	void fmc_ibus_disable(void);
功能描述	禁能IBUS缓存区
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 IBUS cache */
```

```
fmc_ibus_disable();
```

### 函数 fmc\_ibus\_reset\_enable

函数fmc\_ibus\_reset\_enable描述见下表：

表 3-394. 函数 fmc\_ibus\_reset\_enable

函数名称	fmc_ibus_reset_enable
函数原型	void fmc_ibus_reset_enable (void);
功能描述	使能复位IBUS缓存区
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 reset IBUS cache */
```

```
fmc_ibus_reset_enable();
```

### 函数 fmc\_ibus\_reset\_disable

函数fmc\_ibus\_reset\_disable描述见下表：

表 3-395. 函数 fmc\_ibus\_reset\_disable

函数名称	fmc_ibus_reset_disable
函数原型	void fmc_ibus_reset_disable (void);
功能描述	禁能复位IBUS缓存区
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 reset IBUS cache */
```

```
fmc_ibus_reset_disable();
```

### 函数 fmc\_dbus\_enable

函数fmc\_dbus\_enable描述见下表：

**表 3-396. 函数 fmc\_dbus\_enable**

函数名称	fmc_dbus_enable
函数原型	void fmc_dbus_enable(void);
功能描述	使能DBUS缓存区
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 DBUS cache */
```

```
fmc_dbus_enable();
```

### 函数 fmc\_dbus\_disable

函数fmc\_dbus\_disable描述见下表：

**表 3-397. 函数 fmc\_dbus\_disable**

函数名称	fmc_dbus_disable
函数原型	void fmc_dbus_disable(void);
功能描述	禁能DBUS缓存区
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 DBUS cache */
```

```
fmc_dbus_disable();
```

### 函数 fmc\_dbus\_reset\_enable

函数fmc\_dbus\_reset\_enable描述见下表：

**表 3-398. 函数 fmc\_dbus\_reset\_enable**

函数名称	fmc_dbus_reset_enable
函数原型	void fmc_dbus_reset_enable(void);
功能描述	使能复位DBUS缓存区
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 reset DBUS cache */
```

```
fmc_dbus_reset_enable();
```

### 函数 fmc\_dbus\_reset\_disable

函数fmc\_dbus\_reset\_disable描述见下表：

**表 3-399. 函数 fmc\_dbus\_reset\_disable**

函数名称	fmc_dbus_reset_disable
函数原型	void fmc_dbus_reset_disable(void);
功能描述	禁能复位DBUS缓存区
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 reset DBUS cache */
```

```
fmc_dbus_reset_disable();
```

### 函数 fmc\_blank\_check

函数fmc\_blank\_check描述见下表：

表 3-400. 函数 fmc\_blank\_check

函数名称	fmc_blank_check
函数原形	fmc_state_enum fmc_blank_check(uint32_t address, uint8_t length);
功能描述	通过查空命令检查flash页是否为空
先决条件	-
被调用函数	-
输入参数{in}	
address	检查的起始地址
输入参数{in}	
length	读取长度为2 <sup>length</sup> 个双字，需要检查的flash区域必须在一个页面内，且不应超过1KB的边界
输出参数{out}	
-	-
返回值	
fmc_state_enum	FMC状态
FMC_READY	操作完成
FMC_BUSY	操作进行中
FMC_PGSEERR	编程序列错误
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_CBCMDERR	被选中区域不空白错误

例如：

```
/* check whether flash page is blank or not by check blank command */
```

```
fmc_state_enum state;
```

```
state = fmc_blank_check(0x8001000, 4);
```

### 函数 fmc\_page\_erase

函数fmc\_page\_erase描述见下表：

表 3-401. 函数 `fmc_page_erase`

函数名称	<code>fmc_page_erase</code>
函数原型	<code>fmc_state_enum fmc_page_erase(uint32_t page_address);</code>
功能描述	页擦除
先决条件	<code>fmc_unlock</code>
被调用函数	-
输入参数{in}	
<code>page_address</code>	页擦除首地址
输出参数{out}	
-	-
返回值	
<code>fmc_state_enum</code>	FMC状态
<code>FMC_READY</code>	操作完成
<code>FMC_BUSY</code>	操作进行中
<code>FMC_PGSEERR</code>	编程序列错误
<code>FMC_PGERR</code>	编程错误
<code>FMC_PGAERR</code>	编程对齐错误
<code>FMC_WPERR</code>	擦写保护错误
<code>FMC_TOERR</code>	超时错误
<code>FMC_CBCMDERR</code>	被选中区域不空白错误

例如：

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

```
fmc_unlock();
```

```
fmc_state_enum state = fmc_page_erase(0x08001000);
```

### 函数 `fmc_mflash_mass_erase`

函数 `fmc_mflash_mass_erase` 描述见下表：

表 3-402. 函数 `fmc_mflash_mass_erase`

函数名称	<code>fmc_mflash_mass_erase</code>
函数原型	<code>fmc_state_enum fmc_mflash_mass_erase(void)</code>
功能描述	擦除主闪存块
先决条件	-
被调用函数	-
输入参数{in}	
-	-
输出参数{out}	
-	-
返回值	
<code>fmc_state_enum</code>	FMC 状态

<i>FMC_READY</i>	操作完成
<i>FMC_BUSY</i>	操作进行中
<i>FMC_PGSERR</i>	编程序列错误
<i>FMC_PGERR</i>	编程错误
<i>FMC_PGAERR</i>	编程对齐错误
<i>FMC_WPERR</i>	擦写保护错误
<i>FMC_TOERR</i>	超时错误
<i>FMC_CBCMDERR</i>	被选中区域不空白错误

Example:

```
/* erase main flash */
```

```
fmc_unlock();
```

```
fmc_state_enum state = fmc_mflash_mass_erase();
```

### 函数 **fmc\_dflash\_mass\_erase**

函数fmc\_dflash\_mass\_erase描述见下表:

**表 3-403. 函数 fmc\_dflash\_mass\_erase**

函数名称	fmc_dflash_mass_erase
函数原型	fmc_state_enum fmc_dflash_mass_erase(void)
功能描述	擦除数据闪存块
先决条件	-
被调用函数	-
输入参数{in}	
-	-
输出参数{out}	
-	-
返回值	
<b>fmc_state_enum</b>	FMC 状态
<i>FMC_READY</i>	操作完成
<i>FMC_BUSY</i>	操作进行中
<i>FMC_PGSERR</i>	编程序列错误
<i>FMC_PGERR</i>	编程错误
<i>FMC_PGAERR</i>	编程对齐错误
<i>FMC_WPERR</i>	擦写保护错误
<i>FMC_TOERR</i>	超时错误
<i>FMC_CBCMDERR</i>	被选中区域不空白错误

```
/* erase data flash */
```

```
fmc_unlock();
```

```
fmc_state_enum state = fmc_dflash_mass_erase();
```

### 函数 **fmc\_mass\_erase**

函数fmc\_mass\_erase描述见下表:

**表 3-404. 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	
<b>fmc_state_enum</b>	FMC状态
<i>FMC_READY</i>	操作完成
<i>FMC_BUSY</i>	操作进行中
<i>FMC_PGSEERR</i>	编程序列错误
<i>FMC_PGERR</i>	编程错误
<i>FMC_PGAERR</i>	编程对齐错误
<i>FMC_WPERR</i>	擦写保护错误
<i>FMC_TOERR</i>	超时错误
<i>FMC_CBCEMDERR</i>	被选中区域不空白错误

例如:

```
/* erase whole chip */
```

```
fmc_unlock();
```

```
fmc_state_enum state = fmc_mass_erase();
```

### 函数 **fmc\_fourword\_program**

函数fmc\_fourword\_program描述见下表:

**表3-405. 函数fmc\_fourword\_program**

函数名称	fmc_fourword_program
函数原型	fmc_state_enum fmc_fourword_program(uint32_t address, uint64_t data_l, uint64_t data_h)
功能描述	在相应地址 4 字编程
先决条件	-
被调用函数	-

输入参数{in}	
<b>address</b>	编程地址
输入参数{in}	
<b>data_l</b>	编程数据低 64 位
输入参数{in}	
<b>data_h</b>	编程数据高 64 位
输出参数{out}	
-	-
返回值	
<b>fmc_state_enum</b>	FMC 状态
<i>FMC_READY</i>	操作完成
<i>FMC_BUSY</i>	操作进行中
<i>FMC_PGSERR</i>	编程序列错误
<i>FMC_PGERR</i>	编程错误
<i>FMC_PGAERR</i>	编程对齐错误
<i>FMC_WPERR</i>	擦写保护错误
<i>FMC_TOERR</i>	超时错误
<i>FMC_CBCMDERR</i>	被选中区域不空白错误

Example:

```
/* program 128-bit word at the corresponding address */
```

```
fmc_unlock();
```

```
fmc_page_erase(0x08001000);
```

```
fmc_state_enum state = fmc_fourword_program(0x08001000, 0x0123456789abcdef,
0x08192a3b4c5d6e7f);
```

### 函数 fmc\_fast\_program

函数fmc\_fast\_program描述见下表:

表 3-406. 函数 fmc\_fast\_program

函数名称	fmc_fast_program
函数原形	fmc_state_enum fmc_fast_program(uint32_t address, uint64_t data[]);
功能描述	在主编程块相应地址快速编程一行数据
先决条件	fmc_unlock
被调用函数	-
输入参数{in}	
<b>address</b>	编程地址
输入参数{in}	
<b>data</b>	编程数据
输出参数{out}	

-	-
返回值	
<b>fmc_state_enum</b>	FMC状态
<i>FMC_READY</i>	操作完成
<i>FMC_BUSY</i>	操作进行中
<i>FMC_PGSEERR</i>	编程序列错误
<i>FMC_PGERR</i>	编程错误
<i>FMC_PGAERR</i>	编程对齐错误
<i>FMC_WPERR</i>	擦写保护错误
<i>FMC_TOERR</i>	超时错误
<i>FMC_CBCMDERR</i>	被选中区域不空白错误

例如:

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

```
static uint64_t data_buffer[32] = {
    0x0000000000000000U, 0x1111111111111111U, 0x2222222222222222U, 0x3333333333333333U,
    0x4444444444444444U, 0x5555555555555555U, 0x6666666666666666U, 0x7777777777777777U,
    0x8888888888888888U, 0x9999999999999999U, 0xAAAAAAAAAAAAAAAAAU, 0xBBBBBBBBBBBBBBBU,
    0xCCCCCCCCCCCCCCCCCU, 0xDDDDDDDDDDDDDDDDDDU, 0xEEEEEEEEEEEEEEEEU, 0xFFFFFFFFFFFUFU,
    0x0011001100110011U, 0x2233223322332233U, 0x4455445544554455U, 0x6677667766776677U,
    0x8899889988998899U, 0xAABBAABBAABBAABBU, 0xCCDDCCDDCCDDCCDDU, 0xEEFFEEFFEEFFEEFFU,
    0x2200220022002200U, 0x3311331133113311U, 0x6644664466446644U, 0x7755775577557755U,
    0xAA88AA88AA88AA88U, 0xBB99BB99BB99BB99U, 0xEECCEECCEECCEECCECU, 0xFFDDFFDDFFDDFFDDU
};

fmc_unlock();

fmc_page_erase(0x08001000);

/* program flash */

fmc_state_enum fmc_state = fmc_fast_program(0x08001000, data_buffer);
```

## 函数 fmc\_otp\_fourword\_program

函数fmc\_otp\_fourword\_program描述见下表：

**表3-407. 函数fmc\_otp\_fourword\_program**

函数名称	fmc_otp_fourword_program
函数原型	fmc_state_enum fmc_otp_fourword_program(uint32_t address, uint64_t data_l, uint64_t data_h)
功能描述	在 OTP 区域相应地址 4 字编程
先决条件	-
被调用函数	-
输入参数{in}	
address	编程地址
输入参数{in}	
data_l	编程数据低 64 位
输入参数{in}	
data_h	编程数据高 64 位
输出参数{out}	
-	-
返回值	
fmc_state_enum	FMC状态
FMC_READY	操作完成
FMC_BUSY	操作进行中
FMC_PGSEERR	编程序列错误
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_CBCMDERR	被选中区域不空白错误

Example:

```
/* program 128-bit word at the corresponding address */
```

```
fmc_unlock();
```

```
fmc_page_erase(0x08001000);
```

```
fmc_state_enum state = fmc_otp_fourword_program(0x1FFF7000, 0x0123456789abcdef,
0x08192a3b4c5d6e7f);
```

## 函数 ob\_unlock

函数ob\_unlock描述见下表：

表 3-408. 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：

```
/* unlock the option bytes operation */
fmc_unlock();
ob_unlock();
```

### 函数 ob\_lock

函数ob\_lock描述见下表：

表 3-409. 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_unlock
被调用函数	-
输入参数{in}	
-	-
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* lock the option bytes operation */
fmc_unlock();
ob_lock();
```



## 函数 ob\_erase

函数ob\_erase描述见下表:

表 3-410. 函数 ob\_erase

函数名称	ob_erase
函数原型	fmc_state_enum 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, 详细请参考 <a href="#">表3-382. 枚举类型fmc_state_enum</a>

例如:

```
/* erase the FMC option bytes */
```

```
fmc_state_enum state;
```

```
fmc_unlock();
```

```
ob_unlock();
```

```
state = ob_erase();
```

## 函数 ob\_write\_protection\_enable

函数ob\_write\_protection\_enable描述见下表:

表 3-411. 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	使能写保护
先决条件	ob_unlock
被调用函数	-
输入参数{in}	
ob_wp	写保护单元, 请参考 <a href="#">表3-386. 枚举类型fmc_ob_wp_enum</a>
输出参数{out}	
-	-
返回值	
fmc_state_enum	FMC状态, 详细请参考 <a href="#">表3-382. 枚举类型fmc_state_enum</a>

例如:

```

/* enable write protection */

fmc_state enum state;

fmc_unlock();

ob_unlock();

state = ob_write_protection_enable(OB_WP_7);

```

### 函数 ob\_security\_protection\_config

函数ob\_security\_protection\_config描述见下表：

**表 3-412. 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}	
<b>ob_spc</b>	安全保护
<i>FMC_NSPC</i>	无安全保护
<i>FMC_LSPC</i>	低等级安全保护
<i>FMC_HSPC</i>	号等级安全保护
输出参数{out}	
-	-
返回值	
<b>fmc_state_enum</b>	FMC状态，详细请参考 <a href="#">表3-382. 枚举类型fmc_state_enum</a>

例如：

```

/* enable low security protection */

fmc_state enum state;

ob_unlock();

state = ob_security_protection_config(FMC_LSPC);

```

### 函数 ob\_user\_write

函数ob\_user\_write描述见下表：

**表 3-413. 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);
功能描述	编辑用户选项字节

先决条件	ob_unlock
被调用函数	-
输入参数{in}	
optionbyte_user	选项字节UER段结构体，详细请参考 <a href="#">表3-380. 结构体optionbyte_user_struct</a>
输出参数{out}	
-	-
返回值	
fmc_state_enum	FMC状态，详细请参考 <a href="#">表3-382. 枚举类型fmc_state_enum</a>

Example:

```

/* configure user option byte */

optionbyte_user_struct ob_user;

fmc_state_enum state;

fmc_unlock();

ob_unlock();

ob_user.nfwdg_hw = OB_FWDGT_HW;

ob_user.nrst_dpslp = OB_DEEPSLEEP_RST;

ob_user.nrst_stdbby = OB_STDBY_RST;

ob_user.lvd0en = OB_LVDOEN_ENABLE;

ob_user.fwdgspd_dpslp = OB_FWDGDPSLP_ENABLE;

ob_user.lvd0t = OB_LVD0T_VALUE0;

state = ob_user_write(ob_user);

```

### 函数 ob\_data\_program

函数ob\_data\_program描述见下表：

表 3-414. 函数 ob\_data\_program

函数名称	ob_data_program
函数原型	fmc_state_enum ob_data_program(uint32_t address, uint8_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}	
address	编程数据选项字节地址
输入参数{in}	
data	所编程数值
输出参数{out}	

-	-
返回值	
fmc_state_enum	FMC状态，详细请参考 <a href="#">表3-382. 枚举类型fmc_state_enum</a>

例如：

```
/* program option bytes data */

fmc_state_enum state;

fmc_unlock();

ob_unlock();

state = ob_data_program(0x56);
```

### 函数 ob\_user1\_write

函数ob\_user1\_write描述见下表：

表 3-415. 函数 ob\_user1\_write

函数名称	ob_user1_write
函数原型	fmc_state_enum ob_user1_write(uint8_t ob_wwdg_hw, uint8_t ob_sram_ecc, uint8_t ob_swd_mode, uint8_t ob_nrst_mod)
功能描述	编程选项字节 USER1
先决条件	-
被调用函数	-
输入参数{in}	
ob_wwdg_hw	窗口看门狗
OB_WWDGT_HW	硬件使能窗口看门狗功能
OB_WWDGT_SW	软件使能窗口看门狗功能
输入参数{in}	
ob_sram_parity_chk	SRAM ECC
OB_SRAM_ECC_DISABLE	禁能 SRAM ECC
OB_SRAM_ECC_ENABLE	使能 SRAM ECC
输入参数{in}	
ob_swd_mode	SWD 功能
OB_SWD_DISABLE	禁能 SWD
OB_SWD_ENABLE	使能 SWD
输入参数{in}	
ob_nrst_mod	nRST 引脚模式
OB_NRST_MODE_VAL_UE1	模式 1
OB_NRST_MODE_VAL	模式 2

UE2	
OB_NRST_MODE_VAL UE3	模式 3
输出参数{out}	
-	-
返回值	
fmc_state_enum	FMC 状态，详细请参考 <a href="#">表 3-382. 枚举类型 fmc_state_enum</a>

```
/* program option bytes USER1 */
```

```
fmc_state_enum state;
```

```
fmc_unlock();
```

```
ob_unlock();
```

```
state = ob_user1_write(OB_WWDGT_HW, OB_SRAM_ECC_DISABLE, OB_SWD_DISABLE,  
OB_NRST_MODE_VALUE1);
```

### 函数 ob\_wwdg0\_write

函数ob\_wwdg0\_write描述见下表：

表 3-416. 函数 ob\_wwdg0\_write

函数名称	ob_wwdg0_write
函数原型	fmc_state_enum ob_wwdg0_write(optionbyte_wwdg0_struct *optionbyte_wwdg0)
功能描述	编程选项字节 WWDG0
先决条件	-
被调用函数	-
输入参数{in}	
optionbyte_wwdg0	选项字节 wwdg0 结构体，详细请参考 <a href="#">表 3-381. 结构体 optionbyte_wwdg0_struct</a>
输出参数{out}	
-	-
返回值	
fmc_state_enum	FMC 状态，详细请参考 <a href="#">表 3-382. 枚举类型 fmc_state_enum</a>

Example:

```
/* program option bytes WWDGT0*/
```

```
optionbyte_wwdg0_struct ob_wwdg0;
```

```
fmc_state_enum state;
```

```
fmc_unlock();
```

```
ob_unlock();
```

```

ob_wwdgt0.psc = OB_WWDG_PSC8;

ob_wwdgt0.wsps = OB_WSPS_VALUE25;

ob_wwdgt0.weps = OB_WEPS_VALUE75;

ob_wwdgt0.ewie = OB_EWIE_ENABLE;

ob_wwdgt0.wdga = OB_WDGA_DISABLE;

state = ob_wwdgt0_write (ob_wwdgt0);

```

### 函数 ob\_wwdgt1\_wwdgt2\_write

函数ob\_wwdgt1\_wwdgt2\_write描述见下表：

**表 3-417. 函数 ob\_wwdgt1\_wwdgt2\_write**

函数名称	ob_wwdgt1_wwdgt2_write
函数原型	fmc_state_enum ob_wwdgt1_wwdgt2_write(uint16_t ob_wwdg_cnt)
功能描述	编程选项字节 WWDG1 \ WWDG2
先决条件	-
被调用函数	-
输入参数{in}	
ob_wwdg_cnt	硬件启动模式下窗口看门狗计数值，0 ~ 0x00003FFF
输出参数{out}	
-	-
返回值	
fmc_state_enum	FMC 状态，详细请参考 <a href="#">表 3-382. 枚举类型 fmc_state_enum</a>

Example:

```

/* program option bytes WWDGT1 and WWDGT2 */

fmc_state_enum state;

fmc_unlock();

ob_unlock();

state = ob_wwdgt1_wwdgt2_write(0x1F);

```

### 函数 ob\_fwdgt\_write

函数ob\_fwdgt\_write描述见下表：

**表3-418. 函数ob\_fwdgt\_write**

函数名称	ob_fwdgt_write
函数原型	fmc_state_enum ob_fwdgt_write(uint16_t ob_fwdgt_wnd, uint8_t ob_fwdgt_psc, uint8_t ob_fwdgt_rst, uint16_t ob_fwdgt_reload)

功能描述	编程选项字节 FWDGT
先决条件	-
被调用函数	-
输入参数{in}	
ob_fwdgt_wnd	硬件启动模式下独立看门狗计数器窗口值, 0 ~ 0x00000FFF
输入参数{in}	
ob_fwdgt_psc	独立看门狗分频系数选择
OB_FWDGT_PSC4	独立看门狗分频系数选择 1/4
OB_FWDGT_PSC8	独立看门狗分频系数选择 1/8
OB_FWDGT_PSC16	独立看门狗分频系数选择 1/16
OB_FWDGT_PSC32	独立看门狗分频系数选择 1/32
OB_FWDGT_PSC64	独立看门狗分频系数选择 1/64
OB_FWDGT_PSC128	独立看门狗分频系数选择 1/128
OB_FWDGT_PSC256	独立看门狗分频系数选择 1/256
输入参数{in}	
ob_fwdgt_rst	独立看门狗产生复位
OB_FWDGT_RST_DISABLE	禁能独立看门狗产生复位
OB_FWDGT_RST_ENABLE	使能独立看门狗产生复位
输入参数{in}	
ob_fwdgt_reload	独立看门狗计数器重加载值, 0 ~ 0x00000FFF
输出参数{out}	
-	-
返回值	
fmc_state_enum	FMC 状态, 详细请参考 <a href="#">表 3-382. 枚举类型 fmc_state_enum</a>

/\* program option bytes FWDGT \*/

fmc\_state\_enum state;

fmc\_unlock();

ob\_unlock();

state = ob\_fwdgt\_write(0x1F, OB\_FWDGT\_PSC4, OB\_FWDGT\_RST\_DISABLE, 0xFF);

### 函数 ob\_user\_get

函数ob\_user\_get描述见下表:

表 3-419. 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（0x0 – 0xFF）

例如：

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

```
uint8_t user;
```

```
user = ob_user_get();
```

### 函数 ob\_data\_get

函数ob\_data\_get描述见下表：

表 3-420. 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 bytes */
```

```
uint16_t data;
```

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

### 函数 ob\_write\_protection\_get

函数ob\_write\_protection\_get描述见下表：

表 3-421. 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	获取选项字节写保护数值
先决条件	-
被调用函数	-
输入参数{in}	
-	-
输出参数{out}	
-	-
返回值	
uint32_t	选项字节写保护数值（0x0 – 0xFFFFFFFF）

例如：

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

```
uint32_t wp;
```

```
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 3-422. 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 bytes security protection */
```

```
FlagStatus flag;
```

```
flag = ob_security_protection_flag_get();
```

### 函数 fmc\_flag\_get

函数fmc\_flag\_get描述见下表：

表 3-423. 函数 fmc\_flag\_get

函数名称	fmc_flag_get
函数原型	FlagStatus fmc_flag_get(uint32_t flag);
功能描述	检查标志是否置位
先决条件	-
被调用函数	-
输入参数{in}	
flag	FMC标志, 请参考 <a href="#">表3-383. 枚举类型fmc_flag_enum</a>
输出参数{out}	
-	-
返回值	
FlagStatus	SET或RESET

例如:

```
/* get FMC end of operation flag */
```

```
FlagStatus flag;
```

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

### 函数 fmc\_flag\_clear

函数fmc\_flag\_clear描述见下表:

表 3-424. 函数 fmc\_flag\_clear

函数名称	fmc_flag_clear
函数原型	void fmc_flag_clear(uint32_t flag);
功能描述	清除FMC标志
先决条件	-
被调用函数	-
输入参数{in}	
flag	FMC标志, 请参考 <a href="#">表3-383. 枚举类型fmc_flag_enum</a>
输出参数{out}	
-	-
返回值	
-	-

例如:

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

```
fmc_flag_clear(FMC_FLAG_PGERR);
```

### 函数 fmc\_interrupt\_enable

函数fmc\_interrupt\_enable描述见下表:

表 3-425. 函数 `fmc_interrupt_enable`

函数名称	<code>fmc_interrupt_enable</code>
函数原型	<code>void fmc_interrupt_enable(uint32_t interrupt);</code>
功能描述	使能FMC中断
先决条件	<code>fmc_unlock</code>
被调用函数	-
输入参数{in}	
<code>interrupt</code>	FMC中断, 请参考 <a href="#">表3-385. 枚举类型fmc_interrupt_enum</a>
输出参数{out}	
-	-
返回值	
-	-

例如:

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

#### 函数 `fmc_interrupt_disable`

函数`fmc_interrupt_disable`描述见下表:

表 3-426. 函数 `fmc_interrupt_disable`

函数名称	<code>fmc_interrupt_disable</code>
函数原型	<code>void fmc_interrupt_disable(uint32_t interrupt);</code>
功能描述	禁能FMC中断
先决条件	<code>fmc_unlock</code>
被调用函数	-
输入参数{in}	
<code>interrupt</code>	FMC中断, 请参考 <a href="#">表3-385. 枚举类型fmc_interrupt_enum</a>
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* disable FMC end of operation interrupt */  
  
fmc_unlock();  
  
fmc_interrupt_disable(FMC_INT_END);
```

## 函数 fmc\_interrupt\_flag\_get

函数fmc\_interrupt\_flag\_get描述见下表：

表 3-427. 函数 fmc\_interrupt\_flag\_get

函数名称	fmc_interrupt_flag_get
函数原型	FlagStatus fmc_interrupt_flag_get(fmc_interrupt_flag_enum flag);
功能描述	获取FMC中断标志状态
先决条件	-
被调用函数	-
输入参数{in}	
flag	中断标志，请参考 <a href="#">表3-384. 枚举类型fmc_interrupt_flag_enum</a>
输出参数{out}	
-	-
返回值	
FlagStatus	SET或RESET

例如：

```
/* get FMC operation error interrupt flag bit */
```

```
FlagStatus flag;
```

```
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 3-428. 函数 fmc\_interrupt\_flag\_clear

函数名称	fmc_interrupt_flag_clear
函数原型	void fmc_interrupt_flag_clear (fmc_interrupt_flag_enum flag);
功能描述	清除FMC中断标志
先决条件	-
被调用函数	-
输入参数{in}	
flag	清除FMC中断标志，请参考 <a href="#">表3-384. 枚举类型fmc_interrupt_flag_enum</a>
输出参数{out}	
-	-
返回值	
-	-

例如：

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

```
fmc_interrupt_flag_clear (FMC_INT_FLAG_PGERR);
```

### 3.15. FWDGT

独立看门狗定时器（FWDGT）是一个硬件计时电路，用来监测由软件故障导致的系统故障。适合于需要独立环境且对计时精度要求不高的场合。章节[3.15.1](#)描述了FWDGT的寄存器列表，章节[3.15.2](#)对FWDGT库函数进行说明。

#### 3.15.1. 外设寄存器说明

FWDGT寄存器列表如下表所示：

表 3-429. 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
FWDGT_RCR	复位控制寄存器

#### 3.15.2. 外设库函数说明

FWDGT库函数列表如下表所示：

表 3-430. FWDGT 库函数

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

#### 函数 fwdgt\_write\_enable

函数fwdgt\_write\_enable描述见下表：

表 3-431. 函数 fwdgt\_write\_enable

函数名称	fwdgt_write_enable
------	--------------------

函数原型	void fwdgt_write_enable(void);
功能描述	使能对寄存器FWDGT_PSC, FWDGT_RLD和FWDGT_WND的写操作
先决条件	-
被调用函数	-
输入参数{in}	
-	-
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* enable write access to FWDGT_PSC、FWDGT_RLD and FWDGT_WND */
fwdgt_write_enable();
```

### 函数 fwdgt\_write\_disable

函数fwdgt\_write\_disable描述见下表:

表 3-432. 函数 fwdgt\_write\_disable

函数名称	fwdgt_write_disable
函数原型	void fwdgt_write_disable(void);
功能描述	除能对寄存器FWDGT_PSC, FWDGT_RLD和FWDGT_WND的写操作
先决条件	-
被调用函数	-
输入参数{in}	
-	-
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* disable write access to FWDGT_PSC、FWDGT_RLD and FWDGT_WND */
fwdgt_write_disable();
```

### 函数 fwdgt\_enable

函数fwdgt\_enable描述见下表:

表 3-433. 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 3-434. 函数 fwdgt\_prescaler\_value\_config

函数名称	fwdgt_prescaler_value_config
函数原型	ErrStatus fwdgt_prescaler_value_config(uint16_t prescaler_value);
功能描述	配置独立看门狗定时器时钟预分频数
先决条件	-
被调用函数	-
输入参数{in}	
prescaler_value	预分频值
FWDGT_PSC_DIVx	FWDGT预分频值设为x（x=4, 8, 16, 32, 64, 128, 256）
输出参数{out}	
-	-
返回值	
ErrStatus	ERROR / SUCCESS

例如：

```
/* set FWDGT prescaler to 4 */
```

```
ErrStatus flag;
```

```
flag = fwdgt_prescaler_value_config(FWDGT_PSC_DIV4);
```

### 函数 fwdgt\_reload\_value\_config

函数fwdgt\_reload\_value\_config描述见下表：

表 3-435. 函数 fwdgt\_reload\_value\_config

函数名称	fwdgt_reload_value_config
函数原型	ErrStatus fwdgt_reload_value_config(uint16_t reload_value);
功能描述	配置独立看门狗定时器计数器重装载值
先决条件	-
被调用函数	-
输入参数{in}	
reload_value	重装载值，数值范围为0x0000 - 0x0FFF
输出参数{out}	
-	-
返回值	
ErrStatus	ERROR / SUCCESS

例如：

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

```
ErrStatus flag;
```

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

#### 函数 fwdgt\_window\_value\_config

函数fwdgt\_window\_value\_config描述见下表：

表 3-436. 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 3-437. 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 3-438. 函数 fwdgt\_config**

函数名称	fwdgt_config
函数原型	ErrStatus fwdgt_config(uint16_t reload_value, uint8_t prescaler_value);
功能描述	设置FWDGT重装载值和预分频值
先决条件	-
被调用函数	-
输入参数{in}	
reload_value	重装载值(0x0000 - 0x0FFF)
输入参数{in}	
prescaler_value	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

<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	
<b>ErrStatus</b>	ERROR or SUCCESS-

例如:

```
/* configure FWDGT counter clock: 32KHz(IRC32K) / 64 = 0.5 KHz */
```

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

### 函数 fwdgt\_reset\_output

函数fwdgt\_reset\_output描述见下表:

表 3-439. 函数 fwdgt\_reset\_output

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

例如:

```
/* configure the FWDGT reset output */
```

```
void fwdgt_reset_output(void);
```

### 函数 fwdgt\_interrupt\_output

函数fwdgt\_interrupt\_output描述见下表:

表 3-440. 函数 fwdgt\_interrupt\_output

函数名称	fwdgt_interrupt_output
函数原型	void fwdgt_interrupt_output(void);
功能描述	配置FWDGT中断输出
先决条件	-

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

例如：

```
/* configure the FWDGT interrupt output */
```

```
void fwdgt_interrupt_output(void);
```

### 函数 fwdgt\_flag\_get

函数fwdgt\_flag\_get描述见下表：

表 3-441. 函数 fwdgt\_flag\_get

函数名称	fwdgt_flag_get
函数原型	FlagStatus fwdgt_flag_get(uint16_t flag);
功能描述	获取FWDGT标志位状态
先决条件	-
被调用函数	-
输入参数{in}	
flag	需要获取状态的FWDGT标志位
FWDGT_FLAG_PUD	预分频值更新进行中
FWDGT_FLAG_RU D	重装载值更新进行中
FWDGT_FLAG_WU D	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);
```

## 3.16. GPIO

GPIO用来实现各片上设备的逻辑输入/输出功能。章节[3.16.1](#)描述了GPIO的寄存器列表，章节[3.16.2](#)对GPIO库函数进行。

### 3.16.1. 外设寄存器说明

GPIO 寄存器列表如下表所示：

表 3-442. 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
GPIOx_HLD	引脚电平保持配置寄存器
GPIOx_FUNCSEL	引脚选择功能寄存器
GPIOx_HLDCTL	引脚电平保持控制寄存器

### 3.16.2. 外设库函数说明

GPIO库函数列表如下表所示：

表 3-443. 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

库函数名称	库函数描述
gpio_input_port_get	获取一组端口的输入值
gpio_output_bit_get	获取引脚的输出值
gpio_output_port_get	获取一组端口的输出值
gpio_af_set	设置GPIO复用功能
gpio_pin_lock	相应的引脚配置被锁定
gpio_bit_toggle	翻转GPIO引脚状态
gpio_port_toggle	翻转一组GPIO状态
gpio_bit_hold_enable	GPIO端口的位保持功能使能
gpio_bit_hold_disable	GPIO端口的位保持功能除能
gpio_tsel_function_enable	时间戳选择功能使能
gpio_tsel_function_disable	时间戳选择功能除能
gpio_one_line_function_enable	一线功能使能
gpio_one_line_function_disable	一线功能除能
gpio_sample_hold_enable	数据输出功能使能

### 函数 gpio\_deinit

函数gpio\_deinit描述见下表：

**表 3-444. 函数 gpio\_deinit**

函数名称	gpio_deinit
函数原型	void gpio_deinit(uint32_t gpio_periph);
功能描述	复位外设GPIOx
先决条件	-
被调用函数	rcu_periph_reset_enable / rcu_periph_reset_disable
输入参数{in}	
gpio_periph	GPIO端口
GPIOx	端口选择(x = A,B,C,D,E,F,G,N)
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 3-445. 函数 `gpio_mode_set`

函数名称	<code>gpio_mode_set</code>
函数原型	<code>void gpio_mode_set(uint32_t gpio_periph, uint32_t mode, uint32_t pull_up_down, uint32_t pin);</code>
功能描述	设置GPIO模式
先决条件	-
被调用函数	-
输入参数{in}	
<code>gpio_periph</code>	GPIO端口
<code>GPIOx</code>	端口选择(x = A,B,C,D,E,F,G,N)
输入参数{in}	
<code>mode</code>	GPIO引脚模式
<code>GPIO_MODE_INPUT</code>	输入模式
<code>GPIO_MODE_OUTPUT</code>	输出模式
<code>GPIO_MODE_AF</code>	备用功能模式
<code>GPIO_MODE_ANALOG</code>	模拟模式
输入参数{in}	
<code>pull_up_down</code>	GPIO引脚上拉下拉电阻设置
<code>GPIO_PUPD_NONE</code>	悬空模式，无上拉和下拉
<code>GPIO_PUPD_PULLUP</code>	带上拉电阻
<code>GPIO_PUPD_PULLDOWN</code>	带下拉电阻
输入参数{in}	
<code>pin</code>	GPIO pin
<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 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 3-446. 函数 gpio\_output\_options\_set

函数名称	gpio_output_options_set
函数原型	void gpio_output_options_set(uint32_t gpio_periph, uint8_t otype, uint32_t speed, uint32_t pin);
功能描述	设置GPIO输出模式和速度
先决条件	-
被调用函数	-
输入参数{in}	
gpio_periph	GPIO端口
GPIOx	端口选择(x = A,B,C,D,E,F,G,N)
输入参数{in}	
otype	GPIO引脚输出模式
GPIO_OTYPE_PP	推挽输出模式
GPIO_OTYPE_OD	开漏输出模式
输入参数{in}	
speed	GPIO引脚输出最大速度
GPIO_OSPEED_LOW	低速驱动
GPIO_OSPEED_HIGH	高速驱动
GPIO_OSPEED_LARGE_CURRENT	高电流驱动
输入参数{in}	
pin	GPIO引脚
GPIO_PIN_x	引脚选择(x=0..15)
GPIO_PIN_ALL	所有引脚
输出参数{out}	
-	-
返回值	
-	-

例如：

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

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

## 函数 gpio\_bit\_set

函数gpio\_bit\_set描述见下表：

表 3-447. 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}	
<code>gpio_periph</code>	GPIO端口
<code>GPIOx</code>	端口选择(x = A,B,C,D,E,F,G,N)
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:

```
/* 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 3-448. 函数 `gpio_bit_reset`

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

```
/* 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 3-449. 函数 gpio\_bit\_write

函数名称	gpio_bit_write
函数原型	void gpio_bit_write(uint32_t gpio_periph, uint32_t pin, bit_status bit_value);
功能描述	将特定的值写入引脚
先决条件	-
被调用函数	-
输入参数{in}	
gpio_periph	GPIO端口
GPIOx	端口选择(x = A,B,C,D,E,F,G,N)
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 3-450. 函数 gpio\_port\_write

函数名称	gpio_port_write
函数原型	void gpio_port_write(uint32_t gpio_periph, uint16_t data);

功能描述	将特定的值写入端口
先决条件	-
被调用函数	-
输入参数{in}	
gpio_periph	GPIO端口
GPIOx	端口选择(x = A,B,C,D,E,F,G,N)
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 3-451. 函数 gpio\_input\_bit\_get

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

例如:

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

```
FlagStatus bit_state;
```

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

### 函数 gpio\_input\_port\_get

函数gpio\_input\_port\_get描述见下表:

表 3-452. 函数 gpio\_input\_port\_get

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

例如:

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

```
uint16_t port_state;
```

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

### 函数 gpio\_output\_bit\_get

函数gpio\_output\_bit\_get描述见下表:

表 3-453. 函数 gpio\_output\_bit\_get

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

-	-
返回值	
FlagStatus	SET / RESET

例如:

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

```
FlagStatus bit_state;
```

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

### 函数 gpio\_output\_port\_get

函数gpio\_output\_port\_get描述见下表:

表 3-454. 函数 gpio\_output\_port\_get

函数名称	gpio_output_port_get
函数原型	uint16_t gpio_output_port_get(uint32_t gpio_periph);
功能描述	获取引脚的输出值
先决条件	-
被调用函数	-
输入参数{in}	
gpio_periph	GPIO端口
GPIOx	端口选择(x = A,B,C,D,E,F,G,N)
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 3-455. 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}	
<b>gpio_periph</b>	GPIO端口
<i>GPIOx</i>	端口选择(x = A,B,C,D,E,F,G,N)
输入参数{in}	
<b>alt_func_num</b>	GPIO 引脚备用功能, 请参见特定设备的数据手册
<i>GPIO_AF_0</i>	CK_OUT, TRACECK, TRACED0, TRACED1, TRACED2, TRACED3, CPTIMERW
<i>GPIO_AF_1</i>	TIMER0, TIMER1, TIMER2, TIMER7
<i>GPIO_AF_2</i>	TIMER_ETI, TIMER0, TIMER7
<i>GPIO_AF_3</i>	TIMER0, TIMER2, TIMER7
<i>GPIO_AF_4</i>	TIMER0, TIMER7, POC
<i>GPIO_AF_5</i>	CFMUREF, ADC0, ADC2, ADCSM1, ADCSM2
<i>GPIO_AF_6</i>	UART0, UART1, UART2, UART3, POC,
<i>GPIO_AF_7</i>	UART3, ADC0, ADC2, TIMER0, TIMER7, I2C
<i>GPIO_AF_8</i>	CMP0, CMP1, CMP2, CMP3, I2C, CAN
<i>GPIO_AF_9</i>	TIMER2, SPI
<i>GPIO_AF_10</i>	GTOC0, GPTIMER0, GPTIMER1, UART3, ADCSM3, ADCSM4
<i>GPIO_AF_11</i>	GTOC0, GTOC1, SPI, GPTIMER0, GPTIMER1
<i>GPIO_AF_12</i>	GTOC2, GPTIMER0, GPTIMER1
<i>GPIO_AF_13</i>	GTOC2, GTOC3, CAN, GPTIMER0, GPTIMER1
<i>GPIO_AF_14</i>	GPTIMER0, GPTIMER1
<i>GPIO_AF_15</i>	EVENTOUT
输入参数{in}	
<b>pin</b>	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 3-456. 函数 gpio\_pin\_lock

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

功能描述	相应的引脚配置被锁定
先决条件	-
被调用函数	-
输入参数{in}	
gpio_periph	GPIO端口
GPIOx	端口选择(x = A,B,C,D,E,F,G,N)
输入参数{in}	
pin	GPIO引脚
GPIO_PIN_x	引脚选择(x=0..15)
GPIO_PIN_ALL	所有引脚
输出参数{out}	
-	-
返回值	
-	-

例如:

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

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

## 函数 gpio\_bit\_toggle

函数gpio\_bit\_toggle描述见下表:

表 3-457. 函数 gpio\_bit\_toggle

函数名称	gpio_bit_toggle
函数原型	void gpio_bit_toggle(uint32_t gpio_periph, uint32_t pin);
功能描述	翻转GPIO引脚状态
先决条件	-
被调用函数	-
输入参数{in}	
gpio_periph	GPIO端口
GPIOx	端口选择(x = A,B,C,D,E,F,G,N)
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 3-458. 函数 gpio\_port\_toggle**

函数名称	gpio_port_toggle
函数原型	void gpio_port_toggle(uint32_t gpio_periph);
功能描述	翻转一组GPIO状态
先决条件	-
被调用函数	-
输入参数{in}	
gpio_periph	GPIO端口
GPIOx	端口选择(x = A,B,C,D,E,F,G,N)
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* toggle GPIOA */
```

```
gpio_port_toggle(GPIOA);
```

### 函数 void gpio\_bit\_hold\_enable

函数gpio\_bit\_hold\_enable描述见下表：

**表 3-459. 函数 void gpio\_bit\_hold\_enable**

函数名称	gpio_bit_hold_enable
函数原型	void gpio_bit_hold_enable(uint32_t gpio_periph, uint32_t pin)
功能描述	GPIO端口的位保持功能使能
先决条件	-
被调用函数	-
输入参数{in}	
gpio_periph	GPIO端口
GPIOx	端口选择(x = A,B,C,D,E,F,G,N)
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* toggle GPIOA */
```

```
gpio_bit_hold_enable (GPIOA);
```

### 函数 void gpio\_bit\_hold\_disable

函数gpio\_bit\_hold\_disable描述见下表：

**表 3-460. 函数 void gpio\_bit\_hold\_enable**

函数名称	gpio_bit_hold_disable
函数原型	void gpio_bit_hold_disable(uint32_t gpio_periph, uint32_t pin)
功能描述	GPIO端口的位保持功能除能
先决条件	-
被调用函数	-
输入参数{in}	
gpio_periph	GPIO端口
GPIOx	端口选择(x = A,B,C,D,E,F,G,N)
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* toggle GPIOA */
```

```
gpio_bit_hold_disable (GPIOA);
```

### 函数 gpio\_tsel\_function\_enable

函数gpio\_tsel\_function\_enable描述见下表：

**表 3-461. 函数 void gpio\_tsel\_function\_enable**

函数名称	gpio_tsel_function_enable
函数原型	void gpio_tsel_function_enable(void)
功能描述	时间戳选择功能使能
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 tsel function */
```

```
gpio_tsel_function_enable (void);
```

### 函数 gpio\_tsel\_function\_disable

函数gpio\_tsel\_function\_disable描述见下表:

**表 3-462. 函数 void gpio\_tsel\_function\_disable**

函数名称	gpio_tsel_function_disable
函数原型	void gpio_tsel_function_disable(void)
功能描述	时间戳选择功能除能
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 tsel function */
```

```
gpio_tsel_function_disable (void);
```

### 函数 gpio\_one\_line\_function\_enable

函数gpio\_one\_line\_function\_enable描述见下表:

**表 3-463. 函数 void gpio\_one\_line\_function\_enable**

函数名称	gpio_one_line_function_enable
函数原型	void gpio_one_line_function_enable (void)
功能描述	一线功能使能
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 one line function */
```

gpio\_one\_line\_function\_enable (void);

### 函数 gpio\_one\_line\_function\_disable

函数gpio\_one\_line\_function\_disable描述见下表：

**表 3-464. 函数 gpio\_one\_line\_function\_disable**

函数名称	gpio_one_line_function_disable
函数原型	void gpio_one_line_function_disable (void)
功能描述	一线功能除能
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 one line function */
```

```
gpio_one_line_function_disable (void);
```

### 函数 gpio\_sample\_hold\_enable

函数gpio\_smple\_hold\_enable描述见下表：

**表 3-465. 函数 void gpio\_smple\_hold\_enable**

函数名称	gpio_sample_hold_enable
函数原型	void gpio_smple_hold_enable (void)
功能描述	数据输出功能使能
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 data ouput function */
```

```
gpio_sample_hold_enable(void);
```

## 3.17. GPTIMER

通用定时器（GPTIMER0 / 1）是2通道定时器，支持输入捕获和输出比较。可以产生PWM信号控制功率因数校正（PFC）。通用定时器含有一个16位无符号计数器。章节[3.17.1](#)描述了CMP的寄存器列表，章节[3.17.2](#)对GPTIMER库函数进行说明。

### 3.17.1. 外设寄存器说明

GPTIMER寄存器列表如下表所示：

**表 3-466. GPTIMER 寄存器**

寄存器名称	寄存器描述
GPTIMER_WP	GPTIMER 寄存器写保护
GPTIMER_SWEN	GPTIMER 软件使能寄存器
GPTIMER_SWDIS	GPTIMER 软件禁能寄存器
GPTIMER_SWRST	GPTIMER 软件复位寄存器
GPTIMER_ESSEL	GPTIMER 计数器使能源选择寄存器
GPTIMER_DSSEL	GPTIMER 计数器禁能源选择寄存器
GPTIMER_RSSEL	GPTIMER 计数器复位源选择寄存器
GPTIMER_CH0CSSEL	GPTIMER 通道 0 捕获源选择寄存器
GPTIMER_CH1CSSEL	GPTIMER 通道 1 捕获源选择寄存器
GPTIMER_CTL0	GPTIMER 控制寄存器 0
GPTIMER_CUPEVSSEL L	GPTIMER 向上计数事件源选择寄存器
GPTIMER_CDNEVSSEL L	GPTIMER 向下计数事件源选择寄存器
GPTIMER_CH0CTL	GPTIMER 通道 0 控制寄存器
GPTIMER_CH1CTL	GPTIMER 通道 1 控制寄存器
GPTIMER_CHCTL	GPTIMER 通道控制寄存器
GPTIMER_DMAINTEN	GPTIMER DMA 和中断使能寄存器
GPTIMER_INTF	GPTIMER 中断标志寄存器
GPTIMER_UPSSEL	GPTIMER 更新源选择寄存器
GPTIMER_CNT	GPTIMER 计数器寄存器
GPTIMER_PSC	GPTIMER 预分频寄存器
GPTIMER_CAR	GPTIMER 计数器自动重载寄存器
GPTIMER_CH0CV	GPTIMER 通道 0 捕获/比较寄存器
GPTIMER_CH1CV	GPTIMER 通道 1 捕获/比较寄存器
GPTIMER_CH0COMV_ ADD	GPTIMER 通道 0 附加比较寄存器寄存器
GPTIMER_CH1COMV_ ADD	GPTIMER 通道 1 附加比较寄存器寄存器
GPTIMER_DTCTL	GPTIMER 死区控制寄存器

GPTIMER_ADCTL	GPTIMER ADC 触发控制寄存器
GPTIMER_ADCCR1	GPTIMER ADC 触发比较值 1 寄存器
GPTIMER_ADCCR2	GPTIMER ADC 触发比较值 2 寄存器
GPTIMER_ADCTRGS	GPTIMER ADC 触发跳过寄存器
GPTIMER_ADDINTSC TL0	GPTIMER 附加中断跳过控制寄存器 0
GPTIMER_ADDINTSC TL1	GPTIMER 附加中断跳过控制寄存器 1
GPTIMER_IADCTSS	GPTIMER 中断和 ADC 触发信号跳过寄存器寄存器
GPTIMER_CREP	GPTIMER 重复计数器寄存器寄存器
GPTIMER_SYN_RSTCT L	GPTIMER 同步复位控制寄存器
GPTIMER_DMCFG	GPTIMER DMA 配置寄存器
GPTIMER_DMATB	GPTIMER DMA 发送缓冲区寄存器
GPTIMER_CFG	GPTIMER 配置寄存器

### 3.17.2. 外设库函数说明

GPTIMER库函数列表如下表所示：

**表 3-467. GPTIMER 库函数**

库函数名称	库函数描述
gptimer_deinit	复位 GPTIMER
gptimer_struct_para_init	初始化 GPTIMER 初始参数为默认值
gptimer_init	初始化 GPTIMER 计数器
gptimer_register_write_protect_enable	使能 GPTIMER 软件写保护功能
gptimer_register_write_protect_disable	禁能 GPTIMER 软件写保护功能
gptimer_clock_source_select	选择 GPTIMER 时钟源
gptimer_clock_polarity_config	配置 GPTIMER 时钟极性
gptimer_counter_software_enable	软件使能 GPTIMER 计数器
gptimer_counter_software_disable	软件禁能 GPTIMER 计数器
gptimer_counter_software_reset	软件复位 GPTIMER 计数器
gptimer_counter_enable_source_struct_para_init	初始化 GPTIMER 计数器使能源初始参数为默认值

gptimer_counter_enable_source_config	初始化 GPTIMER 计数器使能源
gptimer_counter_disable_source_struct_para_init	初始化 GPTIMER 计数器禁能源初始参数为默认值
gptimer_counter_disable_source_config	初始化 GPTIMER 计数器禁能源
gptimer_counter_reset_source_struct_para_init	初始化 GPTIMER 计数器复位源初始参数为默认值
gptimer_counter_reset_source_config	初始化 GPTIMER 计数器复位源
gptimer_counter_up_source_struct_para_init	初始化 GPTIMER 计数器向上计数源初始参数为默认值
gptimer_counter_up_source_config	初始化 GPTIMER 计数器向上计数源
gptimer_counter_down_source_struct_para_init	初始化 GPTIMER 计数器向下计数源初始参数为默认值
gptimer_counter_down_source_config	初始化 GPTIMER 计数器向下计数源
gptimer_counter_up_input_level_source_select	选择 GPTIMER 计数器向上计数输入电平源
gptimer_counter_down_input_level_source_select	选择 GPTIMER 计数器向下计数输入电平源
gptimer_enable	使能 GPTIMER
gptimer_disable	禁能 GPTIMER
gptimer_auto_reload_shadow_enable	使能自动重载影子功能
gptimer_auto_reload_shadow_disable	禁能自动重载影子功能
gptimer_update_event_enable	使能更新事件
gptimer_update_event_disable	禁能更新事件
gptimer_counter_alignment	设置计数器对齐模式
gptimer_counter_up_direction	设置计数器向上计数方向
gptimer_counter_down_direction	设置计数器向下计数方向
gptimer_counter_direction_force_set_enable	使能计数器方向强制设置

gptimer_counter_directi on_force_set_disable	禁能计数器方向强制设置
gptimer_register_global _update_source_select	选择寄存器全局更新源
gptimer_register_local _update_source_select	选择寄存器本地更新源
gptimer_prescaler_conf ig	配置预分频
gptimer_update_repetiti on_value_config	配置更新重复计数器值
gptimer_update_repetiti on_counter_read	读取更新重复计数器值
gptimer_autoreload_val ue_config	配置自动重载寄存器值
gptimer_counter_value_ config	配置计数器寄存器值
gptimer_counter_read	读取计数器值
gptimer_prescaler_read	读取预分频值
gptimer_single_pulse_ mode_config	配置单脉冲模式
gptimer_dma_enable	使能 DMA
gptimer_dma_disable	禁能 DMA
gptimer_dma_transfer_ config	配置 DMA 传输
gptimer_channel_output _struct_para_init	初始化 GPTIMER 通道输出初始参数为默认值
gptimer_channel_output _config	配置通道输出模式
gptimer_channel_output _force_duty_config	配置通道输出强制占空比
gptimer_channel_output _compare_value_config	配置通道输出比较值
gptimer_channel_output _additional_compare_v alue_config	配置通道输出附加比较值
gptimer_channel_output _shadow_config	配置通道输出影子功能
gptimer_channel_output _additional_shadow_co nfig	配置通道输出附加影子功能
gptimer_channel_output _control_shadow_confi	配置通道输出控制影子功能

g	
gptimer_two_channel_output_high_check_enable	使能两通道同时输出高电平检测功能
gptimer_two_channel_output_high_check_disable	禁能两通道同时输出高电平检测功能
gptimer_two_channel_output_low_check_enable	使能两通道同时输出低电平检测功能
gptimer_two_channel_output_low_check_disable	禁能两通道同时输出低电平检测功能
gptimer_period_signal_output_enable	使能计数器周期同步信号输出
gptimer_period_signal_output_disable	禁能计数器周期同步信号输出
gptimer_stop_output_set_select	配置停止输出源
gptimer_stop_output_recover_time_select	选择输出停止恢复时间
gptimer_channel_complementary_output_struct_para_init	初始化 GPTIMER 互补输出初始参数为默认值
gptimer_channel_complementary_output_config	配置 GPTIMER 通道互补输出功能
gptimer_channel_io_direction_config	配置通道为输入或者输出
gptimer_channel_io_state_config	配置通道为输入或者输出状态使能
gptimer_capture_source_struct_para_init	初始化 GPTIMER 捕获源初始参数为默认值
gptimer_capture_source_config	初始化 GPTIMER 捕获源
gptimer_channel_input_capture_filter_config	配置通道输入捕获滤波
gptimer_channel_input_capture_prescaler_config	配置通道输入捕获分频
gptimer_channel_capture_value_register_read	读取通道捕获比较寄存器值
gptimer_counter_sync_	选择计数器同步复位组

control_select	
gptimer_counter_sync_reset_source_select	选择计数器同步复位源
gptimer_trigger_adc_compare_enable	使能比较事件产生 ADC 转换触发信号
gptimer_trigger_adc_compare_disable	禁能比较事件产生 ADC 转换触发信号
gptimer_trigger_adc_compare_value_config	配置触发 ADC 比较寄存器值
gptimer_trigger_adc_compare_shadow_enable	使能 ADC 比较寄存器影子功能
gptimer_trigger_adc_compare_shadow_disable	禁能 ADC 比较寄存器影子功能
gptimer_trigger_adc_adsr_enable	使能触发 ADC adsr 信号
gptimer_trigger_adc_adsr_disable	禁能触发 ADC adsr 信号
gptimer_trigger_adc_adsr_select	选择触发 ADC adsr 信号
gptimer_trigger_adc_skippping_config	配置触发 ADC 跳过值
gptimer_trigger_adc_skippping_time_select	选择触发 ADC 跳过功能
gptimer_trigger_adc_skippping_counter_read	读取触发 ADC 跳过计数器
gptimer_additional_interrupt_skipping_config	配置附加中断跳过初始值
gptimer_additional_interrupt_skipping_time_select	选择附加中断跳过功能
gptimer_additional_interrupt_skipping_counter_read	读取附加中断跳过计数器
gptimer_flow_interrupt_skipping_source_select	选择溢出中断跳过源
gptimer_flow_interrupt_skipping_link_enable	使能溢出中断跳过链接功能
gptimer_flow_interrupt_skipping_link_disable	禁能溢出中断跳过链接功能
gptimer_flow_interrupt_skipping_num_config	配置溢出中断重复跳过值
gptimer_flow_interrupt_skipping_counter_read	读取溢出中断跳过计数器值



skipping_counter_read	
gptimer_write_chxval_register_config	配置写 CHxVAL 寄存器选择
gptimer_flag_get	获取 GPTIMER 标志位
gptimer_flag_clear	复位 GPTIMER 标志位
gptimer_interrupt_enable	使能 GPTIMER 中断
gptimer_interrupt_disable	禁能 GPTIMER 中断
gptimer_interrupt_flag_get	获取中断 GPTIMER 标志位
gptimer_interrupt_flag_clear	清除中断 GPTIMER 标志位

### 结构体类型 gptimer\_parameter\_struct

表 3-468. 结构体类型 gptimer\_parameter\_struct

成员名称	功能描述
clock_source	时钟源
clock_polarity	时钟极性
prescaler	预分频值
alignedmode	对齐模式
counterdirection	计数方向
period	周期值
clockdivision	时钟分频值
repetitioncounter	计数器重复值

### 结构体类型 gptimer\_counter\_enable\_source\_parameter\_struct

表 3-469. 结构体类型 gptimer\_counter\_enable\_source\_parameter\_struct

成员名称	功能描述
enable_source_eti0	eti0 作为计数器使能源
enable_source_eti1	eti1 作为计数器使能源
enable_source_eti2	eti2 作为计数器使能源
enable_source_eti3	eti3 作为计数器使能源
enable_source_ch0	channel 0 作为计数器使能源
enable_source_ch1	channel 1 作为计数器使能源
enable_source_evsel0	evsel0 作为计数器使能源
enable_source_evsel1	evsel1 作为计数器使能源
enable_source_evsel2	evsel2 作为计数器使能源
enable_source_evsel3	evsel3 作为计数器使能源
enable_source_evsel4	evsel4 作为计数器使能源
enable_source_evsel5	evsel5 作为计数器使能源

enable_source_evsel6	evsel6 作为计数器使能源
enable_source_evsel7	evsel7 作为计数器使能源
enable_source_softwar e	软件事件作为计数器使能源

### 结构体类型 `gptimer_counter_disable_source_parameter_struct`

表 3-470. 结构体类型 `gptimer_counter_disable_source_parameter_struct`

成员名称	功能描述
disable_source_eti0	eti0 作为计数器禁能源
disable_source_eti1	eti1 作为计数器禁能源
disable_source_eti2	eti2 作为计数器禁能源
disable_source_eti3	eti3 作为计数器禁能源
disable_source_ch0	channel 0 作为计数器禁能源
disable_source_ch1	channel 1 作为计数器禁能源
disable_source_evsel0	evsel0 作为计数器禁能源
disable_source_evsel1	evsel1 作为计数器禁能源
disable_source_evsel2	evsel2 作为计数器禁能源
disable_source_evsel3	evsel3 作为计数器禁能源
disable_source_evsel4	evsel4 作为计数器禁能源
disable_source_evsel5	evsel5 作为计数器禁能源
disable_source_evsel6	evsel6 作为计数器禁能源
disable_source_evsel7	evsel7 作为计数器禁能源
disable_source_softwar e	软件事件作为计数器禁能源

### 结构体类型 `gptimer_counter_reset_source_parameter_struct`

表 3-471. 结构体类型 `gptimer_counter_reset_source_parameter_struct`

成员名称	功能描述
reset_source_eti0	eti0 作为计数器复位源
reset_source_eti1	eti1 作为计数器复位源
reset_source_eti2	eti2 作为计数器复位源
reset_source_eti3	eti3 作为计数器复位源
reset_source_ch0	channel 0 作为计数器复位源
reset_source_ch1	channel 1 作为计数器复位源
reset_source_evsel0	evsel0 作为计数器复位源
reset_source_evsel1	evsel1 作为计数器复位源
reset_source_evsel2	evsel2 作为计数器复位源
reset_source_evsel3	evsel3 作为计数器复位源
reset_source_evsel4	evsel4 作为计数器复位源
reset_source_evsel5	evsel5 作为计数器复位源

reset_source_evsel6	evsel6 作为计数器复位源
reset_source_evsel7	elcn 作为计数器复位源
reset_source_com_cap_sync	比较、捕获或同步复位作为计数器复位源
reset_source_software	软件事件作为计数器复位源

### 结构体类型 `gptimer_capture_source_parameter_struct`

表 3-472. 结构体类型 `gptimer_capture_source_parameter_struct`

成员名称	功能描述
capture_source_eti0	eti0 作为捕获源
capture_source_eti1	eti1 作为捕获源
capture_source_eti2	eti2 作为捕获源
capture_source_eti3	eti3 作为捕获源
capture_source_ch0	channel 0 作为捕获源
capture_source_ch1	channel 1 作为捕获源
capture_source_evsel0	evsel0 作为捕获源
capture_source_evsel1	evsel1 作为捕获源
capture_source_evsel2	evsel2 作为捕获源
capture_source_evsel3	evsel3 作为捕获源
capture_source_evsel4	evsel4 作为捕获源
capture_source_evsel5	evsel5 作为捕获源
capture_source_evsel6	evsel6 作为捕获源
capture_source_evsel7	evsel7 作为捕获源

### 结构体类型 `gptimer_counter_up_source_parameter_struct`

表 3-473. 结构体类型 `gptimer_counter_up_source_parameter_struct`

成员名称	功能描述
up_count_source_eti0	eti0 作为向上计数源
up_count_source_eti1	eti1 作为向上计数源
up_count_source_eti2	eti2 作为向上计数源
up_count_source_eti3	eti3 作为向上计数源
up_count_source_ch0	channel 0 作为向上计数源
up_count_source_ch1	channel 1 作为向上计数源
up_count_source_evsel0	evsel0 作为向上计数源
up_count_source_evsel1	evsel1 作为向上计数源
up_count_source_evsel2	evsel2 作为向上计数源
up_count_source_evsel3	evsel3 作为向上计数源

up_count_source_evsel 4	evsel4 作为向上计数源
up_count_source_evsel 5	evsel5 作为向上计数源
up_count_source_evsel 6	evsel6 作为向上计数源
up_count_source_evsel 7	evsel7 作为向上计数源

### 结构体类型 `gptimer_counter_down_source_parameter_struct`

表 3-474. 结构体类型 `gptimer_counter_down_source_parameter_struct`

成员名称	功能描述
down_count_source_eti 0	eti0 作为向下计数源
down_count_source_eti 1	eti1 作为向下计数源
down_count_source_eti 2	eti2 作为向下计数源
down_count_source_eti 3	eti3 作为向下计数源
down_count_source_ch 0	channel 0 作为向下计数源
down_count_source_ch 1	channel 1 作为向下计数源
down_count_source_ev sel0	evsel0 作为向下计数源
down_count_source_ev sel1	evsel1 作为向下计数源
down_count_source_ev sel2	evsel2 作为向下计数源
down_count_source_ev sel3	evsel3 作为向下计数源
down_count_source_ev sel4	evsel4 作为向下计数源
down_count_source_ev sel5	evsel5 作为向下计数源
down_count_source_ev sel6	evsel6 作为向下计数源
down_count_source_ev sel7	evsel7 作为向下计数源

## 结构体类型 `gptimer_oc_parameter_struct`

表 3-475. 结构体类型 `gptimer_oc_parameter_struct`

成员名称	功能描述
<code>chxcv_up_output_level</code>	向上计数 <code>chxcv</code> 匹配时输出电平
<code>chxcv_down_output_level</code>	向下计数 <code>chxcv</code> 匹配时输出电平
<code>chxcomv_add_up_output_level</code>	向上计数 <code>chxcomv_add</code> 匹配时输出电平
<code>chxcomv_add_down_output_level</code>	向下计数 <code>chxcomv_add</code> 匹配时输出电平
<code>period_end_output_level</code>	周期结束输出电平
<code>cnt_enable_disable_oc_en</code>	计数器启动或停止输出控制使能
<code>cnt_enable_output_level</code>	计数器使能输出初始电平
<code>cnt_disable_output_level</code>	计数器禁能输出电平
<code>output_stop_output_level</code>	输出停止输出电平
<code>force_duty_output_mode</code>	强制占空比输出模式
<code>force_duty_output_time</code>	强制占空比输出时刻
<code>force_duty_end_output_level</code>	强制占空比结束时输出电平
<code>compare_period_end_priority_control</code>	比较和周期结束优先级控制

## 结构体类型 `gptimer_com_oc_parameter_struct`

表 3-476. 结构体类型 `gptimer_com_oc_parameter_struct`

成员名称	功能描述
<code>complementary_mode</code>	互补输出模式
<code>deadtime_enable</code>	死区使能
<code>deadtime_mode</code>	死区模式
<code>rising_deadtime_value</code>	上升沿死区值
<code>falling_deadtime_value</code>	下降沿死区值
<code>rising_deadtime_shadow</code>	上升沿死区影子
<code>falling_deadtime_shadow</code>	下降沿死区影子

## 函数 gptimer\_deinit

函数 gptimer\_deinit 描述见下表：

**Table 3-1. Function gptimer\_deinit**

函数名称	gptimer_deinit
函数原型	void gptimer_deinit(uint32_t timer_periph)
功能描述	复位 GPTIMER
先决条件	-
被调用函数	rcu_periph_reset_enable(),rcu_periph_reset_disable()
输入参数{in}	
timer_periph	GPTIMER 外设
GPTIMERx	x=0,1
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* deinit GPTIMER0 */
```

```
gptimer_deinit(GPTIMER0);
```

## 函数 gptimer\_struct\_para\_init

函数 gptimer\_struct\_para\_init 描述见下表：

**Table 3-2. Function gptimer\_struct\_para\_init**

函数名称	gptimer_struct_para_init
函数原型	void gptimer_struct_para_init(gptimer_parameter_struct *initpara)
功能描述	初始化 GPTIMER 初始参数为默认值
先决条件	-
被调用函数	-
输入参数{in}	
initpara	参考 <a href="#">表 3-468. 结构体类型 gptimer_parameter_struct</a>
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* init parameter struct with a default value */
```

```
gptimer_parameter_struct para;
```

```
gptimer_struct_para_init(&para);
```

## 函数 gptimer\_init

函数 gptimer\_init 描述见下表：

**Table 3-3. Function gptimer\_init**

函数名称	gptimer_init
函数原型	void gptimer_init(uint32_t timer_periph, gptimer_parameter_struct *initpara)
功能描述	初始化 GPTIMER 计数器
先决条件	-
被调用函数	-
输入参数{in}	
timer_periph	GPTIMER 外设
GPTIMERx	x=0,1
输入参数{in}	
initpara	参考 <a href="#">表 3-468. 结构体类型 gptimer_parameter_struct</a>
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* initialize GPTIMER0 counter */
```

```
gptimer_parameter_struct para;
```

```
gptimer_init(GPTIMER0, para);
```

## 函数 gptimer\_register\_write\_protect\_enable

函数 gptimer\_register\_write\_protect\_enable 描述见下表：

**Table 3-4. Function gptimer\_register\_write\_protect\_enable**

函数名称	gptimer_register_write_protect_enable
函数原型	void gptimer_register_write_protect_enable(uint32_t timer_periph, uint32_t register_protect)
功能描述	使能 GPTIMER 软件写保护功能
先决条件	-
被调用函数	-
输入参数{in}	
timer_periph	GPTIMER 外设
GPTIMERx	x=0,1
输入参数{in}	

<b>register_protect</b>	寄存器写保护模式
<i>GPTIMER_WRITE_PR</i> <i>OTECT_SWEN</i>	GPTIMER_SWEN 寄存器写保护
<i>GPTIMER_WRITE_PR</i> <i>OTECT_SWDIS</i>	GPTIMER_SWDIS 寄存器写保护
<i>GPTIMER_WRITE_PR</i> <i>OTECT_SWRST</i>	GPTIMER_SWRST 寄存器写保护
<i>GPTIMER_WRITE_PR</i> <i>OTECT_OTHER</i>	其它寄存器写保护
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* enable GPTIMER0 register write protect function */
```

```
gptimer_register_write_protect_enable(GPTIMER0,  
GPTIMER_WRITE_PROTECT_OTHER);
```

### 函数 gptimer\_register\_write\_protect\_disable

函数 gptimer\_register\_write\_protect\_disable 描述见下表：

**Table 3-5. Function gptimer\_register\_write\_protect\_disable**

<b>函数名称</b>	gptimer_register_write_protect_disable
<b>函数原型</b>	void gptimer_register_write_protect_disable(uint32_t timer_periph, uint32_t register_protect)
<b>功能描述</b>	禁能 GPTIMER 软件写保护功能
<b>先决条件</b>	-
<b>被调用函数</b>	-
输入参数{in}	
<b>timer_periph</b>	GPTIMER 外设
<i>GPTIMERx</i>	x=0,1
输入参数{in}	
<b>register_protect</b>	寄存器写保护模式
<i>GPTIMER_WRITE_PR</i> <i>OTECT_SWEN</i>	GPTIMER_SWEN 寄存器写保护
<i>GPTIMER_WRITE_PR</i> <i>OTECT_SWDIS</i>	GPTIMER_SWDIS 寄存器写保护
<i>GPTIMER_WRITE_PR</i> <i>OTECT_SWRST</i>	GPTIMER_SWRST 寄存器写保护
<i>GPTIMER_WRITE_PR</i> <i>OTECT_OTHER</i>	其它寄存器写保护



输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* disable GPTIMER0 register write protect function */
```

```
gptimer_register_write_protect_disable(GPTIMER0,
GPTIMER_WRITE_PROTECT_OTHER);
```

### 函数 gptimer\_clock\_source\_select

函数 gptimer\_clock\_source\_select 描述见下表：

**Table 3-6. Function gptimer\_clock\_source\_select**

函数名称	gptimer_clock_source_select
函数原型	void gptimer_clock_source_select(uint32_t timer_periph, uint32_t clock_source)
功能描述	选择 GPTIMER 时钟源
先决条件	-
被调用函数	-
输入参数{in}	
timer_periph	GPTIMER 外设
GPTIMERx	x=0,1
输入参数{in}	
clock_source	时钟源选择
GPTIMER_CLOCK_SOURCE_CK_GPTIMER	选择 CK_GPTIMER 作为时钟源
GPTIMER_CLOCK_SOURCE_ETI0	选择 ETI0 作为时钟源
GPTIMER_CLOCK_SOURCE_ETI1	选择 ETI1 作为时钟源
GPTIMER_CLOCK_SOURCE_ETI2	选择 ETI2 作为时钟源
GPTIMER_CLOCK_SOURCE_ETI3	选择 ETI3 作为时钟源
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* select a GPTIMER0 clock source */
```

```
gptimer_clock_source_select(GPTIMER0, GPTIMER_CLOCK_SOURCE_CK_GPTIMER);
```

### 函数 gptimer\_clock\_polarity\_config

函数 gptimer\_clock\_polarity\_config 描述见下表：

**Table 3-7. Function gptimer\_clock\_polarity\_config**

函数名称	gptimer_clock_polarity_config
函数原型	void gptimer_clock_polarity_config(uint32_t timer_periph, uint32_t clock_polarity)
功能描述	配置 GPTIMER 时钟极性
先决条件	-
被调用函数	-
输入参数{in}	
timer_periph	GPTIMER 外设
GPTIMERx	x=0,1
输入参数{in}	
clock_polarity	时钟极性选择
GPTIMER_CLOCK_POLARITY_RISING	选择上升沿作为时钟极性
GPTIMER_CLOCK_POLARITY_FALLING	选择下降沿作为时钟极性
GPTIMER_CLOCK_POLARITY_BOTH	选择双边沿作为时钟极性
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* configure a GPTIMER0 clock polarity */
```

```
gptimer_clock_polarity_config(GPTIMER0, GPTIMER_CLOCK_POLARITY_RISING);
```

### 函数 gptimer\_counter\_software\_enable

函数 gptimer\_counter\_software\_enable 描述见下表：

**Table 3-8. Function gptimer\_counter\_software\_enable**

函数名称	gptimer_counter_software_enable
函数原型	void gptimer_counter_software_enable(uint32_t timer_periph, uint32_t timer_cnt)
功能描述	软件使能 GPTIMER 计数器
先决条件	-
被调用函数	-

输入参数{in}	
<b>timer_periph</b>	GPTIMER 外设
<i>GPTIMERx</i>	x=0,1
输入参数{in}	
<b>timer_cnt</b>	GPTIMER 计数器
<i>GPTIMER0_COUNT</i>	GPTIMER0 计数器
<i>GPTIMER1_COUNT</i>	GPTIMER1 计数器
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* software enable GPTIMER0 counter */
```

```
gptimer_counter_software_enable(GPTIMER0, GPTIMER0_COUNT);
```

### 函数 gptimer\_counter\_software\_disable

函数 gptimer\_counter\_software\_disable 描述见下表:

**Table 3-9. Function gptimer\_counter\_software\_disable**

<b>函数名称</b>	gptimer_counter_software_disable
<b>函数原型</b>	void gptimer_counter_software_disable(uint32_t timer_periph, uint32_t timer_cnt)
<b>功能描述</b>	软件禁能 GPTIMER 计数器
<b>先决条件</b>	-
<b>被调用函数</b>	-
输入参数{in}	
<b>timer_periph</b>	GPTIMER 外设
<i>GPTIMERx</i>	x=0,1
输入参数{in}	
<b>timer_cnt</b>	GPTIMER 计数器
<i>GPTIMER0_COUNT</i>	GPTIMER0 计数器
<i>GPTIMER1_COUNT</i>	GPTIMER1 计数器
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* software disable GPTIMER0 counter */
```

```
gptimer_counter_software_disable(GPTIMER0, GPTIMER0_COUNT);
```

**函数 gptimer\_counter\_software\_reset**

函数 gptimer\_counter\_software\_reset 描述见下表:

**Table 3-10. Function gptimer\_counter\_software\_reset**

函数名称	gptimer_counter_software_reset
函数原型	void gptimer_counter_software_reset(uint32_t timer_periph, uint32_t timer_cnt)
功能描述	软件复位 GPTIMR 计数器
先决条件	-
被调用函数	-
输入参数{in}	
timer_periph	GPTIMER 外设
GPTIMERx	x=0,1
输入参数{in}	
timer_cnt	GPTIMER 计数器
GPTIMER0_COUNT	GPTIMER0 计数器
GPTIMER1_COUNT	GPTIMER1 计数器
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* software reset GPTIMER0 counter */
```

```
gptimer_counter_software_reset(GPTIMER0, GPTIMER0_COUNT);
```

**函数 gptimer\_counter\_enable\_source\_struct\_para\_init**

函数 gptimer\_counter\_enable\_source\_struct\_para\_init 描述见下表:

**Table 3-11. Function gptimer\_counter\_enable\_source\_struct\_para\_init**

函数名称	gptimer_counter_enable_source_struct_para_init
函数原型	void gptimer_counter_enable_source_struct_para_init(gptimer_counter_enable_source_parameter_struct *counter_enable_source_para)
功能描述	初始化 GPTIMER 计数器使能源初始参数为默认值
先决条件	-
被调用函数	-
输入参数{in}	
counter_enable_source_struct_para	计数器使能源初始参数结构体, 参考 <a href="#">表 3-469. 结构体类型 gptimer_counter_enable_source_parameter_struct</a>
输出参数{out}	

-	-
返回值	
-	-

例如：

```
/* init parameter struct with a default value */
```

```
gptimer_counter_enable_source_parameter_struct para;
```

```
gptimer_counter_enable_source_struct_para_init(&para);
```

### 函数 gptimer\_counter\_enable\_source\_config

函数 gptimer\_counter\_enable\_source\_config 描述见下表：

**Table 3-12. Function gptimer\_counter\_enable\_source\_config**

函数名称	gptimer_counter_enable_source_config
函数原型	void gptimer_counter_enable_source_config(uint32_t timer_periph, gptimer_counter_enable_source_parameter_struct *counter_enable_source_para)
功能描述	初始化 GPTIMER 计数器使能源
先决条件	-
被调用函数	-
输入参数{in}	
timer_periph	GPTIMER 外设
GPTIMERx	x=0,1
输入参数{in}	
counter_enable_source_struct_para	计数器使能源初始参数结构体，参考 <a href="#">表 3-469. 结构体类型 gptimer_counter_enable_source_parameter_struct</a>
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* initialize GPTIMER0 counter enable source */
```

```
gptimer_counter_enable_source_parameter_struct para;
```

```
gptimer_counter_enable_source_config(GPTIMER0, &para);
```

### 函数 gptimer\_counter\_disable\_source\_struct\_para\_init

函数 gptimer\_counter\_disable\_source\_struct\_para\_init 描述见下表：

Table 3-13. Function gptimer\_counter\_disable\_source\_struct\_para\_init

函数名称	gptimer_counter_disable_source_struct_para_init
函数原型	void gptimer_counter_disable_source_struct_para_init(gptimer_counter_disable_source_parameter_struct *counter_disable_source_para)
功能描述	初始化 GPTIMER 计数器禁能源初始参数为默认值
先决条件	-
被调用函数	-
输入参数{in}	
counter_disable_source_para	计数器禁能源初始参数结构体，参考 <a href="#">表 3-470. 结构体类型 gptimer_counter_disable_source_parameter_struct</a>
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* init parameter struct with a default value */
```

```
gptimer_counter_disable_source_parameter_struct para;
```

```
gptimer_counter_disable_source_struct_para_init(&para);
```

### 函数 gptimer\_counter\_disable\_source\_config

函数 gptimer\_counter\_disable\_source\_config 描述见下表：

Table 3-14. Function gptimer\_counter\_disable\_source\_config

函数名称	gptimer_counter_disable_source_config
函数原型	void gptimer_counter_disable_source_config(uint32_t timer_periph, gptimer_counter_disable_source_parameter_struct *counter_disable_source_para)
功能描述	初始化 GPTIMER 计数器禁能源
先决条件	-
被调用函数	-
输入参数{in}	
timer_periph	GPTIMER 外设
GPTIMERx	x=0,1
输入参数{in}	
counter_disable_source_para	计数器禁能源初始参数结构体，参考 <a href="#">表 3-470. 结构体类型 gptimer_counter_disable_source_parameter_struct</a>
输出参数{out}	
-	-
返回值	

-	-
---	---

例如：

```
/* initialize GPTIMER0 counter disable source */
```

```
gptimer_counter_disable_source_parameter_struct para;
```

```
gptimer_counter_disable_source_config(GPTIMER0, &para);
```

### 函数 gptimer\_counter\_reset\_source\_struct\_para\_init

函数 gptimer\_counter\_reset\_source\_struct\_para\_init 描述见下表：

**Table 3-15. Function gptimer\_counter\_reset\_source\_struct\_para\_init**

函数名称	gptimer_counter_reset_source_struct_para_init
函数原型	void gptimer_counter_reset_source_struct_para_init(gptimer_counter_reset_source_parameter_struct *counter_reset_source_para)
功能描述	初始化 GPTIMER 计数器复位源初始参数为默认值
先决条件	-
被调用函数	-
输入参数{in}	
counter_reset_source_para	计数器复位源初始参数结构体，参考 <a href="#">表 3-471. 结构体类型 gptimer_counter_reset_source_parameter_struct</a>
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* init parameter struct with a default value */
```

