# Table of Contents

Table of Contents ......................................................................................................................... 1
List of Figures ................................................................................................................................. 4
List of Tables ................................................................................................................................. 5

1. Introduction ............................................................................................................................... 25
      1.1.1. Peripherals............................................................................................................... 25
      1.1.2. Naming rules ........................................................................................................... 26

2. Firmware Library Overview ....................................................................................................... 27
   2.1. File Structure of Firmware Library ................................................................................. 27
      2.1.1. Examples Folder ....................................................................................................... 28
      2.1.2. Firmware Folder ....................................................................................................... 28
      2.1.3. Template Folder ....................................................................................................... 28
      2.1.4. Utilities Folder ......................................................................................................... 31
   2.2. File descriptions of Firmware Library .............................................................................. 31

3. Firmware Library of Standard Peripherals ........................................................................... 33
   3.1. Overview of Firmware Library of Standard Peripherals................................................. 33
   3.2. ADC ................................................................................................................................. 33
      3.2.1. Descriptions of Peripheral registers ....................................................................... 33
      3.2.2. Descriptions of Peripheral functions ..................................................................... 34
   3.3. CAU ................................................................................................................................. 58
      3.3.1. Descriptions of Peripheral registers ....................................................................... 58
      3.3.2. Descriptions of Peripheral functions ..................................................................... 59
   3.4. CMP ................................................................................................................................. 86
      3.4.1. Descriptions of Peripheral registers ....................................................................... 86
      3.4.2. Descriptions of Peripheral functions ..................................................................... 87
   3.5. CRC ................................................................................................................................. 98
      3.5.1. Descriptions of Peripheral registers ....................................................................... 98
      3.5.2. Descriptions of Peripheral functions ..................................................................... 99
   3.6. CTC ................................................................................................................................ 106
      3.6.1. Descriptions of Peripheral registers ....................................................................... 107
      3.6.2. Descriptions of Peripheral functions ..................................................................... 107
   3.7. DBG ................................................................................................................................ 120
      3.7.1. Descriptions of Peripheral registers ....................................................................... 120
3.7.2. Descriptions of Peripheral functions .......................................................... 120

3.8. DAC .................................................................................................................. 124
  3.8.1. Descriptions of Peripheral registers............................................................... 124
  3.8.2. Descriptions of Peripheral functions ......................................................... 125

3.9. DMA/DMAMUX ............................................................................................... 138
  3.9.1. Descriptions of Peripheral registers............................................................... 139
  3.9.2. Descriptions of Peripheral functions ......................................................... 139

3.10. EXTI ................................................................................................................ 181
  3.10.1. Descriptions of Peripheral registers............................................................. 181
  3.10.2. Descriptions of Peripheral functions ....................................................... 181

3.11. FMC ............................................................................................................... 189
  3.11.1. Descriptions of Peripheral registers............................................................. 189
  3.11.2. Descriptions of Peripheral functions ....................................................... 190

3.12. FWDGT .......................................................................................................... 210
  3.12.1. Descriptions of Peripheral registers............................................................. 210
  3.12.2. Descriptions of Peripheral functions ....................................................... 210

3.13. GPIO ............................................................................................................... 216
  3.13.1. Descriptions of Peripheral registers............................................................. 216
  3.13.2. Descriptions of Peripheral functions ....................................................... 216

  3.14.1. Descriptions of Peripheral registers............................................................. 227
  3.14.2. Descriptions of Peripheral functions ....................................................... 227

3.15. LPTIMER ....................................................................................................... 266
  3.15.1. Descriptions of Peripheral registers............................................................. 266
  3.15.2. Descriptions of Peripheral functions ....................................................... 267

3.16. LPUART ........................................................................................................ 285
  3.16.1. Descriptions of Peripheral registers............................................................. 285
  3.16.2. Descriptions of Peripheral functions ....................................................... 286

3.17. MISC ............................................................................................................ 314
  3.17.1. Descriptions of Peripheral registers............................................................. 314
  3.17.2. Descriptions of Peripheral functions ....................................................... 315

3.18. PMU ............................................................................................................... 321
  3.18.1. Descriptions of Peripheral registers............................................................. 321
  3.18.2. Descriptions of Peripheral functions ....................................................... 321

3.19. RCU .............................................................................................................. 339
  3.19.1. Descriptions of Peripheral registers............................................................. 339
  3.19.2. Descriptions of Peripheral functions ....................................................... 339

3.20. RTC ............................................................................................................... 376
  3.20.1. Descriptions of Peripheral registers............................................................. 377
3.20.2. Descriptions of Peripheral functions ................................................................. 377

3.21. SLCD .................................................................................................................. 404
  3.21.1. Descriptions of Peripheral registers ............................................................ 404
  3.21.2. Descriptions of Peripheral functions .......................................................... 405

3.22. SPI .................................................................................................................... 422
  3.22.1. Descriptions of Peripheral registers ............................................................ 422
  3.22.2. Descriptions of Peripheral functions .......................................................... 423

3.23. SYSCFG ....................................................................................................... 446
  3.23.1. Descriptions of Peripheral registers ............................................................ 446
  3.23.2. Descriptions of Peripheral functions .......................................................... 446

3.24. TIMER .......................................................................................................... 452
  3.24.1. Descriptions of Peripheral registers ............................................................ 452
  3.24.2. Descriptions of Peripheral functions .......................................................... 452

3.25. TRNG ............................................................................................................ 499
  3.25.1. Descriptions of Peripheral registers ............................................................ 499
  3.25.2. Descriptions of Peripheral functions .......................................................... 499

3.26. USART ......................................................................................................... 504
  3.26.1. Descriptions of Peripheral registers ............................................................ 504
  3.26.2. Descriptions of Peripheral functions .......................................................... 505

3.27. VREF ........................................................................................................... 555
  3.27.1. Descriptions of Peripheral registers ............................................................ 555
  3.27.2. Descriptions of Peripheral functions .......................................................... 556

3.28. WWDGT ....................................................................................................... 560
  3.28.1. Descriptions of Peripheral registers ............................................................ 560
  3.28.2. Descriptions of Peripheral functions .......................................................... 560

4. Revision history .................................................................................................. 566
List of Figures

Figure 2-1. File structure of firmware library of GD32L23x ................................................................. 27
Figure 2-2. Select peripheral example files ................................................................................................. 29
Figure 2-3. Copy the peripheral example files ............................................................................................. 29
Figure 2-4. Open the project file .................................................................................................................. 30
Figure 2-5. Configure project files ................................................................................................................ 30
Figure 2-6. Compile-debug-download .......................................................................................................... 31
List of Tables

Table 1-1. Peripherals .............................................................................................................................................. 25
Table 2-1. Function descriptions of Firmware Library .......................................................................................... 31
Table 3-1. Peripheral function format of Firmware Library ................................................................................ 33
Table 3-2. ADC Registers ...................................................................................................................................... 33
Table 3-3. ADC firmware function ...................................................................................................................... 34
Table 3-4. Function adc_deinit .......................................................................................................................... 35
Table 3-5. Function adc_enable .......................................................................................................................... 35
Table 3-6. Function adc_disable ........................................................................................................................ 36
Table 3-7. Function adc_calibration_enable ...................................................................................................... 36
Table 3-8. Function adc_dma_mode_enable ........................................................................................................ 37
Table 3-9. Function adc_dma_mode_disable ...................................................................................................... 37
Table 3-10. Function adc_discontinuous_mode_config ....................................................................................... 38
Table 3-11. Function adc_special_function_config .......................................................................................... 38
Table 3-12. Function adc_channel_16_to_19 ...................................................................................................... 39
Table 3-13. Function adc_data_alignment_config ............................................................................................. 40
Table 3-14. Function adc_channel_length_config ............................................................................................ 40
Table 3-15. Function adc_regular_channel_config ........................................................................................... 41
Table 3-16. Function adc_inserted_channel_config ........................................................................................... 42
Table 3-17. Function adc_inserted_channel_offset_config ................................................................................ 43
Table 3-18. Function adc_external_trigger_config .......................................................................................... 44
Table 3-19. Function adc_external_trigger_source_config ................................................................................. 44
Table 3-20. Function adc_software_trigger_enable ........................................................................................... 46
Table 3-21. Function adc_regular_data_read ...................................................................................................... 46
Table 3-22. Function adc_inserted_data_read .................................................................................................... 47
Table 3-23. Function adc_watchdog_single_channel_enable ................................................................................ 47
Table 3-24. Function adc_watchdog_group_channel_enable ................................................................................ 48
Table 3-25. Function adc_watchdog_disable ...................................................................................................... 48
Table 3-26. Function adc_watchdog_threshold_config ....................................................................................... 49
Table 3-27. Function adc_resolution_config ...................................................................................................... 49
Table 3-28. Function adc_oversample_mode_config ............................................................................................ 50
Table 3-29. Function adc_oversample_mode_enable ........................................................................................... 52
Table 3-30. Function adc_oversample_mode_disable .......................................................................................... 52
Table 3-31. Function adc_charge_pulse_width_counter ...................................................................................... 53
Table 3-32. Function adc_charge_flag_get .......................................................................................................... 53
Table 3-33. Function adc_regular_software_startconv_flag_get ........................................................................... 54
Table 3-34. Function adc_inserted_software_startconv_flag_get ......................................................................... 54
Table 3-35. Function adc_flag_get ....................................................................................................................... 55
Table 3-36. Function adc_flag_clear .................................................................................................................. 55
Table 3-37. Function adc_interrupt_enable ....................................................................................................... 56
Table 3-38. Function adc_interrupt_disable ..................................................................................................... 56
<table>
<thead>
<tr>
<th>Table</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>3-39.</td>
<td>Function adc_interrupt_flag_get</td>
</tr>
<tr>
<td>3-40.</td>
<td>Function adc_interrupt_flag_clear</td>
</tr>
<tr>
<td>3-41.</td>
<td>CAU Registers</td>
</tr>
<tr>
<td>3-42.</td>
<td>CAU firmware function</td>
</tr>
<tr>
<td>3-43.</td>
<td>Structure cau_key_parameter_struct</td>
</tr>
<tr>
<td>3-44.</td>
<td>Structure cau_iv_parameter_struct</td>
</tr>
<tr>
<td>3-45.</td>
<td>Structure cau_context_parameter_struct</td>
</tr>
<tr>
<td>3-46.</td>
<td>Structure cau_parameter_struct</td>
</tr>
<tr>
<td>3-47.</td>
<td>Function cau_deinit</td>
</tr>
<tr>
<td>3-48.</td>
<td>Function cau_struct_para_init</td>
</tr>
<tr>
<td>3-49.</td>
<td>Function cau_key_struct_para_init</td>
</tr>
<tr>
<td>3-50.</td>
<td>Function cau_iv_struct_para_init</td>
</tr>
<tr>
<td>3-51.</td>
<td>Function cau_context_struct_para_init</td>
</tr>
<tr>
<td>3-52.</td>
<td>Function cau_enable</td>
</tr>
<tr>
<td>3-53.</td>
<td>Function cau_disable</td>
</tr>
<tr>
<td>3-54.</td>
<td>Function cau_dma_enable</td>
</tr>
<tr>
<td>3-55.</td>
<td>Function cau_dma_disable</td>
</tr>
<tr>
<td>3-56.</td>
<td>Function cau_init</td>
</tr>
<tr>
<td>3-57.</td>
<td>Function cau_aes_keysize_config</td>
</tr>
<tr>
<td>3-58.</td>
<td>Function cau_key_init</td>
</tr>
<tr>
<td>3-59.</td>
<td>Function cau_iv_init</td>
</tr>
<tr>
<td>3-60.</td>
<td>Function cau_phase_config</td>
</tr>
<tr>
<td>3-61.</td>
<td>Function cau_fifo_flush</td>
</tr>
<tr>
<td>3-62.</td>
<td>Function cau_enable_state_get</td>
</tr>
<tr>
<td>3-63.</td>
<td>Function cau_data_write</td>
</tr>
<tr>
<td>3-64.</td>
<td>Function cau_data_read</td>
</tr>
<tr>
<td>3-65.</td>
<td>Function cau_context_save</td>
</tr>
<tr>
<td>3-66.</td>
<td>Function cau_context_restore</td>
</tr>
<tr>
<td>3-67.</td>
<td>Function cau_aes_ecb</td>
</tr>
<tr>
<td>3-68.</td>
<td>Function cau_aes_cbc</td>
</tr>
<tr>
<td>3-69.</td>
<td>Function cau_aes_cmc</td>
</tr>
<tr>
<td>3-70.</td>
<td>Function cau_aes_cfb</td>
</tr>
<tr>
<td>3-71.</td>
<td>Function cau_aes_ofb</td>
</tr>
<tr>
<td>3-72.</td>
<td>Function cau_aes_gcm</td>
</tr>
<tr>
<td>3-73.</td>
<td>Function cau_aes_ccm</td>
</tr>
<tr>
<td>3-74.</td>
<td>Function cau_tdes_ecb</td>
</tr>
<tr>
<td>3-75.</td>
<td>Function cau_tdes_cbc</td>
</tr>
<tr>
<td>3-76.</td>
<td>Function cau_des_ecb</td>
</tr>
<tr>
<td>3-77.</td>
<td>Function cau_des_cbc</td>
</tr>
<tr>
<td>3-78.</td>
<td>Function cau_interrupt_enable</td>
</tr>
<tr>
<td>3-79.</td>
<td>Function cau_interrupt_disable</td>
</tr>
<tr>
<td>3-80.</td>
<td>Function cau_interrupt_flag_get</td>
</tr>
<tr>
<td>3-81.</td>
<td>Function cau_flag_get</td>
</tr>
<tr>
<td>3-82.</td>
<td>CMP Registers</td>
</tr>
</tbody>
</table>
Table 3-83. CMP firmware function ................................................................. 87
Table 3-84. Enum operating_mode_enum .......................................................... 87
Table 3-85. Enum inverting_input_enum ............................................................ 87
Table 3-86. Enum cmp1_plus_input_enum ......................................................... 88
Table 3-87. Enum cmp_hysteresis_enum ............................................................ 88
Table 3-88. Enum cmp_output_enum ............................................................... 88
Table 3-89. Enum cmp_output_inv_enum ......................................................... 88
Table 3-90. Enum blanking_source_enum ......................................................... 89
Table 3-91. Enum cmp_output_state_enum ...................................................... 89
Table 3-92. Function cmp_deinit .................................................................. 89
Table 3-93. Function cmp_mode_init ................................................................ 89
Table 3-94. Function cmp1_plus_selection ..................................................... 90
Table 3-95. Function cmp_output_init ............................................................ 91
Table 3-96. Function cmp_blanking_init .......................................................... 92
Table 3-97. Function cmp_enable .................................................................. 93
Table 3-98. Function cmp_disable .................................................................. 93
Table 3-99. Function cmp_window_enable ...................................................... 94
Table 3-100. Function cmp_window_disable .................................................. 94
Table 3-101. Function cmp_voltage_scaler_enable ....................................... 95
Table 3-102. Function cmp_voltage_scaler_disable ....................................... 95
Table 3-103. Function cmp_scaler_bridge_enable ......................................... 96
Table 3-104. Function cmp_scaler_bridge_disable ......................................... 96
Table 3-105. Function cmp_lock_enable ....................................................... 97
Table 3-106. Function cmp_output_level_get ............................................... 98
Table 3-107. CRC Registers ......................................................................... 98
Table 3-108. CRC firmware function ............................................................ 99
Table 3-109. Function crc_deinit ................................................................ 99
Table 3-110. Function crc_reverse_output_data_enable ............................... 100
Table 3-111. Function crc_reverse_output_data_disable ................................ 100
Table 3-112. Function crc_data_register_reset .............................................. 101
Table 3-113. Function crc_data_register_read .............................................. 101
Table 3-114. Function crc_free_data_register_read ....................................... 102
Table 3-115. Function crc_free_data_register_write ...................................... 102
Table 3-116. Function crc_init_data_register_write ....................................... 103
Table 3-117. Function crc_input_data_reverse_config .................................... 103
Table 3-118. Function crc_polynomial_size_set ............................................ 104
Table 3-119. Function crc_polynomial_set .................................................... 104
Table 3-120. Function crc_single_data_calculate ........................................... 105
Table 3-121. Function crc_block_data_calculate ........................................... 105
Table 3-122. CTC Registers ......................................................................... 107
Table 3-123. CTC firmware function ............................................................. 107
Table 3-124. Function ctc_deinit ................................................................. 108
Table 3-125. Function ctc_counter_enable .................................................... 108
Table 3-126. Function ctc_counter_disable .................................................. 109
Table 3-127. Function ctc irc48m trim value config ......................................................... 109
Table 3-128. Function ctc software ressource pulse generate ............................................. 110
Table 3-129. Function ctc hardware trim mode config ...................................................... 110
Table 3-130. Function ctc ressource polarity config ......................................................... 111
Table 3-131. Function ctc ressource signal select .......................................................... 111
Table 3-132. Function ctc ressource prescaler config ....................................................... 112
Table 3-133. Function ctc clock limit value config ........................................................ 113
Table 3-134. Function ctc counter reload value config .................................................... 113
Table 3-135. Function ctc counter capture value read ...................................................... 114
Table 3-136. Function ctc counter direction read ............................................................ 114
Table 3-137. Function ctc counter reload value read ....................................................... 115
Table 3-138. Function ctc irc48m trim value read ............................................................ 115
Table 3-139. Function ctc flag get ................................................................................. 116
Table 3-140. Function ctc flag clear .............................................................................. 116
Table 3-141. Function ctc interrupt enable ..................................................................... 117
Table 3-142. Function ctc interrupt disable ..................................................................... 118
Table 3-143. Function ctc interrupt flag get .................................................................... 118
Table 3-144. Function ctc interrupt flag clear ................................................................. 119
Table 3-145. DBG Registers ......................................................................................... 120
Table 3-146. DBG firmware function ............................................................................ 120
Table 3-147. Enum dbg periph enum ........................................................................... 120
Table 3-148. Function dbg_deinit ................................................................................ 121
Table 3-149. Function dbg_id_get ................................................................................ 121
Table 3-150. Function dbg low power enable ............................................................... 121
Table 3-151. Function dbg low power disable ............................................................... 122
Table 3-152. Function dbg periph enable ..................................................................... 123
Table 3-153. Function dbg periph disable ..................................................................... 124
Table 3-154. DBG Registers ....................................................................................... 124
Table 3-155. DAC firmware function ............................................................................ 125
Table 3-156. Function dac_deinit ................................................................................ 125
Table 3-157. Function dac_enable .............................................................................. 125
Table 3-158. Function dac_disable .............................................................................. 126
Table 3-159. Function dac dma enable ........................................................................ 127
Table 3-160. Function dac dma disable ....................................................................... 127
Table 3-161. Function dac output buffer enable ............................................................ 128
Table 3-162. Function dac output buffer disable ............................................................. 128
Table 3-163. Function dac output value get ................................................................. 129
Table 3-164. Function dac data set .............................................................................. 129
Table 3-165. Function dac trigger enable ..................................................................... 130
Table 3-166. Function dac trigger disable ................................................................... 131
Table 3-167. Function dac trigger source config ........................................................... 131
Table 3-168. Function dac software trigger enable ....................................................... 132
Table 3-169. Function dac software trigger disable ....................................................... 132
Table 3-170. Function dac wave mode config ............................................................... 133
Table 3-171. Function dac_wave_bit_width_config .......................................................... 133
Table 3-172. Function dac_lfsr_noise_config .............................................................. 134
Table 3-173. Function dac_triangle_noise_config ....................................................... 135
Table 3-174. Function dac_flag_get ............................................................................. 135
Table 3-175. Function dac_flag_clear ......................................................................... 136
Table 3-176. Function dac_interrupt_enable ............................................................... 136
Table 3-177. Function dac_interrupt_disable ............................................................. 137
Table 3-178. Function dac_interrupt_flag_get ............................................................. 137
Table 3-179. Function dac_interrupt_flag_clear ........................................................ 138
Table 3-180. DMA Registers ...................................................................................... 139
Table 3-181. DMAMUX Registers .............................................................................. 139
Table 3-182. DMA firmware function ...................................................................... 139
Table 3-183. DMAMUX firmware function ............................................................... 140
Table 3-184. Structure dma_parameter_struct .......................................................... 141
Table 3-185. Structure dmamux_sync_parameter_struct .......................................... 141
Table 3-186. Structure dmamux_gen_parameter_struct ............................................ 142
Table 3-187. Enum dmamux_interrupt_enum .............................................................. 142
Table 3-188. Enum dmamux_flag_enum ..................................................................... 142
Table 3-189. Enum dmamux_interrupt_flag_enum ..................................................... 143
Table 3-190. Enum dma_channel_enum .................................................................... 144
Table 3-191. Enum dmamux_multiplexer_channel_enum .......................................... 144
Table 3-192. Enum dmamux_generator_channel_enum .............................................. 144
Table 3-193. Function dma_deinit ........................................................................... 145
Table 3-194. Function dma_para_init ...................................................................... 145
Table 3-195. Function dma_init ................................................................................ 146
Table 3-196. Function dma_circulation_enable .......................................................... 147
Table 3-197. Function dma_circulation_disable ....................................................... 147
Table 3-198. Function dma_memory_to_memory_enable .......................................... 148
Table 3-199. Function dma_memory_to_memory_disable ....................................... 148
Table 3-200. Function dma_channel_enable ............................................................. 149
Table 3-201. Function dma_channel_disable ............................................................. 149
Table 3-202. Function dma_periph_address_config ............................................... 150
Table 3-203. Function dma_memory_address_config .............................................. 150
Table 3-204. Function dma_transfer_number_config .............................................. 151
Table 3-205. Function dma_transfer_number_get ..................................................... 151
Table 3-206. Function dma_priority_config .............................................................. 152
Table 3-207. Function dma_memory_width_config .................................................. 152
Table 3-208. Function dma_periph_width_config .................................................... 152
Table 3-209. Function dma_memory_increase_enable .............................................. 153
Table 3-210. Function dma_memory_increase_disable ............................................. 154
Table 3-211. Function dma_periph_increase_enable ................................................. 154
Table 3-212. Function dma_periph_increase_disable .............................................. 155
Table 3-213. Function dma_transfer_direction_config .......................................... 156
Table 3-214. Function dma_flag_get ........................................................................ 157
Table 3-215. Function dma_flag_clear................................................................. 157
Table 3-216. Function dma_interrupt_flag_get .................................................. 158
Table 3-217. Function dma_interrupt_flag_clear .................................................. 159
Table 3-218. Function dma_interrupt_enable ...................................................... 159
Table 3-219. Function dma_interrupt_disable ..................................................... 160
Table 3-220. Function dmamux_sync_struct_para_init........................................... 161
Table 3-221. Function dmamux_synchronization_init ........................................... 161
Table 3-222. Function dmamux_synchronization_enable ....................................... 162
Table 3-223. Function dmamux_synchronization_disable ..................................... 162
Table 3-224. Function dmamux_event_generation_enable ...................................... 163
Table 3-225. Function dmamux_event_generation_disable ..................................... 164
Table 3-226. Function dmamux_gen_struct_para_init .......................................... 164
Table 3-227. Function dmamux_request_generator_init ....................................... 165
Table 3-228. Function dmamux_request_generator_channel_enable ........................ 165
Table 3-229. Function dmamux_request_generator_channel_disable ...................... 166
Table 3-230. Function dmamux_synchronization_polarity_config .......................... 166
Table 3-231. Function dmamux_request_forward_number_config ........................... 167
Table 3-232. Function dmamux_sync_id_config .................................................... 168
Table 3-233. Function dmamux_request_id_config .............................................. 170
Table 3-234. Function dmamux_trigger_polarity_config ...................................... 172
Table 3-235. Function dmamux_request_generate_number_config ......................... 173
Table 3-236. Function dmamux_trigger_id_config ............................................. 173
Table 3-237. Function dmamux_flag_get ............................................................. 175
Table 3-238. Function dmamux_flag_clear ........................................................ 176
Table 3-239. Function dmamux_interrupt_flag_get ............................................ 177
Table 3-240. Function dmamux_interrupt_flag_clear ......................................... 178
Table 3-241. Function dmamux_interrupt_enable ............................................... 179
Table 3-242. Function dmamux_interrupt_disable ............................................. 180
Table 3-243. EXTI Registers ............................................................................. 181
Table 3-244. EXTI firmware function .................................................................. 181
Table 3-245. exti_line_enum ............................................................................. 182
Table 3-246. exti_mode_enum ............................................................................ 183
Table 3-247. exti_trig_type_enum ..................................................................... 183
Table 3-248. Function exti_deinit ..................................................................... 183
Table 3-249. Function exti_init ........................................................................ 183
Table 3-250. Function exti_interrupt_enable .................................................... 184
Table 3-251. Function exti_interrupt_disable .................................................... 185
Table 3-252. Function exti_event_enable ........................................................... 185
Table 3-253. Function exti_event_disable .......................................................... 186
Table 3-254. Function exti_software_interrupt_enable ...................................... 186
Table 3-255. Function exti_software_interrupt_disable ...................................... 187
Table 3-256. Function exti_flag_get .................................................................. 187
Table 3-257. Function exti_flag_clear .............................................................. 188
Table 3-258. Function exti_interrupt_flag_get .................................................... 188
Table 3-259. Function exti_interrupt_flag_clear ................................................................. 189
Table 3-260. FMC Registers .............................................................................................. 189
Table 3-261. FMC firmware function ................................................................................ 190
Table 3-262. fmc_state_enum ......................................................................................... 191
Table 3-263. Function fmc_unlock ................................................................................... 191
Table 3-264. Function fmc_lock ....................................................................................... 192
Table 3-265. Function fmc_slp_unlock ............................................................................. 192
Table 3-266. Function fmc_wscnt_set ............................................................................. 193
Table 3-267. Function fmc_prefetch_enable ................................................................. 193
Table 3-268. Function fmc_prefetch_disable ............................................................... 194
Table 3-269. Function fmc_low_power_enable ............................................................ 194
Table 3-270. Function fmc_low_power_disable ............................................................. 195
Table 3-271. Function fmc_enter_slp_enable ............................................................... 195
Table 3-272. Function fmc_enter_slp_disable ............................................................... 196
Table 3-273. Function fmc_enter_sleep_enable ............................................................ 196
Table 3-274. Function fmc_enter_sleep_disable ............................................................ 197
Table 3-275. Function fmc_page_erase ......................................................................... 197
Table 3-276. Function fmc_mass_erase ......................................................................... 198
Table 3-277. Function fmc_word_program .................................................................... 198
Table 3-278. Function fmc_fast_program ................................................................. 199
Table 3-279. Function ob_unlock ................................................................................... 200
Table 3-280. Function ob_lock ....................................................................................... 201
Table 3-281. Function ob_erase ...................................................................................... 201
Table 3-282. Function ob_write_protection_enable ..................................................... 202
Table 3-283. Function ob_security_protection_config .................................................... 202
Table 3-284. Function ob_user_write ............................................................................ 203
Table 3-285. Function ob_data_program ....................................................................... 204
Table 3-286. Function ob_user_get ................................................................................ 204
Table 3-287. Function ob_data_get ................................................................................ 205
Table 3-288. Function ob_write_protection_get .......................................................... 205
Table 3-289. Function ob_security_protection_flag_get ............................................... 206
Table 3-290. Function fmc_flag_get ............................................................................... 206
Table 3-291. Function fmc_flag_clear .......................................................................... 207
Table 3-292. Function fmc_interrupt_enable .............................................................. 208
Table 3-293. Function fmc_interrupt_disable ............................................................ 208
Table 3-294. Function fmc_interrupt_flag_get ............................................................. 209
Table 3-295. Function fmc_interrupt_flag_clear ......................................................... 209
Table 3-296. FWDGT Registers .................................................................................... 210
Table 3-297. FWDGT firmware function ....................................................................... 211
Table 3-298. Function fwdgt_write_enable ................................................................. 211
Table 3-299. Function fwdgt_write_disable ................................................................. 211
Table 3-300. Function fwdgt_enable ............................................................................. 212
Table 3-301. Function fwdgt_prescaler_value_config .................................................. 212
Table 3-302. Function fwdgt_reload_value_config ...................................................... 213
Table 3-303. Function fwdgt_window_value_config ................................................................. 213
Table 3-304. Function fwdgt_counter_reload ........................................................................... 214
Table 3-305. Function fwdgt_config ......................................................................................... 214
Table 3-306. Function fwdgt_flag_get ......................................................................................... 215
Table 3-307. GPIO Registers ..................................................................................................... 216
Table 3-308. GPIO firmware function ......................................................................................... 216
Table 3-309. Function gpio_deinit ............................................................................................. 217
Table 3-310. Function gpio_mode_set ......................................................................................... 217
Table 3-311. Function gpio_output_options_set ......................................................................... 218
Table 3-312. Function gpio_bit_set .............................................................................................. 219
Table 3-313. Function gpio_bit_reset ......................................................................................... 220
Table 3-314. Function gpio_bit_write ......................................................................................... 220
Table 3-315. Function gpio_port_write ....................................................................................... 221
Table 3-316. Function gpio_input_bit_get .................................................................................... 222
Table 3-317. Function gpio_input_port_get .................................................................................. 222
Table 3-318. Function gpio_output_bit_get ................................................................................... 223
Table 3-319. Function gpio_output_port_get .............................................................................. 223
Table 3-320. Function gpio_af_set ............................................................................................. 224
Table 3-321. Function gpio_pin_lock .......................................................................................... 225
Table 3-322. Function gpio_bit_toggle ....................................................................................... 226
Table 3-323. Function gpio_port_toggle ..................................................................................... 226
Table 3-324. I2C Registers ......................................................................................................... 227
Table 3-325. I2C firmware function ........................................................................................... 227
Table 3-326. i2c_interrupt_flag_enum ....................................................................................... 229
Table 3-327. Function i2c_deinit ............................................................................................... 229
Table 3-328. Function i2c_timing_config .................................................................................... 230
Table 3-329. Function i2c_digital_noise_filter_config .............................................................. 231
Table 3-330. Function i2c_analog_noise_filter_enable .............................................................. 232
Table 3-331. Function i2c_analog_noise_filter_disable ............................................................ 232
Table 3-332. Function i2c_master_clock_config ........................................................................ 233
Table 3-333. Function i2c_master_addressing .......................................................................... 233
Table 3-334. Function i2c_address10_header_enable ............................................................... 234
Table 3-335. Function i2c_address10_header_disable .............................................................. 234
Table 3-336. Function i2c_address10_enable ............................................................................ 235
Table 3-337. Function i2c_address10_disable .......................................................................... 235
Table 3-338. Function i2c_automatic_end_enable ..................................................................... 236
Table 3-339. Function i2c_automatic_end_disable .................................................................... 236
Table 3-340. Function i2c_slave_response_to_gcall_enable ...................................................... 237
Table 3-341. Function i2c_slave_response_to_gcall_disable ...................................................... 238
Table 3-342. Function i2c_stretch_scl_low_enable .................................................................... 238
Table 3-343. Function i2c_stretch_scl_low_disable ................................................................... 239
Table 3-344. Function i2c_address_config ............................................................................... 239
Table 3-345. Function i2c_address_bit_compare_config ............................................................. 240
Table 3-346. Function i2c_address_disable .............................................................................. 241
Table 3-347. Function i2c_second_address_config ................................................. 241
Table 3-348. Function i2c_second_address_disable ............................................. 242
Table 3-349. Function i2c_received_address_get ................................................ 243
Table 3-350. Function i2c_slave_byte_control_enable ....................................... 243
Table 3-351. Function i2c_slave_byte_control_disable ...................................... 244
Table 3-352. Function i2c_nack_enable ............................................................... 244
Table 3-353. Function i2c_nack_disable .............................................................. 245
Table 3-354. Function i2c_wakeup_from_deepsleep_enable ............................... 245
Table 3-355. Function i2c_wakeup_from_deepsleep_disable ................................ 246
Table 3-356. Function i2c_enable ....................................................................... 246
Table 3-357. Function i2c_disable ...................................................................... 247
Table 3-358. Function i2c_start_on_bus ................................................................ 247
Table 3-359. Function i2c_start_on_bus .............................................................. 248
Table 3-360. Function i2c_data_transmit ............................................................. 248
Table 3-361. Function i2c_data_receive .............................................................. 249
Table 3-362. Function i2c_reload_enable ............................................................. 249
Table 3-363. Function i2c_reload_disable ............................................................ 250
Table 3-364. Function i2c_transfer_byte_number_config ................................... 250
Table 3-365. Function i2c_dma_enable ................................................................ 251
Table 3-366. Function i2c_dma_disable .............................................................. 252
Table 3-367. Function i2c_pec_transfer .............................................................. 252
Table 3-368. Function i2c_pec_enable ................................................................. 253
Table 3-369. Function i2c_pec_disable ............................................................... 253
Table 3-370. Function i2c_pec_value_get ............................................................ 254
Table 3-371. Function i2c_smbus_alert_enable .................................................... 254
Table 3-372. Function i2c_smbus_alert_disable .................................................. 255
Table 3-373. Function i2c_smbus_default_addr_enable ...................................... 255
Table 3-374. Function i2c_smbus_default_addr_disable ...................................... 256
Table 3-375. Function i2c_smbus_host_addr_enable ......................................... 256
Table 3-376. Function i2c_smbus_host_addr_disable ......................................... 257
Table 3-377. Function i2c_extented_clock_timeout_enable ............................... 257
Table 3-378. Function i2c_extented_clock_timeout_disable .............................. 258
Table 3-379. Function i2c_clock_timeout_enable ............................................... 258
Table 3-380. Function i2c_clock_timeout_disable ............................................. 259
Table 3-381. Function i2c_bus_timeout_b_config .............................................. 259
Table 3-382. Function i2c_bus_timeout_a_config .............................................. 260
Table 3-383. Function i2c_idle_clock_timeout_config ...................................... 260
Table 3-384. Function i2c_flag_get .................................................................... 261
Table 3-385. Function i2c_flag_clear .................................................................. 262
Table 3-386. Function i2c_interrupt_enable ...................................................... 263
Table 3-387. Function i2c_interrupt_disable ..................................................... 263
Table 3-388. Function i2c_interrupt_flag_get ..................................................... 264
Table 3-389. Function i2c_interrupt_flag_clear ............................................... 265
Table 3-390. LPTIMER Registers ..................................................................... 266
Table 3-391. LPTIMER firmware function
Table 3-392. Structure lptimer_parameter_struct
Table 3-393. Function lptimer_deinit
Table 3-394. Function lptimer_struct_para_init
Table 3-395. Function lptimer_init
Table 3-396. Function lptimer_inputremap
Table 3-397. Function lptimer_register_shadow_enable
Table 3-398. Function lptimer_register_shadow_disable
Table 3-399. Function lptimer_timeout_enable
Table 3-400. Function lptimer_timeout_disable
Table 3-401. Function lptimer_continue_start
Table 3-402. Function lptimer_single_start
Table 3-403. Function lptimer_stop
Table 3-404. Function lptimer_counter_read
Table 3-405. Function lptimer_autoreload_read
Table 3-406. Function lptimer_compare_read
Table 3-407. Function lptimer_autoreload_value_config
Table 3-408. Function lptimer_compare_value_config
Table 3-409. Function lptimer_decodemode0_enable
Table 3-410. Function lptimer_decodemode1_enable
Table 3-411. Function lptimer_decodemode_disable
Table 3-412. Function lptimer_highlevelcounter_enable
Table 3-413. Function lptimer_highlevelcounter_disable
Table 3-414. Function lptimer_flag_get
Table 3-415. Function lptimer_flag_clear
Table 3-416. Function lptimer_interrupt_enable
Table 3-417. Function lptimer_interrupt_disable
Table 3-418. Function lptimer_interrupt_flag_get
Table 3-419. Function lptimer_interrupt_flag_clear
Table 3-420. LPUART Registers
Table 3-421. LPUART firmware function
Table 3-422. Enum lpuart_flag_enum
Table 3-423. Enum lpuart_interrupt_flag_enum
Table 3-424. Enum lpuart_interrupt_enum
Table 3-425. Enum lpuart_invert_enum
Table 3-426. Function lpuart_deinit
Table 3-427. Function lpuart_baudrate_set
Table 3-428. Function lpuart_parity__config
Table 3-429. Function lpuart_word_length_set
Table 3-430. Function lpuart_stop_bit_set
Table 3-431. Function lpuart_enable
Table 3-432. Function lpuart_disable
Table 3-433. Function lpuart_transmit_config
Table 3-434. Function lpuart_receive_config
Table 3-435. Function lpuart_data_first_config ................................................................. 293
Table 3-436. Function lpuart_invert_config ........................................................................ 294
Table 3-437. Function lpuart_overrun_enable ................................................................... 295
Table 3-438. Function lpuart_overrun_disable .................................................................... 295
Table 3-439. Function lpuart_data_transmit ................................................................. 296
Table 3-440. Function lpuart_data_receive ........................................................................ 296
Table 3-441. Function lpuart_command_enable ............................................................... 297
Table 3-442. Function lpuart_address_config .................................................................. 297
Table 3-443. Function lpuart_address_detection_mode_config ....................................... 298
Table 3-444. Function lpuart_mute_mode_enable ......................................................... 298
Table 3-445. Function lpuart_mute_mode_disable ........................................................ 299
Table 3-446. Function lpuart_mute_mode_wakeup_config ............................................. 299
Table 3-447. Function lpuart_halfduplex_enable .......................................................... 300
Table 3-448. Function lpuart_halfduplex_disable .......................................................... 300
Table 3-449. Function lpuart_hardware_flow_rts_config ............................................... 301
Table 3-450. Function lpuart_hardware_flow_cts_config ............................................... 301
Table 3-451. Function lpuart_hardware_flow_coherence_config .................................... 302
Table 3-452. Function lpuart_rs485_driver_enable ....................................................... 303
Table 3-453. Function lpuart_rs485_driver_disable ....................................................... 303
Table 3-454. Function lpuart_driver_assertime_config ................................................. 304
Table 3-455. Function lpuart_driver_deassertime_config ............................................. 304
Table 3-456. Function lpuart_depolarity_config ............................................................. 305
Table 3-457. Function lpuart_dma_receive_config ....................................................... 305
Table 3-458. Function lpuart_dma_transmit_config ...................................................... 306
Table 3-459. Function lpuart_reception_error_dma_disable ......................................... 306
Table 3-460. Function lpuart_reception_error_dma_enable ......................................... 307
Table 3-461. Function lpuart_wakeup_enable ............................................................... 307
Table 3-462. Function lpuart_wakeup_disable .............................................................. 308
Table 3-463. Function lpuart_wakeup_mode_config ..................................................... 308
Table 3-464. Function lpuart_flag_get .............................................................................. 309
Table 3-465. Function lpuart_flag_clear ........................................................................... 310
Table 3-466. Function lpuart_interrupt_enable ............................................................ 311
Table 3-467. Function lpuart_interrupt_disable ............................................................ 311
Table 3-468. Function lpuart_interrupt_flag_get .......................................................... 312
Table 3-469. Function lpuart_interrupt_flag_clear ....................................................... 313
Table 3-470. NVIC Registers ......................................................................................... 314
Table 3-471. SysTick Registers ....................................................................................... 315
Table 3-472. MISC firmware function ............................................................................. 315
Table 3-473. IRQn_Type ................................................................................................. 315
Table 3-474. Function nvic IRQ enable ........................................................................... 317
Table 3-475. Function nvic IRQ disable ......................................................................... 317
Table 3-476. Function nvic_system reset ....................................................................... 318
Table 3-477. Function nvic_vector_table_set ............................................................... 318
Table 3-478. Function system lowpower_set ............................................................... 319
<table>
<thead>
<tr>
<th>Table 3-479. Function system_lowpower_reset</th>
<th>320</th>
</tr>
</thead>
<tbody>
<tr>
<td>Table 3-480. Function systick_clksource_set</td>
<td>320</td>
</tr>
<tr>
<td>Table 3-481. PMU Registers</td>
<td>321</td>
</tr>
<tr>
<td>Table 3-482. PMU firmware function</td>
<td>321</td>
</tr>
<tr>
<td>Table 3-483. Function pmu_deinit</td>
<td>322</td>
</tr>
<tr>
<td>Table 3-484. Function pmu_lvd_select</td>
<td>323</td>
</tr>
<tr>
<td>Table 3-485. Function pmu_lvd_disable</td>
<td>323</td>
</tr>
<tr>
<td>Table 3-486. Function pmu_ldo_output_select</td>
<td>324</td>
</tr>
<tr>
<td>Table 3-487. Function pmu_vc_enable</td>
<td>324</td>
</tr>
<tr>
<td>Table 3-488. Function pmu_vc_disable</td>
<td>325</td>
</tr>
<tr>
<td>Table 3-489. Function pmu_vcr_select</td>
<td>325</td>
</tr>
<tr>
<td>Table 3-490. Function pmu_low_power_enable</td>
<td>326</td>
</tr>
<tr>
<td>Table 3-491. Function pmu_low_power_disable</td>
<td>326</td>
</tr>
<tr>
<td>Table 3-492. Function pmu_to_sleepmode</td>
<td>327</td>
</tr>
<tr>
<td>Table 3-493. Function pmu_to_deepsleepmode</td>
<td>328</td>
</tr>
<tr>
<td>Table 3-494. Function pmu_to_standbymode</td>
<td>328</td>
</tr>
<tr>
<td>Table 3-495. Function pmu_wakeup_pin_enable</td>
<td>329</td>
</tr>
<tr>
<td>Table 3-496. Function pmu_wakeup_pin_disable</td>
<td>330</td>
</tr>
<tr>
<td>Table 3-497. Function pmu_backup_write_enable</td>
<td>330</td>
</tr>
<tr>
<td>Table 3-498. Function pmu_backup_write_disable</td>
<td>331</td>
</tr>
<tr>
<td>Table 3-499. Function pmu_sram_power_config</td>
<td>331</td>
</tr>
<tr>
<td>Table 3-500. Function pmu_core1_power_config</td>
<td>332</td>
</tr>
<tr>
<td>Table 3-501. Function pmu_deepsleep2_retention_enable</td>
<td>332</td>
</tr>
<tr>
<td>Table 3-502. Function pmu_deepsleep2_retention_disable</td>
<td>333</td>
</tr>
<tr>
<td>Table 3-503. Function pmu_deepsleep2_sram_power_config</td>
<td>333</td>
</tr>
<tr>
<td>Table 3-504. Function pmu_deepsleep_wait_time_config</td>
<td>334</td>
</tr>
<tr>
<td>Table 3-505. Function pmu_wakeuptime_core1_software_enable</td>
<td>334</td>
</tr>
<tr>
<td>Table 3-506. Function pmu_wakeuptime_core1_software_disable</td>
<td>335</td>
</tr>
<tr>
<td>Table 3-507. Function pmu_wakeuptime_sram_software_enable</td>
<td>335</td>
</tr>
<tr>
<td>Table 3-508. Function pmu_wakeuptime_sram_software_disable</td>
<td>336</td>
</tr>
<tr>
<td>Table 3-509. Function pmu_wakeuptime_deepsleep2_software_enable</td>
<td>336</td>
</tr>
<tr>
<td>Table 3-510. Function pmu_wakeuptime_deepsleep2_software_disable</td>
<td>337</td>
</tr>
<tr>
<td>Table 3-511. Function pmu_flag_get</td>
<td>337</td>
</tr>
<tr>
<td>Table 3-512. Function pmu_flag_clear</td>
<td>338</td>
</tr>
<tr>
<td>Table 3-513. RCU Registers</td>
<td>339</td>
</tr>
<tr>
<td>Table 3-514. RCU firmware function</td>
<td>339</td>
</tr>
<tr>
<td>Table 3-515. Enum rcu_periph_enum</td>
<td>341</td>
</tr>
<tr>
<td>Table 3-516. Enum rcu_periph_sleep_enum</td>
<td>342</td>
</tr>
<tr>
<td>Table 3-517. Enum rcu_periph_reset_enum</td>
<td>342</td>
</tr>
<tr>
<td>Table 3-518. Enum rcu_flag_enum</td>
<td>343</td>
</tr>
<tr>
<td>Table 3-519. Enum rcu_int_flag_enum</td>
<td>343</td>
</tr>
<tr>
<td>Table 3-520. Enum rcu_int_flag_clear_enum</td>
<td>344</td>
</tr>
<tr>
<td>Table 3-521. Enum rcu_int_enum</td>
<td>345</td>
</tr>
<tr>
<td>Table 3-522. Enum rcu_osci_type_enum</td>
<td>345</td>
</tr>
</tbody>
</table>
Table 3-523. Enum rcu_clock_freq_enum................................................................. 345
Table 3-524. Enum uart_idx_enum...................................................................... 346
Table 3-525. Enum i2c_idx_enum...................................................................... 346
Table 3-526. Function rcu_deinit................................................................. 346
Table 3-527. Function rcu_periph_clock_enable........................................... 346
Table 3-528. Function rcu_periph_clock_disable......................................... 347
Table 3-529. Function rcu_periph_clock_sleep_enable................................. 348
Table 3-530. Function rcu_periph_clock_sleep_disable............................... 349
Table 3-531. Function rcu_periph_reset_enable........................................... 350
Table 3-532. Function rcu_periph_reset_disable.......................................... 351
Table 3-533. Function rcu_bkp_reset_enable......................................................... 352
Table 3-534. Function rcu_bkp_reset_disable............................................. 352
Table 3-535. Function rcu_system_clock_source_config.............................. 353
Table 3-536. Function rcu_system_clock_source_get.................................... 353
Table 3-537. Function rcu_abh_clock_config............................................... 354
Table 3-538. Function rcu_abp1_clock_config............................................. 354
Table 3-539. Function rcu_abp2_clock_config............................................. 355
Table 3-540. Function rcu_adc_clock_config............................................... 355
Table 3-541. Function rcu_ckout_config....................................................... 357
Table 3-542. Function rcu_pll_config......................................................... 358
Table 3-543. Function rcu_usart_clock_config........................................... 358
Table 3-544. Function rcu_i2c_clock_config.............................................. 359
Table 3-545. Function rcu_lptimer_clock_config....................................... 360
Table 3-546. Function rcu_lpuart_clock_config........................................ 360
Table 3-547. Function rcu irc16mdiv_clock_config................................... 361
Table 3-548. Function rcu_usbd_clock_config........................................... 362
Table 3-549. Function rcu rtc_clock_config.............................................. 362
Table 3-550. Function rcu_pll_source_ck_pdiv_config............................... 363
Table 3-551. Function rcu_xtal_drive_capability_config............................ 363
Table 3-552. Function rcu lp ldo_config..................................................... 364
Table 3-553. Function rcu lp bandgap_config............................................. 364
Table 3-554. Function rcu_flag_get............................................................. 365
Table 3-555. Function rcu_all_reset_flag_clear......................................... 366
Table 3-556. Function rcu_interrupt_flag_get........................................... 367
Table 3-557. Function rcu_interrupt_flag_clear......................................... 367
Table 3-558. Function rcu_interrupt_enable............................................. 368
Table 3-559. Function rcu_interrupt_disable............................................ 369
Table 3-560. Function rcu_osci_stab_wait....................................................... 370
Table 3-561. Function rcu_osci_on.............................................................. 370
Table 3-562. Function rcu_osci_off............................................................. 371
Table 3-563. Function rcu_osci_bypass_mode_enable................................... 372
Table 3-564. Function rcu_osci_bypass_mode_disable................................ 372
Table 3-565. Function rcu_hxtal_clock_monitor_enable............................. 373
Table 3-566. Function rcu_hxtal_clock_monitor_disable............................ 373
<table>
<thead>
<tr>
<th>Table</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>3-567</td>
<td>Function rtc_lxtal_clock_monitor_enable</td>
<td>374</td>
</tr>
<tr>
<td>3-568</td>
<td>Function rtc_lxtal_clock_monitor_disable</td>
<td>374</td>
</tr>
<tr>
<td>3-569</td>
<td>Function rtc_icrc16m_adjust_value_set</td>
<td>375</td>
</tr>
<tr>
<td>3-570</td>
<td>Function rtc_voltage_key_unlock</td>
<td>375</td>
</tr>
<tr>
<td>3-571</td>
<td>Function rtc_clock_freq_get</td>
<td>376</td>
</tr>
<tr>
<td>3-572</td>
<td>RTC Registers</td>
<td>377</td>
</tr>
<tr>
<td>3-573</td>
<td>RTC firmware function</td>
<td>377</td>
</tr>
<tr>
<td>3-574</td>
<td>rtc_parameter_struct</td>
<td>378</td>
</tr>
<tr>
<td>3-575</td>
<td>rtc_alarm_struct</td>
<td>379</td>
</tr>
<tr>
<td>3-576</td>
<td>rtc_timestamp_struct</td>
<td>379</td>
</tr>
<tr>
<td>3-577</td>
<td>rtc tamper_struct</td>
<td>379</td>
</tr>
<tr>
<td>3-578</td>
<td>Function rtc_deinit</td>
<td>380</td>
</tr>
<tr>
<td>3-579</td>
<td>Function rtc_init</td>
<td>380</td>
</tr>
<tr>
<td>3-580</td>
<td>Function rtc_init_mode_enter</td>
<td>381</td>
</tr>
<tr>
<td>3-581</td>
<td>Function rtc_init_mode_exit</td>
<td>382</td>
</tr>
<tr>
<td>3-582</td>
<td>Function rtc_register_sync_wait</td>
<td>382</td>
</tr>
<tr>
<td>3-583</td>
<td>Function rtc_current_time_get</td>
<td>383</td>
</tr>
<tr>
<td>3-584</td>
<td>Function rtc_subsecond_get</td>
<td>383</td>
</tr>
<tr>
<td>3-585</td>
<td>Function rtc_alarm_config</td>
<td>384</td>
</tr>
<tr>
<td>3-586</td>
<td>Function rtc_alarm_subsecond_config</td>
<td>384</td>
</tr>
<tr>
<td>3-587</td>
<td>Function rtc_alarm_enable</td>
<td>386</td>
</tr>
<tr>
<td>3-588</td>
<td>Function rtc_alarm_disable</td>
<td>386</td>
</tr>
<tr>
<td>3-589</td>
<td>Function rtc_alarm_get</td>
<td>387</td>
</tr>
<tr>
<td>3-590</td>
<td>Function rtc_alarm_subsecond_get</td>
<td>387</td>
</tr>
<tr>
<td>3-591</td>
<td>Function rtc_timestamp_enable</td>
<td>388</td>
</tr>
<tr>
<td>3-592</td>
<td>Function rtc_timestamp_disable</td>
<td>388</td>
</tr>
<tr>
<td>3-593</td>
<td>Function rtc_timestamp_internalevent_config</td>
<td>389</td>
</tr>
<tr>
<td>3-594</td>
<td>Function rtc_timestamp_get</td>
<td>389</td>
</tr>
<tr>
<td>3-595</td>
<td>Function rtc_timestamp_subsecond_get</td>
<td>390</td>
</tr>
<tr>
<td>3-596</td>
<td>Function rtc_timestamp_enable</td>
<td>390</td>
</tr>
<tr>
<td>3-597</td>
<td>Function rtc tamper_disable</td>
<td>391</td>
</tr>
<tr>
<td>3-598</td>
<td>Function rtc tamper_mask</td>
<td>391</td>
</tr>
<tr>
<td>3-599</td>
<td>Function rtc tamper_without_bkp_reset</td>
<td>392</td>
</tr>
<tr>
<td>3-600</td>
<td>Function rtc_output_pin_select</td>
<td>393</td>
</tr>
<tr>
<td>3-601</td>
<td>Function rtc_alarm_output_config</td>
<td>393</td>
</tr>
<tr>
<td>3-602</td>
<td>rtc_calibration_output_config</td>
<td>394</td>
</tr>
<tr>
<td>3-603</td>
<td>rtc_hour_adjust</td>
<td>395</td>
</tr>
<tr>
<td>3-604</td>
<td>rtc_second_adjust</td>
<td>395</td>
</tr>
<tr>
<td>3-605</td>
<td>rtc_bypass_shadow_enable</td>
<td>396</td>
</tr>
<tr>
<td>3-606</td>
<td>rtc_bypass_shadow_disable</td>
<td>396</td>
</tr>
<tr>
<td>3-607</td>
<td>rtc_refclock_detection_enable</td>
<td>397</td>
</tr>
<tr>
<td>3-608</td>
<td>rtc_refclock_detection_disable</td>
<td>397</td>
</tr>
<tr>
<td>3-609</td>
<td>Function rtc_wakeup_enable</td>
<td>398</td>
</tr>
<tr>
<td>3-610</td>
<td>Function rtc_wakeup_disable</td>
<td>398</td>
</tr>
<tr>
<td>Table</td>
<td>Function/Description</td>
<td>Page</td>
</tr>
<tr>
<td>-------</td>
<td>-------------------------------------------------------------------------------------</td>
<td>------</td>
</tr>
<tr>
<td>3-611</td>
<td>Function rtc_wakeup_clock_set</td>
<td>399</td>
</tr>
<tr>
<td>3-612</td>
<td>Function rtc_wakeup_timer_set</td>
<td>399</td>
</tr>
<tr>
<td>3-613</td>
<td>Function rtc_wakeup_timer_get</td>
<td>400</td>
</tr>
<tr>
<td>3-614</td>
<td>rtc_smooth_calibration_config</td>
<td>400</td>
</tr>
<tr>
<td>3-615</td>
<td>Function rtc_interrupt_enable</td>
<td>401</td>
</tr>
<tr>
<td>3-616</td>
<td>Function rtc_interrupt_disable</td>
<td>402</td>
</tr>
<tr>
<td>3-617</td>
<td>Function rtc_flag_get</td>
<td>403</td>
</tr>
<tr>
<td>3-618</td>
<td>Function rtc_flag_clear</td>
<td>404</td>
</tr>
<tr>
<td>3-619</td>
<td>SLCD Registers</td>
<td>404</td>
</tr>
<tr>
<td>3-620</td>
<td>SLCD firmware function</td>
<td>405</td>
</tr>
<tr>
<td>3-621</td>
<td>Enum slcd_data_register_enum</td>
<td>406</td>
</tr>
<tr>
<td>3-622</td>
<td>Function slcd_deinit</td>
<td>406</td>
</tr>
<tr>
<td>3-623</td>
<td>Function slcd_enable</td>
<td>406</td>
</tr>
<tr>
<td>3-624</td>
<td>Function slcd_disable</td>
<td>407</td>
</tr>
<tr>
<td>3-625</td>
<td>Function slcd_init</td>
<td>407</td>
</tr>
<tr>
<td>3-626</td>
<td>Function slcd_enhance_mode_enable</td>
<td>409</td>
</tr>
<tr>
<td>3-627</td>
<td>Function slcd_enhance_mode_disable</td>
<td>410</td>
</tr>
<tr>
<td>3-628</td>
<td>Function slcd_bias_voltage_select</td>
<td>410</td>
</tr>
<tr>
<td>3-629</td>
<td>Function slcd_duty_select</td>
<td>411</td>
</tr>
<tr>
<td>3-630</td>
<td>Function slcd_clock_config</td>
<td>411</td>
</tr>
<tr>
<td>3-631</td>
<td>Function slcd_blink_mode_config</td>
<td>413</td>
</tr>
<tr>
<td>3-632</td>
<td>Function slcd_contrast_ratio_config</td>
<td>414</td>
</tr>
<tr>
<td>3-633</td>
<td>Function slcd_dead_time_config</td>
<td>415</td>
</tr>
<tr>
<td>3-634</td>
<td>Function slcd_pulse_on_duration_config</td>
<td>416</td>
</tr>
<tr>
<td>3-635</td>
<td>Function slcd_com_seg_remap</td>
<td>416</td>
</tr>
<tr>
<td>3-636</td>
<td>Function slcd_voltage_source_select</td>
<td>417</td>
</tr>
<tr>
<td>3-637</td>
<td>Function slcd_high_drive_config</td>
<td>417</td>
</tr>
<tr>
<td>3-638</td>
<td>Function slcd_high_drive_config</td>
<td>418</td>
</tr>
<tr>
<td>3-639</td>
<td>Function slcd_data_update_request</td>
<td>418</td>
</tr>
<tr>
<td>3-640</td>
<td>Function slcd_flag_get</td>
<td>419</td>
</tr>
<tr>
<td>3-641</td>
<td>Function slcd_flag_clear</td>
<td>420</td>
</tr>
<tr>
<td>3-642</td>
<td>Function slcd_interrupt_enable</td>
<td>420</td>
</tr>
<tr>
<td>3-643</td>
<td>Function slcd_interrupt_disable</td>
<td>421</td>
</tr>
<tr>
<td>3-644</td>
<td>Function slcd_interrupt_flag_get</td>
<td>421</td>
</tr>
<tr>
<td>3-645</td>
<td>Function slcd_interrupt_flag_clear</td>
<td>422</td>
</tr>
<tr>
<td>3-646</td>
<td>Function slcd_interrupt_flag_clear</td>
<td>422</td>
</tr>
<tr>
<td>3-647</td>
<td>Function slcd_interrupt_flag_clear</td>
<td>422</td>
</tr>
<tr>
<td>3-648</td>
<td>Function slcd_interrupt_flag_clear</td>
<td>422</td>
</tr>
<tr>
<td>3-649</td>
<td>Function slcd_interrupt_flag_clear</td>
<td>422</td>
</tr>
<tr>
<td>3-650</td>
<td>Function slcd_interrupt_flag_clear</td>
<td>422</td>
</tr>
<tr>
<td>3-651</td>
<td>Function slcd_interrupt_flag_clear</td>
<td>422</td>
</tr>
<tr>
<td>3-652</td>
<td>Function slcd_interrupt_flag_clear</td>
<td>422</td>
</tr>
<tr>
<td>3-653</td>
<td>Function slcd_interrupt_flag_clear</td>
<td>422</td>
</tr>
<tr>
<td>3-654</td>
<td>Function slcd_interrupt_flag_clear</td>
<td>422</td>
</tr>
<tr>
<td>3-655</td>
<td>Function slcd_interrupt_flag_clear</td>
<td>422</td>
</tr>
<tr>
<td>3-656</td>
<td>Function slcd_interrupt_flag_clear</td>
<td>422</td>
</tr>
<tr>
<td>3-657</td>
<td>Function slcd_interrupt_flag_clear</td>
<td>422</td>
</tr>
<tr>
<td>3-658</td>
<td>Function slcd_interrupt_flag_clear</td>
<td>422</td>
</tr>
<tr>
<td>3-659</td>
<td>Function slcd_interrupt_flag_clear</td>
<td>422</td>
</tr>
<tr>
<td>3-660</td>
<td>Function slcd_interrupt_flag_clear</td>
<td>422</td>
</tr>
<tr>
<td>3-661</td>
<td>Function slcd_interrupt_flag_clear</td>
<td>422</td>
</tr>
<tr>
<td>3-662</td>
<td>Function slcd_interrupt_flag_clear</td>
<td>422</td>
</tr>
<tr>
<td>3-663</td>
<td>Function slcd_interrupt_flag_clear</td>
<td>422</td>
</tr>
<tr>
<td>3-664</td>
<td>Function slcd_interrupt_flag_clear</td>
<td>422</td>
</tr>
<tr>
<td>3-665</td>
<td>Function slcd_interrupt_flag_clear</td>
<td>422</td>
</tr>
<tr>
<td>3-666</td>
<td>Function slcd_interrupt_flag_clear</td>
<td>422</td>
</tr>
<tr>
<td>3-667</td>
<td>Function slcd_interrupt_flag_clear</td>
<td>422</td>
</tr>
<tr>
<td>3-668</td>
<td>Function slcd_interrupt_flag_clear</td>
<td>422</td>
</tr>
<tr>
<td>3-669</td>
<td>Function slcd_interrupt_flag_clear</td>
<td>422</td>
</tr>
<tr>
<td>3-670</td>
<td>Function slcd_interrupt_flag_clear</td>
<td>422</td>
</tr>
<tr>
<td>3-671</td>
<td>Function slcd_interrupt_flag_clear</td>
<td>422</td>
</tr>
<tr>
<td>3-672</td>
<td>Function slcd_interrupt_flag_clear</td>
<td>422</td>
</tr>
<tr>
<td>3-673</td>
<td>Function slcd_interrupt_flag_clear</td>
<td>422</td>
</tr>
<tr>
<td>3-674</td>
<td>Function slcd_interrupt_flag_clear</td>
<td>422</td>
</tr>
<tr>
<td>3-675</td>
<td>Function slcd_interrupt_flag_clear</td>
<td>422</td>
</tr>
<tr>
<td>3-676</td>
<td>Function slcd_interrupt_flag_clear</td>
<td>422</td>
</tr>
<tr>
<td>3-677</td>
<td>Function slcd_interrupt_flag_clear</td>
<td>422</td>
</tr>
<tr>
<td>3-678</td>
<td>Function slcd_interrupt_flag_clear</td>
<td>422</td>
</tr>
<tr>
<td>3-679</td>
<td>Function slcd_interrupt_flag_clear</td>
<td>422</td>
</tr>
<tr>
<td>3-680</td>
<td>Function slcd_interrupt_flag_clear</td>
<td>422</td>
</tr>
<tr>
<td>3-681</td>
<td>Function slcd_interrupt_flag_clear</td>
<td>422</td>
</tr>
<tr>
<td>3-682</td>
<td>Function slcd_interrupt_flag_clear</td>
<td>422</td>
</tr>
<tr>
<td>3-683</td>
<td>Function slcd_interrupt_flag_clear</td>
<td>422</td>
</tr>
<tr>
<td>3-684</td>
<td>Function slcd_interrupt_flag_clear</td>
<td>422</td>
</tr>
<tr>
<td>3-685</td>
<td>SPI/I2S Registers</td>
<td>422</td>
</tr>
<tr>
<td>3-686</td>
<td>SPI/I2S firmware function</td>
<td>423</td>
</tr>
<tr>
<td>3-687</td>
<td>spi_parameter_struct</td>
<td>424</td>
</tr>
<tr>
<td>3-688</td>
<td>Function spi_i2s_deinit</td>
<td>424</td>
</tr>
<tr>
<td>3-689</td>
<td>Function spi_struct_para_init</td>
<td>425</td>
</tr>
<tr>
<td>3-690</td>
<td>Function spi_init</td>
<td>425</td>
</tr>
<tr>
<td>3-691</td>
<td>Function spi_enable</td>
<td>426</td>
</tr>
<tr>
<td>3-692</td>
<td>Function spi_disable</td>
<td>427</td>
</tr>
<tr>
<td>3-693</td>
<td>Function i2s_init</td>
<td>427</td>
</tr>
</tbody>
</table>
Table 3-694. Function i2s_psc_config ................................................................. 428
Table 3-695. Function i2s_enable ..................................................................... 430
Table 3-696. Function i2s_disable ................................................................. 430
Table 3-697. Function spi_nss_output_enable .............................................. 431
Table 3-698. Function spi_nss_output_disable ............................................ 431
Table 3-699. Function spi_nss_internal_high ............................................... 432
Table 3-700. Function spi_nss_internal_low ............................................... 432
Table 3-701. Function spi_dma_enable ....................................................... 433
Table 3-702. Function spi_dma_disable ....................................................... 433
Table 3-703. Function spi_i2s_data_frame_format_config ............................ 434
Table 3-704. Function spi_i2s_data_transmit ............................................. 434
Table 3-705. Function spi_i2s_data_receive .............................................. 435
Table 3-706. Function spi_bidirectional_transfer_config ............................. 436
Table 3-707. Function spi_crc_polynomial_set .......................................... 436
Table 3-708. Function spi_crc_polynomial_get ......................................... 437
Table 3-709. Function spi_crc_on .................................................................. 437
Table 3-710. Function spi_crc_off ............................................................... 438
Table 3-711. Function spi_crc_next ............................................................. 438
Table 3-712. Function spi_crc_get ............................................................... 439
Table 3-713. Function spi_i2s_interrupt_enable .......................................... 440
Table 3-714. Function spi_i2s_interrupt_disable ....................................... 440
Table 3-715. Function spi_i2s_interrupt_flag_get ...................................... 441
Table 3-716. Function spi_i2s_flag_get ........................................................ 442
Table 3-717. Function spi_crc_error_clear ................................................ 443
Table 3-718. Function qsqi_enable ............................................................... 443
Table 3-719. Function qsqi_enable ............................................................... 444
Table 3-720. Function qsqi_write_enable ..................................................... 444
Table 3-721. Function qsqi_read_enable ...................................................... 445
Table 3-722. Function qsqi_io23_output_enable ......................................... 445
Table 3-723. Function qsqi_io23_output_disable ....................................... 446
Table 3-693. SYSCFG Registers .................................................................. 446
Table 3-694. SYSCFG firmware function .................................................... 447
Table 3-695. Function syscfg_deinit ........................................................... 447
Table 3-696. Function syscfg_exti_line_config .......................................... 447
Table 3-697. Function syscfg_pin_remap_enable ....................................... 448
Table 3-698. Function syscfg_pin_remap_disable ...................................... 448
Table 3-699. Function syscfg_high_current_enable ................................... 449
Table 3-700. Function syscfg_high_current_disable .................................. 450
Table 3-701. Function irq_latency_set ....................................................... 450
Table 3-702. Function syscfg_bootmode_get .............................................. 451
Table 3-703. TIMERx Registers ................................................................... 452
Table 3-704. TIMERx firmware function .................................................... 453
Table 3-705. Structure timer_parameter_struct ....................................... 454
Table 3-706. Structure timer_oc_parameter_struct .................................... 455
Table 3-707. Structure timer_ic_parameter_struct .......................................................... 455
Table 3-708. Function timer_deinit ............................................................................. 455
Table 3-709. Function timer_struct_para_init ............................................................. 456
Table 3-710. Function timer_init ................................................................................ 456
Table 3-711. Function timer_enable .......................................................................... 457
Table 3-712. Function timer_disable ....................................................................... 458
Table 3-713. Function timer_auto_reload_shadow_enable ....................................... 458
Table 3-714. Function timer_auto_reload_shadow_disable ....................................... 459
Table 3-715. Function timer_update_event_enable ................................................... 459
Table 3-716. Function timer_update_event_disable .................................................. 460
Table 3-717. Function timer_counter_alignment ...................................................... 460
Table 3-718. Function timer_counter_up_direction ................................................... 461
Table 3-719. timer_counter_down_direction ............................................................ 462
Table 3-720. Function timer_prescaler_config ........................................................... 462
Table 3-721. Function timer_autoreloader_value_config .......................................... 463
Table 3-722. Function timer_counter_value_config .................................................. 463
Table 3-723. Function timer_counter_read ............................................................... 464
Table 3-724. Function timer_prescaler_read ............................................................. 465
Table 3-725. Function timer_single_pulse_mode_config                              465
Table 3-726. Function timer_update_source_config ................................................... 466
Table 3-727. Function timer_dma_enable ................................................................ 467
Table 3-728. Function timer_dma_disable ................................................................ 467
Table 3-729. Function timer_channel_dma_request_source_select ................................ 468
Table 3-730. Function timer_dma_transfer_config .................................................... 469
Table 3-731. Function timer_event_software_generate ............................................. 470
Table 3-732. Function timer_channel_output_struct_para_init .................................. 471
Table 3-733. Function timer_channel_output_config ................................................ 472
Table 3-734. Function timer_channel_output_mode_config ....................................... 472
Table 3-735. Function timer_channel_output_pulse_value_config ............................. 473
Table 3-736. Function timer_channel_output_shadow_config .................................... 474
Table 3-737. Function timer_channel_output_fast_config ....................................... 475
Table 3-738. Function timer_channel_output_clear_config ...................................... 476
Table 3-739. Function timer_channel_output_polarity_config .................................... 477
Table 3-740. Function timer_channel_output_state_config ...................................... 477
Table 3-741. Function timer_channel_input_struct_para_init .................................... 478
Table 3-742. Function timer_input_capture_config ................................................... 479
Table 3-743. Function timer_channel_input_capture_prescaler_config ..................... 480
Table 3-744. Function timer_channel_capture_value_register_read ........................... 481
Table 3-745. Function timer_input_pwm_capture_config .......................................... 481
Table 3-746. Function timer_hall_mode_config ......................................................... 482
Table 3-747. Function timer_input_trigger_source_select ....................................... 483
Table 3-748. Function timer_master_output_trigger_source_select ........................... 484
Table 3-749. Function timer_slave_mode_select ...................................................... 485
Table 3-750. Function timer_master_slave_mode_config .......................................... 486
| Table 3-751. Function timer_external_trigger_config | 486 |
| Table 3-752. Function timer_quadrature_decoder_mode_config | 487 |
| Table 3-753. Function timer_internal_trigger_as_external_clock_config | 488 |
| Table 3-754. Function timer_external_trigger_as_external_clock_config | 489 |
| Table 3-755. Function timer_external_clock_mode0_config | 490 |
| Table 3-756. Function timer_external_clock_mode1_config | 491 |
| Table 3-757. Function timer_external_clock_mode1_disable | 492 |
| Table 3-758. Function timer_channel_remap_config | 493 |
| Table 3-759. Function timer_write_chxval_register_config | 494 |
| Table 3-760. Function timer_flag_get | 494 |
| Table 3-761. Function timer_flag_clear | 495 |
| Table 3-762. Function timer_interrupt_enable | 496 |
| Table 3-763. Function timer_interrupt_disable | 496 |
| Table 3-764. Function timer_interrupt_flag_get | 497 |
| Table 3-765. Function timer_interrupt_flag_clear | 498 |
| Table 3-766 TRNG Registers | 499 |
| Table 3-767. TRNG firmware function | 499 |
| Table 3-768. Enum trng_flag_enum | 499 |
| Table 3-769. Enum trng_int_flag_enum | 500 |
| Table 3-770. Function trng_deinit | 500 |
| Table 3-771. Function trng_enable | 500 |
| Table 3-772 Function trng_disable | 501 |
| Table 3-773 Function trng_get_true_random_data | 501 |
| Table 3-774 trng_flag_get | 502 |
| Table 3-775 trng_interrupt_enable | 502 |
| Table 3-776 trng_interrupt_disable | 503 |
| Table 3-777 trng_interrupt_flag_get | 503 |
| Table 3-778 trng_interrupt_flag_clear | 504 |
| Table 3-779. USART Registers | 504 |
| Table 3-780. USART firmware function | 505 |
| Table 3-781. Enum usart_flag_enum | 507 |
| Table 3-782. Enum usart_interrupt_flag_enum | 508 |
| Table 3-783. Enum usart_interrupt_enum | 508 |
| Table 3-784. Enum usart_invert_enum | 509 |
| Table 3-785. Function usart_deinit | 509 |
| Table 3-786. Function usart_baudrate_set | 510 |
| Table 3-787. Function usart_parity_config | 510 |
| Table 3-788. Function usart_word_length_set | 511 |
| Table 3-789. Function usart_stop_bit_set | 511 |
| Table 3-790. Function usart_enable | 512 |
| Table 3-791. Function usart_disable | 513 |
| Table 3-792. Function usart_transmit_config | 513 |
| Table 3-793. Function usart_receive_config | 514 |
| Table 3-794. Function usart_data_first_config | 514 |
Table 3-795. Function usart_invert_config ................................................................. 515
Table 3-796. Function usart_overrun_enable ............................................................ 516
Table 3-797. Function usart_overrun_disable ............................................................ 517
Table 3-798. Function usart_oversample_config ....................................................... 517
Table 3-799. Function usart_sample_bit_config .......................................................... 518
Table 3-800. Function usart_receiver_timeout_enable .............................................. 518
Table 3-801. Function usart_receiver_timeout_disable .......................................... 519
Table 3-802. Function usart_receiver_timeout_threshold_config ................................ 519
Table 3-803. Function usart_data_transmit ................................................................. 520
Table 3-804. Function usart_data_receive ................................................................. 521
Table 3-805. Function usart_command_enable ........................................................... 521
Table 3-806. Function usart_autobaud_detection_enable ....................................... 522
Table 3-807. Function usart_autobaud_detection_disable ..................................... 522
Table 3-808. Function usart_autobaud_detection_mode_config ............................... 523
Table 3-809. Function usart_address_config ............................................................ 524
Table 3-810. Function usart_address_detection_mode_config .................................. 524
Table 3-811. Function usart_mute_mode_enable ....................................................... 525
Table 3-812. Function usart_mute_mode_disable ....................................................... 525
Table 3-813. Function usart_mute_mode_wakeup_config .......................................... 526
Table 3-814. Function usart_lin_mode_enable ......................................................... 527
Table 3-815. Function usart_lin_mode_disable ......................................................... 527
Table 3-816. Function usart_lin_break_detection_length_config ......................... 528
Table 3-817. Function usart_halfduplex_enable ...................................................... 528
Table 3-818. Function usart_halfduplex_disable ...................................................... 529
Table 3-819. Function usart_clock_enable ............................................................... 529
Table 3-820. Function usart_clock_disable ............................................................... 530
Table 3-821. Function usart_synchronous_clock_config ......................................... 530
Table 3-822. Function usart_guard_time_config ....................................................... 531
Table 3-823. Function usart_smartcard_mode_enable ............................................. 532
Table 3-824. Function usart_smartcard_mode_disable ............................................ 532
Table 3-825. Function usart_smartcard_mode_nack_enable ..................................... 533
Table 3-826. Function usart_smartcard_mode_nack_disable .................................... 533
Table 3-827. Function usart_smartcard_mode_early_nack_enable ............................. 534
Table 3-828. Function usart_smartcard_mode_early_nack_disable ............................ 534
Table 3-829. Function usart_smartcard_autoretry_config ....................................... 535
Table 3-830. Function usart_block_length_config .................................................... 535
Table 3-831. Function usart_irda_mode_enable ....................................................... 536
Table 3-832. Function usart_irda_mode_disable ....................................................... 536
Table 3-833. Function usart_prescaler_config .......................................................... 537
Table 3-834. Function usart_irda_lowpower_config .............................................. 537
Table 3-835. Function usart_hardware_flow_rts_config .......................................... 538
Table 3-836. Function usart_hardware_flow_cts_config .......................................... 539
Table 3-837. Function usart_hardware_flow_coherence_config .................................. 539
Table 3-838. Function usart_rs485_driver_enable .................................................... 540
Table 3-839. Function uart_rs485_driver_disable .......................................................... 540
Table 3-840. Function uart_driver_asserttime_config ............................................... 541
Table 3-841. Function uart_driver_deasserttime_config ......................................... 542
Table 3-842. Function uart_depolarity_config .............................................................. 542
Table 3-843. Function uart_dma_receive_config ........................................................ 543
Table 3-844. Function uart_dma_transmit_config ....................................................... 543
Table 3-845. Function uart Reception_error_dma_disable ....................................... 544
Table 3-846. Function uart Reception_error_dma_enable ......................................... 545
Table 3-847. Function uart_wakeup_enable ................................................................. 545
Table 3-848. Function uart_wakeup_disable ................................................................. 546
Table 3-849. Function uart_wakeup_mode_config ....................................................... 546
Table 3-850. Function uart_receive_fifo_enable ......................................................... 547
Table 3-851. Function uart_receive_fifo_disable ......................................................... 547
Table 3-852. Function uart_receive_fifo_counter_number ....................................... 548
Table 3-853. Function uart_flag_get ............................................................................. 548
Table 3-854. Function uart_flag_clear .......................................................................... 550
Table 3-855. Function uart_interrupt_enable .............................................................. 551
Table 3-856. Function uart_interrupt_disable ............................................................ 552
Table 3-857. Function uart_interrupt_flag_get ........................................................... 553
Table 3-858. Function uart_interrupt_flag_clear ......................................................... 554
Table 3-859. WWDGT Registers .................................................................................. 555
Table 3-860. VREF firmware function ......................................................................... 556
Table 3-861. Function vref_deinit .............................................................................. 556
Table 3-862. Function vref_enable .............................................................................. 556
Table 3-863. Function vref_disable .............................................................................. 557
Table 3-864. Function vref_high_impedance_mode_enable ....................................... 557
Table 3-865. Function vref_high_impedance_mode_disable ....................................... 558
Table 3-866. Function vref_status_get ........................................................................ 558
Table 3-867. Function vref_calib_value_set ................................................................. 559
Table 3-868. Function vref_calib_value_get ................................................................. 559
Table 3-869. WWDGT Registers .................................................................................. 560
Table 3-870. WWDGT firmware function .................................................................... 560
Table 3-871. Function wwdgt_deinit .......................................................................... 561
Table 3-872. Function wwdgt_enable .......................................................................... 561
Table 3-873. Function wwdgt_counter_update .......................................................... 562
Table 3-874. Function wwdgt_config ......................................................................... 562
Table 3-875. Function wwdgt_interrupt_enable ........................................................ 563
Table 3-876. Function wwdgt_flag_get ........................................................................ 564
Table 3-877. Function wwdgt_flag_clear .................................................................... 564
Table 4-1. Revision history .......................................................................................... 566
1. **Introduction**

This manual introduces firmware library of GD32L23x devices which are 32-bit microcontrollers based on the ARM processor.

The firmware library is a firmware function package, including program, data structure and macro definitions, all the performance features of peripherals of GD32L23x devices are involved in the package. The peripheral driving code and firmware examples on evaluation board are also included in firmware library. Users need not learn each peripherals in details and it’s easy to apply a peripheral by using the firmware library. Using firmware library can greatly reduce programming time, thereby reducing development costs.

The driving code of each peripheral is concluded by a group of functions, which describes all the performance features of the peripheral. Users can drive a peripheral by a group of APIs (application programming interface), all the APIs are standardized about the code structure, function name and parameter names.

All the driving source code accord with MISRA-C:2004 standard (example files accord with extended ANSI-C standard), and will not be influenced by differences of IDEs, except the startup files which are written differently according to the IDEs.

The commonly used firmware library includes all the functions of all the peripherals, so the code size and the execution speed may not be the optimal. For most applications, users can use the library functions directly, while for the applications which are strict with the code size and execution speed, the firmware library can be used as the reference resource of how to configure a peripheral, and users adjust the code according to actual needs.

The overall structure of the firmware library user manual is shown as below:

- Rules of user manual and firmware library;
- Firmware library overview;
- Functions and registers descriptions of firmware library.

### 1.1. Rules of User Manual and Firmware Library

#### 1.1.1. Peripherals

<table>
<thead>
<tr>
<th>Peripherals</th>
<th>Descriptions</th>
</tr>
</thead>
<tbody>
<tr>
<td>ADC</td>
<td>Analog-to-digital converter</td>
</tr>
<tr>
<td>CAU</td>
<td>Cryptographic Acceleration Unit</td>
</tr>
<tr>
<td>CMP</td>
<td>Comparator</td>
</tr>
<tr>
<td>CRC</td>
<td>CRC calculation unit</td>
</tr>
<tr>
<td>CTC</td>
<td>Clock trim controller</td>
</tr>
</tbody>
</table>
1.1.2. Naming rules

The firmware library naming rules are shown as below:

- The peripherals are shortened in XXX format, such as: ADC. More shorten information of peripherals refer to **Peripherals**;
- The name of source file and header file are started with “gd32l23x_”, such as: gd32l23x_adc.h;
- The constants used only in one file should be defined in the used file; the constants used in many files should be defined in corresponding header file. All the constants are written in uppercase of English letters;
- Registers are handled as constants. The naming of them are written in uppercase of English letters. In most cases, register names are shortened accord with the user manual;
- Variables are written in lowercase, when concluded by several words, underlines should be adapted among words;
- The naming of peripheral functions are started with the peripheral abbreviation added with an underline, when the function name is concluded by several words, underlines should be adapted among words, and all the peripheral functions are written in lowercase.

<table>
<thead>
<tr>
<th>Peripherals</th>
<th>Descriptions</th>
</tr>
</thead>
<tbody>
<tr>
<td>DBG</td>
<td>Debug</td>
</tr>
<tr>
<td>DAC</td>
<td>Digital-to-analog converter</td>
</tr>
<tr>
<td>DMA</td>
<td>Direct memory access controller</td>
</tr>
<tr>
<td>DMAMUX</td>
<td>DMA request multiplexer</td>
</tr>
<tr>
<td>EXTI</td>
<td>Interrupt/event controller</td>
</tr>
<tr>
<td>FMC</td>
<td>Flash memory controller</td>
</tr>
<tr>
<td>FWDGT</td>
<td>Free watchdog timer</td>
</tr>
<tr>
<td>GPIO/AFIO</td>
<td>General-purpose and alternate-function I/Os</td>
</tr>
<tr>
<td>I2C</td>
<td>Inter-integrated circuit interface</td>
</tr>
<tr>
<td>LPTIMER</td>
<td>Low power timer</td>
</tr>
<tr>
<td>LPUART</td>
<td>Low-power universal asynchronous receiver /transmitter</td>
</tr>
<tr>
<td>MISC</td>
<td>Nested Vectored Interrupt Controller</td>
</tr>
<tr>
<td>PMU</td>
<td>Power management unit</td>
</tr>
<tr>
<td>RCU</td>
<td>Reset and clock unit</td>
</tr>
<tr>
<td>RTC</td>
<td>Real-time Clock</td>
</tr>
<tr>
<td>SLCD</td>
<td>Segment LCD controller</td>
</tr>
<tr>
<td>SPI/I2S</td>
<td>Serial peripheral interface/Inter-IC sound</td>
</tr>
<tr>
<td>SYSCFG</td>
<td>System configuration</td>
</tr>
<tr>
<td>TIMER</td>
<td>TIMER</td>
</tr>
<tr>
<td>TRNG</td>
<td>True random number generator</td>
</tr>
<tr>
<td>USART</td>
<td>Universal synchronous/asynchronous receiver /transmitter</td>
</tr>
<tr>
<td>VREF</td>
<td>VREF</td>
</tr>
<tr>
<td>WWDGT</td>
<td>Window watchdog timer</td>
</tr>
</tbody>
</table>
2. Firmware Library Overview

2.1. File Structure of Firmware Library

GD32L23x Firmware Library, the file structure is shown as below:

Figure 2-1. File structure of firmware library of GD32L23x
**2.1.1. Examples Folder**

Examples folder, each of GD32 peripheral has a subfolder. Each subfolder contains one or more examples of the peripheral, to show how to use the peripheral correctly. Each of the example subfolder includes the files shown as below:

- readme.txt: the description and using guide of the example;
- gd32l23x_libopt.h: the header file configures all the peripherals used in the example, included by different “DEFINE” sentences (all the peripherals are enabled by default);
- gd32l23x_it.c: the source file include all the interrupt service routines (if no interrupt is used, then all the function bodies are empty);
- gd32l23x_it.h: the header file include all the prototypes of the interrupt service routines;
- systick.c: the source file include the precise time delay functions by using systick;
- systick.h: the header file include the prototype of the precise time delay functions by using systick;
- main.c: example code. Note: all the examples are not influenced by software IDEs.

**2.1.2. Firmware Folder**

Firmware folder includes all the subfolder and files which are the core part of the firmware:

- CMSIS subfolder includes the Cortex M23 kernel support files, the startup file based on the Cortex M23 kernel processor, the global header file of GD32L23x and system configuration file;
- GD32L23x_standard_peripheral subfolder:
  - Include subfolder includes all the header files of firmware libray, users need not modify this folder;
  - Source subfolder includes all the source files of firmware library, users need not modify this folder;

**Note:** All the codes accord with MISRA-C:2004 standard, and will not be influenced by different software IDEs.

**2.1.3. Template Folder**

Template folder includes a simple demo of how to use LED, how to print by USART and use key to control, (IAR_project is run in IAR, and Keil_project is run in Keil5). User can use the project template to compile the firmware examples, the steps are shown as below:

**Select files**

Open “Examples” folder, select the module to be tested, such as ADC, open “ADC” folder, select an example of ADC, such as “Analog_watchdog”, shown as below:
Copy files

Open “Template” folder, keep the folders of “IAR_project” and “Keil_project”, and delete the other files, then copy all the files in “SPI_master_transmit_slave_receive_interrupt” folder to the “Template” subfolder, shown as below:

Open a project

GD provides project in Keil and IAR, users can open project in different IDEs according to their need, such as “Keil_project”, open \Template\Keil_project\Project.uvprojx, shown as below:
Because different module and different functions adopt different files, users should add or delete the files in project according to the copied files, shown as below:

**Figure 2-4. Open the project file**

- First compile the project, if there is no error, then select the right jumper cap according to the description of readme, download the project to the target board, and there will be the phenomenon showed accord with the description of readme. The usage of IDE can refer to corresponding software user guide. If users are using Keil, the figure is shown as below:
2.1.4. Utilities Folder

Utilities folder includes files about the firmware examples on evaluation board:

- gd32e133r_eval.h is related header file of the evaluation board about running the firmware examples;
- gd32e233r_eval.c is related source file of the evaluation board about running the firmware examples.

**Note:** All the codes accord with MISRA-C:2004 standard, and will not be influenced by different software IDEs.

2.2. File descriptions of Firmware Library

The major files about the firmware library are listed and described in the table below.

<table>
<thead>
<tr>
<th>Files</th>
<th>Descriptions</th>
</tr>
</thead>
<tbody>
<tr>
<td>gd32l23x_libopt.h</td>
<td>The header file about all the header files of peripherals. It is the only one file which is necessity to be included in the user's application, to connect the firmware library and the application.</td>
</tr>
<tr>
<td>main.c</td>
<td>Example of main function.</td>
</tr>
<tr>
<td>gd32l23x_it.h</td>
<td>Header file, including all the prototypes of interrupt service routines.</td>
</tr>
<tr>
<td>gd32l23x_it.c</td>
<td>Source files about interrupt service routines of peripherals. User can written his own interrupt functions in this file. For the different interrupt service requests to the same interrupt vector, users can confirm the interrupt source by functions of judging interrupt flags of peripherals. The functions are included in the firmware library.</td>
</tr>
<tr>
<td>gd32l23x_xxx.h</td>
<td>The header file of peripheral xxx, including functions about peripheral xxx, and the variables used for functions.</td>
</tr>
<tr>
<td>gd32l23x_xxx.c</td>
<td>The C source file for driving peripheral xxx.</td>
</tr>
<tr>
<td>Files</td>
<td>Descriptions</td>
</tr>
<tr>
<td>------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>systick.h</td>
<td>The header file of systick.c, including prototypes of systick configuration function and delay function.</td>
</tr>
<tr>
<td>systick.c</td>
<td>The source file about systick configuration function and delay function.</td>
</tr>
<tr>
<td>readme.txt</td>
<td>Description document about how to configure and how to use the firmware example.</td>
</tr>
</tbody>
</table>
3. Firmware Library of Standard Peripherals

3.1. Overview of Firmware Library of Standard Peripherals

The description format of firmware functions are shown as below:

<table>
<thead>
<tr>
<th>Function name</th>
<th>Name of peripheral function</th>
</tr>
</thead>
<tbody>
<tr>
<td>Function prototype</td>
<td>Declaration prototype</td>
</tr>
<tr>
<td>Function descriptions</td>
<td>Explain the function how to work</td>
</tr>
<tr>
<td>Precondition</td>
<td>Requirements should meet before calling this function</td>
</tr>
<tr>
<td>The called functions</td>
<td>Other firmware functions called in this function</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Input parameter (in)</th>
</tr>
</thead>
<tbody>
<tr>
<td>xxxx</td>
</tr>
<tr>
<td>Description</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Output parameter (out)</th>
</tr>
</thead>
<tbody>
<tr>
<td>xxxx</td>
</tr>
<tr>
<td>Description</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Return value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Description</td>
</tr>
</tbody>
</table>

3.2. ADC

The 12-bit ADC is an analog-to-digital converter using the successive approximation method. The ADC registers are listed in chapter 3.2.1, the ADC firmware functions are introduced in chapter 3.2.2.

3.2.1. Descriptions of Peripheral registers

ADC registers are listed in the table shown as below:

<table>
<thead>
<tr>
<th>Registers</th>
<th>Descriptions</th>
</tr>
</thead>
<tbody>
<tr>
<td>ADC_STAT</td>
<td>Status register</td>
</tr>
<tr>
<td>ADC_CTL0</td>
<td>Control register 0</td>
</tr>
<tr>
<td>ADC_CTL1</td>
<td>Control register 1</td>
</tr>
<tr>
<td>ADC_SAMPT0</td>
<td>Sample time register 0</td>
</tr>
<tr>
<td>ADC_SAMPT1</td>
<td>Sample time register 1</td>
</tr>
<tr>
<td>ADC_IOFFx</td>
<td>Inserted channel data offset register x(x=0..3)</td>
</tr>
<tr>
<td>ADC_WDHT</td>
<td>Watchdog high threshold register</td>
</tr>
</tbody>
</table>
### Descriptions of Peripheral functions

ADC firmware functions are listed in the table shown as below:

#### Table 3-3. ADC firmware function

<table>
<thead>
<tr>
<th>Function name</th>
<th>Function description</th>
</tr>
</thead>
<tbody>
<tr>
<td>adc_deinit</td>
<td>reset ADC</td>
</tr>
<tr>
<td>adc_enable</td>
<td>enable ADC</td>
</tr>
<tr>
<td>adc_disable</td>
<td>disable ADC</td>
</tr>
<tr>
<td>adc_calibration_enable</td>
<td>ADC calibration and reset calibration</td>
</tr>
<tr>
<td>adc_dma_mode_enable</td>
<td>enable DMA request</td>
</tr>
<tr>
<td>adc_dma_mode_disable</td>
<td>disable DMA request</td>
</tr>
<tr>
<td>adc_discontinuous_mode_config</td>
<td>configure ADC discontinuous mode</td>
</tr>
<tr>
<td>adc_special_function_config</td>
<td>configure ADC special function</td>
</tr>
<tr>
<td>adc_channel_16_to_19</td>
<td>configure temperature sensor, internal reference voltage channel, VBAT channel or VSLCD channel</td>
</tr>
<tr>
<td>adc_data_alignment_config</td>
<td>configure ADC data alignment</td>
</tr>
<tr>
<td>adc_channel_length_config</td>
<td>configure the length of regular channel group or inserted channel group</td>
</tr>
<tr>
<td>adc_regular_channel_config</td>
<td>configure ADC regular channel</td>
</tr>
<tr>
<td>adc_inserted_channel_config</td>
<td>configure ADC inserted channel</td>
</tr>
<tr>
<td>adc_inserted_channel_offset_config</td>
<td>configure ADC inserted channel offset</td>
</tr>
<tr>
<td>adc_external_trigger_config</td>
<td>configure ADC external trigger</td>
</tr>
<tr>
<td>adc_external_trigger_source_config</td>
<td>configure ADC external trigger source</td>
</tr>
<tr>
<td>adc_software_trigger_enable</td>
<td>enable ADC software trigger</td>
</tr>
<tr>
<td>adc_regular_data_read</td>
<td>read ADC regular group data register</td>
</tr>
<tr>
<td>adc_inserted_data_read</td>
<td>read ADC inserted group data register</td>
</tr>
<tr>
<td>adc_watchdog_single_channel_enable</td>
<td>enable ADC analog watchdog single channel</td>
</tr>
<tr>
<td>adc_watchdog_group_channel_enable</td>
<td>enable ADC analog watchdog group channel</td>
</tr>
<tr>
<td>adc_watchdog_disable</td>
<td>disable ADC analog watchdog</td>
</tr>
</tbody>
</table>
### Function Description

<table>
<thead>
<tr>
<th>Function name</th>
<th>Function description</th>
</tr>
</thead>
<tbody>
<tr>
<td>adc_watchdog_threshold_config</td>
<td>configure ADC analog watchdog threshold</td>
</tr>
<tr>
<td>adc_resolution_config</td>
<td>configure ADC resolution</td>
</tr>
<tr>
<td>adc_oversample_mode_config</td>
<td>configure ADC oversample mode</td>
</tr>
<tr>
<td>adc_oversample_mode_enable</td>
<td>enable ADC oversample mode</td>
</tr>
<tr>
<td>adc_oversample_mode_disable</td>
<td>disable ADC oversample mode</td>
</tr>
<tr>
<td>adc_charge_pulse_width_counter</td>
<td>configure ADC charge pulse width counter</td>
</tr>
<tr>
<td>adc_charge_flag_get</td>
<td>get ADC charge flag</td>
</tr>
<tr>
<td>adc_regular_software_startconv_flag_get</td>
<td>get the flag of ADC regular channel software start conversion</td>
</tr>
<tr>
<td>adc_inserted_software_startconv_flag_get</td>
<td>get the flag of ADC inserted channel software start conversion</td>
</tr>
<tr>
<td>adc_flag_get</td>
<td>get ADC flag</td>
</tr>
<tr>
<td>adc_flag_clear</td>
<td>clear ADC flag</td>
</tr>
<tr>
<td>adc_interrupt_enable</td>
<td>enable ADC interrupt</td>
</tr>
<tr>
<td>adc_interrupt_disable</td>
<td>disable ADC interrupt</td>
</tr>
<tr>
<td>adc_interrupt_flag_get</td>
<td>get the ADC interrupt flag</td>
</tr>
<tr>
<td>adc_interrupt_flag_clear</td>
<td>clear the ADC interrupt flag</td>
</tr>
</tbody>
</table>

### adc_deinit

The description of adc_deinit is shown as below:

**Table 3-4. Function adc_deinit**

<table>
<thead>
<tr>
<th>Function name</th>
<th>adc_deinit</th>
</tr>
</thead>
<tbody>
<tr>
<td>Function prototype</td>
<td>void adc_deinit(void);</td>
</tr>
<tr>
<td>Function descriptions</td>
<td>reset ADC</td>
</tr>
<tr>
<td>Precondition</td>
<td></td>
</tr>
<tr>
<td>The called functions</td>
<td>rcu_periph_reset_enable / rcu_periph_reset_disable</td>
</tr>
<tr>
<td>Output parameter(out)</td>
<td></td>
</tr>
<tr>
<td>Return value</td>
<td></td>
</tr>
</tbody>
</table>

Example:

```c
/* reset ADC */
adc_deinit();
```

### adc_enable

The description of adc_enable is shown as below:

**Table 3-5. Function adc_enable**

<table>
<thead>
<tr>
<th>Function name</th>
<th>adc_enable</th>
</tr>
</thead>
</table>

Function prototype: `void adc_enable(void);`

Function descriptions: enable ADC

Precondition: -

The called functions: `adc_charge_pulse_width_counter`

### Output parameter(out)

- -

### Return value

- -

Example:

/* enable ADC */

`adc_enable();`

### adc_disable

The description of `adc_disable` is shown as below:

**Table 3-6. Function adc_disable**

<table>
<thead>
<tr>
<th>Function name</th>
<th>adc_disable</th>
</tr>
</thead>
<tbody>
<tr>
<td>Function prototype</td>
<td>void adc_disable(void);</td>
</tr>
<tr>
<td>Function descriptions</td>
<td>disable ADC</td>
</tr>
<tr>
<td>Precondition</td>
<td>-</td>
</tr>
<tr>
<td>The called functions</td>
<td>-</td>
</tr>
</tbody>
</table>

### Output parameter(out)

- -

### Return value

- -

Example:

/* disable ADC */

`adc_disable();`

### adc_calibration_enable

The description of `adc_calibration_enable` is shown as below:

**Table 3-7. Function adc_calibration_enable**

<table>
<thead>
<tr>
<th>Function name</th>
<th>adc_calibration_enable</th>
</tr>
</thead>
<tbody>
<tr>
<td>Function prototype</td>
<td>void adc_calibration_enable(void);</td>
</tr>
<tr>
<td>Function descriptions</td>
<td>ADC calibration and reset calibration</td>
</tr>
<tr>
<td>Precondition</td>
<td>-</td>
</tr>
<tr>
<td>The called functions</td>
<td>-</td>
</tr>
</tbody>
</table>

### Output parameter(out)

- -

Example:

/* disable ADC */

`adc_calibration_enable();`
Example:

```c
/* ADC calibration and reset calibration */
adc_calibration_enable();
```

### adc_dma_mode_enable

The description of `adc_dma_mode_enable` is shown as below:

<table>
<thead>
<tr>
<th>Function name</th>
<th>adc_dma_mode_enable</th>
</tr>
</thead>
<tbody>
<tr>
<td>Function prototype</td>
<td>void adc_dma_mode_enable(void);</td>
</tr>
<tr>
<td>Function descriptions</td>
<td>enable DMA request</td>
</tr>
<tr>
<td>Precondition</td>
<td>-</td>
</tr>
<tr>
<td>The called functions</td>
<td>-</td>
</tr>
<tr>
<td>Output parameter(out)</td>
<td>-</td>
</tr>
<tr>
<td>Return value</td>
<td>-</td>
</tr>
</tbody>
</table>

Example:

```c
/* enable ADC DMA request */
adc_dma_mode_enable();
```

### adc_dma_mode_disable

The description of `adc_dma_mode_disable` is shown as below:

<table>
<thead>
<tr>
<th>Function name</th>
<th>adc_dma_mode_disable</th>
</tr>
</thead>
<tbody>
<tr>
<td>Function prototype</td>
<td>void adc_dma_mode_disable(void);</td>
</tr>
<tr>
<td>Function descriptions</td>
<td>disable DMA request</td>
</tr>
<tr>
<td>Precondition</td>
<td>-</td>
</tr>
<tr>
<td>The called functions</td>
<td>-</td>
</tr>
<tr>
<td>Output parameter(out)</td>
<td>-</td>
</tr>
<tr>
<td>Return value</td>
<td>-</td>
</tr>
</tbody>
</table>

Example:

```c
/* disable ADC DMA request */
adc_dma_mode_disable();
```
adc_dma_mode_disable();

**adc_discontinuous_mode_config**

The description of `adc_discontinuous_mode_config` is shown as below:

**Table 3-10. Function adc_discontinuous_mode_config**

<table>
<thead>
<tr>
<th>Function name</th>
<th>adc_discontinuous_mode_config</th>
</tr>
</thead>
<tbody>
<tr>
<td>Function prototype</td>
<td>void adc_discontinuous_mode_config(uint8_t adc_channel_group, uint8_t length);</td>
</tr>
<tr>
<td>Function descriptions</td>
<td>configure ADC discontinuous mode</td>
</tr>
<tr>
<td>Precondition</td>
<td>-</td>
</tr>
<tr>
<td>The called functions</td>
<td>-</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Input parameter (in)</th>
</tr>
</thead>
<tbody>
<tr>
<td>adc_channel_group</td>
</tr>
<tr>
<td>ADC_REGULAR_CHANNEL</td>
</tr>
<tr>
<td>ADC_INSERTED_CHANNEL</td>
</tr>
<tr>
<td>ADC_CHANNEL_DISCON_DISABLE</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Input parameter (in)</th>
</tr>
</thead>
<tbody>
<tr>
<td>length</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Output parameter (out)</th>
</tr>
</thead>
<tbody>
<tr>
<td>-</td>
</tr>
</tbody>
</table>

Return value -

Example:

/* configure ADC regular channel group discontinuous mode */
adc_discontinuous_mode_config(ADC_REGULAR_CHANNEL, 6);

**adc_special_function_config**

The description of `adc_special_function_config` is shown as below:

**Table 3-11. Function adc_special_function_config**

<table>
<thead>
<tr>
<th>Function name</th>
<th>adc_special_function_config</th>
</tr>
</thead>
<tbody>
<tr>
<td>Function prototype</td>
<td>void adc_special_function_config(uint32_t function, ControlStatus newvalue);</td>
</tr>
<tr>
<td>Function descriptions</td>
<td>configure ADC special function</td>
</tr>
<tr>
<td>Precondition</td>
<td>-</td>
</tr>
<tr>
<td>The called functions</td>
<td>-</td>
</tr>
</tbody>
</table>
Input parameter(in)

<table>
<thead>
<tr>
<th>function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ADC_SCAN_MODE</td>
<td>scan mode select</td>
</tr>
<tr>
<td>ADC_INSERTED_CHANNEL_AUTO</td>
<td>inserted channel group convert automatically</td>
</tr>
<tr>
<td>ADC_CONTINUOUS_MODE</td>
<td>continuous mode select</td>
</tr>
</tbody>
</table>

Input parameter(in)

<table>
<thead>
<tr>
<th>newvalue</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ENABLE</td>
<td>enable function</td>
</tr>
<tr>
<td>DISABLE</td>
<td>disable function</td>
</tr>
</tbody>
</table>

Output parameter(out)

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Return value

- -

Example:

/* enable ADC scan mode */

adc_special_function_config(ADC_SCAN_MODE, ENABLE);

**adc_channel_16_to_19**

The description of adc_channel_16_to_19 is shown as below:

**Table 3-12. Function adc_channel_16_to_19**

<table>
<thead>
<tr>
<th>Function name</th>
<th>adc_channel_16_to_19</th>
</tr>
</thead>
<tbody>
<tr>
<td>Function prototype</td>
<td>void adc_channel_16_to_19(uint32_t function, ControlStatus newvalue);</td>
</tr>
<tr>
<td>Function descriptions</td>
<td>configure temperature sensor, internal reference voltage channel, VBAT channel or VSLCD channel</td>
</tr>
<tr>
<td>Precondition</td>
<td>-</td>
</tr>
<tr>
<td>The called functions</td>
<td>-</td>
</tr>
</tbody>
</table>

Input parameter(in)

<table>
<thead>
<tr>
<th>function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ADC_TEMP_CHANNEL_SWITCH</td>
<td>channel 16 (temperature sensor) switch of ADC</td>
</tr>
<tr>
<td>ADC_INTERNAL_CHANNEL_SWITCH</td>
<td>channel 17 (internal reference voltage) switch of ADC</td>
</tr>
<tr>
<td>ADC_VBAT_CHANNEL_SWITCH</td>
<td>channel 18 (1/3 voltage of external battery) switch of ADC</td>
</tr>
<tr>
<td>ADC_VSLCD_CHANNEL_SWITCH</td>
<td>channel 19 (1/3 voltage of VSLCD) switch of ADC</td>
</tr>
</tbody>
</table>
newvalue | control value  
---|---  
ENABLE | enable function  
DISABLE | disable function  

### Output parameter (out)

```
-  
```

### Return value

```
-  
```

Example:

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

#### adc_data_alignment_config

The description of `adc_data_alignment_config` is shown as below:

<table>
<thead>
<tr>
<th>Function name</th>
<th>adc_data_alignment_config</th>
</tr>
</thead>
<tbody>
<tr>
<td>Function prototype</td>
<td>void adc_data_alignment_config(uint32_t data_alignment);</td>
</tr>
<tr>
<td>Function descriptions</td>
<td>configure ADC data alignment</td>
</tr>
<tr>
<td>Precondition</td>
<td>-</td>
</tr>
<tr>
<td>The called functions</td>
<td>-</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Input parameter (in)</th>
</tr>
</thead>
<tbody>
<tr>
<td>data_alignment</td>
</tr>
<tr>
<td>ADC_DATAALIGN_RIGHT</td>
</tr>
<tr>
<td>ADC_DATAALIGN_LEFT</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Output parameter (out)</th>
</tr>
</thead>
<tbody>
<tr>
<td>-</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Return value</th>
</tr>
</thead>
<tbody>
<tr>
<td>-</td>
</tr>
</tbody>
</table>

Example:

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

#### adc_channel_length_config

The description of `adc_channel_length_config` is shown as below:

<table>
<thead>
<tr>
<th>Function name</th>
<th>adc_channel_length_config</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Function prototype: void adc_channel_length_config(uint8_t adc_channel_group, uint32_t length);

Function descriptions: configure the length of regular channel group or inserted channel group

Precondition:

The called functions:

Input parameter (in):
- adc_channel_group
  - select the channel group
  - ADC_REGULAR_CHANNEL
    - regular channel group
  - ADC_INSERTED_CHANNEL
    - inserted channel group

Input parameter (in):
- length
  - the length of the channel, regular channel 1-16, inserted channel 1-4

Output parameter (out):

Return value:

Example:

/* configure the length of ADC regular channel */
adc_channel_length_config(ADC_REGULAR_CHANNEL, 4);

adc_regular_channel_config

The description of adc_regular_channel_config is shown as below:

Table 3-15. Function adc_regular_channel_config

<table>
<thead>
<tr>
<th>Function name</th>
<th>adc_regular_channel_config</th>
</tr>
</thead>
<tbody>
<tr>
<td>Function prototype</td>
<td>void adc_regular_channel_config(uint8_t rank, uint8_t adc_channel, uint32_t sample_time);</td>
</tr>
<tr>
<td>Function descriptions</td>
<td>configure ADC regular channel</td>
</tr>
<tr>
<td>Precondition</td>
<td>-</td>
</tr>
<tr>
<td>The called functions</td>
<td>-</td>
</tr>
</tbody>
</table>

Input parameter (in):
- rank
  - the regular group sequence rank, this parameter must be between 0 to 15

Input parameter (in):
- adc_channel
  - the selected ADC channel
  - ADC_CHANNEL_x
    - ADC Channel x (x=0..19)

Input parameter (in):
- sample_time
  - the sample time value
  - ADC_SAMPLETIME_2 POINT5
    - 2.5 cycles
  - ADC_SAMPLETIME_7.5
    - 7.5 cycles
<table>
<thead>
<tr>
<th>7POINT5</th>
<th>13.5 cycles</th>
</tr>
</thead>
<tbody>
<tr>
<td>ADC_SAMPETIME_13POINT5</td>
<td>13.5 cycles</td>
</tr>
<tr>
<td>ADC_SAMPETIME_28POINT5</td>
<td>28.5 cycles</td>
</tr>
<tr>
<td>ADC_SAMPETIME_41POINT5</td>
<td>41.5 cycles</td>
</tr>
<tr>
<td>ADC_SAMPETIME_55POINT5</td>
<td>55.5 cycles</td>
</tr>
<tr>
<td>ADC_SAMPETIME_71POINT5</td>
<td>71.5 cycles</td>
</tr>
<tr>
<td>ADC_SAMPETIME_239POINT5</td>
<td>239.5 cycles</td>
</tr>
</tbody>
</table>

Output parameter (out)

- -

Return value

- -

Example:

/* configure ADC regular channel */

adc_regular_channel_config(1, ADC_CHANNEL_0, ADC_SAMPETIME_7POINT5);

adc_inserted_channel_config

The description of adc_inserted_channel_config is shown as below:

Table 3-16. Function adc_inserted_channel_config

<table>
<thead>
<tr>
<th>Function name</th>
<th>adc_inserted_channel_config</th>
</tr>
</thead>
<tbody>
<tr>
<td>Function prototype</td>
<td>void adc_inserted_channel_config(uint8_t rank, uint8_t adc_channel, uint32_t sample_time);</td>
</tr>
<tr>
<td>Function descriptions</td>
<td>configure ADC inserted channel</td>
</tr>
<tr>
<td>Precondition</td>
<td>-</td>
</tr>
<tr>
<td>The called functions</td>
<td>-</td>
</tr>
</tbody>
</table>

Input parameter (in)

<table>
<thead>
<tr>
<th>rank</th>
<th>the inserted group sequencer rank, this parameter must be between 0 to 3</th>
</tr>
</thead>
</table>

Input parameter (in)

<table>
<thead>
<tr>
<th>adc_channel</th>
<th>the selected ADC channel</th>
</tr>
</thead>
<tbody>
<tr>
<td>ADC_CHANNEL_x</td>
<td>ADC Channelx (x=0..19)</td>
</tr>
</tbody>
</table>

Input parameter (in)

<table>
<thead>
<tr>
<th>sample_time</th>
<th>the sample time value</th>
</tr>
</thead>
<tbody>
<tr>
<td>ADC_SAMPETIME_2POINT5</td>
<td>2.5 cycles</td>
</tr>
<tr>
<td>ADC_SAMPETIME_7POINT5</td>
<td>7.5 cycles</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>-------</td>
<td>---</td>
</tr>
<tr>
<td>7POINT5</td>
<td></td>
</tr>
<tr>
<td>ADC_SAMPLETIME_13POINT5</td>
<td>13.5 cycles</td>
</tr>
<tr>
<td>ADC_SAMPLETIME_28POINT5</td>
<td>28.5 cycles</td>
</tr>
<tr>
<td>ADC_SAMPLETIME_41POINT5</td>
<td>41.5 cycles</td>
</tr>
<tr>
<td>ADC_SAMPLETIME_55POINT5</td>
<td>55.5 cycles</td>
</tr>
<tr>
<td>ADC_SAMPLETIME_71POINT5</td>
<td>71.5 cycles</td>
</tr>
<tr>
<td>ADC_SAMPLETIME_239POINT5</td>
<td>239.5 cycles</td>
</tr>
</tbody>
</table>

Output parameter(out)
- -

Return value
- -

Example:

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

adc_inserted_channel_offset_config

The description of adc_inserted_channel_offset_config is shown as below:

**Table 3-17. Function adc_inserted_channel_offset_config**

<table>
<thead>
<tr>
<th>Function name</th>
<th>adc_inserted_channel_offset_config</th>
</tr>
</thead>
</table>
| Function prototype | void adc_inserted_channel_offset_config(uint8_t inserted_channel, uint16_t offset);
| Function descriptions | configure ADC inserted channel offset |
| Precondition | - |
| The called functions | - |

**Input parameter(in)**

<table>
<thead>
<tr>
<th>inserted_channel</th>
<th>inserted channel select</th>
</tr>
</thead>
<tbody>
<tr>
<td>ADC_INSERTED_CHANNEL_x</td>
<td>inserted channel, x=0,1,2,3</td>
</tr>
</tbody>
</table>

**Input parameter(in)**

| offset | the offset data |

**Output parameter(out)**
- -

Return value
- -
Example:

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

**adc_external_trigger_config**

The description of `adc_external_trigger_config` is shown as below:

<table>
<thead>
<tr>
<th>Table 3-18. Function <code>adc_external_trigger_config</code></th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Function name</strong></td>
</tr>
<tr>
<td><strong>Function prototype</strong></td>
</tr>
<tr>
<td><strong>Function descriptions</strong></td>
</tr>
<tr>
<td><strong>Precondition</strong></td>
</tr>
<tr>
<td><strong>The called functions</strong></td>
</tr>
</tbody>
</table>

**Input parameter**(in)

<table>
<thead>
<tr>
<th><code>adc_channel_group</code></th>
<th>select the channel group</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>ADC_REGULAR_CHANNEL</code></td>
<td>regular channel group</td>
</tr>
<tr>
<td><code>ADC_INSERTED_CHANNEL</code></td>
<td>inserted channel group</td>
</tr>
</tbody>
</table>

**Input parameter**(in)

| `newvalue` | control value |
| `ENABLE` | enable function |
| `DISABLE` | disable function |

**Output parameter**(out)

| - | - |

**Return value**

| - | - |

Example:

/* enable ADC inserted channel group external trigger */
adc_external_trigger_config(ADC_INSERTED_CHANNEL_0, ENABLE);

**adc_external_trigger_source_config**

The description of `adc_external_trigger_source_config` is shown as below:

<table>
<thead>
<tr>
<th>Table 3-19. Function <code>adc_external_trigger_source_config</code></th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Function name</strong></td>
</tr>
<tr>
<td><strong>Function prototype</strong></td>
</tr>
<tr>
<td><strong>Function descriptions</strong></td>
</tr>
</tbody>
</table>
### Precondition

- 

### The called functions

- 

#### Input parameter(in)

**adc_channel_group**

- select the channel group

**ADC_REGULAR_CHANNEL**

- regular channel group

**ADC_INSERTED_CHANNEL**

- inserted channel group

#### Input parameter(in)

**external_trigger_source**

- regular or inserted group trigger source

**ADC_EXTTRIG_REGULAR_TIMER8_CH0**

- external trigger TIMER8 CH0 event select for regular channel

**ADC_EXTTRIG_REGULAR_TIMER8_CH1**

- external trigger TIMER8 CH1 event select for regular channel

**ADC_EXTTRIG_REGULAR_TIMER1_CH1**

- external trigger TIMER1 CH1 event select for regular channel

**ADC_EXTTRIG_REGULAR_TIMER2_TRGO**

- external trigger TIMER2 TRGO event select for regular channel

**ADC_EXTTRIG_REGULAR_TIMER11_CH0**

- external trigger TIMER11 CH0 event select for regular channel

**ADC_EXTTRIG_REGULAR_EXTI_11**

- external trigger interrupt line 11 select for regular channel

**ADC_EXTTRIG_REGULAR_NONE**

- external trigger software event select for regular channel

**ADC_EXTTRIG_INSERTED_TIMER1_TRGO**

- TIMER1 TRGO event select for inserted channel

**ADC_EXTTRIG_INSERTED_TIMER1_CH0**

- TIMER1 CH0 event select for inserted channel

**ADC_EXTTRIG_INSERTED_TIMER2_CH3**

- TIMER2 CH3 event select for inserted channel

**ADC_EXTTRIG_INSERTED_EXTI_15**

- external interrupt line 15 event select for inserted channel

**ADC_EXTTRIG_INSERTED_NONE**

- external trigger software event select for inserted channel

#### Output parameter(out)

- 

#### Return value

- 

---

Example:

/* configure ADC regular channel external trigger source */
adc_external_trigger_source_config(ADC_REGULAR_CHANNEL, ADC_EXTTRIG_REGULAR_T8_CH0);

**adc_software_trigger_enable**

The description of adc_software_trigger_enable is shown as below:

**Table 3-20. Function adc_software_trigger_enable**

<table>
<thead>
<tr>
<th>Function name</th>
<th>adc_software_trigger_enable</th>
</tr>
</thead>
<tbody>
<tr>
<td>Function prototype</td>
<td>void adc_software_trigger_enable(uint8_t adc_channel_group);</td>
</tr>
<tr>
<td>Function description</td>
<td>enable ADC software trigger</td>
</tr>
<tr>
<td>Precondition</td>
<td>-</td>
</tr>
<tr>
<td>The called functions</td>
<td>-</td>
</tr>
</tbody>
</table>

**Input parameter (in)**

- `adc_channel_group`
  - select the channel group
  - `ADC_REGULAR_CHANNEL` regular channel group
  - `ADC_INSERTED_CHANNEL` inserted channel group

**Output parameter (out)**

- -

**Return value**

- -

Example:

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

**adc_regular_data_read**

The description of adc_regular_data_read is shown as below:

**Table 3-21. Function adc_regular_data_read**

<table>
<thead>
<tr>
<th>Function name</th>
<th>adc_regular_data_read</th>
</tr>
</thead>
<tbody>
<tr>
<td>Function prototype</td>
<td>uint16_t adc_regular_data_read(void);</td>
</tr>
<tr>
<td>Function description</td>
<td>read ADC regular group data register</td>
</tr>
<tr>
<td>Precondition</td>
<td>-</td>
</tr>
<tr>
<td>The called functions</td>
<td>-</td>
</tr>
</tbody>
</table>

**Output parameter (out)**

- -

**Return value**

- `uint16_t` the conversion value: 0~0xFFFF

Example:
/* read ADC regular group data register */

uint16_t adc_value = 0;
adc_value = adc_regular_data_read();

**adc_inserted_data_read**

The description of adc_inserted_data_read is shown as below:

<table>
<thead>
<tr>
<th>Function name</th>
<th>adc_inserted_data_read</th>
</tr>
</thead>
<tbody>
<tr>
<td>Function prototype</td>
<td>uint16_t adc_inserted_data_read(uint8_t inserted_channel);</td>
</tr>
<tr>
<td>Function descriptions</td>
<td>read ADC inserted group data register</td>
</tr>
<tr>
<td>Precondition</td>
<td>-</td>
</tr>
<tr>
<td>The called functions</td>
<td>-</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Input parameter(in)</th>
</tr>
</thead>
<tbody>
<tr>
<td>inserted_channel</td>
</tr>
<tr>
<td>ADC_INSERTED_CHANNEL_x</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Output parameter(out)</th>
</tr>
</thead>
<tbody>
<tr>
<td>-</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Return value</th>
</tr>
</thead>
<tbody>
<tr>
<td>uint16_t</td>
</tr>
<tr>
<td>the conversion value: 0~0xFFFF</td>
</tr>
</tbody>
</table>

Example:

/* read ADC inserted group data register */

uint16_t adc_value = 0;
adc_value = adc_inserted_data_read(ADC_INSERTED_CHANNEL_0);

**adc_watchdog_single_channel_enable**

The description of adc_watchdog_single_channel_enable is shown as below:

<table>
<thead>
<tr>
<th>Function name</th>
<th>adc_watchdog_single_channel_enable</th>
</tr>
</thead>
<tbody>
<tr>
<td>Function prototype</td>
<td>void adc_watchdog_single_channel_enable(uint8_t adc_channel);</td>
</tr>
<tr>
<td>Function descriptions</td>
<td>enable ADC analog watchdog single channel</td>
</tr>
<tr>
<td>Precondition</td>
<td>-</td>
</tr>
<tr>
<td>The called functions</td>
<td>-</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Input parameter(in)</th>
</tr>
</thead>
<tbody>
<tr>
<td>adc_channel</td>
</tr>
<tr>
<td>ADC_CHANNEL_x</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Output parameter(out)</th>
</tr>
</thead>
<tbody>
<tr>
<td>-</td>
</tr>
</tbody>
</table>

Example:

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

`adc_watchdog_single_channel_enable(ADC_CHANNEL_1);`

**adc_watchdog_group_channel_enable**

The description of `adc_watchdog_group_channel_enable` is shown as below:

Table 3-24. Function `adc_watchdog_group_channel_enable`

<table>
<thead>
<tr>
<th>Function Name</th>
<th>adc_watchdog_group_channel_enable</th>
</tr>
</thead>
<tbody>
<tr>
<td>Function prototype</td>
<td>void adc_watchdog_group_channel_enable(uint8_t adc_channel_group);</td>
</tr>
<tr>
<td>Function descriptions</td>
<td>enable ADC analog watchdog group channel</td>
</tr>
<tr>
<td>Precondition</td>
<td>-</td>
</tr>
<tr>
<td>The called functions</td>
<td>-</td>
</tr>
</tbody>
</table>

Input parameter (in)

- `adc_channel_group` the channel group use analog watchdog
  - `ADC_REGULAR_CHANNEL` regular channel group
  - `ADC_INSERTED_CHANNEL` inserted channel group
  - `ADC_REGULAR_INSERTED_CHANNEL` both regular and inserted group

Output parameter (out)

- -

Return value

- -

Example:

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

`adc_watchdog_group_channel_enable(ADC_REGULAR_CHANNEL);`

**adc_watchdog_disable**

The description of `adc_watchdog_disable` is shown as below:

Table 3-25. Function `adc_watchdog_disable`

<table>
<thead>
<tr>
<th>Function Name</th>
<th>adc_watchdog_disable</th>
</tr>
</thead>
<tbody>
<tr>
<td>Function prototype</td>
<td>void adc_watchdog_disable(void);</td>
</tr>
<tr>
<td>Function descriptions</td>
<td>disable ADC analog watchdog</td>
</tr>
<tr>
<td>Precondition</td>
<td>-</td>
</tr>
<tr>
<td>The called functions</td>
<td>-</td>
</tr>
</tbody>
</table>
Input parameter (in)

- -

Output parameter (out)

- -

Return value

- -

Example:

/* disable ADC analog watchdog */

adc_watchdog_disable();

**adc_watchdog_threshold_config**

The description of **adc_watchdog_threshold_config** is shown as below:

<table>
<thead>
<tr>
<th>Function name</th>
<th>adc_watchdog_threshold_config</th>
</tr>
</thead>
<tbody>
<tr>
<td>Function prototype</td>
<td>void adc_watchdog_threshold_config(uint16_t low_threshold, uint16_t high_threshold);</td>
</tr>
<tr>
<td>Function descriptions</td>
<td>configure ADC analog watchdog threshold</td>
</tr>
<tr>
<td>Precondition</td>
<td>-</td>
</tr>
<tr>
<td>The called functions</td>
<td>-</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Input parameter (in)</th>
</tr>
</thead>
<tbody>
<tr>
<td>low_threshold</td>
</tr>
<tr>
<td>analog watchdog low threshold, 0..4095</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Input parameter (in)</th>
</tr>
</thead>
<tbody>
<tr>
<td>high_threshold</td>
</tr>
<tr>
<td>analog watchdog high threshold, 0..4095</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Output parameter (out)</th>
</tr>
</thead>
<tbody>
<tr>
<td>-</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Return value</th>
</tr>
</thead>
<tbody>
<tr>
<td>-</td>
</tr>
</tbody>
</table>

Example:

/* configure ADC analog watchdog threshold */

adc_watchdog_threshold_config(0x0400, 0x0A00);

**adc_resolution_config**

The description of **adc_resolution_config** is shown as below:

<table>
<thead>
<tr>
<th>Function name</th>
<th>adc_resolution_config</th>
</tr>
</thead>
<tbody>
<tr>
<td>Function prototype</td>
<td>void adc_resolution_config(uint32_t resolution);</td>
</tr>
<tr>
<td>Function descriptions</td>
<td>configure ADC resolution</td>
</tr>
</tbody>
</table>
Precondition

The called functions

<table>
<thead>
<tr>
<th>Input parameter(in)</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>resolution</td>
<td>ADC resolution</td>
</tr>
<tr>
<td>ADC_RESOLUTION_12B</td>
<td>12-bit ADC resolution</td>
</tr>
<tr>
<td>ADC_RESOLUTION_10B</td>
<td>10-bit ADC resolution</td>
</tr>
<tr>
<td>ADC_RESOLUTION_8B</td>
<td>8-bit ADC resolution</td>
</tr>
<tr>
<td>ADC_RESOLUTION_6B</td>
<td>6-bit ADC resolution</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Output parameter(out)</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>-</td>
<td>-</td>
</tr>
</tbody>
</table>

Return value

- -

Example:

/* configure ADC resolution */
adc_resolution_config(ADC_RESOLUTION_12B);

**adc_oversample_mode_config**

The description of adc_oversample_mode_config is shown as below:

<table>
<thead>
<tr>
<th>Function name</th>
<th>adc_oversample_mode_config</th>
</tr>
</thead>
<tbody>
<tr>
<td>Function prototype</td>
<td>void adc_oversample_mode_config(uint32_t mode, uint16_t shift, uint8_t ratio);</td>
</tr>
<tr>
<td>Function descriptions</td>
<td>configure ADC oversample mode</td>
</tr>
<tr>
<td>Precondition</td>
<td>-</td>
</tr>
<tr>
<td>The called functions</td>
<td>-</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Input parameter(in)</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>mode</td>
<td>ADC oversampling mode</td>
</tr>
<tr>
<td>ADC_OVERSAMPLING_ALL_CONVERT</td>
<td>all oversampled conversions for a channel are done consecutively after a trigger</td>
</tr>
<tr>
<td>ADC_OVERSAMPLING_ONE_CONVERT</td>
<td>each oversampled conversion for a channel needs a trigger</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Input parameter(in)</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>shift</td>
<td>ADC oversampling shift</td>
</tr>
<tr>
<td>ADC_OVERSAMPLING_SHIFT_NONE</td>
<td>no oversampling shift</td>
</tr>
</tbody>
</table>
### ADC OVERSAMPLING Details

<table>
<thead>
<tr>
<th>Description</th>
<th>ADC OVERSAMPLING</th>
<th>Details</th>
</tr>
</thead>
<tbody>
<tr>
<td>1-bit oversampling shift</td>
<td>_SHIF 1B</td>
<td>1-bit oversampling shift</td>
</tr>
<tr>
<td>2-bit oversampling shift</td>
<td>_SHIF 2B</td>
<td>2-bit oversampling shift</td>
</tr>
<tr>
<td>3-bit oversampling shift</td>
<td>_SHIF 3B</td>
<td>3-bit oversampling shift</td>
</tr>
<tr>
<td>4-bit oversampling shift</td>
<td>_SHIF 4B</td>
<td>4-bit oversampling shift</td>
</tr>
<tr>
<td>5-bit oversampling shift</td>
<td>_SHIF 5B</td>
<td>5-bit oversampling shift</td>
</tr>
<tr>
<td>6-bit oversampling shift</td>
<td>_SHIF 6B</td>
<td>6-bit oversampling shift</td>
</tr>
<tr>
<td>7-bit oversampling shift</td>
<td>_SHIF 7B</td>
<td>7-bit oversampling shift</td>
</tr>
<tr>
<td>8-bit oversampling shift</td>
<td>_SHIF 8B</td>
<td>8-bit oversampling shift</td>
</tr>
</tbody>
</table>

#### Input parameter (in)

<table>
<thead>
<tr>
<th>Description</th>
<th>ADC OVERSAMPLING</th>
<th>Details</th>
</tr>
</thead>
<tbody>
<tr>
<td>ADC oversampling ratio</td>
<td>_RATIO_MUL2</td>
<td>oversampling ratio multiple 2</td>
</tr>
<tr>
<td>ADC oversampling ratio</td>
<td>_RATIO_MUL4</td>
<td>oversampling ratio multiple 4</td>
</tr>
<tr>
<td>ADC oversampling ratio</td>
<td>_RATIO_MUL8</td>
<td>oversampling ratio multiple 8</td>
</tr>
<tr>
<td>ADC oversampling ratio</td>
<td>_RATIO_MUL16</td>
<td>oversampling ratio multiple 16</td>
</tr>
<tr>
<td>ADC oversampling ratio</td>
<td>_RATIO_MUL32</td>
<td>oversampling ratio multiple 32</td>
</tr>
<tr>
<td>ADC oversampling ratio</td>
<td>_RATIO_MUL64</td>
<td>oversampling ratio multiple 64</td>
</tr>
<tr>
<td>ADC oversampling ratio</td>
<td>_RATIO_MUL128</td>
<td>oversampling ratio multiple 128</td>
</tr>
<tr>
<td>ADC oversampling ratio</td>
<td>_RATIO_MUL256</td>
<td>oversampling ratio multiple 256</td>
</tr>
</tbody>
</table>

#### Output parameter (out)

- 

#### Return value

- 

Example:

```c
/* configure ADC oversample mode: 16 times sample, 4 bits shift */
adc_oversample_mode_config(ADC_OVERSAMPLING_ALL_CONVERT,
```
adc_oversample_mode_enable

The description of adc_oversample_mode_enable is shown as below:

Table 3-29. Function adc_oversample_mode_enable

<table>
<thead>
<tr>
<th>Function name</th>
<th>adc_oversample_mode_enable</th>
</tr>
</thead>
<tbody>
<tr>
<td>Function prototype</td>
<td>void adc_oversample_mode_enable(void);</td>
</tr>
<tr>
<td>Function descriptions</td>
<td>enable ADC oversample mode</td>
</tr>
<tr>
<td>Precondition</td>
<td>-</td>
</tr>
<tr>
<td>The called functions</td>
<td>-</td>
</tr>
<tr>
<td>Input parameter(in)</td>
<td>-</td>
</tr>
<tr>
<td>Output parameter(out)</td>
<td>-</td>
</tr>
<tr>
<td>Return value</td>
<td>-</td>
</tr>
</tbody>
</table>

Example:

/* enable ADC oversample mode */
adc_oversample_mode_enable();

adc_oversample_mode_disable

The description of adc_oversample_mode_disable is shown as below:

Table 3-30. Function adc_oversample_mode_disable

<table>
<thead>
<tr>
<th>Function name</th>
<th>adc_oversample_mode_disable</th>
</tr>
</thead>
<tbody>
<tr>
<td>Function prototype</td>
<td>void adc_oversample_mode_disable(void);</td>
</tr>
<tr>
<td>Function descriptions</td>
<td>disable ADC oversample mode</td>
</tr>
<tr>
<td>Precondition</td>
<td>-</td>
</tr>
<tr>
<td>The called functions</td>
<td>-</td>
</tr>
<tr>
<td>Input parameter(in)</td>
<td>-</td>
</tr>
<tr>
<td>Output parameter(out)</td>
<td>-</td>
</tr>
<tr>
<td>Return value</td>
<td>-</td>
</tr>
</tbody>
</table>

Example:

/* disable ADC oversample mode */
adc_oversample_mode_disable();
adc_charge_pulse_width_counter

The description of adc_charge_pulse_width_counter is shown as below:

Table 3-31. Function adc_charge_pulse_width_counter

<table>
<thead>
<tr>
<th>Function name</th>
<th>adc_charge_pulse_width_counter</th>
</tr>
</thead>
<tbody>
<tr>
<td>Function prototype</td>
<td>void adc_charge_pulse_width_counter(uint32_t value);</td>
</tr>
<tr>
<td>Function descriptions</td>
<td>configure ADC charge pulse width counter</td>
</tr>
<tr>
<td>Precondition</td>
<td>-</td>
</tr>
<tr>
<td>The called functions</td>
<td>-</td>
</tr>
<tr>
<td>Input parameter(in)</td>
<td>value</td>
</tr>
<tr>
<td>Output parameter(out)</td>
<td>-</td>
</tr>
<tr>
<td>Return value</td>
<td>-</td>
</tr>
</tbody>
</table>

Example:

/* configure ADC charge pulse width counter */
adc_charge_pulse_width_counter(100);

adc_charge_flag_get

The description of adc_charge_flag_get is shown as below:

Table 3-32. Function adc_charge_flag_get

<table>
<thead>
<tr>
<th>Function name</th>
<th>adc_charge_flag_get</th>
</tr>
</thead>
<tbody>
<tr>
<td>Function prototype</td>
<td>FlagStatus adc_charge_flag_get(uint32_t flag);</td>
</tr>
<tr>
<td>Function descriptions</td>
<td>get the ADC charge flag</td>
</tr>
<tr>
<td>Precondition</td>
<td>-</td>
</tr>
<tr>
<td>The called functions</td>
<td>-</td>
</tr>
<tr>
<td>Input parameter(in)</td>
<td>flag</td>
</tr>
<tr>
<td>Output parameter(out)</td>
<td>ADC_FLAG_CHARGE</td>
</tr>
<tr>
<td>Return value</td>
<td>-</td>
</tr>
</tbody>
</table>

Example:

/* get the ADC charge flag */
FlagStatus flag_value;
flag_value = adc_charge_flag_get(ADC_FLAG_CHARGE);
adc_regular_software_startconv_flag_get

The description of adc_regular_software_startconv_flag_get is shown as below:

<table>
<thead>
<tr>
<th>Function name</th>
<th>adc_regular_software_startconv_flag_get</th>
</tr>
</thead>
<tbody>
<tr>
<td>Function prototype</td>
<td>FlagStatus adc_regular_software_startconv_flag_get(void);</td>
</tr>
<tr>
<td>Function descriptions</td>
<td>get the flag of ADC regular channel software start conversion</td>
</tr>
<tr>
<td>Precondition</td>
<td>-</td>
</tr>
<tr>
<td>The called functions</td>
<td>-</td>
</tr>
<tr>
<td>Input parameter (in)</td>
<td>-</td>
</tr>
<tr>
<td>Output parameter (out)</td>
<td>-</td>
</tr>
<tr>
<td>Return value</td>
<td>FlagStatus SET or RESET</td>
</tr>
</tbody>
</table>

Example:

/* get the flag of ADC regular channel software start conversion */
FlagStatus flag_value;
flag_value = adc_regular_software_startconv_flag_get();

adc_inserted_software_startconv_flag_get

The description of adc_inserted_software_startconv_flag_get is shown as below:

<table>
<thead>
<tr>
<th>Function name</th>
<th>adc_inserted_software_startconv_flag_get</th>
</tr>
</thead>
<tbody>
<tr>
<td>Function prototype</td>
<td>FlagStatus adc_inserted_software_startconv_flag_get(void);</td>
</tr>
<tr>
<td>Function descriptions</td>
<td>get the flag of ADC inserted channel software start conversion</td>
</tr>
<tr>
<td>Precondition</td>
<td>-</td>
</tr>
<tr>
<td>The called functions</td>
<td>-</td>
</tr>
<tr>
<td>Input parameter (in)</td>
<td>-</td>
</tr>
<tr>
<td>Output parameter (out)</td>
<td>-</td>
</tr>
<tr>
<td>Return value</td>
<td>FlagStatus SET or RESET</td>
</tr>
</tbody>
</table>

Example:

/* get the flag of ADC inserted channel software start conversion */
FlagStatus flag_value;
flag_value = adc_inserted_software_startconv_flag_get();

**adc_flag_get**

The description of `adc_flag_get` is shown as below:

<table>
<thead>
<tr>
<th>Function name</th>
<th>adc_flag_get</th>
</tr>
</thead>
<tbody>
<tr>
<td>Function prototype</td>
<td>FlagStatus adc_flag_get(uint32_t flag);</td>
</tr>
<tr>
<td>Function descriptions</td>
<td>get ADC flag</td>
</tr>
<tr>
<td>Precondition</td>
<td>-</td>
</tr>
<tr>
<td>The called functions</td>
<td>-</td>
</tr>
</tbody>
</table>

**Input parameter(in)**

| flag | ADC flag |
| ADC_FLAG_WDE | analog watchdog event flag |
| ADC_FLAG_EOC | end of group conversion flag |
| ADC_FLAG_EOIC | end of inserted group conversion flag |
| ADC_FLAG_STIC | start flag of inserted channel group |
| ADC_FLAG_STRC | start flag of regular channel group |

**Output parameter(out)**

- |

**Return value**

FlagStatus SET or RESET

**Example:**

/* get the ADC analog watchdog flag*/

FlagStatus flag_value;

flag_value = adc_flag_get(ADC_FLAG_WDE);

**adc_flag_clear**

The description of `adc_flag_clear` is shown as below:

<table>
<thead>
<tr>
<th>Function name</th>
<th>adc_flag_clear</th>
</tr>
</thead>
<tbody>
<tr>
<td>Function prototype</td>
<td>void adc_flag_clear(uint32_t flag);</td>
</tr>
<tr>
<td>Function descriptions</td>
<td>clear ADC flag</td>
</tr>
<tr>
<td>Precondition</td>
<td>-</td>
</tr>
<tr>
<td>The called functions</td>
<td>-</td>
</tr>
</tbody>
</table>

**Input parameter(in)**

| flag | ADC flag |
| ADC_FLAG_WDE | analog watchdog event flag |
| ADC_FLAG_EOC | end of group conversion flag |
ADC_FLAG_EOIC  end of inserted group conversion flag
ADC_FLAG_STIC  start flag of inserted channel group
ADC_FLAG_STRC  start flag of regular channel group

Output parameter(out)
- -

Return value
- -

Example:

/* clear the ADC analog watchdog flag */
adc_flag_clear(ADC_FLAG_WDE);

adc_interrupt_enable

The description of adc_interrupt_enable is shown as below:

<table>
<thead>
<tr>
<th>Function name</th>
<th>adc_interrupt_enable</th>
</tr>
</thead>
<tbody>
<tr>
<td>Function prototype</td>
<td>void adc_interrupt_enable(uint32_t interrupt);</td>
</tr>
<tr>
<td>Function descriptions</td>
<td>enable ADC interrupt</td>
</tr>
<tr>
<td>Precondition</td>
<td>-</td>
</tr>
<tr>
<td>The called functions</td>
<td>-</td>
</tr>
</tbody>
</table>

Input parameter(in)

<table>
<thead>
<tr>
<th>interrupt</th>
<th>ADC interrupt</th>
</tr>
</thead>
<tbody>
<tr>
<td>ADC_INT_WDE</td>
<td>analog watchdog interrupt</td>
</tr>
<tr>
<td>ADC_INT_EOC</td>
<td>end of group conversion interrupt</td>
</tr>
<tr>
<td>ADC_INT_EOIC</td>
<td>end of inserted group conversion interrupt</td>
</tr>
</tbody>
</table>

Output parameter(out)
- -

Return value
- -

Example:

/* enable ADC analog watchdog interrupt */
adc_interrupt_enable(ADC_INT_WDE);

adc_interrupt_disable

The description of adc_interrupt_disable is shown as below:

<table>
<thead>
<tr>
<th>Function name</th>
<th>adc_interrupt_disable</th>
</tr>
</thead>
<tbody>
<tr>
<td>Function prototype</td>
<td>void adc_interrupt_disable(uint32_t interrupt);</td>
</tr>
</tbody>
</table>
Function descriptions: disable ADC interrupt

Precondition: -

The called functions: -

<table>
<thead>
<tr>
<th>Input parameter (in)</th>
<th>ADC interrupt</th>
</tr>
</thead>
<tbody>
<tr>
<td>ADC_INT_WDE</td>
<td>analog watchdog interrupt</td>
</tr>
<tr>
<td>ADC_INT_EOC</td>
<td>end of group conversion interrupt</td>
</tr>
<tr>
<td>ADC_INT_EOIC</td>
<td>end of inserted group conversion interrupt</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Output parameter (out)</th>
<th>-</th>
</tr>
</thead>
<tbody>
<tr>
<td>Return value</td>
<td>-</td>
</tr>
</tbody>
</table>

Example:

/* disable ADC analog watchdog interrupt */
adc_interrupt_disable(ADC_INT_WDE);

adc_interrupt_flag_get

The description of adc_interrupt_flag_get is shown as below:

<table>
<thead>
<tr>
<th>Function name</th>
<th>adc_interrupt_flag_get</th>
</tr>
</thead>
<tbody>
<tr>
<td>Function prototype</td>
<td>FlagStatus adc_interrupt_flag_get(uint32_t int_flag);</td>
</tr>
<tr>
<td>Function descriptions</td>
<td>get ADC interrupt</td>
</tr>
<tr>
<td>Precondition</td>
<td>-</td>
</tr>
<tr>
<td>The called functions</td>
<td>-</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Input parameter (in)</th>
<th>ADC interrupt flag</th>
</tr>
</thead>
<tbody>
<tr>
<td>ADC_INT_FLAG_WDE</td>
<td>analog watchdog interrupt flag</td>
</tr>
<tr>
<td>ADC_INT_FLAG_EOC</td>
<td>end of group conversion interrupt flag</td>
</tr>
<tr>
<td>ADC_INT_FLAG_EOIC</td>
<td>end of inserted group conversion interrupt flag</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Output parameter (out)</th>
<th>-</th>
</tr>
</thead>
<tbody>
<tr>
<td>Return value</td>
<td>FlagStatus</td>
</tr>
</tbody>
</table>

Example:

/* get the ADC analog watchdog interrupt flag */

FlagStatus flag_value;

flag_value = adc_interrupt_flag_get(ADC_INT_FLAG_WDE);
The description of `adc_interrupt_flag_clear` is shown as below:

### Table 3-40. Function `adc_interrupt_flag_clear`

<table>
<thead>
<tr>
<th>Function name</th>
<th>adc_interrupt_flag_clear</th>
</tr>
</thead>
<tbody>
<tr>
<td>Function prototype</td>
<td>void adc_interrupt_flag_clear(uint32_t int_flag);</td>
</tr>
<tr>
<td>Function descriptions</td>
<td>clear ADC interrupt</td>
</tr>
<tr>
<td>Precondition</td>
<td>-</td>
</tr>
<tr>
<td>The called functions</td>
<td>-</td>
</tr>
<tr>
<td>Input parameter(in)</td>
<td>int_flag</td>
</tr>
<tr>
<td></td>
<td>ADC interrupt flag</td>
</tr>
<tr>
<td></td>
<td><code>ADC_INT_FLAG_WDE</code> analog watchdog interrupt flag</td>
</tr>
<tr>
<td></td>
<td><code>ADC_INT_FLAG_EOC</code> end of group conversion interrupt flag</td>
</tr>
<tr>
<td></td>
<td><code>ADC_INT_FLAG_EOIC</code> end of inserted group conversion interrupt flag</td>
</tr>
<tr>
<td>Output parameter(out)</td>
<td>-</td>
</tr>
<tr>
<td></td>
<td>-</td>
</tr>
<tr>
<td>Return value</td>
<td>-</td>
</tr>
</tbody>
</table>

Example:

```c
/* clear the ADC analog watchdog interrupt flag */
adc_interrupt_flag_clear(ADC_INT_FLAG_WDE);
```

### 3.3. CAU

The Cryptographic Acceleration Unit supports acceleration of DES, Triple-DES or AES (128,192, or 256) algorithms. The CAU registers are listed in chapter 3.3.1 the CAU firmware functions are introduced in chapter 3.3.2

#### 3.3.1. Descriptions of Peripheral registers

CAU registers are listed in the table shown as below:

### Table 3-41. CAU Registers

<table>
<thead>
<tr>
<th>Registers</th>
<th>Descriptions</th>
</tr>
</thead>
<tbody>
<tr>
<td>CAU_CTL</td>
<td>control register</td>
</tr>
<tr>
<td>CAU_STAT0</td>
<td>status register 0</td>
</tr>
<tr>
<td>CAU_DI</td>
<td>data input register</td>
</tr>
<tr>
<td>CAU_DO</td>
<td>data output register</td>
</tr>
<tr>
<td>CAU_DMAEN</td>
<td>DMA enable register</td>
</tr>
<tr>
<td>CAU_INTEN</td>
<td>interrupt enable register</td>
</tr>
<tr>
<td>CAU_STAT1</td>
<td>status register 1</td>
</tr>
</tbody>
</table>
### Registers Descriptions

<table>
<thead>
<tr>
<th>Registers</th>
<th>Descriptions</th>
</tr>
</thead>
<tbody>
<tr>
<td>CAU_INTF</td>
<td>interrupt flag register</td>
</tr>
<tr>
<td>CAU_KEY0H</td>
<td>key 0 high register</td>
</tr>
<tr>
<td>CAU_KEY0L</td>
<td>key 0 low register</td>
</tr>
<tr>
<td>CAU_KEY1H</td>
<td>key 1 high register</td>
</tr>
<tr>
<td>CAU_KEY1L</td>
<td>key 1 low register</td>
</tr>
<tr>
<td>CAU_KEY2H</td>
<td>key 2 high register</td>
</tr>
<tr>
<td>CAU_KEY2L</td>
<td>key 2 low register</td>
</tr>
<tr>
<td>CAU_KEY3H</td>
<td>key 3 high register</td>
</tr>
<tr>
<td>CAU_KEY3L</td>
<td>key 3 low register</td>
</tr>
<tr>
<td>CAU_IV0H</td>
<td>initial vector 0 high register</td>
</tr>
<tr>
<td>CAU_IV0L</td>
<td>initial vector 0 low register</td>
</tr>
<tr>
<td>CAU_IV1H</td>
<td>initial vector 1 high register</td>
</tr>
<tr>
<td>CAU_IV1L</td>
<td>initial vector 1 low register</td>
</tr>
<tr>
<td>CAU_GCMCCMCTXSx</td>
<td>GCM or CCM mode context switch register x (x = 0..7)</td>
</tr>
<tr>
<td>CAU_GCMCTXSX</td>
<td>GCM mode context switch register x (x = 0..7)</td>
</tr>
</tbody>
</table>

### 3.3.2. Descriptions of Peripheral functions

CAU firmware functions are listed in the table shown as below:

<table>
<thead>
<tr>
<th>Function name</th>
<th>Function description</th>
</tr>
</thead>
<tbody>
<tr>
<td>cau_deinit</td>
<td>reset the CAU peripheral</td>
</tr>
<tr>
<td>cau_struct_para_init</td>
<td>initialize the CAU encrypt and decrypt parameter struct with the default values</td>
</tr>
<tr>
<td>cau_key_struct_para_init</td>
<td>initialize the key parameter struct with the default values</td>
</tr>
<tr>
<td>cau_iv_struct_para_init</td>
<td>initialize the vectors parameter struct with the default values</td>
</tr>
<tr>
<td>cau_context_struct_para_init</td>
<td>initialize the context parameter struct with the default values</td>
</tr>
<tr>
<td>cau_enable</td>
<td>enable the CAU peripheral</td>
</tr>
<tr>
<td>cau_disable</td>
<td>disable the CAU peripheral</td>
</tr>
<tr>
<td>cau_dma_enable</td>
<td>enable the CAU DMA interface</td>
</tr>
<tr>
<td>cau_dma_disable</td>
<td>disable the CAU DMA interface</td>
</tr>
<tr>
<td>cau_init</td>
<td>initialize the CAU peripheral</td>
</tr>
<tr>
<td>cau_aes_keysize_config</td>
<td>configure key size if use AES algorithm</td>
</tr>
<tr>
<td>cau_key_init</td>
<td>initialize the key parameters</td>
</tr>
<tr>
<td>cau_iv_init</td>
<td>initialize the vectors parameters</td>
</tr>
<tr>
<td>cau_phase_config</td>
<td>configure phase</td>
</tr>
<tr>
<td>cau_fifo_flush</td>
<td>flush the IN and OUT FIFOs</td>
</tr>
<tr>
<td>cau_enable_state_get</td>
<td>return whether CAU peripheral is enabled or disabled</td>
</tr>
<tr>
<td>cau_data_write</td>
<td>write data to the IN FIFO</td>
</tr>
<tr>
<td>Function name</td>
<td>Function description</td>
</tr>
<tr>
<td>-------------------</td>
<td>----------------------------------------------------------</td>
</tr>
<tr>
<td>cau_data_read</td>
<td>return the last data entered into the output FIFO</td>
</tr>
<tr>
<td>cau_context_save</td>
<td>save context before context switching</td>
</tr>
<tr>
<td>cau_context_restore</td>
<td>restore context after context switching</td>
</tr>
<tr>
<td>cau_aes_ecb</td>
<td>encrypt and decrypt using AES in ECB mode</td>
</tr>
<tr>
<td>cau_aes_cbc</td>
<td>encrypt and decrypt using AES in CBC mode</td>
</tr>
<tr>
<td>cau_aes_ctr</td>
<td>encrypt and decrypt using AES in CTR mode</td>
</tr>
<tr>
<td>cau_aes_cfb</td>
<td>encrypt and decrypt using AES in CFB mode</td>
</tr>
<tr>
<td>cau_aes_ofb</td>
<td>encrypt and decrypt using AES in OFB mode</td>
</tr>
<tr>
<td>cau_aes_gcm</td>
<td>encrypt and decrypt using AES in GCM mode</td>
</tr>
<tr>
<td>cau_aes_ccm</td>
<td>encrypt and decrypt using AES in CCM mode</td>
</tr>
<tr>
<td>cau_tdes_ecb</td>
<td>encrypt and decrypt using TDES in ECB mode</td>
</tr>
<tr>
<td>cau_tdes_cbc</td>
<td>encrypt and decrypt using TDES in CBC mode</td>
</tr>
<tr>
<td>cau_des_ecb</td>
<td>encrypt and decrypt using DES in ECB mode</td>
</tr>
<tr>
<td>cau_des_cbc</td>
<td>encrypt and decrypt using DES in CBC mode</td>
</tr>
<tr>
<td>cau_interrupt_enable</td>
<td>enable the CAU interrupts</td>
</tr>
<tr>
<td>cau_interrupt_disable</td>
<td>disable the CAU interrupts</td>
</tr>
<tr>
<td>cau_interrupt_flag_get</td>
<td>get the interrupt flag</td>
</tr>
<tr>
<td>cau_flag_get</td>
<td>get the CAU flag status</td>
</tr>
</tbody>
</table>

**Structure cau_key_parameter_struct**

**Table 3-43. Structure cau_key_parameter_struct**

<table>
<thead>
<tr>
<th>Member name</th>
<th>Function description</th>
</tr>
</thead>
<tbody>
<tr>
<td>key_0_high</td>
<td>key 0 high</td>
</tr>
<tr>
<td>key_0_low</td>
<td>key 0 low</td>
</tr>
<tr>
<td>key_1_high</td>
<td>key 1 high</td>
</tr>
<tr>
<td>key_1_low</td>
<td>key 1 low</td>
</tr>
<tr>
<td>key_2_high</td>
<td>key 2 high</td>
</tr>
<tr>
<td>key_2_low</td>
<td>key 2 low</td>
</tr>
<tr>
<td>key_3_high</td>
<td>key 3 high</td>
</tr>
<tr>
<td>key_3_low</td>
<td>key 3 low</td>
</tr>
</tbody>
</table>

**Structure cau_iv_parameter_struct**

**Table 3-44. Structure cau_iv_parameter_struct**

<table>
<thead>
<tr>
<th>Member name</th>
<th>Function description</th>
</tr>
</thead>
<tbody>
<tr>
<td>iv_0_high</td>
<td>init vector 0 high</td>
</tr>
<tr>
<td>iv_0_low</td>
<td>init vector 0 low</td>
</tr>
<tr>
<td>iv_1_high</td>
<td>init vector 1 high</td>
</tr>
<tr>
<td>iv_1_low</td>
<td>init vector 1 low</td>
</tr>
</tbody>
</table>
Structure `cau_context_parameter_struct`

Table 3-45. Structure `cau_context_parameter_struct`

<table>
<thead>
<tr>
<th>Member name</th>
<th>Function description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ctl_config</td>
<td>current configuration</td>
</tr>
<tr>
<td>iv_0_high</td>
<td>init vector 0 high</td>
</tr>
<tr>
<td>iv_0_low</td>
<td>init vector 0 low</td>
</tr>
<tr>
<td>iv_1_high</td>
<td>init vector 1 high</td>
</tr>
<tr>
<td>iv_1_low</td>
<td>init vector 1 low</td>
</tr>
<tr>
<td>key_0_high</td>
<td>key 0 high</td>
</tr>
<tr>
<td>key_0_low</td>
<td>key 0 low</td>
</tr>
<tr>
<td>key_1_high</td>
<td>key 1 high</td>
</tr>
<tr>
<td>key_1_low</td>
<td>key 1 low</td>
</tr>
<tr>
<td>key_2_high</td>
<td>key 2 high</td>
</tr>
<tr>
<td>key_2_low</td>
<td>key 2 low</td>
</tr>
<tr>
<td>key_3_high</td>
<td>key 3 high</td>
</tr>
<tr>
<td>key_3_low</td>
<td>key 3 low</td>
</tr>
<tr>
<td>gcmccmctxs[8]</td>
<td>GCM or CCM mode context switch</td>
</tr>
<tr>
<td>gcmctxs[8]</td>
<td>GCM mode context switch</td>
</tr>
</tbody>
</table>

Structure `cau_parameter_struct`

Table 3-46. Structure `cau_parameter_struct`

<table>
<thead>
<tr>
<th>Member name</th>
<th>Function description</th>
</tr>
</thead>
<tbody>
<tr>
<td>alg_dir</td>
<td>algorithm directory</td>
</tr>
<tr>
<td>*key</td>
<td>key</td>
</tr>
<tr>
<td>key_size</td>
<td>key size in bytes</td>
</tr>
<tr>
<td>*iv</td>
<td>initialization vector</td>
</tr>
<tr>
<td>iv_size</td>
<td>iv size in bytes</td>
</tr>
<tr>
<td>*input</td>
<td>input data</td>
</tr>
<tr>
<td>in_length</td>
<td>input data length in bytes</td>
</tr>
<tr>
<td>*aad</td>
<td>additional authentication data</td>
</tr>
<tr>
<td>aad_size</td>
<td>aad size</td>
</tr>
</tbody>
</table>

cau_deinit

The description of `cau_deinit` is shown as below:

Table 3-47. Function `cau_deinit`

<table>
<thead>
<tr>
<th>Function name</th>
<th>cau_deinit</th>
</tr>
</thead>
<tbody>
<tr>
<td>Function prototype</td>
<td>void cau_deinit(void);</td>
</tr>
<tr>
<td>Function descriptions</td>
<td>reset the CAU peripheral</td>
</tr>
<tr>
<td>Precondition</td>
<td>-</td>
</tr>
<tr>
<td>The called functions</td>
<td>rcu_periph_reset_enable/rcu_periph_reset_disable/</td>
</tr>
</tbody>
</table>
Example:

 /* reset the CAU peripheral */
 cau_deinit();

cau_struct_para_init

The description of cau_struct_para_init is shown as below:

Table 3-48. Function cau_struct_para_init

<table>
<thead>
<tr>
<th>Function name</th>
<th>cau_struct_para_init</th>
</tr>
</thead>
<tbody>
<tr>
<td>Function prototype</td>
<td>void cau_struct_para_init(cau_parameter_struct *cau_parameter);</td>
</tr>
<tr>
<td>Function descriptions</td>
<td>initialize the CAU encrypt and decrypt parameter struct with the default values</td>
</tr>
<tr>
<td>Precondition</td>
<td>-</td>
</tr>
<tr>
<td>The called functions</td>
<td>-</td>
</tr>
<tr>
<td>Input parameter(in)</td>
<td>-</td>
</tr>
<tr>
<td>Output parameter(out)</td>
<td>-</td>
</tr>
<tr>
<td>cau_parameter</td>
<td>structure for encrypt and decrypt parameters, refer to structure Table 3-46. Structure cau_parameter_struct</td>
</tr>
<tr>
<td>Return value</td>
<td>-</td>
</tr>
</tbody>
</table>

Example:

cau_parameter_struct text;

 /* initialize CAU encrypt and decrypt parameter struct */
 cau_struct_para_init(&text);

cau_key_struct_para_init

The description of cau_key_struct_para_init is shown as below:

Table 3-49. Function cau_key_struct_para_init

<table>
<thead>
<tr>
<th>Function name</th>
<th>cau_key_struct_para_init</th>
</tr>
</thead>
<tbody>
<tr>
<td>Function prototype</td>
<td>void cau_key_struct_para_init(cau_key_parameter_struct *key_initpara);</td>
</tr>
</tbody>
</table>

Example:

cau_key_parameter_struct key;

 /* initialize CAU encrypt and decrypt key struct */
 cau_key_struct_para_init(&key);
Function descriptions: initialize the key parameter struct with the default values

Precondition: -

The called functions: -

Input parameter (in): -

Output parameter (out):

key_initpara: structure for keys initialization of the cau, refer to structure Table 3-43. Structure cau_key_parameter_struct

Return value: -

Example:

/* initialize the key parameter struct */
cau_key_parameter_struct key_initpara;
cau_key_struct_para_init(&key_initpara);

cau_iv_struct_para_init

The description of cau_iv_struct_para_init is shown as below:

Table 3-50. Function cau_iv_struct_para_init

<table>
<thead>
<tr>
<th>Function name</th>
<th>cau_iv_struct_para_init</th>
</tr>
</thead>
<tbody>
<tr>
<td>Function prototype</td>
<td>void cau_iv_struct_para_init(cau_iv_parameter_struct *iv_initpara);</td>
</tr>
<tr>
<td>Function descriptions</td>
<td>initialize the vectors parameter struct with the default values</td>
</tr>
<tr>
<td>Precondition</td>
<td>-</td>
</tr>
<tr>
<td>The called functions</td>
<td>-</td>
</tr>
<tr>
<td>Input parameter (in)</td>
<td>-</td>
</tr>
<tr>
<td>Output parameter (out)</td>
<td>iv_initpara: structure for vectors initialization of the cau, refer to structure Table 3-44. Structure cau_iv_parameter_struct</td>
</tr>
<tr>
<td>Return value</td>
<td>-</td>
</tr>
</tbody>
</table>

Example:

/* initialize the vectors parameter struct */
cau_iv_parameter_struct iv_initpara;
cau_iv_struct_para_init(&iv_initpara);

cau_context_struct_para_init

The description of cau_context_struct_para_init is shown as below:
Table 3-51. Function cau_context_struct_para_init

<table>
<thead>
<tr>
<th>Function name</th>
<th>cau_context_struct_para_init</th>
</tr>
</thead>
<tbody>
<tr>
<td>Function prototype</td>
<td>void cau_context_struct_para_init(cau_context_parameter_struct *cau_context);</td>
</tr>
<tr>
<td>Function descriptions</td>
<td>initialize the context parameter struct with the default values</td>
</tr>
<tr>
<td>Precondition</td>
<td>-</td>
</tr>
<tr>
<td>The called functions</td>
<td>-</td>
</tr>
<tr>
<td>Input parameter(in)</td>
<td>-</td>
</tr>
<tr>
<td>Output parameter(out)</td>
<td>-</td>
</tr>
<tr>
<td>cau_context</td>
<td>structure for cau context swapping, refer to structure Table 3-45. Structure cau_context_parameter_struct</td>
</tr>
<tr>
<td>Return value</td>
<td>-</td>
</tr>
</tbody>
</table>

Example:
/* initialize the context parameter struct */
cau_context_parameter_struct context_initpara;
cau_context_struct_para_init(&context_initpara);

**cau_enable**

The description of cau_enable is shown as below:

Table 3-52. Function cau_enable

<table>
<thead>
<tr>
<th>Function name</th>
<th>cau_enable</th>
</tr>
</thead>
<tbody>
<tr>
<td>Function prototype</td>
<td>void cau_enable(void);</td>
</tr>
<tr>
<td>Function descriptions</td>
<td>enable the CAU peripheral</td>
</tr>
<tr>
<td>Precondition</td>
<td>-</td>
</tr>
<tr>
<td>The called functions</td>
<td>-</td>
</tr>
<tr>
<td>Input parameter(in)</td>
<td>-</td>
</tr>
<tr>
<td>Output parameter(out)</td>
<td>-</td>
</tr>
<tr>
<td>Return value</td>
<td>-</td>
</tr>
</tbody>
</table>

Example:
/* enable the CAU peripheral */
cau_enable();
cau_disable

The description of cau_disable is shown as below:

<table>
<thead>
<tr>
<th>Table 3-53. Function cau_disable</th>
</tr>
</thead>
<tbody>
<tr>
<td>Function name</td>
</tr>
<tr>
<td>Function prototype</td>
</tr>
<tr>
<td>Function descriptions</td>
</tr>
<tr>
<td>Precondition</td>
</tr>
<tr>
<td>The called functions</td>
</tr>
<tr>
<td>Input parameter(in)</td>
</tr>
<tr>
<td>Output parameter(out)</td>
</tr>
<tr>
<td>Return value</td>
</tr>
</tbody>
</table>

Example:

`/* disable the CAU peripheral */`

cau_disable();

cau_dma_enable

The description of cau_dma_enable is shown as below:

<table>
<thead>
<tr>
<th>Table 3-54. Function cau_dma_enable</th>
</tr>
</thead>
<tbody>
<tr>
<td>Function name</td>
</tr>
<tr>
<td>Function prototype</td>
</tr>
<tr>
<td>Function descriptions</td>
</tr>
<tr>
<td>Precondition</td>
</tr>
<tr>
<td>The called functions</td>
</tr>
<tr>
<td>dma_req</td>
</tr>
<tr>
<td>CAU_DMA_INFIFO</td>
</tr>
<tr>
<td>CAU_DMA_OUTFIFO</td>
</tr>
<tr>
<td>Output parameter(out)</td>
</tr>
<tr>
<td>Return value</td>
</tr>
</tbody>
</table>

Example:

`/* enable the CAU DMA interface */`

cau_dma_enable(CAU_DMA_INFIFO);
### cau_dma_disable

The description of cau_dma_disable is shown as below:

#### Table 3-55. Function cau_dma_disable

<table>
<thead>
<tr>
<th>Function name</th>
<th>cau_dma_disable</th>
</tr>
</thead>
<tbody>
<tr>
<td>Function prototype</td>
<td>void cau_dma_disable(uint32_t dma_req);</td>
</tr>
<tr>
<td>Function descriptions</td>
<td>disable the CAU DMA interface</td>
</tr>
<tr>
<td>Precondition</td>
<td>-</td>
</tr>
<tr>
<td>The called functions</td>
<td>-</td>
</tr>
</tbody>
</table>

**Input parameter (in)**

- `dma_req` specify the CAU DMA transfer request to be disabled
- `CAU_DMA_INFIFO` DMA for incoming (Rx) data transfer
- `CAU_DMA_OUTFIFO` DMA for outgoing (Tx) data transfer

**Output parameter (out)**

- -

**Return value**

- -

Example:

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

### cau_init

The description of cau_init is shown as below:

#### Table 3-56. Function cau_init

<table>
<thead>
<tr>
<th>Function name</th>
<th>cau_init</th>
</tr>
</thead>
<tbody>
<tr>
<td>Function prototype</td>
<td>void cau_init(uint32_t algo_dir, uint32_t algo_mode, uint32_t swapping);</td>
</tr>
<tr>
<td>Function descriptions</td>
<td>initialize the CAU peripheral</td>
</tr>
<tr>
<td>Precondition</td>
<td>-</td>
</tr>
<tr>
<td>The called functions</td>
<td>-</td>
</tr>
</tbody>
</table>

**Input parameter (in)**

- `algo_dir` algorithm direction
- `CAU_ENCRYPT` encrypt
- `CAU_DECRYPT` decrypt

**Input parameter (in)**

- `algo_mode` algorithm mode selection
- `CAU_MODE_TDES_ECB` TDES-ECB (3DES Electronic codebook)
- `CAU_MODE_TDES_CBC` TDES-CBC (3DES Cipher block chaining)
- `CAU_MODE_DES_EC` DES-ECB (simple DES Electronic codebook)
<table>
<thead>
<tr>
<th>Input parameter (in)</th>
<th>Output parameter (out)</th>
<th>Return value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Input parameter (in)</td>
<td></td>
<td>Return value</td>
</tr>
<tr>
<td>swapping</td>
<td></td>
<td>Return value</td>
</tr>
<tr>
<td>CAU_SWAPPING_32BIT</td>
<td>no swapping</td>
<td>Return value</td>
</tr>
<tr>
<td>CAU_SWAPPING_16BIT</td>
<td>half-word swapping</td>
<td>Return value</td>
</tr>
<tr>
<td>CAU_SWAPPING_8BIT</td>
<td>bytes swapping</td>
<td>Return value</td>
</tr>
<tr>
<td>CAU_SWAPPING_1BIT</td>
<td>bit swapping</td>
<td>Return value</td>
</tr>
</tbody>
</table>

Example:

/* initialize the CAU peripheral */
cau_init(CAU_ENCRYPT, CAU_MODE_TDES_ECB, CAU_SWAPPING_32BIT);

**cau_aes_keysize_config**

The description of cau_aes_keysize_config is shown as below:

**Table 3-57. Function cau_aes_keysize_config**

<table>
<thead>
<tr>
<th>Function name</th>
<th>cau_aes_keysize_config</th>
</tr>
</thead>
<tbody>
<tr>
<td>Function prototype</td>
<td>void cau_aes_keysize_config(uint32_t key_size);</td>
</tr>
</tbody>
</table>
Function descriptions: configure key size if used AES algorithm

Precondition: 

The called functions: 

Input parameter (in)

<table>
<thead>
<tr>
<th>key_size</th>
<th>key length selection when aes mode</th>
</tr>
</thead>
<tbody>
<tr>
<td>CAU_KEYSIZE_128BIT</td>
<td>128 bit key length</td>
</tr>
<tr>
<td>CAU_KEYSIZE_192BIT</td>
<td>192 bit key length</td>
</tr>
<tr>
<td>CAU_KEYSIZE_256BIT</td>
<td>256 bit key length</td>
</tr>
</tbody>
</table>

Output parameter (out): 

Return value: 

Example:

/* configure key size if used AES algorithm */

cau_aes_keysize_config(CAU_KEYSIZE_128BIT);

cau_key_init

The description of cau_key_init is shown as below:

Table 3-58. Function cau_key_init

<table>
<thead>
<tr>
<th>Function name</th>
<th>cau_key_init</th>
</tr>
</thead>
<tbody>
<tr>
<td>Function prototype</td>
<td>void cau_key_init(cau_key_parameter_struct* key_initpara);</td>
</tr>
<tr>
<td>Function descriptions</td>
<td>initialize the key parameters</td>
</tr>
<tr>
<td>Precondition:</td>
<td>-</td>
</tr>
<tr>
<td>The called functions:</td>
<td>-</td>
</tr>
</tbody>
</table>

Input parameter (in)

key_initpara: structure for keys initialization of the cau, refer to structure Table 3-43. Structure cau_key_parameter_struct

Output parameter (out): 

Return value: 

Example:

/* initialize the key parameters */

cau_key_parameter_struct key_initpara;

key_initpara->key_0_high = 0x12345678;

key_initpara->key_0_low = 0x12345678;

key_initpara->key_1_high = 0x12345678;
key_initpara->key_1_low = 0x12345678;
key_initpara->key_2_high = 0x12345678;
key_initpara->key_2_low = 0x12345678;
key_initpara->key_3_high = 0x12345678;
key_initpara->key_4_low = 0x12345678;
cau_key_init(&key_initpara);

cau_iv_init

The description of cau_iv_init is shown as below:

Table 3-59. Function cau_iv_init

<table>
<thead>
<tr>
<th>Function name</th>
<th>cau_iv_init</th>
</tr>
</thead>
<tbody>
<tr>
<td>Function prototype</td>
<td>void cau_iv_init(cau_iv_parameter_struct* iv_initpara);</td>
</tr>
<tr>
<td>Function descriptions</td>
<td>initialize the vectors parameters</td>
</tr>
<tr>
<td>Precondition</td>
<td>-</td>
</tr>
<tr>
<td>The called functions</td>
<td>-</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Input parameter(in)</th>
</tr>
</thead>
<tbody>
<tr>
<td>iv_initpara</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Output parameter(out)</th>
</tr>
</thead>
<tbody>
<tr>
<td>-</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Return value</th>
</tr>
</thead>
<tbody>
<tr>
<td>-</td>
</tr>
</tbody>
</table>

Example:

/* initialize the vectors parameters */
cau_iv_parameter_struct iv_initpara;
iv_initpara->iv_0_high = 0x12345678;
iv_initpara->iv_0_low = 0x12345678;
iv_initpara->iv_1_high = 0x12345678;
iv_initpara->iv_1_low = 0x12345678;
cau_iv_init(&iv_initpara);

cau_phase_config

The description of cau_phase_config is shown as below:
Table 3-60. Function cau_phase_config

<table>
<thead>
<tr>
<th>Function name</th>
<th>cau_phase_config</th>
</tr>
</thead>
<tbody>
<tr>
<td>Function prototype</td>
<td>void cau_phase_config(uint32_t phase);</td>
</tr>
<tr>
<td>Function descriptions</td>
<td>configure phase</td>
</tr>
<tr>
<td>Precondition</td>
<td>-</td>
</tr>
<tr>
<td>The called functions</td>
<td>-</td>
</tr>
</tbody>
</table>

**Input parameter (in)**

<table>
<thead>
<tr>
<th>phase</th>
<th>gcm or ccm phase</th>
</tr>
</thead>
<tbody>
<tr>
<td>CAU_PREPARE_PHASE</td>
<td>prepare phase</td>
</tr>
<tr>
<td>CAU_AAD_PHASE</td>
<td>AAD phase</td>
</tr>
<tr>
<td>CAU_ENCRYPT_DECRYPT_PHASE</td>
<td>encryption/decryption phase</td>
</tr>
<tr>
<td>CAU_TAG_PHASE</td>
<td>tag phase</td>
</tr>
</tbody>
</table>

**Output parameter (out)**

- -

**Return value**

- -

Example:

```c
/* select prepare phase */
cau_phase_config(CAU_PREPARE_PHASE);
```

### cau_fifo_flush

The description of cau_fifo_flush is shown as below:

Table 3-61. Function cau_fifo_flush

<table>
<thead>
<tr>
<th>Function name</th>
<th>cau_fifo_flush</th>
</tr>
</thead>
<tbody>
<tr>
<td>Function prototype</td>
<td>void cau_fifo_flush(void);</td>
</tr>
<tr>
<td>Function descriptions</td>
<td>flush the IN and OUT FIFOs</td>
</tr>
<tr>
<td>Precondition</td>
<td>-</td>
</tr>
<tr>
<td>The called functions</td>
<td>-</td>
</tr>
</tbody>
</table>

**Input parameter (in)**

- -

**Output parameter (out)**

- -

**Return value**

- -

Example:

```c
/* flush the IN and OUT FIFOs */
cau_fifo_flush();
```
**cau_enable_state_get**

The description of cau_enable_state_get is shown as below:

<table>
<thead>
<tr>
<th>Function name</th>
<th>cau_enable_state_get</th>
</tr>
</thead>
<tbody>
<tr>
<td>Function prototype</td>
<td>ControlStatus cau_enable_state_get(void);</td>
</tr>
<tr>
<td>Function descriptions</td>
<td>return whether CAU peripheral is enabled or disabled</td>
</tr>
<tr>
<td>Precondition</td>
<td>-</td>
</tr>
<tr>
<td>The called functions</td>
<td>-</td>
</tr>
<tr>
<td>Input parameter(in)</td>
<td>-</td>
</tr>
<tr>
<td>Output parameter(out)</td>
<td>-</td>
</tr>
<tr>
<td>Return value</td>
<td>ControlStatus ENABLE or DISABLE</td>
</tr>
</tbody>
</table>

Example:

```c
/* return whether CAU peripheral is enabled or disabled */
ControlStatus state = cau_enable_state_get();
```

**cau_data_write**

The description of cau_data_write is shown as below:

<table>
<thead>
<tr>
<th>Function name</th>
<th>cau_data_write</th>
</tr>
</thead>
<tbody>
<tr>
<td>Function prototype</td>
<td>void cau_data_write(uint32_t data);</td>
</tr>
<tr>
<td>Function descriptions</td>
<td>write data to the IN FIFO</td>
</tr>
<tr>
<td>Precondition</td>
<td>-</td>
</tr>
<tr>
<td>The called functions</td>
<td>-</td>
</tr>
<tr>
<td>Input parameter(in)</td>
<td>data</td>
</tr>
<tr>
<td>Output parameter(out)</td>
<td>-</td>
</tr>
<tr>
<td>Return value</td>
<td>-</td>
</tr>
</tbody>
</table>

Example:

```c
/* write data to the IN FIFO */
cau_data_write(0x10);
```
### cau_data_read

The description of cau_data_read is shown as below:

**Table 3-64. Function cau_data_read**

<table>
<thead>
<tr>
<th>Function name</th>
<th>cau_data_read</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Function prototype</strong></td>
<td>uint32_t cau_data_read(void);</td>
</tr>
<tr>
<td><strong>Function descriptions</strong></td>
<td>return the last data entered into the output FIFO</td>
</tr>
<tr>
<td><strong>Precondition</strong></td>
<td>-</td>
</tr>
<tr>
<td><strong>The called functions</strong></td>
<td>-</td>
</tr>
<tr>
<td><strong>Input parameter(in)</strong></td>
<td>-</td>
</tr>
<tr>
<td><strong>Output parameter(out)</strong></td>
<td>-</td>
</tr>
<tr>
<td><strong>Return value</strong></td>
<td>-</td>
</tr>
</tbody>
</table>

**Example:**

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

uint32_t data;

data = cau_data_read();
```

### cau_context_save

The description of cau_context_save is shown as below:

**Table 3-65. Function cau_context_save**

<table>
<thead>
<tr>
<th>Function name</th>
<th>cau_context_save</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Function prototype</strong></td>
<td>void cau_context_save(cau_context_parameter_struct <em>cau_context, cau_key_parameter_struct</em> key_initpara);</td>
</tr>
<tr>
<td><strong>Function descriptions</strong></td>
<td>save context before context switching</td>
</tr>
<tr>
<td><strong>Precondition</strong></td>
<td>-</td>
</tr>
<tr>
<td><strong>The called functions</strong></td>
<td>-</td>
</tr>
<tr>
<td><strong>Input parameter(in)</strong></td>
<td>-</td>
</tr>
<tr>
<td><strong>key_initpara</strong></td>
<td>structure for keys initialization of the cau, refer to structure Table 3-43. Structure cau_key_parameter_struct</td>
</tr>
<tr>
<td><strong>Output parameter(out)</strong></td>
<td>-</td>
</tr>
<tr>
<td><strong>cau_context</strong></td>
<td>structure for cau context swapping, refer to structure Table 3-45. Structure cau_context_parameter_struct</td>
</tr>
<tr>
<td><strong>Return value</strong></td>
<td>-</td>
</tr>
</tbody>
</table>

**Example:**
cau_context_parameter_struct context;
cau_key_parameter_struct key;
cau_parameter_struct cau_parameter;
unsigned int keyaddr;

......
keyaddr = (unsigned int)(cau_parameter->key);
cau_key_struct_para_init(&key);
k.key.key_1_high = __REV(*(unsigned int *)(keyaddr));
keyaddr += 4U;
k.key.key_1_low= __REV(*(unsigned int *)(keyaddr));
/* save context before context switching */
cau_context_save(&context, &key);

cau_context_restore

The description of cau_context_restore is shown as below:

Table 3-66. Function cau_context_restore

<table>
<thead>
<tr>
<th>Function name</th>
<th>cau_context_restore</th>
</tr>
</thead>
<tbody>
<tr>
<td>Function prototype</td>
<td>void cau_context_restore(cau_context_parameter_struct *cau_context);</td>
</tr>
<tr>
<td>Function descriptions</td>
<td>restore context after context switching</td>
</tr>
<tr>
<td>Precondition</td>
<td>-</td>
</tr>
<tr>
<td>The called functions</td>
<td>-</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Input parameter(in)</th>
</tr>
</thead>
<tbody>
<tr>
<td>cau_context</td>
</tr>
<tr>
<td>structure for cau context swapping, refer to structure Table 3-45. Structure cau_context_parameter_struct</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Output parameter(out)</th>
</tr>
</thead>
<tbody>
<tr>
<td>-</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Return value</th>
</tr>
</thead>
<tbody>
<tr>
<td>-</td>
</tr>
</tbody>
</table>

Example:

cau_context_parameter_struct context;

......

cau_context_save(&context, &key);

......

/* restore context after context switching */
cau_context_restore(&context);

**cau_aes_ecb**

The description of cau_aes_ecb is shown as below:

**Table 3-67. Function cau_aes_ecb**

<table>
<thead>
<tr>
<th>Function name</th>
<th>cau_aes_ecb</th>
</tr>
</thead>
<tbody>
<tr>
<td>Function prototype</td>
<td>ErrStatus cau_aes_ecb(cau_parameter_struct *cau_parameter, uint8_t *output);</td>
</tr>
<tr>
<td>Function descriptions</td>
<td>encrypt and decrypt using AES in ECB mode</td>
</tr>
<tr>
<td>Precondition</td>
<td>-</td>
</tr>
<tr>
<td>The called functions</td>
<td>-</td>
</tr>
<tr>
<td>Input parameter(in)</td>
<td>cau_parameter - structure for encrypt and decrypt parameters, refer to structure <strong>Table 3-46. Structure cau_parameter_struct</strong></td>
</tr>
<tr>
<td>Output parameter(out)</td>
<td>output - pointer to the returned buffer</td>
</tr>
<tr>
<td>Return value</td>
<td>ErrStatus - SUCCESS or ERROR</td>
</tr>
</tbody>
</table>

Example:

cau_parameter_struct text;

uint8_t encrypt_result[TEXT_SIZE];

ErrStatus status;

......

key_addr = key_select[i];

key_size = keysize[i];

text.alg_dir = CAU_ENCRYPT;

text.key = key_addr;

text.key_size = key_size;

text.input = plaintext;

text.in_length = TEXT_SIZE;

"/* encryption in ECB mode */

status = cau_aes_ecb(&text, encrypt_result);

**cau_aes_cbc**

The description of cau_aes_cbc is shown as below:
Table 3-68. Function cau_aes_cbc

<table>
<thead>
<tr>
<th>Function name</th>
<th>cau_aes_cbc</th>
</tr>
</thead>
<tbody>
<tr>
<td>Function prototype</td>
<td>ErrStatus cau_aes_cbc(cau_parameter_struct *cau_parameter, uint8_t *output);</td>
</tr>
<tr>
<td>Function descriptions</td>
<td>encrypt and decrypt using AES in CBC mode</td>
</tr>
<tr>
<td>Precondition</td>
<td>-</td>
</tr>
<tr>
<td>The called functions</td>
<td>-</td>
</tr>
</tbody>
</table>

Input parameter(in)

- cau_parameter: structure for encrypt and decrypt parameters, refer to structure Table 3-46.

Output parameter(out)

- output: pointer to the returned buffer

Return value

- ErrStatus: SUCCESS or ERROR

Example:

```c
cau_parameter_struct text;
uint8_t encrypt_result[TEXT_SIZE];
ErrStatus status;
......
key_addr = key_select[i];
key_size = keysize[i];
text.alg_dir = CAU_ENCRYPT;
text.key = key_addr;
text.key_size = key_size;
text.iv = vectors;
text.input = plaintext;
text.in_length = TEXT_SIZE;
/* encryption in CBC mode */
status = cau_aes_cbc(&text, encrypt_result);
```

cau_aes_ctr

The description of cau_aes_ctr is shown as below:

Table 3-69. Function cau_aes_ctr

<table>
<thead>
<tr>
<th>Function name</th>
<th>cau_aes_ctr</th>
</tr>
</thead>
</table>
Function prototype

| Function prototype | ErrStatus cau_aes_ctr(cau_parameter_struct *cau_parameter, uint8_t *output); |

Function descriptions

encrypt and decrypt using AES in CTR mode

Precondition

-

The called functions

-

Input parameter (in)

| cau_parameter | structure for encrypt and decrypt parameters, refer to structure Table 3-46. |

Structure cau_parameter_struct

Output parameter (out)

| output | pointer to the returned buffer |

Return value

| ErrStatus | SUCCESS or ERROR |

Example:

cau_parameter_struct text;
uint8_t encrypt_result[TEXT_SIZE];
ErrStatus status;

......

key_addr = key_select[i];
key_size = keysize[i];
text.alg_dir = CAU_ENCRYPT;
text.key = key_addr;
text.key_size = key_size;
text.iv = vectors;
text.input = plaintext;
text.in_length = TEXT_SIZE;

/* encryption in CTR mode */
status = cau_aes_ctr(&text, encrypt_result);

cau_aes_cfb

The description of cau_aes_cfb is shown as below:

Table 3-70. Function cau_aes_cfb

<table>
<thead>
<tr>
<th>Function name</th>
<th>cau_aes_cfb</th>
</tr>
</thead>
<tbody>
<tr>
<td>Function prototype</td>
<td>ErrStatus cau_aes_cfb(cau_parameter_struct *cau_parameter, uint8_t *output);</td>
</tr>
</tbody>
</table>
Function descriptions | encrypt and decrypt using AES in CFB mode
---|---
Precondition | -
The called functions | -

### Input parameter(in)
- **cau_parameter**: structure for encrypt and decrypt parameters, refer to structure Table 3-46.

### Output parameter(out)
- **output**: pointer to the returned buffer

### Return value
- **ErrStatus**: SUCCESS or ERROR

Example:

```c
cau_parameter_struct cau_cfb_parameter;
uint8_t encrypt_result[TEXT_SIZE];
ErrStatus status;

......

/* encryption in CFB mode */
cau_cfb_parameter.alg_dir = CAU_ENCRYPT;
cau_cfb_parameter.key = (uint8_t *)key_128;
cau_cfb_parameter.key_size = KEY_SIZE;
cau_cfb_parameter.iv = (uint8_t *)vectors;
cau_cfb_parameter.iv_size = IV_SIZE;
cau_cfb_parameter.input = (uint8_t *)plaintext;
status = cau_aes_cfb(&cau_cfb_parameter, encrypt_result);
```

### cau_aes_ofb

The description of `cau_aes_ofb` is shown as below:

#### Table 3-71. Function cau_aes_ofb

<table>
<thead>
<tr>
<th>Function name</th>
<th>cau_aes_ofb</th>
</tr>
</thead>
<tbody>
<tr>
<td>Function prototype</td>
<td>ErrStatus cau_aes_ofb(cau_parameter_struct *cau_parameter, uint8_t *output);</td>
</tr>
<tr>
<td>Function descriptions</td>
<td>encrypt and decrypt using AES in OFB mode</td>
</tr>
<tr>
<td>Precondition</td>
<td>-</td>
</tr>
</tbody>
</table>
The called functions | - |

Input parameter(in)
Example:

```c
cau_parameter_struct cau_ofb_parameter;

uint8_t encrypt_result[TEXT_SIZE];

ErrStatus status;

......

/* encryption in OFB mode */
cau_ofb_parameter.alg_dir = CAU_ENCRYPT;
cau_ofb_parameter.key = (uint8_t *)key_128;
cau_ofb_parameter.key_size = KEY_SIZE;
cau_ofb_parameter.iv = (uint8_t *)vectors;
cau_ofb_parameter.iv_size = IV_SIZE;
cau_ofb_parameter.input = (uint8_t *)plaintext;
cau_ofb_parameter.in_length = PLAINTEXT_SIZE;
status = cau_aes_ofb(&cau_ofb_parameter, encrypt_result);
```

**cau_aes_gcm**

The description of `cau_aes_gcm` is shown as below:

**Table 3-72. Function cau_aes_gcm**

<table>
<thead>
<tr>
<th>Function name</th>
<th>cau_aes_gcm</th>
</tr>
</thead>
<tbody>
<tr>
<td>Function prototype</td>
<td>ErrStatus cau_aes_gcm(cau_parameter_struct *cau_parameter, uint8_t *output, uint8_t *tag);</td>
</tr>
<tr>
<td>Function descriptions</td>
<td>encrypt and decrypt using AES in GCM mode</td>
</tr>
<tr>
<td>Precondition</td>
<td>-</td>
</tr>
<tr>
<td>The called functions</td>
<td>-</td>
</tr>
<tr>
<td>Input parameter(in)</td>
<td>-</td>
</tr>
<tr>
<td><code>cau_parameter</code></td>
<td>structure for encrypt and decrypt parameters, refer to structure Table 3-46. Structure cau_parameter_struct</td>
</tr>
<tr>
<td><code>output</code></td>
<td>pointer to the returned buffer</td>
</tr>
</tbody>
</table>
### GD32L23x Firmware Library User Guide

#### cau_gcm_parameter

<table>
<thead>
<tr>
<th>Output parameter(out)</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>tag</td>
<td>pointer to the returned tag buffer</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Return value</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>ErrStatus</td>
<td>SUCCESS or ERROR</td>
</tr>
</tbody>
</table>

**Example:**

```c
cau_parameter_struct cau_gcm_parameter;
uint8_t encrypt_result[TEXT_SIZE];
uint8_t gcm_tag[GCM_TAG_SIZE];
ErrStatus status;

......

/* encryption in GCM mode */
cau_gcm_parameter.alg_dir = CAU_ENCRYPT;
ciau_gcm_parameter.key = (uint8_t *)key_128;
ciau_gcm_parameter.key_size = KEY_SIZE;
ciau_gcm_parameter.iv = (uint8_t *)vectors;
ciau_gcm_parameter.iv_size = IV_SIZE;
ciau_gcm_parameter.input = (uint8_t *)plaintext;
ciau_gcm_parameter.in_length = PLAINTEXT_SIZE;
ciau_gcm_parameter.aad = (uint8_t *)aadmessage;
ciau_gcm_parameter.aad_size = AAD_SIZE;
status = cau_aes_gcm(&cau_gcm_parameter, encrypt_result, gcm_tag);
```

**cau_aes_ccm**

The description of cau_aes_ccm is shown as below:

**Table 3-73. Function cau_aes_ccm**

<table>
<thead>
<tr>
<th>Function name</th>
<th>cau_aes_ccm</th>
</tr>
</thead>
</table>

<table>
<thead>
<tr>
<th>Function prototype</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>ErrStatus cau_aes_ccm(cau_parameter_struct *cau_parameter, uint8_t *output, uint8_t_t tag[], uint32_t tag_size, uint8_t_t aad_buf[]);</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Function descriptions</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>encrypt and decrypt using AES in CCM mode</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Precondition</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>-</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>The called functions</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>-</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Input parameter(in)</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>cau_parameter</td>
<td>structure for encrypt and decrypt parameters, refer to structure Table 3-46,</td>
</tr>
</tbody>
</table>

79
**Structure cau_parameter_struct**

<table>
<thead>
<tr>
<th>Input parameter(in)</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>tag_size</td>
<td>tag size (in bytes)</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Output parameter(out)</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>output</td>
<td>pointer to the returned buffer</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Output parameter(out)</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>tag</td>
<td>pointer to the returned tag buffer</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Output parameter(out)</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>aad_buf</td>
<td>pointer to the user buffer used when formatting aad block</td>
</tr>
</tbody>
</table>

**Return value**

| ErrStatus | SUCCESS or ERROR |

Example:

```c
cau_parameter_struct cau_ccm_parameter;
uint8_t encrypt_result[TEXT_SIZE];
uint8_t ccm_tag[CCM_TAG_SIZE];
uint8_t aad_buf[AAD_SIZE + 21];
ErrStatus status;

/* encryption in CCM mode */
cau_ccm_parameter.alg_dir = CAU_ENCRYPT;
cau_ccm_parameter.key = (uint8_t *)ccm_key_128;
cau_ccm_parameter.key_size = KEY_SIZE;
cau_ccm_parameter.iv = (uint8_t *)ccm_vectors;
cau_ccm_parameter.iv_size = CCM_IV_SIZE;
cau_ccm_parameter.input = (uint8_t *)plaintext;
cau_ccm_parameter.in_length = PLAINTEXT_SIZE;
cau_ccm_parameter.aad = (uint8_t *)aadmessage;
cau_ccm_parameter.aad_size = AAD_SIZE;
status = cau_aes_ccm(&cau_ccm_parameter, encrypt_result, ccm_tag, CCM_TAG_SIZE, (uint8_t *)aad_buf);
```

**cau_tdes_ecb**

The description of cau_tdes_ecb is shown as below:
### Table 3-74. Function cau_tdes_ecb

<table>
<thead>
<tr>
<th>Function name</th>
<th>cau_tdes_ecb</th>
</tr>
</thead>
<tbody>
<tr>
<td>Function prototype</td>
<td>ErrStatus cau_tdes_ecb(cau_parameter_struct *cau_parameter, uint8_t *output);</td>
</tr>
<tr>
<td>Function descriptions</td>
<td>encrypt and decrypt using TDES in ECB mode</td>
</tr>
<tr>
<td>Precondition</td>
<td>-</td>
</tr>
<tr>
<td>The called functions</td>
<td>-</td>
</tr>
</tbody>
</table>

**Input parameter (in)**

| cau_parameter | structure for encrypt and decrypt parameters, refer to structure Table 3-46. |

**Output parameter (out)**

| output | pointer to the returned buffer |

**Return value**

| ErrStatus | SUCCESS or ERROR |

Example:

cau_parameter_struct text;
uint8_t encrypt_result[DATA_SIZE];
ErrStatus status;
......
text.alg_dir = CAU_ENCRYPT;
text.key = tdes_key;
text.input = plaintext;
text.in_length = DATA_SIZE;

/* encryption in ECB mode */
status = cau_tdes_ecb(&text, encrypt_result);

---

cau_tdes_cbc

The description of cau_tdes_cbc is shown as below:

### Table 3-75. Function cau_tdes_cbc

<table>
<thead>
<tr>
<th>Function name</th>
<th>cau_tdes_cbc</th>
</tr>
</thead>
<tbody>
<tr>
<td>Function prototype</td>
<td>ErrStatus cau_tdes_cbc(cau_parameter_struct *cau_parameter, uint8_t *output);</td>
</tr>
<tr>
<td>Function descriptions</td>
<td>encrypt and decrypt using TDES in CBC mode</td>
</tr>
<tr>
<td>Precondition</td>
<td>-</td>
</tr>
<tr>
<td>The called functions</td>
<td>-</td>
</tr>
</tbody>
</table>

**Input parameter (in)**

-
cau_parameter | structure for encrypt and decrypt parameters, refer to structure *Table 3-46.*  
| **Structure cau_parameter_struct**  

<table>
<thead>
<tr>
<th>Output parameter(out)</th>
</tr>
</thead>
</table>
| output                | pointer to the returned buffer  

<table>
<thead>
<tr>
<th>Return value</th>
</tr>
</thead>
</table>
| ErrStatus    | SUCCESS or ERROR  

### **Example:**

```c
cau_parameter_struct text;
uint8_t encrypt_result[DATA_SIZE];
ErrStatus status;
......
text.alg_dir = CAU_ENCRYPT;
text.key = tdes_key;
text.iv = vectors;
text.input = plaintext;
text.in_length = DATA_SIZE;
/* encryption in CBC mode */
status = cau_tdes_cbc(&text, encrypt_result);
```

**cau_des_ecb**

The description of `cau_des_ecb` is shown as below:

<table>
<thead>
<tr>
<th><strong>Table 3-76. Function cau_des_ecb</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Function name</strong></td>
</tr>
<tr>
<td><strong>Function prototype</strong></td>
</tr>
<tr>
<td><strong>Function descriptions</strong></td>
</tr>
<tr>
<td><strong>Precondition</strong></td>
</tr>
<tr>
<td><strong>The called functions</strong></td>
</tr>
<tr>
<td><strong>Input parameter(in)</strong></td>
</tr>
</tbody>
</table>
| cau_parameter | structure for encrypt and decrypt parameters, refer to structure *Table 3-46.*  
| **Structure cau_parameter_struct**  

<table>
<thead>
<tr>
<th>Output parameter(out)</th>
</tr>
</thead>
</table>
| output                | pointer to the returned buffer  

<table>
<thead>
<tr>
<th>Return value</th>
</tr>
</thead>
</table>
| ErrStatus    | SUCCESS or ERROR  

```c
```
Example:

cau_parameter_struct text;
uint8_t encrypt_result[DATA_SIZE];
ErrStatus status;
……
text.alg_dir = CAU_ENCRYPT;
text.key = des_key;
text.input = plaintext;
text.in_length = DATA_SIZE;
/* encryption in ECB mode */
status = cau_des_ecb(&text, encrypt_result);

cau_des_cbc

The description of cau_des_cbc is shown as below:

Table 3-77. Function cau_des_cbc

<table>
<thead>
<tr>
<th>Function name</th>
<th>cau_des_cbc</th>
</tr>
</thead>
<tbody>
<tr>
<td>Function prototype</td>
<td>ErrStatus cau_des_cbc(cau_parameter_struct *cau_parameter, uint8_t *output);</td>
</tr>
<tr>
<td>Function descriptions</td>
<td>encrypt and decrypt using DES in CBC mode</td>
</tr>
<tr>
<td>Precondition</td>
<td>-</td>
</tr>
<tr>
<td>The called functions</td>
<td>-</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Input parameter(in)</th>
</tr>
</thead>
<tbody>
<tr>
<td>cau_parameter</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Output parameter(out)</th>
</tr>
</thead>
<tbody>
<tr>
<td>output</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Return value</th>
</tr>
</thead>
<tbody>
<tr>
<td>ErrStatus</td>
</tr>
</tbody>
</table>

Example:

cau_parameter_struct text;
uint8_t encrypt_result[DATA_SIZE];
ErrStatus status;
……
text.alg_dir = CAU_ENCRYPT;
text.key = des_key;

text.iv = vectors;

text.input = plaintext;

text.in_length = DATA_SIZE;

/* encryption in CBC mode */

status = cau_des_cipher(&text, encrypt_result);

cau_interrupt_enable

The description of cau_interrupt_enable is shown as below:

<table>
<thead>
<tr>
<th>Function name</th>
<th>cau_interrupt_enable</th>
</tr>
</thead>
<tbody>
<tr>
<td>Function prototype</td>
<td>void cau_interrupt_enable(uint32_t interrupt);</td>
</tr>
<tr>
<td>Function descriptions</td>
<td>enable the CAU interrupts</td>
</tr>
<tr>
<td>Precondition</td>
<td>-</td>
</tr>
<tr>
<td>The called functions</td>
<td>-</td>
</tr>
</tbody>
</table>

**Input parameter(in)**

<table>
<thead>
<tr>
<th>interrupt</th>
<th>specify the CAU interrupt source to be enabled</th>
</tr>
</thead>
<tbody>
<tr>
<td>CAU_INT_INFIFO</td>
<td>input FIFO interrupt</td>
</tr>
<tr>
<td>CAU_INT_OUTFIFO</td>
<td>output FIFO interrupt</td>
</tr>
</tbody>
</table>

**Output parameter(out)**

- -

**Return value**

- -

Example:

/* enable cau interrupt */

cau_interrupt_enable(CAU_INT_INFIFO);

cau_interrupt_disable

The description of cau_interrupt_disable is shown as below:

<table>
<thead>
<tr>
<th>Function name</th>
<th>cau_interrupt_disable</th>
</tr>
</thead>
<tbody>
<tr>
<td>Function prototype</td>
<td>void cau_interrupt_disable(uint32_t interrupt);</td>
</tr>
<tr>
<td>Function descriptions</td>
<td>disable the CAU interrupts</td>
</tr>
<tr>
<td>Precondition</td>
<td>-</td>
</tr>
<tr>
<td>The called functions</td>
<td>-</td>
</tr>
</tbody>
</table>

**Input parameter(in)**

<table>
<thead>
<tr>
<th>interrupt</th>
<th>specify the CAU interrupt source to be disabled</th>
</tr>
</thead>
</table>
CAU_INT_INFIFO | input FIFO interrupt
--- | ---
CAU_INT_OUTFIFO | output FIFO interrupt

### Return value

- 
- 

### Example:

```c
/* disable cau interrupt */
cau_interrupt_disable(CAU_INT_INFIFO);
```

### `cau_interrupt_flag_get`

The description of `cau_interrupt_flag_get` is shown as below:

#### Table 3-80. Function `cau_interrupt_flag_get`

<table>
<thead>
<tr>
<th>Function name</th>
<th><code>cau_interrupt_flag_get</code></th>
</tr>
</thead>
<tbody>
<tr>
<td>Function prototype</td>
<td><code>FlagStatus cau_interrupt_flag_get(uint32_t int_flag);</code></td>
</tr>
<tr>
<td>Function descriptions</td>
<td>get the interrupt flag</td>
</tr>
<tr>
<td>Precondition</td>
<td>-</td>
</tr>
<tr>
<td>The called functions</td>
<td>-</td>
</tr>
</tbody>
</table>

#### Input parameter(in)

<table>
<thead>
<tr>
<th>int_flag</th>
<th>CAU interrupt flag</th>
</tr>
</thead>
<tbody>
<tr>
<td>CAU_INT_FLAG_INFIFO</td>
<td>input FIFO interrupt</td>
</tr>
<tr>
<td>CAU_INT_FLAG_OUTFIFO</td>
<td>output FIFO interrupt</td>
</tr>
</tbody>
</table>

#### Output parameter(out)

- 
- 

#### Return value

- FlagStatus
  - SET or RESET

Example:

```c
/* get the CAU interrupt flag status */
FlagStatus status = RESET;
status = cau_interrupt_flag_get(CAU_INT_FLAG_INFIFO);
```

### `cau_flag_get`

The description of `cau_flag_get` is shown as below:

#### Table 3-81. Function `cau_flag_get`

<table>
<thead>
<tr>
<th>Function name</th>
<th><code>cau_flag_get</code></th>
</tr>
</thead>
</table>

Function prototype

FlagStatus cau_flag_get(uint32_t flag);

Function descriptions

get the CAU flag status

Precondition

-

The called functions

-

Input parameter (in)

<table>
<thead>
<tr>
<th>flag</th>
<th>CAU flag status</th>
</tr>
</thead>
<tbody>
<tr>
<td>CAU_FLAG_INFIFO_EMPTY</td>
<td>input FIFO empty</td>
</tr>
<tr>
<td>CAU_FLAG_INFIFO_N O_FULL</td>
<td>input FIFO is not full</td>
</tr>
<tr>
<td>CAU_FLAG_OUTFIFO_NO_EMPTY</td>
<td>output FIFO not empty</td>
</tr>
<tr>
<td>CAU_FLAG_OUTFIFO_FULL</td>
<td>output FIFO is full</td>
</tr>
<tr>
<td>CAU_FLAG_BUSY</td>
<td>the CAU core is busy</td>
</tr>
<tr>
<td>CAU_FLAG_INFIFO</td>
<td>input FIFO flag status</td>
</tr>
<tr>
<td>CAU_FLAG_OUTFIFO</td>
<td>output FIFO flag status</td>
</tr>
</tbody>
</table>

Output parameter (out)

-

Return value

FlagStatus

SET or RESET

Example:

/* get the CAU flag status */

FlagStatus status = RESET;

status = cau_flag_get(CAU_FLAG_INFIFO_EMPTY);

3.4. CMP

The general purpose comparator can work either standalone (all terminal are available on I/Os) or together with the timers. It provides a trigger source when an analog signal is in a certain condition, achieves some current control by working together with a PWM output of a TIMER.

The CMP registers are listed in chapter 3.4.1, the CMP firmware functions are introduced in chapter 3.4.2.

3.4.1. Descriptions of Peripheral registers

CMP registers are listed in the table shown as below:

Table 3-82. CMP Registers

<table>
<thead>
<tr>
<th>Registers</th>
<th>Descriptions</th>
</tr>
</thead>
<tbody>
<tr>
<td>CMP0_CS</td>
<td>CMP0 Control/Status register</td>
</tr>
</tbody>
</table>
### 3.4.2. Descriptions of Peripheral functions

CMP firmware functions are listed in the table shown below:

**Table 3-83. CMP firmware function**

<table>
<thead>
<tr>
<th>Function name</th>
<th>Function description</th>
</tr>
</thead>
<tbody>
<tr>
<td>cmp_deinit</td>
<td>deinitialize comparator</td>
</tr>
<tr>
<td>cmp_mode_init</td>
<td>initialize comparator mode</td>
</tr>
<tr>
<td>cmp1_plus_selection</td>
<td>select the plus input for CMP1</td>
</tr>
<tr>
<td>cmp_output_init</td>
<td>initialize comparator output</td>
</tr>
<tr>
<td>cmp_blanking_init</td>
<td>initialize comparator blanking function</td>
</tr>
<tr>
<td>cmp_enable</td>
<td>enable comparator</td>
</tr>
<tr>
<td>cmp_disable</td>
<td>disable comparator</td>
</tr>
<tr>
<td>cmp_window_enable</td>
<td>enable the window mode</td>
</tr>
<tr>
<td>cmp_window_disable</td>
<td>disable the window mode</td>
</tr>
<tr>
<td>cmp_voltage_scaler_enable</td>
<td>enable the voltage scaler</td>
</tr>
<tr>
<td>cmp_voltage_scaler_disable</td>
<td>disable the voltage scaler</td>
</tr>
<tr>
<td>cmp_scaler_bridge_enable</td>
<td>enable the scaler bridge</td>
</tr>
<tr>
<td>cmp_scaler_bridge_disable</td>
<td>disable the scaler bridge</td>
</tr>
<tr>
<td>cmp_lock_enable</td>
<td>lock the comparator</td>
</tr>
<tr>
<td>cmp_output_level_get</td>
<td>get output level</td>
</tr>
</tbody>
</table>

**Enum operating_mode_enum**

**Table 3-84. Enum operating_mode_enum**

<table>
<thead>
<tr>
<th>Member name</th>
<th>Function description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CMP_HIGHSPEED</td>
<td>high speed mode</td>
</tr>
<tr>
<td>CMP_MIDDLESPEED</td>
<td>medium speed mode</td>
</tr>
<tr>
<td>CMP_LOWSPEED</td>
<td>low speed mode</td>
</tr>
</tbody>
</table>

**Enum inverting_input_enum**

**Table 3-85. Enum inverting_input_enum**

<table>
<thead>
<tr>
<th>Member name</th>
<th>Function description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CMP_1_4VREFINT</td>
<td>VREFINT /4 input</td>
</tr>
<tr>
<td>CMP_1_2VREFINT</td>
<td>VREFINT /2 input</td>
</tr>
<tr>
<td>CMP_3_4VREFINT</td>
<td>VREFINT *3/4 input</td>
</tr>
<tr>
<td>CMP_VREFINT</td>
<td>VREFINT input</td>
</tr>
<tr>
<td>CMP_PA0_PA2</td>
<td>PA0 input for selecting CMP0, PA2 input for selecting CMP1</td>
</tr>
<tr>
<td>CMP_DACOUT_PA4</td>
<td>DAC_OUT0(PA4) input</td>
</tr>
<tr>
<td>CMP_PB3</td>
<td>PB3 input only for CMP1</td>
</tr>
</tbody>
</table>
### Enum cmp1_plus_input_enum

<table>
<thead>
<tr>
<th>Member name</th>
<th>Function description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CMP1_PA3</td>
<td>PA3 input</td>
</tr>
<tr>
<td>CMP1_PB4</td>
<td>PB4 input</td>
</tr>
<tr>
<td>CMP1_PB5</td>
<td>PB5 input</td>
</tr>
<tr>
<td>CMP1_PB6</td>
<td>PB6 input</td>
</tr>
<tr>
<td>CMP1_PB7</td>
<td>PB7 input</td>
</tr>
</tbody>
</table>

### Enum cmp_hysteresis_enum

<table>
<thead>
<tr>
<th>Member name</th>
<th>Function description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CMP_HYSTERESIS_NO</td>
<td>output no hysteresis</td>
</tr>
<tr>
<td>CMP_HYSTERESIS_LOW</td>
<td>output low hysteresis</td>
</tr>
<tr>
<td>CMP_HYSTERESIS_MIDDLE</td>
<td>output middle hysteresis</td>
</tr>
<tr>
<td>CMP_HYSTERESIS_HIGH</td>
<td>output high hysteresis</td>
</tr>
</tbody>
</table>

### Enum cmp_output_enum

<table>
<thead>
<tr>
<th>Member name</th>
<th>Function description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CMP_OUTPUT_NONE</td>
<td>output no selection</td>
</tr>
<tr>
<td>CMP_OUTPUT_TIMER1IC3</td>
<td>TIMER 1 channel3 input capture</td>
</tr>
<tr>
<td>CMP_OUTPUT_TIMER2IC0</td>
<td>TIMER 2 channel0 input capture</td>
</tr>
<tr>
<td>CMP_OUTPUT_LPTIMERIC0_IC1</td>
<td>LPTIMER channel0 or channel1 input capture</td>
</tr>
</tbody>
</table>

### Enum cmp_output_inv_enum

<table>
<thead>
<tr>
<th>Member name</th>
<th>Function description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CMP_OUTPUT_POLARITY_INVERTED</td>
<td>output is inverted</td>
</tr>
<tr>
<td>CMP_OUTPUT_POLARITY_NOINVERTED</td>
<td>output is not inverted</td>
</tr>
</tbody>
</table>

### Enum blanking_source_enum

<table>
<thead>
<tr>
<th>Member name</th>
<th>Function description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CMP_BLANKING_NONE</td>
<td>output no selection</td>
</tr>
<tr>
<td>CMP_BLANKING_TIMER1_OC1</td>
<td>TIMER 1 output channel 1</td>
</tr>
<tr>
<td>CMP_BLANKING_TIMER2_OC1</td>
<td>TIMER 2 output channel 1</td>
</tr>
<tr>
<td>CMP_BLANKING_TIMER8_OC1</td>
<td>TIMER 8 output channel 1</td>
</tr>
<tr>
<td>CMP_BLANKING_TIMER11_OC1</td>
<td>TIMER 11 output channel 1</td>
</tr>
</tbody>
</table>
Enum `cmp_output_state_enum`

<table>
<thead>
<tr>
<th>Member name</th>
<th>Function description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CMP_OUTPUTLEVEL_LOW</td>
<td>the output is low</td>
</tr>
<tr>
<td>CMP_OUTPUTLEVEL_HIGH</td>
<td>the output is high</td>
</tr>
</tbody>
</table>

`cmp_deinit`

The description of `cmp_deinit` is shown as below:

Table 3-92. Function `cmp_deinit`

<table>
<thead>
<tr>
<th>Function name</th>
<th>cmp_deinit</th>
</tr>
</thead>
<tbody>
<tr>
<td>Function prototype</td>
<td>void cmp_deinit(uint32_t cmp_periph);</td>
</tr>
<tr>
<td>Function descriptions</td>
<td>deinitialize comparator</td>
</tr>
<tr>
<td>Precondition</td>
<td>-</td>
</tr>
<tr>
<td>The called functions</td>
<td>-</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Input parameter(in)</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>cmp_periph</td>
<td>comparator periph</td>
</tr>
<tr>
<td>CMP0</td>
<td>comparator 0</td>
</tr>
<tr>
<td>CMP1</td>
<td>comparator 1</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Output parameter(out)</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>-</td>
<td>-</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Return value</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>-</td>
<td>-</td>
</tr>
</tbody>
</table>

Example:

/* CMP deinitialize*/

cmp_deinit(CMP0);

`cmp_mode_init`

The description of `cmp_mode_init` is shown as below:

Table 3-93. Function `cmp_mode_init`

<table>
<thead>
<tr>
<th>Function name</th>
<th>cmp_mode_init</th>
</tr>
</thead>
<tbody>
<tr>
<td>Function prototype</td>
<td>void cmp_mode_init(uint32_t cmp_periph, operating_mode_enum operating_mode, inverting_input_enum inverting_input, cmp_hysteresis_enum output_hysteresis)</td>
</tr>
<tr>
<td>Function descriptions</td>
<td>initialize comparator mode</td>
</tr>
<tr>
<td>Precondition</td>
<td>-</td>
</tr>
<tr>
<td>The called functions</td>
<td>-</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Input parameter(in)</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>cmp_periph</td>
<td>comparator periph</td>
</tr>
</tbody>
</table>
### CMP0 & CMP1

<table>
<thead>
<tr>
<th>Comparator</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CMP0</td>
<td>comparator 0</td>
</tr>
<tr>
<td>CMP1</td>
<td>comparator 1</td>
</tr>
</tbody>
</table>

### Input parameter (in)

- **operating_mode**: operating_mode, refer to [Table 3-84. Enum operating_mode_enum](#)
  - CMP_HIGHSPEED: high speed mode
  - CMP_MIDDLESPEED: medium speed mode
  - CMP_LOWSPEED: low speed mode

### Input parameter (in)

- **inverting_input**: inverting_input, refer to [Table 3-85. Enum inverting_input_enum](#)
  - CMP_1_4VREFINT: VREFINT *1/4 input
  - CMP_1_2VREFINT: VREFINT *1/2 input
  - CMP_3_4VREFINT: VREFINT *3/4 input
  - CMP_VREFINT: VREFINT input
  - CMP_PA0_PA2: PA0 input when selecting CMP0, PA2 input when selecting CMP1
  - CMP_DACOUT_PA4: DAC_OUT(PA4) input
  - CMP_PB3: PB3 input when selecting CMP1

### Input parameter (in)

- **output_hysteresis**: hysteresis, refer to [Table 3-87. Enum cmp_hysteresis_enum](#)
  - CMP_HYSTERESIS_NO: output no hysteresis
  - CMP_HYSTERESIS_LOW: output low hysteresis
  - CMP_HYSTERESIS_MIDDLE: output middle hysteresis
  - CMP_HYSTERESIS_HIGH: output high hysteresis

### Output parameter (out)

<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>---</td>
<td>---</td>
</tr>
</tbody>
</table>

### Return value

| - | - |

Example:

```c
/* CMP0 mode initialize*/
cmp_mode_init(CMP0,CMP_HIGHSPEED,CMP_1_4VREFINT, CMP_HYSTERESIS_NO);
```

### cmp1_plus_selection

The description of cmp1_plus_selection is shown as below:

#### Table 3-94. Function cmp1_plus_selection

<table>
<thead>
<tr>
<th>Function name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Cmp1_plus_selection</td>
<td>selecte the plus input for CMP1</td>
</tr>
</tbody>
</table>

```c
// CMP0 mode initialize*/
cmp_mode_init(CMP0,CMP_HIGHSPEED,CMP_1_4VREFINT, CMP_HYSTERESIS_NO);
```
**Precondition**

- 

**The called functions**

- 

### Input parameter(in)

- **plus_input**
  - comparator 1 plus input, refer to Table 3-86. Enum cmp1_plus_input_enum

| CMP1_PA3   | selecte PA3 as plus input for CMP1 |
| CMP1_PB4   | selecte PB4 as plus input for CMP1 |
| CMP1_PB5   | selecte PB5 as plus input for CMP1 |
| CMP1_PB6   | selecte PB6 as plus input for CMP1 |
| CMP1_PB7   | selecte PB7 as plus input for CMP1 |

### Output parameter(out)

- 

**Return value**

- 

---

Example:

```c
/* selecte the plus input for CMP1*/

cmp1_plus_selection(CMP1_PA3);
```

**cmp_output_init**

The description of cmp_output_init is shown as below:

### Table 3-95. Function cmp_output_init

<table>
<thead>
<tr>
<th>Function name</th>
<th>cmp_output_init</th>
</tr>
</thead>
<tbody>
<tr>
<td>Function prototype</td>
<td>void cmp_output_init(uint32_t cmp_periph, cmp_output_enum output_slection, uint32_t output_polarity);</td>
</tr>
<tr>
<td>Function descriptions</td>
<td>initialize comparator output</td>
</tr>
<tr>
<td>Precondition</td>
<td>-</td>
</tr>
<tr>
<td>The called functions</td>
<td>-</td>
</tr>
</tbody>
</table>

### Input parameter(in)

- **cmp_periph**
  - comparator periph

| CMP0   | comparator 0 |
| CMP1   | comparator 1 |

### Input parameter(in)

- **output_slection**
  - output_slection, refer to Table 3-86. Enum cmp_output_enum

<p>| CMP_OUTPUT_NONE               | output no selection |
| CMP_OUTPUT_TIMER_1IC3         | TIMER 1 channel3 input capture |
| CMP_OUTPUT_TIMER_2IC0         | TIMER 2 channel0 input capture |
| CMP_OUTPUT_LPTIMERIC0_IC1     | LPTIMER channel0 or channel1 input capture |</p>
<table>
<thead>
<tr>
<th>Input parameter(in)</th>
<th>Output polarity, refer to Table 3-89, Enum cmp_output_inv_enum</th>
</tr>
</thead>
<tbody>
<tr>
<td>output_polarity</td>
<td>output is inverted</td>
</tr>
<tr>
<td>CMP_OUTPUT_POLARITY_INVERTED</td>
<td>output is inverted</td>
</tr>
<tr>
<td>CMP_OUTPUT_POLARITY_NOINVERTED</td>
<td>output is not inverted</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Output parameter(out)</th>
</tr>
</thead>
<tbody>
<tr>
<td>-</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Return value</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>-</td>
<td>-</td>
</tr>
</tbody>
</table>

Example:

```c
/* CMP0 output initialize*/

cmp_output_init(CMP0,CMP_OUTPUT_TIMER1IC3,
CMP_OUTPUT_POLARITY_NOINVERTED);
```

cmpBlanking_init

The description of cmpBlanking_init is shown as below:

**Table 3-96. Function cmpBlanking_init**

<table>
<thead>
<tr>
<th>Function name</th>
<th>cmpBlanking_init</th>
</tr>
</thead>
<tbody>
<tr>
<td>Function prototype</td>
<td>void cmpBlanking_init(uint32_t cmp_periph, blanking_source_enum blanking_source_selection);</td>
</tr>
<tr>
<td>Function descriptions</td>
<td>initialize comparator blanking function</td>
</tr>
<tr>
<td>Precondition</td>
<td>-</td>
</tr>
<tr>
<td>The called functions</td>
<td>-</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Input parameter(in)</th>
</tr>
</thead>
<tbody>
<tr>
<td>cmp_periph</td>
</tr>
<tr>
<td>CMP0</td>
</tr>
<tr>
<td>CMP1</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Input parameter(in)</th>
<th>blanking source selection, refer to Table 3-90, Enum blanking_source_enum</th>
</tr>
</thead>
<tbody>
<tr>
<td>CMP_BLANKING_NONE</td>
<td>output no selection</td>
</tr>
<tr>
<td>CMP_BLANKING_TIMER1_OC1</td>
<td>TIMER 1 output channel 1</td>
</tr>
<tr>
<td>CMP_BLANKING_TIMER2_OC1</td>
<td>TIMER 2 output channel 1</td>
</tr>
<tr>
<td>CMP_BLANKING_TIMER8_OC1</td>
<td>TIMER 8 output channel 1</td>
</tr>
<tr>
<td>CMP_BLANKING_TIMER11_OC1</td>
<td>TIMER 11 output channel 1</td>
</tr>
</tbody>
</table>
### cmp_enable

The description of `cmp_enable` is shown as below:

<table>
<thead>
<tr>
<th>Function name</th>
<th>cmp_enable</th>
</tr>
</thead>
<tbody>
<tr>
<td>Function prototype</td>
<td>void cmp_enable(uint32_t cmp_periph);</td>
</tr>
<tr>
<td>Function descriptions</td>
<td>enable comparator</td>
</tr>
<tr>
<td>Precondition</td>
<td>-</td>
</tr>
<tr>
<td>The called functions</td>
<td>-</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Input parameter(in)</th>
</tr>
</thead>
<tbody>
<tr>
<td>cmp_periph</td>
</tr>
<tr>
<td>CMP0</td>
</tr>
<tr>
<td>CMP1</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Output parameter(out)</th>
</tr>
</thead>
<tbody>
<tr>
<td>-</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Return value</th>
</tr>
</thead>
<tbody>
<tr>
<td>-</td>
</tr>
</tbody>
</table>

Example:

/* initialize comparator blanking function */
cmp_blanking_init(CMP_BLANKING_NONE);

cmp_enable(CMP0);

### cmp_disable

The description of `cmp_disable` is shown as below:

<table>
<thead>
<tr>
<th>Function name</th>
<th>cmp_disable</th>
</tr>
</thead>
<tbody>
<tr>
<td>Function prototype</td>
<td>void cmp_disable(uint32_t cmp_periph);</td>
</tr>
<tr>
<td>Function descriptions</td>
<td>disable comparator</td>
</tr>
<tr>
<td>Precondition</td>
<td>-</td>
</tr>
<tr>
<td>The called functions</td>
<td>-</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Input parameter(in)</th>
</tr>
</thead>
</table>

Example:

/* enable CMP0*/
cmp_enable(CMP0);
Example:

/* disable CMP0 */

cmp_disable(CMP0);

**cmp_window_enable**

The description of cmp_window_enable is shown as below:

<table>
<thead>
<tr>
<th>Function name</th>
<th>cmp_window_enable</th>
</tr>
</thead>
<tbody>
<tr>
<td>Function prototype</td>
<td>void cmp_window_enable(void);</td>
</tr>
<tr>
<td>Function descriptions</td>
<td>enable the window mode</td>
</tr>
<tr>
<td>Precondition</td>
<td>-</td>
</tr>
<tr>
<td>The called functions</td>
<td>-</td>
</tr>
</tbody>
</table>

| Input parameter(in) | - |
| Output parameter(out)| - |

**Return value**

Example:

/* enable the window mode */

cmp_window_enable();

**cmp_window_disable**

The description of cmp_window_disable is shown as below:

<table>
<thead>
<tr>
<th>Function name</th>
<th>cmp_window_disable</th>
</tr>
</thead>
<tbody>
<tr>
<td>Function prototype</td>
<td>void cmp_window_disable(void);</td>
</tr>
<tr>
<td>Function descriptions</td>
<td>disable the window mode</td>
</tr>
<tr>
<td>Precondition</td>
<td>-</td>
</tr>
<tr>
<td>The called functions</td>
<td>-</td>
</tr>
</tbody>
</table>

| Output parameter(out)| - |

**Return value**
**cmp_voltage_scaler_enable**

The description of *cmp_voltage_scaler_enable* is shown as below:

**Table 3-101. Function cmp_voltage_scaler_enable**

<table>
<thead>
<tr>
<th>Function name</th>
<th>cmp_voltage_scaler_enable</th>
</tr>
</thead>
<tbody>
<tr>
<td>Function prototype</td>
<td>void cmp_voltage_scaler_enable(uint32_t cmp_periph);</td>
</tr>
<tr>
<td>Function descriptions</td>
<td>enable comparator the voltage scaler</td>
</tr>
<tr>
<td>Precondition</td>
<td>-</td>
</tr>
<tr>
<td>The called functions</td>
<td>-</td>
</tr>
</tbody>
</table>

**Input parameter(in)**

<table>
<thead>
<tr>
<th>cmp_periph</th>
<th>comparator periph</th>
</tr>
</thead>
<tbody>
<tr>
<td>CMP0</td>
<td>comparator 0</td>
</tr>
<tr>
<td>CMP1</td>
<td>comparator 1</td>
</tr>
</tbody>
</table>

**Output parameter(out)**

| -                 | -                 |

**Return value**

| -                 | -                 |

Example:

/* enable CMP0 the voltage scaler */

cmp_voltage_scaler_enable(CMP0);

**cmp_voltage_scaler_disable**

The description of *cmp_voltage_scaler_disable* is shown as below:

**Table 3-102. Function cmp_voltage_scaler_disable**

<table>
<thead>
<tr>
<th>Function name</th>
<th>cmp_voltage_scaler_disable</th>
</tr>
</thead>
<tbody>
<tr>
<td>Function prototype</td>
<td>void cmp_voltage_scaler_disable(uint32_t cmp_periph);</td>
</tr>
<tr>
<td>Function descriptions</td>
<td>disable comparator the voltage scaler</td>
</tr>
<tr>
<td>Precondition</td>
<td>-</td>
</tr>
</tbody>
</table>
The called functions

<table>
<thead>
<tr>
<th>Input parameter(in)</th>
</tr>
</thead>
<tbody>
<tr>
<td>cmp_periph</td>
</tr>
<tr>
<td>CMP0</td>
</tr>
<tr>
<td>CMP1</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Output parameter(out)</th>
</tr>
</thead>
<tbody>
<tr>
<td>-</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Return value</th>
</tr>
</thead>
<tbody>
<tr>
<td>-</td>
</tr>
</tbody>
</table>

Example:

/* disable CMP0 the voltage scaler */
cmp_voltage_scaler_disable(CMP0);

**cmp_scaler_bridge_enable**

The description of `cmp_scaler_bridge_enable` is shown as below:

**Table 3-103. Function cmp_scaler_bridge_enable**

<table>
<thead>
<tr>
<th>Function name</th>
</tr>
</thead>
<tbody>
<tr>
<td>cmp_scaler_bridge_enable</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Function prototype</th>
</tr>
</thead>
<tbody>
<tr>
<td>void cmp_scaler_bridge_enable(uint32_t cmp_periph);</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Function descriptions</th>
</tr>
</thead>
<tbody>
<tr>
<td>enable comparator the scaler bridge</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Precondition</th>
</tr>
</thead>
<tbody>
<tr>
<td>-</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>The called functions</th>
</tr>
</thead>
<tbody>
<tr>
<td>-</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Input parameter(in)</th>
</tr>
</thead>
<tbody>
<tr>
<td>cmp_periph</td>
</tr>
<tr>
<td>CMP0</td>
</tr>
<tr>
<td>CMP1</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Output parameter(out)</th>
</tr>
</thead>
<tbody>
<tr>
<td>-</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Return value</th>
</tr>
</thead>
<tbody>
<tr>
<td>-</td>
</tr>
</tbody>
</table>

Example:

/* enable CMP0 the scaler bridge */
cmp_scaler_bridge_enable(CMP0);

**cmp_scaler_bridge_disable**

The description of `cmp_scaler_bridge_disable` is shown as below:

**Table 3-104. Function cmp_scaler_bridge_disable**

<table>
<thead>
<tr>
<th>Function name</th>
</tr>
</thead>
<tbody>
<tr>
<td>cmp_scaler_bridge_disable</td>
</tr>
</tbody>
</table>
Function prototype: `void cmp_scaler_bridge_disable(uint32_t cmp_periph);`

Function descriptions: disable comparator the scaler bridge

Precondition: 

The called functions: 

<table>
<thead>
<tr>
<th>Input parameter(in)</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>cmp_periph</td>
<td>comparator periph</td>
</tr>
<tr>
<td>CMP0</td>
<td>comparator 0</td>
</tr>
<tr>
<td>CMP1</td>
<td>comparator 1</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Output parameter(out)</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Return value</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Example:

/* disable CMP0 the scaler bridge */

cmp_scaler_bridge_disable(CMP0);

cmp_lock_enable

The description of cmp_lock_enable is shown as below:

Table 3-105. Function cmp_lock_enable

<table>
<thead>
<tr>
<th>Function name</th>
<th>cmp_lock_enable</th>
</tr>
</thead>
<tbody>
<tr>
<td>Function prototype</td>
<td>void cmp_lock_enable(uint32_t cmp_periph);</td>
</tr>
<tr>
<td>Function descriptions</td>
<td>lock the comparator</td>
</tr>
<tr>
<td>Precondition</td>
<td>-</td>
</tr>
<tr>
<td>The called functions</td>
<td>-</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Input parameter(in)</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>cmp_periph</td>
<td>comparator periph</td>
</tr>
<tr>
<td>CMP0</td>
<td>comparator 0</td>
</tr>
<tr>
<td>CMP1</td>
<td>comparator 1</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Output parameter(out)</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Return value</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Example:

/* lock CMP0 register */

cmp_lock_enable(CMP0);

cmp_output_level_get

The description of cmp_output_level_get is shown as below:
Table 3-106. Function cmp_output_level_get

<table>
<thead>
<tr>
<th>Function name</th>
<th>cmp_output_level_get</th>
</tr>
</thead>
<tbody>
<tr>
<td>Function prototype</td>
<td>uint32_t cmp_output_level_get(uint32_t cmp_periph);</td>
</tr>
<tr>
<td>Function descriptions</td>
<td>get output level</td>
</tr>
<tr>
<td>Precondition</td>
<td>-</td>
</tr>
<tr>
<td>The called functions</td>
<td>-</td>
</tr>
</tbody>
</table>

| Input parameter(in)    |                          |
|CMP_periph              | comparator periph        |
|CMP0                    | comparator 0             |
|CMP1                    | comparator 1             |

| Output parameter(out)  |                          |
|CMP_OUTPUTLEVEL_HIGH    | Non-inverting input above inverting input and the output is high |
|CMP_OUTPUTLEVEL_LOW     | Non-inverting input below inverting input and the output is low |

Return value

uint32_t the output level, refer to Table 3-91, Enum cmp_output_state_enum

Example:

"/* get CMP0 output level */
cmp_output_level_get(CMP0);

3.5. CRC

A cyclic redundancy check (CRC) is an error-detecting code commonly used in digital networks and storage devices to detect accidental changes to raw data. The CRC registers are listed in chapter 3.5.1, the CRC firmware functions are introduced in chapter 3.5.2.

3.5.1. Descriptions of Peripheral registers

CRC registers are listed in the table shown as below:

<table>
<thead>
<tr>
<th>Registers</th>
<th>Descriptions</th>
</tr>
</thead>
<tbody>
<tr>
<td>CRC_DATA</td>
<td>CRC data register</td>
</tr>
<tr>
<td>CRC_FDATA</td>
<td>CRC free data register</td>
</tr>
<tr>
<td>CRC_CTL</td>
<td>CRC control register</td>
</tr>
<tr>
<td>CRC_IDATA</td>
<td>CRC initialization data register</td>
</tr>
<tr>
<td>CRC_POLY</td>
<td>CRC polynomial register</td>
</tr>
</tbody>
</table>
3.5.2. Descriptions of Peripheral functions

CRC firmware functions are listed in the table shown as below:

Table 3-108. CRC firmware function

<table>
<thead>
<tr>
<th>Function name</th>
<th>Function description</th>
</tr>
</thead>
<tbody>
<tr>
<td>crc_deinit</td>
<td>deinit CRC calculation unit</td>
</tr>
<tr>
<td>crc_reverse_output_data_enable</td>
<td>enable the reverse operation of output data</td>
</tr>
<tr>
<td>crc_reverse_output_data_disable</td>
<td>disable the reverse operation of output data</td>
</tr>
<tr>
<td>crc_data_register_reset</td>
<td>reset data register to the value of initialization data register</td>
</tr>
<tr>
<td>crc_data_register_read</td>
<td>read the data register</td>
</tr>
<tr>
<td>crc_free_data_register_read</td>
<td>read the free data register</td>
</tr>
<tr>
<td>crc_free_data_register_write</td>
<td>write the free data register</td>
</tr>
<tr>
<td>crc_init_data_register_write</td>
<td>write the initial value register</td>
</tr>
<tr>
<td>crc_input_data_reverse_config</td>
<td>configure the CRC input data function</td>
</tr>
<tr>
<td>crc_polynomial_size_set</td>
<td>configure the CRC size of polynomial function</td>
</tr>
<tr>
<td>crc_polynomial_set</td>
<td>configure the CRC polynomial value function</td>
</tr>
<tr>
<td>crc_single_data_calculate</td>
<td>CRC calculate single data</td>
</tr>
<tr>
<td>crc_block_data_calculate</td>
<td>CRC calculate a data array</td>
</tr>
</tbody>
</table>

**crc_deinit**

The description of crc_deinit is shown as below:

Table 3-109. Function crc_deinit

<table>
<thead>
<tr>
<th>Function name</th>
<th>crc_deinit</th>
</tr>
</thead>
<tbody>
<tr>
<td>Function prototype</td>
<td>void crc_deinit(void);</td>
</tr>
<tr>
<td>Function descriptions</td>
<td>deinit CRC calculation unit</td>
</tr>
<tr>
<td>Precondition</td>
<td>-</td>
</tr>
<tr>
<td>The called functions</td>
<td>-</td>
</tr>
<tr>
<td>Input parameter(in)</td>
<td>-</td>
</tr>
<tr>
<td>Output parameter(out)</td>
<td>-</td>
</tr>
<tr>
<td>Return value</td>
<td>-</td>
</tr>
</tbody>
</table>

Example:

```c
/* reset crc */
crc_deinit();
```

**crc_reverse_output_data_enable**

The description of crc_reverse_output_data_enable is shown as below:
Table 3-110. Function crc_reverse_output_data_enable

<table>
<thead>
<tr>
<th>Function name</th>
<th>crc_reverse_output_data_enable</th>
</tr>
</thead>
<tbody>
<tr>
<td>Function prototype</td>
<td>void crc_reverse_output_data_enable(void);</td>
</tr>
<tr>
<td>Function descriptions</td>
<td>enable the reverse operation of output data</td>
</tr>
<tr>
<td>Precondition</td>
<td>-</td>
</tr>
<tr>
<td>The called functions</td>
<td>-</td>
</tr>
<tr>
<td>Input parameter(in)</td>
<td>-</td>
</tr>
<tr>
<td>Output parameter(out)</td>
<td>-</td>
</tr>
<tr>
<td>Return value</td>
<td>-</td>
</tr>
</tbody>
</table>

Example:

```c
/* enable CRC reverse operation of output data */
crc_reverse_output_data_enable();
```

crc_reverse_output_data_disable

The description of crc_reverse_output_data_disable is shown as below:

Table 3-111. Function crc_reverse_output_data_disable

<table>
<thead>
<tr>
<th>Function name</th>
<th>crc_reverse_output_data_disable</th>
</tr>
</thead>
<tbody>
<tr>
<td>Function prototype</td>
<td>void crc_reverse_output_data_disable(void);</td>
</tr>
<tr>
<td>Function descriptions</td>
<td>disable the reverse operation of output data</td>
</tr>
<tr>
<td>Precondition</td>
<td>-</td>
</tr>
<tr>
<td>The called functions</td>
<td>-</td>
</tr>
<tr>
<td>Input parameter(in)</td>
<td>-</td>
</tr>
<tr>
<td>Output parameter(out)</td>
<td>-</td>
</tr>
<tr>
<td>Return value</td>
<td>-</td>
</tr>
</tbody>
</table>

Example:

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

crc_data_register_reset

The description of crc_data_register_reset is shown as below:
Table 3-112. Function crc_data_register_reset

<table>
<thead>
<tr>
<th>Function name</th>
<th>crc_data_register_reset</th>
</tr>
</thead>
<tbody>
<tr>
<td>Function prototype</td>
<td>void crc_data_register_reset(void);</td>
</tr>
<tr>
<td>Function descriptions</td>
<td>reset data register to the value of initialization data register</td>
</tr>
<tr>
<td>Precondition</td>
<td>-</td>
</tr>
<tr>
<td>The called functions</td>
<td>-</td>
</tr>
</tbody>
</table>

**Input parameter(in)**

-  

**Output parameter(out)**

-  

**Return value**

-  

Example:

/* reset crc data register */

crc_data_register_reset();

**crc_data_register_read**

The description of crc_data_register_read is shown as below:

Table 3-113. Function crc_data_register_read

<table>
<thead>
<tr>
<th>Function name</th>
<th>crc_data_register_read</th>
</tr>
</thead>
<tbody>
<tr>
<td>Function prototype</td>
<td>uint32_t crc_data_register_read(void);</td>
</tr>
<tr>
<td>Function descriptions</td>
<td>read the data register</td>
</tr>
<tr>
<td>Precondition</td>
<td>-</td>
</tr>
<tr>
<td>The called functions</td>
<td>-</td>
</tr>
</tbody>
</table>

**Input parameter(in)**

-  

**Output parameter(out)**

-  

**Return value**

uint32_t 32-bit value of the data register (0-0xFFFFFFFF)

Example:

/* read crc data register */

uint32_t crc_value = 0;

crc_value = crc_data_register_read();

**crc_free_data_register_read**

The description of crc_free_data_register_read is shown as below:
Table 3-114. Function crc_free_data_register_read

<table>
<thead>
<tr>
<th>Function name</th>
<th>crc_free_data_register_read</th>
</tr>
</thead>
<tbody>
<tr>
<td>Function prototype</td>
<td>uint8_t crc_free_data_register_read(void);</td>
</tr>
<tr>
<td>Function descriptions</td>
<td>read the free data register</td>
</tr>
<tr>
<td>Precondition</td>
<td>-</td>
</tr>
<tr>
<td>The called functions</td>
<td>-</td>
</tr>
</tbody>
</table>

| Input parameter(in)          | -                           |
| Output parameter(out)        | -                           |

| Return value                 | uint8_t 8-bit value of the free data register (0-0xFF) |

Example:

/* read crc free data register */

uint8_t crc_value = 0;

crc_value = crc_free_data_register_read();

**crc_free_data_register_write**

The description of crc_free_data_register_write is shown as below:

Table 3-115. Function crc_free_data_register_write

<table>
<thead>
<tr>
<th>Function name</th>
<th>crc_free_data_register_write</th>
</tr>
</thead>
<tbody>
<tr>
<td>Function prototype</td>
<td>void crc_free_data_register_write(uint8_t free_data);</td>
</tr>
<tr>
<td>Function descriptions</td>
<td>write the free data register</td>
</tr>
<tr>
<td>Precondition</td>
<td>-</td>
</tr>
<tr>
<td>The called functions</td>
<td>-</td>
</tr>
</tbody>
</table>

| Input parameter(in)          | free_data specify 8-bit data |
| Output parameter(out)        | -                           |

| Return value                 | -                           |

Example:

/* write the free data register */

crc_free_data_register_write(0x11);

**crc_init_data_register_write**

The description of crc_init_data_register_write is shown as below:
Table 3-116. Function crc_init_data_register_write

<table>
<thead>
<tr>
<th>Function name</th>
<th>crc_init_data_register_write</th>
</tr>
</thead>
<tbody>
<tr>
<td>Function prototype</td>
<td>void crc_init_data_register_write(uint32_t init_data)</td>
</tr>
<tr>
<td>Function descriptions</td>
<td>write the initialization data register</td>
</tr>
<tr>
<td>Precondition</td>
<td>-</td>
</tr>
<tr>
<td>The called functions</td>
<td>-</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Input parameter (in)</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>init_data</td>
<td>specify 32-bit data</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Output parameter (out)</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>-</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Return value</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>-</td>
<td></td>
</tr>
</tbody>
</table>

Example:

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

crc_input_data_reverse_config

The description of crc_input_data_reverse_config is shown as below:

Table 3-117. Function crc_input_data_reverse_config

<table>
<thead>
<tr>
<th>Function name</th>
<th>crc_input_data_reverse_config</th>
</tr>
</thead>
<tbody>
<tr>
<td>Function prototype</td>
<td>void crc_input_data_reverse_config(uint32_t data_reverse)</td>
</tr>
<tr>
<td>Function descriptions</td>
<td>configure the crc input data function</td>
</tr>
<tr>
<td>Precondition</td>
<td>-</td>
</tr>
<tr>
<td>The called functions</td>
<td>-</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Input parameter (in)</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>data_reverse</td>
<td>specify input data reverse function</td>
</tr>
<tr>
<td>CRC_INPUT_DATA_NOT</td>
<td>input data is not reversed</td>
</tr>
<tr>
<td>CRC_INPUT_DATA_BYTE</td>
<td>input data is reversed on 8 bits</td>
</tr>
<tr>
<td>CRC_INPUT_DATA_HALFWORD</td>
<td>input data is reversed on 16 bits</td>
</tr>
<tr>
<td>CRC_INPUT_DATA_WORD</td>
<td>input data is reversed on 32 bits</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Output parameter (out)</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>-</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Return value</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>-</td>
<td></td>
</tr>
</tbody>
</table>

Example:
/* configure the crc input data */
crc_input_data_reverse_config(CRC_INPUT_DATA_WORD);

crc_polynomial_size_set

The description of crc_polynomial_size_set is shown as below:

Table 3-118. Function crc_polynomial_size_set

<table>
<thead>
<tr>
<th>Function name</th>
<th>crc_polynomial_size_set</th>
</tr>
</thead>
<tbody>
<tr>
<td>Function prototype</td>
<td>void crc_polynomial_size_set(uint32_t poly_size)</td>
</tr>
<tr>
<td>Function descriptions</td>
<td>configure the CRC size of polynomial function</td>
</tr>
<tr>
<td>Precondition</td>
<td>-</td>
</tr>
<tr>
<td>The called functions</td>
<td>-</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Input parameter(in)</th>
</tr>
</thead>
<tbody>
<tr>
<td>poly_size</td>
</tr>
<tr>
<td>CRC_CTL_PS_32</td>
</tr>
<tr>
<td>CRC_CTL_PS_16</td>
</tr>
<tr>
<td>CRC_CTL_PS_8</td>
</tr>
<tr>
<td>CRC_CTL_PS_7</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Output parameter(out)</th>
</tr>
</thead>
<tbody>
<tr>
<td>-</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Return value</th>
</tr>
</thead>
<tbody>
<tr>
<td>-</td>
</tr>
</tbody>
</table>

Example:

/* configure the CRC polynomial size*/
crc_polynomial_size_set(CRC_CTL_PS_7);

crc_polynomial_set

The description of crc_polynomial_set is shown as below:

Table 3-119. Function crc_polynomial_set

<table>
<thead>
<tr>
<th>Function name</th>
<th>crc_polynomial_set</th>
</tr>
</thead>
<tbody>
<tr>
<td>Function prototype</td>
<td>void crc_polynomial_set(uint32_t poly)</td>
</tr>
<tr>
<td>Function descriptions</td>
<td>configure the CRC polynomial value function</td>
</tr>
<tr>
<td>Precondition</td>
<td>-</td>
</tr>
<tr>
<td>The called functions</td>
<td>-</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Input parameter(in)</th>
</tr>
</thead>
<tbody>
<tr>
<td>poly</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Output parameter(out)</th>
</tr>
</thead>
<tbody>
<tr>
<td>-</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Return value</th>
</tr>
</thead>
<tbody>
<tr>
<td>-</td>
</tr>
</tbody>
</table>
Example:

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

\textbf{crc\_single\_data\_calculate}

The description of \texttt{crc\_single\_data\_calculate} is shown as below:

\begin{table}[h]
\centering
\begin{tabular}{|c|p{0.6\textwidth}|}
\hline
Function name & \texttt{crc\_single\_data\_calculate} \\
Function prototype & uint32\_t \ texttt{crc\_single\_data\_calculate}(uint32\_t \texttt{sdata}, uint8\_t \texttt{data}\_\texttt{format}); \\
Function descriptions & \texttt{CRC calculate a 32-bit data} \\
Precondition & - \\
The called functions & - \\
sdata & \texttt{Input parameter(in)} \\
& specify 32-bit data \\
data\_format & \texttt{Input parameter(in)} \\
\texttt{INPUT\_FORMAT\_WORD} & input data format in word format \\
\texttt{INPUT\_FORMAT\_HALF\_WORD} & input data format in half-word format \\
\texttt{INPUT\_FORMAT\_BYTE} & input data format in byte format \\
Output parameter\(\texttt{(out)}\) & - \\
\hline
\texttt{uint32\_t} & \texttt{Return value} \\
& 32-bit CRC calculate value (0-0xFFFFFFFF) \\
\hline
\end{tabular}
\end{table}

Example:

/* CRC calculate a 32-bit data */
uint32\_t val = 0, valcrc = 0;
val = (uint32\_t)0xabcd1234;
valcrc = \texttt{crc\_single\_data\_calculate}(val, \texttt{INPUT\_FORMAT\_WORD});

\textbf{crc\_block\_data\_calculate}

The description of \texttt{crc\_block\_data\_calculate} is shown as below:

\begin{table}[h]
\centering
\begin{tabular}{|c|p{0.6\textwidth}|}
\hline
Function name & \texttt{crc\_block\_data\_calculate} \\
\hline
\end{tabular}
\end{table}
Function prototype
uint32_t crc_block_data_calculate(uint32_t array[], uint32_t size, uint8_t data_format);

Function descriptions
calculate the CRC value of an array of 32-bit values

Precondition
-

The called functions

Input parameter (in)
array
pointer to an array of 32-bit data words

size
size of the array

Input parameter (in)
data_format
input data format

<table>
<thead>
<tr>
<th>data_format</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>INPUT_FORMAT_WORD</td>
<td>input data in word format</td>
</tr>
<tr>
<td>INPUT_FORMAT_HALFWORD</td>
<td>input data in half-word format</td>
</tr>
<tr>
<td>INPUT_FORMAT_BYTE</td>
<td>input data in byte format</td>
</tr>
</tbody>
</table>

Output parameter (out)
-

Return value
uint32_t
32-bit CRC calculate value (0-0xFFFFFFFF)

Example:

/* 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((uint32_t*) data_buffer, BUFFER_SIZE, INPUT_FORMAT_WORD);

3.6. CTC

The CTC unit trims the frequency of the IRC48M which is based on an external accurate reference signal source. It can adjust the calibration value to provide a precise IRC48M clock automatically or manually. The CTC registers are listed in chapter 3.6.1, the CTC firmware functions are introduced in chapter 3.6.2.
3.6.1. Descriptions of Peripheral registers

CTC registers are listed in the table shown as below:

Table 3-122. CTC Registers

<table>
<thead>
<tr>
<th>Registers</th>
<th>Descriptions</th>
</tr>
</thead>
<tbody>
<tr>
<td>CTC_CTL0</td>
<td>CTC control register 0</td>
</tr>
<tr>
<td>CTC_CTL1</td>
<td>CTC control register 1</td>
</tr>
<tr>
<td>CTC_STAT</td>
<td>CTC status register</td>
</tr>
<tr>
<td>CTC_INTC</td>
<td>CTC interrupt clear register</td>
</tr>
</tbody>
</table>

3.6.2. Descriptions of Peripheral functions

CTC registers are listed in the table shown as below:

Table 3-123. CTC firmware function

<table>
<thead>
<tr>
<th>Function name</th>
<th>Function description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ctc_deinit</td>
<td>reset CTC clock trim controller</td>
</tr>
<tr>
<td>ctc_counter_enable</td>
<td>enable CTC trim counter</td>
</tr>
<tr>
<td>ctc_counter_disable</td>
<td>disable CTC trim counter</td>
</tr>
<tr>
<td>ctc_irc48m_trim_value_config</td>
<td>configure the IRC48M trim value</td>
</tr>
<tr>
<td>ctc_software_refsource_pulse_generate</td>
<td>generate software reference source sync pulse</td>
</tr>
<tr>
<td>ctc_hardware_trim_mode_config</td>
<td>configure hardware automatically trim mode</td>
</tr>
<tr>
<td>ctc_refsource_polarity_config</td>
<td>configure reference signal source polarity</td>
</tr>
<tr>
<td>ctc_refsource_signal_select</td>
<td>select reference signal source</td>
</tr>
<tr>
<td>ctc_refsource_prescaler_config</td>
<td>configure reference signal source prescaler</td>
</tr>
<tr>
<td>ctc_clock_limit_value_config</td>
<td>configure clock trim base limit value</td>
</tr>
<tr>
<td>ctc_counter_reload_value_config</td>
<td>configure CTC counter reload value</td>
</tr>
<tr>
<td>ctc_counter_capture_value_read</td>
<td>read CTC counter capture value when reference sync pulse occurred</td>
</tr>
<tr>
<td>ctc_counter_direction_read</td>
<td>read CTC trim counter direction when reference sync pulse occurred</td>
</tr>
<tr>
<td>ctc_counter_reload_value_read</td>
<td>read CTC counter reload value</td>
</tr>
<tr>
<td>ctc_irc48m_trim_value_read</td>
<td>read the IRC48M trim value</td>
</tr>
<tr>
<td>ctc_interrupt_enable</td>
<td>enable the CTC interrupt</td>
</tr>
<tr>
<td>ctc_interrupt_disable</td>
<td>disable the CTC interrupt</td>
</tr>
<tr>
<td>ctc_interrupt_flag_get</td>
<td>get CTC interrupt flag</td>
</tr>
<tr>
<td>ctc_interrupt_flag_clear</td>
<td>clear CTC interrupt flag</td>
</tr>
<tr>
<td>ctc_flag_get</td>
<td>get CTC flag</td>
</tr>
<tr>
<td>ctc_flag_clear</td>
<td>clear CTC flag</td>
</tr>
</tbody>
</table>
ctc_deinit

The description of ctc_deinit is shown as below:

Table 3-124. Function ctc_deinit

<table>
<thead>
<tr>
<th>Function name</th>
<th>ctc_deinit</th>
</tr>
</thead>
<tbody>
<tr>
<td>Function prototype</td>
<td>void ctc_deinit(void)</td>
</tr>
<tr>
<td>Function descriptions</td>
<td>reset CTC clock trim controller</td>
</tr>
<tr>
<td>Precondition</td>
<td>-</td>
</tr>
<tr>
<td>The called functions</td>
<td>rcu_periph_reset_enable / rcu_periph_reset_disable</td>
</tr>
</tbody>
</table>

| Input parameter(in)    | -                                 |
| Output parameter(out)  | -                                 |
| Return value           | -                                 |

Example:

/* reset CTC */
ctc_deinit();

ctc_counter_enable

The description of ctc_counter_enable is shown as below:

Table 3-125. Function ctc_counter_enable

<table>
<thead>
<tr>
<th>Function name</th>
<th>ctc_counter_enable</th>
</tr>
</thead>
<tbody>
<tr>
<td>Function prototype</td>
<td>void ctc_counter_enable(void);</td>
</tr>
<tr>
<td>Function descriptions</td>
<td>enable CTC counter</td>
</tr>
<tr>
<td>Precondition</td>
<td>-</td>
</tr>
<tr>
<td>The called functions</td>
<td>-</td>
</tr>
</tbody>
</table>

| Input parameter(in)    | -                                 |
| Output parameter(out)  | -                                 |
| Return value           | -                                 |

Example:

/* enable CTC trim counter*/
ctc_counter_enable();
The description of `ctc_counter_disable` is shown as below:

Table 3-126. Function `ctc_counter_disable`

<table>
<thead>
<tr>
<th>Function name</th>
<th>ctc_counter_disable</th>
</tr>
</thead>
<tbody>
<tr>
<td>Function prototype</td>
<td><code>void ctc_counter_disable(void);</code></td>
</tr>
<tr>
<td>Function descriptions</td>
<td>disable CTC counter</td>
</tr>
<tr>
<td>Precondition</td>
<td>-</td>
</tr>
<tr>
<td>The called functions</td>
<td>-</td>
</tr>
<tr>
<td>Input parameter(in)</td>
<td>-</td>
</tr>
<tr>
<td>Output parameter(out)</td>
<td>-</td>
</tr>
<tr>
<td>Return value</td>
<td>-</td>
</tr>
</tbody>
</table>

Example:

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

The description of `ctc_irc48m_trim_value_config` is shown as below:

Table 3-127. Function `ctc_irc48m_trim_value_config`

<table>
<thead>
<tr>
<th>Function name</th>
<th>ctc_irc48m_trim_value_config</th>
</tr>
</thead>
<tbody>
<tr>
<td>Function prototype</td>
<td><code>void ctc_irc48m_trim_value_config(uint8_t trim_value);</code></td>
</tr>
<tr>
<td>Function descriptions</td>
<td>configure the IRC48M trim value</td>
</tr>
<tr>
<td>Precondition</td>
<td>-</td>
</tr>
<tr>
<td>The called functions</td>
<td>-</td>
</tr>
<tr>
<td>Input parameter(in)</td>
<td>trim_value 0~63</td>
</tr>
<tr>
<td>Output parameter(out)</td>
<td>-</td>
</tr>
<tr>
<td>Return value</td>
<td>-</td>
</tr>
</tbody>
</table>

Example:

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

The description of `ctc_software_refsource_pulse_generate` is shown as below:

Table 3-128. Function `ctc_software_refsource_pulse_generate`

<table>
<thead>
<tr>
<th>Function name</th>
<th><code>ctc_software_refsource_pulse_generate</code></th>
</tr>
</thead>
<tbody>
<tr>
<td>Function prototype</td>
<td><code>void ctc_software_refsource_pulse_generate(void);</code></td>
</tr>
<tr>
<td>Function descriptions</td>
<td>generate software reference source sync pulse</td>
</tr>
<tr>
<td>Precondition</td>
<td>-</td>
</tr>
<tr>
<td>The called functions</td>
<td>-</td>
</tr>
<tr>
<td>Input parameter(in)</td>
<td>-</td>
</tr>
<tr>
<td>Output parameter(out)</td>
<td>-</td>
</tr>
<tr>
<td>Return value</td>
<td>-</td>
</tr>
</tbody>
</table>

Example:

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

**ctc_hardware_trim_mode_config**

The description of `ctc_hardware_trim_mode_config` is shown as below:

Table 3-129. Function `ctc_hardware_trim_mode_config`

<table>
<thead>
<tr>
<th>Function name</th>
<th><code>ctc_hardware_trim_mode_config</code></th>
</tr>
</thead>
<tbody>
<tr>
<td>Function prototype</td>
<td><code>void ctc_hardware_trim_mode_config(uint32_t hardmode);</code></td>
</tr>
<tr>
<td>Function descriptions</td>
<td>configure hardware automatically trim mode</td>
</tr>
<tr>
<td>Precondition</td>
<td>-</td>
</tr>
<tr>
<td>The called functions</td>
<td>-</td>
</tr>
<tr>
<td>Input parameter(in)</td>
<td>hardmode</td>
</tr>
<tr>
<td>Output parameter(out)</td>
<td>-</td>
</tr>
<tr>
<td>Return value</td>
<td>-</td>
</tr>
</tbody>
</table>

Example:

```c
/* enable CTC hardware trim */
ctc_hardware_trim_mode_config(CTC_HARDWARE_TRIM_MODE_ENABLE);
```
ctc_hardware_trim_mode_config(CTC_HARDWARE_TRIM_MODE_ENABLE);

**ctc_refsource_polarity_config**

The description of `ctc_refsource_polarity_config` is shown as below:

**Table 3-130. Function ctc_refsource_polarity_config**

<table>
<thead>
<tr>
<th>Function name</th>
<th>ctc_refsource_polarity_config</th>
</tr>
</thead>
<tbody>
<tr>
<td>Function prototype</td>
<td>void ctc_refsource_polarity_config(uint32_t polarity);</td>
</tr>
<tr>
<td>Function descriptions</td>
<td>configure reference signal source polarity</td>
</tr>
<tr>
<td>Precondition</td>
<td>-</td>
</tr>
<tr>
<td>The called functions</td>
<td>-</td>
</tr>
<tr>
<td><strong>Input parameter(in)</strong></td>
<td></td>
</tr>
<tr>
<td>polarity</td>
<td>reference signal source polarity</td>
</tr>
<tr>
<td>CTC_REFSOURCE_POLARITY_FALLING</td>
<td>reference signal source polarity is falling edge</td>
</tr>
<tr>
<td>CTC_REFSOURCE_POLARITY_RISING</td>
<td>reference signal source polarity is rising edge</td>
</tr>
<tr>
<td><strong>Output parameter(out)</strong></td>
<td>-</td>
</tr>
<tr>
<td><strong>Return value</strong></td>
<td>-</td>
</tr>
</tbody>
</table>

Example:

/* set reference source polarity */

ttc_refsource_polarity_config(CTC_REFSOURCE_POLARITY_RISING);

**ctc_refsource_signal_select**

The description of `ctc_refsource_signal_select` is shown as below:

**Table 3-131. Function ctc_refsource_signal_select**

<table>
<thead>
<tr>
<th>Function name</th>
<th>ctc_refsource_signal_select</th>
</tr>
</thead>
<tbody>
<tr>
<td>Function prototype</td>
<td>void ctc_refsource_signal_select(uint32_t refs);</td>
</tr>
<tr>
<td>Function descriptions</td>
<td>select reference signal source</td>
</tr>
<tr>
<td>Precondition</td>
<td>-</td>
</tr>
<tr>
<td>The called functions</td>
<td>-</td>
</tr>
<tr>
<td><strong>Input parameter(in)</strong></td>
<td></td>
</tr>
<tr>
<td>refs</td>
<td>reference signal source</td>
</tr>
<tr>
<td>CTC_REFSOURCE_GPIO</td>
<td>GPIO is selected</td>
</tr>
<tr>
<td>CTC_REFSOURCE_LXTAL</td>
<td>LXTAL is selected</td>
</tr>
<tr>
<td>CTC_REFSOURCE_USBD_SOF</td>
<td>USBD_SOF is selected</td>
</tr>
</tbody>
</table>
/* reference signal selection */

ctc_refsource_signal_select(CTC_REFSOURCE_LXTAL);

```c

/* reference signal selection */

ctc_refsource_signal_select(CTC_REFSOURCE_LXTAL);
```

### ctc_refsource_prescaler_config

The description of `ctc_refsource_prescaler_config` is shown as below:

#### Table 3-132. Function ctc_refsource_prescaler_config

<table>
<thead>
<tr>
<th>Function name</th>
<th>ctc_refsource_prescaler_config</th>
</tr>
</thead>
<tbody>
<tr>
<td>Function prototype</td>
<td>void ctc_refsource_prescaler_config(uint32_t prescaler);</td>
</tr>
<tr>
<td>Function descriptions</td>
<td>configure reference signal source prescaler</td>
</tr>
<tr>
<td>Precondition</td>
<td></td>
</tr>
<tr>
<td>The called functions</td>
<td></td>
</tr>
<tr>
<td>Input parameter(in)</td>
<td></td>
</tr>
<tr>
<td>prescaler</td>
<td>Prescaler factor</td>
</tr>
<tr>
<td><code>CTC_REFSOURCE_P_SC_OFF</code></td>
<td>reference signal not divided</td>
</tr>
<tr>
<td><code>CTC_REFSOURCE_P_SC_DIV2</code></td>
<td>reference signal divided by 2</td>
</tr>
<tr>
<td><code>CTC_REFSOURCE_P_SC_DIV4</code></td>
<td>reference signal divided by 4</td>
</tr>
<tr>
<td><code>CTC_REFSOURCE_P_SC_DIV8</code></td>
<td>reference signal divided by 8</td>
</tr>
<tr>
<td><code>CTC_REFSOURCE_P_SC_DIV16</code></td>
<td>reference signal divided by 16</td>
</tr>
<tr>
<td><code>CTC_REFSOURCE_P_SC_DIV32</code></td>
<td>reference signal divided by 32</td>
</tr>
<tr>
<td><code>CTC_REFSOURCE_P_SC_DIV64</code></td>
<td>reference signal divided by 64</td>
</tr>
<tr>
<td><code>CTC_REFSOURCE_P_SC_DIV128</code></td>
<td>reference signal divided by 128</td>
</tr>
</tbody>
</table>

#### Output parameter(out)

- -

#### Return value

- -
/* configure reference signal source prescaler */

ctc_refsource_prescaler_config(CTC_REFSOURCE_PSC_DIV2);

ctc_clock_limit_value_config

The description of ctc_clock_limit_value_config is shown as below:

Table 3-133. Function ctc_clock_limit_value_config

<table>
<thead>
<tr>
<th>Function name</th>
<th>ctc_clock_limit_value_config</th>
</tr>
</thead>
<tbody>
<tr>
<td>Function prototype</td>
<td>void ctc_clock_limit_value_config(uint8_t limit_value);</td>
</tr>
<tr>
<td>Function descriptions</td>
<td>configure clock trim base limit value</td>
</tr>
<tr>
<td>Precondition</td>
<td>-</td>
</tr>
<tr>
<td>The called functions</td>
<td>-</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Input parameter(in)</th>
</tr>
</thead>
<tbody>
<tr>
<td>limit_value</td>
</tr>
<tr>
<td>0x00 - 0xFF</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Output parameter(out)</th>
</tr>
</thead>
<tbody>
<tr>
<td>-</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Return value</th>
</tr>
</thead>
<tbody>
<tr>
<td>-</td>
</tr>
</tbody>
</table>

Example:

/* configure clock trim base limit value */

ctc_clock_limit_value_config(0x1F);

ctc_counter_reload_value_config

The description of ctc_counter_reload_value_config is shown as below:

Table 3-134. Function ctc_counter_reload_value_config

<table>
<thead>
<tr>
<th>Function name</th>
<th>ctc_counter_reload_value_config</th>
</tr>
</thead>
<tbody>
<tr>
<td>Function prototype</td>
<td>void ctc_counter_reload_value_config(uint16_t reload_value);</td>
</tr>
<tr>
<td>Function descriptions</td>
<td>configure CTC counter reload value</td>
</tr>
<tr>
<td>Precondition</td>
<td>-</td>
</tr>
<tr>
<td>The called functions</td>
<td>-</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Input parameter(in)</th>
</tr>
</thead>
<tbody>
<tr>
<td>reload_value</td>
</tr>
<tr>
<td>0x0000 - 0xFFFF</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Output parameter(out)</th>
</tr>
</thead>
<tbody>
<tr>
<td>-</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Return value</th>
</tr>
</thead>
<tbody>
<tr>
<td>-</td>
</tr>
</tbody>
</table>

Example:

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

**ctc_counter_capture_value_read**

The description of ctc_counter_capture_value_read is shown as below:

<table>
<thead>
<tr>
<th>Function name</th>
<th>uint16_t ctc_counter_capture_value_read(void);</th>
</tr>
</thead>
<tbody>
<tr>
<td>Function prototype</td>
<td>uint16_t ctc_counter_capture_value_read(void);</td>
</tr>
<tr>
<td>Function descriptions</td>
<td>read CTC counter capture value when reference sync pulse occurred</td>
</tr>
<tr>
<td>Precondition</td>
<td>-</td>
</tr>
<tr>
<td>The called functions</td>
<td>-</td>
</tr>
<tr>
<td>Input parameter(in)</td>
<td>-</td>
</tr>
<tr>
<td>Output parameter(out)</td>
<td>-</td>
</tr>
<tr>
<td>Return value</td>
<td>the 16-bit CTC counter capture value (0x0000 - 0xFFFF)</td>
</tr>
</tbody>
</table>

Example:

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

**ctc_counter_direction_read**

The description of ctc_counter_direction_read is shown as below:

<table>
<thead>
<tr>
<th>Function name</th>
<th>ctc_counter_direction_read</th>
</tr>
</thead>
<tbody>
<tr>
<td>Function prototype</td>
<td>FlagStatus ctc_counter_direction_read(void);</td>
</tr>
<tr>
<td>Function descriptions</td>
<td>read CTC trim counter direction when reference sync pulse occurred</td>
</tr>
<tr>
<td>Precondition</td>
<td>-</td>
</tr>
<tr>
<td>The called functions</td>
<td>-</td>
</tr>
<tr>
<td>Input parameter(in)</td>
<td>-</td>
</tr>
<tr>
<td>Output parameter(out)</td>
<td>-</td>
</tr>
<tr>
<td>Return value</td>
<td>FlagStatus SET / RESET</td>
</tr>
</tbody>
</table>

Example:

```c
/* read ctc counter direction */
```
FlagStatus ctc_direction = SET;

ctc_direction = ctc_counter_direction_read();

**ctc_counter_reload_value_read**

The description of **ctc_counter_reload_value_read** is shown as below:

**Table 3-137. Function ctc_counter_reload_value_read**

<table>
<thead>
<tr>
<th>Function name</th>
<th>ctc_counter_reload_value_read</th>
</tr>
</thead>
<tbody>
<tr>
<td>Function prototype</td>
<td>uint16_t ctc_counter_reload_value_read(void);</td>
</tr>
<tr>
<td>Function descriptions</td>
<td>read CTC counter reload value</td>
</tr>
<tr>
<td>Precondition</td>
<td>-</td>
</tr>
<tr>
<td>The called functions</td>
<td>-</td>
</tr>
<tr>
<td>Input parameter(in)</td>
<td>-</td>
</tr>
<tr>
<td>Output parameter(out)</td>
<td>-</td>
</tr>
<tr>
<td>Return value</td>
<td>uint16_t - Read 16-bit data of counter reload value (0x0000 - 0xFFFF)</td>
</tr>
</tbody>
</table>

Example:

`/* read CTC counter reload value */`

`uint16_t ctc_reload_value = 0;`

`ctc_reload_value = ctc_counter_reload_value_read();`

**ctc irc48m_trim_value_read**

The description of **ctc irc48m Trim_value_read** is shown as below:

**Table 3-138. Function ctc irc48m Trim_value_read**

<table>
<thead>
<tr>
<th>Function name</th>
<th>ctc irc48m Trim_value_read</th>
</tr>
</thead>
<tbody>
<tr>
<td>Function prototype</td>
<td>uint8_t ctc irc48m Trim_value_read(void);</td>
</tr>
<tr>
<td>Function descriptions</td>
<td>read the IRC48M trim value</td>
</tr>
<tr>
<td>Precondition</td>
<td>-</td>
</tr>
<tr>
<td>The called functions</td>
<td>-</td>
</tr>
<tr>
<td>Input parameter(in)</td>
<td>-</td>
</tr>
<tr>
<td>Output parameter(out)</td>
<td>-</td>
</tr>
<tr>
<td>Return value</td>
<td>uint8_t - the 7-bit IRC48M trim value (0-63)</td>
</tr>
</tbody>
</table>

Example:
/* read the IRC48M trim value */

uint8_t ctc_trim_value = 0;

ctc_trim_value = ctc irc48m trim_value_read();

**ctc_flag_get**

The description of ctc_flag_get is shown as below:

Table 3-139. Function ctc_flag_get

<table>
<thead>
<tr>
<th>Function name</th>
<th>ctc_flag_get</th>
</tr>
</thead>
<tbody>
<tr>
<td>Function prototype</td>
<td>FlagStatus ctc_flag_get(uint32_t flag);</td>
</tr>
<tr>
<td>Function descriptions</td>
<td>get CTC flag</td>
</tr>
<tr>
<td>Precondition</td>
<td>-</td>
</tr>
<tr>
<td>The called functions</td>
<td>-</td>
</tr>
</tbody>
</table>

### Input parameter(in)

<table>
<thead>
<tr>
<th>flag</th>
<th>CTC status flag</th>
</tr>
</thead>
<tbody>
<tr>
<td>CTC_FLAG_CKOK</td>
<td>clock trim OK interrupt flag</td>
</tr>
<tr>
<td>CTC_FLAG_CKWARN</td>
<td>clock trim warning interrupt flag</td>
</tr>
<tr>
<td>CTC_FLAG_ERR</td>
<td>error interrupt flag</td>
</tr>
<tr>
<td>CTC_FLAG_EREF</td>
<td>expect reference interrupt flag</td>
</tr>
<tr>
<td>CTC_FLAG_CKERR</td>
<td>clock trim error bit</td>
</tr>
<tr>
<td>CTC_FLAG_REFMISS</td>
<td>reference sync pulse miss flag</td>
</tr>
<tr>
<td>CTC_FLAG_TRIMERR</td>
<td>trim value error flag</td>
</tr>
</tbody>
</table>

### Output parameter(out)

- |

### Return value

FlagStatus SET or RESET

Example:

/* get CTC flag status */

FlagStatus state = ctc_flag_get(CTC_FLAG_CKOK);

**ctc_flag_clear**

The description of ctc_flag_clear is shown as below:

Table 3-140. Function ctc_flag_clear

<table>
<thead>
<tr>
<th>Function name</th>
<th>ctc_flag_clear</th>
</tr>
</thead>
<tbody>
<tr>
<td>Function prototype</td>
<td>void ctc_flag_clear(uint32_t flag);</td>
</tr>
<tr>
<td>Function descriptions</td>
<td>clear CTC flag</td>
</tr>
<tr>
<td>Precondition</td>
<td>-</td>
</tr>
<tr>
<td>The called functions</td>
<td>-</td>
</tr>
</tbody>
</table>

### Input parameter(in)
### CTC status flag

<table>
<thead>
<tr>
<th>Flag</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CTC_FLAG_CKOK</td>
<td>clock trim OK interrupt flag</td>
</tr>
<tr>
<td>CTC_FLAG_CKWARN</td>
<td>clock trim warning interrupt flag</td>
</tr>
<tr>
<td>CTC_FLAG_ERR</td>
<td>error interrupt flag</td>
</tr>
<tr>
<td>CTC_FLAG_EREF</td>
<td>expect reference interrupt flag</td>
</tr>
<tr>
<td>CTC_FLAG_CKERR</td>
<td>clock trim error bit</td>
</tr>
<tr>
<td>CTC_FLAG_REFMISS</td>
<td>reference sync pulse miss flag</td>
</tr>
<tr>
<td>CTC_FLAG_TRIMERR</td>
<td>trim value error flag</td>
</tr>
</tbody>
</table>

#### Example:

```c
/* clear CTC flag status */
ctc_flag_clear(CTC_FLAG_CKOK);
```

### `ctc_interrupt_enable`

The description of `ctc_interrupt_enable` is shown as below:

#### Table 3-141. Function `ctc_interrupt_enable`

<table>
<thead>
<tr>
<th>Function name</th>
<th>ctc_interrupt_enable</th>
</tr>
</thead>
<tbody>
<tr>
<td>Function prototype</td>
<td>void ctc_interrupt_enable(uint32_t interrupt);</td>
</tr>
<tr>
<td>Function descriptions</td>
<td>enable the CTC interrupt</td>
</tr>
<tr>
<td>Precondition</td>
<td>-</td>
</tr>
<tr>
<td>The called functions</td>
<td>-</td>
</tr>
</tbody>
</table>

#### Input parameter (in)

<table>
<thead>
<tr>
<th>interrupt</th>
<th>CTC interrupt</th>
</tr>
</thead>
<tbody>
<tr>
<td>CTC_INT_CKOK</td>
<td>clock trim OK interrupt</td>
</tr>
<tr>
<td>CTC_INT_CKWARN</td>
<td>clock trim warning interrupt</td>
</tr>
<tr>
<td>CTC_INT_ERR</td>
<td>error interrupt</td>
</tr>
<tr>
<td>CTC_INT_EREF</td>
<td>expect reference interrupt</td>
</tr>
</tbody>
</table>

#### Output parameter (out)

- -

#### Return value

- -

#### Example:

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

The description of ctc_interrupt_disable is shown as below:

Table 3-142. Function ctc_interrupt_disable

<table>
<thead>
<tr>
<th>Function name</th>
<th>ctc_interrupt_disable</th>
</tr>
</thead>
<tbody>
<tr>
<td>Function prototype</td>
<td>void ctc_interrupt_disable(uint32_t interrupt);</td>
</tr>
<tr>
<td>Function descriptions</td>
<td>disable the CTC interrupt</td>
</tr>
<tr>
<td>Precondition</td>
<td>-</td>
</tr>
<tr>
<td>The called functions</td>
<td>-</td>
</tr>
</tbody>
</table>

**Input parameter (in)**

<table>
<thead>
<tr>
<th>interrupt</th>
<th>CTC interrupt</th>
</tr>
</thead>
<tbody>
<tr>
<td>CTC_INT_CKOK</td>
<td>clock trim OK interrupt</td>
</tr>
<tr>
<td>CTC_INT_CKWARN</td>
<td>clock trim warning interrupt</td>
</tr>
<tr>
<td>CTC_INT_ERR</td>
<td>error interrupt</td>
</tr>
<tr>
<td>CTC_INT_EREF</td>
<td>expect reference interrupt</td>
</tr>
</tbody>
</table>

**Output parameter (out)**

<table>
<thead>
<tr>
<th>Return value</th>
<th>-</th>
</tr>
</thead>
</table>

Example:

/* disable CTC clock trim OK interrupt */

ctc_interrupt_disable(CTC_INT_CKOK);

ctc_interrupt_flag_get

The description of ctc_interrupt_flag_get is shown as below:

Table 3-143. Function ctc_interrupt_flag_get

<table>
<thead>
<tr>
<th>Function name</th>
<th>ctc_interrupt_flag_get</th>
</tr>
</thead>
<tbody>
<tr>
<td>Function prototype</td>
<td>FlagStatus ctc_interrupt_flag_get(uint32_t int_flag);</td>
</tr>
<tr>
<td>Function descriptions</td>
<td>get CTC interrupt flag</td>
</tr>
<tr>
<td>Precondition</td>
<td>-</td>
</tr>
<tr>
<td>The called functions</td>
<td>-</td>
</tr>
</tbody>
</table>

**Input parameter (in)**

<table>
<thead>
<tr>
<th>int_flag</th>
<th>CTC interrupt flag</th>
</tr>
</thead>
<tbody>
<tr>
<td>CTC_INT_FLAG_CKOK</td>
<td>clock trim OK interrupt</td>
</tr>
<tr>
<td>CTC_INT_FLAG_CKWARN</td>
<td>clock trim warning interrupt</td>
</tr>
<tr>
<td>CTC_INT_FLAG_ERR</td>
<td>error interrupt</td>
</tr>
<tr>
<td>CTC_INT_FLAG_EREF</td>
<td>expect reference interrupt</td>
</tr>
<tr>
<td>CTC_INT_FLAG_CKE</td>
<td>clock trim error bit interrupt</td>
</tr>
</tbody>
</table>
### CTC_INT_FLAG_REF_MISS
- reference sync pulse miss interrupt

### CTC_INT_FLAG_TRIM_ERR
- trim value error interrupt

#### Output parameter (out)
- 
- 

#### Return value
- FlagStatus
- SET or RESET

Example:

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

FlagStatus state = ctc_interrupt_flag_get(CTC_INT_FLAG_CKOK);
```

#### ctc_interrupt_flag_clear

The description of `ctc_interrupt_flag_clear` is shown as below:

**Table 3-144. Function ctc_interrupt_flag_clear**

<table>
<thead>
<tr>
<th>Function name</th>
<th>ctc_interrupt_flag_clear</th>
</tr>
</thead>
<tbody>
<tr>
<td>Function prototype</td>
<td>void ctc_interrupt_flag_clear(uint32_t int_flag);</td>
</tr>
<tr>
<td>Function descriptions</td>
<td>clear CTC interrupt flag</td>
</tr>
<tr>
<td>Precondition</td>
<td>-</td>
</tr>
<tr>
<td>The called functions</td>
<td>-</td>
</tr>
</tbody>
</table>

#### Input parameter (in)

<table>
<thead>
<tr>
<th>int_flag</th>
<th>CTC interrupt flag</th>
</tr>
</thead>
<tbody>
<tr>
<td>CTC_INT_FLAG_CKO</td>
<td>clock trim OK interrupt</td>
</tr>
<tr>
<td>CTC_INT_FLAG_CKW</td>
<td>clock trim warning interrupt</td>
</tr>
<tr>
<td>CTC_INT_FLAG_ERR</td>
<td>error interrupt</td>
</tr>
<tr>
<td>CTC_INT_FLAG_EREF</td>
<td>expect reference interrupt</td>
</tr>
<tr>
<td>CTC_INT_FLAG_CKE</td>
<td>clock trim error bit interrupt</td>
</tr>
<tr>
<td>CTC_INT_FLAG_REF_MISS</td>
<td>reference sync pulse miss interrupt</td>
</tr>
<tr>
<td>CTC_INT_FLAG_TRIM_ERR</td>
<td>trim value error interrupt</td>
</tr>
</tbody>
</table>

#### Output parameter (out)
- 
- 

#### Return value
- 
- 

Example:
/* clear CTC interrupt flag status */

ctc_interrupt_flag_clear(CTC_INT_FLAG_CKOK);

3.7. **DBG**

The DBG hold unit helps debugger to debug power saving mode. The DBG registers are listed in chapter **3.7.1**, the DBG firmware functions are introduced in chapter **3.7.2**.

### 3.7.1. Descriptions of Peripheral registers

DBG registers are listed in the table shown as below:

<table>
<thead>
<tr>
<th>Registers</th>
<th>Descriptions</th>
</tr>
</thead>
<tbody>
<tr>
<td>DBG_ID</td>
<td>DBG ID code register</td>
</tr>
<tr>
<td>DBG_CNTL0</td>
<td>DBG control register0</td>
</tr>
<tr>
<td>DBG_CNTL1</td>
<td>DBG control register1</td>
</tr>
</tbody>
</table>

### 3.7.2. Descriptions of Peripheral functions

DBG firmware functions are listed in the table shown as below:

<table>
<thead>
<tr>
<th>Function name</th>
<th>Function description</th>
</tr>
</thead>
<tbody>
<tr>
<td>dbg_deinit</td>
<td>reset DBG register</td>
</tr>
<tr>
<td>dbg_id_get</td>
<td>read DBG_ID code register</td>
</tr>
<tr>
<td>dbg_low_power_enable</td>
<td>enable low power behavior when the MCU is in debug mode</td>
</tr>
<tr>
<td>dbg_low_power_disable</td>
<td>disable low power behavior when the MCU is in debug mode</td>
</tr>
<tr>
<td>dbg_periph_enable</td>
<td>enable peripheral behavior when the MCU is in debug mode</td>
</tr>
<tr>
<td>dbg_periph_disable</td>
<td>disable peripheral behavior when the MCU is in debug mode</td>
</tr>
</tbody>
</table>

**Enum dbg_periph_enum**

<table>
<thead>
<tr>
<th>Member name</th>
<th>Function description</th>
</tr>
</thead>
<tbody>
<tr>
<td>DBG_FWDGT_HOLD</td>
<td>debug FWDGT kept when core is halted</td>
</tr>
<tr>
<td>DBG_WWDGT_HOLD</td>
<td>debug WWDGT kept when core is halted</td>
</tr>
<tr>
<td>DBG_TIMER1_HOLD</td>
<td>hold TIMER1 counter when core is halted</td>
</tr>
<tr>
<td>DBG_TIMER2_HOLD</td>
<td>hold TIMER2 counter when core is halted</td>
</tr>
<tr>
<td>DBG_I2C0_HOLD</td>
<td>hold I2C0 smbus when core is halted</td>
</tr>
<tr>
<td>DBG_I2C1_HOLD</td>
<td>hold I2C1 smbus when core is halted</td>
</tr>
<tr>
<td>DBG_TIMER5_HOLD</td>
<td>hold TIMERS5 counter when core is halted</td>
</tr>
<tr>
<td>DBG_TIMER6_HOLD</td>
<td>hold TIMERS6 counter when core is halted</td>
</tr>
</tbody>
</table>
### Member name | Function description
---|---
DBG_TIMER8_HOLD | hold TIMER8 counter when core is halted
DBG_TIMER11_HOLD | hold TIMER11 counter when core is halted
DBG_RTC_HOLD | hold RTC counter when core is halted
DBG_LPTIMER_HOLD | hold LPTIMER counter when core is halted
DBG_I2C2_HOLD | hold I2C2 smbus when core is halted

#### dbg_deinit

The description of dbg_deinit is shown as below:

**Table 3-148. Function dbg_deinit**

<table>
<thead>
<tr>
<th>Function name</th>
<th>Function prototype</th>
<th>Function descriptions</th>
<th>Precondition</th>
<th>The called functions</th>
</tr>
</thead>
<tbody>
<tr>
<td>dbg_deinit</td>
<td>void dbg_deinit(void);</td>
<td>deinitialize the DBG</td>
<td>-</td>
<td>-</td>
</tr>
</tbody>
</table>

**Input parameter(in)**

-  

**Output parameter(out)**

-  

**Return value**

-  

**Example:**

```c
/* deinitialize the DBG*/

dbg_deinit();
```

#### dbg_id_get

The description of dbg_id_get is shown as below:

**Table 3-149. Function dbg_id_get**

<table>
<thead>
<tr>
<th>Function name</th>
<th>Function prototype</th>
<th>Function descriptions</th>
<th>Precondition</th>
<th>The called functions</th>
</tr>
</thead>
<tbody>
<tr>
<td>dbg_id_get</td>
<td>uint32_t dbg_id_get(void);</td>
<td>Read DBG_ID code register</td>
<td>-</td>
<td>-</td>
</tr>
</tbody>
</table>

**Input parameter(in)**

-  

**Output parameter(out)**

-  

**Return value**

uint32_t DBG_ID code (0-0xFFFFFFFF)
Example:

```
/* read DBG_ID code register */

uint32_t id_value = 0;
id_value = dbg_id_get();
```

**dbg_low_power_enable**

The description of `dbg_low_power_enable` is shown as below:

<table>
<thead>
<tr>
<th>Function name</th>
<th>dbg_low_power_enable</th>
</tr>
</thead>
<tbody>
<tr>
<td>Function prototype</td>
<td>void dbg_low_power_enable(uint32_t dbg_low_power);</td>
</tr>
<tr>
<td>Function descriptions</td>
<td>Enable low power behavior when the mcu is in debug mode</td>
</tr>
<tr>
<td>Precondition</td>
<td>-</td>
</tr>
<tr>
<td>The called functions</td>
<td>-</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Input parameter(in)</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>dbg_low_power</code></td>
</tr>
<tr>
<td><code>DBG_LOW_POWER_SLEEP</code></td>
</tr>
<tr>
<td><code>DBG_LOW_POWER_DEEPSLEEP</code></td>
</tr>
<tr>
<td><code>DBG_LOW_POWER_STANDBY</code></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Output parameter(out)</th>
</tr>
</thead>
<tbody>
<tr>
<td>-</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Return value</th>
</tr>
</thead>
<tbody>
<tr>
<td>-</td>
</tr>
</tbody>
</table>

Example:

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

dbg_low_power_enable(DBG_LOW_POWER_SLEEP);
```

**dbg_low_power_disable**

The description of `dbg_low_power_disable` is shown as below:

<table>
<thead>
<tr>
<th>Function name</th>
<th>dbg_low_power_disable</th>
</tr>
</thead>
<tbody>
<tr>
<td>Function prototype</td>
<td>void dbg_low_power_disable(uint32_t dbg_low_power);</td>
</tr>
<tr>
<td>Function descriptions</td>
<td>Disable low power behavior when the mcu is in debug mode</td>
</tr>
<tr>
<td>Precondition</td>
<td>-</td>
</tr>
<tr>
<td>The called functions</td>
<td>-</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Input parameter(in)</th>
</tr>
</thead>
</table>

122
**dbg_low_power**

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>DBG_LOW_POWER_SLEEP</strong></td>
<td>keep debugger connection during sleep mode</td>
</tr>
<tr>
<td><strong>DBG_LOW_POWER_DEEPSLEEP</strong></td>
<td>keep debugger connection during deepsleep mode</td>
</tr>
<tr>
<td><strong>DBG_LOW_POWER_STANDBY</strong></td>
<td>keep debugger connection during standby mode</td>
</tr>
</tbody>
</table>

**Output parameter (out)**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>-</td>
<td>-</td>
</tr>
</tbody>
</table>

| Return value | - |

**Example:**

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

dbg_periph_enable(DBG_TIMER1_HOLD);
```

**dbg_periph_enable**

The description of **dbg_periph_enable** is shown as below:

**Table 3-152. Function dbg_periph_enable**

<table>
<thead>
<tr>
<th>Function name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>dbg_periph_enable</strong></td>
<td>Enable peripheral behavior when the mcu is in debug mode</td>
</tr>
</tbody>
</table>

**Function prototype**

```
void dbg_periph_enable(dbg_periph_enum dbg_periph);
```

**Precondition**

-  

**The called functions**

<table>
<thead>
<tr>
<th>Input parameter (in)</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>dbg_periph</strong></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>DBG_FWDGT_HOLD</strong></td>
<td>debug FWDGT kept when core is halted</td>
</tr>
<tr>
<td><strong>DBG_WWDGT_HOLD</strong></td>
<td>debug WWDGT kept when core is halted</td>
</tr>
<tr>
<td><strong>DBG_TIMERx_HOLD</strong></td>
<td>x=1,2,5,6,8,11, hold TIMERx counter when core is halted</td>
</tr>
<tr>
<td><strong>DBG_I2Cx_HOLD</strong></td>
<td>x=0,1,2, hold I2Cx smbus when core is halted</td>
</tr>
<tr>
<td><strong>DBG_RTC_HOLD</strong></td>
<td>hold RTC counter when core is halted</td>
</tr>
<tr>
<td><strong>DBG_LPTIMER_HOLD</strong></td>
<td>hold LPTIMER counter when core is halted</td>
</tr>
</tbody>
</table>

**Output parameter (out)**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>-</td>
<td>-</td>
</tr>
</tbody>
</table>

| Return value | - |

**Example:**

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

dbg_low_power_disable(DBG_LOW_POWER_SLEEP);
```
**dbg_periph_disable**

The description of `dbg_periph_disable` is shown as below:

<table>
<thead>
<tr>
<th>Function name</th>
<th>dbg_periph_disable</th>
</tr>
</thead>
<tbody>
<tr>
<td>Function prototype</td>
<td>void dbg_periph_disable(dbg_periph_enum dbg_periph);</td>
</tr>
<tr>
<td>Function descriptions</td>
<td>Disable peripheral behavior when the mcu is in debug mode</td>
</tr>
<tr>
<td>Precondition</td>
<td>-</td>
</tr>
<tr>
<td>The called functions</td>
<td>-</td>
</tr>
</tbody>
</table>

**Input parameter(in)**

<table>
<thead>
<tr>
<th><code>dbg_periph</code></th>
<th>peripheral refer to <a href="#">Table 3-147, Enum dbg_periph_enum</a></th>
</tr>
</thead>
<tbody>
<tr>
<td><code>DBG_FWDGT_HOLD</code></td>
<td>debug FWDGT kept when core is halted</td>
</tr>
<tr>
<td><code>DBG_WWDGT_HOLD</code></td>
<td>debug WWDGT kept when core is halted</td>
</tr>
<tr>
<td><code>DBG_TIMERx_HOLD</code></td>
<td>x=1,2,5,6,8,11, hold TIMERx counter when core is halted</td>
</tr>
<tr>
<td><code>DBG_I2Cx_HOLD</code></td>
<td>x=0,1,2, hold I2Cx smbus when core is halted</td>
</tr>
<tr>
<td><code>DBG_RTC_HOLD</code></td>
<td>hold RTC counter when core is halted</td>
</tr>
<tr>
<td><code>DBG_LPTIMER_HOLD</code></td>
<td>hold LPTIMER counter when core is halted</td>
</tr>
</tbody>
</table>

**Output parameter(out)**

- -

**Return value**

- -

Example:

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

dbg_periph_disable(DBG_TIMER1_HOLD);
```

3.8. **DAC**

The Digital-to-analog converter converts 12-bit digital data to a voltage on the external pins. The DAC registers are listed in chapter 3.8.1, the DAC firmware functions are introduced in chapter 3.8.2.

3.8.1. **Descriptions of Peripheral registers**

DAC registers are listed in the table shown as below:

<table>
<thead>
<tr>
<th>Registers</th>
<th>Descriptions</th>
</tr>
</thead>
<tbody>
<tr>
<td>DAC_CTL0</td>
<td>DAC control register 0</td>
</tr>
<tr>
<td>DAC_SWT</td>
<td>DAC software trigger register</td>
</tr>
<tr>
<td>DAC_R12DH</td>
<td>DAC_OUT 12-bit right-aligned data holding register</td>
</tr>
<tr>
<td>DAC_L12DH</td>
<td>DAC_OUT 12-bit left-aligned data holding register</td>
</tr>
</tbody>
</table>
### DAC Registers Descriptions

<table>
<thead>
<tr>
<th>Registers</th>
<th>Descriptions</th>
</tr>
</thead>
<tbody>
<tr>
<td>DAC_R8DH</td>
<td>DAC_OUT 8-bit right-aligned data holding register</td>
</tr>
<tr>
<td>DAC_DO</td>
<td>DAC_OUT data output register</td>
</tr>
<tr>
<td>DAC_STAT0</td>
<td>DAC Status register 0</td>
</tr>
</tbody>
</table>

#### 3.8.2. Descriptions of Peripheral functions

DAC firmware functions are listed in the table shown as below:

**Table 3-155. DAC firmware function**

<table>
<thead>
<tr>
<th>Function name</th>
<th>Function description</th>
</tr>
</thead>
<tbody>
<tr>
<td>dac_deinit</td>
<td>deinitialize DAC</td>
</tr>
<tr>
<td>dac_enable</td>
<td>enable DAC</td>
</tr>
<tr>
<td>dac_disable</td>
<td>disable DAC</td>
</tr>
<tr>
<td>dac_dma_enable</td>
<td>dac_dma_enable</td>
</tr>
<tr>
<td>dac_dma_disable</td>
<td>dac_dma_disable</td>
</tr>
<tr>
<td>dac_output_buffer_enable</td>
<td>enable DAC output buffer</td>
</tr>
<tr>
<td>dac_output_buffer_disable</td>
<td>disable DAC output buffer</td>
</tr>
<tr>
<td>dac_output_value_get</td>
<td>get DAC output value</td>
</tr>
<tr>
<td>dac_data_set</td>
<td>set DAC data holding register value</td>
</tr>
<tr>
<td>dac_trigger_enable</td>
<td>enable DAC trigger</td>
</tr>
<tr>
<td>dac_trigger_disable</td>
<td>disable DAC trigger</td>
</tr>
<tr>
<td>dac_trigger_source_config</td>
<td>configure DAC trigger source</td>
</tr>
<tr>
<td>dac_software_trigger_enable</td>
<td>enable DAC software trigger</td>
</tr>
<tr>
<td>dac_software_trigger_disable</td>
<td>disable DAC software trigger</td>
</tr>
<tr>
<td>dac_wave_mode_config</td>
<td>configure DAC wave mode</td>
</tr>
<tr>
<td>dac_wave_bit_width_config</td>
<td>configure DAC wave bit width</td>
</tr>
<tr>
<td>dac_lfsr_noise_config</td>
<td>configure DAC LFSR noise mode</td>
</tr>
<tr>
<td>dac_triangle_noise_config</td>
<td>configure DAC triangle noise mode</td>
</tr>
<tr>
<td>dac_flag_get</td>
<td>get the specified DAC flag (DAC DMA underrun flag)</td>
</tr>
<tr>
<td>dac_flag_clear</td>
<td>clear the specified DAC flag (DAC DMA underrun flag)</td>
</tr>
<tr>
<td>dac_interrupt_enable</td>
<td>enable DAC interrupt (DAC DMA underrun interrupt)</td>
</tr>
<tr>
<td>dac_interrupt_disable</td>
<td>disable DAC interrupt (DAC DMA underrun interrupt)</td>
</tr>
<tr>
<td>dac_interrupt_flag_get</td>
<td>get DAC interrupt flag</td>
</tr>
<tr>
<td>dac_interrupt_flag_clear</td>
<td>clear DAC interrupt flag</td>
</tr>
</tbody>
</table>

**dac_deinit**

The description of dac_deinit is shown as below:

**Table 3-156. Function dac_deinit**

<table>
<thead>
<tr>
<th>Function name</th>
<th>dac_deinit</th>
</tr>
</thead>
<tbody>
<tr>
<td>Function prototype</td>
<td>void dac_deinit(void)</td>
</tr>
<tr>
<td>Function descriptions</td>
<td>deinitialize DAC</td>
</tr>
</tbody>
</table>
The called functions
rcu_periph_reset_enable / rcu_periph_reset_disable

### Input parameter(in)
- -

### Output parameter(out)
- -

### Return value
- -

Example:

```c
/* deinitialize DAC */
dac_deinit();
```

### dac_enable

The description of dac_enable is shown as below:

Table 3-157. Function dac_enable

<table>
<thead>
<tr>
<th>Function name</th>
<th>dac_enable</th>
</tr>
</thead>
<tbody>
<tr>
<td>Function prototype</td>
<td>void dac_enable(void)</td>
</tr>
<tr>
<td>Function descriptions</td>
<td>enable DAC</td>
</tr>
<tr>
<td>Precondition</td>
<td>-</td>
</tr>
<tr>
<td>The called functions</td>
<td>-</td>
</tr>
</tbody>
</table>

### Input parameter(in)

- -

### Output parameter(out)

- -

### Return value

- -

Example:

```c
/* enable DAC */
dac_enable();
```

### dac_disable

The description of dac_disable is shown as below:

Table 3-158. Function dac_disable

<table>
<thead>
<tr>
<th>Function name</th>
<th>dac_disable</th>
</tr>
</thead>
<tbody>
<tr>
<td>Function prototype</td>
<td>void dac_disable(void)</td>
</tr>
<tr>
<td>Function descriptions</td>
<td>disable DAC</td>
</tr>
<tr>
<td>Precondition</td>
<td>-</td>
</tr>
</tbody>
</table>
The called functions

Input parameter (in)
- -

Output parameter (out)
- -

Return value
- -

Example:

/* disable DAC */
dac_disable();

dac_dma_enable

The description of dac_dma_enable is shown as below:

Table 3-159. Function dac_dma_enable

<table>
<thead>
<tr>
<th>Function name</th>
<th>dac_dma_enable</th>
</tr>
</thead>
<tbody>
<tr>
<td>Function prototype</td>
<td>void dac_dma_enable(void)</td>
</tr>
<tr>
<td>Function descriptions</td>
<td>enable DAC DMA function</td>
</tr>
<tr>
<td>Precondition</td>
<td>-</td>
</tr>
<tr>
<td>The called functions</td>
<td>-</td>
</tr>
<tr>
<td>Input parameter (in)</td>
<td>-</td>
</tr>
<tr>
<td>Output parameter (out)</td>
<td>-</td>
</tr>
<tr>
<td>Return value</td>
<td>-</td>
</tr>
</tbody>
</table>

Example:

/* enable DAC DMA function */
dac_dma_enable();

dac_dma_disable

The description of dac_dma_disable is shown as below:

Table 3-160. Function dac_dma_disable

<table>
<thead>
<tr>
<th>Function name</th>
<th>dac_dma_disable</th>
</tr>
</thead>
<tbody>
<tr>
<td>Function prototype</td>
<td>void dac_dma_disable(void)</td>
</tr>
<tr>
<td>Function descriptions</td>
<td>disable DAC DMA function</td>
</tr>
<tr>
<td>Precondition</td>
<td>-</td>
</tr>
<tr>
<td>The called functions</td>
<td>-</td>
</tr>
</tbody>
</table>
Example:

/* disable DAC DMA function */
dac_dma_disable();

**dac_output_buffer_enable**

The description of dac_output_buffer_enable is shown as below:

<table>
<thead>
<tr>
<th>Function name</th>
<th>dac_output_buffer_enable</th>
</tr>
</thead>
<tbody>
<tr>
<td>Function prototype</td>
<td>void dac_output_buffer_enable(void)</td>
</tr>
<tr>
<td>Function descriptions</td>
<td>enable DAC output buffer</td>
</tr>
<tr>
<td>Precondition</td>
<td>-</td>
</tr>
<tr>
<td>The called functions</td>
<td>-</td>
</tr>
<tr>
<td>Input parameter(in)</td>
<td>-</td>
</tr>
<tr>
<td>Output parameter(out)</td>
<td>-</td>
</tr>
<tr>
<td>Return value</td>
<td>-</td>
</tr>
</tbody>
</table>

Example:

/* enable DAC output buffer */
dac_output_buffer_enable();

**dac_output_buffer_disable**

The description of dac_output_buffer_disable is shown as below:

<table>
<thead>
<tr>
<th>Function name</th>
<th>dac_output_buffer_disable</th>
</tr>
</thead>
<tbody>
<tr>
<td>Function prototype</td>
<td>void dac_output_buffer_disable(void)</td>
</tr>
<tr>
<td>Function descriptions</td>
<td>disable DAC output buffer</td>
</tr>
<tr>
<td>Precondition</td>
<td>-</td>
</tr>
<tr>
<td>The called functions</td>
<td>-</td>
</tr>
<tr>
<td>Input parameter(in)</td>
<td>-</td>
</tr>
</tbody>
</table>
Example:

```c
/* disable DAC output buffer */
dac_output_buffer_disable();
```

**dac_output_value_get**

The description of dac_output_value_get is shown as below:

**Table 3-163. Function dac_output_value_get**

<table>
<thead>
<tr>
<th>Function name</th>
<th>dac_output_value_get</th>
</tr>
</thead>
<tbody>
<tr>
<td>Function prototype</td>
<td>uint16_t dac_output_value_get(void)</td>
</tr>
<tr>
<td>Function descriptions</td>
<td>get DAC output value</td>
</tr>
<tr>
<td>Precondition</td>
<td>-</td>
</tr>
<tr>
<td>The called functions</td>
<td>-</td>
</tr>
<tr>
<td>Input parameter(in)</td>
<td>-</td>
</tr>
<tr>
<td>Output parameter(out)</td>
<td>-</td>
</tr>
<tr>
<td>Return value</td>
<td>-</td>
</tr>
</tbody>
</table>

| uint16_t               | DAC output data (0~4095)               |

Example:

```c
/* get DAC output value */
uint16_t data = 0;
data = dac_output_value_get();
```

**dac_data_set**

The description of dac_data_set is shown as below:

**Table 3-164. Function dac_data_set**

<table>
<thead>
<tr>
<th>Function name</th>
<th>dac_data_set</th>
</tr>
</thead>
<tbody>
<tr>
<td>Function prototype</td>
<td>void dac_data_set(uint32_t dac_align, uint16_t data)</td>
</tr>
<tr>
<td>Function descriptions</td>
<td>set DAC data holding register value</td>
</tr>
<tr>
<td>Precondition</td>
<td>-</td>
</tr>
<tr>
<td>The called functions</td>
<td>-</td>
</tr>
<tr>
<td>Input parameter(in)</td>
<td>-</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>uint16_t</th>
<th>-</th>
</tr>
</thead>
</table>
Input parameter(in)

<table>
<thead>
<tr>
<th>dac_align</th>
<th>DAC align mode</th>
</tr>
</thead>
<tbody>
<tr>
<td>DAC_ALIGN_12B_R</td>
<td>data right 12b alignment</td>
</tr>
<tr>
<td>DAC_ALIGN_12B_L</td>
<td>data left 12b alignment</td>
</tr>
<tr>
<td>DAC_ALIGN_8B_R</td>
<td>data right 8b alignment</td>
</tr>
</tbody>
</table>

Input parameter(in)

| data         | The data sending to holding register |

Output parameter(out)

- \[\]

Return value

- \[\]

Example:

/* set the DAC specified data holding register value */

dac_data_set(DAC_ALIGN_8B_R, 0xff);

dac_trigger_enable

The description of dac_trigger_enable is shown as below:

Table 3-165. Function dac_trigger_enable

<table>
<thead>
<tr>
<th>Function name</th>
<th>dac_trigger_enable</th>
</tr>
</thead>
<tbody>
<tr>
<td>Function prototype</td>
<td>void dac_trigger_enable(void)</td>
</tr>
<tr>
<td>Function descriptions</td>
<td>enable DAC trigger</td>
</tr>
<tr>
<td>Precondition</td>
<td>-</td>
</tr>
<tr>
<td>The called functions</td>
<td>-</td>
</tr>
</tbody>
</table>

Input parameter(in)

- \[\]

Output parameter(out)

- \[\]

Return value

- \[\]

Example:

/* enable DAC trigger */

dac_trigger_enable();

dac_trigger_disable

The description of dac_trigger_disable is shown as below:
### Table 3-166. Function dac_trigger_disable

<table>
<thead>
<tr>
<th>Function name</th>
<th>dac_trigger_disable</th>
</tr>
</thead>
<tbody>
<tr>
<td>Function prototype</td>
<td>void dac_trigger_disable(void)</td>
</tr>
<tr>
<td>Function descriptions</td>
<td>disable DAC trigger</td>
</tr>
<tr>
<td>Precondition</td>
<td>-</td>
</tr>
<tr>
<td>The called functions</td>
<td>-</td>
</tr>
<tr>
<td>Input parameter(in)</td>
<td>-</td>
</tr>
<tr>
<td>Output parameter(out)</td>
<td>-</td>
</tr>
<tr>
<td>Return value</td>
<td>-</td>
</tr>
</tbody>
</table>

**Example:**

```c
/* disable DAC trigger */
dac_trigger_disable();
```

### dac_trigger_source_config

The description of dac_trigger_source_config is shown as below:

**Table 3-167. Function dac_trigger_source_config**

<table>
<thead>
<tr>
<th>Function name</th>
<th>dac_trigger_source_config</th>
</tr>
</thead>
<tbody>
<tr>
<td>Function prototype</td>
<td>void dac_trigger_source_config(uint32_t triggersource)</td>
</tr>
<tr>
<td>Function descriptions</td>
<td>configure DAC trigger source</td>
</tr>
<tr>
<td>Precondition</td>
<td>-</td>
</tr>
<tr>
<td>The called functions</td>
<td>-</td>
</tr>
<tr>
<td>Input parameter(in)</td>
<td>-</td>
</tr>
<tr>
<td>Input parameter(in)</td>
<td>triggersource</td>
</tr>
<tr>
<td>triggersource</td>
<td>external triggers of DAC</td>
</tr>
</tbody>
</table>

- DAC_TRIGGER_T1_T RGO
- DAC_TRIGGER_T2_T RGO
- DAC_TRIGGER_T6_T RGO
- DAC_TRIGGER_T5_T RGO
- DAC_TRIGGER_EXTI_9
- DAC_TRIGGER_SOFTWARE

- TIMER1 TRGO
- TIMER2 TRGO
- TIMER6 TRGO
- TIMER5 TRGO
- EXTI interrupt line9 event
- software trigger
Output parameter(out)

- - -

Return value

- - -

Example:

/* set DAC trigger source */
dac_trigger_source_config(DAC_TRIGGER_T1_TRGO);

**dac_software_trigger_enable**

The description of dac_software_trigger_enable is shown as below:

<table>
<thead>
<tr>
<th>Function name</th>
<th>dac_software_trigger_enable</th>
</tr>
</thead>
<tbody>
<tr>
<td>Function prototype</td>
<td>void dac_software_trigger_enable(void)</td>
</tr>
<tr>
<td>Function descriptions</td>
<td>enable DAC software trigger</td>
</tr>
<tr>
<td>Precondition</td>
<td>-</td>
</tr>
<tr>
<td>The called functions</td>
<td>-</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Input parameter(in)</th>
</tr>
</thead>
<tbody>
<tr>
<td>- - -</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Output parameter(out)</th>
</tr>
</thead>
<tbody>
<tr>
<td>- - -</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Return value</th>
</tr>
</thead>
<tbody>
<tr>
<td>- - -</td>
</tr>
</tbody>
</table>

Example:

/* enable DAC software trigger */
dac_software_trigger_enable();

**dac_software_trigger_disable**

The description of dac_software_trigger_disable is shown as below:

<table>
<thead>
<tr>
<th>Function name</th>
<th>dac_software_trigger_disable</th>
</tr>
</thead>
<tbody>
<tr>
<td>Function prototype</td>
<td>void dac_software_trigger_disable(void)</td>
</tr>
<tr>
<td>Function descriptions</td>
<td>disable DAC software trigger</td>
</tr>
<tr>
<td>Precondition</td>
<td>-</td>
</tr>
<tr>
<td>The called functions</td>
<td>-</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Input parameter(in)</th>
</tr>
</thead>
<tbody>
<tr>
<td>- - -</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Output parameter(out)</th>
</tr>
</thead>
<tbody>
<tr>
<td>- - -</td>
</tr>
</tbody>
</table>
Example:

```c
/* disable DAC software trigger */
dac_software_trigger_disable();
```

**dac_wave_mode_config**

The description of `dac_wave_mode_config` is shown as below:

<table>
<thead>
<tr>
<th>Function name</th>
<th>dac_wave_mode_config</th>
</tr>
</thead>
<tbody>
<tr>
<td>Function prototype</td>
<td>void dac_wave_mode_config(uint32_t wave_mode)</td>
</tr>
<tr>
<td>Function descriptions</td>
<td>configure DAC wave mode</td>
</tr>
<tr>
<td>Precondition</td>
<td>-</td>
</tr>
<tr>
<td>The called functions</td>
<td>-</td>
</tr>
</tbody>
</table>

**Input parameter(in)**

<table>
<thead>
<tr>
<th>wave_mode</th>
<th>DAC wave mode</th>
</tr>
</thead>
<tbody>
<tr>
<td>DAC_WAVE_DISABLE</td>
<td>wave mode disable</td>
</tr>
<tr>
<td>DAC_WAVE_MODE_LFSR</td>
<td>LFSR noise mode</td>
</tr>
<tr>
<td>DAC_WAVE_MODE_TRIANGLE</td>
<td>triangle noise mode</td>
</tr>
</tbody>
</table>

**Output parameter(out)**

| -                          | -                                          |

**Return value**

Example:

```c
/* configure DAC wave mode */
dac_wave_mode_config(DAC_WAVE_DISABLE);
```

**dac_wave_bit_width_config**

The description of `dac_wave_bit_width_config` is shown as below:

<table>
<thead>
<tr>
<th>Function name</th>
<th>dac_wave_bit_width_config</th>
</tr>
</thead>
<tbody>
<tr>
<td>Function prototype</td>
<td>void dac_wave_bit_width_config(uint32_t bit_width)</td>
</tr>
<tr>
<td>Function descriptions</td>
<td>configure DAC wave bit width</td>
</tr>
</tbody>
</table>

Example:

```c
/* configure DAC wave bit width */
dac_wave_bit_width_config(bit_width);
```
Precondition

The called functions

Input parameter(in)

Input parameter(in)

bit_width

DAC noise wave bit width

DAC_WAVE_BIT_WIDTH

TH_x

x = 1..12

Output parameter(out)

Return value

Example:

/* configure DAC wave bit width */
dac_wave_bit_width_config(DAC_WAVE_BIT_WIDTH_1);

dac_lfsr_noise_config

The description of dac_lfsr_noise_config is shown as below:

Table 3-172. Function dac_lfsr_noise_config

<table>
<thead>
<tr>
<th>Function name</th>
<th>dac_lfsr_noise_config</th>
</tr>
</thead>
<tbody>
<tr>
<td>Function prototype</td>
<td>void dac_lfsr_noise_config(uint32_t unmask_bits)</td>
</tr>
<tr>
<td>Function descriptions</td>
<td>configure DAC LFSR noise mode</td>
</tr>
<tr>
<td>Precondition</td>
<td>-</td>
</tr>
<tr>
<td>The called functions</td>
<td>-</td>
</tr>
<tr>
<td>Input parameter(in)</td>
<td>-</td>
</tr>
<tr>
<td>Input parameter(in)</td>
<td>-</td>
</tr>
<tr>
<td>unmask_bits</td>
<td>LFSR noise unmask bits</td>
</tr>
<tr>
<td>DAC_LFSR_BIT0</td>
<td>unmask the LFSR bit0</td>
</tr>
<tr>
<td>DAC_LFSR_BITSx_0</td>
<td>unmask the LFSR bits[x:0],x=1..11</td>
</tr>
<tr>
<td>Output parameter(out)</td>
<td>-</td>
</tr>
<tr>
<td>Return value</td>
<td>-</td>
</tr>
</tbody>
</table>

Example:

/* configure DAC LFSR noise mode */
dac_lfsr_noise_config(DAC_LFSR_BIT0);
**dac_triangle_noise_config**

The description of dac_triangle_noise_config is shown as below:

### Table 3-173. Function dac_triangle_noise_config

<table>
<thead>
<tr>
<th>Function name</th>
<th>dac_triangle_noise_config</th>
</tr>
</thead>
<tbody>
<tr>
<td>Function prototype</td>
<td>void dac_triangle_noise_config(uint32_t amplitude)</td>
</tr>
<tr>
<td>Function descriptions</td>
<td>configure DAC triangle noise mode</td>
</tr>
<tr>
<td>Precondition</td>
<td>-</td>
</tr>
<tr>
<td>The called functions</td>
<td>-</td>
</tr>
<tr>
<td>Input parameter (in)</td>
<td>-</td>
</tr>
<tr>
<td>amplitude</td>
<td>the amplitude of triangle noise</td>
</tr>
<tr>
<td>DAC_TRIANGLE_AMPLITUDE_x</td>
<td>$x = 2^n \cdot 1 (n = 1..12)$</td>
</tr>
<tr>
<td>Output parameter (out)</td>
<td>-</td>
</tr>
<tr>
<td>Return value</td>
<td>-</td>
</tr>
</tbody>
</table>

Example:

```c
/* configure DAC triangle noise mode */
dac_triangle_noise_config(DAC_TRIANGLE_AMPLITUDE_1);
```

**dac_flag_get**

The description of dac_flag_get is shown as below:

### Table 3-174. Function dac_flag_get

<table>
<thead>
<tr>
<th>Function name</th>
<th>dac_flag_get</th>
</tr>
</thead>
<tbody>
<tr>
<td>Function prototype</td>
<td>FlagStatus dac_flag_get(uint32_t flag)</td>
</tr>
<tr>
<td>Function descriptions</td>
<td>get DAC flag(DAC DMA underrun interrupt flag)</td>
</tr>
<tr>
<td>Precondition</td>
<td>-</td>
</tr>
<tr>
<td>The called functions</td>
<td>-</td>
</tr>
<tr>
<td>Input parameter (in)</td>
<td>-</td>
</tr>
<tr>
<td>dac_flag</td>
<td>DAC flag</td>
</tr>
<tr>
<td>DAC_FLAG_DDUDR</td>
<td>DMA underrun flag</td>
</tr>
<tr>
<td>Output parameter (out)</td>
<td>-</td>
</tr>
<tr>
<td>Return value</td>
<td>-</td>
</tr>
<tr>
<td>FlagStatus</td>
<td>the state of DAC</td>
</tr>
<tr>
<td>SET</td>
<td>DMA underrun error condition occurred</td>
</tr>
<tr>
<td>RESET</td>
<td>No DMA underrun error condition occurred</td>
</tr>
</tbody>
</table>
Example:

/* get the specified DAC flag (DAC DMA underrun flag) */

FlagStatus dac_falg = RESET;
dac_falg = dac_flag_get(DAC_FLAG_DDUDR);

dac_flag_clear

The description of dac_flag_clear is shown as below:

<table>
<thead>
<tr>
<th>Table 3-175. Function dac_flag_clear</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Function name</strong></td>
</tr>
<tr>
<td><strong>Function prototype</strong></td>
</tr>
<tr>
<td><strong>Function descriptions</strong></td>
</tr>
<tr>
<td><strong>Precondition</strong></td>
</tr>
<tr>
<td><strong>The called functions</strong></td>
</tr>
<tr>
<td><strong>Input parameter</strong>(in)</td>
</tr>
<tr>
<td><strong>flag</strong></td>
</tr>
<tr>
<td><strong>DAC_FLAG_DDUDR</strong></td>
</tr>
<tr>
<td><strong>Output parameter</strong>(out)</td>
</tr>
<tr>
<td><strong>Return value</strong></td>
</tr>
</tbody>
</table>

Example:

/* clear the specified DAC flag (DAC DMA underrun flag) */
dac_flag_clear(DAC_FLAG_DDUDR);

dac_interrupt_enable

The description of dac_interrupt_enable is shown as below:

<table>
<thead>
<tr>
<th>Table 3-176. Function dac_interrupt_enable</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Function name</strong></td>
</tr>
<tr>
<td><strong>Function prototype</strong></td>
</tr>
<tr>
<td><strong>Function descriptions</strong></td>
</tr>
<tr>
<td><strong>Precondition</strong></td>
</tr>
<tr>
<td><strong>The called functions</strong></td>
</tr>
<tr>
<td><strong>Input parameter</strong>(in)</td>
</tr>
<tr>
<td><strong>interrupt</strong></td>
</tr>
<tr>
<td><strong>DAC_INT_DDUDRIE</strong></td>
</tr>
<tr>
<td><strong>Output parameter</strong>(out)</td>
</tr>
<tr>
<td><strong>Return value</strong></td>
</tr>
</tbody>
</table>
Example:

```
/* enable DAC interrupt(DAC DMA underrun interrupt) */
dac_interrupt_enable(DAC_INT_DDUDRIE);
```

**dac_interrupt_disable**

The description of `dac_interrupt_disable` is shown as below:

**Table 3-177. Function dac_interrupt_disable**

<table>
<thead>
<tr>
<th>Function name</th>
<th>dac_interrupt_disable</th>
</tr>
</thead>
<tbody>
<tr>
<td>Function prototype</td>
<td>void dac_interrupt_disable(uint32_t interrupt)</td>
</tr>
<tr>
<td>Function description</td>
<td>disable DAC interrupt(DAC DMA underrun interrupt)</td>
</tr>
<tr>
<td>Precondition</td>
<td>-</td>
</tr>
<tr>
<td>The called functions</td>
<td>-</td>
</tr>
<tr>
<td>Input parameter(in)</td>
<td>interrupt</td>
</tr>
<tr>
<td></td>
<td>the DAC interrupt</td>
</tr>
<tr>
<td></td>
<td>DAC_INT_DDUDRIE DMA underrun interrupt disable</td>
</tr>
<tr>
<td>Output parameter(out)</td>
<td>-</td>
</tr>
<tr>
<td>Return value</td>
<td>-</td>
</tr>
</tbody>
</table>

Example:

```
/* disable DAC interrupt(DAC DMA underrun interrupt) */
dac_interrupt_disable(DAC_INT_DDUDRIE);
```

**dac_interrupt_flag_get**

The description of `dac_interrupt_flag_get` is shown as below:

**Table 3-178. Function dac_interrupt_flag_get**

<table>
<thead>
<tr>
<th>Function name</th>
<th>dac_interrupt_flag_get</th>
</tr>
</thead>
<tbody>
<tr>
<td>Function prototype</td>
<td>FlagStatus dac_interrupt_flag_get(uint32_t int_flag)</td>
</tr>
<tr>
<td>Function description</td>
<td>get DAC interrupt flag (DAC DMA underrun interrupt flag)</td>
</tr>
<tr>
<td>Precondition</td>
<td>-</td>
</tr>
<tr>
<td>The called functions</td>
<td>-</td>
</tr>
<tr>
<td>Input parameter(in)</td>
<td>int_flag</td>
</tr>
<tr>
<td></td>
<td>DAC interrupt flag</td>
</tr>
<tr>
<td></td>
<td>DAC_INT_FLAG_DDU DR DMA underrun interrupt flag</td>
</tr>
<tr>
<td>Output parameter(out)</td>
<td>-</td>
</tr>
<tr>
<td>Return value</td>
<td></td>
</tr>
<tr>
<td>--------------------</td>
<td>--------------------------</td>
</tr>
<tr>
<td>FlagStatus</td>
<td>the state of DAC</td>
</tr>
<tr>
<td>SET</td>
<td>DMA underrun error interrupt occurred</td>
</tr>
<tr>
<td>RESET</td>
<td>No DMA underrun error interrupt occurred</td>
</tr>
</tbody>
</table>

Example:

/* get the specified DAC interrupt flag (DAC DMA underrun interrupt flag) */

FlagStatus dac_falg = RESET;
dac_falg = dac_interrupt_flag_get(DAC_INT_FLAG_DDUDR);

**dac_interrupt_flag_clear**

The description of **dac_interrupt_flag_clear** is shown as below:

**Table 3-179. Function dac_interrupt_flag_clear**

<table>
<thead>
<tr>
<th>Function name</th>
<th>dac_interrupt_flag_clear</th>
</tr>
</thead>
<tbody>
<tr>
<td>Function prototype</td>
<td>void dac_interrupt_flag_clear(uint32_t int_flag)</td>
</tr>
<tr>
<td>Function descriptions</td>
<td>clear the specified DAC interrupt flag (DAC DMA underrun interrupt flag)</td>
</tr>
<tr>
<td>Precondition</td>
<td>-</td>
</tr>
<tr>
<td>The called functions</td>
<td>-</td>
</tr>
<tr>
<td>Input parameter(in)</td>
<td></td>
</tr>
<tr>
<td>int_flag</td>
<td>DAC interrupt flag</td>
</tr>
<tr>
<td>DAC_INT_FLAG_DDUDR</td>
<td>DMA underrun interrupt flag</td>
</tr>
<tr>
<td>Output parameter(out)</td>
<td>-</td>
</tr>
<tr>
<td>Return value</td>
<td>-</td>
</tr>
</tbody>
</table>

Example:

/* clear the specified DAC interrupt flag (DAC DMA underrun interrupt flag) */
dac_interrupt_flag_clear(DAC_INT_FLAG_DDUDR);

**3.9. DMA/DMAMUX**

The direct memory access (DMA) controller provides a hardware method of transferring data between peripherals and/or memory without intervention from the CPU, thereby freeing up bandwidth for other system functions. The DMA registers are listed in chapter **3.9.1**, the DMA firmware functions are introduced in chapter **3.9.2**.

DMAMUX is a transmission scheduler for DMA requests. The DMAMUX request multiplexer
GD32L23x Firmware Library User Guide

is used for routing a DMA request line between the peripherals / generated DMA request (from the DMAMUX request generator) and the DMA controller. The DMAMUX registers are listed in chapter 3.9.1, the DMAMUX firmware functions are introduced in chapter 3.9.2.

3.9.1. Descriptions of Peripheral registers

DMA registers are listed in the table shown as below:

Table 3-180. DMA Registers

<table>
<thead>
<tr>
<th>Registers</th>
<th>Descriptions</th>
</tr>
</thead>
<tbody>
<tr>
<td>DMA_INTF</td>
<td>Interrupt flag register</td>
</tr>
<tr>
<td>DMA_INTC</td>
<td>Interrupt flag clear register</td>
</tr>
<tr>
<td>DMA_CHxCTL (x=0..6)</td>
<td>Channel x control register</td>
</tr>
<tr>
<td>DMA_CHxCNT (x=0..6)</td>
<td>Channel x counter register</td>
</tr>
<tr>
<td>DMA_CHxPADDR (x=0..6)</td>
<td>Channel x peripheral base address register</td>
</tr>
<tr>
<td>DMA_CHxMADDR (x=0..6)</td>
<td>Channel x memory base address register</td>
</tr>
</tbody>
</table>

DMAMUX registers are listed in the table shown as below:

Table 3-181. DMAMUX Registers

<table>
<thead>
<tr>
<th>Registers</th>
<th>Descriptions</th>
</tr>
</thead>
<tbody>
<tr>
<td>DMAMUX_RM_CHxCFG (x=0..6)</td>
<td>Request multiplexer channel x configuration register</td>
</tr>
<tr>
<td>DMAMUX_RM_INTF</td>
<td>Request multiplexer channel interrupt flag register</td>
</tr>
<tr>
<td>DMAMUX_RM_INTFC</td>
<td>Request multiplexer channel interrupt flag clear register</td>
</tr>
<tr>
<td>DMAMUX_RG_CHxCFG (x=0..6)</td>
<td>Request generator channel x configuration register</td>
</tr>
<tr>
<td>DMAMUX_RG_INTF</td>
<td>Request generator channel interrupt flag register</td>
</tr>
<tr>
<td>DMAMUX_RG_INTFC</td>
<td>Request generator channel interrupt flag clear register</td>
</tr>
</tbody>
</table>

3.9.2. Descriptions of Peripheral functions

DMA firmware functions are listed in the table shown as below:

Table 3-182. DMA firmware function

<table>
<thead>
<tr>
<th>Function name</th>
<th>Function description</th>
</tr>
</thead>
<tbody>
<tr>
<td>dma_deinit</td>
<td>deinitialize DMA a channel registers</td>
</tr>
<tr>
<td>Function name</td>
<td>Function description</td>
</tr>
<tr>
<td>--------------------------</td>
<td>---------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>dma_struct_para_init</td>
<td>initialize the parameters of DMA struct with the default values</td>
</tr>
<tr>
<td>dma_init</td>
<td>initialize DMA channel</td>
</tr>
<tr>
<td>dma_circulation_enable</td>
<td>enable DMA circulation mode</td>
</tr>
<tr>
<td>dma_circulation_disable</td>
<td>disable DMA circulation mode</td>
</tr>
<tr>
<td>dma_memory_to_memory_enable</td>
<td>enable memory to memory mode</td>
</tr>
<tr>
<td>dma_memory_to_memory_disable</td>
<td>disable memory to memory mode</td>
</tr>
<tr>
<td>dma_channel_enable</td>
<td>enable DMA channel</td>
</tr>
<tr>
<td>dma_channel_disable</td>
<td>disable DMA channel</td>
</tr>
<tr>
<td>dma_periph_address_config</td>
<td>set DMA peripheral base address</td>
</tr>
<tr>
<td>dma_memory_address_config</td>
<td>set DMA memory base address</td>
</tr>
<tr>
<td>dma_transfer_number_config</td>
<td>set the number of remaining data to be transferred by the DMA</td>
</tr>
<tr>
<td>dma_transfer_number_get</td>
<td>get the number of remaining data to be transferred by the DMA</td>
</tr>
<tr>
<td>dma_priority_config</td>
<td>configure priority level of DMA channel</td>
</tr>
<tr>
<td>dma_memory_width_config</td>
<td>configure transfer data size of memory</td>
</tr>
<tr>
<td>dma_periph_width_config</td>
<td>configure transfer data size of peripheral</td>
</tr>
<tr>
<td>dma_memory_increase_enable</td>
<td>enable next address incrementation algorithm of memory</td>
</tr>
<tr>
<td>dma_memory_increase_disable</td>
<td>disable next address incrementation algorithm of memory</td>
</tr>
<tr>
<td>dma_periph_increase_enable</td>
<td>enable next address incrementation algorithm of peripheral</td>
</tr>
<tr>
<td>dma_periph_increase_disable</td>
<td>disable next address incrementation algorithm of peripheral</td>
</tr>
<tr>
<td>dma_transfer_direction_config</td>
<td>configure the direction of data transfer on the channel</td>
</tr>
<tr>
<td>dma_flag_get</td>
<td>check DMA flag is set or not</td>
</tr>
<tr>
<td>dma_flag_clear</td>
<td>clear DMA a channel flag</td>
</tr>
<tr>
<td>dma_interrupt_flag_get</td>
<td>check DMA flag and interrupt enable bit is set or not</td>
</tr>
<tr>
<td>dma_interrupt_flag_clear</td>
<td>clear DMA a channel flag</td>
</tr>
<tr>
<td>dma_interrupt_enable</td>
<td>enable DMA interrupt</td>
</tr>
<tr>
<td>dma_interrupt_disable</td>
<td>disable DMA interrupt</td>
</tr>
</tbody>
</table>

DMAMUX firmware functions are listed in the table shown as below:

Table 3-183. DMAMUX firmware function

<table>
<thead>
<tr>
<th>Function name</th>
<th>Function description</th>
</tr>
</thead>
<tbody>
<tr>
<td>dmamux_sync_struct_para_init</td>
<td>initialize the parameters of DMAMUX synchronization mode structure with the default values</td>
</tr>
<tr>
<td>dmamux_synchronization_init</td>
<td>initialize DMAMUX request multiplexer channel synchronization mode</td>
</tr>
<tr>
<td>dmamux_synchronization_enable</td>
<td>enable synchronization mode</td>
</tr>
<tr>
<td>dmamux_synchronization_disable</td>
<td>disable synchronization mode</td>
</tr>
<tr>
<td>dmamux_event_generation_enable</td>
<td>enable event generation</td>
</tr>
<tr>
<td>dmamux_event_generation_disable</td>
<td>disable event generation</td>
</tr>
<tr>
<td>dmamux_gen_struct_para_init</td>
<td>initialize the parameters of DMAMUX request generator</td>
</tr>
</tbody>
</table>
### Function names and descriptions

<table>
<thead>
<tr>
<th>Function name</th>
<th>Function description</th>
</tr>
</thead>
<tbody>
<tr>
<td>dmamux_request_generator_init</td>
<td>initialize DMAMUX request generator channel</td>
</tr>
<tr>
<td>dmamux_request_generator_channel_enable</td>
<td>enable DMAMUX request generator channel</td>
</tr>
<tr>
<td>dmamux_request_generator_channel_disable</td>
<td>disable DMAMUX request generator channel</td>
</tr>
<tr>
<td>dmamux_synchronization_polarity_config</td>
<td>configure synchronization input polarity</td>
</tr>
<tr>
<td>dmamux_request_forward_number_config</td>
<td>configure number of DMA requests to forward</td>
</tr>
<tr>
<td>dmamux_sync_id_config</td>
<td>configure synchronization input identification</td>
</tr>
<tr>
<td>dmamux_request_id_config</td>
<td>configure multiplexer input identification</td>
</tr>
<tr>
<td>dmamux_trigger_polarity_config</td>
<td>configure trigger input polarity</td>
</tr>
<tr>
<td>dmamux_request_generate_number_config</td>
<td>configure number of DMA requests to be generated</td>
</tr>
<tr>
<td>dmamux_trigger_id_config</td>
<td>configure trigger input identification</td>
</tr>
<tr>
<td>dmamux_flag_get</td>
<td>get DMAMUX flag</td>
</tr>
<tr>
<td>dmamux_flag_clear</td>
<td>clear DMAMUX flag</td>
</tr>
<tr>
<td>dmamux_interrupt_flag_get</td>
<td>get DMAMUX interrupt flag</td>
</tr>
<tr>
<td>dmamux_interrupt_flag_clear</td>
<td>clear DMAMUX interrupt flag</td>
</tr>
<tr>
<td>dmamux_interrupt_enable</td>
<td>enable DMAMUX interrupt</td>
</tr>
<tr>
<td>dmamux_interrupt_disable</td>
<td>disable DMAMUX interrupt</td>
</tr>
</tbody>
</table>

### Structure dma_parameter_struct

#### Table 3-184. Structure dma_parameter_struct

<table>
<thead>
<tr>
<th>Member name</th>
<th>Function description</th>
</tr>
</thead>
<tbody>
<tr>
<td>periph_addr</td>
<td>peripheral base address</td>
</tr>
<tr>
<td>periph_width</td>
<td>transfer data size of peripheral</td>
</tr>
<tr>
<td>memory_addr</td>
<td>memory base address</td>
</tr>
<tr>
<td>memory_width</td>
<td>transfer data size of memory</td>
</tr>
<tr>
<td>number</td>
<td>channel transfer number</td>
</tr>
<tr>
<td>priority</td>
<td>channel priority level</td>
</tr>
<tr>
<td>periph_inc</td>
<td>peripheral increasing mode</td>
</tr>
<tr>
<td>memory_inc</td>
<td>memory increasing mode</td>
</tr>
<tr>
<td>direction</td>
<td>channel data transfer direction</td>
</tr>
<tr>
<td>request</td>
<td>channel input identification</td>
</tr>
</tbody>
</table>

### Structure dmamux_sync_parameter_struct

#### Table 3-185. Structure dmamux_sync_parameter_struct

<table>
<thead>
<tr>
<th>Member name</th>
<th>Function description</th>
</tr>
</thead>
</table>

---

141
sync_id | synchronization input identification
---|---
sync_polarity | synchronization input polarity
request_number | number of DMA requests to forward

### Structure dmamux_gen_parameter_struct

Table 3-186. Structure dmamux_gen_parameter_struct

<table>
<thead>
<tr>
<th>Member name</th>
<th>Function description</th>
</tr>
</thead>
<tbody>
<tr>
<td>trigger_id</td>
<td>trigger input identification</td>
</tr>
<tr>
<td>trigger_polarity</td>
<td>DMAMUX request generator trigger polarity</td>
</tr>
<tr>
<td>request_number</td>
<td>number of DMA requests to be generated</td>
</tr>
</tbody>
</table>

### Enum dmamux_interrupt_enum

Table 3-187. Enum dmamux_interrupt_enum

<table>
<thead>
<tr>
<th>Member name</th>
<th>Function description</th>
</tr>
</thead>
<tbody>
<tr>
<td>DMAMUX_INT_MU XCH0_SO</td>
<td>DMAMUX request multiplexer channel 0 synchronization overrun interrupt</td>
</tr>
<tr>
<td>DMAMUX_INT_MU XCH1_SO</td>
<td>DMAMUX request multiplexer channel 1 synchronization overrun interrupt</td>
</tr>
<tr>
<td>DMAMUX_INT_MU XCH2_SO</td>
<td>DMAMUX request multiplexer channel 2 synchronization overrun interrupt</td>
</tr>
<tr>
<td>DMAMUX_INT_MU XCH3_SO</td>
<td>DMAMUX request multiplexer channel 3 synchronization overrun interrupt</td>
</tr>
<tr>
<td>DMAMUX_INT_MU XCH4_SO</td>
<td>DMAMUX request multiplexer channel 4 synchronization overrun interrupt</td>
</tr>
<tr>
<td>DMAMUX_INT_MU XCH5_SO</td>
<td>DMAMUX request multiplexer channel 5 synchronization overrun interrupt</td>
</tr>
<tr>
<td>DMAMUX_INT_MU XCH6_SO</td>
<td>DMAMUX request multiplexer channel 6 synchronization overrun interrupt</td>
</tr>
<tr>
<td>DMAMUX_INT_GE NCH0_TO</td>
<td>DMAMUX request generator channel 0 trigger overrun interrupt</td>
</tr>
<tr>
<td>DMAMUX_INT_GE NCH1_TO</td>
<td>DMAMUX request generator channel 1 trigger overrun interrupt</td>
</tr>
<tr>
<td>DMAMUX_INT_GE NCH2_TO</td>
<td>DMAMUX request generator channel 2 trigger overrun interrupt</td>
</tr>
<tr>
<td>DMAMUX_INT_GE NCH3_TO</td>
<td>DMAMUX request generator channel 3 trigger overrun interrupt</td>
</tr>
</tbody>
</table>

### Enum dmamux_flag_enum

Table 3-188. Enum dmamux_flag_enum

<table>
<thead>
<tr>
<th>Member name</th>
<th>Function description</th>
</tr>
</thead>
<tbody>
<tr>
<td>DMAMUX_FLAG_M</td>
<td>DMAMUX request multiplexer channel 0 synchronization overrun flag</td>
</tr>
</tbody>
</table>
### UXCH0_SO
- **DMAMUX_FLAG_MUXCH0_SO**: DMAMUX request multiplexer channel 0 synchronization overrun flag
- **DMAMUX_FLAG_MUXCH1_SO**: DMAMUX request multiplexer channel 1 synchronization overrun flag
- **DMAMUX_FLAG_MUXCH2_SO**: DMAMUX request multiplexer channel 2 synchronization overrun flag
- **DMAMUX_FLAG_MUXCH3_SO**: DMAMUX request multiplexer channel 3 synchronization overrun flag
- **DMAMUX_FLAG_MUXCH4_SO**: DMAMUX request multiplexer channel 4 synchronization overrun flag
- **DMAMUX_FLAG_MUXCH5_SO**: DMAMUX request multiplexer channel 5 synchronization overrun flag
- **DMAMUX_FLAG_MUXCH6_SO**: DMAMUX request multiplexer channel 6 synchronization overrun flag
- **DMAMUX_FLAG_GENCH0_TO**: DMAMUX request generator channel 0 trigger overrun flag
- **DMAMUX_FLAG_GENCH1_TO**: DMAMUX request generator channel 1 trigger overrun flag
- **DMAMUX_FLAG_GENCH2_TO**: DMAMUX request generator channel 2 trigger overrun flag
- **DMAMUX_FLAG_GENCH3_TO**: DMAMUX request generator channel 3 trigger overrun flag

### Enum dmamux_interrupt_flag_enum

#### Table 3-189. Enum dmamux_interrupt_flag_enum

<table>
<thead>
<tr>
<th>Member name</th>
<th>Function description</th>
</tr>
</thead>
<tbody>
<tr>
<td>DMAMUX_INT_FLA_G_MUXCH0_SO</td>
<td>DMAMUX request multiplexer channel 0 synchronization overrun interrupt flag</td>
</tr>
<tr>
<td>DMAMUX_INT_FLA_G_MUXCH1_SO</td>
<td>DMAMUX request multiplexer channel 1 synchronization overrun interrupt flag</td>
</tr>
<tr>
<td>DMAMUX_INT_FLA_G_MUXCH2_SO</td>
<td>DMAMUX request multiplexer channel 2 synchronization overrun interrupt flag</td>
</tr>
<tr>
<td>DMAMUX_INT_FLA_G_MUXCH3_SO</td>
<td>DMAMUX request multiplexer channel 3 synchronization overrun interrupt flag</td>
</tr>
<tr>
<td>DMAMUX_INT_FLA_G_MUXCH4_SO</td>
<td>DMAMUX request multiplexer channel 4 synchronization overrun interrupt flag</td>
</tr>
<tr>
<td>DMAMUX_INT_FLA_G_MUXCH5_SO</td>
<td>DMAMUX request multiplexer channel 5 synchronization overrun interrupt flag</td>
</tr>
<tr>
<td>DMAMUX_INT_FLA_G_MUXCH6_SO</td>
<td>DMAMUX request multiplexer channel 6 synchronization overrun interrupt flag</td>
</tr>
<tr>
<td>DMAMUX_INT_FLA_G_GENCH0_TO</td>
<td>DMAMUX request generator channel 0 trigger overrun interrupt flag</td>
</tr>
<tr>
<td>DMAMUX_INT_FLA</td>
<td>DMAMUX request generator channel 1 trigger overrun interrupt flag</td>
</tr>
</tbody>
</table>
Enum dma_channel_enum

Table 3-190. Enum dma_channel_enum

<table>
<thead>
<tr>
<th>Member name</th>
<th>Function description</th>
</tr>
</thead>
<tbody>
<tr>
<td>DMA_CH0</td>
<td>DMA Channel 0</td>
</tr>
<tr>
<td>DMA_CH1</td>
<td>DMA Channel 1</td>
</tr>
<tr>
<td>DMA_CH2</td>
<td>DMA Channel 2</td>
</tr>
<tr>
<td>DMA_CH3</td>
<td>DMA Channel 3</td>
</tr>
<tr>
<td>DMA_CH4</td>
<td>DMA Channel 4</td>
</tr>
<tr>
<td>DMA_CH5</td>
<td>DMA Channel 5</td>
</tr>
<tr>
<td>DMA_CH6</td>
<td>DMA Channel 6</td>
</tr>
</tbody>
</table>

Enum dmamux_multiplexer_channel_enum

Table 3-191. Enum dmamux_multiplexer_channel_enum

<table>
<thead>
<tr>
<th>Member name</th>
<th>Function description</th>
</tr>
</thead>
<tbody>
<tr>
<td>DMAMUX_MUXCH 0</td>
<td>DMAMUX request multiplexer Channel0</td>
</tr>
<tr>
<td>DMAMUX_MUXCH 1</td>
<td>DMAMUX request multiplexer Channel1</td>
</tr>
<tr>
<td>DMAMUX_MUXCH 2</td>
<td>DMAMUX request multiplexer Channel2</td>
</tr>
<tr>
<td>DMAMUX_MUXCH 3</td>
<td>DMAMUX request multiplexer Channel3</td>
</tr>
<tr>
<td>DMAMUX_MUXCH 4</td>
<td>DMAMUX request multiplexer Channel4</td>
</tr>
<tr>
<td>DMAMUX_MUXCH 5</td>
<td>DMAMUX request multiplexer Channel5</td>
</tr>
<tr>
<td>DMAMUX_MUXCH 6</td>
<td>DMAMUX request multiplexer Channel6</td>
</tr>
</tbody>
</table>

Enum dmamux_generator_channel_enum

Table 3-192. Enum dmamux_generator_channel_enum

<table>
<thead>
<tr>
<th>Member name</th>
<th>Function description</th>
</tr>
</thead>
<tbody>
<tr>
<td>DMAMUX_GENCH0</td>
<td>DMAMUX request generator Channel0</td>
</tr>
<tr>
<td>DMAMUX_GENCH1</td>
<td>DMAMUX request generator Channel1</td>
</tr>
<tr>
<td>DMAMUX_GENCH2</td>
<td>DMAMUX request generator Channel2</td>
</tr>
</tbody>
</table>
dma_deinit

The description of dma_deinit is shown as below:

**Table 3-193. Function dma_deinit**

<table>
<thead>
<tr>
<th>Function name</th>
<th>dma_deinit</th>
</tr>
</thead>
<tbody>
<tr>
<td>Function prototype</td>
<td>void dma_deinit(dma_channel_enum channelx);</td>
</tr>
<tr>
<td>Function descriptions</td>
<td>deinitialize DMA a channel registers</td>
</tr>
<tr>
<td>Precondition</td>
<td>-</td>
</tr>
<tr>
<td>The called functions</td>
<td>-</td>
</tr>
</tbody>
</table>

**Input parameter**:

- channelx: DMA channel

**Output parameter**:

- 

**Return value**:

- 

Example:

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

dma_struct_para_init

The description of dma_struct_para_init is shown as below:

**Table 3-194. Function dma_para_init**

<table>
<thead>
<tr>
<th>Function name</th>
<th>dma_struct_para_init</th>
</tr>
</thead>
<tbody>
<tr>
<td>Function prototype</td>
<td>void dma_struct_para_init(dma_parameter_struct* init_struct);</td>
</tr>
<tr>
<td>Function descriptions</td>
<td>initialize the parameters of DMA struct with the default values</td>
</tr>
<tr>
<td>Precondition</td>
<td>-</td>
</tr>
<tr>
<td>The called functions</td>
<td>-</td>
</tr>
</tbody>
</table>

**Input parameter**:

- init_struct: the initialization data needed to initialize DMA channel, refer to Table 3-184. Structure dma_parameter_struct

**Output parameter**:

- 

**Return value**:

- 

Example:

```c
/* initialize the parameters of DMA */
dma_parameter_struct dma_init_struct;
```
dma_init

The description of dma_init is shown as below:

Table 3-195. Function dma_init

<table>
<thead>
<tr>
<th>Function name</th>
<th>dma_init</th>
</tr>
</thead>
<tbody>
<tr>
<td>Function prototype</td>
<td>void dma_init(dma_channel_enum channelx, dma_parameter_struct init_struct);</td>
</tr>
<tr>
<td>Function descriptions</td>
<td>initialize DMA channel</td>
</tr>
<tr>
<td>Precondition</td>
<td>-</td>
</tr>
<tr>
<td>The called functions</td>
<td>-</td>
</tr>
<tr>
<td>Input parameter(in)</td>
<td></td>
</tr>
<tr>
<td>channelx</td>
<td>DMA channel</td>
</tr>
<tr>
<td>DMA_CHx(x=0..6)</td>
<td>DMA channel selection, refer to Table 3-190. Enum dma_channel_enum</td>
</tr>
<tr>
<td>init_struct</td>
<td>Structure for initialization, the structure members can refer to Table 3-184. Structure dma_parameter_struct</td>
</tr>
<tr>
<td>Output parameter(out)</td>
<td></td>
</tr>
<tr>
<td>Return value</td>
<td>-</td>
</tr>
</tbody>
</table>

Example:

/* DMA channel0 initialize */
da_ma_parameter_struct dma_init_struct;
da_ma_struct_para_init(&dma_init_struct);
da_ma_init STRUCT dma_init_struct;
da_ma_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_INCREMENT_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_INCREMENT_ENABLE; dma_init STRUCT periph_width = DMA_PERIPHERAL_WIDTH_8BIT; dma_init STRUCT priority = DMA_PRIORITY_ULTRA_HIGH; dma_init(DMA_CH0, dma_init STRUCT);

da_ma_circulation_enable

The description of dma_circulation_enable is shown as below:
Table 3-196. Function dma_circulation_enable

<table>
<thead>
<tr>
<th>Function name</th>
<th>dma_circulation_enable</th>
</tr>
</thead>
<tbody>
<tr>
<td>Function prototype</td>
<td>void dma_circulation_enable(dma_channel_enum channelx);</td>
</tr>
<tr>
<td>Function descriptions</td>
<td>enable DMA circulation mode</td>
</tr>
<tr>
<td>Precondition</td>
<td>corresponding channel enable bit CHEN should be 0</td>
</tr>
<tr>
<td>The called functions</td>
<td>-</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>channelx</th>
<th>DMA channel</th>
</tr>
</thead>
<tbody>
<tr>
<td>DMA_CHx(x=0..6)</td>
<td>DMA channel selection, refer to Table 3-190. Enum dma_channel_enum</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Input parameter(in)</th>
</tr>
</thead>
<tbody>
<tr>
<td>channelx</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Output parameter(out)</th>
</tr>
</thead>
<tbody>
<tr>
<td>-</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Return value</th>
</tr>
</thead>
<tbody>
<tr>
<td>-</td>
</tr>
</tbody>
</table>

Example:

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

dma_circulation_disable

The description of dma_circulation_disable is shown as below:

Table 3-197. Function dma_circulation_disable

<table>
<thead>
<tr>
<th>Function name</th>
<th>dma_circulation_disable</th>
</tr>
</thead>
<tbody>
<tr>
<td>Function prototype</td>
<td>void dma_circulation_disable(dma_channel_enum channelx);</td>
</tr>
<tr>
<td>Function descriptions</td>
<td>disable DMA circulation mode</td>
</tr>
<tr>
<td>Precondition</td>
<td>corresponding channel enable bit CHEN should be 0</td>
</tr>
<tr>
<td>The called functions</td>
<td>-</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>channelx</th>
<th>DMA channel</th>
</tr>
</thead>
<tbody>
<tr>
<td>DMA_CHx(x=0..6)</td>
<td>DMA channel selection, refer to Table 3-190. Enum dma_channel_enum</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Input parameter(in)</th>
</tr>
</thead>
<tbody>
<tr>
<td>channelx</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Output parameter(out)</th>
</tr>
</thead>
<tbody>
<tr>
<td>-</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Return value</th>
</tr>
</thead>
<tbody>
<tr>
<td>-</td>
</tr>
</tbody>
</table>

Example:

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

dma_memory_to_memory_enable

The description of dma_memory_to_memory_enable is shown as below:
Table 3-198. Function dma_memory_to_memory_enable

<table>
<thead>
<tr>
<th>Function name</th>
<th>dma_memory_to_memory_enable</th>
</tr>
</thead>
<tbody>
<tr>
<td>Function prototype</td>
<td>void dma_memory_to_memory_enable(dma_channel_enum channelx);</td>
</tr>
<tr>
<td>Function descriptions</td>
<td>enable memory to memory mode</td>
</tr>
<tr>
<td>Precondition</td>
<td>corresponding channel enable bit CHEN should be 0</td>
</tr>
<tr>
<td>The called functions</td>
<td></td>
</tr>
<tr>
<td>Input parameter(in)</td>
<td></td>
</tr>
<tr>
<td>channelx</td>
<td>DMA channel</td>
</tr>
<tr>
<td>DMA_CHx(x=0..6)</td>
<td>DMA channel selection, refer to Table 3-190. Enum dma_channel_enum</td>
</tr>
<tr>
<td>Output parameter(out)</td>
<td></td>
</tr>
<tr>
<td>-</td>
<td></td>
</tr>
<tr>
<td>Return value</td>
<td></td>
</tr>
<tr>
<td>-</td>
<td></td>
</tr>
</tbody>
</table>

Example:

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

**dma_memory_to_memory_disable**

The description of **dma_memory_to_memory_disable** is shown as below:

Table 3-199. Function dma_memory_to_memory_disable

<table>
<thead>
<tr>
<th>Function name</th>
<th>dma_memory_to_memory_disable</th>
</tr>
</thead>
<tbody>
<tr>
<td>Function prototype</td>
<td>void dma_memory_to_memory_disable(dma_channel_enum channelx);</td>
</tr>
<tr>
<td>Function descriptions</td>
<td>disable memory to memory mode</td>
</tr>
<tr>
<td>Precondition</td>
<td>corresponding channel enable bit CHEN should be 0</td>
</tr>
<tr>
<td>The called functions</td>
<td></td>
</tr>
<tr>
<td>Input parameter(in)</td>
<td></td>
</tr>
<tr>
<td>channelx</td>
<td>DMA channel</td>
</tr>
<tr>
<td>DMA_CHx(x=0..6)</td>
<td>DMA channel selection, refer to Table 3-190. Enum dma_channel_enum</td>
</tr>
<tr>
<td>Output parameter(out)</td>
<td></td>
</tr>
<tr>
<td>-</td>
<td></td>
</tr>
<tr>
<td>Return value</td>
<td></td>
</tr>
<tr>
<td>-</td>
<td></td>
</tr>
</tbody>
</table>

Example:

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

**dma_channel_enable**

The description of **dma_channel_enable** is shown as below:
Table 3-200. Function dma_channel_enable

<table>
<thead>
<tr>
<th>Function name</th>
<th>dma_channel_enable</th>
</tr>
</thead>
<tbody>
<tr>
<td>Function prototype</td>
<td>void dma_channel_enable(dma_channel_enum channelx);</td>
</tr>
<tr>
<td>Function descriptions</td>
<td>enable DMA channel</td>
</tr>
<tr>
<td>Precondition</td>
<td>-</td>
</tr>
<tr>
<td>The called functions</td>
<td>-</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>channelx</th>
<th>DMA channel</th>
</tr>
</thead>
<tbody>
<tr>
<td>DMA_CHx(x=0..6)</td>
<td>DMA channel selection, refer to Table 3-190. Enum dma_channel_enum</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Output parameter(out)</th>
<th>-</th>
</tr>
</thead>
</table>

| Return value | - |

Example:

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

dma_channel_disable

The description of dma_channel_disable is shown as below:

Table 3-201. Function dma_channel_disable

<table>
<thead>
<tr>
<th>Function name</th>
<th>dma_channel_disable</th>
</tr>
</thead>
<tbody>
<tr>
<td>Function prototype</td>
<td>void dma_channel_disable(dma_channel_enum channelx);</td>
</tr>
<tr>
<td>Function descriptions</td>
<td>disable DMA channel</td>
</tr>
<tr>
<td>Precondition</td>
<td>-</td>
</tr>
<tr>
<td>The called functions</td>
<td>-</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>channelx</th>
<th>DMA channel</th>
</tr>
</thead>
<tbody>
<tr>
<td>DMA_CHx(x=0..6)</td>
<td>DMA channel selection, refer to Table 3-190. Enum dma_channel_enum</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Output parameter(out)</th>
<th>-</th>
</tr>
</thead>
</table>

| Return value | - |

Example:

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

dma_periph_address_config

The description of dma_periph_address_config is shown as below:
Table 3-202. Function dma_periph_address_config

<table>
<thead>
<tr>
<th>Function name</th>
<th>dma_periph_address_config</th>
</tr>
</thead>
<tbody>
<tr>
<td>Function prototype</td>
<td>void dma_periph_address_config(dma_channel_enum channelx, uint32_t address);</td>
</tr>
<tr>
<td>Function descriptions</td>
<td>set DMA peripheral base address</td>
</tr>
<tr>
<td>Precondition</td>
<td>corresponding channel enable bit CHEN should be 0</td>
</tr>
<tr>
<td>The called functions</td>
<td>-</td>
</tr>
</tbody>
</table>

channelx: DMA channel

DMA_CHx(x=0..6): DMA channel selection, refer to Table 3-190. Enum dma_channel_enum

<table>
<thead>
<tr>
<th>Input parameter(in)</th>
<th>address</th>
</tr>
</thead>
<tbody>
<tr>
<td>peripheral base address, 0 − 0xFFFFFFFF</td>
<td></td>
</tr>
</tbody>
</table>

| Output parameter(out) | - |

| Return value | - |

Example:

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

dma_memory_address_config

The description of dma_memory_address_config is shown as below:

Table 3-203. Function dma_memory_address_config

<table>
<thead>
<tr>
<th>Function name</th>
<th>dma_memory_address_config</th>
</tr>
</thead>
<tbody>
<tr>
<td>Function prototype</td>
<td>void dma_memory_address_config(dma_channel_enum channelx, uint32_t address);</td>
</tr>
<tr>
<td>Function descriptions</td>
<td>set DMA memory base address</td>
</tr>
<tr>
<td>Precondition</td>
<td>corresponding channel enable bit CHEN should be 0</td>
</tr>
<tr>
<td>The called functions</td>
<td>-</td>
</tr>
</tbody>
</table>

channelx: DMA channel

DMA_CHx(x=0..6): DMA channel selection, refer to Table 3-190. Enum dma_channel_enum

<table>
<thead>
<tr>
<th>Input parameter(in)</th>
<th>address</th>
</tr>
</thead>
<tbody>
<tr>
<td>memory base address, 0 − 0xFFFFFFFF</td>
<td></td>
</tr>
</tbody>
</table>

| Output parameter(out) | - |

| Return value | - |

Example:
/* configure DMA channel0 memory address */
uint8_t g_destbuf[TRANSFER_NUM];
dma_memory_address_config(DMA_CH0, (uint32_t) g_destbuf);

**dma_transfer_number_config**

The description of `dma_transfer_number_config` is shown as below:

<table>
<thead>
<tr>
<th>Function name</th>
<th>dma_transfer_number_config</th>
</tr>
</thead>
<tbody>
<tr>
<td>Function prototype</td>
<td>void dma_transfer_number_config(dma_channel_enum channelx, uint32_t number);</td>
</tr>
<tr>
<td>Function descriptions</td>
<td>set the number of remaining data to be transferred by the DMA</td>
</tr>
<tr>
<td>Precondition</td>
<td>corresponding channel enable bit CHEN should be 0</td>
</tr>
<tr>
<td>The called functions</td>
<td>-</td>
</tr>
</tbody>
</table>

Table 3-204. Function `dma_transfer_number_config`

<table>
<thead>
<tr>
<th>Input parameter (in)</th>
<th>DMA channel</th>
</tr>
</thead>
<tbody>
<tr>
<td>channelx</td>
<td>DMA channel selection, refer to Table 3-190. Enum <code>dma_channel_enum</code></td>
</tr>
<tr>
<td>number</td>
<td>data transfer number(0x0-0xFFFF)</td>
</tr>
</tbody>
</table>

Output parameter (out)

| Return value | - |

Example:

/* configure DMA channel0 transfer number */
#define TRANSFER_NUM 0x400
dma_transfer_number_config(DMA_CH0, TRANSFER_NUM);

**dma_transfer_number_get**

The description of `dma_transfer_number_get` is shown as below:

<table>
<thead>
<tr>
<th>Function name</th>
<th>dma_transfer_number_get</th>
</tr>
</thead>
<tbody>
<tr>
<td>Function prototype</td>
<td>uint32_t dma_transfer_number_get(dma_channel_enum channelx);</td>
</tr>
<tr>
<td>Function descriptions</td>
<td>get the number of remaining data to be transferred by the DMA</td>
</tr>
<tr>
<td>Precondition</td>
<td>-</td>
</tr>
<tr>
<td>The called functions</td>
<td>-</td>
</tr>
</tbody>
</table>

Table 3-205. Function `dma_transfer_number_get`

<table>
<thead>
<tr>
<th>Input parameter (in)</th>
<th>DMA channel</th>
</tr>
</thead>
<tbody>
<tr>
<td>channelx</td>
<td>DMA channel selection, refer to Table 3-190. Enum <code>dma_channel_enum</code></td>
</tr>
</tbody>
</table>

Output parameter (out)
Return value

uint32_t | DMA data transmission remaining quantity (0x0-0xFFFF)

Example:

```c
/* get DMA channel0 transfer number */
uint32_t number = 0;
number = dma_transfer_number_get(DMA0, DMA_CH0);
```

dma_priority_config

The description of dma_priority_config is shown as below:

**Table 3-206. Function dma_priority_config**

<table>
<thead>
<tr>
<th>Function name</th>
<th>dma_priority_config</th>
</tr>
</thead>
<tbody>
<tr>
<td>Function prototype</td>
<td>void dma_priority_config(dma_channel_enum channelx, uint32_t priority);</td>
</tr>
<tr>
<td>Function descriptions</td>
<td>configure priority level of DMA channel</td>
</tr>
<tr>
<td>Precondition</td>
<td>corresponding channel enable bit CHEN should be 0</td>
</tr>
<tr>
<td>The called functions</td>
<td>-</td>
</tr>
<tr>
<td>Input parameter(in)</td>
<td></td>
</tr>
<tr>
<td>channelx</td>
<td>DMA channel</td>
</tr>
<tr>
<td>DMA_CHx(x=0..6)</td>
<td>DMA channel selection, refer to <em>Table 3-190. Enum dma_channel_enum</em></td>
</tr>
<tr>
<td>priority</td>
<td>priority Level of this channel</td>
</tr>
<tr>
<td>DMA_PRIORITY_LOW</td>
<td>low priority</td>
</tr>
<tr>
<td>DMA_PRIORITY_MEDIUM</td>
<td>medium priority</td>
</tr>
<tr>
<td>DMA_PRIORITY_HIGH</td>
<td>high priority</td>
</tr>
<tr>
<td>DMA_PRIORITY_ULTRA_HIGH</td>
<td>ultra high priority</td>
</tr>
<tr>
<td>Output parameter(out)</td>
<td>-</td>
</tr>
</tbody>
</table>

Example:

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

dma_memory_width_config

The description of dma_memory_width_config is shown as below:

**Table 3-207. Function dma_memory_width_config**

| Function name          | dma_memory_width_config                                 |

```c
/* configure DMA channel0 memory width */
dma_memory_width_config(DMA_CH0, DMA_MEMORY_WIDTH_32);   
```
### Function prototype

```c
void dma_memory_width_config(dma_channel_enum channelx, uint32_t mwidth);
```

### Function descriptions
configure transfer data size of memory

### Precondition
- corresponding channel enable bit CHEN should be 0

### The called functions
- 

#### Input parameter (in)
- 

<table>
<thead>
<tr>
<th>channelx</th>
<th>DMA channel</th>
</tr>
</thead>
<tbody>
<tr>
<td>DMA_CHx(x=0..6)</td>
<td>DMA channel selection, refer to <a href="#">Table 3-190. Enum dma_channel_enum</a></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>mwidth</th>
<th>transfer data width of memory</th>
</tr>
</thead>
<tbody>
<tr>
<td>DMA_MEMORY_WIDTH_8BIT</td>
<td>transfer data width of memory is 8-bit</td>
</tr>
<tr>
<td>DMA_MEMORY_WIDTH_16BIT</td>
<td>transfer data width of memory is 16-bit</td>
</tr>
<tr>
<td>DMA_MEMORY_WIDTH_32BIT</td>
<td>transfer data width of memory is 32-bit</td>
</tr>
</tbody>
</table>

#### Output parameter (out)
- 

#### Return value
- 

Example:

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

### dma_periph_width_config

The description of `dma_periph_width_config` is shown as below:

### Table 3-208. Function `dma_periph_width_config`

<table>
<thead>
<tr>
<th>Function name</th>
<th>dma_periph_width_config</th>
</tr>
</thead>
<tbody>
<tr>
<td>Function prototype</td>
<td>void dma_periph_width_config(dma_channel_enum channelx, uint32_t pwidth);</td>
</tr>
<tr>
<td>Function descriptions</td>
<td>configure transfer data width of peripheral</td>
</tr>
<tr>
<td>Precondition</td>
<td>corresponding channel enable bit CHEN should be 0</td>
</tr>
<tr>
<td>The called functions</td>
<td>-</td>
</tr>
</tbody>
</table>

#### Input parameter (in)
- 

<table>
<thead>
<tr>
<th>channelx</th>
<th>DMA channel</th>
</tr>
</thead>
<tbody>
<tr>
<td>DMA_CHx(x=0..6)</td>
<td>DMA channel selection, refer to <a href="#">Table 3-190. Enum dma_channel_enum</a></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>pwidth</th>
<th>transfer data width of peripheral</th>
</tr>
</thead>
<tbody>
<tr>
<td>DMA_PERIPHERAL_WIDTH_8BIT</td>
<td>transfer data width of peripheral is 8-bit</td>
</tr>
</tbody>
</table>
GD32L23x Firmware Library User Guide

<table>
<thead>
<tr>
<th>DMA_PERIPHERAL_WIDTH_16BIT</th>
<th>transfer data width of peripheral is 16-bit</th>
</tr>
</thead>
<tbody>
<tr>
<td>DMA_PERIPHERAL_WIDTH_32BIT</td>
<td>transfer data width of peripheral is 32-bit</td>
</tr>
</tbody>
</table>

Output parameter(out)

- -

Return value

- -

Example:

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

**dma_memory_increase_enable**

The description of *dma_memory_increase_enable* is shown as below:

**Table 3-209. Function dma_memory_increase_enable**

<table>
<thead>
<tr>
<th>Function name</th>
<th>dma_memory_increase_enable</th>
</tr>
</thead>
<tbody>
<tr>
<td>Function prototype</td>
<td>void dma_memory_increase_enable(dma_channel_enum channelx);</td>
</tr>
<tr>
<td>Function descriptions</td>
<td>enable next address increasement algorithm of memory</td>
</tr>
<tr>
<td>Precondition</td>
<td>corresponding channel enable bit CHEN should be 0</td>
</tr>
<tr>
<td>The called functions</td>
<td>-</td>
</tr>
</tbody>
</table>

Input parameter(in)

channelx DMA channel

DMA_CHx(x=0..6) DMA channel selection, refer to *Table 3-190. Enum dma_channel_enum*

Output parameter(out)

- -

Return value

- -

Example:

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

**dma_memory_increase_disable**

The description of *dma_memory_increase_disable* is shown as below:

**Table 3-210. Function dma_memory_increase_disable**

<table>
<thead>
<tr>
<th>Function name</th>
<th>dma_memory_increase_disable</th>
</tr>
</thead>
<tbody>
<tr>
<td>Function prototype</td>
<td>void dma_memory_increase_disable(dma_channel_enum channelx);</td>
</tr>
<tr>
<td>Function descriptions</td>
<td>disable next address increasement algorithm of memory</td>
</tr>
<tr>
<td>Precondition</td>
<td>corresponding channel enable bit CHEN should be 0</td>
</tr>
</tbody>
</table>
The called functions

<table>
<thead>
<tr>
<th>Input parameter(in)</th>
<th>Output parameter(out)</th>
<th>Return value</th>
</tr>
</thead>
<tbody>
<tr>
<td>channelx DMA channel</td>
<td>DMA channel selection, refer to Table 3-190. Enum dma_channel_enum</td>
<td>-</td>
</tr>
<tr>
<td>DMA_CHx(x=0..6)</td>
<td>-</td>
<td>-</td>
</tr>
</tbody>
</table>

Example:

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

dma_periph_increase_enable

The description of dma_periph_increase_enable is shown as below:

Table 3-211. Function dma_periph_increase_enable

<table>
<thead>
<tr>
<th>Function name</th>
<th>dma_periph_increase_enable</th>
</tr>
</thead>
<tbody>
<tr>
<td>Function prototype</td>
<td>void dma_periph_increase_enable(dma_channel_enum channelx);</td>
</tr>
<tr>
<td>Function descriptions</td>
<td>enable next address increasement algorithm of peripheral</td>
</tr>
<tr>
<td>Precondition</td>
<td>corresponding channel enable bit CHEN should be 0</td>
</tr>
<tr>
<td>The called functions</td>
<td>-</td>
</tr>
<tr>
<td>Input parameter(in)</td>
<td></td>
</tr>
<tr>
<td>channelx DMA channel</td>
<td></td>
</tr>
<tr>
<td>DMA_CHx(x=0..6)</td>
<td>DMA channel selection, refer to Table 3-190. Enum dma_channel_enum</td>
</tr>
<tr>
<td>Output parameter(out)</td>
<td>-</td>
</tr>
<tr>
<td>Return value</td>
<td>-</td>
</tr>
</tbody>
</table>

Example:

`/* enable next address increasement algorithm of DMA channel0 */
dma_periph_increase_enable(DMA_CH0);`

dma_periph_increase_disable

The description of dma_periph_increase_disable is shown as below:

Table 3-212. Function dma_periph_increase_disable

<table>
<thead>
<tr>
<th>Function name</th>
<th>dma_periph_increase_disable</th>
</tr>
</thead>
<tbody>
<tr>
<td>Function prototype</td>
<td>void dma_periph_increase_disable(dma_channel_enum channelx);</td>
</tr>
<tr>
<td>Function descriptions</td>
<td>disable next address increasement algorithm of peripheral</td>
</tr>
<tr>
<td>Precondition</td>
<td>corresponding channel enable bit CHEN should be 0</td>
</tr>
</tbody>
</table>
The called functions: dma_transfer_direction_config

<table>
<thead>
<tr>
<th>Input parameter(in)</th>
<th>Output parameter(out)</th>
<th>Return value</th>
</tr>
</thead>
<tbody>
<tr>
<td>channelx DMA channel</td>
<td>DMA channel selection, refer to Table 3-190. Enum dma_channel_enum</td>
<td>-</td>
</tr>
<tr>
<td>DMA_CHx(x=0..6)</td>
<td>-</td>
<td>-</td>
</tr>
</tbody>
</table>

Example:

`/* disable next address incrementation algorithm of DMA channel0 */
dma_periph_increase_disable(DMA_CH0);`

**dma_transfer_direction_config**

The description of dma_transfer_direction_config is shown as below:

**Table 3-213. Function dma_transfer_direction_config**

<table>
<thead>
<tr>
<th>Function name</th>
<th>dma_transfer_direction_config</th>
</tr>
</thead>
</table>

Function prototype: void dma_transfer_direction_config(dma_channel_enum channelx, uint32_t direction);

Function description: configure the direction of data transfer on the channel

Precondition: corresponding channel enable bit CHEN should be 0

The called functions: -

<table>
<thead>
<tr>
<th>Input parameter(in)</th>
<th>Output parameter(out)</th>
<th>Return value</th>
</tr>
</thead>
<tbody>
<tr>
<td>channelx DMA channel</td>
<td>DMA channel selection, refer to Table 3-190. Enum dma_channel_enum</td>
<td>-</td>
</tr>
<tr>
<td>DMA_CHx(x=0..6)</td>
<td>-</td>
<td>-</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>direction specify the direction of data transfer</th>
</tr>
</thead>
<tbody>
<tr>
<td>DMA_PERIPHERAL_TO_MEMORY read from peripheral and write to memory</td>
</tr>
<tr>
<td>DMA_MEMORY_TO_PERIPHERAL read from memory and write to peripheral</td>
</tr>
</tbody>
</table>

Example:

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

The description of dma_flag_get is shown as below:

**Table 3-214. Function dma_flag_get**

<table>
<thead>
<tr>
<th>Function name</th>
<th>dma_flag_get</th>
</tr>
</thead>
<tbody>
<tr>
<td>Function prototype</td>
<td>FlagStatus dma_flag_get(dma_channel_enum channelx, uint32_t flag);</td>
</tr>
<tr>
<td>Function descriptions</td>
<td>check DMA flag is set or not</td>
</tr>
<tr>
<td>Precondition</td>
<td>-</td>
</tr>
<tr>
<td>The called functions</td>
<td>-</td>
</tr>
<tr>
<td><strong>Input parameter(in)</strong></td>
<td></td>
</tr>
<tr>
<td>channelx</td>
<td>DMA channel</td>
</tr>
<tr>
<td>DMA_CHx(x=0..6)</td>
<td>DMA channel selection, refer to <a href="#">Table 3-190. Enum dma_channel_enum</a></td>
</tr>
<tr>
<td><strong>Input parameter(in)</strong></td>
<td></td>
</tr>
<tr>
<td>flag</td>
<td>specify get which flag</td>
</tr>
<tr>
<td>DMA_FLAG_G</td>
<td>global interrupt flag of channel</td>
</tr>
<tr>
<td>DMA_FLAG_FTF</td>
<td>full transfer finish flag of channel</td>
</tr>
<tr>
<td>DMA_FLAG_HTF</td>
<td>half transfer finish flag of channel</td>
</tr>
<tr>
<td>DMA_FLAG_ERR</td>
<td>error flag of channel</td>
</tr>
<tr>
<td><strong>Output parameter(out)</strong></td>
<td>-</td>
</tr>
<tr>
<td><strong>Return value</strong></td>
<td>FlagStatus SET or RESET</td>
</tr>
</tbody>
</table>

Example:

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

dma_flag_clear

The description of dma_flag_clear is shown as below:

**Table 3-215. Function dma_flag_clear**

<table>
<thead>
<tr>
<th>Function name</th>
<th>dma_flag_clear</th>
</tr>
</thead>
<tbody>
<tr>
<td>Function prototype</td>
<td>void dma_flag_clear(dma_channel_enum channelx, uint32_t flag);</td>
</tr>
<tr>
<td>Function descriptions</td>
<td>clear DMA a channel flag</td>
</tr>
<tr>
<td>Precondition</td>
<td>-</td>
</tr>
<tr>
<td>The called functions</td>
<td>-</td>
</tr>
<tr>
<td><strong>Input parameter(in)</strong></td>
<td></td>
</tr>
<tr>
<td>channelx</td>
<td>DMA channel</td>
</tr>
<tr>
<td>DMA_CHx(x=0..6)</td>
<td>DMA channel selection, refer to <a href="#">Table 3-190. Enum dma_channel_enum</a></td>
</tr>
<tr>
<td><strong>Input parameter(in)</strong></td>
<td></td>
</tr>
<tr>
<td>flag</td>
<td>specify get which flag</td>
</tr>
</tbody>
</table>
GD32L23x Firmware Library User Guide

<table>
<thead>
<tr>
<th>DMA_FLAG_G</th>
<th>global interrupt flag of channel</th>
</tr>
</thead>
<tbody>
<tr>
<td>DMA_FLAG_FTF</td>
<td>full transfer finish flag of channel</td>
</tr>
<tr>
<td>DMA_FLAG_HTF</td>
<td>half transfer finish flag of channel</td>
</tr>
<tr>
<td>DMA_FLAG_ERR</td>
<td>error flag of channel</td>
</tr>
</tbody>
</table>

Output parameter[out]

- -

Return value

- -

Example:

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

**dma_interrupt_flag_get**

The description of *dma_interrupt_flag_get* is shown as below:

**Table 3-216. Function dma_interrupt_flag_get**

<table>
<thead>
<tr>
<th>Function name</th>
<th>dma_interrupt_flag_get</th>
</tr>
</thead>
<tbody>
<tr>
<td>Function prototype</td>
<td>FlagStatus dma_interrupt_flag_get(dma_channel_enum channelx, uint32_t flag);</td>
</tr>
<tr>
<td>Function descriptions</td>
<td>check DMA flag and interrupt enable bit is set or not</td>
</tr>
<tr>
<td>Precondition</td>
<td>-</td>
</tr>
<tr>
<td>The called functions</td>
<td>-</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Input parameter[in]</th>
</tr>
</thead>
<tbody>
<tr>
<td>channelx</td>
</tr>
<tr>
<td>DMA_CHx(x=0..6)</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Input parameter[in]</th>
</tr>
</thead>
<tbody>
<tr>
<td>flag</td>
</tr>
<tr>
<td>DMA_INT_FLAG_FTF</td>
</tr>
<tr>
<td>DMA_INT_FLAG_HTF</td>
</tr>
<tr>
<td>DMA_INT_FLAG_ERR</td>
</tr>
</tbody>
</table>

Output parameter[out]

- -

Return value

FlagStatus | SET or RESET |

Example:

```c
/* get DMA interrupt_flag */
if(dma_interrupt_flag_get(DMA_CH3, DMA_INT_FLAG_FTF)){
    dma_interrupt_flag_clear(DMA_CH3, DMA_INT_FLAG_G);
}
```
dma_interrupt_flag_clear

The description of dma_interrupt_flag_clear is shown as below:

Table 3-217. Function dma_interrupt_flag_clear

<table>
<thead>
<tr>
<th>Function name</th>
<th>dma_interrupt_flag_clear</th>
</tr>
</thead>
<tbody>
<tr>
<td>Function prototype</td>
<td>void dma_interrupt_flag_clear(dma_channel_enum channelx, uint32_t flag);</td>
</tr>
<tr>
<td>Function descriptions</td>
<td>clear DMA a channel flag</td>
</tr>
<tr>
<td>Precondition</td>
<td>-</td>
</tr>
<tr>
<td>The called functions</td>
<td>-</td>
</tr>
</tbody>
</table>

| Input parameter(in)    |                              |
| channelx               | DMA channel                  |
| DMA_CHx(x=0..6)        | DMA channel selection, refer to Table 3-190. Enum dma_channel_enum |

| Input parameter(in)    |                              |
| flag                   | specify get which flag       |
| DMA_INT_FLAG_G         | global interrupt flag of channel |
| DMA_INT_FLAG_FTF       | full transfer finish interrupt flag of channel |
| DMA_INT_FLAG_HTF       | half transfer finish interrupt flag of channel |
| DMA_INT_FLAG_ERR       | error interrupt flag of channel |

| Output parameter(out)  |                              |
|                        | -                            |

| Return value           | -                            |

Example:

/* get DMA interrupt_flag */
if(dma_interrupt_flag_get(DMA_CH3, DMA_INT_FLAG_FTF)){
   dma_interrupt_flag_clear(DMA_CH3, DMA_INT_FLAG_G);
}

dma_interrupt_enable

The description of dma_interrupt_enable is shown as below:

Table 3-218. Function dma_interrupt_enable

<table>
<thead>
<tr>
<th>Function name</th>
<th>dma_interrupt_enable</th>
</tr>
</thead>
<tbody>
<tr>
<td>Function prototype</td>
<td>void dma_interrupt_enable(dma_channel_enum channelx, uint32_t source);</td>
</tr>
<tr>
<td>Function descriptions</td>
<td>enable DMA interrupt</td>
</tr>
<tr>
<td>Precondition</td>
<td>-</td>
</tr>
<tr>
<td>The called functions</td>
<td>-</td>
</tr>
</tbody>
</table>

| Input parameter(in)    |                              |
| channelx               | DMA channel                  |
| DMA_CHx(x=0..6)        | DMA channel selection, refer to Table 3-190. Enum dma_channel_enum |

| Input parameter(in)    |                              |
GD32L23x Firmware Library User Guide

source | DMA interrupt source
---|---
DMA_INT_FTF | full transfer finish interrupt of channel
DMA_INT_HTF | half transfer finish interrupt of channel
DMA_INT_ERR | error interrupt of channel

Output parameter(out)
- -

Return value
- -

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

dma_interrupt_disable

The description of dma_interrupt_disable is shown as below:

Table 3-219. Function dma_interrupt_disable

<table>
<thead>
<tr>
<th>Function name</th>
<th>dma_interrupt_disable</th>
</tr>
</thead>
<tbody>
<tr>
<td>Function prototype</td>
<td>void dma_interrupt_disable(dma_channel_enum channelx, uint32_t source);</td>
</tr>
<tr>
<td>Function descriptions</td>
<td>disable DMA interrupt</td>
</tr>
<tr>
<td>Precondition</td>
<td>-</td>
</tr>
<tr>
<td>The called functions</td>
<td>-</td>
</tr>
<tr>
<td>Input parameter(in)</td>
<td></td>
</tr>
<tr>
<td>channelx</td>
<td>DMA channel</td>
</tr>
<tr>
<td>DMA_CHx(x=0..6)</td>
<td>DMA channel selection, refer to Table 3-190. Enum dma_channel_enum</td>
</tr>
</tbody>
</table>

Input parameter(in)

source | DMA interrupt source
---|---
DMA_INT_FTF | full transfer finish interrupt of channel
DMA_INT_HTF | half transfer finish interrupt of channel
DMA_INT_ERR | error interrupt of channel

Output parameter(out)
- -

Return value
- -

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

dmamux_sync_struct_para_init

The description of dmamux_sync_struct_para_init is shown as below:
### Table 3-220. Function dmamux_sync_struct_para_init

<table>
<thead>
<tr>
<th>Function name</th>
<th>dmamux_sync_struct_para_init</th>
</tr>
</thead>
<tbody>
<tr>
<td>Function prototype</td>
<td>void dmamux_sync_struct_para_init(dmamux_sync_parameter_struct *init_struct);</td>
</tr>
<tr>
<td>Function descriptions</td>
<td>initialize the parameters of DMAMUX synchronization mode structure with the default values</td>
</tr>
<tr>
<td>Precondition</td>
<td>-</td>
</tr>
<tr>
<td>The called functions</td>
<td>-</td>
</tr>
</tbody>
</table>

**Input parameter (in)**
- init_struct

**Output parameter (out)**
- init_struct

The initialization data needed to initialize DMUX request multiplexer channel synchronization mode, refer to **Table 3-185. Structure dmamux_sync_parameter_struct**

**Return value**
- -

Example:

```c
/* initialize DMAMUX synchronization mode structure */
dmamux_sync_parameter_struct dmamux_sync_init_struct;
dmamux_sync_struct_para_init(&dmamux_sync_init_struct);
```

**dmamux_synchronization_init**

The description of dmamux_synchronization_init is shown as below:

### Table 3-221. Function dmamux_synchronization_init

<table>
<thead>
<tr>
<th>Function name</th>
<th>dmamux_synchronization_init</th>
</tr>
</thead>
<tbody>
<tr>
<td>Function prototype</td>
<td>void dmamux_synchronization_init(dmamux_multiplexer_channel_enum channelx, dmamux_sync_parameter_struct *init_struct);</td>
</tr>
<tr>
<td>Function descriptions</td>
<td>initialize DMAMUX request multiplexer channel synchronization mode</td>
</tr>
<tr>
<td>Precondition</td>
<td>-</td>
</tr>
<tr>
<td>The called functions</td>
<td>-</td>
</tr>
<tr>
<td><strong>Input parameter (in)</strong></td>
<td></td>
</tr>
<tr>
<td>channelx</td>
<td>specify which DMAMUX request multiplexer channel is initialized</td>
</tr>
<tr>
<td><strong>DMAMUX_MUXCHx(x=0..6)</strong></td>
<td>DMAMUX channel selection, refer to <strong>Table 3-191. Enum dmamux_multiplexer_channel_enum</strong></td>
</tr>
<tr>
<td><strong>Input parameter (in)</strong></td>
<td></td>
</tr>
<tr>
<td>init_struct</td>
<td>the initialization data needed to initialize DMAMUX request multiplexer channel synchronization mode, refer to <strong>Table 3-185. Structure dmamux_sync_parameter_struct</strong></td>
</tr>
<tr>
<td><strong>Output parameter (out)</strong></td>
<td></td>
</tr>
<tr>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td><strong>Return value</strong></td>
<td>-</td>
</tr>
</tbody>
</table>
Example:

```c
/* 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);
```

dmamux_synchronization_enable

The description of dmamux_synchronization_enable is shown as below:

<table>
<thead>
<tr>
<th>Function name</th>
<th>dmamux_synchronization_enable</th>
</tr>
</thead>
<tbody>
<tr>
<td>Function prototype</td>
<td>void dmamux_synchronization_enable(dmamux_multiplexer_channel_enum channelx);</td>
</tr>
<tr>
<td>Function descriptions</td>
<td>enable synchronization mode</td>
</tr>
<tr>
<td>Precondition</td>
<td>-</td>
</tr>
<tr>
<td>The called functions</td>
<td>-</td>
</tr>
</tbody>
</table>

**Table 3-222. Function dmamux_synchronization_enable**

<table>
<thead>
<tr>
<th>Input parameter(in)</th>
<th>channelx</th>
<th>specify which DMAMUX request multiplexer channel is initialized</th>
</tr>
</thead>
<tbody>
<tr>
<td>DMAMUX_MUXCHx(x=0..6)</td>
<td>DMAMUX channel selection, refer to Table 3-191. Enum dmamux_multiplexer_channel_enum</td>
<td></td>
</tr>
</tbody>
</table>

**Output parameter(out)**

| - | - | |

**Return value**

| - | - | |

Example:

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

dmamux_synchronization_disable

The description of dmamux_synchronization_disable is shown as below:

<table>
<thead>
<tr>
<th>Function name</th>
<th>dmamux_synchronization_disable</th>
</tr>
</thead>
<tbody>
<tr>
<td>Function prototype</td>
<td>void dmamux_synchronization_disable(dmamux_multiplexer_channel_enum channelx);</td>
</tr>
</tbody>
</table>

```c
/* enable synchronization mode */
dmamux_synchronization_disable(DMAMUX_MUXCH0);
```
**Function descriptions**

disable synchronization mode

**Precondition**
-

**The called functions**
-

<table>
<thead>
<tr>
<th><strong>Input parameter (in)</strong></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>channelx</strong></td>
<td>specify which DMAMUX request multiplexer channel is initialized</td>
</tr>
<tr>
<td><strong>DMAMUX_MUXCHx(x=0..6)</strong></td>
<td>DMAMUX channel selection, refer to Table 3-191, Enum dmamux_multiplexer_channel_enum</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th><strong>Output parameter (out)</strong></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>-</td>
<td>-</td>
</tr>
</tbody>
</table>

**Return value**
-

Example:
/* disable synchronization mode */
dmamux_synchronization_disable(DMAMUX_MUXCH0);

dmamux_event_generation_enable

The description of dmamux_event_generation_enable is shown as below:

**Table 3-224. Function dmamux_event_generation_enable**

<table>
<thead>
<tr>
<th><strong>Function name</strong></th>
<th>dmamux_event_generation_enable</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Function prototype</strong></td>
<td>void dmamux_event_generation_enable(dmamux_multiplexer_channel_enum channelx);</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th><strong>Function descriptions</strong></th>
<th>enable event generation</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Precondition</strong></td>
<td>-</td>
</tr>
<tr>
<td><strong>The called functions</strong></td>
<td>-</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th><strong>Input parameter (in)</strong></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>channelx</strong></td>
<td>specify which DMAMUX request multiplexer channel is initialized</td>
</tr>
<tr>
<td><strong>DMAMUX_MUXCHx(x=0..6)</strong></td>
<td>DMAMUX channel selection, refer to Table 3-191, Enum dmamux_multiplexer_channel_enum</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th><strong>Output parameter (out)</strong></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>-</td>
<td>-</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th><strong>Return value</strong></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>-</td>
<td>-</td>
</tr>
</tbody>
</table>

Example:
/* enable event generation */
dmamux_event_generation_enable(DMAMUX_MUXCH0);

dmamux_event_generation_disable

The description of dmamux_event_generation_disable is shown as below:
Table 3-225. Function dmamux_event_generation_disable

<table>
<thead>
<tr>
<th>Function name</th>
<th>dmamux_event_generation_disable</th>
</tr>
</thead>
<tbody>
<tr>
<td>Function prototype</td>
<td>void dmamux_event_generation_disable(dmamux_multiplexer_channel_enum channelx);</td>
</tr>
<tr>
<td>Function descriptions</td>
<td>disable event generation</td>
</tr>
<tr>
<td>Precondition</td>
<td>-</td>
</tr>
<tr>
<td>The called functions</td>
<td>-</td>
</tr>
</tbody>
</table>

**Input parameter (in)**

<table>
<thead>
<tr>
<th>channelx</th>
<th>specify which DMAMUX request multiplexer channel is initialized</th>
</tr>
</thead>
<tbody>
<tr>
<td>DMAMUX_MUXCHx(x=0..6)</td>
<td>DMAMUX channel selection, refer to Table 3-191, Enum dmamux_multiplexer_channel_enum</td>
</tr>
</tbody>
</table>

**Output parameter (out)**

| -                      | -                              |

**Return value**

| -                      | -                              |

Example:

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

dmamux_gen_struct_para_init

The description of dmamux_gen_struct_para_init is shown as below:

Table 3-226. Function dmamux_gen_struct_para_init

<table>
<thead>
<tr>
<th>Function name</th>
<th>dmamux_gen_struct_para_init</th>
</tr>
</thead>
<tbody>
<tr>
<td>Function prototype</td>
<td>void dmamux_gen_struct_para_init(dmamux_gen_parameter_struct *init_struct);</td>
</tr>
<tr>
<td>Function descriptions</td>
<td>initialize the parameters of DMAMUX request generator structure with the default values</td>
</tr>
<tr>
<td>Precondition</td>
<td>-</td>
</tr>
<tr>
<td>The called functions</td>
<td>-</td>
</tr>
</tbody>
</table>

**Input parameter (in)**

| -                      | -                              |

**Output parameter (out)**

| init_struct            | the initialization data needed to initialize DMAMUX request generator channel, refer to Table 3-186, Structure dmamux_gen_parameter_struct |

**Return value**

| -                      | -                              |

Example:

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

dmamux_request_generator_init

The description of dmamux_request_generator_init is shown as below:

Table 3-227. Function dmamux_request_generator_init

<table>
<thead>
<tr>
<th>Function name</th>
<th>dmamux_request_generator_init</th>
</tr>
</thead>
<tbody>
<tr>
<td>Function prototype</td>
<td>void dmamux_request_generator_init(dmamux_generator_channel_enum channelx, dmamux_gen_parameter_struct *init_struct);</td>
</tr>
<tr>
<td>Function descriptions</td>
<td>initialize DMAMUX request generator channel</td>
</tr>
<tr>
<td>Precondition</td>
<td>-</td>
</tr>
<tr>
<td>The called functions</td>
<td>-</td>
</tr>
</tbody>
</table>

**Input parameter(in)**

channelx, specify which DMAMUX request generator channel is initialized

DMAMUX_GENCHx(x=0..3), DMAMUX generation channel selection, refer to Table 3-192. Enum dmamux_generator_channel_enum

**Input parameter(in)**

init_struct, the initialization data needed to initialize DMAMUX request generator channel, refer to Table 3-186. Structure dmamux_gen_parameter_struct

**Output parameter(out)**

- |

**Return value**

- |

Example:

```c
/* 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);
```

dmamux_request_generator_chennel_enable

The description of dmamux_request_generator_chennel_enable is shown as below:

Table 3-228. Function dmamux_request_generator_chennel_enable

<table>
<thead>
<tr>
<th>Function name</th>
<th>dmamux_request_generator_chennel_enable</th>
</tr>
</thead>
<tbody>
<tr>
<td>Function prototype</td>
<td>void dmamux_request_generator_chennel_enable(dmamux_generator_channel_enum channelx);</td>
</tr>
<tr>
<td>Function descriptions</td>
<td>enable DMAMUX request generator channel</td>
</tr>
<tr>
<td>Precondition</td>
<td>-</td>
</tr>
</tbody>
</table>
### The called functions

<table>
<thead>
<tr>
<th>Input parameter(in)</th>
<th>channelx</th>
<th>specify which DMAMUX request generator channel is initialized</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>DMAMUX_GENCHx(x=0..3)</td>
<td>DMAMUX generation channel selection, refer to Table 3-192. Enum dmamux_generator_channel_enum</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Output parameter(out)</th>
<th>-</th>
</tr>
</thead>
</table>

**Return value**

- 

Example:

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

#### dmamux_request_generator_channel_disable

The description of `dmamux_request_generator_channel_disable` is shown as below:

**Table 3-229. Function dmamux_request_generator_channel_disable**

<table>
<thead>
<tr>
<th>Function name</th>
<th>dmamux_request_generator_channel_disable</th>
</tr>
</thead>
<tbody>
<tr>
<td>Function prototype</td>
<td>void</td>
</tr>
<tr>
<td>Function descriptions</td>
<td>disable DMAMUX request generator channel</td>
</tr>
<tr>
<td>Precondition</td>
<td>-</td>
</tr>
<tr>
<td>The called functions</td>
<td>-</td>
</tr>
</tbody>
</table>

**Input parameter(in)**

<table>
<thead>
<tr>
<th>Input parameter(in)</th>
<th>channelx</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>DMAMUX_GENCHx(x=0..3)</td>
</tr>
</tbody>
</table>

DMAMUX generation channel selection, refer to Table 3-192. Enum dmamux_generator_channel_enum

**Output parameter(out)**

- 

**Return value**

- 

Example:

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

#### dmamux_synchronization_polarity_config

The description of `dmamux_synchronization_polarity_config` is shown as below:

**Table 3-230. Function dmamux_synchronization_polarity_config**

<table>
<thead>
<tr>
<th>Function name</th>
<th>dmamux_synchronization_polarity_config</th>
</tr>
</thead>
</table>

- 

Example:

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

```c
void dmamux_synchronization_polarity_config(dmamux_multiplexer_channel_enum channelx, uint32_t polarity);
```

Function descriptions
configure synchronization input polarity

Precondition
-

The called functions
-

Input parameter(in)

<table>
<thead>
<tr>
<th>channelx</th>
<th>description</th>
</tr>
</thead>
<tbody>
<tr>
<td>DMAMUX_MUXCHx(x=0..6)</td>
<td>specify which DMAMUX request multiplexer channel is initialized</td>
</tr>
</tbody>
</table>

Input parameter(in)

<table>
<thead>
<tr>
<th>polarity</th>
<th>description</th>
</tr>
</thead>
<tbody>
<tr>
<td>DMAMUX_SYNC_NO_EVENT</td>
<td>no event detection</td>
</tr>
<tr>
<td>DMAMUX_SYNC_RISING</td>
<td>rising edge</td>
</tr>
<tr>
<td>DMAMUX_SYNC_FALLING</td>
<td>falling edge</td>
</tr>
<tr>
<td>DMAMUX_SYNC_RISING_FALLING</td>
<td>rising and falling edges</td>
</tr>
</tbody>
</table>

Output parameter(out)
-

Return value
-

Example:

```c
/* configure synchronization input polarity */
dmamux_synchronization_polarity_config(DMAMUX_MUXCH0, DMAMUX_SYNC_RISING);
```

dmamux_request_forward_number_config

The description of dmamux_request_forward_number_config is shown as below:

Table 3-231. Function dmamux_request_forward_number_config

<table>
<thead>
<tr>
<th>Function name</th>
<th>dmamux_request_forward_number_config</th>
</tr>
</thead>
<tbody>
<tr>
<td>Function prototype</td>
<td>void dmamux_request_forward_number_config(dmamux_multiplexer_channel_enum channelx, uint32_t number);</td>
</tr>
<tr>
<td>Function descriptions</td>
<td>configure number of DMA requests to forward</td>
</tr>
<tr>
<td>Precondition</td>
<td>-</td>
</tr>
<tr>
<td>The called functions</td>
<td>-</td>
</tr>
</tbody>
</table>

Input parameter(in)

<table>
<thead>
<tr>
<th>channelx</th>
<th>description</th>
</tr>
</thead>
<tbody>
<tr>
<td>DMAMUX_MUXCHx(x=0..6)</td>
<td>specify which DMAMUX request multiplexer channel is initialized</td>
</tr>
</tbody>
</table>

DMAMUX channel selection, refer to Table 3-191. Enum dmamux_multiplexer_channel_enum
Example:

```c
/* configure number of DMA requests to forward */
dmamux_request_forward_number_config(DMAMUX_MUXCH0, 4);
```

**dmamux_sync_id_config**

The description of `dmamux_sync_id_config` is shown as below:

**Table 3-232. Function dmamux_sync_id_config**

<table>
<thead>
<tr>
<th>Function name</th>
<th>dmamux_sync_id_config</th>
</tr>
</thead>
<tbody>
<tr>
<td>Function prototype</td>
<td><code>void dmamux_sync_id_config(dmamux_multiplexer_channel_enum channelx, uint32_t id);</code></td>
</tr>
<tr>
<td>Function descriptions</td>
<td>configure synchronization input identification</td>
</tr>
<tr>
<td>Precondition</td>
<td>-</td>
</tr>
<tr>
<td>The called functions</td>
<td>-</td>
</tr>
</tbody>
</table>

**Input parameter(in)**

<table>
<thead>
<tr>
<th>channelx</th>
<th>specify which DMAMUX request multiplexer channel is initialized</th>
</tr>
</thead>
<tbody>
<tr>
<td>DMAMUX_MUXCHx(x=0..6)</td>
<td>DMAMUX channel selection, refer to <a href="#">Table 3-191. Enum</a></td>
</tr>
</tbody>
</table>

**Input parameter(in)**

<table>
<thead>
<tr>
<th>id</th>
<th>synchronization input identification</th>
</tr>
</thead>
<tbody>
<tr>
<td>DMAMUX_SYNC EXTI0</td>
<td>synchronization input is EXTI0</td>
</tr>
<tr>
<td>DMAMUX_SYNC EXTI1</td>
<td>synchronization input is EXTI1</td>
</tr>
<tr>
<td>DMAMUX_SYNC EXTI2</td>
<td>synchronization input is EXTI2</td>
</tr>
<tr>
<td>DMAMUX_SYNC EXTI3</td>
<td>synchronization input is EXTI3</td>
</tr>
<tr>
<td>DMAMUX_SYNC EXTI4</td>
<td>synchronization input is EXTI4</td>
</tr>
<tr>
<td>DMAMUX_SYNC EXTI5</td>
<td>synchronization input is EXTI5</td>
</tr>
<tr>
<td>DMAMUX_SYNC EXTI6</td>
<td>synchronization input is EXTI6</td>
</tr>
<tr>
<td>DMAMUX_SYNC EXTI7</td>
<td>synchronization input is EXTI7</td>
</tr>
</tbody>
</table>
### DMAMUX_SYNC_EXTI_SYNCHRONIZATION_INPUT
- **8** synchronization input is EXTI8
- **9** synchronization input is EXTI9
- **10** synchronization input is EXTI10
- **11** synchronization input is EXTI11
- **12** synchronization input is EXTI12
- **13** synchronization input is EXTI13
- **14** synchronization input is EXTI14
- **15** synchronization input is EXTI15
- **0_OUT** synchronization input is Evt0_out
- **1_OUT** synchronization input is Evt1_out
- **2_OUT** synchronization input is Evt2_out
- **3_OUT** synchronization input is Evt3_out
- **TIMER11_CH0_O** synchronization input is TIMER11_CH0_O

### Output parameter(out)

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>-</td>
</tr>
</tbody>
</table>

### Return value

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>-</td>
</tr>
</tbody>
</table>

Example:

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

**dmamux_request_id_config**

The description of dmamux_request_id_config is shown as below:
### Table 3-233. Function dmamux_request_id_config

<table>
<thead>
<tr>
<th>Function name</th>
<th>dmamux_request_id_config</th>
</tr>
</thead>
<tbody>
<tr>
<td>Function prototype</td>
<td>void dmamux_request_id_config(dmamux_multiplexer_channel_enum channelx, uint32_t id);</td>
</tr>
<tr>
<td>Function descriptions</td>
<td>configure multiplexer input identification</td>
</tr>
<tr>
<td>Precondition</td>
<td>-</td>
</tr>
<tr>
<td>The called functions</td>
<td>-</td>
</tr>
<tr>
<td><strong>Input parameter(in)</strong></td>
<td></td>
</tr>
<tr>
<td>channelx</td>
<td>specify which DMAMUX request multiplexer channel is initialized</td>
</tr>
<tr>
<td>DMAMUX_MUXCHx(x=0..6)</td>
<td>DMAMUX channel selection, refer to Table 3-191. Enum dmamux_multiplexer_channel_enum</td>
</tr>
<tr>
<td><strong>Input parameter(in)</strong></td>
<td></td>
</tr>
<tr>
<td>id</td>
<td>input DMA request identification</td>
</tr>
</tbody>
</table>

- DMA_REQUEST_M2M: memory to memory transfer
- DMA_REQUEST_GENERATOR0: DMAMUX request generator 0
- DMA_REQUEST_GENERATOR1: DMAMUX request generator 1
- DMA_REQUEST_GENERATOR2: DMAMUX request generator 2
- DMA_REQUEST_GENERATOR3: DMAMUX request generator 3
- DMA_REQUEST_ADC: DMAMUX ADC request
- DMA_REQUEST_DAC: DMAMUX DAC request
- DMA_REQUEST_I2C0_RX: DMAMUX I2C0 RX request
- DMA_REQUEST_I2C0_TX: DMAMUX I2C0 TX request
- DMA_REQUEST_I2C1_RX: DMAMUX I2C1 RX request
- DMA_REQUEST_I2C1_TX: DMAMUX I2C1 TX request
- DMA_REQUEST_I2C2_RX: DMAMUX I2C2 RX request
- DMA_REQUEST_I2C2_TX: DMAMUX I2C2 TX request
- DMA_REQUEST_SPI0_RX: DMAMUX SPI0 RX request
- DMA_REQUEST_SPI0_TX: DMAMUX SPI0 TX request
- DMA_REQUEST_SPI1_RX: DMAMUX SPI1 RX request
- DMA_REQUEST_SPI1_TX: DMAMUX SPI1 TX request
<table>
<thead>
<tr>
<th>function</th>
<th>description</th>
</tr>
</thead>
<tbody>
<tr>
<td>DMA_REQUEST_TIME _TX</td>
<td>DMAMUX TIMER1 CH0 request</td>
</tr>
<tr>
<td>DMA_REQUEST_TIME R1_CH</td>
<td>DMAMUX TIMER1 CH1 request</td>
</tr>
<tr>
<td>DMA_REQUEST_TIME R1_CH1</td>
<td>DMAMUX TIMER1 CH2 request</td>
</tr>
<tr>
<td>DMA_REQUEST_TIME R1_CH2</td>
<td>DMAMUX TIMER1 CH3 request</td>
</tr>
<tr>
<td>DMA_REQUEST_TIME R1_CH3</td>
<td>DMAMUX TIMER1 UP request</td>
</tr>
<tr>
<td>DMA_REQUEST_TIME R2_CH0</td>
<td>DMAMUX TIMER2 CH0 request</td>
</tr>
<tr>
<td>DMA_REQUEST_TIME R2_CH1</td>
<td>DMAMUX TIMER2 CH1 request</td>
</tr>
<tr>
<td>DMA_REQUEST_TIME R2_CH2</td>
<td>DMAMUX TIMER2 CH2 request</td>
</tr>
<tr>
<td>DMA_REQUEST_TIME R2_CH3</td>
<td>DMAMUX TIMER2 CH3 request</td>
</tr>
<tr>
<td>DMA_REQUEST_TIME R2_TRIG</td>
<td>DMAMUX TIMER2 TRIG request</td>
</tr>
<tr>
<td>DMA_REQUEST_TIME R2_UP</td>
<td>DMAMUX TIMER2 UP request</td>
</tr>
<tr>
<td>DMA_REQUEST_TIME R5_UP</td>
<td>DMAMUX TIMER5 UP request</td>
</tr>
<tr>
<td>DMA_REQUEST_TIME R6_UP</td>
<td>DMAMUX TIMER6 UP request</td>
</tr>
<tr>
<td>DMA_REQUEST_CAU _IN</td>
<td>DMAMUX CAU IN request</td>
</tr>
<tr>
<td>DMA_REQUEST_CAU _OUT</td>
<td>DMAMUX CAU OUT request</td>
</tr>
<tr>
<td>DMA_REQUEST_USA RT0_RX</td>
<td>DMAMUX USART0 RX request</td>
</tr>
<tr>
<td>DMA_REQUEST_USA RT0_TX</td>
<td>DMAMUX USART0 TX request</td>
</tr>
<tr>
<td>DMA_REQUEST_USA RT1_RX</td>
<td>DMAMUX USART1 RX request</td>
</tr>
<tr>
<td>DMA_REQUEST_USA RT1_TX</td>
<td>DMAMUX USART1 TX request</td>
</tr>
<tr>
<td>DMA_REQUEST_UAR T3_RX</td>
<td>DMAMUX UART3 RX request</td>
</tr>
<tr>
<td>DMA_REQUEST_UAR T3_TX</td>
<td>DMAMUX UART3 TX request</td>
</tr>
</tbody>
</table>
### DMA_REQUEST_UAR

- **T4_RX**: DMAMUX UART4 RX request
- **T4_TX**: DMAMUX UART4 TX request
- **ART_RX**: DMAMUX LPUART RX request
- **ART_TX**: DMAMUX LPUART TX request

<table>
<thead>
<tr>
<th>Output parameter(out)</th>
<th>-</th>
</tr>
</thead>
</table>

| Return value | - |

#### Example:

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

### dmamux_trigger_polarity_config

The description of `dmamux_trigger_polarity_config` is shown as below:

**Table 3-234. Function dmamux_trigger_polarity_config**

<table>
<thead>
<tr>
<th>Function name</th>
<th>dmamux_trigger_polarity_config</th>
</tr>
</thead>
<tbody>
<tr>
<td>Function prototype</td>
<td>void dmamux_trigger_polarity_config(dmamux_generator_channel_enum channelx, uint32_t polarity);</td>
</tr>
<tr>
<td>Function descriptions</td>
<td>configure trigger input polarity</td>
</tr>
<tr>
<td>Precondition</td>
<td>-</td>
</tr>
<tr>
<td>The called functions</td>
<td>-</td>
</tr>
</tbody>
</table>

**Input parameter(in)**

<table>
<thead>
<tr>
<th>channelx</th>
<th>specify which DMAMUX request generator channel is initialized</th>
</tr>
</thead>
<tbody>
<tr>
<td>DMAMUX_GENCHx(x=0..3)</td>
<td>DMAMUX generation channel selection, refer to Table 3-192. Enum dmamux_generator_channel_enum</td>
</tr>
</tbody>
</table>

**Input parameter(in)**

<table>
<thead>
<tr>
<th>polarity</th>
<th>trigger input polarity</th>
</tr>
</thead>
<tbody>
<tr>
<td>DMAMUX_GEN_NO_EVENT</td>
<td>no event detection</td>
</tr>
<tr>
<td>DMAMUX_GEN_RISING</td>
<td>rising edge</td>
</tr>
<tr>
<td>DMAMUX_GEN_FALLING</td>
<td>falling edge</td>
</tr>
<tr>
<td>DMAMUX_GEN_RISING_FALLING</td>
<td>rising and falling edges</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Output parameter(out)</th>
<th>-</th>
</tr>
</thead>
</table>
dmamux_request_generate_number_config

The description of dmamux_request_generate_number_config is shown as below:

Table 3-235. Function dmamux_request_generate_number_config

<table>
<thead>
<tr>
<th>Function name</th>
<th>dmamux_request_generate_number_config</th>
</tr>
</thead>
<tbody>
<tr>
<td>Function prototype</td>
<td>void</td>
</tr>
<tr>
<td></td>
<td>dmamux_request_generate_number_config(dmamux_generator_channel_e num channelx, uint32_t number);</td>
</tr>
<tr>
<td>Function descriptions</td>
<td>configure number of DMA requests to be generated</td>
</tr>
<tr>
<td>Precondition</td>
<td>-</td>
</tr>
<tr>
<td>The called functions</td>
<td>-</td>
</tr>
<tr>
<td>Input parameter (in)</td>
<td></td>
</tr>
<tr>
<td>channelx</td>
<td>specify which DMAMUX request generator channel is initialized</td>
</tr>
<tr>
<td>DMAMUX_GENCHx(x=0..3)</td>
<td>DMAMUX generation channel selection, refer to Table 3-192. Enum dmamux_generator_channel_enum</td>
</tr>
<tr>
<td>number</td>
<td>DMA requests number to be generated (1 - 32)</td>
</tr>
<tr>
<td>Output parameter (out)</td>
<td></td>
</tr>
<tr>
<td>Return value</td>
<td></td>
</tr>
</tbody>
</table>

Example:

/* configure number of DMA requests to be generated */
dmamux_request_generate_number_config(DMAMUX_GENCH0, 1);

dmamux_trigger_id_config

The description of dmamux_trigger_id_config is shown as below:

Table 3-236. Function dmamux_trigger_id_config

<table>
<thead>
<tr>
<th>Function name</th>
<th>dmamux_trigger_id_config</th>
</tr>
</thead>
<tbody>
<tr>
<td>Function prototype</td>
<td>void dmamux_trigger_id_config(dmamux_generator_channel_enum channelx, uint32_t id);</td>
</tr>
<tr>
<td>Function descriptions</td>
<td>configure trigger input identification</td>
</tr>
<tr>
<td>Precondition</td>
<td>-</td>
</tr>
<tr>
<td>The called functions</td>
<td>-</td>
</tr>
</tbody>
</table>
### Input parameter \( \text{in} \)

<table>
<thead>
<tr>
<th>channelx</th>
<th>DMAMUX generation channel selection, refer to Table 3-192. Enum <code>dmamux_generator_channel_enum</code></th>
</tr>
</thead>
<tbody>
<tr>
<td>DMAMUX_GENCHx(x=0..3)</td>
<td>specify which DMAMUX request generator channel is initialized</td>
</tr>
</tbody>
</table>

#### `enum dmamux_generator_channel_enum`:

- `DMAMUX_TRIGGER EXTI0`  
  trigger input is EXTI0
- `DMAMUX_TRIGGER EXTI1`  
  trigger input is EXTI1
- `DMAMUX_TRIGGER EXTI2`  
  trigger input is EXTI2
- `DMAMUX_TRIGGER EXTI3`  
  trigger input is EXTI3
- `DMAMUX_TRIGGER EXTI4`  
  trigger input is EXTI4
- `DMAMUX_TRIGGER EXTI5`  
  trigger input is EXTI5
- `DMAMUX_TRIGGER EXTI6`  
  trigger input is EXTI6
- `DMAMUX_TRIGGER EXTI7`  
  trigger input is EXTI7
- `DMAMUX_TRIGGER EXTI8`  
  trigger input is EXTI8
- `DMAMUX_TRIGGER EXTI9`  
  trigger input is EXTI9
- `DMAMUX_TRIGGER EXTI10`  
  trigger input is EXTI10
- `DMAMUX_TRIGGER EXTI11`  
  trigger input is EXTI11
- `DMAMUX_TRIGGER EXTI12`  
  trigger input is EXTI12
- `DMAMUX_TRIGGER EXTI13`  
  trigger input is EXTI13
- `DMAMUX_TRIGGER EXTI14`  
  trigger input is EXTI14
- `DMAMUX_TRIGGER EXTI15`  
  trigger input is EXTI15
- `DMAMUX_TRIGGER EVT0_OUT`  
  trigger input is Evt0_out
- `DMAMUX_TRIGGER EVT1_OUT`  
  trigger input is Evt1_out
- `DMAMUX_TRIGGER EVT2_OUT`  
  trigger input is Evt2_out
EVT2_OUT
DMAMUX_TRIGGER_EVT3_OUT
DMAMUX_TRIGGER_TIMER11_CH0_O

<table>
<thead>
<tr>
<th>Output parameter(out)</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>trigger input is Evt3_out</td>
<td></td>
</tr>
<tr>
<td>trigger input is TIMER11_CH0_O</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Return value</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>-</td>
<td></td>
</tr>
</tbody>
</table>

Example:

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

dmamux_flag_get

The description of dmamux_flag_get is shown as below:

Table 3-237. Function dmamux_flag_get

<table>
<thead>
<tr>
<th>Function name</th>
<th>dmamux_flag_get</th>
</tr>
</thead>
<tbody>
<tr>
<td>Function prototype</td>
<td>FlagStatus dmamux_flag_get(dmamux_flag_enum flag);</td>
</tr>
<tr>
<td>Function descriptions</td>
<td>get DMAMUX flag</td>
</tr>
<tr>
<td>Precondition</td>
<td>-</td>
</tr>
<tr>
<td>The called functions</td>
<td>-</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Input parameter(in)</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>flag</td>
<td></td>
</tr>
<tr>
<td>DMAMUX_FLAG_MUX CH0_SO</td>
<td>DMAMUX request multiplexer channel 0 synchronization overrun flag</td>
</tr>
<tr>
<td>DMAMUX_FLAG_MUX CH1_SO</td>
<td>DMAMUX request multiplexer channel 1 synchronization overrun flag</td>
</tr>
<tr>
<td>DMAMUX_FLAG_MUX CH2_SO</td>
<td>DMAMUX request multiplexer channel 2 synchronization overrun flag</td>
</tr>
<tr>
<td>DMAMUX_FLAG_MUX CH3_SO</td>
<td>DMAMUX request multiplexer channel 3 synchronization overrun flag</td>
</tr>
<tr>
<td>DMAMUX_FLAG_MUX CH4_SO</td>
<td>DMAMUX request multiplexer channel 4 synchronization overrun flag</td>
</tr>
<tr>
<td>DMAMUX_FLAG_MUX CH5_SO</td>
<td>DMAMUX request multiplexer channel 5 synchronization overrun flag</td>
</tr>
<tr>
<td>DMAMUX_FLAG_MUX CH6_SO</td>
<td>DMAMUX request multiplexer channel 6 synchronization overrun flag</td>
</tr>
<tr>
<td>DMAMUX_FLAG_GEN CH0_TO</td>
<td>DMAMUX request generator channel 0 trigger overrun flag</td>
</tr>
<tr>
<td>DMAMUX_FLAG_GEN CH1_TO</td>
<td>DMAMUX request generator channel 1 trigger overrun flag</td>
</tr>
</tbody>
</table>
DMAMUX_FLAG_GEN
CH2_TO
DMAMUX request generator channel 2 trigger overrun flag

DMAMUX_FLAG_GEN
CH3_TO
DMAMUX request generator channel 3 trigger overrun flag

Output parameter\(\text{(out)}\)

- -

Return value

FlagStatus
SET or RESET

Example:

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

\textbf{dmamux\_flag\_clear}

The description of \texttt{dmamux\_flag\_clear} is shown as below:

\textbf{Table 3-238. Function \texttt{dmamux\_flag\_clear}}

<table>
<thead>
<tr>
<th>Function name</th>
<th>\texttt{dmamux_flag_clear}</th>
</tr>
</thead>
<tbody>
<tr>
<td>Function prototype</td>
<td>\texttt{void dmamux_flag_clear(dmamux_flag_enum flag);}</td>
</tr>
<tr>
<td>Function descriptions</td>
<td>clear DMAMUX flag</td>
</tr>
<tr>
<td>Preconditions</td>
<td>-</td>
</tr>
<tr>
<td>The called functions</td>
<td>-</td>
</tr>
</tbody>
</table>

\textbf{Input parameter\(\text{(in)}\)}

<table>
<thead>
<tr>
<th>flag</th>
<th>flag type, refer to \textit{Table 3-188. Enum dmamux_flag_enum}</th>
</tr>
</thead>
<tbody>
<tr>
<td>DMAMUX_FLAG_MUX CH0_SO</td>
<td>DMAMUX request multiplexer channel 0 synchronization overrun flag</td>
</tr>
<tr>
<td>DMAMUX_FLAG_MUX CH1_SO</td>
<td>DMAMUX request multiplexer channel 1 synchronization overrun flag</td>
</tr>
<tr>
<td>DMAMUX_FLAG_MUX CH2_SO</td>
<td>DMAMUX request multiplexer channel 2 synchronization overrun flag</td>
</tr>
<tr>
<td>DMAMUX_FLAG_MUX CH3_SO</td>
<td>DMAMUX request multiplexer channel 3 synchronization overrun flag</td>
</tr>
<tr>
<td>DMAMUX_FLAG_MUX CH4_SO</td>
<td>DMAMUX request multiplexer channel 4 synchronization overrun flag</td>
</tr>
<tr>
<td>DMAMUX_FLAG_MUX CH5_SO</td>
<td>DMAMUX request multiplexer channel 5 synchronization overrun flag</td>
</tr>
<tr>
<td>DMAMUX_FLAG_MUX CH6_SO</td>
<td>DMAMUX request multiplexer channel 6 synchronization overrun flag</td>
</tr>
<tr>
<td>DMAMUX_FLAG_GEN CH0_TO</td>
<td>DMAMUX request generator channel 0 trigger overrun flag</td>
</tr>
<tr>
<td>DMAMUX_FLAG_GEN CH1_TO</td>
<td>DMAMUX request generator channel 1 trigger overrun flag</td>
</tr>
</tbody>
</table>
**DMAMUX_FLAG_GEN CH2_TO**

DMAMUX request generator channel 2 trigger overrun flag

**DMAMUX_FLAG_GEN CH3_TO**

DMAMUX request generator channel 3 trigger overrun flag

**Output parameter(out)**

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>-</td>
<td>-</td>
</tr>
</tbody>
</table>

**Return value**

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>-</td>
<td>-</td>
</tr>
</tbody>
</table>

**Example:**

/* clear DMAMUX flag */
dmamux_flag_clear(DMAMUX_FLAG_GENCH0_TO);

dmamux_interrupt_flag_get

The description of dmamux_interrupt_flag_get is shown as below:

**Table 3-239. Function dmamux_interrupt_flag_get**

<table>
<thead>
<tr>
<th>Function name</th>
<th>dmamux_interrupt_flag_get</th>
</tr>
</thead>
<tbody>
<tr>
<td>Function prototype</td>
<td>FlagStatus dmamux_interrupt_flag_get(dmamux_interrupt_flag_enum int_flag);</td>
</tr>
<tr>
<td>Function descriptions</td>
<td>get DMAMUX interrupt flag</td>
</tr>
<tr>
<td>Precondition</td>
<td>-</td>
</tr>
<tr>
<td>The called functions</td>
<td>-</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Input parameter(in)</th>
<th>int_flag</th>
</tr>
</thead>
<tbody>
<tr>
<td>DMAMUX_INT_FLAG_MUXCH0_SO</td>
<td>DMAMUX request multiplexer channel 0 synchronization overrun interrupt flag</td>
</tr>
<tr>
<td>DMAMUX_INT_FLAG_MUXCH1_SO</td>
<td>DMAMUX request multiplexer channel 1 synchronization overrun interrupt flag</td>
</tr>
<tr>
<td>DMAMUX_INT_FLAG_MUXCH2_SO</td>
<td>DMAMUX request multiplexer channel 2 synchronization overrun interrupt flag</td>
</tr>
<tr>
<td>DMAMUX_INT_FLAG_MUXCH3_SO</td>
<td>DMAMUX request multiplexer channel 3 synchronization overrun interrupt flag</td>
</tr>
<tr>
<td>DMAMUX_INT_FLAG_MUXCH4_SO</td>
<td>DMAMUX request multiplexer channel 4 synchronization overrun interrupt flag</td>
</tr>
<tr>
<td>DMAMUX_INT_FLAG_MUXCH5_SO</td>
<td>DMAMUX request multiplexer channel 5 synchronization overrun interrupt flag</td>
</tr>
<tr>
<td>DMAMUX_INT_FLAG_MUXCH6_SO</td>
<td>DMAMUX request multiplexer channel 6 synchronization overrun interrupt flag</td>
</tr>
<tr>
<td>DMAMUX_INT_FLAG_GENCH0_TO</td>
<td>DMAMUX request generator channel 0 trigger overrun interrupt flag</td>
</tr>
<tr>
<td>DMAMUX_INT_FLAG_GENCH1_TO</td>
<td>DMAMUX request generator channel 1 trigger overrun interrupt flag</td>
</tr>
</tbody>
</table>
DMAMUX_INT_FLAG_GENCH2_TO  
DMAMUX request generator channel 2 trigger overrun interrupt flag

DMAMUX_INT_FLAG_GENCH3_TO  
DMAMUX request generator channel 3 trigger overrun interrupt flag

Output parameter(out)
-  -

Return value
FlagStatus  SET or RESET

Example:

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

dmamux_interrupt_flag_clear

The description of dmamux_interrupt_flag_clear is shown as below:

**Table 3-240. Function dmamux_interrupt_flag_clear**

<table>
<thead>
<tr>
<th>Function name</th>
<th>dmamux_interrupt_flag_clear</th>
</tr>
</thead>
<tbody>
<tr>
<td>Function prototype</td>
<td>FlagStatus dmamux_interrupt_flag_get(dmamux_interrupt_flag_enum int_flag);</td>
</tr>
<tr>
<td>Function descriptions</td>
<td>clear DMAMUX interrupt flag</td>
</tr>
<tr>
<td>Precondition</td>
<td>-</td>
</tr>
<tr>
<td>The called functions</td>
<td>-</td>
</tr>
</tbody>
</table>

**Input parameter(in)**

<table>
<thead>
<tr>
<th>int_flag</th>
<th>flag type, refer to <strong>Table 3-189. Enum dmamux_interrupt_flag_enum</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td>DMAMUX_INT_FLAG_MUXCH0_SO</td>
<td>DMAMUX request multiplexer channel 0 synchronization overrun interrupt flag</td>
</tr>
<tr>
<td>DMAMUX_INT_FLAG_MUXCH1_SO</td>
<td>DMAMUX request multiplexer channel 1 synchronization overrun interrupt flag</td>
</tr>
<tr>
<td>DMAMUX_INT_FLAG_MUXCH2_SO</td>
<td>DMAMUX request multiplexer channel 2 synchronization overrun interrupt flag</td>
</tr>
<tr>
<td>DMAMUX_INT_FLAG_MUXCH3_SO</td>
<td>DMAMUX request multiplexer channel 3 synchronization overrun interrupt flag</td>
</tr>
<tr>
<td>DMAMUX_INT_FLAG_MUXCH4_SO</td>
<td>DMAMUX request multiplexer channel 4 synchronization overrun interrupt flag</td>
</tr>
<tr>
<td>DMAMUX_INT_FLAG_MUXCH5_SO</td>
<td>DMAMUX request multiplexer channel 5 synchronization overrun interrupt flag</td>
</tr>
<tr>
<td>DMAMUX_INT_FLAG_MUXCH6_SO</td>
<td>DMAMUX request multiplexer channel 6 synchronization overrun interrupt flag</td>
</tr>
<tr>
<td>DMAMUX_INT_FLAG_GENCH0_TO</td>
<td>DMAMUX request generator channel 0 trigger overrun interrupt flag</td>
</tr>
</tbody>
</table>
DMAMUX_INT_FLAG_GENCH1_TO

DMAMUX request generator channel 1 trigger overrun interrupt flag

DMAMUX_INT_FLAG_GENCH2_TO

DMAMUX request generator channel 2 trigger overrun interrupt flag

DMAMUX_INT_FLAG_GENCH3_TO

DMAMUX request generator channel 3 trigger overrun interrupt flag

Output parameter(out)

- -

Return value

- -

Example:

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

dmamux_interrupt_enable

The description of dmamux_interrupt_enable is shown as below:

Table 3-241. Function dmamux_interrupt_enable

<table>
<thead>
<tr>
<th>Function name</th>
<th>dmamux_interrupt_enable</th>
</tr>
</thead>
<tbody>
<tr>
<td>Function prototype</td>
<td>void dmamux_interrupt_enable(dmamux_interrupt_enum interrupt);</td>
</tr>
<tr>
<td>Function descriptions</td>
<td>enable DMAMUX interrupt</td>
</tr>
<tr>
<td>Precondition</td>
<td>-</td>
</tr>
<tr>
<td>The called functions</td>
<td>-</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>interrupt</th>
<th>specify which interrupt to enable</th>
</tr>
</thead>
<tbody>
<tr>
<td>DMAMUX_INT_MUXC_H0_SO</td>
<td>DMAMUX request multiplexer channel 0 synchronization overrun interrupt</td>
</tr>
<tr>
<td>DMAMUX_INT_MUXC_H1_SO</td>
<td>DMAMUX request multiplexer channel 1 synchronization overrun interrupt</td>
</tr>
<tr>
<td>DMAMUX_INT_MUXC_H2_SO</td>
<td>DMAMUX request multiplexer channel 2 synchronization overrun interrupt</td>
</tr>
<tr>
<td>DMAMUX_INT_MUXC_H3_SO</td>
<td>DMAMUX request multiplexer channel 3 synchronization overrun interrupt</td>
</tr>
<tr>
<td>DMAMUX_INT_MUXC_H4_SO</td>
<td>DMAMUX request multiplexer channel 4 synchronization overrun interrupt</td>
</tr>
<tr>
<td>DMAMUX_INT_MUXC_H5_SO</td>
<td>DMAMUX request multiplexer channel 5 synchronization overrun interrupt</td>
</tr>
<tr>
<td>DMAMUX_INT_MUXC_H6_SO</td>
<td>DMAMUX request multiplexer channel 6 synchronization overrun interrupt</td>
</tr>
<tr>
<td>DMAMUX_INT_GENC</td>
<td>DMAMUX request generator channel 0 trigger overrun interrupt</td>
</tr>
</tbody>
</table>
Example:

```c
/* enable DMAMUX interrupt */
dmamux_interrupt_enable(DMAMUX_INT_MUXCH0_SO);
```

**dmamux_interrupt_disable**

The description of dmamux_interrupt_disable is shown as below:

<table>
<thead>
<tr>
<th>Function name</th>
<th>dmamux_interrupt_disable</th>
</tr>
</thead>
<tbody>
<tr>
<td>Function prototype</td>
<td>void dmamux_interrupt_disable(dmamux_interrupt_enum interrupt);</td>
</tr>
<tr>
<td>Function descriptions</td>
<td>disable DMAMUX interrupt</td>
</tr>
<tr>
<td>Precondition</td>
<td>-</td>
</tr>
<tr>
<td>The called functions</td>
<td>-</td>
</tr>
<tr>
<td>Input parameter(in)</td>
<td>specify which interrupt to disable</td>
</tr>
<tr>
<td><strong>DMAMUX_INT_MUXC H0_SO</strong></td>
<td>DMAMUX request multiplexer channel 0 synchronization overrun interrupt</td>
</tr>
<tr>
<td><strong>DMAMUX_INT_MUXC H1_SO</strong></td>
<td>DMAMUX request multiplexer channel 1 synchronization overrun interrupt</td>
</tr>
<tr>
<td><strong>DMAMUX_INT_MUXC H2_SO</strong></td>
<td>DMAMUX request multiplexer channel 2 synchronization overrun interrupt</td>
</tr>
<tr>
<td><strong>DMAMUX_INT_MUXC H3_SO</strong></td>
<td>DMAMUX request multiplexer channel 3 synchronization overrun interrupt</td>
</tr>
<tr>
<td><strong>DMAMUX_INT_MUXC H4_SO</strong></td>
<td>DMAMUX request multiplexer channel 4 synchronization overrun interrupt</td>
</tr>
<tr>
<td><strong>DMAMUX_INT_MUXC H5_SO</strong></td>
<td>DMAMUX request multiplexer channel 5 synchronization overrun interrupt</td>
</tr>
<tr>
<td><strong>DMAMUX_INT_MUXC H6_SO</strong></td>
<td>DMAMUX request multiplexer channel 6 synchronization overrun interrupt</td>
</tr>
<tr>
<td><strong>DMAMUX_INT_GENC H0_TO</strong></td>
<td>DMAMUX request generator channel 0 trigger overrun interrupt</td>
</tr>
</tbody>
</table>
DMAMUX_INT_GENC
H1_TO

DMAMUX request generator channel 1 trigger overrun interrupt

DMAMUX_INT_GENC
H2_TO

DMAMUX request generator channel 2 trigger overrun interrupt

DMAMUX_INT_GENC
H3_TO

DMAMUX request generator channel 3 trigger overrun interrupt

Output parameter(out)

- -

Return value

- -

Example:

/* disable DMAMUX interrupt */
dmamux_interrupt_disable(DMAMUX_INT_MUXCH0_SO);

3.10. **EXTI**

EXTI is the interrupt/event controller in the MCU. It contains up to 30 independent edge detectors and generates interrupt requests or events to the processor. The EXTI registers are listed in chapter **3.10.1**, the EXTI firmware functions are introduced in chapter **3.10.2**.

3.10.1. **Descriptions of Peripheral registers**

EXTI registers are listed in the table shown as below:

<table>
<thead>
<tr>
<th>Registers</th>
<th>Descriptions</th>
</tr>
</thead>
<tbody>
<tr>
<td>EXTI_INTEN</td>
<td>Interrupt enable register</td>
</tr>
<tr>
<td>EXTI_EVEN</td>
<td>Event enable register</td>
</tr>
<tr>
<td>EXTI_RTEN</td>
<td>Rising edge trigger enable register</td>
</tr>
<tr>
<td>EXTI_FTEN</td>
<td>Falling edge trigger enable register</td>
</tr>
<tr>
<td>EXTI_SWIEV</td>
<td>Software interrupt event register</td>
</tr>
<tr>
<td>EXTI_PD</td>
<td>Pending register</td>
</tr>
</tbody>
</table>

3.10.2. **Descriptions of Peripheral functions**

EXTI firmware functions are listed in the table shown as below:

<table>
<thead>
<tr>
<th>Function name</th>
<th>Function description</th>
</tr>
</thead>
<tbody>
<tr>
<td>exti_deinit</td>
<td>deinitialize the EXTI, reset the value of all EXTI registers with initial values</td>
</tr>
<tr>
<td>exti_init</td>
<td>initialize the EXTI, enable the configuration of EXTI initialize</td>
</tr>
<tr>
<td>Function name</td>
<td>Function description</td>
</tr>
<tr>
<td>-------------------------------------</td>
<td>--------------------------------------------</td>
</tr>
<tr>
<td>exti_interrupt_enable</td>
<td>enable the interrupts from EXTI line x</td>
</tr>
<tr>
<td>exti_interrupt_disable</td>
<td>disable the interrupts from EXTI line x</td>
</tr>
<tr>
<td>exti_event_enable</td>
<td>enable the events from EXTI line x</td>
</tr>
<tr>
<td>exti_event_disable</td>
<td>disable the events from EXTI line x</td>
</tr>
<tr>
<td>exti_software_interrupt_enable</td>
<td>enable EXTI software interrupt event</td>
</tr>
<tr>
<td>exti_software_interrupt_disable</td>
<td>disable EXTI software interrupt event</td>
</tr>
<tr>
<td>exti_flag_get</td>
<td>get EXTI lines pending flag</td>
</tr>
<tr>
<td>exti_flag_clear</td>
<td>clear EXTI lines pending flag</td>
</tr>
<tr>
<td>exti_interrupt_flag_get</td>
<td>get EXTI lines flag when the interrupt flag is set</td>
</tr>
<tr>
<td>exti_interrupt_flag_clear</td>
<td>clear EXTI lines pending flag</td>
</tr>
</tbody>
</table>

**Enum exti_line_enum**

<table>
<thead>
<tr>
<th>enum name</th>
<th>Function description</th>
</tr>
</thead>
<tbody>
<tr>
<td>EXTI_0</td>
<td>EXTI line 0</td>
</tr>
<tr>
<td>EXTI_1</td>
<td>EXTI line 1</td>
</tr>
<tr>
<td>EXTI_2</td>
<td>EXTI line 2</td>
</tr>
<tr>
<td>EXTI_3</td>
<td>EXTI line 3</td>
</tr>
<tr>
<td>EXTI_4</td>
<td>EXTI line 4</td>
</tr>
<tr>
<td>EXTI_5</td>
<td>EXTI line 5</td>
</tr>
<tr>
<td>EXTI_6</td>
<td>EXTI line 6</td>
</tr>
<tr>
<td>EXTI_7</td>
<td>EXTI line 7</td>
</tr>
<tr>
<td>EXTI_8</td>
<td>EXTI line 8</td>
</tr>
<tr>
<td>EXTI_9</td>
<td>EXTI line 9</td>
</tr>
<tr>
<td>EXTI_10</td>
<td>EXTI line 10</td>
</tr>
<tr>
<td>EXTI_11</td>
<td>EXTI line 11</td>
</tr>
<tr>
<td>EXTI_12</td>
<td>EXTI line 12</td>
</tr>
<tr>
<td>EXTI_13</td>
<td>EXTI line 13</td>
</tr>
<tr>
<td>EXTI_14</td>
<td>EXTI line 14</td>
</tr>
<tr>
<td>EXTI_15</td>
<td>EXTI line 15</td>
</tr>
<tr>
<td>EXTI_16</td>
<td>EXTI line 16</td>
</tr>
<tr>
<td>EXTI_17</td>
<td>EXTI line 17</td>
</tr>
<tr>
<td>EXTI_19</td>
<td>EXTI line 19</td>
</tr>
<tr>
<td>EXTI_25</td>
<td>EXTI line 25</td>
</tr>
<tr>
<td>EXTI_26</td>
<td>EXTI line 26</td>
</tr>
<tr>
<td>EXTI_27</td>
<td>EXTI line 27</td>
</tr>
<tr>
<td>EXTI_28</td>
<td>EXTI line 28</td>
</tr>
<tr>
<td>EXTI_29</td>
<td>EXTI line 29</td>
</tr>
</tbody>
</table>
Enum exti_mode_enum

Table 3-246. exti_mode_enum

<table>
<thead>
<tr>
<th>enum name</th>
<th>Function description</th>
</tr>
</thead>
<tbody>
<tr>
<td>EXTI_INTERRUPT</td>
<td>EXTI interrupt mode</td>
</tr>
<tr>
<td>EXTI_EVENT</td>
<td>EXTI event mode</td>
</tr>
</tbody>
</table>

Enum exti_trig_type_enum

Table 3-247. exti_trig_type_enum

<table>
<thead>
<tr>
<th>enum name</th>
<th>Function description</th>
</tr>
</thead>
<tbody>
<tr>
<td>EXTI_TRIG_RISING</td>
<td>EXTI rising edge trigger</td>
</tr>
<tr>
<td>EXTI_TRIG_FALLING</td>
<td>EXTI falling edge trigger</td>
</tr>
<tr>
<td>EXTI_TRIG_BOTH</td>
<td>EXTI rising and falling edge trigger</td>
</tr>
<tr>
<td>EXTI_TRIG_NONE</td>
<td>EXTI without rising edge or falling edge trigger</td>
</tr>
</tbody>
</table>

exti_deinit

The description of exti_deinit is shown as below:

Table 3-248. Function exti_deinit

<table>
<thead>
<tr>
<th>Function name</th>
<th>exti_deinit</th>
</tr>
</thead>
<tbody>
<tr>
<td>Function prototype</td>
<td>void exti_deinit(void);</td>
</tr>
<tr>
<td>Function descriptions</td>
<td>deinitialize the EXTI, reset the value of all EXTI registers with initial values</td>
</tr>
<tr>
<td>Precondition</td>
<td>-</td>
</tr>
<tr>
<td>The called functions</td>
<td>-</td>
</tr>
<tr>
<td>Input parameter(in)</td>
<td>-</td>
</tr>
<tr>
<td>Output parameter(out)</td>
<td>-</td>
</tr>
<tr>
<td>Return value</td>
<td>-</td>
</tr>
</tbody>
</table>

Example:

/* deinitialize the EXTI */

exti_deinit();

exti_init

The description of exti_init is shown as below:

Table 3-249. Function exti_init

<table>
<thead>
<tr>
<th>Function name</th>
<th>exti_init</th>
</tr>
</thead>
<tbody>
<tr>
<td>Function prototype</td>
<td>void exti_init(exti_line_enum linex, exti_mode_enum mode, exti_trig_type_enum trig_type);</td>
</tr>
</tbody>
</table>
Function descriptions | initialize the EXTI, enable the configuration of EXTI initialize
---|---
Precondition | -
The called functions | -

### Input parameter(in)

<table>
<thead>
<tr>
<th>linex</th>
<th>EXTI line x, refer to Table 3-245, exti_line_enum</th>
</tr>
</thead>
<tbody>
<tr>
<td>EXTI_x</td>
<td>x=0,1,2,...29</td>
</tr>
</tbody>
</table>

### Input parameter(in)

<table>
<thead>
<tr>
<th>mode</th>
<th>EXTI mode, refer to Table 3-246, exti_mode_enum</th>
</tr>
</thead>
<tbody>
<tr>
<td>EXTI_INTERRUPT</td>
<td>interrupt mode</td>
</tr>
<tr>
<td>EXTI_EVENT</td>
<td>event mode</td>
</tr>
</tbody>
</table>

### Input parameter(in)

<table>
<thead>
<tr>
<th>trig_type</th>
<th>trigger type, refer to Table 3-247, exti_trig_type_enum</th>
</tr>
</thead>
<tbody>
<tr>
<td>EXTI_TRIG_RISING</td>
<td>rising edge trigger</td>
</tr>
<tr>
<td>EXTI_TRIG_FALLING</td>
<td>falling edge trigger</td>
</tr>
<tr>
<td>EXTI_TRIG_BOTH</td>
<td>rising edge and falling edge trigger</td>
</tr>
<tr>
<td>EXTI_TRIG_NONE</td>
<td>without rising edge or falling edge trigger</td>
</tr>
</tbody>
</table>

### Output parameter(out)

- -

### Return value

- -

Example:

```c
/* configure EXTI_0 */
exti_init(EXTI_0, EXTI_INTERRUPT, EXTI_TRIG_BOTH);
```

### exti_interrupt_enable

The description of exti_interrupt_enable is shown as below:

#### Table 3-250. Function exti_interrupt_enable

<table>
<thead>
<tr>
<th>Function name</th>
<th>exti_interrupt_enable</th>
</tr>
</thead>
<tbody>
<tr>
<td>Function prototype</td>
<td>void exti_interrupt_enable(exti_line_enum linex);</td>
</tr>
<tr>
<td>Function descriptions</td>
<td>enable the interrupts from EXTI line x</td>
</tr>
<tr>
<td>Precondition</td>
<td>-</td>
</tr>
<tr>
<td>The called functions</td>
<td>-</td>
</tr>
</tbody>
</table>

### Input parameter(in)

<table>
<thead>
<tr>
<th>linex</th>
<th>EXTI line x, refer to Table 3-245, exti_line_enum</th>
</tr>
</thead>
<tbody>
<tr>
<td>EXTI_x</td>
<td>x=0,1,2,...29</td>
</tr>
</tbody>
</table>

### Output parameter(out)

- -

### Return value

- -
Example:

/* enable the interrupts from EXTI line 0 */

exti_interrupt_enable(EXTI_0);

**exti_interrupt_disable**

The description of exti_interrupt_disable is shown as below:

<table>
<thead>
<tr>
<th>Table 3-251. Function exti_interrupt_disable</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Function name</strong></td>
</tr>
<tr>
<td><strong>Function prototype</strong></td>
</tr>
<tr>
<td><strong>Function descriptions</strong></td>
</tr>
<tr>
<td><strong>Precondition</strong></td>
</tr>
<tr>
<td><strong>The called functions</strong></td>
</tr>
<tr>
<td><strong>Input parameter(in)</strong></td>
</tr>
<tr>
<td>linex</td>
</tr>
<tr>
<td>EXTI_x</td>
</tr>
<tr>
<td><strong>Output parameter(out)</strong></td>
</tr>
<tr>
<td>-</td>
</tr>
<tr>
<td><strong>Return value</strong></td>
</tr>
<tr>
<td>-</td>
</tr>
</tbody>
</table>

Example:

/* disable the interrupts from EXTI line 0 */

exti_interrupt_disable(EXTI_0);

**exti_event_enable**

The description of exti_event_enable is shown as below:

<table>
<thead>
<tr>
<th>Table 3-252. Function exti_event_enable</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Function name</strong></td>
</tr>
<tr>
<td><strong>Function prototype</strong></td>
</tr>
<tr>
<td><strong>Function descriptions</strong></td>
</tr>
<tr>
<td><strong>Precondition</strong></td>
</tr>
<tr>
<td><strong>The called functions</strong></td>
</tr>
<tr>
<td><strong>Input parameter(in)</strong></td>
</tr>
<tr>
<td>linex</td>
</tr>
<tr>
<td>EXTI_x</td>
</tr>
<tr>
<td><strong>Output parameter(out)</strong></td>
</tr>
<tr>
<td>-</td>
</tr>
<tr>
<td><strong>Return value</strong></td>
</tr>
<tr>
<td>-</td>
</tr>
</tbody>
</table>
Example:

```c
/* enable the events from EXTI line 0 */
exti_event_enable(EXTI_0);
```

**exti_event_disable**

The description of exti_event_disable is shown as below:

<table>
<thead>
<tr>
<th>Function name</th>
<th>exti_event_disable</th>
</tr>
</thead>
<tbody>
<tr>
<td>Function prototype</td>
<td>void exti_event_disable(exti_line_enum linex);</td>
</tr>
<tr>
<td>Function descriptions</td>
<td>disable the events from EXTI line x</td>
</tr>
<tr>
<td>Precondition</td>
<td>-</td>
</tr>
<tr>
<td>The called functions</td>
<td>-</td>
</tr>
</tbody>
</table>

**Input parameter(in)**

<table>
<thead>
<tr>
<th>linex</th>
<th>EXTI line x, refer to Table 3-245, exti_line_enum</th>
</tr>
</thead>
<tbody>
<tr>
<td>EXTI_x</td>
<td>x=0,1,2..29</td>
</tr>
</tbody>
</table>

**Output parameter(out)**

- -

**Return value**

- -

Example:

```c
/* disable the events from EXTI line 0 */
exti_event_disable(EXTI_0);
```

**exti_software_interrupt_enable**

The description of exti_software_interrupt_enable is shown as below:

<table>
<thead>
<tr>
<th>Function name</th>
<th>exti_software_interrupt_enable</th>
</tr>
</thead>
<tbody>
<tr>
<td>Function prototype</td>
<td>void exti_software_interrupt_enable(exti_line_enum linex);</td>
</tr>
<tr>
<td>Function descriptions</td>
<td>enable EXTI software interrupt event</td>
</tr>
<tr>
<td>Precondition</td>
<td>-</td>
</tr>
<tr>
<td>The called functions</td>
<td>-</td>
</tr>
</tbody>
</table>

**Input parameter(in)**

<table>
<thead>
<tr>
<th>linex</th>
<th>EXTI line x, refer to Table 3-245, exti_line_enum</th>
</tr>
</thead>
<tbody>
<tr>
<td>EXTI_x</td>
<td>x=0,1,2..29</td>
</tr>
</tbody>
</table>

**Output parameter(out)**

- -

**Return value**

- -
Example:

/* enable EXTI line 0 software interrupt */
exti_software_interrupt_enable(EXTI_0);

**exti_software_interrupt_disable**

The description of **exti_software_interrupt_disable** is shown as below:

Table 3-255. Function **exti_software_interrupt_disable**

<table>
<thead>
<tr>
<th>Function name</th>
<th>exti_software_interrupt_disable</th>
</tr>
</thead>
<tbody>
<tr>
<td>Function prototype</td>
<td>void exti_software_interrupt_disable(exti_line_enum linex);</td>
</tr>
<tr>
<td>Function descriptions</td>
<td>disable EXTI software interrupt event</td>
</tr>
<tr>
<td>Precondition</td>
<td>-</td>
</tr>
<tr>
<td>The called functions</td>
<td>-</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Input parameter(in)</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>linex</td>
<td>EXTI line x, refer to <strong>Table 3-245, exti_line_enum</strong></td>
</tr>
<tr>
<td>EXTI_x</td>
<td>x=0,1,2..29</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Output parameter(out)</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>-</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Return value</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>-</td>
<td></td>
</tr>
</tbody>
</table>

Example:

/* disable EXTI line 0 software interrupt */
exti_software_interrupt_disable(EXTI_0);

**exti_flag_get**

The description of **exti_flag_get** is shown as below:

Table 3-256. Function **exti_flag_get**

<table>
<thead>
<tr>
<th>Function name</th>
<th>exti_flag_get</th>
</tr>
</thead>
<tbody>
<tr>
<td>Function prototype</td>
<td>FlagStatus exti_flag_get(exti_line_enum linex);</td>
</tr>
<tr>
<td>Function descriptions</td>
<td>get EXTI lines pending flag</td>
</tr>
<tr>
<td>Precondition</td>
<td>-</td>
</tr>
<tr>
<td>The called functions</td>
<td>-</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Input parameter(in)</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>linex</td>
<td>EXTI line x, refer to <strong>Table 3-245, exti_line_enum</strong></td>
</tr>
<tr>
<td>EXTI_x</td>
<td>x=0,1,2..29</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Output parameter(out)</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>-</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Return value</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>FlagStatus</td>
<td>SET or RESET</td>
</tr>
</tbody>
</table>

Example:

```c
/* get EXTI line 0 flag status */
FlagStatus state = exti_flag_get(EXTI_0);
```

**exti_flag_clear**

The description of exti_flag_clear is shown as below:

**Table 3-257. Function exti_flag_clear**

<table>
<thead>
<tr>
<th>Function name</th>
<th>exti_flag_clear</th>
</tr>
</thead>
<tbody>
<tr>
<td>Function prototype</td>
<td>void exti_flag_clear(exti_line_enum linex);</td>
</tr>
<tr>
<td>Function description</td>
<td>clear EXTI lines pending flag</td>
</tr>
<tr>
<td>Precondition</td>
<td>-</td>
</tr>
<tr>
<td>The called functions</td>
<td>-</td>
</tr>
</tbody>
</table>

| Input parameter(in)   | linex EXTI line x, refer to Table 3-245, exti_line_enum |
|                       | EXTI_x x=0,1,2..29 |

| Output parameter(out) | - |
| Return value          | - |

Example:

```c
/* clear EXTI line 0 flag status */
exti_flag_clear(EXTI_0);
```

**exti_interrupt_flag_get**

The description of exti_interrupt_flag_get is shown as below:

**Table 3-258. Function exti_interrupt_flag_get**

<table>
<thead>
<tr>
<th>Function name</th>
<th>exti_interrupt_flag_get</th>
</tr>
</thead>
<tbody>
<tr>
<td>Function prototype</td>
<td>FlagStatus exti_interrupt_flag_get(exti_line_enum linex);</td>
</tr>
<tr>
<td>Function description</td>
<td>get EXTI lines flag when the interrupt flag is set</td>
</tr>
<tr>
<td>Precondition</td>
<td>-</td>
</tr>
<tr>
<td>The called functions</td>
<td>-</td>
</tr>
</tbody>
</table>

| Input parameter(in)   | linex EXTI line x, refer to Table 3-245, exti_line_enum |
|                       | EXTI_x x=0,1,2..29 |

| Output parameter(out) | - |
| Return value          | FlagStatus SET or RESET |
Example:

```c
/* get EXTI line 0 interrupt flag status */
FlagStatus state = exti_interrupt_flag_get(EXTI_0);
```

**exti_interrupt_flag_clear**

The description of exti_interrupt_flag_clear is shown as below:

**Table 3-259. Function exti_interrupt_flag_clear**

<table>
<thead>
<tr>
<th>Function name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>exti_interrupt_flag_clear</td>
<td>-</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Function prototype</th>
<th>void exti_interrupt_flag_clear(exti_line_enum linex);</th>
</tr>
</thead>
<tbody>
<tr>
<td>Function description</td>
<td>clear EXTI lines pending flag</td>
</tr>
<tr>
<td>Precondition</td>
<td>-</td>
</tr>
<tr>
<td>The called functions</td>
<td>-</td>
</tr>
</tbody>
</table>
| Input parameter (in) | linex:
EXTI line x, refer to Table 3-245, exti_line_enum |
| Output parameter (out) | - |
| Return value | - |

Example:

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

### 3.11. FMC

There is flash controller and option byte for GD32L23x series. The FMC registers are listed in chapter 3.11.1 the FMC firmware functions are introduced in chapter 3.11.2.

#### 3.11.1. Descriptions of Peripheral registers

FMC registers are listed in the table shown as below:

**Table 3-260. FMC Registers**

<table>
<thead>
<tr>
<th>Registers</th>
<th>Descriptions</th>
</tr>
</thead>
<tbody>
<tr>
<td>FMC_WS</td>
<td>FMC wait state register</td>
</tr>
<tr>
<td>FMC_KEY</td>
<td>FMC unlock key register</td>
</tr>
<tr>
<td>FMC_OBKEY</td>
<td>FMC option bytes unlock key register</td>
</tr>
<tr>
<td>FMC_STAT</td>
<td>FMC status register</td>
</tr>
<tr>
<td>FMC_CTL</td>
<td>FMC control register</td>
</tr>
</tbody>
</table>
### Registers

<table>
<thead>
<tr>
<th>Registers</th>
<th>Descriptions</th>
</tr>
</thead>
<tbody>
<tr>
<td>FMC_ADDR</td>
<td>FMC address register</td>
</tr>
<tr>
<td>FMC_OBSTAT</td>
<td>FMC option bytes status register</td>
</tr>
<tr>
<td>FMC_WP</td>
<td>FMC erase/write protection register</td>
</tr>
<tr>
<td>FMC_SLPKEY</td>
<td>FMC flash sleep or power-down mode unlock key register</td>
</tr>
<tr>
<td>FMC_PID</td>
<td>FMC product ID register</td>
</tr>
</tbody>
</table>

### 3.11.2. Descriptions of Peripheral functions

FMC firmware functions are listed in the table shown as below:

#### Table 3-261. FMC firmware function

<table>
<thead>
<tr>
<th>Function name</th>
<th>Function description</th>
</tr>
</thead>
<tbody>
<tr>
<td>fmc_unlock</td>
<td>unlock the main FMC operation</td>
</tr>
<tr>
<td>fmc_lock</td>
<td>lock the main FMC operation</td>
</tr>
<tr>
<td>fmc_slp_unlock</td>
<td>unlock operation of RUN_SLP bit in FMC_WS register</td>
</tr>
<tr>
<td>fmc_ws cnt_set</td>
<td>set the wait state</td>
</tr>
<tr>
<td>fmc_prefetch_enable</td>
<td>enable pre-fetch</td>
</tr>
<tr>
<td>fmc_prefetch_disable</td>
<td>disable pre-fetch</td>
</tr>
<tr>
<td>fmc_low_power_enable</td>
<td>enable low power</td>
</tr>
<tr>
<td>fmc_low_power_disable</td>
<td>disable low power</td>
</tr>
<tr>
<td>fmc_enter_slp_enable</td>
<td>flash enter sleep/power-down mode during MCU run/low-power run mode</td>
</tr>
<tr>
<td>fmc_enter_slp_disable</td>
<td>flash enter IDLE mode during MCU run/low-power run mode</td>
</tr>
<tr>
<td>fmc_enter_sleep_enable</td>
<td>flash enter sleep mode when MCU enter deep-sleep mode or RUN_SLP bit is set</td>
</tr>
<tr>
<td>fmc_enter_sleep_disable</td>
<td>flash enter power-down mode when MCU enter deep-sleep mode or RUN_SLP bit is set</td>
</tr>
<tr>
<td>fmc_page_erase</td>
<td>erase FMC page</td>
</tr>
<tr>
<td>fmc_mass_erase</td>
<td>erase FMC whole chip</td>
</tr>
<tr>
<td>fmc_word_program</td>
<td>FMC program a word at the corresponding address</td>
</tr>
<tr>
<td>fmc_fast_program</td>
<td>FMC fast program a row words (32 double words) at the corresponding address</td>
</tr>
<tr>
<td>ob_unlock</td>
<td>unlock the option byte operation</td>
</tr>
<tr>
<td>ob_lock</td>
<td>lock the option byte operation</td>
</tr>
<tr>
<td>ob_erase</td>
<td>erase the option byte</td>
</tr>
<tr>
<td>ob_write_protection_enable</td>
<td>enable write protection</td>
</tr>
<tr>
<td>ob_security_protection_config</td>
<td>configure the option bytes security protection</td>
</tr>
<tr>
<td>ob_user_write</td>
<td>program option bytes USER</td>
</tr>
<tr>
<td>ob_data_program</td>
<td>program option bytes DATA</td>
</tr>
<tr>
<td>ob_user_get</td>
<td>get the value of option bytes USER</td>
</tr>
<tr>
<td>ob_data_get</td>
<td>get the value of option bytes DATA</td>
</tr>
<tr>
<td>ob_write_protection_get</td>
<td>get the value of option bytes write protection</td>
</tr>
<tr>
<td>Function name</td>
<td>Function description</td>
</tr>
<tr>
<td>---------------------------------------------------</td>
<td>--------------------------------------------</td>
</tr>
<tr>
<td>ob_security_protection_flag_get</td>
<td>get option bytes security protection state</td>
</tr>
<tr>
<td>fmc_flag_get</td>
<td>get FMC flag status</td>
</tr>
<tr>
<td>fmc_flag_clear</td>
<td>clear the FMC flag</td>
</tr>
<tr>
<td>fmc_interrupt_enable</td>
<td>enable FMC interrupt</td>
</tr>
<tr>
<td>fmc_interrupt_disable</td>
<td>disable FMC interrupt</td>
</tr>
<tr>
<td>fmc_interrupt_flag_get</td>
<td>get FMC interrupt flag</td>
</tr>
<tr>
<td>fmc_interrupt_flag_clear</td>
<td>clear FMC interrupt flag</td>
</tr>
</tbody>
</table>

**fmc_state_enum**

Table 3-262. fmc_state_enum

<table>
<thead>
<tr>
<th>enum name</th>
<th>enum description</th>
</tr>
</thead>
<tbody>
<tr>
<td>FMC_READY</td>
<td>the operation has been completed</td>
</tr>
<tr>
<td>FMC_BUSY</td>
<td>the operation is in progress</td>
</tr>
<tr>
<td>FMC_PGERR</td>
<td>program error</td>
</tr>
<tr>
<td>FMC_PGAERR</td>
<td>program alignment error</td>
</tr>
<tr>
<td>FMC_WPERR</td>
<td>erase/program protection error</td>
</tr>
<tr>
<td>FMC_TOERR</td>
<td>timeout error</td>
</tr>
<tr>
<td>FMC_OB_HSPC</td>
<td>high security protection</td>
</tr>
<tr>
<td>FMC_SLP</td>
<td>sleep or power-down status</td>
</tr>
</tbody>
</table>

**fmc_unlock**

The description of fmc_unlock is shown as below:

Table 3-263. Function fmc_unlock

<table>
<thead>
<tr>
<th>Function name</th>
<th>fmc_unlock</th>
</tr>
</thead>
<tbody>
<tr>
<td>Function prototype</td>
<td>void fmc_unlock(void);</td>
</tr>
<tr>
<td>Function descriptions</td>
<td>unlock the main FMC operation</td>
</tr>
<tr>
<td>Precondition</td>
<td>-</td>
</tr>
<tr>
<td>The called functions</td>
<td>-</td>
</tr>
<tr>
<td>Input parameter(in)</td>
<td>-</td>
</tr>
<tr>
<td>Output parameter(out)</td>
<td>-</td>
</tr>
<tr>
<td>Return value</td>
<td>-</td>
</tr>
</tbody>
</table>

Example:

/* unlock the main FMC operation */

fmc_unlock();
fmc_lock

The description of fmc_lock is shown as below:

Table 3-264. Function fmc_lock

<table>
<thead>
<tr>
<th>Function name</th>
<th>Function prototype</th>
<th>Function descriptions</th>
<th>Precondition</th>
<th>The called functions</th>
</tr>
</thead>
<tbody>
<tr>
<td>fmc_lock</td>
<td>void fmc_lock(void);</td>
<td>lock the main FMC operation</td>
<td>-</td>
<td>-</td>
</tr>
</tbody>
</table>

Input parameter(in)

- -

Output parameter(out)

- -

Return value

- -

Example:

/* lock the main FMC operation */

fmc_lock();

fmc_slp_unlock

The description of fmc_slp_unlock is shown as below:

Table 3-265. Function fmc_slp_unlock

<table>
<thead>
<tr>
<th>Function name</th>
<th>Function prototype</th>
<th>Function descriptions</th>
<th>Precondition</th>
<th>The called functions</th>
</tr>
</thead>
<tbody>
<tr>
<td>fmc_slp_unlock</td>
<td>void fmc_slp_unlock(void);</td>
<td>unlock operation of RUN_SLP bit in FMC_WS register</td>
<td>-</td>
<td>-</td>
</tr>
</tbody>
</table>

Input parameter(in)

- -

Output parameter(out)

- -

Return value

- -

Example:

/* unlock RUN_SLP bit in FMC_WS register */

fmc_slp_unlock();
fmc_wscnt_set

The description of fmc_wscnt_set is shown as below:

Table 3-266. Function fmc_wscnt_set

<table>
<thead>
<tr>
<th>Function name</th>
<th>fmc_wscnt_set</th>
</tr>
</thead>
<tbody>
<tr>
<td>Function prototype</td>
<td>void fmc_wscnt_set(uint32_t wscnt);</td>
</tr>
<tr>
<td>Function descriptions</td>
<td>set the wait state</td>
</tr>
<tr>
<td>Precondition</td>
<td>-</td>
</tr>
<tr>
<td>The called functions</td>
<td>-</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Input parameter(in)</th>
<th>wait state counter value</th>
</tr>
</thead>
<tbody>
<tr>
<td>wscnt</td>
<td></td>
</tr>
<tr>
<td>FMC_WAIT_STATE_0</td>
<td>0 wait state added</td>
</tr>
<tr>
<td>FMC_WAIT_STATE_1</td>
<td>1 wait state added</td>
</tr>
<tr>
<td>FMC_WAIT_STATE_2</td>
<td>2 wait state added</td>
</tr>
<tr>
<td>FMC_WAIT_STATE_3</td>
<td>3 wait state added</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Output parameter(out)</th>
<th>-</th>
</tr>
</thead>
</table>

Return value: -

Example:

/* set the wait state */

fmc_wscnt_set(FMC_WAIT_STATE_1);

fmc_prefetch_enable

The description of fmc_prefetch_enable is shown as below:

Table 3-267. Function fmc_prefetch_enable

<table>
<thead>
<tr>
<th>Function name</th>
<th>fmc_prefetch_enable</th>
</tr>
</thead>
<tbody>
<tr>
<td>Function prototype</td>
<td>void fmc_prefetch_enable(void);</td>
</tr>
<tr>
<td>Function descriptions</td>
<td>enable pre-fetch</td>
</tr>
<tr>
<td>Precondition</td>
<td>-</td>
</tr>
<tr>
<td>The called functions</td>
<td>-</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Input parameter(in)</th>
<th>-</th>
</tr>
</thead>
</table>

<table>
<thead>
<tr>
<th>Output parameter(out)</th>
<th>-</th>
</tr>
</thead>
</table>

Return value: -

Example:

/* enable pre-fetch */
fmc_prefetch_enable();

**fmc_prefetch_disable**

The description of `fmc_prefetch_disable` is shown as below:

<table>
<thead>
<tr>
<th>Function name</th>
<th>fmc_prefetch_disable</th>
</tr>
</thead>
<tbody>
<tr>
<td>Function prototype</td>
<td>void fmc_prefetch_disable(void);</td>
</tr>
<tr>
<td>Function descriptions</td>
<td>disable pre-fetch</td>
</tr>
<tr>
<td>Precondition</td>
<td>-</td>
</tr>
<tr>
<td>The called functions</td>
<td>-</td>
</tr>
<tr>
<td>Input parameter(in)</td>
<td>-</td>
</tr>
<tr>
<td>Output parameter(out)</td>
<td>-</td>
</tr>
<tr>
<td>Return value</td>
<td>-</td>
</tr>
</tbody>
</table>

Example:

/* disable pre-fetch */

fmc_prefetch_disable();

**fmc_low_power_enable**

The description of `fmc_low_power_enable` is shown as below:

<table>
<thead>
<tr>
<th>Function name</th>
<th>fmc_low_power_enable</th>
</tr>
</thead>
<tbody>
<tr>
<td>Function prototype</td>
<td>void fmc_low_power_enable(void);</td>
</tr>
<tr>
<td>Function descriptions</td>
<td>enable low power</td>
</tr>
<tr>
<td>Precondition</td>
<td>-</td>
</tr>
<tr>
<td>The called functions</td>
<td>-</td>
</tr>
<tr>
<td>Input parameter(in)</td>
<td>-</td>
</tr>
<tr>
<td>Output parameter(out)</td>
<td>-</td>
</tr>
<tr>
<td>Return value</td>
<td>-</td>
</tr>
</tbody>
</table>

Example:

/* enable low power */

fmc_low_power_enable();
**fmc_low_power_disable**

The description of `fmc_low_power_disable` is shown as below:

**Table 3-270. Function fmc_low_power_disable**

<table>
<thead>
<tr>
<th>Function name</th>
<th>fmc_low_power-disable</th>
</tr>
</thead>
<tbody>
<tr>
<td>Function prototype</td>
<td>void fmc_low_power_disable(void);</td>
</tr>
<tr>
<td>Function descriptions</td>
<td>disable low power</td>
</tr>
<tr>
<td>Precondition</td>
<td>-</td>
</tr>
<tr>
<td>The called functions</td>
<td>-</td>
</tr>
<tr>
<td>Input parameter(in)</td>
<td>-</td>
</tr>
<tr>
<td>Output parameter(out)</td>
<td>-</td>
</tr>
<tr>
<td>Return value</td>
<td>-</td>
</tr>
</tbody>
</table>

Example:

```c
/* disable low power */
fmc_low_power_disable();
```

**fmc_enter_slp_enable**

The description of `fmc_enter_slp_enable` is shown as below:

**Table 3-271. Function fmc_enter_slp_enable**

<table>
<thead>
<tr>
<th>Function name</th>
<th>fmc_enter_slp_enable</th>
</tr>
</thead>
<tbody>
<tr>
<td>Function prototype</td>
<td>void fmc_enter_slp_enable(void);</td>
</tr>
<tr>
<td>Function descriptions</td>
<td>flash enter sleep/power-down mode during MCU run/low-power run mode</td>
</tr>
<tr>
<td>Precondition</td>
<td>-</td>
</tr>
<tr>
<td>The called functions</td>
<td>-</td>
</tr>
<tr>
<td>Input parameter(in)</td>
<td>-</td>
</tr>
<tr>
<td>Output parameter(out)</td>
<td>-</td>
</tr>
<tr>
<td>Return value</td>
<td>-</td>
</tr>
</tbody>
</table>

Example:

```c
fmc_slp_unlock();
/* flash enter sleep/power-down mode during MCU run/low-power run mode */
fmc_enter_slp_enable();
```
### fmc_enter_slp_disable

The description of `fmc_enter_slp_disable` is shown as below:

**Table 3-272. Function fmc_enter_slp_disable**

<table>
<thead>
<tr>
<th>Function name</th>
<th>fmc_enter_slp_disable</th>
</tr>
</thead>
<tbody>
<tr>
<td>Function prototype</td>
<td>void fmc_enter_slp_disable(void);</td>
</tr>
<tr>
<td>Function descriptions</td>
<td>flash enter IDLE mode during MCU run/low-power run mode</td>
</tr>
<tr>
<td>Precondition</td>
<td>-</td>
</tr>
<tr>
<td>The called functions</td>
<td>-</td>
</tr>
<tr>
<td>Input parameter(in)</td>
<td>-</td>
</tr>
<tr>
<td>Output parameter(out)</td>
<td>-</td>
</tr>
<tr>
<td>Return value</td>
<td>-</td>
</tr>
</tbody>
</table>

Example:

```c
fmc_slp_unlock();
/* flash enter IDLE mode during MCU run/low-power run mode */
fmc_enter_slp_disable();
```

### fmc_enter_sleep_enable

The description of `fmc_enter_sleep_enable` is shown as below:

**Table 3-273. Function fmc_enter_sleep_enable**

<table>
<thead>
<tr>
<th>Function name</th>
<th>fmc_enter_sleep_enable</th>
</tr>
</thead>
<tbody>
<tr>
<td>Function prototype</td>
<td>void fmc_enter_sleep_enable(void);</td>
</tr>
<tr>
<td>Function descriptions</td>
<td>flash enter sleep mode when MCU enter deep-sleep mode or RUN_SLP bit is set</td>
</tr>
<tr>
<td>Precondition</td>
<td>-</td>
</tr>
<tr>
<td>The called functions</td>
<td>-</td>
</tr>
<tr>
<td>Input parameter(in)</td>
<td>-</td>
</tr>
<tr>
<td>Output parameter(out)</td>
<td>-</td>
</tr>
<tr>
<td>Return value</td>
<td>-</td>
</tr>
</tbody>
</table>

Example:

```c
/* flash enter sleep mode when MCU enter deep-sleep mode or RUN_SLP bit is set */
fmc_enter_sleep_enable();
```
fmc_enter_sleep_disable

The description of fmc_enter_sleep_disable is shown as below:

Table 3-274. Function fmc_enter_sleep_disable

<table>
<thead>
<tr>
<th>Function name</th>
<th>fmc_enter_sleep_disable</th>
</tr>
</thead>
<tbody>
<tr>
<td>Function prototype</td>
<td>void fmc_enter_sleep_disable(void);</td>
</tr>
<tr>
<td>Function descriptions</td>
<td>flash enter power-down mode when MCU enter deep-sleep mode or RUN_SLP bit is set</td>
</tr>
<tr>
<td>Precondition</td>
<td>-</td>
</tr>
<tr>
<td>The called functions</td>
<td>-</td>
</tr>
<tr>
<td>Input parameter(in)</td>
<td>-</td>
</tr>
<tr>
<td>Output parameter(out)</td>
<td>-</td>
</tr>
<tr>
<td>Return value</td>
<td>-</td>
</tr>
</tbody>
</table>

Example:

/* flash enter power-down mode when MCU enter deep-sleep mode or RUN_SLP bit is set */
fmc_enter_sleep_disable();

fmc_page_erase

The description of fmc_page_erase is shown as below:

Table 3-275. Function fmc_page_erase

<table>
<thead>
<tr>
<th>Function name</th>
<th>fmc_page_erase</th>
</tr>
</thead>
<tbody>
<tr>
<td>Function prototype</td>
<td>fmc_state_enum fmc_page_erase(uint32_t page_address);</td>
</tr>
<tr>
<td>Function descriptions</td>
<td>erase page</td>
</tr>
<tr>
<td>Precondition</td>
<td>fmc_unlock</td>
</tr>
<tr>
<td>The called functions</td>
<td>-</td>
</tr>
<tr>
<td>Input parameter(in)</td>
<td>page_address</td>
</tr>
<tr>
<td></td>
<td>the page address to be erased</td>
</tr>
<tr>
<td>Output parameter(out)</td>
<td>-</td>
</tr>
<tr>
<td>Return value</td>
<td>-</td>
</tr>
</tbody>
</table>

Example:

fmc_unlock();

/* erase page */
fmc_state_enum state = fmc_page_erase(0x08004000);

fmc_mass_erase

The description of fmc_mass_erase is shown as below:

**Table 3-276. Function fmc_mass_erase**

<table>
<thead>
<tr>
<th>Function name</th>
<th>fmc_mass_erase</th>
</tr>
</thead>
<tbody>
<tr>
<td>Function prototype</td>
<td>fmc_state_enum fmc_mass_erase(void);</td>
</tr>
<tr>
<td>Function descriptions</td>
<td>erase whole chip</td>
</tr>
<tr>
<td>Precondition</td>
<td>fmc_unlock</td>
</tr>
<tr>
<td>The called functions</td>
<td></td>
</tr>
<tr>
<td>Input parameter(in)</td>
<td></td>
</tr>
<tr>
<td>Output parameter(out)</td>
<td></td>
</tr>
<tr>
<td>Return value</td>
<td>fmc_state_enum state of FMC, the enum members can refer to members of the enum</td>
</tr>
</tbody>
</table>

**Example:**

fmc_unlock();

/* erase whole chip */

fmc_state_enum state = fmc_mass_erase();

fmc_word_program

The description of fmc_word_program is shown as below:

**Table 3-277. Function fmc_word_program**

<table>
<thead>
<tr>
<th>Function name</th>
<th>fmc_word_program</th>
</tr>
</thead>
<tbody>
<tr>
<td>Function prototype</td>
<td>fmc_state_enum fmc_word_program(uint32_t address, uint32_t data);</td>
</tr>
<tr>
<td>Function descriptions</td>
<td>program a word at the corresponding address</td>
</tr>
<tr>
<td>Precondition</td>
<td>fmc_unlock</td>
</tr>
<tr>
<td>The called functions</td>
<td></td>
</tr>
<tr>
<td>Input parameter(in)</td>
<td>address the address to be programmed</td>
</tr>
<tr>
<td>Input parameter(in)</td>
<td>data the data to be programmed</td>
</tr>
<tr>
<td>Output parameter(out)</td>
<td></td>
</tr>
<tr>
<td>Return value</td>
<td>fmc_state_enum state of FMC, the enum members can refer to members of the enum</td>
</tr>
</tbody>
</table>
Example:

fmc_unlock();

fmc_page_erase(0x08004000);

/* program a word at the corresponding address */

fmc_state_enum fmc_state = fmc_word_program(0x08004000, 0xaabbcddd);

### fmc_fast_program

The description of fmc_fast_program is shown as below:

**Table 3-278. Function fmc_fast_program**

<table>
<thead>
<tr>
<th>Function name</th>
<th>fmc_fast_program</th>
</tr>
</thead>
<tbody>
<tr>
<td>Function prototype</td>
<td>fmc_state_enum fmc_fast_program(uint32_t address, uint64_t data[]);</td>
</tr>
<tr>
<td>Function descriptions</td>
<td>FMC fast program one row data (32 double-word) starting at the corresponding address</td>
</tr>
<tr>
<td>Precondition</td>
<td>fmc_unlock</td>
</tr>
<tr>
<td>The called functions</td>
<td>-</td>
</tr>
</tbody>
</table>

**Input parameter (in)**

- **address**
  - the address to be programmed

**Input parameter (in)**

- **data**
  - the data to be programmed

**Output parameter (out)**

- -

**Return value**

- fmc_state_enum
  - state of FMC, the enum members can refer to members of the enum

Example:

/* data buffer for fast programming */

```c
static uint64_t data_buffer[32] = {
  0x0000000000000000U, 0x1111111111111111U, 0x2222222222222222U,
  0x3333333333333333U, 0x4444444444444444U, 0x5555555555555555U,
  0x6666666666666666U, 0x7777777777777777U,
  0x8888888888888888U, 0x9999999999999999U, 0xAABBABBBBBBBBBBU,
  0xCCCCCCCCCCCCCCCU, 0xDDDDDDDDDDDDDDDDU,
  0xEEEEEEEDEEEEEEEEU, 0xFFFFFFFFFFFFFU,
};
```
fmc_unlock();

fmc_page_erase(0x08004000);

/* program flash */

fmc_state_enum fmc_state = fmc_fast_program(0x08004000, data_buffer);

**ob_unlock**

The description of ob_unlock is shown as below:

<table>
<thead>
<tr>
<th>Function name</th>
<th>ob_unlock</th>
</tr>
</thead>
<tbody>
<tr>
<td>Function prototype</td>
<td>void ob_unlock(void);</td>
</tr>
<tr>
<td>Function descriptions</td>
<td>unlock the option byte operation</td>
</tr>
<tr>
<td>Precondition</td>
<td>fmc_unlock</td>
</tr>
<tr>
<td>The called functions</td>
<td>-</td>
</tr>
<tr>
<td>Input parameter (in)</td>
<td>-</td>
</tr>
<tr>
<td>Output parameter (out)</td>
<td>-</td>
</tr>
<tr>
<td>Return value</td>
<td>-</td>
</tr>
</tbody>
</table>

Example:

fmc_unlock();

/* unlock the option byte operation */

ob_unlock();

**ob_lock**

The description of ob_lock is shown as below:
Table 3-280. Function ob_lock

<table>
<thead>
<tr>
<th>Function name</th>
<th>ob_lock</th>
</tr>
</thead>
<tbody>
<tr>
<td>Function prototype</td>
<td>void ob_lock(void);</td>
</tr>
<tr>
<td>Function descriptions</td>
<td>lock the option byte operation</td>
</tr>
<tr>
<td>Precondition</td>
<td>fmc_lock</td>
</tr>
<tr>
<td>The called functions</td>
<td>-</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Input parameter(in)</th>
</tr>
</thead>
<tbody>
<tr>
<td>-</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Output parameter(out)</th>
</tr>
</thead>
<tbody>
<tr>
<td>-</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Return value</th>
</tr>
</thead>
<tbody>
<tr>
<td>-</td>
</tr>
</tbody>
</table>

Example:

fmc_lock();

/* lock the option byte operation */

ob_lock();

**ob_erase**

The description of ob_erase is shown as below:

Table 3-281. Function ob_erase

<table>
<thead>
<tr>
<th>Function name</th>
<th>ob_erase</th>
</tr>
</thead>
<tbody>
<tr>
<td>Function prototype</td>
<td>void ob_erase(void);</td>
</tr>
<tr>
<td>Function descriptions</td>
<td>erase the option byte</td>
</tr>
<tr>
<td>Precondition</td>
<td>ob_unlock</td>
</tr>
<tr>
<td>The called functions</td>
<td>-</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Input parameter(in)</th>
</tr>
</thead>
<tbody>
<tr>
<td>-</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Output parameter(out)</th>
</tr>
</thead>
<tbody>
<tr>
<td>-</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Return value</th>
</tr>
</thead>
<tbody>
<tr>
<td>-</td>
</tr>
</tbody>
</table>

Example:

fmc_unlock();

ob_unlock();

/* erase the option byte */

fmc_state_enum fmc_state = ob_erase();

Table 3-262. fmc_state_enum.
ob_write_protection_enable

The description of `ob_write_protection_enable` is shown as below:

<table>
<thead>
<tr>
<th>Function name</th>
<th><code>ob_write_protection_enable</code></th>
</tr>
</thead>
<tbody>
<tr>
<td>Function prototype</td>
<td><code>fmc_state_enum ob_write_protection_enable(uint32_t ob_wp);</code></td>
</tr>
<tr>
<td>Function descriptions</td>
<td>enable erase/write protection</td>
</tr>
<tr>
<td>Precondition</td>
<td><code>ob_unlock</code></td>
</tr>
<tr>
<td>The called functions</td>
<td></td>
</tr>
</tbody>
</table>

**Input parameter (in)**
- `ob_wp` specify sector to be write protected

**Output parameter (out)**
- 

Return value
- `fmc_state_enum` state of FMC, the enum members can refer to members of the enum

Example:

```c
fmc_unlock();
ob_unlock();
/* enable write protection */
fmc_state_enum fmc_state = ob_write_protection_enable(OB_WP_1);
```

ob_security_protection_config

The description of `ob_security_protection_config` is shown as below:

<table>
<thead>
<tr>
<th>Function name</th>
<th><code>ob_security_protection_config</code></th>
</tr>
</thead>
<tbody>
<tr>
<td>Function prototype</td>
<td><code>fmc_state_enum ob_security_protection_config(uint8_t ob_spc);</code></td>
</tr>
<tr>
<td>Function descriptions</td>
<td>configure security protection</td>
</tr>
<tr>
<td>Precondition</td>
<td><code>ob_unlock</code></td>
</tr>
<tr>
<td>The called functions</td>
<td></td>
</tr>
</tbody>
</table>

**Input parameter (in)**
- `ob_spc` specify security protection
  - `FMC_NSPC` no security protection
  - `FMC_LSPC` low security protection
  - `FMC_HSPC` high security protection

**Output parameter (out)**
- 

Return value
- `fmc_state_enum` state of FMC, the enum members can refer to members of the enum
Example:

```c
fmc_state_enum fmc_state;
fmc_unlock();
ob_unlock();
/* enable security protection */
fmc_state = ob_security_protection_config(FMC_USPC);
```

### ob_user_write

The description of `ob_user_write` is shown as below:

**Table 3-284. Function ob_user_write**

<table>
<thead>
<tr>
<th>Function name</th>
<th>ob_user_write</th>
</tr>
</thead>
<tbody>
<tr>
<td>Function prototype</td>
<td><code>fmc_state_enum ob_user_write(uint8_t ob_fwdgt, uint8_t ob_deepsleep, uint8_t ob_stdby, uint8_t ob_bor_th);</code></td>
</tr>
<tr>
<td>Function descriptions</td>
<td>program option bytes USER</td>
</tr>
<tr>
<td>Precondition</td>
<td><code>ob_unlock</code></td>
</tr>
<tr>
<td>The called functions</td>
<td>-</td>
</tr>
</tbody>
</table>

**Input parameter (in)**

<table>
<thead>
<tr>
<th>ob_fwdgt</th>
<th>option bytes free watchdog value</th>
</tr>
</thead>
<tbody>
<tr>
<td>OB_FWDGT_SW</td>
<td>software free watchdog</td>
</tr>
<tr>
<td>OB_FWDGT_HW</td>
<td>hardware free watchdog</td>
</tr>
</tbody>
</table>

**Input parameter (in)**

<table>
<thead>
<tr>
<th>ob_deepsleep</th>
<th>option bytes deepsleep reset value</th>
</tr>
</thead>
<tbody>
<tr>
<td>OB_DEEPSLEEP_NRS</td>
<td>no reset when entering deepsleep mode</td>
</tr>
<tr>
<td>OB_DEEPSLEEP_RST</td>
<td>generate a reset instead of entering deepsleep mode</td>
</tr>
</tbody>
</table>

**Input parameter (in)**

<table>
<thead>
<tr>
<th>ob_stdby</th>
<th>option bytes standby reset value</th>
</tr>
</thead>
<tbody>
<tr>
<td>OB_STDBY_NRST</td>
<td>no reset when entering standby mode</td>
</tr>
<tr>
<td>OB_STDBY_RST</td>
<td>generate a reset instead of entering standby mode</td>
</tr>
</tbody>
</table>

**Input parameter (in)**

<table>
<thead>
<tr>
<th>ob_bor_th</th>
<th>option bytes BOR threshold value</th>
</tr>
</thead>
<tbody>
<tr>
<td>OB_BOR_TH_VALUE0</td>
<td>BOR threshold value 0</td>
</tr>
<tr>
<td>OB_BOR_TH_VALUE1</td>
<td>BOR threshold value 1</td>
</tr>
<tr>
<td>OB_BOR_TH_VALUE2</td>
<td>BOR threshold value 2</td>
</tr>
<tr>
<td>OB_BOR_TH_VALUE3</td>
<td>BOR threshold value 3</td>
</tr>
<tr>
<td>OB_BOR_TH_VALUE4</td>
<td>BOR threshold value 4</td>
</tr>
</tbody>
</table>

**Output parameter (out)**
Return value

<table>
<thead>
<tr>
<th>fmc_state_enum</th>
<th>state of FMC, the enum members can refer to members of the enum</th>
</tr>
</thead>
</table>

Example:

defmc_unlock();

defob_unlock();

/* program the FMC user option byte */

defmc_state_enum fmc_state = ob_user_write(OB_FWDGT_HW, OB_DEEPSLEEP_NRST, OB_STDBY_NRST, OB_BOR_TH_VALUE0);

**ob_data_program**

The description of **ob_data_program** is shown as below:

**Table 3-285. Function ob_data_program**

<table>
<thead>
<tr>
<th>Function name</th>
<th>ob_data_program</th>
</tr>
</thead>
<tbody>
<tr>
<td>Function prototype</td>
<td>fmc_state_enum ob_data_program(uint16_t data);</td>
</tr>
<tr>
<td>Function descriptions</td>
<td>program option bytes DATA</td>
</tr>
<tr>
<td>Preconditions</td>
<td>ob_unlock</td>
</tr>
<tr>
<td>The called functions</td>
<td>-</td>
</tr>
</tbody>
</table>

**Input parameter (in)**

| data            | the data to be programmed |

**Output parameter (out)**

| -                | - |

**Return value**

<table>
<thead>
<tr>
<th>fmc_state_enum</th>
<th>state of FMC, the enum members can refer to members of the enum</th>
</tr>
</thead>
</table>

Example:


defmc_unlock();

defob_unlock();

/* program option bytes data */

defmc_state_enum fmc_state = ob_data_program(0xdd22);

**ob_user_get**

The description of **ob_user_get** is shown as below:

**Table 3-286. Function ob_user_get**

<table>
<thead>
<tr>
<th>Function name</th>
<th>ob_user_get</th>
</tr>
</thead>
</table>

Function prototype: `uint8_t ob_user_get(void);`

Function descriptions: get the value of option bytes USER

Precondition: -

The called functions: -

| Input parameter(in) | - |
| Output parameter(out) | - |

Return value: `uint8_t`

Example:

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

uint8_t user = ob_user_get();
```

### ob_data_get

The description of `ob_data_get` is shown as below:

#### Table 3-287. Function ob_data_get

 Function name | ob_data_get  
--- | ---
 Function prototype | `uint16_t ob_data_get(void);`
 Function descriptions | get the value of option bytes DATA
 Precondition | -
 The called functions | -

| Input parameter(in) | - |
| Output parameter(out) | - |

Return value: `uint16_t`

Example:

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

uint16_t data = ob_data_get();
```

### ob_write_protection_get

The description of `ob_write_protection_get` is shown as below:

#### Table 3-288. Function ob_write_protection_get

 Function name | ob_write_protection_get  
--- | ---
 Function prototype | `uint32_t ob_write_protection_get(void);`
### Function descriptions

- **get the value of option bytes write protection**

### Precondition

- **The called functions**

### Input parameter (in)

- **Output parameter (out)**

### Return value

- **uint32_t**

  - the FMC write protection option byte value (0x0 – 0xFFFF FFFF)

---

**Example:**

```c
/* get the FMC option byte write protection */
uint32_t wp = ob_write_protection_get();
```

**ob_security_protection_flag_get**

The description of `ob_security_protection_flag_get` is shown as below:

**Table 3-289. Function ob_security_protection_flag_get**

<table>
<thead>
<tr>
<th>Function name</th>
<th>ob_security_protection_flag_get</th>
</tr>
</thead>
<tbody>
<tr>
<td>Function prototype</td>
<td>FlagStatus ob_security_protection_flag_get(void);</td>
</tr>
<tr>
<td>Function descriptions</td>
<td>get the FMC option byte security protection state</td>
</tr>
<tr>
<td>Precondition</td>
<td>-</td>
</tr>
<tr>
<td>The called functions</td>
<td>-</td>
</tr>
<tr>
<td>Input parameter (in)</td>
<td>-</td>
</tr>
<tr>
<td>Output parameter (out)</td>
<td>-</td>
</tr>
<tr>
<td>Return value</td>
<td>FlagStatus SET or RESET</td>
</tr>
</tbody>
</table>

**Example:**

```c
/* get the FMC option byte security protection level */
FlagStatus spc_flag = ob_security_protection_flag_get();
```

**fmc_flag_get**

The description of `fmc_flag_get` is shown as below:

**Table 3-290. Function fmc_flag_get**

<table>
<thead>
<tr>
<th>Function name</th>
<th>fmc_flag_get</th>
</tr>
</thead>
<tbody>
<tr>
<td>Function prototype</td>
<td>FlagStatus fmc_flag_get(uint32_t flag);</td>
</tr>
<tr>
<td>Function descriptions</td>
<td>get FMC flag status</td>
</tr>
</tbody>
</table>
### Precondition

- The called functions

### Input parameter (in)

<table>
<thead>
<tr>
<th>flag</th>
<th>FMC flag</th>
</tr>
</thead>
<tbody>
<tr>
<td>FMC_FLAG_BUSY</td>
<td>FMC busy flag</td>
</tr>
<tr>
<td>FMC_FLAG_PGERR</td>
<td>FMC programming error flag</td>
</tr>
<tr>
<td>FMC_FLAG_PGAERR</td>
<td>FMC program alignment error flag</td>
</tr>
<tr>
<td>FMC_FLAG_WPERR</td>
<td>FMC write protection error flag</td>
</tr>
<tr>
<td>FMC_FLAG_END</td>
<td>FMC end of programming flag</td>
</tr>
<tr>
<td>FMC_FLAG_SLP</td>
<td>FMC sleep or power down flag</td>
</tr>
</tbody>
</table>

### Output parameter (out)

- 

### Return value

| FlagStatus | SET or RESET |

### Example:

```c
/* get FMC end flag */
FlagStatus flag = fmc_flag_get(FMC_FLAG_END);
```

### fmc_flag_clear

The description of fmc_flag_clear is shown as below:

#### Table 3-291. Function fmc_flag_clear

<table>
<thead>
<tr>
<th>Function name</th>
<th>fmc_flag_clear</th>
</tr>
</thead>
<tbody>
<tr>
<td>Function prototype</td>
<td>void fmc_flag_clear(uint32_t flag);</td>
</tr>
<tr>
<td>Function descriptions</td>
<td>clear the FMC flag</td>
</tr>
<tr>
<td>Precondition</td>
<td>-</td>
</tr>
<tr>
<td>The called functions</td>
<td>-</td>
</tr>
</tbody>
</table>

### Input parameter (in)

<table>
<thead>
<tr>
<th>flag</th>
<th>FMC flag</th>
</tr>
</thead>
<tbody>
<tr>
<td>FMC_FLAG_BUSY</td>
<td>FMC busy flag</td>
</tr>
<tr>
<td>FMC_FLAG_PGERR</td>
<td>FMC programming error flag</td>
</tr>
<tr>
<td>FMC_FLAG_PGAERR</td>
<td>FMC program alignment error flag</td>
</tr>
<tr>
<td>FMC_FLAG_WPERR</td>
<td>FMC write protection error flag</td>
</tr>
<tr>
<td>FMC_FLAG_END</td>
<td>FMC end of programming flag</td>
</tr>
<tr>
<td>FMC_FLAG_SLP</td>
<td>FMC sleep or power down flag</td>
</tr>
</tbody>
</table>

### Output parameter (out)

- 

### Return value

- 

### Example:

/* get FMC end flag */
FlagStatus flag = fmc_flag_get(FMC_FLAG_END);
fmc_flag_clear(FMC_FLAG_END);

fmc_interrupt_enable

The description of fmc_interrupt_enable is shown as below:

Table 3-292. Function fmc_interrupt_enable

<table>
<thead>
<tr>
<th>Function name</th>
<th>fmc_interrupt_enable</th>
</tr>
</thead>
<tbody>
<tr>
<td>Function prototype</td>
<td>void fmc_interrupt_enable(uint32_t interrupt);</td>
</tr>
<tr>
<td>Function descriptions</td>
<td>enable FMC interrupt</td>
</tr>
<tr>
<td>Precondition</td>
<td>-</td>
</tr>
<tr>
<td>The called functions</td>
<td>-</td>
</tr>
</tbody>
</table>

Input parameter (in)

<table>
<thead>
<tr>
<th>interrupt</th>
<th>FMC interrupt</th>
</tr>
</thead>
<tbody>
<tr>
<td>FMC_INT_END</td>
<td>FMC end of program interrupt</td>
</tr>
<tr>
<td>FMC_INT_ERR</td>
<td>FMC error interrupt</td>
</tr>
</tbody>
</table>

Output parameter (out)

- -

Return value

Example:

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

fmc_interrupt_disable

The description of fmc_interrupt_disable is shown as below:

Table 3-293. Function fmc_interrupt_disable

<table>
<thead>
<tr>
<th>Function name</th>
<th>fmc_interrupt_disable</th>
</tr>
</thead>
<tbody>
<tr>
<td>Function prototype</td>
<td>void fmc_interrupt_disable(uint32_t interrupt);</td>
</tr>
<tr>
<td>Function descriptions</td>
<td>disable FMC interrupt</td>
</tr>
<tr>
<td>Precondition</td>
<td>-</td>
</tr>
<tr>
<td>The called functions</td>
<td>-</td>
</tr>
</tbody>
</table>

Input parameter (in)

<table>
<thead>
<tr>
<th>interrupt</th>
<th>FMC interrupt</th>
</tr>
</thead>
<tbody>
<tr>
<td>FMC_INT_END</td>
<td>FMC end of program interrupt</td>
</tr>
<tr>
<td>FMC_INT_ERR</td>
<td>FMC error interrupt</td>
</tr>
</tbody>
</table>

Output parameter (out)

- -

Return value
Example:

/* disable FMC end interrupt */

fmc_interrupt_disable(FMC_INT_END);

**fmc_interrupt_flag_get**

The description of fmc_interrupt_flag_get is shown as below:

<table>
<thead>
<tr>
<th>Table 3-294. Function fmc_interrupt_flag_get</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Function name</strong></td>
</tr>
<tr>
<td><strong>Function prototype</strong></td>
</tr>
<tr>
<td><strong>Function descriptions</strong></td>
</tr>
<tr>
<td><strong>Precondition</strong></td>
</tr>
<tr>
<td><strong>The called functions</strong></td>
</tr>
<tr>
<td><strong>Input parameter</strong>(in)</td>
</tr>
<tr>
<td>flag</td>
</tr>
<tr>
<td>FMC_INT_FLAG_PGE</td>
</tr>
<tr>
<td>FMC_INT_FLAG_PGA</td>
</tr>
<tr>
<td>FMC_INT_FLAG_WPE</td>
</tr>
<tr>
<td>FMC_INT_FLAG_END</td>
</tr>
<tr>
<td><strong>Output parameter</strong>(out)</td>
</tr>
<tr>
<td><strong>Return value</strong></td>
</tr>
<tr>
<td>FlagStatus</td>
</tr>
</tbody>
</table>

Example:

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

FlagStatus flag = fmc_interrupt_flag_get(FMC_INT_FLAG_PGERR);

**fmc_interrupt_flag_clear**

The description of fmc_interrupt_flag_clear is shown as below:

<table>
<thead>
<tr>
<th>Table 3-295. Function fmc_interrupt_flag_clear</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Function name</strong></td>
</tr>
<tr>
<td><strong>Function prototype</strong></td>
</tr>
<tr>
<td><strong>Function descriptions</strong></td>
</tr>
<tr>
<td><strong>Precondition</strong></td>
</tr>
<tr>
<td><strong>The called functions</strong></td>
</tr>
</tbody>
</table>
### Input parameter (in)