```
gptimer_counter_reset_source_parameter_struct para;
```

```
gptimer_counter_reset_source_struct_para_init(&para);
```

### 函数 gptimer\_counter\_reset\_source\_config

函数 gptimer\_counter\_reset\_source\_config 描述见下表：

**Table 3-16. Function gptimer\_counter\_reset\_source\_config**

函数名称	gptimer_counter_reset_source_config
函数原型	void gptimer_counter_reset_source_config(uint32_t timer_periph, gptimer_counter_reset_source_parameter_struct *counter_reset_source_para)
功能描述	初始化 GPTIMER 计数器复位源

先决条件	-
被调用函数	-
输入参数{in}	
timer_periph	GPTIMER 外设
GPTIMERx	x=0,1
输入参数{in}	
counter_reset_source _para	计数器复位源初始参数结构体，参考 <a href="#">表 3-471. 结构体类型</a> <a href="#">gptimer_counter_reset_source_parameter_struct</a>
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* initialize GPTIMER0 counter reset source */
```

```
gptimer_counter_rest_source_parameter_struct para;
```

```
gptimer_counter_reset_source_config(GPTIMER0, &para);
```

### 函数 gptimer\_counter\_up\_source\_struct\_para\_init

函数 gptimer\_counter\_up\_source\_struct\_para\_init 描述见下表：

**Table 3-17. Function gptimer\_counter\_up\_source\_struct\_para\_init**

函数名称	gptimer_counter_up_source_struct_para_init
函数原型	void gptimer_counter_up_source_struct_para_init(gptimer_counter_up_source _parameter_struct *counter_up_source_para)
功能描述	初始化 GPTIMER 计数器向上计数源初始参数为默认值
先决条件	-
被调用函数	-
输入参数{in}	
counter_up_source_p ara	计数器向上计数源初始参数结构体，参考 <a href="#">表 3-473. 结构体类型</a> <a href="#">gptimer_counter_up_source_parameter_struct</a>
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* init parameter struct with a default value */
```

```
gptimer_counter_up_source_parameter_struct para;
```



```
gptimer_counter_up_source_struct_para_init(&para);
```

## 函数 gptimer\_counter\_up\_source\_config

函数 gptimer\_counter\_up\_source\_config 描述见下表：

**Table 3-18. Function gptimer\_counter\_up\_source\_config**

函数名称	gptimer_counter_up_source_config
函数原型	void gptimer_counter_up_source_config(uint32_t timer_periph, gptimer_counter_up_source_parameter_struct *counter_up_source_para)
功能描述	初始化 GPTIMER 计数器向上计数源
先决条件	-
被调用函数	-
输入参数{in}	
timer_periph	GPTIMER 外设
GPTIMERx	x=0,1
输入参数{in}	
counter_up_source_p ara	计数器向上计数源初始参数结构体，参考 <a href="#">表 3-473. 结构体类型 gptimer_counter_up_source_parameter_struct</a>
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* initialize GPTIMER0 counter up source */
```

```
gptimer_counter_up_source_parameter_struct para;
```

```
gptimer_counter_up_source_config(GPTIMER0, &para);
```

## 函数 gptimer\_counter\_down\_source\_struct\_para\_init

函数 gptimer\_counter\_down\_source\_struct\_para\_init 描述见下表：

**Table 3-19. Function gptimer\_counter\_down\_source\_struct\_para\_init**

函数名称	gptimer_counter_down_source_struct_para_init
函数原型	void gptimer_counter_down_source_struct_para_init(gptimer_counter_down_s ource_parameter_struct *counter_down_source_para)
功能描述	初始化 GPTIMER 计数器向下计数源初始参数为默认值
先决条件	-
被调用函数	-
输入参数{in}	
counter_down_source	计数器向下计数源初始参数结构体，参考 <a href="#">表 3-474. 结构体类型</a>

<b>_para</b>	<a href="#"><u>gptimer_counter_down_source_parameter_struct</u></a>
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* init parameter struct with a default value */
```

```
gptimer_counter_down_source_parameter_struct para;
```

```
gptimer_counter_down_source_struct_para_init(&para);
```

### 函数 gptimer\_counter\_down\_source\_config

函数 gptimer\_counter\_down\_source\_config 描述见下表：

**Table 3-20. Function gptimer\_counter\_down\_source\_config**

函数名称	gptimer_counter_down_source_config
函数原型	void gptimer_counter_down_source_config(uint32_t timer_periph, gptimer_counter_down_source_parameter_struct *counter_down_source_para)
功能描述	初始化 GPTIMER 计数器向下计数源
先决条件	-
被调用函数	-
输入参数{in}	
timer_periph	GPTIMER 外设
GPTIMERx	x=0,1
输入参数{in}	
counter_down_source _para	计数器向下计数源初始参数结构体，参考 <a href="#">表 3-474. 结构体类型</a> <a href="#"><u>gptimer_counter_down_source_parameter_struct</u></a>
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* initialize GPTIMER0 counter down source */
```

```
gptimer_counter_down_source_parameter_struct para;
```

```
gptimer_counter_down_source_config(GPTIMER0, &para);
```

### 函数 gptimer\_counter\_up\_input\_level\_source\_select

函数 gptimer\_counter\_up\_input\_level\_source\_select 描述见下表：

Table 3-21. Function gptimer\_counter\_up\_input\_level\_source\_select

函数名称	gptimer_counter_up_input_level_source_select
函数原型	void gptimer_counter_up_input_level_source_select(uint32_t timer_periph, uint32_t input_source)
功能描述	选择 GPTIMER 计数器向上计数输入电平源
先决条件	-
被调用函数	-
输入参数{in}	
timer_periph	GPTIMER 外设
GPTIMERx	x=0,1
输入参数{in}	
input_source	计数器向上计数输入电平源
GPTIMER_COUNTER_UP_DISABLE	计数器向上计数输入电平源禁能
GPTIMER_COUNTER_UP_CH0_LOW	channel 0 低电平作为向上计数输入电平源使能
GPTIMER_COUNTER_UP_CH0_HIGH	channel 0 高电平作为向上计数输入电平源使能
GPTIMER_COUNTER_UP_CH1_LOW	channel 1 低电平作为向上计数输入电平源使能
GPTIMER_COUNTER_UP_CH1_HIGH	channel 1 高电平作为向上计数输入电平源使能
GPTIMER_COUNTER_UP_ETI0_LOW	eti0 低电平作为向上计数输入电平源使能
GPTIMER_COUNTER_UP_ETI0_HIGH	eti0 高电平作为向上计数输入电平源使能
GPTIMER_COUNTER_UP_ETI1_LOW	eti1 低电平作为向上计数输入电平源使能
GPTIMER_COUNTER_UP_ETI1_HIGH	eti1 高电平作为向上计数输入电平源使能
GPTIMER_COUNTER_UP_ETI2_LOW	eti2 低电平作为向上计数输入电平源使能
GPTIMER_COUNTER_UP_ETI2_HIGH	eti2 高电平作为向上计数输入电平源使能
GPTIMER_COUNTER_UP_ETI3_LOW	eti3 低电平作为向上计数输入电平源使能
GPTIMER_COUNTER_UP_ETI3_HIGH	eti3 高电平作为向上计数输入电平源使能
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* select counter up input level source */
```

```
gptimer_counter_up_input_level_source_select(GPTIMER0,
GPTIMER_COUNTER_UP_CH0_HIGH);
```

### 函数 gptimer\_counter\_down\_input\_level\_source\_select

函数 gptimer\_counter\_down\_input\_level\_source\_select 描述见下表:

**Table 3-22. Function gptimer\_counter\_down\_input\_level\_source\_select**

函数名称	gptimer_counter_down_input_level_source_select
函数原型	void gptimer_counter_down_input_level_source_select(uint32_t timer_periph, uint32_t input_source)
功能描述	选择 GPTIMER 计数器向下计数输入电平源
先决条件	-
被调用函数	-
输入参数{in}	
timer_periph	GPTIMER 外设
GPTIMERx	x=0,1
输入参数{in}	
input_source	计数器向下技术输入电平源
GPTIMER_COUNTER_DOWN_DISABLE	计数器向下技术输入电平源禁能
GPTIMER_COUNTER_DOWN_CH0_LOW	channel 0 低电平作为向下计数输入电平源使能
GPTIMER_COUNTER_DOWN_CH0_HIGH	channel 0 高电平作为向下计数输入电平源使能
GPTIMER_COUNTER_DOWN_CH1_LOW	channel 1 低电平作为向下计数输入电平源使能
GPTIMER_COUNTER_DOWN_CH1_HIGH	channel 1 高电平作为向下计数输入电平源使能
GPTIMER_COUNTER_DOWN_ETI0_LOW	eti0 低电平作为向下计数输入电平源使能
GPTIMER_COUNTER_DOWN_ETI0_HIGH	eti0 高电平作为向下计数输入电平源使能
GPTIMER_COUNTER_DOWN_ETI1_LOW	eti1 低电平作为向下计数输入电平源使能
GPTIMER_COUNTER_DOWN_ETI1_HIGH	eti1 高电平作为向下计数输入电平源使能
GPTIMER_COUNTER_DOWN_ETI2_LOW	eti2 低电平作为向下计数输入电平源使能
GPTIMER_COUNTER_DOWN_ETI2_HIGH	eti2 高电平作为向下计数输入电平源使能

<i>DOWN_ETI2_HIGH</i>	
<i>GPTIMER_COUNTER_DOWN_ETI3_LOW</i>	eti3 低电平作为向下计数输入电平源使能
<i>GPTIMER_COUNTER_DOWN_ETI3_HIGH</i>	eti3 高电平作为向下计数输入电平源使能
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* select counter down input level source */
```

```
gptimer_counter_down_input_level_source_select(GPTIMER0,
GPTIMER_COUNTER_DOWN_CH0_LOW);
```

### 函数 gptimer\_enable

函数 gptimer\_enable 描述见下表：

**Table 3-23. Function gptimer\_enable**

函数名称	gptimer_enable
函数原型	void gptimer_enable(uint32_t timer_periph)
功能描述	使能 GPTIMER
先决条件	-
被调用函数	-
输入参数{in}	
timer_periph	GPTIMER 外设
<i>GPTIMERx</i>	x=0,1
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* enable GPTIMER0 */
```

```
gptimer_enable(GPTIMER0);
```

### 函数 gptimer\_disable

函数 gptimer\_disable 描述见下表：

**Table 3-24. Function gptimer\_disable**

函数名称	gptimer_disable
------	-----------------

函数原型	void gptimer_disable(uint32_t timer_periph)
功能描述	禁能 GPTIMER
先决条件	-
被调用函数	-
输入参数{in}	
timer_periph	GPTIMER 外设
GPTIMERx	x=0,1
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* disable GPTIMER0 */
gptimer_disable(GPTIMER0);
```

### 函数 gptimer\_auto\_reload\_shadow\_enable

函数 gptimer\_auto\_reload\_shadow\_enable 描述见下表:

**Table 3-25. Function gptimer\_auto\_reload\_shadow\_enable**

函数名称	gptimer_auto_reload_shadow_enable
函数原型	void gptimer_auto_reload_shadow_enable(uint32_t timer_periph)
功能描述	使能自动重装载影子功能
先决条件	-
被调用函数	-
输入参数{in}	
timer_periph	GPTIMER 外设
GPTIMERx	x=0,1
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* enable GPTIMER0 the auto reload shadow function */
gptimer_auto_reload_shadow_enable(GPTIMER0);
```

### 函数 gptimer\_auto\_reload\_shadow\_disable

函数 gptimer\_auto\_reload\_shadow\_disable 描述见下表:

**Table 3-26. Function gptimer\_auto\_reload\_shadow\_disable**

函数名称	gptimer_auto_reload_shadow_disable
函数原型	void gptimer_auto_reload_shadow_disable(uint32_t timer_periph)
功能描述	禁能自动重载影子功能
先决条件	-
被调用函数	-
输入参数{in}	
timer_periph	GPTIMER 外设
GPTIMERx	x=0,1
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* disable GPTIMER0 the auto reload shadow function */
```

```
gptimer_auto_reload_shadow_disable(GPTIMER0);
```

### 函数 gptimer\_update\_event\_enable

函数 gptimer\_update\_event\_enable 描述见下表：

**Table 3-27. Function gptimer\_update\_event\_enable**

函数名称	gptimer_update_event_enable
函数原型	void gptimer_update_event_enable(uint32_t timer_periph)
功能描述	使能更新事件
先决条件	-
被调用函数	-
输入参数{in}	
timer_periph	GPTIMER 外设
GPTIMERx	x=0,1
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* enable GPTIMER0 the update event */
```

```
gptimer_update_event_enable(GPTIMER0);
```

### 函数 gptimer\_update\_event\_disable

函数 gptimer\_update\_event\_disable 描述见下表：

Table 3-28. Function gptimer\_update\_event\_disable

函数名称	gptimer_update_event_disable
函数原型	void gptimer_update_event_disable(uint32_t timer_periph)
功能描述	禁能更新事件
先决条件	-
被调用函数	-
输入参数{in}	
timer_periph	GPTIMER 外设
GPTIMERx	x=0,1
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* disable GPTIMER0 the update event */
gptimer_update_event_disable(GPTIMER0);
```

### 函数 gptimer\_counter\_alignment

函数 gptimer\_counter\_alignment 描述见下表：

Table 3-29. Function gptimer\_counter\_alignment

函数名称	gptimer_counter_alignment
函数原型	void gptimer_counter_alignment(uint32_t timer_periph, uint16_t aligned)
功能描述	设置计数器对齐模式
先决条件	-
被调用函数	-
输入参数{in}	
timer_periph	GPTIMER 外设
GPTIMERx	x=0,1
输入参数{in}	
aligned	计数器对齐模式
GPTIMER_COUNTER_EDGE	边沿对齐
GPTIMER_COUNTER_CENTER	中央对齐
输出参数{out}	
-	-
返回值	
-	-

例如：



```
/* set GPTIMER0 counter alignment mode */
```

```
gptimer_counter_alignment(GPTIMER0);
```

### 函数 gptimer\_counter\_up\_direction

函数 gptimer\_counter\_up\_direction 描述见下表：

**Table 3-30. Function gptimer\_counter\_up\_direction**

函数名称	gptimer_counter_up_direction
函数原型	void gptimer_counter_up_direction(uint32_t timer_periph)
功能描述	设置计数器向上计数方向
先决条件	-
被调用函数	-
输入参数{in}	
timer_periph	GPTIMER 外设
GPTIMERx	x=0,1
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* set GPTIMER0 counter up direction */
```

```
gptimer_counter_up_direction(GPTIMER0);
```

### 函数 gptimer\_counter\_down\_direction

函数 gptimer\_counter\_down\_direction 描述见下表：

**Table 3-31. Function gptimer\_counter\_down\_direction**

函数名称	gptimer_counter_down_direction
函数原型	void gptimer_counter_down_direction(uint32_t timer_periph)
功能描述	设置计数器向下计数方向
先决条件	-
被调用函数	-
输入参数{in}	
timer_periph	GPTIMER 外设
GPTIMERx	x=0,1
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* set GPTIMER0 counter down direction */
```

```
gptimer_counter_down_direction(GPTIMER0);
```

### 函数 gptimer\_counter\_direction\_force\_set\_enable

函数 gptimer\_counter\_direction\_force\_set\_enable 描述见下表：

**Table 3-32. Function gptimer\_counter\_direction\_force\_set\_enable**

函数名称	gptimer_counter_direction_force_set_enable
函数原型	void gptimer_counter_direction_force_set_enable(uint32_t timer_periph)
功能描述	使能计数器方向强制设置
先决条件	-
被调用函数	-
输入参数{in}	
timer_periph	GPTIMER 外设
GPTIMERx	x=0,1
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* enable GPTIMER0 counter direction force set */
```

```
gptimer_counter_direction_force_set_enable(GPTIMER0);
```

### 函数 gptimer\_counter\_direction\_force\_set\_disable

函数 gptimer\_counter\_direction\_force\_set\_disable 描述见下表：

**Table 3-33. Function gptimer\_counter\_direction\_force\_set\_disable**

函数名称	gptimer_counter_direction_force_set_disable
函数原型	void gptimer_counter_direction_force_set_disable(uint32_t timer_periph)
功能描述	禁能计数器方向强制设置
先决条件	-
被调用函数	-
输入参数{in}	
timer_periph	GPTIMER 外设
GPTIMERx	x=0,1
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* disable GPTIMER0 counter direction force set */
```

```
gptimer_counter_direction_force_set_disable(GPTIMER0);
```

### 函数 gptimer\_register\_global\_update\_source\_select

函数 gptimer\_register\_global\_update\_source\_select 描述见下表：

**Table 3-34. Function gptimer\_register\_global\_update\_source\_select**

函数名称	gptimer_register_global_update_source_select
函数原型	void gptimer_register_global_update_source_select(uint32_t timer_periph, uint32_t source)
功能描述	选择寄存器全局更新源
先决条件	-
被调用函数	-
输入参数{in}	
timer_periph	GPTIMER 外设
GPTIMERx	x=0,1
输入参数{in}	
source	全局更新源
GPTIMER_GLOBAL_UPS_SEL_FLOW	上溢或下溢事件作为全局更新源事件
GPTIMER_GLOBAL_UPS_SEL_OVERFLOW	上溢事件作为全局更新源事件
GPTIMER_GLOBAL_UPS_SEL_UNDERFLOW	下溢事件作为全局更新源事件
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* select register global update source */
```

```
gptimer_register_global_update_source_select(GPTIMER0,  
GPTIMER_GLOBAL_UPS_SEL_FLOW);
```

### 函数 gptimer\_register\_local\_update\_source\_select

函数 gptimer\_register\_local\_update\_source\_select 描述见下表：

**Table 3-35. Function gptimer\_register\_local\_update\_source\_select**

函数名称	gptimer_register_local_update_source_select
函数原型	void gptimer_register_local_update_source_select(uint32_t timer_periph, uint32_t reg, uint32_t source)

功能描述	选择寄存器本地更新源
先决条件	-
被调用函数	-
输入参数{in}	
timer_periph	GPTIMER 外设
GPTIMERx	x=0,1
输入参数{in}	
reg	本地更新源
GPTIMER_LOCAL_UP_CH0CV	CH0CV 本地更新
GPTIMER_LOCAL_UP_CH0COMV_ADD	CH0COMV_ADD 本地更新
GPTIMER_LOCAL_UP_CH1CV	CH1CV 本地更新
GPTIMER_LOCAL_UP_CH1COMV_ADD	CH1COMV_ADD 本地更新
GPTIMER_LOCAL_UP_ADCCR1	ADC triggered comparison value 1 本地更新
GPTIMER_LOCAL_UP_ADCCR2	ADC triggered comparison value 2 本地更新
GPTIMER_LOCAL_UP_CAR	car 本地更新
GPTIMER_LOCAL_UP_CH0	channel 0 输出模式本地更新
GPTIMER_LOCAL_UP_CH1	channel 1 输出模式本地更新
输入参数{in}	
source	更新源
GPTIMER_LOCAL_UP_S_SEL_DISABLE	本地更新事件源选择禁能
GPTIMER_LOCAL_UP_S_SEL_OVERFLOW	上溢事件作为本地更新事件源
GPTIMER_LOCAL_UP_S_SEL_UNDERFLOW	下溢事件作为本地更新事件源
GPTIMER_LOCAL_UP_S_SEL_FLOW	溢出事件作为本地更新事件源
GPTIMER_LOCAL_UP_S_SEL_CHCOM	通道比较事件作为本地更新事件源（只有 GPTIMER_LOCAL_UP_CH0COMV_ADD 有效）
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* select register local update source */
```

```
gptimer_register_local_update_source_select(GPTIMER0, GPTIMER_LOCAL_UP_CH0CV,
GPTIMER_LOCAL_UPS_SEL_OVERFLOW);
```

### 函数 gptimer\_prescaler\_config

函数 gptimer\_prescaler\_config 描述见下表：

**Table 3-36. Function gptimer\_prescaler\_config**

函数名称	gptimer_prescaler_config
函数原型	void gptimer_prescaler_config(uint32_t timer_periph, uint16_t prescaler)
功能描述	配置预分频
先决条件	-
被调用函数	-
输入参数{in}	
timer_periph	GPTIMER 外设
GPTIMERx	x=0,1
输入参数{in}	
prescaler	分频值，0~65535
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* configure GPTIMER0 prescaler */
```

```
gptimer_prescaler_config(GPTIMER0, 0x3fff);
```

### 函数 gptimer\_update\_repetition\_value\_config

函数 gptimer\_update\_repetition\_value\_config 描述见下表：

**Table 3-37. Function gptimer\_update\_repetition\_value\_config**

函数名称	gptimer_update_repetition_value_config
函数原型	void gptimer_update_repetition_value_config(uint32_t timer_periph, uint16_t repetition, uint32_t load_mode)
功能描述	配置更新重复计数器值
先决条件	-
被调用函数	-
输入参数{in}	
timer_periph	GPTIMER 外设
GPTIMERx	x=0,1

输入参数{in}	
<b>repetition</b>	计数器重复值, 0~4095
输入参数{in}	
<b>load_mode</b>	加载模式
<i>GPTIMER_UPREP_LO AD_IMMEDIATELY</i>	重复值立即加载
<i>GPTIMER_UPREP_LO AD_NEXT_UPEVENT</i>	重复值加载在下次更新事件
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* configure GPTIMER0 repetition register value */
```

```
gptimer_update_repetition_value_config(GPTIMER0, 0,
GPTIMER_UPREP_LOAD_IMMEDIATELY);
```

### 函数 gptimer\_update\_repetition\_counter\_read

函数 gptimer\_update\_repetition\_counter\_read 描述见下表:

**Table 3-38. Function gptimer\_update\_repetition\_counter\_read**

<b>函数名称</b>	gptimer_update_repetition_counter_read
<b>函数原型</b>	uint16_t gptimer_update_repetition_counter_read(uint32_t timer_periph)
<b>功能描述</b>	读取更新重复计数器值
<b>先决条件</b>	-
<b>被调用函数</b>	-
输入参数{in}	
<b>timer_periph</b>	GPTIMER 外设
<i>GPTIMERx</i>	x=0,1
输出参数{out}	
-	-
返回值	
<b>uint16_t</b>	重复值

例如:

```
/* read GPTIMER0 repetition register value */
```

```
uint16_t value;
```

```
value = gptimer_update_repetition_counter_read (GPTIMER0);
```

## 函数 gptimer\_autoreload\_value\_config

函数 gptimer\_autoreload\_value\_config 描述见下表：

**Table 3-39. Function gptimer\_autoreload\_value\_config**

函数名称	gptimer_autoreload_value_config
函数原型	void gptimer_autoreload_value_config(uint32_t timer_periph, uint32_t autoreload)
功能描述	配置自动重装载寄存器值
先决条件	-
被调用函数	-
输入参数{in}	
timer_periph	GPTIMER 外设
GPTIMERx	x=0,1
输入参数{in}	
autoreload	the counter auto-reload value,0~65535
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* configure GPTIMER0 autoreload register value */
gptimer_autoreload_value_config (GPTIMER0, 0xff);
```

## 函数 gptimer\_counter\_value\_config

函数 gptimer\_counter\_value\_config 描述见下表：

**Table 3-40. Function gptimer\_counter\_value\_config**

函数名称	gptimer_counter_value_config
函数原型	void gptimer_counter_value_config(uint32_t timer_periph, uint32_t counter)
功能描述	配置计数器寄存器值
先决条件	-
被调用函数	-
输入参数{in}	
timer_periph	GPTIMER 外设
GPTIMERx	x=0,1
输入参数{in}	
counter	计数器值，0~65535
输出参数{out}	
-	-

返回值	
-	-

例如：

```
/* configure GPTIMER0 counter register value */
```

```
gptimer_counter_value_config (GPTIMER0, 0xff);
```

### 函数 gptimer\_counter\_read

函数 gptimer\_counter\_read 描述见下表：

**Table 3-41. Function gptimer\_counter\_read**

函数名称	gptimer_counter_read
函数原型	uint32_t gptimer_counter_read(uint32_t timer_periph)
功能描述	读取计数器值
先决条件	-
被调用函数	-
输入参数{in}	
timer_periph	GPTIMER 外设
GPTIMERx	x=0,1
输出参数{out}	
-	-
返回值	
uint32_t	计数器值

例如：

```
/* read GPTIMER0 counter value */
```

```
uint32_t value;
```

```
value = gptimer_counter_read(GPTIMER0);
```

### 函数 gptimer\_prescaler\_read

函数 gptimer\_prescaler\_read 描述见下表：

**Table 3-42. Function gptimer\_prescaler\_read**

函数名称	gptimer_prescaler_read
函数原型	uint16_t gptimer_prescaler_read(uint32_t timer_periph)
功能描述	读取预分频值
先决条件	-
被调用函数	-
输入参数{in}	
timer_periph	GPTIMER 外设
GPTIMERx	x=0,1



输出参数{out}	
-	-
返回值	
uint16_t	预分频寄存器值

例如:

```
/* read GPTIMER0 prescaler value */
```

```
uint16_t value;
```

```
value = gptimer_prescaler_read (GPTIMER0);
```

### 函数 gptimer\_single\_pulse\_mode\_config

函数 gptimer\_single\_pulse\_mode\_config 描述见下表:

**Table 3-43. Function gptimer\_single\_pulse\_mode\_config**

函数名称	gptimer_single_pulse_mode_config
函数原型	void gptimer_single_pulse_mode_config(uint32_t timer_periph, uint32_t spmode)
功能描述	配置单脉冲模式
先决条件	-
被调用函数	-
输入参数{in}	
timer_periph	GPTIMER 外设
GPTIMERx	x=0,1
输入参数{in}	
spmode	单脉冲模式选择
GPTIMER_SP_MODE_SINGLE	单脉冲模式
GPTIMER_SP_MODE_REPETITIVE	重复脉冲模式
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* configure GPTIMER0 single pulse mode */
```

```
gptimer_single_pulse_mode_config (GPTIMER0, GPTIMER_SP_MODE_SINGLE);
```

### 函数 gptimer\_dma\_enable

函数 gptimer\_dma\_enable 描述见下表:

Table 3-44. Function gptimer\_dma\_enable

函数名称	gptimer_dma_enable
函数原型	void gptimer_dma_enable(uint32_t timer_periph, uint32_t dma)
功能描述	使能 DMA
先决条件	-
被调用函数	-
输入参数{in}	
timer_periph	GPTIMER 外设
GPTIMERx	x=0,1
输入参数{in}	
dma	DMA 响应
GPTIMER_DMA_CH0D	channel 0 捕获/比较 DMA 响应
GPTIMER_DMA_CH1D	channel 1 捕获/比较 DMA 响应
GPTIMER_DMA_CH0C OMADD	channel 0 附加比较 DMA 响应
GPTIMER_DMA_CH1C OMADD	channel 1 附加比较 DMA 响应
GPTIMER_DMA_REPE D	重复计数器结束 DMA 响应
GPTIMER_DMA_OVER FLOWD	计数器上溢 DMA 响应
GPTIMER_DMA_UNDE RFLOWD	计数器下溢 DMA 响应
GPTIMER_DMA_OVUN FLOWD	计数器上溢/下溢 DMA 响应
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* enable DMA */
```

```
gptimer_dma_enable (GPTIMER0, GPTIMER_DMA_CH0D);
```

### 函数 gptimer\_dma\_disable

函数 gptimer\_dma\_disable 描述见下表：

Table 3-45. Function gptimer\_dma\_disable

函数名称	gptimer_dma_disable
函数原型	void gptimer_dma_disable(uint32_t timer_periph, uint32_t dma)
功能描述	禁能 DMA

先决条件	-
被调用函数	-
输入参数{in}	
timer_periph	GPTIMER 外设
GPTIMERx	x=0,1
输入参数{in}	
dma	DMA 响应
GPTIMER_DMA_CH0D	channel 0 捕获/比较 DMA 响应
GPTIMER_DMA_CH1D	channel 1 捕获/比较 DMA 响应
GPTIMER_DMA_CH0C OMADD	channel 0 附加比较 DMA 响应
GPTIMER_DMA_CH1C OMADD	channel 1 附加比较 DMA 响应
GPTIMER_DMA_REPE D	重复计数器结束 DMA 响应
GPTIMER_DMA_OVER FLOWD	计数器上溢 DMA 响应
GPTIMER_DMA_UNDE RFLOWD	计数器下溢 DMA 响应
GPTIMER_DMA_OVUN FLOWD	计数器上溢/下溢 DMA 响应
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* disable DMA */
```

```
gptimer_dma_disable (GPTIMER0, GPTIMER_DMA_CH0D);
```

### 函数 gptimer\_dma\_transfer\_config

函数 gptimer\_dma\_transfer\_config 描述见下表:

**Table 3-46. Function gptimer\_dma\_transfer\_config**

函数名称	gptimer_dma_transfer_config
函数原型	void gptimer_dma_transfer_config(uint32_t timer_periph, uint32_t dma_baseaddr, uint32_t dma_lenth)
功能描述	配置 DMA 传输
先决条件	-
被调用函数	-
输入参数{in}	

<b>timer_periph</b>	GPTIMER 外设
<i>GPTIMERx</i>	x=0,1
<b>输入参数{in}</b>	
<b>dma_baseaddr</b>	DMA 传输地址
<i>GPTIMER_DMACFG_D MATA_WP</i>	DMA 传输地址是 GPTIMER_WP
<i>GPTIMER_DMACFG_D MATA_SWEN</i>	DMA 传输地址是 GPTIMER_SWEN
<i>GPTIMER_DMACFG_D MATA_SWDIS</i>	DMA 传输地址是 GPTIMER_SWDIS
<i>GPTIMER_DMACFG_D MATA_SWRST</i>	DMA 传输地址是 GPTIMER_SWRST
<i>GPTIMER_DMACFG_D MATA_ESSEL</i>	DMA 传输地址是 GPTIMER_ESSEL
<i>GPTIMER_DMACFG_D MATA_DSSEL</i>	DMA 传输地址是 GPTIMER_DSSEL
<i>GPTIMER_DMACFG_D MATA_RSSEL</i>	DMA 传输地址是 GPTIMER_RSSEL
<i>GPTIMER_DMACFG_D MATA_CH0CSSEL</i>	DMA 传输地址是 GPTIMER_CH0CSSEL
<i>GPTIMER_DMACFG_D MATA_CH1CSSEL</i>	DMA 传输地址是 GPTIMER_CH0CSSEL
<i>GPTIMER_DMACFG_D MATA_CTL1</i>	DMA 传输地址是 GPTIMER_CTL0
<i>GPTIMER_DMACFG_D MATA_UPSSR</i>	DMA 传输地址是 GPTIMER_CUPEVSSEL
<i>GPTIMER_DMACFG_D MATA_DNSSR</i>	DMA 传输地址是 GPTIMER_CDNEVSSEL
<i>GPTIMER_DMACFG_D MATA_CH0CTL</i>	DMA 传输地址是 GPTIMER_CH0CTL
<i>GPTIMER_DMACFG_D MATA_CH1CTL</i>	DMA 传输地址是 GPTIMER_CH1CTL
<i>GPTIMER_DMACFG_D MATA_CHCTL</i>	DMA 传输地址是 GPTIMER_CHCTL
<i>GPTIMER_DMACFG_D MATA_DMAINTEN</i>	DMA 传输地址是 GPTIMER_DMAINTEN
<i>GPTIMER_DMACFG_D MATA_INTF</i>	DMA 传输地址是 GPTIMER_INTF
<i>GPTIMER_DMACFG_D MATA_UPSSEL</i>	DMA 传输地址是 GPTIMER_UPSSEL
<i>GPTIMER_DMACFG_D MATA_CNT</i>	DMA 传输地址是 GPTIMER_CNT

GPTIMER_DMACFG_D MATA_PSC	DMA 传输地址是 GPTIMER_PSC
GPTIMER_DMACFG_D MATA_CAR	DMA 传输地址是 GPTIMER_CAR
GPTIMER_DMACFG_D MATA_CH0CV	DMA 传输地址是 GPTIMER_CH0CV
GPTIMER_DMACFG_D MATA_CH1CV	DMA 传输地址是 GPTIMER_CH1CV
GPTIMER_DMACFG_D MATA_CH0COMV_ADD D	DMA 传输地址是 GPTIMER_CH0COMV_ADD
GPTIMER_DMACFG_D MATA_CH1COMV_ADD D	DMA 传输地址是 GPTIMER_CH1COMV_ADD
GPTIMER_DMACFG_D MATA_DTCTL	DMA 传输地址是 GPTIMER_DTCTL
GPTIMER_DMACFG_D MATA_ADCTL	DMA 传输地址是 GPTIMER_ADCTL
GPTIMER_DMACFG_D MATA_ADCCR1	DMA 传输地址是 GPTIMER_ADCCR1
GPTIMER_DMACFG_D MATA_ADCCR2	DMA 传输地址是 GPTIMER_ADCCR2
GPTIMER_DMACFG_D MATA_ADCTRGS	DMA 传输地址是 GPTIMER_ADCTRGS
GPTIMER_DMACFG_D MATA_ADDINTSCTL0	DMA 传输地址是 GPTIMER_ADDINTSCTL0
GPTIMER_DMACFG_D MATA_ADDINTSCTL1	DMA 传输地址是 GPTIMER_ADDINTSCTL1
GPTIMER_DMACFG_D MATA_IADCTSS	DMA 传输地址是 GPTIMER_IADCTSS
GPTIMER_DMACFG_D MATA_CREP	DMA 传输地址是 GPTIMER_CREP
GPTIMER_DMACFG_D MATA_SYNRSTCTL	DMA 传输地址是 GPTIMER_SYNRSTCTL
输入参数{in}	
dma_lenth	DMA 传输长度
GPTIMER_DMACFG_D MATC_xTRANSFER(x= 1..35)	DMA 传输 x 次
输出参数{out}	
-	-
返回值	

-	-
---	---

例如：

```
/* configure the GPTIMER0 DMA transfer */
```

```
gptimer_dma_transfer_config (GPTIMER0, GPTIMER_DMACFG_DMATA_ADDINTSCTL0,  
GPTIMER_DMACFG_DMATC_2TRANSFER);
```

### 函数 gptimer\_channel\_output\_struct\_para\_init

函数 gptimer\_channel\_output\_struct\_para\_init 描述见下表：

**Table 3-47. Function gptimer\_channel\_output\_struct\_para\_init**

函数名称	gptimer_channel_output_struct_para_init
函数原型	void gptimer_channel_output_struct_para_init(gptimer_oc_parameter_struct *ocpara)
功能描述	初始化 GPTIMER 通道输出初始参数为默认值
先决条件	-
被调用函数	-
输入参数{in}	
ocpara	GPTIMER 通道输出初始参数结构体，参考 <a href="#">表 3-475. 结构体类型 gptimer_oc_parameter_struct</a>
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* init parameter struct with a default value */
```

```
gptimer_oc_parameter_struct para;
```

```
gptimer_channel_output_struct_para_init (&para);
```

### 函数 gptimer\_channel\_output\_config

函数 gptimer\_channel\_output\_config 描述见下表：

**Table 3-48. Function gptimer\_channel\_output\_config**

函数名称	gptimer_channel_output_config
函数原型	void gptimer_channel_output_config(uint32_t timer_periph, uint16_t channel, gptimer_oc_parameter_struct *ocpara)
功能描述	配置通道输出模式
先决条件	-
被调用函数	-

输入参数{in}	
<b>timer_periph</b>	GPTIMER 外设
<i>GPTIMERx</i>	x=0,1
输入参数{in}	
<b>channel</b>	GPTIMER 通道
<i>GPTIMER_CH0</i>	channel 0
<i>GPTIMER_CH1</i>	channel 1
输入参数{in}	
<b>ocpara</b>	GPTIMER 通道输出初始参数结构体，参考 <a href="#">表 3-475. 结构体类型 gptimer_oc_parameter_struct</a>
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* configure GPTIMER0 channel output mode */
```

```
gptimer_oc_parameter_struct para;
```

```
gptimer_channel_output_config (GPTIMER0, GPTIMER_CH0, &para);
```

### 函数 gptimer\_channel\_output\_force\_duty\_config

函数 gptimer\_channel\_output\_force\_duty\_config 描述见下表：

**Table 3-49. Function gptimer\_channel\_output\_force\_duty\_config**

<b>函数名称</b>	gptimer_channel_output_force_duty_config
<b>函数原型</b>	void gptimer_channel_output_force_duty_config(uint32_t timer_periph, uint16_t channel, uint32_t duty, uint32_t output_time)
<b>功能描述</b>	配置通道输出强制占空比
<b>先决条件</b>	-
<b>被调用函数</b>	-
输入参数{in}	
<b>timer_periph</b>	GPTIMER 外设
<i>GPTIMERx</i>	x=0,1
输入参数{in}	
<b>channel</b>	GPTIMER 通道
<i>GPTIMER_CH0</i>	channel 0
<i>GPTIMER_CH1</i>	channel 1
输入参数{in}	
<b>duty</b>	占空比模式
<i>GPTIMER_COMPARE_DUTY_OUTPUT</i>	输出占空比通过比较匹配

<i>GPTIMER_FORCE_0_DUTY_OUTPUT</i>	强制为 0 占空比
<i>GPTIMER_FORCE_10_0_DUTY_OUTPUT</i>	强制为 100% 占空比
输入参数{in}	
<b>output_time</b>	输出时刻
<i>GPTIMER_FORCE_DUTY_OUTPUT_IMMEDIATELY</i>	强制占空比输出立即生效
<i>GPTIMER_FORCE_DUTY_OUTPUT_NEXT_PERIOD</i>	强制占空比输出在下个周期
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* configure GPTIMER0 channel output force duty */
```

```
gptimer_channel_output_force_duty_config      (GPTIMER0,      GPTIMER_CH0,
GPTIMER_COMPARE_DUTY_OUTPUT,
GPTIMER_FORCE_DUTY_OUTPUT_IMMEDIATELY);
```

### 函数 **gptimer\_channel\_output\_compare\_value\_config**

函数 **gptimer\_channel\_output\_compare\_value\_config** 描述见下表：

**Table 3-50. Function **gptimer\_channel\_output\_compare\_value\_config****

函数名称	<b>gptimer_channel_output_compare_value_config</b>
函数原型	<code>void gptimer_channel_output_compare_value_config(uint32_t timer_periph, uint16_t channel, uint16_t value)</code>
功能描述	配置通道输出比较值
先决条件	-
被调用函数	-
输入参数{in}	
<b>timer_periph</b>	GPTIMER 外设
<i>GPTIMERx</i>	x=0,1
输入参数{in}	
<b>channel</b>	GPTIMER 通道
<i>GPTIMER_CH0</i>	channel 0
<i>GPTIMER_CH1</i>	channel 1
输入参数{in}	
<b>value</b>	0~65535



输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* configure GPTIMER0 channel output compare value */
```

```
gptimer_channel_output_compare_value_config (GPTIMER0, GPTIMER_CH0, 0xff);
```

### 函数 **gptimer\_channel\_output\_additional\_compare\_value\_config**

函数 **gptimer\_channel\_output\_additional\_compare\_value\_config** 描述见下表：

**Table 3-51. Function **gptimer\_channel\_output\_additional\_compare\_value\_config****

函数名称	<b>gptimer_channel_output_additional_compare_value_config</b>
函数原型	<code>void gptimer_channel_output_additional_compare_value_config(uint32_t timer_periph, uint16_t channel, uint16_t value)</code>
功能描述	配置通道输出附加比较值
先决条件	-
被调用函数	-
输入参数{in}	
<b>timer_periph</b>	GPTIMER 外设
<i>GPTIMERx</i>	x=0,1
输入参数{in}	
<b>channel</b>	GPTIMER 通道
<i>GPTIMER_CH0</i>	channel 0
<i>GPTIMER_CH1</i>	channel 1
输入参数{in}	
<b>value</b>	0~65535
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* configure GPTIMER0 channel output additional compare value */
```

```
gptimer_channel_output_additional_compare_value_config (GPTIMER0, GPTIMER_CH0, 0xff);
```

### 函数 **gptimer\_channel\_output\_shadow\_config**

函数 **gptimer\_channel\_output\_shadow\_config** 描述见下表：

Table 3-52. Function gptimer\_channel\_output\_shadow\_config

函数名称	gptimer_channel_output_shadow_config
函数原型	void gptimer_channel_output_shadow_config(uint32_t timer_periph, uint16_t channel, uint16_t ocshadow)
功能描述	配置通道输出影子功能
先决条件	-
被调用函数	-
输入参数{in}	
timer_periph	GPTIMER 外设
GPTIMERx	x=0,1
输入参数{in}	
channel	GPTIMER 通道
GPTIMER_CH0	channel 0
GPTIMER_CH1	channel 1
输入参数{in}	
ocshadow	通道输出比较影子功能
GPTIMER_OC_SHADOW_ENABLE	通道输出比较影子功能使能
GPTIMER_OC_SHADOW_DISABLE	通道输出比较影子功能禁能
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* configure GPTIMER0 channel output shadow function */
```

```
gptimer_channel_output_shadow_config(GPTIMER0, GPTIMER_CH0,  
GPTIMER_OC_SHADOW_ENABLE);
```

### 函数 gptimer\_channel\_output\_additional\_shadow\_config

函数 gptimer\_channel\_output\_additional\_shadow\_config 描述见下表:

Table 3-53. Function gptimer\_channel\_output\_additional\_shadow\_config

函数名称	gptimer_channel_output_additional_shadow_config
函数原型	void gptimer_channel_output_additional_shadow_config(uint32_t timer_periph, uint16_t channel, uint16_t ocshadow)
功能描述	配置通道输出附加影子功能
先决条件	-
被调用函数	-
输入参数{in}	

<b>timer_periph</b>	GPTIMER 外设
<i>GPTIMERx</i>	x=0,1
输入参数{in}	
<b>channel</b>	GPTIMER 通道
<i>GPTIMER_CH0</i>	channel 0
<i>GPTIMER_CH1</i>	channel 1
输入参数{in}	
<b>ocshadow</b>	通道输出比较影子功能
<i>GPTIMER_OC_SHADOW_ENABLE</i>	通道输出比较影子功能使能
<i>GPTIMER_OC_SHADOW_DISABLE</i>	通道输出比较影子功能禁能
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* configure GPTIMER0 channel output additional shadow function */
```

```
gptimer_channel_output_additional_shadow_config (GPTIMER0, GPTIMER_CH0,
GPTIMER_OC_SHADOW_ENABLE);
```

### 函数 gptimer\_channel\_output\_control\_shadow\_config

函数 gptimer\_channel\_output\_control\_shadow\_config 描述见下表：

**Table 3-54. Function gptimer\_channel\_output\_control\_shadow\_config**

函数名称	gptimer_channel_output_control_shadow_config
函数原型	void gptimer_channel_output_control_shadow_config(uint32_t timer_periph, uint16_t channel, uint16_t ocshadow)
功能描述	配置通道输出控制影子功能
先决条件	-
被调用函数	-
输入参数{in}	
<b>timer_periph</b>	GPTIMER 外设
<i>GPTIMERx</i>	x=0,1
输入参数{in}	
<b>channel</b>	GPTIMER 通道
<i>GPTIMER_CH0</i>	channel 0
<i>GPTIMER_CH1</i>	channel 1
输入参数{in}	
<b>ocshadow</b>	通道输出比较影子功能

<code>GPTIMER_OC_SHADOW_ENABLE</code>	通道输出比较影子功能使能
<code>GPTIMER_OC_SHADOW_DISABLE</code>	通道输出比较影子功能禁能
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* configure GPTIMER0 channel output additional shadow function */
```

```
gptimer_channel_output_control_shadow_config (GPTIMER0, GPTIMER_CH0,
GPTIMER_OC_SHADOW_ENABLE);
```

### 函数 `gptimer_two_channel_output_high_check_enable`

函数 `gptimer_two_channel_output_high_check_enable` 描述见下表：

**Table 3-55. Function `gptimer_two_channel_output_high_check_enable`**

函数名称	<code>gptimer_two_channel_output_high_check_enable</code>
函数原型	<code>void gptimer_two_channel_output_high_check_enable(uint32_t timer_periph)</code>
功能描述	使能两通道同时输出高电平检测功能
先决条件	-
被调用函数	-
输入参数{in}	
<code>timer_periph</code>	GPTIMER 外设
<code>GPTIMERx</code>	$x=0,1$
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* enable GPTIMER0 two channel simultaneous output high check function */
```

```
gptimer_two_channel_output_high_check_enable (GPTIMER0);
```

### 函数 `gptimer_two_channel_output_high_check_disable`

函数 `gptimer_two_channel_output_high_check_disable` 描述见下表：

**Table 3-56. Function `gptimer_two_channel_output_high_check_disable`**

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

函数原型	void gptimer_two_channel_output_high_check_disable(uint32_t timer_periph)
功能描述	禁能两通道同时输出高电平检测功能
先决条件	-
被调用函数	-
输入参数{in}	
timer_periph	GPTIMER 外设
GPTIMERx	x=0,1
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* disable GPTIMER0 two channel simultaneous output high check function */
```

```
gptimer_two_channel_output_high_check_disable (GPTIMER0);
```

### 函数 gptimer\_two\_channel\_output\_low\_check\_enable

函数 gptimer\_two\_channel\_output\_low\_check\_enable 描述见下表：

**Table 3-57. Function gptimer\_two\_channel\_output\_low\_check\_enable**

函数名称	gptimer_two_channel_output_low_check_enable
函数原型	void gptimer_two_channel_output_low_check_enable(uint32_t timer_periph)
功能描述	使能两通道同时输出低电平检测功能
先决条件	-
被调用函数	-
输入参数{in}	
timer_periph	GPTIMER 外设
GPTIMERx	x=0,1
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* enable GPTIMER0 two channel simultaneous output low check function */
```

```
gptimer_two_channel_output_low_check_enable (GPTIMER0);
```

### 函数 gptimer\_two\_channel\_output\_low\_check\_disable

函数 gptimer\_two\_channel\_output\_low\_check\_disable 描述见下表：

**Table 3-58. Function gptimer\_two\_channel\_output\_low\_check\_disable**

函数名称	gptimer_two_channel_output_low_check_disable
函数原型	void gptimer_two_channel_output_low_check_disable(uint32_t timer_periph)
功能描述	禁能两通道同时输出低电平检测功能
先决条件	-
被调用函数	-
输入参数{in}	
timer_periph	GPTIMER 外设
GPTIMERx	x=0,1
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* disable GPTIMER0 two channel simultaneous output low check function */
```

```
gptimer_two_channel_output_low_check_disable (GPTIMER0);
```

### 函数 gptimer\_period\_signal\_output\_enable

函数 gptimer\_period\_signal\_output\_enable 描述见下表：

**Table 3-59. Function gptimer\_period\_signal\_output\_enable**

函数名称	gptimer_period_signal_output_enable
函数原型	void gptimer_period_signal_output_enable(uint32_t timer_periph)
功能描述	使能计数器周期同步信号输出
先决条件	-
被调用函数	-
输入参数{in}	
timer_periph	GPTIMER 外设
GPTIMERx	x=0,1
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* enable GPTIMER0 counter cycle synchronization signal output */
```

```
gptimer_period_signal_output_enable (GPTIMER0);
```

## 函数 gptimer\_period\_signal\_output\_disable

函数 gptimer\_period\_signal\_output\_disable 描述见下表：

**Table 3-60. Function gptimer\_period\_signal\_output\_disable**

函数名称	gptimer_period_signal_output_disable
函数原型	void gptimer_period_signal_output_disable(uint32_t timer_periph)
功能描述	禁能计数器周期同步信号输出
先决条件	-
被调用函数	-
输入参数{in}	
timer_periph	GPTIMER 外设
GPTIMERx	x=0,1
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* disable GPTIMER0 counter cycle synchronization signal output */
```

```
gptimer_period_signal_output_disable (GPTIMER0);
```

## 函数 gptimer\_stop\_output\_set\_select

函数 gptimer\_stop\_output\_set\_select 描述见下表：

**Table 3-61. Function gptimer\_stop\_output\_set\_select**

函数名称	gptimer_stop_output_set_select
函数原型	void gptimer_stop_output_set_select(uint32_t timer_periph, uint32_t group)
功能描述	配置停止输出源
先决条件	-
被调用函数	-
输入参数{in}	
timer_periph	GPTIMER 外设
GPTIMERx	x=0,1
输入参数{in}	
group	停止输出源选择
GPTIMER_OUTPUT_S TOP_GTOC0	输出停止源 GTOC0
GPTIMER_OUTPUT_S TOP_GTOC1	输出停止源 GTOC1
GPTIMER_OUTPUT_S	输出停止源 GTOC2

TOP_GTOC2	
GPTIMER_OUTPUT_S TOP_GTOC3	输出停止源 GTOC3
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* configure GPTIMER0 stop output source */
```

```
gptimer_stop_output_set_select (GPTIMER0, GPTIMER_OUTPUT_STOP_GTOC0);
```

### 函数 gptimer\_stop\_output\_recover\_time\_select

函数 gptimer\_stop\_output\_recover\_time\_select 描述见下表：

**Table 3-62. Function gptimer\_stop\_output\_recover\_time\_select**

函数名称	gptimer_stop_output_recover_time_select
函数原型	void gptimer_stop_output_recover_time_select(uint32_t timer_periph, uint32_t time)
功能描述	选择输出停止恢复时间
先决条件	-
被调用函数	-
输入参数{in}	
timer_periph	GPTIMER 外设
GPTIMERx	x=0,1
输入参数{in}	
time	output stop recover time 停止输出恢复时刻
GPTIMER_OUTPUT_R ECOVER_PERIOD_EN D	周期结束停止输出恢复
GPTIMER_OUTPUT_R ECOVER_CREP_END	重复计数器结束停止输出恢复
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* GPTIMER0 output stop recover time select */
```

```
gptimer_stop_output_recover_time_select (GPTIMER0,  
GPTIMER_OUTPUT_RECOVER_PERIOD_END);
```



**函数 gptimer\_channel\_complementary\_output\_struct\_para\_init**

函数 gptimer\_channel\_complementary\_output\_struct\_para\_init 描述见下表：

**Table 3-63. Function gptimer\_channel\_complementary\_output\_struct\_para\_init**

函数名称	gptimer_channel_complementary_output_struct_para_init
函数原型	void gptimer_channel_complementary_output_struct_para_init(gptimer_com_oc_parameter_struct *comocpara)
功能描述	初始化 GPTIMER 互补输出初始参数为默认值
先决条件	-
被调用函数	-
输入参数{in}	
comocpara	GPTIMER 互补输出初始参数结构体，参考 <a href="#">表 3-476. 结构体类型 gptimer_com_oc_parameter_struct</a>
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* init parameter struct with a default value */
```

```
gptimer_com_oc_parameter_struct para;
```

```
gptimer_channel_complementary_output_struct_para_init (&para);
```

**函数 gptimer\_channel\_complementary\_output\_config**

函数 gptimer\_channel\_complementary\_output\_config 描述见下表：

**Table 3-64. Function gptimer\_channel\_complementary\_output\_config**

函数名称	gptimer_channel_complementary_output_config
函数原型	void gptimer_channel_complementary_output_config(uint32_t timer_periph, gptimer_com_oc_parameter_struct *comocpara)
功能描述	配置 GPTIMER 通道互补输出功能
先决条件	-
被调用函数	-
输入参数{in}	
timer_periph	GPTIMER 外设
GPTIMERx	x=0,1
输入参数{in}	
comocpara	GPTIMER 互补输出初始参数结构体，参考 <a href="#">表 3-476. 结构体类型 gptimer_com_oc_parameter_struct</a>
输出参数{out}	

-	-
返回值	
-	-

例如:

```
/* configure GPTIMER0 channel output function */
```

```
gptimer_com_oc_parameter_struct para;
```

```
gptimer_channel_complementary_output_config (GPTIMER0, GPTIMER_CH0, &para);
```

### 函数 gptimer\_channel\_io\_direction\_config

函数 gptimer\_channel\_io\_direction\_config 描述见下表:

**Table 3-65. Function gptimer\_channel\_io\_direction\_config**

函数名称	gptimer_channel_io_direction_config	
函数原型	void gptimer_channel_io_direction_config(uint32_t timer_periph, uint16_t channel, uint32_t io_direction)	
功能描述	配置通道为输入或者输出	
先决条件	-	
被调用函数	-	
输入参数{in}		
timer_periph	GPTIMER 外设	
GPTIMERx	x=0,1	
输入参数{in}		
channel	GPTIMER 通道	
GPTIMER_CH0	channel 0	
GPTIMER_CH1	channel 1	
输入参数{in}		
io_direction	输入或输出选择	
GPTIMER_CHANNEL_OUTPUT	通道输出	
GPTIMER_CHANNEL_INPUT	通道输入	
输出参数{out}		
-	-	
返回值		
-	-	

例如:

```
/* configure channel input */
```

```
gptimer_channel_io_direction_config (GPTIMER0, GPTIMER_CH0, GPTIMER_CHANNEL_INPUT);
```

**函数 gptimer\_channel\_io\_state\_config**

函数 gptimer\_channel\_io\_state\_config 描述见下表：

**Table 3-66. Function gptimer\_channel\_io\_state\_config**

函数名称	gptimer_channel_io_state_config
函数原型	void gptimer_channel_io_state_config(uint32_t timer_periph, uint16_t channel, uint32_t state)
功能描述	配置通道为输入或者输出状态使能
先决条件	-
被调用函数	-
输入参数{in}	
timer_periph	GPTIMER 外设
GPTIMERx	x=0,1
输入参数{in}	
channel	GPTIMER 通道
GPTIMER_CH0	channel 0
GPTIMER_CH1	channel 1
输入参数{in}	
state	使能或禁能
GPTIMER_CHANNEL_ENABLE	通道使能
GPTIMER_CHANNEL_DISABLE	通道禁能
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* configure channel input enable state */
```

```
gptimer_channel_io_state_config          (GPTIMER0,          GPTIMER_CH0,
gptimer_channel_io_state_config          GPTIMER_CHANNEL_ENABLE);
```

**函数 gptimer\_capture\_source\_struct\_para\_init**

函数 gptimer\_capture\_source\_struct\_para\_init 描述见下表：

**Table 3-67. Function gptimer\_capture\_source\_struct\_para\_init**

函数名称	gptimer_capture_source_struct_para_init
函数原型	void gptimer_capture_source_struct_para_init(gptimer_capture_source_parameter_struct *counter_capture_source_para)

功能描述	初始化 GPTIMER 捕获源初始参数为默认值
先决条件	-
被调用函数	-
输入参数{in}	
counter_capture_source_para	GPTIMER 捕获源初始参数结构体, 参考 <a href="#">表 3-472. 结构体类型 gptimer_capture_source_parameter_struct</a>
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* init parameter struct with a default value */
gptimer_capture_source_parameter_struct para;
gptimer_capture_source_struct_para_init (&para);
```

### 函数 gptimer\_capture\_source\_config

函数 gptimer\_capture\_source\_config 描述见下表:

**Table 3-68. Function gptimer\_capture\_source\_config**

函数名称	gptimer_capture_source_config
函数原型	void gptimer_capture_source_config(uint32_t timer_periph, uint16_t channel, gptimer_capture_source_parameter_struct *capture_source_para)
功能描述	初始化 GPTIMER 捕获源
先决条件	-
被调用函数	-
输入参数{in}	
timer_periph	GPTIMER 外设
GPTIMERx	x=0,1
输入参数{in}	
channel	GPTIMER 通道
GPTIMER_CH0	channel 0
GPTIMER_CH1	channel 1
输入参数{in}	
counter_capture_source_para	GPTIMER 捕获源初始参数结构体, 参考 <a href="#">表 3-472. 结构体类型 gptimer_capture_source_parameter_struct</a>
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* initialize GPTIMER0 counter capture source */
```

```
gptimer_capture_source_config (GPTIMER0, GPTIMER_CH0, capture_source_eti0);
```

### 函数 gptimer\_channel\_input\_capture\_filter\_config

函数 gptimer\_channel\_input\_capture\_filter\_config 描述见下表：

**Table 3-69. Function gptimer\_channel\_input\_capture\_filter\_config**

函数名称	gptimer_channel_input_capture_filter_config
函数原型	void gptimer_channel_input_capture_filter_config(uint32_t timer_periph, uint32_t channel, uint32_t number)
功能描述	配置通道输入捕获滤波
先决条件	-
被调用函数	-
输入参数{in}	
timer_periph	GPTIMER 外设
GPTIMERx	x=0,1
输入参数{in}	
channel	GPTIMER 通道
GPTIMER_CH0	channel 0
GPTIMER_CH1	channel 1
输入参数{in}	
number	0~15
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* configure GPTIMER0 channel input capture filter */
```

```
gptimer_channel_input_capture_filter_config (GPTIMER0, GPTIMER_CH0, 0);
```

### 函数 gptimer\_channel\_input\_capture\_prescaler\_config

函数 gptimer\_channel\_input\_capture\_prescaler\_config 描述见下表：

**Table 3-70. Function gptimer\_channel\_input\_capture\_prescaler\_config**

函数名称	gptimer_channel_input_capture_prescaler_config
函数原型	void gptimer_channel_input_capture_prescaler_config(uint32_t timer_periph, uint16_t channel, uint32_t prescaler)
功能描述	配置通道输入捕获分频
先决条件	-

被调用函数	-
输入参数{in}	
timer_periph	GPTIMER 外设
GPTIMERx	x=0,1
输入参数{in}	
channel	GPTIMER 通道
GPTIMER_CH0	channel 0
GPTIMER_CH1	channel 1
输入参数{in}	
prescaler	分频值
GPTIMER_IC_PSC_DIV1	不分频
GPTIMER_IC_PSC_DIV2	除以 2
GPTIMER_IC_PSC_DIV4	除以 4
GPTIMER_IC_PSC_DIV8	除以 8
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* configure GPTIMER0 channel input capture prescaler */
```

```
gptimer_channel_input_capture_prescaler_config (GPTIMER0, GPTIMER_CH0,
GPTIMER_IC_PSC_DIV1);
```

### 函数 gptimer\_channel\_capture\_value\_register\_read

函数 gptimer\_channel\_capture\_value\_register\_read 描述见下表：

**Table 3-71. Function gptimer\_channel\_capture\_value\_register\_read**

函数名称	gptimer_channel_capture_value_register_read
函数原型	uint32_t gptimer_channel_capture_value_register_read(uint32_t timer_periph, uint16_t channel)
功能描述	读取通道捕获比较寄存器值
先决条件	-
被调用函数	-
输入参数{in}	
timer_periph	GPTIMER 外设
GPTIMERx	x=0,1

输入参数{in}	
<b>channel</b>	GPTIMER 通道
<i>GPTIMER_CH0</i>	channel 0
<i>GPTIMER_CH1</i>	channel 1
输出参数{out}	
-	-
返回值	
<b>uint32_t</b>	捕获值

例如:

```
/* read channel capture compare register value */
```

```
uint32_t value;
```

```
value = gptimer_channel_capture_value_register_read (GPTIMER0, GPTIMER_CH0);
```

### 函数 gptimer\_counter\_sync\_control\_select

函数 gptimer\_counter\_sync\_control\_select 描述见下表:

**Table 3-72. Function gptimer\_counter\_sync\_control\_select**

<b>函数名称</b>	gptimer_counter_sync_control_select
<b>函数原型</b>	void gptimer_counter_sync_control_select(uint32_t timer_periph, uint32_t set)
<b>功能描述</b>	选择计数器同步复位组
<b>先决条件</b>	-
<b>被调用函数</b>	-
输入参数{in}	
<b>timer_periph</b>	GPTIMER 外设
<i>GPTIMERx</i>	x=0,1
输入参数{in}	
<b>group</b>	同步复位组
<i>GPTIMER_CNT_RESE T_GTOC0</i>	计数器复位组 GTOC0
<i>GPTIMER_CNT_RESE T_GTOC1</i>	计数器复位组 GTOC1
<i>GPTIMER_CNT_RESE T_GTOC2</i>	计数器复位组 GTOC2
<i>GPTIMER_CNT_RESE T_GTOC3</i>	计数器复位组 GTOC3
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* select GPTIMER0 counter sync reset set */
```

```
gptimer_counter_sync_control_select (GPTIMER0, GPTIMER_CNT_RESET_GTOC0);
```

### 函数 **gptimer\_counter\_sync\_reset\_source\_select**

函数 **gptimer\_counter\_sync\_reset\_source\_select** 描述见下表:

**Table 3-73. Function **gptimer\_counter\_sync\_reset\_source\_select****

函数名称	<b>gptimer_counter_sync_reset_source_select</b>
函数原型	<code>void gptimer_counter_sync_reset_source_select(uint32_t timer_periph, uint32_t source)</code>
功能描述	选择计数器同步复位源
先决条件	-
被调用函数	-
输入参数{in}	
<b>timer_periph</b>	GPTIMER 外设
<i>GPTIMERx</i>	x=0,1
输入参数{in}	
<b>group</b>	同步复位源
<i>GPTIMER_SYNC_RES ET_CH0CV</i>	ch0cv 作为同步复位源
<i>GPTIMER_SYNC_RES ET_CH1CV</i>	ch1cv 作为同步复位源
<i>GPTIMER_SYNC_RES ET_CH0COMV_ADD</i>	ch0comval_add 作为同步复位源
<i>GPTIMER_SYNC_RES ET_CH1COMV_ADD</i>	ch1comval_add 作为同步复位源
<i>GPTIMER_SYNC_RES ET_UF</i>	overflow 作为同步复位源
<i>GPTIMER_SYNC_RES ET_OF</i>	underflow 作为同步复位源
<i>GPTIMER_SYNC_RES ET_CH</i>	通道输入作为同步复位源
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* select GPTIMER0 counter sync reset source */
```

```
gptimer_counter_sync_reset_source_select
```

```
(GPTIMER0,
```



```
GPTIMER_SYNC_RESET_CH0CV);
```

### 函数 `gptimer_trigger_adc_compare_enable`

函数 `gptimer_trigger_adc_compare_enable` 描述见下表：

**Table 3-74. Function `gptimer_trigger_adc_compare_enable`**

函数名称	<code>gptimer_trigger_adc_compare_enable</code>
函数原型	<code>void gptimer_trigger_adc_compare_enable(uint32_t timer_periph, uint32_t compare_time)</code>
功能描述	使能比较事件产生 ADC 转换触发信号
先决条件	-
被调用函数	-
输入参数{in}	
<b>timer_periph</b>	GPTIMER 外设
<i>GPTIMERx</i>	x=0,1
输入参数{in}	
<b>compare_time</b>	比较时刻
<i>GPTIMER_ADT1UEN</i>	使能向上计数时 ADTCV1[15:0]匹配事件作为 ADC 触发信号
<i>GPTIMER_ADT1DEN</i>	使能向下计数时 ADTCV1[15:0]匹配事件作为 ADC 触发信号
<i>GPTIMER_ADT2UEN</i>	使能向上计数时 ADTCV2[15:0]匹配事件作为 ADC 触发信号
<i>GPTIMER_ADT2DEN</i>	使能向下计数时 ADTCV2[15:0]匹配事件作为 ADC 触发信号
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* enable GPTIMER0 compare event produce a A/D converter trigger signal */
```

```
gptimer_trigger_adc_compare_enable (GPTIMER0, GPTIMER_ADT1UEN);
```

### 函数 `gptimer_trigger_adc_compare_disable`

函数 `gptimer_trigger_adc_compare_disable` 描述见下表：

**Table 3-75. Function `gptimer_trigger_adc_compare_disable`**

函数名称	<code>gptimer_trigger_adc_compare_disable</code>
函数原型	<code>void gptimer_trigger_adc_compare_disable(uint32_t timer_periph, uint32_t compare_time)</code>
功能描述	禁能比较事件产生 ADC 转换触发信号
先决条件	-
被调用函数	-
输入参数{in}	
<b>timer_periph</b>	GPTIMER 外设

<i>GPTIMERx</i>	<i>x</i> =0,1
输入参数{in}	
<b>compare_time</b>	compare time
<i>GPTIMER_ADT1UEN</i>	使能向上计数时 ADTCV1[15:0]匹配事件作为 ADC 触发信号
<i>GPTIMER_ADT1DEN</i>	使能向下计数时 ADTCV1[15:0]匹配事件作为 ADC 触发信号
<i>GPTIMER_ADT2UEN</i>	使能向上计数时 ADTCV2[15:0]匹配事件作为 ADC 触发信号
<i>GPTIMER_ADT2DEN</i>	使能向下计数时 ADTCV2[15:0]匹配事件作为 ADC 触发信号
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* disable GPTIMER0 compare event produce a A/D converter trigger signal */
```

```
gptimer_trigger_adc_compare_disable (GPTIMER0, GPTIMER_ADT1UEN);
```

### 函数 gptimer\_trigger\_adc\_compare\_value\_config

函数 gptimer\_trigger\_adc\_compare\_value\_config 描述见下表：

**Table 3-76. Function gptimer\_trigger\_adc\_compare\_value\_config**

函数名称	gptimer_trigger_adc_compare_value_config
函数原型	void gptimer_trigger_adc_compare_value_config(uint32_t timer_periph, uint32_t compare, uint16_t value)
功能描述	配置触发 ADC 比较寄存器值
先决条件	-
被调用函数	-
输入参数{in}	
<b>timer_periph</b>	GPTIMER 外设
<i>GPTIMERx</i>	<i>x</i> =0,1
输入参数{in}	
<b>compare</b>	比较值选择
<i>GPTIMER_ADC_COMPARE1</i>	选择 GPTIMERx_ADCCR1 ADTCV1
<i>GPTIMER_ADC_COMPARE2</i>	选择 GPTIMERx_ADCCR2 ADTCV2
输入参数{in}	
<b>value</b>	0~65535
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* config GPTIMER0 trigger adc compare regiseter value */
```

```
gptimer_trigger_adc_compare_value_config (GPTIMER0, GPTIMER_ADC_COMPARE1, 0xff);
```

### 函数 gptimer\_trigger\_adc\_compare\_shadow\_enable

函数 gptimer\_trigger\_adc\_compare\_shadow\_enable 描述见下表:

**Table 3-77. Function gptimer\_trigger\_adc\_compare\_shadow\_enable**

函数名称	gptimer_trigger_adc_compare_shadow_enable
函数原型	void gptimer_trigger_adc_compare_shadow_enable(uint32_t timer_periph, uint32_t compare)
功能描述	使能 ADC 比较寄存器影子功能
先决条件	-
被调用函数	-
输入参数{in}	
timer_periph	GPTIMER 外设
GPTIMERx	x=0,1
输入参数{in}	
compare	比较值选择
GPTIMER_ADC_COMPARE1	选择 GPTIMERx_ADCCR1 ADTCV1
GPTIMER_ADC_COMPARE2	选择 GPTIMERx_ADCCR2 ADTCV2
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* enable GPTIMER0 adc compare regiseter shadow function */
```

```
gptimer_trigger_adc_compare_shadow_enable (GPTIMER0, GPTIMER_ADC_COMPARE1);
```

### 函数 gptimer\_trigger\_adc\_compare\_shadow\_disable

函数 gptimer\_trigger\_adc\_compare\_shadow\_disable 描述见下表:

**Table 3-78. Function gptimer\_trigger\_adc\_compare\_shadow\_disable**

函数名称	gptimer_trigger_adc_compare_shadow_disable
函数原型	void gptimer_trigger_adc_compare_shadow_disable(uint32_t timer_periph, uint32_t compare)
功能描述	禁能 ADC 比较寄存器影子功能

先决条件	-
被调用函数	-
输入参数{in}	
timer_periph	GPTIMER 外设
GPTIMERx	x=0,1
输入参数{in}	
compare	比较值选择
GPTIMER_ADC_COMPARE1	选择 GPTIMERx_ADCCR1 ADTCV1
GPTIMER_ADC_COMPARE2	选择 GPTIMERx_ADCCR2 ADTCV2
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* disable GPTIMER0 adc compare register shadow function */
```

```
gpgptimer_trigger_adc_compare_shadow_disable(GPTIMER0,
GPTIMER_ADC_COMPARE1);
```

### 函数 gptimer\_trigger\_adc\_adsm\_enable

函数 gptimer\_trigger\_adc\_adsm\_enable 描述见下表:

**Table 3-79. Function gptimer\_trigger\_adc\_adsm\_enable**

函数名称	gptimer_trigger_adc_adsm_enable
函数原型	void gptimer_trigger_adc_adsm_enable(uint32_t timer_periph, uint32_t adsm)
功能描述	使能触发 ADC adsm 信号
先决条件	-
被调用函数	-
输入参数{in}	
timer_periph	GPTIMER 外设
GPTIMERx	x=0,1
输入参数{in}	
adsm	adsm 信号选择
GPTIMER_ADCSM3	ADC 监视信号 3
GPTIMER_ADCSM4	ADC 监视信号 4
输出参数{out}	
-	-
返回值	

-	-
---	---

例如：

```
/* enable GPTIMER0 trigger adc adsm signal */
```

```
gptimer_trigger_adc_adsm_enable (GPTIMER0, GPTIMER_ADCSM3);
```

### 函数 **gptimer\_trigger\_adc\_adsm\_disable**

函数 **gptimer\_trigger\_adc\_adsm\_disable** 描述见下表：

**Table 3-80. Function **gptimer\_trigger\_adc\_adsm\_disable****

函数名称	<b>gptimer_trigger_adc_adsm_disable</b>
函数原型	void <b>gptimer_trigger_adc_adsm_disable</b> (uint32_t timer_periph, uint32_t adsm)
功能描述	禁能触发 ADC adsm 信号
先决条件	-
被调用函数	-
输入参数{in}	
<b>timer_periph</b>	GPTIMER 外设
<i>GPTIMERx</i>	x=0,1
输入参数{in}	
<b>adsm</b>	adsm 信号选择
<i>GPTIMER_ADCSM3</i>	ADC 监视信号 3
<i>GPTIMER_ADCSM4</i>	ADC 监视信号 4
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* disable GPTIMER0 trigger adc adsm signal */
```

```
gptimer_trigger_adc_adsm_disable (GPTIMER0, GPTIMER_ADCSM3);
```

### 函数 **gptimer\_trigger\_adc\_adsm\_select**

函数 **gptimer\_trigger\_adc\_adsm\_select** 描述见下表：

**Table 3-81. Function **gptimer\_trigger\_adc\_adsm\_select****

函数名称	<b>gptimer_trigger_adc_adsm_select</b>
函数原型	void <b>gptimer_trigger_adc_adsm_select</b> (uint32_t timer_periph, uint32_t adsm, uint32_t trg_source)
功能描述	选择触发 ADC adsm 信号
先决条件	-

被调用函数	-
输入参数{in}	
timer_periph	GPTIMER 外设
GPTIMERx	x=0,1
输入参数{in}	
adsm	adsm 信号选择
GPTIMER_ADCSM3	ADC 监视信号 3
GPTIMER_ADCSM4	ADC 监视信号 4
输入参数{in}	
trg_source	触发源选择
GPTIMER_ADSM_ADT CV1	ADC 监视信号 ADTCV1
GPTIMER_ADSM_ADT CV2	ADC 监视信号 ADTCV2
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* select GPTIMER0 trigger adc adsm signal */
```

```
gptimer_trigger_adc_adsm_select          (GPTIMER0,          GPTIMER_ADCSM3,
GPTIMER_ADSM_ADTCV1);
```

### 函数 gptimer\_trigger\_adc\_skipping\_config

函数 gptimer\_trigger\_adc\_skipping\_config 描述见下表：

**Table 3-82. Function gptimer\_trigger\_adc\_skipping\_config**

函数名称	gptimer_trigger_adc_skipping_config
函数原型	void gptimer_trigger_adc_skipping_config(uint32_t timer_periph, uint32_t skipping_count, uint32_t value, uint32_t initial_value, uint32_t skipping_source)
功能描述	配置触发 ADC 跳过值
先决条件	-
被调用函数	-
输入参数{in}	
timer_periph	GPTIMER 外设
GPTIMERx	x=0,1
输入参数{in}	
skipping_count	跳过计数器选择
GPTIMER_ADC_SKIP_	ADC 跳过计数器 1

COUNT1	
GPTIMER_ADC_SKIP_COUNT2	ADC 跳过计数器 2
输入参数{in}	
value	跳过值, 0~15
输入参数{in}	
initial_value	初始值, 0~15
输入参数{in}	
skipping_source	跳过源选择
GPTIMER_ADC_SKIP_NO	不计数, 不跳过
GPTIMER_ADC_SKIP_ADTCV1	ADTCV1[15:0]比较匹配
GPTIMER_ADC_SKIP_ADTCV2	the ADTCV2[15:0]比较匹配
GPTIMER_ADC_SKIP_ADTCV1_OR_ADTCV2	ADTCV1[15:0]或者 ADTCV2[15:0]比较匹配
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* config GPTIMER0 trigger adc skipping value */
```

```
gptimer_trigger_adc_skipping_config (GPTIMER0, GPTIMER_ADC_SKIP_COUNT1, 2, 2, GPTIMER_ADC_SKIP_ADTCV1);
```

### 函数 gptimer\_trigger\_adc\_skipping\_time\_select

函数 gptimer\_trigger\_adc\_skipping\_time\_select 描述见下表:

**Table 3-83. Function gptimer\_trigger\_adc\_skipping\_time\_select**

函数名称	gptimer_trigger_adc_skipping_time_select
函数原型	void gptimer_trigger_adc_skipping_time_select(uint32_t timer_periph, uint32_t compare, uint32_t skipping_time)
功能描述	选择触发 ADC 跳过功能
先决条件	-
被调用函数	-
输入参数{in}	
timer_periph	GPTIMER 外设
GPTIMERx	x=0,1
输入参数{in}	

<b>compare</b>	比较值选择
<i>GPTIMER_ADC_COMPARE1</i>	选择 GPTIMERx_ADCCR1 ADTCV1
<i>GPTIMER_ADC_COMPARE2</i>	选择 GPTIMERx_ADCCR2 ADTCV2
<b>输入参数{in}</b>	
<b>skipping_time</b>	跳过时刻选择
<i>GPTIMER_ADC_REQ_NO</i>	不跳过
<i>GPTIMER_ADC_REQ_CNT1_ZERO</i>	当跳过计数器 1 等于 0 时产生 ADC 响应源
<i>GPTIMER_ADC_REQ_CNT2_ZERO</i>	当跳过计数器 2 等于 0 时产生 ADC 响应源
<i>GPTIMER_ADC_REQ_CNT1_OR_CNT2_ZERO</i>	当跳过计数器 1/2 等于 0 时产生 ADC 响应源
<i>GPTIMER_ADC_REQ_CNT1_SVAL</i>	当跳过计数器 1 等于跳过值时产生 ADC 响应源
<i>GPTIMER_ADC_REQ_CNT2_SVAL</i>	当跳过计数器 2 等于跳过值时产生 ADC 响应源
<i>GPTIMER_ADC_REQ_CNT1_OR_CNT2_SVAL</i>	当跳过计数器 1/2 等于跳过值时产生 ADC 响应源
<b>输出参数{out}</b>	
-	-
<b>返回值</b>	
-	-

例如：

```
/* select GPTIMER0 trigger adc skipping function */
```

```
gptimer_trigger_adc_skipping_time_select (GPTIMER0, GPTIMER_ADC_COMPARE1,
GPTIMER_ADC_REQ_CNT1_ZERO);
```

### 函数 **gptimer\_trigger\_adc\_skipping\_counter\_read**

函数 **gptimer\_trigger\_adc\_skipping\_counter\_read** 描述见下表：

**Table 3-84. Function **gptimer\_trigger\_adc\_skipping\_counter\_read****

<b>函数名称</b>	<b>gptimer_trigger_adc_skipping_counter_read</b>
<b>函数原型</b>	<code>uint16_t gptimer_trigger_adc_skipping_counter_read(uint32_t timer_periph, uint32_t skipping_count)</code>
<b>功能描述</b>	读取触发 ADC 跳过计数器
<b>先决条件</b>	-



被调用函数	-
输入参数{in}	
timer_periph	GPTIMER 外设
GPTIMERx	x=0,1
输入参数{in}	
skipping_count	跳过计数器选择
GPTIMER_ADDINT_SK IP_COUNT1	附加中断跳过计数器 1
GPTIMER_ADDINT_SK IP_COUNT2	附加中断跳过计数器 2
输出参数{out}	
-	-
返回值	
skipping counter	

例如:

```
/* read GPTIMER0 trigger adc skipping counter */
```

```
gptimer_trigger_adc_skipping_counter_read                      (GPTIMER0,  
GPTIMER_ADDINT_SKIP_COUNT1);
```

### 函数 gptimer\_additional\_interrupt\_skipping\_config

函数 gptimer\_additional\_interrupt\_skipping\_config 描述见下表:

**Table 3-85. Function gptimer\_additional\_interrupt\_skipping\_config**

函数名称	gptimer_additional_interrupt_skipping_config
函数原型	void gptimer_additional_interrupt_skipping_config(uint32_t timer_periph, uint32_t skipping_count, uint32_t value, uint32_t initial_value, uint32_t skipping_source)
功能描述	配置附加中断跳过初始值
先决条件	-
被调用函数	-
输入参数{in}	
timer_periph	GPTIMER 外设
GPTIMERx	x=0,1
输入参数{in}	
skipping_count	跳过计数器选择
GPTIMER_ADDINT_SK IP_COUNT1	附加中断跳过计数器 1
GPTIMER_ADDINT_SK IP_COUNT2	附加中断跳过计数器 2
输入参数{in}	

<b>value</b>	跳过值, 0~15
<b>输入参数{in}</b>	
<b>initial_value</b>	初始值, 0~15
<b>输入参数{in}</b>	
<b>skipping_source</b>	跳过源选择
<i>GPTIMER_ADDINT_SK IP_NO</i>	无附加中断跳过计数器跳过源
<i>GPTIMER_ADDINT_SK IP_OVERFLOW</i>	上溢作为附加中断跳过计数器跳过源
<i>GPTIMER_ADDINT_SK IP_UNDERFLOW</i>	下溢作为附加中断跳过计数器跳过源
<i>GPTIMER_ADDINT_SK IP_FLOW</i>	上溢/下溢作为附加中断跳过计数器跳过源
<b>输出参数{out}</b>	
-	-
<b>返回值</b>	
-	-

例如:

```
/* config GPTIMER0 additional interrupt skipping initial value */
```

```
gptimer_additional_interrupt_skipping_config(GPTIMER0,
GPTIMER_ADDINT_SKIP_COUNT1, 2, 2, GPTIMER_ADDINT_SKIP_OVERFLOW);
```

### 函数 **gptimer\_additional\_interrupt\_skipping\_time\_select**

函数 **gptimer\_additional\_interrupt\_skipping\_time\_select** 描述见下表:

**Table 3-86. Function **gptimer\_additional\_interrupt\_skipping\_time\_select****

<b>函数名称</b>	<b>gptimer_additional_interrupt_skipping_time_select</b>
<b>函数原型</b>	<code>void gptimer_additional_interrupt_skipping_time_select(uint32_t timer_periph, uint32_t event, uint32_t skipping_time)</code>
<b>功能描述</b>	选择附加中断跳过功能
<b>先决条件</b>	-
<b>被调用函数</b>	-
<b>输入参数{in}</b>	
<b>timer_periph</b>	GPTIMER 外设
<i>GPTIMERx</i>	x=0,1
<b>输入参数{in}</b>	
<b>event</b>	跳过事件
<i>GPTIMER_ADDINT_SK IP_EVENT_CH0CV</i>	附加中断跳过事件 ch0cv
<i>GPTIMER_ADDINT_SK</i>	附加中断跳过事件 ch1cv

IP_EVENT_CH1CV	
GPTIMER_ADDINT_SK IP_EVENT_CH0COMV _ADD	附加中断跳过事件 ch0comv_add
GPTIMER_ADDINT_SK IP_EVENT_CH1COMV _ADD	附加中断跳过事件 ch1comv_add
GPTIMER_ADDINT_SK IP_EVENT_UNDERFL OW	附加中断跳过事件 underflow
GPTIMER_ADDINT_SK IP_EVENT_OVERFLOW W	附加中断跳过事件 overflow
GPTIMER_ADDINT_SK IP_EVENT_ADTCV1	附加中断跳过事件 adtcv1
GPTIMER_ADDINT_SK IP_EVENT_ADTCV2	附加中断跳过事件 adtcv2
输入参数{in}	
skipping_time	跳过时刻
GPTIMER_ADDINT_RE Q_NO	不跳过
GPTIMER_ADDINT_RE Q_CNT1_ZERO	当跳过计数器 1 等于 0 时产生附加中断响应源
GPTIMER_ADDINT_RE Q_CNT2_ZERO	当跳过计数器 2 等于 0 时产生附加中断响应源
GPTIMER_ADDINT_RE Q_CNT1_OR_CNT2_Z ERO	当跳过计数器 1/2 等于 0 时产生附加中断响应源
GPTIMER_ADDINT_RE Q_CNT1_SVAL	当跳过计数器 1 等于跳过值时产生附加中断响应源
GPTIMER_ADDINT_RE Q_CNT2_SVAL	当跳过计数器 2 等于跳过值时产生附加中断响应源
GPTIMER_ADDINT_RE Q_CNT1_OR_CNT2_S VAL	当跳过计数器 1/2 等于跳过值时产生附加中断响应源
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* select GPTIMER0 additional interrupt skipping function */
```

gptimer\_additional\_interrupt\_skipping\_time\_select (GPTIMER0,  
GPTIMER\_ADDINT\_SKIP\_EVENT\_CH0CV, GPTIMER\_ADDINT\_REQ\_CNT1\_ZERO);

### 函数 gptimer\_additional\_interrupt\_skipping\_counter\_read

函数 gptimer\_additional\_interrupt\_skipping\_counter\_read 描述见下表:

**Table 3-87. Function gptimer\_additional\_interrupt\_skipping\_counter\_read**

函数名称	gptimer_additional_interrupt_skipping_counter_read
函数原型	uint16_t gptimer_additional_interrupt_skipping_counter_read(uint32_t timer_periph, uint32_t skipping_count)
功能描述	读取附加中断跳过计数器
先决条件	-
被调用函数	-
输入参数{in}	
timer_periph	GPTIMER 外设
GPTIMERx	x=0,1
输入参数{in}	
skipping_count	跳过计数器选择
GPTIMER_ADDINT_SK IP_COUNT1	附加中断跳过计数器 1
GPTIMER_ADDINT_SK IP_COUNT2	附加中断跳过计数器 2
输出参数{out}	
-	-
返回值	
skipping counter	

例如:

```
/* read GPTIMER0 additional interrupt skipping counter */
```

```
gptimer_additional_interrupt_skipping_counter_read (GPTIMER0,  
GPTIMER_ADDINT_SKIP_COUNT1);
```

### 函数 gptimer\_flow\_interrupt\_skipping\_source\_select

函数 gptimer\_flow\_interrupt\_skipping\_source\_select 描述见下表:

**Table 3-88. Function gptimer\_flow\_interrupt\_skipping\_source\_select**

函数名称	gptimer_flow_interrupt_skipping_source_select
函数原型	void gptimer_flow_interrupt_skipping_source_select(uint32_t timer_periph, uint32_t skipping_source)
功能描述	选择溢出中断跳过源
先决条件	-
被调用函数	-

输入参数{in}	
<b>timer_periph</b>	GPTIMER 外设
<i>GPTIMERx</i>	x=0,1
输入参数{in}	
<b>skipping_source</b>	跳过源选择
<i>GPTIMER_FLOW_SKIP_SOURCE_NO</i>	无溢出跳过源
<i>GPTIMER_FLOW_SKIP_SOURCE_OVERFLOW</i>	上溢作为溢出跳过源
<i>GPTIMER_FLOW_SKIP_SOURCE_UNDERFLOW</i>	下溢作为溢出跳过源
<i>GPTIMER_FLOW_SKIP_SOURCE_FLOW</i>	上溢/下溢作为溢出跳过源
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* select GPTIMER0 flow interrupt skipping source */
```

```
gptimer_flow_interrupt_skipping_source_select(GPTIMER0, GPTIMER_FLOW_SKIP_SOURCE_OVERFLOW);
```

### 函数 gptimer\_flow\_interrupt\_skipping\_link\_enable

函数 gptimer\_flow\_interrupt\_skipping\_link\_enable 描述见下表:

**Table 3-89. Function gptimer\_flow\_interrupt\_skipping\_link\_enable**

<b>函数名称</b>	gptimer_flow_interrupt_skipping_link_enable
<b>函数原型</b>	void gptimer_flow_interrupt_skipping_link_enable(uint32_t timer_periph, uint32_t skipping_link_source)
<b>功能描述</b>	使能溢出中断跳过链接功能
<b>先决条件</b>	-
<b>被调用函数</b>	-
输入参数{in}	
<b>timer_periph</b>	GPTIMER 外设
<i>GPTIMERx</i>	x=0,1
输入参数{in}	
<b>skipping_link_source</b>	跳过链接源
<i>GPTIMER_SKIP_LINK_CH0CV</i>	中断跳过链接 channel 0 输入捕获/比较匹配

<code>GPTIMER_SKIP_LINK_CH1CV</code>	中断跳过链接 channel 1 输入捕获/比较匹配
<code>GPTIMER_SKIP_LINK_CH0COMV_ADD</code>	中断跳过链接 channel 0 附加值比较匹配
<code>GPTIMER_SKIP_LINK_CH1COMV_ADD</code>	中断跳过链接 channel 1 附加值比较匹配
<code>GPTIMER_SKIP_LINK_ADCCR1</code>	中断跳过链接通过 ADTCV1[15:0]产生触发信号
<code>GPTIMER_SKIP_LINK_ADCCR2</code>	中断跳过链接通过 ADTCV2[15:0]产生触发信号
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* enable GPTIMER0 flow interrupt skipping link function */
```

```
gptimer_flow_interrupt_skipping_link_enable (GPTIMER0, GPTIMER_SKIP_LINK_CH0CV);
```

### 函数 `gptimer_flow_interrupt_skipping_link_disable`

函数 `gptimer_flow_interrupt_skipping_link_disable` 描述见下表：

**Table 3-90. Function `gptimer_flow_interrupt_skipping_link_disable`**

函数名称	<code>gptimer_flow_interrupt_skipping_link_disable</code>
函数原型	<code>void gptimer_flow_interrupt_skipping_link_disable(uint32_t timer_periph, uint32_t skipping_link_source)</code>
功能描述	禁能溢出中断跳过链接功能
先决条件	-
被调用函数	-
输入参数{in}	
<code>timer_periph</code>	GPTIMER 外设
<code>GPTIMERx</code>	x=0,1
输入参数{in}	
<code>skipping_link_source</code>	跳过链接源
<code>GPTIMER_SKIP_LINK_CH0CV</code>	中断跳过链接 channel 0 输入捕获/比较匹配
<code>GPTIMER_SKIP_LINK_CH1CV</code>	中断跳过链接 channel 1 输入捕获/比较匹配
<code>GPTIMER_SKIP_LINK_CH0COMV_ADD</code>	中断跳过链接 channel 0 附加值比较匹配
<code>GPTIMER_SKIP_LINK_CH1COMV_ADD</code>	中断跳过链接 channel 1 附加值比较匹配

<code>GPTIMER_SKIP_LINK_ADCCR1</code>	中断跳过链接通过 ADTCV1[15:0]产生触发信号
<code>GPTIMER_SKIP_LINK_ADCCR2</code>	中断跳过链接通过 ADTCV2[15:0]产生触发信号
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* disable GPTIMER0 flow interrupt skipping link function */
```

```
gptimer_flow_interrupt_skipping_link_disable (GPTIMER0, GPTIMER_SKIP_LINK_CH0CV);
```

### 函数 `gptimer_flow_interrupt_skipping_num_config`

函数 `gptimer_flow_interrupt_skipping_num_config` 描述见下表：

**Table 3-91. Function `gptimer_flow_interrupt_skipping_num_config`**

函数名称	<code>gptimer_flow_interrupt_skipping_num_config</code>
函数原型	<code>void gptimer_flow_interrupt_skipping_num_config(uint32_t timer_periph, uint32_t number)</code>
功能描述	配置溢出中断重复跳过值
先决条件	-
被调用函数	-
输入参数{in}	
<code>timer_periph</code>	GPTIMER 外设
<code>GPTIMERx</code>	x=0,1
输入参数{in}	
<code>number</code>	中断跳过值，0~7
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* configure GPTIMER0 flow interrupt repetition skipping number */
```

```
gptimer_flow_interrupt_skipping_num_config (GPTIMER0, 7);
```

### 函数 `gptimer_flow_interrupt_skipping_counter_read`

函数 `gptimer_flow_interrupt_skipping_counter_read` 描述见下表：

Table 3-92. Function gptimer\_flow\_interrupt\_skipping\_counter\_read

函数名称	gptimer_flow_interrupt_skipping_counter_read
函数原型	uint16_t gptimer_flow_interrupt_skipping_counter_read(uint32_t timer_periph)
功能描述	读取溢出中断跳过计数器值
先决条件	-
被调用函数	-
输入参数{in}	
timer_periph	GPTIMER 外设
GPTIMERx	x=0,1
输出参数{out}	
-	-
返回值	
uint16_t	跳过计数值

例如：

```
/* read GPTIMER0 flow interrupt skipping counter value */
```

```
uint16_t value;
```

```
value = gptimer_flow_interrupt_skipping_counter_read (GPTIMER0);
```

### 函数 gptimer\_write\_chxval\_register\_config

函数 gptimer\_write\_chxval\_register\_config 描述见下表：

Table 3-93. Function gptimer\_write\_chxval\_register\_config

函数名称	gptimer_write_chxval_register_config
函数原型	void gptimer_write_chxval_register_config(uint32_t timer_periph, uint16_t ccsel)
功能描述	配置写 CHxVAL 寄存器选择
先决条件	-
被调用函数	-
输入参数{in}	
timer_periph	GPTIMER 外设
GPTIMERx	x=0,1
输入参数{in}	
ccsel	写 CHxVAL 寄存器保护使能或禁能
GPTIMER_CHVSEL_DISABLE	无影响
GPTIMER_CHVSEL_ENABLE	当写 CHxVAL 寄存器，如果写入的值和 CHxVAL 之前值一样，则写入值无效
输出参数{out}	
-	-



返回值	
-	-

例如：

```
/* configure GPTIMER0 write CHxVAL register selection */
```

```
gptimer_write_chxval_register_config (GPTIMER0, GPTIMER_CHVSEL_ENABLE);
```

## 函数 gptimer\_flag\_get

函数 gptimer\_flag\_get 描述见下表：

**Table 3-94. 函数 gptimer\_flag\_get**

函数名称	gptimer_flag_get
函数原型	FlagStatus gptimer_flag_get(uint32_t timer_periph, uint32_t flag)
功能描述	获取 GPTIMER 标志位
先决条件	-
被调用函数	-
输入参数{in}	
timer_periph	GPTIMER 外设
GPTIMERx	x=0,1
输入参数{in}	
flag	GPTIMER 标志位
GPTIMER_FLAG_CH0	GPTIMER channel 0 捕获或比较标志位
GPTIMER_FLAG_CH1	GPTIMER channel 1 捕获或比较标志位
GPTIMER_FLAG_OVRFLOW	GPTIMER 计数器上溢标志位
GPTIMER_FLAG_UNDRFLOW	GPTIMER 计数器下溢标志位
GPTIMER_FLAG_CH0COMADD	GPTIMER 附加 channel 0 比较标志位
GPTIMER_FLAG_CH1COMADD	GPTIMER 附加 channel 1 比较标志位
GPTIMER_FLAG_UP	GPTIMER 计数器复位标志位
GPTIMER_FLAG_ADT1CMU	GPTIMER ADC 触发 1 向上计数比较匹配标志位
GPTIMER_FLAG_ADT1CMD	GPTIMER ADC 触发 1 向下计数比较匹配标志位
GPTIMER_FLAG_ADT2CMU	GPTIMER ADC 触发 2 向上计数比较匹配标志位
GPTIMER_FLAG_ADT2CMD	GPTIMER ADC 触发 2 向下计数比较匹配标志位
GPTIMER_FLAG_CHHOUT	GPTIMER 输出同时高错误标志位

<i>GPTIMER_FLAG_CHL</i> <i>OUT</i>	GPTIMER 输出同时低错误标志位
<i>GPTIMER_FLAG_RCE</i> <i>ND</i>	GPTIMER 重复计数器结束标志位
<i>GPTIMER_FLAG_DIR</i>	GPTIMER 计数器方向标志位
<i>GPTIMER_FLAG_OST</i>	GPTIMER 输出停止标志位
输出参数{out}	
-	-
返回值	
<b>FlagStatus: SET or RESET</b>	

例如:

```
/* get GPTIMER0 flags */
```

```
gptimer_flag_get (GPTIMER0, GPTIMER_FLAG_CH0);
```

### 函数 gptimer\_flag\_clear

函数 gptimer\_flag\_clear 描述见下表:

**Table 3-95. Function gptimer\_flag\_clear**

函数名称	gptimer_flag_clear
函数原型	void gptimer_flag_clear(uint32_t timer_periph, uint32_t flag)
功能描述	复位 GPTIMER 标志位
先决条件	-
被调用函数	-
输入参数{in}	
<b>timer_periph</b>	GPTIMER 外设
<i>GPTIMERx</i>	x=0,1
输入参数{in}	
<b>flag</b>	GPTIMER 标志位
<i>GPTIMER_FLAG_CH0</i>	GPTIMER channel 0 捕获或比较标志位
<i>GPTIMER_FLAG_CH1</i>	GPTIMER channel 1 捕获或比较标志位
<i>GPTIMER_FLAG_OVE</i> <i>RFLOW</i>	GPTIMER 计数器上溢标志位
<i>GPTIMER_FLAG_UND</i> <i>ERFLOW</i>	GPTIMER 计数器下溢标志位
<i>GPTIMER_FLAG_CH0</i> <i>COMADD</i>	GPTIMER 附加 channel 0 比较标志位
<i>GPTIMER_FLAG_CH1</i> <i>COMADD</i>	GPTIMER 附加 channel 1 比较标志位
<i>GPTIMER_FLAG_UP</i>	GPTIMER 计数器复位标志位

<i>GPTIMER_FLAG_ADT1CMU</i>	GPTIMER ADC 触发 1 向上计数比较匹配标志位
<i>GPTIMER_FLAG_ADT1CMD</i>	GPTIMER ADC 触发 1 向下计数比较匹配标志位
<i>GPTIMER_FLAG_ADT2CMU</i>	GPTIMER ADC 触发 2 向上计数比较匹配标志位
<i>GPTIMER_FLAG_ADT2CMD</i>	GPTIMER ADC 触发 2 向下计数比较匹配标志位
<i>GPTIMER_FLAG_CHHOUT</i>	GPTIMER 输出同时高错误标志位
<i>GPTIMER_FLAG_CHLOWT</i>	GPTIMER 输出同时低错误标志位
<i>GPTIMER_FLAG_RCEND</i>	GPTIMER 重复计数器结束标志位
<i>GPTIMER_FLAG_DIR</i>	GPTIMER 计数器方向标志位
<i>GPTIMER_FLAG_OST</i>	GPTIMER 输出停止标志位
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* reset GPTIMER0 flags */
```

```
gptimer_flag_clear (GPTIMER0, GPTIMER_FLAG_CH0);
```

## 函数 gptimer\_interrupt\_enable

函数 gptimer\_interrupt\_enable 描述见下表：

**Table 3-96. Function gptimer\_interrupt\_enable**

函数名称	gptimer_interrupt_enable
函数原型	void gptimer_interrupt_enable(uint32_t timer_periph, uint32_t interrupt)
功能描述	使能 GPTIMER 中断
先决条件	-
被调用函数	-
输入参数{in}	
timer_periph	GPTIMER 外设
GPTIMERx	x=0,1
输入参数{in}	
interrupt	GPTIMER 中断使能源
GPTIMER_INT_CH0	GPTIMER channel 0 捕获或者比较中断
GPTIMER_INT_CH1	GPTIMER channel 1 捕获或者比较中断

<i>GPTIMER_INT_OVERFLOW</i>	GPTIMER 计数器上溢中断
<i>GPTIMER_INT_UNDERFLOW</i>	GPTIMER 计数器下溢中断
<i>GPTIMER_INT_CH0COMADD</i>	GPTIMER 附加 channel 0 比较中断
<i>GPTIMER_INT_CH1COMADD</i>	GPTIMER 附加 channel 1 比较中断
<i>GPTIMER_INT_UP</i>	GPTIMER 计数器复位中断
<i>GPTIMER_INT_CREP</i>	GPTIMER 重复计数结束中断
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* enable the GPTIMER0 interrupt */
```

```
gptimer_interrupt_enable (GPTIMER0, GPTIMER_INT_CH0);
```

### 函数 `gptimer_interrupt_disable`

函数 `gptimer_interrupt_disable` 描述见下表:

**Table 3-97. Function `gptimer_interrupt_disable`**

函数名称	<code>gptimer_interrupt_disable</code>
函数原型	<code>void gptimer_interrupt_disable(uint32_t timer_periph, uint32_t interrupt)</code>
功能描述	禁能 GPTIMER 中断
先决条件	-
被调用函数	-
输入参数{in}	
<b>timer_periph</b>	GPTIMER 外设
<i>GPTIMERx</i>	x=0,1
输入参数{in}	
<b>interrupt</b>	GPTIMER 中断使能源
<i>GPTIMER_INT_CH0</i>	GPTIMER channel 0 捕获或者比较中断
<i>GPTIMER_INT_CH1</i>	GPTIMER channel 1 捕获或者比较中断
<i>GPTIMER_INT_OVERFLOW</i>	GPTIMER 计数器上溢中断
<i>GPTIMER_INT_UNDERFLOW</i>	GPTIMER 计数器下溢中断
<i>GPTIMER_INT_CH0COMADD</i>	GPTIMER 附加 channel 0 比较中断

<i>GPTIMER_INT_CH1COMADD</i>	GPTIMER 附加 channel 1 比较中断
<i>GPTIMER_INT_UP</i>	GPTIMER 计数器复位中断
<i>GPTIMER_INT_CREP</i>	GPTIMER 重复计数结束中断
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* disable the GPTIMER0 interrupt */
```

```
gptimer_interrupt_disable (GPTIMER0, GPTIMER_INT_CH0);
```

### 函数 gptimer\_interrupt\_flag\_get

函数 gptimer\_interrupt\_flag\_get 描述见下表:

**Table 3-98. Function gptimer\_interrupt\_flag\_get**

函数名称	gptimer_interrupt_flag_get
函数原型	FlagStatus gptimer_interrupt_flag_get(uint32_t timer_periph, uint32_t int_flag)
功能描述	获取中断 GPTIMER 标志位
先决条件	-
被调用函数	-
输入参数{in}	
timer_periph	GPTIMER 外设
<i>GPTIMERx</i>	x=0,1
输入参数{in}	
int_flag	GPTIMER 中断标志位
<i>GPTIMER_INT_FLAG_CH0</i>	GPTIMER channel 0 捕获或者比较中断标志位
<i>GPTIMER_INT_FLAG_CH1</i>	GPTIMER channel 1 捕获或者比较中断标志位
<i>GPTIMER_INT_FLAG_OVERFLOW</i>	GPTIMER 计数器上溢中断标志位
<i>GPTIMER_INT_FLAG_UNDERFLOW</i>	GPTIMER 计数器下溢中断标志位
<i>GPTIMER_INT_FLAG_CH0COMADD</i>	GPTIMER 附加 channel 0 比较中断标志位
<i>GPTIMER_INT_FLAG_CH1COMADD</i>	GPTIMER 附加 channel 1 比较中断标志位
<i>GPTIMER_INT_FLAG_UP</i>	GPTIMER 计数器复位中断标志位

<b>GPTIMER_INT_FLAG_CREP</b>	GPTIMER 重复计数结束中断标志位
输出参数{out}	
-	-
返回值	
<b>FlagStatus: SET or RESET</b>	

例如:

```
/* get GPTIMER0 interrupt flag */
```

```
gptimer_interrupt_flag_get (GPTIMER0, GPTIMER_INT_FLAG_CH0);
```

### 函数 gptimer\_interrupt\_flag\_clear

函数 gptimer\_interrupt\_flag\_clear 描述见下表:

**Table 3-99. Function gptimer\_interrupt\_flag\_clear**

<b>函数名称</b>	<b>gptimer_interrupt_flag_clear</b>
<b>函数原型</b>	<code>void gptimer_interrupt_flag_clear(uint32_t timer_periph, uint32_t int_flag)</code>
<b>功能描述</b>	清除中断 GPTIMER 标志位
<b>先决条件</b>	-
<b>被调用函数</b>	-
<b>输入参数{in}</b>	
<b>timer_periph</b>	GPTIMER 外设
<i>GPTIMERx</i>	x=0,1
<b>输入参数{in}</b>	
<b>int_flag</b>	GPTIMER 中断标志位
<i>GPTIMER_INT_FLAG_CH0</i>	GPTIMER channel 0 捕获或者比较中断标志位
<i>GPTIMER_INT_FLAG_CH1</i>	GPTIMER channel 1 捕获或者比较中断标志位
<i>GPTIMER_INT_FLAG_OVERFLOW</i>	GPTIMER 计数器上溢中断标志位
<i>GPTIMER_INT_FLAG_UNDERFLOW</i>	GPTIMER 计数器下溢中断标志位
<i>GPTIMER_INT_FLAG_CH0COMADD</i>	GPTIMER 附加 channel 0 比较中断标志位
<i>GPTIMER_INT_FLAG_CH1COMADD</i>	GPTIMER 附加 channel 1 比较中断标志位
<i>GPTIMER_INT_FLAG_UP</i>	GPTIMER 计数器复位中断标志位
<i>GPTIMER_INT_FLAG_CREP</i>	GPTIMER 重复计数结束中断标志位

输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* clear GPTIMER0 interrupt flag */
```

```
gptimer_interrupt_flag_clear (GPTIMER0, GPTIMER_INT_FLAG_CH0);
```

## 3.18. GTOC

GTOC可以生成请求来关闭GPTIMER输出引脚。被关闭的引脚可输出高阻（Hi-Z）态。输出关闭请求源可以来自芯片上的不同模块，如CMP和GPTIMER等。章节[3.18.1](#)描述了CMP的寄存器列表，章节[3.18.2](#)对CMP库函数进行说明。

### 3.18.1. 外设寄存器说明

GTOC寄存器列表如下表所示：

表 3-477. GTOC 寄存器

寄存器名称	寄存器描述
GTOCx_CFG	GTOCx配置寄存器
GTOCx_OCRCTL	GTOCx输出关闭请求控制寄存器
GTOCx_WP	GTOCx写保护寄存器
GTOCx_ECRCTL	GTOCx扩展关闭请求控制寄存器

### 3.18.2. 外设库函数说明

GTOC库函数列表如下表所示：

表 3-478. GTOC 库函数

库函数名称	库函数描述
gtoc_deinit	复位GTOC
gtoc_output_closing_request_enable	使能GTOC输出关闭请求
gtoc_gptimer_trigger_status_get	获取GPTIMER外部触发输入状态
gtoc_software_request_generate	生成软件输出关闭请求
gtoc_software_request_stop	停止软件输出关闭请求
gtoc_input_detection_mode_select	选择GTOCx_IN输入检测模式
gtoc_input_polarity_config	配置GTOCx_IN输入极性
gtoc_digital_filter_enable	使能GTOC数字滤波
gtoc_digital_filter_disable	禁能GTOC数字滤波
gtoc_digital_filter_config	配置GTOC数字滤波

库函数名称	库函数描述
gtoc_output_closing_request_mask	屏蔽GTOC输出关闭请求
gtoc_register_write_enable	使能GTOC扩展关闭请求控制寄存器写
gtoc_register_write_disable	禁能GTOC扩展关闭请求控制寄存器写
gtoc_extended_closing_request_enable	使能GTOC扩展关闭请求
gtoc_extended_closing_request_disable	禁能GTOC扩展关闭请求
gtoc_extended_closing_request_config	配置GTOC扩展关闭请求
gtoc_extended_closing_request_mask	屏蔽GTOC扩展关闭请求
gtoc_flag_get	获取GTOC标志位
gtoc_flag_clear	清除GTOC标志位
gtoc_interrupt_flag_get	获取GTOC中断标志位
gtoc_interrupt_flag_clear	清除GTOC中断标志位

### 函数 gtoc\_deinit

函数gtoc\_deinit描述见下表：

表 3-479. 函数 gtoc\_deinit

函数名称	gtoc_deinit
函数原型	void gtoc_deinit(uint32_t gtoc_periph)
功能描述	复位 GTOC
先决条件	-
被调用函数	-
输入参数{in}	
gtoc_periph	GTOC 外设
GTOCx	x=0,1,2,3
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* deinitialize GTOC0 */
```

```
dac_deinit(GTOC0);
```

### 函数 gtoc\_output\_closing\_request\_enable

函数gtoc\_output\_closing\_request\_enable描述见下表：

表 3-480. 函数 gtoc\_output\_closing\_request\_enable

函数名称	gtoc_output_closing_request_enable
函数原型	void gtoc_output_closing_request_enable(uint32_t gtoc_periph, uint16_t gtoc_ocr_source)



功能描述	使能 GTOC 输出关闭请求
先决条件	-
被调用函数	-
输入参数{in}	
gtoc_periph	GTOC 外设
GTOCx	x=0,1,2,3
输入参数{in}	
gtoc_ocr_source	输出关闭请求源
GTOC_OCR_SOU RCE_GTOCPIN	GTOCx_IN 引脚输入检测
GTOC_OCR_SOU RCE_GPTIMER	GPTIMER 输出错误检测
GTOC_OCR_SOU RCE_HXTALSTUC K	HXTAL 卡住检测
GTOC_OCR_SOU RCE_LOCKUP	CPU 锁定检测
GTOC_OCR_SOU RCE_CMP0	CMP0 有效边沿检测
GTOC_OCR_SOU RCE_CMP1	CMP1 有效边沿检测
GTOC_OCR_SOU RCE_CMP2	CMP2 有效边沿检测
GTOC_OCR_SOU RCE_CMP3	CMP3 有效边沿检测
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* enable GTOC0 output closing request generated by GTOCx_IN pin input detection */
gtoc_output_closing_request_enable(GTOC0, GTOC_OCR_SOURCE_GTOCPIN);
```

### 函数 gtoc\_gptimer\_trigger\_status\_get

函数gtoc\_gptimer\_trigger\_status\_get描述见下表：

表 3-481. 函数 gtoc\_gptimer\_trigger\_status\_get

函数名称	gtoc_gptimer_trigger_status_get
函数原型	uint32_t gtoc_gptimer_trigger_status_get(uint32_t gtoc_periph)
功能描述	获取 GPTIMER 外部触发输入状态

先决条件	-
被调用函数	-
输入参数{in}	
gtoc_periph	GTOC 外设
GTOCx	x=0,1,2,3
输出参数{out}	
-	-
返回值	
uint32_t	触发输入状态
GPTIMER_TRIGGE R_STATUS_HIGH	触发输入是高
GPTIMER_TRIGGE R_STATUS_LOW	触发输入是低

例如:

```
uint32_t status;
```

```
/* get GPTIMER external trigger input status from GTOC0 */
```

```
status = gtoc_gptimer_trigger_status_get(GTOC0);
```

### 函数 gtoc\_software\_request\_generate

函数gtoc\_software\_request\_generate描述见下表:

表 3-482. 函数 gtoc\_software\_request\_generate

函数名称	gtoc_software_request_generate
函数原型	void gtoc_software_request_generate(uint32_t gtoc_periph)
功能描述	生成软件输出关闭请求
先决条件	-
被调用函数	-
输入参数{in}	
gtoc_periph	GTOC 外设
GTOCx	x=0,1,2,3
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* generate software output closing request from GTOC0 */
```

```
gtoc_software_request_generate(GTOC0);
```

## 函数 gtoc\_software\_request\_stop

函数gtoc\_software\_request\_stop描述见下表：

表 3-483. 函数 gtoc\_software\_request\_stop

函数名称	gtoc_software_request_stop
函数原型	void gtoc_software_request_stop(uint32_t gtoc_periph)
功能描述	停止软件输出关闭请求
先决条件	-
被调用函数	-
输入参数{in}	
gtoc_periph	GTOC 外设
GTOCx	x=0,1,2,3
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* stop software output closing request from GTOC0 */
```

```
gtoc_software_request_stop(GTOC0);
```

## 函数 gtoc\_input\_detection\_mode\_select

函数gtoc\_input\_detection\_mode\_select描述见下表：

表 3-484. 函数 gtoc\_input\_detection\_mode\_select

函数名称	gtoc_input_detection_mode_select
函数原型	void gtoc_input_detection_mode_select(uint32_t gtoc_periph, uint32_t detection_mode)
功能描述	选择 GTOCx_IN 输入检测模式
先决条件	-
被调用函数	-
输入参数{in}	
gtoc_periph	GTOC 外设
GTOCx	x=0,1,2,3
输入参数{in}	
detection_mode	GTOCx_IN 输入检测模式
GTOC_INPUT_DETECTION_LEVEL	电平检测
GTOC_INPUT_DETECTION_EDGE	边沿检测
输出参数{out}	

-	-
返回值	
-	-

例如:

```
/* select GTOC0_IN input detection mode */
```

```
gtoc_input_detection_mode_select(GTOC0, GTOC_INPUT_DETECTION_LEVEL);
```

### 函数 **gtoc\_input\_polarity\_config**

函数gtoc\_input\_polarity\_config描述见下表:

**表 3-485. 函数 gtoc\_input\_polarity\_config**

函数名称	gtoc_input_polarity_config
函数原型	void gtoc_input_polarity_config(uint32_t gtoc_periph, uint32_t input_polarity)
功能描述	配置 GTOCx_IN 输入极性
先决条件	-
被调用函数	-
输入参数{in}	
gtoc_periph	GTOC 外设
GTOCx	x=0,1,2,3
输入参数{in}	
input_polarity	GTOCx_IN 输入极性
GTOC_INPUT_POLARITY_NONINVERTED	GTOCx_IN 输入不反相
GTOC_INPUT_DETECTION_EDGE	GTOCx_IN 输入反相
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* configure GTOC0_IN input polarity */
```

```
gtoc_input_polarity_config(GTOC0, GTOC_INPUT_POLARITY_NONINVERTED);
```

### 函数 **gtoc\_digital\_filter\_enable**

函数gtoc\_digital\_filter\_enable描述见下表:

**表 3-486. 函数 gtoc\_digital\_filter\_enable**

函数名称	gtoc_digital_filter_enable
------	----------------------------

函数原型	void gtoc_digital_filter_enable(uint32_t gtoc_periph)
功能描述	使能 GTOC 数字滤波
先决条件	-
被调用函数	-
输入参数{in}	
gtoc_periph	GTOC 外设
GTOCx	x=0,1,2,3
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* enable GTOC0 digital filter */
gtoc_digital_filter_enable(GTOC0);
```

### 函数 gtoc\_digital\_filter\_disable

函数gtoc\_digital\_filter\_disable描述见下表：

表 3-487. 函数 gtoc\_digital\_filter\_disable

函数名称	gtoc_digital_filter_disable
函数原型	void gtoc_digital_filter_disable(uint32_t gtoc_periph)
功能描述	禁能 GTOC 数字滤波
先决条件	-
被调用函数	-
输入参数{in}	
gtoc_periph	GTOC 外设
GTOCx	x=0,1,2,3
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* disable GTOC0 digital filter */
gtoc_digital_filter_disable(GTOC0);
```

### 函数 gtoc\_digital\_filter\_config

函数gtoc\_digital\_filter\_config描述见下表：

表 3-488. 函数 `gtoc_digital_filter_config`

函数名称	<code>gtoc_digital_filter_config</code>
函数原型	<code>void gtoc_digital_filter_config(uint32_t gtoc_periph, gtoc_filter_sampling_freq_enum sampling_frequency, uint32_t sampling_number)</code>
功能描述	配置 GTOC 数字滤波
先决条件	-
被调用函数	-
输入参数{in}	
<code>gtoc_periph</code>	GTOC 外设
<code>GTOCx</code>	$x=0,1,2,3$
输入参数{in}	
<code>sampling_frequency</code>	采样频率
<code>GTOC_SAMPLING_FREQUENCY_DIV1</code>	GTOC 数字滤波器采样频率 $f_{HCLK}$
<code>GTOC_SAMPLING_FREQUENCY_DIV8</code>	GTOC 数字滤波器采样频率 $f_{HCLK}/8$
<code>GTOC_SAMPLING_FREQUENCY_DIV32</code>	GTOC 数字滤波器采样频率 $f_{HCLK}/32$
<code>GTOC_SAMPLING_FREQUENCY_DIV128</code>	GTOC 数字滤波器采样频率 $f_{HCLK}/128$
<code>GTOC_SAMPLING_FREQUENCY_DIV2</code>	GTOC 数字滤波器采样频率 $f_{HCLK}/2$
<code>GTOC_SAMPLING_FREQUENCY_DIV4</code>	GTOC 数字滤波器采样频率 $f_{HCLK}/4$
<code>GTOC_SAMPLING_FREQUENCY_DIV16</code>	GTOC 数字滤波器采样频率 $f_{HCLK}/16$
<code>GTOC_SAMPLING_FREQUENCY_DIV64</code>	GTOC 数字滤波器采样频率 $f_{HCLK}/64$
<code>GTOC_SAMPLING_FREQUENCY_DIV256</code>	GTOC 数字滤波器采样频率 $f_{HCLK}/256$
<code>GTOC_SAMPLING</code>	GTOC 数字滤波器采样频率 $f_{HCLK}/512$

<code>_FREQUENCY_DIV12</code>	
输入参数{in}	
<code>sampling_number</code>	GTOCx 数字滤波采样数
<code>GTOC_SAMPLING_NUM_3_TIMES</code>	GTOC 数字滤波采样数是 3 次
<code>GTOC_SAMPLING_NUM_4_TIMES</code>	GTOC 数字滤波采样数是 4 次
<code>GTOC_SAMPLING_NUM_5_TIMES</code>	GTOC 数字滤波采样数是 5 次
<code>GTOC_SAMPLING_NUM_6_TIMES</code>	GTOC 数字滤波采样数是 6 次
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* configure GTOC0 digital filter */
```

```
gtoc_digital_filter_config(GTOC0, GTOC_SAMPLING_FREQUENCY_DIV1,
GTOC_SAMPLING_NUM_3_TIMES);
```

### 函数 `gtoc_output_closing_request_mask`

函数 `gtoc_output_closing_request_mask` 描述见下表：

表 3-489. 函数 `gtoc_output_closing_request_mask`

函数名称	<code>gtoc_output_closing_request_mask</code>
函数原型	<code>void gtoc_output_closing_request_mask(uint32_t gtoc_periph, uint32_t mask_source)</code>
功能描述	屏蔽 GTOC 输出关闭请求
先决条件	-
被调用函数	-
输入参数{in}	
<code>gtoc_periph</code>	GTOC 外设
<code>GTOCx</code>	x=0,1,2,3
输入参数{in}	
<code>mask_source</code>	输出关闭请求的屏蔽源
<code>GTOC_OCRMKSEL_NOT_MASKED</code>	不屏蔽输出关闭请求
<code>GTOC_OCRMKSEL_GPTIMER0_CH0</code>	GPTIMER0_CH0 信号屏蔽输出关闭请求

GTOC_OCRMKSE L_GPTIMER0_CH1	GPTIMER0_CH1 信号屏蔽输出关闭请求
GTOC_OCRMKSE L_GPTIMER1_CH0	GPTIMER1_CH0 信号屏蔽输出关闭请求
GTOC_OCRMKSE L_GPTIMER1_CH1	GPTIMER1_CH1 信号屏蔽输出关闭请求
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* mask GTOC output closing request */
```

```
gtoc_output_closing_request_mask(GTOC0, GTOC_OCRMKSEL_GPTIMER0_CH0);
```

### 函数 gtoc\_register\_write\_enable

函数gtoc\_register\_write\_enable描述见下表：

表 3-490. 函数 gtoc\_register\_write\_enable

函数名称	gtoc_register_write_enable
函数原型	void gtoc_register_write_enable(uint32_t gtoc_periph)
功能描述	使能 GTOC 扩展关闭请求控制寄存器写
先决条件	-
被调用函数	-
输入参数{in}	
gtoc_periph	GTOC 外设
GTOCx	x=0,1,2,3
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* enable GTOC0 extended closing request control register write */
```

```
gtoc_register_write_enable(GTOC0);
```

### 函数 gtoc\_register\_write\_disable

函数gtoc\_register\_write\_disable描述见下表：

表 3-491. 函数 gtoc\_register\_write\_disable

函数名称	gtoc_register_write_disable
------	-----------------------------



函数原型	void gtoc_register_write_disable(uint32_t gtoc_periph)
功能描述	禁能 GTOC 扩展关闭请求控制寄存器写
先决条件	-
被调用函数	-
输入参数{in}	
gtoc_periph	GTOC 外设
GTOCx	x=0,1,2,3
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* disable GTOC0 extended closing request control register write */
gtoc_register_write_disable(GTOC0);
```

### 函数 gtoc\_extended\_closing\_request\_enable

函数gtoc\_extended\_closing\_request\_enable描述见下表：

表 3-492. 函数 gtoc\_extended\_closing\_request\_enable

函数名称	gtoc_extended_closing_request_enable
函数原型	void gtoc_extended_closing_request_enable(uint32_t gtoc_periph)
功能描述	使能 GTOC 扩展关闭请求
先决条件	-
被调用函数	-
输入参数{in}	
gtoc_periph	GTOC 外设
GTOCx	x=0,1,2,3
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* enable extended closing request from GTOC0 */
gtoc_extended_closing_request_enable(GTOC0);
```

### 函数 gtoc\_extended\_closing\_request\_disable

函数gtoc\_extended\_closing\_request\_disable描述见下表：

表 3-493. 函数 `gtoc_extended_closing_request_disable`

函数名称	<code>gtoc_extended_closing_request_disable</code>
函数原型	<code>void gtoc_extended_closing_request_disable(uint32_t gtoc_periph)</code>
功能描述	禁能 GTOC 扩展关闭请求
先决条件	-
被调用函数	-
输入参数{in}	
<b>gtoc_periph</b>	GTOC peripheral
<i>GTOCx</i>	x=0,1,2,3
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* disable extended closing request from GTOC0 */
```

```
gtoc_extended_closing_request_disable(GTOC0);
```

### 函数 `gtoc_extended_closing_request_config`

函数 `gtoc_extended_closing_request_config` 描述见下表：

表 3-494. 函数 `gtoc_extended_closing_request_config`

函数名称	<code>gtoc_extended_closing_request_config</code>
函数原型	<code>void gtoc_extended_closing_request_config(uint32_t gtoc_periph, gtoc_ext_closing_req_source_enum gtoc_ecr_source, uint32_t valid_level)</code>
功能描述	配置 GTOC 扩展关闭请求
先决条件	-
被调用函数	-
输入参数{in}	
<b>gtoc_periph</b>	GTOC 外设
<i>GTOCx</i>	x=0,1,2,3
输入参数{in}	
<b>gtoc_ecr_source</b>	GTOC 扩展关闭请求源
<i>GTOC_ECR_SOU_RCE_CMP0</i>	CMP0 电平检测
<i>GTOC_ECR_SOU_RCE_CMP1</i>	CMP1 电平检测
<i>GTOC_ECR_SOU_RCE_CMP2</i>	CMP2 电平检测
<i>GTOC_ECR_SOU_RCE_CMP3</i>	CMP3 电平检测

<i>GTOC_ECR_SOU</i> <i>RCE_GTOCPIN</i>	GTOCx_IN 输入电平检测
输入参数{in}	
<b>valid_level</b>	有效电平
<i>GTOC_VALID_LEV</i> <i>EL_LOW</i>	低电平有效
<i>GTOC_VALID_LEV</i> <i>EL_HIGH</i>	高电平有效
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* configure extended closing request from GTOC0 */
```

```
gtoc_extended_closing_request_disable(GTOC0,          GTOC_ECR_SOURCE_CMP0,
GTOC_VALID_LEVEL_LOW);
```

### 函数 `gtoc_extended_closing_request_mask`

函数 `gtoc_extended_closing_request_mask` 描述见下表:

**表 3-495. 函数 `gtoc_extended_closing_request_mask`**

函数名称	<code>gtoc_extended_closing_request_mask</code>
函数原型	<code>void gtoc_extended_closing_request_mask(uint32_t gtoc_periph, uint32_t mask_source)</code>
功能描述	屏蔽 GTOC 扩展关闭请求
先决条件	-
被调用函数	-
输入参数{in}	
<b>gtoc_periph</b>	GTOC 外设
<i>GTOCx</i>	x=0,1,2,3
输入参数{in}	
<b>mask_source</b>	扩展关闭请求的屏蔽源
<i>GTOC_ECRMKSE</i> <i>L_NOT_MASKED</i>	不屏蔽扩展关闭请求
<i>GTOC_ECRMKSE</i> <i>L_GPTIMER0_CH0</i>	GPTIMER0_CH0 信号屏蔽扩展关闭请求
<i>GTOC_ECRMKSE</i> <i>L_GPTIMER0_CH1</i>	GPTIMER0_CH1 信号屏蔽扩展关闭请求
<i>GTOC_ECRMKSE</i> <i>L_GPTIMER1_CH0</i>	GPTIMER1_CH0 信号屏蔽扩展关闭请求

<i>GTOC_ECRMKSEL_GPTIMER1_CH1</i>	GPTIMER1_CH1 信号屏蔽扩展关闭请求
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* mask extended closing request from GTOC0 */
```

```
gtoc_extended_closing_request_mask(GTOC0, GTOC_ECRMKSEL_GPTIMER0_CH0);
```

### 函数 **gtoc\_flag\_get**

函数gtoc\_flag\_get描述见下表：

表 3-496. 函数 **gtoc\_flag\_get**

函数名称	gtoc_flag_get
函数原型	FlagStatus gtoc_flag_get(uint32_t gtoc_periph, uint32_t flag)
功能描述	获取 GTOC 标志位
先决条件	-
被调用函数	-
输入参数{in}	
gtoc_periph	GTOC 外设
<i>GTOCx</i>	x=0,1,2,3
输入参数{in}	
flag	GTOC 标志位
<i>GTOC_FLAG_INIF</i>	GTOCx_IN 输入中断标志位
<i>GTOC_FLAG_OFV EIF</i>	GPTIMER 输出错误或者 CMP 有效边沿中断标志位
<i>GTOC_FLAG_HXT ALSDF</i>	HXTAL 卡住检测标志位
<i>GTOC_FLAG_LOC KUPDF</i>	CPU 锁定检测标志位
输出参数{out}	
-	-
返回值	
FlagStatus	SET 或 RESET

例如：

```
FlagStatus status;
```

```
/* get GTOC0 flag */
```

```
status = gtoc_flag_get(GTOC0, GTOC_FLAG_INIF);
```

**函数 gtoc\_flag\_clear**

函数gtoc\_flag\_clear描述见下表：

**表 3-497. 函数 gtoc\_flag\_clear**

函数名称	gtoc_flag_clear
函数原型	void gtoc_flag_clear(uint32_t gtoc_periph, uint32_t flag)
功能描述	清除 GTOC 标志位
先决条件	-
被调用函数	-
输入参数{in}	
gtoc_periph	GTOC 外设
GTOCx	x=0,1,2,3
输入参数{in}	
flag	GTOC 标志位
GTOC_FLAG_INIF	GTOCx_IN 输入中断标志位
GTOC_FLAG_OFV EIF	GPTIMER 输出错误或者 CMP 有效边沿中断标志位
GTOC_FLAG_HXT ALSDF	HXTAL 卡住检测标志位
GTOC_FLAG_LOC KUPDF	CPU 锁定检测标志位
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* clear GTOC0 flag */
gtoc_flag_clear(GTOC0, GTOC_FLAG_INIF);
```

**函数 gtoc\_interrupt\_flag\_get**

函数gtoc\_interrupt\_flag\_get描述见下表：

**表 3-498. 函数 gtoc\_interrupt\_flag\_get**

函数名称	gtoc_interrupt_flag_get
函数原型	FlagStatus gtoc_interrupt_flag_get(uint32_t gtoc_periph, uint32_t int_flag)
功能描述	获取 GTOC 中断标志位
先决条件	-
被调用函数	-
输入参数{in}	
gtoc_periph	GTOC 外设

<i>GTOCx</i>	<i>x</i> =0,1,2,3
输入参数{in}	
<b>int_flag</b>	GTOC 中断标志位
<i>GTOC_INT_FLAG_INIF</i>	<i>GTOCx_IN</i> 输入中断标志位
<i>GTOC_INT_FLAG_OFVEIF</i>	GPTIMER 输出错误或者 CMP 有效边沿中断标志位
输出参数{out}	
-	-
返回值	
<b>FlagStatus</b>	SET 或 RESET

例如:

```
FlagStatus status;
```

```
/* get GTOC0 interrupt flag */
```

```
status = gtoc_interrupt_flag_get(GTOC0, GTOC_INT_FLAG_INIF);
```

### 函数 **gtoc\_interrupt\_flag\_clear**

函数 **gtoc\_interrupt\_flag\_clear** 描述见下表:

表 3-499. 函数 **gtoc\_interrupt\_flag\_clear**

函数名称	<b>gtoc_interrupt_flag_clear</b>
函数原型	<code>void gtoc_interrupt_flag_clear(uint32_t gtoc_periph, uint32_t int_flag)</code>
功能描述	清除 GTOC 中断标志位
先决条件	-
被调用函数	-
输入参数{in}	
<b>gtoc_periph</b>	GTOC 外设
<i>GTOCx</i>	<i>x</i> =0,1,2,3
输入参数{in}	
<b>int_flag</b>	GTOC 中断标志位
<i>GTOC_INT_FLAG_INIF</i>	<i>GTOCx_IN</i> 输入中断标志位
<i>GTOC_INT_FLAG_OFVEIF</i>	GPTIMER 输出错误或者 CMP 有效边沿中断标志位
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* clear GTOC0 interrupt flag */
```

```
gtoc_interrupt_flag_clear(GTOC0, GTOC_INT_FLAG_INIF);
```

## 3.19. 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[3.19.1](#)描述了I2C的寄存器列表，章节[3.19.2](#)对I2C库函数进行说明。

### 3.19.1. 外设寄存器说明

I2C寄存器列表如下表所示：

**表 3-500. 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

### 3.19.2. 外设库函数说明

I2C库函数列表如下表所示：

**表 3-501. 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_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

库函数名称	库函数描述
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_receivied_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
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_mode_enable	使能 SMBus 模式
i2c_smbus_mode_disable	禁能 SMBus 模式
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



库函数名称	库函数描述
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 3-502. 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 3-503. 函数 i2c\_deinit

函数名称	i2c_deinit
函数原型	void i2c_deinit(void);
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}	
-	-
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* reset I2C */
```

```
i2c_deinit();
```

### 函数 i2c\_timing\_config

函数i2c\_timing\_config描述见下表：

表 3-504. 函数 i2c\_timing\_config

函数名称	i2c_timing_config
函数原型	void i2c_timing_config(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}	
<b>psc</b>	0-0xf, 时序分频
输入参数{in}	
<b>scl_dely</b>	0-0xf, 数据建立时间
输入参数{in}	
<b>sda_dely</b>	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(0x1, 0x2, 0x1);
```

### 函数 i2c\_digital\_noise\_filter\_config

函数i2c\_digital\_noise\_filter\_config描述见下表：

表 3-505. 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 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}	
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：

```
/* I2C digital filter filters spikes with a length of up to 1  $t_{I2CCLK}$  */
```

```
i2c_digital_noise_filter_config(FILTER_LENGTH_1);
```

### 函数 i2c\_master\_clock\_config

函数i2c\_master\_clock\_config描述见下表：

表 3-506. 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 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}	
sclh	0-0xff, SCL 高电平周期

输入参数{in}	
<b>scll</b>	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(0x0f, 0x0f);
```

### 函数 i2c\_master\_addressing

函数i2c\_master\_addressing描述见下表:

表 3-507. 函数 i2c\_master\_addressing

函数名称	i2c_master_addressing
函数原型	void i2c_master_addressing(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}	
<b>address</b>	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}	
<b>trans_direction</b>	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(0x82, I2C_MASTER_TRANSMIT);
```

### 函数 i2c\_address10\_header\_enable

函数i2c\_address10\_header\_enable描述见下表:

表 3-508. 函数 i2c\_address10\_header\_enable

函数名称	i2c_address10_header_enable
函数原型	void i2c_address10_header_enable(void);
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}	
-	-
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();
```

### 函数 i2c\_address10\_header\_disable

函数i2c\_address10\_header\_disable描述见下表：

表 3-509. 函数 i2c\_address10\_header\_disable

函数名称	i2c_address10_header_disable
函数原型	void i2c_address10_header_disable(void);
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}	
-	-
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();
```

### 函数 i2c\_address10\_enable

函数i2c\_address10\_enable描述见下表：

表 3-510. 函数 i2c\_address10\_enable

函数名称	i2c_address10_enable
函数原型	void i2c_address10_enable(void);
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}	
-	-
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* enable 10-bit addressing mode in master mode */
```

```
i2c_address10_enable();
```

### 函数 i2c\_address10\_disable

函数i2c\_address10\_disable描述见下表：

表 3-511. 函数 i2c\_address10\_disable

函数名称	i2c_address10_disable
函数原型	void i2c_address10_disable(void);
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}	
-	-
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* disable 10-bit addressing mode in master mode */
```

```
i2c_address10_disable();
```

### 函数 i2c\_automatic\_end\_enable

函数i2c\_automatic\_end\_enable描述见下表：

表 3-512. 函数 i2c\_automatic\_end\_enable

函数名称	i2c_automatic_end_enable
函数原型	void i2c_automatic_end_enable(void);
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}	
-	-
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* enable I2C automatic end mode in master mode */
```

```
i2c_automatic_end_enable();
```

### 函数 i2c\_automatic\_end\_disable

函数i2c\_automatic\_end\_disable描述见下表：

表 3-513. 函数 i2c\_automatic\_end\_disable

函数名称	i2c_automatic_end_disable
函数原型	void i2c_automatic_end_disable(void);
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}	
-	-
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* disable I2C automatic end mode in master mode */
```

```
i2c_automatic_end_disable();
```

### 函数 i2c\_slave\_response\_to\_gcall\_enable

函数i2c\_slave\_response\_to\_gcall\_enable描述见下表：

表 3-514. 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(void);
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}	
-	-
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* enable the response to a general call */
```

```
i2c_slave_response_to_gcall_enable();
```

### 函数 i2c\_slave\_response\_to\_gcall\_disable

函数i2c\_slave\_response\_to\_gcall\_disable描述见下表：

表 3-515. 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(void);
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}	
-	-
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* disable the response to a general call */
```

```
i2c_slave_response_to_gcall_disable();
```

### 函数 i2c\_stretch\_scl\_low\_enable

函数i2c\_stretch\_scl\_low\_enable描述见下表：



表 3-516. 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(void);
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}	
-	-
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();
```

### 函数 i2c\_stretch\_scl\_low\_disable

函数i2c\_stretch\_scl\_low\_disable描述见下表：

表 3-517. 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(void);
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}	
-	-
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();
```

### 函数 i2c\_address\_config

函数i2c\_address\_config描述见下表：

表 3-518. 函数 i2c\_address\_config

函数名称	i2c_address_config
函数原型	void i2c_address_config(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}	
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(0x82, I2C_ADDFORMAT_7BITS);
```

### 函数 i2c\_address\_bit\_compare\_config

函数i2c\_address\_bit\_compare\_config描述见下表：

表 3-519. 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 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}	
compare_bits	需要进行比较的位
ADDRESS_NO_COMPARE	ADDRESS[7:1]所有位都被屏蔽
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

PARE	
ADDRESS_BIT4_COM PARE	地址的第 4 位需要进行比较
ADDRESS_BIT5_COM PARE	地址的第 5 位需要进行比较
ADDRESS_BIT6_COM PARE	地址的第 6 位需要进行比较
ADDRESS_BIT7_COM PARE	地址的第 7 位需要进行比较
ADDRESS_ALL_COMPARE	无屏蔽，所有位都需要被比较
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(ADDRESS_BIT1_COMPARE);
```

### 函数 i2c\_address\_disable

函数i2c\_address\_disable描述见下表：

表 3-520. 函数 i2c\_address\_disable

函数名称	i2c_address_disable
函数原型	void i2c_address_disable(void);
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}	
-	-
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* disable i2c address in slave mode */
```

```
i2c_address_disable();
```

## 函数 i2c\_second\_address\_config

函数i2c\_second\_address\_config描述见下表：

表 3-521. 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 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}	
address	I2C 地址
输入参数{in}	
addr_mask	不需要进行比较的地址
ADDRESS2_NO_MASK	无屏蔽，全部都需要进行比较
ADDRESS2_MASK_BIT1	ADDRESS2[1]屏蔽， ADDRESS2[7:2]进行比较
ADDRESS2_MASK_BIT1_2	ADDRESS2[2:1]屏蔽， ADDRESS2[7:3]进行比较
ADDRESS2_MASK_BIT1_3	ADDRESS2[3:1]屏蔽， ADDRESS2[7:4]进行比较
ADDRESS2_MASK_BIT1_4	ADDRESS2[4:1]屏蔽， ADDRESS2[7:5]进行比较
ADDRESS2_MASK_BIT1_5	ADDRESS2[5:1]屏蔽， ADDRESS2[7:6]进行比较
ADDRESS2_MASK_BIT1_6	ADDRESS2[6:1]屏蔽， ADDRESS2[7]进行比较
ADDRESS2_MASK_ALL	ADDRESS2[7:1]屏蔽
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* configure i2c second slave address */
```

```
i2c_second_address_config(0x82, ADDRESS2_MASK_BIT1_2);
```

## 函数 i2c\_second\_address\_disable

函数i2c\_second\_address\_disable描述见下表：

表 3-522. 函数 i2c\_second\_address\_disable

函数名称	i2c_second_address_disable
函数原型	void i2c_second_address_disable(void);
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}	
-	-
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* disable i2c second address in slave mode */
```

```
i2c_second_address_disable();
```

### 函数 i2c\_receved\_address\_get

函数i2c\_receved\_address\_get描述见下表：

表 3-523. 函数 i2c\_receved\_address\_get

函数名称	i2c_receved_address_get
函数原型	uint32_t i2c_receved_address_get(void);
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}	
-	-
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();
```

### 函数 i2c\_slave\_byte\_control\_enable

函数i2c\_slave\_byte\_control\_enable描述见下表：

表 3-524. 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(void);
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}	
-	-
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* enable slave byte control */
i2c_slave_byte_control_enable();
```

### 函数 i2c\_slave\_byte\_control\_disable

函数i2c\_slave\_byte\_control\_disable描述见下表：

表 3-525. 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(void);
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}	
-	-
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* disable slave byte control */
i2c_slave_byte_control_disable();
```

### 函数 i2c\_nack\_enable

函数i2c\_nack\_enable描述见下表：

表 3-526. 函数 i2c\_nack\_enable

函数名称	i2c_nack_enable
函数原型	void i2c_nack_enable(void);
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}	
-	-
输出参数{out}	
-	-
返回值	
-	-

例如：

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

```
i2c_nack_enable();
```

### 函数 i2c\_wakeup\_from\_deepsleep\_enable

函数i2c\_wakeup\_from\_deepsleep\_enable描述见下表：

表 3-527. 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(void);
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}	
-	-
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* enable wakeup from Deep-sleep mode */
```

```
i2c_wakeup_from_deepsleep_enable();
```

### 函数 i2c\_wakeup\_from\_deepsleep\_disable

函数i2c\_wakeup\_from\_deepsleep\_disable描述见下表：

表 3-528. 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(void);
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}	
-	-
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* disable wakeup from Deep-sleep mode */
```

```
i2c_wakeup_from_deepsleep_disable();
```

### 函数 i2c\_enable

函数i2c\_enable描述见下表：

表 3-529. 函数 i2c\_enable

函数名称	i2c_enable
函数原型	void i2c_enable(void);
功能描述	使能 I2C
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 I2C */
```

```
i2c_enable();
```

### 函数 i2c\_disable

函数i2c\_disable描述见下表：



表 3-530. 函数 i2c\_disable

函数名称	i2c_disable
函数原型	void i2c_disable(void);
功能描述	禁能 I2C
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 I2C */
```

```
i2c_disable();
```

### 函数 i2c\_start\_on\_bus

函数i2c\_start\_on\_bus描述见下表：

表 3-531. 函数 i2c\_start\_on\_bus

函数名称	i2c_start_on_bus
函数原型	void i2c_start_on_bus(void);
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}	
-	-
输出参数{out}	
-	-
返回值	
-	-

例如：

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

```
i2c_start_on_bus();
```

### 函数 i2c\_stop\_on\_bus

函数i2c\_stop\_on\_bus描述见下表：

表 3-532. 函数 i2c\_stop\_on\_bus

函数名称	i2c_stop_on_bus
函数原型	void i2c_stop_on_bus(void);
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}	
-	-
输出参数{out}	
-	-
返回值	
-	-

例如：

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

```
i2c_stop_on_bus();
```

### 函数 i2c\_data\_transmit

函数i2c\_data\_transmit描述见下表：

表 3-533. 函数 i2c\_data\_transmit

函数名称	i2c_data_transmit
函数原型	void i2c_data_transmit(uint8_t data);
功能描述	发送数据
先决条件	-
被调用函数	-
输入参数{in}	
data	transmit data
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* I2C transmit data */
```

```
i2c_data_transmit(0x80);
```

### 函数 i2c\_data\_receive

函数i2c\_data\_receive描述见下表：

表 3-534. 函数 i2c\_data\_receive

函数名称	i2c_data_receive
函数原型	uint8_t i2c_data_receive(void);
功能描述	接收数据
先决条件	-
被调用函数	-
输入参数{in}	
-	-
输出参数{out}	
-	-
返回值	
uint32_t	received data (0x00-0xFF)

例如：

```
/* I2C receive data */
uint32_t i2c_receiver;
i2c_receiver = i2c_data_receive();
```

### 函数 i2c\_reload\_enable

函数i2c\_reload\_enable描述见下表：

表 3-535. 函数 i2c\_reload\_enable

函数名称	i2c_reload_enable
函数原型	void i2c_reload_enable(void);
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}	
-	-
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* enable I2C reload mode */
i2c_reload_enable();
```

### 函数 i2c\_reload\_disable

函数i2c\_reload\_disable描述见下表：

表 3-536. 函数 i2c\_reload\_disable

函数名称	i2c_reload_disable
函数原型	void i2c_reload_disable(void);
功能描述	禁能 I2C 重载模式
先决条件	-
被调用函数	-
输入参数{in}	
-	-
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* disable I2C reload mode */
```

```
i2c_reload_disable();
```

### 函数 i2c\_transfer\_byte\_number\_config

函数i2c\_transfer\_byte\_number\_config描述见下表：

表 3-537. 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(uint8_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}	
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(0xFF);
```

### 函数 i2c\_dma\_enable

函数i2c\_dma\_enable描述见下表：

表 3-538. 函数 i2c\_dma\_enable

函数名称	i2c_dma_enable
函数原型	void i2c_dma_enable(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}	
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(I2C_DMA_RECEIVE);
```

### 函数 i2c\_dma\_disable

函数i2c\_dma\_disable描述见下表：

表 3-539. 函数 i2c\_dma\_disable

函数名称	i2c_dma_disable
函数原型	void i2c_dma_disable(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}	
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(I2C_DMA_RECEIVE);
```

## 函数 i2c\_pec\_transfer

函数i2c\_pec\_transfer描述见下表：

**表 3-540. 函数 i2c\_pec\_transfer**

函数名称	i2c_pec_transfer
函数原型	void i2c_pec_transfer(void);
功能描述	I2C 传输 PEC 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：

```
/* I2C transfers PEC value */
```

```
i2c_pec_transfer();
```

## 函数 i2c\_pec\_enable

函数i2c\_pec\_enable描述见下表：

**表 3-541. 函数 i2c\_pec\_enable**

函数名称	i2c_pec_enable
函数原型	void i2c_pec_enable(void);
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}	
-	-
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* enable I2C PEC calculation */
```

```
i2c_pec_enable();
```

## 函数 i2c\_pec\_disable

函数i2c\_pec\_disable描述见下表：

**表 3-542. 函数 i2c\_pec\_disable**

函数名称	i2c_pec_disable
函数原型	void i2c_pec_disable(void);
功能描述	禁用报文错误校验
先决条件	-
被调用函数	-
输入参数{in}	
-	-
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* disable I2C PEC calculation */
```

```
i2c_pec_disable();
```

## 函数 i2c\_pec\_value\_get

函数i2c\_pec\_value\_get描述见下表：

**表 3-543. 函数 i2c\_pec\_value\_get**

函数名称	i2c_pec_value_get
函数原型	uint32_t i2c_pec_value_get(void);
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}	
-	-
输出参数{out}	
-	-
返回值	
uint32_t	PEC 值

例如：

```
/* get packet error checking value */
```

```
uint32_t pec_value;
```

```
pec_value = i2c_pec_value_get();
```

## 函数 i2c\_smbus\_mode\_enable

函数i2c\_smbus\_mode\_enable描述见下表：

**表 3-544. 函数 i2c\_smbus\_mode\_enable**

函数名称	i2c_smbus_mode_enable
函数原型	void i2c_smbus_mode_enable(void);
功能描述	使能 SMBus 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 SMBus mode */
i2c_smbus_mode_enable();
```

## 函数 i2c\_smbus\_mode\_disable

函数i2c\_smbus\_mode\_disable描述见下表：

**表 3-545. 函数 i2c\_smbus\_mode\_disable**

函数名称	i2c_smbus_mode_disable
函数原型	void i2c_smbus_mode_disable(void);
功能描述	禁能 SMBus 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 SMBus mode */
i2c_smbus_mode_disable();
```



## 函数 i2c\_smbus\_alert\_enable

函数i2c\_smbus\_alert\_enable描述见下表：

**表 3-546. 函数 i2c\_smbus\_alert\_enable**

函数名称	i2c_smbus_alert_enable
函数原型	void i2c_smbus_alert_enable(void);
功能描述	使能 SMBus 报警
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 SMBus Alert */

i2c_smbus_alert_enable();
```

## 函数 i2c\_smbus\_alert\_disable

函数i2c\_smbus\_alert\_disable描述见下表：

**表 3-547. 函数 i2c\_smbus\_alert\_disable**

函数名称	i2c_smbus_alert_disable
函数原型	void i2c_smbus_alert_disable(void);
功能描述	禁能 SMBus 报警
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 SMBus Alert */

i2c_smbus_alert_disable();
```

## 函数 i2c\_smbus\_default\_addr\_enable

函数i2c\_smbus\_default\_addr\_enable描述见下表：

**表 3-548. 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(void);
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}	
-	-
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* enable SMBus device default address */
```

```
i2c_smbus_default_addr_enable();
```

## 函数 i2c\_smbus\_default\_addr\_disable

函数i2c\_smbus\_default\_addr\_disable描述见下表：

**表 3-549. 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(void);
功能描述	禁能 SMBus 设备默认地址
先决条件	-
被调用函数	-
输入参数{in}	
-	-
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* disable SMBus device default address */
```

```
i2c_smbus_default_addr_disable();
```

## 函数 i2c\_smbus\_host\_addr\_enable

函数i2c\_smbus\_host\_addr\_enable描述见下表：

**表 3-550. 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(void);
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}	
-	-
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* enable SMBus Host address */
```

```
i2c_smbus_host_addr_enable();
```

## 函数 i2c\_smbus\_host\_addr\_disable

函数i2c\_smbus\_host\_addr\_disable描述见下表：

**表 3-551. 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(void);
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}	
-	-
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* disable SMBus Host address */
```

```
i2c_smbus_host_addr_disable();
```

### 函数 i2c\_extented\_clock\_timeout\_enable

函数i2c\_extented\_clock\_timeout\_enable描述见下表:

表 3-552. 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(void);
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}	
-	-
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* enable extended clock timeout detection */
```

```
i2c_extented_clock_timeout_enable();
```

### 函数 i2c\_extented\_clock\_timeout\_disable

函数i2c\_extented\_clock\_timeout\_disable描述见下表:

表 3-553. 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(void);
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}	
-	-
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* disable extended clock timeout detection */
```

```
i2c_extented_clock_timeout_disable();
```

## 函数 i2c\_clock\_timeout\_enable

函数i2c\_clock\_timeout\_enable描述见下表：

**表 3-554. 函数 i2c\_clock\_timeout\_enable**

函数名称	i2c_clock_timeout_enable
函数原型	void i2c_clock_timeout_enable(void);
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}	
-	-
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* enable clock timeout detection */
```

```
i2c_clock_timeout_enable();
```

## 函数 i2c\_clock\_timeout\_disable

函数i2c\_clock\_timeout\_disable描述见下表：

**表 3-555. 函数 i2c\_clock\_timeout\_disable**

函数名称	i2c_clock_timeout_disable
函数原型	void i2c_clock_timeout_disable(void);
功能描述	禁能时钟超时检测
先决条件	-
被调用函数	-
输入参数{in}	
-	-
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* disable clock timeout detection */
```

```
i2c_clock_timeout_disable();
```

## 函数 i2c\_bus\_timeout\_b\_config

函数i2c\_bus\_timeout\_b\_config描述见下表：

**表 3-556. 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 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}	
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(0xff);
```

## 函数 i2c\_bus\_timeout\_a\_config

函数i2c\_bus\_timeout\_a\_config描述见下表：

**表 3-557. 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 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}	
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(0xff);
```

## 函数 i2c\_idle\_clock\_timeout\_config

函数i2c\_idle\_clock\_timeout\_config描述见下表：

表 3-558. 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 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}	
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(BUSTOA_DETECT_SCL_LOW);
```

## 函数 i2c\_flag\_get

函数i2c\_flag\_get描述见下表：

表 3-559. 函数 i2c\_flag\_get

函数名称	i2c_flag_get
函数原型	FlagStatus i2c_flag_get(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}	
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

<i>I2C_FLAG_TC</i>	主机模式下传输完成标志
<i>I2C_FLAG_TCR</i>	传输完成重载标志
<i>I2C_FLAG_BERR</i>	总线错误标志
<i>I2C_FLAG_LOSTARB</i>	仲裁丢失标志
<i>I2C_FLAG_OUERR</i>	从机模式下，过载/欠载错误标志
<i>I2C_FLAG_PECERR</i>	PEC 错误标志
<i>I2C_FLAG_TIMEOUT</i>	超时标志
<i>I2C_FLAG_SMBALT</i>	SMBus 报警标志
<i>I2C_FLAG_I2CBSY</i>	忙标志
<i>I2C_FLAG_TR</i>	从机模式下，I2C 作为发送器还是接收器标志
输出参数{out}	
-	-
返回值	
FlagStatus	SET / RESET

例如：

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

```
FlagStatus flag_state = RESET;
```

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

### 函数 i2c\_flag\_clear

函数i2c\_flag\_clear描述见下表：

**表 3-560. 函数 i2c\_flag\_clear**

函数名称	i2c_flag_clear
函数原型	void i2c_flag_clear(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}	
flag	I2C 标志位
<i>I2C_FLAG_ADDSEND</i>	从机模式下，接收到的地址与自身地址匹配
<i>I2C_FLAG_NACK</i>	NACK 标志
<i>I2C_FLAG_STPDET</i>	从机模式下检测到 STOP 信号
<i>I2C_FLAG_BERR</i>	总线错误标志
<i>I2C_FLAG_LOSTARB</i>	仲裁丢失标志
<i>I2C_FLAG_OUERR</i>	从机模式下，过载/欠载错误标志
<i>I2C_FLAG_PECERR</i>	PEC 错误标志
<i>I2C_FLAG_TIMEOUT</i>	超时标志
<i>I2C_FLAG_SMBALT</i>	SMBus 报警标志
输出参数{out}	



-	-
返回值	
-	-

例如：

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

```
i2c_flag_clear(I2C_FLAG_BERR);
```

### 函数 i2c\_interrupt\_enable

函数i2c\_interrupt\_enable描述见下表：

表 3-561. 函数 i2c\_interrupt\_enable

函数名称	i2c_interrupt_enable
函数原型	void i2c_interrupt_enable(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}	
interrupt	I2C 中断
I2C_INT_ERR	错误中断
I2C_INT_TC	发送完成中断
I2C_INT_STPDET	检测到 STOP 中断
I2C_INT_NACK	接收到 NACK 中断
I2C_INT_ADDM	地址匹配中断
I2C_INT_RBNE	接收中断
I2C_INT_TI	发送中断
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* enable I2C transmit interrupt */
```

```
i2c_interrupt_enable(I2C_INT_TI);
```

### 函数 i2c\_interrupt\_disable

函数i2c\_interrupt\_disable描述见下表：

表 3-562. 函数 i2c\_interrupt\_disable

函数名称	i2c_interrupt_disable
函数原型	void i2c_interrupt_disable(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}	
<b>interrupt</b>	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 I2C transmit interrupt */
```

```
i2c_interrupt_disable(I2C_INT_TI);
```

### 函数 i2c\_interrupt\_flag\_get

函数 i2c\_interrupt\_flag\_get 描述见下表：

表 3-563. 函数 i2c\_interrupt\_flag\_get

函数名称	i2c_interrupt_flag_get
函数原型	FlagStatus i2c_interrupt_flag_get(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}	
<b>int_flag</b>	I2C 中断标志位，参考 <a href="#">表 3-502. 枚举类型 i2c_interrupt_flag_enum</a> 。
<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

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_OVERR	从机模式下，过载/欠载错误中断标志
I2C_INT_FLAG_PECERR	PEC 错误中断标志
I2C_INT_FLAG_TIMEOUT	超时中断标志
I2C_INT_FLAG_SMBAL	SMBus 报警中断标志
输出参数{out}	
-	-
返回值	
FlagStatus	SET / RESET

例如：

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

```
FlagStatus flag_state = RESET;
```

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

### 函数 i2c\_interrupt\_flag\_clear

函数i2c\_interrupt\_flag\_clear描述见下表：

表 3-564. 函数 i2c\_interrupt\_flag\_clear

函数名称	i2c_interrupt_flag_clear
函数原型	void i2c_interrupt_flag_clear(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}	
int_flag	I2C 中断标志位，参考 <a href="#">表 3-502. 枚举类型 i2c_interrupt_flag_enum</a> 。
I2C_INT_FLAG_ADDS END	从机模式下，接收到的地址与自身地址匹配中断标志
I2C_INT_FLAG_NACK	NACK 中断标志
I2C_INT_FLAG_STOP ET	从机模式下检测到 STOP 信号中断标志
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_OVERR	从机模式下，过载/欠载错误中断标志

<i>R</i>	
<i>I2C_INT_FLAG_PECERR</i>	PEC 错误中断标志
<i>I2C_INT_FLAG_TIMEOUT</i>	超时中断标志
<i>I2C_INT_FLAG_SMBAL</i>	SMBus 报警中断标志
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* clear a bus error flag */
i2c_interrupt_flag_clear(I2C_INT_FLAG_BERR);
```

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

### 3.20.1. 外设寄存器说明

表 3-565. NVIC 寄存器

寄存器名称	寄存器描述
ISER <sup>(1)</sup>	中断使能寄存器
ICER <sup>(1)</sup>	中断禁能寄存器
ISPR <sup>(1)</sup>	中断挂起寄存器
ICPR <sup>(1)</sup>	中断清除寄存器
IABR <sup>(1)</sup>	中断活动状态寄存器
ITNS <sup>(1)</sup>	中断不安全状态寄存器
IPR <sup>(1)</sup>	中断优先级寄存器
STIR <sup>(1)</sup>	软触发中断寄存器
CPUID <sup>(2)</sup>	CPUID 寄存器
ICSR <sup>(2)</sup>	中断控制及状态寄存器
VTOR <sup>(2)</sup>	向量表偏移量寄存器
AIRCR <sup>(2)</sup>	应用程序中断及复位控制寄存器
SCR <sup>(2)</sup>	系统控制寄存器
CCR <sup>(2)</sup>	配置与控制寄存器
SHPR <sup>(2)</sup>	系统异常优先级寄存器
SHCSR <sup>(2)</sup>	系统异常控制及状态寄存器

1. 参考 core\_cm33.h 文件中定义的结构体类型 NVIC\_Type
2. 参考 core\_cm33.h 文件中定义的结构体类型 SCB\_Type

表 3-566. SysTick 寄存器

寄存器名称	寄存器描述
CTRL <sup>(1)</sup>	Systick控制和状态寄存器
LOAD <sup>(1)</sup>	Systick重载值寄存器
VAL <sup>(1)</sup>	Systick当前值寄存器
CALIB <sup>(1)</sup>	Systick校准寄存器

1. 参考 core\_cm33.h 文件中定义的结构体类型 SysTick\_Type

### 3.20.2. 外设库函数说明

MISC库函数列表如下表所示：

表 3-567. MISC 库函数

库函数名称	库函数描述
nvic_priority_group_set	配置中断优先级组
nvic_irq_enable	使能NVIC的中断
nvic_irq_disable	禁能NVIC的中断
nvic_system_reset	复位MCU
nvic_vector_table_set	设置向量表地址
system_lowpower_set	设置系统低功耗模式状态
system_lowpower_reset	复位系统低功耗模式状态
systick_clksource_set	设置系统定时器时钟源

### 枚举类型 IRQn\_Type

表 3-568. 枚举类型 IRQn\_Type

成员名称	功能描述
WWDGT_IRQn	窗口看门狗中断
LVD0_IRQn	连接到 EXTI 线 16 的 LVD0 中断
LVD1_IRQn	连接到 EXTI 线 17 的 LVD1 中断
FMC_IRQn	FMC 全局中断
RCU_IRQn	RCU 全局中断
EXTI0_IRQn	EXTI 线 0 中断
EXTI1_IRQn	EXTI 线 1 中断
EXTI2_IRQn	EXTI 线 2 中断
EXTI3_IRQn	EXTI 线 3 中断
EXTI4_IRQn	EXTI 线 4 中断
DMA0_Channel0_IRQn	DMA0 通道 0 全局中断
DMA0_Channel1_IRQn	DMA0 通道 1 全局中断
DMA0_Channel2_IRQn	DMA0 通道 2 全局中断

DMA0_Channel3_IRQn	DMA0 通道 3 全局中断
DMA0_Channel4_IRQn	DMA0 通道 4 全局中断
DMA0_Channel5_IRQn	DMA0 通道 5 全局中断
ADC0_IRQn	ADC0 中断
CAN_TX_IRQn	CAN 发送中断
CAN_RX0_IRQn	CAN 接收 0 中断
CAN_RX1_IRQn	CAN 接收 1 中断
CAN_EWMC_IRQn	CAN EWMC 中断
EXTI5_9_IRQn	EXTI 线[9:5]中断
TIMER0_BRK_IRQn	TIMER0 中止中断
TIMER0_UP_IRQn	TIMER0 更新中断
TIMER0_TRG_CMT_IRQn	TIMER0 触发和通道换相中断
TIMER0_Channel_IRQn	TIMER0 捕获比较中断
TIMER1_IRQn	TIMER1 全局中断
TIMER2_IRQn	TIMER2 全局中断
GPTIMER0_IRQn	GPTIMER0 中断
I2C_EV_IRQn	I2C 事件中断
I2C_ER_IRQn	I2C 错误中断
SPI_IRQn	SPI 全局中断
UART0_IRQn	UART0 全局中断
UART1_IRQn	UART1 全局中断
EXTI10_15_IRQn	EXTI 线[15:10]中断
TIMER7_BRK_IRQn	TIMER7 中止中断
TIMER7_UP_IRQn	TIMER7 更新中断
TIMER7_TRG_CMT_IRQn	TIMER7 触发和通道换相中断
TIMER7_Channel_IRQn	TIMER7 捕获比较
TMU_IRQn	TMU 中断
GPTIMER1_IRQn	GPTIMER1 中断
UART2_IRQn	UART2 中断
UART3_IRQn	UART3 中断
CPTIMER0_IRQn	CPTIMER0 全局中断
CPTIMER1_IRQn	CPTIMER1 全局中断
DMA1_Channel0_IRQn	DMA1 通道 0 全局中断
DMA1_Channel1_IRQn	DMA1 通道 1 全局中断
DMA1_Channel2_IRQn	DMA1 通道 2 全局中断
DMA1_Channel3_IRQn	DMA1 通道 3 全局中断
DMA1_Channel4_IRQn	DMA1 通道 4 全局中断
DMA1_Channel5_IRQn	DMA1 通道 5 全局中断
DMAMUX_OVERRUN_IRQn	DMAMUX 溢出中断
CPTIMERW_IRQn	CPTIMERW 全局中断
CFMU_IRQn	CFMU 中断

I2C_WKUP_IRQn	连接到 EXTI 线 23 的 I2C 唤醒中断
FWDGT_IRQn	连接到 EXTI 线 22 的 FWDGT 中断
CMP0_IRQn	CMP0 中断
CMP1_IRQn	CMP1 中断
CMP2_IRQn	CMP2 中断
CMP3_IRQn	CMP3 中断
ADC2_IRQn	ADC2 中断
EVIC_IRQn	EVIC 全局中断
GTOC0_IRQn	GTOC0 中断
GTOC1_IRQn	GTOC1 中断
GTOC2_IRQn	GTOC2 中断
GTOC3_IRQn	GTOC3 中断
CMP0_EXTI_IRQn	连接到 EXTI 线 18 的 CMP0 中断
CMP1_EXTI_IRQn	连接到 EXTI 线 19 的 CMP0 中断
CMP2_EXTI_IRQn	连接到 EXTI 线 20 的 CMP0 中断
CMP3_EXTI_IRQn	连接到 EXTI 线 21 的 CMP0 中断
SRAMC_ECC_IRQ	SRAMC_ECC 中断