<table>
<thead>
<tr>
<th>flag</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>FMC_INT_FLAG_PGE</td>
<td>FMC program operation error flag</td>
</tr>
<tr>
<td>FMC_INT_FLAG_PGA</td>
<td>FMC program alignment error flag</td>
</tr>
<tr>
<td>FMC_INT_FLAG_WPE</td>
<td>FMC erase/program protection error flag</td>
</tr>
<tr>
<td>FMC_INT_FLAG_END</td>
<td>FMC end of operation flag</td>
</tr>
</tbody>
</table>

### Output parameter (out)

- `-` -

### Return value

- `-` -

Example:

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

fmc_interrupt_flag_get(FMC_INT_FLAG_PGERR);
```

### 3.12. FWDGT

The free watchdog timer (FWDGT) is a hardware timing circuitry that can be used to detect system failures due to software malfunctions. It’s suitable for the situation that requires an independent environment and lower timing accuracy. The FWDGT registers are listed in chapter 3.12.1 the FWDGT firmware functions are introduced in chapter 3.12.2.

#### 3.12.1. Descriptions of Peripheral registers

FWDGT registers are listed in the table shown as below:

<table>
<thead>
<tr>
<th>Registers</th>
<th>Descriptions</th>
</tr>
</thead>
<tbody>
<tr>
<td>FWDGT_CTL</td>
<td>Control register</td>
</tr>
<tr>
<td>FWDGT_PSC</td>
<td>Prescaler register</td>
</tr>
<tr>
<td>FWDGT_RLD</td>
<td>Reload register</td>
</tr>
<tr>
<td>FWDGT_STAT</td>
<td>Status register</td>
</tr>
<tr>
<td>FWDGT_WND</td>
<td>window register</td>
</tr>
</tbody>
</table>

#### 3.12.2. Descriptions of Peripheral functions

FWDGT firmware functions are listed in the table shown as below:
### Table 3-297. FWDGT firmware function

<table>
<thead>
<tr>
<th>Function name</th>
<th>Function description</th>
</tr>
</thead>
<tbody>
<tr>
<td>fwdgt_write_enable</td>
<td>enable write access to FWDGT_PSC and FWDGT_RLD and FWDGT_WND</td>
</tr>
<tr>
<td>fwdgt_write_disable</td>
<td>disable write access to FWDGT_PSC and FWDGT_RLD and FWDGT_WND</td>
</tr>
<tr>
<td>fwdgt_enable</td>
<td>start the FWDGT counter</td>
</tr>
<tr>
<td>fwdgt_prescaler_value_config</td>
<td>configure the FWDGT counter prescaler value</td>
</tr>
<tr>
<td>fwdgt_reload_value_config</td>
<td>configure the FWDGT counter reload value</td>
</tr>
<tr>
<td>fwdgt_window_value_config</td>
<td>configure the FWDGT counter window value</td>
</tr>
<tr>
<td>fwdgt_counter_reload</td>
<td>reload the counter of FWDGT</td>
</tr>
<tr>
<td>fwdgt_config</td>
<td>configure counter reload value, and prescaler divider value</td>
</tr>
<tr>
<td>fwdgt_flag_get</td>
<td>get flag state of FWDGT</td>
</tr>
</tbody>
</table>

#### fwdgt_write_enable

The description of `fwdgt_write_enable` is shown as below:

**Table 3-298. Function fwdgt_write_enable**

<table>
<thead>
<tr>
<th>Function name</th>
<th>fwdgt_write_enable</th>
</tr>
</thead>
<tbody>
<tr>
<td>Function prototype</td>
<td>void fwdgt_write_enable(void);</td>
</tr>
<tr>
<td>Function descriptions</td>
<td>enable write access to FWDGT_PSC and FWDGT_RLD and FWDGT_WND</td>
</tr>
<tr>
<td>Precondition</td>
<td>-</td>
</tr>
<tr>
<td>The called functions</td>
<td>-</td>
</tr>
<tr>
<td>Input parameter(in)</td>
<td>-</td>
</tr>
<tr>
<td>Output parameter(out)</td>
<td>-</td>
</tr>
<tr>
<td>Return value</td>
<td>-</td>
</tr>
</tbody>
</table>

Example:

```c
/* enable write access to FWDGT_PSC and FWDGT_RLD and FWDGT_WND */

fwdgt_write_enable();
```

#### fwdgt_write_disable

The description of `fwdgt_write_disable` is shown as below:

**Table 3-299. Function fwdgt_write_disable**

<table>
<thead>
<tr>
<th>Function name</th>
<th>fwdgt_write_disable</th>
</tr>
</thead>
<tbody>
<tr>
<td>Function prototype</td>
<td>void fwdgt_write_disable(void);</td>
</tr>
<tr>
<td>Function descriptions</td>
<td>disable write access to FWDGT_PSC,FWDGT_RLD and FWDGT_WND</td>
</tr>
</tbody>
</table>
Precondition: -

The called functions: -

<table>
<thead>
<tr>
<th>Input parameter(in)</th>
<th>-</th>
<th>-</th>
</tr>
</thead>
</table>

<table>
<thead>
<tr>
<th>Output parameter(out)</th>
<th>-</th>
<th>-</th>
</tr>
</thead>
</table>

Return value: -

Example:

/* disable write access to FWDGT_PSC, FWDGT_RLD and FWDGT_WND */
fwdgt_write_disable();

**fwdgt_enable**

The description of fwdgt_enable is shown as below:

<table>
<thead>
<tr>
<th>Function name</th>
<th>fwdgt_enable</th>
</tr>
</thead>
<tbody>
<tr>
<td>Function prototype</td>
<td>void fwdgt_enable(void);</td>
</tr>
<tr>
<td>Function descriptions</td>
<td>start the FWDGT counter</td>
</tr>
<tr>
<td>Precondition</td>
<td>-</td>
</tr>
</tbody>
</table>

The called functions: -

<table>
<thead>
<tr>
<th>Input parameter(in)</th>
<th>-</th>
<th>-</th>
</tr>
</thead>
</table>

<table>
<thead>
<tr>
<th>Output parameter(out)</th>
<th>-</th>
<th>-</th>
</tr>
</thead>
</table>

Return value: -

Example:

/* start the free watchdog timer counter */
fwdgt_enable();

**fwdgt_prescaler_value_config**

The description of fwdgt_prescaler_value_config is shown as below:

<table>
<thead>
<tr>
<th>Function name</th>
<th>fwdgt_prescaler_value_config</th>
</tr>
</thead>
<tbody>
<tr>
<td>Function prototype</td>
<td>ErrStatus fwdgt_prescaler_value_config(uint16_t prescaler_value);</td>
</tr>
<tr>
<td>Function descriptions</td>
<td>configure the FWDGT counter clock prescaler value</td>
</tr>
<tr>
<td>Precondition</td>
<td>-</td>
</tr>
</tbody>
</table>

Example:

/* configure the FWDGT counter clock prescaler value */
fwdgt_prescaler_value_config(uint16_t prescaler_value);
The called functions

<table>
<thead>
<tr>
<th>Function name</th>
<th>fwdgt_prescaler_value_config</th>
</tr>
</thead>
</table>

**Input parameter (in)**

<table>
<thead>
<tr>
<th>prescaler_value</th>
<th>specify prescaler value</th>
</tr>
</thead>
<tbody>
<tr>
<td>FWDGT_PSC_DIVx</td>
<td>FWDGT prescaler set to x(x=4,8,16,32,64,128,256)</td>
</tr>
</tbody>
</table>

**Output parameter (out)**

- 

**Return value**

ErrStatus | ERROR / SUCCESS |

Example:

/* set FWDGT prescaler to 4 */

ErrStatus flag;

flag = fwdgt_prescaler_value_config(FWDGT_PSC_DIV4);

**fwdgt_reload_value_config**

The description of fwdgt_reload_value_config is shown as below:

**Table 3-302. Function fwdgt_reload_value_config**

<table>
<thead>
<tr>
<th>Function name</th>
<th>fwdgt_reload_value_config</th>
</tr>
</thead>
</table>

**Function prototype**

ErrStatus fwdgt_reload_value_config(uint16_t reload_value);

**Function descriptions**

configure the FWDGT counter reload value

**Precondition**

- 

**The called functions**

- 

**Input parameter (in)**

<table>
<thead>
<tr>
<th>reload_value</th>
<th>reload_value, specify reload value(0x0000 - 0xFFF)</th>
</tr>
</thead>
</table>

**Output parameter (out)**

- 

**Return value**

ErrStatus | ERROR / SUCCESS |

Example:

/* set FWDGT reload value to 0xFFF */

ErrStatus flag;

flag = fwdgt_reload_value_config(0xFFF);

**fwdgt_window_value_config**

The description of fwdgt_window_value_config is shown as below:

**Table 3-303. Function fwdgt_window_value_config**

<table>
<thead>
<tr>
<th>Function name</th>
<th>fwdgt_window_value_config</th>
</tr>
</thead>
</table>

**Function prototype**

ErrStatus fwdgt_window_value_config(uint16_t window_value);
**Function descriptions**
configure the FWDGT counter window value

<table>
<thead>
<tr>
<th>Precondition</th>
<th>-</th>
</tr>
</thead>
<tbody>
<tr>
<td>The called functions</td>
<td>-</td>
</tr>
</tbody>
</table>

**Input parameter (in)**

<table>
<thead>
<tr>
<th>window_value</th>
<th>window_value, specify window value(0x0000 - 0x0FFF)</th>
</tr>
</thead>
</table>

**Output parameter (out)**

| - | - |

**Return value**

| ErrStatus | ERROR / SUCCESS |

Example:

/* set FWDGT window value to 0xFFF */

ErrStatus flag;

flag = fwdgt_window_value_config(0xFFF);

**fwdgt_counter_reload**

The description of fwdgt_counter_reload is shown as below:

**Table 3-304. Function fwdgt_counter_reload**

<table>
<thead>
<tr>
<th>Function name</th>
<th>fwdgt_counter_reload</th>
</tr>
</thead>
</table>

| Function prototype | void fwdgt_counter_reload(void); |

<table>
<thead>
<tr>
<th>Function descriptions</th>
<th>reload the counter of FWDGT</th>
</tr>
</thead>
<tbody>
<tr>
<td>Precondition</td>
<td>-</td>
</tr>
<tr>
<td>The called functions</td>
<td>-</td>
</tr>
</tbody>
</table>

**Input parameter (in)**

| - | - |

**Output parameter (out)**

| - | - |

**Return value**

| - | - |

Example:

/* reload FWDGT counter */

fwdgt_counter_reload();

**fwdgt_config**

The description of fwdgt_config is shown as below:

**Table 3-305. Function fwdgt_config**

<table>
<thead>
<tr>
<th>Function name</th>
<th>fwdgt_config</th>
</tr>
</thead>
</table>

| Function prototype | ErrStatus fwdgt_config(uint16_t reload_value, uint8_t prescaler_div); |

Example:

/* reload FWDGT counter */

fwdgt_config(uint16_t reload_value, uint8_t prescaler_div);
Function descriptions: configure counter reload value, and prescaler divider value

Precondition:
- 

The called functions:
- 

Input parameter (in):
- 

reload_value: specify reload value (0x0000 - 0x0FFF)

Input parameter (in):
- 

prescaler_div: FWDGT prescaler value -
- FWDGT_PSC_DIV4: FWDGT prescaler set to 4
- FWDGT_PSC_DIV8: FWDGT prescaler set to 8
- FWDGT_PSC_DIV16: FWDGT prescaler set to 16
- FWDGT_PSC_DIV32: FWDGT prescaler set to 32
- FWDGT_PSC_DIV64: FWDGT prescaler set to 64
- FWDGT_PSC_DIV128: FWDGT prescaler set to 128
- FWDGT_PSC_DIV256: FWDGT prescaler set to 256

Output parameter (out):
- 

Error Status: ERROR or SUCCESS

Example:

/* configure FWDGT counter clock: 40KHz(IRC40K) / 64 = 0.625 KHz */
fwdgt_config(2*500, FWDGT_PSC_DIV64);

fwdgt_flag_get

The description of fwdgt_flag_get is shown as below:

Table 3-306. Function fwdgt_flag_get

<table>
<thead>
<tr>
<th>Function name</th>
<th>fwdgt_flag_get</th>
</tr>
</thead>
<tbody>
<tr>
<td>Function prototype</td>
<td>FlagStatus fwdgt_flag_get(uint16_t flag);</td>
</tr>
<tr>
<td>Function descriptions</td>
<td>get flag state of FWDGT</td>
</tr>
<tr>
<td>Precondition</td>
<td>-</td>
</tr>
<tr>
<td>The called functions</td>
<td>-</td>
</tr>
</tbody>
</table>

Input parameter (in):
- 

flag: flag to get
- FWDGT_FLAG_PUD: a write operation to FWDGT_PSC register is going on
- FWDGT_FLAG_RUD: a write operation to FWDGT_RLD register is going on
- FWDGT_FLAG_WUD: a write operation to FWDGT_WND register is going on

Output parameter (out):
- 

Return value:
- FlagStatus: SET or RESET
Example:

/* test if a prescaler value update is on going */

FlagStatus status;
status = fwdgt_flag_get(FWDGT_FLAG_PUD);

3.13. GPIO

GPIO is used to implement logic input/output functions for the devices. The GPIO registers are listed in chapter 3.13.1, the GPIO firmware functions are introduced in chapter 3.13.2.

3.13.1. Descriptions of Peripheral registers

GPIO registers are listed in the table shown as below:

Table 3-307. GPIO Registers

<table>
<thead>
<tr>
<th>Registers</th>
<th>Descriptions</th>
</tr>
</thead>
<tbody>
<tr>
<td>GPIOx_CTL</td>
<td>GPIO port control register</td>
</tr>
<tr>
<td>GPIOx_OMODE</td>
<td>GPIO port output mode register</td>
</tr>
<tr>
<td>GPIOx_OSPD</td>
<td>GPIO port output speed register</td>
</tr>
<tr>
<td>GPIOx_PUD</td>
<td>GPIO port pull-up/pull-down register</td>
</tr>
<tr>
<td>GPIOx_ISTAT</td>
<td>GPIO port input status register</td>
</tr>
<tr>
<td>GPIOx_OCTL</td>
<td>GPIO port output control register</td>
</tr>
<tr>
<td>GPIOx_BOP</td>
<td>GPIO port bit operation register</td>
</tr>
<tr>
<td>GPIOx_LOCK</td>
<td>GPIO port configuration lock register</td>
</tr>
<tr>
<td>GPIOx_AFSEL0</td>
<td>GPIO alternate function selected register 0</td>
</tr>
<tr>
<td>GPIOx_AFSEL1</td>
<td>GPIO alternate function selected register 1</td>
</tr>
<tr>
<td>GPIOx_BC</td>
<td>GPIO bit clear register</td>
</tr>
<tr>
<td>GPIOx_TG</td>
<td>GPIO port bit toggle register</td>
</tr>
</tbody>
</table>

3.13.2. Descriptions of Peripheral functions

GPIO firmware functions are listed in the table shown as below:

Table 3-308. GPIO firmware function

<table>
<thead>
<tr>
<th>Function name</th>
<th>Function description</th>
</tr>
</thead>
<tbody>
<tr>
<td>gpio_deinit</td>
<td>reset GPIO port</td>
</tr>
<tr>
<td>gpio_mode_set</td>
<td>set GPIO mode</td>
</tr>
<tr>
<td>gpio_output_options_set</td>
<td>set GPIO output type and speed</td>
</tr>
<tr>
<td>gpio_bit_set</td>
<td>set GPIO pin</td>
</tr>
<tr>
<td>gpio_bit_reset</td>
<td>reset GPIO pin bit</td>
</tr>
<tr>
<td>gpio_bit_write</td>
<td>write data to the specified GPIO pin</td>
</tr>
<tr>
<td>gpio_port_write</td>
<td>write data to the specified GPIO port</td>
</tr>
<tr>
<td>Function name</td>
<td>Function description</td>
</tr>
<tr>
<td>-----------------------</td>
<td>------------------------------------------</td>
</tr>
<tr>
<td>gpio_input_bit_get</td>
<td>get GPIO pin input status</td>
</tr>
<tr>
<td>gpio_input_port_get</td>
<td>get GPIO port input status</td>
</tr>
<tr>
<td>gpio_output_bit_get</td>
<td>get GPIO pin output status</td>
</tr>
<tr>
<td>gpio_output_port_get</td>
<td>get GPIO port output status</td>
</tr>
<tr>
<td>gpio_af_set</td>
<td>set GPIO alternate function</td>
</tr>
<tr>
<td>gpio_pin_lock</td>
<td>lock GPIO pin bit</td>
</tr>
<tr>
<td>gpio_bit_toggle</td>
<td>toggle GPIO pin status</td>
</tr>
<tr>
<td>gpio_port_toggle</td>
<td>toggle GPIO port status</td>
</tr>
</tbody>
</table>

**gpio_deinit**

The description of `gpio_deinit` is shown as below:

**Table 3-309. Function gpio_deinit**

<table>
<thead>
<tr>
<th>Function name</th>
<th>gpio_deinit</th>
</tr>
</thead>
<tbody>
<tr>
<td>Function prototype</td>
<td>void gpio_deinit(uint32_t gpio_periph);</td>
</tr>
<tr>
<td>Function descriptions</td>
<td>reset GPIO port</td>
</tr>
<tr>
<td>Precondition</td>
<td>-</td>
</tr>
<tr>
<td>The called functions</td>
<td>rcu_periph_reset_enable / rcu_periph_reset_disable</td>
</tr>
</tbody>
</table>

**Input parameter (in)**

<table>
<thead>
<tr>
<th>gpio_periph</th>
<th>GPIO port</th>
</tr>
</thead>
<tbody>
<tr>
<td>GPIOx</td>
<td>GPIOx(x = A,B,C,D,F)</td>
</tr>
</tbody>
</table>

**Output parameter (out)**

| - | - |

**Return value**

- -

Example:

/* reset GPIOA */

gpio_deinit(GPIOA);

**gpio_mode_set**

The description of `gpio_mode_set` is shown as below:

**Table 3-310. Function gpio_mode_set**

<table>
<thead>
<tr>
<th>Function name</th>
<th>gpio_mode_set</th>
</tr>
</thead>
<tbody>
<tr>
<td>Function prototype</td>
<td>void gpio_mode_set(uint32_t gpio_periph, uint32_t mode, uint32_t pull_up_down, uint32_t pin);</td>
</tr>
<tr>
<td>Function descriptions</td>
<td>set GPIO mode</td>
</tr>
<tr>
<td>Precondition</td>
<td>-</td>
</tr>
<tr>
<td>The called functions</td>
<td>-</td>
</tr>
</tbody>
</table>

**Input parameter (in)**
```markdown
<table>
<thead>
<tr>
<th>Function name</th>
<th>gpio_output_options_set</th>
</tr>
</thead>
<tbody>
<tr>
<td>Function prototype</td>
<td>void gpio_output_options_set(uint32_t gpio_periph, uint8_t otype, uint32_t speed, uint32_t_t pin);</td>
</tr>
<tr>
<td>Function descriptions</td>
<td>set GPIO output type and speed</td>
</tr>
<tr>
<td>Precondition</td>
<td>-</td>
</tr>
<tr>
<td>The called functions</td>
<td>-</td>
</tr>
<tr>
<td>Input parameter(in)</td>
<td></td>
</tr>
<tr>
<td>gpio_periph</td>
<td>GPIO port</td>
</tr>
<tr>
<td>GPIOx</td>
<td>GPIOx(x = A,B,C,D,F)</td>
</tr>
</tbody>
</table>
```

Example:

/* config PA0 as input mode with pullup*/
gpio_mode_set(GPIOA, GPIO_MODE_INPUT, GPIO_PUPD_PULLUP, GPIO_PIN_0);

gpio_output_options_set

The description of gpio_output_options_set is shown as below:

### Table 3-311. Function gpio_output_options_set

<table>
<thead>
<tr>
<th>Function name</th>
<th>gpio_output_options_set</th>
</tr>
</thead>
<tbody>
<tr>
<td>Function prototype</td>
<td>void gpio_output_options_set(uint32_t gpio_periph, uint8_t otype, uint32_t speed, uint32_t_t pin);</td>
</tr>
<tr>
<td>Function descriptions</td>
<td>set GPIO output type and speed</td>
</tr>
<tr>
<td>Precondition</td>
<td>-</td>
</tr>
<tr>
<td>The called functions</td>
<td>-</td>
</tr>
<tr>
<td>Input parameter(in)</td>
<td></td>
</tr>
<tr>
<td>gpio_periph</td>
<td>GPIO port</td>
</tr>
<tr>
<td>GPIOx</td>
<td>GPIOx(x = A,B,C,D,F)</td>
</tr>
</tbody>
</table>
### gpio_output_options_set

<table>
<thead>
<tr>
<th>Input parameter (in)</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>otype</strong></td>
</tr>
<tr>
<td>GPIO_OTYPE_PP</td>
</tr>
<tr>
<td>GPIO_OTYPE_OD</td>
</tr>
<tr>
<td><strong>speed</strong></td>
</tr>
<tr>
<td>GPIO_OSPEED_2MHZ</td>
</tr>
<tr>
<td>GPIO_OSPEED_10MHZ</td>
</tr>
<tr>
<td>GPIO_OSPEED_50MHZ</td>
</tr>
<tr>
<td><strong>pin</strong></td>
</tr>
<tr>
<td>GPIO_PIN_x</td>
</tr>
<tr>
<td>GPIO_PIN_ALL</td>
</tr>
</tbody>
</table>

#### Example:

```c
/* config PA0 as push pull mode */
gpio_output_options_set(GPIOA, GPIO_OTYPE_PP, GPIO_OSPEED_2MHZ, GPIO_PIN_0);
```

### gpio_bit_set

The description of gpio_bit_set is shown as below:

#### Table 3-312. Function gpio_bit_set

<table>
<thead>
<tr>
<th>Function name</th>
<th>gpio_bit_set</th>
</tr>
</thead>
<tbody>
<tr>
<td>Function prototype</td>
<td>void gpio_bit_set(uint32_t gpio_periph,uint32_t pin);</td>
</tr>
<tr>
<td>Function descriptions</td>
<td>set GPIO pin bit</td>
</tr>
<tr>
<td>Precondition</td>
<td>-</td>
</tr>
<tr>
<td>The called functions</td>
<td>-</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Input parameter (in)</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>gpio_periph</strong></td>
</tr>
<tr>
<td>GPIO port</td>
</tr>
<tr>
<td><strong>GPIOx</strong></td>
</tr>
<tr>
<td>GPIOx(x = A,B,C,D,F)</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Input parameter (in)</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>pin</strong></td>
</tr>
<tr>
<td>GPIO pin</td>
</tr>
<tr>
<td><strong>GPIO_PIN_x</strong></td>
</tr>
<tr>
<td>GPIO_PIN_x(x=0..15)</td>
</tr>
<tr>
<td><strong>GPIO_PIN_ALL</strong></td>
</tr>
<tr>
<td>All pins</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Output parameter (out)</th>
</tr>
</thead>
<tbody>
<tr>
<td>-</td>
</tr>
</tbody>
</table>
Example:

/\* set PA0*/

gpio_bit_set(GPIOA, GPIO_PIN_0);

**gpio_bit_reset**

The description of gpio_bit_reset is shown as below:

<table>
<thead>
<tr>
<th>Function name</th>
<th>gpio_bit_reset</th>
</tr>
</thead>
<tbody>
<tr>
<td>Function prototype</td>
<td>void gpio_bit_reset(uint32_t gpio_periph,uint32_t pin);</td>
</tr>
<tr>
<td>Function descriptions</td>
<td>reset GPIO pin</td>
</tr>
<tr>
<td>Precondition</td>
<td>-</td>
</tr>
<tr>
<td>The called functions</td>
<td>-</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Input parameter(in)</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>gpio_periph</td>
<td>GPIO port</td>
</tr>
<tr>
<td>GPIOx</td>
<td>GPIOx(x = A,B,C,D,F)</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Input parameter(in)</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>pin</td>
<td>GPIO pin</td>
</tr>
<tr>
<td>GPIO_PIN_x</td>
<td>GPIO_PIN_x(x=0..15)</td>
</tr>
<tr>
<td>GPIO_PIN_ALL</td>
<td>All pins</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Output parameter(out)</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>-</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Return value</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>-</td>
</tr>
</tbody>
</table>

Example:

/\* reset PA0*/

gpio_bit_set(GPIOA, GPIO_PIN_0);

**gpio_bit_write**

The description of gpio_bit_write is shown as below:

<table>
<thead>
<tr>
<th>Function name</th>
<th>gpio_bit_write</th>
</tr>
</thead>
<tbody>
<tr>
<td>Function prototype</td>
<td>void gpio_bit_write(uint32_t gpio_periph,uint32_t pin,bit_status bit_value);</td>
</tr>
<tr>
<td>Function descriptions</td>
<td>write data to the specified GPIO pin</td>
</tr>
<tr>
<td>Precondition</td>
<td>-</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Input parameter(in)</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>gpio_periph</td>
<td>GPIO port</td>
</tr>
<tr>
<td>GPIOx</td>
<td>GPIOx(x = A,B,C,D,F)</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Input parameter(in)</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>pin</td>
<td>GPIO pin</td>
</tr>
<tr>
<td>GPIO_PIN_x</td>
<td>GPIO_PIN_x(x=0..15)</td>
</tr>
<tr>
<td>GPIO_PIN_ALL</td>
<td>All pins</td>
</tr>
</tbody>
</table>
The called functions

- 

Input parameter (in)

<table>
<thead>
<tr>
<th>gpio_periph</th>
<th>GPIO port</th>
</tr>
</thead>
<tbody>
<tr>
<td>GPIOx</td>
<td>GPIOx(x = A,B,C,D,F)</td>
</tr>
</tbody>
</table>

Input parameter (in)

<table>
<thead>
<tr>
<th>pin</th>
<th>GPIO pin</th>
</tr>
</thead>
<tbody>
<tr>
<td>GPIO_PIN_x</td>
<td>GPIO_PIN_x(x=0..15)</td>
</tr>
<tr>
<td>GPIO_PIN_ALL</td>
<td>All pins</td>
</tr>
</tbody>
</table>

Input parameter (in)

<table>
<thead>
<tr>
<th>bit_value</th>
<th>SET or RESET</th>
</tr>
</thead>
<tbody>
<tr>
<td>RESET</td>
<td>clear the port pin</td>
</tr>
<tr>
<td>SET</td>
<td>set the port pin</td>
</tr>
</tbody>
</table>

Output parameter (out)

- 

Return value

- 

Example:

/* write 1 to PA0 */

gpio_bit_write(GPIOA, GPIO_PIN_0, SET);

gpio_port_write

The description of gpio_port_write is shown as below:

Table 3-315. Function gpio_port_write

<table>
<thead>
<tr>
<th>Function name</th>
<th>gpio_port_write</th>
</tr>
</thead>
<tbody>
<tr>
<td>Function prototype</td>
<td>void gpio_port_write(uint32_t gpio_periph,uint16_t data);</td>
</tr>
<tr>
<td>Function descriptions</td>
<td>write data to the specified GPIO port</td>
</tr>
<tr>
<td>Precondition</td>
<td>-</td>
</tr>
<tr>
<td>The called functions</td>
<td>-</td>
</tr>
</tbody>
</table>

Input parameter (in)

<table>
<thead>
<tr>
<th>gpio_periph</th>
<th>GPIO port</th>
</tr>
</thead>
<tbody>
<tr>
<td>GPIOx</td>
<td>GPIOx(x = A,B,C,D,F)</td>
</tr>
</tbody>
</table>

Input parameter (in)

| data              | specify the value to be written to the port output data register |

Output parameter (out)

- 

Return value

- 

Example:

/*write 1010 0101 1010 0101 to Port A */
```c
gpio_port_write(GPIOA, 0xA5A5);
```

**gpio_input_bit_get**

The description of `gpio_input_bit_get` is shown as below:

**Table 3-316. Function gpio_input_bit_get**

<table>
<thead>
<tr>
<th>Function name</th>
<th>gpio_input_bit_get</th>
</tr>
</thead>
<tbody>
<tr>
<td>Function prototype</td>
<td>FlagStatus gpio_input_bit_get(uint32_t gpio_periph,uint32_t pin);</td>
</tr>
<tr>
<td>Function descriptions</td>
<td>get GPIO pin input status</td>
</tr>
<tr>
<td>Precondition</td>
<td>-</td>
</tr>
<tr>
<td>The called functions</td>
<td>-</td>
</tr>
<tr>
<td><strong>Input parameter(in)</strong></td>
<td></td>
</tr>
<tr>
<td>gpio_periph</td>
<td>GPIO port</td>
</tr>
<tr>
<td>GPIOx</td>
<td>GPIOx(x = A,B,C,D,F)</td>
</tr>
<tr>
<td>pin</td>
<td>GPIO pin</td>
</tr>
<tr>
<td>GPIO_PIN_x</td>
<td>GPIO_PIN_x(x=0..15)</td>
</tr>
<tr>
<td>GPIO_PIN_ALL</td>
<td>All pins</td>
</tr>
<tr>
<td><strong>Output parameter(out)</strong></td>
<td></td>
</tr>
<tr>
<td></td>
<td>-</td>
</tr>
<tr>
<td><strong>Return value</strong></td>
<td>FlagStatus</td>
</tr>
<tr>
<td></td>
<td>SET / RESET</td>
</tr>
</tbody>
</table>

Example:

```c
/* get status of PA0 */
FlagStatus bit_state;
bit_state = gpio_input_bit_get(GPIOA, GPIO_PIN_0);
```

**gpio_input_port_get**

The description of `gpio_input_port_get` is shown as below:

**Table 3-317. Function gpio_input_port_get**

<table>
<thead>
<tr>
<th>Function name</th>
<th>gpio_input_port_get</th>
</tr>
</thead>
<tbody>
<tr>
<td>Function prototype</td>
<td>uint16_t gpio_input_port_get(uint32_t gpio_periph);</td>
</tr>
<tr>
<td>Function descriptions</td>
<td>get GPIO all pins input status</td>
</tr>
<tr>
<td>Precondition</td>
<td>-</td>
</tr>
<tr>
<td>The called functions</td>
<td>-</td>
</tr>
<tr>
<td><strong>Input parameter(in)</strong></td>
<td></td>
</tr>
<tr>
<td>gpio_periph</td>
<td>GPIO port</td>
</tr>
<tr>
<td>GPIOx</td>
<td>GPIOx(x = A,B,C,D,F)</td>
</tr>
<tr>
<td><strong>Output parameter(out)</strong></td>
<td></td>
</tr>
</tbody>
</table>

```c
```
Example:

/* get input value of Port A */
uint16_t port_state;
port_state = gpio_input_port_get(GPIOA);

**gpio_output_bit_get**

The description of gpio_output_bit_get is shown as below:

Table 3-318. Function gpio_output_bit_get

<table>
<thead>
<tr>
<th>Function name</th>
<th>gpio_output_bit_get</th>
</tr>
</thead>
<tbody>
<tr>
<td>Function prototype</td>
<td>FlagStatus gpio_output_bit_get(uint32_t gpio_periph,uint32_t pin);</td>
</tr>
<tr>
<td>Function descriptions</td>
<td>get GPIO pin output status</td>
</tr>
<tr>
<td>Precondition</td>
<td>-</td>
</tr>
<tr>
<td>The called functions</td>
<td>-</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Input parameter (in)</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>gpio_periph</td>
<td>GPIO port</td>
</tr>
<tr>
<td>GPIOx</td>
<td>GPIOx(x = A,B,C,D,F)</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Input parameter (in)</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>pin</td>
<td>GPIO pin</td>
</tr>
<tr>
<td>GPIO_PIN_x</td>
<td>GPIO_PIN_x(x=0..15)</td>
</tr>
<tr>
<td>GPIO_PIN_ALL</td>
<td>All pins</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Output parameter (out)</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>-</td>
<td>-</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Return value</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>FlagStatus</td>
<td>SET / RESET</td>
</tr>
</tbody>
</table>

Example:

/* get output status of PA0 */

FlagStatus bit_state;
bit_state = gpio_output_bit_get(GPIOA, GPIO_PIN_0);

**gpio_output_port_get**

The description of gpio_output_port_get is shown as below:

Table 3-319. Function gpio_output_port_get

<table>
<thead>
<tr>
<th>Function name</th>
<th>gpio_output_port_get</th>
</tr>
</thead>
<tbody>
<tr>
<td>Function prototype</td>
<td>uint16_t gpio_output_port_get(uint32_t gpio_periph);</td>
</tr>
</tbody>
</table>
**Function descriptions**

get GPIO all pins output status

**Precondition**

-

**The called functions**

-

**Input parameter**(in)

<table>
<thead>
<tr>
<th>gpio_periph</th>
<th>GPIO port</th>
</tr>
</thead>
<tbody>
<tr>
<td>GPIOx</td>
<td>GPIOx(x = A,B,C,D,F)</td>
</tr>
</tbody>
</table>

**Output parameter**(out)

-

**Return value**

Uint16_t 0x0000-0xFFFF

Example:

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

uint16_t port_state;

port_state = gpio_output_port_get(GPIOA);
```

**gpio_af_set**

The description of gpio_af_set is shown as below:

**Table 3-320. Function gpio_af_set**

<table>
<thead>
<tr>
<th>Function name</th>
<th>gpio_af_set</th>
</tr>
</thead>
<tbody>
<tr>
<td>Function prototype</td>
<td>void gpio_af_set(uint32_t gpio_periph, uint32_t alt_func_num, uint32_t pin);</td>
</tr>
<tr>
<td>Function descriptions</td>
<td>set GPIO alternate function</td>
</tr>
<tr>
<td>Precondition</td>
<td>-</td>
</tr>
<tr>
<td>The called functions</td>
<td>-</td>
</tr>
</tbody>
</table>

**Input parameter**(in)

<table>
<thead>
<tr>
<th>gpio_periph</th>
<th>GPIO port</th>
</tr>
</thead>
<tbody>
<tr>
<td>GPIOx</td>
<td>GPIOx(x = A,B,C,D,F)</td>
</tr>
</tbody>
</table>

**Input parameter**(in)

<table>
<thead>
<tr>
<th>alt_func_num</th>
<th>GPIO pin af function, please refer to specific device datasheet</th>
</tr>
</thead>
<tbody>
<tr>
<td>GPIO_AF_0</td>
<td>RTC_OUT, CK_OUT, SWDIO, SWCLK</td>
</tr>
<tr>
<td>GPIO_AF_1</td>
<td>TIMER1, TIMER2</td>
</tr>
<tr>
<td>GPIO_AF_2</td>
<td>TIMER8, TIMER11, LPTIMER, LPUART</td>
</tr>
<tr>
<td>GPIO_AF_3</td>
<td>SLCD</td>
</tr>
<tr>
<td>GPIO_AF_4</td>
<td>I2C0, I2C1, I2C2</td>
</tr>
<tr>
<td>GPIO_AF_5</td>
<td>SPI0, SPI1, I2S1</td>
</tr>
<tr>
<td>GPIO_AF_6</td>
<td>SPI1, CMP0, CMP1, I2S1</td>
</tr>
<tr>
<td>GPIO_AF_7</td>
<td>USART0, USART1, LPUART, UART3, UART4</td>
</tr>
<tr>
<td>GPIO_AF_8</td>
<td>USART1, LPUART, UART3, UART4, I2C1, CTC, CMP1</td>
</tr>
<tr>
<td>GPIO_AF_9</td>
<td>EVENTOUT</td>
</tr>
</tbody>
</table>

**Input parameter**(in)
**GD32L23x Firmware Library User Guide**

<table>
<thead>
<tr>
<th>pin</th>
<th>GPIO pin</th>
</tr>
</thead>
<tbody>
<tr>
<td>GPIO_PIN_x</td>
<td>GPIO_PIN_x(x=0..15)</td>
</tr>
<tr>
<td>GPIO_PIN_ALL</td>
<td>All pins</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Output parameter(out)</th>
</tr>
</thead>
<tbody>
<tr>
<td>-</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Return value</th>
</tr>
</thead>
<tbody>
<tr>
<td>-</td>
</tr>
</tbody>
</table>

Example:

/* set PA0 alternate function 0 */

gpio_af_set(GPIOA, GPIO_AF_0, GPIO_PIN_0);

**gpio_pin_lock**

The description of gpio_pin_lock is shown as below:

**Table 3-321. Function gpio_pin_lock**

<table>
<thead>
<tr>
<th>Function name</th>
<th>gpio_pin_lock</th>
</tr>
</thead>
<tbody>
<tr>
<td>Function prototype</td>
<td>void gpio_pin_lock(uint32_t gpio_periph, uint32_t pin);</td>
</tr>
<tr>
<td>Function descriptions</td>
<td>lock GPIO pin bit</td>
</tr>
<tr>
<td>Precondition</td>
<td>-</td>
</tr>
<tr>
<td>The called functions</td>
<td>-</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Input parameter(in)</th>
</tr>
</thead>
<tbody>
<tr>
<td>gpio_periph</td>
</tr>
<tr>
<td>GPIOx</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Input parameter(in)</th>
</tr>
</thead>
<tbody>
<tr>
<td>pin</td>
</tr>
<tr>
<td>GPIO_PIN_x</td>
</tr>
<tr>
<td>GPIO_PIN_ALL</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Output parameter(out)</th>
</tr>
</thead>
<tbody>
<tr>
<td>-</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Return value</th>
</tr>
</thead>
<tbody>
<tr>
<td>-</td>
</tr>
</tbody>
</table>

Example:

/* lock PA0 */

gpio_pin_lock(GPIOA, GPIO_PIN_0);

**gpio_bit_toggle**

The description of gpio_bit_toggle is shown as below:
Table 3-322. Function gpio_bit_toggle

<table>
<thead>
<tr>
<th>Function name</th>
<th>gpio_bit_toggle</th>
</tr>
</thead>
<tbody>
<tr>
<td>Function prototype</td>
<td>void gpio_bit_toggle(uint32_t gpio_periph, uint32_t pin);</td>
</tr>
<tr>
<td>Function descriptions</td>
<td>toggle GPIO pin status</td>
</tr>
<tr>
<td>Precondition</td>
<td>-</td>
</tr>
<tr>
<td>The called functions</td>
<td>-</td>
</tr>
</tbody>
</table>

**Input parameter (in)**

<table>
<thead>
<tr>
<th>gpio_periph</th>
<th>GPIO port</th>
</tr>
</thead>
<tbody>
<tr>
<td>GPIOx</td>
<td>GPIOx(x = A,B,C,D,F)</td>
</tr>
</tbody>
</table>

**Input parameter (in)**

<table>
<thead>
<tr>
<th>pin</th>
<th>GPIO pin</th>
</tr>
</thead>
<tbody>
<tr>
<td>GPIO_PIN_x</td>
<td>GPIO_PIN_x(x=0..15)</td>
</tr>
<tr>
<td>GPIO_PIN_ALL</td>
<td>GPIO_PIN_ALL</td>
</tr>
</tbody>
</table>

**Output parameter (out)**

| - | - |

**Return value**

| - | - |

Example:

/* toggle PA0 */

gpio_bit_toggle(GPIOA, GPIO_PIN_0);

gpio_port_toggle

The description of gpio_port_toggle is shown as below:

Table 3-323. Function gpio_port_toggle

<table>
<thead>
<tr>
<th>Function name</th>
<th>gpio_port_toggle</th>
</tr>
</thead>
<tbody>
<tr>
<td>Function prototype</td>
<td>void gpio_port_toggle(uint32_t gpio_periph);</td>
</tr>
<tr>
<td>Function descriptions</td>
<td>toggle GPIO port status</td>
</tr>
<tr>
<td>Precondition</td>
<td>-</td>
</tr>
<tr>
<td>The called functions</td>
<td>-</td>
</tr>
</tbody>
</table>

**Input parameter (in)**

<table>
<thead>
<tr>
<th>gpio_periph</th>
<th>GPIO port</th>
</tr>
</thead>
<tbody>
<tr>
<td>GPIOx</td>
<td>GPIOx(x = A,B,C,D,F)</td>
</tr>
</tbody>
</table>

**Output parameter (out)**

| - | - |

**Return value**

| - | - |

Example:

/* toggle GPIOA*/

gpio_port_toggle(GPIOA);
3.14. **I2C**

The I2C (inter-integrated circuit) module provides an I2C interface which is an industry standard two-line serial interface for MCU to communicate with external I2C interface. The I2C registers are listed in chapter 3.14.1, the I2C firmware functions are introduced in chapter 3.14.2.

### 3.14.1. Descriptions of Peripheral registers

I2C registers are listed in the table shown as below:

<table>
<thead>
<tr>
<th>Registers</th>
<th>Descriptions</th>
</tr>
</thead>
<tbody>
<tr>
<td>I2C_CTL0</td>
<td>Control register 0</td>
</tr>
<tr>
<td>I2C_CTL1</td>
<td>Control register 1</td>
</tr>
<tr>
<td>I2C_SADDR0</td>
<td>Slave address register 0</td>
</tr>
<tr>
<td>I2C_SADDR1</td>
<td>Slave address register 1</td>
</tr>
<tr>
<td>I2C_TIMING</td>
<td>Timing register</td>
</tr>
<tr>
<td>I2C_TIMEOUT</td>
<td>Timeout register</td>
</tr>
<tr>
<td>I2C_STAT</td>
<td>Status register</td>
</tr>
<tr>
<td>I2C_STATC</td>
<td>I2C status clear register</td>
</tr>
<tr>
<td>I2C_PEC</td>
<td>PEC register</td>
</tr>
<tr>
<td>I2C_RDATA</td>
<td>Receive data register</td>
</tr>
<tr>
<td>I2C_TDATA</td>
<td>Transmit data register</td>
</tr>
<tr>
<td>I2C_CTL2</td>
<td>Control register 2</td>
</tr>
</tbody>
</table>

### 3.14.2. Descriptions of Peripheral functions

I2C firmware functions are listed in the table shown as below:

<table>
<thead>
<tr>
<th>Function name</th>
<th>Function description</th>
</tr>
</thead>
<tbody>
<tr>
<td>i2c_deinit</td>
<td>reset I2C</td>
</tr>
<tr>
<td>i2c_timing_config</td>
<td>configure the timing parameters</td>
</tr>
<tr>
<td>i2c_digital_noise_filter_config</td>
<td>configure digital noise filter</td>
</tr>
<tr>
<td>i2c_analog_noise_filter_enable</td>
<td>enable analog noise filter</td>
</tr>
<tr>
<td>i2c_analog_noise_filter_disable</td>
<td>disable analog noise filter</td>
</tr>
<tr>
<td>i2c_master_clock_config</td>
<td>configure the SCL high and low period of clock in master mode</td>
</tr>
<tr>
<td>i2c_master_addressing</td>
<td>configure I2C slave address and transfer direction in master mode</td>
</tr>
<tr>
<td>i2c_address10_header_enable</td>
<td>10-bit address header executes read direction only in master receive mode</td>
</tr>
<tr>
<td>Function name</td>
<td>Function description</td>
</tr>
<tr>
<td>-------------------------------</td>
<td>---------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>i2c_address10_header_disable</td>
<td>10-bit address header executes complete sequence in master receive mode</td>
</tr>
<tr>
<td>i2c_address10_enable</td>
<td>enable 10-bit addressing mode in master mode</td>
</tr>
<tr>
<td>i2c_address10_disable</td>
<td>disable 10-bit addressing mode in master mode</td>
</tr>
<tr>
<td>i2c_automatic_end_enable</td>
<td>enable I2C automatic end mode in master mode</td>
</tr>
<tr>
<td>i2c_automatic_end_disable</td>
<td>disable I2C automatic end mode in master mode</td>
</tr>
<tr>
<td>i2c_slave_response_to_gcall_enable</td>
<td>enable the response to a general call</td>
</tr>
<tr>
<td>i2c_slave_response_to_gcall_disable</td>
<td>disable the response to a general call</td>
</tr>
<tr>
<td>i2c_stretch_scl_low_enable</td>
<td>enable to stretch SCL low when data is not ready in slave mode</td>
</tr>
<tr>
<td>i2c_stretch_scl_low_disable</td>
<td>disable to stretch SCL low when data is not ready in slave mode</td>
</tr>
<tr>
<td>i2c_address_config</td>
<td>configure I2C slave address</td>
</tr>
<tr>
<td>i2c_address_bit_compare_config</td>
<td>define which bits of ADDRESS[7:1] need to compare with the incoming address byte</td>
</tr>
<tr>
<td>i2c_address_disable</td>
<td>disable I2C address in slave mode</td>
</tr>
<tr>
<td>i2c_second_address_config</td>
<td>configure I2C second slave address</td>
</tr>
<tr>
<td>i2c_second_address_disable</td>
<td>disable I2C second address in slave mode</td>
</tr>
<tr>
<td>i2c_recevied_address_get</td>
<td>get received match address in slave mode</td>
</tr>
<tr>
<td>i2c_slave_byte_control_enable</td>
<td>enable slave byte control</td>
</tr>
<tr>
<td>i2c_slave_byte_control_disable</td>
<td>disable slave byte control</td>
</tr>
<tr>
<td>i2c_nack_enable</td>
<td>generate a NACK in slave mode</td>
</tr>
<tr>
<td>i2c_nack_disable</td>
<td>generate an ACK in slave mode</td>
</tr>
<tr>
<td>i2c_wakeup_from_deepsleep_enable</td>
<td>enable wakeup from Deep-sleep mode</td>
</tr>
<tr>
<td>i2c_wakeup_from_deepsleep_disable</td>
<td>disable wakeup from Deep-sleep mode</td>
</tr>
<tr>
<td>i2c_enable</td>
<td>enable I2C</td>
</tr>
<tr>
<td>i2c_disable</td>
<td>disable I2C</td>
</tr>
<tr>
<td>i2c_start_on_bus</td>
<td>generate a START condition on I2C bus</td>
</tr>
<tr>
<td>i2c_stop_on_bus</td>
<td>generate a STOP condition on I2C bus</td>
</tr>
<tr>
<td>i2c_data_transmit</td>
<td>I2C transmit data</td>
</tr>
<tr>
<td>i2c_data_receive</td>
<td>I2C receive data</td>
</tr>
<tr>
<td>i2c_reload_enable</td>
<td>enable I2C reload mode</td>
</tr>
<tr>
<td>i2c_reload_disable</td>
<td>disable I2C reload mode</td>
</tr>
<tr>
<td>i2c_transfer_byte_number_config</td>
<td>configure number of bytes to be transferred</td>
</tr>
<tr>
<td>i2c_dma_enable</td>
<td>enable I2C DMA for transmission or reception</td>
</tr>
<tr>
<td>i2c_dma_disable</td>
<td>disable I2C DMA for transmission or reception</td>
</tr>
<tr>
<td>i2c_piec_transfer</td>
<td>I2C transfers PEC value</td>
</tr>
<tr>
<td>i2c_piec_enable</td>
<td>enable I2C PEC calculation</td>
</tr>
<tr>
<td>i2c_piec_disable</td>
<td>disable I2C PEC calculation</td>
</tr>
<tr>
<td>i2c_piec_value_get</td>
<td>get packet error checking value</td>
</tr>
<tr>
<td>i2c_smbus_alert_enable</td>
<td>enable SMBus Alert</td>
</tr>
</tbody>
</table>
Function name | Function description
---|---
i2c_smbus_alert_disable | disable SMBus Alert
i2c_smbus_default_addr_enable | enable SMBus device default address
i2c_smbus_default_addr_disable | disable SMBus device default address
i2c_smbus_host_addr_enable | enable SMBus Host address
i2c_smbus_host_addr_disable | disable SMBus Host address
i2c_extended_clock_timeout_enable | enable extended clock timeout detection
i2c_extended_clock_timeout_disable | disable extended clock timeout detection
i2c_clock_timeout_enable | enable clock timeout detection
i2c_clock_timeout_disable | disable clock timeout detection
i2c_bus_timeout_b_config | configure bus timeout B
i2c_bus_timeout_a_config | configure bus timeout A
i2c_idle_clock_timeout_config | configure idle clock timeout detection
i2c_flag_get | get I2C flag status
i2c_flag_clear | clear I2C flag status
i2c_interrupt_enable | enable I2C interrupt
i2c_interrupt_disable | disable I2C interrupt
i2c_interrupt_flag_get | get I2C interrupt flag status
i2c_interrupt_flag_clear | clear I2C interrupt flag status

Table 3-326. i2c_interrupt_flag_enum

<table>
<thead>
<tr>
<th>Member name</th>
<th>Function description</th>
</tr>
</thead>
<tbody>
<tr>
<td>I2C_INT_FLAG_TI</td>
<td>transmit interrupt flag</td>
</tr>
<tr>
<td>I2C_INT_FLAG_RBNE</td>
<td>I2C_RDATA is not empty during receiving interrupt flag</td>
</tr>
<tr>
<td>I2C_INT_FLAG_ADDSEND</td>
<td>address received matches in slave mode interrupt flag</td>
</tr>
<tr>
<td>I2C_INT_FLAG_NACK</td>
<td>not acknowledge interrupt flag</td>
</tr>
<tr>
<td>I2C_INT_FLAG_STPDET</td>
<td>stop condition detected in slave mode interrupt flag</td>
</tr>
<tr>
<td>I2C_INT_FLAG_TC</td>
<td>transfer complete in master mode interrupt flag</td>
</tr>
<tr>
<td>I2C_INT_FLAG_TCR</td>
<td>transfer complete reload interrupt flag</td>
</tr>
<tr>
<td>I2C_INT_FLAG_BERR</td>
<td>bus error interrupt flag</td>
</tr>
<tr>
<td>I2C_INT_FLAG_LOSTARB</td>
<td>arbitration lost interrupt flag</td>
</tr>
<tr>
<td>I2C_INT_FLAG_OUERR</td>
<td>overrun/underrun error in slave mode interrupt flag</td>
</tr>
<tr>
<td>I2C_INT_FLAG_PECERR</td>
<td>PEC error interrupt flag</td>
</tr>
<tr>
<td>I2C_INT_FLAG_TIMEOUT</td>
<td>timeout interrupt flag</td>
</tr>
<tr>
<td>I2C_INT_FLAG_SMBALT</td>
<td>SMBus Alert interrupt flag</td>
</tr>
</tbody>
</table>

i2c_deinit

The description of i2c_deinit is shown as below:

Table 3-327. Function i2c_deinit

| Function name | i2c_deinit |
Function prototype
void i2c_deinit(uint32_t i2c_periph);

Function descriptions
reset I2C

Precondition
-

The called functions
rcu_periph_reset_enable / rcu_periph_reset_disable

<table>
<thead>
<tr>
<th>Input parameter(in)</th>
</tr>
</thead>
<tbody>
<tr>
<td>i2c_periph</td>
</tr>
<tr>
<td>I2C peripheral</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Output parameter(out)</th>
</tr>
</thead>
<tbody>
<tr>
<td>-</td>
</tr>
</tbody>
</table>

Example:

/* reset I2C0 */

i2c_deinit(I2C0);

i2c_timing_config

The description of i2c_timing_config is shown as below:

**Table 3-328. Function i2c_timing_config**

<table>
<thead>
<tr>
<th>Function name</th>
<th>i2c_timing_config</th>
</tr>
</thead>
<tbody>
<tr>
<td>Function prototype</td>
<td>void i2c_timing_config(uint32_t i2c_periph, uint32_t psc, uint32_t scl_dely, uint32_t sda_dely);</td>
</tr>
<tr>
<td>Function descriptions</td>
<td>configure the timing parameters</td>
</tr>
<tr>
<td>Precondition</td>
<td>-</td>
</tr>
<tr>
<td>The called functions</td>
<td>-</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Input parameter(in)</th>
</tr>
</thead>
<tbody>
<tr>
<td>i2c_periph</td>
</tr>
<tr>
<td>I2C peripheral</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Input parameter(in)</th>
</tr>
</thead>
<tbody>
<tr>
<td>psc</td>
</tr>
<tr>
<td>0-0xf, timing prescaler</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Input parameter(in)</th>
</tr>
</thead>
<tbody>
<tr>
<td>scl_dely</td>
</tr>
<tr>
<td>0-0xf, data setup time</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Input parameter(in)</th>
</tr>
</thead>
<tbody>
<tr>
<td>sda_dely</td>
</tr>
<tr>
<td>0-0xf, data hold time</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Output parameter(out)</th>
</tr>
</thead>
<tbody>
<tr>
<td>-</td>
</tr>
</tbody>
</table>

Example:

/* configure the timing parameters */
i2c_timings_config(I2C0, 0x1, 0x2, 0x1);

i2c_digital_noise_filter_config

The description of i2c_digital_noise_filter_config is shown as below:

Table 3-329. Function i2c_digital_noise_filter_config

<table>
<thead>
<tr>
<th>Function name</th>
<th>i2c_digital_noise_filter_config</th>
</tr>
</thead>
<tbody>
<tr>
<td>Function prototype</td>
<td>void i2c_digital_noise_filter_config(uint32_t i2c_periph, uint32_t filter_length);</td>
</tr>
<tr>
<td>Function descriptions</td>
<td>configure digital noise filter</td>
</tr>
<tr>
<td>Precondition</td>
<td>-</td>
</tr>
<tr>
<td>The called functions</td>
<td>-</td>
</tr>
</tbody>
</table>

**Input parameter (in)**

<table>
<thead>
<tr>
<th>i2c_periph</th>
<th>I2C peripheral</th>
</tr>
</thead>
<tbody>
<tr>
<td>I2Cx</td>
<td>(x=0,1,2)</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>filter_length</th>
<th>filter_length</th>
</tr>
</thead>
<tbody>
<tr>
<td>FILTER_DISABLE</td>
<td>digital filter is disabled</td>
</tr>
<tr>
<td>FILTER_LENGTH_1</td>
<td>digital filter is enabled and filter spikes with a length of up to 1 tI2CCLK</td>
</tr>
<tr>
<td>FILTER_LENGTH_2</td>
<td>digital filter is enabled and filter spikes with a length of up to 2 tI2CCLK</td>
</tr>
<tr>
<td>FILTER_LENGTH_3</td>
<td>digital filter is enabled and filter spikes with a length of up to 3 tI2CCLK</td>
</tr>
<tr>
<td>FILTER_LENGTH_4</td>
<td>digital filter is enabled and filter spikes with a length of up to 4 tI2CCLK</td>
</tr>
<tr>
<td>FILTER_LENGTH_5</td>
<td>digital filter is enabled and filter spikes with a length of up to 5 tI2CCLK</td>
</tr>
<tr>
<td>FILTER_LENGTH_6</td>
<td>digital filter is enabled and filter spikes with a length of up to 6 tI2CCLK</td>
</tr>
<tr>
<td>FILTER_LENGTH_7</td>
<td>digital filter is enabled and filter spikes with a length of up to 7 tI2CCLK</td>
</tr>
<tr>
<td>FILTER_LENGTH_8</td>
<td>digital filter is enabled and filter spikes with a length of up to 8 tI2CCLK</td>
</tr>
<tr>
<td>FILTER_LENGTH_9</td>
<td>digital filter is enabled and filter spikes with a length of up to 9 tI2CCLK</td>
</tr>
<tr>
<td>FILTER_LENGTH_10</td>
<td>digital filter is enabled and filter spikes with a length of up to 10 tI2CCLK</td>
</tr>
<tr>
<td>FILTER_LENGTH_11</td>
<td>digital filter is enabled and filter spikes with a length of up to 11 tI2CCLK</td>
</tr>
<tr>
<td>FILTER_LENGTH_12</td>
<td>digital filter is enabled and filter spikes with a length of up to 12 tI2CCLK</td>
</tr>
<tr>
<td>FILTER_LENGTH_13</td>
<td>digital filter is enabled and filter spikes with a length of up to 13 tI2CCLK</td>
</tr>
<tr>
<td>FILTER_LENGTH_14</td>
<td>digital filter is enabled and filter spikes with a length of up to 14 tI2CCLK</td>
</tr>
<tr>
<td>FILTER_LENGTH_15</td>
<td>digital filter is enabled and filter spikes with a length of up to 15 tI2CCLK</td>
</tr>
</tbody>
</table>

**Output parameter (out)**

| - | - |

**Return value**

- -

Example:

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

i2c_digital_noise_filter_config(I2C0, FILTER_LENGTH_1);
The description of `i2c_analog_noise_filter_enable` is shown as below:

**Table 3-330. Function `i2c_analog_noise_filter_enable`**

<table>
<thead>
<tr>
<th>Function name</th>
<th><code>i2c_analog_noise_filter_enable</code></th>
</tr>
</thead>
<tbody>
<tr>
<td>Function prototype</td>
<td><code>void i2c_analog_noise_filter_enable(uint32_t i2c_periph);</code></td>
</tr>
<tr>
<td>Function descriptions</td>
<td>enable analog noise filter</td>
</tr>
<tr>
<td>Precondition</td>
<td>-</td>
</tr>
<tr>
<td>The called functions</td>
<td>-</td>
</tr>
</tbody>
</table>

**Input parameter (in)**

<table>
<thead>
<tr>
<th>i2c_periph</th>
<th>I2C peripheral</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>i2C</code> (x=0,1,2)</td>
<td></td>
</tr>
</tbody>
</table>

**Output parameter (out)**

| -                          | -              |

| Return value               | -              |

Example:

```c
/* enable analog noise filter */
i2c_analog_noise_filter_enable(I2C0);
```

The description of `i2c_analog_noise_filter_disable` is shown as below:

**Table 3-331. Function `i2c_analog_noise_filter_disable`**

<table>
<thead>
<tr>
<th>Function name</th>
<th><code>i2c_analog_noise_filter_disable</code></th>
</tr>
</thead>
<tbody>
<tr>
<td>Function prototype</td>
<td><code>void i2c_analog_noise_filter_disable(uint32_t i2c_periph);</code></td>
</tr>
<tr>
<td>Function descriptions</td>
<td>disable analog noise filter</td>
</tr>
<tr>
<td>Precondition</td>
<td>-</td>
</tr>
<tr>
<td>The called functions</td>
<td>-</td>
</tr>
</tbody>
</table>

**Input parameter (in)**

<table>
<thead>
<tr>
<th>i2c_periph</th>
<th>I2C peripheral</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>i2C</code> (x=0,1,2)</td>
<td></td>
</tr>
</tbody>
</table>

**Output parameter (out)**

| -                          | -              |

| Return value               | -              |

Example:

```c
/* disable analog noise filter */
i2c_analog_noise_filter_disable(I2C0);
```
i2c_master_clock_config

The description of i2c_master_clock_config is shown as below:

Table 3-332. Function i2c_master_clock_config

<table>
<thead>
<tr>
<th>Function name</th>
<th>i2c_master_clock_config</th>
</tr>
</thead>
<tbody>
<tr>
<td>Function prototype</td>
<td>void i2c_master_clock_config(uint32_t i2c_periph, uint32_t sclh, uint32_t scll);</td>
</tr>
<tr>
<td>Function descriptions</td>
<td>configure the SCL high and low period of clock in master mode</td>
</tr>
<tr>
<td>Precondition</td>
<td>-</td>
</tr>
<tr>
<td>The called functions</td>
<td>-</td>
</tr>
</tbody>
</table>

Input parameter (in)

<table>
<thead>
<tr>
<th>i2c_periph</th>
<th>I2C peripheral</th>
</tr>
</thead>
<tbody>
<tr>
<td>i2Cx</td>
<td>(x=0,1,2)</td>
</tr>
</tbody>
</table>

Input parameter (in)

<table>
<thead>
<tr>
<th>sclh</th>
<th>0-0xff, SCL high period</th>
</tr>
</thead>
</table>

Input parameter (in)

<table>
<thead>
<tr>
<th>scll</th>
<th>0-0xff, SCL low period</th>
</tr>
</thead>
</table>

Output parameter (out)

| -      | -                       |

Return value

Example:

/* configure the SCL and SDA period of clock in master mode */

i2c_master_clock_config(I2C0, 0x0f, 0x0f);

i2c_master_addressing

The description of i2c_master_addressing is shown as below:

Table 3-333. Function i2c_master_addressing

<table>
<thead>
<tr>
<th>Function name</th>
<th>i2c_master_addressing</th>
</tr>
</thead>
<tbody>
<tr>
<td>Function prototype</td>
<td>void i2c_master_addressing(uint32_t i2c_periph, uint32_t address, uint32_t trans_direction);</td>
</tr>
<tr>
<td>Function descriptions</td>
<td>configure i2c slave address and transfer direction in master mode</td>
</tr>
<tr>
<td>Precondition</td>
<td>-</td>
</tr>
<tr>
<td>The called functions</td>
<td>-</td>
</tr>
</tbody>
</table>

Input parameter (in)

<table>
<thead>
<tr>
<th>i2c_periph</th>
<th>I2C peripheral</th>
</tr>
</thead>
<tbody>
<tr>
<td>i2Cx</td>
<td>(x=0,1,2)</td>
</tr>
</tbody>
</table>

Input parameter (in)

<table>
<thead>
<tr>
<th>address</th>
<th>0-0x3FF except reserved address, I2C slave address to be sent</th>
</tr>
</thead>
</table>

Input parameter (in)
trans_direction | I2C transfer direction in master mode  
---|---
I2C_MASTER_TRANSMIT | master transmit  
I2C_MASTER_RECEIVE | master receive  

Output parameter(out)  
-  

Return value  
-  

Example:  
/* send slave address to I2C bus */  
i2c_master_addressing(I2C0, 0x82, I2C_MASTER_TRANSMIT);  

**i2c_address10_header_enable**  
The description of `i2c_address10_header_enable` is shown as below:  

<table>
<thead>
<tr>
<th>Function name</th>
<th>i2c_address10_header_enable</th>
</tr>
</thead>
<tbody>
<tr>
<td>Function prototype</td>
<td>void i2c_address10_header_enable(uint32_t i2c_periph);</td>
</tr>
<tr>
<td>Function descriptions</td>
<td>10-bit address header executes read direction only in master receive mode</td>
</tr>
<tr>
<td>Precondition</td>
<td>-</td>
</tr>
<tr>
<td>The called functions</td>
<td>-</td>
</tr>
<tr>
<td>Input parameter(in)</td>
<td></td>
</tr>
<tr>
<td>i2c_periph</td>
<td>I2C peripheral</td>
</tr>
<tr>
<td>I2Cx</td>
<td>(x=0,1,2)</td>
</tr>
<tr>
<td>Output parameter(out)</td>
<td>-</td>
</tr>
<tr>
<td>Return value</td>
<td>-</td>
</tr>
</tbody>
</table>

Example:  
/* 10-bit address header executes read direction only in master receive mode */  
i2c_address10_header_enable(I2C0);  

**i2c_address10_header_disable**  
The description of `i2c_address10_header_disable` is shown as below:  

<table>
<thead>
<tr>
<th>Function name</th>
<th>i2c_address10_header_disable</th>
</tr>
</thead>
<tbody>
<tr>
<td>Function prototype</td>
<td>void i2c_address10_header_disable(uint32_t i2c_periph);</td>
</tr>
<tr>
<td>Function descriptions</td>
<td>10-bit address header executes complete sequence in master receive mode</td>
</tr>
</tbody>
</table>
Precondition

The called functions

<table>
<thead>
<tr>
<th>Input parameter(in)</th>
</tr>
</thead>
<tbody>
<tr>
<td>i2c_periph</td>
</tr>
<tr>
<td>I2Cx</td>
</tr>
</tbody>
</table>

Output parameter(out)

Return value

Example:

/* 10-bit address header executes complete sequence in master receive mode */

i2c_address10_header_disable(I2C0);

i2c_address10_enable

The description of i2c_address10_enable is shown as below:

Table 3-336. Function i2c_address10_enable

<table>
<thead>
<tr>
<th>Function name</th>
<th>i2c_address10_enable</th>
</tr>
</thead>
<tbody>
<tr>
<td>Function prototype</td>
<td>void i2c_address10_enable(uint32_t i2c_periph);</td>
</tr>
<tr>
<td>Function descriptions</td>
<td>enable 10-bit addressing mode in master mode</td>
</tr>
</tbody>
</table>

Precondition

The called functions

Input parameter(in)

<table>
<thead>
<tr>
<th>i2c_periph</th>
<th>I2C peripheral</th>
</tr>
</thead>
<tbody>
<tr>
<td>I2Cx</td>
<td>(x=0,1,2)</td>
</tr>
</tbody>
</table>

Output parameter(out)

Return value

Example:

/* enable 10-bit addressing mode in master mode */

i2c_address10_enable(I2C0);

i2c_address10_disable

The description of i2c_address10_disable is shown as below:

Table 3-337. Function i2c_address10_disable

<table>
<thead>
<tr>
<th>Function name</th>
<th>i2c_address10_disable</th>
</tr>
</thead>
<tbody>
<tr>
<td>Function prototype</td>
<td>void i2c_address10_disable(uint32_t i2c_periph);</td>
</tr>
</tbody>
</table>
Function descriptions: disable 10-bit addressing mode in master mode

Precondition: -

The called functions: -

Input parameter (in):

<table>
<thead>
<tr>
<th>i2c_periph</th>
<th>I2C peripheral</th>
</tr>
</thead>
<tbody>
<tr>
<td>i2Cx</td>
<td>(x=0,1,2)</td>
</tr>
</tbody>
</table>

Output parameter (out):

- -

Return value:

- -

Example:

/* disable 10-bit addressing mode in master mode */

i2c_address10_disable(I2C0);

**i2c_automatic_end_enable**

The description of i2c_automatic_end_enable is shown as below:

Table 3-338. Function i2c_automatic_end_enable

<table>
<thead>
<tr>
<th>Function name</th>
<th>i2c_automatic_end_enable</th>
</tr>
</thead>
<tbody>
<tr>
<td>Function prototype</td>
<td>void i2c_automatic_end_enable(uint32_t i2c_periph);</td>
</tr>
<tr>
<td>Function descriptions</td>
<td>enable I2C automatic end mode in master mode</td>
</tr>
<tr>
<td>Precondition</td>
<td>-</td>
</tr>
<tr>
<td>The called functions</td>
<td>-</td>
</tr>
</tbody>
</table>

Input parameter (in):

<table>
<thead>
<tr>
<th>i2c_periph</th>
<th>I2C peripheral</th>
</tr>
</thead>
<tbody>
<tr>
<td>i2Cx</td>
<td>(x=0,1,2)</td>
</tr>
</tbody>
</table>

Output parameter (out):

- -

Return value:

- -

Example:

/* enable I2C automatic end mode in master mode */

i2c_automatic_end_enable(I2C0);

**i2c_automatic_end_disable**

The description of i2c_automatic_end_disable is shown as below:

Table 3-339. Function i2c_automatic_end_disable

<table>
<thead>
<tr>
<th>Function name</th>
<th>i2c_automatic_end_disable</th>
</tr>
</thead>
</table>
Function prototype
void i2c_automatic_end_disable(uint32_t i2c_periph);

Function descriptions
disable I2C automatic end mode in master mode

Precondition
-

The called functions
-

<table>
<thead>
<tr>
<th>Input parameter(in)</th>
</tr>
</thead>
<tbody>
<tr>
<td>i2c_periph</td>
</tr>
<tr>
<td>I2C peripheral</td>
</tr>
<tr>
<td>I2Cx</td>
</tr>
<tr>
<td>(x=0,1,2)</td>
</tr>
</tbody>
</table>

Output parameter(out)
-

Return value
-

Example:

/* disable I2C automatic end mode in master mode */
i2c_automatic_end_disable(I2C0);

i2c_slave_response_to_gcall_enable

The description of i2c_slave_response_to_gcall_enable is shown as below:

Table 3-340. Function i2c_slave_response_to_gcall_enable

<table>
<thead>
<tr>
<th>Function name</th>
<th>i2c_slave_response_to_gcall_enable</th>
</tr>
</thead>
<tbody>
<tr>
<td>Function prototype</td>
<td>void i2c_slave_response_to_gcall_enable(uint32_t i2c_periph);</td>
</tr>
<tr>
<td>Function descriptions</td>
<td>enable the response to a general call</td>
</tr>
<tr>
<td>Precondition</td>
<td>-</td>
</tr>
<tr>
<td>The called functions</td>
<td>-</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Input parameter(in)</th>
</tr>
</thead>
<tbody>
<tr>
<td>i2c_periph</td>
</tr>
<tr>
<td>I2C peripheral</td>
</tr>
<tr>
<td>I2Cx</td>
</tr>
<tr>
<td>(x=0,1,2)</td>
</tr>
</tbody>
</table>

Output parameter(out)
-

Return value
-

Example:

/* enable the response to a general call */
i2c_slave_response_to_gcall_enable(I2C0);

i2c_slave_response_to_gcall_disable

The description of i2c_slave_response_to_gcall_disable is shown as below:
Table 3-341. Function i2c_slave_response_to_gcall_disable

<table>
<thead>
<tr>
<th>Function name</th>
<th>i2c_slave_response_to_gcall_disable</th>
</tr>
</thead>
<tbody>
<tr>
<td>Function prototype</td>
<td>void i2c_slave_response_to_gcall_disable(uint32_t i2c_periph);</td>
</tr>
<tr>
<td>Function descriptions</td>
<td>disable the response to a general call</td>
</tr>
<tr>
<td>Precondition</td>
<td>-</td>
</tr>
<tr>
<td>The called functions</td>
<td>-</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Input parameter(in)</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>i2c_periph</td>
<td>I2C peripheral</td>
</tr>
<tr>
<td>i2Cx</td>
<td>(x=0,1,2)</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Output parameter(out)</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Return value</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Example:

/* disable the response to a general call */

i2c_slave_response_to_gcall_disable(I2C0);

**i2c_stretch_scl_low_enable**

The description of i2c_stretch_scl_low_enable is shown as below:

Table 3-342. Function i2c_stretch_scl_low_enable

<table>
<thead>
<tr>
<th>Function name</th>
<th>i2c_stretch_scl_low_enable</th>
</tr>
</thead>
<tbody>
<tr>
<td>Function prototype</td>
<td>void i2c_stretch_scl_low_enable(uint32_t i2c_periph);</td>
</tr>
<tr>
<td>Function descriptions</td>
<td>enable to stretch SCL low when data is not ready in slave mode</td>
</tr>
<tr>
<td>Precondition</td>
<td>-</td>
</tr>
<tr>
<td>The called functions</td>
<td>-</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Input parameter(in)</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>i2c_periph</td>
<td>I2C peripheral</td>
</tr>
<tr>
<td>i2Cx</td>
<td>(x=0,1,2)</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Output parameter(out)</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Return value</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Example:

/* enable to stretch SCL low when data is not ready in slave mode */

i2c_stretch_scl_low_enable(I2C0);

**i2c_stretch_scl_low_disable**

The description of i2c_stretch_scl_low_disable is shown as below:
Table 3-343. Function i2c_stretch_scl_low_disable

<table>
<thead>
<tr>
<th>Function name</th>
<th>i2c_stretch_scl_low_disable</th>
</tr>
</thead>
<tbody>
<tr>
<td>Function prototype</td>
<td>void i2c_stretch_scl_low_disable(uint32_t i2c_periph);</td>
</tr>
<tr>
<td>Function descriptions</td>
<td>disable to stretch SCL low when data is not ready in slave mode</td>
</tr>
<tr>
<td>Precondition</td>
<td>-</td>
</tr>
<tr>
<td>The called functions</td>
<td>-</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Input parameter (in)</th>
</tr>
</thead>
<tbody>
<tr>
<td>i2c_periph</td>
</tr>
<tr>
<td>i2c_periph (x=0,1,2)</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Output parameter (out)</th>
</tr>
</thead>
<tbody>
<tr>
<td>-</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Return value</th>
</tr>
</thead>
<tbody>
<tr>
<td>-</td>
</tr>
</tbody>
</table>

Example:
/* disable to stretch SCL low when data is not ready in slave mode */
i2c_stretch_scl_low_disable(I2C0);

i2c_address_config

The description of i2c_address_config is shown as below:

Table 3-344. Function i2c_address_config

<table>
<thead>
<tr>
<th>Function name</th>
<th>i2c_address_config</th>
</tr>
</thead>
<tbody>
<tr>
<td>Function prototype</td>
<td>void i2c_address_config(uint32_t i2c_periph, uint32_t address, uint32_t addr_format);</td>
</tr>
<tr>
<td>Function descriptions</td>
<td>configure i2c slave address</td>
</tr>
<tr>
<td>Precondition</td>
<td>-</td>
</tr>
<tr>
<td>The called functions</td>
<td>-</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Input parameter (in)</th>
</tr>
</thead>
<tbody>
<tr>
<td>i2c_periph</td>
</tr>
<tr>
<td>i2c_periph (x=0,1,2)</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>address</th>
<th>I2C address</th>
</tr>
</thead>
</table>

<table>
<thead>
<tr>
<th>addr_format</th>
<th>7bits or 10bits</th>
</tr>
</thead>
<tbody>
<tr>
<td>I2C_ADDFORMAT_7BITS</td>
<td>7bits</td>
</tr>
<tr>
<td>I2C_ADDFORMAT_10BITS</td>
<td>10bits</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Output parameter (out)</th>
</tr>
</thead>
<tbody>
<tr>
<td>-</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Return value</th>
</tr>
</thead>
<tbody>
<tr>
<td>-</td>
</tr>
</tbody>
</table>
Example:

/* configure I2C slave address */
i2c_address_config(I2C0, 0x82, I2C_ADDFORMAT_7BITS);

**i2c_address_bit_compare_config**

The description of `i2c_address_bit_compare_config` is shown as below:

**Table 3-345. Function i2c_address_bit_compare_config**

<table>
<thead>
<tr>
<th>Function name</th>
<th>i2c_address_bit_compare_config</th>
</tr>
</thead>
<tbody>
<tr>
<td>Function prototype</td>
<td>void i2c_address_bit_compare_config(uint32_t i2c_periph, uint32_t compare_bits);</td>
</tr>
<tr>
<td>Function descriptions</td>
<td>define which bits of ADDRESS[7:1] need to compare with the incoming address byte</td>
</tr>
<tr>
<td>Precondition</td>
<td>-</td>
</tr>
<tr>
<td>The called functions</td>
<td>-</td>
</tr>
</tbody>
</table>

**Input parameter (in)**

<table>
<thead>
<tr>
<th>i2c_periph</th>
<th>I2C peripheral</th>
</tr>
</thead>
<tbody>
<tr>
<td>I2Cx</td>
<td>(x=0,1,2)</td>
</tr>
</tbody>
</table>

**Input parameter (in)**

<table>
<thead>
<tr>
<th>compare_bits</th>
<th>the bits need to compare</th>
</tr>
</thead>
<tbody>
<tr>
<td>ADDRESS_BIT1_COMPARE</td>
<td>address bit1 needs compare</td>
</tr>
<tr>
<td>ADDRESS_BIT2_COMPARE</td>
<td>address bit2 needs compare</td>
</tr>
<tr>
<td>ADDRESS_BIT3_COMPARE</td>
<td>address bit3 needs compare</td>
</tr>
<tr>
<td>ADDRESS_BIT4_COMPARE</td>
<td>address bit4 needs compare</td>
</tr>
<tr>
<td>ADDRESS_BIT5_COMPARE</td>
<td>address bit5 needs compare</td>
</tr>
<tr>
<td>ADDRESS_BIT6_COMPARE</td>
<td>address bit6 needs compare</td>
</tr>
<tr>
<td>ADDRESS_BIT7_COMPARE</td>
<td>address bit7 needs compare</td>
</tr>
</tbody>
</table>

**Output parameter (out)**

| - | - |

**Return value**

| - | - |
/* bit 1 of ADDRESS[7:1] need to compare with the incoming address byte */

i2c_address_bit_compare_config(I2C0, ADDRESS_BIT1COMPARE);

**i2c_address_disable**

The description of `i2c_address_disable` is shown as below:

**Table 3-346. Function i2c_address_disable**

<table>
<thead>
<tr>
<th>Function name</th>
<th>i2c_address_disable</th>
</tr>
</thead>
<tbody>
<tr>
<td>Function prototype</td>
<td>void i2c_address_disable(uint32_t i2c_periph);</td>
</tr>
<tr>
<td>Function descriptions</td>
<td>disable i2c address in slave mode</td>
</tr>
<tr>
<td>Precondition</td>
<td>-</td>
</tr>
<tr>
<td>The called functions</td>
<td>-</td>
</tr>
<tr>
<td>Input parameter(in)</td>
<td></td>
</tr>
<tr>
<td>i2c_periph</td>
<td>I2C peripheral</td>
</tr>
<tr>
<td>i2Cx</td>
<td>(x=0,1,2)</td>
</tr>
<tr>
<td>Output parameter(out)</td>
<td>-</td>
</tr>
<tr>
<td>Return value</td>
<td>-</td>
</tr>
</tbody>
</table>

Example:

/* disable i2c address in slave mode */

i2c_address_disable(I2C0);

**i2c_second_address_config**

The description of `i2c_second_address_config` is shown as below:

**Table 3-347. Function i2c_second_address_config**

<table>
<thead>
<tr>
<th>Function name</th>
<th>i2c_second_address_config</th>
</tr>
</thead>
<tbody>
<tr>
<td>Function prototype</td>
<td>void i2c_second_address_config(uint32_t i2c_periph, uint32_t address, uint32_t addr_mask);</td>
</tr>
<tr>
<td>Function descriptions</td>
<td>configure i2c second slave address</td>
</tr>
<tr>
<td>Precondition</td>
<td>-</td>
</tr>
<tr>
<td>The called functions</td>
<td>-</td>
</tr>
<tr>
<td>Input parameter(in)</td>
<td></td>
</tr>
<tr>
<td>i2c_periph</td>
<td>I2C peripheral</td>
</tr>
<tr>
<td>i2Cx</td>
<td>(x=0,1,2)</td>
</tr>
<tr>
<td>address</td>
<td>I2C address</td>
</tr>
<tr>
<td>addr_mask</td>
<td>the bits not need to compare</td>
</tr>
</tbody>
</table>
### ADDRESS2_NO_MASK
- No mask, all the bits must be compared

### ADDRESS2_MASK_BIT
- ADDRESS2[1] is masked, only ADDRESS2[7:2] are compared
- ADDRESS2[2:1] is masked, only ADDRESS2[7:3] are compared
- ADDRESS2[3:1] is masked, only ADDRESS2[7:4] are compared
- ADDRESS2[4:1] is masked, only ADDRESS2[7:5] are compared
- ADDRESS2[5:1] is masked, only ADDRESS2[7:6] are compared
- ADDRESS2[6:1] is masked, only ADDRESS2[7] are compared
- All the ADDRESS2[7:1] bits are masked

<table>
<thead>
<tr>
<th>Output parameter(out)</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>-</td>
</tr>
</tbody>
</table>

| Return value         | -|

Example:

```c
/* configure i2c second slave address */
i2c_second_address_config(I2C0, 0x82, ADDRESS2_MASK_BIT1_2);
```

### i2c_second_address_disable

The description of `i2c_second_address_disable` is shown as below:

#### Table 3-348. Function i2c_second_address_disable

<table>
<thead>
<tr>
<th>Function name</th>
<th>i2c_second_address_disable</th>
</tr>
</thead>
<tbody>
<tr>
<td>Function prototype</td>
<td>void i2c_second_address_disable(uint32_t i2c_periph);</td>
</tr>
<tr>
<td>Function descriptions</td>
<td>disable i2c second address in slave mode</td>
</tr>
<tr>
<td>Precondition</td>
<td>-</td>
</tr>
<tr>
<td>The called functions</td>
<td>-</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Input parameter(in)</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>i2c_periph</td>
<td>I2C peripheral</td>
</tr>
<tr>
<td>i2Cx</td>
<td>(x=0,1,2)</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Output parameter(out)</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>-</td>
</tr>
</tbody>
</table>

| Return value            | -                        |

Example:
i2c_second_address_disable(I2C0);

i2c_recevied_address_get

The description of i2c_recevied_address_get is shown as below:

**Table 3-349. Function i2c_recevied_address_get**

<table>
<thead>
<tr>
<th>Function name</th>
<th>i2c_recevied_address_get</th>
</tr>
</thead>
<tbody>
<tr>
<td>Function prototype</td>
<td>uint32_t i2c_recevied_address_get(uint32_t i2c_periph);</td>
</tr>
<tr>
<td>Function descriptions</td>
<td>get received match address in slave mode</td>
</tr>
<tr>
<td>Precondition</td>
<td>-</td>
</tr>
<tr>
<td>The called functions</td>
<td>-</td>
</tr>
<tr>
<td>Input parameter[in]</td>
<td></td>
</tr>
<tr>
<td>i2c_periph</td>
<td>I2C peripheral</td>
</tr>
<tr>
<td>i2C0</td>
<td>(x=0,1,2)</td>
</tr>
<tr>
<td>Input parameter[out]</td>
<td></td>
</tr>
<tr>
<td>Return value</td>
<td></td>
</tr>
<tr>
<td>uint32_t</td>
<td>0x00..0x7F</td>
</tr>
</tbody>
</table>

Example:

/\* get received match address in slave mode */

uint32_t address;

address = i2c_recevied_address_get(I2C0);

i2c_slave_byte_control_enable

The description of i2c_slave_byte_control_enable is shown as below:

**Table 3-350. Function i2c_slave_byte_control_enable**

<table>
<thead>
<tr>
<th>Function name</th>
<th>i2c_slave_byte_control_enable</th>
</tr>
</thead>
<tbody>
<tr>
<td>Function prototype</td>
<td>void i2c_slave_byte_control_enable(uint32_t i2c_periph);</td>
</tr>
<tr>
<td>Function descriptions</td>
<td>enable slave byte control</td>
</tr>
<tr>
<td>Precondition</td>
<td>-</td>
</tr>
<tr>
<td>The called functions</td>
<td>-</td>
</tr>
<tr>
<td>Input parameter[in]</td>
<td></td>
</tr>
<tr>
<td>i2c_periph</td>
<td>I2C peripheral</td>
</tr>
<tr>
<td>i2C0</td>
<td>(x=0,1,2)</td>
</tr>
<tr>
<td>Output parameter[out]</td>
<td></td>
</tr>
<tr>
<td>Return value</td>
<td></td>
</tr>
</tbody>
</table>
Example:

/* enable slave byte control */

i2c_slave_byte_control_enable(I2C0);

### i2c_slave_byte_control_disable

The description of `i2c_slave_byte_control_disable` is shown as below:

<table>
<thead>
<tr>
<th>Table 3-351. Function <code>i2c_slave_byte_control_disable</code></th>
</tr>
</thead>
<tbody>
<tr>
<td>Function name</td>
</tr>
<tr>
<td>Function prototype</td>
</tr>
<tr>
<td>Function descriptions</td>
</tr>
<tr>
<td>Precondition</td>
</tr>
<tr>
<td>The called functions</td>
</tr>
<tr>
<td>Input parameter(in)</td>
</tr>
<tr>
<td>i2c_periph</td>
</tr>
<tr>
<td>i2Cx</td>
</tr>
<tr>
<td>Output parameter(out)</td>
</tr>
<tr>
<td>-</td>
</tr>
<tr>
<td>Return value</td>
</tr>
</tbody>
</table>

Example:

/* disable slave byte control */

i2c_slave_byte_control_disable(I2C0);

### i2c_nack_enable

The description of `i2c_nack_enable` is shown as below:

<table>
<thead>
<tr>
<th>Table 3-352. Function <code>i2c_nack_enable</code></th>
</tr>
</thead>
<tbody>
<tr>
<td>Function name</td>
</tr>
<tr>
<td>Function prototype</td>
</tr>
<tr>
<td>Function descriptions</td>
</tr>
<tr>
<td>Precondition</td>
</tr>
<tr>
<td>The called functions</td>
</tr>
<tr>
<td>Input parameter(in)</td>
</tr>
<tr>
<td>i2c_periph</td>
</tr>
<tr>
<td>i2Cx</td>
</tr>
<tr>
<td>Output parameter(out)</td>
</tr>
<tr>
<td>-</td>
</tr>
<tr>
<td>Return value</td>
</tr>
</tbody>
</table>
Example:

/* generate a NACK in slave mode */
i2c_nack_enable(I2C0);

i2c_nack_disable

The description of i2c_nack_disable is shown as below:

Table 3-353. Function i2c_nack_disable

<table>
<thead>
<tr>
<th>Function name</th>
<th>i2c_nack_disable</th>
</tr>
</thead>
<tbody>
<tr>
<td>Function prototype</td>
<td>void i2c_nack_disable(uint32_t i2c_periph);</td>
</tr>
<tr>
<td>Function descriptions</td>
<td>generate a ACK in slave mode</td>
</tr>
<tr>
<td>Precondition</td>
<td>-</td>
</tr>
<tr>
<td>The called functions</td>
<td>-</td>
</tr>
<tr>
<td>Input parameter(in)</td>
<td></td>
</tr>
<tr>
<td>i2c_periph</td>
<td>I2C peripheral</td>
</tr>
<tr>
<td>I2Cx</td>
<td>(x=0,1,2)</td>
</tr>
<tr>
<td>Output parameter(out)</td>
<td>-</td>
</tr>
<tr>
<td>Return value</td>
<td>-</td>
</tr>
</tbody>
</table>

Example:

/* generate a ACK in slave mode */
i2c_nack_disable(I2C0);

i2c_wakeup_from_deepsleep_enable

The description of i2c_wakeup_from_deepsleep_enable is shown as below:

Table 3-354. Function i2c_wakeup_from_deepsleep_enable

<table>
<thead>
<tr>
<th>Function name</th>
<th>i2c_wakeup_from_deepsleep_enable</th>
</tr>
</thead>
<tbody>
<tr>
<td>Function prototype</td>
<td>void i2c_wakeup_from_deepsleep_enable(uint32_t i2c_periph);</td>
</tr>
<tr>
<td>Function descriptions</td>
<td>enable wakeup from Deep-sleep mode</td>
</tr>
<tr>
<td>Precondition</td>
<td>-</td>
</tr>
<tr>
<td>The called functions</td>
<td>-</td>
</tr>
<tr>
<td>Input parameter(in)</td>
<td></td>
</tr>
<tr>
<td>i2c_periph</td>
<td>I2C peripheral</td>
</tr>
<tr>
<td>I2Cx</td>
<td>(x=0)</td>
</tr>
<tr>
<td>Output parameter(out)</td>
<td>-</td>
</tr>
<tr>
<td>Return value</td>
<td>-</td>
</tr>
</tbody>
</table>
Example:

/* enable wakeup from Deep-sleep mode */
i2c_wakeup_from_deepsleep_enable(I2C0);

**i2c_wakeup_from_deepsleep_disable**

The description of i2c_wakeup_from_deepsleep_disable is shown as below:

**Table 3-355. Function i2c_wakeup_from_deepsleep_disable**

<table>
<thead>
<tr>
<th>Function name</th>
<th>i2c_wakeup_from_deepsleep_disable</th>
</tr>
</thead>
<tbody>
<tr>
<td>Function prototype</td>
<td>void i2c_wakeup_from_deepsleep_disable(uint32_t i2c_periph);</td>
</tr>
<tr>
<td>Function descriptions</td>
<td>disable wakeup from Deep-sleep mode</td>
</tr>
<tr>
<td>Precondition</td>
<td>-</td>
</tr>
<tr>
<td>The called functions</td>
<td>-</td>
</tr>
</tbody>
</table>

**Input parameter(in)**

<table>
<thead>
<tr>
<th>i2c_periph</th>
<th>I2C peripheral</th>
</tr>
</thead>
<tbody>
<tr>
<td>I2Cx</td>
<td>(x=0)</td>
</tr>
</tbody>
</table>

**Output parameter(out)**

| - | - |

**Return value**

| - | - |

Example:

/* disable wakeup from Deep-sleep mode */
i2c_wakeup_from_deepsleep_disable(I2C0);

**i2c_enable**

The description of i2c_enable is shown as below:

**Table 3-356. Function i2c_enable**

<table>
<thead>
<tr>
<th>Function name</th>
<th>i2c_enable</th>
</tr>
</thead>
<tbody>
<tr>
<td>Function prototype</td>
<td>void i2c_enable(uint32_t i2c_periph);</td>
</tr>
<tr>
<td>Function descriptions</td>
<td>enable I2C</td>
</tr>
<tr>
<td>Precondition</td>
<td>-</td>
</tr>
<tr>
<td>The called functions</td>
<td>-</td>
</tr>
</tbody>
</table>

**Input parameter(in)**

<table>
<thead>
<tr>
<th>i2c_periph</th>
<th>I2C peripheral</th>
</tr>
</thead>
<tbody>
<tr>
<td>I2Cx</td>
<td>(x=0,1,2)</td>
</tr>
</tbody>
</table>

**Output parameter(out)**

| - | - |

**Return value**

| - | - |
Example:

/* enable I2C0 */
i2c_enable(I2C0);

**i2c_disable**

The description of i2c_disable is shown as below:

<table>
<thead>
<tr>
<th>Function name</th>
<th>i2c_disable</th>
</tr>
</thead>
<tbody>
<tr>
<td>Function prototype</td>
<td>void i2c_disable(uint32_t i2c_periph);</td>
</tr>
<tr>
<td>Function descriptions</td>
<td>disable I2C</td>
</tr>
<tr>
<td>Precondition</td>
<td>-</td>
</tr>
<tr>
<td>The called functions</td>
<td>-</td>
</tr>
</tbody>
</table>

**Input parameter (in)**

<table>
<thead>
<tr>
<th>i2c_periph</th>
<th>I2C peripheral</th>
</tr>
</thead>
<tbody>
<tr>
<td>I2Cx</td>
<td>(x=0,1,2)</td>
</tr>
</tbody>
</table>

**Output parameter (out)**

| - | - |

**Return value**

- -

Example:

/* disable I2C0 */
i2c_disable(I2C0);

**i2c_start_on_bus**

The description of i2c_start_on_bus is shown as below:

<table>
<thead>
<tr>
<th>Function name</th>
<th>i2c_start_on_bus</th>
</tr>
</thead>
<tbody>
<tr>
<td>Function prototype</td>
<td>void i2c_start_on_bus(uint32_t i2c_periph);</td>
</tr>
<tr>
<td>Function descriptions</td>
<td>generate a START condition on I2C bus</td>
</tr>
<tr>
<td>Precondition</td>
<td>-</td>
</tr>
<tr>
<td>The called functions</td>
<td>-</td>
</tr>
</tbody>
</table>

**Input parameter (in)**

<table>
<thead>
<tr>
<th>i2c_periph</th>
<th>I2C peripheral</th>
</tr>
</thead>
<tbody>
<tr>
<td>I2Cx</td>
<td>(x=0,1,2)</td>
</tr>
</tbody>
</table>

**Output parameter (out)**

| - | - |

**Return value**

- -

247
Example:

/* I2C0 send a start condition to I2C bus */
i2c_start_on_bus(I2C0);

### i2c_stop_on_bus

The description of `i2c_stop_on_bus` is shown as below:

<table>
<thead>
<tr>
<th>Function name</th>
<th>i2c_stop_on_bus</th>
</tr>
</thead>
<tbody>
<tr>
<td>Function prototype</td>
<td>void i2c_stop_on_bus(uint32_t i2c_periph);</td>
</tr>
<tr>
<td>Function descriptions</td>
<td>generate a STOP condition on I2C bus</td>
</tr>
<tr>
<td>Precondition</td>
<td>-</td>
</tr>
<tr>
<td>The called functions</td>
<td>-</td>
</tr>
</tbody>
</table>

**Input parameter (in)**

<table>
<thead>
<tr>
<th>i2c_periph</th>
<th>I2C peripheral</th>
</tr>
</thead>
<tbody>
<tr>
<td>I2Cx</td>
<td>(x=0,1,2)</td>
</tr>
</tbody>
</table>

**Output parameter (out)**

| - | - |

**Return value**

| - | - |

Example:

/* I2C0 generate a STOP condition to I2C bus */
i2c_stop_on_bus(I2C0);

### i2c_data_transmit

The description of `i2c_data_transmit` is shown as below:

<table>
<thead>
<tr>
<th>Function name</th>
<th>i2c_data_transmit</th>
</tr>
</thead>
<tbody>
<tr>
<td>Function prototype</td>
<td>void i2c_data_transmit(uint32_t i2c_periph, uint32_t data);</td>
</tr>
<tr>
<td>Function descriptions</td>
<td>I2C transmit data</td>
</tr>
<tr>
<td>Precondition</td>
<td>-</td>
</tr>
<tr>
<td>The called functions</td>
<td>-</td>
</tr>
</tbody>
</table>

**Input parameter (in)**

<table>
<thead>
<tr>
<th>i2c_periph</th>
<th>I2C peripheral</th>
</tr>
</thead>
<tbody>
<tr>
<td>I2Cx</td>
<td>(x=0,1,2)</td>
</tr>
</tbody>
</table>

**Input parameter (in)**

<table>
<thead>
<tr>
<th>data</th>
<th>transmit data</th>
</tr>
</thead>
</table>

**Output parameter (out)**

| - | - |
Example:

/* I2C0 transmit data */
i2c_data_transmit(I2C0, 0x80);

i2c_data_receive

The description of i2c_data_receive is shown as below:

<table>
<thead>
<tr>
<th>Table 3-361. Function i2c_data_receive</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Function name</strong></td>
</tr>
<tr>
<td><strong>Function prototype</strong></td>
</tr>
<tr>
<td><strong>Function descriptions</strong></td>
</tr>
<tr>
<td><strong>Precondition</strong></td>
</tr>
<tr>
<td><strong>The called functions</strong></td>
</tr>
<tr>
<td><strong>Input parameter</strong></td>
</tr>
<tr>
<td>i2c_periph</td>
</tr>
<tr>
<td>I2Cx</td>
</tr>
<tr>
<td><strong>Output parameter</strong></td>
</tr>
<tr>
<td>-</td>
</tr>
<tr>
<td><strong>Return value</strong></td>
</tr>
<tr>
<td>uint32_t</td>
</tr>
</tbody>
</table>

Example:

/* I2C0 receive data */
uint32_t i2c_receiver;
i2c_receiver = i2c_data_receive(I2C0);

i2c_reload_enable

The description of i2c_reload_enable is shown as below:

<table>
<thead>
<tr>
<th>Table 3-362. Function i2c_reload_enable</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Function name</strong></td>
</tr>
<tr>
<td><strong>Function prototype</strong></td>
</tr>
<tr>
<td><strong>Function descriptions</strong></td>
</tr>
<tr>
<td><strong>Precondition</strong></td>
</tr>
<tr>
<td><strong>The called functions</strong></td>
</tr>
<tr>
<td><strong>Input parameter</strong></td>
</tr>
<tr>
<td>i2c_periph</td>
</tr>
<tr>
<td>I2Cx</td>
</tr>
</tbody>
</table>
Output parameter(out)  
- -  

Return value - -  

Example:  
/* enable I2C reload mode */  
i2c_reload_enable(I2C0);  

**i2c_reload_disable**  
The description of `i2c_reload_disable` is shown as below:  

<table>
<thead>
<tr>
<th>Function name</th>
<th>i2c_reload_disable</th>
</tr>
</thead>
<tbody>
<tr>
<td>Function prototype</td>
<td>void i2c_reload_disable(uint32_t i2c_periph);</td>
</tr>
<tr>
<td>Function descriptions</td>
<td>disable I2C reload mode</td>
</tr>
<tr>
<td>Precondition</td>
<td>-</td>
</tr>
<tr>
<td>The called functions</td>
<td>-</td>
</tr>
<tr>
<td>Input parameter(in)</td>
<td></td>
</tr>
<tr>
<td>i2c_periph</td>
<td>I2C peripheral</td>
</tr>
<tr>
<td>I2Cx</td>
<td>(x=0,1,2)</td>
</tr>
<tr>
<td>Output parameter(out)</td>
<td>-</td>
</tr>
<tr>
<td>Return value</td>
<td>-</td>
</tr>
</tbody>
</table>

Example:  
/* disable I2C reload mode */  
i2c_reload_disable(I2C0);  

**i2c_transfer_byte_number_config**  
The description of `i2c_transfer_byte_number_config` is shown as below:  

<table>
<thead>
<tr>
<th>Function name</th>
<th>i2c_transfer_byte_number_config</th>
</tr>
</thead>
<tbody>
<tr>
<td>Function prototype</td>
<td>void i2c_transfer_byte_number_config(uint32_t i2c_periph, uint32_t byte_number);</td>
</tr>
<tr>
<td>Function descriptions</td>
<td>configure number of bytes to be transferred</td>
</tr>
<tr>
<td>Precondition</td>
<td>-</td>
</tr>
<tr>
<td>The called functions</td>
<td>-</td>
</tr>
<tr>
<td>Input parameter(in)</td>
<td></td>
</tr>
</tbody>
</table>
### i2c_periph

<table>
<thead>
<tr>
<th>i2c_periph</th>
<th>I2C peripheral</th>
</tr>
</thead>
<tbody>
<tr>
<td>I2Cx</td>
<td>(x=0,1,2)</td>
</tr>
</tbody>
</table>

**Input parameter (in)**

| byte_number | 0x0-0xFF, number of bytes to be transferred |

**Output parameter (out)**

| -           | -                                           |

**Return value**

| -           | -                                           |

**Example:**

```c
/* configure number of bytes to be transferred */
i2c_transfer_byte_number_config(I2C0, 0xFF);
```

### i2c_dma_enable

The description of `i2c_dma_enable` is shown as below:

**Table 3-365. Function i2c_dma_enable**

<table>
<thead>
<tr>
<th>Function name</th>
<th>i2c_dma_enable</th>
</tr>
</thead>
<tbody>
<tr>
<td>Function prototype</td>
<td>void i2c_dma_enable(uint32_t i2c_periph, uint8_t dma);</td>
</tr>
<tr>
<td>Function descriptions</td>
<td>enable I2C DMA for transmission or reception</td>
</tr>
<tr>
<td>Precondition</td>
<td>-</td>
</tr>
<tr>
<td>The called functions</td>
<td>-</td>
</tr>
</tbody>
</table>

**Input parameter (in)**

<table>
<thead>
<tr>
<th>i2c_periph</th>
<th>I2C peripheral</th>
</tr>
</thead>
<tbody>
<tr>
<td>I2Cx</td>
<td>(x=0,1,2)</td>
</tr>
</tbody>
</table>

**Input parameter (in)**

<table>
<thead>
<tr>
<th>dma</th>
<th>I2C DMA</th>
</tr>
</thead>
<tbody>
<tr>
<td>I2C_DMA_TRANSMIT</td>
<td>transmit data using DMA</td>
</tr>
<tr>
<td>I2C_DMA_RECEIVE</td>
<td>receive data using DMA</td>
</tr>
</tbody>
</table>

**Output parameter (out)**

| -           | -                                           |

**Return value**

| -           | -                                           |

**Example:**

```c
/* enable I2C DMA for transmission or reception */
i2c_dma_enable(I2C0, I2C_DMA_RECEIVE);
```

### i2c_dma_disable

The description of `i2c_dma_disable` is shown as below:
### Table 3-366. Function i2c_dma_disable

<table>
<thead>
<tr>
<th>Function name</th>
<th>i2c_dma_disable</th>
</tr>
</thead>
<tbody>
<tr>
<td>Function prototype</td>
<td>void i2c_dma_disable(uint32_t i2c_periph, uint8_t dma);</td>
</tr>
<tr>
<td>Function descriptions</td>
<td>disable I2C DMA for transmission or reception</td>
</tr>
</tbody>
</table>

**Precondition**
- 

**The called functions**
- 

**Input parameter (in)**

<table>
<thead>
<tr>
<th>i2c_periph</th>
<th>I2C peripheral</th>
</tr>
</thead>
<tbody>
<tr>
<td>i2Cx</td>
<td>(x=0,1,2)</td>
</tr>
</tbody>
</table>

**Input parameter (in)**

<table>
<thead>
<tr>
<th>dma</th>
<th>I2C DMA</th>
</tr>
</thead>
<tbody>
<tr>
<td>I2C_DMA_TRANSMIT</td>
<td>transmit data using DMA</td>
</tr>
<tr>
<td>I2C_DMA_RECEIVE</td>
<td>receive data using DMA</td>
</tr>
</tbody>
</table>

**Output parameter (out)**
- 

**Return value**
- 

Example:

```c
/* disable I2C DMA for transmission or reception */

i2c_dma_disable(I2C0, I2C_DMA_RECEIVE);
```

### i2c_pec_transfer

The description of i2c_pec_transfer is shown as below:

**Function prototype**

```c
void i2c_pec_transfer(uint32_t i2c_periph);
```

**Function descriptions**

I2C transfers PEC value

**Precondition**
- 

**The called functions**
- 

**Input parameter (in)**

<table>
<thead>
<tr>
<th>i2c_periph</th>
<th>I2C peripheral</th>
</tr>
</thead>
<tbody>
<tr>
<td>i2Cx</td>
<td>(x=0,1,2)</td>
</tr>
</tbody>
</table>

**Output parameter (out)**
- 

**Return value**
- 

Example:

```c
/* I2C transfers PEC value */

i2c_pec_transfer(I2C0);
```
i2c_pec_enable

The description of i2c_pec_enable is shown as below:

Table 3-368. Function i2c_pec_enable

<table>
<thead>
<tr>
<th>Function name</th>
<th>i2c_pec_enable</th>
</tr>
</thead>
<tbody>
<tr>
<td>Function prototype</td>
<td>void i2c_pec_enable(uint32_t i2c_periph);</td>
</tr>
<tr>
<td>Function descriptions</td>
<td>enable I2C PEC calculation</td>
</tr>
<tr>
<td>Precondition</td>
<td>-</td>
</tr>
<tr>
<td>The called functions</td>
<td>-</td>
</tr>
</tbody>
</table>

Input parameter(in)

<table>
<thead>
<tr>
<th>i2c_periph</th>
<th>I2C peripheral</th>
</tr>
</thead>
<tbody>
<tr>
<td>I2Cx</td>
<td>(x=0,1,2)</td>
</tr>
</tbody>
</table>

Output parameter(out)

- -

Return value

- -

Example:

/* enable I2C PEC calculation */
i2c_pec_enable(I2C0);

i2c_pec_disable

The description of i2c_pec_disable is shown as below:

Table 3-369. Function i2c_pec_disable

<table>
<thead>
<tr>
<th>Function name</th>
<th>i2c_pec_disable</th>
</tr>
</thead>
<tbody>
<tr>
<td>Function prototype</td>
<td>void i2c_pec_disable(uint32_t i2c_periph);</td>
</tr>
<tr>
<td>Function descriptions</td>
<td>disable I2C PEC calculation</td>
</tr>
<tr>
<td>Precondition</td>
<td>-</td>
</tr>
<tr>
<td>The called functions</td>
<td>-</td>
</tr>
</tbody>
</table>

Input parameter(in)

<table>
<thead>
<tr>
<th>i2c_periph</th>
<th>I2C peripheral</th>
</tr>
</thead>
<tbody>
<tr>
<td>I2Cx</td>
<td>(x=0,1,2)</td>
</tr>
</tbody>
</table>

Output parameter(out)

- -

Return value

- -

Example:

/* disable I2C PEC calculation */
i2c_pec_disable(I2C0);
i2c_pec_value_get

The description of i2c_pec_value_get is shown as below:

Table 3-370. Function i2c_pec_value_get

<table>
<thead>
<tr>
<th>Function name</th>
<th>i2c_pec_value_get</th>
</tr>
</thead>
<tbody>
<tr>
<td>Function prototype</td>
<td>uint32_t i2c_pec_value_get(uint32_t i2c_periph);</td>
</tr>
<tr>
<td>Function descriptions</td>
<td>get packet error checking value</td>
</tr>
<tr>
<td>Precondition</td>
<td>-</td>
</tr>
<tr>
<td>The called functions</td>
<td>-</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Input parameter (in)</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>i2c_periph</td>
<td>I2C peripheral I2Cx (x=0,1,2)</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Output parameter (out)</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>-</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Return value</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>uint32_t</td>
<td>PEC value</td>
</tr>
</tbody>
</table>

Example:

/* I2C0 get packet error checking value */

uint32_t pec_value;
pec_value = i2c_pec_value_get(I2C0);

i2c_smbus_alert_enable

The description of i2c_smbus_alert_enable is shown as below:

Table 3-371. Function i2c_smbus_alert_enable

<table>
<thead>
<tr>
<th>Function name</th>
<th>i2c_smbus_alert_enable</th>
</tr>
</thead>
<tbody>
<tr>
<td>Function prototype</td>
<td>void i2c_smbus_alert_enable(uint32_t i2c_periph);</td>
</tr>
<tr>
<td>Function descriptions</td>
<td>enable SMBus Alert</td>
</tr>
<tr>
<td>Precondition</td>
<td>-</td>
</tr>
<tr>
<td>The called functions</td>
<td>-</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Input parameter (in)</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>i2c_periph</td>
<td>I2C peripheral I2Cx (x=0,1,2)</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Output parameter (out)</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>-</td>
<td></td>
</tr>
</tbody>
</table>

Return value

Example:

/* enable SMBus Alert */
i2c_smbus_alert_disable

The description of `i2c_smbus_alert_disable` is shown as below:

### Table 3-372. Function `i2c_smbus_alert_disable`

| Function name                  | i2c_smbus_alert_disable
|-------------------------------|------------------
| Function prototype            | `void i2c_smbus_alert_disable(uint32_t i2c_periph);`
| Function descriptions         | disable SMBus Alert
| Precondition                  | -                
| The called functions          | -                

#### Input parameter (in)

<table>
<thead>
<tr>
<th>i2c_periph</th>
<th>I2C peripheral</th>
</tr>
</thead>
<tbody>
<tr>
<td>I2Cx</td>
<td>(x=0,1,2)</td>
</tr>
</tbody>
</table>

#### Output parameter (out)

- 

#### Return value

- 

Example:

```c
/* disable SMBus Alert */
i2c_smbus_alert_disable(I2C0);
```

### i2c_smbus_default_addr_enable

The description of `i2c_smbus_default_addr_enable` is shown as below:

### Table 3-373. Function `i2c_smbus_default_addr_enable`

<table>
<thead>
<tr>
<th>Function name</th>
<th>i2c_smbus_default_addr_enable</th>
</tr>
</thead>
</table>
| Function prototype            | `void i2c_smbus_default_addr_enable(uint32_t i2c_periph);`
| Function descriptions         | enable SMBus device default address |
| Precondition                  | -                             |
| The called functions          | -                             |

#### Input parameter (in)

<table>
<thead>
<tr>
<th>i2c_periph</th>
<th>I2C peripheral</th>
</tr>
</thead>
<tbody>
<tr>
<td>I2Cx</td>
<td>(x=0,1,2)</td>
</tr>
</tbody>
</table>

#### Output parameter (out)

- 

#### Return value

- 

Example:

```c
/* enable SMBus device default address */
i2c_smbus_default_addr_enable(I2C0);
```
i2c_smbus_default_addr_enable(I2C0);

i2c_smbus_default_addr_disable

The description of i2c_smbus_default_addr_disable is shown as below:

Table 3-374. Function i2c_smbus_default_addr_disable

<table>
<thead>
<tr>
<th>Function name</th>
<th>i2c_smbus_default_addr_disable</th>
</tr>
</thead>
<tbody>
<tr>
<td>Function prototype</td>
<td>void i2c_smbus_default_addr_disable(uint32_t i2c_periph);</td>
</tr>
<tr>
<td>Function descriptions</td>
<td>disable SMBus device default address</td>
</tr>
<tr>
<td>Precondition</td>
<td>-</td>
</tr>
<tr>
<td>The called functions</td>
<td>-</td>
</tr>
<tr>
<td>Input parameter(in)</td>
<td></td>
</tr>
<tr>
<td>i2c_periph</td>
<td>I2C peripheral</td>
</tr>
<tr>
<td>i2Cx</td>
<td>(x = 0, 1, 2)</td>
</tr>
<tr>
<td>Output parameter(out)</td>
<td></td>
</tr>
<tr>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>Return value</td>
<td>-</td>
</tr>
</tbody>
</table>

Example:

/* disable SMBus device default address */
i2c_smbus_default_addr_disable(I2C0);

i2c_smbus_host_addr_enable

The description of i2c_smbus_host_addr_enable is shown as below:

Table 3-375. Function i2c_smbus_host_addr_enable

<table>
<thead>
<tr>
<th>Function name</th>
<th>i2c_smbus_host_addr_enable</th>
</tr>
</thead>
<tbody>
<tr>
<td>Function prototype</td>
<td>void i2c_smbus_host_addr_enable(uint32_t i2c_periph);</td>
</tr>
<tr>
<td>Function descriptions</td>
<td>enable SMBus Host address</td>
</tr>
<tr>
<td>Precondition</td>
<td>-</td>
</tr>
<tr>
<td>The called functions</td>
<td>-</td>
</tr>
<tr>
<td>Input parameter(in)</td>
<td></td>
</tr>
<tr>
<td>i2c_periph</td>
<td>I2C peripheral</td>
</tr>
<tr>
<td>i2Cx</td>
<td>(x = 0, 1, 2)</td>
</tr>
<tr>
<td>Output parameter(out)</td>
<td></td>
</tr>
<tr>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>Return value</td>
<td>-</td>
</tr>
</tbody>
</table>

Example:

/* enable SMBus Host address */
i2c_smbus_host_addr_enable(I2C0);

**i2c_smbus_host_addr_disable**

The description of `i2c_smbus_host_addr_disable` is shown as below:

### Table 3-376. Function i2c_smbus_host_addr_disable

<table>
<thead>
<tr>
<th>Function name</th>
<th>i2c_smbus_host_addr_disable</th>
</tr>
</thead>
<tbody>
<tr>
<td>Function prototype</td>
<td>void i2c_smbus_host_addr_disable(uint32_t i2c_periph);</td>
</tr>
<tr>
<td>Function descriptions</td>
<td>disable SMBus Host address</td>
</tr>
<tr>
<td>Precondition</td>
<td>-</td>
</tr>
<tr>
<td>The called functions</td>
<td>-</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Input parameter (in)</th>
</tr>
</thead>
<tbody>
<tr>
<td>i2c_periph</td>
</tr>
<tr>
<td>i2Cx (x=0,1,2)</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Output parameter (out)</th>
</tr>
</thead>
<tbody>
<tr>
<td>-</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Return value</th>
</tr>
</thead>
<tbody>
<tr>
<td>-</td>
</tr>
</tbody>
</table>

Example:

```c
/* disable SMBus Host address */

i2c_smbus_host_addr_disable(I2C0);
```

**i2c_extented_clock_timeout_enable**

The description of `i2c_extented_clock_timeout_enable` is shown as below:

### Table 3-377. Function i2c_extented_clock_timeout_enable

<table>
<thead>
<tr>
<th>Function name</th>
<th>i2c_extented_clock_timeout_enable</th>
</tr>
</thead>
<tbody>
<tr>
<td>Function prototype</td>
<td>void i2c_extented_clock_timeout_enable(uint32_t i2c_periph);</td>
</tr>
<tr>
<td>Function descriptions</td>
<td>enable extended clock timeout detection</td>
</tr>
<tr>
<td>Precondition</td>
<td>-</td>
</tr>
<tr>
<td>The called functions</td>
<td>-</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Input parameter (in)</th>
</tr>
</thead>
<tbody>
<tr>
<td>i2c_periph</td>
</tr>
<tr>
<td>i2Cx (x=0,1,2)</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Output parameter (out)</th>
</tr>
</thead>
<tbody>
<tr>
<td>-</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Return value</th>
</tr>
</thead>
<tbody>
<tr>
<td>-</td>
</tr>
</tbody>
</table>

Example:

```c
/* enable extended clock timeout detection */

i2c_extented_clock_timeout_enable(I2C0);
```
i2c_extented_clock_timeout_enable(I2C0);

i2c_extented_clock_timeout_disable

The description of i2c_extented_clock_timeout_disable is shown as below:

**Table 3-378. Function i2c_extented_clock_timeout_disable**

<table>
<thead>
<tr>
<th>Function name</th>
<th>i2c_extented_clock_timeout_disable</th>
</tr>
</thead>
<tbody>
<tr>
<td>Function prototype</td>
<td>void i2c_extented_clock_timeout_disable(uint32_t i2c_periph);</td>
</tr>
<tr>
<td>Function descriptions</td>
<td>disable extended clock timeout detection</td>
</tr>
<tr>
<td>Precondition</td>
<td>-</td>
</tr>
<tr>
<td>The called functions</td>
<td>-</td>
</tr>
</tbody>
</table>

**Input parameter(in)**

<table>
<thead>
<tr>
<th>i2c_periph</th>
<th>I2C peripheral</th>
</tr>
</thead>
<tbody>
<tr>
<td>i2Cx</td>
<td>(x=0,1,2)</td>
</tr>
</tbody>
</table>

**Output parameter(out)**

- -

**Return value**

- -

Example:

/* disable extended clock timeout detection */

i2c_extented_clock_timeout_disable(I2C0);

i2c_clock_timeout_enable

The description of i2c_clock_timeout_enable is shown as below:

**Table 3-379. Function i2c_clock_timeout_enable**

<table>
<thead>
<tr>
<th>Function name</th>
<th>i2c_clock_timeout_enable</th>
</tr>
</thead>
<tbody>
<tr>
<td>Function prototype</td>
<td>void i2c_clock_timeout_enable(uint32_t i2c_periph);</td>
</tr>
<tr>
<td>Function descriptions</td>
<td>enable clock timeout detection</td>
</tr>
<tr>
<td>Precondition</td>
<td>-</td>
</tr>
<tr>
<td>The called functions</td>
<td>-</td>
</tr>
</tbody>
</table>

**Input parameter(in)**

<table>
<thead>
<tr>
<th>i2c_periph</th>
<th>I2C peripheral</th>
</tr>
</thead>
<tbody>
<tr>
<td>i2Cx</td>
<td>(x=0,1,2)</td>
</tr>
</tbody>
</table>

**Output parameter(out)**

- -

**Return value**

- -

Example:

/* enable clock timeout detection */
i2c_clock_timeout_enable(I2C0);

i2c_clock_timeout_disable

The description of i2c_clock_timeout_disable is shown as below:

Table 3-380. Function i2c_clock_timeout_disable

<table>
<thead>
<tr>
<th>Function name</th>
<th>i2c_clock_timeout_disable</th>
</tr>
</thead>
<tbody>
<tr>
<td>Function prototype</td>
<td>void i2c_clock_timeout_disable(uint32_t i2c_periph);</td>
</tr>
<tr>
<td>Function descriptions</td>
<td>disable clock timeout detection</td>
</tr>
<tr>
<td>Precondition</td>
<td>-</td>
</tr>
<tr>
<td>The called functions</td>
<td>-</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Input parameter(in)</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>i2c_periph</td>
<td>I2C peripheral</td>
</tr>
<tr>
<td>i2Cx</td>
<td>(x=0,1,2)</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Output parameter(out)</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>-</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Return value</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>-</td>
<td></td>
</tr>
</tbody>
</table>

Example:

/* disable clock timeout detection */

i2c_clock_timeout_disable(I2C0);

i2c_bus_timeout_b_config

The description of i2c_bus_timeout_b_config is shown as below:

Table 3-381. Function i2c_bus_timeout_b_config

<table>
<thead>
<tr>
<th>Function name</th>
<th>i2c_bus_timeout_b_config</th>
</tr>
</thead>
<tbody>
<tr>
<td>Function prototype</td>
<td>void i2c_bus_timeout_b_config(uint32_t i2c_periph, uint32_t timeout);</td>
</tr>
<tr>
<td>Function descriptions</td>
<td>configure bus timeout B</td>
</tr>
<tr>
<td>Precondition</td>
<td>-</td>
</tr>
<tr>
<td>The called functions</td>
<td>-</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Input parameter(in)</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>i2c_periph</td>
<td>I2C peripheral</td>
</tr>
<tr>
<td>i2Cx</td>
<td>(x=0,1,2)</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Input parameter(in)</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>timeout</td>
<td>0-0xffff, bus timeout B</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Output parameter(out)</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>-</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Return value</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>-</td>
<td></td>
</tr>
</tbody>
</table>

Example:
/* configure bus timeout B */

i2c_bus_timeout_b_config(I2C0, 0xff);

**i2c_bus_timeout_a_config**

The description of **i2c_bus_timeout_a_config** is shown as below:

<table>
<thead>
<tr>
<th>Table 3-382. Function i2c_bus_timeout_a_config</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Function name</strong></td>
</tr>
<tr>
<td><strong>Function prototype</strong></td>
</tr>
<tr>
<td><strong>Function descriptions</strong></td>
</tr>
<tr>
<td><strong>Precondition</strong></td>
</tr>
<tr>
<td><strong>The called functions</strong></td>
</tr>
<tr>
<td><strong>Input parameter(in)</strong></td>
</tr>
<tr>
<td>i2c_periph</td>
</tr>
<tr>
<td>i2Cx</td>
</tr>
<tr>
<td><strong>Input parameter(in)</strong></td>
</tr>
<tr>
<td>timeout</td>
</tr>
<tr>
<td><strong>Output parameter(out)</strong></td>
</tr>
<tr>
<td><strong>Return value</strong></td>
</tr>
</tbody>
</table>

Example:

/* configure bus timeout A */

i2c_bus_timeout_a_config(I2C0, 0xff);

**i2c_idle_clock_timeout_config**

The description of **i2c_idle_clock_timeout_config** is shown as below:

<table>
<thead>
<tr>
<th>Table 3-383. Function i2c_idle_clock_timeout_config</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Function name</strong></td>
</tr>
<tr>
<td><strong>Function prototype</strong></td>
</tr>
<tr>
<td><strong>Function descriptions</strong></td>
</tr>
<tr>
<td><strong>Precondition</strong></td>
</tr>
<tr>
<td><strong>The called functions</strong></td>
</tr>
<tr>
<td><strong>Input parameter(in)</strong></td>
</tr>
<tr>
<td>i2c_periph</td>
</tr>
<tr>
<td>i2Cx</td>
</tr>
<tr>
<td><strong>Input parameter(in)</strong></td>
</tr>
<tr>
<td>timeout</td>
</tr>
<tr>
<td><strong>BUSTOA_DETECT_SC</strong></td>
</tr>
</tbody>
</table>
BUSTOA is used to detect both SCL and SDA high timeout when the bus is idle.