### 函数 `nvic_priority_group_set`

函数 `nvic_priority_group_set` 描述见下表：

**表 3-569. 函数 `nvic_priority_group_set`**

函数名称	<code>nvic_priority_group_set</code>
函数原型	<code>void nvic_priority_group_set(uint32_t nvic_prigroup);</code>
功能描述	配置优先级组的位长度
先决条件	-
被调用函数	-
输入参数{in}	
<code>nvic_prigroup</code>	优先级组
<code>NVIC_PRIGROUP_PRE0_SUB4</code>	0位用于抢占优先级，4位用于次优先级
<code>NVIC_PRIGROUP_PRE1_SUB3</code>	1位用于抢占优先级，3位用于次优先级
<code>NVIC_PRIGROUP_PRE2_SUB2</code>	2位用于抢占优先级，2位用于次优先级
<code>NVIC_PRIGROUP_PRE3_SUB1</code>	3位用于抢占优先级，1位用于次优先级
<code>NVIC_PRIGROUP_PRE4_SUB0</code>	4位用于抢占优先级，0位用于次优先级
输出参数{out}	
-	-
返回值	

-	-
---	---

例如:

```
/* priority group configuration , 0 bits for pre-emption priority 4 bits for subpriority */
```

```
nvic_priority_group_set(NVIC_PRIGROUP_PRE0_SUB4);
```

### 函数 nvic\_irq\_enable

函数nvic\_irq\_enable描述见下表:

表 3-570. 函数 nvic\_irq\_enable

函数名称	nvic_irq_enable
函数原型	void nvic_irq_enable(uint8_t nvic_irq, uint8_t nvic_irq_pre_priority, uint8_t nvic_irq_sub_priority);
功能描述	使能NVIC中断
先决条件	-
被调用函数	nvic_priority_group_set
输入参数{in}	
nvic_irq	NVIC中断, 参考枚举类型 <a href="#">表3-568. 枚举类型IRQn_Type</a>
输入参数{in}	
nvic_irq_pre_priority	抢占优先级
nvic_irq_sub_priority	次优先级
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* enable window watchDog timer interrupt, pre-priority is 1, sub-priority is 1 */
```

```
nvic_irq_enable(WWDGT_IRQn, 1, 1);
```

### 函数 nvic\_irq\_disable

函数nvic\_irq\_disable描述见下表:

表 3-571. 函数 nvic\_irq\_disable

函数名称	nvic_irq_disable
函数原型	void nvic_irq_disable(uint8_t nvic_irq);
功能描述	禁能NVIC中断
先决条件	-
被调用函数	NVIC_DisableIRQ
输入参数{in}	
nvic_irq	NVIC中断, 参考枚举类型 <a href="#">表3-568. 枚举类型IRQn_Type</a>



输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* disable window watchDog timer interrupt */
```

```
nvic_irq_disable(WWDGT_IRQn);
```

**表 3-572. 函数 nvic\_system\_reset**

函数名称	nvic_system_reset
函数原型	void nvic_system_reset(void);
功能描述	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 3-573. 函数 nvic\_vector\_table\_set**

函数名称	nvic_vector_table_set
函数原型	void nvic_vector_table_set(uint32_t nvic_vect_tab, uint32_t offset);
功能描述	设置向量表地址
先决条件	-
被调用函数	-
输入参数{in}	
nvic_vect_tab	RAM或者FLASH基地址
NVIC_VECTTAB_RAM	RAM基地址
NVIC_VECTTAB_FLASH	FLASH基地址

输入参数{in}	
offset	向量表偏移量（向量表地址=基地址+偏移量）
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* set vector table address = NVIC_VECTTAB_FLASH + 0x200 */
```

```
nvic_vector_table_set(NVIC_VECTTAB_FLASH, 0x200);
```

### 函数 system\_lowpower\_set

函数system\_lowpower\_set描述见下表：

表 3-574. 函数 system\_lowpower\_set

函数名称	system_lowpower_set
函数原型	void system_lowpower_set(uint8_t lowpower_mode);
功能描述	设置系统低功耗模式状态
先决条件	-
被调用函数	-
输入参数{in}	
lowpower_mode	系统低功耗模式的状态
SCB_LPM_SLEEP_EXIT_ISR	该位为1时，退出ISR时一直处于低功耗模式
SCB_LPM_DEEPSLEEP	该位为1时，系统处于deep sleep模式
SCB_LPM_WAKEUP_BY_ALL_INT	该位为1时，低功耗模式可以被所有中断唤醒（无论中断是否被使能）
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* the system always enter low power mode by exiting from ISR */
```

```
system_lowpower_set(SCB_LPM_SLEEP_EXIT_ISR);
```

### 函数 system\_lowpower\_reset

函数system\_lowpower\_reset描述见下表：

表 3-575. 函数 `system_lowpower_reset`

函数名称	<code>system_lowpower_reset</code>
函数原型	<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 3-576. 函数 `systick_clksource_set`

函数名称	<code>systick_clksource_set</code>
函数原型	<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HCLK时钟
<code>SYSTICK_CLKSOURCE_HCLK_DIV8</code>	SysTick时钟源为HCLK时钟的8分频
输出参数{out}	
-	-
返回值	

-	-
---	---

例如:

```
/* systick clock source is HCLK/8 */
```

```
systick_cksourceset(SYSTICK_CLKSOURCE_HCLK_DIV8);
```

## 3.21. PMU

电源管理单元提供了三种省电模式，包括睡眠模式，深度睡眠模式和待机模式。章节 [3.21.1](#) 描述了 PMU 的寄存器列表，章节 [3.21.2](#) 对 PMU 库函数进行说明。

### 3.21.1. 外设寄存器说明

PMU 寄存器列表如下表所示:

表 3-577. PMU 寄存器

寄存器名称	寄存器描述
PMU_CTL	控制寄存器
PMU_CS	电源控制和状态寄存器
PMU_LVD1CTL	低电压检测器 1 控制寄存器
PMU_LVD2CTL	低电压检测器 2 控制寄存器

### 3.21.2. 外设库函数说明

PMU 库函数列表如下表所示:

表 3-578. PMU 库函数

库函数名称	库函数描述
pmu_deinit	复位外设 PMU
pmu_to_sleepmode	进入睡眠模式
pmu_to_deepsleepmode	进入深度睡眠模式
pmu_to_standbymode	进入待机模式
pmu_deepsleepmode_vcore_output	深度睡眠时 LDO 输出电压选择
pmu_wakeup_pin_enable	WKUP 引脚唤醒使能
pmu_wakeup_pin_disable	WKUP 引脚唤醒失能
pmu_lvd_enable	低电压检测器使能
pmu_lvd_disable	低电压检测器失能
pmu_lvd_interrupt_select	选择低电压检测器的中断产生条件
pmu_lvd_interrupt_type	选择低电压检测器中断类型
pmu_lvd_interrupt_reset_enable	使能低电压检测器产生中断或复位
pmu_lvd_interrupt_reset_disable	失能低电压检测器产生中断或复位
pmu_lvd_output_enable	使能低电压检测器比较结果输出

pmu_lvd_output_disable	失能低电压检测器比较结果输出
pmu_lvd_mode_select	低电压检测模式选择
pmu_lvd_reset_select	复位信号解除选择
pmu_lvd_level_select	检测阈值选择
pmu_lvd_digital_filter_enable	数字噪声滤波器使能
pmu_lvd_digital_filter_disable	数字噪声滤波器失能
pmu_lvd_sample_clock_select	采样时钟选择
pmu_flag_get	获取标志位
pmu_flag_clear	标志位清除

## pmu\_deinit

函数pmu\_deinit描述见下表：

**表 3-579. 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 registers */
pmu_deinit();
```

## pmu\_to\_sleepmode

函数pmu\_to\_sleepmode描述见下表：

**表 3-580. 函数 pmu\_to\_sleepmode**

函数名称	pmu_to_sleepmode
函数原型	void pmu_to_sleepmode(uint8_t sleepmodecmd)
功能描述	进入睡眠模式
先决条件	-
被调用函数	-
输入参数{in}	
sleepmodecmd	进入睡眠模式命令

WFI_CMD	WFI 命令
WFE_CMD	WFE 命令
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* PMU work at sleep mode */
pmu_to_sleepmode(WFI_CMD);
```

### pmu\_to\_deepsleepmode

函数pmu\_to\_deepsleepmode描述见下表：

表 3-581. 函数 pmu\_to\_deepsleepmode

函数名称	pmu_to_deepsleepmode
函数原型	void pmu_to_deepsleepmode(uint8_t deepsleepmodecmd)
功能描述	进入深度睡眠模式
先决条件	-
被调用函数	-
输入参数{in}	
deepsleepmodecmd	进入深度睡眠模式命令
WFI_CMD	WFI 命令
WFE_CMD	WFE 命令
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* PMU work at deepsleep mode */
pmu_to_deepsleepmode(WFI_CMD);
```

### pmu\_to\_standbymode

函数pmu\_to\_standbymode描述见下表：

表 3-582. 函数 pmu\_to\_standbymode

函数名称	pmu_to_standbymode
函数原型	void pmu_to_standbymode(void)
功能描述	进入待机模式
先决条件	-

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

例如：

```
/* PMU work at standby mode */
```

```
pmu_to_standby();
```

### pmu\_deepsleepmode\_vcore\_output

函数pmu\_deepsleepmode\_vcore\_output描述见下表：

表 3-583. 函数 pmu\_deepsleepmode\_vcore\_output

函数名称	pmu_deepsleepmode_vcore_output
函数原型	void pmu_deepsleepmode_vcore_output(uint32_t dslldovs)
功能描述	深度睡眠时 LDO 输出电压选择
先决条件	-
被调用函数	-
输入参数{in}	
<b>dslldovs</b>	深度睡眠时 LDO 输出电压
<i>PMU_DSLDOVS_0</i>	LDO 输出电压 0.9V
<i>PMU_DSLDOVS_1</i>	LDO 输出电压 1.0V
<i>PMU_DSLDOVS_2</i>	LDO 输出电压 1.1V
<i>PMU_DSLDOVS_3</i>	LDO 输出电压 1.2V
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* Deep-sleep mode voltage scaling selection */
```

```
pmu_deepsleepmode_vcore_output(PMU_DSLDOVS_0);
```

### pmu\_wakeup\_pin\_enable

函数pmu\_wakeup\_pin\_enable描述见下表：

表 3-584. 函数 pmu\_wakeup\_pin\_enable

函数名称	pmu_wakeup_pin_enable
------	-----------------------

函数原型	void pmu_wakeup_pin_enable(void)
功能描述	WKUP 引脚唤醒使能
先决条件	-
被调用函数	-
输入参数{in}	
-	-
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* enable PMU wakeup pin */
pmu_wakeup_pin_enable();
```

### pmu\_wakeup\_pin\_disable

函数pmu\_wakeup\_pin\_disable描述见下表：

表 3-585. 函数 pmu\_wakeup\_pin\_disable

函数名称	pmu_wakeup_pin_disable
函数原型	void pmu_wakeup_pin_disable(void)
功能描述	WKUP 引脚唤醒失能
先决条件	-
被调用函数	-
输入参数{in}	
-	-
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* disable PMU wakeup pin */
pmu_wakeup_pin_disable();
```

### pmu\_lvd\_enable

函数pmu\_lvd\_enable描述见下表：

表 3-586. 函数 pmu\_lvd\_enable

函数名称	pmu_lvd_enable
函数原型	void pmu_lvd_enable(uint8_t lvd_select)



功能描述	低电压检测器使能
先决条件	-
被调用函数	-
输入参数{in}	
lvd_select	低电压检测器选择
PMU_LVD_1	选择 LVD1
PMU_LVD_2	选择 LVD2
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* enable LVD1 */
```

```
pmu_lvd_enable(PMU_LVD_1);
```

### pmu\_lvd\_disable

函数pmu\_lvd\_disable描述见下表：

表 3-587. 函数 pmu\_lvd\_disable

函数名称	pmu_lvd_disable
函数原型	void pmu_lvd_disable(uint8_t lvd_select)
功能描述	低电压检测器失能
先决条件	-
被调用函数	-
输入参数{in}	
lvd_select	低电压检测器选择
PMU_LVD_1	选择 LVD1
PMU_LVD_2	选择 LVD2
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* disable LVD1 */
```

```
pmu_lvd_disable(PMU_LVD_1);
```

### pmu\_lvd\_interrupt\_select

函数pmu\_lvd\_interrupt\_select描述见下表：

表 3-588. 函数 pmu\_lvd\_interrupt\_select

函数名称	pmu_lvd_interrupt_select
函数原型	void pmu_lvd_interrupt_select(uint8_t lvd_select, uint32_t select_condition)
功能描述	选择低电压检测器的中断产生条件
先决条件	-
被调用函数	-
输入参数{in}	
lvd_select	低电压检测器选择
PMU_LVD_1	选择 LVD1
PMU_LVD_2	选择 LVD2
输入参数{in}	
select_condition	中断产生触发模式
PMU_LVD_INT_RISING	vcc >= vdetx(上升)
PMU_LVD_INT_FALLING	vcc < vdetx(下降)
PMU_LVD_INT_BOTH	vcc >= vdetx(上升) 或 vcc < vdetx(下降)
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* select the interrupt generation condition for the LVD1 */
```

```
pmu_lvd_interrupt_select(PMU_LVD_1, PMU_LVD_INT_RISING);
```

### pmu\_lvd\_interrupt\_type

函数pmu\_lvd\_interrupt\_type描述见下表:

表 3-589. 函数 pmu\_lvd\_interrupt\_type

函数名称	pmu_lvd_interrupt_type
函数原型	void pmu_lvd_interrupt_type(uint8_t lvd_select, uint32_t interrupt_type)
功能描述	选择低电压检测器中断类型
先决条件	-
被调用函数	-
输入参数{in}	
lvd_select	低电压检测器选择
PMU_LVD_1	选择 LVD1
PMU_LVD_2	选择 LVD2
输入参数{in}	

<b>interrupt_type</b>	选择低电压检测器中断类型
<i>PMU_LVD_INT_MASKABLE</i>	可屏蔽中断
<i>PMU_LVD_INT_NON_MASKABLE</i> 不	不可屏蔽中断
<b>输出参数{out}</b>	
-	-
<b>返回值</b>	
-	-

例如：

```
/* select the interrupt type for the LVD1 */
```

```
pmu_lvd_interrupt_type(PMU_LVD_1, PMU_LVD_INT_MASKABLE);
```

### pmu\_lvd\_interrupt\_reset\_enable

函数pmu\_lvd\_interrupt\_reset\_enable描述见下表：

表 3-590. 函数 pmu\_lvd\_interrupt\_reset\_enable

<b>函数名称</b>	pmu_lvd_interrupt_reset_enable
<b>函数原型</b>	void pmu_lvd_interrupt_reset_enable(uint8_t lvd_select)
<b>功能描述</b>	使能低电压检测器产生中断或复位
<b>先决条件</b>	-
<b>被调用函数</b>	-
<b>输入参数{in}</b>	
<b>lvd_select</b>	低电压检测器选择
<i>PMU_LVD_1</i>	选择 LVD1
<i>PMU_LVD_2</i>	选择 LVD2
<b>输出参数{out}</b>	
-	-
<b>返回值</b>	
-	-

例如：

```
/* enable low voltage detector interrupt or reset */
```

```
pmu_lvd_interrupt_reset_enable(PMU_LVD_1);
```

### pmu\_lvd\_interrupt\_reset\_disable

函数pmu\_lvd\_interrupt\_reset\_disable描述见下表：

表 3-591. 函数 pmu\_lvd\_interrupt\_reset\_disable

<b>函数名称</b>	pmu_lvd_interrupt_reset_disable
-------------	---------------------------------

函数原型	void pmu_lvd_interrupt_reset_disable(uint8_t lvd_select)
功能描述	失能低电压检测器产生中断或复位
先决条件	-
被调用函数	-
输入参数{in}	
lvd_select	低电压检测器选择
PMU_LVD_1	选择 LVD1
PMU_LVD_2	选择 LVD2
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* disable low voltage detector interrupt or reset */
```

```
pmu_lvd_interrupt_reset_disable(PMU_LVD_1);
```

### pmu\_lvd\_output\_enable

函数pmu\_lvd\_output\_enable描述见下表:

表 3-592. 函数 pmu\_lvd\_output\_enable

函数名称	pmu_lvd_output_enable
函数原型	void pmu_lvd_output_enable(uint8_t lvd_select)
功能描述	使能低电压检测器比较结果输出
先决条件	-
被调用函数	-
输入参数{in}	
lvd_select	低电压检测器选择
PMU_LVD_1	选择 LVD1
PMU_LVD_2	选择 LVD2
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* enable LVD1 comparison result output */
```

```
pmu_lvd_output_enable(PMU_LVD_1);
```

**pmu\_lvd\_output\_disable**

函数pmu\_lvd\_output\_disable描述见下表:

**表 3-593. 函数 pmu\_lvd\_output\_disable**

函数名称	pmu_lvd_output_disable
函数原型	void pmu_lvd_output_disable(uint8_t lvd_select)
功能描述	失能低电压检测器比较结果输出
先决条件	-
被调用函数	-
输入参数{in}	
lvd_select	低电压检测器选择
PMU_LVD_1	选择 LVD1
PMU_LVD_2	选择 LVD2
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* disable LVD1 comparison result output */
```

```
pmu_lvd_output_disable(PMU_LVD_1);
```

**pmu\_lvd\_mode\_select**

函数pmu\_lvd\_mode\_select描述见下表:

**表 3-594. 函数 pmu\_lvd\_mode\_select**

函数名称	pmu_lvd_mode_select
函数原型	void pmu_lvd_mode_select(uint8_t lvd_select, uint32_t mode)
功能描述	低电压检测模式选择
先决条件	-
被调用函数	-
输入参数{in}	
lvd_select	低电压检测器选择
PMU_LVD_1	选择 LVD1
PMU_LVD_2	选择 LVD2
输入参数{in}	
mode	低电压检测模式
PMU_LVD_RESET	当电压低于 Vdetx 时, 低电压检测器产生复位
PMU_LVD_INTERRUPT	当电压低于 Vdetx 时, 低电压检测器产生中断
输出参数{out}	

-	-
返回值	
-	-

例如：

```
/* select low voltage detector mode */
```

```
pmu_lvd_mode_select(PMU_LVD_1, PMU_LVD_RESET);
```

### pmu\_lvd\_reset\_select

函数pmu\_lvd\_reset\_select描述见下表：

表 3-595. 函数 pmu\_lvd\_reset\_select

函数名称	pmu_lvd_reset_select
函数原型	void pmu_lvd_reset_select(uint8_t lvd_select, uint32_t mode)
功能描述	复位信号解除选择
先决条件	-
被调用函数	-
输入参数{in}	
lvd_select	低电压检测器选择
PMU_LVD_1	选择 LVD1
PMU_LVD_2	选择 LVD2
输入参数{in}	
mode	复位信号解除选择
PMU_LVD_RESET_ASSERTION	在检测到 VCC > Vdet2 后，复位信号才会在稳定时间（tLVD2）后解除（在 VCC <= Vdet2 时 MCU 一直处于复位状态）。
PMU_LVD_RESET_RISE	LVD2 复位信号产生后，在经过一个稳定时间（tLVD2）后，复位信号将解除（MCU 产生复位一次）
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* select low voltage detector reset negation */
```

```
pmu_lvd_reset_select(PMU_LVD_1, PMU_LVD_RESET_RISE);
```

### pmu\_lvd\_level\_select

函数pmu\_lvd\_level\_select描述见下表：

表 3-596. 函数 pmu\_lvd\_level\_select

函数名称	pmu_lvd_level_select
------	----------------------

函数原型	void pmu_lvd_level_select(uint8_t lvd_select, uint32_t level)
功能描述	检测阈值选择
先决条件	-
被调用函数	-
输入参数{in}	
lvd_select	低电压检测器选择
PMU_LVD_1	选择 LVD1
PMU_LVD_2	选择 LVD2
输入参数{in}	
level	检测阈值选择
PMU_LVDT_0	检测阈值为 4.62v / 4.54v
PMU_LVDT_1	检测阈值为 4.55v / 4.47v
PMU_LVDT_2	检测阈值为 4.40v / 4.32v
PMU_LVDT_3	检测阈值为 3.03v / 2.95v
PMU_LVDT_4	检测阈值为 2.96v / 2.88v
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* select low voltage detector threshold */
```

```
pmu_lvd_level_select(PMU_LVD_1, PMU_LVDT_0);
```

### pmu\_lvd\_digital\_filter\_enable

函数pmu\_lvd\_digital\_filter\_enable描述见下表：

表 3-597. 函数 pmu\_lvd\_digital\_filter\_enable

函数名称	pmu_lvd_digital_filter_enable
函数原型	void pmu_lvd_digital_filter_enable(uint8_t lvd_select)
功能描述	数字噪声滤波器使能
先决条件	-
被调用函数	-
输入参数{in}	
lvd_select	低电压检测器选择
PMU_LVD_1	选择 LVD1
PMU_LVD_2	选择 LVD2
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* enable LVD1 digital filter */
```

```
pmu_lvd_digital_filter_enable(PMU_LVD_1);
```

### pmu\_lvd\_digital\_filter\_disable

函数pmu\_lvd\_digital\_filter\_disable描述见下表：

表 3-598. 函数 pmu\_lvd\_digital\_filter\_disable

函数名称	pmu_lvd_digital_filter_disable
函数原型	void pmu_lvd_digital_filter_disable(uint8_t lvd_select)
功能描述	数字噪声滤波器失能
先决条件	-
被调用函数	-
输入参数{in}	
lvd_select	低电压检测器选择
PMU_LVD_1	选择 LVD1
PMU_LVD_2	选择 LVD2
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* disable LVD1 digital filter */
```

```
pmu_lvd_digital_filter_disable(PMU_LVD_1);
```

### pmu\_lvd\_sample\_clock\_select

函数pmu\_lvd\_sample\_clock\_select描述见下表：

表 3-599. 函数 pmu\_lvd\_sample\_clock\_select

函数名称	pmu_lvd_sample_clock_select
函数原型	void pmu_lvd_sample_clock_select(uint8_t lvd_select, uint32_t clock)
功能描述	采样时钟选择
先决条件	-
被调用函数	-
输入参数{in}	
lvd_select	低电压检测器选择
PMU_LVD_1	选择 LVD1
PMU_LVD_2	选择 LVD2
输入参数{in}	



<b>clock</b>	采样时钟
<i>PMU_LVDSAMP_DIV2</i>	PCLK1 频率的 1 / 2
<i>PMU_LVDSAMP_DIV4</i>	PCLK1 频率的 1 / 4
<i>PMU_LVDSAMP_DIV8</i>	PCLK1 频率的 1 / 8
<i>PMU_LVDSAMP_DIV16</i>	PCLK1 频率的 1 / 16
<b>输出参数{out}</b>	
-	-
<b>返回值</b>	
-	-

例如：

```
/* select low voltage detector sampling clock */
```

```
pmu_lvd_sample_clock_select(PMU_LVD_1, PMU_LVDSAMP_DIV2);
```

### pmu\_flag\_get

函数pmu\_flag\_get描述见下表：

表 3-600. 函数 pmu\_flag\_get

<b>函数名称</b>	pmu_flag_get
<b>函数原型</b>	FlagStatus pmu_flag_get(uint32_t flag)
<b>功能描述</b>	获取标志位
<b>先决条件</b>	-
<b>被调用函数</b>	-
<b>输入参数{in}</b>	
<b>flag</b>	标志
<i>PMU_FLAG_WAKEUP</i>	唤醒标志
<i>PMU_FLAG_STANDBY</i>	待机标志
<i>PMU_FLAG_LVD1DET</i> <i>F</i>	低电压检测器 1 检测标志
<i>PMU_FLAG_LVD1OSF</i>	低电压检测器 1 输出信号标志
<i>PMU_FLAG_LVD2DET</i> <i>F</i>	低电压检测器 2 检测标志
<i>PMU_FLAG_LVD2OSF</i>	低电压检测器 2 输出信号标志
<b>输出参数{out}</b>	
-	-
<b>返回值</b>	
<b>FlagStatus</b>	SET or RESET

例如：

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

```
pmu_flag_get(PMU_FLAG_WAKEUP);
```

### pmu\_flag\_clear

函数pmu\_flag\_clear描述见下表：

表 3-601. 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	标志位清除
先决条件	-
被调用函数	-
输入参数{in}	
flag	标志
PMU_FLAG_RESET_WAKEUP	唤醒标志复位
PMU_FLAG_RESET_STANDBY	待机标志复位
输出参数{out}	
-	-
返回值	
-	-

例如：

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

```
pmu_flag_clear(PMU_FLAG_RESET_WAKEUP);
```

## 3.22. POC

端口输出控制器（POC）可以禁能TIMER和GPTIMER通道引脚输出，并适用于各种场合。被禁能的引脚可以选择高阻（Hi-Z）输出或者GPIO模块控制输出。章节[3.22.1](#)描述了POC的寄存器列表，章节[3.22.2](#)对POC库函数进行说明。

### 3.22.1. 外设寄存器说明

POC寄存器列表如下表所示：

表 3-602. POC 寄存器

寄存器名称	寄存器描述
POC_STAT0	POC 状态寄存器 0
POC_INnDCFG	POC 输入 n 检测配置寄存器
POC_CTL0	POC 控制寄存器 0

POC_SWDRG	POC 软件禁能请求生成寄存器
POC_CDCFG0	POC 互补检测配置寄存器 0
POC_CDCFG1	POC 互补检测配置寄存器 1
POC_ODMODE0	POC 输出禁能模式寄存器 0
POC_ODMODE1	POC 输出禁能模式寄存器 1
POC_REQSEL0	POC 请求选择寄存器 0
POC_REQSEL1	POC 请求选择寄存器 1
POC_STAT1	POC 状态寄存器 1
POC_CTL1	POC 控制寄存器 1
POC_EXTCTL0	POC 扩展控制寄存器 0
POC_EXTCTL1	POC 扩展控制寄存器 1
POC_INnDMK	POC 输入 n 检测屏蔽寄存器
POC_CMPnDMK	POC 比较器 n 检测屏蔽寄存器

### 3.22.2. 外设库函数说明

GTOC库函数列表如下表所示：

**表 3-603. GTOC 库函数**

库函数名称	库函数描述
poc_deinit	复位 POC
poc_input_detection_config	配置 POC_INn 输入检测
poc_input_dreq_status_config	配置 POC_INn 禁能请求状态
poc_input_polarity_config	配置 POC_INn 输入极性
poc_sys_fault_dreq_status_config	配置系统故障禁能请求状态
poc_software_request_generate	软件生成禁能请求
poc_software_request_stop	软件停止禁能请求
poc_complementary_detection_struct_para_init	初始化 POC 互补检测结构体
poc_timer0_complementary_detection_config	配置 TIMER0 互补通道检测
poc_timer7_complementary_detection_config	配置 TIMER7 互补通道检测
poc_timer0_output_disable_mode_select	选择 TIMER0 的输出禁能模式
poc_timer7_output_disable_mode_select	选择 TIMER7 的输出禁能模式
poc_timer1_output_disable_mode_select	选择 TIMER1 的输出禁能模式
poc_timer2_output_disable_mode_select	选择 TIMER2 的输出禁能模式
poc_gptimer0_output_disable_mode_select	选择 GPTIMER0 的输出禁能模式
poc_gptimer1_output_disable_mode_select	选择 GPTIMER1 的输出禁能模式
poc_request_struct_para_init	初始化 POC 请求结构体
poc_request_select	请求选择目标定时器
poc_cmp_dreq_status_config	配置比较器禁能请求状态
poc_cmp_dreq_status_extended_config	配置比较器扩展禁能请求状态
poc_input_detection_mask	屏蔽 POC_INn 输入检测
poc_cmp_output_detection_mask	屏蔽比较器输出检测

poc_flag_get	获取 POC 标志位
poc_flag_clear	清除 POC 标志位
poc_interrupt_enable	使能 POC 中断
poc_interrupt_disable	禁能 POC 中断
poc_interrupt_flag_get	获取 POC 中断标志位
poc_interrupt_flag_clear	清除 POC 中断标志位

### 结构体类型 poc\_request\_struct

表 3-604. 结构体类型 poc\_request\_struct

成员名称	功能描述
req_pocin0	POC_IN0 引脚输入检测生成的请求
req_pocin1	POC_IN1 引脚输入检测生成的请求
req_pocin2	POC_IN2 引脚输入检测生成的请求
req_pocin3	POC_IN3 引脚输入检测生成的请求
req_pocin4	POC_IN4 引脚输入检测生成的请求
req_pocin5	POC_IN5 引脚输入检测生成的请求
req_comparator	比较器输出检测生成的请求

### 结构体类型 poc\_complementary\_detection\_struct

表 3-605. 结构体类型 poc\_complementary\_detection\_struct

成员名称	功能描述
ccdreqstatus	同时导通禁能请求状态
polarityselen	定时器极性选择使能
mch2polarity	多模式通道 2 有效极性选择
ch2polarity	通道 2 有效极性选择
mch1polarity	多模式通道 1 有效极性选择
ch1polarity	通道 1 有效极性选择
mch0polarity	多模式通道 0 有效极性选择
ch0polarity	通道 0 有效极性选择

### 枚举类型 poc\_pin\_enum

表 3-606. 枚举类型 poc\_pin\_enum

成员名称	功能描述
POC_IN0	POC_IN0 输入引脚
POC_IN1	POC_IN1 输入引脚
POC_IN2	POC_IN2 输入引脚
POC_IN3	POC_IN3 输入引脚
POC_IN4	POC_IN4 输入引脚
POC_IN5	POC_IN5 输入引脚

### 枚举类型 `cmp_output_enum`

表 3-607. 枚举类型 `cmp_output_enum`

成员名称	功能描述
CMP0_OUT	比较器 0 输出
CMP1_OUT	比较器 1 输出
CMP2_OUT	比较器 2 输出
CMP3_OUT	比较器 3 输出

### 枚举类型 `target_timer_enum`

表 3-608. 枚举类型 `target_timer_enum`

成员名称	功能描述
TARGET_TIMER0	TIMER0 为目标定时器
TARGET_TIMER7	TIMER7 为目标定时器
TARGET_TIMER1	TIMER1 为目标定时器
TARGET_TIMER2	TIMER2 为目标定时器
TARGET_GPTIMER0	GPTIMER0 为目标定时器
TARGET_GPTIMER1	GPTIMER1 为目标定时器

### 枚举类型 `poc_interrupt_enum`

表 3-609. 枚举类型 `poc_interrupt_enum`

成员名称	功能描述
POC_INT_IN0	POC_IN0 输入检测中断
POC_INT_IN1	POC_IN1 输入检测中断
POC_INT_IN2	POC_IN2 输入检测中断
POC_INT_IN3	POC_IN3 输入检测中断
POC_INT_IN4	POC_IN4 输入检测中断
POC_INT_IN5	POC_IN5 输入检测中断
POC_INT_TIMER0CC	TIMER0 同时导通检测中断
POC_INT_TIMER7CC	TIMER7 同时导通检测中断

### 函数 `poc_deinit`

函数 `poc_deinit` 描述见下表：

表 3-610. 函数 `poc_deinit`

函数名称	<code>poc_deinit</code>
函数原型	<code>void poc_deinit(void)</code>
功能描述	复位 POC
先决条件	<code>rcu_periph_reset_enable/ rcu_periph_reset_disable</code>
被调用函数	-
输出参数{out}	

-	-
返回值	
-	-

例如：

```
/* deinitialize POC */
```

```
dac_deinit();
```

### 函数 `poc_input_detection_config`

函数 `poc_input_detection_config` 描述见下表：

表 3-611. 函数 `poc_input_detection_config`

函数名称	<code>poc_input_detection_config</code>
函数原型	<code>void poc_input_detection_config(poc_pin_enum poc_pin, uint32_t detection_mode, uint32_t sampling_number)</code>
功能描述	配置 POC_INn 输入检测
先决条件	-
被调用函数	-
输入参数{in}	
<code>poc_pin</code>	参考 <a href="#">表 3-606. 枚举类型 <code>poc_pin_enum</code></a>
输入参数{in}	
<code>detection_mode</code>	输入检测模式
<code>POC_EDGE_DETECTION_DIV1</code>	采样频率为 $f_{HCLK}$ ，在下降沿（INPL=0）或上升沿（INPL=1）生成请求
<code>POC_LEVEL_DETECTION_DIV8</code>	采样频率为 $f_{HCLK}/8$ ，当低电平（INPL=0）或高电平（INPL=1）被连续采样一定次数后产生一个请求
<code>POC_LEVEL_DETECTION_DIV16</code>	采样频率为 $f_{HCLK}/16$ ，当低电平（INPL=0）或高电平（INPL=1）被连续采样一定次数后产生一个请求
<code>POC_LEVEL_DETECTION_DIV128</code>	采样频率为 $f_{HCLK}/128$ ，当低电平（INPL=0）或高电平（INPL=1）被连续采样一定次数后产生一个请求
<code>POC_LEVEL_DETECTION_DIV1</code>	采样频率为 $f_{HCLK}$ ，当低电平（INPL=0）或高电平（INPL=1）被连续采样一定次数后产生一个请求
<code>POC_LEVEL_DETECTION_DIV2</code>	采样频率为 $f_{HCLK}/2$ ，当低电平（INPL=0）或高电平（INPL=1）被连续采样一定次数后产生一个请求
<code>POC_LEVEL_DETECTION_DIV4</code>	采样频率为 $f_{HCLK}/4$ ，当低电平（INPL=0）或高电平（INPL=1）被连续采样一定次数后产生一个请求
<code>POC_LEVEL_DETECTION_DIV256</code>	采样频率为 $f_{HCLK}/256$ ，当低电平（INPL=0）或高电平（INPL=1）被连续采样一定次数后产生一个请求
<code>POC_LEVEL_DETECTION_DIV512</code>	采样频率为 $f_{HCLK}/512$ ，当低电平（INPL=0）或高电平（INPL=1）被连续采样一定次数后产生一个请求
<code>POC_LEVEL_DETECTION_DIV32</code>	采样频率为 $f_{HCLK}/32$ ，当低电平（INPL=0）或高电平（INPL=1）被连续采样一

<i>ECTION_DIV32</i>	定次数后产生一个请求
<i>POC_LEVEL_DETECTION_DIV64</i>	采样频率为 $f_{HCLK}/64$ ，当低电平（INPL=0）或高电平（INPL=1）被连续采样一定次数后产生一个请求
<i>POC_EDGE_DETECTION_DIV2</i>	采样频率为 $f_{HCLK}/2$ ，在下降沿（INPL=0）或上升沿（INPL=1）生成请求
<i>POC_EDGE_DETECTION_DIV4</i>	采样频率为 $f_{HCLK}/4$ ，在下降沿（INPL=0）或上升沿（INPL=1）生成请求
输入参数{in}	
<b>sampling_number</b>	有效采样数
<i>POC_SAMPLING_NUM_16_TIMES</i>	采样数 16 次
<i>POC_SAMPLING_NUM_4_TIMES</i>	采样数 4 次
<i>POC_SAMPLING_NUM_8_TIMES</i>	采样数 8 次
<i>POC_SAMPLING_NUM_9_TIMES</i>	采样数 9 次
<i>POC_SAMPLING_NUM_10_TIMES</i>	采样数 10 次
<i>POC_SAMPLING_NUM_11_TIMES</i>	采样数 11 次
<i>POC_SAMPLING_NUM_12_TIMES</i>	采样数 12 次
<i>POC_SAMPLING_NUM_13_TIMES</i>	采样数 13 次
<i>POC_SAMPLING_NUM_14_TIMES</i>	采样数 14 次
<i>POC_SAMPLING_NUM_15_TIMES</i>	采样数 15 次
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* configure POC_IN0 input detection */
```

```
poc_input_detection_config(POC_IN0, POC_LEVEL_DETECTION_DIV8,
POC_SAMPLING_NUM_16_TIMES);
```

### 函数 **poc\_input\_dreq\_status\_config**

函数 `poc_input_dreq_status_config` 描述见下表：

表 3-612. 函数 `poc_input_dreq_status_config`

函数名称	<code>poc_input_dreq_status_config</code>
函数原型	<code>void poc_input_dreq_status_config(poc_pin_enum poc_pin, uint32_t indreq_status)</code>
功能描述	配置 POC_INn 禁能请求状态
先决条件	-
被调用函数	-
输入参数{in}	
<code>poc_pin</code>	参考 <a href="#">表 3-606. 枚举类型 <code>poc_pin_enum</code></a>
输入参数{in}	
<code>indreq_status</code>	POC_INn 禁能请求状态
<code>POC_INn_DREQ_DISABLE</code>	禁能
<code>POC_INn_DREQ_ENABLE</code>	使能
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* configure POC_IN0 disabling request status */
```

```
poc_input_dreq_status_config(POC_IN0, POC_INn_DREQ_ENABLE);
```

### 函数 `poc_input_polarity_config`

函数 `poc_input_polarity_config` 描述见下表：

表 3-613. 函数 `poc_input_polarity_config`

函数名称	<code>poc_input_polarity_config</code>
函数原型	<code>void poc_input_polarity_config(poc_pin_enum poc_pin, uint32_t input_polarity)</code>
功能描述	配置 POC_INn 输入极性
先决条件	-
被调用函数	-
输入参数{in}	
<code>poc_pin</code>	参考 <a href="#">表 3-606. 枚举类型 <code>poc_pin_enum</code></a>
输入参数{in}	
<code>input_polarity</code>	POC_INn 输入极性
<code>POC_INPUT_POLARITY_NONINVERTED</code>	POC_INn 输入不反相
<code>POC_INPUT_POLARITY_INVERTED</code>	POC_INn 输入反相



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

例如：

```
/* configure POC_IN0 input polarity */
```

```
poc_input_polarity_config(POC_IN0, POC_INPUT_POLARITY_INVERTED);
```

### 函数 **poc\_sys\_fault\_dreq\_status\_config**

函数 **poc\_sys\_fault\_dreq\_status\_config** 描述见下表：

**表 3-614. 函数 **poc\_sys\_fault\_dreq\_status\_config****

函数名称	poc_sys_fault_dreq_status_config
函数原型	void poc_sys_fault_dreq_status_config(uint32_t hxtalsdreq_status, uint32_t lockupdreq_status)
功能描述	配置系统故障禁能请求状态
先决条件	-
被调用函数	-
输入参数{in}	
<b>hxtalsdreq_status</b>	HXTAL 卡住禁能请求状态
<i>HXTALS_DREQ_DISABLE</i>	禁能
<i>HXTALS_DREQ_ENABLE</i>	使能
输入参数{in}	
<b>lockupdreq_status</b>	CPU 锁定禁能请求状态
<i>LOCKUP_DREQ_DISABLE</i>	禁能
<i>LOCKUP_DREQ_ENABLE</i>	使能
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* configure system fault disabling request status */
```

```
poc_sys_fault_dreq_status_config(HXTALS_DREQ_ENABLE, LOCKUP_DREQ_ENABLE);
```

## 函数 `poc_software_request_generate`

函数 `poc_software_request_generate` 描述见下表：

**表 3-615. 函数 `poc_software_request_generate`**

函数名称	<code>poc_software_request_generate</code>
函数原型	<code>void poc_software_request_generate(target_timer_enum target_timer)</code>
功能描述	软件生成禁能请求
先决条件	-
被调用函数	-
输入参数{in}	
<code>target_timer</code>	参考 <a href="#">表 3-608. 枚举类型 <code>target_timer_enum</code></a>
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* generate software disabling request for TIMER0 */
poc_software_request_generate(TARGET_TIMER0);
```

## 函数 `poc_software_request_stop`

函数 `poc_software_request_stop` 描述见下表：

**表 3-616. 函数 `poc_software_request_stop`**

函数名称	<code>poc_software_request_stop</code>
函数原型	<code>void poc_software_request_stop (target_timer_enum target_timer)</code>
功能描述	软件停止禁能请求
先决条件	-
被调用函数	-
输入参数{in}	
<code>target_timer</code>	参考 <a href="#">表 3-608. 枚举类型 <code>target_timer_enum</code></a>
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* stop software disabling request for TIMER0 */
poc_software_request_stop(TARGET_TIMER0);
```

## 函数 `poc_complementary_detection_struct_para_init`

函数 `poc_complementary_detection_struct_para_init` 描述见下表：

**表 3-617. 函数 `poc_complementary_detection_struct_para_init`**

函数名称	<code>poc_complementary_detection_struct_para_init</code>
函数原型	<pre>void poc_complementary_detection_struct_para_init(poc_complementary_detection_struct *detpara)</pre>
功能描述	初始化 POC 互补检测结构体
先决条件	-
被调用函数	-
输入参数{in}	
detpara	POC 互补检测结构体参数，结构体成员参考 <a href="#">表 3-605. 结构体类型 <code>poc_complementary_detection_struct</code></a>
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* initialize POC complementary detection struct with a default value */
```

```
poc_complementary_detection_struct para;
```

```
poc_complementary_detection_struct_para_init(&para);
```

## 函数 `poc_timer0_complementary_detection_config`

函数 `poc_timer0_complementary_detection_config` 描述见下表：

**表 3-618. 函数 `poc_timer0_complementary_detection_config`**

函数名称	<code>poc_timer0_complementary_detection_config</code>
函数原型	<pre>void poc_timer0_complementary_detection_config(poc_complementary_detection_struct *detpara)</pre>
功能描述	配置 TIMER0 互补通道检测
先决条件	-
被调用函数	-
输入参数{in}	
detpara	POC 互补检测结构体参数，结构体成员参考 <a href="#">表 3-605. 结构体类型 <code>poc_complementary_detection_struct</code></a>
输出参数{out}	
-	-
返回值	

-	-
---	---

例如：

```
/* configure TIMER0 complementary channel detection */
```

```
poc_complementary_detection_struct para;
```

```
poc_timer0_complementary_detection_config(&para);
```

### 函数 `poc_timer7_complementary_detection_config`

函数 `poc_timer7_complementary_detection_config` 描述见下表：

**表 3-619. 函数 `poc_timer7_complementary_detection_config`**

函数名称	<code>poc_timer7_complementary_detection_config</code>
函数原型	void <code>poc_timer7_complementary_detection_config(poc_complementary_detection_struct *detpara)</code>
功能描述	配置 TIMER7 互补通道检测
先决条件	-
被调用函数	-
输入参数{in}	
detpara	POC 互补检测结构体参数，结构体成员参考 <a href="#">表 3-605. 结构体类型 <code>poc_complementary_detection_struct</code></a>
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* configure TIMER7 complementary channel detection */
```

```
poc_complementary_detection_struct para;
```

```
poc_timer7_complementary_detection_config(&para);
```

### 函数 `poc_timer0_output_disable_mode_select`

函数 `poc_timer0_output_disable_mode_select` 描述见下表：

**表 3-620. 函数 `poc_timer0_output_disable_mode_select`**

函数名称	<code>poc_timer0_output_disable_mode_select</code>
函数原型	void <code>poc_timer0_output_disable_mode_select(uint8_t ch0mch0_pin_mode, uint8_t ch1mch1_pin_mode, uint8_t ch2mch2_pin_mode, uint8_t ch3mch3_pin_mode)</code>
功能描述	选择 TIMER0 的输出禁能模式

先决条件	-
被调用函数	-
输入参数{in}	
ch0mch0_pin_mode	引脚输出模式
POC_TIMER_CHANNEL_OUT	定时器通道输出
POC_HIGH_IMPEDANCE_OUT	高阻输出
POC_GPIO_OUT	GPIO 输出
输入参数{in}	
ch1mch1_pin_mode	引脚输出模式
POC_TIMER_CHANNEL_OUT	定时器通道输出
POC_HIGH_IMPEDANCE_OUT	高阻输出
POC_GPIO_OUT	GPIO 输出
输入参数{in}	
ch2mch2_pin_mode	引脚输出模式
POC_TIMER_CHANNEL_OUT	定时器通道输出
POC_HIGH_IMPEDANCE_OUT	高阻输出
POC_GPIO_OUT	GPIO 输出
输入参数{in}	
ch3mch3_pin_mode	引脚输出模式
POC_TIMER_CHANNEL_OUT	定时器通道输出
POC_HIGH_IMPEDANCE_OUT	高阻输出
POC_GPIO_OUT	GPIO 输出
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* select output disable mode for TIMER0 */
```

```
poc_timer0_output_disable_mode_select(POC_HIGH_IMPEDANCE_OUT,
```

POC\_HIGH\_IMPEDANCE\_OUT,  
POC\_HIGH\_IMPEDANCE\_OUT);

POC\_HIGH\_IMPEDANCE\_OUT,

### 函数 `poc_timer7_output_disable_mode_select`

函数 `poc_timer7_output_disable_mode_select` 描述见下表:

**表 3-621. 函数 `poc_timer7_output_disable_mode_select`**

函数名称	<code>poc_timer7_output_disable_mode_select</code>
函数原型	<code>void poc_timer7_output_disable_mode_select(uint8_t ch0mch0_pin_mode, uint8_t ch1mch1_pin_mode, uint8_t ch2mch2_pin_mode, uint8_t ch3mch3_pin_mode)</code>
功能描述	选择 TIMER7 的输出禁能模式
先决条件	-
被调用函数	-
输入参数{in}	
<code>ch0mch0_pin_mode</code>	引脚输出模式
<code>POC_TIMER_CHANNEL_OUT</code>	定时器通道输出
<code>POC_HIGH_IMPEDANCE_OUT</code>	高阻输出
<code>POC_GPIO_OUT</code>	GPIO 输出
输入参数{in}	
<code>ch1mch1_pin_mode</code>	引脚输出模式
<code>POC_TIMER_CHANNEL_OUT</code>	定时器通道输出
<code>POC_HIGH_IMPEDANCE_OUT</code>	高阻输出
<code>POC_GPIO_OUT</code>	GPIO 输出
输入参数{in}	
<code>ch2mch2_pin_mode</code>	引脚输出模式
<code>POC_TIMER_CHANNEL_OUT</code>	定时器通道输出
<code>POC_HIGH_IMPEDANCE_OUT</code>	高阻输出
<code>POC_GPIO_OUT</code>	GPIO 输出
输入参数{in}	
<code>ch3mch3_pin_mode</code>	引脚输出模式
<code>POC_TIMER_CHANNEL_OUT</code>	定时器通道输出

NNEL_OUT	
POC_HIGH_IMPEDANCE_OUT	高阻输出
POC_GPIO_OUT	GPIO 输出
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* select output disable mode for TIMER7 */
```

```
poc_timer7_output_disable_mode_select(POC_HIGH_IMPEDANCE_OUT,
POC_HIGH_IMPEDANCE_OUT,
POC_HIGH_IMPEDANCE_OUT,
POC_HIGH_IMPEDANCE_OUT);
```

### 函数 poc\_timer1\_output\_disable\_mode\_select

函数poc\_timer1\_output\_disable\_mode\_select描述见下表:

表 3-622. 函数 poc\_timer1\_output\_disable\_mode\_select

函数名称	poc_timer1_output_disable_mode_select
函数原型	void poc_timer1_output_disable_mode_select(uint8_t ch0_pin_mode, uint8_t ch1_pin_mode, uint8_t ch2_pin_mode, uint8_t ch3_pin_mode)
功能描述	选择 TIMER1 的输出禁能模式
先决条件	-
被调用函数	-
输入参数{in}	
ch0_pin_mode	引脚输出模式
POC_TIMER_CHA NNEL_OUT	定时器通道输出
POC_HIGH_IMPEDANCE_OUT	高阻输出
POC_GPIO_OUT	GPIO 输出
输入参数{in}	
ch1_pin_mode	引脚输出模式
POC_TIMER_CHA NNEL_OUT	定时器通道输出
POC_HIGH_IMPEDANCE_OUT	高阻输出
POC_GPIO_OUT	GPIO 输出
输入参数{in}	
ch2_pin_mode	引脚输出模式

POC_TIMER_CHA NNEL_OUT	定时器通道输出
POC_HIGH_IMPED ANCE_OUT	高阻输出
POC_GPIO_OUT	GPIO 输出
输入参数{in}	
ch3_pin_mode	引脚输出模式
POC_TIMER_CHA NNEL_OUT	定时器通道输出
POC_HIGH_IMPED ANCE_OUT	高阻输出
POC_GPIO_OUT	GPIO 输出
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* select output disable mode for TIMER1 */
```

```
poc_timer1_output_disable_mode_select(POC_HIGH_IMPEDANCE_OUT,  
POC_HIGH_IMPEDANCE_OUT, POC_HIGH_IMPEDANCE_OUT,  
POC_HIGH_IMPEDANCE_OUT);
```

### 函数 poc\_timer2\_output\_disable\_mode\_select

函数poc\_timer2\_output\_disable\_mode\_select描述见下表：

表 3-623. 函数 poc\_timer2\_output\_disable\_mode\_select

函数名称	poc_timer2_output_disable_mode_select
函数原型	void poc_timer2_output_disable_mode_select(uint8_t ch0_pin_mode, uint8_t ch1_pin_mode, uint8_t ch2_pin_mode, uint8_t ch3_pin_mode)
功能描述	选择 TIMER2 的输出禁能模式
先决条件	-
被调用函数	-
输入参数{in}	
ch0_pin_mode	引脚输出模式
POC_TIMER_CHA NNEL_OUT	定时器通道输出
POC_HIGH_IMPED ANCE_OUT	高阻输出
POC_GPIO_OUT	GPIO 输出
输入参数{in}	



<b>ch1_pin_mode</b>	引脚输出模式
<i>POC_TIMER_CHA</i> <i>NNEL_OUT</i>	定时器通道输出
<i>POC_HIGH_IMPED</i> <i>ANCE_OUT</i>	高阻输出
<i>POC_GPIO_OUT</i>	GPIO 输出
输入参数{in}	
<b>ch2_pin_mode</b>	引脚输出模式
<i>POC_TIMER_CHA</i> <i>NNEL_OUT</i>	定时器通道输出
<i>POC_HIGH_IMPED</i> <i>ANCE_OUT</i>	高阻输出
<i>POC_GPIO_OUT</i>	GPIO 输出
输入参数{in}	
<b>ch3_pin_mode</b>	引脚输出模式
<i>POC_TIMER_CHA</i> <i>NNEL_OUT</i>	定时器通道输出
<i>POC_HIGH_IMPED</i> <i>ANCE_OUT</i>	高阻输出
<i>POC_GPIO_OUT</i>	GPIO 输出
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* select output disable mode for TIMER2 */
```

```
poc_timer2_output_disable_mode_select(POC_HIGH_IMPEDANCE_OUT,  
POC_HIGH_IMPEDANCE_OUT, POC_HIGH_IMPEDANCE_OUT,  
POC_HIGH_IMPEDANCE_OUT);
```

### 函数 **poc\_gptimer0\_output\_disable\_mode\_select**

函数 **poc\_gptimer0\_output\_disable\_mode\_select** 描述见下表：

**表 3-624. 函数 **poc\_gptimer0\_output\_disable\_mode\_select****

函数名称	<b>poc_gptimer0_output_disable_mode_select</b>
函数原型	<code>void poc_gptimer0_output_disable_mode_select(uint8_t ch0_pin_mode, uint8_t ch1_pin_mode)</code>
功能描述	选择 GPTIMER0 的输出禁能模式
先决条件	-
被调用函数	-

输入参数{in}	
ch0_pin_mode	引脚输出模式
POC_TIMER_CHA NNEL_OUT	定时器通道输出
POC_HIGH_IMPED ANCE_OUT	高阻输出
POC_GPIO_OUT	GPIO 输出
输入参数{in}	
ch1_pin_mode	引脚输出模式
POC_TIMER_CHA NNEL_OUT	定时器通道输出
POC_HIGH_IMPED ANCE_OUT	高阻输出
POC_GPIO_OUT	GPIO 输出
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* select output disable mode for GPTIMER0 */
```

```
poc_gptimer0_output_disable_mode_select(POC_HIGH_IMPEDANCE_OUT,  
POC_HIGH_IMPEDANCE_OUT);
```

### 函数 poc\_gptimer1\_output\_disable\_mode\_select

函数poc\_gptimer1\_output\_disable\_mode\_select描述见下表：

**表 3-625. 函数 poc\_gptimer1\_output\_disable\_mode\_select**

函数名称	poc_gptimer1_output_disable_mode_select
函数原型	void poc_gptimer1_output_disable_mode_select(uint8_t ch0_pin_mode, uint8_t ch1_pin_mode)
功能描述	选择 GPTIMER1 的输出禁能模式
先决条件	-
被调用函数	-
输入参数{in}	
ch0_pin_mode	引脚输出模式
POC_TIMER_CHA NNEL_OUT	定时器通道输出
POC_HIGH_IMPED ANCE_OUT	高阻输出
POC_GPIO_OUT	GPIO 输出

输入参数{in}	
ch1_pin_mode	引脚输出模式
POC_TIMER_CHA NNEL_OUT	定时器通道输出
POC_HIGH_IMPED ANCE_OUT	高阻输出
POC_GPIO_OUT	GPIO 输出
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* select output disable mode for GPTIMER1 */
```

```
poc_gptimer1_output_disable_mode_select(POC_HIGH_IMPEDANCE_OUT,  
POC_HIGH_IMPEDANCE_OUT);
```

### 函数 poc\_request\_struct\_para\_init

函数poc\_request\_struct\_para\_init描述见下表:

表 3-626. 函数 poc\_request\_struct\_para\_init

函数名称	poc_request_struct_para_init
函数原型	void poc_request_struct_para_init(poc_request_struct *request)
功能描述	初始化 POC 请求结构体
先决条件	-
被调用函数	-
输入参数{in}	
request	POC 请求结构体参数，结构体成员参考 <a href="#">表 3-604. 结构体类型</a> <a href="#">poc_request_struct</a>
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* initialize POC request struct with a default value */
```

```
poc_request_struct poc_request;
```

```
poc_request_struct_para_init(&poc_request);
```

**函数 poc\_request\_select**

函数poc\_request\_select描述见下表：

**表 3-627. 函数 poc\_request\_select**

函数名称	poc_request_select
函数原型	void poc_request_select(poc_request_struct *request, target_timer_enum target_timer)
功能描述	请求选择目标定时器
先决条件	-
被调用函数	-
输入参数{in}	
request	POC 请求结构体参数，结构体成员参考 <a href="#">表 3-604. 结构体类型 poc_request_struct</a>
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* request select TIMER0 */
poc_request_struct poc_request;
poc_request_select(&poc_request, TARGET_TIMER0);
```

**函数 poc\_cmp\_dreq\_status\_config**

函数poc\_cmp\_dreq\_status\_config描述见下表：

**表 3-628. 函数 poc\_cmp\_dreq\_status\_config**

函数名称	poc_cmp_dreq_status_config
函数原型	void poc_cmp_dreq_status_config(uint32_t cmp0dreq_status, uint32_t cmp1dreq_status, uint32_t cmp2dreq_status, uint32_t cmp3dreq_status)
功能描述	配置比较器禁能请求状态
先决条件	-
被调用函数	-
输入参数{in}	
cmp0dreq_status	比较器 0 禁能请求状态
CMP0_DREQ_DISABLE	禁能
CMP0_DREQ_ENABLE	使能
输入参数{in}	
cmp1dreq_status	比较器 1 禁能请求状态

<i>CMP1_DREQ_DISABLE</i>	禁能
<i>CMP1_DREQ_ENABLE</i>	使能
输入参数{in}	
<b>cmp2dreq_status</b>	比较器 2 禁能请求状态
<i>CMP2_DREQ_DISABLE</i>	禁能
<i>CMP2_DREQ_ENABLE</i>	使能
输入参数{in}	
<b>cmp1dreq_status</b>	比较器 3 禁能请求状态
<i>CMP3_DREQ_DISABLE</i>	禁能
<i>CMP3_DREQ_ENABLE</i>	使能
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* configure comparator disabling request status */
```

```
poc_cmp_dreq_status_config(CMP0_DREQ_DISABLE,          CMP1_DREQ_DISABLE,
CMP2_DREQ_ENABLE, CMP3_DREQ_ENABLE);
```

### 函数 **poc\_cmp\_dreq\_status\_extended\_config**

函数 **poc\_cmp\_dreq\_status\_extended\_config** 描述见下表：

**表 3-629. 函数 **poc\_cmp\_dreq\_status\_extended\_config****

函数名称	<b>poc_cmp_dreq_status_extended_config</b>
函数原型	void poc_cmp_dreq_status_extended_config(uint32_t cmp0dreq_status, uint32_t cmp1dreq_status, uint32_t cmp2dreq_status, uint32_t cmp3dreq_status, target_timer_enum target_timer)
功能描述	配置比较器扩展禁能请求状态
先决条件	-
被调用函数	-
输入参数{in}	
<b>cmp0dreq_status</b>	比较器 0 禁能请求状态
<i>CMP0_DREQ_DISABLE</i>	禁能

<code>CMP0_DREQ_ENABLE</code>	使能
输入参数{in}	
<code>cmp1dreq_status</code>	比较器 1 禁能请求状态
<code>CMP1_DREQ_DISABLE</code>	禁能
<code>CMP1_DREQ_ENABLE</code>	使能
输入参数{in}	
<code>cmp2dreq_status</code>	比较器 2 禁能请求状态
<code>CMP2_DREQ_DISABLE</code>	禁能
<code>CMP2_DREQ_ENABLE</code>	使能
输入参数{in}	
<code>cmp3dreq_status</code>	比较器 3 禁能请求状态
<code>CMP3_DREQ_DISABLE</code>	禁能
<code>CMP3_DREQ_ENABLE</code>	使能
输入参数{in}	
<code>target_timer</code>	参考 <a href="#">表 3-608</a> 枚举类型 <code>target_timer_enum</code>
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* configure comparator disabling request status for TIMER0 */
```

```
poc_cmp_dreq_status_extended_config(CMP0_DREQ_DISABLE, CMP1_DREQ_DISABLE,
CMP2_DREQ_ENABLE, CMP3_DREQ_ENABLE, TARGET_TIMER0);
```

### 函数 `poc_input_detection_mask`

函数 `poc_input_detection_mask` 描述见下表：

表 3-630. 函数 `poc_input_detection_mask`

函数名称	<code>poc_input_detection_mask</code>
函数原型	<code>void poc_input_detection_mask(poc_pin_enum poc_pin, uint32_t mask_source)</code>
功能描述	屏蔽 POC_INn 输入检测
先决条件	-

被调用函数	-
输入参数{in}	
poc_pin	参考 <a href="#">表 3-606. 枚举类型 poc_pin_enum</a>
输入参数{in}	
mask_source	屏蔽源选择
POC_REQUEST_NOT_MASKED	不屏蔽检测
POC_MASK_SOURCE_TIMER0_CH0	TIMER0_CH0 信号屏蔽检测
POC_MASK_SOURCE_TIMER0_CH1	TIMER0_CH1 信号屏蔽检测
POC_MASK_SOURCE_TIMER0_CH2	TIMER0_CH2 信号屏蔽检测
POC_MASK_SOURCE_TIMER0_CH3	TIMER0_CH3 信号屏蔽检测
POC_MASK_SOURCE_TIMER0_MCH0	TIMER0_MCH0 信号屏蔽检测
POC_MASK_SOURCE_TIMER0_MCH1	TIMER0_MCH1 信号屏蔽检测
POC_MASK_SOURCE_TIMER0_MCH2	TIMER0_MCH2 信号屏蔽检测
POC_MASK_SOURCE_TIMER0_MCH3	TIMER0_MCH3 信号屏蔽检测
POC_MASK_SOURCE_TIMER7_CH0	TIMER7_CH0 信号屏蔽检测
POC_MASK_SOURCE_TIMER7_CH1	TIMER7_CH1 信号屏蔽检测
POC_MASK_SOURCE_TIMER7_CH2	TIMER7_CH2 信号屏蔽检测
POC_MASK_SOURCE_TIMER7_CH3	TIMER7_CH3 信号屏蔽检测
POC_MASK_SOURCE_TIMER7_MCH0	TIMER7_MCH0 信号屏蔽检测
POC_MASK_SOURCE_TIMER7_MCH1	TIMER7_MCH1 信号屏蔽检测
POC_MASK_SOURCE_TIMER7_MCH2	TIMER7_MCH2 信号屏蔽检测

<i>RCE_TIMER7_MC H2</i>	
<i>POC_MASK_SOU RCE_TIMER7_MC H3</i>	TIMER7_MCH3 信号屏蔽检测
<i>POC_MASK_SOU RCE_TIMER1_CH0</i>	TIMER1_CH0 信号屏蔽检测
<i>POC_MASK_SOU RCE_TIMER1_CH1</i>	TIMER1_CH1 信号屏蔽检测
<i>POC_MASK_SOU RCE_TIMER1_CH2</i>	TIMER1_CH2 信号屏蔽检测
<i>POC_MASK_SOU RCE_TIMER1_CH3</i>	TIMER1_CH3 信号屏蔽检测
<i>POC_MASK_SOU RCE_TIMER2_CH0</i>	TIMER2_CH0 信号屏蔽检测
<i>POC_MASK_SOU RCE_TIMER2_CH1</i>	TIMER2_CH1 信号屏蔽检测
<i>POC_MASK_SOU RCE_TIMER2_CH2</i>	TIMER2_CH2 信号屏蔽检测
<i>POC_MASK_SOU RCE_TIMER2_CH3</i>	TIMER2_CH3 信号屏蔽检测
<i>POC_MASK_SOU RCE_GPTIMER0_ CH0</i>	GPTIMER0_CH0 信号屏蔽检测
<i>POC_MASK_SOU RCE_GPTIMER0_ CH1</i>	GPTIMER0_CH1 信号屏蔽检测
<i>POC_MASK_SOU RCE_GPTIMER1_ CH0</i>	GPTIMER1_CH0 信号屏蔽检测
<i>POC_MASK_SOU RCE_GPTIMER1_ CH1</i>	GPTIMER1_CH1 信号屏蔽检测
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* mask POC_IN0 input detection */
```

```
poc_input_detection_mask(POC_IN0, POC_MASK_SOURCE_TIMER0_CH0);
```



## 函数 poc\_cmp\_output\_detection\_mask

函数poc\_cmp\_output\_detection\_mask描述见下表：

表 3-631. 函数 poc\_cmp\_output\_detection\_mask

函数名称	poc_cmp_output_detection_mask
函数原型	void poc_cmp_output_detection_mask(cmp_output_enum cmp_output, uint32_t mask_source)
功能描述	屏蔽比较器输出检测
先决条件	-
被调用函数	-
输入参数{in}	
cmp_output	参考 <a href="#">表 3-607. 枚举类型 cmp_output_enum</a>
输入参数{in}	
mask_source	屏蔽源选择
POC_REQUEST_NOT_MASKED	不屏蔽检测
POC_MASK_SOURCE_TIMER0_CH0	TIMER0_CH0 信号屏蔽检测
POC_MASK_SOURCE_TIMER0_CH1	TIMER0_CH1 信号屏蔽检测
POC_MASK_SOURCE_TIMER0_CH2	TIMER0_CH2 信号屏蔽检测
POC_MASK_SOURCE_TIMER0_CH3	TIMER0_CH3 信号屏蔽检测
POC_MASK_SOURCE_TIMER0_MCH0	TIMER0_MCH0 信号屏蔽检测
POC_MASK_SOURCE_TIMER0_MCH1	TIMER0_MCH1 信号屏蔽检测
POC_MASK_SOURCE_TIMER0_MCH2	TIMER0_MCH2 信号屏蔽检测
POC_MASK_SOURCE_TIMER0_MCH3	TIMER0_MCH3 信号屏蔽检测
POC_MASK_SOURCE_TIMER7_CH0	TIMER7_CH0 信号屏蔽检测
POC_MASK_SOURCE_TIMER7_CH1	TIMER7_CH1 信号屏蔽检测
POC_MASK_SOURCE_TIMER7_CH2	TIMER7_CH2 信号屏蔽检测

POC_MASK_SOU RCE_TIMER7_CH3	TIMER7_CH3 信号屏蔽检测
POC_MASK_SOU RCE_TIMER7_MC H0	TIMER7_MCH0 信号屏蔽检测
POC_MASK_SOU RCE_TIMER7_MC H1	TIMER7_MCH1 信号屏蔽检测
POC_MASK_SOU RCE_TIMER7_MC H2	TIMER7_MCH2 信号屏蔽检测
POC_MASK_SOU RCE_TIMER7_MC H3	TIMER7_MCH3 信号屏蔽检测
POC_MASK_SOU RCE_TIMER1_CH0	TIMER1_CH0 信号屏蔽检测
POC_MASK_SOU RCE_TIMER1_CH1	TIMER1_CH1 信号屏蔽检测
POC_MASK_SOU RCE_TIMER1_CH2	TIMER1_CH2 信号屏蔽检测
POC_MASK_SOU RCE_TIMER1_CH3	TIMER1_CH3 信号屏蔽检测
POC_MASK_SOU RCE_TIMER2_CH0	TIMER2_CH0 信号屏蔽检测
POC_MASK_SOU RCE_TIMER2_CH1	TIMER2_CH1 信号屏蔽检测
POC_MASK_SOU RCE_TIMER2_CH2	TIMER2_CH2 信号屏蔽检测
POC_MASK_SOU RCE_TIMER2_CH3	TIMER2_CH3 信号屏蔽检测
POC_MASK_SOU RCE_GPTIMER0_ CH0	GPTIMER0_CH0 信号屏蔽检测
POC_MASK_SOU RCE_GPTIMER0_ CH1	GPTIMER0_CH1 信号屏蔽检测
POC_MASK_SOU RCE_GPTIMER1_ CH0	GPTIMER1_CH0 信号屏蔽检测
POC_MASK_SOU RCE_GPTIMER1_ CH1	GPTIMER1_CH1 信号屏蔽检测

输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* mask CMP0 output detection */
```

```
poc_cmp_output_detection_mask(CMP0_OUT, POC_MASK_SOURCE_TIMER0_CH0);
```

### 函数 poc\_flag\_get

函数poc\_flag\_get描述见下表：

表 3-632. 函数 poc\_flag\_get

函数名称	poc_flag_get
函数原型	FlagStatus poc_flag_get(uint32_t flag)
功能描述	获取 POC 标志位
先决条件	-
被调用函数	-
输入参数{in}	
flag	POC 标志位
POC_FLAG_IN0IF	POC_IN0 输入检测标志位
POC_FLAG_IN1IF	POC_IN1 输入检测标志位
POC_FLAG_IN2IF	POC_IN2 输入检测标志位
POC_FLAG_IN3IF	POC_IN3 输入检测标志位
POC_FLAG_IN4IF	POC_IN4 输入检测标志位
POC_FLAG_IN5IF	POC_IN5 输入检测标志位
POC_FLAG_HXTA LSDF	HXTAL 卡住检测标志位
POC_FLAG_LOCK UPDF	CPU 锁定检测标志位
POC_FLAG_TIME R0_CCIF	TIMER0 同时导通检测标志位
POC_FLAG_TIME R7_CCIF	TIMER7 同时导通检测标志位
POC_FLAG_CMP0 DF	比较器 0 检测标志位
POC_FLAG_CMP1 DF	比较器 1 检测标志位
POC_FLAG_CMP2 DF	比较器 2 检测标志位
POC_FLAG_CMP3	比较器 3 检测标志位

<i>DF</i>	
输出参数{out}	
-	-
返回值	
<b>FlagStatus</b>	SET 或者 RESET

例如:

```
FlagStatus status;
```

```
/* get POC_IN0 input detection flag */
```

```
status = poc_flag_get(POC_FLAG_IN0IF);
```

### 函数 poc\_flag\_clear

函数poc\_flag\_clear描述见下表:

表 3-633. 函数 poc\_flag\_clear

函数名称	poc_flag_clear
函数原型	void poc_flag_clear(uint32_t flag)
功能描述	清除 POC 标志位
先决条件	-
被调用函数	-
输入参数{in}	
<b>flag</b>	POC 标志位
<i>POC_FLAG_IN0IF</i>	POC_IN0 输入检测标志位
<i>POC_FLAG_IN1IF</i>	POC_IN1 输入检测标志位
<i>POC_FLAG_IN2IF</i>	POC_IN2 输入检测标志位
<i>POC_FLAG_IN3IF</i>	POC_IN3 输入检测标志位
<i>POC_FLAG_IN4IF</i>	POC_IN4 输入检测标志位
<i>POC_FLAG_IN5IF</i>	POC_IN5 输入检测标志位
<i>POC_FLAG_HXTA LSDF</i>	HXTAL 卡住检测标志位
<i>POC_FLAG_LOCK UPDF</i>	CPU 锁定检测标志位
<i>POC_FLAG_TIME R0_CCIF</i>	TIMER0 同时导通检测标志位
<i>POC_FLAG_TIME R7_CCIF</i>	TIMER7 同时导通检测标志位
<i>POC_FLAG_CMP0 DF</i>	比较器 0 检测标志位
<i>POC_FLAG_CMP1 DF</i>	比较器 1 检测标志位
<i>POC_FLAG_CMP2</i>	比较器 2 检测标志位

DF	
POC_FLAG_CMP3 DF	比较器 3 检测标志位
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* clear POC_IN0 input detection flag */
poc_flag_clear(POC_FLAG_IN0IF);
```

### 函数 poc\_interrupt\_enable

函数poc\_interrupt\_enable描述见下表：

表 3-634. 函数 poc\_interrupt\_enable

函数名称	poc_interrupt_enable
函数原型	void poc_interrupt_enable(uint32_t flag)
功能描述	使能 POC 中断
先决条件	-
被调用函数	-
输入参数{in}	
interrupt	参考 <a href="#">表 3-609. 枚举类型 poc_interrupt_enum</a>
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* enable POC_IN0 input detection interrupt */
poc_interrupt_enable(POC_INT_IN0);
```

### 函数 poc\_interrupt\_disable

函数poc\_interrupt\_disable描述见下表：

表 3-635. 函数 poc\_interrupt\_disable

函数名称	poc_interrupt_disable
函数原型	void poc_interrupt_disable(uint32_t flag)
功能描述	禁能 POC 中断
先决条件	-
被调用函数	-

输入参数{in}	
interrupt	参考 <a href="#">表 3-609. 枚举类型 poc_interrupt_enum</a>
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* disable POC_IN0 input detection interrupt */
```

```
poc_interrupt_disable (POC_INT_IN0);
```

### 函数 poc\_interrupt\_flag\_get

函数poc\_interrupt\_flag\_get描述见下表:

表 3-636. 函数 poc\_interrupt\_flag\_get

函数名称	poc_interrupt_flag_get
函数原型	FlagStatus poc_interrupt_flag_get (uint32_t flag)
功能描述	获取 POC 中断标志位
先决条件	-
被调用函数	-
输入参数{in}	
flag	POC 中断标志位
POC_INT_FLAG_I N0IF	POC_IN0 输入中断标志位
POC_INT_FLAG_I N1IF	POC_IN1 输入中断标志位
POC_INT_FLAG_I N2IF	POC_IN2 输入中断标志位
POC_INT_FLAG_I N3IF	POC_IN3 输入中断标志位
POC_INT_FLAG_I N4IF	POC_IN4 输入中断标志位
POC_INT_FLAG_I N5IF	POC_IN5 输入中断标志位
POC_INT_FLAG_TI MER0_CCIF	TIMER0 同时导通中断标志位
POC_INT_FLAG_TI MER7_CCIF	TIMER7 同时导通中断标志位
输出参数{out}	
-	-
返回值	

<b>FlagStatus</b>	SET 或者 RESET
-------------------	--------------

例如：

```
FlagStatus status;
```

```
/* get POC_IN0 input interrupt flag */
```

```
status = poc_interrupt_flag_get(POC_INT_FLAG_IN0IF);
```

### 函数 **poc\_interrupt\_flag\_clear**

函数 **poc\_interrupt\_flag\_clear** 描述见下表：

**表 3-637. 函数 **poc\_interrupt\_flag\_clear****

函数名称	poc_interrupt_flag_clear
函数原型	void poc_interrupt_flag_clear(uint32_t int_flag)
功能描述	清除 POC 中断标志位
先决条件	-
被调用函数	-
输入参数{in}	
<b>flag</b>	POC 中断标志位
<i>POC_INT_FLAG_IN0IF</i>	POC_IN0 输入中断标志位
<i>POC_INT_FLAG_IN1IF</i>	POC_IN1 输入中断标志位
<i>POC_INT_FLAG_IN2IF</i>	POC_IN2 输入中断标志位
<i>POC_INT_FLAG_IN3IF</i>	POC_IN3 输入中断标志位
<i>POC_INT_FLAG_IN4IF</i>	POC_IN4 输入中断标志位
<i>POC_INT_FLAG_IN5IF</i>	POC_IN5 输入中断标志位
<i>POC_INT_FLAG_TIMER0_CCIF</i>	TIMER0 同时导通中断标志位
<i>POC_INT_FLAG_TIMER7_CCIF</i>	TIMER7 同时导通中断标志位
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* clear POC_IN0 input interrupt flag */
```

poc\_interrupt\_flag\_clear (POC\_INT\_FLAG\_IN0IF);

### 3.23. 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[3.23.1](#)描述了RCU的寄存器列表，章节[3.23.2](#)对RCU库函数进行说明。

#### 3.23.1. 外设寄存器说明

RCU寄存器列表如下表所示：

表 3-638. RCU 寄存器

寄存器名称	寄存器描述
RCU_CTL	控制寄存器
RCU_CFG0	时钟配置寄存器0
RCU_INT	时钟中断寄存器
RCU_APB2RST	APB2复位寄存器
RCU_APB1RST	APB1复位寄存器
RCU_AHBMEN	AHB使能寄存器
RCU_APB2EN	APB2使能寄存器
RCU_APB1EN	APB1使能寄存器
RCU_RSTSCK	复位源/时钟寄存器
RCU_AHBRST	AHB复位寄存器
RCU_CFG1	时钟配置寄存器1
RCU_CFG2	时钟配置寄存器2

#### 3.23.2. 外设库函数说明

RCU库函数列表如下表所示：

表 3-639. 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	使能外设复位
rcu_periph_reset_disable	禁能外设复位
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



库函数名称	库函数描述
rcu_ahb_clock_config	配置 AHB 时钟预分频选择
rcu_apb1_clock_config	配置 APB1 时钟预分频选择
rcu_apb2_clock_config	配置 APB2 时钟预分频选择
rcu_ckout_config	配置 CKOUT 时钟源选择
rcu_pll_config	配置 PLL 时钟
rcu_system_reset_enable	使能 RCU 系统复位
rcu_system_reset_disable	禁能 RCU 系统复位
rcu_adc_clock_config	配置 CK_ADCPRE 时钟源和分频因子
rcu_i2c_clock_source_config	配置 CK_I2C 的时钟源选择
rcu_osc_i_stab_wait	等待振荡器稳定标志位置位或振荡器起振超时
rcu_osc_i_on	打开振荡器
rcu_osc_i_off	关闭振荡器
rcu_osc_i_bypass_mode_enable	使能振荡器时钟旁路模式
rcu_osc_i_bypass_mode_disable	禁能振荡器时钟旁路模式
rcu_hxtal_frequency_scale_select	选择 HXTAL 频率范围
rcu_irc32m_adjust_value_set	设置内部 32MHz 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_pll_clock_monitor_enable	使能 PLL 时钟监视器
rcu_pll_clock_monitor_disable	禁能 PLL 时钟监视器
rcu_clock_freq_get	获取系统时钟、总线频率
rcu_flag_get	获取时钟稳定和外设复位标志
rcu_all_reset_flag_clear	清除所有复位标志位
rcu_interrupt_flag_get	获取时钟稳定中断和时钟阻塞中断标志
rcu_interrupt_flag_clear	清除中断标志
rcu_interrupt_enable	使能时钟稳定中断
rcu_interrupt_disable	禁能时钟稳定中断

### 枚举类型 rcu\_periph\_enum

表 3-640. 枚举类型 rcu\_periph\_enum

成员名称	功能描述
RCU_DMA0	DMA0时钟
RCU_DMA1	DMA1时钟
RCU_DMAMUX	DMAMUX时钟
RCU_CRC	CRC时钟
RCU_GPIOA	GPIOA时钟
RCU_GPIOB	GPIOB时钟
RCU_GPIOC	GPIOC时钟
RCU_GPIOD	GPIOD时钟
RCU_GPIOE	GPIOE时钟

成员名称	功能描述
RCU_GPIOF	GPIOF时钟
RCU_GPIOG	GPIOG时钟
RCU_GPION	GPION时钟
RCU_POC	POC时钟
RCU_GTOC	GTOC时钟
RCU_SVPWM	SVPWM时钟
RCU_TMU	TMU时钟
RCU_SYSCFG	SYSCFG时钟
RCU_CMP	CMP时钟
RCU_ADC0	ADC0时钟
RCU_ADC2	ADC2时钟
RCU_TIMER0	TIMER0时钟
RCU_SPI	SPI时钟
RCU_TIMER7	TIMER7时钟
RCU_TIMER1	TIMER1时钟
RCU_TIMER2	TIMER2时钟
RCU_GPTIMER0	GPTIMER0时钟
RCU_GPTIMER1	GPTIMER1时钟
RCU_EVIC	EVIC时钟
RCU_CAN	CAN时钟
RCU_CPTIMER0	CPTIMER0时钟
RCU_CPTIMER1	CPTIMER1时钟
RCU_CPTIMERW	CPTIMERW时钟
RCU_WWDGT	WWDGT时钟
RCU_UART0	UART0时钟
RCU_UART1	UART1时钟
RCU_UART2	UART2时钟
RCU_UART3	UART3时钟
RCU_I2C	I2C时钟
RCU_CFMU	CFMU时钟
RCU_PMU	PMU时钟
RCU_DAC	DAC时钟

### 枚举类型 `rcu_periph_reset_enum`

表 3-641. 枚举类型 `rcu_periph_reset_enum`

成员名称	功能描述
RCU_DMA0RST	DMA0时钟复位
RCU_DMA1RST	DMA1时钟复位
RCU_DMAMUXRST	DMAMUX时钟复位
RCU_CRCRST	CRC时钟复位

成员名称	功能描述
RCU_GPIOARST	GPIOA时钟复位
RCU_GPIOBRST	GPIOB时钟复位
RCU_GPIOCRST	GPIOC时钟复位
RCU_GPIODRST	GPIOD时钟复位
RCU_GPIOERST	GPIOE时钟复位
RCU_GPIOFRST	GPIOF时钟复位
RCU_GPIOGRST	GPIOG时钟复位
RCU_GPIONRST	GPION时钟复位
RCU_POCRST	POC时钟复位
RCU_GTOCRST	GTOC时钟复位
RCU_SVPWMRST	SVPWM时钟复位
RCU_TMURST	TMU时钟复位
RCU_SYSCFGRST	系统配置复位
RCU_CMPRST	比较器时钟复位
RCU_ADC0RST	ADC0时钟复位
RCU_ADC2RST	ADC2时钟复位
RCU_TIMER0RST	TIMER0时钟复位
RCU_SPIRST	SPI时钟复位
RCU_TIMER7RST	TIMER7时钟复位
RCU_TIMER1RST	TIMER1时钟复位
RCU_TIMER2RST	TIMER2时钟复位
RCU_GPTIMER0RST	GPTIMER0时钟复位
RCU_GPTIMER1RST	GPTIMER1时钟复位
RCU_CANRST	CAN时钟复位
RCU_CPTIMER0RST	CPTIMER0时钟复位
RCU_CPTIMER1RST	CPTIMER1时钟复位
RCU_CPTIMERWRST	CPTIMERW时钟复位
RCU_WWDGTRST	WWDGT时钟复位
RCU_UART0RST	UART0时钟复位
RCU_UART1RST	UART1时钟复位
RCU_UART2RST	UART2时钟复位
RCU_UART3RST	UART3时钟复位
RCU_I2CRST	I2C时钟复位
RCU_CFMURST	CFMU时钟复位
RCU_PMURST	PMU时钟复位
RCU_DACRST	DAC时钟复位
RCU_DMA0RST	DMA0时钟复位
RCU_DMA1RST	DMA1时钟复位
RCU_DMAMUXRST	DMAMUX时钟复位
RCU_CRCRST	CRC时钟复位

成员名称	功能描述
RCU_GPIOARST	GPIOA时钟复位

### 枚举类型 `rcu_periph_sleep_enum`

表 3-642. 枚举类型 `rcu_periph_sleep_enum`

成员名称	功能描述
RCU_SRAM_SLP	SRAM接口时钟
RCU_FMC_SLP	FMC时钟

### 枚举类型 `rcu_flag_enum`

表 3-643. 枚举类型 `rcu_flag_enum`

成员名称	功能描述
RCU_FLAG_IRC32MSTB	IRC32M振荡器稳定标志
RCU_FLAG_HXTALSTB	外部高速晶振稳定标志
RCU_FLAG_PLLSTB	PLL稳定标志
RCU_FLAG_IRC32KSTB	IRC32K振荡器稳定标志
RCU_FLAG_LVD0RST	LVD0复位标志
RCU_FLAG_LOCKUPRST	CPU Lockup复位标志
RCU_FLAG_LVD1RST	LVD1复位标志
RCU_FLAG_LVD2RST	LVD2复位标志
RCU_FLAG_LOHRST	LOH复位标志
RCU_FLAG_LOPRST	LOP复位标志
RCU_FLAG_V11RST	V1.1复位标志
RCU_FLAG_RSTFC	复位标志清除
RCU_FLAG_OBLRST	选项字节复位标志
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 3-644. 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_IRC32MSTB	IRC32M时钟稳定中断标志
RCU_INT_FLAG_HXTALSTB	外部高速晶振时钟稳定中断标志
RCU_INT_FLAG_PLLSTB	PLL时钟稳定中断标志
RCU_INT_FLAG_PLLM	PLL时钟监视器中断标志

成员名称	功能描述
RCU_INT_FLAG_CKM	外部高速晶振时钟阻塞中断标志

### 枚举类型 `rcu_int_flag_clear_enum`

表 3-645. 枚举类型 `rcu_int_flag_clear_enum`

成员名称	功能描述
RCU_INT_FLAG_IRC32KSTB_CLR	IRC32K时钟稳定中断清除标志
RCU_INT_FLAG_IRC32MSTB_CLR	IRC32M时钟稳定中断清除标志
RCU_INT_FLAG_HXTALSTB_CLR	外部高速晶振时钟稳定中断清除标志
RCU_INT_FLAG_PLLSTB_CLR	PLL时钟稳定中断清除标志
RCU_INT_FLAG_PLLM_CLR	清除PLL时钟监视器中断
RCU_INT_FLAG_CKM_CLR	外部高速晶振时钟阻塞中断清除标志

### 枚举类型 `rcu_int_enum`

表 3-646. 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_IRC32MSTB	IRC32M时钟稳定中断
RCU_INT_HXTALSTB	外部高速晶振时钟稳定中断
RCU_INT_PLLSTB	PLL时钟稳定中断
RCU_INT_PLLM	PLL监视器中断

### 枚举类型 `rcu_osci_type_enum`

表 3-647. 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_IRC32M	IRC32M振荡器
RCU_IRC32K	IRC32K振荡器
RCU_PLL_CK	锁相环时钟

### 枚举类型 `rcu_clock_freq_enum`

表 3-648. 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

成员名称	功能描述
CK_APB2	APB2时钟

### 函数 rcu\_deinit

函数rcu\_deinit描述见下表：

表 3-649. 函数 rcu\_deinit

函数名称	rcu_deinit
函数原型	void rcu_deinit(void)
功能描述	deinitialize the RCU
先决条件	-
被调用函数	-
输入参数{in}	
-	-
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* deinitialize the RCU */
rcu_deinit();
```

### 函数 rcu\_periph\_clock\_enable

函数rcu\_periph\_clock\_enable描述见下表：

表 3-650. 函数 rcu\_periph\_clock\_enable

函数名称	rcu_periph_clock_enable
函数原型	void rcu_periph_clock_enable(rcu_periph_enum periph)
功能描述	使能外设时钟
先决条件	-
被调用函数	-
输入参数{in}	
periph	RCU 外设，参考 <a href="#">表 3-640. 枚举类型 rcu_periph_enum</a>
RCU_TMU	TMU 时钟
RCU_SVPWM	SVPWM 时钟
RCU_GTOC	GTOC 时钟
RCU_POC	POC 时钟
RCU_GPIOx (x = A,B,C,D,E,F,G,N)	GPIO 口时钟
RCU_CRC	CRC 时钟

<i>RCU_DMAMUX</i>	DMAMUX 时钟
<i>RCU_DMAx (x = 0,1)</i>	DMA 时钟
<i>RCU_CAN</i>	CAN 时钟
<i>RCU_EVIC</i>	EVIC 时钟
<i>RCU_TIMERx (x = 0,1,2,7)</i>	TIMER 时钟
<i>RCU_GPTIMERx (x = 0,1)</i>	GPTIMER 时钟
<i>RCU_CPTIMERx (x = 0,1,W)</i>	CPTIMER 时钟
<i>RCU_SPI</i>	SPI 时钟
<i>RCU_WWDGT</i>	WWDGT 时钟
<i>RCU_UARTx(x = 0,1,2,3)</i>	UART 时钟
<i>RCU_I2C</i>	I2C 时钟
<i>RCU_SYSCFG</i>	系统配置时钟
<i>RCU_PMU</i>	PMU 时钟
<i>RCU_CFMU</i>	CFMU 时钟
<i>RCU_DAC</i>	DAC 时钟
<i>RCU_ADCx (x = 0,2)</i>	ADC 时钟
<i>RCU_CMP</i>	CMP 时钟
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* enable the USART0 clock */
```

```
rcu_periph_clock_enable(RCU_USART0);
```

## 函数 `rcu_periph_clock_disable`

函数 `rcu_periph_clock_disable` 描述见下表:

**表 3-651. 函数 `rcu_periph_clock_disable`**

函数名称	<code>rcu_periph_clock_disable</code>
函数原型	<code>void rcu_periph_clock_disable(rcu_periph_enum periph)</code>
功能描述	disable the peripherals clock
先决条件	-
被调用函数	-
输入参数{in}	
<b>periph</b>	RCU 外设, 参考 <a href="#">表 3-640. 枚举类型 <code>rcu_periph_enum</code></a>

<i>RCU_TMU</i>	TMU 时钟
<i>RCU_SVPWM</i>	SVPWM 时钟
<i>RCU_GTOC</i>	GTOC 时钟
<i>RCU_POC</i>	POC 时钟
<i>RCU_GPIOx</i> ( <i>x</i> = <i>A,B,C,D,E,F,G,N</i> )	GPIO 口时钟
<i>RCU_CRC</i>	CRC 时钟
<i>RCU_DMAMUX</i>	DMAMUX 时钟
<i>RCU_DMAx</i> ( <i>x</i> = 0,1)	DMA 时钟
<i>RCU_CAN</i>	CAN 时钟
<i>RCU_EVIC</i>	EVIC 时钟
<i>RCU_TIMERx</i> ( <i>x</i> = 0,1,2,7)	TIMER 时钟
<i>RCU_GPTIMERx</i> ( <i>x</i> = 0,1)	GPTIMER 时钟
<i>RCU_CPTIMERx</i> ( <i>x</i> = 0,1,W)	CPTIMER 时钟
<i>RCU_SPI</i>	SPI 时钟
<i>RCU_WWDGT</i>	WWDGT 时钟
<i>RCU_UARTx</i> ( <i>x</i> =0,1,2,3)	UART 时钟
<i>RCU_I2C</i>	I2C 时钟
<i>RCU_SYSCFG</i>	系统配置时钟
<i>RCU_PMU</i>	PMU 时钟
<i>RCU_CFMU</i>	CFMU 时钟
<i>RCU_DAC</i>	DAC 时钟
<i>RCU_ADCx</i> ( <i>x</i> = 0,2)	ADC 时钟
<i>RCU_CMP</i>	CMP 时钟
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* disable the USART0 clock */
```

```
rcu_periph_clock_disable(RCU_USART0);
```

### 函数 `rcu_periph_clock_sleep_enable`

函数 `rcu_periph_clock_sleep_enable` 描述见下表:



表 3-652. 函数 rcu\_periph\_clock\_sleep\_enable

函数名称	rcu_periph_clock_sleep_enable
函数原型	void rcu_periph_clock_sleep_enable(rcu_periph_sleep_enum periph)
功能描述	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 <a href="#">表 3-642. 枚举类型 rcu_periph_sleep_enum</a>
RCU_FMC_SLP	FMC 时钟
RCU_SRAM_SLP	SRAM 时钟
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* enable the FMC clock when in sleep mode */
rcu_periph_clock_sleep_enable(RCU_FMC_SLP);
```

### 函数 rcu\_periph\_clock\_sleep\_disable

函数rcu\_periph\_clock\_sleep\_disable描述见下表：

表 3-653. 函数 rcu\_periph\_clock\_sleep\_disable

函数名称	rcu_periph_clock_sleep_disable
函数原型	void rcu_periph_clock_sleep_disable(rcu_periph_sleep_enum periph)
功能描述	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 <a href="#">表 3-642. 枚举类型 rcu_periph_sleep_enum</a>
RCU_SRAM_SLP	SRAM 时钟
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* disable the SRAM clock when in sleep mode */
rcu_periph_clock_sleep_disable(RCU_SRAM_SLP);
```

## 函数 rcu\_periph\_reset\_enable

函数rcu\_periph\_reset\_enable描述见下表:

表 3-654. 函数 rcu\_periph\_reset\_enable

函数名称	rcu_periph_reset_enable
函数原型	void rcu_periph_reset_enable(rcu_periph_reset_enum periph_reset)
功能描述	使能外设复位
先决条件	-
被调用函数	-
输入参数{in}	
<b>periph_reset</b>	RCU 外设复位, 参考 <a href="#">表 3-641. 枚举类型 rcu_periph_reset_enum</a>
<i>RCU_TMURST</i>	TMU 时钟
<i>RCU_SVPWMRST</i>	SVPWM 时钟
<i>RCU_GTOCRST</i>	GTOC 时钟
<i>RCU_POCRST</i>	POC 时钟
<i>RCU_GPIOxRST</i> (x = A,B,C,D,E,F,G,N)	GPIO 口时钟
<i>RCU_CRCRST</i>	CRC 时钟
<i>RCU_DMAMUXRST</i>	DMAMUX 时钟
<i>RCU_DMAxRST</i> (x = 0,1)	DMA 时钟
<i>RCU_CANRST</i>	CAN 时钟
<i>RCU_TIMERxRST</i> (x = 0,1,2,7)	TIMER 时钟
<i>RCU_GPTIMERxRST</i> (x = 0,1)	GPTIMER 时钟
<i>RCU_CPTIMERxRST</i> (x = 0,1,W)	CPTIMER 时钟
<i>RCU_SPIRST</i>	SPI 时钟
<i>RCU_WWDGTRST</i>	WWDGT 时钟
<i>RCU_UARTxRST</i> (x = 0,1,2,3)	UART 时钟
<i>RCU_I2CRST</i>	I2C 时钟
<i>RCU_SYSCFGRST</i>	系统配置时钟
<i>RCU_PMURST</i>	PMU 时钟
<i>RCU_CFMURST</i>	CFMU 时钟
<i>RCU_DACRST</i>	DAC 时钟
<i>RCU_ADCxRST</i> (x = 0,2)	ADC 时钟
<i>RCU_CMPRST</i>	CMP 时钟
输出参数{out}	

-	-
返回值	
-	-

例如:

```
/* enable SPI reset */
```

```
rcu_periph_reset_enable(RCU_SPIRST);
```

### 函数 rcu\_periph\_reset\_disable

函数rcu\_periph\_reset\_disable描述见下表:

表 3-655. 函数 rcu\_periph\_reset\_disable

函数名称	rcu_periph_reset_disable
函数原型	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 <a href="#">表 3-641. 枚举类型 rcu_periph_reset_enum</a>
RCU_TMURST	TMU 时钟
RCU_SVPWMRST	SVPWM 时钟
RCU_GTOCRST	GTOC 时钟
RCU_POCRST	POC 时钟
RCU_GPIOxRST (x = A,B,C,D,E,F,G,N)	GPIO 口时钟
RCU_CRCRST	CRC 时钟
RCU_DMAMUXRST	DMAMUX 时钟
RCU_DMAxRST (x = 0,1)	DMA 时钟
RCU_CANRST	CAN 时钟
RCU_TIMERxRST (x = 0,1,2,7)	TIMER 时钟
RCU_GPTIMERxRST (x = 0,1)	GPTIMER 时钟
RCU_CPTIMERxRST (x = 0,1,W)	CPTIMER 时钟
RCU_SPIRST	SPI 时钟
RCU_WWDGTRST	WWDGT 时钟
RCU_UARTxRST(x = 0,1,2,3)	UART 时钟
RCU_I2CRST	I2C 时钟

<i>RCU_SYSCFGRST</i>	系统配置时钟
<i>RCU_PMURST</i>	PMU 时钟
<i>RCU_CFMURST</i>	CFMU 时钟
<i>RCU_DACRST</i>	DAC 时钟
<i>RCU_ADCxRST</i> (x = 0,2)	ADC 时钟
<i>RCU_CMPRST</i>	CMP 时钟
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* disable SPI reset */
```

```
rcu_periph_reset_disable(RCU_SPIRST);
```

### 函数 **rcu\_system\_clock\_source\_config**

函数 **rcu\_system\_clock\_source\_config** 描述见下表:

**表 3-656. 函数 **rcu\_system\_clock\_source\_config****

函数名称	<b>rcu_system_clock_source_config</b>
函数原型	<code>void rcu_system_clock_source_config(uint32_t ck_sys)</code>
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}	
<b>ck_sys</b>	系统时钟源选择
<i>RCU_CKSYSSRC_IRC32M</i>	选择 CK_IRC32M 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
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 3-657. 函数 rcu\_system\_clock\_source\_get

函数名称	rcu_system_clock_source_get
函数原型	uint32_t rcu_system_clock_source_get(void)
功能描述	获取系统时钟源选择状态
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_IRC32M/RCU_SCSS_HXTAL/RCU_SCSS_PLL
RCU_SCSS_IRC32M	CK_IRC32M is selected as the CK_SYS source

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 3-658. 函数 rcu\_ahb\_clock\_config

函数名称	rcu_ahb_clock_config
函数原型	void rcu_ahb_clock_config(uint32_t ck_ahb)
功能描述	配置 AHB 时钟预分频选择
先决条件	-
被调用函数	-
输入参数{in}	
ck_ahb	AHB 预分频选择
RCU_AHB_CKSYS_DIVx (x = 1, 2, 4, 8, 16, 64, 128, 256, 512)	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 3-659. 函数 rcu\_apb1\_clock\_config

函数名称	rcu_apb1_clock_config
函数原型	void rcu_apb1_clock_config(uint32_t ck_apb1)
功能描述	配置 APB1 时钟预分频选择
先决条件	-
被调用函数	-
输入参数{in}	
ck_apb1	APB1 预分频选择
RCU_APB1_CKAHB_D IV1	选择 CK_AHB 为 CK_APB1
RCU_APB1_CKAHB_D IV2	选择 CK_AHB/2 为 CK_APB1
RCU_APB1_CKAHB_D IV4	选择 CK_AHB/4 为 CK_APB1
RCU_APB1_CKAHB_D IV8	选择 CK_AHB/8 为 CK_APB1
RCU_APB1_CKAHB_D IV16	选择 CK_AHB/16 为 CK_APB1
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* configure CK_AHB/16 as CK_APB1 */
```

```
rcu_apb1_clock_config(RCU_APB1_CKAHB_DIV16);
```

### 函数 rcu\_apb2\_clock\_config

函数rcu\_apb2\_clock\_config描述见下表:

表 3-660. 函数 rcu\_apb2\_clock\_config

函数名称	rcu_apb2_clock_config
函数原型	void rcu_apb2_clock_config(uint32_t ck_apb2)

功能描述	配置 APB2 时钟预分频选择
先决条件	-
被调用函数	-
输入参数{in}	
ck_apb2	APB2 预分频选择
RCU_APB2_CKAHB_D IV1	选择 CK_AHB 为 CK_APB2
RCU_APB2_CKAHB_D IV2	选择 CK_AHB / 2 为 CK_APB2
RCU_APB2_CKAHB_D IV4	选择 CK_AHB / 4 为 CK_APB2
RCU_APB2_CKAHB_D IV8	选择 CK_AHB / 8 为 CK_APB2
RCU_APB2_CKAHB_D IV16	选择 CK_AHB / 16 为 CK_APB2
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* configure CK_AHB/8 as CK_APB2 */
```

```
rcu_apb2_clock_config(RCU_APB2_CKAHB_DIV8);
```

### 函数 rcu\_ckout\_config

函数rcu\_ckout\_config描述见下表：

表 3-661. 函数 rcu\_ckout\_config

函数名称	rcu_ckout_config
函数原型	void rcu_ckout_config(uint32_t ckout_src, uint32_t ckout_div)
功能描述	配置 CKOUT 时钟源选择
先决条件	-
被调用函数	-
输入参数{in}	
ckout_src	CKOUT 时钟源选择
RCU_CKOUTSRC_NO NE	无时钟输出
RCU_CKOUTSRC_IRC 32K	选择 IRC32K 时钟
RCU_CKOUTSRC_CK SYS	选择系统时钟

<i>RCU_CKOUTSRC_IRC32M</i>	选择内部高速 32M 晶振
<i>RCU_CKOUTSRC_HXTAL</i>	选择 HXTAL 时钟
<i>RCU_CKOUTSRC_CKPLL_DIV1</i>	选择 CK_PLL 时钟
<i>RCU_CKOUTSRC_CKPLL_DIV8</i>	选择 CK_PLL/8 时钟
输入参数{in}	
<b>ckout_div</b>	CK_OUT 分频
<i>RCU_CKOUT_DIVx</i> (x=1,2,4,8,16,32,64,128)	CK_OUT 除以 x
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 3-662. 函数 rcu\_pll\_config

函数名称	rcu_pll_config
函数原型	void rcu_pll_config(uint32_t pll_src, uint32_t predv_div, 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}	
<b>pll_src</b>	PLL 时钟源选择
<i>RCU_PLLSRC_IRC32M</i>	IRC32M 时钟被选择为 PLL 时钟的时钟源
<i>RCU_PLLSRC_HXTAL</i>	HXTAL 时钟被选择为 PLL 时钟的时钟源
输入参数{in}	
<b>predv_div</b>	PREDV 分频因子
<i>RCU_PREDV_DIVx</i> , x = 1..8	x = 1..8
输入参数{in}	
<b>pll_mul</b>	PLL 时钟倍频因子



<i>RCU_PLL_MULx</i> ( <i>x</i> = 4,4.5,5,5.5...63.5,64)	PLL 时钟 * <i>x</i>
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* configure the PLL */
```

```
rcu_pll_config(RCU_PLLSRC_HXTAL, RCU_PREDV_DIV1, RCU_PLL_MUL10);
```

### 函数 rcu\_system\_reset\_enable

函数rcu\_system\_reset\_enable描述见下表:

表 3-663. 函数 rcu\_system\_reset\_enable

函数名称	rcu_system_reset_enable
函数原型	void rcu_system_reset_enable(uint32_t reset_source)
功能描述	使能 RCU 系统复位
先决条件	-
被调用函数	-
输入参数{in}	
reset_source	复位源
RCU_SYSRST_LOCKUP	CPU lock-up 复位
RCU_SYSRST_LOH	LOH 复位
RCU_SYSRST_LOP	LOP 复位
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* enable RCU system reset */
```

```
rcu_system_reset_enable(RCU_SYSRST_LOCKUP);
```

### 函数 rcu\_system\_reset\_disable

函数rcu\_system\_reset\_disable描述见下表:

表 3-664. 函数 rcu\_system\_reset\_disable

函数名称	rcu_system_reset_disable
函数原型	void rcu_system_reset_disable(uint32_t reset_source)

功能描述	禁能 RCU 系统复位
先决条件	-
被调用函数	-
输入参数{in}	
reset_source	复位源
RCU_SYSRST_LOCKUP	CPU lock-up 复位
RCU_SYSRST_LOH	LOH 复位
RCU_SYSRST_LOP	LOP 复位
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* disable RCU system reset */
```

```
rcu_system_reset_disable(RCU_SYSRST_LOCKUP);
```

### 函数 rcu\_adc\_clock\_config

函数rcu\_adc\_clock\_config描述见下表：

表 3-665. 函数 rcu\_adc\_clock\_config

函数名称	rcu_adc_clock_config
函数原型	void rcu_adc_clock_config(uint32_t ck_adc, uint32_t adc_psc)
功能描述	配置 CK_ADCPRE 时钟源和分频因子
先决条件	-
被调用函数	-
输入参数{in}	
ck_adc	ADC 时钟源选择
RCU_CK_ADCPRE_PCLK2	选择 CK_PCLK2 为 CK_ADC 时钟源
RCU_CK_ADCPRE_HXTAL	选择 CK_HXTAL 为 CK_ADC 时钟源
RCU_CK_ADCPRE_PLL	选择 CK_PLL 为 CK_ADC 时钟源
RCU_CK_ADCPRE_IRC32M	选择 IRC32M 为 CK_ADC 时钟源
输入参数{in}	
adc_psc	ADC 预分频器因子
RCU_CK_ADCPRE_DIVx(x = 2,3...31,32)	ADC 预分频器选择 CK_ADCPRE/ x

输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* configure the ADC prescaler factor */
```

```
rcu_adc_clock_config(RCU_CK_ADCPRE_PLL, RCU_CK_ADCPRE_DIV2);
```

### 函数 rcu\_i2c\_clock\_config

函数rcu\_i2c\_clock\_config描述见下表：

表 3-666. 函数 rcu\_i2c\_clock\_config

函数名称	rcu_i2c_clock_source_config
函数原型	void rcu_i2c_clock_source_config(uint32_t ck_i2c)
功能描述	配置 CK_I2C 时钟源
先决条件	-
被调用函数	-
输入参数{in}	
ck_i2c	i2c 时钟源选择
RCU_CK_I2CSRC_PCLK1	选择 CK_PCLK1 为 CK_I2C 时钟源
RCU_CK_I2CSRC_PLL	选择 CK_PLL 为 CK_I2C 时钟源
RCU_CK_I2CSRC_IRC32M	选择 CK_IRC32M 为 CK_I2C 时钟源
RCU_CK_I2CSRC_HXTAL	选择 CK_HXTAL 为 CK_I2C 时钟源
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* configure the CK_I2C clock source */
```

```
rcu_i2c_clock_source_config(RCU_CK_I2CSRC_PCLK1);
```

### 函数 rcu\_osc\_i2c\_wait

函数rcu\_osc\_i2c\_wait描述见下表：

表 3-667. 函数 rcu\_osc\_i2c\_wait

函数名称	rcu_osc_i2c_wait
------	------------------

函数原型	ErrStatus rcu_osci_stab_wait(rcu_osci_type_enum osci)
功能描述	等待振荡器稳定标志位置位或振荡器起振超时
先决条件	-
被调用函数	-
输入参数{in}	
osci	振荡器类型，参考 <a href="#">表3-647. 枚举类型rcu_osci_type_enum</a>
RCU_HXTAL	外部高速振荡器
RCU_IRC32M	IRC32M 振荡器
RCU_IRC32K	IRC32K 振荡器
RCU_PLL_CK	锁相环时钟
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 3-668. 函数 rcu\_osci\_on

函数名称	rcu_osci_on
函数原形	void rcu_osci_on(rcu_osci_type_enum osci);
功能描述	打开振荡器
先决条件	-
被调用函数	-
输入参数{in}	
osci	振荡器类型，参考 <a href="#">表3-647. 枚举类型rcu_osci_type_enum</a>
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* turn on the high speed crystal oscillator */
rcu_osci_on(RCU_HXTAL);
```

## 函数 rcu\_osc\_off

函数rcu\_osc\_off描述见下表:

表 3-669. 函数 rcu\_osc\_off

函数名称	rcu_osc_off
函数原型	void rcu_osc_off(rcu_osc_type_enum osci)
功能描述	关闭振荡器
先决条件	-
被调用函数	-
输入参数{in}	
osci	振荡器类型, 参考 <a href="#">表 3-647. 枚举类型 rcu_osc_type_enum</a>
RCU_HXTAL	外部高速振荡器
RCU_IRC32M	IRC32M 振荡器
RCU_IRC32K	IRC32K 振荡器
RCU_PLL_CK	锁相环时钟
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 3-670. 函数 rcu\_osc\_bypass\_mode\_enable

函数名称	rcu_osc_bypass_mode_enable
函数原型	void rcu_osc_bypass_mode_enable(void)
功能描述	使能振荡器时钟旁路模式, HXTALEN 应在使能振荡器时钟旁路模式前先复位
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 the high speed crystal oscillator bypass mode */
```

```
rcu_osci_bypass_mode_enable();
```

### 函数 rcu\_osci\_bypass\_mode\_disable

函数rcu\_osci\_bypass\_mode\_disable描述见下表：

表 3-671. 函数 rcu\_osci\_bypass\_mode\_disable

函数名称	rcu_osci_bypass_mode_enable
函数原型	void rcu_osci_bypass_mode_enable(void)
功能描述	禁能振荡器时钟旁路模式
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 high speed crystal oscillator bypass mode */
```

```
rcu_osci_bypass_mode_disable();
```

### 函数 rcu\_hxtal\_frequency\_scale\_select

函数rcu\_hxtal\_frequency\_scale\_select描述见下表：

表 3-672. 函数 rcu\_hxtal\_frequency\_scale\_select

函数名称	rcu_hxtal_frequency_scale_select
函数原型	void rcu_hxtal_frequency_scale_select(uint32_t hxtal_scal)
功能描述	选择 HXTAL 频率范围
先决条件	-
被调用函数	-
输入参数{in}	
hxtal_scal	HXTAL 频率范围
HXTAL_SCALE_1M_T O_10M	HXTAL 范围是 1-10MHz
HXTAL_SCALE_10M_T O_24M	HXTAL 范围是 10-24MHz
输出参数{out}	
-	-
返回值	

-	-
---	---

例如：

```
/* HXTAL frequency scale select */
```

```
rcu_hxtal_frequency_scale_select(HXTAL_SCALE_1M_TO_10M);
```

### 函数 rcu\_irc32m\_adjust\_value\_set

函数rcu\_irc32m\_adjust\_value\_set描述见下表：

表 3-673. 函数 rcu\_irc32m\_adjust\_value\_set

函数名称	rcu_irc32m_adjust_value_set
函数原型	void rcu_irc32m_adjust_value_set(uint32_t irc32m_adjval)
功能描述	设置内部 32MHz RC 振荡器时钟调整值
先决条件	-
被调用函数	-
输入参数{in}	
irc32m_adjval	IRC32M 调整值（0 到 0x1F 之间）
0x00 - 0x1F	IRC32M 调整值
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* set the IRC8M adjust value */
```

```
rcu_irc8m_adjust_value_set(0x10);
```

### 函数 rcu\_hxtal\_clock\_monitor\_enable

函数rcu\_hxtal\_clock\_monitor\_enable描述见下表：

表 3-674. 函数 rcu\_hxtal\_clock\_monitor\_enable

函数名称	rcu_hxtal_clock_monitor_enable
函数原型	void rcu_hxtal_clock_monitor_enable(void)
功能描述	使能 HXTAL 时钟监视器
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 3-675. 函数 rcu\_hxtal\_clock\_monitor\_disable

函数名称	rcu_hxtal_clock_monitor_disable
函数原型	void rcu_hxtal_clock_monitor_disable(void)
功能描述	禁能 HXTAL 时钟监视器
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\_pll\_clock\_monitor\_disable

函数rcu\_pll\_clock\_monitor\_disable描述见下表:

表 3-676. 函数 rcu\_pll\_clock\_monitor\_disable

函数名称	rcu_pll_clock_monitor_disable
函数原型	void rcu_pll_clock_monitor_disable(void)
功能描述	禁能 PLL 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 PLL clock monitor */
```

```
rcu_pll_clock_monitor_disable();
```

### 函数 rcu\_clock\_freq\_get

函数rcu\_clock\_freq\_get描述见下表:

表 3-677. 函数 rcu\_clock\_freq\_get

函数名称	rcu_clock_freq_get
函数原型	uint32_t rcu_clock_freq_get(rcu_clock_freq_enum clock)
功能描述	获取系统时钟、总线频率
先决条件	-
被调用函数	-
输入参数{in}	
clock	要获取的时钟频率, 参考 <a href="#">表 3-647. 枚举类型 rcu_osc_type_enum</a>
CK_SYS	系统时钟
CK_AHB	AHB 时钟
CK_APB1	APB1 时钟
CK_APB2	APB2 时钟
输出参数{out}	
-	-
返回值	
ck_freq	系统时钟/AHB 时钟/APB1 时钟/APB2 时钟频率

例如:

```
uint32_t temp_freq;
```

```
/* get the system clock frequency */
```

```
temp_freq = rcu_clock_freq_get(CK_SYS);
```

### 函数 rcu\_flag\_get

函数rcu\_flag\_get描述见下表:

表 3-678. 函数 rcu\_flag\_get

函数名称	rcu_flag_get
函数原型	FlagStatus rcu_flag_get(rcu_flag_enum flag)
功能描述	获取时钟稳定和外设复位标志
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 <a href="#">表 3-643. 枚举类型 rcu_flag_enum</a>

<i>RCU_FLAG_IRC32MS TB</i>	IRC32M 振荡器稳定标志
<i>RCU_FLAG_HXTALST B</i>	外部高速晶振稳定标志
<i>RCU_FLAG_PLLSTB</i>	PLL 稳定标志
<i>RCU_FLAG_IRC32KST B</i>	IRC32K 振荡器稳定标志
<i>RCU_FLAG_LVD0RST</i>	LVD0 复位标志
<i>RCU_FLAG_LOCKUPR ST</i>	CPU Lockup 复位标志
<i>RCU_FLAG_LVD1RST</i>	LVD1 复位标志
<i>RCU_FLAG_LVD2RST</i>	LVD2 复位标志
<i>RCU_FLAG_LOHRST</i>	LOH 复位标志
<i>RCU_FLAG_LOPRST</i>	LOP 复位标志
<i>RCU_FLAG_V11RST</i>	V1.1 复位标志
<i>RCU_FLAG_OBLRST</i>	选项字节复位标志
<i>RCU_FLAG_EPRST</i>	外部引脚复位标志
<i>RCU_FLAG_PORRST</i>	电源复位标志
<i>RCU_FLAG_SWRST</i>	软件复位标志
<i>RCU_FLAG_FWDGTR ST</i>	独立看门狗复位标志
<i>RCU_FLAG_WWDGTR ST</i>	窗口看门狗复位标志
<i>RCU_FLAG_LPRST</i>	低功耗复位标志
输出参数{out}	
-	-
返回值	
-	-

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 3-679. 函数 rcu\_all\_reset\_flag\_clear

函数名称	rcu_all_reset_flag_clear
函数原型	void rcu_all_reset_flag_clear(void)
功能描述	清除所有复位标志位

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 3-680. 函数 rcu\_interrupt\_flag\_get

函数名称	rcu_interrupt_flag_get
函数原型	FlagStatus rcu_interrupt_flag_get(rcu_int_flag_enum int_flag)
功能描述	获取时钟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 <a href="#">表 3-644. 枚举类型 rcu_int_flag_enum</a>
RCU_INT_FLAG_IRC3 2KSTB	IRC32K 时钟稳定中断标志
RCU_INT_FLAG_IRC3 2MSTB	IRC32M 时钟稳定中断标志
RCU_INT_FLAG_HXTA LSTB	外部高速晶振时钟稳定中断标志
RCU_INT_FLAG_PLLS TB	PLL 时钟稳定中断标志
RCU_INT_FLAG_CKM	PLL 时钟监视器中断标志
RCU_INT_FLAG_PLLM	外部高速晶振时钟阻塞中断标志
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 3-681. 函数 rcu\_interrupt\_flag\_clear

函数名称	rcu_interrupt_flag_clear
函数原型	void rcu_interrupt_flag_clear(rcu_int_flag_clear_enum int_flag)
功能描述	clear the interrupt flags
先决条件	-
被调用函数	-
输入参数{in}	
int_flag	时钟稳定和阻塞中断标志清除，参考 <a href="#">表 3-645. 枚举类型 rcu_int_flag_clear_enum</a>
RCU_INT_FLAG_IRC3 2KSTB_CLR	IRC32K 时钟稳定中断清除标志
RCU_INT_FLAG_IRC3 2MSTB_CLR	IRC32M 时钟稳定中断清除标志
RCU_INT_FLAG_HXTA LSTB_CLR	外部高速晶振时钟稳定中断清除标志
RCU_INT_FLAG_PLLS TB_CLR	PLL 时钟稳定中断清除标志
RCU_INT_FLAG_CKM _CLR	清除 PLL 时钟监视器中断
RCU_INT_FLAG_PLLM _CLR	外部高速晶振时钟阻塞中断清除标志
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* clear the interrupt HXTAL stabilization interrupt flag */
rcu_interrupt_flag_clear(RCU_INT_FLAG_HXTALSTB_CLR);
```

### 函数 rcu\_interrupt\_enable

函数rcu\_interrupt\_enable描述见下表：

表 3-682. 函数 rcu\_interrupt\_enable

函数名称	rcu_interrupt_enable
------	----------------------

函数原型	void rcu_interrupt_enable(rcu_int_enum interrupt)
功能描述	enable the stabilization interrupt
先决条件	-
被调用函数	-
输入参数{in}	
interrupt	时钟稳定中断, 参考 <a href="#">表 3-646. 枚举类型 rcu_int_enum</a>
RCU_INT_IRC32KSTB	IRC32K 时钟稳定中断
RCU_INT_IRC32MSTB	IRC32M 时钟稳定中断
RCU_INT_HXTALSTB	外部高速晶振时钟稳定中断
RCU_INT_PLLSTB	PLL 时钟稳定中断
RCU_INT_PLLM	PLL 监视器中断
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 3-683. 函数 rcu\_interrupt\_disable

函数名称	rcu_interrupt_disable
函数原型	void rcu_interrupt_disable(rcu_int_enum interrupt)
功能描述	disable the stabilization interrupt
先决条件	-
被调用函数	-
输入参数{in}	
interrupt	时钟稳定中断, 参考 <a href="#">表 3-646. 枚举类型 rcu_int_enum</a>
RCU_INT_IRC32KSTB	IRC32K 时钟稳定中断
RCU_INT_IRC32MSTB	IRC32M 时钟稳定中断
RCU_INT_HXTALSTB	外部高速晶振时钟稳定中断
RCU_INT_PLLSTB	PLL 时钟稳定中断
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* disable the HXTAL stabilization interrupt */
```

```
rcu_interrupt_disable(RCU_INT_HXTALSTB);
```

## 3.24. SPI

SPI模块可以通过SPI协议与外部设备进行通信。SPI寄存器在[3.24.1](#)章中列出，SPI固件函数在[3.24.2](#)章中介绍。

### 3.24.1. 外设寄存器说明

SPI/I2S寄存器列表如下表所示：

**表 3-684. SPI/I2S 寄存器**

寄存器名称	寄存器描述
SPI_CTL0	控制寄存器0
SPI_CTL1	控制寄存器1
SPI_STAT	状态寄存器
SPI_DATA	数据寄存器
SPI_CRCPOLY	CRC多项式寄存器
SPI_RCRC	接收CRC寄存器
SPI_TCRC	发送CRC寄存器
SPI_QCTL	四线SPI控制寄存器

### 3.24.2. 外设库函数说明

SPI/I2S库函数列表如下表所示：

**表 3-685. SPI/I2S 库函数**

库函数名称	库函数描述
spi_deinit	复位外设SPI
spi_struct_para_init	将SPI结构体中所有参数初始化为默认值
spi_init	初始化外设SPI
spi_enable	使能外设SPI
spi_disable	失能外设SPI
spi_nss_output_enable	使能外设SPINSS输出
spi_nss_output_disable	失能外设SPINSS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_data_frame_format_config	配置外设SPI数据帧格式
spi_data_transmit	发送数据

库函数名称	库函数描述
spi_data_receive	接收数据
spi_bidirectional_transfer_config	配置外设SPI的数据传输方向
spi_format_error_clear	清除TI模式格式错误标志
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_on	打开外设SPI的CRC功能
spi_crc_off	关闭外设SPI的CRC功能
spi_crc_next	设置外设SPI下一次传输数据为CRC值
spi_crc_get	外设SPI获取CRC值
spi_crc_error_clear	清除SPI CRC错误标志状态
spi_ti_mode_enable	使能SPI TI模式
spi_ti_mode_disable	禁能SPI TI模式
spi_nssp_mode_enable	使能SPI NSS脉冲模式
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_flag_get	获取外设SPI标志状态
spi_interrupt_enable	使能外设SPI中断
spi_interrupt_disable	失能外设SPI中断
spi_interrupt_flag_get	获取外设SPI中断状态

### 结构体 spi\_parameter\_struct

表 3-686. 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_16BIT, SPI_FRAME_SIZE_8BIT)
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_CLOCK_POLARITY_LOW_PHASE_1EDGE, SPI_CLOCK_POLARITY_HIGH_PHASE_1EDGE, SPI_CLOCK_POLARITY_LOW_PHASE_2EDGE, SPI_CLOCK_POLARITY_HIGH_PHASE_2EDGE)

prescale	预分频器配置 (SPI_PSC_n (n=2,4,8,16,32,64,128,256))
----------	--

### 函数 spi\_deinit

函数spi\_deinit描述见下表:

表 3-687. 函数 spi\_deinit

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

例如:

```
/* reset SPI */
```

```
spi_deinit();
```

### 函数 spi\_struct\_para\_init

函数spi\_struct\_para\_init描述见下表:

表 3-688. 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	
先决条件	-
被调用函数	-
输入参数{in}	
*spi_struct	一个已经定义的spi_parameter_struct结构体变量地址
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 3-689. 函数 spi\_init**

函数名称	spi_init
函数原形	void spi_init(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_struct	初始化结构体，结构体成员参考 <a href="#">表3-686. 结构体spi_parameter_struct</a>
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* initialize SPI */
```

```
spi_parameter_struct spi_init_struct;
```

```
spi_init_struct.trans_mode      = SPI_TRANSMODE_BDTRANSMIT;
```

```
spi_init_struct.device_mode     = SPI_MASTER;
```

```
spi_init_struct.frame_size     = SPI_FRAME_SIZE_8BIT;
```

```
spi_init_struct.clock_polarity_phase = SPI_CLOCK_POLARITY_HIGH_PHASE_2EDGE;
```

```
spi_init_struct.nss             = SPI_NSS_SOFT;
```

```
spi_init_struct.prescale       = SPI_PSC_8;
```

```
spi_init_struct.endian         = SPI_ENDIAN_MSB;
```

```
spi_init(&spi_init_struct);
```

### 函数 spi\_enable

函数spi\_enable描述见下表：

**表 3-690. 函数 spi\_enable**

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

功能描述	使能外设SPI
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 SPI */
```

```
spi_enable();
```

### 函数 spi\_disable

函数spi\_disable描述见下表：

表 3-691. 函数 spi\_disable

函数名称	spi_disable
函数原形	void spi_disable(void);
功能描述	禁能外设SPI
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 SPI */
```

```
spi_disable();
```

### 函数 spi\_nss\_output\_enable

函数spi\_nss\_output\_enable描述见下表：

表 3-692. 函数 spi\_nss\_output\_enable

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

例如：

```
/* enable SPI NSS output */
```

```
spi_nss_output_enable();
```

### 函数 spi\_nss\_output\_disable

函数spi\_nss\_output\_disable描述见下表：

表 3-693. 函数 spi\_nss\_output\_disable

函数名称	spi_nss_output_disable
函数原形	void spi_nss_output_disable(void);
功能描述	禁能外设SPI NSS输出
先决条件	-
被调用函数	-
输入参数{in}	
-	-
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* disable SPI NSS output */
```

```
spi_nss_output_disable();
```

### 函数 spi\_nss\_internal\_high

函数spi\_nss\_internal\_high描述见下表：

表 3-694. 函数 spi\_nss\_internal\_high

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

例如：

```
/* SPI NSS pin is pulled high level in software mode */
```

```
spi_nss_internal_high();
```

### 函数 spi\_nss\_internal\_low

函数spi\_nss\_internal\_low描述见下表：

表 3-695. 函数 spi\_nss\_internal\_low

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

例如：

```
/* SPI NSS pin is pulled low level in software mode */
```

```
spi_nss_internal_low();
```

### 函数 spi\_dma\_enable

函数spi\_dma\_enable描述见下表：

表 3-696. 函数 spi\_dma\_enable

函数名称	spi_dma_enable
函数原形	void spi_dma_enable(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}	
<b>dma</b>	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 SPI transmit data DMA function */
```

```
spi_dma_enable(SPI_DMA_TRANSMIT);
```

### 函数 spi\_dma\_disable

函数spi\_dma\_disable描述见下表：

**表 3-697. 函数 spi\_dma\_disable**

函数名称	spi_dma_disable
函数原形	void spi_dma_disable(uint8_t dma);
功能描述	禁能外设SPI的DMA功能
先决条件	-
被调用函数	-
输入参数{in}	
<b>dma</b>	SPI DMA模式
<i>SPI_DMA_TRANSMIT</i>	SPI发送缓冲区DMA使能
<i>SPI_DMA_RECEIVE</i>	SPI接收缓冲区DMA使能
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* disable SPI transmit data DMA function */
```

```
spi_dma_disable(SPI_DMA_TRANSMIT);
```

## 函数 spi\_data\_frame\_format\_config

函数spi\_data\_frame\_format\_config描述见下表：

表 3-698. 函数 spi\_data\_frame\_format\_config

函数名称	spi_data_frame_format_config
函数原形	void spi_data_frame_format_config(uint16_t frame_format);
功能描述	配置外设SPI数据帧格式
先决条件	-
被调用函数	-
输入参数{in}	
frame_format	SPI帧大小
SPI_FRAME_SIZE_16BIT	SPI 16位数据帧格式
SPI_FRAME_SIZE_8BIT	SPI 8位数据帧格式
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* configure SPI data frame format size is 16 bits */
```

```
spi_data_frame_format_config(SPI_FRAME_SIZE_16BIT);
```

## 函数 spi\_data\_transmit

函数spi\_data\_transmit描述见下表：

表 3-699. 函数 spi\_data\_transmit

函数名称	spi_data_transmit
函数原形	void spi_data_transmit(uint16_t data);
功能描述	SPI发送数据
先决条件	-
被调用函数	-
输入参数{in}	
data	16位数据
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* SPI transmit data */
```

```
spi_data_transmit(SPI_send_array[send_n]);
```

### 函数 spi\_data\_receive

函数spi\_data\_receive描述见下表:

表 3-700. 函数 spi\_data\_receive

函数名称	spi_data_receive
函数原形	uint16_t spi_data_receive(void);
功能描述	SPI接收数据
先决条件	-
被调用函数	-
输入参数{in}	
-	-
输出参数{out}	
-	-
返回值	
-	16位数据

例如:

```
/* SPI receive data */
```

```
SPI_receive_array[receive_n] = spi_data_receive();
```

### 函数 spi\_bidirectional\_transfer\_config

函数spi\_bidirectional\_transfer\_config描述见下表:

表 3-701. 函数 spi\_bidirectional\_transfer\_config

函数名称	spi_bidirectional_transfer_config
函数原形	void spi_bidirectional_transfer_config(uint32_t transfer_direction);
功能描述	配置外设SPI的数据传输方向
先决条件	-
被调用函数	-
输入参数{in}	
transfer_direction	SPI双向传输输出使能
SPI_BIDIRECTIONAL_TRANSMIT	SPI工作在只发送模式
SPI_BIDIRECTIONAL_RECEIVE	SPI工作在只接收模式
输出参数{out}	
-	-
返回值	

-	-
---	---

例如:

```
/* SPI works in transmit-only mode */
```

```
spi_bidirectional_transfer_config(SPI_BIDIRECTIONAL_TRANSMIT);
```

### 函数 spi\_format\_error\_clear

函数spi\_format\_error\_clear描述见下表:

表 3-702. 函数 spi\_format\_error\_clear

函数名称	spi_format_error_clear
函数原形	void spi_format_error_clear(void);
功能描述	配置
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 TI Mode Format Error flag status */
```

```
spi_format_error_clear();
```

### 函数 spi\_crc\_polynomial\_set

函数spi\_crc\_polynomial\_set描述见下表:

表 3-703. 函数 spi\_crc\_polynomial\_set

函数名称	spi_crc_polynomial_set
函数原形	void spi_crc_polynomial_set(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}	
crc_poly	CRC多项式值
输出参数{out}	
-	-
返回值	
-	-



例如：

```
/* set SPI CRC polynomial */
```

```
spi_crc_polynomial_set(CRC_VALUE);
```

### 函数 spi\_crc\_polynomial\_get

函数spi\_crc\_polynomial\_get描述见下表：

表 3-704. 函数 spi\_crc\_polynomial\_get

函数名称	spi_crc_polynomial_get
函数原形	uint16_t spi_crc_polynomial_get(void);
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}	
-	-
输出参数{out}	
-	-
返回值	
uint16_t	16位CRC多项式值（0-0xFFFF）

例如：

```
/* get SPI CRC polynomial */
```

```
uint16_t crc_val;
```

```
crc_val = spi_crc_polynomial_get();
```

### 函数 spi\_crc\_on

函数spi\_crc\_on描述见下表：

表 3-705. 函数 spi\_crc\_on

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

例如：

```
/* turn on SPI CRC function */
```

```
spi_crc_on();
```

### 函数 spi\_crc\_off

函数spi\_crc\_off描述见下表：

**表 3-706. 函数 spi\_crc\_off**

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

例如：

```
/* turn off SPI CRC function */
```

```
spi_crc_off();
```

### 函数 spi\_crc\_next

函数spi\_crc\_next描述见下表：

**表 3-707. 函数 spi\_crc\_next**

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

例如：

```
/* SPI next data is CRC value */
```

```
spi_crc_next();
```

### 函数 spi\_crc\_get

函数spi\_crc\_get描述见下表:

表 3-708. 函数 spi\_crc\_get

函数名称	spi_crc_get
函数原形	uint16_t spi_crc_get(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}	
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 SPI CRC send value */
```

```
uint16_t crc_val;
```

```
crc_val = spi_crc_get(SPI_CRC_TX);
```

### 函数 spi\_crc\_error\_clear

函数spi\_crc\_error\_clear描述见下表:

表 3-709. 函数 spi\_crc\_error\_clear

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

-	-
---	---

例如：

```
/* clear SPI CRC error flag status */
```

```
spi_crc_error_clear();
```

### 函数 spi\_ti\_mode\_enable

函数spi\_ti\_mode\_enable描述见下表：

表 3-710. 函数 spi\_ti\_mode\_enable

函数名称	spi_ti_mode_enable
函数原形	void spi_ti_mode_enable(void);
功能描述	使能SPI TI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 SPI TI mode */
```

```
spi_ti_mode_enable();
```

### 函数 spi\_ti\_mode\_disable

函数spi\_ti\_mode\_disable描述见下表：

表 3-711. 函数 spi\_ti\_mode\_disable

函数名称	spi_ti_mode_disable
函数原形	void spi_ti_mode_disable(void);
功能描述	禁能SPI TI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 SPI TI mode */  
  
spi_ti_mode_disable();
```

### 函数 spi\_nssp\_mode\_enable

函数spi\_nssp\_mode\_enable描述见下表：

表 3-712. 函数 spi\_nssp\_mode\_enable

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

例如：

```
/* enable SPI NSS pulse mode */  
  
spi_nssp_mode_enable();
```

### 函数 spi\_nssp\_mode\_disable

函数spi\_nssp\_mode\_disable描述见下表：

表 3-713. 函数 spi\_nssp\_mode\_disable

函数名称	spi_nssp_mode_disable
函数原形	void spi_nssp_mode_disable(void);
功能描述	禁能SPI NSS脉冲模式
先决条件	-
被调用函数	-
输入参数{in}	
-	-
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* disable SPI NSS pulse mode */
```

```
spi_nssp_mode_disable();
```

### 函数 spi\_quad\_enable

函数spi\_quad\_enable描述见下表:

表 3-714. 函数 spi\_quad\_enable

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

例如:

```
/* enable SPI quad wire mode */
```

```
spi_quad_enable();
```

### 函数 spi\_quad\_disable

函数spi\_quad\_disable描述见下表:

表 3-715. 函数 spi\_quad\_disable

函数名称	spi_quad_disable
函数原形	spi_quad_disable(void);
功能描述	禁能四线SPI模式
先决条件	-
被调用函数	-
输入参数{in}	
-	-
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* disable SPI quad wire mode */
```

```
spi_quad_disable();
```

### 函数 `spi_quad_write_enable`

函数 `spi_quad_write_enable` 描述见下表:

表 3-716. 函数 `spi_quad_write_enable`

函数名称	<code>spi_quad_write_enable</code>
函数原形	<code>void spi_quad_write_enable(void);</code>
功能描述	使能四线SPI写
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 SPI quad wire write */
```

```
spi_quad_write_enable();
```

### 函数 `spi_quad_read_enable`

函数 `spi_quad_read_enable` 描述见下表:

表 3-717. 函数 `spi_quad_read_enable`

函数名称	<code>spi_quad_read_enable</code>
函数原形	<code>void spi_quad_read_enable(void);</code>
功能描述	使能四线SPI读
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 SPI quad wire read */
```

```
spi_quad_read_enable();
```

**函数 spi\_flag\_get**

函数spi\_flag\_get描述见下表：

**表 3-718. 函数 spi\_flag\_get**

函数名称	spi_flag_get
函数原形	FlagStatus spi_flag_get(uint32_t flag);
功能描述	获取外设SPI标志状态
先决条件	-
被调用函数	-
输入参数{in}	
flag	SPI/I2S标志状态
SPI_FLAG_TBE	发送缓冲区空标志
SPI_FLAG_RBNE	接收缓冲区非空标志
SPI_FLAG_TRANS	通信进行中标志
SPI_INT_FLAG_RXORERR	接收过载错误标志
SPI_FLAG_CONFERR	配置错误标志
SPI_FLAG_CRCERR	CRC错误标志
I2S_FLAG_RXORERR	接收过载错误标志
I2S_FLAG_TXURERR	发送欠载错误标志
I2S_FLAG_CH	通道标志
输出参数{out}	
-	-
返回值	
FlagStatus	SET或RESET

例如：

```
/* get SPI transmit buffer empty flag status */
while(RESET == spi_flag_get(SPI_FLAG_TBE));
spi_data_transmit(SPI_send_array[send_n++]);
```

**函数 spi\_interrupt\_enable**

函数spi\_interrupt\_enable描述见下表：

**表 3-719. 函数 spi\_interrupt\_enable**

函数名称	spi_interrupt_enable
函数原形	void spi_interrupt_enable(uint32_t interrupt);



功能描述	使能外设SPI中断
先决条件	-
被调用函数	-
输入参数{in}	
interrupt	SPI/I2S中断
SPI_INT_TBE	发送缓冲区空中断使能
SPI_INT_RBNE	接收缓冲区非空中断使能
SPI_INT_ERR	错误中断使能
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* enable SPI transmit buffer empty interrupt */
```

```
spi_interrupt_enable(SPI_INT_TBE);
```

### 函数 spi\_interrupt\_disable

函数spi\_interrupt\_disable描述见下表：

表 3-720. 函数 spi\_interrupt\_disable

函数名称	spi_interrupt_disable
函数原形	void spi_interrupt_disable(uint32_t interrupt);
功能描述	禁能外设SPI中断
先决条件	-
被调用函数	-
输入参数{in}	
interrupt	SPI/I2S中断
SPI_INT_TBE	发送缓冲区空中断使能
SPI_INT_RBNE	接收缓冲区非空中断使能
SPI_INT_ERR	错误中断使能
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* disable SPI transmit buffer empty interrupt */
```

```
spi_interrupt_disable(SPI_INT_TBE);
```

**函数 spi\_interrupt\_flag\_get**

函数spi\_interrupt\_flag\_get描述见下表：

**表 3-721. 函数 spi\_interrupt\_flag\_get**

函数名称	spi_interrupt_flag_get
函数原形	FlagStatus spi_interrupt_flag_get(uint8_t int_flag);
功能描述	获取外设SPI中断状态
先决条件	-
被调用函数	-
输入参数{in}	
int_flag	SPI/I2S中断状态
SPI_INT_FLAG_TB E	发送缓冲区空中断
SPI_INT_FLAG_RB NE	接收缓冲区非空中断
SPI_INT_FLAG_RX ORERR	接收过载错误中断
SPI_INT_FLAG_CO NFERR	配置错误中断
SPI_INT_FLAG_CR CERR	CRC错误中断
I2S_INT_FLAG_TX URERR	发送欠载错误中断
输出参数{out}	
-	-
返回值	
FlagStatus	SET 或 RESET

例如：

```

/* get SPI transmit buffer empty interrupt status */

if(RESET != spi_interrupt_flag_get(SPI_INT_FLAG_TBE)){
    while(RESET == spi_flag_get(SPI_FLAG_TBE));
    spi_data_transmit(SPI_send_array[send_n++]);
}

```

**3.25. SVPWM**

章节[3.26.1](#)描述了SVPWM的寄存器列表，章节[3.26.2](#)对SVPWM库函数进行说明。

### 3.25.1. 外设寄存器说明

SVPWM寄存器列表如下表所示：

表 3-722. SYSCFG 寄存器

寄存器名称	寄存器描述
SVPWM_CTL0	控制寄存器0
SVPWM_UALPHA	Ualpha寄存器
SVPWM_UBETA	Ubeta寄存器
SVPWM_CAR	自动重载寄存器
SVPWM_TA	Ta寄存器
SVPWM_TB	Tb寄存器
SVPWM_TC	Tc寄存器
SVPWM_SEC	扇区寄存器
SVPWM_STAT	状态寄存器

### 3.25.2. 外设库函数说明

SVPWM库函数列表如下表所示：

表 3-723. SVPWM 库函数

库函数名称	库函数描述
svpwm_deinit	初始化SVPWM结构体
svpwm_init	配置SVPWM
svpwm_enable	使能SVPWM
svpwm_flag_get	获取SVPWM标志位
svpwm_alpha_beta_write	写 alpha beta 值
svpwm_ta_tb_tc_read	读取ta、tb、tc值
svpwm_sector_read	读取扇区值

#### 结构体 svpwm\_parameter\_struct

表 3-724. 结构体 svpwm\_parameter\_struct

成员名称	功能描述
switch_mode	SVPWM开关模式(SVPWM_SWITCH_MODE0, SVPWM_SWITCH_MODE1)
working_mode	SVPWM 模式定义(SVPWM_MODE_SEVEN_SEGMENT, SVPWM_MODE_FIVE_SEGMENT)
period_count	周期值

#### 函数 svpwm\_deinit

函数svpwm\_deinit描述见下表：

表 3-725. 函数 `svpwm_deinit`

函数名称	<code>syscfg_deinit</code>
函数原形	<code>void svpwm_deinit(void);</code>
功能描述	复位SVPWM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 SVPWM registers */
svpwm_deinit();
```

### 函数 `svpwm_init`

函数`svpwm_init`描述见下表：

表 3-726. 函数 `svpwm_init`

函数名称	<code>svpwm_init</code>
函数原型	<code>void svpwm_init(svpwm_parameter_struct *init_struct);</code>
功能描述	配置SVPWM
先决条件	-
被调用函数	-
输入参数{in}	
<code>init_struct</code>	参考 <a href="#">结构体<code>svpwm_parameter_struct</code></a>
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* initialize SVPWM */
svpwm_parameter_struct init_struct;
svpwm_init (&init_struct);
```

### 函数 `svpwm_enable`

函数`svpwm_enable`描述见下表：

表 3-727. 函数 `svpwm_enable`

函数名称	<code>svpwm_enable</code>
函数原形	<code>void svpwm_enable(void);</code>
功能描述	使能SVPWM
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 SVPWM registers */
svpwm_enable();
```

### 函数 `svpwm_flag_get`

函数`svpwm_flag_get`描述见下表：

表 3-728. 函数 `svpwm_flag_get`

函数名称	<code>svpwm_flag_get</code>
函数原型	<code>FlagStatus svpwm_flag_get(uint32_t flag);</code>
功能描述	获取外设SVPWM的状态标志
先决条件	-
被调用函数	-
输入参数{in}	
<b>flag</b>	状态标志
<code>SVPWM_FLAG_OA_L</code>	A相占空比小于0下溢标志位
<code>SVPWM_FLAG_OB_L</code>	B相占空比小于0下溢标志位
<code>SVPWM_FLAG_OC_L</code>	C相占空比小于0下溢标志位
<code>SVPWM_FLAG_OA_H</code>	A相占空比大于COUNT上溢标志位
<code>SVPWM_FLAG_OB_H</code>	B相占空比大于COUNT上溢标志位
<code>SVPWM_FLAG_OC_H</code>	C相占空比大于COUNT上溢标志位
<code>SVPWM_FLAG_OS</code>	SVPWM运行状态标志位

<i>F</i>	
输出参数{out}	
-	-
返回值	
FlagStatus	SET或者RESET

例如：

```
/* get SVPWM status flag */
```

```
FlagStatus Flag_status = RESET;
```

```
Flag_status = svpwm_flag_get (SVPWM_FLAG_OAL);
```

### 函数 svpwm\_alpha\_beta\_write

函数svpwm\_alpha\_beta\_write描述见下表：

表 3-729. 函数 svpwm\_alpha\_beta\_write

函数名称	svpwm_alpha_beta_write
函数原型	void svpwm_alpha_beta_write(float alpha, float beta);
功能描述	写入alpha beta数据
先决条件	-
被调用函数	-
输入参数{in}	
alpha	电压alpha值
输入参数{in}	
beta	电压beta值
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* write alpha and beta data in iq format */
```

```
svpwm_alpha_beta_write (1000,1000);
```

### 函数 svpwm\_ta\_tb\_tc\_read

函数svpwm\_ta\_tb\_tc\_read描述见下表：

表 3-730. 函数 svpwm\_ta\_tb\_tc\_read

函数名称	svpwm_ta_tb_tc_read
函数原型	void svpwm_ta_tb_tc_read(uint16_t *ta, uint16_t *tb, uint16_t *tc);
功能描述	读取 ta tb tc.

先决条件	-
被调用函数	-
输入参数{in}	
*ta	ta的地址
输入参数{in}	
*tb	tb的地址
输入参数{in}	
*tc	tc的地址
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* read ta, tb, tc data in uint16 format */
uint16_t ta,tb,tc;
svpwm_ta_tb_tc_read (&ta,&tb,&tc);
```

### 函数 svpwm\_sector\_read

函数svpwm\_sector\_read描述见下表：

表 3-731. 函数 svpwm\_sector\_read

函数名称	svpwm_sector_read
函数原形	uint8_t svpwm_sector_read();
功能描述	读取扇区
先决条件	-
被调用函数	-
输入参数{in}	
-	-
输出参数{out}	
-	-
返回值	
uint8_t	扇区值

例如：

```
/* read sector data */
uint8_t sector;
sector = svpwm_sector_read();
```

## 3.26. SYSCFG

章节 [3.26.1](#) 描述了SYSCFG的寄存器列表，章节 [3.26.2](#) 对SYSCFG库函数进行说明。

### 3.26.1. 外设寄存器说明

SYSCFG寄存器列表如下表所示：

**表 3-732. SYSCFG 寄存器**

寄存器名称	寄存器描述
SYSCFG_CFG0	配置寄存器0
SYSCFG_CFG1	配置寄存器1
SYSCFG_EXTISS0	EXTI源选择寄存器0
SYSCFG_EXTISS1	EXTI源选择寄存器1
SYSCFG_EXTISS2	EXTI源选择寄存器2
SYSCFG_EXTISS3	EXTI源选择寄存器3
SYSCFG_STAT	状态寄存器
SYSCFG_CFG2	配置寄存器2
SYSCFG_CFG3	配置寄存器3
SYSCFG_CFG4	配置寄存器4
SYSCFG_TIMERCISEL0	TIMER输入源选择寄存器0
SYSCFG_TIMERCISEL1	TIMER输入源选择寄存器1
SYSCFG_CFG5	配置寄存器5
SYSCFG_SRAMCFG	SRAM配置寄存器
SYSCFG_PRCFG	保护配置寄存器
SYSCFG_TADSRCFG	TIMERx ADC启动请求配置寄存器
SYSCFG_TIMER0TRGCFG0	TIMERx配置寄存器0, x=0
SYSCFG_TIMER0TRGCFG1	TIMERx配置寄存器1, x=0
SYSCFG_TIMER0TRGCFG2	TIMERx配置寄存器2, x=0
SYSCFG_TIMER1TRGCFG0	TIMERx配置寄存器0, x=1
SYSCFG_TIMER1TRGCFG1	TIMERx配置寄存器1, x=1
SYSCFG_TIMER1TRGCFG2	TIMERx配置寄存器2, x=1
SYSCFG_TIMER2TRGCFG0	TIMERx配置寄存器0, x=2
SYSCFG_TIMER2TRGCFG1	TIMERx配置寄存器1, x=2
SYSCFG_TIMER2TRGCFG2	TIMERx配置寄存器2, x=2
SYSCFG_TIMER7TRGCFG0	TIMERx配置寄存器0, x=7
SYSCFG_TIMER7TRGCFG1	TIMERx配置寄存器1, x=7
SYSCFG_TIMER7TRGCFG2	TIMERx配置寄存器2, x=7
SYSCFG_CFG6	配置寄存器6



## 3.26.2. 外设库函数说明

SYSCFG库函数列表如下表所示：

表 3-733. SYSCFG 库函数

库函数名称	库函数描述
syscfg_deinit	复位SYSCFG寄存器
syscfg_i2c_fast_mode_plus_enable	使能I2C Fm+模式
syscfg_i2c_fast_mode_plus_disable	禁能I2C Fm+模式
syscfg_exti_line_config	配置GPIO引脚作为EXTI
syscfg_timer_input_source_select	选择TIMER输入源
syscfg_lockup_enable	使能模块锁定
syscfg_sram_remap_set	配置SRAM重映射大小
syscfg_sram_remap_size_get	获取SRAM重映射大小
syscfg_ck_cmp_sel_set	配置CK_CMP选择
syscfg_ck_cmp_sel_get	获取CK_CMP选择
syscfg_register_write_enable	寄存器写使能
syscfg_register_write_disable	寄存器写禁能
syscfg_adc_signal_monitor_output_enable	使能ADC信号监控输出
syscfg_adc_signal_monitor_output_disable	禁能ADC信号监控输出
syscfg_adc_signal_monitor_select	配置ADC信号监控输出
syscfg_boot_mode_get	获取启动模式
syscfg_sram_ecc_error_address_get	获取SRAM ECC错误地址
syscfg_sram_ecc_error_bit_get	获取SRAM ECC单比特纠错位
syscfg_interrupt_enable	使能SYSCFG外设中断
syscfg_interrupt_disable	除能SYSCFG外设中断
syscfg_interrupt_flag_get	获取中断标志
syscfg_interrupt_flag_clear	清除中断标志

## 枚举类型 exti\_gpio\_enum

表 3-734. 枚举类型 exti\_gpio\_enum

枚举名称	枚举描述
EXTI_SOURCE_EXTI0_PA8	EXTI0 PA8
EXTI_SOURCE_EXTI0_PC10	EXTI0 PC10
EXTI_SOURCE_EXTI0_PN2	EXTI0 PN2
EXTI_SOURCE_EXTI0_PC8	EXTI0 PC8
EXTI_SOURCE_EXTI1_PA9	EXTI1 PA9
EXTI_SOURCE_EXTI1_PC11	EXTI1 PC11
EXTI_SOURCE_EXTI1_PE13	EXTI1 PE13
EXTI_SOURCE_EXTI1_PC9	EXTI1 PC9
EXTI_SOURCE_EXTI2_PA0	EXTI2 PA0

枚举名称	枚举描述
EXTI_SOURCE_EXTI2_PC12	EXTI2 PC12
EXTI_SOURCE_EXTI2_PF14	EXTI2 PF14
EXTI_SOURCE_EXTI2_PG12	EXTI2 PG12
EXTI_SOURCE_EXTI3_PF12	EXTI3 PF12
EXTI_SOURCE_EXTI4_PA1	EXTI4 PA1
EXTI_SOURCE_EXTI4_PE14	EXTI4 PE14
EXTI_SOURCE_EXTI4_PF9	EXTI4 PF9
EXTI_SOURCE_EXTI4_PF13	EXTI4 PF13
EXTI_SOURCE_EXTI5_PD8	EXTI5 PD8
EXTI_SOURCE_EXTI5_PG14	EXTI5 PG14
EXTI_SOURCE_EXTI5_PN7	EXTI5 PN7
EXTI_SOURCE_EXTI6_PB1	EXTI6 PB1
EXTI_SOURCE_EXTI6_PC6	EXTI6 PC6
EXTI_SOURCE_EXTI6_PD2	EXTI6 PD2
EXTI_SOURCE_EXTI6_PG13	EXTI6 PG13
EXTI_SOURCE_EXTI7_PB0	EXTI7 PB0
EXTI_SOURCE_EXTI7_PC7	EXTI7 PC7
EXTI_SOURCE_EXTI7_PN5	EXTI7 PN5
EXTI_SOURCE_EXTI8_PD4	EXTI8 PD4
EXTI_SOURCE_EXTI8_PE8	EXTI8 PE8
EXTI_SOURCE_EXTI8_PF8	EXTI8 PF8
EXTI_SOURCE_EXTI8_PG15	EXTI8 PG15
EXTI_SOURCE_EXTI9_PD5	EXTI9 PD5
EXTI_SOURCE_EXTI9_PD9	EXTI9 PD9
EXTI_SOURCE_EXTI9_PE9	EXTI9 PE9
EXTI_SOURCE_EXTI9_PF11	EXTI9 PF11
EXTI_SOURCE_EXTI10_PB2	EXTI10 PB2
EXTI_SOURCE_EXTI10_PC0	EXTI10 PC0
EXTI_SOURCE_EXTI10_PD10	EXTI10 PD10
EXTI_SOURCE_EXTI10_PE10	EXTI10 PE10
EXTI_SOURCE_EXTI10_PF10	EXTI10 PF10
EXTI_SOURCE_EXTI11_PC1	EXTI11 PC1
EXTI_SOURCE_EXTI11_PD11	EXTI11 PD11
EXTI_SOURCE_EXTI11_PG11	EXTI11 PG11
EXTI_SOURCE_EXTI12_PC2	EXTI12 PC2
EXTI_SOURCE_EXTI12_PD12	EXTI12 PD12
EXTI_SOURCE_EXTI12_PE12	EXTI12 PE12
EXTI_SOURCE_EXTI13_PC3	EXTI13 PC3
EXTI_SOURCE_EXTI13_PD13	EXTI13 PD13
EXTI_SOURCE_EXTI14_PB14	EXTI14 PB14

枚举名称	枚举描述
EXTI_SOURCE_EXTI14_PC4	EXTI14 PC4
EXTI_SOURCE_EXTI14_PD14	EXTI14 PD14
EXTI_SOURCE_EXTI14_PE11	EXTI14 PE11
EXTI_SOURCE_EXTI15_PB15	EXTI15 PB15
EXTI_SOURCE_EXTI15_PC5	EXTI15 PC5

### 枚举类型 `syscfg_interrupt_enum`

表 3-735. 枚举类型 `syscfg_interrupt_enum`

枚举名称	枚举描述
SYSCFG_INT_ECC_ME0	SRAM 多位不可纠错中断使能
SYSCFG_INT_ECC_SE0	SRAM 单比特可纠错中断使能
SYSCFG_INT_ECC_ME1	CAN 接收 FIFO 多位不可纠错中断使能
SYSCFG_INT_ECC_SE1	CAN 接收 FIFO 单比特可纠错中断使能
SYSCFG_INT_ECC_ME2	CAN 滤波器 SRAM 多位不可纠错中断使能
SYSCFG_INT_ECC_SE2	CAN 滤波器 SRAM 单比特可纠错中断使能
SYSCFG_INT_ECC_ME3	Flash SRAM 多位不可纠错中断使能
SYSCFG_INT_ECC_SE3	Flash SRAM 单比特可纠错中断使能
SYSCFG_INT_ECC_ME4	Flash 多位不可纠错中断使能
SYSCFG_INT_NMI_HXTAL	HXTAL 时钟监控 NMI 中断使能
SYSCFG_INT_NMI_PIN	NMI 引脚中断使能
SYSCFG_INT_NMI_WWDGT	WWDGT NMI 中断使能
SYSCFG_INT_NMI_FWDGT	FWDGT NMI 中断使能
SYSCFG_INT_NMI_LVD2	LVD2 NMI 中断使能
SYSCFG_INT_NMI_LVD1	LVD1 NMI 中断使能

### 枚举类型 `syscfg_adcs_m_enum`

表 3-736. 枚举类型 `syscfg_adcs_m_enum`

枚举名称	枚举描述
SYSCFG_ADCSM1_TIMER0_TRGOF	ADCSM1 监控源是 TIMER0_TRGOF
SYSCFG_ADCSM1_TIMER0_TRGUF	ADCSM1 监控源是 TIMER0_TRGUF
SYSCFG_ADCSM1_TIMER0_TRGA	ADCSM1 监控源是 TIMER0_TRGA
SYSCFG_ADCSM1_TIMER0_TRGB	ADCSM1 监控源是 TIMER0_TRGB
SYSCFG_ADCSM1_TIMER0_TRGAB	ADCSM1 监控源是 TIMER0_TRGAB
SYSCFG_ADCSM1_TIMER7_TRGOF	ADCSM1 监控源是 TIMER7_TRGOF
SYSCFG_ADCSM1_TIMER7_TRGUF	ADCSM1 监控源是 TIMER7_TRGUF
SYSCFG_ADCSM1_TIMER7_TRGA	ADCSM1 监控源是 TIMER7_TRGA
SYSCFG_ADCSM1_TIMER7_TRGB	ADCSM1 监控源是 TIMER7_TRGB
SYSCFG_ADCSM1_TIMER7_TRGAB	ADCSM1 监控源是 TIMER7_TRGAB
SYSCFG_ADCSM2_TIMER0_TRGOF	ADCSM2 监控源是 TIMER0_TRGOF

枚举名称	枚举描述
SYSCFG_ADCSM2_TIMER0_TRGUF	ADCSM2 监控源是 TIMER0_TRGUF
SYSCFG_ADCSM2_TIMER0_TRGA	ADCSM2 监控源是 TIMER0_TRGA
SYSCFG_ADCSM2_TIMER0_TRGB	ADCSM2 监控源是 TIMER0_TRGB
SYSCFG_ADCSM2_TIMER0_TRGAB	ADCSM2 监控源是 TIMER0_TRGAB
SYSCFG_ADCSM2_TIMER7_TRGOF	ADCSM2 监控源是 TIMER7_TRGOF
SYSCFG_ADCSM2_TIMER7_TRGUF	ADCSM2 监控源是 TIMER7_TRGUF
SYSCFG_ADCSM2_TIMER7_TRGA	ADCSM2 监控源是 TIMER7_TRGA
SYSCFG_ADCSM2_TIMER7_TRGB	ADCSM2 监控源是 TIMER7_TRGB
SYSCFG_ADCSM2_TIMER7_TRGAB	ADCSM2 监控源是 TIMER7_TRGAB

### 枚举类型 timer\_channel\_input\_enum

表 3-737. 枚举类型 timer\_channel\_input\_enum

枚举名称	枚举描述
TIMER7_CIO_INPUT_TIMER7_CH0	选择 TIMER7 CH0 作为 TIMER7 CIO
TIMER7_CIO_INPUT_CMP1_OUT	选择 CMP1 输出作为 TIMER7 CIO
TIMER7_CI1_INPUT_TIMER7_CH1	选择 TIMER7 CH1 作为 TIMER7 CI1
TIMER7_CI2_INPUT_TIMER7_CH2	选择 TIMER7 CH2 作为 TIMER7 CI2
TIMER7_CI3_INPUT_TIMER7_CH3	选择 TIMER7 CH3 作为 TIMER7 CI3
TIMER0_CIO_INPUT_TIMER0_CH0	选择 TIMER0 CH0 作为 TIMER0 CIO
TIMER0_CIO_INPUT_CMP0_OUT	选择 CMP0 输出作为 TIMER0 CIO
TIMER0_CI1_INPUT_TIMER0_CH1	选择 TIMER0 CH1 作为 TIMER0 CI1
TIMER0_CI2_INPUT_TIMER0_CH2	选择 TIMER0 CH2 作为 TIMER0 CI2
TIMER0_CI3_INPUT_TIMER0_CH3	选择 TIMER0 CH3 作为 TIMER0 CI3
TIMER2_CIO_INPUT_TIMER2_CH0	选择 TIMER2 CH0 作为 TIMER2 CIO
TIMER2_CIO_INPUT_CMP0_OUT	选择 CMP0 输出作为 TIMER2 CIO
TIMER2_CIO_INPUT_CMP1_OUT	选择 CMP1 输出作为 TIMER2 CIO
TIMER2_CIO_INPUT_CMP0_1_OUT	选择 CMP0 和 CMP1 输出作为 TIMER2 CIO
TIMER2_CIO_INPUT_EXT_TRIGGER0	选择外部触发器 0 作为 TIMER2 CIO
TIMER2_CIO_INPUT_EXT_TRIGGER1	选择外部触发器 1 作为 TIMER2 CIO
TIMER2_CIO_INPUT_EXT_TRIGGER2	选择外部触发器 2 作为 TIMER2 CIO
TIMER2_CIO_INPUT_EXT_TRIGGER3	选择外部触发器 3 作为 TIMER2 CIO
TIMER2_CI1_INPUT_TIMER2_CH1	选择 TIMER2 CH1 作为 TIMER2 CI1
TIMER2_CI1_INPUT_EXT_TRIGGER0	选择外部触发器 0 作为 TIMER2 CI1
TIMER2_CI1_INPUT_EXT_TRIGGER1	选择外部触发器 1 作为 TIMER2 CI1
TIMER2_CI1_INPUT_EXT_TRIGGER2	选择外部触发器 2 作为 TIMER2 CI1
TIMER2_CI1_INPUT_EXT_TRIGGER3	选择外部触发器 3 作为 TIMER2 CI1
TIMER2_CI2_INPUT_TIMER2_CH2	选择 TIMER2 CH2 作为 TIMER2 CI2
TIMER2_CI3_INPUT_TIMER2_CH3	选择 TIMER2 CH3 作为 TIMER2 CI3
TIMER2_CI3_INPUT_IRC32K	选择 IRC32K 作为 TIMER2 CI3
TIMER2_CI3_INPUT_CK_OUT	选择时钟输出作为 TIMER2 CI3

枚举名称	枚举描述
TIMER1_CIO_INPUT_TIMER1_CH0	选择 TIMER1 CH0 作为 TIMER1 CIO
TIMER1_CIO_INPUT_EXT_TRIGGER0	选择外部触发器 0 作为 TIMER1 CIO
TIMER1_CIO_INPUT_EXT_TRIGGER1	选择外部触发器 1 作为 TIMER1 CIO
TIMER1_CIO_INPUT_EXT_TRIGGER2	选择外部触发器 2 作为 TIMER1 CIO
TIMER1_CIO_INPUT_EXT_TRIGGER3	选择外部触发器 3 作为 TIMER1 CIO
TIMER1_C11_INPUT_TIMER1_CH1	选择 TIMER1 CH1 作为 TIMER1 C11
TIMER1_C11_INPUT_EXT_TRIGGER0	选择外部触发器 0 作为 TIMER1 C11
TIMER1_C11_INPUT_EXT_TRIGGER1	选择外部触发器 1 作为 TIMER1 C11
TIMER1_C11_INPUT_EXT_TRIGGER2	选择外部触发器 2 作为 TIMER1 C11
TIMER1_C11_INPUT_EXT_TRIGGER3	选择外部触发器 3 作为 TIMER1 C11
TIMER1_C12_INPUT_TIMER1_CH2	选择 TIMER1 CH2 作为 TIMER1 C12
TIMER1_C13_INPUT_TIMER1_CH3	选择 TIMER1 CH3 作为 TIMER1 C13
TIMER1_C13_INPUT_CMP0_OUT	选择 CMP0 输出作为 TIMER1 C13
TIMER1_C13_INPUT_CMP1_OUT	选择 CMP1 输出作为 TIMER1 C13
TIMER1_C13_INPUT_CMP0_1_OUT	选择 CMP0 和 CMP1 输出作为 TIMER1 C13

### 枚举类型 `ck_cmp_sel_enum`

表 3-738. 枚举类型 `ck_cmp_sel_enum`

枚举名称	枚举描述
CK_CMP_DIV2	CK_CMP=fPCLK/2
CK_CMP_DIV4	CK_CMP=fPCLK/4

### 函数 `syscfg_deinit`

函数 `syscfg_deinit` 描述见下表：

表 3-739. 函数 `syscfg_deinit`

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

例如：

```
/* reset SYSCFG registers */
```

```
syscfg_deinit();
```

### 函数 `syscfg_i2c_fast_mode_plus_enable`

函数 `syscfg_i2c_fast_mode_plus_enable` 描述见下表：

表 3-740. 函数 `syscfg_i2c_fast_mode_plus_enable`

函数名称	<code>syscfg_i2c_fast_mode_plus_enable</code>
函数原型	<code>void syscfg_i2c_fast_mode_plus_enable(void);</code>
功能描述	使能I2C Fm+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 I2C fust mode plus function */
```