<table>
<thead>
<tr>
<th>Output parameter{out}</th>
</tr>
</thead>
<tbody>
<tr>
<td>-</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Return value</th>
</tr>
</thead>
<tbody>
<tr>
<td>-</td>
</tr>
</tbody>
</table>

Example:

/* configure idle clock timeout detection */

i2c_idle_clock_timeout_config(I2C0, BUSTOA_DETECT_SCL_LOW);

### i2c_flag_get

The description of `i2c_flag_get` is shown as below:

<table>
<thead>
<tr>
<th>Function name</th>
<th>i2c_flag_get</th>
</tr>
</thead>
<tbody>
<tr>
<td>Function prototype</td>
<td>FlagStatus i2c_flag_get(uint32_t i2c_periph, uint32_t flag);</td>
</tr>
<tr>
<td>Function descriptions</td>
<td>get I2C flag status</td>
</tr>
<tr>
<td>Precondition</td>
<td>-</td>
</tr>
<tr>
<td>The called functions</td>
<td>-</td>
</tr>
</tbody>
</table>

#### Input parameter{in}

<table>
<thead>
<tr>
<th>i2c_periph</th>
<th>I2C peripheral</th>
</tr>
</thead>
<tbody>
<tr>
<td>I2Cx(x=0,1,2)</td>
<td></td>
</tr>
</tbody>
</table>

#### Input parameter{in}

<table>
<thead>
<tr>
<th>flag</th>
<th>I2C flags</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>I2C_FLAG_TBE</code></td>
<td>I2C_TDATA is empty during transmitting</td>
</tr>
<tr>
<td><code>I2C_FLAG_TI</code></td>
<td>transmit interrupt</td>
</tr>
<tr>
<td><code>I2C_FLAG_RBNE</code></td>
<td>I2C_RDATA is not empty during receiving</td>
</tr>
<tr>
<td><code>I2C_FLAG_ADDSEND</code></td>
<td>address received matches in slave mode</td>
</tr>
<tr>
<td><code>I2C_FLAG_NACK</code></td>
<td>not acknowledge flag</td>
</tr>
<tr>
<td><code>I2C_FLAG_STPDET</code></td>
<td>STOP condition detected in slave mode</td>
</tr>
<tr>
<td><code>I2C_FLAG_TC</code></td>
<td>transfer complete in master mode</td>
</tr>
<tr>
<td><code>I2C_FLAG_TCR</code></td>
<td>transfer complete reload</td>
</tr>
<tr>
<td><code>I2C_FLAG_BERR</code></td>
<td>bus error</td>
</tr>
<tr>
<td><code>I2C_FLAG_LOSTARB</code></td>
<td>arbitration Lost</td>
</tr>
<tr>
<td><code>I2C_FLAG_OUERR</code></td>
<td>overrun/underrun error in slave mode</td>
</tr>
<tr>
<td><code>I2C_FLAG_PECERR</code></td>
<td>PEC error</td>
</tr>
<tr>
<td><code>I2C_FLAG_TIMEOUT</code></td>
<td>timeout flag</td>
</tr>
<tr>
<td><code>I2C_FLAG_SMBALT</code></td>
<td>SMBus Alert</td>
</tr>
<tr>
<td><code>I2C_FLAG_I2CBSY</code></td>
<td>busy flag</td>
</tr>
<tr>
<td><code>I2C_FLAG_TR</code></td>
<td>whether the I2C is a transmitter or a receiver in slave mode</td>
</tr>
</tbody>
</table>
### Output parameter(out)

<p>| |</p>
<table>
<thead>
<tr>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>-</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Return value</th>
</tr>
</thead>
<tbody>
<tr>
<td>FlagStatus</td>
</tr>
<tr>
<td>SET / RESET</td>
</tr>
</tbody>
</table>

Example:

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

FlagStatus flag_state = RESET;

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

### i2c_flag_clear

The description of i2c_flag_clear is shown as below:

**Table 3-385. Function i2c_flag_clear**

<table>
<thead>
<tr>
<th>Function name</th>
<th>i2c_flag_clear</th>
</tr>
</thead>
<tbody>
<tr>
<td>Function prototype</td>
<td>void i2c_flag_clear(uint32_t i2c_periph, uint32_t flag);</td>
</tr>
<tr>
<td>Function descriptions</td>
<td>clear I2C flag status</td>
</tr>
<tr>
<td>Precondition</td>
<td>-</td>
</tr>
<tr>
<td>The called functions</td>
<td>-</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Input parameter(in)</th>
</tr>
</thead>
<tbody>
<tr>
<td>i2c_periph</td>
</tr>
<tr>
<td>i2C peripheral</td>
</tr>
<tr>
<td>i2Cx</td>
</tr>
<tr>
<td>(x=0,1,2)</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Input parameter(in)</th>
</tr>
</thead>
<tbody>
<tr>
<td>flag</td>
</tr>
<tr>
<td>I2C flags</td>
</tr>
<tr>
<td>I2C_FLAG_ADDSEND</td>
</tr>
<tr>
<td>I2C_FLAG_NACK</td>
</tr>
<tr>
<td>I2C_FLAG_STPDET</td>
</tr>
<tr>
<td>I2C_FLAG_BERR</td>
</tr>
<tr>
<td>I2C_FLAG_LOSTARB</td>
</tr>
<tr>
<td>I2C_FLAG_OUERR</td>
</tr>
<tr>
<td>I2C_FLAG_PECERR</td>
</tr>
<tr>
<td>I2C_FLAG_TIMEOUT</td>
</tr>
<tr>
<td>I2C_FLAG_SMBALT</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Output parameter(out)</th>
</tr>
</thead>
<tbody>
<tr>
<td>-</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Return value</th>
</tr>
</thead>
<tbody>
<tr>
<td>-</td>
</tr>
</tbody>
</table>

Example:

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

i2c_flag_clear(I2C0, I2C_FLAG_BERR);
```
i2c_interrupt_enable

The description of i2c_interrupt_enable is shown as below:

Table 3-386. Function i2c_interrupt_enable

<table>
<thead>
<tr>
<th>Function name</th>
<th>i2c_interrupt_enable</th>
</tr>
</thead>
<tbody>
<tr>
<td>Function prototype</td>
<td>void i2c_interrupt_enable(uint32_t i2c_periph, uint32_t interrupt);</td>
</tr>
<tr>
<td>Function descriptions</td>
<td>enable I2C interrupt</td>
</tr>
<tr>
<td>Precondition</td>
<td>-</td>
</tr>
<tr>
<td>The called functions</td>
<td>-</td>
</tr>
</tbody>
</table>

Input parameter(in)

<table>
<thead>
<tr>
<th>i2c_periph</th>
<th>I2C peripheral</th>
</tr>
</thead>
<tbody>
<tr>
<td>I2Cx</td>
<td>(x=0,1,2)</td>
</tr>
</tbody>
</table>

Output parameter(out)

- Return value -

Example:

/* enable I2C0 transmit interrupt */
i2c_interrupt_enable(I2C0, I2C_INT_TI);

i2c_interrupt_disable

The description of i2c_interrupt_disable is shown as below:

Table 3-387. Function i2c_interrupt_disable

<table>
<thead>
<tr>
<th>Function name</th>
<th>i2c_interrupt_disable</th>
</tr>
</thead>
<tbody>
<tr>
<td>Function prototype</td>
<td>void i2c_interrupt_disable(uint32_t i2c_periph, uint32_t interrupt);</td>
</tr>
<tr>
<td>Function descriptions</td>
<td>disable I2C interrupt</td>
</tr>
<tr>
<td>Precondition</td>
<td>-</td>
</tr>
<tr>
<td>The called functions</td>
<td>-</td>
</tr>
</tbody>
</table>

Input parameter(in)

<table>
<thead>
<tr>
<th>i2c_periph</th>
<th>I2C peripheral</th>
</tr>
</thead>
<tbody>
<tr>
<td>I2Cx</td>
<td>(x=0,1,2)</td>
</tr>
</tbody>
</table>
### GD32L23x Firmware Library User Guide

**Input parameter(in)**

<table>
<thead>
<tr>
<th>interrupt</th>
<th>I2C interrupts</th>
</tr>
</thead>
<tbody>
<tr>
<td>I2C_INT_ERR</td>
<td>error interrupt</td>
</tr>
<tr>
<td>I2C_INT_TC</td>
<td>transfer complete interrupt</td>
</tr>
<tr>
<td>I2C_INT_STPDET</td>
<td>stop detection interrupt</td>
</tr>
<tr>
<td>I2C_INT_NACK</td>
<td>not acknowledge received interrupt</td>
</tr>
<tr>
<td>I2C_INT_ADDRM</td>
<td>address match interrupt</td>
</tr>
<tr>
<td>I2C_INT_RBNE</td>
<td>receive interrupt</td>
</tr>
<tr>
<td>I2C_INT_TI</td>
<td>transmit interrupt</td>
</tr>
</tbody>
</table>

**Output parameter(out)**

| -                  | -                              |

**Return value**

| -                  | -                              |

Example:

```
/* disable I2C0 transmit interrupt */
i2c_interrupt_disable(I2C0, I2C_INT_TI);
```

**i2c_interrupt_flag_get**

The description of i2c_interrupt_flag_get is shown as below:

**Table 3-388. Function i2c_interrupt_flag_get**

<table>
<thead>
<tr>
<th>Function name</th>
<th>i2c_interrupt_flag_get</th>
</tr>
</thead>
<tbody>
<tr>
<td>Function prototype</td>
<td>FlagStatus i2c_interrupt_flag_get(uint32_t i2c_periph, i2c_interrupt_flag_enum int_flag);</td>
</tr>
<tr>
<td>Function descriptions</td>
<td>get I2C interrupt flag status</td>
</tr>
<tr>
<td>Precondition</td>
<td>-</td>
</tr>
<tr>
<td>The called functions</td>
<td>-</td>
</tr>
</tbody>
</table>

**Input parameter(in)**

- **i2c_periph**
  - I2C peripheral
  - I2Cx (x=0,1,2)

**Input parameter(in)**

- **int_flag**
  - I2C interrupt flags, refer to **Table 3-326. i2c_interrupt_flag_enum**
  - I2C_INT_FLAG_TI: transmit interrupt flag
  - I2C_INT_FLAG_RBNE: I2C_RDATA is not empty during receiving interrupt flag
  - I2C_INT_FLAG_ADDS: address received matches in slave mode interrupt flag
  - I2C_INT_FLAG_NACK: not acknowledge interrupt flag
  - I2C_INT_FLAG_STPD: stop condition detected in slave mode interrupt flag
  - I2C_INT_FLAG_TC: transfer complete in master mode interrupt flag
  - I2C_INT_FLAG_TCR: transfer complete reload interrupt flag
### GD32L23x Firmware Library User Guide

**I2C_INT_FLAG_BERR**  
bus error interrupt flag

**I2C_INT_FLAG_LOSTARB**  
arbitration lost interrupt flag

**I2C_INT_FLAG_OUER**  
overrun/underrun error in slave mode interrupt flag

**I2C_INT_FLAG_PECERR**  
PEC error interrupt flag

**I2C_INT_FLAG_TIMEOUT**  
timeout interrupt flag

**I2C_INT_FLAG_SMBA**  
SMBus Alert interrupt flag

<table>
<thead>
<tr>
<th>Output parameter(out)</th>
<th>-</th>
</tr>
</thead>
<tbody>
<tr>
<td>Return value</td>
<td>FlagStatus: SET / RESET</td>
</tr>
</tbody>
</table>

Example:

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

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

### i2c_interrupt_flag_clear

The description of `i2c_interrupt_flag_clear` is shown as below:

**Table 3-389. Function i2c_interrupt_flag_clear**

<table>
<thead>
<tr>
<th>Function name</th>
<th>i2c_interrupt_flag_clear</th>
</tr>
</thead>
<tbody>
<tr>
<td>Function prototype</td>
<td>void i2c_interrupt_flag_clear(uint32_t i2c_periph, i2c_interrupt_flag_enum int_flag);</td>
</tr>
<tr>
<td>Function descriptions</td>
<td>clear I2C interrupt flag status</td>
</tr>
<tr>
<td>Precondition</td>
<td>-</td>
</tr>
<tr>
<td>The called functions</td>
<td>-</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Input parameter(in)</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>i2c_periph</td>
<td>I2C peripheral</td>
</tr>
<tr>
<td>i2Cx</td>
<td>(x=0,1,2)</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Input parameter(in)</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>int_flag</td>
<td>I2C interrupt flags, refer to Table 3-326, i2c_interrupt_flag_enum.</td>
</tr>
<tr>
<td>I2C_INT_FLAG_ADDSEND</td>
<td>address received matches in slave mode interrupt flag</td>
</tr>
<tr>
<td>I2C_INT_FLAG_NACK</td>
<td>not acknowledge interrupt flag</td>
</tr>
<tr>
<td>I2C_INT_FLAG_STPDET</td>
<td>stop condition detected in slave mode interrupt flag</td>
</tr>
<tr>
<td>I2C_INT_FLAG_BERR</td>
<td>bus error interrupt flag</td>
</tr>
</tbody>
</table>
I2C_INT_FLAG_LOSTARB arbitration lost interrupt flag
I2C_INT_FLAG_OUERR overrun/underrun error in slave mode interrupt flag
I2C_INT_FLAG_PECERR PEC error interrupt flag
I2C_INT_FLAG_TIMEOUT timeout interrupt flag
I2C_INT_FLAG_SMBALERT SMBus Alert interrupt flag

Output parameter(out)
- -

Return value
- -

Example:
/* clear a bus error flag */
i2c_interrupt_flag_clear(I2C0, I2C_INT_FLAG_BERR);

3.15. LPTIMER

The LPTIMER is a 32-bit timer and it is able to keep running in all power modes except for Standby mode with its diversity of clock sources. The LPTIMER registers are listed in chapter 3.15.1, the LPTIMER firmware functions are introduced in chapter 3.15.2.

3.15.1. Descriptions of Peripheral registers

LPTIMER registers are listed in the table shown as below:

<table>
<thead>
<tr>
<th>Registers</th>
<th>Descriptions</th>
</tr>
</thead>
<tbody>
<tr>
<td>LPTIMER_INTF</td>
<td>interrupt flag register</td>
</tr>
<tr>
<td>LPTIMER_INTC</td>
<td>interrupt flag clear register</td>
</tr>
<tr>
<td>LPTIMER_INTEN</td>
<td>interrupt enable register</td>
</tr>
<tr>
<td>LPTIMER_CTL0</td>
<td>control register 0</td>
</tr>
<tr>
<td>LPTIMER_CTL1</td>
<td>control register 1</td>
</tr>
<tr>
<td>LPTIMER_CMPV</td>
<td>compare value register</td>
</tr>
<tr>
<td>LPTIMER_CAR</td>
<td>counter auto reload register</td>
</tr>
<tr>
<td>LPTIMER_CNT</td>
<td>counter register</td>
</tr>
<tr>
<td>LPTIMER_EIRMP</td>
<td>external input remap register</td>
</tr>
<tr>
<td>LPTIMER_INHLCMV</td>
<td>input high level counter max value register</td>
</tr>
</tbody>
</table>
3.15.2. Descriptions of Peripheral functions

The description format of firmware functions are shown as below:

Table 3-391. LPTIMER firmware function

<table>
<thead>
<tr>
<th>Function name</th>
<th>Function description</th>
</tr>
</thead>
<tbody>
<tr>
<td>lptimer_deinit</td>
<td>deinit the LPTIMER</td>
</tr>
<tr>
<td>lptimer_structpara_init</td>
<td>initialize LPTIMER init parameter struct with a default value</td>
</tr>
<tr>
<td>lptimer_init</td>
<td>initialize LPTIMER counter</td>
</tr>
<tr>
<td>lptimer_inputremap</td>
<td>configure external input remap</td>
</tr>
<tr>
<td>lptimer_register_shadow_enable</td>
<td>enable the LPTIMER_CAR and LPTIMER_CMPV registers shadow function</td>
</tr>
<tr>
<td>lptimer_register_shadow_disable</td>
<td>disable the LPTIMER_CAR and LPTIMER_CMPV registers shadow function</td>
</tr>
<tr>
<td>lptimer_timeout_enable</td>
<td>enable the LPTIMER TIMEOUT function</td>
</tr>
<tr>
<td>lptimer_timeout_disable</td>
<td>disable the LPTIMER TIMEOUT function</td>
</tr>
<tr>
<td>lptimer_countinue_start</td>
<td>LPTIMER start with countinue mode</td>
</tr>
<tr>
<td>lptimer_single_start</td>
<td>LPTIMER start with single mode</td>
</tr>
<tr>
<td>lptimer_stop</td>
<td>stop LPTIMER</td>
</tr>
<tr>
<td>lptimer_counter_read</td>
<td>read LPTIMER current counter value</td>
</tr>
<tr>
<td>lptimer_autoreload_read</td>
<td>read LPTIMER auto reload value</td>
</tr>
<tr>
<td>lptimer_compare_read</td>
<td>read LPTIMER compare value</td>
</tr>
<tr>
<td>lptimer_autoreload_value_config</td>
<td>configure LPTIMER autoreload register value</td>
</tr>
<tr>
<td>lptimer_compare_value_config</td>
<td>configure LPTIMER compare value</td>
</tr>
<tr>
<td>lptimer_decodemode0_enable</td>
<td>enable decode mode 0</td>
</tr>
<tr>
<td>lptimer_decodemode1_enable</td>
<td>enable decode mode 1</td>
</tr>
<tr>
<td>lptimer_decodemode_disable</td>
<td>disable decode mode 0/1</td>
</tr>
<tr>
<td>lptimer_highlevelcounter_enable</td>
<td>enable external input high level counter</td>
</tr>
<tr>
<td>lptimer_highlevelcounter_disable</td>
<td>disable external input high level counter</td>
</tr>
<tr>
<td>lptimer_flag_get</td>
<td>get LPTIMER flags</td>
</tr>
<tr>
<td>lptimer_flag_clear</td>
<td>clear LPTIMER flags</td>
</tr>
<tr>
<td>lptimer_interrupt_enable</td>
<td>enable the LPTIMER interrupt</td>
</tr>
<tr>
<td>lptimer_interrupt_disable</td>
<td>disable the LPTIMER interrupt</td>
</tr>
<tr>
<td>lptimer_interrupt_flag_get</td>
<td>get LPTIMER interrupt flag</td>
</tr>
<tr>
<td>lptimer_interrupt_flag_clear</td>
<td>clear LPTIMER interrupt flag</td>
</tr>
</tbody>
</table>

Structure lptimer_parameter_struct

Table 3-392. Structure lptimer_parameter_struct

<table>
<thead>
<tr>
<th>Member name</th>
<th>Function description</th>
</tr>
</thead>
<tbody>
<tr>
<td>clocksource</td>
<td>clock source (LPTIMER_INTERNALCLK, LPTIMER_EXTERNALCLK)</td>
</tr>
<tr>
<td>prescaler</td>
<td>counter clock prescaler (LPTIMER_PSC_x, x=1,2,4,8..128)</td>
</tr>
<tr>
<td>extclockpolarity</td>
<td>external clock polarity of the active edge for the counter</td>
</tr>
</tbody>
</table>
### Member name | Function description
---|---
| extclockfilter | external clock sampling time to configure the clock glitch filter (LPTIMER_EXTERNALCLK_FILTEROFF, LPTIMER_EXTERNALCLK_FILTER_2, LPTIMER_EXTERNALCLK_FILTER_4, LPTIMER_EXTERNALCLK_FILTER_8)
| triggermode | trigger mode (LPTIMER_TRIGGER_SOFTWARE, LPTIMER_TRIGGER_EXTERNALRISING, LPTIMER_TRIGGEREXTERNALFALLING, LPTIMER_TRIGGER_EXTERNALBOTH)
| extriggersource | external trigger source (LPTIMER_EXTRIGGER_GPIO, LPTIMER_EXTRIGGER_RTCALARM0, LPTIMER_EXTRIGGER_RTCALARM1, LPTIMER_EXTRIGGER_RTCTAMP0, LPTIMER_EXTRIGGER_RTCTAMP1, LPTIMER_EXTRIGGER_RTCTAMP2, LPTIMER_EXTRIGGER_CMP0_OUT, LPTIMER_EXTRIGGER_CMP1_OUT)
| extriggerfilter | external trigger filter (LPTIMER_TRIGGER_FILTEROFF, LPTIMER_TRIGGER_FILTER_2, LPTIMER_TRIGGER_FILTER_4, LPTIMER_TRIGGER_FILTER_8)
| outputpolarity | output polarity (LPTIMER_OUTPUT_NOTINVERTED, LPTIMER_OUTPUT_INVERTED)
| outputmode | output mode (LPTIMER_OUTPUT_PWMORSINGLE, LPTIMER_OUTPUT_SET)
| countersource | counter source (LPTIMER_COUNTER_INTERNAL, LPTIMER_COUNTER_EXTERNAL)

### lptimer_deinit

The description of lptimer_deinit is shown as below:

Table 3-393. Function lptimer_deinit

<table>
<thead>
<tr>
<th>Function name</th>
<th>lptimer_deinit</th>
</tr>
</thead>
<tbody>
<tr>
<td>Function prototype</td>
<td>void lptimer_deinit(void);</td>
</tr>
<tr>
<td>Function descriptions</td>
<td>deinit LPTIMER</td>
</tr>
<tr>
<td>Precondition</td>
<td>-</td>
</tr>
<tr>
<td>The called functions</td>
<td>rcu_periph_reset_enable / rcu_periph_reset_disable</td>
</tr>
<tr>
<td>Input parameter(in)</td>
<td>-</td>
</tr>
<tr>
<td>Output parameter(out)</td>
<td>-</td>
</tr>
<tr>
<td>Return value</td>
<td>-</td>
</tr>
</tbody>
</table>
Example:

```c
/* deinit LPTIMER */
lptimer_deinit();
```

**lptimer_struct_para_init**

The description of `lptimer_struct_para_init` is shown as below:

**Table 3-394. Function lptimer_struct_para_init**

<table>
<thead>
<tr>
<th>Function name</th>
<th>lptimer_struct_para_init</th>
</tr>
</thead>
<tbody>
<tr>
<td>Function prototype</td>
<td>void lptimer_struct_para_init(lptimer_parameter_struct *initpara);</td>
</tr>
<tr>
<td>Function descriptions</td>
<td>initialize LPTIMER init parameter struct with a default value</td>
</tr>
<tr>
<td>Precondition</td>
<td>-</td>
</tr>
<tr>
<td>The called functions</td>
<td>-</td>
</tr>
</tbody>
</table>

**Input parameter(in)**

- **initpara**: LPTIMER init parameter struct, the structure members can refer to Table 3-392, Structure lptimer_parameter_struct.

**Output parameter(out)**

- -

**Return value**

- -

Example:

```c
/* initialize LPTIMER init parameter struct with a default value */
lptimer_parameter_struct lp_timer_initpara;
lptimer_struct_para_init(&lp_timer_initpara);
```

**lptimer_init**

The description of `lptimer_init` is shown as below:

**Table 3-395. Function lptimer_init**

<table>
<thead>
<tr>
<th>Function name</th>
<th>lptimer_init</th>
</tr>
</thead>
<tbody>
<tr>
<td>Function prototype</td>
<td>void lptimer_init(lptimer_parameter_struct *initpara);</td>
</tr>
<tr>
<td>Function descriptions</td>
<td>initialize LPTIMER counter</td>
</tr>
<tr>
<td>Precondition</td>
<td>-</td>
</tr>
<tr>
<td>The called functions</td>
<td>-</td>
</tr>
</tbody>
</table>

**Input parameter(in)**

- **initpara**: TIMER init parameter struct, the structure members can refer to Table 3-392, Structure lptimer_parameter_struct.

**Output parameter(out)**

- -

**Return value**

- -
Example:

/* initialize LPTIMER */

lptimer_parameter_struct lptimer_structure;
lptimer_structure.clocksource      = LPTIMER_INTERNALCLK;
lptimer_structure.prescaler        = LPTIMER_PSC_16;
lptimer_structure.extclockpolarity = LPTIMER_EXTERNALCLK_RISING;
lptimer_structure.extclockfilter   = LPTIMER_EXTERNALCLK_FILTEROFF;
lptimer_structure.triggermode      = LPTIMER_TRIGGER_SOFTWARE;
lptimer_structure.extriggersource  = LPTIMER_EXTRIGGER_GPIO;
lptimer_structure.extriggerfilter  = LPTIMER_TRIGGER_FILTEROFF;
lptimer_structure.outputpolarity   = LPTIMER_OUTPUT_NOTINVERTED;
lptimer_structure.outputmode       = LPTIMER_OUTPUT_PWMORSINGLE;
lptimer_structure.countersource    = LPTIMER_COUNTER_INTERNAL;
lptimer_init(&lptimer_structure);

lptimer_inputremap

The description of lptimer_inputremap is shown as below:

Table 3-396. Function lptimer_inputremap

<table>
<thead>
<tr>
<th>Function name</th>
<th>lptimer_inputremap</th>
</tr>
</thead>
<tbody>
<tr>
<td>Function prototype</td>
<td>void lptimer_inputremap(uint32_t input0remap, uint32_t input1remap);</td>
</tr>
<tr>
<td>Function descriptions</td>
<td>configure external input remap</td>
</tr>
<tr>
<td>Precondition</td>
<td>-</td>
</tr>
<tr>
<td>The called functions</td>
<td>-</td>
</tr>
</tbody>
</table>

**Input parameter(in)**

<table>
<thead>
<tr>
<th>input0remap</th>
<th>external input0 remap</th>
</tr>
</thead>
<tbody>
<tr>
<td>LPTIMER_INPUT0_GPIO</td>
<td>external input is remaped to GPIO</td>
</tr>
<tr>
<td>LPTIMER_INPUT0_CMP0_OUT</td>
<td>external input is remaped to CMP0_OUT</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>input1remap</th>
<th>external input1 remap</th>
</tr>
</thead>
<tbody>
<tr>
<td>LPTIMER_INPUT1_GPIO</td>
<td>external input is remaped to GPIO</td>
</tr>
<tr>
<td>LPTIMER_INPUT1_CMP</td>
<td>external input is remaped to CMP1_OUT</td>
</tr>
</tbody>
</table>
### GD32L23x Firmware Library User Guide

<table>
<thead>
<tr>
<th>P1_OUT</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Output parameter(out)</td>
<td></td>
</tr>
<tr>
<td>-</td>
<td></td>
</tr>
<tr>
<td>Return value</td>
<td></td>
</tr>
<tr>
<td>-</td>
<td></td>
</tr>
</tbody>
</table>

Example:

```c
/* configure LPTIMER inputs is connected to GPIO */
lptimer_inputremap(LPTIMER_INPUT0_GPIO, LPTIMER_INPUT1_GPIO);
```

### lptimer_register_shadow_enable

The description of `lptimer_register_shadow_enable` is shown as below:

**Table 3-397. Function lptimer_register_shadow_enable**

<table>
<thead>
<tr>
<th>Function name</th>
<th>lptimer_register_shadow_enable</th>
</tr>
</thead>
<tbody>
<tr>
<td>Function prototype</td>
<td>void lptimer_register_shadow_enable(void);</td>
</tr>
<tr>
<td>Function descriptions</td>
<td>enable the LPTIMER_CAR and LPTIMER_CMPV registers shadow function</td>
</tr>
<tr>
<td>Precondition</td>
<td>-</td>
</tr>
<tr>
<td>The called functions</td>
<td>-</td>
</tr>
<tr>
<td>Input parameter(in)</td>
<td>-</td>
</tr>
<tr>
<td>Output parameter(out)</td>
<td>-</td>
</tr>
<tr>
<td>Return value</td>
<td>-</td>
</tr>
</tbody>
</table>

Example:

```c
/* enable the LPTIMER_CAR and LPTIMER_CMPV registers shadow function */
lptimer_register_shadow_enable();
```

### lptimer_register_shadow_disable

The description of `lptimer_register_shadow_disable` is shown as below:

**Table 3-398. Function lptimer_register_shadow_disable**

<table>
<thead>
<tr>
<th>Function name</th>
<th>lptimer_register_shadow_disable</th>
</tr>
</thead>
<tbody>
<tr>
<td>Function prototype</td>
<td>void lptimer_register_shadow_disable(void);</td>
</tr>
<tr>
<td>Function descriptions</td>
<td>disable the LPTIMER_CAR and LPTIMER_CMPV registers shadow function</td>
</tr>
<tr>
<td>Precondition</td>
<td>-</td>
</tr>
<tr>
<td>The called functions</td>
<td>-</td>
</tr>
<tr>
<td>Input parameter(in)</td>
<td>-</td>
</tr>
</tbody>
</table>
Example:

/* disable the LPTIMER_CAR and LPTIMER_CMPV registers shadow function */
lptimer_register_shadow_disable();

**Iptimer_timeout_enable**

The description of `lptimer_timeout_enable` is shown as below:

<table>
<thead>
<tr>
<th>Function name</th>
<th>lptimer_timeout_enable</th>
</tr>
</thead>
<tbody>
<tr>
<td>Function prototype</td>
<td>void lptimer_timeout_enable(void);</td>
</tr>
<tr>
<td>Function descriptions</td>
<td>enable the LPTIMER TIMEOUT function</td>
</tr>
<tr>
<td>Precondition</td>
<td>-</td>
</tr>
<tr>
<td>The called functions</td>
<td>-</td>
</tr>
</tbody>
</table>

| Input parameter (in)        | -                      |
|                            | -                      |

| Output parameter (out)      | -                      |
|                            | -                      |

| Return value                | -                      |
|                            | -                      |

Example:

/* enable the LPTIMER TIMEOUT function */
lptimer_timeout_enable();

**Iptimer_timeout_disable**

The description of `lptimer_timeout_disable` is shown as below:

<table>
<thead>
<tr>
<th>Function name</th>
<th>lptimer_timeout_disable</th>
</tr>
</thead>
<tbody>
<tr>
<td>Function prototype</td>
<td>void lptimer_timeout_disable(void);</td>
</tr>
<tr>
<td>Function descriptions</td>
<td>disable the LPTIMER TIMEOUT function</td>
</tr>
<tr>
<td>Precondition</td>
<td>-</td>
</tr>
<tr>
<td>The called functions</td>
<td>-</td>
</tr>
</tbody>
</table>

| Input parameter (in)        | -                       |
|                            | -                       |

| Output parameter (out)      | -                       |
|                            | -                       |

| Return value                | -                       |
|                            | -                       |
Output parameter(out)

- |

Return value

- |

Example:

/* disable the LPTIMER TIMEOUT function */

lptimer_timeout_disable();

**lptimer_countinue_start**

The description of lptimer_countinue_start is shown as below:

<table>
<thead>
<tr>
<th>Function name</th>
<th>lptimer_countinue_start</th>
</tr>
</thead>
<tbody>
<tr>
<td>Function prototype</td>
<td>void lptimer_countinue_start(uint32_t autoreload, uint32_t compare);</td>
</tr>
<tr>
<td>Function descriptions</td>
<td>LPTIMER countinue start</td>
</tr>
<tr>
<td>Precondition</td>
<td>-</td>
</tr>
<tr>
<td>The called functions</td>
<td>-</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Input parameter(in)</th>
<th>autoreload</th>
</tr>
</thead>
<tbody>
<tr>
<td>auto reload value, 0~0xFFFFF</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Input parameter(in)</th>
<th>compare</th>
</tr>
</thead>
<tbody>
<tr>
<td>compare value, 0~0xFFFFF</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Output parameter(out)</th>
<th>-</th>
</tr>
</thead>
</table>

| Return value | - |

Example:

/* LPTIMER countinue start */

lptimer_countinue_start(0x0000FFFF, 0x00007FFF);

**lptimer_single_start**

The description of lptimer_single_start is shown as below:

<table>
<thead>
<tr>
<th>Function name</th>
<th>lptimer_single_start</th>
</tr>
</thead>
<tbody>
<tr>
<td>Function prototype</td>
<td>void lptimer_single_start(uint32_t autoreload, uint32_t compare);</td>
</tr>
<tr>
<td>Function descriptions</td>
<td>LPTIMER single start</td>
</tr>
<tr>
<td>Precondition</td>
<td>-</td>
</tr>
<tr>
<td>The called functions</td>
<td>-</td>
</tr>
</tbody>
</table>

| Input parameter(in) | - |

Example:

/* LPTIMER single start */

lptimer_single_start(0x0000FFFF, 0x00007FFF);
autoreload | auto reload value, 0~0xFFFFFFFF  
---|---
compare | compare value, 0~0xFFFFFFFF  
---|---
Input parameter\(\text{(in)}\) |  
Output parameter\(\text{(out)}\) |  
---|---
Input parameter\(\text{(in)}\) |  
Output parameter\(\text{(out)}\) |  
---|---
Return value |  
---|---

Example:

/* LPTIMER single start */

lptimer_single_start(0x0000FFFF, 0x00007FFF);

lptimer_stop

The description of lptimer_stop is shown as below:

<table>
<thead>
<tr>
<th>Function name</th>
<th>lptimer_stop</th>
</tr>
</thead>
<tbody>
<tr>
<td>Function prototype</td>
<td>void lptimer_stop(void);</td>
</tr>
<tr>
<td>Function descriptions</td>
<td>stop LPTIMER</td>
</tr>
<tr>
<td>Precondition</td>
<td>-</td>
</tr>
<tr>
<td>The called functions</td>
<td>-</td>
</tr>
<tr>
<td>Input parameter(\text{(in)})</td>
<td>-</td>
</tr>
<tr>
<td>Output parameter(\text{(out)})</td>
<td>-</td>
</tr>
<tr>
<td>Return value</td>
<td>-</td>
</tr>
</tbody>
</table>

Example:

/* stop LPTIMER */

lptimer_stop();

lptimer_counter_read

The description of lptimer_counter_read is shown as below:

<table>
<thead>
<tr>
<th>Function name</th>
<th>lptimer_counter_read</th>
</tr>
</thead>
<tbody>
<tr>
<td>Function prototype</td>
<td>uint32_t lptimer_counter_read(void);</td>
</tr>
<tr>
<td>Function descriptions</td>
<td>read LPTIMER current counter value</td>
</tr>
<tr>
<td>Precondition</td>
<td>-</td>
</tr>
<tr>
<td>The called functions</td>
<td>-</td>
</tr>
</tbody>
</table>
**Table 3-405. Function lptimer_autoreload_read**

<table>
<thead>
<tr>
<th>Function name</th>
<th>lptimer_autoreload_read</th>
</tr>
</thead>
<tbody>
<tr>
<td>Function prototype</td>
<td>uint32_t lptimer_autoreload_read(void);</td>
</tr>
<tr>
<td>Function descriptions</td>
<td>read LPTIMER auto reload value</td>
</tr>
<tr>
<td>Precondition</td>
<td>-</td>
</tr>
<tr>
<td>The called functions</td>
<td>-</td>
</tr>
<tr>
<td>Input parameter(in)</td>
<td>-</td>
</tr>
<tr>
<td>Output parameter(out)</td>
<td>-</td>
</tr>
<tr>
<td>Return value</td>
<td>uint32_t auto reload value (0~0xFFFFFFFF)</td>
</tr>
</tbody>
</table>

Example:

/* read LPTIMER auto reload value */
uint32_t i = 0;
i = lptimer_autoreload_read();

**Iptimer_compare_read**

The description of lptimer_compare_read is shown as below:

**Table 3-406. Function lptimer_compare_read**

<table>
<thead>
<tr>
<th>Function name</th>
<th>lptimer_compare_read</th>
</tr>
</thead>
<tbody>
<tr>
<td>Function prototype</td>
<td>uint32_t lptimer_compare_read(void);</td>
</tr>
<tr>
<td>Function descriptions</td>
<td>read LPTIMER compare value</td>
</tr>
<tr>
<td>Precondition</td>
<td>-</td>
</tr>
</tbody>
</table>

Example:

/* read LPTIMER compare value */
uint32_t i = 0;
i = lptimer_compare_read();
The called functions

<table>
<thead>
<tr>
<th>Function name</th>
<th>lptimer_autoreload_value_config</th>
</tr>
</thead>
<tbody>
<tr>
<td>Function prototype</td>
<td>void lptimer_autoreload_value_config(uint32_t autoreload);</td>
</tr>
<tr>
<td>Function descriptions</td>
<td>configure LPTIMER autoreload register value</td>
</tr>
<tr>
<td>Precondition</td>
<td>-</td>
</tr>
<tr>
<td>The called functions</td>
<td>-</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Input parameter(in)</th>
<th>autoreload</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>autoreload value (0-0xFFFFFFFF)</td>
</tr>
</tbody>
</table>

| Output parameter(out) | - |

| Return value          | - |

Example:

/* configure LPTIMER autoreload register value */

lptimer_autoreload_value_config(0x000000FF);

The description of lptimer_autoreload_value_config is shown as below:

Table 3-408. Function lptimer_compare_value_config

<table>
<thead>
<tr>
<th>Function name</th>
<th>lptimer_compare_value_config</th>
</tr>
</thead>
<tbody>
<tr>
<td>Function prototype</td>
<td>void lptimer_compare_value_config(uint32_t compare);</td>
</tr>
<tr>
<td>Function descriptions</td>
<td>configure LPTIMER compare value</td>
</tr>
<tr>
<td>Precondition</td>
<td>-</td>
</tr>
</tbody>
</table>

Example:

/* configure LPTIMER compare value */

lptimer_compare_value_config(0x000000FF);
The called functions -

<table>
<thead>
<tr>
<th>Input parameter (in)</th>
</tr>
</thead>
<tbody>
<tr>
<td>counter</td>
</tr>
<tr>
<td>compare value (0-0xFFFFFFFF)</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Output parameter (out)</th>
</tr>
</thead>
<tbody>
<tr>
<td>-</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Return value</th>
</tr>
</thead>
<tbody>
<tr>
<td>-</td>
</tr>
</tbody>
</table>

Example:

/* configure LPTIMER compare value */

lptimer_compare_value_config(0x000000FF);

**lptimer_decodemode0_enable**

The description of **lptimer_decodemode0_enable** is shown as below:

**Table 3-409. Function lptimer_decodemode0_enable**

<table>
<thead>
<tr>
<th>Function name</th>
<th>lptimer_decodemode0_enable</th>
</tr>
</thead>
<tbody>
<tr>
<td>Function prototype</td>
<td>void lptimer_decodemode0_enable(void);</td>
</tr>
<tr>
<td>Function descriptions</td>
<td>enable decode mode 0</td>
</tr>
<tr>
<td>Precondition</td>
<td>-</td>
</tr>
<tr>
<td>The called functions</td>
<td>-</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Input parameter (in)</th>
</tr>
</thead>
<tbody>
<tr>
<td>-</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Output parameter (out)</th>
</tr>
</thead>
<tbody>
<tr>
<td>-</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Return value</th>
</tr>
</thead>
<tbody>
<tr>
<td>-</td>
</tr>
</tbody>
</table>

Example:

/* enable decode mode 0 */

lptimer_decodemode0_enable();

**lptimer_decodemode1_enable**

The description of **lptimer_decodemode1_enable** is shown as below:

**Table 3-410. Function lptimer_decodemode1_enable**

<table>
<thead>
<tr>
<th>Function name</th>
<th>lptimer_decodemode1_enable</th>
</tr>
</thead>
<tbody>
<tr>
<td>Function prototype</td>
<td>void lptimer_decodemode1_enable(void);</td>
</tr>
<tr>
<td>Function descriptions</td>
<td>enable decode mode 1</td>
</tr>
<tr>
<td>Precondition</td>
<td>-</td>
</tr>
<tr>
<td>The called functions</td>
<td>-</td>
</tr>
</tbody>
</table>
### lptimer_decodemode_disable

The description of lptimer_decodemode_disable is shown as below:

#### Table 3-411. Function lptimer_decodemode_disable

<table>
<thead>
<tr>
<th>Function name</th>
<th>lptimer_decodemode_disable</th>
</tr>
</thead>
<tbody>
<tr>
<td>Function prototype</td>
<td>void lptimer_decodemode_disable(void);</td>
</tr>
<tr>
<td>Function descriptions</td>
<td>disable decode mode 0/1</td>
</tr>
<tr>
<td>Precondition</td>
<td>-</td>
</tr>
<tr>
<td>The called functions</td>
<td>-</td>
</tr>
<tr>
<td>Input parameter(in)</td>
<td>-</td>
</tr>
<tr>
<td>Output parameter(out)</td>
<td>-</td>
</tr>
<tr>
<td>Return value</td>
<td>-</td>
</tr>
</tbody>
</table>

Example:

```c
/* disable decode mode 0/1 */
lptimer_decodemode_disable();
```

### lptimer_highlevelcounter_enable

The description of lptimer_highlevelcounter_enable is shown as below:

#### Table 3-412. Function lptimer_highlevelcounter_enable

<table>
<thead>
<tr>
<th>Function name</th>
<th>lptimer_highlevelcounter_enable</th>
</tr>
</thead>
<tbody>
<tr>
<td>Function prototype</td>
<td>void lptimer_highlevelcounter_enable(uint32_t maxvalue);</td>
</tr>
<tr>
<td>Function descriptions</td>
<td>enable external input high level counter</td>
</tr>
<tr>
<td>Precondition</td>
<td>-</td>
</tr>
<tr>
<td>The called functions</td>
<td>-</td>
</tr>
<tr>
<td>Input parameter(in)</td>
<td>-</td>
</tr>
</tbody>
</table>

Example:

```c
/* enable decode mode */
lptimer_decodemode1_enable();
```
maxvalue | input high level counter max value, 0x0~0xFFFFFFFF
---|---
Output parameter(out) | -
- | -
Return value | -

Example:

/* enable external input high level counter */
lptimer_highlevelcounter_enable(0x00007FFF);

**Iptimer_highlevelcounter_disable**

The description of Iptimer_highlevelcounter_disable is shown as below:

<table>
<thead>
<tr>
<th>Function name</th>
<th>Iptimer_highlevelcounter_disable</th>
</tr>
</thead>
<tbody>
<tr>
<td>Function prototype</td>
<td>void Iptimer_highlevelcounter_disable(void);</td>
</tr>
<tr>
<td>Function descriptions</td>
<td>disable external input high level counter</td>
</tr>
<tr>
<td>Precondition</td>
<td>-</td>
</tr>
<tr>
<td>The called functions</td>
<td>-</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Input parameter(in)</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>-</td>
<td>-</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Output parameter(out)</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>-</td>
<td>-</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Return value</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>-</td>
<td>-</td>
</tr>
</tbody>
</table>

Example:

/* disable external input high level counter */
lptimer_highlevelcounter_disable();

**Iptimer_flag_get**

The description of Iptimer_flag_get is shown as below:

<table>
<thead>
<tr>
<th>Function name</th>
<th>Iptimer_flag_get</th>
</tr>
</thead>
<tbody>
<tr>
<td>Function prototype</td>
<td>FlagStatus lptimer_flag_get(uint32_t flag);</td>
</tr>
<tr>
<td>Function descriptions</td>
<td>get LPTIMER flags</td>
</tr>
<tr>
<td>Precondition</td>
<td>-</td>
</tr>
<tr>
<td>The called functions</td>
<td>-</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Input parameter(in)</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>flag</td>
<td>the LPTIMER flag</td>
</tr>
</tbody>
</table>
### LPTIMER_flag_CMPV
- **Description:** Compare value register match flag

### LPTIMER_flag_CARM
- **Description:** Counter auto reload register match flag

### LPTIMER_flag_ETED
- **Description:** External trigger edge event flag

### LPTIMER_flag_CMPV_UP
- **Description:** Compare value register update flag

### LPTIMER_flag_CARU
- **Description:** Counter auto reload register update flag

### LPTIMER_flag_UP
- **Description:** LPTIMER counter direction change down to up flag

### LPTIMER_flag_DOWN
- **Description:** LPTIMER counter direction change up to down flag

### LPTIMER_flag_HLCM_VUP
- **Description:** Input high level counter max value register update flag

### LPTIMER_flag_INHLO
- **Description:** LPTIMER_INx(x=0,1) high level counter overflow flag

### LPTIMER_flag_INHLOE
- **Description:** The high level of LPTIMER_IN0 and LPTIMER_IN1 overlap error flag

### LPTIMER_flag_INRF_OE
- **Description:** The falling and rising edges of LPTIMER_IN0 and LPTIMER_IN1 overlap error flag

### LPTIMER_flag_IN0E
- **Description:** LPTIMER_IN0 error flag

### LPTIMER_flag_IN1E
- **Description:** LPTIMER_IN1 error flag

#### Function Parameters

**Output parameter (out):**
- -

**Return value:**
- FlagStatus

**FlagStatus:**
- SET or RESET

#### Example

```c
/* get LPTIMER flag */
FlagStatus Flag_status = RESET;
Flag_status = lptimer_flag_get(LPTIMER_FLAG_CMPVM);
```

### lptimer_flag_clear

The description of lptimer_flag_clear is shown as below:

#### Table 3.415. Function lptimer_flag_clear

<table>
<thead>
<tr>
<th>Function name</th>
<th>lptimer_flag_clear</th>
</tr>
</thead>
<tbody>
<tr>
<td>Function prototype</td>
<td>void lptimer_flag_clear(uint32_t flag);</td>
</tr>
<tr>
<td>Function descriptions</td>
<td>Clear LPTIMER flags</td>
</tr>
<tr>
<td>Precondition</td>
<td>-</td>
</tr>
<tr>
<td>The called functions</td>
<td>-</td>
</tr>
<tr>
<td>Input parameter (in)</td>
<td>-</td>
</tr>
</tbody>
</table>
**LPTIMER FLAGs**

- **LPTIMER_FLAG_CMPVM** - compare value register match flag
- **LPTIMER_FLAG_CARM** - counter auto reload register match flag
- **LPTIMER_FLAG_ETED** - external trigger edge event flag
- **LPTIMER_FLAG_CMPV_UP** - compare value register update flag
- **LPTIMER_FLAG_CARU_P** - counter auto reload register update flag
- **LPTIMER_FLAG_UP** - LPTIMER counter direction change down to up flag
- **LPTIMER_FLAG_DOWN** - LPTIMER counter direction change up to down flag
- **LPTIMER_FLAG_HLCMVUP** - input high level counter max value register update flag
- **LPTIMER_FLAG_INHLOE** - LPTIMER_INx(x=0,1) high level counter overflow flag
- **LPTIMER_FLAG_INHLOE** - the high level of LPTIMER_IN0 and LPTIMER_IN1 overlap error flag
- **LPTIMER_FLAG_INRFOE** - the falling and rising edges of LPTIMER_IN0 and LPTIMER_IN1 overlap error flag
- **LPTIMER_FLAG_IN0E** - LPTIMER_IN0 error flag
- **LPTIMER_FLAG_IN1E** - LPTIMER_IN1 error flag

**Output parameter(out)**
- -

**Return value**
- -

**Example:**

```c
/* clear LPTIMER flag */

lptimer_flag_clear(LPTIMER_FLAG_CMPVM);
```

**lptimer_interrupt_enable**

The description of lptimer_interrupt_enable is shown as below:

**Table 3-416. Function lptimer_interrupt_enable**

<table>
<thead>
<tr>
<th>Function name</th>
<th>lptimer_interrupt_enable</th>
</tr>
</thead>
<tbody>
<tr>
<td>Function prototype</td>
<td>void lptimer_interrupt_enable(uint32_t interrupt);</td>
</tr>
<tr>
<td>Function descriptions</td>
<td>enable the LPTIMER interrupt</td>
</tr>
<tr>
<td>Precondition</td>
<td>-</td>
</tr>
<tr>
<td>The called functions</td>
<td>-</td>
</tr>
</tbody>
</table>

**Input parameter(in)**
<table>
<thead>
<tr>
<th>interrupt</th>
<th>LPTIMER interrupt source</th>
</tr>
</thead>
<tbody>
<tr>
<td>LPTIMER_INT_CMPVM</td>
<td>compare value register match interrupt</td>
</tr>
<tr>
<td>LPTIMER_INT_CARM</td>
<td>counter auto reload register match interrupt</td>
</tr>
<tr>
<td>LPTIMER_INT_ETEDEV</td>
<td>external trigger edge event interrupt</td>
</tr>
<tr>
<td>LPTIMER_INT_CMPVU</td>
<td>compare value register update interrupt</td>
</tr>
<tr>
<td>LPTIMER_INT_CARUP</td>
<td>counter auto reload register update interrupt</td>
</tr>
<tr>
<td>LPTIMER_INT_UP</td>
<td>LPTIMER counter direction change down to up interrupt</td>
</tr>
<tr>
<td>LPTIMER_INT_DOWN</td>
<td>LPTIMER counter direction change up to down interrupt</td>
</tr>
<tr>
<td>LPTIMER_INT_HLCMV UP</td>
<td>input high level counter max value register update interrupt</td>
</tr>
<tr>
<td>LPTIMER_INT_INHLCO</td>
<td>LPTIMER_INx(x=0,1) high level counter overflow interrupt</td>
</tr>
<tr>
<td>LPTIMER_INT_INHLOE</td>
<td>the high level of LPTIMER_IN0 and LPTIMER_IN1 overlap error interrupt</td>
</tr>
<tr>
<td>LPTIMER_INT_INRFOE</td>
<td>the falling and rising edges of LPTIMER_IN0 and LPTIMER_IN1 overlap error interrupt</td>
</tr>
<tr>
<td>LPTIMER_INT_IN0E</td>
<td>LPTIMER_IN0 error interrupt</td>
</tr>
<tr>
<td>LPTIMER_INT_IN1E</td>
<td>LPTIMER_IN1 error interrupt</td>
</tr>
</tbody>
</table>

**Output parameter** (out)

- -

**Return value**

- -

Example:

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

Iptimer_interrupt_enable(LPTIMER_INT_CMPVM);
```

**Iptimer_interrupt_disable**

The description of Iptimer_interrupt_disable is shown as below:

**Table 3-417. Function Iptimer_interrupt_disable**

<table>
<thead>
<tr>
<th>Function name</th>
<th>Iptimer_interrupt_disabled</th>
</tr>
</thead>
<tbody>
<tr>
<td>Function prototype</td>
<td>void Iptimer_interrupt_disabled(uint32_t interrupt);</td>
</tr>
<tr>
<td>Function descriptions</td>
<td>disable the LPTIMER interrupt</td>
</tr>
<tr>
<td>Precondition</td>
<td>-</td>
</tr>
<tr>
<td>The called functions</td>
<td>-</td>
</tr>
</tbody>
</table>

**Input parameter** (in)

<table>
<thead>
<tr>
<th>interrupt</th>
<th>LPTIMER interrupt source</th>
</tr>
</thead>
<tbody>
<tr>
<td>LPTIMER_INT_CMPVM</td>
<td>compare value register match interrupt</td>
</tr>
<tr>
<td>LPTIMER_INT_CARM</td>
<td>counter auto reload register match interrupt</td>
</tr>
<tr>
<td>LPTIMER_INT_ETEDEV</td>
<td>external trigger edge event interrupt</td>
</tr>
<tr>
<td>LPTIMER_INT_CMPVU</td>
<td>compare value register update interrupt</td>
</tr>
</tbody>
</table>
GD32L23x Firmware Library User Guide

LPTIMER_INT_CARUP  counter auto reload register update interrupt
LPTIMER_INT_UP    LPTIMER counter direction change down to up interrupt
LPTIMER_INT_DOWN  LPTIMER counter direction change up to down interrupt
LPTIMER_INT_HLCMV  input high level counter max value register update interrupt
LPTIMER_INT_INHLCO LPTIMER_INx(x=0,1) high level counter overflow interrupt
LPTIMER_INT_INHLOE the high level of LPTIMER_IN0 and LPTIMER_IN1 overlap error interrupt
LPTIMER_INT_INRFOE the falling and rising edges of LPTIMER_IN0 and LPTIMER_IN1 overlap error interrupt
LPTIMER_INT_IN0E   LPTIMER_IN0 error interrupt
LPTIMER_INT_IN1E   LPTIMER_IN1 error interrupt

Output parameter(out)
- -

Return value
- -

Example:

/* disable the LPTIMER interrupt */
lptimer_interrupt_disable(LPTIMER_INT_CMPVM);

lptimer_interrupt_flag_get

The description of lptimer_interrupt_flag_get is shown as below:

Table 3-418. Function lptimer_interrupt_flag_get

<table>
<thead>
<tr>
<th>Function name</th>
<th>lptimer_interrupt_flag_get</th>
</tr>
</thead>
<tbody>
<tr>
<td>Function prototype</td>
<td>FlagStatus lptimer_interrupt_flag_get(uint32_t int_flag);</td>
</tr>
<tr>
<td>Function descriptions</td>
<td>get LPTIMER interrupt flag</td>
</tr>
<tr>
<td>Precondition</td>
<td>-</td>
</tr>
<tr>
<td>The called functions</td>
<td>-</td>
</tr>
</tbody>
</table>

Input parameter(in)

<table>
<thead>
<tr>
<th>int_flag</th>
<th>the LPTIMER interrupt flag</th>
</tr>
</thead>
<tbody>
<tr>
<td>LPTIMER_INT_FLAG_CMPVM</td>
<td>compare value register match interrupt flag</td>
</tr>
<tr>
<td>LPTIMER_INT_FLAG_ARM</td>
<td>counter auto reload register match interrupt flag</td>
</tr>
<tr>
<td>LPTIMER_INT_FLAG_TEDEV</td>
<td>external trigger edge event interrupt flag</td>
</tr>
<tr>
<td>LPTIMER_INT_FLAG_MPVUP</td>
<td>compare value register update interrupt flag</td>
</tr>
<tr>
<td>LPTIMER_INT_FLAG_ARUP</td>
<td>counter auto reload register update interrupt flag</td>
</tr>
<tr>
<td>LPTIMER_INT_FLAG_U</td>
<td>LPTIMER counter direction change down to up interrupt flag</td>
</tr>
</tbody>
</table>
### LPTIMER_INT_FLAG_D
- **LPTIMER_INT_FLAG_DOWN**: LPTIMER counter direction change up to down interrupt flag

### LPTIMER_INT_FLAG_H
- **LPTIMER_INT_FLAG_HCMVUP**: input high level counter max value register update interrupt flag

### LPTIMER_INT_FLAG_I
- **LPTIMER_INT_FLAG_INHLOE**: LPTIMER_INx(x=0,1) high level counter overflow interrupt flag
- **LPTIMER_INT_FLAG_INLCVUP**: input high level counter max value register update interrupt flag
- **LPTIMER_INT_FLAG_INLPC**: the high level of LPTIMER_IN0 and LPTIMER_IN1 overlap error interrupt flag
- **LPTIMER_INT_FLAG_INLPOE**: the falling and rising edges of LPTIMER_IN0 and LPTIMER_IN1 overlap error interrupt flag
- **LPTIMER_INT_FLAG_INNOE**: LPTIMER_IN0 error interrupt flag
- **LPTIMER_INT_FLAG_INN1E**: LPTIMER_IN1 error interrupt flag

#### Output parameter (out)
- -

#### Return value
- **FlagStatus**: SET or RESET

### Example:

```c
/* get LPTIMER interrupt flag */

FlagStatus Flag_interrupt = RESET;
Flag_interrupt = lptimer_interrupt_flag_get(LPTIMER_INT_FLAG_CMPVM);

### lptimer_interrupt_flag_clear

The description of `lptimer_interrupt_flag_clear` is shown as below:

#### Table 3-419. Function lptimer_interrupt_flag_clear

<table>
<thead>
<tr>
<th>Function name</th>
<th>lptimer_interrupt_flag_clear</th>
</tr>
</thead>
<tbody>
<tr>
<td>Function prototype</td>
<td>void lptimer_interrupt_flag_clear(uint32_t int_flag);</td>
</tr>
<tr>
<td>Function descriptions</td>
<td>clear LPTIMER interrupt flag</td>
</tr>
<tr>
<td>Precondition</td>
<td>-</td>
</tr>
<tr>
<td>The called functions</td>
<td>-</td>
</tr>
</tbody>
</table>

#### Input parameter (in)

<table>
<thead>
<tr>
<th>int_flag</th>
<th>the LPTIMER interrupt flag</th>
</tr>
</thead>
<tbody>
<tr>
<td>LPTIMER_INT_FLAG_CMPVM</td>
<td>compare value register match interrupt flag</td>
</tr>
<tr>
<td>LPTIMER_INT_FLAG_ARM</td>
<td>counter auto reload register match interrupt flag</td>
</tr>
<tr>
<td>LPTIMER_INT_FLAG_E TEDEV</td>
<td>external trigger edge event interrupt flag</td>
</tr>
</tbody>
</table>
### LPTIMER_INT_FLAG_P
- **LPTIMER_INT_FLAG_CMPVM**
  - Description: compare value register update interrupt flag
- **LPTIMER_INT_FLAG_ARUP**
  - Description: counter auto reload register update interrupt flag
- **LPTIMER_INT_FLAG_UP**
  - Description: LPTIMER counter direction change down to up interrupt flag
- **LPTIMER_INT_FLAG_DWN**
  - Description: LPTIMER counter direction change up to down interrupt flag
- **LPTIMER_INT_FLAG_LCMVUP**
  - Description: input high level counter max value register update interrupt flag
- **LPTIMER_INT_FLAG_INHLOE**
  - Description: LPTIMER_INx(x=0,1) high level counter overflow interrupt flag
- **LPTIMER_INT_FLAG_INHLOE**
  - Description: the high level of LPTIMER_IN0 and LPTIMER_IN1 overlap error interrupt flag
- **LPTIMER_INT_FLAG_INRFOE**
  - Description: the falling and rising edges of LPTIMER_IN0 and LPTIMER_IN1 overlap error interrupt flag
- **LPTIMER_INT_FLAG_INOE**
  - Description: LPTIMER_IN0 error interrupt flag
- **LPTIMER_INT_FLAG_IN1E**
  - Description: LPTIMER_IN1 error interrupt flag

### Output parameter(out)
- `-`

### Return value
- `-`

Example:
```c
/* clear LPTIMER interrupt flag */
lptimer_interrupt_flag_clear(LPTIMER_INT_FLAG_CMPVM);
```

### 3.16. LPUART

The Low power Universal Asynchronous Receiver/Transmitter (LPUART) provides a flexible serial data exchange interface. The LPUART registers are listed in chapter 3.16.1, the LPUART firmware functions are introduced in chapter 3.16.2.

### 3.16.1. Descriptions of Peripheral registers

LPUART registers are listed in the table shown as below:

<table>
<thead>
<tr>
<th>Registers</th>
<th>Descriptions</th>
</tr>
</thead>
<tbody>
<tr>
<td>LPUART_CTL0</td>
<td>Control register 0</td>
</tr>
</tbody>
</table>
### Descriptions of Peripheral functions

LPUART firmware functions are listed in the table shown as below:

<table>
<thead>
<tr>
<th>Function name</th>
<th>Function description</th>
</tr>
</thead>
<tbody>
<tr>
<td>lpuart_deinit</td>
<td>reset LPUART</td>
</tr>
<tr>
<td>lpuart_baudrate_set</td>
<td>configure LPUART baud rate value</td>
</tr>
<tr>
<td>lpuart_parity_config</td>
<td>configure LPUART parity</td>
</tr>
<tr>
<td>lpuart_word_length_set</td>
<td>configure LPUART word length</td>
</tr>
<tr>
<td>lpuart_stop_bit_set</td>
<td>configure LPUART stop bit length</td>
</tr>
<tr>
<td>lpuart_enable</td>
<td>enable LPUART</td>
</tr>
<tr>
<td>lpuart_disable</td>
<td>disable LPUART</td>
</tr>
<tr>
<td>lpuart_transmit_config</td>
<td>configure LPUART transmitter</td>
</tr>
<tr>
<td>lpuart_receive_config</td>
<td>configure LPUART receiver</td>
</tr>
<tr>
<td>lpuart_data_first_config</td>
<td>data is transmitted/received with the LSB/MSB first</td>
</tr>
<tr>
<td>lpuart_invert_config</td>
<td>configure LPUART inverted</td>
</tr>
<tr>
<td>lpuart_overrun_enable</td>
<td>enable the LPUART overrun function</td>
</tr>
<tr>
<td>lpuart_overrun_disable</td>
<td>disable the LPUART overrun function</td>
</tr>
<tr>
<td>lpuart_data_transmit</td>
<td>LPUART transmit data function</td>
</tr>
<tr>
<td>lpuart_data_receive</td>
<td>LPUART receive data function</td>
</tr>
<tr>
<td>lpuart_command_enable</td>
<td>enable LPUART command</td>
</tr>
<tr>
<td>lpuart_address_config</td>
<td>configure address of the LPUART</td>
</tr>
<tr>
<td>lpuart_address_detection_mode_config</td>
<td>configure address detection mode</td>
</tr>
<tr>
<td>lpuart_mute_mode_enable</td>
<td>enable mute mode</td>
</tr>
<tr>
<td>lpuart_mute_mode_disable</td>
<td>disable mute mode</td>
</tr>
<tr>
<td>lpuart_mute_mode_wakeup_config</td>
<td>configure wakeup method in mute mode</td>
</tr>
<tr>
<td>lpuart_halfduplex_enable</td>
<td>enable half-duplex mode</td>
</tr>
<tr>
<td>lpuart_halfduplex_disable</td>
<td>disable half-duplex mode</td>
</tr>
<tr>
<td>lpuart_hardware_flow_rts_config</td>
<td>configure hardware flow control RTS</td>
</tr>
<tr>
<td>lpuart_hardware_flow_cts_config</td>
<td>configure hardware flow control CTS</td>
</tr>
</tbody>
</table>
### Function names and descriptions

<table>
<thead>
<tr>
<th>Function name</th>
<th>Function description</th>
</tr>
</thead>
<tbody>
<tr>
<td>lpuart_hardware_flow_coherence_config</td>
<td>configure hardware flow control coherence mode</td>
</tr>
<tr>
<td>lpuart_rs485_driver_enable</td>
<td>enable RS485 driver</td>
</tr>
<tr>
<td>lpuart_rs485_driver_disable</td>
<td>disable RS485 driver</td>
</tr>
<tr>
<td>lpuart_driver_assertime_config</td>
<td>configure driver enable assertion time</td>
</tr>
<tr>
<td>lpuart_driver_deassertime_config</td>
<td>configure driver enable de-assertion time</td>
</tr>
<tr>
<td>lpuart_depolarity_config</td>
<td>configure driver enable polarity mode</td>
</tr>
<tr>
<td>lpuart_dma_receive_config</td>
<td>configure LPUART DMA for reception</td>
</tr>
<tr>
<td>lpuart_dma_transmit_config</td>
<td>configure LPUART DMA for transmission</td>
</tr>
<tr>
<td>lpuart_reception_error_dma_disable</td>
<td>disable DMA on reception error</td>
</tr>
<tr>
<td>lpuart_reception_error_dma_enable</td>
<td>enable DMA on reception error</td>
</tr>
<tr>
<td>lpuart_wakeup_enable</td>
<td>enable LPUART to wakeup the mcu from deep-sleep mode</td>
</tr>
<tr>
<td>lpuart_wakeup_disable</td>
<td>disable LPUART to wakeup the mcu from deep-sleep mode</td>
</tr>
<tr>
<td>lpuart_wakeup_mode_config</td>
<td>configure the LPUART wakeup mode from deep-sleep mode</td>
</tr>
<tr>
<td>lpuart_flag_get</td>
<td>get flag in STAT/CHC register</td>
</tr>
<tr>
<td>lpuart_flag_clear</td>
<td>clear LPUART status</td>
</tr>
<tr>
<td>lpuart_interrupt_enable</td>
<td>enable LPUART interrupt</td>
</tr>
<tr>
<td>lpuart_interrupt_disable</td>
<td>disable LPUART interrupt</td>
</tr>
<tr>
<td>lpuart_interrupt_flag_get</td>
<td>get LPUART interrupt and flag status</td>
</tr>
<tr>
<td>lpuart_interrupt_flag_clear</td>
<td>clear LPUART interrupt flag</td>
</tr>
</tbody>
</table>

### Enum lpuart_flag_enum

#### Table 3-422. Enum lpuart_flag_enum

<table>
<thead>
<tr>
<th>Member name</th>
<th>Function description</th>
</tr>
</thead>
<tbody>
<tr>
<td>LPUART_FLAG_REA</td>
<td>receive enable acknowledge flag</td>
</tr>
<tr>
<td>LPUART_FLAG_TEA</td>
<td>transmit enable acknowledge flag</td>
</tr>
<tr>
<td>LPUART_FLAG_WU</td>
<td>wakeup from Deep-sleep mode flag</td>
</tr>
<tr>
<td>LPUART_FLAG_RWU</td>
<td>receiver wakeup from mute mode</td>
</tr>
<tr>
<td>LPUART_FLAG_AM</td>
<td>ADDR match flag</td>
</tr>
<tr>
<td>LPUART_FLAG_BSY</td>
<td>busy flag</td>
</tr>
<tr>
<td>LPUART_FLAG_CTS</td>
<td>CTS level</td>
</tr>
<tr>
<td>LPUART_FLAG_CTSF</td>
<td>CTS change flag</td>
</tr>
<tr>
<td>LPUART_FLAG_TBE</td>
<td>transmit data buffer empty</td>
</tr>
<tr>
<td>LPUART_FLAG_TC</td>
<td>transmission complete</td>
</tr>
<tr>
<td>LPUART_FLAG_RBNE</td>
<td>read data buffer not empty</td>
</tr>
<tr>
<td>LPUART_FLAG_IDLE</td>
<td>IDLE line detected flag</td>
</tr>
<tr>
<td>LPUART_FLAG_ORERR</td>
<td>overrun error</td>
</tr>
<tr>
<td>LPUART_FLAG_NERR</td>
<td>noise error flag</td>
</tr>
<tr>
<td>LPUART_FLAG_FERR</td>
<td>frame error flag</td>
</tr>
<tr>
<td>LPUART_FLAG_PERR</td>
<td>parity error flag</td>
</tr>
<tr>
<td>LPUART_FLAG_EPERR</td>
<td>early parity error flag</td>
</tr>
</tbody>
</table>
### Enum lpuart_interrupt_flag_enum

#### Table 3-423. Enum lpuart_interrupt_flag_enum

<table>
<thead>
<tr>
<th>Member name</th>
<th>Function description</th>
</tr>
</thead>
<tbody>
<tr>
<td>LPUART_INT_FLAG_AM</td>
<td>address match interrupt and flag</td>
</tr>
<tr>
<td>LPUART_INT_FLAG_PERR</td>
<td>parity error interrupt and flag</td>
</tr>
<tr>
<td>LPUART_INT_FLAG_TBE</td>
<td>transmitter buffer empty interrupt and flag</td>
</tr>
<tr>
<td>LPUART_INT_FLAG_TC</td>
<td>transmission complete interrupt and flag</td>
</tr>
<tr>
<td>LPUART_INT_FLAG_RBNE</td>
<td>read data buffer not empty interrupt and flag</td>
</tr>
<tr>
<td>LPUART_INT_FLAG_RBNE_ORE_RR</td>
<td>read data buffer not empty interrupt and overrun error flag</td>
</tr>
<tr>
<td>LPUART_INT_FLAG_IDLE</td>
<td>IDLE line detected interrupt and flag</td>
</tr>
<tr>
<td>LPUART_INT_FLAG_WU</td>
<td>wakeup from deep-sleep mode interrupt and flag</td>
</tr>
<tr>
<td>LPUART_INT_FLAG_CTS</td>
<td>CTS interrupt and flag</td>
</tr>
<tr>
<td>LPUART_INT_FLAG_ERR_NERR</td>
<td>error interrupt and noise error flag</td>
</tr>
<tr>
<td>LPUART_INT_FLAG_ERR_ORE_RR</td>
<td>error interrupt and overrun error</td>
</tr>
<tr>
<td>LPUART_INT_FLAG_ERR_FERR</td>
<td>error interrupt and frame error flag</td>
</tr>
</tbody>
</table>

### Enum lpuart_interrupt_enum

#### Table 3-424. Enum lpuart_interrupt_enum

<table>
<thead>
<tr>
<th>Member name</th>
<th>Function description</th>
</tr>
</thead>
<tbody>
<tr>
<td>LPUART_INT_AM</td>
<td>address match interrupt</td>
</tr>
<tr>
<td>LPUART_INT_PERR</td>
<td>parity error interrupt</td>
</tr>
<tr>
<td>LPUART_INT_TBE</td>
<td>transmitter buffer empty interrupt</td>
</tr>
<tr>
<td>LPUART_INT_TC</td>
<td>transmission complete interrupt</td>
</tr>
<tr>
<td>LPUART_INT_RBNE</td>
<td>read data buffer not empty interrupt and overrun error interrupt</td>
</tr>
<tr>
<td>LPUART_INT_IDLE</td>
<td>IDLE line detected interrupt</td>
</tr>
<tr>
<td>LPUART_INT_WU</td>
<td>wakeup from deep-sleep mode interrupt</td>
</tr>
<tr>
<td>LPUART_INT_CTS</td>
<td>CTS interrupt</td>
</tr>
<tr>
<td>LPUART_INT_ERR</td>
<td>error interrupt</td>
</tr>
</tbody>
</table>

### Enum lpuart_invert_enum

#### Table 3-425. Enum lpuart_invert_enum

<table>
<thead>
<tr>
<th>Member name</th>
<th>Function description</th>
</tr>
</thead>
<tbody>
<tr>
<td>LPUART_DINV_ENABLE</td>
<td>data bit level inversion</td>
</tr>
<tr>
<td>LPUART_DINV_DISABLE</td>
<td>data bit level not inversion</td>
</tr>
<tr>
<td>LPUART_TXPIN_ENABLE</td>
<td>TX pin level inversion</td>
</tr>
<tr>
<td>LPUART_TXPIN_DISABLE</td>
<td>TX pin level not inversion</td>
</tr>
<tr>
<td>LPUART_RXPIN_ENABLE</td>
<td>RX pin level inversion</td>
</tr>
<tr>
<td>LPUART_RXPIN_DISABLE</td>
<td>RX pin level not inversion</td>
</tr>
<tr>
<td>Member name</td>
<td>Function description</td>
</tr>
<tr>
<td>---------------------------</td>
<td>----------------------------</td>
</tr>
<tr>
<td>LPUART_SWAP_ENABLE</td>
<td>swap TX/RX pins</td>
</tr>
<tr>
<td>LPUART_SWAP_DISABLE</td>
<td>not swap TX/RX pins</td>
</tr>
</tbody>
</table>

**Ipuart_deinit**

The description of Ipuart_deinit is shown as below:

**Table 3-426. Function Ipuart_deinit**

<table>
<thead>
<tr>
<th>Function name</th>
<th>Ipuart_deinit</th>
</tr>
</thead>
<tbody>
<tr>
<td>Function prototype</td>
<td>void Ipuart_deinit(void)</td>
</tr>
<tr>
<td>Function descriptions</td>
<td>reset LPUART</td>
</tr>
<tr>
<td>Precondition</td>
<td></td>
</tr>
<tr>
<td>The called functions</td>
<td>rcu_periph_reset_enable / rcu_periph_reset_disable</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Input parameter(in)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
</tr>
<tr>
<td>Output parameter(out)</td>
</tr>
<tr>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Return value</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
</tr>
</tbody>
</table>

Example:

/* reset LPUART */

Ipuart_deinit();

**Ipuart_baudrate_set**

The description of Ipuart_baudrate_set is shown as below:

**Table 3-427. Function Ipuart_baudrate_set**

<table>
<thead>
<tr>
<th>Function name</th>
<th>Ipuart_baudrate_set</th>
</tr>
</thead>
<tbody>
<tr>
<td>Function prototype</td>
<td>void Ipuart_baudrate_set(uint32_t baudval);</td>
</tr>
<tr>
<td>Function descriptions</td>
<td>configure LPUART baud rate value</td>
</tr>
<tr>
<td>Precondition</td>
<td></td>
</tr>
<tr>
<td>The called functions</td>
<td>rcu_clock_freq_get</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Input parameter(in)</th>
</tr>
</thead>
<tbody>
<tr>
<td>baudval</td>
</tr>
<tr>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Output parameter(out)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Return value</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
</tr>
</tbody>
</table>

Example:

/* configure LPUART baud rate value */
Ipuart_baudrate_set(115200);

Ipuart_parity_config

The description of Ipuart_parity_config is shown as below:

Table 3-428. Function Ipuart_parity_config

<table>
<thead>
<tr>
<th>Function name</th>
<th>Ipuart_parity_config</th>
</tr>
</thead>
<tbody>
<tr>
<td>Function prototype</td>
<td>void Ipuart_parity_config( uint32_t paritycfg );</td>
</tr>
<tr>
<td>Function descriptions</td>
<td>configure LPUART parity</td>
</tr>
<tr>
<td>Precondition</td>
<td>-</td>
</tr>
<tr>
<td>The called functions</td>
<td>-</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Input parameter(in)</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>paritycfg</td>
<td>configure LPUART parity</td>
</tr>
<tr>
<td>LPUART_PM_NONE</td>
<td>no parity</td>
</tr>
<tr>
<td>LPUART_PM_ODD</td>
<td>odd parity</td>
</tr>
<tr>
<td>LPUART_PM_EVEN</td>
<td>even parity</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Output parameter(out)</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>-</td>
<td>-</td>
</tr>
</tbody>
</table>

Return value

Example:

/* configure LPUART parity */

Ipuart_parity_config(LPUART_PM_EVEN);

Ipuart_word_length_set

The description of Ipuart_word_length_set is shown as below:

Table 3-429. Function Ipuart_word_length_set

<table>
<thead>
<tr>
<th>Function name</th>
<th>Ipuart_word_length_set</th>
</tr>
</thead>
<tbody>
<tr>
<td>Function prototype</td>
<td>void Ipuart_word_length_set( uint32_t wlen );</td>
</tr>
<tr>
<td>Function descriptions</td>
<td>configure LPUART word length</td>
</tr>
<tr>
<td>Precondition</td>
<td>-</td>
</tr>
<tr>
<td>The called functions</td>
<td>-</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Input parameter(in)</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>wlen</td>
<td>LPART word length configure</td>
</tr>
<tr>
<td>LPUART_WL_7BIT</td>
<td>7 bits</td>
</tr>
<tr>
<td>LPUART_WL_8BIT</td>
<td>8 bits</td>
</tr>
<tr>
<td>LPUART_WL_9BIT</td>
<td>9 bits</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Output parameter(out)</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>-</td>
<td>-</td>
</tr>
</tbody>
</table>

Return value
Example:

/* configure LPUART word length */

lpuart_word_length_set(LPUART_WL_9BIT);

**lpuart_stop_bit_set**

The description of `lpuart_stop_bit_set` is shown as below:

<table>
<thead>
<tr>
<th>Function name</th>
<th>lpuart_stop_bit_set</th>
</tr>
</thead>
<tbody>
<tr>
<td>Function prototype</td>
<td>void lpuart_stop_bit_set(uint32_t stblen);</td>
</tr>
<tr>
<td>Function descriptions</td>
<td>configure LPUART stop bit length</td>
</tr>
<tr>
<td>Precondition</td>
<td>-</td>
</tr>
<tr>
<td>The called functions</td>
<td>-</td>
</tr>
<tr>
<td>Input parameter(in)</td>
<td>stblen</td>
</tr>
<tr>
<td></td>
<td>LPUART stop bit configure</td>
</tr>
<tr>
<td></td>
<td>USART_STB_1BIT 1 bit</td>
</tr>
<tr>
<td></td>
<td>USART_STB_2BIT 2 bits</td>
</tr>
<tr>
<td>Output parameter(out)</td>
<td>-</td>
</tr>
<tr>
<td>Return value</td>
<td>-</td>
</tr>
</tbody>
</table>

Example:

/* configure LPUART stop bit length */

lpuart_stop_bit_set(LPUART_STB_1_5BIT);

**lpuart_enable**

The description of `lpuart_enable` is shown as below:

<table>
<thead>
<tr>
<th>Function name</th>
<th>lpuart_enable</th>
</tr>
</thead>
<tbody>
<tr>
<td>Function prototype</td>
<td>void lpuart_enable(void);</td>
</tr>
<tr>
<td>Function descriptions</td>
<td>enable LPUART</td>
</tr>
<tr>
<td>Precondition</td>
<td>-</td>
</tr>
<tr>
<td>The called functions</td>
<td>-</td>
</tr>
<tr>
<td>Input parameter(in)</td>
<td>-</td>
</tr>
<tr>
<td>Output parameter(out)</td>
<td>-</td>
</tr>
</tbody>
</table>
Example:

```c
/* enable LPUART */

lpuart_enable();
```

### lpuart_disable

The description of `lpuart_disable` is shown as below:

**Table 3-432. Function lpuart_disable**

<table>
<thead>
<tr>
<th>Function name</th>
<th>Function prototype</th>
<th>Function descriptions</th>
<th>Precondition</th>
<th>The called functions</th>
</tr>
</thead>
<tbody>
<tr>
<td>lpuart_disable</td>
<td>void lpuart_disable(void);</td>
<td>disable LPUART</td>
<td>-</td>
<td>-</td>
</tr>
</tbody>
</table>

**Input parameter(in)**

- -

**Output parameter(out)**

- -

**Return value**

- -

Example:

```c
/* disable LPUART */

lpuart_disable();
```

### lpuart_transmit_config

The description of `lpuart_transmit_config` is shown as below:

**Table 3-433. Function lpuart_transmit_config**

<table>
<thead>
<tr>
<th>Function name</th>
<th>Function prototype</th>
<th>Function descriptions</th>
<th>Precondition</th>
<th>The called functions</th>
</tr>
</thead>
<tbody>
<tr>
<td>lpuart_transmit_config</td>
<td>void lpuart_transmit_config(uint32_t txconfig);</td>
<td>configure LPUART transmitter</td>
<td>-</td>
<td>-</td>
</tr>
</tbody>
</table>

**Input parameter(in)**

- txconfig

- `LPUART_TRANSMIT_ENABLE`  
  enable LPUART transmission

- `LPUART_TRANSMIT_DISABLE`  
  disable LPUART transmission
Example:

/* configure LPUART transmitter */
lpuart_transmit_config(LPUART_TRANSMIT_ENABLE);

**lpuart_receive_config**

The description of lpuart_receive_config is shown as below:

**Table 3-434. Function lpuart_receive_config**

<table>
<thead>
<tr>
<th>Function name</th>
<th>lpuart_receive_config</th>
</tr>
</thead>
<tbody>
<tr>
<td>Function prototype</td>
<td>void lpuart_receive_config(uint32_t rxconfig);</td>
</tr>
<tr>
<td>Function descriptions</td>
<td>configure LPUART receiver</td>
</tr>
<tr>
<td>Precondition</td>
<td>-</td>
</tr>
<tr>
<td>The called functions</td>
<td>-</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>input parameter (in)</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>rxconfig</td>
<td>enable or disable LPUART receiver</td>
</tr>
<tr>
<td>LPUART_RECEIVE_ENABLE</td>
<td>enable LPUART reception</td>
</tr>
<tr>
<td>LPUART_RECEIVE_DISABLE</td>
<td>disable LPUART reception</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>output parameter (out)</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>-</td>
<td>-</td>
</tr>
</tbody>
</table>

Example:

/* configure LPUART receiver */
lpuart_reconfigure(LPUART_RECEIVE_ENABLE);

**lpuart_data_first_config**

The description of lpuart_data_first_config is shown as below:

**Table 3-435. Function lpuart_data_first_config**

<table>
<thead>
<tr>
<th>Function name</th>
<th>lpuart_data_first_config</th>
</tr>
</thead>
<tbody>
<tr>
<td>Function prototype</td>
<td>void lpuart_data_first_config(uint32_t msbf);</td>
</tr>
<tr>
<td>Function descriptions</td>
<td>data is transmitted/received with the LSB/MSB first</td>
</tr>
<tr>
<td>Precondition</td>
<td>-</td>
</tr>
<tr>
<td>The called functions</td>
<td>-</td>
</tr>
</tbody>
</table>
GD32L23x Firmware Library User Guide

### Input parameter (in)

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>msbf</td>
<td>LSB/MSB</td>
</tr>
<tr>
<td>LPUART_MSBF_LSB</td>
<td>LSB first</td>
</tr>
<tr>
<td>LPUART_MSBF_MSB</td>
<td>MSB first</td>
</tr>
</tbody>
</table>

### Output parameter (out)

- -

### Return value

- -

Example:

```c
/* configure LSB of data first */

lpuart_data_first_config(LPUART_MSBF_LSB);
```

#### lpuart_invert_config

The description of lpuart_invert_config is shown as below:

**Table 3-436. Function lpuart_invert_config**

<table>
<thead>
<tr>
<th>Function name</th>
<th>lpuart_invert_config</th>
</tr>
</thead>
<tbody>
<tr>
<td>Function prototype</td>
<td>void lpuart_invert_config(lpuart_invert_enum invertpara);</td>
</tr>
<tr>
<td>Function descriptions</td>
<td>configure LPUART inverted</td>
</tr>
<tr>
<td>Precondition</td>
<td>-</td>
</tr>
<tr>
<td>The called functions</td>
<td>-</td>
</tr>
</tbody>
</table>

### Input parameter (in)

<table>
<thead>
<tr>
<th>invertpara</th>
<th>refer to Table 3-425, Enum lpuart_invert_enum</th>
</tr>
</thead>
<tbody>
<tr>
<td>LPUART_DINV_ENAB</td>
<td>data bit level inversion</td>
</tr>
<tr>
<td>LPUART_DINV_DISAB</td>
<td>data bit level not inversion</td>
</tr>
<tr>
<td>LPUART_TXPIN_ENAB</td>
<td>TX pin level inversion</td>
</tr>
<tr>
<td>LPUART_TXPIN_DISAB</td>
<td>TX pin level not inversion</td>
</tr>
<tr>
<td>LPUART_RXPIN_ENAB</td>
<td>RX pin level inversion</td>
</tr>
<tr>
<td>LPUART_RXPIN_DISAB</td>
<td>RX pin level not inversion</td>
</tr>
<tr>
<td>LPUART_SWAP_ENAB</td>
<td>swap TX/RX pins</td>
</tr>
<tr>
<td>LPUART_SWAP_DISAB</td>
<td>not swap TX/RX pins</td>
</tr>
</tbody>
</table>

### Output parameter (out)

- -
Example:

/* configure LPUART inversion */
lpuart_invert_config(LPUART_DINV_ENABLE);

**lpuart_overrun_enable**

The description of `lpuart_overrun_enable` is shown as below:

<table>
<thead>
<tr>
<th>Function name</th>
<th><code>lpuart_overrun_enable</code></th>
</tr>
</thead>
<tbody>
<tr>
<td>Function prototype</td>
<td><code>void lpuart_overrun_enable(void);</code></td>
</tr>
<tr>
<td>Function descriptions</td>
<td>enable the LPUART overrun</td>
</tr>
<tr>
<td>Precondition</td>
<td>-</td>
</tr>
<tr>
<td>The called functions</td>
<td>-</td>
</tr>
<tr>
<td>Input parameter(in)</td>
<td>-</td>
</tr>
<tr>
<td>Output parameter(out)</td>
<td>-</td>
</tr>
<tr>
<td>Return value</td>
<td>-</td>
</tr>
</tbody>
</table>

Example:

/* enable LPUART overrun */
lpuart_overrun_enable();

**lpuart_overrun_disable**

The description of `lpuart_overrun_disable` is shown as below:

<table>
<thead>
<tr>
<th>Function name</th>
<th><code>lpuart_overrun_disable</code></th>
</tr>
</thead>
<tbody>
<tr>
<td>Function prototype</td>
<td><code>void lpuart_overrun_disable(void);</code></td>
</tr>
<tr>
<td>Function descriptions</td>
<td>disable the LPUART overrun function</td>
</tr>
<tr>
<td>Precondition</td>
<td>-</td>
</tr>
<tr>
<td>The called functions</td>
<td>-</td>
</tr>
<tr>
<td>Input parameter(in)</td>
<td>-</td>
</tr>
<tr>
<td>Output parameter(out)</td>
<td>-</td>
</tr>
<tr>
<td>Return value</td>
<td>-</td>
</tr>
</tbody>
</table>
Example:

/* disable LPUART overrun */
lpuart_overrun_disable();

**lpuart_data_transmit**

The description of lpuart_data_transmit is shown as below:

<table>
<thead>
<tr>
<th>Table 3-439. Function lpuart_data_transmit</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Function name</strong></td>
</tr>
<tr>
<td><strong>Function prototype</strong></td>
</tr>
<tr>
<td><strong>Function descriptions</strong></td>
</tr>
<tr>
<td><strong>Precondition</strong></td>
</tr>
<tr>
<td><strong>The called functions</strong></td>
</tr>
<tr>
<td><strong>Input parameter</strong>(in)</td>
</tr>
<tr>
<td><strong>data</strong></td>
</tr>
<tr>
<td><strong>Output parameter</strong>(out)</td>
</tr>
<tr>
<td><strong>Return value</strong></td>
</tr>
</tbody>
</table>

Example:

/* LPUART transmit data */
lpuart_data_transmit(0xA0);

**lpuart_data_receive**

The description of lpuart_data_receive is shown as below:

<table>
<thead>
<tr>
<th>Table 3-440. Function lpuart_data_receive</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Function name</strong></td>
</tr>
<tr>
<td><strong>Function prototype</strong></td>
</tr>
<tr>
<td><strong>Function descriptions</strong></td>
</tr>
<tr>
<td><strong>Precondition</strong></td>
</tr>
<tr>
<td><strong>The called functions</strong></td>
</tr>
<tr>
<td><strong>Input parameter</strong>(in)</td>
</tr>
<tr>
<td><strong>Output parameter</strong>(out)</td>
</tr>
<tr>
<td><strong>Return value</strong></td>
</tr>
<tr>
<td><strong>data</strong></td>
</tr>
</tbody>
</table>
Example:

/* LPUART receive data */

uint16_t temp;

temp = Lpuart_data_receive();

**Ipuart_command_enable**

The description of Ipuart_command_enable is shown as below:

**Table 3-441. Function Ipuart_command_enable**

<table>
<thead>
<tr>
<th>Function name</th>
<th>Ipuart_command_enable</th>
</tr>
</thead>
<tbody>
<tr>
<td>Function prototype</td>
<td>void Ipuart_command_enable(uint32_t cmdtype);</td>
</tr>
<tr>
<td>Function descriptions</td>
<td>enable LPUART command</td>
</tr>
<tr>
<td>Precondition</td>
<td>-</td>
</tr>
<tr>
<td>The called functions</td>
<td>-</td>
</tr>
</tbody>
</table>

**Input parameter(in)**

<table>
<thead>
<tr>
<th>cmdtype</th>
<th>command type</th>
</tr>
</thead>
<tbody>
<tr>
<td>LPUART_CMD_MMCMD</td>
<td>mute mode command</td>
</tr>
<tr>
<td>LPUART_CMD_RXFCMD</td>
<td>receive data flush command</td>
</tr>
</tbody>
</table>

**Output parameter(out)**

| -                        | -                             |

**Return value**

Example:

/* enable LPUART command */

Ipuart_command_enable(LPUART_CMD_ABDCMD);

**Ipuart_address_config**

The description of Ipuart_address_config is shown as below:

**Table 3-442. Function Ipuart_address_config**

<table>
<thead>
<tr>
<th>Function name</th>
<th>Ipuart_address_config</th>
</tr>
</thead>
<tbody>
<tr>
<td>Function prototype</td>
<td>void Ipuart_address_config(uint8_t addr);</td>
</tr>
<tr>
<td>Function descriptions</td>
<td>configure address of the LPUART</td>
</tr>
<tr>
<td>Precondition</td>
<td>-</td>
</tr>
<tr>
<td>The called functions</td>
<td>-</td>
</tr>
</tbody>
</table>

**Input parameter(in)**

| addr                    | address of LPUART (0-0xFF) |

**Output parameter(out)**
Example:

```c
/* configure address of the LPUART */

lpuart_address_config(0x00);
```

**Lpuart_address_detection_mode_config**

The description of `lpuart_address_detection_mode_config` is shown as below:

<table>
<thead>
<tr>
<th>Table 3-443. Function <code>lpuart_address_detection_mode_config</code></th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Function name</strong></td>
</tr>
<tr>
<td><strong>Function prototype</strong></td>
</tr>
<tr>
<td><strong>Function descriptions</strong></td>
</tr>
<tr>
<td><strong>Precondition</strong></td>
</tr>
<tr>
<td><strong>The called functions</strong></td>
</tr>
<tr>
<td><strong>Input parameter(in)</strong></td>
</tr>
<tr>
<td><code>addmod</code></td>
</tr>
<tr>
<td><code>LPUART_ADDM_4BIT</code></td>
</tr>
<tr>
<td><code>LPUART_ADDM_FULL_BIT</code></td>
</tr>
<tr>
<td><strong>Output parameter(out)</strong></td>
</tr>
<tr>
<td><strong>Return value</strong></td>
</tr>
</tbody>
</table>

Example:

```c
/* configure address detection mode */

lpuart_address_config(LPUART_ADDM_4BIT);
```

**Lpuart_mute_mode_enable**

The description of `lpuart_mute_mode_enable` is shown as below:

<table>
<thead>
<tr>
<th>Table 3-444. Function <code>lpuart_mute_mode_enable</code></th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Function name</strong></td>
</tr>
<tr>
<td><strong>Function prototype</strong></td>
</tr>
<tr>
<td><strong>Function descriptions</strong></td>
</tr>
<tr>
<td><strong>Precondition</strong></td>
</tr>
<tr>
<td><strong>The called functions</strong></td>
</tr>
<tr>
<td><strong>Input parameter(in)</strong></td>
</tr>
</tbody>
</table>

Example:

/* enable LPUART receiver in mute mode */

lpuart_mute_mode_enable();

**lpuart_mute_mode_disable**

The description of lpuart_mute_mode_disable is shown as below:

Table 3-445. Function lpuart_mute_mode_disable

<table>
<thead>
<tr>
<th>Function name</th>
<th>lpuart_mute_mode_disable</th>
</tr>
</thead>
<tbody>
<tr>
<td>Function prototype</td>
<td>void lpuart_mute_mode_disable(void);</td>
</tr>
<tr>
<td>Function descriptions</td>
<td>disable mute mode</td>
</tr>
<tr>
<td>Precondition</td>
<td>-</td>
</tr>
<tr>
<td>The called functions</td>
<td>-</td>
</tr>
</tbody>
</table>

| Input parameter(in)           | -                                              |
| Output parameter(out)         | -                                              |
| Return value                  | -                                              |

Example:

/* disable LPUART receiver in mute mode */

lpuart_mute_mode_disable();

**lpuart_mute_mode_wakeup_config**

The description of lpuart_mute_mode_wakeup_config is shown as below:

Table 3-446. Function lpuart_mute_mode_wakeup_config

<table>
<thead>
<tr>
<th>Function name</th>
<th>lpuart_mute_mode_wakeup_config</th>
</tr>
</thead>
<tbody>
<tr>
<td>Function prototype</td>
<td>void lpuart_mute_mode_wakeup_config(uint32_t wmethod);</td>
</tr>
<tr>
<td>Function descriptions</td>
<td>configure wakeup method in mute mode</td>
</tr>
<tr>
<td>Precondition</td>
<td>-</td>
</tr>
<tr>
<td>The called functions</td>
<td>-</td>
</tr>
</tbody>
</table>

| Input parameter(in)           | wmethod                                        |
| Output parameter(out)         | two methods be used to enter or exit the mute mode |
| Return value                  | -                                              |
LPUART_WM_IDLE  idle line
LPUART_WM_ADDR  address mask

Output parameter(out)
- -

Return value
- -

Example:

/* configure LPUART wakeup method in mute mode */
luart_mute_mode_wakeup_config(LPUART_WM_IDLE);

Ipuart_halfduplex_enable

The description of Ipuart_halfduplex_enable is shown as below:

Table 3-447. Function Ipuart_halfduplex_enable

<table>
<thead>
<tr>
<th>Function name</th>
<th>Ipuart_halfduplex_enable</th>
</tr>
</thead>
<tbody>
<tr>
<td>Function prototype</td>
<td>void Ipuart_halfduplex_enable(void);</td>
</tr>
<tr>
<td>Function descriptions</td>
<td>enable half-duplex mode</td>
</tr>
<tr>
<td>Precondition</td>
<td>-</td>
</tr>
<tr>
<td>The called functions</td>
<td>-</td>
</tr>
<tr>
<td>Input parameter(in)</td>
<td>-</td>
</tr>
<tr>
<td>Output parameter(out)</td>
<td>-</td>
</tr>
<tr>
<td>Return value</td>
<td>-</td>
</tr>
</tbody>
</table>

Example:

/* enable LPUART half duplex mode*/
luart_halfduplex_enable();

Ipuart_halfduplex_disable

The description of Ipuart_halfduplex_disable is shown as below:

Table 3-448. Function Ipuart_halfduplex_disable

<table>
<thead>
<tr>
<th>Function name</th>
<th>Ipuart_halfduplex_disable</th>
</tr>
</thead>
<tbody>
<tr>
<td>Function prototype</td>
<td>void Ipuart_halfduplex_disable(void);</td>
</tr>
<tr>
<td>Function descriptions</td>
<td>disable half-duplex mode</td>
</tr>
<tr>
<td>Precondition</td>
<td>-</td>
</tr>
<tr>
<td>The called functions</td>
<td>-</td>
</tr>
<tr>
<td>Input parameter(in)</td>
<td>-</td>
</tr>
</tbody>
</table>
Output parameter(output)

Return value

Example:

/* disable LPUART half duplex mode*/

lpuart_halfduplex_disable();

**Ipuart_hardware_flow_rts_config**

The description of Ipuart_hardware_flow_rts_config is shown as below:

<table>
<thead>
<tr>
<th>Function name</th>
<th>Ipuart_hardware_flow_rts_config</th>
</tr>
</thead>
<tbody>
<tr>
<td>Function prototype</td>
<td>void Ipuart_hardware_flow_rts_config(uint32_t rtsconfig);</td>
</tr>
<tr>
<td>Function descriptions</td>
<td>configure hardware flow control RTS</td>
</tr>
<tr>
<td>Precondition</td>
<td>-</td>
</tr>
<tr>
<td>The called functions</td>
<td>-</td>
</tr>
<tr>
<td>Input parameter(in)</td>
<td>-</td>
</tr>
<tr>
<td>Input parameter(in)</td>
<td>rtsconfig enable or disable RTS</td>
</tr>
<tr>
<td>LPUART_RTS_ENABLE</td>
<td>enable RTS</td>
</tr>
<tr>
<td>LPUART_RTS_DISABLE</td>
<td>disable RTS</td>
</tr>
<tr>
<td>Output parameter(out)</td>
<td>-</td>
</tr>
<tr>
<td>Return value</td>
<td>-</td>
</tr>
</tbody>
</table>

Example:

/* configure LPUART hardware flow control RTS */

lpuart_hardware_flow_rts_config(LPUART_RTS_ENABLE);

**Ipuart_hardware_flow_cts_config**

The description of Ipuart_hardware_flow_cts_config is shown as below:

<table>
<thead>
<tr>
<th>Function name</th>
<th>Ipuart_hardware_flow_cts_config</th>
</tr>
</thead>
<tbody>
<tr>
<td>Function prototype</td>
<td>void Ipuart_hardware_flow_cts_config(uint32_t ctsconfig);</td>
</tr>
</tbody>
</table>
### Function descriptions
- configure hardware flow control CTS

### Precondition
- 

### The called functions
- 

### Input parameter (in)
- 

### Input parameter (in)
- 

### ctsconfig
- enable or disable CTS

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>LPUART_CTS_ENABL</strong></td>
<td>enable CTS</td>
</tr>
<tr>
<td><strong>LPUART_CTS_DISABL</strong></td>
<td>disable CTS</td>
</tr>
</tbody>
</table>

### Output parameter (out)
- 

### Return value
- 

**Example:**

```c
/* configure LPUART hardware flow control CTS */
lpuart_hardware_flow_cts_config(LPUART_CTS_ENABLE);
```

### lpuart_hardware_flow_coherence_config

The description of `lpuart_hardware_flow_coherence_config` is shown as below:

**Table 3-451. Function lpuart_hardware_flow_coherence_config**

<table>
<thead>
<tr>
<th>Function name</th>
<th>lpuart_hardware_flow_coherence_config</th>
</tr>
</thead>
<tbody>
<tr>
<td>Function prototype</td>
<td>void lpuart_hardware_flow_coherence_config(uint32_t hcm);</td>
</tr>
<tr>
<td>Function descriptions</td>
<td>configure hardware flow control coherence mode</td>
</tr>
<tr>
<td>Precondition</td>
<td>-</td>
</tr>
<tr>
<td>The called functions</td>
<td>-</td>
</tr>
</tbody>
</table>

### Input parameter (in)
- 

### hcm
- Hardware flow control coherence mode

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>LPUART_HCM_NONE</strong></td>
<td>nRTS signal equals to the RBNE status register</td>
</tr>
<tr>
<td><strong>LPUART_HCM_EN</strong></td>
<td>nRTS signal is set when the last data bit has been sampled</td>
</tr>
</tbody>
</table>

### Output parameter (out)
- 

### Return value
- 

**Example:**

```c
/* configure hardware flow control coherence mode */
lpuart_hardware_flow_coherence_config(LPUART_HCM_NONE);
```
Ipuart_rs485_driver_enable

The description of Ipuart_rs485_driver_enable is shown as below:

**Table 3-452. Function Ipuart_rs485_driver_enable**

<table>
<thead>
<tr>
<th>Function name</th>
<th>Ipuart_rs485_driver_enable</th>
</tr>
</thead>
<tbody>
<tr>
<td>Function prototype</td>
<td>void Ipuart_rs485_driver_enable(void);</td>
</tr>
<tr>
<td>Function descriptions</td>
<td>enable RS485 driver</td>
</tr>
<tr>
<td>Precondition</td>
<td>-</td>
</tr>
<tr>
<td>The called functions</td>
<td>-</td>
</tr>
<tr>
<td>Input parameter(in)</td>
<td>-</td>
</tr>
<tr>
<td>Output parameter(out)</td>
<td>-</td>
</tr>
<tr>
<td>Return value</td>
<td>-</td>
</tr>
</tbody>
</table>

Example:

/* enable LPUART RS485 driver */
Ipuart_rs485_driver_enable();

Ipuart_rs485_driver_disable

The description of Ipuart_rs485_driver_disable is shown as below:

**Table 3-453. Function Ipuart_rs485_driver_disable**

<table>
<thead>
<tr>
<th>Function name</th>
<th>Ipuart_rs485_driver_disable</th>
</tr>
</thead>
<tbody>
<tr>
<td>Function prototype</td>
<td>void Ipuart_rs485_driver_disable(void);</td>
</tr>
<tr>
<td>Function descriptions</td>
<td>disable RS485 driver</td>
</tr>
<tr>
<td>Precondition</td>
<td>-</td>
</tr>
<tr>
<td>The called functions</td>
<td>-</td>
</tr>
<tr>
<td>Input parameter(in)</td>
<td>-</td>
</tr>
<tr>
<td>Output parameter(out)</td>
<td>-</td>
</tr>
<tr>
<td>Return value</td>
<td>-</td>
</tr>
</tbody>
</table>

Example:

/* disable LPUART RS485 driver */
Ipuart_rs485_driver_disable();
Ipuart_driver_assertime_config

The description of lpuart_driver_assertime_config is shown as below:

<table>
<thead>
<tr>
<th>Function name</th>
<th>lpuart_driver_assertime_config</th>
</tr>
</thead>
<tbody>
<tr>
<td>Function prototype</td>
<td>void lpuart_driver_assertime_config(uint32_t deatime);</td>
</tr>
<tr>
<td>Function descriptions</td>
<td>configure driver enable assertion time</td>
</tr>
<tr>
<td>Precondition</td>
<td>-</td>
</tr>
<tr>
<td>The called functions</td>
<td>-</td>
</tr>
</tbody>
</table>

**Input parameter (in)**

deatime: driver enable assertion time (0x00-0x0000001F)

**Output parameter (out)**

- -

Return value -

Example:

/* set LPUART driver assertime */

lpuart_driver_assertime_config(0x0000001F);

Ipuart_driver_deassertime_config

The description of lpuart_driver_deassertime_config is shown as below:

<table>
<thead>
<tr>
<th>Function name</th>
<th>lpuart_driver_deassertime_config</th>
</tr>
</thead>
<tbody>
<tr>
<td>Function prototype</td>
<td>void lpuart_driver_deassertime_config(uint32_t dedtime);</td>
</tr>
<tr>
<td>Function descriptions</td>
<td>configure driver enable de-assertion time</td>
</tr>
<tr>
<td>Precondition</td>
<td>-</td>
</tr>
<tr>
<td>The called functions</td>
<td>-</td>
</tr>
</tbody>
</table>

**Input parameter (in)**

dedtime: driver enable de-assertion time (0x00-0x0000001F)

**Output parameter (out)**

- -

Return value -

Example:

/* set LPUART driver deassertime */

lpuart_driver_deassertime_config(0x0000001F);
Ipuart_depolarity_config

The description of Ipuart_depolarity_config is shown as below:

Table 3-456. Function Ipuart_depolarity_config

<table>
<thead>
<tr>
<th>Function name</th>
<th>Ipuart_depolarity_config</th>
</tr>
</thead>
<tbody>
<tr>
<td>Function prototype</td>
<td>void Ipuart_depolarity_config(uint32_t dep);</td>
</tr>
<tr>
<td>Function descriptions</td>
<td>configure driver enable polarity mode</td>
</tr>
<tr>
<td>Precondition</td>
<td>-</td>
</tr>
<tr>
<td>The called functions</td>
<td>-</td>
</tr>
</tbody>
</table>

Input parameter(in)

<table>
<thead>
<tr>
<th>dep</th>
<th>DE signal</th>
</tr>
</thead>
<tbody>
<tr>
<td>LPUART_DEP_HIGH</td>
<td>DE signal is active high</td>
</tr>
<tr>
<td>LPUART_DEP_LOW</td>
<td>DE signal is active low</td>
</tr>
</tbody>
</table>

Output parameter(out)

- |

Return value

Example:

/* configure driver enable polarity mode */

Ipuart_driver_depolarity_config(LPUART_DEP_HIGH);

Ipuart_dma_receive_config

The description of Ipuart_dma_receive_config is shown as below:

Table 3-457. Function Ipuart_dma_receive_config

<table>
<thead>
<tr>
<th>Function name</th>
<th>Ipuart_dma_receive_config</th>
</tr>
</thead>
<tbody>
<tr>
<td>Function prototype</td>
<td>void Ipuart_dma_receive_config(uint32_t dmacmd);</td>
</tr>
<tr>
<td>Function descriptions</td>
<td>configure LPUART DMA for reception</td>
</tr>
<tr>
<td>Precondition</td>
<td>-</td>
</tr>
<tr>
<td>The called functions</td>
<td>-</td>
</tr>
</tbody>
</table>

Input parameter(in)

<table>
<thead>
<tr>
<th>dmacmd</th>
<th>enable or disable DMA for reception</th>
</tr>
</thead>
<tbody>
<tr>
<td>LPUART_DENR_ENABLE</td>
<td>DMA enable for reception</td>
</tr>
<tr>
<td>LPUART_DENR_DISABLE</td>
<td>DMA disable for reception</td>
</tr>
</tbody>
</table>

Output parameter(out)

- |

Return value

-
Example:

/* LPUART DMA enable for reception */
lpuart_dma_receive_config(LPUART_DENR_ENABLE);

**lpuart_dma_transmit_config**

The description of `lpuart_dma_transmit_config` is shown as below:

**Table 3-458. Function lpuart_dma_transmit_config**

<table>
<thead>
<tr>
<th>Function name</th>
<th>lpuart_dma_transmit_config</th>
</tr>
</thead>
<tbody>
<tr>
<td>Function prototype</td>
<td>void lpuart_dma_transmit_config(uint32_t dmacmd);</td>
</tr>
<tr>
<td>Function descriptions</td>
<td>configure LPUART DMA for transmission</td>
</tr>
<tr>
<td>Precondition</td>
<td>-</td>
</tr>
<tr>
<td>The called functions</td>
<td>-</td>
</tr>
</tbody>
</table>

**Input parameter(in)**

- dmacmd: enable or disable DMA for transmission
- `LPUART_DENT_ENABLE`: DMA enable for transmission
- `LPUART_DENT_DISABLE`: DMA disable for transmission

| Output parameter(out)         | -                        |
| Return value                  | -                        |

Example:

/* LPUART DMA enable for transmission */
lpuart_dma_transmit_config(LPUART_DENT_ENABLE);

**lpuart_reception_error_dma_disable**

The description of `lpuart_reception_error_dma_disable` is shown as below:

**Table 3-459. Function lpuart_reception_error_dma_disable**

<table>
<thead>
<tr>
<th>Function name</th>
<th>lpuart_reception_error_dma_disable</th>
</tr>
</thead>
<tbody>
<tr>
<td>Function prototype</td>
<td>void lpuart_reception_error_dma_disable(void);</td>
</tr>
<tr>
<td>Function descriptions</td>
<td>disable DMA on reception error</td>
</tr>
<tr>
<td>Precondition</td>
<td>-</td>
</tr>
<tr>
<td>The called functions</td>
<td>-</td>
</tr>
</tbody>
</table>

**Input parameter(in)**

- -

**Output parameter(out)**

- -
Example:

/* disable DMA on reception error */
lpuart_reception_error_dma_disable();

**lpuart_reception_error_dma_enable**

The description of lpuart_reception_error_dma_enable is shown as below:

Table 3-460. Function lpuart_reception_error_dma_enable

<table>
<thead>
<tr>
<th>Function name</th>
<th>lpuart_reception_error_dma_enable</th>
</tr>
</thead>
<tbody>
<tr>
<td>Function prototype</td>
<td>void lpuart_reception_error_dma_enable(void);</td>
</tr>
<tr>
<td>Function descriptions</td>
<td>enable DMA on reception error</td>
</tr>
<tr>
<td>Precondition</td>
<td>-</td>
</tr>
<tr>
<td>The called functions</td>
<td>-</td>
</tr>
<tr>
<td>Input parameter(in)</td>
<td>-</td>
</tr>
<tr>
<td>Output parameter(out)</td>
<td>-</td>
</tr>
<tr>
<td>Return value</td>
<td>-</td>
</tr>
</tbody>
</table>

Example:

/* enable DMA on reception error */
lpuart_reception_error_dma_enable();

**lpuart_wakeup_enable**

The description of lpuart_wakeup_enable is shown as below:

Table 3-461. Function lpuart_wakeup_enable

<table>
<thead>
<tr>
<th>Function name</th>
<th>lpuart_wakeup_enable</th>
</tr>
</thead>
<tbody>
<tr>
<td>Function prototype</td>
<td>void lpuart_wakeup_enable(void);</td>
</tr>
<tr>
<td>Function descriptions</td>
<td>enable LPUART to wakeup the mcu from deep-sleep mode</td>
</tr>
<tr>
<td>Precondition</td>
<td>-</td>
</tr>
<tr>
<td>The called functions</td>
<td>-</td>
</tr>
<tr>
<td>Input parameter(in)</td>
<td>-</td>
</tr>
<tr>
<td>Output parameter(out)</td>
<td>-</td>
</tr>
<tr>
<td>Return value</td>
<td>-</td>
</tr>
</tbody>
</table>
Example:

/* LPUART wake up enable */

Ipuart_wakeup_enable();

**Ipuart_wakeup_disable**

The description of `Ipuart_wakeup_disable` is shown as below:

<table>
<thead>
<tr>
<th>Table 3-462. Function <code>Ipuart_wakeup_disable</code></th>
</tr>
</thead>
<tbody>
<tr>
<td>Function name</td>
</tr>
<tr>
<td>Function prototype</td>
</tr>
<tr>
<td>Function descriptions</td>
</tr>
<tr>
<td>Precondition</td>
</tr>
<tr>
<td>The called functions</td>
</tr>
<tr>
<td>Input parameter(in)</td>
</tr>
<tr>
<td>Output parameter(out)</td>
</tr>
<tr>
<td>Return value</td>
</tr>
</tbody>
</table>

Example:

/* LPUART wake up disable */

Ipuart_wakeup_disable();

**Ipuart_wakeup_mode_config**

The description of `Ipuart_wakeup_mode_config` is shown as below:

<table>
<thead>
<tr>
<th>Table 3-463. Function <code>Ipuart_wakeup_mode_config</code></th>
</tr>
</thead>
<tbody>
<tr>
<td>Function name</td>
</tr>
<tr>
<td>Function prototype</td>
</tr>
<tr>
<td>Function descriptions</td>
</tr>
<tr>
<td>Precondition</td>
</tr>
<tr>
<td>The called functions</td>
</tr>
<tr>
<td>Input parameter(in)</td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
</tbody>
</table>
GD32L23x Firmware Library User Guide

<table>
<thead>
<tr>
<th>Output parameter(out)</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>-</td>
<td>-</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Return value</th>
</tr>
</thead>
<tbody>
<tr>
<td>-</td>
</tr>
</tbody>
</table>

Example:

/* configure LPUART wake up mode */

lpuart_wakeup_mode_config(LPUART_WUM_ADDR);

### lpuart_flag_get

The description of lpuart_flag_get is shown as below:

#### Table 3-464. Function lpuart_flag_get

<table>
<thead>
<tr>
<th>Function name</th>
<th>lpuart_flag_get</th>
</tr>
</thead>
<tbody>
<tr>
<td>Function prototype</td>
<td>FlagStatus lpuart_flag_get(lpuart_flag_enum flag);</td>
</tr>
<tr>
<td>Function descriptions</td>
<td>get flag in STAT/CHC register</td>
</tr>
<tr>
<td>Precondition</td>
<td>-</td>
</tr>
<tr>
<td>The called functions</td>
<td>-</td>
</tr>
</tbody>
</table>

#### Input parameter(in)

- |

#### Input parameter(in)

- |

#### flag

<table>
<thead>
<tr>
<th>LPUART_FLAG_PERR</th>
<th>parity error flag</th>
</tr>
</thead>
<tbody>
<tr>
<td>LPUART_FLAG_FERR</td>
<td>frame error flag</td>
</tr>
<tr>
<td>LPUART_FLAG_NERR</td>
<td>noise error flag</td>
</tr>
<tr>
<td>LPUART_FLAG_ORER</td>
<td>overrun error</td>
</tr>
<tr>
<td>LPUART_FLAG_IDLE</td>
<td>idle line detected flag</td>
</tr>
<tr>
<td>LPUART_FLAG_RBNE</td>
<td>read data buffer not empty</td>
</tr>
<tr>
<td>LPUART_FLAG_TC</td>
<td>transmission completed</td>
</tr>
<tr>
<td>LPUART_FLAG_TBE</td>
<td>transmit data register empty</td>
</tr>
<tr>
<td>LPUART_FLAG_CTSF</td>
<td>CTS change flag</td>
</tr>
<tr>
<td>LPUART_FLAG_CTS</td>
<td>CTS level</td>
</tr>
<tr>
<td>LPUART_FLAG_BSY</td>
<td>busy flag</td>
</tr>
<tr>
<td>LPUART_FLAG_AM</td>
<td>address match flag</td>
</tr>
<tr>
<td>LPUART_FLAG_RWU</td>
<td>receiver wakeup from mute mode</td>
</tr>
<tr>
<td>LPUART_FLAG_WU</td>
<td>wakeup from deep-sleep mode flag</td>
</tr>
<tr>
<td>LPUART_FLAG_TEA</td>
<td>transmit enable acknowledge flag</td>
</tr>
<tr>
<td>LPUART_FLAG_REA</td>
<td>receive enable acknowledge flag</td>
</tr>
<tr>
<td>LPUART_FLAG_EPER</td>
<td>early parity error flag</td>
</tr>
</tbody>
</table>

*Table 3-422. Enum lpuart_flag_enum* only one among these parameters can be selected
Output parameter (out)

-  

Return value

FlagStatus | SET or RESET

Example:

/* get flag LPUART state */

FlagStatus status;

status = lpuart_flag_get(LPUART_FLAG_TBE);

**Ipuart_flag_clear**

The description of lpuart_flag_clear is shown as below:

<table>
<thead>
<tr>
<th>Function name</th>
<th>lpuart_flag_clear</th>
</tr>
</thead>
<tbody>
<tr>
<td>Function prototype</td>
<td>void lpuart_flag_clear(lpuart_flag_enum flag);</td>
</tr>
<tr>
<td>Function descriptions</td>
<td>clear LPUART status</td>
</tr>
<tr>
<td>Precondition</td>
<td>-</td>
</tr>
<tr>
<td>The called functions</td>
<td>-</td>
</tr>
</tbody>
</table>

**Table 3-465. Function lpuart_flag_clear**

<table>
<thead>
<tr>
<th>flag</th>
<th>LPUART flags, refer to <a href="#">Table 3-422. Enum lpuart_flag_enum</a> only one among these parameters can be selected</th>
</tr>
</thead>
<tbody>
<tr>
<td>LPUART_FLAG_PERR</td>
<td>parity error flag</td>
</tr>
<tr>
<td>LPUART_FLAG_FERR</td>
<td>frame error flag</td>
</tr>
<tr>
<td>LPUART_FLAG_NERR</td>
<td>noise detected flag</td>
</tr>
<tr>
<td>LPUART_FLAG_ORER</td>
<td>overrun error flag</td>
</tr>
<tr>
<td>LPUART_FLAG_IDLE</td>
<td>idle line detected flag</td>
</tr>
<tr>
<td>LPUART_FLAG_TC</td>
<td>transmission complete flag</td>
</tr>
<tr>
<td>LPUART_FLAG_CTSF</td>
<td>CTS change flag</td>
</tr>
<tr>
<td>LPUART_FLAG_AM</td>
<td>address match flag</td>
</tr>
<tr>
<td>LPUART_FLAG_WU</td>
<td>wakeup from deep-sleep mode flag</td>
</tr>
<tr>
<td>LPUART_FLAG_EPER</td>
<td>early parity error flag</td>
</tr>
</tbody>
</table>

Input parameter (in)

<table>
<thead>
<tr>
<th>flag</th>
<th></th>
</tr>
</thead>
</table>

Output parameter (out)

-  

Return value

-  

Example:

/* clear LPUART flag */
Ipuart_flag_clear(LPUART_FLAG_TC);

Ipuart_interrupt_enable

The description of Ipuart_interrupt_enable is shown as below:

Table 3-466. Function Ipuart_interrupt_enable

<table>
<thead>
<tr>
<th>Function name</th>
<th>Ipuart_interrupt_enable</th>
</tr>
</thead>
<tbody>
<tr>
<td>Function prototype</td>
<td>void Ipuart_interrupt_enable(Ipuart_interrupt_enum interrupt);</td>
</tr>
<tr>
<td>Function descriptions</td>
<td>enable LPUART interrupt</td>
</tr>
<tr>
<td>Precondition</td>
<td>-</td>
</tr>
<tr>
<td>The called functions</td>
<td>-</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Input parameter(in)</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>interrupt</td>
<td></td>
</tr>
<tr>
<td>LPUART_INT_IDLE</td>
<td>idle interrupt</td>
</tr>
<tr>
<td>LPUART_INT_RBNE</td>
<td>read data buffer not empty interrupt and overrun error interrupt enable interrupt</td>
</tr>
<tr>
<td>LPUART_INT_TC</td>
<td>transmission complete interrupt</td>
</tr>
<tr>
<td>LPUART_INT_TBE</td>
<td>transmit data register empty interrupt</td>
</tr>
<tr>
<td>LPUART_INT_PERR</td>
<td>parity error interrupt</td>
</tr>
<tr>
<td>LPUART_INT_AM</td>
<td>address match interrupt</td>
</tr>
<tr>
<td>LPUART_INT_ERR</td>
<td>error interrupt enable in multibuffer communication</td>
</tr>
<tr>
<td>LPUART_INT_CTS</td>
<td>CTS interrupt</td>
</tr>
<tr>
<td>LPUART_INT_WU</td>
<td>wakeup from deep-sleep mode interrupt</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Output parameter(out)</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>-</td>
<td></td>
</tr>
</tbody>
</table>

| Return value                   | -                     |

Example:

/* enable LPUART TBE interrupt */

Ipuart_interrupt_enable(LPUART_INT_TBE);

Ipuart_interrupt_disable

The description of Ipuart_interrupt_disable is shown as below:

Table 3-467. Function Ipuart_interrupt_disable

<table>
<thead>
<tr>
<th>Function name</th>
<th>Ipuart_interrupt_disable</th>
</tr>
</thead>
<tbody>
<tr>
<td>Function prototype</td>
<td>void Ipuart_interrupt_disable(Ipuart_interrupt_enum interrupt);</td>
</tr>
<tr>
<td>Function descriptions</td>
<td>disable LPUART interrupt</td>
</tr>
<tr>
<td>Precondition</td>
<td>-</td>
</tr>
<tr>
<td>The called functions</td>
<td>-</td>
</tr>
</tbody>
</table>
GD32L23x Firmware Library User Guide

### Input parameter(in)

<table>
<thead>
<tr>
<th>interrupt</th>
<th>description</th>
</tr>
</thead>
<tbody>
<tr>
<td>LPUART_INT_IDLE</td>
<td>idle interrupt</td>
</tr>
<tr>
<td>LPUART_INT_RBNE</td>
<td>read data buffer not empty interrupt and overrun error interrupt enable interrupt</td>
</tr>
<tr>
<td>LPUART_INT_TC</td>
<td>transmission complete interrupt</td>
</tr>
<tr>
<td>LPUART_INT_TBE</td>
<td>transmit data register empty interrupt</td>
</tr>
<tr>
<td>LPUART_INT_PERR</td>
<td>parity error interrupt</td>
</tr>
<tr>
<td>LPUART_INT_AM</td>
<td>address match interrupt</td>
</tr>
<tr>
<td>LPUART_INT_ERR</td>
<td>error interrupt enable in multibuffer communication</td>
</tr>
<tr>
<td>LPUART_INT_CTS</td>
<td>CTS interrupt</td>
</tr>
<tr>
<td>LPUART_INT_WU</td>
<td>wakeup from deep-sleep mode interrupt</td>
</tr>
</tbody>
</table>

### Output parameter(out)

| - | - |

### Return value

| - | - |

Example:

```c
/* disable LPUART TBE interrupt */

lpuart_interrupt_disable(LPUART_INT_TBE);
```

#### Lpuart_interrupt_flag_get

The description of `lpuart_interrupt_flag_get` is shown as below:

### Table 3-468. Function `lpuart_interrupt_flag_get`

<table>
<thead>
<tr>
<th>Function name</th>
<th><code>lpuart_interrupt_flag_get</code></th>
</tr>
</thead>
<tbody>
<tr>
<td>Function prototype</td>
<td><code>FlagStatus lpuart_interrupt_flag_get(lpuart_interrupt_flag_enum int_flag);</code></td>
</tr>
<tr>
<td>Function descriptions</td>
<td>get LPUART interrupt and flag status</td>
</tr>
<tr>
<td>Precondition</td>
<td>-</td>
</tr>
<tr>
<td>The called functions</td>
<td>-</td>
</tr>
</tbody>
</table>

### Input parameter(in)

<table>
<thead>
<tr>
<th>int_flag</th>
<th>LPUART interrupt flag, refer to <code>Table 3-423. Enum</code> <code>lpuart_interrupt_flag_enum</code>, only one among these parameters can be selected</th>
</tr>
</thead>
<tbody>
<tr>
<td>LPUART_INT_FLAG_AM</td>
<td>address match interrupt and flag</td>
</tr>
<tr>
<td>LPUART_INT_FLAG_PERR</td>
<td>parity error interrupt and flag</td>
</tr>
<tr>
<td>LPUART_INT_FLAG_TBE</td>
<td>transmitter buffer empty interrupt and flag</td>
</tr>
<tr>
<td>LPUART_INT_FLAG_T</td>
<td>transmission complete interrupt and flag</td>
</tr>
</tbody>
</table>
CLPUART_INT_FLAG_RBNE
read data buffer not empty interrupt and flag

CLPUART_INT_FLAG_RBNE_ORERR
read data buffer not empty interrupt and overrun error flag

CLPUART_INT_FLAG_IDLE
IDLE line detected interrupt and flag

CLPUART_INT_FLAG_WU
wakeup from deep-sleep mode interrupt and flag

CLPUART_INT_FLAG_CTS
CTS interrupt and flag

CLPUART_INT_FLAG_ERR
error interrupt and noise error flag

CLPUART_INT_FLAG_EE_OERRR
error interrupt and overrun error

CLPUART_INT_FLAG_EE_RERR
error interrupt and frame error flag

Output parameter(out)
-

Return value
FlagStatus
SET or RESET

Example:

/* get the LPUART interrupt flag status */

FlagStatus status;
status = lpuart_interrupt_flag_get(LPUART_INT_FLAG_RBNE);

Ipuart_interrupt_flag_clear

The description of lpuart_interrupt_flag_clear is shown as below:

Table 3-469. Function lpuart_interrupt_flag_clear

<table>
<thead>
<tr>
<th>Function name</th>
<th>lpuart_interrupt_flag_clear</th>
</tr>
</thead>
<tbody>
<tr>
<td>Function prototype</td>
<td>void lpuart_interrupt_flag_clear(lpuart_interrupt_flag_enum int_flag);</td>
</tr>
<tr>
<td>Function descriptions</td>
<td>clear LPUART interrupt flag</td>
</tr>
<tr>
<td>Precondition</td>
<td>-</td>
</tr>
<tr>
<td>The called functions</td>
<td>-</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Input parameter(in)</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>int_flag</strong></td>
</tr>
<tr>
<td>LPUART interrupt flag, refer to Table 3-423. Enum lpuart_interrupt_flag_enum, only one among these parameters can be selected</td>
</tr>
<tr>
<td>LPUART_INT_FLAG_PERR</td>
</tr>
<tr>
<td>parity error flag</td>
</tr>
</tbody>
</table>
### LPUART_INT_FLAG_E

- **RR_FERR**: frame error flag
- **RR_NERR**: noise detected flag
- **BNE_ORERR**: read data buffer not empty interrupt and overrun error flag
- **RR_ORERR**: error interrupt and overrun error
- **IDL**: idle line detected flag
- **TC**: transmission complete flag
- **TS**: CTS change flag
- **AM**: address match flag
- **WU**: wakeup from deep-sleep mode flag

### Output parameter (out)

- **-**: -

### Return value

- **-**: -

---

**Example:**

```c
/* clear the LPUART interrupt flag */
lpuart_interrupt_flag_clear(LPUART_INT_FLAG_TC);
```

### 3.17. MISC

MISC is a software package that provide the interfaces for NVIC and SysTick. The NVIC and SysTick registers are listed in chapter 3.17.1, the MISC firmware functions are introduced in chapter 3.17.2.

### 3.17.1. Descriptions of Peripheral registers

**Table 3-470. NVIC Registers**

<table>
<thead>
<tr>
<th>Registers</th>
<th>Descriptions</th>
</tr>
</thead>
<tbody>
<tr>
<td>ISER(1)</td>
<td>Interrupt Set Enable Register</td>
</tr>
<tr>
<td>ICER(1)</td>
<td>Interrupt Clear Enable Register</td>
</tr>
<tr>
<td>ISPR(1)</td>
<td>Interrupt Set Pending Register</td>
</tr>
<tr>
<td>ICPR(1)</td>
<td>Interrupt Clear Pending Register</td>
</tr>
<tr>
<td>IABR(1)</td>
<td>Interrupt Active bit Register</td>
</tr>
</tbody>
</table>
1. refer to the structure NVIC_Type, is defined in the core_cm32.h file
2. refer to the structure SCB_Type, is defined in the core_cm32.h file

Table 3-471. SysTick Registers

<table>
<thead>
<tr>
<th>Registers</th>
<th>Descriptions</th>
</tr>
</thead>
<tbody>
<tr>
<td>CTRL(1)</td>
<td>SysTick Control and Status Register</td>
</tr>
<tr>
<td>LOAD(1)</td>
<td>SysTick Reload Value Register</td>
</tr>
<tr>
<td>VAL(1)</td>
<td>SysTick Current Value Register</td>
</tr>
<tr>
<td>CALIB(1)</td>
<td>SysTick Calibration Register</td>
</tr>
</tbody>
</table>

1. refer to the structure SysTick_Type, is defined in the core_cm32.h file

3.17.2. Descriptions of Peripheral functions

MISC firmware functions are listed in the table shown as below:

Table 3-472. MISC firmware function

<table>
<thead>
<tr>
<th>Function name</th>
<th>Function description</th>
</tr>
</thead>
<tbody>
<tr>
<td>nvic_irq_enable</td>
<td>enable NVIC request</td>
</tr>
<tr>
<td>nvic_irq_disable</td>
<td>disable NVIC request</td>
</tr>
<tr>
<td>nvic_system_reset</td>
<td>initiates a system reset request to reset the MCU</td>
</tr>
<tr>
<td>nvic_vector_table_set</td>
<td>set the NVIC vector table address</td>
</tr>
<tr>
<td>system_lowpower_set</td>
<td>set the state of the low power mode</td>
</tr>
<tr>
<td>system_lowpower_reset</td>
<td>reset the state of the low power mode</td>
</tr>
<tr>
<td>systick_clicksource_set</td>
<td>set the systick clock source</td>
</tr>
</tbody>
</table>

Enum IRQn_Type

Table 3-473. IRQn_Type

<table>
<thead>
<tr>
<th>Member name</th>
<th>Function description</th>
</tr>
</thead>
<tbody>
<tr>
<td>WWDGT_IRQn</td>
<td>window watchDog timer interrupt</td>
</tr>
<tr>
<td>LVD_IRQn</td>
<td>LVD through EXTI line detect interrupt</td>
</tr>
<tr>
<td>TAMPER_STAMP_IRQn</td>
<td>RTC Tamper and Timestamp from EXTI interrupt</td>
</tr>
<tr>
<td>RTC_WKUP_IRQn</td>
<td>RTC Wakeup interrupt</td>
</tr>
<tr>
<td>Member name</td>
<td>Function description</td>
</tr>
<tr>
<td>--------------------------------</td>
<td>------------------------</td>
</tr>
<tr>
<td>FMC_IRQHandler</td>
<td>FMC interrupt</td>
</tr>
<tr>
<td>RCU_CTC_IRQHandler</td>
<td>RCU and CTC interrupt</td>
</tr>
<tr>
<td>EXTI0_IRQn</td>
<td>EXTI line 0 interrupts</td>
</tr>
<tr>
<td>EXTI1_IRQn</td>
<td>EXTI line 1 interrupts</td>
</tr>
<tr>
<td>EXTI2_IRQn</td>
<td>EXTI line 2 interrupts</td>
</tr>
<tr>
<td>EXTI3_IRQn</td>
<td>EXTI line 3 interrupts</td>
</tr>
<tr>
<td>EXTI4_IRQn</td>
<td>EXTI line 4 interrupts</td>
</tr>
<tr>
<td>DMA_Channel0_IRQHandler</td>
<td>DMA channel 0 interrupt</td>
</tr>
<tr>
<td>DMA_Channel1_IRQHandler</td>
<td>DMA channel 1 interrupts</td>
</tr>
<tr>
<td>DMA_Channel2_IRQHandler</td>
<td>DMA channel 2 interrupts</td>
</tr>
<tr>
<td>DMA_Channel3_IRQHandler</td>
<td>DMA channel 3 interrupts</td>
</tr>
<tr>
<td>DMA_Channel4_IRQHandler</td>
<td>DMA channel 4 interrupts</td>
</tr>
<tr>
<td>DMA_Channel5_IRQHandler</td>
<td>DMA channel 5 interrupts</td>
</tr>
<tr>
<td>DMA_Channel6_IRQHandler</td>
<td>DMA channel 6 interrupts</td>
</tr>
<tr>
<td>ADC_IRQn</td>
<td>ADC interrupts</td>
</tr>
<tr>
<td>USBD_HP_IRQn</td>
<td>USBD High Priority</td>
</tr>
<tr>
<td>USBD_LP_IRQn</td>
<td>USBD Low Priority</td>
</tr>
<tr>
<td>TIMER1_IRQHandler</td>
<td>TIMER1 interrupt</td>
</tr>
<tr>
<td>TIMER2_IRQHandler</td>
<td>TIMER2 interrupt</td>
</tr>
<tr>
<td>TIMER8_IRQHandler</td>
<td>TIMER8 interrupt</td>
</tr>
<tr>
<td>TIMER11_IRQHandler</td>
<td>TIMER11 interrupt</td>
</tr>
<tr>
<td>TIMER5_IRQn</td>
<td>TIMER5 interrupt</td>
</tr>
<tr>
<td>TIMER6_IRQn</td>
<td>TIMER6 interrupt</td>
</tr>
<tr>
<td>USART0_IRQHandler</td>
<td>USART0 interrupt</td>
</tr>
<tr>
<td>USART1_IRQHandler</td>
<td>USART1 interrupt</td>
</tr>
<tr>
<td>UART3_IRQn</td>
<td>UART3 interrupt</td>
</tr>
<tr>
<td>UART4_IRQn</td>
<td>UART4 interrupt</td>
</tr>
<tr>
<td>I2C0_EV_IRQn</td>
<td>I2C0 event interrupt</td>
</tr>
<tr>
<td>I2C0_ER_IRQn</td>
<td>I2C0 error interrupt</td>
</tr>
<tr>
<td>I2C1_EV_IRQn</td>
<td>I2C1 event interrupt</td>
</tr>
<tr>
<td>I2C1_ER_IRQn</td>
<td>I2C1 error interrupt</td>
</tr>
<tr>
<td>SPI0_IRQn</td>
<td>SPI0 interrupt</td>
</tr>
<tr>
<td>SPI1_IRQn</td>
<td>SPI1 interrupt</td>
</tr>
<tr>
<td>DAC_IRQn</td>
<td>DAC interrupt</td>
</tr>
<tr>
<td>I2C2_EV_IRQn</td>
<td>I2C2 event interrupt</td>
</tr>
<tr>
<td>I2C2_ER_IRQn</td>
<td>I2C2 error interrupt</td>
</tr>
<tr>
<td>RTC_Alarm_IRQHandler</td>
<td>RTC Alarm interrupt</td>
</tr>
<tr>
<td>USBD_WAKEUP_IRQn</td>
<td>USBD Wakeup interrupt</td>
</tr>
<tr>
<td>EXTI9_10_IRQn</td>
<td>EXTI line 5 to 9 interrupts</td>
</tr>
<tr>
<td>EXTI11_15_IRQn</td>
<td>EXTI line 10 to 15 interrupts</td>
</tr>
<tr>
<td>DMAMUX_IRQn</td>
<td>DMAMUX interrupt</td>
</tr>
<tr>
<td>Member name</td>
<td>Function description</td>
</tr>
<tr>
<td>--------------------------</td>
<td>------------------------------------</td>
</tr>
<tr>
<td>CMP0_IRQHandler</td>
<td>Comparator 0 interrupt</td>
</tr>
<tr>
<td>CMP1_IRQHandler</td>
<td>Comparator 1 interrupt</td>
</tr>
<tr>
<td>I2C0_WKUP_IRQHandler</td>
<td>I2C0 Wakeup interrupt</td>
</tr>
<tr>
<td>I2C2_WKUP_IRQHandler</td>
<td>I2C2 Wakeup interrupt</td>
</tr>
<tr>
<td>USART0_WKUP_IRQHandler</td>
<td>USART0 Wakeup interrupt</td>
</tr>
<tr>
<td>LPUART_IRQn</td>
<td>LPUART global interrupt</td>
</tr>
<tr>
<td>CAU_IRQn</td>
<td>CAU interrupt</td>
</tr>
<tr>
<td>TRNG_IRQn</td>
<td>TRNG interrupt</td>
</tr>
<tr>
<td>SLCD_IRQn</td>
<td>SLCD interrupt</td>
</tr>
<tr>
<td>USART1_WKUP_IRQn</td>
<td>USART1 Wakeup interrupt</td>
</tr>
<tr>
<td>I2C1_WKUP_IRQn</td>
<td>I2C1 Wakeup interrupt</td>
</tr>
<tr>
<td>LPUART_WKUP_IRQn</td>
<td>LPUART Wakeup interrupt</td>
</tr>
<tr>
<td>LPTIMER_IRQn</td>
<td>LPTIMER interrupt</td>
</tr>
</tbody>
</table>

### nvic_irq_enable

The description of `nvic_irq_enable` is shown as below:

**Table 3-474. Function nvic_irq_enable**

<table>
<thead>
<tr>
<th>Function name</th>
<th>nvic_irq_enable</th>
</tr>
</thead>
<tbody>
<tr>
<td>Function prototype</td>
<td>void nvic_irq_enable(uint8_t nvic_irq, uint8_t nvic_irq_pre_priority);</td>
</tr>
<tr>
<td>Function descriptions</td>
<td>enable NVIC request</td>
</tr>
<tr>
<td>Precondition</td>
<td>-</td>
</tr>
<tr>
<td>The called functions</td>
<td>NVIC_SetPriority, NVIC_EnableIRQ</td>
</tr>
</tbody>
</table>

#### Input parameter (in)

- `nvic_irq` : NVIC interrupt, refer to enum `Table 3-473,IRQn_Type`

#### Input parameter (in)

- `nvic_irq_pre_priority` : the pre-emption priority needed to set (0~3)

#### Output parameter (out)

- `Return value` : -

Example:

```c
/* enable window watchDog timer interrupt, priority is 1 */
nvic_irq_enable(WWDGT_IRQn, 1);
```

### nvic_irq_disable

The description of `nvic_irq_disable` is shown as below:

**Table 3-475. Function nvic_irq_disable**

<table>
<thead>
<tr>
<th>Function name</th>
<th>nvic_irq_disable</th>
</tr>
</thead>
</table>

Function prototype: void nvic_irq_disable(uint8_t nvic_irq);

Function descriptions: disable NVIC request

Precondition: -

The called functions: -

<table>
<thead>
<tr>
<th>Input parameter (in)</th>
<th>Output parameter (out)</th>
<th>Return value</th>
</tr>
</thead>
<tbody>
<tr>
<td>nvic_irq</td>
<td>NVIC interrupt, refer to enum</td>
<td>-</td>
</tr>
</tbody>
</table>

Example:

/* disable window watchDog timer interrupt */
nvic_irq_disable(WWDGT_IRQn);

nvic_system_reset

The description of nvic_system_reset is shown as below:

Table 3-476. Function nvic_system_reset

<table>
<thead>
<tr>
<th>Function name</th>
<th>nvic_system_reset</th>
</tr>
</thead>
<tbody>
<tr>
<td>Function prototype</td>
<td>void nvic_system_reset(void);</td>
</tr>
<tr>
<td>Function descriptions</td>
<td>initiates a system reset request to reset the MCU</td>
</tr>
<tr>
<td>Precondition</td>
<td>-</td>
</tr>
<tr>
<td>The called functions</td>
<td>NVIC_SystemReset</td>
</tr>
<tr>
<td>Input parameter (in)</td>
<td>-</td>
</tr>
<tr>
<td>Output parameter (out)</td>
<td>-</td>
</tr>
<tr>
<td>Return value</td>
<td>-</td>
</tr>
</tbody>
</table>

Example:

/* reset the MCU */
nvic_system_reset();

nvic_vector_table_set

The description of nvic_vector_table_set is shown as below:

Table 3-477. Function nvic_vector_table_set

<table>
<thead>
<tr>
<th>Function name</th>
<th>nvic_vector_table_set</th>
</tr>
</thead>
<tbody>
<tr>
<td>Function prototype</td>
<td>void nvic_vector_table_set(uint32_t nvic_vict_tab, uint32_t offset);</td>
</tr>
<tr>
<td>Function descriptions</td>
<td>set the NVIC vector table address</td>
</tr>
</tbody>
</table>
Precondition

The called functions

**Input parameter(in)**

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>nvic_vict_tab</td>
<td>the RAM or FLASH base address</td>
</tr>
<tr>
<td>NVIC_VECTTAB_RAM</td>
<td>RAM base address</td>
</tr>
<tr>
<td>NVIC_VECTTAB_FLASH</td>
<td>Flash base address</td>
</tr>
</tbody>
</table>

**Input parameter(in)**

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>offset</td>
<td>Vector Table offset (vector table start address= base address+offset)</td>
</tr>
</tbody>
</table>

**Output parameter(out)**

- -

**Return value**

- -

Example:

```c
/* set vector table address = NVIC_VECTTAB_FLASH + 0x200 */
nvic_vector_table_set(NVIC_VECTTAB_FLASH, 0x200);
```

**system_lowpower_set**

The description of system_lowpower_set is shown as below:

**Table 3-478. Function system_lowpower_set**

<table>
<thead>
<tr>
<th>Function name</th>
<th>system_lowpower_set</th>
</tr>
</thead>
<tbody>
<tr>
<td>Function prototype</td>
<td>void system_lowpower_set(uint8_t lowpower_mode);</td>
</tr>
<tr>
<td>Function descriptions</td>
<td>set the state of the low power mode</td>
</tr>
<tr>
<td>Precondition</td>
<td>-</td>
</tr>
<tr>
<td>The called functions</td>
<td>-</td>
</tr>
</tbody>
</table>

**Input parameter(in)**

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>lowpower_mode</td>
<td>the low power mode state</td>
</tr>
<tr>
<td>SCB_LPM_SLEEP_EXIT_ISR</td>
<td>if chose this para, the system always enter low power mode by exiting from ISR</td>
</tr>
<tr>
<td>SCB_LPM_DEEPSLEEP</td>
<td>if chose this para, the system will enter the DEEPSLEEP mode</td>
</tr>
<tr>
<td>SCB_LPM_WAKE_BY_ALL_INT</td>
<td>if chose this para, the lowpower mode can be woke up by all the enable and disable interrupts</td>
</tr>
</tbody>
</table>

**Output parameter(out)**

- -

**Return value**

- -

Example:

```c
/* the system always enter low power mode by exiting from ISR */
system_lowpower_set(SCB_LPM_SLEEP_EXIT_ISR);
```
system_lowpower_reset

The description of system_lowpower_reset is shown as below:

Table 3-479. Function system_lowpower_reset

<table>
<thead>
<tr>
<th>Function name</th>
<th>system_lowpower_reset</th>
</tr>
</thead>
<tbody>
<tr>
<td>Function prototype</td>
<td>void system_lowpower_reset(uint8_t lowpower_mode);</td>
</tr>
<tr>
<td>Function descriptions</td>
<td>reset the state of the low power mode</td>
</tr>
<tr>
<td>Precondition</td>
<td>-</td>
</tr>
<tr>
<td>The called functions</td>
<td>-</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Input parameter (in)</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>lowpower_mode</td>
<td>the low power mode state</td>
</tr>
<tr>
<td>SCB_LPM_SLEEP_EXIT_ISR</td>
<td>if chose this para, the system will exit low power mode by exiting from ISR</td>
</tr>
<tr>
<td>SCB_LPM_DEEPSLEEP_P</td>
<td>if chose this para, the system will enter the SLEEP mode</td>
</tr>
<tr>
<td>SCB_LPM_WAKE_BY_ALL_INT</td>
<td>if chose this para, the lowpower mode only can be woke up by the enable interrupts</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Output parameter (out)</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Return value</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Example:

/* the system will exit low power mode by exiting from ISR */
system_lowpower_reset(SCB_LPM_SLEEP_EXIT_ISR);

systick_clksource_set

The description of systick_clksource_set is shown as below:

Table 3-480. Function systick_clksource_set

<table>
<thead>
<tr>
<th>Function name</th>
<th>systick_clksource_set</th>
</tr>
</thead>
<tbody>
<tr>
<td>Function prototype</td>
<td>void systick_clksource_set(uint32_t systick_clksource);</td>
</tr>
<tr>
<td>Function descriptions</td>
<td>set the systick clock source</td>
</tr>
<tr>
<td>Precondition</td>
<td>-</td>
</tr>
<tr>
<td>The called functions</td>
<td>-</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Input parameter (in)</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>systick_clksource</td>
<td>the systick clock source needed to choose</td>
</tr>
<tr>
<td>SYSTICK_CLKSOURCE_E_HCLK</td>
<td>systick clock source is from HCLK</td>
</tr>
<tr>
<td>SYSTICK_CLKSOURCE_E_HCLK_DIV8</td>
<td>systick clock source is from HCLK/8</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Output parameter (out)</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
</tr>
</tbody>
</table>


Example:

/* systick clock source is HCLK/8 */
systick_clksource_set(SYSYSTICK_CLKSOURCE_HCLK_DIV8);

3.18. PMU

According to the Power management unit (PMU), provides ten types of power saving modes, including Run, Run1, Run2, Sleep, Sleep1, Sleep2, Sleep-sleep, Sleep-2, Deep-sleep 1, Deep-sleep 2 and Standby mode. The PMU registers are listed in chapter 3.18.1, the PMU firmware functions are introduced in chapter 3.18.2.

3.18.1. Descriptions of Peripheral registers

PMU registers are listed in the table shown as below:

Table 3-481. PMU Registers

<table>
<thead>
<tr>
<th>Registers</th>
<th>Descriptions</th>
</tr>
</thead>
<tbody>
<tr>
<td>PMU_CTL0</td>
<td>PMU control 0 register</td>
</tr>
<tr>
<td>PMU_CS</td>
<td>PMU control and status register</td>
</tr>
<tr>
<td>PMU_CTL1</td>
<td>PMU control 1 register</td>
</tr>
<tr>
<td>PMU_STAT</td>
<td>PMU status register</td>
</tr>
<tr>
<td>PMU_PAR</td>
<td>PMU parameter register</td>
</tr>
</tbody>
</table>

3.18.2. Descriptions of Peripheral functions

PMU firmware functions are listed in the table shown as below:

Table 3-482. PMU firmware function

<table>
<thead>
<tr>
<th>Function name</th>
<th>Function description</th>
</tr>
</thead>
<tbody>
<tr>
<td>pmu_deinit</td>
<td>reset PMU</td>
</tr>
<tr>
<td>pmu_lvd_select</td>
<td>select low voltage detector threshold</td>
</tr>
<tr>
<td>pmu_lvd_disable</td>
<td>disable PMU lvd</td>
</tr>
<tr>
<td>pmu_ldo_output_select</td>
<td>select LDO output voltage</td>
</tr>
<tr>
<td>pmu_vc_enable</td>
<td>enable VBAT battery charging</td>
</tr>
<tr>
<td>pmu_vc_disable</td>
<td>disable VBAT battery charging</td>
</tr>
<tr>
<td>pmu_vcr_select</td>
<td>select PMU VBAT battery charging resistor</td>
</tr>
<tr>
<td>pmu_low_power_enable</td>
<td>enable low power in Run/Sleep mode</td>
</tr>
<tr>
<td>pmu_low_power_disable</td>
<td>disable low power in Run/Sleep mode</td>
</tr>
<tr>
<td>pmu_to_sleepmode</td>
<td>PMU work in sleep mode</td>
</tr>
<tr>
<td>Function name</td>
<td>Function description</td>
</tr>
<tr>
<td>-----------------------------------</td>
<td>-----------------------------------------------</td>
</tr>
<tr>
<td>pmu_to_deepsleepmode</td>
<td>PMU work at Deep-sleep/Deep-sleep 1/Deep-sleep 2 mode</td>
</tr>
<tr>
<td>pmu_to_standbymode</td>
<td>pmu work at standby mode</td>
</tr>
<tr>
<td>pmu_wakeup_pin_enable</td>
<td>enable PMU wakeup pin</td>
</tr>
<tr>
<td>pmu_wakeup_pin_disable</td>
<td>disable PMU wakeup pin</td>
</tr>
<tr>
<td>pmu_backup_write_enable</td>
<td>enable backup domain write</td>
</tr>
<tr>
<td>pmu_backup_write_disable</td>
<td>disable backup domain write</td>
</tr>
<tr>
<td>pmu_sram_power_config</td>
<td>configure power state of SRAM1</td>
</tr>
<tr>
<td>pmu_core1_power_config</td>
<td>configure power state of COREOFF1 domain</td>
</tr>
<tr>
<td>pmu_deepsleep2_retention_enable</td>
<td>have retention register in Deep-sleep 2</td>
</tr>
<tr>
<td>pmu_deepsleep2_retention_disable</td>
<td>no retention register in Deep-sleep 2</td>
</tr>
<tr>
<td>pmu_deepsleep2_sram_power_config</td>
<td>configure SRAM1 power state when enter Deep-sleep 2</td>
</tr>
<tr>
<td>pmu_deepsleep2_wait_time_config</td>
<td>configure IRC16M counter before enter Deep-sleep mode</td>
</tr>
<tr>
<td>pmu_wakeuptime_core1_software_enable</td>
<td>use software value signal when wake up COREOFF1 domain</td>
</tr>
<tr>
<td>pmu_wakeuptime_core1ソフトウェア</td>
<td>use hardware ack signal when wake up COREOFF1 domain</td>
</tr>
<tr>
<td>pmu_wakeuptime_sramソフトウェア</td>
<td>use software value signal when wake up SRAM1</td>
</tr>
<tr>
<td>pmu_wakeuptime_sramソフトウェア</td>
<td>use hardware ack signal when wake up SRAM1</td>
</tr>
<tr>
<td>pmu_wakeuptime_deepsleep2ソフトウェア</td>
<td>use software value signal when wake up Deep-sleep 2</td>
</tr>
<tr>
<td>pmu_wakeuptime_deepsleep2ソフトウェア</td>
<td>use hardware ack signal when wake up Deep-sleep 2</td>
</tr>
<tr>
<td>pmu_flag_get</td>
<td>get PMU flag status</td>
</tr>
<tr>
<td>pmu_flag_clear</td>
<td>clear PMU flag status</td>
</tr>
</tbody>
</table>

**pmu_deinit**

The description of pmu_deinit is shown as below:

**Table 3-483. Function pmu_deinit**

<table>
<thead>
<tr>
<th>Function name</th>
<th>pmu_deinit</th>
</tr>
</thead>
<tbody>
<tr>
<td>Function prototype</td>
<td>void pmu_deinit(void);</td>
</tr>
<tr>
<td>Function descriptions</td>
<td>reset PMU</td>
</tr>
<tr>
<td>Precondition</td>
<td>-</td>
</tr>
<tr>
<td>The called functions</td>
<td>rcu_periph_reset_enable / rcu_periph_reset_disable</td>
</tr>
</tbody>
</table>

**Input parameter(in)**

- |

**Output parameter(out)**

- |

**Return value**

322
Example:

/* reset PMU */

pmu_deinit();

**pmu_lvd_select**

The description of `pmu_lvd_select` is shown as below:

**Table 3-484. Function pmu_lvd_select**

<table>
<thead>
<tr>
<th>Function name</th>
<th>pmu_lvd_select</th>
</tr>
</thead>
<tbody>
<tr>
<td>Function prototype</td>
<td>void pmu_lvd_select(uint32_t lvdt_n);</td>
</tr>
<tr>
<td>Function descriptions</td>
<td>select low voltage detector threshold</td>
</tr>
<tr>
<td>Precondition</td>
<td>-</td>
</tr>
<tr>
<td>The called functions</td>
<td>-</td>
</tr>
<tr>
<td><strong>Input parameter(in)</strong></td>
<td></td>
</tr>
<tr>
<td>lvdt_n</td>
<td>voltage threshold value</td>
</tr>
<tr>
<td>PMU_LVDT_0</td>
<td>voltage threshold is 2.1V</td>
</tr>
<tr>
<td>PMU_LVDT_1</td>
<td>voltage threshold is 2.3V</td>
</tr>
<tr>
<td>PMU_LVDT_2</td>
<td>voltage threshold is 2.4V</td>
</tr>
<tr>
<td>PMU_LVDT_3</td>
<td>voltage threshold is 2.6V</td>
</tr>
<tr>
<td>PMU_LVDT_4</td>
<td>voltage threshold is 2.7V</td>
</tr>
<tr>
<td>PMU_LVDT_5</td>
<td>voltage threshold is 2.9V</td>
</tr>
<tr>
<td>PMU_LVDT_6</td>
<td>voltage threshold is 3.0V</td>
</tr>
<tr>
<td>PMU_LVDT_7</td>
<td>input analog voltage on PB7 (compared with 0.8V)</td>
</tr>
<tr>
<td><strong>Output parameter(out)</strong></td>
<td></td>
</tr>
<tr>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td><strong>Return value</strong></td>
<td></td>
</tr>
<tr>
<td>-</td>
<td>-</td>
</tr>
</tbody>
</table>

Example:

/* select low voltage detector threshold as 3.0V */

pmu_lvd_select(PMU_LVDT_6);

**pmu_lvd_disable**

The description of `pmu_lvd_disable` is shown as below:

**Table 3-485. Function pmu_lvd_disable**

<table>
<thead>
<tr>
<th>Function name</th>
<th>pmu_lvd_disable</th>
</tr>
</thead>
<tbody>
<tr>
<td>Function prototype</td>
<td>void pmu_lvd_disable(void);</td>
</tr>
<tr>
<td>Function descriptions</td>
<td>disable PMU lvd</td>
</tr>
</tbody>
</table>
The called functions

Input parameter (in)

Output parameter (out)

Return value

Example:

/* disable PMU lvd */

pmu_lvd_disable();

pmu_ldo_output_select

The description of pmu_ldo_output_select is shown as below:

<table>
<thead>
<tr>
<th>Function name</th>
<th>pmu_ldo_output_select</th>
</tr>
</thead>
<tbody>
<tr>
<td>Function prototype</td>
<td>void pmu_ldo_output_select(uint32_t ldo_output);</td>
</tr>
<tr>
<td>Function descriptions</td>
<td>select LDO output voltage</td>
</tr>
<tr>
<td>Precondition</td>
<td>-</td>
</tr>
<tr>
<td>The called functions</td>
<td>-</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Ido_output</th>
<th>output voltage mode</th>
</tr>
</thead>
<tbody>
<tr>
<td>PMU_LDOVS_LOW</td>
<td>LDO output voltage low mode</td>
</tr>
<tr>
<td>PMU_LDOVS_HIGH</td>
<td>LDO output voltage high mode</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Output parameter (out)</th>
<th>-</th>
</tr>
</thead>
<tbody>
<tr>
<td>Return value</td>
<td>-</td>
</tr>
</tbody>
</table>

Example:

/* select output low voltage mode */

pmu_ldo_output_select(PMU_LDOVS_LOW);

pmu_vc_enable

The description of pmu_vc_enable is shown as below:

<table>
<thead>
<tr>
<th>Function name</th>
<th>pmu_vc_enable</th>
</tr>
</thead>
<tbody>
<tr>
<td>Function prototype</td>
<td>void pmu_vc_enable(void);</td>
</tr>
<tr>
<td>Function descriptions</td>
<td>enable VBAT battery charging</td>
</tr>
<tr>
<td>-------------------------------</td>
<td>------------------------------</td>
</tr>
<tr>
<td>Precondition</td>
<td>-</td>
</tr>
<tr>
<td>The called functions</td>
<td>-</td>
</tr>
<tr>
<td>Input parameter(in)</td>
<td>-</td>
</tr>
<tr>
<td>Output parameter(out)</td>
<td>-</td>
</tr>
<tr>
<td>Return value</td>
<td>-</td>
</tr>
</tbody>
</table>

Example:

```c
/* enable VBAT battery charging */

pmu_vc_enable();
```

**pmu_vc_disable**

The description of `pmu_vc_disable` is shown as below:

**Table 3-488. Function pmu_vc_disable**

<table>
<thead>
<tr>
<th>Function name</th>
<th>pmu_vc_disable</th>
</tr>
</thead>
<tbody>
<tr>
<td>Function prototype</td>
<td>void pmu_vc_disable(void);</td>
</tr>
<tr>
<td>Function descriptions</td>
<td>disable VBAT battery charging</td>
</tr>
<tr>
<td>Precondition</td>
<td>-</td>
</tr>
<tr>
<td>The called functions</td>
<td>-</td>
</tr>
<tr>
<td>Input parameter(in)</td>
<td>-</td>
</tr>
<tr>
<td>Output parameter(out)</td>
<td>-</td>
</tr>
<tr>
<td>Return value</td>
<td>-</td>
</tr>
</tbody>
</table>

Example:

```c
/* disable VBAT battery charging */

pmu_vc_disable();
```

**pmu_vcr_select**

The description of `pmu_vcr_select` is shown as below:

**Table 3-489. Function pmu_vcr_select**

<table>
<thead>
<tr>
<th>Function name</th>
<th>pmu_vcr_select</th>
</tr>
</thead>
<tbody>
<tr>
<td>Function prototype</td>
<td>void pmu_vcr_select(uint32_t resistor);</td>
</tr>
<tr>
<td>Function descriptions</td>
<td>select PMU VBAT battery charging resistor</td>
</tr>
</tbody>
</table>
The called functions

- Precondition
- The called functions

<table>
<thead>
<tr>
<th>Input parameter(in)</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>PMU_VCRSEL_5K</td>
<td>5 kOhms resistor is selected for charging VBAT battery</td>
</tr>
<tr>
<td>PMU_VCRSEL_1P5K</td>
<td>1.5 kOhms resistor is selected for charging VBAT battery</td>
</tr>
</tbody>
</table>

- Output parameter(out)

- Return value

Example:

/* select PMU VBAT battery charging resistor */

pmu_vcr_select(PMU_VCRSEL_5K);

**pmu_low_power_enable**

The description of pmu_low_power_enable is shown as below:

**Table 3-490. Function pmu_low_power_enable**

<table>
<thead>
<tr>
<th>Function name</th>
<th>pmu_low_power_enable</th>
</tr>
</thead>
<tbody>
<tr>
<td>Function prototype</td>
<td>void pmu_low_power_enable(void);</td>
</tr>
<tr>
<td>Function descriptions</td>
<td>enable low power in Run/Sleep mode</td>
</tr>
<tr>
<td>Precondition</td>
<td>-</td>
</tr>
<tr>
<td>The called functions</td>
<td>-</td>
</tr>
<tr>
<td>Input parameter(in)</td>
<td>-</td>
</tr>
<tr>
<td>Output parameter(out)</td>
<td>-</td>
</tr>
<tr>
<td>Return value</td>
<td>-</td>
</tr>
</tbody>
</table>

Example:

/* enable low power in Run/Sleep mode */

pmu_low_power_enable();

**pmu_low_power_disable**

The description of pmu_low_power_disable is shown as below:

**Table 3-491. Function pmu_low_power_disable**

<table>
<thead>
<tr>
<th>Function name</th>
<th>pmu_low_power_disable</th>
</tr>
</thead>
<tbody>
<tr>
<td>Function prototype</td>
<td>void pmu_low_power_disable(void);</td>
</tr>
</tbody>
</table>
### pmu_to_sleepmode

The description of pmu_to_sleepmode is shown as below:

#### Table 3-492. Function pmu_to_sleepmode

<table>
<thead>
<tr>
<th>Function name</th>
<th>pmu_to_sleepmode</th>
</tr>
</thead>
<tbody>
<tr>
<td>Function prototype</td>
<td>void pmu_to_sleepmode(uint32_t lowdrive, uint8_t sleepmodecmd);</td>
</tr>
<tr>
<td>Function descriptions</td>
<td>PMU work at sleep mode</td>
</tr>
<tr>
<td>Precondition</td>
<td>-</td>
</tr>
<tr>
<td>The called functions</td>
<td>-</td>
</tr>
<tr>
<td>Input parameter(in)</td>
<td></td>
</tr>
<tr>
<td>lowdrive</td>
<td>NPLDO work mode</td>
</tr>
<tr>
<td>PMU_LDNPDSP_NOR</td>
<td>low-driver mode disable</td>
</tr>
<tr>
<td>MALDRIVE</td>
<td></td>
</tr>
<tr>
<td>PMU_LDNPDSP_LOW</td>
<td>low-driver mode enable</td>
</tr>
<tr>
<td>DRIVE</td>
<td></td>
</tr>
<tr>
<td>Output parameter(out)</td>
<td></td>
</tr>
<tr>
<td>sleepmodecmd</td>
<td>command to enter sleep mode</td>
</tr>
<tr>
<td>WFI_CMD</td>
<td>use WFI command</td>
</tr>
<tr>
<td>WFE_CMD</td>
<td>use WFE command</td>
</tr>
<tr>
<td>Return value</td>
<td>-</td>
</tr>
</tbody>
</table>

Example:

/* PMU work at sleep mode */

pmu_to_sleepmode(PMU_LDNP_LOWDRIVE, WFI_CMD);
pmu_to_deepsleepmode

The description of pmu_to_deepsleepmode is shown as below:

Table 3-493. Function pmu_to_deepsleepmode

<table>
<thead>
<tr>
<th>Function name</th>
<th>pmu_to_deepsleepmode</th>
</tr>
</thead>
<tbody>
<tr>
<td>Function prototype</td>
<td>void pmu_to_deepsleepmode(uint32_t lowdrive, uint8_t deepsleepmodecmd, uint8_t deepsleepmode);</td>
</tr>
<tr>
<td>Function descriptions</td>
<td>PMU work at Deep-sleep/Deep-sleep 1/Deep-sleep 2 mode</td>
</tr>
<tr>
<td>Precondition</td>
<td>-</td>
</tr>
<tr>
<td>The called functions</td>
<td>-</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Input parameter(in)</th>
</tr>
</thead>
<tbody>
<tr>
<td>lowdrive</td>
</tr>
<tr>
<td>PMU_LDNPDSP_NORMADRIVE</td>
</tr>
<tr>
<td>PMU_LDNPDSP_LOWDRIVE</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Input parameter(in)</th>
</tr>
</thead>
<tbody>
<tr>
<td>deepsleepmodecmd</td>
</tr>
<tr>
<td>WFI_CMD</td>
</tr>
<tr>
<td>WFE_CMD</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Input parameter(in)</th>
</tr>
</thead>
<tbody>
<tr>
<td>deepsleepmode</td>
</tr>
<tr>
<td>PMU_DEEPSLEEP</td>
</tr>
<tr>
<td>PMU_DEEPSLEEP1</td>
</tr>
<tr>
<td>PMU_DEEPSLEEP2</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Output parameter(out)</th>
</tr>
</thead>
<tbody>
<tr>
<td>-</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Return value</th>
</tr>
</thead>
<tbody>
<tr>
<td>-</td>
</tr>
</tbody>
</table>

Example:

/* PMU work at Deep-sleep mode */

pmu_to_deepsleepmode(PMU_LDNPDSP_NORMALDRIVE, WFI_CMD, PMU_DEEPSLEEP);

pmu_to_standbymode

The description of pmu_to_standbymode is shown as below:

Table 3-494. Function pmu_to_standbymode

<table>
<thead>
<tr>
<th>Function name</th>
<th>pmu_to_standbymode</th>
</tr>
</thead>
<tbody>
<tr>
<td>Function prototype</td>
<td>void pmu_to_standbymode(uint8_t standbymodecmd);</td>
</tr>
<tr>
<td>Function descriptions</td>
<td>pmu work at standby mode</td>
</tr>
</tbody>
</table>
The called functions

standbymodecmd  command to enter standby mode
  WFI_CMD  use WFI command
  WFE_CMD  use WFE command

Output parameter(out)

Return value

Example:

/* PMU work at standby mode */

pmu_to_standby(WFI_CMD);

pmu_wakeup_pin_enable

The description of pmu_wakeup_pin_enable is shown as below:

<table>
<thead>
<tr>
<th>Function name</th>
<th>pmu_wakeup_pin_enable</th>
</tr>
</thead>
<tbody>
<tr>
<td>Function prototype</td>
<td>void pmu_wakeup_pin_enable(uint32_t wakeup_pin);</td>
</tr>
<tr>
<td>Function descriptions</td>
<td>enable wakeup pin</td>
</tr>
</tbody>
</table>

| Precondition     | - |
| The called functions | - |

<table>
<thead>
<tr>
<th>Input parameter(in)</th>
</tr>
</thead>
<tbody>
<tr>
<td>wakeup_pin</td>
</tr>
<tr>
<td>PMU_WAKEUP_PIN0</td>
</tr>
<tr>
<td>PMU_WAKEUP_PIN1</td>
</tr>
<tr>
<td>PMU_WAKEUP_PIN2</td>
</tr>
<tr>
<td>PMU_WAKEUP_PIN3</td>
</tr>
<tr>
<td>PMU_WAKEUP_PIN4</td>
</tr>
</tbody>
</table>

Output parameter(out)

Return value

Example:

/* enable wakeup pin0 */

pmu_wakeup_pin_enable(PMU_WAKEUP_PIN0);
pmu_wakeup_pin_disable

The description of pmu_wakeup_pin_disable is shown as below:

Table 3-496. Function pmu_wakeup_pin_disable

<table>
<thead>
<tr>
<th>Function name</th>
<th>pmu_wakeup_pin_disable</th>
</tr>
</thead>
<tbody>
<tr>
<td>Function prototype</td>
<td>void pmu_wakeup_pin_disable(uint32_t wakeup_pin);</td>
</tr>
<tr>
<td>Function descriptions</td>
<td>disable wakeup pin</td>
</tr>
<tr>
<td>Precondition</td>
<td>-</td>
</tr>
<tr>
<td>The called functions</td>
<td>-</td>
</tr>
<tr>
<td></td>
<td><strong>Input parameter</strong>(in)</td>
</tr>
<tr>
<td>wakeup_pin</td>
<td>Wakeup pin</td>
</tr>
<tr>
<td>PMU_WAKEUP_PIN0</td>
<td>WKUP Pin 0 (PA0)</td>
</tr>
<tr>
<td>PMU_WAKEUP_PIN1</td>
<td>WKUP Pin 1 (PC13)</td>
</tr>
<tr>
<td>PMU_WAKEUP_PIN2</td>
<td>WKUP Pin 2 (PA2)</td>
</tr>
<tr>
<td>PMU_WAKEUP_PIN3</td>
<td>WKUP Pin 3 (PB2)</td>
</tr>
<tr>
<td>PMU_WAKEUP_PIN4</td>
<td>WKUP Pin 4 (PC6)</td>
</tr>
<tr>
<td></td>
<td><strong>Output parameter</strong>(out)</td>
</tr>
<tr>
<td></td>
<td>-</td>
</tr>
<tr>
<td></td>
<td><strong>Return value</strong></td>
</tr>
<tr>
<td></td>
<td>-</td>
</tr>
</tbody>
</table>

Example:

/* disable wakeup pin0 */

pmu_wakeup_pin_disable(PMU_WAKEUP_PIN0);

pmu_backup_write_enable

The description of pmu_backup_write_enable is shown as below:

Table 3-497. Function pmu_backup_write_enable

<table>
<thead>
<tr>
<th>Function name</th>
<th>pmu_backup_write_enable</th>
</tr>
</thead>
<tbody>
<tr>
<td>Function prototype</td>
<td>void pmu_backup_write_enable(void);</td>
</tr>
<tr>
<td>Function descriptions</td>
<td>enable backup domain write</td>
</tr>
<tr>
<td>Precondition</td>
<td>-</td>
</tr>
<tr>
<td>The called functions</td>
<td>-</td>
</tr>
<tr>
<td></td>
<td><strong>Input parameter</strong>(in)</td>
</tr>
<tr>
<td></td>
<td>-</td>
</tr>
<tr>
<td></td>
<td><strong>Output parameter</strong>(out)</td>
</tr>
<tr>
<td></td>
<td>-</td>
</tr>
<tr>
<td></td>
<td><strong>Return value</strong></td>
</tr>
<tr>
<td></td>
<td>-</td>
</tr>
</tbody>
</table>

Example:
/* enable backup domain write */

pmu_backup_write_enable();

pmu_backup_write_disable

The description of pmu_backup_write_disable is shown as below:

Table 3-498. Function pmu_backup_write_disable

<table>
<thead>
<tr>
<th>Function name</th>
<th>pmu_backup_write_disable</th>
</tr>
</thead>
<tbody>
<tr>
<td>Function prototype</td>
<td>void pmu_backup_write_disable(void);</td>
</tr>
<tr>
<td>Function descriptions</td>
<td>disable backup domain write</td>
</tr>
<tr>
<td>Precondition</td>
<td>-</td>
</tr>
<tr>
<td>The called functions</td>
<td>-</td>
</tr>
<tr>
<td>Input parameter(in)</td>
<td>-</td>
</tr>
<tr>
<td>Output parameter(out)</td>
<td>-</td>
</tr>
<tr>
<td>Return value</td>
<td>-</td>
</tr>
</tbody>
</table>

Example:

/* disable backup domain write */

pmu_backup_write_disable();

pmu_sram_power_config

The description of pmu_sram_power_config is shown as below:

Table 3-499. Function pmu_sram_power_config

<table>
<thead>
<tr>
<th>Function name</th>
<th>pmu_sram_power_config</th>
</tr>
</thead>
<tbody>
<tr>
<td>Function prototype</td>
<td>void pmu_sram_power_config(uint32_t state);</td>
</tr>
<tr>
<td>Function descriptions</td>
<td>configure power state of SRAM1</td>
</tr>
<tr>
<td>Precondition</td>
<td>-</td>
</tr>
<tr>
<td>The called functions</td>
<td>-</td>
</tr>
<tr>
<td>Input parameter(in)</td>
<td>state</td>
</tr>
<tr>
<td></td>
<td>PMU_SRAM1_SLEEP</td>
</tr>
<tr>
<td></td>
<td>PMU_SRAM1_WAKE</td>
</tr>
<tr>
<td>Output parameter(out)</td>
<td>-</td>
</tr>
<tr>
<td>Return value</td>
<td>-</td>
</tr>
</tbody>
</table>

Example:
pmu_sram_power_config(PMU_SRAM1_WAKE);

pmu_core1_power_config

The description of pmu_core1_power_config is shown as below:

Table 3-500. Function pmu_core1_power_config

<table>
<thead>
<tr>
<th>Function name</th>
<th>pmu_core1_power_config</th>
</tr>
</thead>
<tbody>
<tr>
<td>Function prototype</td>
<td>void pmu_core1_power_config(uint32_t state);</td>
</tr>
<tr>
<td>Function descriptions</td>
<td>configure power state of COREOFF1 domain</td>
</tr>
<tr>
<td>Precondition</td>
<td>-</td>
</tr>
<tr>
<td>The called functions</td>
<td>-</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Input parameter(in)</th>
</tr>
</thead>
<tbody>
<tr>
<td>state</td>
</tr>
<tr>
<td>PMU_CORE1_SLEEP</td>
</tr>
<tr>
<td>COREOFF1 domain go to power-off</td>
</tr>
<tr>
<td>PMU_CORE1_WAKE</td>
</tr>
<tr>
<td>COREOFF1 domain wakeup</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Output parameter(out)</th>
</tr>
</thead>
<tbody>
<tr>
<td>-</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Return value</th>
</tr>
</thead>
<tbody>
<tr>
<td>-</td>
</tr>
</tbody>
</table>

Example:

/* configure power state of COREOFF1 domain */

pmu_core1_power_config(PMU_SRAM1_WAKE);

pmu_deepsleep2_retention_enable

The description of pmu_deepsleep2_retention_enable is shown as below:

Table 3-501. Function pmu_deepsleep2_retention_enable

<table>
<thead>
<tr>
<th>Function name</th>
<th>pmu_deepsleep2_retention_enable</th>
</tr>
</thead>
<tbody>
<tr>
<td>Function prototype</td>
<td>void pmu_deepsleep2_retention_enable(void);</td>
</tr>
<tr>
<td>Function descriptions</td>
<td>have retention register in Deep-sleep 2</td>
</tr>
<tr>
<td>Precondition</td>
<td>-</td>
</tr>
<tr>
<td>The called functions</td>
<td>-</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Input parameter(in)</th>
</tr>
</thead>
<tbody>
<tr>
<td>-</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Output parameter(out)</th>
</tr>
</thead>
<tbody>
<tr>
<td>-</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Return value</th>
</tr>
</thead>
<tbody>
<tr>
<td>-</td>
</tr>
</tbody>
</table>

Example:
pmu_deepsleep2_retention_enable();

**pmu_deepsleep2_retention_disable**

The description of pmu_deepsleep2_retention_disable is shown as below:

<table>
<thead>
<tr>
<th>Function name</th>
<th>pmu_deepsleep2_retention_disable</th>
</tr>
</thead>
<tbody>
<tr>
<td>Function prototype</td>
<td>void pmu_deepsleep2_retention_disable(void);</td>
</tr>
<tr>
<td>Function descriptions</td>
<td>no retention register in Deep-sleep 2</td>
</tr>
</tbody>
</table>

| Precondition | - |
| The called functions | - |
| Input parameter(in) | - |
| Output parameter(out) | - |
| Return value | - |

**Example:**

```c
/* no retention register in Deep-sleep 2 */

pmu_deepsleep2_retention_disable();
```

**pmu_deepsleep2_sram_power_config**

The description of pmu_deepsleep2_sram_power_config is shown as below:

<table>
<thead>
<tr>
<th>Function name</th>
<th>pmu_deepsleep2_sram_power_config</th>
</tr>
</thead>
<tbody>
<tr>
<td>Function prototype</td>
<td>void pmu_deepsleep2_sram_power_config(uint32_t state);</td>
</tr>
<tr>
<td>Function descriptions</td>
<td>configure SRAM1 power state when enter Deep-sleep 2</td>
</tr>
</tbody>
</table>

| Precondition | - |
| The called functions | - |
| Input parameter(in) | - |
| state | - |
| PMU_SRAM1_POWER_OFF | power state of SRAM1 |
| PMU_SRAM1_POWER_REMAIN | SRAM1 power same as Run/Run1/Run2 mode |
| Output parameter(out) | - |

| Return value | - |
Example:

/* configure SRAM1 power state when enter Deep-sleep 2 */

pmu_deepsleep2_sram_power_config(PMU_SRAM1_POWER_OFF);

pmu_deepsleep_wait_time_config

The description of pmu_deepsleep_wait_time_config is shown as below:

Table 3-504. Function pmu_deepsleep_wait_time_config

<table>
<thead>
<tr>
<th>Function name</th>
<th>pmu_deepsleep_wait_time_config</th>
</tr>
</thead>
<tbody>
<tr>
<td>Function prototype</td>
<td>void pmu_deepsleep_wait_time_config(uint32_t wait_time);</td>
</tr>
<tr>
<td>Function descriptions</td>
<td>configure IRC16M counter before enter Deep-sleep mode</td>
</tr>
<tr>
<td>Precondition</td>
<td>-</td>
</tr>
<tr>
<td>The called functions</td>
<td>-</td>
</tr>
</tbody>
</table>

Input parameter(in)

<table>
<thead>
<tr>
<th>wait_time</th>
<th>IRC16M counter before enter Deep-sleep mode</th>
</tr>
</thead>
<tbody>
<tr>
<td>uint32_t</td>
<td>0x0~0x1F</td>
</tr>
</tbody>
</table>

Output parameter(out)

- |

Return value

- |

Example:

/* configure IRC16M counter before enter Deep-sleep mode */

pmu_deepsleep_wait_time_config(1);

pmu_wakeuptime_core1_software_enable

The description of pmu_wakeuptime_core1_software_enable is shown as below:

Table 3-505. Function pmu_wakeuptime_core1_software_enable

<table>
<thead>
<tr>
<th>Function name</th>
<th>pmu_wakeuptime_core1_software_enable</th>
</tr>
</thead>
<tbody>
<tr>
<td>Function prototype</td>
<td>void pmu_wakeuptime_core1_software_enable(uint32_t wakeup_time);</td>
</tr>
<tr>
<td>Function descriptions</td>
<td>use software value signal when wake up COREOFF1 domain</td>
</tr>
<tr>
<td>Precondition</td>
<td>-</td>
</tr>
<tr>
<td>The called functions</td>
<td>-</td>
</tr>
</tbody>
</table>

Input parameter(in)

<table>
<thead>
<tr>
<th>wakeup_time</th>
<th>wakeup time of power switch of COREOFF1 domain</th>
</tr>
</thead>
<tbody>
<tr>
<td>uint32_t</td>
<td>0x0~0xFF</td>
</tr>
</tbody>
</table>

Output parameter(out)

- |
Example:

/* use software value signal when wake up COREOFF1 domain */

pmu_wakeuptime_core1_software_enable(1);

pmu_wakeuptime_core1_software_disable

The description of pmu_wakeuptime_core1_software_disable is shown as below:

Table 3-506. Function pmu_wakeuptime_core1_software_disable

<table>
<thead>
<tr>
<th>Function name</th>
<th>pmu_wakeuptime_core1_software_disable</th>
</tr>
</thead>
<tbody>
<tr>
<td>Function prototype</td>
<td>void pmu_wakeuptime_core1_software_disable(void);</td>
</tr>
<tr>
<td>Function descriptions</td>
<td>use hardware ack signal when wake up COREOFF1 domain</td>
</tr>
<tr>
<td>Precondition</td>
<td>-</td>
</tr>
<tr>
<td>The called functions</td>
<td>-</td>
</tr>
<tr>
<td>Input parameter(in)</td>
<td>-</td>
</tr>
<tr>
<td>Output parameter(out)</td>
<td>-</td>
</tr>
<tr>
<td>Return value</td>
<td>-</td>
</tr>
</tbody>
</table>

Example:

/* use hardware ack signal when wake up COREOFF1 domain */

pmu_wakeuptime_core1_software_disable();

pmu_wakeuptime_sram_software_enable

The description of pmu_wakeuptime_sram_software_enable is shown as below:

Table 3-507. Function pmu_wakeuptime_sram_software_enable

<table>
<thead>
<tr>
<th>Function name</th>
<th>pmu_wakeuptime_sram_software_enable</th>
</tr>
</thead>
<tbody>
<tr>
<td>Function prototype</td>
<td>void pmu_wakeuptime_sram_software_enable(uint32_t wakeup_time);</td>
</tr>
<tr>
<td>Function descriptions</td>
<td>use software value signal when wake up SRAM1</td>
</tr>
<tr>
<td>Precondition</td>
<td>-</td>
</tr>
<tr>
<td>The called functions</td>
<td>-</td>
</tr>
<tr>
<td>Input parameter(in)</td>
<td>wakeup_time</td>
</tr>
<tr>
<td></td>
<td>uint32_t</td>
</tr>
<tr>
<td></td>
<td>0x0~0xFF</td>
</tr>
<tr>
<td>Output parameter(out)</td>
<td>-</td>
</tr>
</tbody>
</table>

Example:

/* use software value signal when wake up SRAM1 */

pmu_wakeuptime_sram_software_enable(0x0);

pmu_wakeuptime_sram_software_disable
/* use software value signal when wake up SRAM1 */

pmu_wakeuptime_sram_software_enable(1);

**pmu_wakeuptime_sram_software_disable**

The description of `pmu_wakeuptime_sram_software_disable` is shown as below:

**Table 3-508. Function pmu_wakeuptime_sram_software_disable**

<table>
<thead>
<tr>
<th>Function name</th>
<th>pmu_wakeuptime_sram_software_disable</th>
</tr>
</thead>
<tbody>
<tr>
<td>Function prototype</td>
<td>void pmu_wakeuptime_sram_software_disable(void);</td>
</tr>
<tr>
<td>Function descriptions</td>
<td>use hardware ack signal when wake up SRAM1</td>
</tr>
<tr>
<td>Precondition</td>
<td>-</td>
</tr>
<tr>
<td>The called functions</td>
<td>-</td>
</tr>
<tr>
<td>Input parameter(in)</td>
<td>-</td>
</tr>
<tr>
<td>Output parameter(out)</td>
<td>-</td>
</tr>
<tr>
<td>Return value</td>
<td>-</td>
</tr>
</tbody>
</table>

/* use hardware ack signal when wake up SRAM1 */

pmu_wakeuptime_sram_software_disable();

**pmu_wakeuptime_deepsleep2_software_enable**

The description of `pmu_wakeuptime_deepsleep2_software_enable` is shown as below:

**Table 3-509. Function pmu_wakeuptime_deepsleep2_software_enable**

<table>
<thead>
<tr>
<th>Function name</th>
<th>pmu_wakeuptime_deepsleep2_software_enable</th>
</tr>
</thead>
<tbody>
<tr>
<td>Function prototype</td>
<td>void pmu_wakeuptime_deepsleep2_software_enable(uint32_t wakeup_time);</td>
</tr>
<tr>
<td>Function descriptions</td>
<td>use software value signal when wake up Deep-sleep 2</td>
</tr>
<tr>
<td>Precondition</td>
<td>-</td>
</tr>
<tr>
<td>The called functions</td>
<td>-</td>
</tr>
<tr>
<td>Input parameter(in)</td>
<td>wakeup_time</td>
</tr>
<tr>
<td>Output parameter(out)</td>
<td>uint32_t</td>
</tr>
<tr>
<td></td>
<td>wakeup time of power switch of COREOFF0 domain</td>
</tr>
<tr>
<td></td>
<td>0x0~0xFF</td>
</tr>
</tbody>
</table>

Example:

/* use hardware ack signal when wake up SRAM1 */

pmu_wakeuptime_deepsleep2_software_enable(0x0);

pmu_wakeuptime_deepsleep2_software_enable(0xFF);
Example:

```c
/* use software value signal when wake up Deep-sleep 2 */

pmu_wakeuptime_deepsleep2_software_enable(1);
```

### pmu_wakeuptime_deepsleep2_software_disable

The description of `pmu_wakeuptime_deepsleep2_software_disable` is shown as below:

<table>
<thead>
<tr>
<th>Function name</th>
<th>pmu_wakeuptime_deepsleep2_software_disable</th>
</tr>
</thead>
<tbody>
<tr>
<td>Function prototype</td>
<td>void pmu_wakeuptime_deepsleep2_software_disable(void);</td>
</tr>
<tr>
<td>Function descriptions</td>
<td>use hardware ack signal when wake up Deep-sleep 2</td>
</tr>
<tr>
<td>Precondition</td>
<td>-</td>
</tr>
<tr>
<td>The called functions</td>
<td>-</td>
</tr>
<tr>
<td>Input parameter(in)</td>
<td>-</td>
</tr>
<tr>
<td>Output parameter(out)</td>
<td>-</td>
</tr>
<tr>
<td>Return value</td>
<td>-</td>
</tr>
</tbody>
</table>

Example:

```c
/* use hardware ack signal when wake up Deep-sleep 2 */

pmu_wakeuptime_deepsleep2_software_disable();
```

### pmu_flag_get

The description of `pmu_flag_get` is shown as below:

<table>
<thead>
<tr>
<th>Function name</th>
<th>pmu_flag_get</th>
</tr>
</thead>
<tbody>
<tr>
<td>Function prototype</td>
<td>FlagStatus pmu_flag_get(uint32_t flag);</td>
</tr>
<tr>
<td>Function descriptions</td>
<td>get PMU flag status</td>
</tr>
<tr>
<td>Precondition</td>
<td>-</td>
</tr>
<tr>
<td>The called functions</td>
<td>-</td>
</tr>
<tr>
<td>Input parameter(in)</td>
<td>flag</td>
</tr>
<tr>
<td>flag</td>
<td>flag</td>
</tr>
<tr>
<td>PMU_FLAG_WAKEUP</td>
<td>wakeup flag</td>
</tr>
<tr>
<td>PMU_FLAG_STANDBY</td>
<td>standby flag</td>
</tr>
</tbody>
</table>
PMU_FLAG_LVD | lvd flag
PMU_FLAG_LDOVSRF | LDO voltage select ready flag
PMU_FLAG_NPRDY | normal-power LDO ready flag
PMU_FLAG_LPRDY | low-power LDO ready flag
PMU_FLAG_SRAM1_SLEEP | SRAM1 is in sleep state flag
PMU_FLAG_SRAM1_ACTIVE | SRAM1 is in active state flag
PMU_FLAG_CORE1_SLEEP | COREOFF1 domain is in sleep state flag
PMU_FLAG_CORE1_ACTIVE | COREOFF1 domain is in active state flag
PMU_FLAG_DEEPSLEEP_EP_2 | Deep-sleep 2 mode status flag

Output parameter(out)
- -

Return value
FlagStatus | SET or RESET

Example:

/* get flag state */
FlagStatus status;
status = pmu_flag_get(PMU_FLAG_WAKEUP);

pmu_flag_clear

The description of pmu_flag_clear is shown as below:

<table>
<thead>
<tr>
<th>Function name</th>
<th>pmu_flag_clear</th>
</tr>
</thead>
<tbody>
<tr>
<td>Function prototype</td>
<td>void pmu_flag_clear(uint32_t flag);</td>
</tr>
<tr>
<td>Function descriptions</td>
<td>clear PMU flag status</td>
</tr>
<tr>
<td>Precondition</td>
<td>-</td>
</tr>
<tr>
<td>The called functions</td>
<td>-</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Input parameter(in)</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>flag</td>
<td>flag</td>
</tr>
<tr>
<td>PMU_FLAG_WAKEUP</td>
<td>wakeup flag</td>
</tr>
<tr>
<td>PMU_FLAG_STANDBY</td>
<td>standby flag</td>
</tr>
<tr>
<td>PMU_FLAG_DEEPSLEEP_EP_2</td>
<td>Deep-sleep 2 mode status flag</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Output parameter(out)</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>-</td>
<td>-</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Return value</th>
</tr>
</thead>
</table>

Example:

```
/* clear flag bit */

pmu_flag_clear(PMU_FLAG_STANDBY);
```

## 3.19. RCU

RCU is the reset and clock unit. Reset Control includes the control of three kinds of reset: power reset, system reset and backup domain reset. The Clock Control unit provides a range of frequencies and clock functions. The RCU registers are listed in chapter 3.19.1, the RCU firmware functions are introduced in chapter 3.19.2.

### 3.19.1. Descriptions of Peripheral registers

**Table 3-513. RCU Registers**

<table>
<thead>
<tr>
<th>Registers</th>
<th>Descriptions</th>
</tr>
</thead>
<tbody>
<tr>
<td>RCU_CTL0</td>
<td>Control register 0</td>
</tr>
<tr>
<td>RCU_CFG0</td>
<td>Clock configuration register 0</td>
</tr>
<tr>
<td>RCU_INT</td>
<td>Clock interrupt register</td>
</tr>
<tr>
<td>RCU_APB2RST</td>
<td>APB2 reset register</td>
</tr>
<tr>
<td>RCU_APB1RST</td>
<td>APB1 reset register</td>
</tr>
<tr>
<td>RCU_AHBEN</td>
<td>AHB enable register</td>
</tr>
<tr>
<td>RCU_APB2EN</td>
<td>APB2 enable register</td>
</tr>
<tr>
<td>RCU_APB1EN</td>
<td>APB1 enable register</td>
</tr>
<tr>
<td>RCU_BDCTL</td>
<td>Backup domain control register</td>
</tr>
<tr>
<td>RCU_RSTSCK</td>
<td>Reset source/clock register</td>
</tr>
<tr>
<td>RCU_AHB1RST</td>
<td>AHB reset register</td>
</tr>
<tr>
<td>RCU_CFG1</td>
<td>Clock configuration register 1</td>
</tr>
<tr>
<td>RCU_CFG2</td>
<td>Clock configuration register 2</td>
</tr>
<tr>
<td>RCU_AHB2EN</td>
<td>AHB2 enable register</td>
</tr>
<tr>
<td>RCU_AHB2RST</td>
<td>AHB2 reset register</td>
</tr>
<tr>
<td>RCU_CTL1</td>
<td>Control register 1</td>
</tr>
<tr>
<td>RCU_VKEY</td>
<td>Voltage key register</td>
</tr>
<tr>
<td>RCU_LPLDO</td>
<td>Low-Power mode LDO voltage register</td>
</tr>
<tr>
<td>RCU_LPB</td>
<td>Low-Power bandgap mode register</td>
</tr>
</tbody>
</table>

### 3.19.2. Descriptions of Peripheral functions

**Table 3-514. RCU firmware function**

<table>
<thead>
<tr>
<th>Function name</th>
<th>Function description</th>
</tr>
</thead>
<tbody>
<tr>
<td>rcu_deinit</td>
<td>deinitialize the RCU</td>
</tr>
<tr>
<td>Function name</td>
<td>Function description</td>
</tr>
<tr>
<td>------------------------------------</td>
<td>---------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>rcu_periph_clock_enable</td>
<td>enable the peripherals clock</td>
</tr>
<tr>
<td>rcu_periph_clock_disable</td>
<td>disable the peripherals clock</td>
</tr>
<tr>
<td>rcu_periph_clock_sleep_enable</td>
<td>enable the peripherals clock when in sleep mode</td>
</tr>
<tr>
<td>rcu_periph_clock_sleep_disable</td>
<td>disable the peripherals clock when in sleep mode</td>
</tr>
<tr>
<td>rcu_periph_reset_enable</td>
<td>enable the peripherals reset</td>
</tr>
<tr>
<td>rcu_periph_reset_disable</td>
<td>disable the peripheral reset</td>
</tr>
<tr>
<td>rcu_bkp_reset_enable</td>
<td>enable the BKP domain reset</td>
</tr>
<tr>
<td>rcu_bkp_reset_disable</td>
<td>disable the BKP domain reset</td>
</tr>
<tr>
<td>rcu_system_clock_source_config</td>
<td>configure the system clock source</td>
</tr>
<tr>
<td>rcu_system_clock_source_get</td>
<td>get the system clock source</td>
</tr>
<tr>
<td>rcu_ahb_clock_config</td>
<td>configure the AHB clock prescaler selection</td>
</tr>
<tr>
<td>rcu_apb1_clock_config</td>
<td>configure the APB1 clock prescaler selection</td>
</tr>
<tr>
<td>rcu_apb2_clock_config</td>
<td>configure the APB2 clock prescaler selection</td>
</tr>
<tr>
<td>rcu_adc_clock_config</td>
<td>configure the ADC clock source and prescaler selection</td>
</tr>
<tr>
<td>rcu_ckout_config</td>
<td>configure the CK_OUT clock source and prescaler</td>
</tr>
<tr>
<td>rcu_pll_config</td>
<td>configure the main PLL clock</td>
</tr>
<tr>
<td>rcu_usart_clock_config</td>
<td>configure the USART clock</td>
</tr>
<tr>
<td>rcu_i2c_clock_config</td>
<td>configure the I2Cx(x=0,1,2) clock source selection</td>
</tr>
<tr>
<td>rcu_lptimer_clock_config</td>
<td>configure the LPTIMER clock source selection</td>
</tr>
<tr>
<td>rcu_lpuart_clock_config</td>
<td>configure the LPUART clock source selection</td>
</tr>
<tr>
<td>rcu_irc16mdiv_clock_config</td>
<td>configure the IRC16MDIV clock selection</td>
</tr>
<tr>
<td>rcu_usbd_clock_config</td>
<td>configure the USBD clock source selection</td>
</tr>
<tr>
<td>rcu_rtc_clock_config</td>
<td>configure the RTC clock source selection</td>
</tr>
<tr>
<td>rcu_pll_source_ck_prediv_config</td>
<td>configure PLL source clocks pre-divider</td>
</tr>
<tr>
<td>rcu_lxtal_drive_capability_config</td>
<td>configure the LXTAL drive capability</td>
</tr>
<tr>
<td>rcu_ip_ldo_config</td>
<td>configure the low power mode LDO voltage selection</td>
</tr>
<tr>
<td>rcu_ip_bandgap_config</td>
<td>configure low power bandgap mode selection</td>
</tr>
<tr>
<td>rcu_flag_get</td>
<td>get the clock stabilization and peripheral reset flags</td>
</tr>
<tr>
<td>rcu_all_reset_flag_clear</td>
<td>clear all the reset flag</td>
</tr>
<tr>
<td>rcu_interrupt_flag_get</td>
<td>get the clock stabilization interrupt and ckm flags</td>
</tr>
<tr>
<td>rcu_interrupt_flag_clear</td>
<td>clear the interrupt flags</td>
</tr>
<tr>
<td>rcu_interrupt_enable</td>
<td>enable the stabilization interrupt</td>
</tr>
<tr>
<td>rcu_interrupt_disable</td>
<td>disable the stabilization interrupt</td>
</tr>
<tr>
<td>rcu_osci_stab_wait</td>
<td>wait for oscillator stabilization flags is SET or oscillator startup is timeout</td>
</tr>
<tr>
<td>rcu_osci_on</td>
<td>turn on the oscillator</td>
</tr>
<tr>
<td>rcu_osci_off</td>
<td>turn off the oscillator</td>
</tr>
<tr>
<td>rcu_osci_bypass_mode_enable</td>
<td>enable the oscillator bypass mode</td>
</tr>
<tr>
<td>rcu_osci_bypass_mode_disable</td>
<td>disable the oscillator bypass mode</td>
</tr>
<tr>
<td>rcu_hxtal_clock_monitor_enable</td>
<td>enable the HXTAL clock monitor</td>
</tr>
<tr>
<td>rcu_hxtal_clock_monitor_disable</td>
<td>disable the HXTAL clock monitor</td>
</tr>
<tr>
<td>Function name</td>
<td>Function description</td>
</tr>
<tr>
<td>----------------------------------</td>
<td>--------------------------------------------</td>
</tr>
<tr>
<td>rcu_lxtal_clock_monitor_enable</td>
<td>enable the LXTAL clock monitor</td>
</tr>
<tr>
<td>rcu_lxtal_clock_monitor_disable</td>
<td>disable the LXTAL clock monitor</td>
</tr>
<tr>
<td>rcu irc16m_adjust_value_set</td>
<td>set the IRC16M adjust value</td>
</tr>
<tr>
<td>rcu_voltage_key_unlock</td>
<td>unlock the voltage key</td>
</tr>
<tr>
<td>rcu_clock_freq_get</td>
<td>get the system clock, bus and peripheral clock frequency</td>
</tr>
</tbody>
</table>

**Enum rcu_periph_enum**

Table 3-515. Enum rcu_periph_enum

<table>
<thead>
<tr>
<th>enum name</th>
<th>Function description</th>
</tr>
</thead>
<tbody>
<tr>
<td>RCU_DMA</td>
<td>DMA clock</td>
</tr>
<tr>
<td>RCU_CAU</td>
<td>CAU clock</td>
</tr>
<tr>
<td>RCU_TRNG</td>
<td>TRNG clock</td>
</tr>
<tr>
<td>RCU_CRC</td>
<td>CRC clock</td>
</tr>
<tr>
<td>RCU_GPIOA</td>
<td>GPIOA clock</td>
</tr>
<tr>
<td>RCU_GPIOB</td>
<td>GPIOB clock</td>
</tr>
<tr>
<td>RCU_GPIOC</td>
<td>GPIOC clock</td>
</tr>
<tr>
<td>RCU_GPIOD</td>
<td>GPIOD clock</td>
</tr>
<tr>
<td>RCU_GPIOF</td>
<td>GPIOF clock</td>
</tr>
<tr>
<td>RCU_SYSCFG</td>
<td>SYSCFG clock</td>
</tr>
<tr>
<td>RCU_CMP</td>
<td>CMP clock</td>
</tr>
<tr>
<td>RCU_ADC</td>
<td>ADC clock</td>
</tr>
<tr>
<td>RCU_TIMER8</td>
<td>TIMER8 clock</td>
</tr>
<tr>
<td>RCU_SPI0</td>
<td>SPI0 clock</td>
</tr>
<tr>
<td>RCU_USART0</td>
<td>USART0 clock</td>
</tr>
<tr>
<td>RCU_DBGMCU</td>
<td>DBGMCU clock</td>
</tr>
<tr>
<td>RCU_TIMER1</td>
<td>TIMER1 clock</td>
</tr>
<tr>
<td>RCU_TIMER2</td>
<td>TIMER2 clock</td>
</tr>
<tr>
<td>RCU_TIMER5</td>
<td>TIMER5 clock</td>
</tr>
<tr>
<td>RCU_TIMER6</td>
<td>TIMER6 clock</td>
</tr>
<tr>
<td>RCU_TIMER11</td>
<td>TIMER11 clock</td>
</tr>
<tr>
<td>RCU_LPTIMER</td>
<td>LPTIMER clock</td>
</tr>
<tr>
<td>RCU_SLCD</td>
<td>SLCD clock</td>
</tr>
<tr>
<td>RCU_WWDGT</td>
<td>WWDGT clock</td>
</tr>
<tr>
<td>RCU_SPI1</td>
<td>SPI1 clock</td>
</tr>
<tr>
<td>RCU_USART1</td>
<td>USART1 clock</td>
</tr>
<tr>
<td>RCU_LPUART</td>
<td>LPUART clock</td>
</tr>
<tr>
<td>RCU_UART3</td>
<td>UART3 clock</td>
</tr>
<tr>
<td>RCU_UART4</td>
<td>UART4 clock</td>
</tr>
<tr>
<td>RCU_I2C0</td>
<td>I2C0 clock</td>
</tr>
<tr>
<td>RCU_I2C1</td>
<td>I2C1 clock</td>
</tr>
<tr>
<td>RCU_USBD</td>
<td>USBD clock</td>
</tr>
<tr>
<td>enum name</td>
<td>Function description</td>
</tr>
<tr>
<td>------------------</td>
<td>----------------------</td>
</tr>
<tr>
<td>RCU_I2C2</td>
<td>I2C2 clock</td>
</tr>
<tr>
<td>RCU_PMU</td>
<td>PMU clock</td>
</tr>
<tr>
<td>RCU_DAC</td>
<td>DAC clock</td>
</tr>
<tr>
<td>RCU_CTC</td>
<td>CTC clock</td>
</tr>
<tr>
<td>RCU_BKP</td>
<td>BKP clock</td>
</tr>
<tr>
<td>RCU_RTC</td>
<td>RTC clock</td>
</tr>
</tbody>
</table>

**Enum rcu_periph_sleep_enum**

Table 3-516. Enum rcu_periph_sleep_enum

<table>
<thead>
<tr>
<th>enum name</th>
<th>Function description</th>
</tr>
</thead>
<tbody>
<tr>
<td>RCU_SRAM0_SLP</td>
<td>SRAM0 clock</td>
</tr>
<tr>
<td>RCU_FMC_SLP</td>
<td>FMC clock</td>
</tr>
<tr>
<td>RCU_SRAM1_SLP</td>
<td>SRAM1 clock</td>
</tr>
</tbody>
</table>

**Enum rcu_periph_reset_enum**

Table 3-517. Enum rcu_periph_reset_enum

<table>
<thead>
<tr>
<th>enum name</th>
<th>Function description</th>
</tr>
</thead>
<tbody>
<tr>
<td>RCU_CAURST</td>
<td>SRAM0 clock</td>
</tr>
<tr>
<td>RCU_TRNGRST</td>
<td>TRNG reset</td>
</tr>
<tr>
<td>RCU_CRCRCST</td>
<td>CRC reset</td>
</tr>
<tr>
<td>RCU_GPIOARST</td>
<td>GPIOA reset</td>
</tr>
<tr>
<td>RCU_GPIOBRST</td>
<td>GPIOB reset</td>
</tr>
<tr>
<td>RCU_GPIOCRST</td>
<td>GPIOC reset</td>
</tr>
<tr>
<td>RCU_GPIODRST</td>
<td>GPIOD reset</td>
</tr>
<tr>
<td>RCU_GPIOFRST</td>
<td>GPIOF reset</td>
</tr>
<tr>
<td>RCU_SYSCFGRST</td>
<td>SYSCFG reset</td>
</tr>
<tr>
<td>RCU_CMPRST</td>
<td>CMP reset</td>
</tr>
<tr>
<td>RCU_ADCRST</td>
<td>ADC reset</td>
</tr>
<tr>
<td>RCU_TIMER8RST</td>
<td>TIMER8 reset</td>
</tr>
<tr>
<td>RCU_SPI0RST</td>
<td>SPI0 reset</td>
</tr>
<tr>
<td>RCU_USART0RST</td>
<td>USART0 reset</td>
</tr>
<tr>
<td>RCU_TIMER1RST</td>
<td>TIMER1 reset</td>
</tr>
<tr>
<td>RCU_TIMER2RST</td>
<td>TIMER2 reset</td>
</tr>
<tr>
<td>RCU_TIMER5RST</td>
<td>TIMER5 reset</td>
</tr>
<tr>
<td>RCU_TIMER6RST</td>
<td>TIMER6 reset</td>
</tr>
<tr>
<td>RCU_TIMER11RST</td>
<td>TIMER11 reset</td>
</tr>
<tr>
<td>RCU_LPTIMERRST</td>
<td>LPTIMER reset</td>
</tr>
<tr>
<td>RCU_SLCDRST</td>
<td>SLCD reset</td>
</tr>
<tr>
<td>RCU_WWDGTRST</td>
<td>WWDGDT reset</td>
</tr>
<tr>
<td>RCU_SPI1RST</td>
<td>SPI1 reset</td>
</tr>
<tr>
<td>enum name</td>
<td>Function description</td>
</tr>
<tr>
<td>--------------------</td>
<td>-----------------------------</td>
</tr>
<tr>
<td>RCU_USART1RST</td>
<td>USART1 reset</td>
</tr>
<tr>
<td>RCU_LPUARTRST</td>
<td>LPUART reset</td>
</tr>
<tr>
<td>RCU_UART3RST</td>
<td>UART3 reset</td>
</tr>
<tr>
<td>RCU_UART4RST</td>
<td>UART4 reset</td>
</tr>
<tr>
<td>RCU_I2C0RST</td>
<td>I2C0 reset</td>
</tr>
<tr>
<td>RCU_I2C1RST</td>
<td>I2C1 reset</td>
</tr>
<tr>
<td>RCU_USBDRST</td>
<td>USBD reset</td>
</tr>
<tr>
<td>RCU_I2C2RST</td>
<td>I2C2 reset</td>
</tr>
<tr>
<td>RCU_PMURST</td>
<td>PMU reset</td>
</tr>
<tr>
<td>RCU_DACRST</td>
<td>DAC reset</td>
</tr>
<tr>
<td>RCU_CTCRST</td>
<td>CTC reset</td>
</tr>
</tbody>
</table>

**Enum `rcu_flag_enum`**

**Table 3-518. Enum `rcu_flag_enum`**

<table>
<thead>
<tr>
<th>enum name</th>
<th>Function description</th>
</tr>
</thead>
<tbody>
<tr>
<td>RCU_FLAG_IRC32KST B</td>
<td>IRC32K stabilization flags</td>
</tr>
<tr>
<td>RCU_FLAG_LXTALST B</td>
<td>LXTAL stabilization flags</td>
</tr>
<tr>
<td>RCU_FLAG_IRC16MS TB</td>
<td>IRC16M stabilization flags</td>
</tr>
<tr>
<td>RCU_FLAG_HXTALST B</td>
<td>HXTAL stabilization flags</td>
</tr>
<tr>
<td>RCU_FLAG_IRC48MS TB</td>
<td>IRC48M stabilization flags</td>
</tr>
<tr>
<td>RCU_FLAG_PLLSTB</td>
<td>PLL stabilization flags</td>
</tr>
<tr>
<td>RCU_FLAG_V12RST</td>
<td>V12 reset flags</td>
</tr>
<tr>
<td>RCU_FLAG_EPRST</td>
<td>EPR reset flags</td>
</tr>
<tr>
<td>RCU_FLAG_PORRST</td>
<td>power reset flags</td>
</tr>
<tr>
<td>RCU_FLAG_SWRST</td>
<td>SW reset flags</td>
</tr>
<tr>
<td>RCU_FLAG_FWDGTR ST</td>
<td>FWDGT reset flags</td>
</tr>
<tr>
<td>RCU_FLAG_WWDGTR ST</td>
<td>WWDGT reset flags</td>
</tr>
<tr>
<td>RCU_FLAG_LPRST</td>
<td>LP reset flags</td>
</tr>
</tbody>
</table>

**Enum `rcu_int_flag_enum`**

**Table 3-519. Enum `rcu_int_flag_enum`**

<table>
<thead>
<tr>
<th>enum name</th>
<th>Function description</th>
</tr>
</thead>
<tbody>
<tr>
<td>RCU_INT_FLAG_IRC3</td>
<td>IRC32K stabilization interrupt flag</td>
</tr>
<tr>
<td>enum name</td>
<td>Function description</td>
</tr>
<tr>
<td>----------------------</td>
<td>-----------------------------------------</td>
</tr>
<tr>
<td>2KSTB</td>
<td></td>
</tr>
<tr>
<td>RCU_INT_FLAG_LXTA</td>
<td>LXTAL stabilization interrupt flag</td>
</tr>
<tr>
<td>LSTB</td>
<td></td>
</tr>
<tr>
<td>RCU_INT_FLAG_IRC1</td>
<td>IRC16M stabilization interrupt flag</td>
</tr>
<tr>
<td>6MSTB</td>
<td></td>
</tr>
<tr>
<td>RCU_INT_FLAG_HXTA</td>
<td>HXTAL stabilization interrupt flag</td>
</tr>
<tr>
<td>LSTB</td>
<td></td>
</tr>
<tr>
<td>RCU_INT_FLAG_PLLS</td>
<td>PLL stabilization interrupt flag</td>
</tr>
<tr>
<td>TB</td>
<td></td>
</tr>
<tr>
<td>RCU_INT_FLAG_IRC4</td>
<td>IRC48M stabilization interrupt flag</td>
</tr>
<tr>
<td>8MSTB</td>
<td></td>
</tr>
<tr>
<td>RCU_INT_FLAG_LXTA</td>
<td>LXTAL clock stuck interrupt flag</td>
</tr>
<tr>
<td>LCKM</td>
<td></td>
</tr>
<tr>
<td>RCU_INT_FLAG_CKM</td>
<td>CKM interrupt flag</td>
</tr>
<tr>
<td>RCU_INT_FLAG_IRC3</td>
<td>IRC32K stabilization interrupt flag</td>
</tr>
<tr>
<td>2KSTB_CLR</td>
<td></td>
</tr>
</tbody>
</table>

**Enum rcu_int_flag_clear_enum**

Table 3-520. Enum rcu_int_flag_clear_enum

<table>
<thead>
<tr>
<th>enum name</th>
<th>Function description</th>
</tr>
</thead>
<tbody>
<tr>
<td>RCU_INT_FLAG_IRC3</td>
<td>IRC32K stabilization interrupt flags clear</td>
</tr>
<tr>
<td>2KSTB_CLR</td>
<td></td>
</tr>
<tr>
<td>RCU_INT_FLAG_LXTA</td>
<td>LXTAL stabilization interrupt flags clear</td>
</tr>
<tr>
<td>LSTB_CLR</td>
<td></td>
</tr>
<tr>
<td>RCU_INT_FLAG_IRC1</td>
<td>IRC16M stabilization interrupt flags clear</td>
</tr>
<tr>
<td>6MSTB_CLR</td>
<td></td>
</tr>
<tr>
<td>RCU_INT_FLAG_HXTA</td>
<td>HXTAL stabilization interrupt flags clear</td>
</tr>
<tr>
<td>LSTB_CLR</td>
<td></td>
</tr>
<tr>
<td>RCU_INT_FLAG_PLLS</td>
<td>PLL stabilization interrupt flags clear</td>
</tr>
<tr>
<td>TB_CLR</td>
<td></td>
</tr>
<tr>
<td>RCU_INT_FLAG_IRC4</td>
<td>IRC48M stabilization interrupt flags clear</td>
</tr>
<tr>
<td>8MSTB_CLR</td>
<td></td>
</tr>
<tr>
<td>RCU_INT_FLAG_LXTA</td>
<td>LXTAL clock stuck interrupt flags clear</td>
</tr>
<tr>
<td>LCKM_CLR</td>
<td></td>
</tr>
<tr>
<td>RCU_INT_FLAG_CKM</td>
<td>CKM interrupt flags clear</td>
</tr>
<tr>
<td>RCU_INT_FLAG_IRC3</td>
<td>IRC32K stabilization interrupt flags clear</td>
</tr>
<tr>
<td>2KSTB_CLR</td>
<td></td>
</tr>
</tbody>
</table>
**Enum rcu_int_enum**

Table 3-521. Enum rcu_int_enum

<table>
<thead>
<tr>
<th>enum name</th>
<th>Function description</th>
</tr>
</thead>
<tbody>
<tr>
<td>RCU_INT_IRC32KSTB</td>
<td>IRC32K stabilization interrupt</td>
</tr>
<tr>
<td>RCU_INT_LXTALSTB</td>
<td>LXTAL stabilization interrupt</td>
</tr>
<tr>
<td>RCU_INT_IRC16MSTB</td>
<td>IRC16M stabilization interrupt</td>
</tr>
<tr>
<td>RCU_INT_HXTALSTB</td>
<td>HXTAL stabilization interrupt</td>
</tr>
<tr>
<td>RCU_INT_PLLSTB</td>
<td>PLL stabilization interrupt</td>
</tr>
<tr>
<td>RCU_INT_IRC48MSTB</td>
<td>IRC48M stabilization interrupt</td>
</tr>
<tr>
<td>RCU_INT_LXTALCKM</td>
<td>LXTAL clock stuck interrupt</td>
</tr>
</tbody>
</table>

**Enum rcu_osci_type_enum**

Table 3-522. Enum rcu_osci_type_enum

<table>
<thead>
<tr>
<th>enum name</th>
<th>Function description</th>
</tr>
</thead>
<tbody>
<tr>
<td>RCU_HXTAL</td>
<td>HXTAL</td>
</tr>
<tr>
<td>RCU_LXTAL</td>
<td>LXTAL</td>
</tr>
<tr>
<td>RCU_IRC16M</td>
<td>IRC16M</td>
</tr>
<tr>
<td>RCU_IRC48M</td>
<td>IRC48M</td>
</tr>
<tr>
<td>RCU_IRC32K</td>
<td>IRC32K</td>
</tr>
<tr>
<td>RCU_PLL_CK</td>
<td>PLL</td>
</tr>
</tbody>
</table>

**Enum rcu_clock_freq_enum**

Table 3-523. Enum rcu_clock_freq_enum

<table>
<thead>
<tr>
<th>enum name</th>
<th>Function description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CK_SYS</td>
<td>system clock</td>
</tr>
<tr>
<td>CK_AHB</td>
<td>AHB clock</td>
</tr>
<tr>
<td>CK_APB1</td>
<td>APB1 clock</td>
</tr>
<tr>
<td>CK_APB2</td>
<td>APB2 clock</td>
</tr>
<tr>
<td>CK_ADC</td>
<td>ADC clock</td>
</tr>
<tr>
<td>CK_USART0</td>
<td>USART0 clock</td>
</tr>
<tr>
<td>CK_I2C0</td>
<td>I2C0 clock</td>
</tr>
<tr>
<td>CK_I2C1</td>
<td>I2C1 clock</td>
</tr>
<tr>
<td>CK_I2C2</td>
<td>I2C2 clock</td>
</tr>
<tr>
<td>CK_LPUART</td>
<td>LPUART clock</td>
</tr>
<tr>
<td>CK_USART1</td>
<td>USART1 clock</td>
</tr>
<tr>
<td>CK_LPTIMER</td>
<td>LPTIMER clock</td>
</tr>
</tbody>
</table>
Enum `usart_idx_enum`

Table 3-524. Enum `usart_idx_enum`

<table>
<thead>
<tr>
<th>enum name</th>
<th>Function description</th>
</tr>
</thead>
<tbody>
<tr>
<td>IDX_USART0</td>
<td>idnex of USART0</td>
</tr>
<tr>
<td>IDX_USART1</td>
<td>idnex of USART1</td>
</tr>
</tbody>
</table>

Enum `i2c_idx_enum`

Table 3-525. Enum `i2c_idx_enum`

<table>
<thead>
<tr>
<th>enum name</th>
<th>Function description</th>
</tr>
</thead>
<tbody>
<tr>
<td>IDX_I2C0</td>
<td>idnex of I2C0</td>
</tr>
<tr>
<td>IDX_I2C1</td>
<td>idnex of I2C1</td>
</tr>
<tr>
<td>IDX_I2C2</td>
<td>idnex of I2C2</td>
</tr>
</tbody>
</table>

`rcu_deinit`

The description of `rcu_deinit` is shown as below:

Table 3-526. Function `rcu_deinit`

<table>
<thead>
<tr>
<th>Function name</th>
<th>Function description</th>
</tr>
</thead>
<tbody>
<tr>
<td>rcu_deinit</td>
<td>deinitialize the RCU, reset the value of all RCU registers into initial values</td>
</tr>
</tbody>
</table>

Function prototype: `void rcu_deinit(void);`

Precondition: -

The called functions: -

Input parameter (in): -

Output parameter (out): -

Return value: -

Example:

```c
/* reset RCU */
rcu_deinit();
```

`rcu_periph_clock_enable`

The description of `rcu_periph_clock_enable` is shown as below:

Table 3-527. Function `rcu_periph_clock_enable`

<table>
<thead>
<tr>
<th>Function name</th>
<th>Function description</th>
</tr>
</thead>
<tbody>
<tr>
<td>rcu_periph_clock_enable</td>
<td>enable the peripherals clock</td>
</tr>
</tbody>
</table>

Function prototype: `void rcu_periph_clock_enable(rcu_periph_enum periph);`

Precondition: -
The called functions

<table>
<thead>
<tr>
<th>periph</th>
<th>Input parameter(in)</th>
</tr>
</thead>
<tbody>
<tr>
<td>RCU_GPIOx</td>
<td>RCU peripherals, refer to Table 3-515, Enum rcu_periph_enum</td>
</tr>
<tr>
<td>RCU_DMA</td>
<td>GPIO ports clock (x=A,B,C,D,F)</td>
</tr>
<tr>
<td>RCU_CAU</td>
<td>DMA clock</td>
</tr>
<tr>
<td>RCU_TRNG</td>
<td>CAU clock</td>
</tr>
<tr>
<td>RCU_CRC</td>
<td>TRNG clock</td>
</tr>
<tr>
<td>RCU_CMP</td>
<td>CRC clock</td>
</tr>
<tr>
<td>RCU_SYSCFG</td>
<td>CMP clock</td>
</tr>
<tr>
<td>RCU_ADC</td>
<td>SYSCFG clock</td>
</tr>
<tr>
<td>RCU_TIMERx</td>
<td>ADC clock</td>
</tr>
<tr>
<td>RCU_LPTIMER</td>
<td>TIMERx clock (x=1,2,5,6,8,11)</td>
</tr>
<tr>
<td>RCU_SPIx</td>
<td>LPTIMER clock</td>
</tr>
<tr>
<td>RCU_USARTx</td>
<td>SPIx clock (x=0,1)</td>
</tr>
<tr>
<td>RCU_WWDGT</td>
<td>USARTx clock (x=0,1)</td>
</tr>
<tr>
<td>RCU_I2Cx</td>
<td>WWDGT clock</td>
</tr>
<tr>
<td>RCU_PMU</td>
<td>I2Cx clock (x=0,1,2)</td>
</tr>
<tr>
<td>RCU_RTC</td>
<td>PMU clock</td>
</tr>
<tr>
<td>RCU_DBGMCU</td>
<td>RTC clock</td>
</tr>
<tr>
<td>RCU_CAU</td>
<td>DBGMCU clock</td>
</tr>
<tr>
<td>RCU_DAC</td>
<td>CAU clock</td>
</tr>
<tr>
<td>RCU_CTC</td>
<td>DAC clock</td>
</tr>
<tr>
<td>RCU_BKP</td>
<td>CTC clock</td>
</tr>
<tr>
<td>RCU_USBD</td>
<td>BKP clock</td>
</tr>
<tr>
<td>RCU_SLCD</td>
<td>USBD clock</td>
</tr>
<tr>
<td>RCU_SLCD</td>
<td>SLCD clock</td>
</tr>
</tbody>
</table>

Output parameter(out)

|                        |                        |

Return value

Example:

/* enable the USART0 clock */
rcu_periph_clock_enable(RCU_USART0);

rcu_periph_clock_disable

The description of rcu_periph_clock_disable is shown as below:

Table 3-528. Function rcu_periph_clock_disable

<table>
<thead>
<tr>
<th>Function name</th>
<th>rcu_periph_clock_disable</th>
</tr>
</thead>
<tbody>
<tr>
<td>Function prototype</td>
<td>void rcu_periph_clock_disable(rcu_periph_enum periph);</td>
</tr>
<tr>
<td>Function descriptions</td>
<td>disable the peripherals clock</td>
</tr>
</tbody>
</table>
### Precondition
- 

### The called functions
- 

#### Input parameter(in)

<table>
<thead>
<tr>
<th>periph</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>RCU_GPIOx</td>
<td>GPIO ports clock (x=A,B,C,D,F)</td>
</tr>
<tr>
<td>RCU_DMA</td>
<td>DMA clock</td>
</tr>
<tr>
<td>RCU_CAU</td>
<td>CAU clock</td>
</tr>
<tr>
<td>RCU_TRNG</td>
<td>TRNG clock</td>
</tr>
<tr>
<td>RCU_CRC</td>
<td>CRC clock</td>
</tr>
<tr>
<td>RCU_CMP</td>
<td>CMP clock</td>
</tr>
<tr>
<td>RCU_SYSCFG</td>
<td>SYSCFG clock</td>
</tr>
<tr>
<td>RCU_ADC</td>
<td>ADC clock</td>
</tr>
<tr>
<td>RCU_TIMERx</td>
<td>TIMERx clock (x=1,2,5,6,8,11)</td>
</tr>
<tr>
<td>RCU_LPTIMER</td>
<td>LPTIMER clock</td>
</tr>
<tr>
<td>RCU_SPIx</td>
<td>SPIx clock (x=0,1)</td>
</tr>
<tr>
<td>RCU_USARTx</td>
<td>USARTx clock (x=0,1)</td>
</tr>
<tr>
<td>RCU_WWDGT</td>
<td>WWDGT clock</td>
</tr>
<tr>
<td>RCU_I2Cx</td>
<td>I2Cx clock (x=0,1,2)</td>
</tr>
<tr>
<td>RCU_PMU</td>
<td>PMU clock</td>
</tr>
<tr>
<td>RCU_RTC</td>
<td>RTC clock</td>
</tr>
<tr>
<td>RCU_DBGMCU</td>
<td>DBGMCU clock</td>
</tr>
<tr>
<td>RCU_CAU</td>
<td>CAU clock</td>
</tr>
<tr>
<td>RCU_DAC</td>
<td>DAC clock</td>
</tr>
<tr>
<td>RCU_CTC</td>
<td>CTC clock</td>
</tr>
<tr>
<td>RCU_BKP</td>
<td>BKP clock</td>
</tr>
<tr>
<td>RCU_USBD</td>
<td>USBD clock</td>
</tr>
<tr>
<td>RCU_SLCD</td>
<td>SLCD clock</td>
</tr>
</tbody>
</table>

#### Output parameter(out)
- 

#### Return value
- 

---

**Example:**

```c
/* disable the USART0 clock */
rcu_periph_clock_disable(RCU_USART0);
```

### rcu_periph_clock_sleep_enable

The description of `rcu_periph_clock_sleep_enable` is shown as below:

#### Table 3-529. Function `rcu_periph_clock_sleep_enable`

<table>
<thead>
<tr>
<th>Function name</th>
<th>rcu_periph_clock_sleep_enable</th>
</tr>
</thead>
<tbody>
<tr>
<td>Function prototype</td>
<td>void rcu_periph_clock_sleep_enable(rcu_periph_sleep_enum periph);</td>
</tr>
</tbody>
</table>
enable the peripherals clock when in sleep mode

Precondition

The called functions

Input parameter(in)

<table>
<thead>
<tr>
<th>periph</th>
<th>RCU peripherals, refer to Table 3-516. Enum rcu_periph_sleep_enum</th>
</tr>
</thead>
<tbody>
<tr>
<td>RCU_FMC_SLP</td>
<td>FMC clock</td>
</tr>
<tr>
<td>RCU_SRAM0_SLP</td>
<td>SRAM0 clock</td>
</tr>
<tr>
<td>RCU_SRAM1_SLP</td>
<td>SRAM1 clock</td>
</tr>
</tbody>
</table>

Output parameter(out)

Return value

Example:

/* enable the FMC clock when in sleep mode */
rcu_periph_clock_sleep_enable(RCU_FMC_SLP);

rcu_periph_clock_sleep_disable

The description of rcu_periph_clock_sleep_disable is shown as below:

Table 3-530. Function rcu_periph_clock_sleep_disable

<table>
<thead>
<tr>
<th>Function name</th>
<th>rcu_periph_clock_sleep_disable</th>
</tr>
</thead>
<tbody>
<tr>
<td>Function prototype</td>
<td>void rcu_periph_clock_sleep_disable(rcu_periph_sleep_enum periph);</td>
</tr>
<tr>
<td>Function descriptions</td>
<td>disable the peripherals clock when in sleep mode</td>
</tr>
<tr>
<td>Precondition</td>
<td>-</td>
</tr>
<tr>
<td>The called functions</td>
<td>-</td>
</tr>
</tbody>
</table>

Input parameter(in)

<table>
<thead>
<tr>
<th>periph</th>
<th>RCU peripherals, refer to Table 3-516. Enum rcu_periph_sleep_enum</th>
</tr>
</thead>
<tbody>
<tr>
<td>RCU_FMC_SLP</td>
<td>FMC clock</td>
</tr>
<tr>
<td>RCU_SRAM0_SLP</td>
<td>SRAM0 clock</td>
</tr>
<tr>
<td>RCU_SRAM1_SLP</td>
<td>SRAM1 clock</td>
</tr>
</tbody>
</table>

Output parameter(out)

Return value

Example:

/* disable the FMC clock when in sleep mode */
rcu_periph_clock_sleep_disable(RCU_FMC_SLP);
**rcu_periph_reset_enable**

The description of `rcu_periph_reset_enable` is shown as below:

<table>
<thead>
<tr>
<th>Function name</th>
<th><code>rcu_periph_reset_enable</code></th>
</tr>
</thead>
<tbody>
<tr>
<td>Function prototype</td>
<td><code>void rcu_periph_reset_enable(rcu_periph_reset_enum periph_reset);</code></td>
</tr>
<tr>
<td>Function descriptions</td>
<td>enable the peripherals reset</td>
</tr>
<tr>
<td>Precondition</td>
<td>-</td>
</tr>
<tr>
<td>The called functions</td>
<td>-</td>
</tr>
</tbody>
</table>

**Input parameter(in)**

<table>
<thead>
<tr>
<th>periph_reset</th>
<th>RCU peripherals reset, refer to Table 3-517, Enum <code>rcu_periph_reset_enum</code></th>
</tr>
</thead>
<tbody>
<tr>
<td><code>RCU_GPIOxRST</code></td>
<td>reset GPIO ports clock (x=A,B,C,D,F)</td>
</tr>
<tr>
<td><code>RCU_CAU</code></td>
<td>reset CAU clock</td>
</tr>
<tr>
<td><code>RCU_CRC</code></td>
<td>reset CRC clock</td>
</tr>
<tr>
<td><code>RCU_TRNG</code></td>
<td>reset TRNG clock</td>
</tr>
<tr>
<td><code>RCU_CMPRST</code></td>
<td>reset CMP clock</td>
</tr>
<tr>
<td><code>RCU_SYSCFG_RST</code></td>
<td>reset SYSCFG clock</td>
</tr>
<tr>
<td><code>RCU_ADC_RST</code></td>
<td>reset ADC clock</td>
</tr>
<tr>
<td><code>RCU_TIMERxRST</code></td>
<td>reset TIMERx clock (x=1,2,5,6,8,11)</td>
</tr>
<tr>
<td><code>RCU_SPIxRST</code></td>
<td>reset SPIx clock (x=0,1)</td>
</tr>
<tr>
<td><code>RCU_USARTxRST</code></td>
<td>reset USARTx clock (x=0,1)</td>
</tr>
<tr>
<td><code>RCU_LPTIMERRST</code></td>
<td>reset LPTIMER clock</td>
</tr>
<tr>
<td><code>RCU_SLCDERRST</code></td>
<td>reset SLCD clock</td>
</tr>
<tr>
<td><code>RCU_WWDGTRST</code></td>
<td>reset WWDGT clock</td>
</tr>
<tr>
<td><code>RCU_LPUARTRST</code></td>
<td>reset LPUART clock</td>
</tr>
<tr>
<td><code>RCU_UARTxRST</code></td>
<td>reset UARTx clock (x=3,4)</td>
</tr>
<tr>
<td><code>RCU_I2CxRST</code></td>
<td>reset I2Cx clock (x=0,1)</td>
</tr>
<tr>
<td><code>RCU_USBDRST</code></td>
<td>reset USBD clock</td>
</tr>
<tr>
<td><code>RCU_PMU_RST</code></td>
<td>reset PMU clock</td>
</tr>
<tr>
<td><code>RCU_DAC_RST</code></td>
<td>reset DAC clock</td>
</tr>
<tr>
<td><code>RCU_CTC_RST</code></td>
<td>reset CTC clock</td>
</tr>
</tbody>
</table>

**Output parameter(out)**

- |

**Return value**

- |

Example:

```
/* enable SPI0 reset */
rcu_periph_reset_enable(RCU_SPI0RST);
```
The description of rcu_periph_reset_disable is shown as below:

### Table 3-532. Function rcu_periph_reset_disable

<table>
<thead>
<tr>
<th>Function name</th>
<th>rcu_periph_reset_disable</th>
</tr>
</thead>
<tbody>
<tr>
<td>Function prototype</td>
<td>void rcu_periph_reset_disable(rcu_periph_reset_enum periph_reset);</td>
</tr>
<tr>
<td>Function description</td>
<td>disable the peripheral reset</td>
</tr>
<tr>
<td>Precondition</td>
<td>-</td>
</tr>
<tr>
<td>The called functions</td>
<td>-</td>
</tr>
</tbody>
</table>

#### Input parameter (in)

<table>
<thead>
<tr>
<th>periph_reset</th>
<th>RCU peripherals reset, refer to <a href="https://example.com">Table 3-517, Enum</a></th>
</tr>
</thead>
<tbody>
<tr>
<td>RCU_GPIOxRST</td>
<td>reset GPIO ports clock (x=A,B,C,D,F)</td>
</tr>
<tr>
<td>RCU_CAU</td>
<td>reset CAU clock</td>
</tr>
<tr>
<td>RCU_CRC</td>
<td>reset CRC clock</td>
</tr>
<tr>
<td>RCU_TRNG</td>
<td>reset TRNG clock</td>
</tr>
<tr>
<td>RCU_CMPRST</td>
<td>reset CMP clock</td>
</tr>
<tr>
<td>RCU_SYSCFGRST</td>
<td>reset SYSCFG clock</td>
</tr>
<tr>
<td>RCU_ADCRST</td>
<td>reset ADC clock</td>
</tr>
<tr>
<td>RCU_TIMERxRST</td>
<td>reset TIMERx clock (x=1,2,5,6,8,11)</td>
</tr>
<tr>
<td>RCU_SPIxRST</td>
<td>reset SPIx clock (x=0,1)</td>
</tr>
<tr>
<td>RCU_USARTxRST</td>
<td>reset USARTx clock (x=0,1)</td>
</tr>
<tr>
<td>RCU_LPTIMERRST</td>
<td>reset LPTIMER clock</td>
</tr>
<tr>
<td>RCU_SLCDRRST</td>
<td>reset SLCD clock</td>
</tr>
<tr>
<td>RCU_WWDGTRST</td>
<td>reset WWDGT clock</td>
</tr>
<tr>
<td>RCU_LPUARTRST</td>
<td>reset LPUART clock</td>
</tr>
<tr>
<td>RCU_UARTxRST</td>
<td>reset UARTx clock (x=3,4)</td>
</tr>
<tr>
<td>RCU_I2CxRST</td>
<td>reset I2Cx clock (x=0,1)</td>
</tr>
<tr>
<td>RCU_USBDST</td>
<td>reset USB clock</td>
</tr>
<tr>
<td>RCU_PMURST</td>
<td>reset PMU clock</td>
</tr>
<tr>
<td>RCU_DACRST</td>
<td>reset DAC clock</td>
</tr>
<tr>
<td>RCU_CTCRST</td>
<td>reset CTC clock</td>
</tr>
</tbody>
</table>

#### Output parameter (out)

| - | - |

#### Return value

| - | - |

Example:

```c
/* disable SPI0 reset */
rcu_periph_reset_disable(RCU_SPI0RST);
```
rcu_bkp_reset_enable

The description of rcu_bkp_reset_enable is shown as below:

Table 3-533. Function rcu_bkp_reset_enable

<table>
<thead>
<tr>
<th>Function name</th>
<th>rcu_bkp_reset_enable</th>
</tr>
</thead>
<tbody>
<tr>
<td>Function prototype</td>
<td>void rcu_bkp_reset_enable(void);</td>
</tr>
<tr>
<td>Function descriptions</td>
<td>enable the BKP domain reset</td>
</tr>
<tr>
<td>Precondition</td>
<td>-</td>
</tr>
<tr>
<td>The called functions</td>
<td>-</td>
</tr>
<tr>
<td>Input parameter(in)</td>
<td>-</td>
</tr>
<tr>
<td>Output parameter(out)</td>
<td>-</td>
</tr>
<tr>
<td>Return value</td>
<td>-</td>
</tr>
</tbody>
</table>

Example:

/* reset the BKP domain */
rcu_bkp_reset_enable();

rcu_bkp_reset_disable

The description of rcu_bkp_reset_disable is shown as below:

Table 3-534. Function rcu_bkp_reset_disable

<table>
<thead>
<tr>
<th>Function name</th>
<th>rcu_bkp_reset_disable</th>
</tr>
</thead>
<tbody>
<tr>
<td>Function prototype</td>
<td>void rcu_bkp_reset_disable(void);</td>
</tr>
<tr>
<td>Function descriptions</td>
<td>disable the BKP domain reset</td>
</tr>
<tr>
<td>Precondition</td>
<td>-</td>
</tr>
<tr>
<td>The called functions</td>
<td>-</td>
</tr>
<tr>
<td>Input parameter(in)</td>
<td>-</td>
</tr>
<tr>
<td>Output parameter(out)</td>
<td>-</td>
</tr>
<tr>
<td>Return value</td>
<td>-</td>
</tr>
</tbody>
</table>

Example:

/* disable the BKP domain reset */
rcu_bkp_reset_disable();
rcu_system_clock_source_config

The description of rcu_system_clock_source_config is shown as below:

Table 3-535. Function rcu_system_clock_source_config

<table>
<thead>
<tr>
<th>Function name</th>
<th>rcu_system_clock_source_config</th>
</tr>
</thead>
<tbody>
<tr>
<td>Function prototype</td>
<td>void rcu_system_clock_source_config(uint32_t ck_sys);</td>
</tr>
<tr>
<td>Function descriptions</td>
<td>configure the system clock source</td>
</tr>
<tr>
<td>Precondition</td>
<td>-</td>
</tr>
<tr>
<td>The called functions</td>
<td>-</td>
</tr>
</tbody>
</table>

Input parameter(in)

<table>
<thead>
<tr>
<th>ck_sys</th>
<th>system clock source select</th>
</tr>
</thead>
<tbody>
<tr>
<td>RCU_CKSYSSRC_IRC 16M</td>
<td>select CK_IRC16M as the CK_SYS source</td>
</tr>
<tr>
<td>RCU_CKSYSSRC_HXTAL</td>
<td>select CK_HXTAL as the CK_SYS source</td>
</tr>
<tr>
<td>RCU_CKSYSSRC_PLL</td>
<td>select CK_PLL as the CK_SYS source</td>
</tr>
<tr>
<td>RCU_CKSYSSRC_IRC 48M</td>
<td>select CK_IRC48M as the CK_SYS source</td>
</tr>
</tbody>
</table>

Output parameter(out)

- |

Return value

- |

Example:

```c
/* configure the CK_HXTAL as the CK_SYS source */
rcu_system_clock_source_config(RCU_CKSYSSRC_HXTAL);
```

rcu_system_clock_source_get

The description of rcu_system_clock_source_get is shown as below:

Table 3-536. Function rcu_system_clock_source_get

<table>
<thead>
<tr>
<th>Function name</th>
<th>rcu_system_clock_source_get</th>
</tr>
</thead>
<tbody>
<tr>
<td>Function prototype</td>
<td>uint32_t rcu_system_clock_source_get(void);</td>
</tr>
<tr>
<td>Function descriptions</td>
<td>get the system clock source</td>
</tr>
<tr>
<td>Precondition</td>
<td>-</td>
</tr>
<tr>
<td>The called functions</td>
<td>-</td>
</tr>
</tbody>
</table>

Input parameter(in)

- |

Output parameter(out)

- |

Return value

uint32_t RCU_SCSS_IRC16M/RCU_SCSS_HXTAL/RCU_SCSS_PLL/RCU_SCSS_
Example:

```c
uint32_t temp_cksys_status;

/* get the CK_SYS source */
temp_cksys_status = rcu_system_clock_source_get();
```

**rcu_ahb_clock_config**

The description of rcu_ahb_clock_config is shown as below:

<table>
<thead>
<tr>
<th>Function name</th>
<th>rcu_ahb_clock_config</th>
</tr>
</thead>
<tbody>
<tr>
<td>Function prototype</td>
<td>void rcu_ahb_clock_config(uint32_t ck_AHB);</td>
</tr>
<tr>
<td>Function descriptions</td>
<td>configure the AHB clock prescaler selection</td>
</tr>
<tr>
<td>Precondition</td>
<td>-</td>
</tr>
<tr>
<td>The called functions</td>
<td>-</td>
</tr>
</tbody>
</table>

**Input parameter(in)**

<table>
<thead>
<tr>
<th>ck_AHB</th>
<th>AHB clock prescaler selection</th>
</tr>
</thead>
<tbody>
<tr>
<td>RCU_AHB_CKSYS_DIVx</td>
<td>select CK_SYS / x, (x=1, 2, 4, 8, 16, 64, 128, 256, 512)</td>
</tr>
</tbody>
</table>

**Output parameter(out)**

| -                          | - |

**Return value**

| -                          | - |

Example:

```c
/* configure CK_SYS/128 */
rcu_ahb_clock_config(RCU_AHB_CKSYS_DIV128);
```

**rcu_apb1_clock_config**

The description of rcu_apb1_clock_config is shown as below:

<table>
<thead>
<tr>
<th>Function name</th>
<th>rcu_apb1_clock_config</th>
</tr>
</thead>
<tbody>
<tr>
<td>Function prototype</td>
<td>void rcu_apb1_clock_config(uint32_t ck_APB1);</td>
</tr>
<tr>
<td>Function descriptions</td>
<td>configure the APB1 clock prescaler selection</td>
</tr>
<tr>
<td>Precondition</td>
<td>-</td>
</tr>
<tr>
<td>The called functions</td>
<td>-</td>
</tr>
</tbody>
</table>

**Input parameter(in)**

<table>
<thead>
<tr>
<th>ck_APB1</th>
<th>APB1 clock prescaler selection</th>
</tr>
</thead>
<tbody>
<tr>
<td>RCU_APB1_CKAHB_DIVx</td>
<td>select (CK_AHB / x) as CK_APB1 (x=1,2,4,8,16)</td>
</tr>
</tbody>
</table>
/* configure CK_AHB/16 as CK_APB1 */
rcu_apb1_clock_config(RCU_APB1_CKAHB_DIV16);

rcu_apb2_clock_config

The description of rcu_apb2_clock_config is shown as below:

Table 3-539. Function rcu_apb2_clock_config

<table>
<thead>
<tr>
<th>Function name</th>
<th>rcu_apb2_clock_config</th>
</tr>
</thead>
<tbody>
<tr>
<td>Function prototype</td>
<td>void rcu_apb2_clock_config(uint32_t ck_apb2);</td>
</tr>
<tr>
<td>Function descriptions</td>
<td>configure the APB2 clock prescaler selection</td>
</tr>
<tr>
<td>Precondition</td>
<td>-</td>
</tr>
<tr>
<td>The called functions</td>
<td>-</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Input parameter(in)</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>ck_apb2</td>
<td>APB2 clock prescaler selection</td>
</tr>
<tr>
<td>RCU_APB2_CKAHB_DIVx</td>
<td>select (CK_AHB / x) as CK_APB2 clock (x=1,2,4,8,16)</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Output parameter(out)</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>-</td>
<td>-</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Return value</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>-</td>
<td>-</td>
</tr>
</tbody>
</table>

Example:

/* configure CK_AHB/8 as CK_APB2 */
rcu_apb2_clock_config(RCU_APB2_CKAHB_DIV8);

rcu_adc_clock_config

The description of rcu_adc_clock_config is shown as below:

Table 3-540. Function rcu_adc_clock_config

<table>
<thead>
<tr>
<th>Function name</th>
<th>rcu_adc_clock_config</th>
</tr>
</thead>
<tbody>
<tr>
<td>Function prototype</td>
<td>void rcu_adc_clock_config(rcu_adc_clock_enum ck_adc);</td>
</tr>
<tr>
<td>Function descriptions</td>
<td>configure the ADC clock prescaler selection</td>
</tr>
<tr>
<td>Precondition</td>
<td>-</td>
</tr>
<tr>
<td>The called functions</td>
<td>-</td>
</tr>
</tbody>
</table>
### Input parameter(in)

<table>
<thead>
<tr>
<th>ck_adc</th>
<th>ADC clock prescaler selection</th>
</tr>
</thead>
<tbody>
<tr>
<td>RCU_ADCCK_IRC16M</td>
<td>select CK_IRC16M as CK_ADC</td>
</tr>
<tr>
<td>RCU_ADCCK_APB2_DIV2</td>
<td>select CK_APB2/2 as CK_ADC</td>
</tr>
<tr>
<td>RCU_ADCCK_APB2_DIV4</td>
<td>select CK_APB2/4 as CK_ADC</td>
</tr>
<tr>
<td>RCU_ADCCK_APB2_DIV6</td>
<td>select CK_APB2/6 as CK_ADC</td>
</tr>
<tr>
<td>RCU_ADCCK_APB2_DIV8</td>
<td>select CK_APB2/8 as CK_ADC</td>
</tr>
<tr>
<td>RCU_ADCCK_APB2_DIV10</td>
<td>select CK_APB2/10 as CK_ADC</td>
</tr>
<tr>
<td>RCU_ADCCK_APB2_DIV12</td>
<td>select CK_APB2/12 as CK_ADC</td>
</tr>
<tr>
<td>RCU_ADCCK_APB2_DIV14</td>
<td>select CK_APB2/14 as CK_ADC</td>
</tr>
<tr>
<td>RCU_ADCCK_APB2_DIV16</td>
<td>select CK_APB2/16 as CK_ADC</td>
</tr>
<tr>
<td>RCU_ADCCK_AHB_DIV3</td>
<td>select CK_AHB/3 as CK_ADC</td>
</tr>
<tr>
<td>RCU_ADCCK_AHB_DIV5</td>
<td>select CK_AHB/5 as CK_ADC</td>
</tr>
<tr>
<td>RCU_ADCCK_AHB_DIV7</td>
<td>select CK_AHB/7 as CK_ADC</td>
</tr>
<tr>
<td>RCU_ADCCK_AHB_DIV9</td>
<td>select CK_AHB/9 as CK_ADC</td>
</tr>
<tr>
<td>RCU_ADCCK_AHB_DIV11</td>
<td>select CK_AHB/11 as CK_ADC</td>
</tr>
<tr>
<td>RCU_ADCCK_AHB_DIV13</td>
<td>select CK_AHB/13 as CK_ADC</td>
</tr>
<tr>
<td>RCU_ADCCK_AHB_DIV15</td>
<td>select CK_AHB/15 as CK_ADC</td>
</tr>
</tbody>
</table>

### Output parameter(out)

- -

### Return value

- -

Example:

/* configure the ADC prescaler factor */
rcu_adc_clock_config(RCU_ADCCK_APB2_DIV2);
rcu_ckout_config

The description of rcu_ckout_config is shown as below:

Table 3-541. Function rcu_ckout_config

<table>
<thead>
<tr>
<th>Function name</th>
<th>rcu_ckout_config</th>
</tr>
</thead>
<tbody>
<tr>
<td>Function prototype</td>
<td>void rcu_ckout_config(uint32_t ckout_src, uint32_t ckout_div);</td>
</tr>
<tr>
<td>Function descriptions</td>
<td>configure the CK_OUT clock source and division factor</td>
</tr>
<tr>
<td>Precondition</td>
<td>-</td>
</tr>
<tr>
<td>The called functions</td>
<td>-</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Input parameter(in)</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>ckout_src</td>
<td>CK_OUT clock source selection</td>
</tr>
<tr>
<td>RCU_CKOUTSRC_NONE</td>
<td>no clock selected</td>
</tr>
<tr>
<td>RCU_CKOUTSRC_IRC 48M</td>
<td>select high speed 48M internal oscillator clock</td>
</tr>
<tr>
<td>RCU_CKOUTSRC_IRC 32K</td>
<td>select high speed 32K internal oscillator clock</td>
</tr>
<tr>
<td>RCU_CKOUTSRC_LX TAL</td>
<td>select LXTAL clock</td>
</tr>
<tr>
<td>RCU_CKOUTSRC_CK SYS</td>
<td>select system clock CK_SYS</td>
</tr>
<tr>
<td>RCU_CKOUTSRC_IRC 16M</td>
<td>select high speed 16M internal oscillator clock</td>
</tr>
<tr>
<td>RCU_CKOUTSRC_HX TAL</td>
<td>select HXTAL clock</td>
</tr>
<tr>
<td>RCU_CKOUTSRC_CK PLL_DIV1</td>
<td>select CK_PLL clock</td>
</tr>
<tr>
<td>RCU_CKOUTSRC_CK PLL_DIV2</td>
<td>Select (CK_PLL / 2) clock</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Input parameter(in)</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>ckout_div</td>
<td>CK_OUT divider</td>
</tr>
<tr>
<td>RCU_CKOUT_DIVx</td>
<td>CK_OUT is divided by x(x=1,2,4,8,16,32,64,128)</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Output parameter(out)</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>-</td>
<td>-</td>
</tr>
</tbody>
</table>

| Return value | - |

Example:

/* configure the HXTAL as CK_OUT clock source */

rcu_ckout_config(RCU_CKOUTSRC_HX TAL, RCU_CKOUT_DIV1);
rcu_pll_config

The description of rcu_pll_config is shown as below:

Table 3-542. Function rcu_pll_config

<table>
<thead>
<tr>
<th>Function name</th>
<th>rcu_pll_config</th>
</tr>
</thead>
<tbody>
<tr>
<td>Function prototype</td>
<td>void rcu_pll_config(uint32_t pll_src, uint32_t pll_mul);</td>
</tr>
<tr>
<td>Function descriptions</td>
<td>configure the main PLL clock</td>
</tr>
<tr>
<td>Precondition</td>
<td>-</td>
</tr>
<tr>
<td>The called functions</td>
<td>-</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Input parameter(in)</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>pll_src</td>
<td>PLL clock source selection</td>
</tr>
<tr>
<td>RCU_PLLSRC_IRC16M</td>
<td>IRC16M clock is selected as source clock of PLL</td>
</tr>
<tr>
<td>RCU_PLLSRC_HXTAL</td>
<td>HXTAL is selected as source clock of PLL</td>
</tr>
<tr>
<td>RCU_PLLSRC_IRC48M</td>
<td>select CK_IRC48M as PLL source clock</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Input parameter(in)</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>pll_mul</td>
<td>PLL clock multiplication factor</td>
</tr>
<tr>
<td>RCU_PLL_MULx</td>
<td>PLL source clock * x (x = 4..127)</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Output parameter(out)</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>-</td>
<td>-</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Return value</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>-</td>
<td>-</td>
</tr>
</tbody>
</table>

Example:

/* configure the PLL */
rcu_pll_config(RCU_PLLSRC_HXTAL, RCU_PLL_MUL10);

rcu_usart_clock_config

The description of rcu_usart_clock_config is shown as below:

Table 3-543. Function rcu_usart_clock_config

<table>
<thead>
<tr>
<th>Function name</th>
<th>rcu_usart_clock_config</th>
</tr>
</thead>
<tbody>
<tr>
<td>Function prototype</td>
<td>void rcu_usart_clock_config(uint32_t usart_idx, uint32_t ck_usart);</td>
</tr>
<tr>
<td>Function descriptions</td>
<td>configure the USARTx(x=0,1) clock source selection</td>
</tr>
<tr>
<td>Precondition</td>
<td>-</td>
</tr>
<tr>
<td>The called functions</td>
<td>-</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Input parameter(in)</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>usart_idx</td>
<td>idnex of USART, refer to Table 3-524, Enum usart_idx_enum</td>
</tr>
<tr>
<td>IDX_USARTx</td>
<td>idnex of USARTx(x=0,1)</td>
</tr>
<tr>
<td>ck_usart</td>
<td>USART clock source selection</td>
</tr>
<tr>
<td>RCU_USARTSRC_CK</td>
<td>CK_USART select CK_APB1/ CK_APB2</td>
</tr>
</tbody>
</table>
### Function `rcu_usart_clock_config`

Configure the USART clock source.

#### Parameters
- `IDX_USARTx`: USART index (x=0,1,2)
- `RCU_USARTSRC_CK SYS`: CK_USART select CK_SYS
- `RCU_USARTSRC_LXTAL`: CK_USART select CK_LXTAL
- `RCU_USARTSRC_IRC16M_DIV`: CK_USART select CK_IRC16MDIV

#### Output Parameters
- None

#### Return Value
- None

#### Example