```
syscfg_i2c_fast_mode_plus_enable();
```

### 函数 `syscfg_i2c_fast_mode_plus_disable`

函数 `syscfg_i2c_fast_mode_plus_disable` 描述见下表：

表 3-741. 函数 `syscfg_i2c_fast_mode_plus_disable`

函数名称	<code>syscfg_i2c_fast_mode_plus_disable</code>
函数原型	<code>void syscfg_i2c_fast_mode_plus_disable(void);</code>
功能描述	禁能I2C Fm+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 I2C fust mode plus function */
```

```
syscfg_i2c_fast_mode_plus_disable();
```

## 函数 syscfg\_exti\_line\_config

函数syscfg\_exti\_line\_config描述见下表：

**表 3-742. 函数 syscfg\_exti\_line\_config**

函数名称	syscfg_exti_line_config
函数原形	void syscfg_exti_line_config(exti_gpio_enum exti_gpio);
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_gpio	指定EXTI使用的GPIO端口，参考 <a href="#">表3-734. 枚举类型 exti_gpio_enum</a>
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* configure the PA8 pin as EXTI0 Line */
syscfg_exti_line_config(EXTI_SOURCE_EXTI0_PA8);
```

## 函数 syscfg\_timer\_input\_source\_select

函数syscfg\_timer\_input\_source\_select描述见下表：

**表 3-743. 函数 syscfg\_timer\_input\_source\_select**

函数名称	syscfg_timer_input_source_select
函数原型	void syscfg_timer_input_source_select(timer_channel_input_enum timer_input);
功能描述	选择TIMER输入源
先决条件	-
被调用函数	-
输入参数{in}	
timer_input	TIMER通道输入选择，参考 <a href="#">表3-737. 枚举类型 timer_channel_input_enum</a>
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* select timer channel input source */
syscfg_timer_input_source_select(TIMER7_CIO_INPUT_TIMER7_CH0);
```

**函数 syscfg\_lockup\_enable**

函数syscfg\_lockup\_enable描述见下表:

**表 3-744. 函数 syscfg\_lockup\_enable**

函数名称	syscfg_lockup_enable
函数原型	void syscfg_lockup_enable(uint32_t lockup);
功能描述	配置SYSCFG锁定功能
先决条件	-
被调用函数	-
输入参数{in}	
lockup	锁定功能
SYSCFG_LOCKUP_LOCKUP	CPU锁定
SYSCFG_LVD_LOCKUP	LVD锁定
SYSCFG_SRAM_LOCKUP	SRAM ECC多位错误锁定
SYSCFG_CANRX_LOCK	CAN接收FIFO ECC多位错误锁定
SYSCFG_CANFILTER_LOCK	CAN滤波器ECC多位错误锁定
SYSCFG_FLASHSRAM_LOCK	FLASH SRAM ECC多位错误锁定
SYSCFG_FLASH_LOCKUP	FLASH ECC多位错误锁定
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* enable module lockup function */
syscfg_lockup_enable(SYSCFG_LOCKUP_LOCKUP);
```

**函数 syscfg\_sram\_remap\_set**

函数syscfg\_sram\_remap\_set描述见下表:

**表 3-745. 函数 syscfg\_sram\_remap\_set**

函数名称	syscfg_sram_remap_set
函数原型	void syscfg_sram_remap_set(uint32_t size);
功能描述	设置SRAM重映射大小
先决条件	-
被调用函数	-
输入参数{in}	
size	SRAM重映射大小
SRAM_REMAP_SIZE_0KB	0KB
SRAM_REMAP_SIZE_8KB	8KB



SRAM_REMAP_SIZE_16KB	16KB
SRAM_REMAP_SIZE_32KB	32KB
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* set sram remap to 32KB */
```

```
syscfg_sram_remap_set(SRAM_REMAP_SIZE_32KB);
```

### 函数 syscfg\_sram\_remap\_size\_get

函数syscfg\_sram\_remap\_size\_get描述见下表：

表 3-746. 函数 syscfg\_sram\_remap\_size\_get

函数名称	syscfg_sram_remap_size_get
函数原型	uint32_t syscfg_sram_remap_size_get(void);
功能描述	获取SRAM重映射大小
先决条件	-
被调用函数	-
输入参数{in}	
-	-
输出参数{out}	
-	-
返回值	
uint32_t	SRAM重映射大小
SRAM_REMAP_SIZE_0KB	0KB
SRAM_REMAP_SIZE_8KB	8KB
SRAM_REMAP_SIZE_16KB	16KB
SRAM_REMAP_SIZE_32KB	32KB

例如：

```
/* get SRAM remap size */
```

```
uint32_t size;
```

```
size = syscfg_sram_remap_size_get();
```

### 函数 syscfg\_ck\_cmp\_sel\_set

函数syscfg\_ck\_cmp\_sel\_set描述见下表：

表 3-747. 函数 syscfg\_ck\_cmp\_sel\_set

函数名称	syscfg_ck_cmp_sel_set
函数原型	void syscfg_ck_cmp_sel_set(ck_cmp_sel_enum sel)
功能描述	配置 CK_CMP 选项
先决条件	-
被调用函数	-
输入参数{in}	
sel	CK_CMP 选项
CK_CMP_DIV2	CK_CMP=fPCLK/2
CK_CMP_DIV4	CK_CMP=fPCLK/4
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* 设置CK_CMP=fPCLK/4 */
```

```
syscfg_ck_cmp_sel_set(CK_CMP_DIV4);
```

### 函数 syscfg\_ck\_cmp\_sel\_get

函数syscfg\_ck\_cmp\_sel\_get描述见下表：

表 3-748. 函数 syscfg\_ck\_cmp\_sel\_get

函数名称	syscfg_ck_cmp_sel_get
函数原型	ck_cmp_sel_enum syscfg_ck_cmp_sel_get(void)
功能描述	获取CK_CMP选项
先决条件	-
被调用函数	-
输入参数{in}	
-	-
输出参数{out}	
-	-
返回值	
sel	CK_CMP 选项
CK_CMP_DIV2	CK_CMP=fPCLK/2
CK_CMP_DIV4	CK_CMP=fPCLK/4

例如：

```
ck_cmp_sel_enum sel;
```

```
sel = syscfg_ck_cmp_sel_get();
```

**函数 syscfg\_register\_write\_enable**

函数syscfg\_register\_write\_enable描述见下表：

**表 3-749. 函数 syscfg\_register\_write\_enable**

函数名称	syscfg_register_write_enable
函数原型	void syscfg_register_write_enable(uint32_t wp_reg);
功能描述	使能寄存器写保护功能
先决条件	-
被调用函数	-
输入参数{in}	
wp_reg	写使能的寄存器
SYSCFG_WRITE_PROTECTION_REG0	RCU_CTL, RCU_CFG0, RCU_INT, RCU_CFG1, RCU_CFG2
SYSCFG_WRITE_PROTECTION_REG1	RCU_APB1RST, RCU_APB2RST, RCU_AHBEN, RCU_APB1EN, RCU_APB2EN, RCU_RSTSCK, RCU_AHBRSRST
SYSCFG_WRITE_PROTECTION_REG2	PMU_LVD1CTL, PMU_LVD2CTL
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* 对RCU_CTL, RCU_CFG0, RCU_INT, RCU_CFG1, RCU_CFG2使能写 */
```

```
syscfg_register_write_enable(SYSCFG_WRITE_PROTECTION_REG0);
```

**函数 syscfg\_register\_write\_disable**

函数syscfg\_register\_write\_disable描述见下表：

**表 3-750. 函数 syscfg\_register\_write\_disable**

函数名称	syscfg_register_write_disable
函数原型	void syscfg_register_write_disable(uint32_t wp_reg);
功能描述	使能寄存器写功能
先决条件	-
被调用函数	-
输入参数{in}	
wp_reg	写使能的寄存器
SYSCFG_WRITE_PROTECTION_REG0	RCU_CTL, RCU_CFG0, RCU_INT, RCU_CFG1, RCU_CFG2
SYSCFG_WRITE_PROTECTION	RCU_APB1RST, RCU_APB2RST, RCU_AHBEN,

_REG1	RCU_APB1EN, RCU_APB2EN, RCU_RSTSCK, RCU_AHBRST
SYSCFG_WRITE_PROTECTION _REG2	PMU_LVD1CTL, PMU_LVD2CTL
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* 对RCU_CTL, RCU_CFG0, RCU_INT, RCU_CFG1, RCU_CFG2禁能写 */
```

```
syscfg_register_write_disable(SYSCFG_WRITE_PROTECTION_REG0);
```

### 函数 syscfg\_adc\_signal\_monitor\_select

函数syscfg\_adc\_signal\_monitor\_select描述见下表：

表 3-751. 函数 syscfg\_adc\_signal\_monitor\_select

函数名称	syscfg_adc_signal_monitor_select
函数原型	void syscfg_adc_signal_monitor_select(syscfg_adcsmd_enum timer_signal);
功能描述	选择ADC信号监控
先决条件	-
被调用函数	-
输入参数{in}	
timer_signal	TIMER信号源，参考 <a href="#">表3-736. 枚举类型 syscfg_adcsmd_enum</a>
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* 配置和使能TIMER0_TRGOF输出到ADCSMD1 */
```

```
syscfg_adc_signal_monitor_select(SYSCFG_ADCSM1_TIMER0_TRGOF);
```

```
syscfg_adc_signal_monitor_output_enable(SYSCFG_TADSRCFG_ADCSM1);
```

### 函数 syscfg\_adc\_signal\_monitor\_output\_enable

函数syscfg\_adc\_signal\_monitor\_output\_enable描述见下表：

表 3-752. 函数 `syscfg_adc_signal_monitor_output_enable`

函数名称	<code>syscfg_adc_signal_monitor_output_enable</code>
函数原型	<code>void syscfg_adc_signal_monitor_output_enable(uint32_t adcsn);</code>
功能描述	使能ADC信号监控输出
先决条件	-
被调用函数	-
输入参数{in}	
<b>adcsn</b>	ADC信号监控
<code>SYSCFG_TADSRCFG_ADCSM1</code>	ADC SM1 引脚输出
<code>SYSCFG_TADSRCFG_ADCSM2</code>	ADC SM2 引脚输出
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* 配置和使能TIMER0_TRGOF输出到ADCSM1 */
```

```
syscfg_adc_signal_monitor_select(SYSCFG_ADCSM1_TIMER0_TRGOF);
```

```
syscfg_adc_signal_monitor_output_enable(SYSCFG_TADSRCFG_ADCSM1);
```

### 函数 `syscfg_adc_signal_monitor_output_disable`

函数`syscfg_adc_signal_monitor_output_disable`描述见下表：

表 3-753. 函数 `syscfg_adc_signal_monitor_output_disable`

函数名称	<code>syscfg_adc_signal_monitor_output_disable</code>
函数原型	<code>void syscfg_adc_signal_monitor_output_disable(uint32_t adcsn);</code>
功能描述	禁能ADC信号监控输出
先决条件	-
被调用函数	-
输入参数{in}	
<b>adcsn</b>	ADC信号监控
<code>SYSCFG_TADSRCFG_ADCSM1</code>	ADC SM1 引脚输出
<code>SYSCFG_TADSRCFG_ADCSM2</code>	ADC SM2 引脚输出
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* 禁能输出到ADCSM1 */
```

```
syscfg_adc_signal_monitor_output_disable(SYSCFG_TADSRCFG_ADCSM1);
```

### 函数 syscfg\_boot\_mode\_get

函数syscfg\_boot\_mode\_get描述见下表：

表 3-754. 函数 syscfg\_boot\_mode\_get

函数名称	syscfg_boot_mode_get
函数原型	uint32_t syscfg_boot_mode_get(void);
功能描述	获取启动模式
先决条件	-
被调用函数	-
输入参数{in}	
-	-
输出参数{out}	
-	-
返回值	
uint32_t	启动模式
SYSCFG_BOOT_SYSTEM_FLASH	从系统存储器启动
SYSCFG_BOOT_MAIN_FLASH	从主flash启动

例如：

```
/* 获取启动模式 */
```

```
uint32_t boot_mode=0U;
```

```
boot_mode = syscfg_boot_mode_get();
```

### 函数 syscfg\_sram\_ecc\_error\_address\_get

函数syscfg\_sram\_ecc\_error\_address\_get描述见下表：

表 3-755. 函数 syscfg\_sram\_ecc\_error\_address\_get

函数名称	syscfg_sram_ecc_error_address_get
函数原型	uint32_t syscfg_sram_ecc_error_address_get(void);
功能描述	获取SRAM ECC错误地址
先决条件	-
被调用函数	-
输入参数{in}	
-	-
输出参数{out}	
-	-

返回值	
<b>addr</b>	错误地址，0-0x1FFF

例如：

```
/* get SRAM ECC error address */
uint32_t error_address = 0U;
error_address = syscfg_sram_ecc_error_address_get();
```

### 函数 syscfg\_sram\_ecc\_error\_bit\_get

函数syscfg\_sram\_ecc\_error\_bit\_get描述见下表：

表 3-756. 函数 syscfg\_sram\_ecc\_error\_bit\_get

函数名称	syscfg_sram_ecc_error_bit_get
函数原型	uint8_t syscfg_sram_ecc_error_bit_get(void);
功能描述	获取SRAM ECC单比特可纠错错误地址
先决条件	-
被调用函数	-
输入参数{in}	
-	-
输出参数{out}	
-	-
返回值	
<b>addr</b>	错误地址，0-0x1FFF

例如：

```
/* get SRAM ECC error address */
uint32_t error_address = 0U;
error_address = syscfg_sram_ecc_error_address_get();
```

### 函数 syscfg\_interrupt\_enable

函数syscfg\_interrupt\_enable描述见下表：

表 3-757. 函数 syscfg\_interrupt\_enable

函数名称	syscfg_interrupt_enable
函数原型	void syscfg_interrupt_enable(syscfg_interrupt_enum interrupt);
功能描述	使能中断
先决条件	-
被调用函数	-
输入参数{in}	
<b>interrupt</b>	使能中断输入，参考 <a href="#">表3-735. 枚举类型syscfg_interrupt_enum</a>

输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* enable the SRAM ECC multi-bits non-correction event flag */
```

```
syscfg_interrupt_enable(SYSCFG_INT_FLAG_ECC_ME0);
```

### 函数 syscfg\_interrupt\_disable

函数syscfg\_interrupt\_disable描述见下表：

表 3-758. 函数 syscfg\_interrupt\_disable

函数名称	syscfg_interrupt_disable
函数原型	void syscfg_interrupt_disable(syscfg_interrupt_enum interrupt);
功能描述	除能中断
先决条件	-
被调用函数	-
输入参数{in}	
interrupt	除能中断输入，参考 <a href="#">表3-735. 枚举类型syscfg_interrupt_enum</a>
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* disable the SRAM ECC multi-bits non-correction event flag */
```

```
syscfg_interrupt_disable(SYSCFG_INT_FLAG_ECC_ME0);
```

### 函数 syscfg\_interrupt\_flag\_get

函数syscfg\_interrupt\_flag\_get描述见下表：

表 3-759. 函数 syscfg\_interrupt\_flag\_get

函数名称	syscfg_interrupt_flag_get
函数原型	FlagStatus syscfg_interrupt_flag_get(syscfg_flag_enum int_flag);
功能描述	获取中断标志
先决条件	-
被调用函数	-
输入参数{in}	
int_flag	中断标志获取



SYSCFG_INT_FLAG_ECC_ME0	SRAM 多位错误标志
SYSCFG_INT_FLAG_ECC_SE0	SRAM 单位错误标志
SYSCFG_INT_FLAG_ECC_ME1	CAN FIFO SRAM 多位错误标志
SYSCFG_INT_FLAG_ECC_SE1	CAN FIFO SRAM 单位错误标志
SYSCFG_INT_FLAG_ECC_ME2	CAN 滤波器 SRAM 多位错误标志
SYSCFG_INT_FLAG_ECC_SE2	CAN 滤波器 SRAM 单位错误标志
SYSCFG_INT_FLAG_ECC_ME3	Flash SRAM 多位错误标志
SYSCFG_INT_FLAG_ECC_SE3	Flash SRAM 单位错误标志
SYSCFG_INT_FLAG_ECC_ME4	Flash 多位错误标志
SYSCFG_INT_FLAG_NMI_HXTAL	HXTAL 时钟监控 NMI 中断标志
SYSCFG_INT_FLAG_NMI_PIN	NMI 引脚中断标志
SYSCFG_INT_FLAG_NMI_WWDGT	WWDGT NMI 中断标志
SYSCFG_INT_FLAG_NMI_FWDGT	FWDGT NMI 中断标志
输出参数{out}	
-	-
返回值	
FlagStatus	中断标志状态
SET	获取到标志
RESET	未获取到标志

例如：

```
/* get the SRAM ECC multi-bits non-correction event flag */
```

```
FlagStatus Flag = RESET;
```

```
Flag = syscfg_interrupt_flag_get(SYSCFG_INT_FLAG_ECC_ME0);
```

### 函数 syscfg\_interrupt\_flag\_clear

函数syscfg\_interrupt\_flag\_clear描述见下表：

表 3-760. 函数 syscfg\_interrupt\_flag\_clear

函数名称	syscfg_interrupt_flag_clear
函数原型	void syscfg_interrupt_flag_clear(uint32_t int_flag);
功能描述	清除中断标志
先决条件	-
被调用函数	-
输入参数{in}	
int_flag	清除中断标志
SYSCFG_INT_FLAG_ECC_ME0	SRAM 多位错误标志
SYSCFG_INT_FLAG_ECC_SE0	SRAM 单位错误标志
SYSCFG_INT_FLAG_ECC_ME1	CAN FIFO SRAM 多位错误标志
SYSCFG_INT_FLAG_ECC_SE1	CAN FIFO SRAM 单位错误标志
SYSCFG_INT_FLAG_ECC_ME2	CAN 滤波器 SRAM 多位错误标志

SYSCFG_INT_FLAG_ECC_SE2	CAN 滤波器 SRAM 单位错误标志
SYSCFG_INT_FLAG_ECC_ME3	Flash SRAM 多位错误标志
SYSCFG_INT_FLAG_ECC_SE3	Flash SRAM 单位错误标志
SYSCFG_INT_FLAG_ECC_ME4	Flash 多位错误标志
SYSCFG_INT_FLAG_NMI_FWDGT	FWDGT NMI 中断标志
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* clear the SRAM ECC multi-bits non-correction event flag */
```

```
syscfg_interrupt_flag_clear (SYSCFG_INT_FLAG_ECC_ME0);
```

## 3.27. 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，TIMER7），通用定时器L0（TIMER1，TIMER2），不同类型的定时器具体功能有所差别。章节[3.27.1](#)描述了TIMER的寄存器列表，章节[3.27.2](#)对TIMER库函数进行说明。

### 3.27.1. 外设寄存器说明

TIMER寄存器列表如下所示：

表 3-761. TIMER 寄存器

寄存器名称	寄存器描述
TIMERx_CTL0	控制寄存器0
TIMERx_CTL1	控制寄存器1
TIMERx_SMCFG	从模式配置寄存器
TIMERx_DMAINTEN	DMA和中断使能寄存器
TIMERx_INTF	中断标志寄存器
TIMERx_SWEVG	软件事件产生寄存器
TIMERx_CHCTL0	通道控制寄存器0
TIMERx_CHCTL1	通道控制寄存器1
TIMERx_CHCTL2	通道控制寄存器2
TIMERx_CNT	计数器寄存器
TIMERx_PSC	预分频寄存器
TIMERx_CAR	计数器自动重载寄存器
TIMERx_CREP	重复计数寄存器
TIMERx_CH0CV	通道0捕获/比较寄存器
TIMERx_CH1CV	通道1捕获/比较寄存器

寄存器名称	寄存器描述
TIMERx_CH2CV	通道2捕获/比较寄存器
TIMERx_CH3CV	通道3捕获/比较寄存器
TIMERx_CCHP	互补通道保护寄存器
TIMERx_MCHCTL0	多模式通道控制寄存器0
TIMERx_MCHCTL1	多模式通道控制寄存器1
TIMERx_MCHCTL2	多模式通道控制寄存器2
TIMERx_MCH0CV	多模式通道0捕获/比较寄存器
TIMERx_MCH1CV	多模式通道1捕获/比较寄存器
TIMERx_MCH2CV	多模式通道2捕获/比较寄存器
TIMERx_MCH3CV	多模式通道3捕获/比较寄存器
TIMERx_CH0COMV_ADD	通道0附加比较寄存器
TIMERx_CH1COMV_ADD	通道1附加比较寄存器
TIMERx_CH2COMV_ADD	通道2附加比较寄存器
TIMERx_CH3COMV_ADD	通道3附加比较寄存器
TIMERx_CTL2	控制寄存器2
TIMERx_AFCTL	附加控制寄存器
TIMERx_IREP	中断重复计数寄存器
TIMERx_ADCRCTL	ADC转换请求控制寄存器
TIMERx_ADCCR1	ADC触发比较寄存器1
TIMERx_ADCCR2	ADC触发比较寄存器2
TIMERx_ADCREP	ADC重复计数寄存器
TIMERx_ADCTL	ADC触发控制寄存器
TIMERx_SHUPM0	影子更新模式寄存器0
TIMERx_SHUPM1	影子更新模式寄存器1
TIMERx_COUNSEL	计数上/下溢模式选择寄存器
TIMERx_DMACFG	DMA配置寄存器
TIMERx_DMATB	DMA发送缓冲区寄存器
TIMERx_CFG	配置寄存器

### 3.27.2. 外设库函数说明

TIMER库函数列表如下表所示：

**表 3-762. 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
timer_auto_reload_shadow_enable	TIMERx自动重载影子使能

库函数名称	库函数描述
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_upif_backup_enable	使能外设TIMERx的更新备份功能
timer_upif_backup_disable	禁能外设TIMERx的更新备份功能
timer_flow_interrupt_source_select	选择外设TIMERx的溢出中断源
timer_update_source_select	选择外设TIMERx的更新事件源
timer_external_input_source_select	选择外设TIMERx的外部输入源
timer_prescaler_config	配置外设TIMERx预分频器
timer_update_repetition_value_config	配置外设TIMERx的更新重复计数器
timer_flow_interrupt_repetition_value_config	配置外设TIMERx的中断重复计数器
timer_flow_interrupt_repetition_value_reload	重载外设TIMERx的溢出中断重复值
timer_autoreload_value_config	配置外设TIMERx的自动重载寄存器
timer_autoreload_value_read	读取外设TIMERx的自动重载值
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_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_ocpre_clr_source_select	选择外设TIMERx的OCPRE_CLR信号源
timer_ocpre_clr_int_source_select	选择外设TIMERx的OCPRE_CLR_INT信号源
timer_channel_composite_asymmetric_pwm_level_config	配置外设TIMERx的复合PWM和非对称PWM的比较时刻
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

库函数名称	库函数描述
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_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_compare_fast_config	配置TIMERx通道输出比较快速功能
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_multi_mode_channel_output_parameter_struct_init	将TIMER多功能通道输出参数结构体中所有参数初始化为默认值
timer_multi_mode_channel_output_config	外设TIMERx的多模式通道输出配置
timer_multi_mode_channel_mode_config	配置外设TIMERx多模式通道模式
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

库函数名称	库函数描述
timer_external_trigger_config	配置TIMERx外部触发输入
timer_quadrature_decoder_mode_config	TIMERx配置为正交编码器模式
timer_non_quadrature_decoder_mode_config	TIMERx配置为非正交编码器模式
timer_internal_clock_config	TIMERx配置为内部时钟
timer_internal_trigger_as_external_clock_config	配置外设TIMERx的内部触发作为外部时钟输入
timer_external_trigger_as_external_clock_config	配置外设TIMERx的外部触发作为外部时钟输入
timer_slave_mode_input_config	配置外设TIMERx从模式输入
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_write_chxval_register_config	配置TIMERx写CHxVAL选择位
timer_output_value_selection_config	配置TIMERx输出值选择
timer_output_match_pulse_select	选择外设TIMERx的输出匹配脉冲
timer_channel_composite_pwm_pulse_config	配置外设TIMERx的复合PWM模式输出比较值
timer_channel_asymmetric_pwm_pulse_config	配置外设TIMERx的非对称PWM模式输出比较值
timer_channel_additional_compare_value_config	配置外设TIMERx通道的附加比较值
timer_channel_additional_compare_value_read	读取外设TIMERx通道的附加比较值
timer_channel_additional_output_shadow_config	配置外设TIMERx的附加输出比较影子寄存器
timer_break_external_input_enable	使能外部中止输入
timer_break_cmp_enable	使能CMP输出中止信号
timer_break_external_input_disable	禁能外部中止输入
timer_break_cmp_disable	禁能CMP输出中止信号
timer_break_external_input_polarity_config	配置外设TIMERx的外部中止输入极性
timer_break_cmp_polarity_config	配置外设TIMERx的CMP输出中止信号极性
timer_break_auto_recover_event_select	选择中止信号恢复事件
timer_trigger_adc_compare_enable	使能外设TIMERx的比较事件产生一个ADC转换触发信号
timer_trigger_adc_compare_disable	禁能外设TIMERx的比较事件产生一个ADC转换触发信号
timer_trigger_adc_flow_enable	使能外设TIMERx的溢出事件产生一个ADC转换触发信号
timer_trigger_adc_flow_disable	禁能外设TIMERx的溢出事件产生一个ADC转换触发信号
timer_trigger_adc_compare_value_config	配置ADC触发比较寄存器

库函数名称	库函数描述
ig	
timer_trigger_adc_repetition_value_config	配置ADC触发重复计数值
timer_trigger_adc_repetition_decrement_select	选择ADC触发重复计数值自减信号源
timer_trigger_adc_repetition_value_reload	重载ADC触发重复计数值
timer_trigger_adc_compare_value_shadow_enable	使能ADC触发比较影子寄存器
timer_trigger_adc_compare_value_shadow_disable	禁能ADC触发比较影子寄存器
timer_trigger_adc_monitor_config	配置ADC触发信号监控
timer_register_update_event_select	选择影子寄存器更新事件
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 3-763. 结构体 timer\_parameter\_struct

成员名称	功能描述
prescaler	预分频值（0~65535）
alignedmode	对齐模式（TIMER_COUNTER_EDGE, TIMER_COUNTER_CENTER_DOWN, TIMER_COUNTER_CENTER_UP, TIMER_COUNTER_CENTER_BOTH）
counterdirection	计数方向（TIMER_COUNTER_UP, TIMER_COUNTER_DOWN）
period	周期（0~0xFFFF）
clockdivision	时钟分频因子（TIMER_CKDIV_DIV1, TIMER_CKDIV_DIV2, TIMER_CKDIV_DIV4）
repetitioncounter	重复计数器值（0~0xFF，用于TIMER_CREP寄存器）

### 结构体 timer\_break\_parameter\_struct

表 3-764. 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）

成员名称	功能描述
deadtime	死区时间（0~255）
breakpolarity	BREAK输入极性（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	BREAK输入使能（TIMER_BREAK_ENABLE, TIMER_BREAK_DISABLE）
breakfilter	BREAK0输入滤波（0~15）

### 结构体 timer\_oc\_parameter\_struct

表 3-765. 结构体 timer\_oc\_parameter\_struct

成员名称	功能描述
outputstate	通道输出状态（TIMER_CCX_ENABLE, TIMER_CCX_DISABLE）
outputnstate	互补通道输出状态（TIMER_CCXN_ENABLE, TIMER_CCXN_DISABLE）
ocpolarity	通道输出极性（TIMER_OC_POLARITY_HIGH, TIMER_OC_POLARITY_LOW）
ocnpolarity	互补通道输出极性（TIMER_OCN_POLARITY_HIGH, TIMER_OCN_POLARITY_LOW）
ocidlestate	空闲状态下通道输出（TIMER_OC_IDLE_STATE_LOW, TIMER_OC_IDLE_STATE_HIGH）
ocnidlestate	空闲状态下互补通道输出（TIMER_OCN_IDLE_STATE_LOW, TIMER_OCN_IDLE_STATE_HIGH）

### 结构体 timer\_omc\_parameter\_struct

表 3-766. 结构体 timer\_omc\_parameter\_struct

成员名称	功能描述
outputmode	多模式通道输出模式选择（TIMER_MCH_MODE_INDEPENDENTLY, TIMER_MCH_MODE_COMPLEMENTARY）
outputstate	多模式通道输出状态（TIMER_MCCX_ENABLE, TIMER_MCCX_DISABLE）
ocpolarity	多模式通道输出极性（TIMER_OMC_POLARITY_HIGH, TIMER_OMC_POLARITY_LOW）

### 结构体 timer\_ic\_parameter\_struct

表 3-767. 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）



成员名称	功能描述
icselection	通道输入模式选择 (TIMER_IC_SELECTION_DIRECTTI, TIMER_IC_SELECTION_INDIRECTTI, TIMER_IC_SELECTION_ITS, TIMER_IC_SELECTION_PAIR)
icprescaler	通道输入捕获预分频 (TIMER_IC_PSC_DIV1, TIMER_IC_PSC_DIV2, TIMER_IC_PSC_DIV4, TIMER_IC_PSC_DIV8)
icfilter	通道输入捕获滤波 (0~15)

## 函数 timer\_deinit

函数timer\_deinit描述见下表:

表 3-768. 函数 timer\_deinit

函数名称	timer_deinit
函数原型	void timer_deinit(uint32_t timer_periph);
功能描述	复位外设TIMER
先决条件	-
被调用函数	rcu_periph_reset_enable / rcu_periph_reset_disable
输入参数{in}	
timer_periph	TIMER外设
TIMERx(x=0,1,2,7)	TIMER外设选择
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* reset TIMER0 */
```

```
timer_deinit (TIMER0);
```

## 函数 timer\_struct\_para\_init

函数timer\_struct\_para\_init描述见下表:

表 3-769. 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 <a href="#">结构体timer_parameter_struct</a> 。
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 3-770. 函数 timer\_init

函数名称	timer_init
函数原型	void timer_init(uint32_t timer_periph, timer_parameter_struct* initpara);
功能描述	初始化外设TIMER
先决条件	-
被调用函数	-
输入参数{in}	
timer_periph	TIMER外设
TIMERx(x=0,1,2,7)	TIMER外设选择
输入参数{in}	
initpara	TIMER初始化结构体，结构体成员参考 <a href="#">结构体timer_parameter_struct</a> 。
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* initialize TIMER0 */
```

```
timer_parameter_struct timer_initpara;
```

```
timer_initpara.prescaler      = 107;
```

```
timer_initpara.alignedmode    = TIMER_COUNTER_EDGE;
```

```
timer_initpara.counterdirection = TIMER_COUNTER_UP;
```

```
timer_initpara.period         = 999;
```

```
timer_initpara.clockdivision   = TIMER_CKDIV_DIV1;
```

```
timer_initpara.repetitioncounter = 1;
```

```
timer_init(TIMERO, &timer_initpara);
```

### 函数 timer\_enable

函数timer\_enable描述见下表:

表 3-771. 函数 timer\_enable

函数名称	timer_enable
函数原型	void timer_enable(uint32_t timer_periph);
功能描述	使能外设TIMER
先决条件	-
被调用函数	-
输入参数{in}	
timer_periph	TIMER外设
TIMERx(x=0,1,2,7)	TIMER外设选择
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* enable TIMERO */
```

```
timer_enable(TIMERO);
```

### 函数 timer\_disable

函数timer\_disable描述见下表:

表 3-772. 函数 timer\_disable

函数名称	timer_disable
函数原型	void timer_disable(uint32_t timer_periph);
功能描述	禁能外设TIMER
先决条件	-
被调用函数	-
输入参数{in}	
timer_periph	TIMER外设
TIMERx(x=0,1,2,7)	TIMER外设选择
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* disable TIMER0 */
```

```
timer_disable(TIMER0);
```

### 函数 timer\_auto\_reload\_shadow\_enable

函数timer\_auto\_reload\_shadow\_enable描述见下表：

**表 3-773. 函数 timer\_auto\_reload\_shadow\_enable**

函数名称	timer_auto_reload_shadow_enable
函数原型	void timer_auto_reload_shadow_enable(uint32_t timer_periph);
功能描述	TIMER自动重载影子使能
先决条件	-
被调用函数	-
输入参数{in}	
timer_periph	TIMER外设
TIMERx(x=0,1,2,7)	TIMER外设选择
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* enable the TIMER0 auto reload shadow function */
```

```
timer_auto_reload_shadow_enable(TIMER0);
```

### 函数 timer\_auto\_reload\_shadow\_disable

函数timer\_auto\_reload\_shadow\_disable描述见下表：

**表 3-774. 函数 timer\_auto\_reload\_shadow\_disable**

函数名称	timer_auto_reload_shadow_disable
函数原型	void timer_auto_reload_shadow_disable (uint32_t timer_periph);
功能描述	TIMER自动重载影子禁能
先决条件	-
被调用函数	-
输入参数{in}	
timer_periph	TIMER外设
TIMERx(x=0,1,2,7)	TIMER外设选择
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* disable the TIMER0 auto reload shadow function */
```

```
timer_auto_reload_shadow_disable(TIMER0);
```

### 函数 timer\_update\_event\_enable

函数timer\_update\_event\_enable描述见下表：

**表 3-775. 函数 timer\_update\_event\_enable**

函数名称	timer_update_event_enable
函数原型	void timer_update_event_enable(uint32_t timer_periph);
功能描述	TIMER更新使能
先决条件	-
被调用函数	-
输入参数{in}	
timer_periph	TIMER外设
TIMERx(x=0,1,2,7)	TIMER外设选择
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* enable TIMER0 the update event */
```

```
timer_update_event_enable(TIMER0);
```

### 函数 timer\_update\_event\_disable

函数timer\_update\_event\_disable描述见下表：

**表 3-776. 函数 timer\_update\_event\_disable**

函数名称	timer_update_event_disable
函数原型	void timer_update_event_disable (uint32_t timer_periph);
功能描述	TIMER更新禁能
先决条件	-
被调用函数	-
输入参数{in}	
timer_periph	TIMER外设
TIMERx(x=0,1,2,7)	TIMER外设选择
输出参数{out}	
-	-
返回值	

-	-
---	---

例如:

```
/* disable TIMER0 the update event */
timer_update_event_disable(TIMER0);
```

### 函数 timer\_counter\_alignment

函数timer\_counter\_alignment描述见下表:

表 3-777. 函数 timer\_counter\_alignment

函数名称	timer_counter_alignment
函数原型	void timer_counter_alignment(uint32_t timer_periph, uint16_t aligned);
功能描述	设置外设TIMER的对齐模式
先决条件	-
被调用函数	-
输入参数{in}	
timer_periph	TIMER外设
TIMERx(x=0, 1, 2, 7)	TIMER外设选择
输入参数{in}	
aligned	对齐模式
TIMER_COUNTER_EDGE	边沿对齐模式
TIMER_COUNTER_CENTER_DOWN	中央对齐向下计数模式
TIMER_COUNTER_CENTER_UP	中央对齐向上计数模式
TIMER_COUNTER_CENTER_BOTH	中央对齐上下计数模式
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* set TIMER0 counter center-aligned and counting up assert mode */
timer_counter_alignment(TIMER0, TIMER_COUNTER_CENTER_UP);
```

### 函数 timer\_counter\_up\_direction

函数timer\_counter\_up\_direction描述见下表:

表 3-778. 函数 timer\_counter\_up\_direction

函数名称	timer_counter_up_direction
函数原型	void timer_counter_up_direction(uint32_t timer_periph);
功能描述	设置外设TIMER向上计数
先决条件	计数器设置为无中央对齐计数模式（边沿对齐模式）
被调用函数	-
输入参数{in}	
timer_periph	TIMER外设
TIMERx(x=0,1,2,7)	TIMER外设选择
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* set TIMER0 counter up direction */
```

```
timer_counter_up_direction(TIMER0);
```

### 函数 timer\_counter\_down\_direction

函数timer\_counter\_down\_direction描述见下表：

表 3-779. 函数 timer\_counter\_down\_direction

函数名称	timer_counter_down_direction
函数原型	void timer_counter_down_direction(uint32_t timer_periph);
功能描述	设置外设TIMER向下计数
先决条件	计数器设置为无中央对齐计数模式（边沿对齐模式）
被调用函数	-
输入参数{in}	
timer_periph	TIMER外设
TIMERx(x=0,1,2,7)	TIMER外设选择
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* set TIMER0 counter down direction */
```

```
timer_counter_down_direction(TIMER0);
```

## 函数 timer\_upif\_backup\_enable

函数timer\_upif\_backup\_enable描述见下表：

**表 3-780. 函数 timer\_upif\_backup\_enable**

函数名称	timer_upif_backup_enable
函数原型	void timer_upif_backup_enable (uint32_t timer_periph);
功能描述	使能外设TIME更新备份
先决条件	-
被调用函数	-
输入参数{in}	
timer_periph	TIMER外设
TIMERx(x=0,1,2,7)	TIMER外设选择
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* enable upif backup */
timer_upif_backup_enable(TIMER0);
```

## 函数 timer\_upif\_backup\_disable

函数timer\_upif\_backup\_disable描述见下表：

**表 3-781. 函数 timer\_upif\_backup\_disable**

函数名称	timer_upif_backup_disable
函数原型	void timer_upif_backup_disable (uint32_t timer_periph);
功能描述	禁能外设TIME更新备份
先决条件	-
被调用函数	-
输入参数{in}	
timer_periph	TIMER外设
TIMERx(x=0,1,2,7)	TIMER外设选择
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* disable upif backup */
timer_upif_backup_disable(TIMER0);
```



## 函数 timer\_flow\_interrupt\_source\_select

函数timer\_flow\_interrupt\_source\_select描述见下表：

表 3-782. 函数 timer\_flow\_interrupt\_source\_select

函数名称	timer_flow_interrupt_source_select
函数原型	void timer_flow_interrupt_source_select (uint32_t timer_periph, uint32_t source);
功能描述	选择外设TIMERx的溢出中断源
先决条件	-
被调用函数	-
输入参数{in}	
timer_periph	TIMER外设
TIMERx(x=0,1,2,7)	TIMER外设选择
输入参数{in}	
source	溢出中断源
TIMER_INT_SEL_FLOW	计数上/下溢中断
TIMER_INT_SEL_OVERFLOW	计数上溢中断
TIMER_INT_SEL_UNDERFLOW	计数下溢中断
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* configure the timer flow interrupt souce selection */
```

```
timer_flow_interrupt_source_select(TIMER0, TIMER_INT_SEL_FLOW);
```

## 函数 timer\_update\_source\_select

函数timer\_update\_source\_select描述见下表：

表 3-783. 函数 timer\_update\_source\_select

函数名称	timer_update_source_select
函数原型	void timer_update_source_select (uint32_t timer_periph, uint32_t source);
功能描述	选择外设TIMERx的更新事件源
先决条件	-
被调用函数	-
输入参数{in}	
timer_periph	TIMER外设

<i>TIMERx</i> ( <i>x</i> =0,1,2,7)	TIMER外设选择
输入参数{in}	
<b>source</b>	更新事件源
<i>TIMER_UPS_SEL_FLOW</i>	计数上/下溢更新
<i>TIMER_UPS_SEL_OVERFLOW</i>	计数上溢更新
<i>TIMER_UPS_SEL_UNDERFLOW</i>	计数下溢更新
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* configure the timer update event souce selection */
```

```
timer_update_source_select(TIMER0, TIMER_UPS_SEL_FLOW);
```

### 函数 timer\_external\_input\_source\_select

函数timer\_external\_input\_source\_select描述见下表:

表 3-784. 函数 timer\_external\_input\_source\_select

函数名称	timer_external_input_source_select
函数原型	void timer_external_input_source_select (uint32_t timer_periph, uint32_t source);
功能描述	选择外设TIMERx的外部输入源
先决条件	-
被调用函数	-
输入参数{in}	
<b>timer_periph</b>	TIMER外设
<i>TIMERx</i> ( <i>x</i> =0,1,2,7)	TIMER外设选择
输入参数{in}	
<b>source</b>	外部触发输入信号
<i>TIMER_ETI_SEL_ETI1</i>	TIMER_ETI1, <i>TIMERx</i> ( <i>x</i> = 0,1,2,7)
<i>TIMER_ETI_SEL_ETI2</i>	TIMER_ETI2, <i>TIMERx</i> ( <i>x</i> = 0,1,2,7)
<i>TIMER_ETI_SEL_ETI3</i>	TIMER_ETI3, <i>TIMERx</i> ( <i>x</i> = 1,2)
<i>TIMER_ETI_SEL_ETI4</i>	TIMER_ETI4, <i>TIMERx</i> ( <i>x</i> = 1,2)

输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* configure the TIMER ETI */
```

```
timer_external_input_source_select(TIMER0, TIMER_ETI_SEL_ETI1);
```

### 函数 timer\_prescaler\_config

函数timer\_prescaler\_config描述见下表：

表 3-785. 函数 timer\_prescaler\_config

函数名称	timer_prescaler_config
函数原型	void timer_prescaler_config(uint32_t timer_periph, uint16_t prescaler, uint32_t pscreload);
功能描述	配置外设TIMER预分频器
先决条件	-
被调用函数	-
输入参数{in}	
timer_periph	TIMER外设
TIMERx(x=0,7)	TIMER外设选择
输入参数{in}	
prescaler	预分频值，0~0xFFFF
输入参数{in}	
pscreload	预分频值加载模式
TIMER_PSC_RELOAD_NOW	预分频值立即加载
TIMER_PSC_RELOAD_UPDATE	预分频值在下次更新事件发生时加载
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* configure TIMER0 prescaler */
```

```
timer_prescaler_config(TIMER0, 3000, TIMER_PSC_RELOAD_NOW);
```

## 函数 timer\_update\_repetition\_value\_config

函数timer\_update\_repetition\_value\_config描述见下表：

表 3-786. 函数 timer\_update\_repetition\_value\_config

函数名称	timer_update_repetition_value_config
函数原型	void timer_update_repetition_value_config (uint32_t timer_periph, uint16_t repetition)
功能描述	配置外设TIMER的更新重复计数器
先决条件	-
被调用函数	-
输入参数{in}	
timer_periph	TIMER外设
TIMERx(x=0,7)	TIMER外设选择
输入参数{in}	
repetition	重复计数器值，取值范围（0~0xFF）
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* configure TIMER0 update repetition register value */
```

```
timer_update_repetition_value_config(TIMER0, 98);
```

## 函数 timer\_flow\_interrupt\_repetition\_value\_config

函数timer\_flow\_interrupt\_repetition\_value\_config描述见下表：

表 3-787. 函数 timer\_flow\_interrupt\_repetition\_value\_config

函数名称	timer_flow_interrupt_repetition_value_config
函数原型	void timer_flow_interrupt_repetition_value_config(uint32_t timer_periph, uint16_t repetition)
功能描述	配置外设TIMER的中断重复计数器
先决条件	-
被调用函数	-
输入参数{in}	
timer_periph	TIMER外设
TIMERx(x=0,7)	TIMER外设选择
输入参数{in}	
repetition	重复计数器值，取值范围（0~0xFF）
输出参数{out}	
-	-

返回值	
-	-

例如：

```
/* configure TIMER0 interrupt repetition register value */
```

```
timer_flow_interrupt_repetition_value_config(TIMER0, 98);
```

### 函数 timer\_flow\_interrupt\_repetition\_value\_reload

函数timer\_flow\_interrupt\_repetition\_value\_reload描述见下表：

**表 3-788. 函数 timer\_flow\_interrupt\_repetition\_value\_reload**

函数名称	timer_flow_interrupt_repetition_value_reload
函数原型	void timer_flow_interrupt_repetition_value_reload (uint32_t timer_periph)
功能描述	重载外设TIMER的中断重复计数器
先决条件	-
被调用函数	-
输入参数{in}	
timer_periph	TIMER外设
TIMERx(x=0,7)	TIMER外设选择
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* timer flow interrupt repetition value immediate reload */
```

```
timer_flow_interrupt_repetition_value_reload(TIMER0);
```

### 函数 timer\_autoreload\_value\_config

函数timer\_autoreload\_value\_config描述见下表：

**表 3-789. 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, uint32_t autoreload);
功能描述	配置外设TIMER的自动重载寄存器
先决条件	-
被调用函数	-
输入参数{in}	
timer_periph	TIMER外设
TIMERx(x=0,1,2,7)	TIMER外设选择
输入参数{in}	

<b>autoreload</b>	计数器自动重载值 0~0xFFFF
<b>输出参数{out}</b>	
-	-
<b>返回值</b>	
-	-

例如：

```
/* configure TIMER autoreload register value */
timer_autoreload_value_config(TIMERO, 3000);
```

### 函数 timer\_autoreload\_value\_read

函数timer\_autoreload\_value\_read描述见下表：

表 3-790. 函数 timer\_autoreload\_value\_read

<b>函数名称</b>	timer_autoreload_value_read
<b>函数原型</b>	Uint32_t timer_autoreload_value_read(uint32_t timer_periph);
<b>功能描述</b>	读取外设TIMER自动重载寄存器值
<b>先决条件</b>	-
<b>被调用函数</b>	-
<b>输入参数{in}</b>	
<b>timer_periph</b>	TIMER外设
<i>TIMERx(x=0,1,2,7)</i>	TIMER外设选择
<b>输出参数{out}</b>	
-	-
<b>返回值</b>	
<b>Uint32_t</b>	计数器自动重载值 0~0xFFFF

例如：

```
/* get TIMER autoreload register value */
Uint32_t i = 0;
i =(uint32_t) timer_autoreload_value_read (TIMERO);
```

### 函数 timer\_counter\_value\_config

函数timer\_counter\_value\_config描述见下表：

表 3-791. 函数 timer\_counter\_value\_config

<b>函数名称</b>	timer_counter_value_config
<b>函数原型</b>	void timer_counter_value_config(uint32_t timer_periph, uint32_t counter);

功能描述	配置外设TIMER的计数器值
先决条件	-
被调用函数	-
输入参数{in}	
timer_periph	TIMER外设
TIMERx(x=0,1,2,7)	TIMER外设选择
输入参数{in}	
counter	计数器值 0~0xFFFF
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* configure TIMER0 counter register value */
timer_counter_value_config(TIMER0, 999);
```

### 函数 timer\_counter\_read

函数timer\_counter\_read描述见下表：

表 3-792. 函数 timer\_counter\_read

函数名称	timer_counter_read
函数原型	Uint32_t timer_counter_read(uint32_t timer_periph);
功能描述	读取外设TIMER的计数器值
先决条件	-
被调用函数	-
输入参数{in}	
timer_periph	TIMER外设
TIMERx(x=0,1,2,7)	TIMER外设选择
输出参数{out}	
-	-
返回值	
Uint32_t	外设TIMER的计数器值 0~0xFFFF

例如：

```
/* read TIMER0 counter value */
Uint32_t i = 0;
i = timer_counter_read(TIMER0);
```

## 函数 timer\_prescaler\_read

函数timer\_prescaler\_read描述见下表:

表 3-793. 函数 timer\_prescaler\_read

函数名称	timer_prescaler_read
函数原型	uint16_t timer_prescaler_read(uint32_t timer_periph);
功能描述	读取外设TIMER的预分频器值
先决条件	-
被调用函数	-
输入参数{in}	
timer_periph	TIMER外设
TIMERx(x=0,1,2,7)	TIMER外设选择
输出参数{out}	
-	-
返回值	
uint16_t	外设TIMER的预分频器值 (0~0xFFFF)

例如:

```
/* read TIMER0 prescaler value */
```

```
uint16_t i = 0;
```

```
i = timer_prescaler_read(TIMER0);
```

## 函数 timer\_single\_pulse\_mode\_config

函数timer\_single\_pulse\_mode\_config描述见下表:

表 3-794. 函数 timer\_single\_pulse\_mode\_config

函数名称	timer_single_pulse_mode_config
函数原型	void timer_single_pulse_mode_config(uint32_t timer_periph, uint32_t spmode);
功能描述	配置外设TIMER的单脉冲模式
先决条件	-
被调用函数	-
输入参数{in}	
timer_periph	TIMER外设
TIMERx(x=0,1,2,7)	TIMER外设选择
输入参数{in}	
spmode	脉冲模式
TIMER_SP_MODE_SINGLE	单脉冲模式计数
TIMER_SP_MODE_REPETITIVE	重复模式计数
输出参数{out}	



-	-
返回值	
-	-

例如:

```
/* configure TIMER0 single pulse mode */
```

```
timer_single_pulse_mode_config(TIMER0, TIMER_SP_MODE_SINGLE);
```

### 函数 timer\_update\_source\_config

函数timer\_update\_source\_config描述见下表:

**表 3-795. 函数 timer\_update\_source\_config**

函数名称	timer_update_source_config
函数原型	void timer_update_source_config(uint32_t timer_periph, uint32_t update);
功能描述	配置外设TIMER的更新源
先决条件	-
被调用函数	-
输入参数{in}	
timer_periph	TIMER外设
TIMEx(x=0,1,2,7)	TIMER外设选择
输入参数{in}	
update	更新源
TIMER_UPDATE_SRC_GLOBAL	下述任一事件产生更新中断或DMA请求: - UPG位被置1 - 计数器溢出/下溢 - 从模式控制器产生的更新
TIMER_UPDATE_SRC_REGULAR	只有计数器溢出/下溢才产生更新中断或DMA请求
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* configure TIMER update only by counter overflow/underflow */
```

```
timer_update_source_config(TIMER0, TIMER_UPDATE_SRC_REGULAR);
```

### 函数 timer\_channel\_control\_shadow\_config

函数timer\_channel\_control\_shadow\_config描述见下表:

表 3-796. 函数 timer\_channel\_control\_shadow\_config

函数名称	timer_channel_control_shadow_config
函数原型	void timer_channel_control_shadow_config(uint32_t timer_periph, ControlStatus newvalue);
功能描述	配置通道控制影子寄存器
先决条件	-
被调用函数	-
输入参数{in}	
timer_periph	TIMER外设
TIMERx(x=0,7)	TIMER外设选择
输入参数{in}	
newvalue	控制状态
ENABLE	使能
DISABLE	禁能
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* channel commutation control shadow register enable */
```

```
timer_channel_control_shadow_config(TIMER0, ENABLE);
```

### 函数 timer\_channel\_control\_shadow\_update\_config

函数timer\_channel\_control\_shadow\_update\_config描述见下表：

表 3-797. 函数 timer\_channel\_control\_shadow\_update\_config

函数名称	timer_channel_control_shadow_update_config
函数原型	void timer_channel_control_shadow_update_config(uint32_t timer_periph, uint32_t ccuctl);
功能描述	配置TIMER通道控制影子寄存器更新控制
先决条件	-
被调用函数	-
输入参数{in}	
timer_periph	TIMER外设
TIMERx(x=0,7)	TIMER外设选择
输入参数{in}	
ccuctl	通道换相控制影子寄存器更新控制
TIMER_UPDATECT L_CCUC	CMTG位被置1时更新影子寄存器
TIMER_UPDATECT	当CMTG位被置1或检测到TRIGI上升沿时，影子寄存器更新

<i>L_CCUTRI</i>	
<i>TIMER_UPDATECT</i> <i>L_CCUOVERFLOW</i>	当计数器上溢事件发生时，影子寄存器更新
<i>TIMER_UPDATECT</i> <i>L_CCUUNDERFLOW</i> <i>W</i>	当计数器下溢事件发生时，影子寄存器更新
<i>TIMER_UPDATECT</i> <i>L_CCUFLOW</i>	当计数器上溢/ 下溢事件发生时，影子寄存器更新
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* configure TIMER0 channel control shadow register update when CMTG bit is set */
timer_channel_control_shadow_update_config(TIMER0, TIMER_UPDATECTL_CCU);
```

### 函数 timer\_ocpre\_clr\_source\_select

函数timer\_ocpre\_clr\_source\_select描述见下表：

表 3-798. 函数 timer\_ocpre\_clr\_source\_select

函数名称	timer_ocpre_clr_source_select
函数原型	void timer_ocpre_clr_source_select(uint32_t timer_periph, uint32_t ocrsel);
功能描述	配置外设TIMERx的OCPRE_CLR信号源
先决条件	-
被调用函数	-
输入参数{in}	
timer_periph	TIMER外设
<i>TIMERx(x=0,7)</i>	参考具体参数
输入参数{in}	
ocrsel	OCPRE_CLR信号源
<i>TIMER_OCPRE_CLR_CMP0</i>	OCPRE_CLR信号源为CMP0
<i>TIMER_OCPRE_CLR_CMP1</i>	OCPRE_CLR信号源为CMP1
<i>TIMER_OCPRE_CLR_CMP2</i>	OCPRE_CLR信号源为CMP2
<i>TIMER_OCPRE_CLR_CMP3</i>	OCPRE_CLR信号源为CMP3
输出参数{out}	
-	-

返回值	
-	-

例如：

```
/* configure the TIMER0 OCPRE_CLR source is CMP0 */
```

```
timer_ocpre_clr_source_select(TIMER0, TIMER_OCPRE_CLR_CMP0);
```

### 函数 timer\_ocpre\_clr\_int\_source\_select

函数timer\_ocpre\_clr\_int\_source\_select描述见下表：

**表 3-799. 函数 timer\_ocpre\_clr\_int\_source\_select**

函数名称	timer_ocpre_clr_int_source_select
函数原型	void timer_ocpre_clr_int_source_select(uint32_t timer_periph, uint32_t selection);
功能描述	配置外设TIMERx的OCPRE_CLR_INT信号源
先决条件	-
被调用函数	-
输入参数{in}	
<b>timer_periph</b>	TIMER外设
<i>TIMERx(x=0,7)</i>	参考具体参数
输入参数{in}	
<b>selection</b>	OCPRE_CLR_INT信号连接
<i>TIMER_SMCFG_OCPRE_CLR_INPUT</i>	OCPRE_CLR_INT信号与OCPRE_CLR输入连接
<i>TIMER_SMCFG_OCPRE_ETI</i>	OCPRE_CLR_INT信号与ETIF连接
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* configure the TIMER0 OCPRE_CLR_INT is connected to the OCPRE_CLR input */
```

```
timer_ocpre_clr_int_source_select(TIMER0, TIMER_SMCFG_OCPRE_CLR_INPUT);
```

### 函数 timer\_channel\_composite\_asymmetric\_pwm\_level\_config

函数timer\_channel\_composite\_asymmetric\_pwm\_level\_config描述见下表：

**表 3-800. 函数 timer\_channel\_composite\_asymmetric\_pwm\_level\_config**

函数名称	timer_channel_composite_asymmetric_pwm_level_config
函数原型	void timer_channel_composite_asymmetric_pwm_level_config(uint32_t

	timer_periph, uint16_t channel, ControlStatus newvalue);
功能描述	配置外设TIMERx的复合PWM和非对称PWM的比较时刻
先决条件	-
被调用函数	-
输入参数{in}	
timer_periph	TIMER外设
TIMERx(x=0,7)	参考具体参数
输入参数{in}	
channel	TIMER通道
TIMER_CH_0	通道0, TIMERx(x=0,7)
TIMER_CH_1	通道1, TIMERx(x=0,7)
TIMER_CH_2	通道2, TIMERx(x=0,7)
TIMER_CH_3	通道3, TIMERx(x=0,7)
输入参数{in}	
newvalue	控制状态
ENABLE	使能
DISABLE	禁能
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* configure TIMER0 CH0 period point match moment OxCPre level in the composite
PWM or asymmetric PWM mode */
```

```
timer_channel_composite_asymmetric_pwm_level_config (TIMER0, TIMER_CH_0, ENAB
LE);
```

### 函数 timer\_dma\_enable

函数timer\_dma\_enable描述见下表:

表 3-801. 函数 timer\_dma\_enable

函数名称	timer_dma_enable
函数原型	void timer_dma_enable(uint32_t timer_periph, uint32_t dma);
功能描述	外设TIMER的DMA使能
先决条件	-
被调用函数	-
输入参数{in}	
timer_periph	TIMER外设
TIMERx(x=0,1,2,7)	参考具体参数
输入参数{in}	

<b>dma</b>	<b>DMA源</b>
<i>TIMER_DMA_UPD</i>	更新DMA请求, <i>TIMERx</i> ( <i>x</i> =0,1,2,7)
<i>TIMER_DMA_CH0</i> <i>D</i>	通道0比较/捕获DMA请求, <i>TIMERx</i> ( <i>x</i> =0,1,2,7)
<i>TIMER_DMA_CH1</i> <i>D</i>	通道1比较/捕获DMA请求, <i>TIMERx</i> ( <i>x</i> =0,1,2,7)
<i>TIMER_DMA_CH2</i> <i>D</i>	通道2比较/捕获DMA请求, <i>TIMERx</i> ( <i>x</i> =0,1,2,7)
<i>TIMER_DMA_CH3</i> <i>D</i>	通道3比较/捕获DMA请求, <i>TIMERx</i> ( <i>x</i> =0,1,2,7)
<i>TIMER_DMA_CMT</i> <i>D</i>	换相DMA更新请求, <i>TIMERx</i> ( <i>x</i> =0,1,2,7)
<i>TIMER_DMA_TRG</i> <i>D</i>	触发DMA请求, <i>TIMER</i> ( <i>x</i> =0,1,2,7)
<i>TIMER_DMA_MCH</i> <i>0D</i>	多模式通道0比较/捕获DMA请求, <i>TIMERx</i> ( <i>x</i> =0,7)
<i>TIMER_DMA_MCH</i> <i>1D</i>	多模式通道1比较/捕获DMA请求, <i>TIMERx</i> ( <i>x</i> =0,7)
<i>TIMER_DMA_MCH</i> <i>2D</i>	多模式通道2比较/捕获DMA请求, <i>TIMERx</i> ( <i>x</i> =0,7)
<i>TIMER_DMA_MCH</i> <i>3D</i>	多模式通道3比较/捕获DMA请求, <i>TIMERx</i> ( <i>x</i> =0,7)
<b>输出参数{out}</b>	
-	-
<b>返回值</b>	
-	-

例如:

```
/* enable the TIMER0 update DMA */
```

```
timer_dma_enable(TIMER0, TIMER_DMA_UPD);
```

### 函数 timer\_dma\_disable

函数timer\_dma\_disable描述见下表:

**表 3-802. 函数 timer\_dma\_disable**

<b>函数名称</b>	timer_dma_disable
<b>函数原型</b>	void timer_dma_disable (uint32_t timer_periph, uint32_t dma);
<b>功能描述</b>	外设TIMER的DMA禁能
<b>先决条件</b>	-
<b>被调用函数</b>	-
<b>输入参数{in}</b>	

<b>timer_periph</b>	TIMER外设
<i>TIMERx(x=0,1,2,7)</i>	参考具体参数
<b>输入参数{in}</b>	
<b>dma</b>	DMA源
<i>TIMER_DMA_UPD</i>	更新DMA请求, TIMERx(x=0,1,2,7)
<i>TIMER_DMA_CH0</i> <i>D</i>	通道0比较/捕获DMA请求, TIMERx(x=0,1,2,7)
<i>TIMER_DMA_CH1</i> <i>D</i>	通道1比较/捕获DMA请求, TIMERx(x=0,1,2,7)
<i>TIMER_DMA_CH2</i> <i>D</i>	通道2比较/捕获DMA请求, TIMERx(x=0,1,2,7)
<i>TIMER_DMA_CH3</i> <i>D</i>	通道3比较/捕获DMA请求, TIMERx(x=0,1,2,7)
<i>TIMER_DMA_CMT</i> <i>D</i>	换相DMA更新请求, TIMERx(x=0,1,2,7)
<i>TIMER_DMA_TRG</i> <i>D</i>	触发DMA请求, TIMERx(x=0,1,2,7)
<i>TIMER_DMA_MCH</i> <i>0D</i>	多模式通道0比较/捕获DMA请求, TIMERx(x=0,7)
<i>TIMER_DMA_MCH</i> <i>1D</i>	多模式通道1比较/捕获DMA请求, TIMERx(x=0,7)
<i>TIMER_DMA_MCH</i> <i>2D</i>	多模式通道2比较/捕获DMA请求, TIMERx(x=0,7)
<i>TIMER_DMA_MCH</i> <i>3D</i>	多模式通道3比较/捕获DMA请求, TIMERx(x=0,7)
<b>输出参数{out}</b>	
-	-
<b>返回值</b>	
-	-

例如:

```
/* disable the TIMER0 update DMA */
```

```
timer_dma_disable(TIMER0, TIMER_DMA_UPD);
```

### 函数 timer\_channel\_dma\_request\_source\_select

函数timer\_channel\_dma\_request\_source\_select描述见下表:

**表 3-803. 函数 timer\_channel\_dma\_request\_source\_select**

<b>函数名称</b>	timer_channel_dma_request_source_select
<b>函数原型</b>	void timer_channel_dma_request_source_select(uint32_t timer_periph, uint32_t dma_request);

功能描述	外设TIMER的通道DMA请求源选择
先决条件	-
被调用函数	-
输入参数{in}	
timer_periph	TIMER外设
TIMERx(x=0,1,2,7)	TIMER外设选择
输入参数{in}	
dma_request	通道的DMA请求源选择
TIMER_DMAREQUEST_CHANNELEVENT	当通道捕获/比较事件发生时，发送通道n的DMA请求
TIMER_DMAREQUEST_UPDATEEVENT	当更新事件发生，发送通道n的DMA请求
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* TIMER0 channel DMA request of channel n is sent when channel event occurs */
```

```
timer_channel_dma_request_source_select (TIMER0, TIMER_DMAREQUEST_CHANNEL  
EVENT);
```

### 函数 timer\_dma\_transfer\_config

函数timer\_dma\_transfer\_config描述见下表：

表 3-804. 函数 timer\_dma\_transfer\_config

函数名称	timer_dma_transfer_config
函数原型	void timer_dma_transfer_config(uint32_t timer_periph, uint32_t dma_baseaddr, uint32_t dma_lenth);
功能描述	配置外设TIMER的DMA模式
先决条件	-
被调用函数	-
输入参数{in}	
timer_periph	TIMER外设
TIMERx(x=0,1,2,7)	参考具体参数
输入参数{in}	
dma_baseaddr	DMA传输起始地址
TIMER_DMACFG_DMATA_CTL0	DMA传输起始地址：TIMER_CTL0, TIMERx(x=0,1,2,7)



<i>TIMER_DMACFG_DMATA_CTL1</i>	DMA传输起始地址: TIMER_CTL1, TIMERx(x=0,1,2,7)
<i>TIMER_DMACFG_DMATA_SMCFG</i>	DMA传输起始地址: TIMER_SMCFG, TIMERx(x=0,1,2,7)
<i>TIMER_DMACFG_DMATA_DMAINTEN</i>	DMA传输起始地址: TIMER_DMAINTEN, TIMERx(x=0,1,2,7)
<i>TIMER_DMACFG_DMATA_INTF</i>	DMA传输起始地址: TIMER_INTF, TIMERx(x=0,1,2,7)
<i>TIMER_DMACFG_DMATA_SWEVG</i>	DMA传输起始地址: TIMER_SWEVG, TIMERx(x=0,1,2,7)
<i>TIMER_DMACFG_DMATA_CHCTL0</i>	DMA传输起始地址: TIMER_CHCTL0, TIMERx(x=0,1,2,7)
<i>TIMER_DMACFG_DMATA_CHCTL1</i>	DMA传输起始地址: TIMER_CHCTL1, TIMERx(x=0,1,2,7)
<i>TIMER_DMACFG_DMATA_CHCTL2</i>	DMA传输起始地址: TIMER_CHCTL2, TIMERx(x=0,1,2,7)
<i>TIMER_DMACFG_DMATA_CNT</i>	DMA传输起始地址: TIMER_CNT, TIMERx(x=0,1,2,7)
<i>TIMER_DMACFG_DMATA_PSC</i>	DMA传输起始地址: TIMER_PSC, TIMERx(x=0,1,2,7)
<i>TIMER_DMACFG_DMATA_CAR</i>	DMA传输起始地址: TIMER_CAR, TIMERx(x=0,1,2,7)
<i>TIMER_DMACFG_DMATA_CREP</i>	DMA传输起始地址: TIMER_CREP0, TIMERx(x=0,7)
<i>TIMER_DMACFG_DMATA_CH0CV</i>	DMA传输起始地址: TIMER_CH0CV, TIMERx(x=0,1,2,7)
<i>TIMER_DMACFG_DMATA_CH1CV</i>	DMA传输起始地址: TIMER_CH1CV, TIMERx(x=0,1,2,7)
<i>TIMER_DMACFG_DMATA_CH2CV</i>	DMA传输起始地址: TIMER_CH2CV, TIMERx(x=0,1,2,7)
<i>TIMER_DMACFG_DMATA_CH3CV</i>	DMA传输起始地址: TIMER_CH3CV, TIMERx(x=0,1,2,7)
<i>TIMER_DMACFG_DMATA_CCHP</i>	DMA传输起始地址: TIMER_CCHP, TIMERx(x=0,7)
<i>TIMER_DMACFG_DMATA_MCHCTL0</i>	DMA传输起始地址: TIMER_MCHCTL0, TIMERx(x=0,7)
<i>TIMER_DMACFG_DMATA_MCHCTL1</i>	DMA传输起始地址: TIMER_MCHCTL1, TIMERx(x=0,7)
<i>TIMER_DMACFG_DMATA_MCHCTL2</i>	DMA传输起始地址: TIMER_MCHCTL2, TIMERx(x=0,7)
<i>TIMER_DMACFG_DMATA_MCH0CV</i>	DMA传输起始地址: TIMER_MCH0CV, TIMERx(x=0,7)

<i>DMATA_MCH0CV</i>	
<i>TIMER_DMACFG_DMATA_MCH1CV</i>	DMA传输起始地址: <i>TIMER_MCH1CV</i> , <i>TIMERx</i> ( <i>x</i> =0,7)
<i>TIMER_DMACFG_DMATA_MCH2CV</i>	DMA传输起始地址: <i>TIMER_MCH2CV</i> , <i>TIMERx</i> ( <i>x</i> =0,7)
<i>TIMER_DMACFG_DMATA_MCH3CV</i>	DMA传输起始地址: <i>TIMER_MCH3CV</i> , <i>TIMERx</i> ( <i>x</i> =0,7)
<i>TIMER_DMACFG_DMATA_CH0COMV_ADD</i>	DMA传输起始地址: <i>TIMER_CH0COMV_ADD</i> , <i>TIMERx</i> ( <i>x</i> =0,7)
<i>TIMER_DMACFG_DMATA_CH1COMV_ADD</i>	DMA传输起始地址: <i>TIMER_CH1COMV_ADD</i> , <i>TIMERx</i> ( <i>x</i> =0,7)
<i>TIMER_DMACFG_DMATA_CH2COMV_ADD</i>	DMA传输起始地址: <i>TIMER_CH2COMV_ADD</i> , <i>TIMERx</i> ( <i>x</i> =0,7)
<i>TIMER_DMACFG_DMATA_CH3COMV_ADD</i>	DMA传输起始地址: <i>TIMER_CH3COMV_ADD</i> , <i>TIMERx</i> ( <i>x</i> =0,7)
<i>TIMER_DMACFG_DMATA_CTL2</i>	DMA传输起始地址: <i>TIMER_CTL2</i> , <i>TIMERx</i> ( <i>x</i> =0,7)
<i>TIMER_DMACFG_DMATA_AFCTL</i>	DMA传输起始地址: <i>TIMER_AFCTL</i> , <i>TIMERx</i> ( <i>x</i> =0,7)
<i>TIMER_DMACFG_DMATA_IREP</i>	DMA传输起始地址: <i>TIMER_IREP</i> , <i>TIMERx</i> ( <i>x</i> =0,7)
<i>TIMER_DMACFG_DMATA_ADCRCTL</i>	DMA传输起始地址: <i>TIMER_ADCRCTL</i> , <i>TIMERx</i> ( <i>x</i> =0,7)
<i>TIMER_DMACFG_DMATA_ADCCR1</i>	DMA传输起始地址: <i>TIMER_ADCCR1</i> , <i>TIMERx</i> ( <i>x</i> =0,7)
<i>TIMER_DMACFG_DMATA_ADCCR2</i>	DMA传输起始地址: <i>TIMER_ADCCR2</i> , <i>TIMERx</i> ( <i>x</i> =0,7)
<i>TIMER_DMACFG_DMATA_ADCREP</i>	DMA传输起始地址: <i>TIMER_ADCREP</i> , <i>TIMERx</i> ( <i>x</i> =0,7)
<i>TIMER_DMACFG_DMATA_ADCTL</i>	DMA传输起始地址: <i>TIMER_ADCTL</i> , <i>TIMERx</i> ( <i>x</i> =0,7)
<i>TIMER_DMACFG_DMATA_SHUPM0</i>	DMA传输起始地址: <i>TIMER_SHUPM0</i> , <i>TIMERx</i> ( <i>x</i> =0,7)
<i>TIMER_DMACFG_DMATA_SHUPM1</i>	DMA传输起始地址: <i>TIMER_SHUPM1</i> , <i>TIMERx</i> ( <i>x</i> =0,7)
<i>TIMER_DMACFG_DMATA_CNT_FLOW_SEL_W_SEL</i>	DMA传输起始地址: <i>TIMER_CNT_FLOW_SEL</i> , <i>TIMERx</i> ( <i>x</i> =0,7)

输入参数{in}	
<b>dma_lenth</b>	DMA传输长度
<i>TIMER_DMACFG_DMATC_xTRANSFER</i>	(x=1~52), DMA传输x次
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* configure the TIMER0 DMA transfer */
```

```
timer_dma_transfer_config(TIMER0, TIMER_DMACFG_DMATA_CTL0, TIMER_DMACFG_DMATC_5TRANSFER);
```

### 函数 timer\_event\_software\_generate

函数timer\_event\_software\_generate描述见下表:

表 3-805. 函数 timer\_event\_software\_generate

函数名称	timer_event_software_generate
函数原型	void timer_event_software_generate(uint32_t timer_periph, uint32_t event);
功能描述	软件产生事件
先决条件	-
被调用函数	-
输入参数{in}	
<b>timer_periph</b>	TIMER外设
<i>TIMERx(x=0,1,2,7)</i>	参考具体参数
输入参数{in}	
<b>event</b>	事件源
<i>TIMER_EVENT_SRC_UPG</i>	更新事件产生, TIMERx(x=0,1,2,7)
<i>TIMER_EVENT_SRC_CH0G</i>	通道0捕获或比较事件发生, TIMERx(x=0,1,2,7)
<i>TIMER_EVENT_SRC_CH1G</i>	通道1捕获或比较事件发生, TIMERx(x=0,1,2,7)
<i>TIMER_EVENT_SRC_CH2G</i>	通道2捕获或比较事件发生, TIMERx(x=0,1,2,7)
<i>TIMER_EVENT_SRC_CH3G</i>	通道3捕获或比较事件发生, TIMERx(x=0,1,2,7)
<i>TIMER_EVENT_SRC_CMTG</i>	通道换相更新事件发生, TIMERx(x=0,7)

TIMER_EVENT_SR C_TRGG	触发事件产生, TIMERx(x=0,1,2,7)
TIMER_EVENT_SR C_BRKG	产生BREAK事件, TIMERx(x=0,7)
TIMER_EVENT_SR C_MCH0G	多模式通道0捕获或比较事件发生, TIMERx(x=0,7)
TIMER_EVENT_SR C_MCH1G	多模式通道1捕获或比较事件发生, TIMERx(x=0,7)
TIMER_EVENT_SR C_MCH2G	多模式通道2捕获或比较事件发生, TIMERx(x=0,7)
TIMER_EVENT_SR C_MCH3G	多模式通道3捕获或比较事件发生, TIMERx(x=0,7)
TIMER_EVENT_SR C_CH0COMADDG	通道0附加比较事件发生, TIMERx(x=0,7)
TIMER_EVENT_SR C_CH1COMADDG	通道1附加比较事件发生, TIMERx(x=0,7)
TIMER_EVENT_SR C_CH2COMADDG	通道2附加比较事件发生, TIMERx(x=0,7)
TIMER_EVENT_SR C_CH3COMADDG	通道3附加比较事件发生, TIMERx(x=0,7)
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* software generate update event*/
```

```
timer_event_software_generate(TIMER0, TIMER_EVENT_SRC_UPG);
```

### 函数 timer\_break\_struct\_para\_init

函数timer\_break\_struct\_para\_init描述见下表:

表 3-806. 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 <a href="#">结构体timer_break_parameter_struct</a> 。
输出参数{out}	

-	-
返回值	
-	-

例如：

```
/* initialize TIMER break parameter struct with a default value */
```

```
timer_break_parameter_struct timer_breakpara;
```

```
timer_break_struct_para_init(&timer_breakpara);
```

### 函数 timer\_break\_config

函数timer\_break\_config描述见下表：

表 3-807. 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
TIMEx(x=0,7)	TIMER外设选择
输入参数{in}	
breakpara	中止功能配置结构体，详见 <a href="#">结构体timer_break_parameter_struct</a> 。
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* configure TIMER0 break function */
```

```
timer_break_parameter_struct timer_breakpara;
```

```
timer_breakpara.runoffstate = TIMER_ROS_STATE_DISABLE;
```

```
timer_breakpara.ideloffstate = TIMER_IOS_STATE_DISABLE;
```

```
timer_breakpara.deadtime = 0U;
```

```
timer_breakpara.outputautostate = TIMER_OUTAUTO_ENABLE;
```

```
timer_breakpara.protectmode = TIMER_CCHP_PROT_OFF;
```

```
timer_breakpara.breakstate = TIMER_BREAK0_ENABLE;
```

```

timer_breakpara.breakfilter      = 0U;

timer_breakpara.breakpolarity    = TIMER_BREAK0_POLARITY_LOW;

timer_break_config(TIMERO, &timer_breakpara);

```

### 函数 timer\_break\_enable

函数timer\_break\_enable描述见下表：

**表 3-808. 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TIMER的中止功能
先决条件	只有在TIMERx_CCHP寄存器的PROT [1:0] =00 时，才可修改
被调用函数	-
输入参数{in}	
timer_periph	TIMER外设
TIMERx(x=0,7)	TIMER外设选择
输出参数{out}	
-	-
返回值	
-	-

例如：

```

/* enable TIMERO BREAK function*/

timer_break_enable(TIMERO);

```

### 函数 timer\_break\_disable

函数timer\_break\_disable描述见下表：

**表 3-809. 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TIMER的中止功能
先决条件	只有在TIMERx_CCHP寄存器的PROT [1:0] =00 时，才可修改
被调用函数	-
输入参数{in}	
timer_periph	TIMER外设
TIMERx(x=0,7)	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

函数timer\_automatic\_output\_enable描述见下表:

表 3-810. 函数 timer\_automatic\_output\_enable

函数名称	timer_automatic_output_enable
函数原型	void timer_automatic_output_enable(uint32_t timer_periph);
功能描述	自动输出使能
先决条件	只有在TIMERx_CCHP寄存器的PROT [1:0] =00 时，才可修改
被调用函数	-
输入参数{in}	
timer_periph	TIMER外设
TIMERx(x=0,7)	TIMER外设选择
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* enable TIMER0 output automatic function */
```

```
timer_automatic_output_enable(TIMER0);
```

### 函数 timer\_automatic\_output\_disable

函数timer\_automatic\_output\_disable描述见下表:

表 3-811. 函数 timer\_automatic\_output\_disable

函数名称	timer_automatic_output_disable
函数原型	void timer_automatic_output_disable (uint32_t timer_periph);
功能描述	自动输出禁能
先决条件	只有在TIMERx_CCHP寄存器的PROT [1:0] = 00时，才可修改
被调用函数	-
输入参数{in}	
timer_periph	TIMER外设
TIMERx(x=0,7)	TIMER外设选择
输出参数{out}	
-	-

返回值	
-	-

例如：

```
/* disable TIMER0 output automatic function */
```

```
timer_automatic_output_disable(TIMER0);
```

### 函数 timer\_primary\_output\_config

函数timer\_primary\_output\_config描述见下表：

**表 3-812. 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}	
<b>timer_periph</b>	TIMER外设
<i>TIMERx(x=0,7)</i>	TIMER外设选择
输入参数{in}	
<b>newvalue</b>	控制状态
<i>ENABLE</i>	使能
<i>DISABLE</i>	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\_output\_struct\_para\_init

函数timer\_channel\_output\_struct\_para\_init描述见下表：

**表 3-813. 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 <a href="#">结构体timer_oc_parameter_struct</a> 。
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 3-814. 函数 timer\_channel\_output\_config

函数名称	timer_channel_output_config
函数原型	void timer_channel_output_config(uint32_t timer_periph, uint16_t channel, timer_oc_parameter_struct* ocpara);
功能描述	外设TIMER的通道输出配置
先决条件	-
被调用函数	-
输入参数{in}	
timer_periph	TIMER外设
TIMERx(x=0,1,2,7)	参考具体参数
输入参数{in}	
channel	TIMER通道
TIMER_CH_0	通道0
TIMER_CH_1	通道1
TIMER_CH_2	通道2
TIMER_CH_3	通道3
输入参数{in}	
ocpara	输出通道结构体，详见 <a href="#">结构体timer_oc_parameter_struct</a> 。
输出参数{out}	
-	-
返回值	
-	-

例如：

```

/* configure TIMER0 channel 0 output function */

timer_oc_parameter_struct timer_ocinitpara;

timer_ocinitpara.outputstate = TIMER_CCX_ENABLE;

timer_ocinitpara.outputnstate = TIMER_CCXN_ENABLE;

timer_ocinitpara.ocpolarity = TIMER_OC_POLARITY_HIGH;

timer_ocinitpara.ocnpolarity = TIMER_OCN_POLARITY_HIGH;

timer_ocinitpara.ocidlestate = TIMER_OC_IDLE_STATE_HIGH;

timer_ocinitpara.ocnidlestate = TIMER_OCN_IDLE_STATE_LOW;

timer_channel_output_config(TIMER0, TIMER_CH_0, & timer_ocinitpara);

```

### 函数 timer\_channel\_output\_mode\_config

函数timer\_channel\_output\_mode\_config描述见下表:

**表 3-815. 函数 timer\_channel\_output\_mode\_config**

函数名称	timer_channel_output_mode_config
函数原型	void timer_channel_output_mode_config(uint32_t timer_periph, uint16_t channel, uint32_t ocmode);
功能描述	配置外设TIMER通道输出比较模式
先决条件	-
被调用函数	-
输入参数{in}	
<b>timer_periph</b>	TIMER外设
TIMERx(x=0,1,2,7)	参考具体参数
输入参数{in}	
<b>channel</b>	TIMER通道
TIMER_CH_0	通道0, TIMERx(x=0,1,2,7)
TIMER_CH_1	通道1, TIMERx(x=0,1,2,7)
TIMER_CH_2	通道2, TIMERx(x=0,1,2,7)
TIMER_CH_3	通道3, TIMERx(x=0,1,2,7)
TIMER_MCH_0	多模式通道0, TIMERx (x=0,7)
TIMER_MCH_1	多模式通道1, TIMERx (x=0,7)
TIMER_MCH_2	多模式通道2, TIMERx (x=0,7)
TIMER_MCH_3	多模式通道3, TIMERx (x=0,7)
输入参数{in}	
<b>ocmode</b>	通道输出比较模式, TIMERx(x=0,1,2,7)
TIMER_OC_MODE_TIMING	时基模式, TIMERx(x=0,1,2,7)
TIMER_OC_MODE	匹配时设置为高, TIMERx(x=0,1,2,7)

<code>_ACTIVE</code>	
<code>TIMER_OC_MODE_INACTIVE</code>	匹配时设置为低, <code>TIMERx(x=0,1,2,7)</code>
<code>TIMER_OC_MODE_TOGGLE</code>	匹配时翻转, <code>TIMERx(x=0,1,2,7)</code>
<code>TIMER_OC_MODE_LOW</code>	强制为低, <code>TIMERx(x=0,1,2,7)</code>
<code>TIMER_OC_MODE_HIGH</code>	强制为高, <code>TIMERx(x=0,1,2,7)</code>
<code>TIMER_OC_MODE_PWM0</code>	PWM模式0, <code>TIMERx(x=0,1,2,7)</code>
<code>TIMER_OC_MODE_PWM1</code>	PWM模式1, <code>TIMERx(x=0,1,2,7)</code>
<code>TIMER_OC_MODE_COMPOSITE_PWM0</code>	复合PWM模式0, <code>TIMERx(x=0,7)</code>
<code>TIMER_OC_MODE_COMPOSITE_PWM1</code>	复合PWM模式1, <code>TIMERx(x=0,7)</code>
<code>TIMER_OC_MODE_ASYMMETRIC_PWM0</code>	非对称PWM模式0, <code>TIMERx(x=0,7)</code>
<code>TIMER_OC_MODE_ASYMMETRIC_PWM1</code>	非对称PWM模式1, <code>TIMERx(x=0,7)</code>
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* configure TIMER0 channel PWM 0 mode */
```

```
timer_channel_output_mode_config(TIMER0,TIMER_CH_0,TIMER_OC_MODE_PWM0);
```

### 函数 `timer_channel_output_pulse_value_config`

函数`timer_channel_output_pulse_value_config`描述见下表:

**表 3-816. 函数 `timer_channel_output_pulse_value_config`**

函数名称	<code>timer_channel_output_pulse_value_config</code>
函数原型	<code>void timer_channel_output_pulse_value_config(uint32_t timer_periph, uint16_t channel, uint32_t pulse);</code>
功能描述	配置外设TIMER的通道输出比较值

先决条件	-
被调用函数	-
输入参数{in}	
timer_periph	TIMER外设
TIMERx(x=0,1,2,7)	参考具体参数
输入参数{in}	
channel	TIMER通道
TIMER_CH_0	通道0, TIMERx(x=0,1,2,7)
TIMER_CH_1	通道1, TIMERx(x=0,1,2,7)
TIMER_CH_2	通道2, TIMERx(x=0,1,2,7)
TIMER_CH_3	通道3, TIMERx(x=0,1,2,7)
TIMER_MCH_0	多模式通道0, TIMERx (x=0,7)
TIMER_MCH_1	多模式通道1, TIMERx (x=0,7)
TIMER_MCH_2	多模式通道2, TIMERx (x=0,7)
TIMER_MCH_3	多模式通道3, TIMERx (x=0,7)
输入参数{in}	
pulse	通道输出比较值 (0~65535)
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* configure TIMER0 channel 0 output pulse value */
```

```
timer_channel_output_pulse_value_config(TIMER0, TIMER_CH_0, 399);
```

### 函数 timer\_channel\_output\_shadow\_config

函数timer\_channel\_output\_shadow\_config描述见下表:

表 3-817. 函数 timer\_channel\_output\_shadow\_config

函数名称	timer_channel_output_shadow_config
函数原型	void timer_channel_output_shadow_config(uint32_t timer_periph, uint16_t channel, uint16_t ocshadow);
功能描述	配置TIMER通道输出比较影子寄存器功能
先决条件	-
被调用函数	-
输入参数{in}	
timer_periph	TIMER外设
TIMERx(x=0,1,2,7)	参考具体参数
输入参数{in}	
channel	TIMER通道

<i>TIMER_CH_0</i>	通道0, <i>TIMERx</i> ( <i>x</i> =0,1,2,7)
<i>TIMER_CH_1</i>	通道1, <i>TIMERx</i> ( <i>x</i> =0,1,2,7)
<i>TIMER_CH_2</i>	通道2, <i>TIMERx</i> ( <i>x</i> =0,1,2,7)
<i>TIMER_CH_3</i>	通道3, <i>TIMERx</i> ( <i>x</i> =0,1,2,7)
<i>TIMER_MCH_0</i>	多模式通道0, <i>TIMERx</i> ( <i>x</i> =0,7)
<i>TIMER_MCH_1</i>	多模式通道1, <i>TIMERx</i> ( <i>x</i> =0,7)
<i>TIMER_MCH_2</i>	多模式通道2, <i>TIMERx</i> ( <i>x</i> =0,7)
<i>TIMER_MCH_3</i>	多模式通道3, <i>TIMERx</i> ( <i>x</i> =0,7)
输入参数{in}	
<b>ocshadow</b>	输出比较影子寄存器功能状态
<i>TIMER_OC_SHADOW_ENABLE</i>	通道输出比较影子寄存器使能
<i>TIMER_OC_SHADOW_DISABLE</i>	通道输出比较影子寄存器禁能
<i>TIMER_OMC_SHADOW_ENABLE</i>	多模式通道输出比较影子寄存器使能
<i>TIMER_OMC_SHADOW_DISABLE</i>	多模式通道输出比较影子寄存器禁能
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/*configure TIMER0 channel 0 output shadow function */
```

```
timer_channel_output_shadow_config(TIMER0, TIMER_CH_0, TIMER_OC_SHADOW_ENABLE);
```

### 函数 timer\_channel\_output\_compare\_fast\_config

函数timer\_channel\_output\_compare\_fast\_config描述见下表:

表 3-818. 函数 timer\_channel\_output\_compare\_fast\_config

函数名称	timer_channel_output_compare_fast_config
函数原型	void timer_channel_output_compare_fast_config (uint32_t timer_periph, uint16_t channel, uint16_t ocfast);
功能描述	配置TIMER通道输出比较快速功能
先决条件	-
被调用函数	-
输入参数{in}	
<b>timer_periph</b>	TIMER外设
<i>TIMERx</i> ( <i>x</i> =0,1,2,7)	参考具体参数

输入参数{in}	
<b>channel</b>	TIMER通道
<i>TIMER_CH_0</i>	通道0, TIMERx(x=0,1,2,7)
<i>TIMER_CH_1</i>	通道1, TIMERx(x=0,1,2,7)
<i>TIMER_CH_2</i>	通道2, TIMERx(x=0,1,2,7)
<i>TIMER_CH_3</i>	通道3, TIMERx(x=0,1,2,7)
<i>TIMER_MCH_0</i>	多模式通道0, TIMERx (x=0,7)
<i>TIMER_MCH_1</i>	多模式通道1, TIMERx (x=0,7)
<i>TIMER_MCH_2</i>	多模式通道2, TIMERx (x=0,7)
<i>TIMER_MCH_3</i>	多模式通道3, TIMERx (x=0,7)
输入参数{in}	
<b>ocfast</b>	输出比较快速功能状态
<i>TIMER_OC_FAST_ENABLE</i>	通道输出比较快速使能
<i>TIMER_OC_FAST_DISABLE</i>	通道输出比较快速禁能
<i>TIMER_OMC_FAST_ENABLE</i>	多模式通道输出比较快速使能
<i>TIMER_OMC_FAST_DISABLE</i>	多模式通道输出比较快速禁能
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/*configure TIMER0 channel 0 output compare fast function */
```

```
timer_channel_output_compare_fast_config(TIMER0, TIMER_CH_0, TIMER_OC_FAST_ENABLE);
```

### 函数 timer\_channel\_output\_clear\_config

函数timer\_channel\_output\_clear\_config描述见下表:

表 3-819. 函数 timer\_channel\_output\_clear\_config

函数名称	timer_channel_output_clear_config
函数原型	void timer_channel_output_clear_config(uint32_t timer_periph, uint16_t channel, uint16_t occlear);
功能描述	配置TIMER的通道输出比较清0功能
先决条件	-
被调用函数	-
输入参数{in}	

<b>timer_periph</b>	TIMER外设
<i>TIMERx(x=0,1,2,7)</i>	参考具体参数
<b>输入参数{in}</b>	
<b>channel</b>	TIMER通道
<i>TIMER_CH_0</i>	通道0, TIMERx(x=0,1,2,7)
<i>TIMER_CH_1</i>	通道1, TIMERx(x=0,1,2,7)
<i>TIMER_CH_2</i>	通道2, TIMERx(x=0,1,2,7)
<i>TIMER_CH_3</i>	通道3, TIMERx(x=0,1,2,7)
<i>TIMER_MCH_0</i>	多模式通道0, TIMERx(x=0,7)
<i>TIMER_MCH_1</i>	多模式通道1, TIMERx (x=0,7)
<i>TIMER_MCH_2</i>	多模式通道2, TIMERx (x=0,7)
<i>TIMER_MCH_3</i>	多模式通道3, TIMERx (x=0,7)
<b>输入参数{in}</b>	
<b>occlear</b>	通道比较输出清0功能状态
<i>TIMER_OC_CLEAR_ENABLE</i>	通道比较输出清0功能使能
<i>TIMER_OC_CLEAR_DISABLE</i>	通道比较输出清0功能禁能
<i>TIMER_OMC_CLEAR_ENABLE</i>	多模式通道比较输出清0功能使能
<i>TIMER_OMC_CLEAR_DISABLE</i>	多模式通道比较输出清0功能禁能
<b>输出参数{out}</b>	
-	-
<b>返回值</b>	
-	-

例如:

```
/* configure TIMER0 channel 0 output clear function */
```

```
timer_channel_output_clear_config(TIMER0, TIMER_CH_0, TIMER_OC_CLEAR_ENABLE);
```

### 函数 timer\_channel\_output\_polarity\_config

函数timer\_channel\_output\_polarity\_config描述见下表:

表 3-820. 函数 timer\_channel\_output\_polarity\_config

<b>函数名称</b>	timer_channel_output_polarity_config
<b>函数原型</b>	void timer_channel_output_polarity_config(uint32_t timer_periph, uint16_t channel, uint16_t ocpolarity);
<b>功能描述</b>	通道输出极性配置
<b>先决条件</b>	-

被调用函数	-
输入参数{in}	
timer_periph	TIMER外设
TIMERx(x=0,1,2,7)	参考具体参数
输入参数{in}	
channel	TIMER通道
TIMER_CH_0	通道0, TIMERx(x=0,1,2,7)
TIMER_CH_1	通道1, TIMERx(x=0,1,2,7)
TIMER_CH_2	通道2, TIMERx(x=0,1,2,7)
TIMER_CH_3	通道3, TIMERx(x=0,1,2,7)
TIMER_MCH_0	多模式通道0, TIMERx (x=0,7)
TIMER_MCH_1	多模式通道1, TIMERx (x=0,7)
TIMER_MCH_2	多模式通道2, TIMERx (x=0,7)
TIMER_MCH_3	多模式通道3, TIMERx (x=0,7)
输入参数{in}	
ocpolarity	通道输出极性
TIMER_OC_POLARITY_HIGH	通道输出极性高电平有效
TIMER_OC_POLARITY_LOW	通道输出极性低电平有效
TIMER_OMC_POLARITY_HIGH	多模式通道输出极性高电平有效
TIMER_OMC_POLARITY_LOW	多模式通道输出极性低电平有效
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* configure TIMER0 channel 0 output polarity */
```

```
timer_channel_output_polarity_config(TIMER0, TIMER_CH_0, TIMER_OC_POLARITY_HIGH);
```

### 函数 timer\_channel\_complementary\_output\_polarity\_config

函数timer\_channel\_complementary\_output\_polarity\_config描述见下表:

表 3-821. 函数 timer\_channel\_complementary\_output\_polarity\_config

函数名称	timer_channel_complementary_output_polarity_config
函数原型	void timer_channel_complementary_output_polarity_config(uint32_t timer_periph, uint16_t channel, uint16_t ocnpolarity);



功能描述	互补通道输出极性配置
先决条件	-
被调用函数	-
输入参数{in}	
timer_periph	TIMER外设
TIMERx(x=0,7)	TIMER外设选择
输入参数{in}	
channel	TIMER通道
TIMER_CH_0	通道0
TIMER_CH_1	通道1
TIMER_CH_2	通道2
TIMER_CH_3	通道3
输入参数{in}	
ocpolarity	互补通道输出极性
TIMER_OCN_POLARITY_HIGH	互补通道输出极性高电平有效
TIMER_OCN_POLARITY_LOW	互补通道输出极性低电平有效
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* configure TIMER0 channel 0 complementary output polarity */
```

```
timer_channel_complementary_output_polarity_config(TIMER0, TIMER_CH_0, TIMER_OCN_POLARITY_HIGH);
```

### 函数 timer\_channel\_output\_state\_config

函数timer\_channel\_output\_state\_config描述见下表：

表 3-822. 函数 timer\_channel\_output\_state\_config

函数名称	timer_channel_output_state_config
函数原型	void timer_channel_output_state_config(uint32_t timer_periph, uint16_t channel, uint32_t state);
功能描述	配置通道状态
先决条件	-
被调用函数	-
输入参数{in}	
timer_periph	TIMER外设
TIMERx(x=0,1,2,7)	参考具体参数

输入参数{in}	
<b>channel</b>	TIMER通道
<i>TIMER_CH_0</i>	通道0, TIMERx(x=0,1,2,7)
<i>TIMER_CH_1</i>	通道1, TIMERx(x=0,1,2,7)
<i>TIMER_CH_2</i>	通道2, TIMERx(x=0,1,2,7)
<i>TIMER_CH_3</i>	通道3, TIMERx(x=0,1,2,7)
<i>TIMER_MCH_0</i>	多模式通道0, TIMERx (x=0,7)
<i>TIMER_MCH_1</i>	多模式通道1, TIMERx (x=0,7)
<i>TIMER_MCH_2</i>	多模式通道2, TIMERx (x=0,7)
<i>TIMER_MCH_3</i>	多模式通道3, TIMERx (x=0,7)
输入参数{in}	
<b>state</b>	通道状态
<i>TIMER_CCX_ENABLE</i>	通道使能
<i>TIMER_CCX_DISABLE</i>	通道禁能
<i>TIMER_MCCX_ENABLE</i>	多模式通道使能
<i>TIMER_MCCX_DISABLE</i>	多模式通道禁能
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* configure TIMER0 channel 0 enable state */
```

```
timer_channel_output_state_config(TIMER0, TIMER_CH_0, TIMER_CCX_ENABLE);
```

### 函数 timer\_channel\_complementary\_output\_state\_config

函数timer\_channel\_complementary\_output\_state\_config描述见下表:

表 3-823. 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}	
<b>timer_periph</b>	TIMER外设

<i>TIMERx(x=0,7)</i>	TIMER外设选择
输入参数{in}	
<b>channel</b>	TIMER通道
<i>TIMER_CH_0</i>	通道0
<i>TIMER_CH_1</i>	通道1
<i>TIMER_CH_2</i>	通道2
<i>TIMER_CH_3</i>	通道3
输入参数{in}	
<b>state</b>	互补通道状态
<i>TIMER_CCXN_ENABLE</i>	互补通道使能
<i>TIMER_CCXN_DISABLE</i>	互补通道禁能
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* configure TIMER0 channel 0 complementary output enable state */
```

```
timer_channel_complementary_output_state_config(TIMER0, TIMER_CH_0, TIMER_CCXN_ENABLE);
```

### 函数 timer\_channel\_input\_struct\_para\_init

函数timer\_channel\_input\_struct\_para\_init描述见下表：

表 3-824. 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}	
<b>icpara</b>	通道输入结构体，详见 <a href="#">结构体timer_ic_parameter_struct</a> 。
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 3-825. 函数 timer\_input\_capture\_config**

函数名称	timer_input_capture_config
函数原型	void timer_input_capture_config(uint32_t timer_periph, uint16_t channel, timer_ic_parameter_struct* icpara);
功能描述	配置TIMER输入捕获参数
先决条件	-
被调用函数	timer_channel_input_capture_prescaler_config
输入参数{in}	
timer_periph	TIMER外设
TIMERx(x=0,1,2,7)	参考具体参数
输入参数{in}	
channel	TIMER通道
TIMER_CH_0	通道0, TIMERx(x=0,1,2,7)
TIMER_CH_1	通道1, TIMERx(x=0,1,2,7)
TIMER_CH_2	通道2, TIMERx(x=0,1,2,7)
TIMER_CH_3	通道3, TIMERx(x=0,1,2,7)
TIMER_MCH_0	多模式通道0, TIMERx (x=0,7)
TIMER_MCH_1	多模式通道1, TIMERx (x=0,7)
TIMER_MCH_2	多模式通道2, TIMERx (x=0,7)
TIMER_MCH_3	多模式通道3, TIMERx (x=0,7)
输入参数{in}	
icpara	通道输入结构体，详见 <a href="#">结构体timer_ic_parameter_struct</a> 。
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* configure TIMER0 input capture parameter */
```

```
timer_ic_parameter_struct timer_icinitpara;
```

```
timer_icinitpara.icpolarity = TIMER_IC_POLARITY_RISING;
```

```
timer_icinitpara.icselection = TIMER_IC_SELECTION_DIRECTTTI;
```

```
timer_icinitpara.icprescaler = TIMER_IC_PSC_DIV1;
```

```
timer_icinitpara.icfilter      = 0x0;
```

```
timer_input_capture_config(TIMER0, TIMER_CH_0, &timer_icinitpara);
```

### 函数 timer\_channel\_input\_capture\_prescaler\_config

函数timer\_channel\_input\_capture\_prescaler\_config描述见下表：

**表 3-826. 函数 timer\_channel\_input\_capture\_prescaler\_config**

函数名称	timer_channel_input_capture_prescaler_config
函数原型	void timer_channel_input_capture_prescaler_config(uint32_t timer_periph, uint16_t channel, uint16_t prescaler);
功能描述	配置TIMER通道输入捕获预分频值
先决条件	-
被调用函数	-
输入参数{in}	
timer_periph	TIMER外设
TIMERx(x=0,1,2,7)	参考具体参数
输入参数{in}	
channel	TIMER通道
TIMER_CH_0	通道0, TIMERx(x=0,1,2,7)
TIMER_CH_1	通道1, TIMERx(x=0,1,2,7)
TIMER_CH_2	通道2, TIMERx(x=0,1,2,7)
TIMER_CH_3	通道3, TIMERx(x=0,1,2,7)
TIMER_MCH_0	多模式通道0, TIMERx (x=0,7)
TIMER_MCH_1	多模式通道1, TIMERx (x=0,7)
TIMER_MCH_2	多模式通道2, TIMERx (x=0,7)
TIMER_MCH_3	多模式通道3, TIMERx (x=0,7)
输入参数{in}	
prescaler	通道输入捕获预分频值
TIMER_IC_PSC_DIV1	不分频
TIMER_IC_PSC_DIV2	2分频
TIMER_IC_PSC_DIV4	4分频
TIMER_IC_PSC_DIV8	8分频
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* configure TIMER0 channel 0 input capture prescaler value */
```

```
timer_channel_input_capture_prescaler_config(TIMER0, TIMER_CH_0, TIMER_IC_PSC_DIV2);
```

### 函数 timer\_channel\_capture\_value\_register\_read

函数timer\_channel\_capture\_value\_register\_read描述见下表：

表 3-827. 函数 timer\_channel\_capture\_value\_register\_read

函数名称	timer_channel_capture_value_register_read
函数原型	uint32_t timer_channel_capture_value_register_read(uint32_t timer_periph, uint16_t channel);
功能描述	读取通道捕获值
先决条件	-
被调用函数	-
输入参数{in}	
timer_periph	TIMER外设
TIMERx(x=0,1,2,7)	参考具体参数
输入参数{in}	
channel	TIMER通道
TIMER_CH_0	通道0, TIMERx(x=0,1,2,7)
TIMER_CH_1	通道1, TIMERx(x=0,1,2,7)
TIMER_CH_2	通道2, TIMERx(x=0,1,2,7)
TIMER_CH_3	通道3, TIMERx(x=0,1,2,7)
TIMER_MCH_0	多模式通道0, TIMERx (x=0,7)
TIMER_MCH_1	多模式通道1, TIMERx (x=0,7)
TIMER_MCH_2	多模式通道2, TIMERx (x=0,7)
TIMER_MCH_3	多模式通道3, TIMERx (x=0,7)
输出参数{out}	
-	-
返回值	
uint32_t	通道输入捕获值 (0~0xFFFFFFFF)

例如：

```
/* read TIMER0 channel 0 capture compare register value */
```

```
uint32_t ch0_value = 0;
```

```
ch0_value = timer_channel_capture_value_register_read(TIMER0, TIMER_CH_0);
```

### 函数 timer\_input\_pwm\_capture\_config

函数timer\_input\_pwm\_capture\_config描述见下表：

表 3-828. 函数 timer\_input\_pwm\_capture\_config

函数名称	timer_input_pwm_capture_config
函数原型	void timer_input_pwm_capture_config(uint32_t timer_periph, uint16_t channel, timer_ic_parameter_struct* icpwm);
功能描述	配置TIMER捕获PWM输入参数
先决条件	-
被调用函数	timer_channel_input_capture_prescaler_config
输入参数{in}	
timer_periph	TIMER外设
TIMERx(x=0,1,2,7)	TIMER外设选择
输入参数{in}	
channel	TIMER通道
TIMER_CH_0	通道0
TIMER_CH_1	通道1
输入参数{in}	
icpwm	输入PWM捕获结构体，详见 <a href="#">结构体timer_ic_parameter_struct</a> 。
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* configure TIMER0 input pwm capture parameter */
timer_ic_parameter_struct timer_icinitpara;

timer_icinitpara.icpolarity = TIMER_IC_POLARITY_RISING;

timer_icinitpara.icselection = TIMER_IC_SELECTION_DIRECTTTI;

timer_icinitpara.icprescaler = TIMER_IC_PSC_DIV1;

timer_icinitpara.icfilter = 0x0;

timer_input_pwm_capture_config(TIMER0, TIMER_CH_0, &timer_icinitpara);
```

### 函数 timer\_hall\_mode\_config

函数timer\_hall\_mode\_config描述见下表：

表 3-829. 函数 timer\_hall\_mode\_config

函数名称	timer_hall_mode_config
函数原型	void timer_hall_mode_config(uint32_t timer_periph, uint32_t hallmode);
功能描述	配置TIMER的HALL接口功能
先决条件	-
被调用函数	-

输入参数{in}	
<b>timer_periph</b>	TIMER外设
<i>TIMERx</i> ( <i>x</i> =0, 1, 2, 7)	TIMER外设选择
输入参数{in}	
<b>hallmode</b>	HALL接口功能状态
<i>TIMER_HALLINTE</i> <i>RFACE_ENABLE</i>	HALL接口使能
<i>TIMER_HALLINTE</i> <i>RFACE_DISABLE</i>	HALL接口禁能
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* configure TIMER0 hall sensor mode */
```

```
timer_hall_mode_config(TIMER0, TIMER_HALLINTERFACE_ENABLE);
```

### 函数 timer\_multi\_mode\_channel\_output\_parameter\_struct\_init

函数timer\_multi\_mode\_channel\_output\_parameter\_struct\_init描述见下表：

表 3-830. 函数 timer\_multi\_mode\_channel\_output\_parameter\_struct\_init

函数名称	timer_multi_mode_channel_output_parameter_struct_init
函数原型	void timer_multi_mode_channel_output_parameter_struct_init(timer_omc_parameter_struct *omcpara);
功能描述	将TIMER多模式通道输出参数结构体中所有参数初始化为默认值
先决条件	-
被调用函数	-
输入参数{in}	
<b>omcpara</b>	多模式通道输出参数结构体，详见 <a href="#">结构体timer_omc_parameter_struct</a> 。
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* initialize TIMER multi mode channel output parameter struct with a default value */
```

```
timer_omc_parameter_struct timer_omcinitpara;
```

```
timer_multi_mode_channel_output_parameter_struct_init(&timer_omcinitpara);
```



**函数 timer\_multi\_mode\_channel\_output\_config**

函数timer\_multi\_mode\_channel\_output\_config描述见下表:

**表 3-831. 函数 timer\_multi\_mode\_channel\_output\_config**

函数名称	timer_multi_mode_channel_output_config
函数原型	void timer_multi_mode_channel_output_config(uint32_t timer_periph, uint16_t channel, timer_omc_parameter_struct *omcpara);
功能描述	外设TIMER的多模式通道输出配置
先决条件	-
被调用函数	-
输入参数{in}	
timer_periph	TIMER外设
TIMERx(x=0,7)	参考具体参数
输入参数{in}	
channel	TIMER通道
TIMER_MCH_0	多模式通道0
TIMER_MCH_1	多模式通道1
TIMER_MCH_2	多模式通道2
TIMER_MCH_3	多模式通道3
输入参数{in}	
omcpara	多模式通道输出参数结构体，详见 <a href="#">结构体timer_omc_parameter_struct</a> 。
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* configure TIMER0 multi mode channel 0 output function */
timer_omc_parameter_struct timer_omcinitpara;

omcpara->outputmode = TIMER_MCH_MODE_INDEPENDENTLY;

omcpara->outputstate = TIMER_MCCX_ENABLE;

omcpara->ocpolarity = TIMER_OMC_POLARITY_HIGH;

timer_multi_mode_channel_output_parameter_struct_init(TIMER0, TIMER_MCH_0, &timer_omcinitpara);
```

**函数 timer\_multi\_mode\_channel\_mode\_config**

函数timer\_multi\_mode\_channel\_mode\_config描述见下表:

表 3-832. 函数 timer\_multi\_mode\_channel\_mode\_config

函数名称	timer_multi_mode_channel_mode_config
函数原型	void timer_multi_mode_channel_mode_config(uint32_t timer_periph, uint32_t channel, uint32_t multi_mode_sel);
功能描述	外设TIMER多模式通道模式选择
先决条件	-
被调用函数	-
输入参数{in}	
timer_periph	TIMER外设
TIMERx(x=0,7)	参考具体参数
输入参数{in}	
channel	TIMER通道
TIMER_MCH_0	多模式通道0
TIMER_MCH_1	多模式通道1
TIMER_MCH_2	多模式通道2
TIMER_MCH_3	多模式通道3
输入参数{in}	
ocmode	通道输出比较模式
TIMER_MCH_MODE_INDEPENDENTLY	多模式通道为独立模式
TIMER_MCH_MODE_COMPLEMENTARY	多模式通道为互补模式
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* select TIMER0 multi mode channel 0 mode */
```

```
timer_multi_mode_channel_mode_config (TIMER0, TIMER_MCH_0, TIMER_MCH_MODE_INDEPENDENTLY);
```

### 函数 timer\_input\_trigger\_source\_select

函数timer\_input\_trigger\_source\_select描述见下表:

表 3-833. 函数 timer\_input\_trigger\_source\_select

函数名称	timer_input_trigger_source_select
函数原型	void timer_input_trigger_source_select(uint32_t timer_periph, uint32_t intrigger);

功能描述	TIMER的输入触发源选择
先决条件	SMC[2:0] = 000
被调用函数	-
输入参数{in}	
timer_periph	TIMER外设
TIMERx(x=0,1,2,7)	TIMER外设选择
输入参数{in}	
intrigger	输入触发源
TIMER_SMCFG_T RGSEL_NONE	禁能输入触发
TIMER_SMCFG_T RGSEL_ITI0	内部触发输入0(ITI0, TIMERx(x=0,1,2,7))
TIMER_SMCFG_T RGSEL_ITI1	内部触发输入1(ITI1, TIMERx(x=0,1,2,7))
TIMER_SMCFG_T RGSEL_ITI2	内部触发输入2(ITI2, TIMERx(x=0,1,2,7))
TIMER_SMCFG_T RGSEL_ITI3	内部触发输入3(ITI3, TIMERx(x=0,1,2,7))
TIMER_SMCFG_T RGSEL_CIOF_ED	TI0的边沿检测(CIOF_ED, TIMERx(x=0,1,2,7))
TIMER_SMCFG_T RGSEL_CIOFE0	滤波后的通道0输入(CIOFE0, TIMERx(x=0,1,2,7))
TIMER_SMCFG_T RGSEL_CI1FE1	滤波后的通道1输入(CI1FE1, TIMERx(x=0,1,2,7))
TIMER_SMCFG_T RGSEL_ETIFP	滤波后的外部触发输入(ETIFP, TIMERx(x=0,1,2,7))
TIMER_SMCFG_T RGSEL_CI2FE2	滤波后的通道2输入(TIMERx(x=0,7))
TIMER_SMCFG_T RGSEL_CI3FE3	滤波后的通道3输入(TIMERx(x=0,7))
TIMER_SMCFG_T RGSEL_MCI0FEM0	滤波后的多模式通道0输入(TIMERx(x=0,7))
TIMER_SMCFG_T RGSEL_MCI1FEM1	滤波后的多模式通道1输入(TIMERx(x=0,7))
TIMER_SMCFG_T RGSEL_MCI2FEM2	滤波后的多模式通道2输入(TIMERx(x=0,7))
TIMER_SMCFG_T RGSEL_MCI3FEM3	滤波后的多模式通道3输入(TIMERx(x=0,7))
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* select TIMER0 input trigger source */
```

```
timer_input_trigger_source_select (TIMER0, TIMER_SMCFG_TRGSEL_ITI0);
```

### 函数 timer\_master\_output\_trigger\_source\_select

函数timer\_master\_output\_trigger\_source\_select描述见下表：

表 3-834. 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 outrigger);
功能描述	选择TIMER主模式输出触发源
先决条件	-
被调用函数	-
输入参数{in}	
timer_periph	TIMER外设
TIMERx(x=0,1,2,7)	TIMER外设选择
输入参数{in}	
outrigger	输出触发源
TIMER_TRI_OUT_SRC_RESET	UPG位作为TRGO
TIMER_TRI_OUT_SRC_ENABLE	TIMER使能信号作为TRGO
TIMER_TRI_OUT_SRC_UPDATE	更新事件作为TRGO
TIMER_TRI_OUT_SRC_CH0	通道0捕获/比较事件作为TRGO
TIMER_TRI_OUT_SRC_O0CPRE	O0CPRE作为触发输出TRGO
TIMER_TRI_OUT_SRC_O1CPRE	O1CPRE作为触发输出TRGO
TIMER_TRI_OUT_SRC_O2CPRE	O2CPRE作为触发输出TRGO
TIMER_TRI_OUT_SRC_O3CPRE	O3CPRE作为触发输出TRGO
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* select TIMER0 master mode output trigger source */
```

```
timer_master_output_trigger_source_select(TIMER0, TIMER_TRI_OUT0_SRC_RESET);
```

### 函数 timer\_slave\_mode\_select

函数timer\_slave\_mode\_select描述见下表：

表 3-835. 函数 timer\_slave\_mode\_select

函数名称	timer_slave_mode_select
函数原型	void timer_slave_mode_select(uint32_t timer_periph, uint32_t slavemode);
功能描述	TIMER从模式配置
先决条件	-
被调用函数	-
输入参数{in}	
timer_periph	TIMER外设
TIMERx(x=0,1,2,7)	TIMER外设选择
输入参数{in}	
slavemode	从模式
TIMER_SLAVE_MODE_DISABLE	关闭从模式
TIMER_ENCODER_MODE0	编码器模式0
TIMER_ENCODER_MODE1	编码器模式1
TIMER_ENCODER_MODE2	编码器模式2
TIMER_SLAVE_MODE_RESTART	复位模式
TIMER_SLAVE_MODE_PAUSE	暂停模式
TIMER_SLAVE_MODE_EVENT	事件模式
TIMER_SLAVE_MODE_EXTERNAL0	外部时钟模式0
TIMER_SLAVE_MODE_RESTART_EVENT	复位+事件模式
TIMER_NONQUAD_MODE0	非正交编码器模式0
TIMER_NONQUAD_MODE1	非正交编码器模式1
TIMER_NONQUAD_MODE2	非正交编码器模式2

<code>_MODE2</code>	
<code>TIMER_NONQUAD</code> <code>_MODE3</code>	非正交编码器模式3
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* select TIMER0 slave mode */
```

```
timer_slave_mode_select(TIMER0, TIMER_ENCODER_MODE0);
```

### 函数 timer\_master\_slave\_mode\_config

函数timer\_master\_slave\_mode\_config描述见下表：

表 3-836. 函数 timer\_master\_slave\_mode\_config

函数名称	timer_master_slave_mode_config
函数原型	void timer_master_slave_mode_config(uint32_t timer_periph, uint32_t masterslave);
功能描述	TIMER主从模式配置
先决条件	-
被调用函数	-
输入参数{in}	
timer_periph	TIMER外设
<code>TIMERx(x=0,1,2,7)</code>	TIMER外设选择
输入参数{in}	
masterslave	主从模式使能状态
<code>TIMER_MASTER_SLAVE_MODE_ENABLE</code>	主从模式使能
<code>TIMER_MASTER_SLAVE_MODE_DISABLE</code>	主从模式禁能
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* configure TIMER0 master slave mode */
```

```
timer_master_slave_mode_config(TIMER0, TIMER_MASTER_SLAVE_MODE_ENABLE);
```

## 函数 timer\_external\_trigger\_config

函数timer\_external\_trigger\_config描述见下表：

表 3-837. 函数 timer\_external\_trigger\_config

函数名称	timer_external_trigger_config
函数原型	void timer_external_trigger_config(uint32_t timer_periph, uint32_t extprescaler, uint32_t expolarity, uint32_t extfilter);
功能描述	配置TIMER外部触发输入
先决条件	-
被调用函数	-
输入参数{in}	
timer_periph	TIMER外设
TIMERx(x=0,1,2,7)	TIMER外设选择
输入参数{in}	
extprescaler	外部触发预分频
TIMER_EXT_TRI_PSC_OFF	不分频
TIMER_EXT_TRI_PSC_DIV2	2分频
TIMER_EXT_TRI_PSC_DIV4	4分频
TIMER_EXT_TRI_PSC_DIV8	8分频
输入参数{in}	
expolarity	外部触发输入极性
TIMER_ETP_FALLING	低电平或者下降沿有效
TIMER_ETP_RISING	高电平或者上升沿有效
输入参数{in}	
extfilter	外部触发滤波控制（0~15）
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* configure TIMER0 external trigger input */
```

```
timer_external_trigger_config(TIMER0, TIMER_EXT_TRI_PSC_DIV2, TIMER_ETP_FALLING, 10);
```

## 函数 timer\_quadrature\_decoder\_mode\_config

函数timer\_quadrature\_decoder\_mode\_config描述见下表：

表 3-838. 函数 timer\_quadrature\_decoder\_mode\_config

函数名称	timer_quadrature_decoder_mode_config
函数原型	void timer_quadrature_decoder_mode_config (uint32_t timer_periph, uint32_t decomode, uint16_t ic0polarity, uint16_t ic1polarity);
功能描述	TIMER配置为编码器模式
先决条件	-
被调用函数	-
输入参数{in}	
timer_periph	TIMER外设
TIMERx(x=0,1,2,7)	TIMER外设选择
输入参数{in}	
decomode	正交编码器模式
TIMER_QUAD_DECODER_MODE0	根据CI1FE1的电平进行CI0FE0边沿计数
TIMER_QUAD_DECODER_MODE1	根据CI0FE0的电平进行CI1FE1边沿计数
TIMER_QUAD_DECODER_MODE2	根据CI1FE1/CI1FE0的电平进行CI0FE0/CI1FE1边沿计数
输入参数{in}	
ic0polarity	IC0输入极性
TIMER_IC_POLARITY_RISING	捕获上升边沿
TIMER_IC_POLARITY_FALLING	捕获下降边沿
TIMER_IC_POLARITY_BOTH_EDGE	捕获双边沿
输入参数{in}	
ic1polarity	IC1输入极性
TIMER_IC_POLARITY_RISING	捕获上升边沿
TIMER_IC_POLARITY_FALLING	捕获下降边沿
TIMER_IC_POLARITY_BOTH_EDGE	捕获双边沿
输出参数{out}	
-	-
返回值	
-	-



例如:

```
/* configure TIMER0 quadrature decoder mode */
```

```
timer_quadrature_decoder_mode_config(TIMER0, TIMER_ENCODER_MODE0, TIMER_IC_POLARITY_RISING, TIMER_IC_POLARITY_RISING);
```

### 函数 timer\_decoder\_mode\_config

函数timer\_decoder\_mode\_config描述见下表:

表 3-839. 函数 timer\_decoder\_mode\_config

函数名称	timer_decoder_mode_config
函数原型	void timer_decoder_mode_config (uint32_t timer_periph, uint32_t decompode, uint16_t ic0polarity, uint16_t ic1polarity);
功能描述	TIMER配置为非正交编码器模式
先决条件	-
被调用函数	-
输入参数{in}	
timer_periph	TIMER外设
TIMERx(x=0,1,2,7)	TIMER外设选择
输入参数{in}	
decompode	编码器模式
TIMER_DECODER_MODE0	非正交编码器模式0
TIMER_DECODER_MODE1	非正交编码器模式1
TIMER_DECODER_MODE2	非正交编码器模式2
TIMER_DECODER_MODE3	非正交编码器模式3
输入参数{in}	
ic0polarity	IC0输入极性
TIMER_IC_POLARITY_RISING	捕获上升边沿
TIMER_IC_POLARITY_FALLING	捕获下降边沿
TIMER_IC_POLARITY_BOTH_EDGE	捕获双边沿
输入参数{in}	
ic1polarity	IC1输入极性
TIMER_IC_POLARITY_RISING	捕获上升边沿
TIMER_IC_POLARITY_FALLING	捕获下降边沿

TY_FALLING	
TIMER_IC_POLARI TY_BOTH_EDGE	捕获双边沿
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* configure TIMER0 non-quadrature decoder mode */
```

```
timer_decoder_mode_config (TIMER0, TIMER_NONQUAD_MODE0, TIMER_IC_POLARI  
TY_RISING, TIMER_IC_POLARITY_RISING);
```

### 函数 timer\_internal\_clock\_config

函数timer\_internal\_clock\_config描述见下表：

表 3-840. 函数 timer\_internal\_clock\_config

函数名称	timer_internal_clock_config
函数原型	void timer_internal_clock_config(uint32_t timer_periph);
功能描述	TIMER配置为内部时钟模式
先决条件	-
被调用函数	-
输入参数{in}	
timer_periph	TIMER外设
TIMERx(x=0,1,2,7)	TIMER外设选择
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* configure TIMER0 internal clock mode */
```

```
timer_internal_clock_config (TIMER0);
```

### 函数 timer\_internal\_trigger\_as\_external\_clock\_config

函数timer\_internal\_trigger\_as\_external\_clock\_config描述见下表：

表 3-841. 函数 timer\_internal\_trigger\_as\_external\_clock\_config

函数名称	timer_internal_trigger_as_external_clock_config
函数原型	void timer_internal_trigger_as_external_clock_config(uint32_t timer_periph, uint32_t intrigger);

功能描述	配置TIMER的内部触发为时钟源
先决条件	-
被调用函数	timer_input_trigger_source_select
输入参数{in}	
timer_periph	TIMER外设
TIMERx(x=0,1,2,7)	TIMER外设选择
输入参数{in}	
intrigger	被选择的内部触发源
TIMER_SMCFG_T RGSEL_ITI0	内部触发输入0
TIMER_SMCFG_T RGSEL_ITI1	内部触发输入1
TIMER_SMCFG_T RGSEL_ITI2	内部触发输入2
TIMER_SMCFG_T RGSEL_ITI3	内部触发输入3
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* configure TIMER0 the internal trigger ITI0 as external clock input */
```

```
timer_internal_trigger_as_external_clock_config(TIMER0, TIMER_SMCFG_TRGSEL_ITI0);
```

### 函数 timer\_external\_trigger\_as\_external\_clock\_config

函数timer\_external\_trigger\_as\_external\_clock\_config描述见下表:

表 3-842. 函数 timer\_external\_trigger\_as\_external\_clock\_config

函数名称	timer_external_trigger_as_external_clock_config
函数原型	void timer_external_trigger_as_external_clock_config(uint32_t timer_periph, uint32_t extrigger, uint16_t expolarity, uint32_t extfilter);
功能描述	配置TIMER的外部触发作为时钟源
先决条件	-
被调用函数	timer_input_trigger_source_select
输入参数{in}	
timer_periph	TIMER外设
TIMERx(x=0,1,2,7)	TIMER外设选择
输入参数{in}	
extrigger	外部触发源
TIMER_SMCFG_T	TI0的边沿检测

<i>RGSEL_CIOF_ED</i>	
<i>TIMER_SMCFG_T RGSEL_CIOFE0</i>	滤波后的通道0输入
<i>TIMER_SMCFG_T RGSEL_CIOFE1</i>	滤波后的通道1输入
输入参数{in}	
<b>expolarity</b>	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>	外部触发双边沿有效
输入参数{in}	
<b>extfilter</b>	滤波参数（0~15）
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* configure TIMER0 the external trigger CIOFE0 as external clock input */
```

```
timer_external_trigger_as_external_clock_config(TIMER0, TIMER_SMCFG_TRGSEL_CIOFE0, TIMER_IC_POLARITY_RISING, 0);
```

### 函数 timer\_slave\_mode\_input\_config

函数timer\_slave\_mode\_input\_config描述见下表：

表 3-843. 函数 timer\_slave\_mode\_input\_config

函数名称	timer_slave_mode_input_config
函数原型	void timer_slave_mode_input_config(uint32_t timer_periph, uint32_t channel, uint16_t extpolarity, uint32_t extfilter)
功能描述	配置TIMER的从模式输入
先决条件	-
被调用函数	-
输入参数{in}	
<b>timer_periph</b>	TIMER外设
<i>TIMERx(x=0,1,2,7)</i>	TIMER外设选择
输入参数{in}	
<b>channel</b>	TIIMER通道
<i>TIMER_CH_0</i>	通道0, TIMERx(x=0,1,2,7)

<i>TIMER_CH_1</i>	通道1, TIMERx(x=0,1,2,7)
<i>TIMER_CH_2</i>	通道2, TIMERx(x=0,1,2,7)
<i>TIMER_CH_3</i>	通道3, TIMERx(x=0,1,2,7)
<i>TIMER_MCH_0</i>	多模式通道0, TIMERx (x=0,7)
<i>TIMER_MCH_1</i>	多模式通道1, TIMERx (x=0,7)
<i>TIMER_MCH_2</i>	多模式通道2, TIMERx (x=0,7)
<i>TIMER_MCH_3</i>	多模式通道3, TIMERx (x=0,7)
<b>输入参数{in}</b>	
<b>extpolarity</b>	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>	外部触发双边沿有效
<i>TIMER_IMC_POLARITY_RISING</i>	外部触发源高电平或者上升沿有效
<i>TIMER_IMC_POLARITY_FALLING</i>	外部触发源低电平或者下降沿有效
<i>TIMER_IMC_POLARITY_BOTH_EDGE</i>	外部触发双边沿有效
<b>输入参数{in}</b>	
<b>extfilter</b>	滤波参数 (0~15)
<b>输出参数{out}</b>	
-	-
<b>返回值</b>	
-	-

例如:

```
/* configure TIMER0 CH0 slave mode input polarity */
```

```
timer_slave_mode_input_config (TIMER0, TIMER_CH_0, 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 3-844. 函数 `timer_external_clock_mode0_config`**

函数名称	<code>timer_external_clock_mode0_config</code>
函数原型	<code>void timer_external_clock_mode0_config(uint32_t timer_periph, uint32_t extprescaler, uint32_t expolarity, uint32_t extfilter);</code>
功能描述	配置TIMER外部时钟模式0, ETI作为时钟源

先决条件	-
被调用函数	timer_external_trigger_config timer_slave_mode_select timer_input_trigger_source_select
输入参数{in}	
timer_periph	TIMER外设
TIMERx(x=0,1,2,7)	TIMER外设选择
输入参数{in}	
extprescaler	ETI触发源预分频值
TIMER_EXT_TRI_P SC_OFF	不分频
TIMER_EXT_TRI_P SC_DIV2	2分频
TIMER_EXT_TRI_P SC_DIV4	4分频
TIMER_EXT_TRI_P SC_DIV8	8分频
输入参数{in}	
expolarity	ETI触发源极性
TIMER_ETP_FALLI NG	下降沿或者低电平有效
TIMER_ETP_RISIN G	上升沿或者高电平有效
输入参数{in}	
extfilter	ETI触发源滤波参数（0~15）
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* configure TIMER0 the external clock mode0 */
```

```
timer_external_clock_mode0_config(TIMER0, TIMER_EXT_TRI_PSC_DIV2, TIMER_ETP_FALLING, 0);
```

### 函数 timer\_external\_clock\_mode1\_config

函数timer\_external\_clock\_mode1\_config描述见下表：

表 3-845. 函数 timer\_external\_clock\_mode1\_config

函数名称	timer_external_clock_mode1_config
函数原型	void timer_external_clock_mode1_config(uint32_t timer_periph, uint32_t

	extprescaler, uint32_t expolarity, uint32_t extfilter);
功能描述	配置TIMER外部时钟模式1
先决条件	-
被调用函数	timer_external_trigger_config
输入参数{in}	
timer_periph	TIMER外设
TIMERx(x=0,1,2,7)	TIMER外设选择
输入参数{in}	
extprescaler	ETI触发源预分频值
TIMER_EXT_TRI_P SC_OFF	不分频
TIMER_EXT_TRI_P SC_DIV2	2分频
TIMER_EXT_TRI_P SC_DIV4	4分频
TIMER_EXT_TRI_P SC_DIV8	8分频
输入参数{in}	
expolarity	ETI触发源极性
TIMER_ETP_FALLI NG	下降沿或者低电平有效
TIMER_ETP_RISIN G	上升沿或者高电平有效
输入参数{in}	
extfilter	ETI触发源滤波参数 (0~15)
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* configure TIMER0 the external clock mode1 */
```

```
timer_external_clock_mode1_config(TIMER0, TIMER_EXT_TRI_PSC_DIV2, TIMER_ETP_FALLING, 0);
```

### 函数 timer\_external\_clock\_mode1\_disable

函数timer\_external\_clock\_mode1\_disable描述见下表:

表 3-846. 函数 timer\_external\_clock\_mode1\_disable

函数名称	timer_external_clock_mode1_disable
函数原型	void timer_external_clock_mode1_disable(uint32_t timer_periph);

功能描述	TIMER外部时钟模式1禁能
先决条件	-
被调用函数	-
输入参数{in}	
timer_periph	TIMER外设
TIMERx(x=0,1,2,7)	TIMER外设选择
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* disable TIMER0 the external clock mode1 */
```

```
timer_external_clock_mode1_disable(TIMER0);
```

### 函数 timer\_write\_chxval\_register\_config

函数timer\_write\_chxval\_register\_config描述见下表：

表 3-847. 函数 timer\_write\_chxval\_register\_config

函数名称	timer_write_chxval_register_config
函数原型	void timer_write_chxval_register_config(uint32_t timer_periph, uint16_t ccsel);
功能描述	配置TIMER写CHxVAL选择位
先决条件	-
被调用函数	-
输入参数{in}	
timer_periph	TIMER外设
TIMERx(x=0,1,2,7)	TIMER外设选择
输入参数{in}	
ccsel	写CHxVAL寄存器选择位
TIMER_CHVSEL_DISABLE	无影响
TIMER_CHVSEL_ENABLE	当写入捕获比较寄存器的值与寄存器当前值相等时，写入操作无效。
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* configure TIMER0 write CHxVAL register selection */
```

```
timer_write_chxval_register_config(TIMER0, TIMER_CHVSEL_ENABLE);
```



**函数 timer\_output\_value\_selection\_config**

函数timer\_output\_value\_selection\_config描述见下表：

**表 3-848. 函数 timer\_output\_value\_selection\_config**

函数名称	timer_output_value_selection_config
函数原型	void timer_output_value_selection_config(uint32_t timer_periph, uint16_t outsel);
功能描述	配置TIMER输出值选择位
先决条件	-
被调用函数	-
输入参数{in}	
timer_periph	TIMER外设
TIMERx(x=0,7)	TIMER外设选择
输入参数{in}	
ccsel	输出值选择位
TIMER_OUTSEL_DISABLE	无影响
TIMER_OUTSEL_ENABLE	如果POEN位与IOS位均为0，则输出无效。
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* configure TIMER output value selection */
```

```
timer_output_value_selection_config(TIMER0, TIMER_OUTSEL_ENABLE);
```

**函数 timer\_output\_match\_pulse\_select**

函数timer\_output\_match\_pulse\_select描述见下表：

**表 3-849. 函数 timer\_output\_match\_pulse\_select**

函数名称	timer_output_match_pulse_select
函数原型	void timer_output_match_pulse_select(uint32_t timer_periph, uint32_t channel, uint16_t pulsesel);
功能描述	通道TIMER输出匹配脉冲选择
先决条件	-
被调用函数	-
输入参数{in}	
timer_periph	TIMER外设
TIMERx(x=0,7)	参考具体参数

输入参数{in}	
<b>channel</b>	TIMER通道
<i>TIMER_CH_0</i>	通道0
<i>TIMER_CH_1</i>	通道1
<i>TIMER_CH_2</i>	通道2
<i>TIMER_CH_3</i>	通道3
输入参数{in}	
<b>pulsesel</b>	输出匹配脉冲选择
<i>TIMER_PULSE_OUTPUT_NORMAL</i>	通道输出正常
<i>TIMER_PULSE_OUTPUT_CNT_UP</i>	仅在向上计数时，通道输出脉冲
<i>TIMER_PULSE_OUTPUT_CNT_DOWN</i>	仅在向下计数时，通道输出脉冲
<i>TIMER_PULSE_OUTPUT_CNT_BOTH</i>	向上/向下计数时，通道输出脉冲
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* configure TIMER0 channel 0 match pulse selection */
```

```
timer_output_match_pulse_select (TIMER0, TIMER_CH_0, TIMER_PULSE_OUTPUT_CNT_UP;
```

### 函数 timer\_channel\_composite\_pwm\_pulse\_value\_config

函数timer\_channel\_composite\_pwm\_pulse\_config描述见下表：

**表 3-850. 函数 timer\_channel\_composite\_pwm\_pulse\_config**

<b>函数名称</b>	timer_channel_composite_pwm_pulse_config
<b>函数原型</b>	void timer_channel_composite_pwm_pulse_config (uint32_t timer_periph, uint32_t channel, uint32_t pulse, uint32_t add_pulse);
<b>功能描述</b>	配置TIMER的复合PWM模式输出脉冲值
<b>先决条件</b>	-
<b>被调用函数</b>	-
输入参数{in}	
<b>timer_periph</b>	TIMER外设
<i>TIMERx(x=0,7)</i>	参考具体参数
输入参数{in}	
<b>channel</b>	TIMER通道

<i>TIMER_CH_0</i>	通道0
<i>TIMER_CH_1</i>	通道1
<i>TIMER_CH_2</i>	通道2
<i>TIMER_CH_3</i>	通道3
输入参数{in}	
<b>pulse</b>	通道比较值（0~0xFFFF）
输入参数{in}	
<b>add_pulse</b>	通道附加比较值（0~0xFFFF）
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* configure TIMER0 channel 0 output pulse value */
```

```
timer_channel_composite_pwm_pulse_config (TIMER0, TIMER_CH_0, 399, 3999);
```

### 函数 timer\_channel\_asymmetric\_pwm\_pulse\_config

函数timer\_channel\_asymmetric\_pwm\_pulse\_config描述见下表：

表 3-851. 函数 timer\_channel\_asymmetric\_pwm\_pulse\_config

函数名称	timer_channel_asymmetric_pwm_pulse_config
函数原型	void timer_channel_asymmetric_pwm_pulse_config (uint32_t timer_periph, uint32_t channel, uint32_t pulse, uint32_t add_pulse);
功能描述	配置TIMER的非对称PWM模式输出脉冲值
先决条件	-
被调用函数	-
输入参数{in}	
<b>timer_periph</b>	TIMER外设
<i>TIMERx(x=0,7)</i>	参考具体参数
输入参数{in}	
<b>channel</b>	TIMER通道
<i>TIMER_CH_0</i>	通道0
<i>TIMER_CH_1</i>	通道1
<i>TIMER_CH_2</i>	通道2
<i>TIMER_CH_3</i>	通道3
输入参数{in}	
<b>pulse</b>	通道比较值（0~0xFFFF）
输入参数{in}	
<b>add_pulse</b>	通道附加比较值（0~0xFFFF）
输出参数{out}	

-	-
返回值	
-	-

例如：

- /\* configure TIMER0 channel 0 output pulse value \*/
- timer\_channel\_asymmetric\_pwm\_pulse\_config (TIMER0, TIMER\_CH\_0, 399, 3999);

### 函数 timer\_channel\_additional\_compare\_value\_config

函数timer\_channel\_additional\_compare\_value\_config描述见下表：

表 3-852. 函数 timer\_channel\_additional\_compare\_value\_config

函数名称	timer_channel_additional_compare_value_config
函数原型	void timer_channel_additional_compare_value_config(uint32_t timer_periph, uint16_t channel, uint32_t value)
功能描述	配置TIMER通道附加比较寄存器
先决条件	-
被调用函数	-
输入参数{in}	
timer_periph	TIMER外设
TIMERx(x=0,7)	参考具体参数
输入参数{in}	
channel	TIMER通道
TIMER_CH_0	通道0
TIMER_CH_1	通道1
TIMER_CH_2	通道2
TIMER_CH_3	通道3
输入参数{in}	
value	通道附加比较值（0~0xFFFF）
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* configure TIMER0 channel 0 additional compare value */
timer_channel_additional_compare_value_config (TIMER0, TIMER_CH_0, 399);
```

### 函数 timer\_channel\_additional\_compare\_value\_read

函数timer\_channel\_additional\_compare\_value\_read描述见下表：

表 3-853. 函数 timer\_channel\_additional\_compare\_value\_read

函数名称	timer_channel_additional_compare_value_read
函数原型	uint16_t timer_channel_additional_compare_value_read(uint32_t timer_periph, uint16_t channel);
功能描述	读取TIMER通道附加输出比较寄存器值
先决条件	-
被调用函数	-
输入参数{in}	
timer_periph	TIMER外设
TIMERx(x=0,7)	TIMER外设选择
输入参数{in}	
channel	TIMER通道
TIMER_CH_0	通道0
TIMER_CH_1	通道1
TIMER_CH_2	通道2
TIMER_CH_3	通道3
输出参数{out}	
-	-
返回值	
Uint16_t	附加输出比较寄存器值，0~0xFFFF

例如：

```
/* get TIMER autoreload register value */
```

```
uint16_t i = 0;
```

```
i = timer_channel_additional_compare_value_read(TIMER0, TIMER_CH_0);
```

### 函数 timer\_channel\_additional\_output\_shadow\_config

函数timer\_channel\_additional\_output\_shadow\_config描述见下表：

表 3-854. 函数 timer\_channel\_additional\_output\_shadow\_config

函数名称	timer_channel_additional_output_shadow_config
函数原型	void timer_channel_additional_output_shadow_config(uint32_t timer_periph, uint16_t channel, uint16_t aocshadow);
功能描述	配置TIMER通道附加输出比较影子寄存器功能
先决条件	-
被调用函数	-
输入参数{in}	
timer_periph	TIMER外设
TIMERx(x=0,7)	参考具体参数
输入参数{in}	
channel	TIMER通道

<i>TIMER_CH_0</i>	通道0
<i>TIMER_CH_1</i>	通道1
<i>TIMER_CH_2</i>	通道2
<i>TIMER_CH_3</i>	通道3
输入参数{in}	
<b>aocshadow</b>	通道附加输出比较影子寄存器状态
<i>TIMER_ADD_SHADOW_ENABLE</i>	通道附加输出比较影子寄存器使能
<i>TIMER_ADD_SHADOW_DISABLE</i>	通道附加输出比较影子寄存器禁能
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/*configure TIMER0 channel 0 additional output shadow function */
```

```
timer_channel_additional_output_shadow_config (TIMER0, TIMER_CH_0, TIMER_OC_SHADOW_ENABLE);
```

### 函数 timer\_break\_external\_input\_enable

函数timer\_break\_external\_input\_enable描述见下表：

表 3-855. 函数 timer\_break\_external\_input\_enable

函数名称	timer_break_external_input_enable
函数原型	void timer_break_external_input_enable(uint32_t timer_periph);
功能描述	使能TIMER中止功能外部输入
先决条件	-
被调用函数	-
输入参数{in}	
<b>timer_periph</b>	TIMER外设
<i>TIMERx(x=0,7)</i>	参考具体参数
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* enable the TIMER break external input */
```

```
timer_break_external_input_enable (TIMER0);
```

**函数 timer\_break\_cmp\_enable**

函数timer\_break\_cmp\_enable描述见下表:

**表 3-856. 函数 timer\_break\_cmp\_enable**

函数名称	timer_break_cmp_enable
函数原型	void timer_break_cmp_enable(uint32_t timer_periph, uint32_t break_cmp);
功能描述	使能TIMER中止功能CMP输入
先决条件	-
被调用函数	-
输入参数{in}	
timer_periph	TIMER外设
TIMERx(x=0,7)	参考具体参数
输入参数{in}	
break_cmp	中止功能CMP选择
TIMER_BREAK_C MPX (X=0..3)	CMP0/1/2/3
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* enable the TIMER break cmp input */
```

```
timer_break_cmp_enable (TIMER0, TIMER_BERAK_CMP0);
```

**函数 timer\_break\_external\_input\_disable**

函数timer\_break\_external\_input\_disable描述见下表:

**表 3-857. 函数 timer\_break\_external\_input\_disable**

函数名称	timer_break_external_input_disable
函数原型	void timer_break_external_input_disable(uint32_t timer_periph);
功能描述	禁能TIMER中止功能外部输入
先决条件	-
被调用函数	-
输入参数{in}	
timer_periph	TIMER外设
TIMERx(x=0,7)	参考具体参数
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* disable the TIMER break enternal input */
```

```
timer_break_external_input_disable (TIMER0);
```

### 函数 timer\_break\_cmp\_disable

函数timer\_break\_cmp\_disable描述见下表:

表 3-858. 函数 timer\_break\_cmp\_disable

函数名称	timer_break_cmp_disable
函数原型	void timer_break_cmp_disable(uint32_t timer_periph, uint32_t break_cmp);
功能描述	禁能TIMER中止功能CMP输入
先决条件	-
被调用函数	-
输入参数{in}	
timer_periph	TIMER外设
TIMERx(x=0,7)	参考具体参数
输入参数{in}	
break_cmp	中止功能CMP选择
TIMER_BREAK_C MPX (X=0..3)	CMP0/1/2/3
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* disable the TIMER break cmp input */
```

```
timer_break_cmp_disable (TIMER0, TIMER_BREAK_CMP0);
```

### 函数 timer\_break\_external\_input\_polarity\_config

函数timer\_break\_external\_input\_polarity\_config描述见下表:

表 3-859. 函数 timer\_break\_external\_input\_polarity\_config

函数名称	timer_break_external_input_polarity_config
函数原型	void timer_break_external_input_polarity_config(uint32_t timer_periph, uint32_t polarity);
功能描述	配置TIMER中止功能输入极性
先决条件	-
被调用函数	-
输入参数{in}	



<b>timer_periph</b>	TIMER外设
<i>TIMERx(x=0,7)</i>	参考具体参数
<b>输入参数{in}</b>	
<b>polarity</b>	中止极性
<i>TIMER_BRKIN_POLARITY_NONINVERTED</i>	输入信号正相
<i>TIMER_BRKIN_POLARITY_INVERTED</i>	输入信号反相
<b>输出参数{out}</b>	
-	-
<b>返回值</b>	
-	-

例如：

```
/* configure the TIMER break external input polarity */
```

```
timer_break_external_input_polarity_config (TIMER0, TIMER_BRKIN_POLARITY_NONINVERTED);
```

### 函数 timer\_break\_cmp\_polarity\_config

函数timer\_break\_cmp\_polarity\_config描述见下表：

表 3-860. 函数 timer\_break\_cmp\_polarity\_config

<b>函数名称</b>	timer_break_cmp_polarity_config
<b>函数原型</b>	void timer_break_cmp_polarity_config(uint32_t timer_periph, uint32_t break_cmp, uint32_t polarity);
<b>功能描述</b>	配置TIMER中止功能CMP输入极性
<b>先决条件</b>	-
<b>被调用函数</b>	-
<b>输入参数{in}</b>	
<b>timer_periph</b>	TIMER外设
<i>TIMERx(x=0,7)</i>	参考具体参数
<b>输入参数{in}</b>	
<i>break_cmp</i>	中止功能CMP选择
<i>TIMER_BREAK_CMPX (X=0..3)</i>	CMP0/1/2/3
<b>输入参数{in}</b>	
<b>polarity</b>	中止极性
<i>TIMER_BRKIN_CMP_P_INVERTED</i>	CMP输入信号反相
<i>TIMER_BRKIN_CMP</i>	CMP输入信号正相

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

例如：

```
/* configure the TIMER break cmp polarity */
```

```
timer_break_cmp_polarity_config (TIMER0, TIMER_BRKIN_CMP0,TIMER_BRKIN_CMP_
NONINVERTED);
```

### 函数 timer\_break\_auto\_recover\_event\_select

函数timer\_break\_auto\_recover\_event\_select描述见下表：

表 3-861. 函数 timer\_break\_auto\_recover\_event\_select

函数名称	timer_break_auto_recover_event_select
函数原型	void timer_break_auto_recover_event_select(uint32_t timer_periph, uint32_t event);
功能描述	选择TIMER中止恢复事件
先决条件	-
被调用函数	-
输入参数{in}	
timer_periph	TIMER外设
TIMERx(x=0,7)	参考具体参数
输入参数{in}	
event	中止恢复事件
TIMER_BRKAU_M ODE_FLOW	在下一次溢出后恢复输出
TIMER_BRKAU_M ODE_OVERFLOW	在下一次上溢后恢复输出
TIMER_BRKAU_M ODE_UNDERFLOW	在下一次下溢后恢复输出
TIMER_BRKAU_M ODE_UPDATE_EVENT	在下一次更新后恢复输出
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* configure TIMER0 break auto recover event selection */
```

```
timer_break_auto_recover_event_select (TIMER0, TIMER_BRKAU_MODE_FLOW);
```

### 函数 timer\_trigger\_adc\_compare\_enable

函数timer\_trigger\_adc\_compare\_enable描述见下表:

表 3-862. 函数 timer\_trigger\_adc\_compare\_enable

函数名称	timer_trigger_adc_compare_enable
函数原型	void timer_trigger_adc_compare_enable(uint32_t timer_periph, uint32_t compare_time);
功能描述	使能TIMER比较事件产生触发ADC转换信号
先决条件	-
被调用函数	-
输入参数{in}	
timer_periph	TIMER外设
TIMERx(x=0,7)	参考具体参数
输入参数{in}	
compare_time	比较时刻
TIMER_ADCRCTL_ADT2DWGTEN	TIMER_TRGB在向下计数比较触发
TIMER_ADCRCTL_ADT2UPGTEN	TIMER_TRGB在向上计数比较触发
TIMER_ADCRCTL_ADT1DWGTEN	TIMER_TRGA在向下计数比较触发
TIMER_ADCRCTL_ADT1UPGTEN	TIMER_TRGA在向上计数比较触发
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* enable TIMER0 compare event produce a ADC converter trigger signal */
```

```
timer_trigger_adc_compare_enable (TIMER0, TIMER_ADCRCTL_ADT2DWGTEN);
```

### 函数 timer\_trigger\_adc\_compare\_disable

函数timer\_trigger\_adc\_compare\_disable描述见下表:

表 3-863. 函数 timer\_trigger\_adc\_compare\_disable

函数名称	timer_trigger_adc_compare_disable
函数原型	void timer_trigger_adc_compare_disable(uint32_t timer_periph, uint32_t

	compare_time);
功能描述	禁能TIMER比较事件产生触发ADC转换信号
先决条件	-
被调用函数	-
输入参数{in}	
timer_periph	TIMER外设
TIMERx(x=0,7)	参考具体参数
输入参数{in}	
compare_time	比较时刻
TIMER_ADCRCTL_ ADT2DWGTEN	TIMER_TRGB在向下计数比较触发
TIMER_ADCRCTL_ ADT2UPGTEN	TIMER_TRGB在向上计数比较触发
TIMER_ADCRCTL_ ADT1DWGTEN	TIMER_TRGA在向下计数比较触发
TIMER_ADCRCTL_ ADT1UPGTEN	TIMER_TRGA在向上计数比较触发
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* disable TIMER0 compare event produce a ADC converter trigger signal */
```

```
timer_trigger_adc_compare_disable (TIMER0, TIMER_ADCRCTL_ADT2DWGTEN);
```

### 函数 timer\_trigger\_adc\_flow\_enable

函数timer\_trigger\_adc\_flow\_enable描述见下表:

表 3-864. 函数 timer\_trigger\_adc\_flow\_enable

函数名称	timer_trigger_adc_flow_enable
函数原型	void timer_trigger_adc_flow_enable(uint32_t timer_periph, uint32_t flow_timer);
功能描述	使能TIMER溢出事件产生ADC转换触发信号
先决条件	-
被调用函数	-
输入参数{in}	
timer_periph	TIMER外设
TIMERx(x=0,7)	参考具体参数
输入参数{in}	
flow_timer	溢出时刻
TIMER_ADCTL_AD	选择下溢信号

<i>TSUF</i>	
<i>TIMER_ADCTL_AD</i> <i>TSOF</i>	选择上溢信号
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* enable TIMER0 flow event produce a ADC converter trigger signal */
```

```
timer_trigger_adc_flow_enable (TIMER0, TIMER_ADCTL_ADTSUF);
```

### 函数 timer\_trigger\_adc\_flow\_disable

函数timer\_trigger\_adc\_flow\_disable描述见下表：

表 3-865. 函数 timer\_trigger\_adc\_flow\_disable

函数名称	timer_trigger_adc_flow_disable
函数原型	void timer_trigger_adc_flow_disable(uint32_t timer_periph, uint32_t flow_timer);
功能描述	禁能TIMER溢出事件产生ADC转换触发信号
先决条件	-
被调用函数	-
输入参数{in}	
<b>timer_periph</b>	TIMER外设
<i>TIMERx(x=0,7)</i>	参考具体参数
输入参数{in}	
<b>flow_timer</b>	溢出时刻
<i>TIMER_ADCTL_AD</i> <i>TSUF</i>	选择下溢信号
<i>TIMER_ADCTL_AD</i> <i>TSOF</i>	选择上溢信号
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* disable TIMER0 flow event produce a ADC converter trigger signal */
```

```
timer_trigger_adc_flow_disable (TIMER0, TIMER_ADCTL_ADTSUF);
```

**函数 timer\_trigger\_adc\_compare\_value\_config**

函数timer\_trigger\_adc\_compare\_value\_config描述见下表:

**表 3-866. 函数 timer\_trigger\_adc\_compare\_value\_config**

函数名称	timer_trigger_adc_compare_value_config
函数原型	void timer_trigger_adc_compare_value_config(uint32_t timer_periph, uint32_t compare, uint16_t value);
功能描述	配置TIMER触发ADC的相应寄存器的比较值
先决条件	-
被调用函数	-
输入参数{in}	
timer_periph	TIMER外设
TIMERx(x=0,7)	参考具体参数
输入参数{in}	
compare	比较寄存器选择
TIMER_ADC_COMPARE1	TIMER_ADC_CCR1
TIMER_ADC_COMPARE2	TIMER_ADC_CCR2
输入参数{in}	
value	触发ADC比较值 (0~0xFFFF)
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* configure TIMER0 trigger ADC compare register value */
timer_trigger_adc_compare_value_config (TIMER0, TIMER_ADC_COMPARE1, 399);
```

**函数 timer\_trigger\_adc\_repetition\_value\_config**

函数timer\_trigger\_adc\_repetition\_value\_config描述见下表:

**表 3-867. 函数 timer\_trigger\_adc\_repetition\_value\_config**

函数名称	timer_trigger_adc_repetition_value_config
函数原型	void timer_trigger_adc_repetition_value_config(uint32_t timer_periph, uint32_t timer_adc_crep, uint16_t value);
功能描述	配置TIMER的ADC触发重复值
先决条件	-
被调用函数	-
输入参数{in}	

<b>timer_periph</b>	TIMER外设
<i>TIMERx(x=0,7)</i>	参考具体参数
输入参数{in}	
<b>timer_adc_crep</b>	选择TIMER触发ADC重复信号
<i>TIMER_ADC_REPA</i>	ADC_REP_A
<i>TIMER_ADC_REPB</i>	ADC_REP_B
<i>TIMER_ADC_REPA</i> <i>B</i>	ADC_REP_AB
输入参数{in}	
<b>value</b>	触发ADC重复值（0~0xF）
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* configure TIMER0 trigger ADC repetition value */
```

```
timer_trigger_adc_repetition_value_config (TIMER0, TIMER_ADC_REPA, 1);
```

### 函数 timer\_trigger\_adc\_repetition\_decrement\_select

函数timer\_trigger\_adc\_repetition\_decrement\_select描述见下表：

**表 3-868. 函数 timer\_trigger\_adc\_repetition\_decrement\_select**

<b>函数名称</b>	timer_trigger_adc_repetition_decrement_select
<b>函数原型</b>	void timer_trigger_adc_repetition_decrement_select(uint32_t timer_periph, uint32_t timer_adc_crep, uint16_t source);
<b>功能描述</b>	选择TIMER触发ADC重复计数值自减信号源
<b>先决条件</b>	-
<b>被调用函数</b>	-
输入参数{in}	
<b>timer_periph</b>	TIMER外设
<i>TIMERx(x=0,7)</i>	参考具体参数
输入参数{in}	
<b>timer_adc_crep</b>	选择TIMER触发ADC重复信号
<i>TIMER_ADC_REPA</i>	ADC_REP_A
<i>TIMER_ADC_REPB</i>	ADC_REP_B
输入参数{in}	
<b>source</b>	自减源。
<i>TIMER_ADC_REP_DEC_OVERFLOW</i>	上溢时自减
<i>TIMER_ADC_REP_</i>	下溢时自减

DEC_UNDERFLOW	
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* select TIMER0 trigger ADC repetition_decrement source */
```

```
timer_trigger_adc_repetition_decrement_select (TIMER0, TIMER_ADC_REPA, TIMER_A  
DC_REP_DEC_OVERFLOW);
```

### 函数 timer\_trigger\_adc\_repetition\_value\_reload

函数timer\_trigger\_adc\_repetition\_value\_reload描述见下表：

表 3-869. 函数 timer\_trigger\_adc\_repetition\_value\_reload

函数名称	timer_trigger_adc_repetition_value_reload
函数原型	void timer_trigger_adc_repetition_value_reload(uint32_t timer_periph);
功能描述	重载ADC触发重复计数值
先决条件	-
被调用函数	-
输入参数{in}	
timer_periph	TIMER外设
TIMER <sub>x</sub> (x=0,7)	TIMER外设选择
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* reload TIMER0 trigger ADC repetition value */
```

```
timer_trigger_adc_repetition_value_reload (TIMER0);
```

### 函数 timer\_trigger\_adc\_compare\_value\_shadow\_enable

函数timer\_trigger\_adc\_compare\_value\_shadow\_enable描述见下表：

表 3-870. 函数 timer\_trigger\_adc\_compare\_value\_shadow\_enable

函数名称	timer_trigger_adc_compare_value_shadow_enable
函数原型	void timer_trigger_adc_compare_value_shadow_enable(uint32_t timer_periph, uint32_t timer_adc_cmpval);
功能描述	使能ADC触发比较影子寄存器
先决条件	-



被调用函数	-
输入参数{in}	
timer_periph	TIMER外设
TIMERx(x=0,7)	TIMER外设选择
输入参数{in}	
timer_adc_cmpval	ADC触发影子寄存器
TIMER_ADCTL_ADTPREEN1	TIMER_ADCCR1
TIMER_ADCTL_ADTPREEN1	TIMER_ADCCR2
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* enable TIMER0 trigger ADC compare regiseter shadow function */
```

```
timer_trigger_adc_compare_value_shadow_enable (TIMER0, TIMER_ADCTL_ADTPREEN1);
```

### 函数 timer\_trigger\_adc\_compare\_value\_shadow\_disable

函数timer\_trigger\_adc\_compare\_value\_shadow\_disable描述见下表:

表 3-871. 函数 timer\_trigger\_adc\_compare\_value\_shadow\_disable

函数名称	timer_trigger_adc_compare_value_shadow_disable
函数原型	void timer_trigger_adc_compare_value_shadow_disable(uint32_t timer_periph, uint32_t timer_adc_cmpval);
功能描述	禁能ADC触发比较影子寄存器
先决条件	-
被调用函数	-
输入参数{in}	
timer_periph	TIMER外设
TIMERx(x=0,7)	TIMER外设选择
输入参数{in}	
timer_adc_cmpval	ADC触发影子寄存器
TIMER_ADCTL_ADTPREEN1	TIMER_ADCCR1
TIMER_ADCTL_ADTPREEN1	TIMER_ADCCR2
输出参数{out}	
-	-

返回值	
-	-

例如：

```
/* disable TIMER0 trigger ADC compare regiseter shadow function */
```

```
timer_trigger_adc_compare_value_shadow_disable (TIMER0, TIMER_ADCTL_ADTPREEN1);
```

### 函数 timer\_trigger\_adc\_monitor\_config

函数timer\_trigger\_adc\_monitor\_config描述见下表：

表 3-872. 函数 timer\_trigger\_adc\_monitor\_config

函数名称	timer_trigger_adc_monitor_config
函数原型	void timer_trigger_adc_monitor_config(uint32_t timer_periph, uint32_t adcsel, uint32_t adcsel_sel, ControlStatus newvalue);
功能描述	配置ADC触发信号监控
先决条件	-
被调用函数	-
输入参数{in}	
<b>timer_periph</b>	TIMER外设
TIMERx(x=0,7)	参考具体参数
输入参数{in}	
<b>adcsel</b>	ADC信号监控引脚
SYSCFG_TADSRCFG_ADCSM1	ADC SM1引脚输出
SYSCFG_TADSRCFG_ADCSM2	ADC SM2引脚输出
输入参数{in}	
<b>adcsel_sel</b>	选择ADC触发信号监控
SYSCFG_ADCSM1_TIMER0_TRGOF	TIMER0_TRGOF
SYSCFG_ADCSM1_TIMER0_TRGUF	TIMER0_TRGUF
SYSCFG_ADCSM1_TIMER0_TRGA	TIMER0_TRGA
SYSCFG_ADCSM1_TIMER0_TRGB	TIMER0_TRGB
SYSCFG_ADCSM1_TIMER0_TRGAB	TIMER0_TRGAB
SYSCFG_ADCSM1_TIMER7_TRGOF	TIMER7_TRGOF

SYSCFG_ADCSM1 _TIMER7_TRGUF	TIMER7_TRGUF
SYSCFG_ADCSM1 _TIMER7_TRGA	TIMER7_TRGA
SYSCFG_ADCSM1 _TIMER7_TRGB	TIMER7_TRGB
SYSCFG_ADCSM1 _TIMER7_TRGAB	TIMER7_TRGAB
SYSCFG_ADCSM2 _TIMER0_TRGOF	TIMER0_TRGOF
SYSCFG_ADCSM2 _TIMER0_TRGUF	TIMER0_TRGUF
SYSCFG_ADCSM2 _TIMER0_TRGA	TIMER0_TRGA
SYSCFG_ADCSM2 _TIMER0_TRGB	TIMER0_TRGB
SYSCFG_ADCSM2 _TIMER0_TRGAB	TIMER0_TRGAB
SYSCFG_ADCSM2 _TIMER7_TRGOF	TIMER7_TRGOF
SYSCFG_ADCSM2 _TIMER7_TRGUF	TIMER7_TRGUF
SYSCFG_ADCSM2 _TIMER7_TRGA	TIMER7_TRGA
SYSCFG_ADCSM2 _TIMER7_TRGB	TIMER7_TRGB
SYSCFG_ADCSM2 _TIMER7_TRGAB	TIMER7_TRGAB
输入参数{in}	
newvalue	控制状态
ENABLE	使能
DISABLE	禁能
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* configure ADC trigger signal monitoring function */
```

```
timer_trigger_adc_monitor_config (TIMER0, SYSCFG_ADCSM1_TIMER0_TRGOF, ENABLE);
```

## 函数 timer\_register\_update\_event\_select

函数timer\_register\_update\_event\_select描述见下表:

表 3-873. 函数 timer\_register\_update\_event\_select

函数名称	timer_register_update_event_select
函数原型	void timer_register_update_event_select(uint32_t timer_periph, uint32_t reg, uint32_t event);
功能描述	选择影子寄存器更新事件
先决条件	-
被调用函数	-
输入参数{in}	
timer_periph	TIMER外设
TIMERx(x=0,7)	参考具体参数
输入参数{in}	
reg	选择影子寄存器
TIMER_CH0CV_UPSEL	选择TIMER_CH0CV的影子寄存器
TIMER_CH1CV_UPSEL	选择TIMER_CH1CV的影子寄存器
TIMER_CH2CV_UPSEL	选择TIMER_CH2CV的影子寄存器
TIMER_CH3CV_UPSEL	选择TIMER_CH3CV的影子寄存器
TIMER_CH0CV_AD D_UPSEL	选择TIMER_CH0COMVADD的影子寄存器
TIMER_CH1CV_AD D_UPSEL	选择TIMER_CH1COMVADD的影子寄存器
TIMER_CH2CV_AD D_UPSEL	选择TIMER_CH2COMVADD的影子寄存器
TIMER_CH3CV_AD D_UPSEL	选择TIMER_CH3COMVADD的影子寄存器
TIMER_ADC_CR1_ UPSEL	选择TIMER_ADCCR1的影子寄存器
TIMER_ADC_CR2_ UPSEL	选择TIMER_ADCCR2的影子寄存器
TIMER_CAR_UPSEL	选择TIMER_CAR的影子寄存器
TIMER_MCH0CV_ UPSEL	选择TIMER_MCH0CV的影子寄存器
TIMER_MCH1CV_ UPSEL	选择TIMER_MCH1CV的影子寄存器
TIMER_MCH2CV_ UPSEL	选择TIMER_MCH2CV的影子寄存器

UPSEL	
TIMER_MCH3CV_UPSEL	选择TIMER_MCH3CV的影子寄存器
输入参数{in}	
event	选择更新事件
TIMER_UPDATE_GLOBAL	计数更新事件
TIMER_UPDATE_NEXT_OVERFLOW	下次上溢事件
TIMER_UPDATE_NEXT_UNDERFLOW	下次下溢事件
TIMER_UPDATE_NEXT_FLOW	下次溢出事件
TIMER_UPDATE_NEXT_ADDCOMPAR_E_EVENT	下次比较事件
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* select TIMER0 register reload event */
```

```
timer_register_update_event_select (TIMER0, TIMER_CH0CV_UPSEL, TIMER_UPDATE_GLOBAL);
```

### 函数 timer\_flag\_get

函数timer\_flag\_get描述见下表:

表 3-874. 函数 timer\_flag\_get

函数名称	timer_flag_get
函数原型	FlagStatus timer_flag_get(uint32_t timer_periph, uint32_t flag);
功能描述	获取外设TIMERx的状态标志
先决条件	-
被调用函数	-
输入参数{in}	
timer_periph	TIMER外设
TIMERx(x=0,1,2,7)	参考具体参数
输入参数{in}	
flag	状态标志
TIMER_FLAG_UP	更新标志, TIMERx(x=0,1,2,7)

<i>TIMER_FLAG_CH0</i>	通道0比较/捕获标志, <i>TIMERx</i> ( <i>x</i> =0,1,2,7)
<i>TIMER_FLAG_CH1</i>	通道1比较/捕获标志, <i>TIMERx</i> ( <i>x</i> =0,1,2,7)
<i>TIMER_FLAG_CH2</i>	通道2比较/捕获标志, <i>TIMERx</i> ( <i>x</i> =0,1,2,7)
<i>TIMER_FLAG_CH3</i>	通道3比较/捕获标志, <i>TIMERx</i> ( <i>x</i> =0,1,2,7)
<i>TIMER_FLAG_CMT</i>	通道换相更新标志, <i>TIMERx</i> ( <i>x</i> =0,7)
<i>TIMER_FLAG_TRG</i>	触发标志, <i>TIMERx</i> ( <i>x</i> =0,1,2,7)
<i>TIMER_FLAG_BRK</i>	BREAK标志, <i>TIMERx</i> ( <i>x</i> =0,7)
<i>TIMER_FLAG_CH0</i> 0	通道0捕获溢出标志, <i>TIMERx</i> ( <i>x</i> =0,1,2,7)
<i>TIMER_FLAG_CH1</i> 0	通道1捕获溢出标志, <i>TIMERx</i> ( <i>x</i> =0,1,2,7)
<i>TIMER_FLAG_CH2</i> 0	通道2捕获溢出标志, <i>TIMERx</i> ( <i>x</i> =0,1,2,7)
<i>TIMER_FLAG_CH3</i> 0	通道3捕获溢出标志, <i>TIMERx</i> ( <i>x</i> =0,1,2,7)
<i>TIMER_FLAG_MCH</i> 00	多模式通道0捕获溢出标志, <i>TIMERx</i> ( <i>x</i> =0,7)
<i>TIMER_FLAG_MCH</i> 10	多模式通道1捕获溢出标志, <i>TIMERx</i> ( <i>x</i> =0,7)
<i>TIMER_FLAG_MCH</i> 20	多模式通道2捕获溢出标志, <i>TIMERx</i> ( <i>x</i> =0,7)
<i>TIMER_FLAG_MCH</i> 30	多模式通道3捕获溢出标志, <i>TIMERx</i> ( <i>x</i> =0,7)
<i>TIMER_FLAG_SYS</i> B	系统源中止标志, <i>TIMERx</i> ( <i>x</i> =0,7)
<i>TIMER_FLAG_UND</i> ERFLOW	<i>TIMER</i> 计数下溢中断标志, <i>TIMERx</i> ( <i>x</i> =0,7)
<i>TIMER_FLAG_OVE</i> RFLOW	<i>TIMER</i> 计数上溢中断标志, <i>TIMERx</i> ( <i>x</i> =0,7)
<i>TIMER_FLAG_MCH</i> 00	多模式通道0捕获溢出标志, <i>TIMERx</i> ( <i>x</i> =0,7,14~16,40~44)
<i>TIMER_FLAG_MCH</i> 10	多模式通道1捕获溢出标志, <i>TIMERx</i> ( <i>x</i> =0,7)
<i>TIMER_FLAG_MCH</i> 20	多模式通道2捕获溢出标志, <i>TIMERx</i> ( <i>x</i> =0,7)
<i>TIMER_FLAG_MCH</i> 30	多模式通道3捕获溢出标志, <i>TIMERx</i> ( <i>x</i> =0,7)
<i>TIMER_FLAG_CH0</i> COMADD	通道0附加比较标志, <i>TIMERx</i> ( <i>x</i> =0,7)
<i>TIMER_FLAG_CH1</i> COMADD	通道1附加比较标志, <i>TIMERx</i> ( <i>x</i> =0,7)
<i>TIMER_FLAG_CH2</i>	通道2附加比较标志, <i>TIMERx</i> ( <i>x</i> =0,7)

COMADD	
TIMER_FLAG_CH3 COMADD	通道3附加比较标志, TIMERx(x=0,7)
输出参数{out}	
-	-
返回值	
FlagStatus	SET或者RESET

例如:

```
/* get TIMER0 update flags */
```

```
FlagStatus Flag_status = RESET;
```

```
Flag_status = timer_flag_get(TIMER0, TIMER_FLAG_UP);
```

### 函数 timer\_flag\_clear

函数timer\_flag\_clear描述见下表:

表 3-875. 函数 timer\_flag\_clear

函数名称	timer_flag_clear
函数原型	void timer_flag_clear(uint32_t timer_periph, uint32_t flag);
功能描述	清除外设TIMERx状态标志
先决条件	-
被调用函数	-
输入参数{in}	
timer_periph	TIMER外设
TIMERx(x=0, 1, 2, 7)	参考具体参数
输入参数{in}	
flag	状态标志
TIMER_FLAG_UP	更新标志, TIMERx(x=0,1,2,7)
TIMER_FLAG_CH0	通道0比较/捕获标志, TIMERx(x=0,1,2,7)
TIMER_FLAG_CH1	通道1比较/捕获标志, TIMERx(x=0,1,2,7)
TIMER_FLAG_CH2	通道2比较/捕获标志, TIMERx(x=0,1,2,7)
TIMER_FLAG_CH3	通道3比较/捕获标志, TIMERx(x=0,1,2,7)
TIMER_FLAG_CMT	通道换相更新标志, TIMERx(x=0,7)
TIMER_FLAG_TRG	触发标志, TIMERx(x=0,1,2,7)
TIMER_FLAG_BRK	BREAK标志, TIMERx(x=0,7)
TIMER_FLAG_CH0 O	通道0捕获溢出标志, TIMERx(x=0,1,2,7)
TIMER_FLAG_CH1 O	通道1捕获溢出标志, TIMERx(x=0,1,2,7)
TIMER_FLAG_CH2 O	通道2捕获溢出标志, TIMERx(x=0,1,2,7)

<code>TIMER_FLAG_CH3</code> <code>0</code>	通道3捕获溢出标志, <code>TIMERx(x=0,1,2,7)</code>
<code>TIMER_FLAG_MCH</code> <code>00</code>	多模式通道0捕获溢出标志, <code>TIMERx(x=0,7)</code>
<code>TIMER_FLAG_MCH</code> <code>10</code>	多模式通道1捕获溢出标志, <code>TIMERx(x=0,7)</code>
<code>TIMER_FLAG_MCH</code> <code>20</code>	多模式通道2捕获溢出标志, <code>TIMERx(x=0,7)</code>
<code>TIMER_FLAG_MCH</code> <code>30</code>	多模式通道3捕获溢出标志, <code>TIMERx(x=0,7)</code>
<code>TIMER_FLAG_SYS</code> <code>B</code>	系统源中止标志, <code>TIMERx(x=0,7)</code>
<code>TIMER_FLAG_UND</code> <code>ERFLOW</code>	<code>TIMER</code> 计数下溢中断标志, <code>TIMERx(x=0,7)</code>
<code>TIMER_FLAG_OVE</code> <code>RFLOW</code>	<code>TIMER</code> 计数上溢中断标志, <code>TIMERx(x=0,7)</code>
<code>TIMER_FLAG_MCH</code> <code>00</code>	多模式通道0捕获溢出标志, <code>TIMERx(x=0,7,14~16,40~44)</code>
<code>TIMER_FLAG_MCH</code> <code>10</code>	多模式通道1捕获溢出标志, <code>TIMERx(x=0,7)</code>
<code>TIMER_FLAG_MCH</code> <code>20</code>	多模式通道2捕获溢出标志, <code>TIMERx(x=0,7)</code>
<code>TIMER_FLAG_MCH</code> <code>30</code>	多模式通道3捕获溢出标志, <code>TIMERx(x=0,7)</code>
<code>TIMER_FLAG_CH0</code> <code>COMADD</code>	通道0附加比较标志, <code>TIMERx(x=0,7)</code>
<code>TIMER_FLAG_CH1</code> <code>COMADD</code>	通道1附加比较标志, <code>TIMERx(x=0,7)</code>
<code>TIMER_FLAG_CH2</code> <code>COMADD</code>	通道2附加比较标志, <code>TIMERx(x=0,7)</code>
<code>TIMER_FLAG_CH3</code> <code>COMADD</code>	通道3附加比较标志, <code>TIMERx(x=0,7)</code>
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* clear TIMER0 update flags */
```

```
timer_flag_clear(TIMER0, TIMER_FLAG_UP);
```



## 函数 timer\_interrupt\_enable

函数timer\_interrupt\_enable描述见下表:

表 3-876. 函数 timer\_interrupt\_enable

函数名称	timer_interrupt_enable
函数原型	void timer_interrupt_enable(uint32_t timer_periph, uint32_t interrupt);
功能描述	外设TIMERx中断使能
先决条件	-
被调用函数	-
输入参数{in}	
timer_periph	TIMER外设
TIMERx(x=0,1,2,7)	参考具体参数
输入参数{in}	
interrupt	中断源
TIMER_INT_UP	更新中断, TIMERx(x=0,1,2,7)
TIMER_INT_CH0	通道0比较/捕获中断, TIMERx(x=0,1,2,7)
TIMER_INT_CH1	通道1比较/捕获中断, TIMERx(x=0,1,2,7)
TIMER_INT_CH2	通道2比较/捕获中断, TIMERx(x=0,1,2,7)
TIMER_INT_CH3	通道3比较/捕获中断, TIMERx(x=0,1,2,7)
TIMER_INT_CMT	换相更新中断, TIMERx(x=0,7)
TIMER_INT_TRG	触发中断, TIMERx(x=0,1,2,7)
TIMER_INT_BRK	中止中断, TIMERx(x=0,7)
TIMER_INT_FLOW	TIMER计数溢出中断, TIMERx(x=0,7)
TIMER_INT_MCH0	多模式通道0比较/捕获中断, TIMERx(x=0,7)
TIMER_INT_MCH1	多模式通道1比较/捕获中断, TIMERx(x=0,7)
TIMER_INT_MCH2	多模式通道2比较/捕获中断, TIMERx(x=0,7)
TIMER_INT_MCH3	多模式通道3比较/捕获中断, TIMERx(x=0,7)
TIMER_INT_CH0C OMADD	通道0附加比较中断, TIMERx(x=0,7)
TIMER_INT_CH1C OMADD	通道1附加比较中断, TIMERx(x=0,7)
TIMER_INT_CH2C OMADD	通道2附加比较中断, TIMERx(x=0,7)
TIMER_INT_CH3C OMADD	通道3附加比较中断, TIMERx(x=0,7)
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* enable the TIMER0 update interrupt */
```

timer\_interrupt\_enable (TIMER0, TIMER\_INT\_UP);

### 函数 timer\_interrupt\_disable

函数timer\_interrupt\_disable描述见下表:

表 3-877. 函数 timer\_interrupt\_disable

函数名称	timer_interrupt_disable
函数原型	void timer_interrupt_disable (uint32_t timer_periph, uint32_t interrupt);
功能描述	外设TIMERx中断禁能
先决条件	-
被调用函数	-
输入参数{in}	
timer_periph	TIMER外设
TIMERx(x=0,1,2,7)	参考具体参数
输入参数{in}	
interrupt	中断源
TIMER_INT_UP	更新中断, TIMERx(x=0,1,2,7)
TIMER_INT_CH0	通道0比较/捕获中断, TIMERx(x=0,1,2,7)
TIMER_INT_CH1	通道1比较/捕获中断, TIMERx(x=0,1,2,7)
TIMER_INT_CH2	通道2比较/捕获中断, TIMERx(x=0,1,2,7)
TIMER_INT_CH3	通道3比较/捕获中断, TIMERx(x=0,1,2,7)
TIMER_INT_CMT	换相更新中断, TIMERx(x=0,7)
TIMER_INT_TRG	触发中断, TIMERx(x=0,1,2,7)
TIMER_INT_BRK	中止中断, TIMERx(x=0,7)
TIMER_INT_FLOW	TIMER计数溢出中断, TIMERx(x=0,7)
TIMER_INT_MCH0	多模式通道0比较/捕获中断, TIMERx(x=0,7)
TIMER_INT_MCH1	多模式通道1比较/捕获中断, TIMERx(x=0,7)
TIMER_INT_MCH2	多模式通道2比较/捕获中断, TIMERx(x=0,7)
TIMER_INT_MCH3	多模式通道3比较/捕获中断, TIMERx(x=0,7)
TIMER_INT_CH0C OMADD	通道0附加比较中断, TIMERx(x=0,7)
TIMER_INT_CH1C OMADD	通道1附加比较中断, TIMERx(x=0,7)
TIMER_INT_CH2C OMADD	通道2附加比较中断, TIMERx(x=0,7)
TIMER_INT_CH3C OMADD	通道3附加比较中断, TIMERx(x=0,7)
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* disable the TIMER0 update interrupt */
```

```
timer_interrupt_disable(TIMER0, TIMER_INT_UP);
```

### 函数 timer\_interrupt\_flag\_get

函数timer\_interrupt\_flag\_get描述见下表：

表 3-878. 函数 timer\_interrupt\_flag\_get

函数名称	timer_interrupt_flag_get
函数原型	FlagStatus timer_interrupt_flag_get(uint32_t timer_periph, uint32_t int_flag);
功能描述	获取外设TIMERx中断标志
先决条件	-
被调用函数	-
输入参数{in}	
timer_periph	TIMER外设
TIMERx(x=0,1,2,7)	参考具体参数
输入参数{in}	
int_flag	中断源
TIMER_INT_FLAG_UP	更新中断, TIMERx(x=0,1,2,7)
TIMER_INT_FLAG_CH0	通道0比较/捕获中断, TIMERx(x=0,1,2,7)
TIMER_INT_FLAG_CH1	通道1比较/捕获中断, TIMERx(x=0,1,2,7)
TIMER_INT_FLAG_CH2	通道2比较/捕获中断, TIMERx(x=0,1,2,7)
TIMER_INT_FLAG_CH3	通道3比较/捕获中断, TIMERx(x=0,1,2,7)
TIMER_INT_FLAG_CMT	换相更新中断, TIMERx(x=0,7)
TIMER_INT_FLAG_TRG	触发中断, TIMERx(x=0,1,2,7)
TIMER_INT_FLAG_BRK	中止中断, TIMERx(x=0,7)
TIMER_INT_FLAG_UNDERFLOW	TIMER计数下溢中断, TIMERx(x=0,7)
TIMER_INT_FLAG_OVERFLOW	TIMER计数上溢中断, TIMERx(x=0,7)
TIMER_INT_FLAG_MCH0	多模式通道0比较/捕获中断, TIMERx(x=0,7)
TIMER_INT_FLAG	多模式通道1比较/捕获中断, TIMERx(x=0,7)

<code>_MCH1</code>	
<code>TIMER_INT_FLAG</code> <code>_MCH2</code>	多模式通道2比较/捕获中断, <code>TIMERx(x=0,7)</code>
<code>TIMER_INT_FLAG</code> <code>_MCH3</code>	多模式通道3比较/捕获中断, <code>TIMERx(x=0,7)</code>
<code>TIMER_INT_FLAG</code> <code>_CH0COMADD</code>	通道0附加比较中断, <code>TIMERx(x=0,7)</code>
<code>TIMER_INT_FLAG</code> <code>_CH1COMADD</code>	通道1附加比较中断, <code>TIMERx(x=0,7)</code>
<code>TIMER_INT_FLAG</code> <code>_CH2COMADD</code>	通道2附加比较中断, <code>TIMERx(x=0,7)</code>
<code>TIMER_INT_FLAG</code> <code>_CH3COMADD</code>	通道3附加比较中断, <code>TIMERx(x=0,7)</code>
输出参数{out}	
-	-
返回值	
<b>FlagStatus</b>	SET或者RESET

例如:

```
/* get TIMER0 update interrupt flag */
```

```
FlagStatus Flag_interrupt = RESET;
```

```
Flag_interrupt = timer_interrupt_flag_get(TIMER0, TIMER_INT_FLAG_UP);
```

### 函数 `timer_interrupt_flag_clear`

函数`timer_interrupt_flag_clear`描述见下表:

表 3-879. 函数 `timer_interrupt_flag_clear`

函数名称	<code>timer_interrupt_flag_clear</code>
函数原型	<code>void timer_interrupt_flag_clear(uint32_t timer_periph, uint32_t int_flag);</code>
功能描述	清除外设 <code>TIMERx</code> 的中断标志
先决条件	-
被调用函数	-
输入参数{in}	
<b>timer_periph</b>	TIMER外设
<code>TIMERx(x=0,1,2,7)</code>	参考具体参数
输入参数{in}	
<b>int_flag</b>	中断源
<code>TIMER_INT_FLAG</code> <code>UP</code>	更新中断, <code>TIMERx(x=0,1,2,7)</code>
<code>TIMER_INT_FLAG</code> <code>_CH0</code>	通道0比较/捕获中断, <code>TIMERx(x=0,1,2,7)</code>

<code>TIMER_INT_FLAG</code> <code>_CH1</code>	通道1比较/捕获中断, <code>TIMERx(x=0,1,2,7)</code>
<code>TIMER_INT_FLAG</code> <code>_CH2</code>	通道2比较/捕获中断, <code>TIMERx(x=0,1,2,7)</code>
<code>TIMER_INT_FLAG</code> <code>_CH3</code>	通道3比较/捕获中断, <code>TIMERx(x=0,1,2,7)</code>
<code>TIMER_INT_FLAG</code> <code>_CMT</code>	换相更新中断, <code>TIMERx(x=0,7)</code>
<code>TIMER_INT_FLAG</code> <code>_TRG</code>	触发中断, <code>TIMERx(x=0,1,2,7)</code>
<code>TIMER_INT_FLAG</code> <code>_BRK</code>	中止中断, <code>TIMERx(x=0,7)</code>
<code>TIMER_INT_FLAG</code> <code>_UNDERFLOW</code>	TIMER计数下溢中断, <code>TIMERx(x=0,7)</code>
<code>TIMER_INT_FLAG</code> <code>_OVERFLOW</code>	TIMER计数上溢中断, <code>TIMERx(x=0,7)</code>
<code>TIMER_INT_FLAG</code> <code>_MCH0</code>	多模式通道0比较/捕获中断, <code>TIMERx(x=0,7)</code>
<code>TIMER_INT_FLAG</code> <code>_MCH1</code>	多模式通道1比较/捕获中断, <code>TIMERx(x=0,7)</code>
<code>TIMER_INT_FLAG</code> <code>_MCH2</code>	多模式通道2比较/捕获中断, <code>TIMERx(x=0,7)</code>
<code>TIMER_INT_FLAG</code> <code>_MCH3</code>	多模式通道3比较/捕获中断, <code>TIMERx(x=0,7)</code>
<code>TIMER_INT_FLAG</code> <code>_CH0COMADD</code>	通道0附加比较中断, <code>TIMERx(x=0,7)</code>
<code>TIMER_INT_FLAG</code> <code>_CH1COMADD</code>	通道1附加比较中断, <code>TIMERx(x=0,7)</code>
<code>TIMER_INT_FLAG</code> <code>_CH2COMADD</code>	通道2附加比较中断, <code>TIMERx(x=0,7)</code>
<code>TIMER_INT_FLAG</code> <code>_CH3COMADD</code>	通道3附加比较中断, <code>TIMERx(x=0,7)</code>
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* clear TIMER0 update interrupt flag */
```

```
timer_interrupt_flag_clear(TIMER0, TIMER_INT_FLAG_UP);
```

## 3.28. TMU

三角函数加速器（TMU）是一个完全可配置的单元，可执行常见的三角运算和算术运算操作。TMU可以减轻CPU的负担。章节[3.28.1](#)描述了TMU的寄存器列表，章节[3.28.2](#)对TMU库函数进行说明。

### 3.28.1. 外设寄存器说明

TMU寄存器列表如下表所示：

**表 3-880. TMU 寄存器**

寄存器名称	寄存器描述
TMU_CS	TMU控制和状态寄存器
TMU_IDATA	TMU输入数据寄存器
TMU_ODATA	TMU输出数据寄存器

### 3.28.2. 外设库函数说明

TMU库函数列表如下表所示：

**表 3-881. TMU 库函数**

函数名称	功能描述
tmu_deinit	复位 TMU 的所有寄存器
tmu_struct_para_init	使用默认值初始化 tmu_parameter_struct 结构体
tmu_init	初始化 TMU
tmu_dma_read_enable	使能 TMU 读请求（DMA）
tmu_dma_read_disable	禁能 TMU 读请求（DMA）
tmu_dma_write_enable	使能 TMU 写中断
tmu_dma_write_disable	禁能 TMU 写中断
tmu_one_q31_write	写一个 q1.31 格式数据
tmu_two_q31_write	写两个 q1.31 格式数据
tmu_two_q15_write	写两个 q1.15 格式数据
tmu_one_f32_write	写一个浮点格式数据
tmu_two_f32_write	写两个浮点格式数据
tmu_one_q31_read	读一个 q1.31 格式数据
tmu_two_q31_read	读两个 q1.31 格式数据
tmu_two_q15_read	读两个 q1.15 格式数据
tmu_one_f32_read	读一个浮点格式数据
tmu_two_f32_read	读两个浮点格式数据
tmu_flag_get	获取 TMU 标志位
tmu_flag_clear	清除 TMU 标志位
tmu_interrupt_enable	TMU 中断使能

函数名称	功能描述
tmu_interrupt_disable	TMU 中断禁能
tmu_interrupt_flag_get	获取 TMU 中断标志位
tmu_interrupt_flag_clear	清除 TMU 中断标志位

### 结构体 tmu\_parameter\_struct

表 3-882. 结构体 tmu\_parameter\_struct

成员名称	功能描述
mode	TMU运行模式 (TMU_MODE_COS,TMU_MODE_SIN,TMU_MODE_ATAN2,TMU_MODE_MODULE, TMU_MODE_SQRT)
scale	缩放因子(TMUSCALING_FACTOR_x(x=1,2,4,8,16,32,64,128))
output_floating	输出浮点格式数据使能(TMUTPUT_FLOAT_DISABLE, TMUTPUT_FLOAT_ENABLE)
input_floating	输入浮点格式数据使能(TMUTPUT_FLOAT_DISABLE, TMUTPUT_FLOAT_ENABLE)
dma_read	读TMU_ODATA寄存器DMA请求(TMUREAD_DMA_DISABLE, TMUREAD_DMA_ENABLE)
dma_write	写TMU_IDATA寄存器DMA请求(TMUWRITE_DMA_DISABLE, TMUWRITE_DMA_ENABLE)
read_times	读TMU_ODATA寄存器的次数(TMUREAD_TIMES_1, TMUREAD_TIMES_2)
write_times	写TMU_IDATA寄存器的次数(TMUWRITE_TIMES_1, TMUWRITE_TIMES_2)
output_width	输出数据宽度(TMUTPUT_WIDTH_32, TMUTPUT_WIDTH_16)
input_width	输入数据宽度(TMUTPUT_WIDTH_32, TMUTPUT_WIDTH_16)

### 函数 tmu\_deinit

函数tmu\_deinit的描述如下表：

表 3-883. 函数 tmu\_deinit

函数名称	tmu_deinit
函数原型	void tmu_deinit(void);
功能描述	复位TMU的所有寄存器
先决条件	-
被调用函数	rcu_periph_reset_enable / rcu_periph_reset_disable
输入参数{in}	
-	-
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* reset TMU */
```

```
tmu_deinit();
```

### 函数 tmu\_struct\_para\_init

函数tmu\_struct\_para\_init的描述如下表：

表 3-884. 函数 tmu\_struct\_para\_init

函数名称	tmu_struct_para_init
函数原型	void tmu_struct_para_init(tmu_parameter_struct* init_struct);
功能描述	使用默认值初始化tmu_parameter_struct结构体
先决条件	-
被调用函数	-
输入参数{in}	
init_struct	指向 tmu_parameter_struct结构体指针，结构体成员参考 <a href="#">结构体 tmu_parameter_struct</a> 。
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* initialize TMU init parameter struct with a default value */
```

```
tmu_parameter_struct tmu_initpara;
```

```
tmu_struct_para_init(&tmu_initpara);
```

### 函数 tmu\_init

函数tmu\_init的描述如下表：

表 3-885. 函数 tmu\_init

函数名称	tmu_init
函数原型	void tmu_init(tmu_parameter_struct* init_struct);
功能描述	初始化TMU
先决条件	-
被调用函数	-
输入参数{in}	
init_struct	指向 tmu_parameter_struct结构体指针，结构体成员参考 <a href="#">结构体 tmu_parameter_struct</a> 。
输出参数{out}	
-	-
返回值	



-	-
---	---

例如：

```

/* initialize TMU */

tmu_parameter_struct tmu_init_struct;

tmu_init_struct.mode = TMU_MODE_COS;

tmu_init_struct.scale = TMU_SCALING_FACTOR_1;

tmu_init_struct.output_floating = TMU_OUTPUT_FLOAT_DISABLE;

tmu_init_struct.input_floating = TMU_INPUT_FLOAT_DISABLE;

tmu_init_struct.dma_read = TMU_READ_DMA_ENABLE;

tmu_init_struct.dma_write = TMU_WRITE_DMA_ENABLE;

tmu_init_struct.read_times = TMU_READ_TIMES_2;

tmu_init_struct.write_times = TMU_WRITE_TIMES_2;

tmu_init_struct.output_width = TMU_OUTPUT_WIDTH_32;

tmu_init_struct.input_width = TMU_INPUT_WIDTH_32;

tmu_init(&tmu_init_struct);

```

### 函数 tmu\_dma\_read\_enable

函数tmu\_dma\_read\_enable的描述如下表：

**表 3-886. 函数 tmu\_dma\_read\_enable**

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

例如：

```

/* enable TMU DMA read request */

tmu_dma_read_enable();

```

## 函数 tmu\_dma\_read\_disable

函数tmu\_dma\_read\_disable的描述如下表：

表 3-887. 函数 tmu\_dma\_read\_disable

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

例如：

```
/* disable TMU DMA read request */
tmu_dma_read_disable();
```

## 函数 tmu\_dma\_write\_enable

函数tmu\_dma\_write\_enable的描述如下表：

表 3-888. 函数 tmu\_dma\_write\_enable

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

例如：

```
/* enable TMU DMA write request */
tmu_dma_write_enable();
```

## 函数 tmu\_dma\_write\_disable

函数tmu\_dma\_write\_disable的描述如下表：

**表 3-889. 函数 tmu\_dma\_write\_disable**

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

例如：

```
/* disable TMU DMA write request */
```

```
tmu_dma_write_disable();
```

## 函数 tmu\_one\_q31\_write

函数tmu\_one\_q31\_write的描述如下表：

**表 3-890. 函数 tmu\_one\_q31\_write**

函数名称	tmu_one_q31_write
函数原型	void tmu_one_q31_write(uint32_t data);
功能描述	写一个q1.31格式数据
先决条件	-
被调用函数	-
输入参数{in}	
<b>data</b>	q1.31格式的输入数据(0x00000000~0xFFFFFFFF)
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* write onedata in q1.31 format */
```

```
int32_t in = -2000;
```

```
tmu_one_q31_write((uint32_t)in);
```

## 函数 tmu\_two\_q31\_write

函数tmu\_two\_q31\_write的描述如下表：

表 3-891. 函数 tmu\_two\_q31\_write

函数名称	tmu_two_q31_write
函数原型	void tmu_two_q31_write(uint32_t data1, uint32_t data2);
功能描述	写两个q1.31格式数据
先决条件	-
被调用函数	-
输入参数{in}	
data1	q1.31格式的第一个输入数据(0x00000000~0xFFFFFFFF)
输入参数{in}	
data2	q1.31格式的第二个输入数据(0x00000000~0xFFFFFFFF)
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* write two data in q1.31 format */
int32_t in1 = -2000;
int32_t in2 = 3000;
tmu_two_q31_write((uint32_t)in1, (uint32_t)in2);
```

## 函数 tmu\_two\_q15\_write

函数tmu\_two\_q15\_write的描述如下表：

表 3-892. 函数 tmu\_two\_q15\_write

函数名称	tmu_two_q15_write
函数原型	void tmu_two_q15_write(uint16_t data1, uint16_t data2);
功能描述	写两个q1.15格式数据
先决条件	-
被调用函数	-
输入参数{in}	
data1	q1.15格式的第一个输入数据(0x0000~0xFFFF)
输入参数{in}	
data2	q1.15格式的第二个输入数据(0x0000~0xFFFF)
输出参数{out}	
-	-
返回值	

-	-
---	---

例如:

```
/* write two data in q1.15 format */
```

```
int16_t in1 = -2000;
```

```
int16_t in2 = 3000;
```

```
tmu_two_q15_write((uint16_t)in1, (uint16_t)in2);
```

### 函数 tmu\_one\_f32\_write

函数tmu\_one\_f32\_write的描述如下表:

**表 3-893. 函数 tmu\_one\_f32\_write**

函数名称	tmu_one_f32_write
函数原型	void tmu_one_f32_write(float data);
功能描述	写一个浮点格式数据
先决条件	-
被调用函数	-
输入参数{in}	
data	浮点格式的输入数据
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* write one data in floating point format */
```

```
float in = -2000.0f;
```

```
tmu_one_f32_write(in);
```

### 函数 tmu\_two\_f32\_write

函数tmu\_two\_f32\_write的描述如下表:

**表 3-894. 函数 tmu\_two\_f32\_write**

函数名称	tmu_two_f32_write
函数原型	void tmu_two_f32_write(float data1, float data2);
功能描述	写两个浮点格式数据
先决条件	-
被调用函数	-
输入参数{in}	

<b>data1</b>	浮点格式的第二个输入数据
<b>输入参数{in}</b>	
<b>data2</b>	浮点格式的第二个输入数据
<b>输出参数{out}</b>	
-	-
<b>返回值</b>	
-	-

例如：

```
/* write two data in floating point format */
```

```
float in1 = -2000.0f;
```

```
float in2 = 3000.0f;
```

```
tmu_two_f32_write(in1, in2);
```

### 函数 tmu\_one\_q31\_read

函数tmu\_one\_q31\_read的描述如下表：

**表 3-895. 函数 tmu\_one\_q31\_read**

<b>函数名称</b>	tmu_one_q31_read
<b>函数原型</b>	void tmu_one_q31_read(uint32_t* p);
<b>功能描述</b>	读一个q1.31格式数据
<b>先决条件</b>	-
<b>被调用函数</b>	-
<b>输入参数{in}</b>	
<b>p</b>	指向输出数据的指针（q1.31格式）
<b>输出参数{out}</b>	
-	-
<b>返回值</b>	
-	-

例如：

```
/* read one data in q1.31 format */
```

```
uint32_t out = 0;
```

```
tmu_one_q31_read (&out);
```

### 函数 tmu\_two\_q31\_read

函数tmu\_two\_q31\_read的描述如下表：

表 3-896. 函数 tmu\_two\_q31\_read

函数名称	tmu_two_q31_read
函数原型	void tmu_two_q31_read(uint32_t* p1, uint32_t* p2);
功能描述	读两个q1.31格式数据
先决条件	-
被调用函数	-
输入参数{in}	
p1	指向第一个输出数据的指针（q1.31格式）
输入参数{in}	
p2	指向第二个输出数据的指针（q1.31格式）
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* read two data in q1.31 format */
```

```
uint32_t out1 = 0;
```

```
uint32_t out2 = 0;
```

```
tmu_two_q31_read(&out1, &out2);
```

### 函数 tmu\_two\_q15\_read

函数tmu\_two\_q15\_read的描述如下表：

表 3-897. 函数 tmu\_two\_q15\_read

函数名称	tmu_two_q15_read
函数原型	void tmu_two_q15_read(uint16_t* p1, uint16_t* p2);
功能描述	读两个q1.15格式数据
先决条件	-
被调用函数	-
输入参数{in}	
p1	指向第一个输出数据的指针（q1.15格式）
输入参数{in}	
p2	指向第二个输出数据的指针（q1.15格式）
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* read two data in q1.15 format */
```

```
uint16_t out1 = 0;
```

```
uint16_t out2 = 0;
```

```
tmu_two_q15_read(&out1, &out2);
```

### 函数 tmu\_one\_f32\_read

函数tmu\_one\_f32\_read的描述如下表：

表 3-898. 函数 tmu\_one\_f32\_read

函数名称	tmu_one_f32_read
函数原型	void tmu_one_f32_read(float* p);
功能描述	读一个浮点格式数据
先决条件	-
被调用函数	-
输入参数{in}	
p	指向输出数据的指针（浮点格式）
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* read one data in floating point format */
```

```
float out = 0.0f;
```

```
tmu_one_f32_read (&out);
```

### 函数 tmu\_two\_f32\_read

函数tmu\_two\_f32\_read的描述如下表：

表 3-899. 函数 tmu\_two\_f32\_read

函数名称	tmu_two_f32_read
函数原型	void tmu_two_f32_read(float* p1, float* p2)
功能描述	读两个浮点格式数据
先决条件	-
被调用函数	-
输入参数{in}	
p1	指向第一个输出数据的指针（浮点格式）
输入参数{in}	
p2	指向第二个输出数据的指针（浮点格式）



输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* read two data in floating point format */
```

```
float out1 = 0.0f;
```

```
float out2 = 0.0f;
```

```
tmu_two_f32_read(&out1, &out2);
```

### 函数 tmu\_flag\_get

函数tmu\_flag\_get的描述如下表：

表 3-900. 函数 tmu\_flag\_get

函数名称	tmu_flag_get
函数原型	FlagStatus tmu_flag_get(uint32_t flag);
功能描述	获取TMU标志位
先决条件	-
被调用函数	-
输入参数{in}	
flag	TMU标志位
TMU_FLAG_OVRF	TMU溢出错误标志位
TMU_FLAG_END	TMU运算结束标志位
输出参数{out}	
-	-
返回值	
FlagStatus	SET or RESET

例如：

```
/* get TMU flag bit */
```

```
FlagStatus flag_value;
```

```
flag_value = tmu_flag_get(TMU_FLAG_OVRF);
```

### 函数 tmu\_flag\_clear

函数tmu\_flag\_clear的描述如下表：

表 3-901. 函数 tmu\_flag\_clear

函数名称	tmu_flag_clear
------	----------------

函数原型	void tmu_flag_clear(uint32_t flag);
功能描述	清除TMU标志位
先决条件	-
被调用函数	-
输入参数{in}	
flag	TMU标志位
TMU_FLAG_OVRF	TMU溢出错误标志位
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* clear TMU flag bit */
```

```
tmu_flag_clear(TMU_FLAG_OVRF);
```

### 函数 tmu\_interrupt\_enable

函数tmu\_interrupt\_enable的描述如下表：

表 3-902. 函数 tmu\_interrupt\_enable

函数名称	tmu_interrupt_enable
函数原型	void tmu_interrupt_enable(uint32_t interrupt);
功能描述	TMU中断使能
先决条件	-
被调用函数	-
输入参数{in}	
interrupt	TMU中断
TMU_INT_OVRF	TMU溢出中断
TMU_INT_END	TMU读TMU_ODATA寄存器请求中断
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* enable TMU interrupt */
```

```
tmu_interrupt_enable(TMU_INT_OVRF);
```

### 函数 tmu\_interrupt\_disable

函数tmu\_interrupt\_disable的描述如下表：

表 3-903. 函数 `tmu_interrupt_disable`

函数名称	<code>tmu_interrupt_disable</code>
函数原型	<code>void tmu_interrupt_disable(uint32_t interrupt);</code>
功能描述	TMU中断禁能
先决条件	-
被调用函数	-
输入参数{in}	
<b>interrupt</b>	TMU中断
<code>TMU_INT_OVRF</code>	TMU溢出中断
<code>TMU_INT_END</code>	TMU读TMU_ODATA寄存器请求中断
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* disable TMU interrupt */
tmu_interrupt_disable(TMU_INT_OVRF);
```

### 函数 `tmu_interrupt_flag_get`

函数`tmu_interrupt_flag_get`的描述如下表：

表 3-904. 函数 `tmu_interrupt_flag_get`

函数名称	<code>tmu_interrupt_flag_get</code>
函数原型	<code>FlagStatus tmu_interrupt_flag_get(uint32_t int_flag);</code>
功能描述	获取TMU中断标志位
先决条件	-
被调用函数	-
输入参数{in}	
<b>int_flag</b>	TMU中断标志位
<code>TMU_INT_FLAG_OVRF</code>	TMU溢出中断标志位
<code>TMU_INT_FLAG_END</code>	TMU读TMU_ODATA寄存器请求中断标志位
输出参数{out}	
-	-
返回值	
<b>FlagStatus</b>	SET或RESET

例如：

```
/* get the TMU interrupt flag bit*/
FlagStatus flag_value;
```

```
flag_value = tmu_interrupt_flag_get(TMU_INT_FLAG_OVRF);
```

### 函数 tmu\_interrupt\_flag\_clear

函数tmu\_interrupt\_flag\_clear的描述如下表:

表 3-905. 函数 tmu\_interrupt\_flag\_clear

函数名称	tmu_interrupt_flag_clear
函数原型	void tmu_interrupt_flag_clear(uint32_t int_flag);
功能描述	清除TMU中断标志位
先决条件	-
被调用函数	-
输入参数{in}	
int_flag	TMU中断标志位
<i>TMU_INT_FLAG_OVRF</i>	TMU溢出中断标志位
输出参数{out}	
-	-
返回值	
-	-

例如:

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

```
tmu_interrupt_flag_clear(TMU_INT_FLAG_OVRF);
```

## 3.29. UART

通用异步收发器(UART)提供了一个灵活方便的串行数据交换接口, 章节[3.29.1](#)描述了UART的寄存器列表, 章节[3.29.2](#)对UART库函数进行说明。

### 3.29.1. 外设寄存器说明

UART寄存器列表如下表所示:

表 3-906. UART 寄存器

寄存器名称	寄存器描述
UART_STAT0	状态寄存器0
UART_DATA	数据寄存器
UART_BAUD	波特率寄存器
UART_CTL0	控制寄存器0
UART_CTL1	控制寄存器1
UART_CTL2	控制寄存器2

寄存器名称	寄存器描述
UART_GP	保护时间和预分频器寄存器
UART_CHC	兼容性控制寄存器

### 3.29.2. 外设库函数说明

UART库函数列表如下表所示：

**表 3-907. UART 库函数**

库函数名称	库函数描述
uart_deinit	复位外设UART
uart_baudrate_set	配置UART波特率
uart_parity_config	配置UART奇偶校验
uart_word_length_set	配置UART字长
uart_stop_bit_set	配置UART停止位
uart_enable	使能UART
uart_disable	失能UART
uart_transmit_config	UART发送配置
uart_receive_config	UART接收配置
uart_data_first_config	配置数据传输时低位在前或高位在前
uart_invert_config	配置UART反转功能
uart_oversample_config	配置UART过采样模式
uart_sample_bit_config	配置UART单次采样方式
uart_data_transmit	UART发送数据功能
uart_data_receive	UART接收数据功能
uart_address_config	配置UART地址
uart_mute_mode_enable	使能UART静默模式
uart_mute_mode_disable	失能UART静默模式
uart_mute_mode_wakeup_config	配置UART静默模式唤醒方式
uart_lin_mode_enable	使能UART LIN模式
uart_lin_mode_disable	失能UART LIN模式
uart_lin_break_detection_length_config	配置UART LIN模式中断帧长度
uart_send_break	发送断开帧
uart_halfduplex_enable	使能UART半双工模式
uart_halfduplex_disable	失能UART半双工模式
uart_irda_mode_enable	使能UART串行红外编解码功能模块
uart_irda_mode_disable	失能UART串行红外编解码功能模块
uart_prescaler_config	在UART IrDA低功耗模式下配置外设时钟分频系数
uart_irda_lowpower_config	配置UART IrDA低功耗模式
uart_parity_check_coherence_config	配置奇偶校验一致性模式
uart_dma_receive_config	配置UART DMA接收

库函数名称	库函数描述
uart_dma_transmit_config	配置UART DMA发送
uart_flag_get	得到STAT/RFCSC寄存器中的标志
uart_flag_clear	清除UART状态
uart_interrupt_enable	使能UART中断
uart_interrupt_disable	失能UART中断
uart_interrupt_flag_get	得到UART中断和标志状态
uart_interrupt_flag_clear	清除UART中断标志位

### 枚举类型 `uart_flag_enum`

表 3-908. 枚举类型 `uart_flag_enum`

成员名称	功能描述
UART_FLAG_LBD	LIN断开检测标志
UART_FLAG_TBE	发送数据寄存器空
UART_FLAG_TC	发送完成
UART_FLAG_RBNE	读数据缓冲区非空
UART_FLAG_IDLE	空闲线检测标志
UART_FLAG_ORERR	溢出错误
UART_FLAG_NERR	噪声错误标志
UART_FLAG_FERR	帧错误
UART_FLAG_PERR	校验错误
UART_FLAG_EPERR	校验错误超前检测标志

### 枚举类型 `uart_interrupt_flag_enum`

表 3-909. 枚举类型 `uart_interrupt_flag_enum`

成员名称	功能描述
UART_INT_FLAG_PERR	奇偶校验错误中断标志
UART_INT_FLAG_TBE	发送寄存器空中断标志
UART_INT_FLAG_TC	发送完成中断标志
UART_INT_FLAG_RBNE	读缓冲区非空中断标志
UART_INT_FLAG_RBNE_ORERR	读缓冲区非空和溢出中断标志
UART_INT_FLAG_IDLE	空闲线检测中断标志
UART_INT_FLAG_LBD	LIN断开检测中断标志
UART_INT_FLAG_ERR_ORERR	溢出错误中断标志
UART_INT_FLAG_ERR_NERR	噪声错误中断标志
UART_INT_FLAG_ERR_FERR	帧错误中断标志

## 枚举类型 `uart_interrupt_enum`

表 3-910. 枚举类型 `uart_interrupt_enum`

成员名称	功能描述
UART_INT_PERR	奇偶校验错误中断使能
UART_INT_TBE	发送寄存器空中断使能
UART_INT_TC	发送完成中断使能
UART_INT_RBNE	读缓冲区非空中断和溢出错误中断使能
UART_INT_IDLE	空闲线检测中断使能
UART_INT_LBD	LIN断开检测中断使能
UART_INT_ERR	错误中断使能

## 枚举类型 `uart_invert_enum`

表 3-911. 枚举类型 `uart_invert_enum`

成员名称	功能描述
UART_DINV_ENABLE	数据位反转
UART_DINV_DISABLE	数据位不反转
UART_TXPIN_ENABLE	TX管脚电平反转
UART_TXPIN_DISABLE	TX管脚电平不反转
UART_RXPIN_ENABLE	RX管脚电平反转
UART_RXPIN_DISABLE	RX管脚电平不反转

## 函数 `uart_deinit`

函数 `uart_deinit` 描述见下表：

表 3-912. 函数 `uart_deinit`

函数名称	<code>uart_deinit</code>
函数原型	<code>void uart_deinit(uint32_t uart_periph);</code>
功能描述	复位外设UARTx
先决条件	-
被调用函数	<code>rcu_periph_reset_enable</code> / <code>rcu_periph_reset_disable</code>
输入参数{in}	
<code>uart_periph</code>	外设UARTx
<code>UARTx</code>	x=0,1,2,3
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* reset UART0 */
```

```
uart_deinit(UART0);
```

### 函数 `uart_baudrate_set`

函数 `uart_baudrate_set` 描述见下表：

表 3-913. 函数 `uart_baudrate_set`

函数名称	<code>uart_baudrate_set</code>
函数原型	<code>void uart_baudrate_set(uint32_t uart_periph, uint32_t baudval);</code>
功能描述	配置UART波特率
先决条件	-
被调用函数	<code>rcu_clock_freq_get</code>
输入参数{in}	
<code>uart_periph</code>	外设UARTx
<code>UARTx</code>	x=0,1,2,3
输入参数{in}	
<code>baudval</code>	波特率值
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* configure UART0 baud rate value */  
uart_baudrate_set(UART0, 115200);
```

### 函数 `uart_parity_config`

函数 `uart_parity_config` 描述见下表：

表 3-914. 函数 `uart_parity_config`

函数名称	<code>uart_parity_config</code>
函数原型	<code>void uart_parity_config(uint32_t uart_periph, uint32_t paritycfg);</code>
功能描述	配置UART奇偶校验
先决条件	-
被调用函数	-
输入参数{in}	
<code>uart_periph</code>	外设UARTx
<code>UARTx</code>	x=0,1,2,3
输入参数{in}	
<code>paritycfg</code>	配置UART奇偶校验
<code>UART_PM_NONE</code>	无校验
<code>UART_PM_ODD</code>	奇校验



UART_PM_EVEN	偶校验
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* configure UART0 parity */
```

```
uart_parity_config(UART0, UART_PM_EVEN);
```

### 函数 uart\_word\_length\_set

函数uart\_word\_length\_set描述见下表:

**表 3-915. 函数 uart\_word\_length\_set**

函数名称	uart_word_length_set
函数原型	void uart_word_length_set(uint32_t uart_periph, uint32_t wlen);
功能描述	配置UART字长
先决条件	-
被调用函数	-
输入参数{in}	
uart_periph	外设UARTx
UARTx	x=0,1,2,3
输入参数{in}	
wlen	配置UART字长
UART_WL_8BIT	8 bits
UART_WL_9BIT	9 bits
UART_WL_7BIT	7 bits
UART_WL_10BIT	10 bits
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* configure UART0 word length */
```

```
uart_word_length_set(UART0, UART_WL_9BIT);
```

### 函数 uart\_stop\_bit\_set

函数uart\_stop\_bit\_set描述见下表:

表 3-916. 函数 `uart_stop_bit_set`

函数名称	<code>uart_stop_bit_set</code>
函数原型	<code>void uart_stop_bit_set(uint32_t uart_periph, uint32_t stblen);</code>
功能描述	配置UART停止位
先决条件	-
被调用函数	-
输入参数{in}	
<code>uart_periph</code>	外设UARTx
<code>UARTx</code>	x=0,1,2,3
输入参数{in}	
<code>stblen</code>	配置UART停止位
<code>UART_STB_1BIT</code>	1 bit
<code>UART_STB_2BIT</code>	2 bit
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* configure UART0 stop bit length */
uart_stop_bit_set(UART0, UART_STB_1BIT);
```

### 函数 `uart_enable`

函数`uart_enable`描述见下表：

表 3-917. 函数 `uart_enable`

函数名称	<code>uart_enable</code>
函数原型	<code>void uart_enable(uint32_t uart_periph);</code>
功能描述	使能UART
先决条件	-
被调用函数	-
输入参数{in}	
<code>uart_periph</code>	外设UARTx
<code>UARTx</code>	x=0,1,2,3
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* enable UART0 */
```

```
uart_enable(UART0);
```

### 函数 `uart_disable`

函数 `uart_disable` 描述见下表：

表 3-918. 函数 `uart_disable`

函数名称	<code>uart_disable</code>
函数原型	<code>void uart_disable(uint32_t uart_periph);</code>
功能描述	失能UART
先决条件	-
被调用函数	-
输入参数{in}	
<code>uart_periph</code>	外设UARTx
<code>UARTx</code>	x=0,1,2,3
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* disable UART0 */
```

```
uart_disable(UART0);
```

### 函数 `uart_transmit_config`

函数 `uart_transmit_config` 描述见下表：

表 3-919. 函数 `uart_transmit_config`

函数名称	<code>uart_transmit_config</code>
函数原型	<code>void uart_transmit_config(uint32_t uart_periph, uint32_t txconfig);</code>
功能描述	UART发送器配置
先决条件	-
被调用函数	-
输入参数{in}	
<code>uart_periph</code>	外设UARTx
<code>UARTx</code>	x=0,1,2,3
输入参数{in}	
<code>txconfig</code>	使能/失能UART发送器
<code>UART_TRANSMIT_ENABLE</code>	使能UART发送
<code>UART_TRANSMIT_DISABLE</code>	失能UART发送

输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* configure UART0 transmitter */
```

```
uart_transmit_config(UART0,UART_TRANSMIT_ENABLE);
```

### 函数 uart\_receive\_config

函数uart\_receive\_config描述见下表：

表 3-920. 函数 uart\_receive\_config

函数名称	uart_receive_config
函数原型	void uart_receive_config(uint32_t uart_periph, uint32_t rxconfig);
功能描述	UART接收器配置
先决条件	-
被调用函数	-
输入参数{in}	
uart_periph	外设UARTx
UARTx	x=0,1,2,3
输入参数{in}	
rxconfig	使能/失能UART接收器
UART_RECEIVE_ENABLE	使能UART接收
UART_RECEIVE_DISABLE	失能UART接收
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* configure UART0 receiver */
```

```
uart_receive_config(UART0, UART_RECEIVE_ENABLE);
```

### 函数 uart\_data\_first\_config

函数uart\_data\_first\_config描述见下表：

表 3-921. 函数 uart\_data\_first\_config

函数名称	uart_data_first_config
------	------------------------

函数原型	void uart_data_first_config(uint32_t uart_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}	
uart_periph	外设UARTx
UARTx	x=0,1,2,3
输入参数{in}	
msbf	数据传输时低位在前/高位在前
UART_MSBF_LSB	数据传输时低位在前
UART_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 */
```

```
uart_data_first_config(UART0, UART_MSBF_LSB);
```

### 函数 uart\_invert\_config

函数uart\_invert\_config描述见下表：

表 3-922. 函数 uart\_invert\_config

函数名称	uart_invert_config
函数原型	void uart_invert_config(uint32_t uart_periph, uart_invert_enum invertpara);
功能描述	配置UART反转功能
先决条件	-
被调用函数	-
输入参数{in}	
uart_periph	外设UARTx
UARTx	x=0,1,2,3
输入参数{in}	
invertpara	参考 <a href="#">表3-911. 枚举类型uart_invert_enum</a>
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* configure UART0 inversion */
```

```
uart_invert_config(UART0, UART_DINV_ENABLE);
```

### 函数 `uart_oversample_config`

函数 `uart_oversample_config` 描述见下表：

表 3-923. 函数 `uart_oversample_config`

函数名称	<code>uart_oversample_config</code>
函数原型	<code>void uart_oversample_config(uint32_t uart_periph, uint32_t oversamp);</code>
功能描述	配置UART过采样模式
先决条件	-
被调用函数	-
输入参数{in}	
<code>uart_periph</code>	外设UARTx
<code>UARTx</code>	x=0,1,2,3
输入参数{in}	
<code>oversamp</code>	过采样值
<code>UART_OVSMOD_8</code>	8倍过采样
<code>UART_OVSMOD_16</code>	16倍过采样
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* config UART0 oversampling by 8 */
```

```
uart_oversample_config(UART0, UART_OVSMOD_8);
```

### 函数 `uart_sample_bit_config`

函数 `uart_sample_bit_config` 描述见下表：

表 3-924. 函数 `uart_sample_bit_config`

函数名称	<code>uart_sample_bit_config</code>
函数原型	<code>void uart_sample_bit_config(uint32_t uart_periph, uint32_t osb);</code>
功能描述	配置UART单次采样方式
先决条件	-
被调用函数	-
输入参数{in}	
<code>uart_periph</code>	外设UARTx
<code>UARTx</code>	x=0,1,2,3
输入参数{in}	

<b>osb</b>	单次采样方式
<i>UART_OSB_1BIT</i>	1次采样方法
<i>UART_OSB_3BIT</i>	3次采样方法
<b>输出参数{out}</b>	
-	-
<b>返回值</b>	
-	-

例如：

```
/* config UART0 1 bit sample mode */
```

```
uart_sample_bit_config(UART0, UART_OSB_1BIT);
```

### 函数 **uart\_data\_transmit**

函数uart\_data\_transmit描述见下表：

**表 3-925. 函数 uart\_data\_transmit**

<b>函数名称</b>	uart_data_transmit
<b>函数原型</b>	void uart_data_transmit(uint32_t uart_periph, uint16_t data);
<b>功能描述</b>	UART发送数据功能
<b>先决条件</b>	-
<b>被调用函数</b>	-
<b>输入参数{in}</b>	
<b>uart_periph</b>	外设UARTx
<i>UARTx</i>	x=0,1,2,3
<b>输入参数{in}</b>	
<b>data</b>	发送的数据（0x00-0x1FF）
<b>输出参数{out}</b>	
-	-
<b>返回值</b>	
-	-

例如：

```
/* UART0 transmit data */
```

```
uart_data_transmit(UART0, 0xAA);
```

### 函数 **uart\_data\_receive**

函数uart\_data\_receive描述见下表：

**表 3-926. 函数 uart\_data\_receive**

<b>函数名称</b>	uart_data_receive
<b>函数原型</b>	uint16_t uart_data_receive(uint32_t uart_periph);

功能描述	UART接收数据功能
先决条件	-
被调用函数	-
输入参数{in}	
uart_periph	外设UARTx
UARTx	x=0,1,2,3
输出参数{out}	
-	-
返回值	
uint16_t	接收到的数据（0x00-0x1FF）

例如：

```
/* UART0 receive data */

uint16_t temp;

temp = uart_data_receive(UART0);
```

### 函数 uart\_address\_config

函数uart\_address\_config描述见下表：

表 3-927. 函数 uart\_address\_config

函数名称	uart_address_config
函数原型	void uart_address_config(uint32_t uart_periph, uint8_t addr);
功能描述	配置UART地址
先决条件	-
被调用函数	-
输入参数{in}	
uart_periph	外设UARTx
UARTx	x=0,1,2,3
输入参数{in}	
addr	UART地址（0x00-0xFF）
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* configure address of the UART0 */

uart_address_config(UART0, 0x00);
```



## 函数 `uart_mute_mode_enable`

函数 `uart_mute_mode_enable` 描述见下表：

**表 3-928. 函数 `uart_mute_mode_enable`**

函数名称	<code>uart_mute_mode_enable</code>
函数原型	<code>void uart_mute_mode_enable(uint32_t uart_periph);</code>
功能描述	使能UART静默模式
先决条件	-
被调用函数	-
输入参数{in}	
<code>uart_periph</code>	外设UARTx
<code>UARTx</code>	x=0,1,2,3
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* enable UART0 receiver in mute mode */
```

```
uart_mute_mode_enable(UART0);
```

## 函数 `uart_mute_mode_disable`

函数 `uart_mute_mode_disable` 描述见下表：

**表 3-929. 函数 `uart_mute_mode_disable`**

函数名称	<code>uart_mute_mode_disable</code>
函数原型	<code>void uart_mute_mode_disable(uint32_t uart_periph);</code>
功能描述	失能UART静默模式
先决条件	-
被调用函数	-
输入参数{in}	
<code>uart_periph</code>	外设UARTx
<code>UARTx</code>	x=0,1,2,3
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* disable UART0 receiver in mute mode */
```

```
uart_mute_mode_disable(UART0);
```

## 函数 `uart_mute_mode_wakeup_config`

函数 `uart_mute_mode_wakeup_config` 描述见下表：

**表 3-930. 函数 `uart_mute_mode_wakeup_config`**

函数名称	<code>uart_mute_mode_wakeup_config</code>
函数原型	<code>void uart_mute_mode_wakeup_config(uint32_t uart_periph, uint32_t wmethod);</code>
功能描述	配置UART静默模式唤醒方式
先决条件	-
被调用函数	-
输入参数{in}	
<code>uart_periph</code>	外设UARTx
<code>UARTx</code>	x=0,1,2,3
输入参数{in}	
<code>wmethod</code>	两种方法用于进入或退出静默模式
<code>UART_WM_IDLE</code>	空闲线唤醒
<code>UART_WM_ADDR</code>	地址匹配唤醒
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* configure UART0 wakeup method in mute mode */
```

```
uart_mute_mode_wakeup_config(UART0, UART_WM_IDLE);
```

## 函数 `uart_lin_mode_enable`

函数 `uart_lin_mode_enable` 描述见下表：

**表 3-931. 函数 `uart_lin_mode_enable`**

函数名称	<code>uart_lin_mode_enable</code>
函数原型	<code>void uart_lin_mode_enable(uint32_t uart_periph);</code>
功能描述	使能UART LIN模式
先决条件	-
被调用函数	-
输入参数{in}	
<code>uart_periph</code>	外设UARTx
<code>UARTx</code>	x=0,1,2,3
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* UART0 LIN mode enable */

uart_lin_mode_enable(UART0);
```

### 函数 `uart_lin_mode_disable`

函数 `uart_lin_mode_disable` 描述见下表：

**表 3-932. 函数 `uart_lin_mode_disable`**

函数名称	<code>uart_lin_mode_disable</code>
函数原型	<code>void uart_lin_mode_disable(uint32_t uart_periph);</code>
功能描述	失能UART LIN模式
先决条件	-
被调用函数	-
输入参数{in}	
<code>uart_periph</code>	外设UARTx
<code>UARTx</code>	x=0,1,2,3
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* UART0 LIN mode disable */

uart_lin_mode_disable(UART0);
```

### 函数 `uart_lin_break_dection_length_config`

函数 `uart_lin_break_dection_length_config` 描述见下表：

**表 3-933. 函数 `uart_lin_break_dection_length_config`**

函数名称	<code>uart_lin_break_dection_length_config</code>
函数原型	<code>void uart_lin_break_dection_length_config(uint32_t uart_periph, uint32_t lflen);</code>
功能描述	配置UART LIN模式中断帧长度
先决条件	-
被调用函数	-
输入参数{in}	
<code>uart_periph</code>	外设UARTx
<code>UARTx</code>	x=0,1,2,3
输入参数{in}	
<code>lflen</code>	LIN模式中中断帧长度
<code>UART_LBLEN_10B</code>	断开帧长度为10 bits

UART_LBLEN_11B	断开帧长度为11 bits
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* configure LIN break frame length */
```

```
uart_lin_break_dection_length_config(UART0, UART_LBLEN_10B);
```

### 函数 uart\_send\_break

函数uart\_send\_break描述见下表：

表 3-934. 函数 uart\_halfduplex\_enable

函数名称	uart_send_break
函数原型	void uart_send_break(uint32_t uart_periph);
功能描述	使能UART半双工模式
先决条件	-
被调用函数	-
输入参数{in}	
uart_periph	外设UARTx
UARTx	x=0,1,2,3
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* send break frame */
```

```
uart_send_break(UART0);
```

### 函数 uart\_halfduplex\_enable

函数uart\_halfduplex\_enable描述见下表：

表 3-935. 函数 uart\_halfduplex\_enable

函数名称	uart_halfduplex_enable
函数原型	void uart_halfduplex_enable(uint32_t uart_periph);
功能描述	使能UART半双工模式
先决条件	-
被调用函数	-
输入参数{in}	

<b>uart_periph</b>	外设UARTx
<i>UARTx</i>	x=0,1,2,3
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* enable UART0 half duplex mode */
```

```
uart_halfduplex_enable(UART0);
```

### 函数 **uart\_halfduplex\_disable**

函数uart\_halfduplex\_disable描述见下表：

**表 3-936. 函数 uart\_halfduplex\_disable**

<b>函数名称</b>	uart_halfduplex_disable
<b>函数原型</b>	void uart_halfduplex_disable(uint32_t uart_periph);
<b>功能描述</b>	失能UART半双工模式
<b>先决条件</b>	-
<b>被调用函数</b>	-
输入参数{in}	
<b>uart_periph</b>	外设UARTx
<i>UARTx</i>	x=0,1,2,3
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* disable UART0 half duplex mode */
```

```
uart_halfduplex_disable(UART0);
```

### 函数 **uart\_irda\_mode\_enable**

函数uart\_irda\_mode\_enable描述见下表：

**表 3-937. 函数 uart\_irda\_mode\_enable**

<b>函数名称</b>	uart_irda_mode_enable
<b>函数原型</b>	void uart_irda_mode_enable(uint32_t uart_periph);
<b>功能描述</b>	使能UART串行红外编解码功能模块
<b>先决条件</b>	-
<b>被调用函数</b>	-

输入参数{in}	
<b>uart_periph</b>	外设UARTx
<i>UARTx</i>	x=0,1,2,3
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* enable UART0 IrDA mode */
```

```
uart_irda_mode_enable(UART0);
```

### 函数 **uart\_irda\_mode\_disable**

函数uart\_irda\_mode\_disable描述见下表：

**表 3-938. 函数 uart\_irda\_mode\_disable**

<b>函数名称</b>	uart_irda_mode_disable
<b>函数原型</b>	void uart_irda_mode_disable(uint32_t uart_periph);
<b>功能描述</b>	失能UART串行红外编解码功能模块
<b>先决条件</b>	-
<b>被调用函数</b>	-
输入参数{in}	
<b>uart_periph</b>	外设UARTx
<i>UARTx</i>	x=0,1,2,3
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* disable UART0 IrDA mode */
```

```
uart_irda_mode_disable(UART0);
```

### 函数 **uart\_prescaler\_config**

函数uart\_prescaler\_config描述见下表：

**表 3-939. 函数 uart\_prescaler\_config**

<b>函数名称</b>	uart_prescaler_config
<b>函数原型</b>	void uart_prescaler_config(uint32_t uart_periph, uint32_t psc);
<b>功能描述</b>	在UART IrDA低功耗模式下配置外设时钟分频系数
<b>先决条件</b>	-

被调用函数	-
输入参数{in}	
uart_periph	外设UARTx
UARTx	x=0,1,2,3
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 UART0 peripheral clock prescaler in UART IrDA low-power mode */
```

```
uart_prescaler_config(UART0, 0x00);
```

### 函数 uart\_irda\_lowpower\_config

函数uart\_irda\_lowpower\_config描述见下表:

表 3-940. 函数 uart\_irda\_lowpower\_config

函数名称	uart_irda_lowpower_config
函数原型	void uart_irda_lowpower_config(uint32_t uart_periph, uint32_t irlp);
功能描述	配置UART IrDA低功耗模式
先决条件	-
被调用函数	-
输入参数{in}	
uart_periph	外设UARTx
UARTx	x=0,1,2,3
输入参数{in}	
irlp	IrDA低功耗模式或正常模式
UART_IRLP_LOW	低功耗模式
UART_IRLP_NORMAL	正常模式
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* configure UART0 IrDA low-power */
```

```
uart_irda_lowpower_config(UART0, UART_IRLP_LOW);
```

## 函数 `uart_parity_check_coherence_config`

函数 `uart_parity_check_coherence_config` 描述见下表：

**表 3-941. 函数 `uart_parity_check_coherence_config`**

函数名称	<code>uart_parity_check_coherence_config</code>
函数原型	<code>void uart_parity_check_coherence_config(uint32_t uart_periph, uint32_t pcm);</code>
功能描述	配置奇偶校验一致性模式
先决条件	-
被调用函数	-
输入参数{in}	
<code>uart_periph</code>	外设UARTx
<code>UARTx</code>	x=0,1,2,3
输入参数{in}	
<code>pcm</code>	驱动使能的极性选择模式
<code>UART_PCM_NONE</code>	奇偶校验失能
<code>UART_PCM_EN</code>	奇偶校验使能
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* enabled UART0 configure parity check coherence */
```

```
uart_parity_check_coherence_config (UART0, UART_PCM_EN);
```

## 函数 `uart_dma_receive_config`

函数 `uart_dma_receive_config` 描述见下表：

**表 3-942. 函数 `uart_dma_receive_config`**

函数名称	<code>uart_dma_receive_config</code>
函数原型	<code>void uart_dma_receive_config(uint32_t uart_periph, uint32_t dmacmd);</code>
功能描述	配置UART DMA接收功能
先决条件	-
被调用函数	-
输入参数{in}	
<code>uart_periph</code>	外设UARTx
<code>UARTx</code>	x=0,1,2,3
输入参数{in}	
<code>dmacmd</code>	DMA使能/失能DMA接收功能
<code>UART_RECEIVE_DMA_ENABLE</code>	使能DMA接收功能



UART_RECEIVE_DMA_DISABLE	失能DMA接收功能
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* UART0 DMA enable for reception */
```

```
uart_dma_receive_config(UART0, UART_RECEIVE_DMA_ENABLE);
```

### 函数 uart\_dma\_transmit\_config

函数uart\_dma\_transmit\_config描述见下表：

表 3-943. 函数 uart\_dma\_transmit\_config

函数名称	uart_dma_transmit_config
函数原型	void uart_dma_transmit_config(uint32_t uart_periph, uint32_t dmacmd);
功能描述	配置UART DMA发送功能
先决条件	-
被调用函数	-
输入参数{in}	
uart_periph	外设UARTx
UARTx	x=0,1,2,3
输入参数{in}	
dmacmd	使能/失能DMA发送功能
UART_TRANSMIT_DMA_ENABLE	使能DMA发送功能
UART_TRANSMIT_DMA_DISABLE	失能DMA发送功能
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* UART0 DMA enable for transmission */
```

```
uart_dma_transmit_config(UART0, UART_TRANSMIT_DMA_ENABLE);
```

### 函数 uart\_flag\_get

函数uart\_flag\_get描述见下表：

表 3-944. 函数 `uart_flag_get`

函数名称	<code>uart_flag_get</code>
函数原型	<code>FlagStatus uart_flag_get(uint32_t uart_periph, uart_flag_enum flag);</code>
功能描述	获取UART STAT/CHC/RFCS寄存器标志位
先决条件	-
被调用函数	-
输入参数{in}	
<code>uart_periph</code>	外设UARTx
<code>UARTx</code>	x=0,1,2,3
输入参数{in}	
<code>flag</code>	UART标志位，参考 <a href="#">表3-908. 枚举类型uart_flag_enum</a> 只能选择一个参数
输出参数{out}	
-	-
返回值	
<code>FlagStatus</code>	SET或RESET

例如：

```
/* get flag UART0 state */
FlagStatus status;

status = uart_flag_get(UART0, UART_FLAG_TBE);
```

### 函数 `uart_flag_clear`

函数`uart_flag_clear`描述见下表：

表 3-945. 函数 `uart_flag_clear`

函数名称	<code>uart_flag_clear</code>
函数原型	<code>void uart_flag_clear(uint32_t uart_periph, uart_flag_enum flag);</code>
功能描述	清除UART状态寄存器标志位
先决条件	-
被调用函数	-
输入参数{in}	
<code>uart_periph</code>	外设UARTx
<code>UARTx</code>	x=0,1,2,3
输入参数{in}	
<code>flag</code>	UART标志位，参考 <a href="#">表3-908. 枚举类型uart_flag_enum</a> 只能选择一个参数
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* clear UART0 flag */

uart_flag_clear(UART0, UART_FLAG_TC);
```

### 函数 `uart_interrupt_enable`

函数 `uart_interrupt_enable` 描述见下表：

表 3-946. 函数 `uart_interrupt_enable`

函数名称	uart_interrupt_enable
函数原型	void uart_interrupt_enable(uint32_t uart_periph, uart_interrupt_enum interrupt);
功能描述	使能UART中断
先决条件	-
被调用函数	-
输入参数{in}	
uart_periph	外设UARTx
UARTx	x=0,1,2,3
输入参数{in}	
interrupt	UART中断UART标志位，参考 <a href="#">表3-910. 枚举类型uart_interrupt_enum</a> 只能选择一个参数
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* enable UART0 TBE interrupt */

uart_interrupt_enable(UART0, UART_INT_TBE);
```

### 函数 `uart_interrupt_disable`

函数 `uart_interrupt_disable` 描述见下表：

表 3-947. 函数 `uart_interrupt_disable`

函数名称	uart_interrupt_disable
函数原型	void uart_interrupt_disable(uint32_t uart_periph, uart_interrupt_enum interrupt);
功能描述	失能UART中断
先决条件	-
被调用函数	-
输入参数{in}	
uart_periph	外设UARTx
UARTx	x=0,1,2,3

输入参数{in}	
interrupt	UART中断UART标志位，参考 <a href="#">表3-910. 枚举类型uart_interrupt_enum</a> 只能选择一个参数
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* disable UART0 TBE interrupt */
```

```
uart_interrupt_disable(UART0, UART_INT_TBE);
```

### 函数 uart\_interrupt\_flag\_get

函数uart\_interrupt\_flag\_get描述见下表：

**表 3-948. 函数 uart\_interrupt\_flag\_get**

函数名称	uart_interrupt_flag_get
函数原型	FlagStatus uart_interrupt_flag_get(uint32_t uart_periph, uart_interrupt_flag_enum int_flag);
功能描述	获取UART中断标志位状态
先决条件	-
被调用函数	-
输入参数{in}	
uart_periph	外设UARTx
UARTx	x=0,1,2,3
输入参数{in}	
int_flag	UART中断标志，参考 <a href="#">表3-909. 枚举类型uart_interrupt_flag_enum</a> 只能选择一个参数
输出参数{out}	
-	-
返回值	
FlagStatus	SET或RESET

例如：

```
/* get the UART0 interrupt flag status */
```

```
FlagStatus status;
```

```
status = uart_interrupt_flag_get(UART0, UART_INT_FLAG_RBNE);
```

### 函数 uart\_interrupt\_flag\_clear

函数uart\_interrupt\_flag\_clear描述见下表：

表 3-949. 函数 `uart_interrupt_flag_clear`

函数名称	<code>uart_interrupt_flag_clear</code>
函数原型	<code>void uart_interrupt_flag_clear(uint32_t uart_periph, uart_interrupt_flag_enum int_flag);</code>
功能描述	清除UART中断标志位状态
先决条件	-
被调用函数	-
输入参数{in}	
<code>uart_periph</code>	外设UARTx
<code>UARTx</code>	x=0,1,2,3
输入参数{in}	
<code>int_flag</code>	UART中断标志, 参考 <a href="#">表3-909. 枚举类型uart_interrupt_flag_enum</a> 只能选择一个参数
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* clear the UART0 interrupt flag */
```

```
uart_interrupt_flag_clear(UART0, UART_INT_FLAG_TC);
```

### 3.30. 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 [3.30.1](#) 描述了 WWDGT 的寄存器列表, 章节 [3.30.2](#) 对 WWDGT 库函数进行说明。

#### 3.30.1. 外设寄存器说明

WWDGT 寄存器列表如下表所示:

表 3-950. 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

#### 3.30.2. 外设库函数说明

WWDGT 库函数列表如下表所示:

表 3-951. 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_counter_update	设置WWDGT计数器更新值
wwdgt_struct_para_init	初始化WWDGT配置结构体的参数为缺省值
wwdgt_cfg_init	初始化WWDGT配置并启动计数器
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

### 结构体 wwdgt\_cfg\_parameter\_struct

表 3-952. 结构体 wwdgt\_cfg\_parameter\_struct

成员名称	功能描述
counter	WWDGT计数器计数值
reset_control	WWDGT复位控制位
ewie_control	WWDGT提前唤醒中断控制位
window	WWDGT窗口起始值
window_end_position	WWDGT窗口结束位置值
prescaler	WWDGT分频值

### 函数 wwdgt\_deinit

函数wwdgt\_deinit描述见下表：

表 3-953. 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\_counter\_update

函数wwdgt\_counter\_update描述见下表：

**表 3-954. 函数 wwdgt\_counter\_update**

函数名称	wwdgt_counter_update
函数原型	void wwdgt_counter_update(uint16_t counter_value);
功能描述	设置WWDGT计数器更新值
先决条件	-
被调用函数	-
输入参数{in}	
counter_value	计数器值，数值范围为0x0000 - 0x3FFF
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* update WWDGT counter to 0x3FFF */
wwdgt_counter_update(0x3FFF);
```

## 函数 wwdgt\_struct\_para\_init

函数wwdgt\_struct\_para\_init描述见下表：

**表 3-955. 函数 wwdgt\_struct\_para\_init**

函数名称	wwdgt_struct_para_init
函数原型	void wwdgt_struct_para_init(wwdgt_cfg_parameter_struct *cfg_struct);
功能描述	初始化WWDGT配置结构体的参数为缺省值
先决条件	-
被调用函数	-
输入参数{in}	
cfg_struct	一个已经定义的wwdgt_cfg_parameter_struct结构体变量地址，参考 <a href="#">表3-952. 结构体wwdgt_cfg_parameter_struct</a>
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* initialize the parameters of WWDGT */
wwdgt_cfg_parameter_struct wwdgt_cfg_struct;
wwdgt_struct_para_init(&wwdgt_cfg_struct);
```

## 函数 wwdgt\_cfg\_init

函数wwdgt\_cfg\_init描述见下表:

**表 3-956. 函数 wwdgt\_cfg\_init**

函数名称	wwdgt_cfg_init
函数原型	void wwdgt_cfg_init(wwdgt_cfg_parameter_struct *cfg_struct);
功能描述	初始化WWDGT配置并启动计数器
先决条件	-
被调用函数	-
输入参数{in}	
cfg_struct	一个已经定义的wwdgt_cfg_parameter_struct结构体变量地址, 参考 <a href="#">表3-952. 结构体wwdgt_cfg_parameter_struct</a>
输出参数{out}	
-	-
返回值	
-	-

例如:

```

/* initialize WWDGT */
wwdgt_cfg_parameter_struct wwdgt_cfg_struct;
wwdgt_struct_para_init(&wwdgt_cfg_struct);

wwdgt_cfg_struct.counter = 16383U;
wwdgt_cfg_struct.reset_control = WWDGT_RESET_ENABLE;
wwdgt_cfg_struct.ewie_control = WWDGT_EWIE_DISABLE;
wwdgt_cfg_struct.window = 12000U;
wwdgt_cfg_struct.window_end_position = WWDGT_CFG_WEPS_THREE_QUARTER;
wwdgt_cfg_struct.prescaler = WWDGT_CFG_PSC_DIV8;
wwdgt_cfg_init(&wwdgt_cfg_struct);

```

## 函数 wwdgt\_flag\_get

函数wwdgt\_flag\_get描述见下表:

**表 3-957. 函数 wwdgt\_flag\_get**

函数名称	wwdgt_flag_get
函数原型	FlagStatus wwdgt_flag_get(uint32_t flag);
功能描述	检查WWDGT中断标志位是否置位
先决条件	-
被调用函数	-
输入参数{in}	
flag	WWDGT中断标志
WWDGT_FLAG_EW	提前唤醒中断标志



<i>IF</i>	
<i>WWDGT_FLAG_RE FEF</i>	刷新错误标志
<i>WWDGT_FLAG_UN DFF</i>	下溢错误标志
输出参数{out}	
-	-
返回值	
<b>FlagStatus</b>	SET or RESET

例如:

```
/* test if early wakeup interrupt flag is set */
```

```
FlagStatus status;
```

```
status = wwdgt_flag_get(WWDGT_FLAG_EWIF);
```

### 函数 **wwdgt\_flag\_clear**

函数 **wwdgt\_flag\_clear** 描述见下表:

**表 3-958. 函数 **wwdgt\_flag\_clear****

函数名称	wwdgt_flag_clear
函数原型	void wwdgt_flag_clear(uint32_t flag);
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}	
<b>flag</b>	WWDGT中断标志
<i>WWDGT_FLAG_EW IF</i>	提前唤醒中断标志
<i>WWDGT_FLAG_RE FEF</i>	刷新错误标志
<i>WWDGT_FLAG_UN DFF</i>	下溢错误标志
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* clear early wakeup interrupt state of WWDGT */
```

```
wwdgt_flag_clear(WWDGT_FLAG_EWIF);
```

## 4. 版本历史

表 4-1. 版本历史

版本号.	说明	日期
1.0	首次发布	2026 年 2 月 28 日

## Important Notice

This document is the property of GigaDevice Semiconductor Inc. and its subsidiaries (the "Company"). This document, including any product of the Company described in this document (the "Product"), is owned by the Company according to the laws of the People's Republic of China and other applicable laws. The Company reserves all rights under such laws and no Intellectual Property Rights are transferred (either wholly or partially) or licensed by the Company (either expressly or impliedly) herein. The names and brands of third party referred thereto (if any) are the property of their respective owner and referred to for identification purposes only.

To the maximum extent permitted by applicable law, the Company makes no representations or warranties of any kind, express or implied, with regard to the merchantability and the fitness for a particular purpose of the Product, nor does the Company assume any liability arising out of the application or use of any Product. Any information provided in this document is provided only for reference purposes. It is the sole responsibility of the user of this document to determine whether the Product is suitable and fit for its applications and products planned, and properly design, program, and test the functionality and safety of its applications and products planned using the Product. The Product is designed, developed, and/or manufactured for ordinary business, industrial, personal, and/or household applications only, and the Product is not designed or intended for use in (i) safety critical applications such as weapons systems, nuclear facilities, atomic energy controller, combustion controller, aeronautic or aerospace applications, traffic signal instruments, pollution control or hazardous substance management; (ii) life-support systems, other medical equipment or systems (including life support equipment and surgical implants); (iii) automotive applications or environments, including but not limited to applications for active and passive safety of automobiles (regardless of front market or aftermarket), for example, EPS, braking, ADAS (camera/fusion), EMS, TCU, BMS, BSG, TPMS, Airbag, Suspension, DMS, ICMS, Domain, ESC, DCDC, e-clutch, advanced-lighting, etc.. Automobile herein means a vehicle propelled by a self-contained motor, engine or the like, such as, without limitation, cars, trucks, motorcycles, electric cars, and other transportation devices; and/or (iv) other uses where the failure of the device or the Product can reasonably be expected to result in personal injury, death, or severe property or environmental damage (collectively "Unintended Uses"). Customers shall take any and all actions to ensure the Product meets the applicable laws and regulations. The Company is not liable for, in whole or in part, and customers shall hereby release the Company as well as its suppliers and/or distributors from, any claim, damage, or other liability arising from or related to all Unintended Uses of the Product. Customers shall indemnify and hold the Company, and its officers, employees, subsidiaries, affiliates as well as its suppliers and/or distributors harmless from and against all claims, costs, damages, and other liabilities, including claims for personal injury or death, arising from or related to any Unintended Uses of the Product.

Information in this document is provided solely in connection with the Product. The Company reserves the right to make changes, corrections, modifications or improvements to this document and the Product described herein at any time without notice. The Company shall have no responsibility whatsoever for conflicts or incompatibilities arising from future changes to them. Information in this document supersedes and replaces information previously supplied in any prior versions of this document.