```c
/* configure the LXTAL as USART0 clock */
rcu_usart_clock_config(IDX_USART0, RCU_USART0SRC_LXTAL);
```

### Function `rcu_i2c_clock_config`

Configure the I2Cx clock source.

#### Parameters
- `i2c_idx`: I2C index (x=0,1,2)
- `ck_i2c`: I2C clock source selection

#### Precondition
- None

#### The Called Functions
- None

#### Input Parameters
- `i2c_idx`
- `ck_i2c`

#### Output Parameters
- None

#### Return Value
- None

#### Example

```c
/* configure the CKAPB1 as I2C0 clock */
rcu_usart_clock_config(IDX_I2C0, RCU_I2CSRC_CKAPB1);
```
## rcu_lptimer_clock_config

The description of `rcu_lptimer_clock_config` is shown as below:

### Table 3-545. Function rcu_lptimer_clock_config

<table>
<thead>
<tr>
<th>Function name</th>
<th>rcu_lptimer_clock_config</th>
</tr>
</thead>
<tbody>
<tr>
<td>Function prototype</td>
<td>void rcu_lptimer_clock_config(uint32_t ck_lptimer);</td>
</tr>
<tr>
<td>Function descriptions</td>
<td>configure the LPTIMER clock source selection</td>
</tr>
<tr>
<td>Precondition</td>
<td>-</td>
</tr>
<tr>
<td>The called functions</td>
<td>-</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Input parameter(in)</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>ck_lptimer</td>
<td>LPTIMER clock source selection</td>
</tr>
<tr>
<td>RCU_LPTIMERSRC_C_KAPB1</td>
<td>CK_LPTIMER select CK_APB1</td>
</tr>
<tr>
<td>RCU_LPTIMERSRC_I_RC32K</td>
<td>CK_LPTIMER select CK_IRC32K</td>
</tr>
<tr>
<td>RCU_LPTIMERSRC_L_XTAL</td>
<td>CK_LPTIMER select CK_LXTAL</td>
</tr>
<tr>
<td>RCU_LPTIMERSRC_I_RC16MDIV</td>
<td>CK_LPTIMER select CK_IRC16MDIV</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Output parameter(out)</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>-</td>
<td>-</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Return value</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>-</td>
<td>-</td>
</tr>
</tbody>
</table>

Example:

```c
/* configure the LPTIMER clock source selection */
rcu_lptimer_clock_config(RCU_LPTIMERSRC_C_KAPB1);
```

## rcu_lpuart_clock_config

The description of `rcu_lpuart_clock_config` is shown as below:

### Table 3-546. Function rcu_lpuart_clock_config

<table>
<thead>
<tr>
<th>Function name</th>
<th>rcu_lpuart_clock_config</th>
</tr>
</thead>
<tbody>
<tr>
<td>Function prototype</td>
<td>void rcu_lpuart_clock_config(uint32_t ck_lpuart);</td>
</tr>
<tr>
<td>Function descriptions</td>
<td>configure the LPUART clock source selection</td>
</tr>
<tr>
<td>Precondition</td>
<td>-</td>
</tr>
<tr>
<td>The called functions</td>
<td>-</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Input parameter(in)</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>ck_lpuart</td>
<td>LPUART clock source selection</td>
</tr>
<tr>
<td>RCU_LPUARTSRC_C_KAPB1</td>
<td>LPUART select CK_APB1</td>
</tr>
<tr>
<td>RCU_LPUARTSRC_C</td>
<td>LPUART select CK_SYS</td>
</tr>
</tbody>
</table>
Example:

/* configure the lpuart clock source selection */
rcu_lpuart_clock_config(RCU_LPUARTSRC_CKAPB1);

rcu_irc16mdiv_clock_config

The description of rcu_irc16mdiv_clock_config is shown as below:

Table 3-547. Function rcu_irc16mdiv_clock_config

<table>
<thead>
<tr>
<th>Function name</th>
<th>Function prototype</th>
<th>Function descriptions</th>
</tr>
</thead>
<tbody>
<tr>
<td>rcu_irc16mdiv_clock_config</td>
<td>void rcu_irc16mdiv_clock_config(uint32_t ck_irc16mdiv);</td>
<td>configure the IRC16MDIV clock selection</td>
</tr>
</tbody>
</table>

Precondition -

The called functions -

Input parameter(in)

<table>
<thead>
<tr>
<th>ck_irc16mdiv</th>
<th>IRC16MDIV clock selection</th>
</tr>
</thead>
<tbody>
<tr>
<td>RCU_IRC16MDIV_NO_NE</td>
<td>CK_IRC16MDIV select CK_IRC16M</td>
</tr>
<tr>
<td>RCU_IRC16MDIV_2</td>
<td>CK_IRC16MDIV select CK_IRC16M divided by 2</td>
</tr>
<tr>
<td>RCU_IRC16MDIV_4</td>
<td>CK_IRC16MDIV select CK_IRC16M divided by 4</td>
</tr>
<tr>
<td>RCU_IRC16MDIV_8</td>
<td>CK_IRC16MDIV select CK_IRC16M divided by 8</td>
</tr>
<tr>
<td>RCU_IRC16MDIV_16</td>
<td>CK_IRC16MDIV select CK_IRC16M divided by 16</td>
</tr>
</tbody>
</table>

Output parameter(out)

- -

Return value - -

Example:

/* configure the IRC16MDIV clock selection */
rcu_irc16mdiv_clock_config(RCU_IRC16MDIV_2);
rcu_usbd_clock_config

The description of rcu_usbd_clock_config is shown as below:

Table 3-548. Function rcu_usbd_clock_config

<table>
<thead>
<tr>
<th>Function name</th>
<th>rcu_usbd_clock_config</th>
</tr>
</thead>
<tbody>
<tr>
<td>Function prototype</td>
<td>void rcu_usbd_clock_config(uint32_t ck_usbd);</td>
</tr>
<tr>
<td>Function descriptions</td>
<td>configure the USBD clock source selection</td>
</tr>
<tr>
<td>Precondition</td>
<td>-</td>
</tr>
<tr>
<td>The called functions</td>
<td>-</td>
</tr>
</tbody>
</table>

Input parameter(in)

- `ck_usbd` USBD clock source selection
  - `RCU_USBDSRC_IRC48M` select CK_IRC48M as USBD source clock
  - `RCU_USBDSRC_PLL` select CK_PLL as USBD source clock

Output parameter(out)

- -

Return value

- -

Example:

/* configure the USBD clock source selection */
rcu_usbd_clock_config(RCU_USBDSRC_IRC48M);

rcu_rtc_clock_config

The description of rcu_rtc_clock_config is shown as below:

Table 3-549. Function rcu_rtc_clock_config

<table>
<thead>
<tr>
<th>Function name</th>
<th>rcu_rtc_clock_config</th>
</tr>
</thead>
<tbody>
<tr>
<td>Function prototype</td>
<td>void rcu_rtc_clock_config(uint32_t rtc_clock_source);</td>
</tr>
<tr>
<td>Function descriptions</td>
<td>configure the RTC clock source selection</td>
</tr>
<tr>
<td>Precondition</td>
<td>-</td>
</tr>
<tr>
<td>The called functions</td>
<td>-</td>
</tr>
</tbody>
</table>

Input parameter(in)

- `rtc_clock_source` RTC clock source selection
  - `RCU_RTCSRC_NONE` no clock selected
  - `RCU_RTCSRC_LXTAL` select CK_LXTAL as RTC source clock
  - `RCU_RTCSRC_IRC40K` select CK_IRC40K as RTC source clock
  - `RCU_RTCSRC_HXTAL_DIV_32` select (CK_HXTAL / 32) as RTC source clock

Output parameter(out)

- -
Example:

/* configure the RTC clock source selection */
rcu_rtc_clock_config(RCU_RTC_SRC_IRC32K);

**rcu_pll_source_ck_prediv_config**

The description of `rcu_pll_source_ck_prediv_config` is shown as below:

<table>
<thead>
<tr>
<th>Function name</th>
<th>rcu_pll_source_ck_prediv_config</th>
</tr>
</thead>
<tbody>
<tr>
<td>Function prototype</td>
<td>void rcu_pll_source_ck_prediv_config(uint32_t pllsource_ck_prediv);</td>
</tr>
<tr>
<td>Function descriptions</td>
<td>configure PLL source clocks pre-divider</td>
</tr>
<tr>
<td>Precondition</td>
<td>-</td>
</tr>
<tr>
<td>The called functions</td>
<td>-</td>
</tr>
</tbody>
</table>

**Input parameter(in)**

- **pllsoure_ck_prediv**
  - PLL source clocks divider used as input of PLL
  - `RCU_PLL_PREDVx` PLL source clocks divided x used as input of PLL (x=1..16)

**Output parameter(out)**

- -

**Return value**

- -

Example:

/* configure PLL source clocks pre-divider */
rcu_pll_source_ck_prediv_config(RCU_PLL_PREDV2);

**rcu_lxtal_drive_capability_config**

The description of `rcu_lxtal_drive_capability_config` is shown as below:

<table>
<thead>
<tr>
<th>Function name</th>
<th>rcu_lxtal_drive_capability_config</th>
</tr>
</thead>
<tbody>
<tr>
<td>Function prototype</td>
<td>void rcu_lxtal_drive_capability_config(uint32_t lxtal_dricap);</td>
</tr>
<tr>
<td>Function descriptions</td>
<td>configure the LXTAL drive capability</td>
</tr>
<tr>
<td>Precondition</td>
<td>-</td>
</tr>
<tr>
<td>The called functions</td>
<td>-</td>
</tr>
</tbody>
</table>

**Input parameter(in)**

- **lxtal_dricap**
  - drive capability of LXTAL
  - `RCU_LXTAL_LOWDRI` lower driving capability
  - `RCU_LXTAL_MED_LO` medium low driving capability
WDRI

<table>
<thead>
<tr>
<th></th>
<th>GD32L23x Firmware Library User Guide</th>
</tr>
</thead>
<tbody>
<tr>
<td>RCU_LXTAL_MED_HI</td>
<td>medium high driving capability</td>
</tr>
<tr>
<td>GHDR</td>
<td></td>
</tr>
<tr>
<td>RCU_LXTAL_HIGHDRI</td>
<td>higher driving capability</td>
</tr>
</tbody>
</table>

Output parameter(out)
- |

Return value
- |

Example:

/* set the LXTAL lower driving capability */
rcu_lxtal_drive_capability_config(RCU_LXTAL_LOWDRI);

rcu_lp_ldo_config

The description of rcu_lp_ldo_config is shown as below:

Table 3-552. Function rcu_lp_ldo_config

<table>
<thead>
<tr>
<th>Function name</th>
<th>rcu_lp_ldo_config</th>
</tr>
</thead>
<tbody>
<tr>
<td>Function prototype</td>
<td>void rcu_lp_ldo_config(uint32_t lp_ldo_voltage);</td>
</tr>
<tr>
<td>Function descriptions</td>
<td>configure the low power mode LDO voltage selection</td>
</tr>
<tr>
<td>Precondition</td>
<td>-</td>
</tr>
<tr>
<td>The called functions</td>
<td>-</td>
</tr>
</tbody>
</table>

Input parameter(in)

<table>
<thead>
<tr>
<th>lp_ldo_voltage</th>
<th>low power mode LDO voltage</th>
</tr>
</thead>
<tbody>
<tr>
<td>RCU_LP_LDO_V_0_8</td>
<td>LP_LDO output voltage 0.8V</td>
</tr>
<tr>
<td>RCU_LP_LDO_V_0_9</td>
<td>LP_LDO output voltage 0.9V</td>
</tr>
</tbody>
</table>

Output parameter(out)
- |

Return value
- |

Example:

/* set the low power mode LDO voltage 0.8V */
rcu_lp_ldo_config(RCU_LP_LDO_V_0_8);

rcu_lp_bandgap_config

The description of rcu_lp_bandgap_config is shown as below:

Table 3-553. Function rcu_lp_bandgap_config

<table>
<thead>
<tr>
<th>Function name</th>
<th>rcu_lp_bandgap_config</th>
</tr>
</thead>
<tbody>
<tr>
<td>Function prototype</td>
<td>void rcu_lp_bandgap_config(uint32_t lp_bandgap_clock);</td>
</tr>
</tbody>
</table>
### Function descriptions
configure low power bandgap mode selection

---

### Precondition
- 

---

### The called functions
- 

---

### Input parameter(in)

<table>
<thead>
<tr>
<th>lp_bandgap_clock</th>
<th>low power bandgap clock</th>
</tr>
</thead>
<tbody>
<tr>
<td>RCU_LPBM_32CLK</td>
<td>The length of holding phase is 3.2ms, 32 clock cycles</td>
</tr>
<tr>
<td>RCU_LPBM_64CLK</td>
<td>The length of holding phase is 6.4ms, 64 clock cycles</td>
</tr>
<tr>
<td>RCU_LPBM_128CLK</td>
<td>The length of holding phase is 12.8ms, 128 clock cycles</td>
</tr>
<tr>
<td>RCU_LPBM_256CLK</td>
<td>The length of holding phase is 25.6ms, 256 clock cycles</td>
</tr>
<tr>
<td>RCU_LPBM_512CLK</td>
<td>The length of holding phase is 51.2ms, 512 clock cycles</td>
</tr>
<tr>
<td>RCU_LPBM_1024CLK</td>
<td>The length of holding phase is 102.4ms, 1024 clock cycles</td>
</tr>
<tr>
<td>RCU_LPBM_2048CLK</td>
<td>The length of holding phase is 204.8ms, 2048 clock cycles</td>
</tr>
</tbody>
</table>

---

### Output parameter(out)
- 

---

### Return value
- 

---

### Example:

```c
/* configure low power bandgap mode selection */
rcu_lp_bandgap_config(RCU_LPBM_64CLK);
```

### rcu_flag_get

The description of `rcu_flag_get` is shown as below:

#### Table 3-554. Function rcu_flag_get

<table>
<thead>
<tr>
<th>Function name</th>
<th>rcu_flag_get</th>
</tr>
</thead>
<tbody>
<tr>
<td>Function prototype</td>
<td>FlagStatus rcu_flag_get(rcu_flag_enum flag);</td>
</tr>
<tr>
<td>Function descriptions</td>
<td>get the clock stabilization and peripheral reset flags</td>
</tr>
<tr>
<td>Precondition</td>
<td>-</td>
</tr>
<tr>
<td>The called functions</td>
<td>-</td>
</tr>
</tbody>
</table>

---

### Input parameter(in)

<table>
<thead>
<tr>
<th>flag</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>RCU_FLAG_IRC32KSTB</td>
<td>IRC32K stabilization flag</td>
</tr>
<tr>
<td>RCU_FLAG_LXTALSTB</td>
<td>LXTAL stabilization flag</td>
</tr>
<tr>
<td>RCU_FLAG_IRC16MS</td>
<td>IRC16M stabilization flag</td>
</tr>
<tr>
<td>RCU_FLAG_HXTALSTB</td>
<td>HXTAL stabilization flag</td>
</tr>
<tr>
<td>RCU_FLAG_PLLSTB</td>
<td>PLL stabilization flag</td>
</tr>
</tbody>
</table>
### RCU_Flag_IRC48MS
- **RCU_FLAG_IRC48MS TB**
  - IRC48M stabilization flag

### RCU_Flag_V12RST
- **RCU_FLAG_V12RST**
  - V12 domain power reset flag

### RCU_Flag_EPRST
- **RCU_FLAG_EPRST**
  - external PIN reset flag

### RCU_Flag_PORRST
- **RCU_FLAG_PORRST**
  - power reset flag

### RCU_Flag_SWRST
- **RCU_FLAG_SWRST**
  - software reset flag

### RCU_Flag_FWDGTR
- **RCU_FLAG_FWDGTR ST**
  - free watchdog timer reset flag

### RCU_Flag_WWDGTR
- **RCU_FLAG_WWDGTR ST**
  - window watchdog timer reset flag

### RCU_Flag_LPRST
- **RCU_FLAG_LPRST**
  - low-power reset flag

---

**Output parameter (out)**
- -

**Return value**
- FlagStatus
  - SET or RESET

---

**Example:**

```c
/* get the clock stabilization flag */
if(RESET != rcu_flag_get(RCU_FLAG_LXTALSTB)) {
}
```

**rcu_all_reset_flag_clear**

The description of **rcu_all_reset_flag_clear** is shown as below:

**Table 3-555. Function rcu_all_reset_flag_clear**

<table>
<thead>
<tr>
<th>Function name</th>
<th>rcu_all_reset_flag_clear</th>
</tr>
</thead>
<tbody>
<tr>
<td>Function prototype</td>
<td>void rcu_all_reset_flag_clear(void);</td>
</tr>
<tr>
<td>Function descriptions</td>
<td>clear all the reset flag</td>
</tr>
<tr>
<td>Precondition</td>
<td>-</td>
</tr>
<tr>
<td>The called functions</td>
<td>-</td>
</tr>
</tbody>
</table>

**Input parameter (in)**
- -

**Output parameter (out)**
- -

**Return value**
- -

**Example:**

```c
/* clear all the reset flag */
rcu_all_reset_flag_clear();
```


rcu_interrupt_flag_get

The description of rcu_interrupt_flag_get is shown as below:

Table 3-556. Function rcu_interrupt_flag_get

<table>
<thead>
<tr>
<th>Function name</th>
<th>rcu_interrupt_flag_get</th>
</tr>
</thead>
<tbody>
<tr>
<td>Function prototype</td>
<td>FlagStatus rcu_interrupt_flag_get(rcu_int_flag_enum int_flag);</td>
</tr>
<tr>
<td>Function descriptions</td>
<td>get the clock stabilization interrupt and ckm flags</td>
</tr>
<tr>
<td>Precondition</td>
<td>-</td>
</tr>
<tr>
<td>The called functions</td>
<td>-</td>
</tr>
</tbody>
</table>

Input parameter(in)

<table>
<thead>
<tr>
<th>int_flag</th>
<th>interrupt and ckm flags, refer to Table 3-519. Enum rcu_int_flag_enum</th>
</tr>
</thead>
<tbody>
<tr>
<td>RCU_INT_FLAG_IRC32KSTB</td>
<td>IRC32K stabilization interrupt flag</td>
</tr>
<tr>
<td>RCU_INT_FLAG_LXTALSTB</td>
<td>LXTAL stabilization interrupt flag</td>
</tr>
<tr>
<td>RCU_INT_FLAG_IRC16MSTB</td>
<td>IRC16M stabilization interrupt flag</td>
</tr>
<tr>
<td>RCU_INT_FLAG_HXTALSTB</td>
<td>HXTAL stabilization interrupt flag</td>
</tr>
<tr>
<td>RCU_INT_FLAG_PLLSTB</td>
<td>PLL stabilization interrupt flag</td>
</tr>
<tr>
<td>RCU_INT_FLAG_IRC48MSTB</td>
<td>IRC48M stabilization interrupt flag</td>
</tr>
<tr>
<td>RCU_INT_FLAG_LXTALCKM</td>
<td>LXTAL clock stuck interrupt flag</td>
</tr>
<tr>
<td>RCU_INT_FLAG_CKM</td>
<td>HXTAL clock stuck interrupt flag</td>
</tr>
</tbody>
</table>

Output parameter(out)

- |

Return value

FlagStatus | SET or RESET |

Example:

/* get the clock stabilization interrupt flag */

if(SET == rcu_interrupt_flag_get(RCU_INT_FLAG_HXTALSTB)){
}

rcu_interrupt_flag_clear

The description of rcu_interrupt_flag_clear is shown as below:

Table 3-557. Function rcu_interrupt_flag_clear

<table>
<thead>
<tr>
<th>Function name</th>
<th>rcu_interrupt_flag_clear</th>
</tr>
</thead>
</table>
## Function prototype

```c
void rcu_interrupt_flag_clear(rcu_int_flag_clear_enum int_flag_clear);
```

## Function descriptions

Clear the interrupt flags

## Precondition

- 

## The called functions

- 

## Input parameter (in)

<table>
<thead>
<tr>
<th>int_flag_clear</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>RCU_INT_FLAG_IRC3 2KSTB_CLR</td>
<td>IRC32K</td>
<td>2K stability interrupt flag clear</td>
</tr>
<tr>
<td>RCU_INT_FLAG_LXTA LSTB_CLR</td>
<td>LXTAL</td>
<td>LXTAL stability interrupt flag clear</td>
</tr>
<tr>
<td>RCU_INT_FLAG_IRC1 6MSTB_CLR</td>
<td>IRC16M</td>
<td>16M stability interrupt flag clear</td>
</tr>
<tr>
<td>RCU_INT_FLAG_HXT ALSTB_CLR</td>
<td>HXTAL</td>
<td>HXTAL stability interrupt flag clear</td>
</tr>
<tr>
<td>RCU_INT_FLAG_PLLS TB_CLR</td>
<td>PLL</td>
<td>PLL stability interrupt flag clear</td>
</tr>
<tr>
<td>RCU_INT_FLAG_IRC4 8MSTB_CLR</td>
<td>IRC48M</td>
<td>48M stability interrupt flag clear</td>
</tr>
<tr>
<td>RCU_INT_FLAG_LXTA LCKM_CLR</td>
<td>LXTAL</td>
<td>LXTAL clock stuck interrupt flag clear</td>
</tr>
<tr>
<td>RCU_INT_FLAG_CKM _CLR</td>
<td>CKM</td>
<td>CKM clock stuck interrupt flag clear</td>
</tr>
</tbody>
</table>

## Output parameter (out)

- 

## Return value

- 

Example:

```c
/* clear the interrupt HXTAL stabilization interrupt flag */
rcu_interrupt_flag_clear(RCU_INT_FLAG_HXTALSTB_CLR);
```

### rcu_interrupt_enable

The description of `rcu_interrupt_enable` is shown as below:

#### Table 3-558. Function `rcu_interrupt_enable`

<table>
<thead>
<tr>
<th>Function name</th>
<th><code>rcu_interrupt_enable</code></th>
</tr>
</thead>
<tbody>
<tr>
<td>Function prototype</td>
<td><code>void rcu_interrupt_enable(rcu_int_enum stab_int);</code></td>
</tr>
<tr>
<td>Function descriptions</td>
<td>enable the stabilization interrupt</td>
</tr>
<tr>
<td>Precondition</td>
<td>-</td>
</tr>
<tr>
<td>The called functions</td>
<td>-</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Input parameter (in)</th>
<th></th>
</tr>
</thead>
</table>
stab_int | clock stabilization interrupt, refer to Table 3-521. Enum rcu_int_enum
---|---
RCU_INT_IRC32KSTB | IRC32K stabilization interrupt enable
RCU_INT_LXTALSTB | LXTAL stabilization interrupt enable
RCU_INT_IRC16MSTB | IRC16M stabilization interrupt enable
RCU_INT_HXTALSTB | HXTAL stabilization interrupt enable
RCU_INT_PLLSTB | PLL stabilization interrupt enable
RCU_INT_IRC48MSTB | IRC48M stabilization interrupt enable
RCU_INT_LXTALCKM | LXTAL clock stuck interrupt enable

Output parameter(out)
- -

Return value
- -

Example:

/* enable the HXTAL stabilization interrupt */
rcu_interrupt_enable(RCU_INT_HXTALSTB);

rcu_interrupt_disable

The description of rcu_interrupt_disable is shown as below:

Table 3-559. Function rcu_interrupt_disable

<table>
<thead>
<tr>
<th>Function name</th>
<th>rcu_interrupt_disable</th>
</tr>
</thead>
<tbody>
<tr>
<td>Function prototype</td>
<td>void rcu_interrupt_disable(rcu_int_enum stab_int);</td>
</tr>
<tr>
<td>Function descriptions</td>
<td>disable the stabilization interrupt</td>
</tr>
<tr>
<td>Precondition</td>
<td>-</td>
</tr>
<tr>
<td>The called functions</td>
<td>-</td>
</tr>
</tbody>
</table>

Input parameter(in)

<table>
<thead>
<tr>
<th>stab_int</th>
<th>clock stabilization interrupt, refer to Table 3-521. Enum rcu_int_enum</th>
</tr>
</thead>
<tbody>
<tr>
<td>RCU_INT_IRC32KSTB</td>
<td>IRC32K stabilization interrupt enable</td>
</tr>
<tr>
<td>RCU_INT_LXTALSTB</td>
<td>LXTAL stabilization interrupt enable</td>
</tr>
<tr>
<td>RCU_INT_IRC16MSTB</td>
<td>IRC16M stabilization interrupt enable</td>
</tr>
<tr>
<td>RCU_INT_HXTALSTB</td>
<td>HXTAL stabilization interrupt enable</td>
</tr>
<tr>
<td>RCU_INT_PLLSTB</td>
<td>PLL stabilization interrupt enable</td>
</tr>
<tr>
<td>RCU_INT_IRC48MSTB</td>
<td>IRC48M stabilization interrupt enable</td>
</tr>
<tr>
<td>RCU_INT_LXTALCKM</td>
<td>LXTAL clock stuck interrupt enable</td>
</tr>
</tbody>
</table>

Output parameter(out)
- -

Return value
- -

Example:

/* disable the HXTAL stabilization interrupt */
rcu_interrupt_disable(RCU_INT_HXTALSTB);

rcu_osci_stab_wait

The description of rcu_osci_stab_wait is shown as below:

Table 3-560. Function rcu_osci_stab_wait

<table>
<thead>
<tr>
<th>Function name</th>
<th>rcu_osci_stab_wait</th>
</tr>
</thead>
<tbody>
<tr>
<td>Function prototype</td>
<td>ErrStatus rcu_osci_stab_wait(rcu_osci_type_enum osci);</td>
</tr>
<tr>
<td>Function descriptions</td>
<td>wait for oscillator stabilization flags is SET or oscillator startup is timeout</td>
</tr>
<tr>
<td>Precondition</td>
<td>-</td>
</tr>
<tr>
<td>The called functions</td>
<td>-</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Input parameter(in)</th>
<th>osci</th>
</tr>
</thead>
<tbody>
<tr>
<td>RCU_HXTAL</td>
<td>high speed crystal oscillator(HXTAL)</td>
</tr>
<tr>
<td>RCU_LXTAL</td>
<td>low speed crystal oscillator(LXTAL)</td>
</tr>
<tr>
<td>RCU_IRC16M</td>
<td>internal 16M RC oscillators(IRC16M)</td>
</tr>
<tr>
<td>RCU_IRC48M</td>
<td>internal 48M RC oscillators(IRC48M)</td>
</tr>
<tr>
<td>RCU_IRC32K</td>
<td>internal 32K RC oscillator(IRC32K)</td>
</tr>
<tr>
<td>RCU_PLL_CK</td>
<td>phase locked loop(PLL)</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Output parameter(out)</th>
<th>-</th>
</tr>
</thead>
</table>

<table>
<thead>
<tr>
<th>Return value</th>
<th>ErrStatus</th>
</tr>
</thead>
<tbody>
<tr>
<td>SUCCESS or ERROR</td>
<td></td>
</tr>
</tbody>
</table>

Example:

/* wait for oscillator stabilization flag */

if(SUCCESS == rcu_osci_stab_wait(RCU_HXTAL)){
    
}

rcu_osci_on

The description of rcu_osci_on is shown as below:

Table 3-561. Function rcu_osci_on

<table>
<thead>
<tr>
<th>Function name</th>
<th>rcu_osci_on</th>
</tr>
</thead>
<tbody>
<tr>
<td>Function prototype</td>
<td>void rcu_osci_on(rcu_osci_type_enum osci);</td>
</tr>
<tr>
<td>Function descriptions</td>
<td>turn on the oscillator</td>
</tr>
<tr>
<td>Precondition</td>
<td>-</td>
</tr>
<tr>
<td>The called functions</td>
<td>-</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Input parameter(in)</th>
<th>osci</th>
</tr>
</thead>
<tbody>
<tr>
<td>RCU_HXTAL</td>
<td>high speed crystal oscillator(HXTAL)</td>
</tr>
</tbody>
</table>
RCU_LXTAL | low speed crystal oscillator (LXTAL)
RCU_IRC16M | internal 16M RC oscillators (IRC16M)
RCU_IRC48M | internal 48M RC oscillators (IRC48M)
RCU_IRC32K | internal 32K RC oscillator (IRC32K)
RCU_PLL_CK | phase locked loop (PLL)

Output parameter (out):
- -

Return value:
- -

Example:

/* turn on the high speed crystal oscillator */
rcu_osci_on(RCU_HXTAL);

rcu_osci_off

The description of rcu_osci_off is shown as below:

Table 3-562. Function rcu_osci_off

<table>
<thead>
<tr>
<th>Function name</th>
<th>rcu_osci_off</th>
</tr>
</thead>
</table>
| Function prototype | void rcu_osci_off(rcu_osci_type_enum osci);
| Function descriptions | turn off the oscillator |
| Precondition | - |
| The called functions | - |

Input parameter (in):

<table>
<thead>
<tr>
<th>osci</th>
<th>oscillator types, refer to Table 3-522, Enum rcu_osci_type_enum</th>
</tr>
</thead>
<tbody>
<tr>
<td>RCU_HXTAL</td>
<td>high speed crystal oscillator (HXTAL)</td>
</tr>
<tr>
<td>RCU_LXTAL</td>
<td>low speed crystal oscillator (LXTAL)</td>
</tr>
<tr>
<td>RCU_IRC16M</td>
<td>internal 16M RC oscillators (IRC16M)</td>
</tr>
<tr>
<td>RCU_IRC48M</td>
<td>internal 48M RC oscillators (IRC48M)</td>
</tr>
<tr>
<td>RCU_IRC32K</td>
<td>internal 32K RC oscillator (IRC32K)</td>
</tr>
<tr>
<td>RCU_PLL_CK</td>
<td>phase locked loop (PLL)</td>
</tr>
</tbody>
</table>

Output parameter (out):
- -

Return value:
- -

Example:

/* turn off the high speed crystal oscillator */
rcu_osci_off(RCU_HXTAL);
rcu_osci_bypass_mode_enable

The description of rcu_osci_bypass_mode_enable is shown as below:

Table 3-563. Function rcu_osci_bypass_mode_enable

<table>
<thead>
<tr>
<th>Function name</th>
<th>rcu_osci_bypass_mode_enable</th>
</tr>
</thead>
<tbody>
<tr>
<td>Function prototype</td>
<td>void rcu_osci_bypass_mode_enable(rcu_osci_type_enum osci);</td>
</tr>
<tr>
<td>Function descriptions</td>
<td>enable the oscillator bypass mode</td>
</tr>
<tr>
<td>Precondition</td>
<td>HXTALEN or LXTALEN must be reset before it</td>
</tr>
<tr>
<td>The called functions</td>
<td>-</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Input parameter(in)</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>osci</td>
<td>oscillator types, refer to Table 3-522. Enum rcu_osci_type_enum</td>
</tr>
<tr>
<td>RCU_HXTAL</td>
<td>high speed crystal oscillator(HXTAL)</td>
</tr>
<tr>
<td>RCU_LXTAL</td>
<td>low speed crystal oscillator(LXTAL)</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Output parameter(out)</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>-</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Return value</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>-</td>
</tr>
</tbody>
</table>

Example:

/* enable the high speed crystal oscillator bypass mode */
rcu_osci_bypass_mode_enable(RCU_HXTAL);

rcu_osci_bypass_mode_disable

The description of rcu_osci_bypass_mode_disable is shown as below:

Table 3-564. Function rcu_osci_bypass_mode_disable

<table>
<thead>
<tr>
<th>Function name</th>
<th>rcu_osci_bypass_mode_disable</th>
</tr>
</thead>
<tbody>
<tr>
<td>Function prototype</td>
<td>void rcu_osci_bypass_mode_disable(rcu_osci_type_enum osci);</td>
</tr>
<tr>
<td>Function descriptions</td>
<td>disable the oscillator bypass mode</td>
</tr>
<tr>
<td>Precondition</td>
<td>HXTALEN or LXTALEN must be reset before it</td>
</tr>
<tr>
<td>The called functions</td>
<td>-</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Input parameter(in)</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>osci</td>
<td>oscillator types, refer to Table 3-522. Enum rcu_osci_type_enum</td>
</tr>
<tr>
<td>RCU_HXTAL</td>
<td>high speed crystal oscillator(HXTAL)</td>
</tr>
<tr>
<td>RCU_LXTAL</td>
<td>low speed crystal oscillator(LXTAL)</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Output parameter(out)</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>-</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Return value</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>-</td>
</tr>
</tbody>
</table>

Example:

/* disable the high speed crystal oscillator bypass mode */
rcu_osci_bypass_mode_disable(RCU_HXTAL);

rcu_hxtal_clock_monitor_enable

The description of rcu_hxtal_clock_monitor_enable is shown as below:

Table 3-565. Function rcu_hxtal_clock_monitor_enable

<table>
<thead>
<tr>
<th>Function name</th>
<th>rcu_hxtal_clock_monitor_enable</th>
</tr>
</thead>
<tbody>
<tr>
<td>Function prototype</td>
<td>void rcu_hxtal_clock_monitor_enable(void);</td>
</tr>
<tr>
<td>Function descriptions</td>
<td>enable the HXTAL clock monitor</td>
</tr>
<tr>
<td>Precondition</td>
<td>-</td>
</tr>
<tr>
<td>The called functions</td>
<td>-</td>
</tr>
<tr>
<td>Input parameter(in)</td>
<td>-</td>
</tr>
<tr>
<td>Output parameter(out)</td>
<td>-</td>
</tr>
<tr>
<td>Return value</td>
<td>-</td>
</tr>
</tbody>
</table>

Example:

/* enable the HXTAL clock monitor */
rcu_hxtal_clock_monitor_enable();

rcu_hxtal_clock_monitor_disable

The description of rcu_hxtal_clock_monitor_disable is shown as below:

Table 3-566. Function rcu_hxtal_clock_monitor_disable

<table>
<thead>
<tr>
<th>Function name</th>
<th>rcu_hxtal_clock_monitor_disable</th>
</tr>
</thead>
<tbody>
<tr>
<td>Function prototype</td>
<td>void rcu_hxtal_clock_monitor_disable(void);</td>
</tr>
<tr>
<td>Function descriptions</td>
<td>disable the HXTAL clock monitor</td>
</tr>
<tr>
<td>Precondition</td>
<td>-</td>
</tr>
<tr>
<td>The called functions</td>
<td>-</td>
</tr>
<tr>
<td>Input parameter(in)</td>
<td>-</td>
</tr>
<tr>
<td>Output parameter(out)</td>
<td>-</td>
</tr>
<tr>
<td>Return value</td>
<td>-</td>
</tr>
</tbody>
</table>

Example:

/* disable the HXTAL clock monitor */
rcu_hxtal_clock_monitor_disable();
rcu_lxtal_clock_monitor_enable

The description of rcu_lxtal_clock_monitor_enable is shown as below:

Table 3-567. Function rcu_lxtal_clock_monitor_enable

<table>
<thead>
<tr>
<th>Function name</th>
<th>rcu_lxtal_clock_monitor_enable</th>
</tr>
</thead>
<tbody>
<tr>
<td>Function prototype</td>
<td>void rcu_lxtal_clock_monitor_enable(void);</td>
</tr>
<tr>
<td>Function descriptions</td>
<td>enable the LXTAL clock monitor</td>
</tr>
<tr>
<td>Precondition</td>
<td>-</td>
</tr>
<tr>
<td>The called functions</td>
<td>-</td>
</tr>
<tr>
<td>Input parameter(in)</td>
<td>-</td>
</tr>
<tr>
<td>Output parameter(out)</td>
<td>-</td>
</tr>
<tr>
<td>Return value</td>
<td>-</td>
</tr>
</tbody>
</table>

Example:

/* enable the LXTAL clock monitor */
rcu_lxtal_clock_monitor_enable();

rcu_lxtal_clock_monitor_disable

The description of rcu_lxtal_clock_monitor_disable is shown as below:

Table 3-568. Function rcu_lxtal_clock_monitor_disable

<table>
<thead>
<tr>
<th>Function name</th>
<th>rcu_lxtal_clock_monitor_disable</th>
</tr>
</thead>
<tbody>
<tr>
<td>Function prototype</td>
<td>void rcu_lxtal_clock_monitor_disable(void);</td>
</tr>
<tr>
<td>Function descriptions</td>
<td>disable the LXTAL clock monitor</td>
</tr>
<tr>
<td>Precondition</td>
<td>-</td>
</tr>
<tr>
<td>The called functions</td>
<td>-</td>
</tr>
<tr>
<td>Input parameter(in)</td>
<td>-</td>
</tr>
<tr>
<td>Output parameter(out)</td>
<td>-</td>
</tr>
<tr>
<td>Return value</td>
<td>-</td>
</tr>
</tbody>
</table>

Example:

/* disable the LXTAL clock monitor */
rcu_lxtal_clock_monitor_disable();
rcu_irc16m_adjust_value_set

The description of rcu_irc16m_adjust_value_set is shown as below:

Table 3-569. Function rcu_irc16m_adjust_value_set

<table>
<thead>
<tr>
<th>Function name</th>
<th>rcu_irc16m_adjust_value_set</th>
</tr>
</thead>
<tbody>
<tr>
<td>Function prototype</td>
<td>void rcu_irc16m_adjust_value_set(uint32_t irc16m_adjval);</td>
</tr>
<tr>
<td>Function descriptions</td>
<td>set the IRC16M adjust value</td>
</tr>
<tr>
<td>Precondition</td>
<td>-</td>
</tr>
<tr>
<td>The called functions</td>
<td>-</td>
</tr>
<tr>
<td>Input parameter(in)</td>
<td>-</td>
</tr>
<tr>
<td>Irc16m_adjval</td>
<td>IRC8M adjust value, must be between 0 and 0x1F</td>
</tr>
<tr>
<td>Output parameter(out)</td>
<td>-</td>
</tr>
<tr>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>Return value</td>
<td>-</td>
</tr>
</tbody>
</table>

Example:

/* set the IRC16M adjust value */
rcu_irc16m_adjust_value_set(0x10);

rcu_voltage_key_unlock

The description of rcu_voltage_key_unlock is shown as below:

Table 3-570. Function rcu_voltage_key_unlock

<table>
<thead>
<tr>
<th>Function name</th>
<th>rcu_voltage_key_unlock</th>
</tr>
</thead>
<tbody>
<tr>
<td>Function prototype</td>
<td>void rcu_voltage_key_unlock(void);</td>
</tr>
<tr>
<td>Function descriptions</td>
<td>unlock the voltage key</td>
</tr>
<tr>
<td>Precondition</td>
<td>-</td>
</tr>
<tr>
<td>The called functions</td>
<td>-</td>
</tr>
<tr>
<td>Input parameter(in)</td>
<td>-</td>
</tr>
<tr>
<td>Output parameter(out)</td>
<td>-</td>
</tr>
<tr>
<td>Return value</td>
<td>-</td>
</tr>
</tbody>
</table>

Example:

/* unlock the voltage key */
rcu_voltage_key_unlock();
The description of `rcu_clock_freq_get` is shown as below:

### Table 3-571. Function `rcu_clock_freq_get`

<table>
<thead>
<tr>
<th>Function name</th>
<th><code>rcu_clock_freq_get</code></th>
</tr>
</thead>
<tbody>
<tr>
<td>Function prototype</td>
<td><code>uint32_t rcu_clock_freq_get(rcu_clock_freq_enum clock) ;</code></td>
</tr>
<tr>
<td>Function descriptions</td>
<td>get the system clock, bus clock and peripheral clock frequency</td>
</tr>
<tr>
<td>Precondition</td>
<td>-</td>
</tr>
<tr>
<td>The called functions</td>
<td>-</td>
</tr>
</tbody>
</table>

#### Input parameter (in)

- `clock`  
  the clock frequency which to get, refer to `Table 3-523. Enum rcu_clock_freq_enum`
  
- `CK_SYS`  
  system clock frequency

- `CK_AHB`  
  AHB clock frequency

- `CK_APB1`  
  APB1 clock frequency

- `CK_APB2`  
  APB2 clock frequency

- `CK_ADC`  
  ADC clock frequency

- `CK_USART0`  
  USART0 clock frequency

- `CK_USART1`  
  USART1 clock frequency

- `CK_LPTIMER`  
  LPTIMER clock frequency

- `CK_LPUART`  
  LPUART clock frequency

#### Output parameter (out)

- `-`

#### Return value

- `uint32_t`  
  clock frequency of system, AHB, APB1, APB2, ADC or USART0/1, LPUART, LPTIMER

Example:

```
uint32_t temp_freq;

/* get the system clock frequency */

temp_freq = rcu_clock_freq_get(CK_SYS);
```

## 3.20. RTC

The Real-time Clock (RTC) is usually used as a clock-calendar. The ones in the Backup Domain consist of a 32-bit up-counter, an alarm, a prescaler, a divider and the RTC clock configuration register. The RTC registers are listed in chapter 3.20.1, the FWDGT firmware functions are introduced in chapter 3.20.2.
3.20.1. Descriptions of Peripheral registers

RTC registers are listed in the table shown as below:

<table>
<thead>
<tr>
<th>Registers</th>
<th>Descriptions</th>
</tr>
</thead>
<tbody>
<tr>
<td>RTC_TIME</td>
<td>RTC time of day register</td>
</tr>
<tr>
<td>RTC_DATE</td>
<td>RTC date register</td>
</tr>
<tr>
<td>RTC_CTL</td>
<td>RTC control register</td>
</tr>
<tr>
<td>RTC_STAT</td>
<td>RTC status register</td>
</tr>
<tr>
<td>RTC_PSC</td>
<td>RTC time prescaler register</td>
</tr>
<tr>
<td>RTC_WUT</td>
<td>RTC wakeup timer register</td>
</tr>
<tr>
<td>RTC_ALRM0TD</td>
<td>RTC alarm 0 time and date register</td>
</tr>
<tr>
<td>RTC_ALRM1TD</td>
<td>RTC alarm 1 time and date register</td>
</tr>
<tr>
<td>RTC_WPK</td>
<td>RTC write protection key register</td>
</tr>
<tr>
<td>RTC_SS</td>
<td>RTC sub second register</td>
</tr>
<tr>
<td>RTC_SHIFTCTL</td>
<td>RTC shift function control register</td>
</tr>
<tr>
<td>RTC_TTS</td>
<td>RTC time of timestamp register</td>
</tr>
<tr>
<td>RTC_DTS</td>
<td>RTC date of timestamp register</td>
</tr>
<tr>
<td>RTC_SSS</td>
<td>RTC sub second of timestamp register</td>
</tr>
<tr>
<td>RTC_HRFC</td>
<td>RTC high resolution frequency compensation register</td>
</tr>
<tr>
<td>RTC_TAMP</td>
<td>RTC tamper register</td>
</tr>
<tr>
<td>RTC_ALRM0SS</td>
<td>RTC alarm 0 sub second register</td>
</tr>
<tr>
<td>RTC_ALRM1SS</td>
<td>RTC alarm 1 sub second register</td>
</tr>
<tr>
<td>RTC_BKP0</td>
<td>RTC backup 0 register</td>
</tr>
<tr>
<td>RTC_BKP1</td>
<td>RTC backup 1 register</td>
</tr>
<tr>
<td>RTC_BKP2</td>
<td>RTC backup 2 register</td>
</tr>
<tr>
<td>RTC_BKP3</td>
<td>RTC backup 3 register</td>
</tr>
<tr>
<td>RTC_BKP4</td>
<td>RTC backup 4 register</td>
</tr>
</tbody>
</table>

3.20.2. Descriptions of Peripheral functions

RTC firmware functions are listed in the table shown as below:

<table>
<thead>
<tr>
<th>Function name</th>
<th>Function description</th>
</tr>
</thead>
<tbody>
<tr>
<td>rtc_deinit</td>
<td>reset most of the RTC registers</td>
</tr>
<tr>
<td>rtc_init</td>
<td>initialize RTC registers</td>
</tr>
<tr>
<td>rtc_init_mode_enter</td>
<td>enter RTC init mode</td>
</tr>
<tr>
<td>rtc_init_mode_exit</td>
<td>exit RTC init mode</td>
</tr>
<tr>
<td>rtc_register_sync_wait</td>
<td>wait until RTC_TIME and RTC_DATE registers are synchronized with APB clock, and the shadow registers are updated</td>
</tr>
</tbody>
</table>
## GD32L23x Firmware Library User Guide

### Function name | Function description
--- | ---
rtc_current_time_get | get current time and date
rtc_subsecond_get | get current subsecond value
rtc_alarm_config | configure RTC alarm
rtc_alarm_subsecond_config | configure subsecond of RTC alarm
rtc_alarm_get | get RTC alarm
rtc_alarm_subsecond_get | get RTC alarm subsecond
rtc_alarm_enable | enable RTC alarm
rtc_alarm_disable | disable RTC alarm
rtc_timestamp_enable | enable RTC time-stamp
rtc_timestamp_disable | disable RTC time-stamp
rtc_timestamp_get | get RTC timestamp time and date
rtc_timestamp_internal_event_config | configure RTC time-stamp internal event
rtc_timestamp_subsecond_get | get RTC time-stamp subsecond
rtc_tamper_enable | enable RTC tamper
rtc_tamper_disable | disable RTC tamper
rtc_tamper_mask | set specified RTC tamper mask function
rtc_tamper_without_bkp_reset | tamperx event does not erase the RTC_BKP registers
rtc_output_pin_select | select the RTC output pin
rtc_alarm_output_config | configure RTC alarm output source
rtc_calibration_output_config | configure RTC calibration output source
rtc_hour_adjust | adjust the daylight saving time by adding or subtracting one hour from the current time
rtc_second_adjust | adjust RTC second or subsecond value of current time
rtc_bypass_shadow_enable | enable RTC bypass shadow registers function
rtc_bypass_shadow_disable | disable RTC bypass shadow registers function
rtc_refclock_detection_enable | enable RTC reference clock detection function
rtc_refclock_detection_disable | disable RTC reference clock detection function
rtc_wakeup_enable | enable RTC wakeup timer
rtc_wakeup_disable | disable RTC wakeup timer
rtc_wakeup_clock_set | set auto wakeup timer clock
rtc_wakeup_timer_set | set auto wakeup timer value
rtc_wakeup_timer_get | get auto wakeup timer value
rtc_smooth_calibration_config | configure RTC smooth calibration
rtc_interrupt_enable | enable specified RTC interrupt
rtc_interrupt_disable | disable specified RTC interrupt
rtc_flag_get | check specified flag
rtc_flag_clear | clear specified flag

### Structure rtc_parameter_struct

#### Table 3-574. rtc_parameter_struct

<table>
<thead>
<tr>
<th>Member name</th>
<th>Function description</th>
</tr>
</thead>
<tbody>
<tr>
<td>rtc_current_time_get</td>
<td></td>
</tr>
<tr>
<td>rtc_subsecond_get</td>
<td></td>
</tr>
<tr>
<td>rtc_alarm_config</td>
<td></td>
</tr>
<tr>
<td>rtc_alarm_subsecond_config</td>
<td></td>
</tr>
<tr>
<td>rtc_alarm_get</td>
<td></td>
</tr>
<tr>
<td>rtc_alarm_subsecond_get</td>
<td></td>
</tr>
<tr>
<td>rtc_alarm_enable</td>
<td></td>
</tr>
<tr>
<td>rtc_alarm_disable</td>
<td></td>
</tr>
<tr>
<td>rtc_timestamp_enable</td>
<td></td>
</tr>
<tr>
<td>rtc_timestamp_disable</td>
<td></td>
</tr>
<tr>
<td>rtc_timestamp_get</td>
<td></td>
</tr>
<tr>
<td>rtc_timestamp_internal_event_config</td>
<td>configure RTC time-stamp internal event</td>
</tr>
<tr>
<td>rtc_timestamp_subsecond_get</td>
<td></td>
</tr>
<tr>
<td>rtc_tamper_enable</td>
<td></td>
</tr>
<tr>
<td>rtc_tamper_disable</td>
<td></td>
</tr>
<tr>
<td>rtc_tamper_mask</td>
<td></td>
</tr>
<tr>
<td>rtc_tamper_without_bkp_reset</td>
<td></td>
</tr>
<tr>
<td>rtc_output_pin_select</td>
<td></td>
</tr>
<tr>
<td>rtc_alarm_output_config</td>
<td></td>
</tr>
<tr>
<td>rtc_calibration_output_config</td>
<td></td>
</tr>
<tr>
<td>rtc_hour_adjust</td>
<td></td>
</tr>
<tr>
<td>rtc_second_adjust</td>
<td></td>
</tr>
<tr>
<td>rtc_bypass_shadow_enable</td>
<td></td>
</tr>
<tr>
<td>rtc_bypass_shadow_disable</td>
<td></td>
</tr>
<tr>
<td>rtc_refclock_detection_enable</td>
<td></td>
</tr>
<tr>
<td>rtc_refclock_detection_disable</td>
<td></td>
</tr>
<tr>
<td>rtc_wakeup_enable</td>
<td></td>
</tr>
<tr>
<td>rtc_wakeup_disable</td>
<td></td>
</tr>
<tr>
<td>rtc_wakeup_clock_set</td>
<td></td>
</tr>
<tr>
<td>rtc_wakeup_timer_set</td>
<td></td>
</tr>
<tr>
<td>rtc_wakeup_timer_get</td>
<td></td>
</tr>
<tr>
<td>rtc_smooth_calibration_config</td>
<td></td>
</tr>
<tr>
<td>rtc_interrupt_enable</td>
<td></td>
</tr>
<tr>
<td>rtc_interrupt_disable</td>
<td></td>
</tr>
<tr>
<td>rtc_flag_get</td>
<td></td>
</tr>
<tr>
<td>rtc_flag_clear</td>
<td></td>
</tr>
</tbody>
</table>

378
<table>
<thead>
<tr>
<th>year</th>
<th>RTC year value: 0x0 - 0x99 (BCD format)</th>
</tr>
</thead>
<tbody>
<tr>
<td>month</td>
<td>RTC month value (BCD format)</td>
</tr>
<tr>
<td>date</td>
<td>RTC date value: 0x1 - 0x31 (BCD format)</td>
</tr>
<tr>
<td>day_of_week</td>
<td>RTC weekday value (BCD format)</td>
</tr>
<tr>
<td>hour</td>
<td>RTC hour value: 0x1 - 0x12 (BCD format) or 0x0 - 0x23 (BCD format)</td>
</tr>
<tr>
<td>minute</td>
<td>RTC minute value: 0x0 - 0x59 (BCD format)</td>
</tr>
<tr>
<td>second</td>
<td>RTC second value: 0x0 - 0x59 (BCD format)</td>
</tr>
<tr>
<td>factor_asyn</td>
<td>RTC asynchronous prescaler value: 0x0 - 0x7F</td>
</tr>
<tr>
<td>factor_syn</td>
<td>RTC synchronous prescaler value: 0x0 - 0x7FFF</td>
</tr>
<tr>
<td>am_pm</td>
<td>RTC AM/PM value</td>
</tr>
<tr>
<td>display_format</td>
<td>RTC time notation</td>
</tr>
</tbody>
</table>

**Structure rtc_alarm_struct**

**Table 3-575. rtc_alarm_struct**

<table>
<thead>
<tr>
<th>Member name</th>
<th>Function description</th>
</tr>
</thead>
<tbody>
<tr>
<td>alarm_mask</td>
<td>RTC alarm mask</td>
</tr>
<tr>
<td>weekday_or_date</td>
<td>specify RTC alarm is on date or weekday</td>
</tr>
<tr>
<td>alarm_day</td>
<td>RTC alarm date or weekday value (BCD format)</td>
</tr>
<tr>
<td>alarm_hour</td>
<td>RTC alarm hour value: 0x1 - 0x12 (BCD format) or 0x0 - 0x23 (BCD format)</td>
</tr>
<tr>
<td>alarm_minute</td>
<td>RTC alarm minute value: 0x0 - 0x59 (BCD format)</td>
</tr>
<tr>
<td>alarm_second</td>
<td>RTC alarm second value: 0x0 - 0x59 (BCD format)</td>
</tr>
<tr>
<td>am_pm</td>
<td>RTC alarm AM/PM value</td>
</tr>
</tbody>
</table>

**Structure rtc_timestamp_struct**

**Table 3-576. rtc_timestamp_struct**

<table>
<thead>
<tr>
<th>Member name</th>
<th>Function description</th>
</tr>
</thead>
<tbody>
<tr>
<td>timestamp_month</td>
<td>RTC time-stamp month value (BCD format)</td>
</tr>
<tr>
<td>timestamp_date</td>
<td>RTC time-stamp date value: 0x1 - 0x31 (BCD format)</td>
</tr>
<tr>
<td>timestamp_day</td>
<td>RTC time-stamp weekday value (BCD format)</td>
</tr>
<tr>
<td>timestamp_hour</td>
<td>RTC time-stamp hour value (BCD format): 0x1 - 0x12 (BCD format) or 0x0 - 0x23 (BCD format)</td>
</tr>
<tr>
<td>timestamp_minute</td>
<td>RTC time-stamp minute value: 0x0 - 0x59 (BCD format)</td>
</tr>
<tr>
<td>timestamp_second</td>
<td>RTC time-stamp second value: 0x0 - 0x59 (BCD format)</td>
</tr>
<tr>
<td>am_pm</td>
<td>RTC time-stamp AM/PM value</td>
</tr>
</tbody>
</table>

**Structure rtc_tamper_struct**

**Table 3-577. rtc_tamper_struct**

<table>
<thead>
<tr>
<th>Member name</th>
<th>Function description</th>
</tr>
</thead>
<tbody>
<tr>
<td>tamper_source</td>
<td>RTC tamper source</td>
</tr>
<tr>
<td>tamper_trigger</td>
<td>RTC tamper trigger</td>
</tr>
<tr>
<td>tamper_filter</td>
<td>RTC tamper consecutive samples needed during a voltage level detection</td>
</tr>
</tbody>
</table>
tamper_sample_frequency
- RTC tamper sampling frequency during a voltage level detection

 tamper_precharge_enable
- RTC tamper precharge feature during a voltage level detection

 tamper_precharge_time
- RTC tamper precharge duration if precharge feature is enabled

 tamper_with_timestamp
- RTC tamper time-stamp feature

**rtc_deinit**

The description of rtc_deinit is shown as below:

**Table 3-578. Function rtc_deinit**

<table>
<thead>
<tr>
<th>Function name</th>
<th>rtc_deinit</th>
</tr>
</thead>
<tbody>
<tr>
<td>Function prototype</td>
<td>ErrStatus rtc_deinit(void);</td>
</tr>
<tr>
<td>Function descriptions</td>
<td>reset most of the RTC registers</td>
</tr>
<tr>
<td>Precondition</td>
<td>-</td>
</tr>
<tr>
<td>The called functions</td>
<td>rcu_periph_reset_enable/rcu_periph_reset_disable</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Input parameter(in)</th>
<th>-</th>
</tr>
</thead>
<tbody>
<tr>
<td>Output parameter(out)</td>
<td>-</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Return value</th>
<th>ErrStatus</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>ERROR or SUCCESS</td>
</tr>
</tbody>
</table>

Example:

`/* reset most of the RTC registers*/`

ErrStatus error_status = rtc_deinit();

**rtc_init**

The description of rtc_init is shown as below:

**Table 3-579. Function rtc_init**

<table>
<thead>
<tr>
<th>Function name</th>
<th>rtc_init</th>
</tr>
</thead>
<tbody>
<tr>
<td>Function prototype</td>
<td>ErrStatus rtc_init(rtc_parameter_struct* rtc_initpara_struct);</td>
</tr>
<tr>
<td>Function descriptions</td>
<td>initialize RTC registers</td>
</tr>
<tr>
<td>Precondition</td>
<td>-</td>
</tr>
<tr>
<td>The called functions</td>
<td>-</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Input parameter(in)</th>
<th>rtc_initpara_struct</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>pointer to a rtc_parameter_struct structure which contains parameters for initialization of the rtc peripheral, the structure members can refer to members of the structure Table 3-574. rtc_parameter_struct</td>
</tr>
</tbody>
</table>
Example:

/* initialize RTC registers */

rtc_parameter_struct rtc_initpara;

rtc_interrupt_disable(RTC_INT_SECOND);

rtc_initpara.factor_asyn = prescaler_a;

rtc_initpara.factor_syn = prescaler_s;

rtc_initpara.year = 0x16;

rtc_initpara.day_of_week = RTC_SATURDAY;

rtc_initpara.month = RTC_APR;

rtc_initpara.date = 0x30;

rtc_initpara.display_format = RTC_24HOUR;

rtc_initpara.am_pm = RTC_AM;

rtc_init(&rtc_initpara);

**rtc_init_mode_enter**

The description of rtc_init_mode_enter is shown as below:

<table>
<thead>
<tr>
<th>Function name</th>
<th>rtc_init_mode_enter</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Function prototype</strong></td>
<td>ErrStatus rtc_init_mode_enter(void);</td>
</tr>
<tr>
<td><strong>Function descriptions</strong></td>
<td>enter RTC init mode</td>
</tr>
<tr>
<td><strong>Precondition</strong></td>
<td>-</td>
</tr>
<tr>
<td><strong>The called functions</strong></td>
<td>-</td>
</tr>
<tr>
<td><strong>Input parameter(in)</strong></td>
<td>-</td>
</tr>
<tr>
<td><strong>Output parameter(out)</strong></td>
<td>-</td>
</tr>
<tr>
<td><strong>Return value</strong></td>
<td>ErrStatus</td>
</tr>
</tbody>
</table>

**Example:**

/* enter RTC init mode */
ErrStatus error_status = rtc_init_mode_enter();

**rtc_init_mode_exit**

The description of rtc_init_mode_exit is shown as below:

**Table 3-581. Function rtc_init_mode_exit**

<table>
<thead>
<tr>
<th>Function name</th>
<th>rtc_init_mode_exit</th>
</tr>
</thead>
<tbody>
<tr>
<td>Function prototype</td>
<td>void rtc_init_mode_exit(void);</td>
</tr>
<tr>
<td>Function descriptions</td>
<td>exit RTC init mode</td>
</tr>
<tr>
<td>Precondition</td>
<td>-</td>
</tr>
<tr>
<td>The called functions</td>
<td>-</td>
</tr>
<tr>
<td>Input parameter(in)</td>
<td>-</td>
</tr>
<tr>
<td>Output parameter(out)</td>
<td>-</td>
</tr>
<tr>
<td>Return value</td>
<td>-</td>
</tr>
</tbody>
</table>

Example:

/*exit RTC init mode*/

rtc_init_mode_exit();

**rtc_register_sync_wait**

The description of rtc_register_sync_wait is shown as below:

**Table 3-582. Function rtc_register_sync_wait**

<table>
<thead>
<tr>
<th>Function name</th>
<th>rtc_register_sync_wait</th>
</tr>
</thead>
<tbody>
<tr>
<td>Function prototype</td>
<td>ErrStatus rtc_register_sync_wait(void);</td>
</tr>
<tr>
<td>Function descriptions</td>
<td>wait until RTC_TIME and RTC_DATE registers are synchronized with APB clock, and the shadow registers are updated</td>
</tr>
<tr>
<td>Precondition</td>
<td>-</td>
</tr>
<tr>
<td>The called functions</td>
<td>-</td>
</tr>
<tr>
<td>Input parameter(in)</td>
<td>-</td>
</tr>
<tr>
<td>Output parameter(out)</td>
<td>-</td>
</tr>
<tr>
<td>Return value</td>
<td>ErrStatus ERROR or SUCCESS</td>
</tr>
</tbody>
</table>

Example:

/*wait until RTC_TIME and RTC_DATE registers are synchronized with APB clock, and the shadow registers are updated*/
ErrStatus error_status = rtc_register_sync_wait();

**rtc_current_time_get**

The description of `rtc_current_time_get` is shown as below:

<table>
<thead>
<tr>
<th>Function name</th>
<th>rtc_current_time_get</th>
</tr>
</thead>
<tbody>
<tr>
<td>Function prototype</td>
<td>void rtc_current_time_get(rtc_parameter_struct* rtc_initpara_struct);</td>
</tr>
<tr>
<td>Function descriptions</td>
<td>get current time and date</td>
</tr>
<tr>
<td>Precondition</td>
<td>-</td>
</tr>
<tr>
<td>The called functions</td>
<td>-</td>
</tr>
</tbody>
</table>

#### Input parameter(in)
- rtc_initpara_struct

#### Output parameter(out)
- rtc_initpara_struct

- return value

| rtc_initpara_struct   | pointer to a rtc_parameter_struct structure which contains parameters for initialization of the rtc peripheral, the structure members can refer to members of the structure Table 3-574. rtc_parameter_struct |

Example:

```c
/*get current time and date*/

rtc_parameter_struct rtc_initpara_struct;

rtc_current_time_get(&rtc_initpara_struct);
```

**rtc_subsecond_get**

The description of `rtc_subsecond_get` is shown as below:

<table>
<thead>
<tr>
<th>Function name</th>
<th>rtc_subsecond_get</th>
</tr>
</thead>
<tbody>
<tr>
<td>Function prototype</td>
<td>uint32_t rtc_subsecond_get(void);</td>
</tr>
<tr>
<td>Function descriptions</td>
<td>get current subsecond value</td>
</tr>
<tr>
<td>Precondition</td>
<td>-</td>
</tr>
<tr>
<td>The called functions</td>
<td>-</td>
</tr>
</tbody>
</table>

#### Input parameter(in)
- -

#### Output parameter(out)
- -

#### Return value
- uint32_t current subsecond value(0x00-0xFFFF)
/*get current subsecond value*/

uint32_t sub_second = rtc_subsecond_get();

**rtc_alarm_config**

The description of rtc_alarm_config is shown as below:

<table>
<thead>
<tr>
<th>Table 3-585. Function rtc_alarm_config</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Function name</strong></td>
</tr>
<tr>
<td><strong>Function prototype</strong></td>
</tr>
<tr>
<td><strong>Function descriptions</strong></td>
</tr>
<tr>
<td><strong>Precondition</strong></td>
</tr>
<tr>
<td><strong>The called functions</strong></td>
</tr>
<tr>
<td><strong>Input parameter(in)</strong></td>
</tr>
<tr>
<td>rtc_alarm</td>
</tr>
<tr>
<td><strong>Input parameter(in)</strong></td>
</tr>
<tr>
<td>rtc_alarm_time</td>
</tr>
<tr>
<td><strong>Output parameter(out)</strong></td>
</tr>
<tr>
<td>Return value</td>
</tr>
</tbody>
</table>

Example:

/*rtc_alarm_config*/

rtc_alarm_struct rtc_alarm_time;

rtc_alarm_config(RTC_ALARM0,&rtc_alarm_time);

**rtc_alarm_subsecond_config**

The description of rtc_alarm_subsecond_config is shown as below:

<table>
<thead>
<tr>
<th>Table 3-586. Function rtc_alarm_subsecond_config</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Function name</strong></td>
</tr>
<tr>
<td><strong>Function prototype</strong></td>
</tr>
<tr>
<td><strong>Function descriptions</strong></td>
</tr>
<tr>
<td><strong>Precondition</strong></td>
</tr>
<tr>
<td><strong>The called functions</strong></td>
</tr>
<tr>
<td><strong>Input parameter(in)</strong></td>
</tr>
<tr>
<td>rtc_alarm</td>
</tr>
<tr>
<td><strong>Input parameter(in)</strong></td>
</tr>
<tr>
<td>mask_subsecond</td>
</tr>
<tr>
<td>-------------------</td>
</tr>
<tr>
<td>RTC_MASKSSC_0_14</td>
</tr>
<tr>
<td>RTC_MASKSSC_1_14</td>
</tr>
<tr>
<td>RTC_MASKSSC_2_14</td>
</tr>
<tr>
<td>RTC_MASKSSC_3_14</td>
</tr>
<tr>
<td>RTC_MASKSSC_4_14</td>
</tr>
<tr>
<td>RTC_MASKSSC_5_14</td>
</tr>
<tr>
<td>RTC_MASKSSC_6_14</td>
</tr>
<tr>
<td>RTC_MASKSSC_7_14</td>
</tr>
<tr>
<td>RTC_MASKSSC_8_14</td>
</tr>
<tr>
<td>RTC_MASKSSC_9_14</td>
</tr>
<tr>
<td>RTC_MASKSSC_10_1</td>
</tr>
<tr>
<td>RTC_MASKSSC_11_1</td>
</tr>
<tr>
<td>RTC_MASKSSC_12_1</td>
</tr>
<tr>
<td>RTC_MASKSSC_13_1</td>
</tr>
<tr>
<td>RTC_MASKSSC_NON</td>
</tr>
</tbody>
</table>

### Input parameter(in)

- **subsecond**
  - alarm subsecond value (0x000 - 0x7FFF)

### Output parameter(out)

- **-**

### Return value

- **-**

Example:

```c
/*configure subsecond of RTC alarm*/

rtc_subsecond_config(RTC_ALRM0,RTC_MASKSSC_9_14, 0x7FFF);
```
**rtc_alarm_enable**

The description of rtc_alarm_enable is shown as below:

### Table 3-587. Function rtc_alarm_enable

<table>
<thead>
<tr>
<th>Function name</th>
<th>rtc_alarm_enable</th>
</tr>
</thead>
<tbody>
<tr>
<td>Function prototype</td>
<td>void rtc_alarm_enable(uint8_t rtc_alarm);</td>
</tr>
<tr>
<td>Function descriptions</td>
<td>enable RTC alarm</td>
</tr>
<tr>
<td>Precondition</td>
<td>-</td>
</tr>
<tr>
<td>The called functions</td>
<td>-</td>
</tr>
<tr>
<td>Input parameter (in)</td>
<td>rtc_alarm</td>
</tr>
<tr>
<td></td>
<td>RTC_ALARM0 or RTC_ALARM1</td>
</tr>
<tr>
<td>Output parameter (out)</td>
<td>-</td>
</tr>
<tr>
<td>Return value</td>
<td>-</td>
</tr>
</tbody>
</table>

**Example:**

```
/* enable RTC alarm */
rtc_alarm_enable(RTC_ALARM0);
```

**rtc_alarm_disable**

The description of rtc_alarm_disable is shown as below:

### Table 3-588. Function rtc_alarm_disable

<table>
<thead>
<tr>
<th>Function name</th>
<th>rtc_alarm_disable</th>
</tr>
</thead>
<tbody>
<tr>
<td>Function prototype</td>
<td>ErrStatus rtc_alarm_disable(uint8_t rtc_alarm);</td>
</tr>
<tr>
<td>Function descriptions</td>
<td>disable RTC alarm</td>
</tr>
<tr>
<td>Precondition</td>
<td>-</td>
</tr>
<tr>
<td>The called functions</td>
<td>-</td>
</tr>
<tr>
<td>Input parameter (in)</td>
<td>rtc_alarm</td>
</tr>
<tr>
<td></td>
<td>RTC_ALARM0 or RTC_ALARM1</td>
</tr>
<tr>
<td>Output parameter (out)</td>
<td>-</td>
</tr>
<tr>
<td>Return value</td>
<td>-</td>
</tr>
<tr>
<td>ErrStatus</td>
<td>ERROR or SUCCESS</td>
</tr>
</tbody>
</table>

**Example:**

```
/* disable RTC alarm */
ErrStatus error_status = rtc_alarm_disable(RTC_ALARM0);```
**rtc_alarm_get**

The description of `rtc_alarm_get` is shown as below:

**Table 3-589. Function rtc_alarm_get**

<table>
<thead>
<tr>
<th>Function name</th>
<th>rtc_alarm_get</th>
</tr>
</thead>
<tbody>
<tr>
<td>Function prototype</td>
<td>void rtc_alarm_get(uint8_t rtc_alarm, rtc_alarm_struct *rtc_alarm_time);</td>
</tr>
<tr>
<td>Function descriptions</td>
<td>get RTC alarm</td>
</tr>
<tr>
<td>Precondition</td>
<td>-</td>
</tr>
<tr>
<td>The called functions</td>
<td>-</td>
</tr>
</tbody>
</table>

**Input parameter(in)**

| rtc_alarm | RTC_ALARM0 or RTC_ALARM1 |

**Output parameter(out)**

| rtc_alarm_time | Pointer to a rtc_alarm_struct structure which contains parameters for RTC alarm configuration, the structure members can refer to members of the structure Table 3-575, rtc_alarm_struct |

**Return value**

Example:

```c
/*disable RTC alarm*/
rtc_alarm_struct rtc_alarm_time;
rtc_alarm_get(RTC_ALARM0,&rtc_alarm_time);
```

**rtc_alarm_subsecond_get**

The description of `rtc_alarm_subsecond_get` is shown as below:

**Table 3-590. Function rtc_alarm_subsecond_get**

<table>
<thead>
<tr>
<th>Function name</th>
<th>rtc_alarm_subsecond_get</th>
</tr>
</thead>
<tbody>
<tr>
<td>Function prototype</td>
<td>uint32_t rtc_alarm_subsecond_get(uint8_t rtc_alarm);</td>
</tr>
<tr>
<td>Function descriptions</td>
<td>get RTC alarm subsecond</td>
</tr>
<tr>
<td>Precondition</td>
<td>-</td>
</tr>
<tr>
<td>The called functions</td>
<td>-</td>
</tr>
</tbody>
</table>

**Input parameter(in)**

| rtc_alarm | RTC_ALARM0 or RTC_ALARM1 |

**Output parameter(out)**

| | |

**Return value**

Example:

```c
/*get RTC alarm subsecond*/
```
uint32_t subsecond = rtc_alarm_subsecond_get(RTC_ALARM0);

**rtc_timestamp_enable**

The description of rtc_timestamp_enable is shown as below:

**Table 3-591. Function rtc_timestamp_enable**

<table>
<thead>
<tr>
<th>Function name</th>
<th>rtc_timestamp_enable</th>
</tr>
</thead>
<tbody>
<tr>
<td>Function prototype</td>
<td>void rtc_timestamp_enable(uint32_t edge);</td>
</tr>
<tr>
<td>Function descriptions</td>
<td>enable RTC time-stamp</td>
</tr>
<tr>
<td>Precondition</td>
<td>-</td>
</tr>
<tr>
<td>The called functions</td>
<td>-</td>
</tr>
</tbody>
</table>

**Input parameter (in)**

<table>
<thead>
<tr>
<th>edge</th>
<th>specify which edge to detect of time-stamp</th>
</tr>
</thead>
</table>

| RTC_TIMESTAMP_RISING_EDGE     | rising edge is valid event edge for timestamp event |
| RTC_TIMESTAMP_FALLING_EDGE    | falling edge is valid event edge for timestamp event |

**Output parameter (out)**

- 

**Return value**

- Example:

  /*enable RTC time-stamp*/
  rtc_timestamp_enable(RTC_TIMESTAMP_RISING_EDGE);

**rtc_timestamp_disable**

The description of rtc_timestamp_disable is shown as below:

**Table 3-592. Function rtc_timestamp_disable**

<table>
<thead>
<tr>
<th>Function name</th>
<th>rtc_timestamp_disable</th>
</tr>
</thead>
<tbody>
<tr>
<td>Function prototype</td>
<td>void rtc_timestamp_disable(void);</td>
</tr>
<tr>
<td>Function descriptions</td>
<td>disable RTC time-stamp</td>
</tr>
<tr>
<td>Precondition</td>
<td>-</td>
</tr>
<tr>
<td>The called functions</td>
<td>-</td>
</tr>
</tbody>
</table>

**Input parameter (in)**

- 

**Output parameter (out)**

- 

**Return value**

- Example:
/* disable RTC time-stamp */

rtc_timestamp_disable();

rtc_timestamp_internalevent_config

The description of rtc_timestamp_internalevent_config is shown as below:

**Table 3-593. Function rtc_timestamp_internalevent_config**

<table>
<thead>
<tr>
<th>Function name</th>
<th>rtc_timestamp_internalevent_config</th>
</tr>
</thead>
<tbody>
<tr>
<td>Function prototype</td>
<td>void rtc_timestamp_internalevent_config(uint32_t mode)</td>
</tr>
<tr>
<td>Function descriptions</td>
<td>configure RTC time-stamp internal event</td>
</tr>
<tr>
<td>Precondition</td>
<td>-</td>
</tr>
<tr>
<td>The called functions</td>
<td>-</td>
</tr>
</tbody>
</table>

**Input parameter(in)**

<table>
<thead>
<tr>
<th>mode</th>
<th>specify which internal or external event to be detected</th>
</tr>
</thead>
<tbody>
<tr>
<td>RTC_ITSEN_DISABLE</td>
<td>disable RTC time-stamp internal event</td>
</tr>
<tr>
<td>RTC_ITSEN_ENABLE</td>
<td>enable RTC time-stamp internal event</td>
</tr>
</tbody>
</table>

**Output parameter(out)**

| - | - |

**Return value**

Example:

/* disable RTC time-stamp internal event */

rtc_timestamp_internalevent_config(RTC_ITSEN_DISABLE);

rtc_timestamp_get

The description of rtc_timestamp_get is shown as below:

**Table 3-594. Function rtc_timestamp_get**

<table>
<thead>
<tr>
<th>Function name</th>
<th>rtc_timestamp_get</th>
</tr>
</thead>
<tbody>
<tr>
<td>Function prototype</td>
<td>void rtc_timestamp_get(rtc_timestamp_struct* rtc_timestamp);</td>
</tr>
<tr>
<td>Function descriptions</td>
<td>get RTC timestamp time and date</td>
</tr>
<tr>
<td>Precondition</td>
<td>-</td>
</tr>
<tr>
<td>The called functions</td>
<td>-</td>
</tr>
</tbody>
</table>

**Input parameter(in)**

| - | - |

**Output parameter(out)**

| rtc_timestamp | Pointer to a rtc_timestamp_struct structure which contains parameters for RTC time-stamp configuration. The structure members can refer to members of the structure Table 3-577, rtc_tamper_struct |

**Return value**
Example:

/* get RTC timestamp time and date */
rtc_timestamp_struct rtc_timestamp;
rtc_timestamp_get(& rtc_timestamp);

**rtc_timestamp_subsecond_get**

The description of rtc_timestamp_subsecond_get is shown as below:

<table>
<thead>
<tr>
<th>Function name</th>
<th>rtc_timestamp_subsecond_get</th>
</tr>
</thead>
<tbody>
<tr>
<td>Function prototype</td>
<td>uint32_t rtc_timestamp_subsecond_get(void);</td>
</tr>
<tr>
<td>Function descriptions</td>
<td>get RTC time-stamp subsecond</td>
</tr>
<tr>
<td>Precondition</td>
<td>-</td>
</tr>
<tr>
<td>The called functions</td>
<td>Input parameter(in)</td>
</tr>
<tr>
<td></td>
<td>-</td>
</tr>
<tr>
<td></td>
<td>Output parameter(out)</td>
</tr>
<tr>
<td></td>
<td>-</td>
</tr>
<tr>
<td>Return value</td>
<td>uint32_t RTC time-stamp subsecond value</td>
</tr>
</tbody>
</table>

Example:

/* get RTC time-stamp subsecond */
uint32_t subsecond = rtc_timestamp_subsecond_get();

**rtc_tamper_enable**

The description of rtc_tamper_enable is shown as below:

<table>
<thead>
<tr>
<th>Function name</th>
<th>rtc_tamper_enable</th>
</tr>
</thead>
<tbody>
<tr>
<td>Function prototype</td>
<td>void rtc_tamper_enable(rtc_tamper_struct* rtc_tamper);</td>
</tr>
<tr>
<td>Function descriptions</td>
<td>enable RTC tamper</td>
</tr>
<tr>
<td>Precondition</td>
<td>-</td>
</tr>
<tr>
<td>The called functions</td>
<td>Input parameter(in)</td>
</tr>
<tr>
<td></td>
<td>rtc_tamper</td>
</tr>
<tr>
<td></td>
<td>pointer to a rtc_tamper_struct structure which contains parameters for RTC tamper configuration, the structure members can refer to members of the structure Table 3-577. rtc_tamper_struct</td>
</tr>
<tr>
<td></td>
<td>Output parameter(out)</td>
</tr>
</tbody>
</table>

[Table 3-595. Function rtc_timestamp_subsecond_get](#)

[Table 3-596. Function rtc_timestamp_enable](#)
Example:

/* enable RTC tamper */
rtc_tamper_struct rtc_tamper
rtc_tamper_enable(& rtc_tamper);

**rtc_tamper_disable**

The description of rtc_tamper_disable is shown as below:

**Table 3-597. Function rtc_tamper_disable**

<table>
<thead>
<tr>
<th>Function name</th>
<th>rtc_tamper_disable</th>
</tr>
</thead>
<tbody>
<tr>
<td>Function prototype</td>
<td>void rtc_tamper_disable(uint32_t source);</td>
</tr>
<tr>
<td>Function descriptions</td>
<td>disable RTC tamper</td>
</tr>
<tr>
<td>Precondition</td>
<td>-</td>
</tr>
<tr>
<td>The called functions</td>
<td>-</td>
</tr>
<tr>
<td>Input parameter(in)</td>
<td></td>
</tr>
<tr>
<td>source</td>
<td>specify which tamper source to be disabled</td>
</tr>
<tr>
<td>RTC_TAMPER0</td>
<td>RTC tamper0</td>
</tr>
<tr>
<td>RTC_TAMPER1</td>
<td>RTC tamper1</td>
</tr>
<tr>
<td>RTC_TAMPER2</td>
<td>RTC tamper2</td>
</tr>
<tr>
<td>Output parameter(out)</td>
<td></td>
</tr>
<tr>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>Return value</td>
<td>-</td>
</tr>
</tbody>
</table>

Example:

/* disable RTC tamper0 */
rtc_tamper_disable(RTC_TAMPER0);

**rtc_tamper_mask**

The description of rtc_tamper_mask is shown as below:

**Table 3-598. Function rtc_tamper_mask**

<table>
<thead>
<tr>
<th>Function name</th>
<th>rtc_tamper_mask</th>
</tr>
</thead>
<tbody>
<tr>
<td>Function prototype</td>
<td>void rtc_tamper_mask(uint32_t source);</td>
</tr>
<tr>
<td>Function descriptions</td>
<td>set specified RTC tamper mask</td>
</tr>
<tr>
<td>Precondition</td>
<td>-</td>
</tr>
<tr>
<td>The called functions</td>
<td>-</td>
</tr>
</tbody>
</table>
**Input parameter (in)**

<table>
<thead>
<tr>
<th>source</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>RTC_TAMPMASK_NONE</td>
<td>both tamper 0~2 would not be masked</td>
</tr>
<tr>
<td>RTC_TAMPMASK_TP0</td>
<td>tamper0 will be masked, both TPIE and TP0IE would be focused to reset</td>
</tr>
<tr>
<td>RTC_TAMPMASK_TP1</td>
<td>Tamper1 will be masked, both TPIE and TP1IE would be focused to reset</td>
</tr>
<tr>
<td>RTC_TAMPMASK_TP2</td>
<td>Tamper2 will be masked, both TPIE and TP2IE would be focused to reset</td>
</tr>
</tbody>
</table>

**Output parameter (out)**

| -                          | -                                                                           |

**Return value**

- -

Example:

```c
/* both tamper 0~3 would not be masked */
rtc_tamper_mask(RTC_TAMPMASK_NONE);
```

**rtc_tamper_without_bkp_reset**

The description of rtc_tamper_without_bkp_reset is shown as below:

**Table 3-599. Function rtc_tamper_without_bkp_reset**

<table>
<thead>
<tr>
<th>Function name</th>
<th>rtc_tamper_without_bkp_reset</th>
</tr>
</thead>
<tbody>
<tr>
<td>Function prototype</td>
<td>void rtc_tamper_without_bkp_reset(uint32_t ne_source);</td>
</tr>
<tr>
<td>Function descriptions</td>
<td>tamperx event does not erase the RTC_BKP registers</td>
</tr>
</tbody>
</table>

**Input parameter (in)**

<table>
<thead>
<tr>
<th>ne_source</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>RTC_TAMPXNOERAS_E_NONE</td>
<td>both tamper0 and tamper1 event will trigger RTC_BKP registers reset</td>
</tr>
<tr>
<td>RTC_TAMPXNOERAS_E_TP0</td>
<td>tamper0 event will not trigger RTC_BKP registers reset</td>
</tr>
<tr>
<td>RTC_TAMPXNOERAS_E_TP1</td>
<td>tamper1 event will not trigger RTC_BKP registers reset</td>
</tr>
<tr>
<td>RTC_TAMPXNOERAS_E_TP2</td>
<td>tamper2 event will not trigger RTC_BKP registers reset</td>
</tr>
<tr>
<td>RTC_TAMPXNOERAS_E_TP0_TP1</td>
<td>Neither tamper0 nor tamper1 event will trigger RTC_BKP registers reset</td>
</tr>
<tr>
<td>RTC_TAMPXNOERAS_E_TP_ALL</td>
<td>tamper 0~3 event all will not trigger RTC_BKP registers reset</td>
</tr>
</tbody>
</table>

**Output parameter (out)**

| -                          | -                                                                           |

**Return value**

- -
Example:

/* tamper 0~3 event all will not trigger RTC_BKP registers reset */

rtc_tamper_without_bkp_reset(RTC_TAMPXNOERASE_TP_ALL);

**rtc_output_pin_select**

The description of rtc_output_pin_select is shown as below:

![Table 3-600. Function rtc_output_pin_select](image)

**rtc_alarm_output_config**

The description of rtc_alarm_output_config is shown as below:

![Table 3-601. Function rtc_alarm_output_config](image)
### RTC_ALARM1_HIGH
when the alarm1 flag is set, the output pin is high

### RTC_ALARM1_LOW
when the alarm1 flag is set, the output pin is low

### RTC_WAKEUP_HIGH
when the wakeup flag is set, the output pin is high

### RTC_WAKEUP_LOW
when the wakeup flag is set, the output pin is low

### Input parameter(in)

<table>
<thead>
<tr>
<th>mode</th>
<th>description</th>
</tr>
</thead>
<tbody>
<tr>
<td>RTC_ALARM_OUTPUT_T_OD</td>
<td>open drain mode</td>
</tr>
<tr>
<td>RTC_ALARM_OUTPUT_T_PP</td>
<td>push pull mode</td>
</tr>
</tbody>
</table>

### Output parameter(out)

<table>
<thead>
<tr>
<th>Return value</th>
</tr>
</thead>
</table>

Example:

/* configure rtc alarm0 output source */

rtc_alarm_output_config(RTC_ALARM0_LOW, RTC_ALARM_OUTPUT_PP);

#### rtc_calibration_output_config

The description of rtc_calibration_output_config is shown as below:

**Table 3-602. rtc_calibration_output_config**

<table>
<thead>
<tr>
<th>Function name</th>
<th>rtc_calibration_output_config</th>
</tr>
</thead>
<tbody>
<tr>
<td>Function prototype</td>
<td>void rtc_calibration_output_config(uint32_t source);</td>
</tr>
<tr>
<td>Function descriptions</td>
<td>configure rtc calibration output source</td>
</tr>
<tr>
<td>Precondition</td>
<td>-</td>
</tr>
<tr>
<td>The called functions</td>
<td>-</td>
</tr>
</tbody>
</table>

**Input parameter(in)**

<table>
<thead>
<tr>
<th>source</th>
<th>description</th>
</tr>
</thead>
<tbody>
<tr>
<td>RTC_CALIBRATION_5_12HZ</td>
<td>when the LSE frequency is 32768Hz and the RTC_PSC is the default value, output 512Hz signal</td>
</tr>
<tr>
<td>RTC_CALIBRATION_1_HZ</td>
<td>when the LSE frequency is 32768Hz and the RTC_PSC is the default value, output 1Hz signal</td>
</tr>
</tbody>
</table>

**Output parameter(out)**

<table>
<thead>
<tr>
<th>Return value</th>
</tr>
</thead>
</table>

Example:

/* when the LSE frequency is 32768Hz and the RTC_PSC is the default value, output 1Hz signal */
rtc_hour_adjust

The description of rtc_hour_adjust is shown as below:

**Table 3-603. rtc_hour_adjust**

<table>
<thead>
<tr>
<th>Function name</th>
<th>rtc_hour_adjust</th>
</tr>
</thead>
<tbody>
<tr>
<td>Function prototype</td>
<td>void rtc_hour_adjust(uint32_t operation);</td>
</tr>
<tr>
<td>Function descriptions</td>
<td>adjust the daylight saving time by adding or subtracting one hour from the current time</td>
</tr>
<tr>
<td>Precondition</td>
<td>-</td>
</tr>
<tr>
<td>The called functions</td>
<td>-</td>
</tr>
<tr>
<td>operation</td>
<td>hour adjustment operation</td>
</tr>
<tr>
<td>RTC_CTL_A1H</td>
<td>add one hour</td>
</tr>
<tr>
<td>RTC_CTL_S1H</td>
<td>subtract one hour</td>
</tr>
<tr>
<td>Input parameter(in)</td>
<td></td>
</tr>
<tr>
<td>add</td>
<td>add 1s to current time or not</td>
</tr>
<tr>
<td>RTC_SHIFT_ADD1S_RST</td>
<td>no effect</td>
</tr>
<tr>
<td>RTC_SHIFT_ADD1S_SET</td>
<td>add 1s to current time</td>
</tr>
<tr>
<td>Output parameter(out)</td>
<td></td>
</tr>
<tr>
<td>-</td>
<td></td>
</tr>
<tr>
<td>Return value</td>
<td></td>
</tr>
<tr>
<td>-</td>
<td></td>
</tr>
</tbody>
</table>

Example:

```c
/* adjust the daylight saving time by adding one hour from the current time */
rtc_hour_adjust(RTC_CTL_A1H);
```

rtc_second_adjust

The description of rtc_second_adjust is shown as below:

**Table 3-604. rtc_second_adjust**

<table>
<thead>
<tr>
<th>Function name</th>
<th>rtc_second_adjust</th>
</tr>
</thead>
<tbody>
<tr>
<td>Function prototype</td>
<td>ErrStatus rtc_second_adjust(uint32_t add, uint32_t minus);</td>
</tr>
<tr>
<td>Function descriptions</td>
<td>adjust RTC second or subsecond value of current time</td>
</tr>
<tr>
<td>Precondition</td>
<td>-</td>
</tr>
<tr>
<td>The called functions</td>
<td>-</td>
</tr>
<tr>
<td>Input parameter(in)</td>
<td></td>
</tr>
<tr>
<td>add</td>
<td>add 1s to current time or not</td>
</tr>
<tr>
<td>RTC_SHIFT_ADD1S_RST</td>
<td>no effect</td>
</tr>
<tr>
<td>RTC_SHIFT_ADD1S_SET</td>
<td>add 1s to current time</td>
</tr>
<tr>
<td>minus</td>
<td>number of subsecond to minus from current time(0x0 - 0x7FFF)</td>
</tr>
</tbody>
</table>
Output parameter(out) | - | -
| | | |
| **Return value** | **ErrStatus** | ERROR or SUCCESS

Example:

/* adjust RTC second or subsecond value of current time */

ErrStatus error_status = rtc_second_adjust(RTC_SHIFT_ADD1S_SET, 0);

**rtc_bypass_shadow_enable**

The description of `rtc_bypass_shadow_enable` is shown as below:

**Table 3-605. rtc_bypass_shadow_enable**

<table>
<thead>
<tr>
<th>Function name</th>
<th>rtc_bypass_shadow_enable</th>
</tr>
</thead>
<tbody>
<tr>
<td>Function prototype</td>
<td>void rtc_bypass_shadow_enable(void);</td>
</tr>
<tr>
<td>Function descriptions</td>
<td>enable RTC bypass shadow registers function</td>
</tr>
<tr>
<td>Precondition</td>
<td>-</td>
</tr>
<tr>
<td>The called functions</td>
<td>-</td>
</tr>
</tbody>
</table>

**Input parameter(in)**

| - | - |

**Output parameter(out)**

| - | - |

**Return value**

| - | - |

Example:

/* enable RTC bypass shadow registers function*/

rtc_bypass_shadow_enable();

**rtc_bypass_shadow_disable**

The description of `rtc_bypass_shadow_disable` shown as below:

**Table 3-606. rtc_bypass_shadow_disable**

<table>
<thead>
<tr>
<th>Function name</th>
<th>rtc_bypass_shadow_disable</th>
</tr>
</thead>
<tbody>
<tr>
<td>Function prototype</td>
<td>void rtc_bypass_shadow_disable(void);</td>
</tr>
<tr>
<td>Function descriptions</td>
<td>disable RTC bypass shadow registers function</td>
</tr>
<tr>
<td>Precondition</td>
<td>-</td>
</tr>
<tr>
<td>The called functions</td>
<td>-</td>
</tr>
</tbody>
</table>

**Input parameter(in)**

| - | - |

**Output parameter(out)**

| - | - |
Example:

/* disable RTC bypass shadow registers function*/
rtc_bypass_shadow_disable();

**rtc_refclock_detection_enable**

The description of rtc_refclock_detection_enable shown as below:

Table 3-607. rtc_refclock_detection_enable

<table>
<thead>
<tr>
<th>Function name</th>
<th>rtc_refclock_detection_enable</th>
</tr>
</thead>
<tbody>
<tr>
<td>Function prototype</td>
<td>ErrStatus rtc_refclock_detection_enable(void);</td>
</tr>
<tr>
<td>Function descriptions</td>
<td>enable RTC reference clock detection function</td>
</tr>
<tr>
<td>Precondition</td>
<td>-</td>
</tr>
<tr>
<td>The called functions</td>
<td>rtc_init_mode_enter/rtc_init_mode_exit</td>
</tr>
</tbody>
</table>

Input parameter(in)

- -

Output parameter(out)

- -

Return value

ErrStatus ERROR or SUCCESS

Example:

/* enable RTC reference clock detection function*/
ErrStatus error_status = rtc_refclock_detection_enable();

**rtc_refclock_detection_disable**

The description of rtc_refclock_detection_disable shown as below:

Table 3-608. rtc_refclock_detection_disable

<table>
<thead>
<tr>
<th>Function name</th>
<th>rtc_refclock_detection_disable</th>
</tr>
</thead>
<tbody>
<tr>
<td>Function prototype</td>
<td>ErrStatus rtc_refclock_detection_disable(void);</td>
</tr>
<tr>
<td>Function descriptions</td>
<td>disable RTC reference clock detection function</td>
</tr>
<tr>
<td>Precondition</td>
<td>-</td>
</tr>
<tr>
<td>The called functions</td>
<td>rtc_init_mode_enter/rtc_init_mode_exit</td>
</tr>
</tbody>
</table>

Input parameter(in)

- -

Output parameter(out)

- -
Example:

/* disable RTC reference clock detection function */

ErrStatus error_status = rtc_refclock_detection_disable();

**rtc_wakeup_enable**

The description of rtc_refclock_detection_disable is shown as below:

Table 3-609. Function rtc_wakeup_enable

<table>
<thead>
<tr>
<th>Function name</th>
<th>rtc_wakeup_enable</th>
</tr>
</thead>
<tbody>
<tr>
<td>Function prototype</td>
<td>void rtc_wakeup_enable(void);</td>
</tr>
<tr>
<td>Function descriptions</td>
<td>enable RTC auto wake up function</td>
</tr>
<tr>
<td>Precondition</td>
<td>-</td>
</tr>
<tr>
<td>The called functions</td>
<td>-</td>
</tr>
<tr>
<td>Input parameter(in)</td>
<td>-</td>
</tr>
<tr>
<td>Output parameter(out)</td>
<td>-</td>
</tr>
<tr>
<td>Return value</td>
<td>-</td>
</tr>
</tbody>
</table>

Example:

/* enable RTC auto wake up function */

rtc_wakeup_enable();

**rtc_wakeup_disable**

The description of rtc_wakeup_disable is shown as below:

Table 3-610. Function rtc_wakeup_disable

<table>
<thead>
<tr>
<th>Function name</th>
<th>rtc_wakeup_disable</th>
</tr>
</thead>
<tbody>
<tr>
<td>Function prototype</td>
<td>ErrStatus rtc_wakeup_disable(void);</td>
</tr>
<tr>
<td>Function descriptions</td>
<td>disable RTC auto wake up function</td>
</tr>
<tr>
<td>Precondition</td>
<td>-</td>
</tr>
<tr>
<td>The called functions</td>
<td>-</td>
</tr>
<tr>
<td>Input parameter(in)</td>
<td>-</td>
</tr>
<tr>
<td>Output parameter(out)</td>
<td>-</td>
</tr>
<tr>
<td>Return value</td>
<td>-</td>
</tr>
</tbody>
</table>
Example:

/* disable RTC auto wakeup function */

ErrStatus error_status = rtc_wakeup_disable();

**rtc_wakeup_clock_set**

The description of rtc_wakeup_clock_set is shown as below:

<table>
<thead>
<tr>
<th>Function name</th>
<th>rtc_wakeup_clock_set</th>
</tr>
</thead>
<tbody>
<tr>
<td>Function prototype</td>
<td>ErrStatus rtc_wakeup_clock_set(uint8_t wakeup_clock);</td>
</tr>
<tr>
<td>Function descriptions</td>
<td>set RTC auto wakeup timer clock</td>
</tr>
<tr>
<td>Precondition</td>
<td>-</td>
</tr>
<tr>
<td>The called functions</td>
<td>-</td>
</tr>
<tr>
<td><strong>Input parameter</strong>(in)</td>
<td></td>
</tr>
<tr>
<td>wakeup_clock</td>
<td>wake up timer clock is RTC clock divided factor</td>
</tr>
<tr>
<td>WAKEUP_RTCCK_DIV 16</td>
<td>RTC auto wakeup timer clock is RTC clock divided by 16</td>
</tr>
<tr>
<td>WAKEUP_RTCCK_DIV 8</td>
<td>RTC auto wakeup timer clock is RTC clock divided by 8</td>
</tr>
<tr>
<td>WAKEUP_RTCCK_DIV 4</td>
<td>RTC auto wakeup timer clock is RTC clock divided by 4</td>
</tr>
<tr>
<td>WAKEUP_RTCCK_DIV 2</td>
<td>RTC auto wakeup timer clock is RTC clock divided by 2</td>
</tr>
<tr>
<td>WAKEUP_CKSPRE</td>
<td>RTC auto wakeup timer clock is ckspre</td>
</tr>
<tr>
<td>WAKEUP_CKSPRE 2 EXP16</td>
<td>RTC auto wakeup timer clock is ckspre and wakeup timer add 2exp16</td>
</tr>
<tr>
<td><strong>Output parameter</strong>(out)</td>
<td></td>
</tr>
<tr>
<td></td>
<td>-</td>
</tr>
<tr>
<td><strong>Return value</strong></td>
<td>ErrStatus ERROR or SUCCESS</td>
</tr>
</tbody>
</table>

Example:

/* RTC auto wakeup timer clock is ckspre */

ErrStatus error_status = rtc_wakeup_clock_set(WAKEUP_RTCCK_DIV8);  

**rtc_wakeup_timer_set**

The description of rtc_wakeup_timer_set is shown as below:

**Table 3-612. Function rtc_wakeup_timer_set**

<table>
<thead>
<tr>
<th>Function name</th>
<th>rtc_wakeup_timer_set</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Return value</strong></td>
<td>ErrStatus ERROR or SUCCESS</td>
</tr>
</tbody>
</table>
Function prototype: ErrStatus rtc_wakeup_timer_set(uint16_t wakeup_timer);

Function descriptions: set wakeup timer value

Precondition: -

The called functions: -

Input parameter(in)

<table>
<thead>
<tr>
<th>wakeup_timer</th>
<th>wakeup timer value</th>
</tr>
</thead>
<tbody>
<tr>
<td>uint16_t</td>
<td>0x0000-0xFFFF</td>
</tr>
</tbody>
</table>

Output parameter(out)

- |

Return value

| ErrStatus | ERROR or SUCCESS |

Example:

/* set wakeup timer value */

ErrStatus error_status = rtc_wakeup_timer_set(0XFFEE);

**rtc_wakeup_timer_get**

The description of rtc_wakeup_timer_set is shown as below:

**Table 3-613. Function rtc_wakeup_timer_get**

<table>
<thead>
<tr>
<th>Function name</th>
<th>rtc_wakeup_timer_get</th>
</tr>
</thead>
<tbody>
<tr>
<td>Function prototype</td>
<td>uint16_t rtc_wakeup_timer_get(void);</td>
</tr>
<tr>
<td>Function descriptions</td>
<td>set wakeup timer value</td>
</tr>
<tr>
<td>Precondition</td>
<td>-</td>
</tr>
<tr>
<td>The called functions</td>
<td>-</td>
</tr>
<tr>
<td>Input parameter(in)</td>
<td>-</td>
</tr>
<tr>
<td>Output parameter(out)</td>
<td>-</td>
</tr>
<tr>
<td>Return value</td>
<td>uint16_t 0-0xFFFF</td>
</tr>
</tbody>
</table>

Example:

/* get wakeup timer value */

uint32_t wakeup_time = rtc_wakeup_timer_get();

**rtc_smooth_calibration_config**

The description of rtc_smooth_calibration_config is shown as below:

**Table 3-614. rtc_smooth_calibration_config**

<table>
<thead>
<tr>
<th>Function name</th>
<th>rtc_smooth_calibration_config</th>
</tr>
</thead>
</table>
### Function prototype

ErrStatus rtc_smooth_calibration_config(uint32_t window, uint32_t plus, uint32_t minus);

### Function descriptions
configure RTC smooth calibration

### Precondition
-

### The called functions
-

#### Input parameter(in)

<table>
<thead>
<tr>
<th>window</th>
<th>select calibration window</th>
</tr>
</thead>
<tbody>
<tr>
<td>RTC_CALIBRATION_WINDOW_32S</td>
<td>2exp20 RTCLCLK cycles, 32s if RTCLCLK = 32768 Hz</td>
</tr>
<tr>
<td>RTC_CALIBRATION_WINDOW_16S</td>
<td>2exp19 RTCLCLK cycles, 16s if RTCLCLK = 32768 Hz</td>
</tr>
<tr>
<td>RTC_CALIBRATION_WINDOW_8S</td>
<td>2exp18 RTCLCLK cycles, 8s if RTCLCLK = 32768 Hz</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>plus</th>
<th>add RTC clock or not</th>
</tr>
</thead>
<tbody>
<tr>
<td>RTC_CALIBRATION_PLUS_SET</td>
<td>add one RTC clock every 2048 rtc clock</td>
</tr>
<tr>
<td>RTC_CALIBRATION_PLUS_RESET</td>
<td>no effect</td>
</tr>
</tbody>
</table>

| minus                       | the RTC clock to minus during the calibration window(0x0 - 0x1FF) |

### Output parameter(out)
-

### Return value
ErrStatus ERROR or SUCCESS

#### Example:

```c
/* configure RTC smooth calibration */

ErrStatus error_status;

error_status = rtc_smooth_calibration_config(RTC_CALIBRATION_WINDOW_32S, RTC_CALIBRATION_PLUS_SET, 0x10);
```

### rtc_interrupt_enable

The description of rtc_interrupt_enable is shown as below:

#### Table 3-615. Function rtc_interrupt_enable

<table>
<thead>
<tr>
<th>Function name</th>
<th>rtc_interrupt_enable</th>
</tr>
</thead>
<tbody>
<tr>
<td>Function prototype</td>
<td>void rtc_interrupt_enable(uint32_t interrupt);</td>
</tr>
<tr>
<td>Function descriptions</td>
<td>enable specified RTC interrupt</td>
</tr>
<tr>
<td>Precondition</td>
<td>-</td>
</tr>
<tr>
<td>The called functions</td>
<td>-</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Input parameter(in)</th>
<th></th>
</tr>
</thead>
</table>

401
interrupt | specify which interrupt source to be enabled
--- | ---
RTC_INT_TIMESTAMP | timestamp interrupt
RTC_INT_ALARM0 | Alarm0 interrupt
RTC_INT_ALARM1 | Alarm1 interrupt
RTC_INT_TAMP_ALL | tamp interrupt
RTC_INT_TAMP0 | Tamper0 detection interrupt
RTC_INT_TAMP1 | Tamper1 detection interrupt
RTC_INT_TAMP2 | Tamper2 detection interrupt
RTC_INT_WAKEUP | wakeup timer interrupt

Output parameter(out)
- -

Return value
- -

Example:

/* enable specified RTC interrupt*/
rtc_interrupt_enable(RTC_INT_TAMP0);

rtc_interrupt_disable

The description of rtc_interrupt_disable is shown as below:

<table>
<thead>
<tr>
<th>Function name</th>
<th>rtc_interrupt_disable</th>
</tr>
</thead>
<tbody>
<tr>
<td>Function prototype</td>
<td>void rtc_interrupt_disable(uint32_t interrupt);</td>
</tr>
<tr>
<td>Function descriptions</td>
<td>disable specified RTC interrupt</td>
</tr>
<tr>
<td>Precondition</td>
<td>-</td>
</tr>
<tr>
<td>The called functions</td>
<td>-</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>interrupt</th>
<th>specify which RTC interrupt to disable</th>
</tr>
</thead>
<tbody>
<tr>
<td>RTC_INT_TIMESTAMP</td>
<td>timestamp interrupt</td>
</tr>
<tr>
<td>RTC_INT_ALARM0</td>
<td>Alarm0 interrupt</td>
</tr>
<tr>
<td>RTC_INT_ALARM1</td>
<td>Alarm1 interrupt</td>
</tr>
<tr>
<td>RTC_INT_TAMP_ALL</td>
<td>tamp interrupt</td>
</tr>
<tr>
<td>RTC_INT_TAMP0</td>
<td>Tamper0 detection interrupt</td>
</tr>
<tr>
<td>RTC_INT_TAMP1</td>
<td>Tamper1 detection interrupt</td>
</tr>
<tr>
<td>RTC_INT_TAMP2</td>
<td>Tamper2 detection interrupt</td>
</tr>
<tr>
<td>RTC_INT_WAKEUP</td>
<td>wakeup timer interrupt</td>
</tr>
</tbody>
</table>

Output parameter(out)
- -

Return value
- -
Example:

/* disable specified RTC interrupt */

rtc_interrupt_disable(RTC_INT_TAMP0);

**rtc_flag_get**

The description of rtc_flag_get is shown as below:

Table 3-617. Function rtc_flag_get

<table>
<thead>
<tr>
<th>Function name</th>
<th>rtc_flag_get</th>
</tr>
</thead>
<tbody>
<tr>
<td>Function prototype</td>
<td>FlagStatus rtc_flag_get(uint32_t flag);</td>
</tr>
<tr>
<td>Function descriptions</td>
<td>check specified flag</td>
</tr>
<tr>
<td>Precondition</td>
<td>-</td>
</tr>
<tr>
<td>The called functions</td>
<td>-</td>
</tr>
</tbody>
</table>

**Input parameter(in)**

<table>
<thead>
<tr>
<th>flag</th>
<th>specify which flag to check</th>
</tr>
</thead>
<tbody>
<tr>
<td>RTC_FLAG_SCP</td>
<td>smooth calibration pending flag</td>
</tr>
<tr>
<td>RTC_FLAG_TP2</td>
<td>RTC tamper 2 detected flag</td>
</tr>
<tr>
<td>RTC_FLAG_TP1</td>
<td>RTC tamper 1 detected flag</td>
</tr>
<tr>
<td>RTC_FLAG_TP0</td>
<td>RTC tamper 0 detected flag</td>
</tr>
<tr>
<td>RTC_FLAG_TSOVR</td>
<td>time-stamp overflow event flag</td>
</tr>
<tr>
<td>RTC_FLAG_TS</td>
<td>time-stamp event flag</td>
</tr>
<tr>
<td>RTC_FLAG_ALARM0</td>
<td>Alarm0 event flag</td>
</tr>
<tr>
<td>RTC_FLAG_ALARM1</td>
<td>Alarm1 event flag</td>
</tr>
<tr>
<td>RTC_FLAG_WT</td>
<td>wakeup timer occurs flag</td>
</tr>
<tr>
<td>RTC_FLAG_INIT</td>
<td>init mode event flag</td>
</tr>
<tr>
<td>RTC_FLAG_RSYN</td>
<td>time and date registers synchronized event flag</td>
</tr>
<tr>
<td>RTC_FLAG_YCM</td>
<td>year parameter configured event flag</td>
</tr>
<tr>
<td>RTC_FLAG_SOP</td>
<td>shift function operation pending flag</td>
</tr>
<tr>
<td>RTC_FLAG_ALARM0W</td>
<td>Alarm0 written available flag</td>
</tr>
<tr>
<td>RTC_FLAG_ALARM1W</td>
<td>Alarm1 written available flag</td>
</tr>
<tr>
<td>RTC_FLAG_WTW</td>
<td>wakeup timer can be written flag</td>
</tr>
</tbody>
</table>

**Output parameter(out)**

- |

**Return value**

| FlagStatus | SET or RESET |

Example:

/* check time-stamp event flag */

FlagStatus = rtc_flag_get(RTC_FLAG_TS);
The description of `rtc_flag_clear` is shown as below:

### Table 3-618. Function `rtc_flag_clear`

<table>
<thead>
<tr>
<th>Function name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>rtc_flag_clear</code></td>
<td>void <code>rtc_flag_clear(uint32_t flag)</code>;</td>
</tr>
</tbody>
</table>

#### Function prototype

- `clear specified flag`

#### Precondition

- `-`

#### The called functions

- `-`

#### Input parameter (in)

<table>
<thead>
<tr>
<th>flag</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>RTC_FLAG_TP2</code></td>
<td>RTC tamper 2 detected flag</td>
</tr>
<tr>
<td><code>RTC_FLAG_TP1</code></td>
<td>RTC tamper 1 detected flag</td>
</tr>
<tr>
<td><code>RTC_FLAG_TP0</code></td>
<td>RTC tamper 0 detected flag</td>
</tr>
<tr>
<td><code>RTC_FLAG_TSOVR</code></td>
<td>time-stamp overflow event flag</td>
</tr>
<tr>
<td><code>RTC_FLAG_TS</code></td>
<td>time-stamp event flag</td>
</tr>
<tr>
<td><code>RTC_FLAG_WT</code></td>
<td>wakeup timer occurs flag</td>
</tr>
<tr>
<td><code>RTC_FLAG_ALARM0</code></td>
<td>Alarm0 event flag</td>
</tr>
<tr>
<td><code>RTC_FLAG_ALARM1</code></td>
<td>Alarm1 event flag</td>
</tr>
<tr>
<td><code>RTC_FLAG_RSYN</code></td>
<td>time and date registers synchronized event flag</td>
</tr>
</tbody>
</table>

#### Return value

- `-`

Example:

```c
/* clear time-stamp event flag */

rtc_flag_clear(RTC_FLAG_TS);
```

## 3.21. SLCD

The SLCD controller directly drives LCD displays by creating the AC segment and commonvoltage signals automatically. The SLCD registers are listed in chapter 3.21.1, the SLCD firmware functions are introduced in chapter 3.21.2.

### 3.21.1. Descriptions of Peripheral registers

SLCD registers are listed in the table shown as below:

### Table 3-619. SLCD Registers

<table>
<thead>
<tr>
<th>Registers</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SLCD_CTL</td>
<td>SLCD control register</td>
</tr>
</tbody>
</table>
### 3.21.2. Descriptions of Peripheral functions

SLCD firmware functions are listed in the table shown as below:

**Table 3-620. SLCD firmware function**

<table>
<thead>
<tr>
<th>Function name</th>
<th>Function description</th>
</tr>
</thead>
<tbody>
<tr>
<td>slcd_deinit</td>
<td>reset SLCD interface</td>
</tr>
<tr>
<td>slcd_enable</td>
<td>enable SLCD interface</td>
</tr>
<tr>
<td>slcd_disable</td>
<td>disable SLCD interface</td>
</tr>
<tr>
<td>slcd_init</td>
<td>initialize SLCD interface</td>
</tr>
<tr>
<td>slcd_enhance_mode_enable</td>
<td>enable SLCD enhance mode</td>
</tr>
<tr>
<td>slcd_enhance_mode_disable</td>
<td>disable SLCD enhance mode</td>
</tr>
<tr>
<td>slcd_bias_voltage_select</td>
<td>select SLCD bias voltage</td>
</tr>
<tr>
<td>slcd_duty_select</td>
<td>select SLCD duty</td>
</tr>
<tr>
<td>slcd_clock_config</td>
<td>configure SLCD input clock</td>
</tr>
<tr>
<td>slcd_blink_mode_config</td>
<td>configure SLCD blink mode</td>
</tr>
<tr>
<td>slcd_contrast_ratio_config</td>
<td>configure SLCD contrast ratio</td>
</tr>
<tr>
<td>slcd_dead_time_config</td>
<td>configure SLCD dead time duration</td>
</tr>
<tr>
<td>slcd_pulse_on_duration_config</td>
<td>configure SLCD pulse on duration</td>
</tr>
<tr>
<td>slcd_com_seg_remap</td>
<td>select SLCD common/segment pad</td>
</tr>
<tr>
<td>slcd_voltage_source_select</td>
<td>select SLCD voltage source</td>
</tr>
<tr>
<td>slcd_high_drive_config</td>
<td>enable or disable permanent high drive</td>
</tr>
<tr>
<td>slcd_data_register_write</td>
<td>write SLCD display data registers</td>
</tr>
<tr>
<td>slcd_data_update_request</td>
<td>update SLCD data request</td>
</tr>
<tr>
<td>slcd_flag_get</td>
<td>get SLCD status flag</td>
</tr>
<tr>
<td>slcd_flag_clear</td>
<td>clear SLCD flag</td>
</tr>
<tr>
<td>slcd_interrupt_enable</td>
<td>enable SLCD interrupt</td>
</tr>
<tr>
<td>slcd_interrupt_disable</td>
<td>disable SLCD interrupt</td>
</tr>
<tr>
<td>slcd_interrupt_flag_get</td>
<td>get SLCD interrupt flag</td>
</tr>
<tr>
<td>slcd_interrupt_flag_clear</td>
<td>clear SLCD interrupt flag</td>
</tr>
</tbody>
</table>
Enum slcd_data_register_enum

Table 3-621. Enum slcd_data_register_enum

<table>
<thead>
<tr>
<th>Member name</th>
<th>Function description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SLCD_DATA_REG0</td>
<td>SLCD display data register 0</td>
</tr>
<tr>
<td>SLCD_DATA_REG1</td>
<td>SLCD display data register 1</td>
</tr>
<tr>
<td>SLCD_DATA_REG2</td>
<td>SLCD display data register 2</td>
</tr>
<tr>
<td>SLCD_DATA_REG3</td>
<td>SLCD display data register 3</td>
</tr>
<tr>
<td>SLCD_DATA_REG4</td>
<td>SLCD display data register 4</td>
</tr>
<tr>
<td>SLCD_DATA_REG5</td>
<td>SLCD display data register 5</td>
</tr>
<tr>
<td>SLCD_DATA_REG6</td>
<td>SLCD display data register 6</td>
</tr>
<tr>
<td>SLCD_DATA_REG7</td>
<td>SLCD display data register 7</td>
</tr>
</tbody>
</table>

slcd_deinit

The description of slcd_deinit is shown as below:

Table 3-622. Function slcd_deinit

<table>
<thead>
<tr>
<th>Function name</th>
<th>slcd_deinit</th>
</tr>
</thead>
<tbody>
<tr>
<td>Function prototype</td>
<td>void slcd_deinit(void);</td>
</tr>
<tr>
<td>Function descriptions</td>
<td>SLCD reset interface</td>
</tr>
<tr>
<td>Precondition</td>
<td>-</td>
</tr>
<tr>
<td>Input parameter(in)</td>
<td>-</td>
</tr>
<tr>
<td>Output parameter(out)</td>
<td>-</td>
</tr>
<tr>
<td>Return value</td>
<td>-</td>
</tr>
</tbody>
</table>

Example:

/* reset the SLCD */
slcd_deinit();

slcd_enable

The description of slcd_enable is shown as below:

Table 3-623. Function slcd_enable

<table>
<thead>
<tr>
<th>Function name</th>
<th>slcd_enable</th>
</tr>
</thead>
<tbody>
<tr>
<td>Function prototype</td>
<td>void slcd_enable(void);</td>
</tr>
<tr>
<td>Function descriptions</td>
<td>enable SLCD interface</td>
</tr>
<tr>
<td>Precondition</td>
<td>-</td>
</tr>
<tr>
<td>Input parameter(in)</td>
<td>-</td>
</tr>
</tbody>
</table>
Example:

/* enable SLCD */
slcd_enable();

**slcd_disable**

The description of `slcd_disable` is shown as below:

**Table 3-624. Function `slcd_disable`**

<table>
<thead>
<tr>
<th>Function name</th>
<th>slcd_disable</th>
</tr>
</thead>
<tbody>
<tr>
<td>Function prototype</td>
<td><code>void slcd_disable(void);</code></td>
</tr>
<tr>
<td>Function descriptions</td>
<td>disable SLCD interface</td>
</tr>
<tr>
<td>Precondition</td>
<td>-</td>
</tr>
</tbody>
</table>

| Input parameter(in) | - |
| Output parameter(out) | - |

| Return value | - |

Example:

/* disable SLCD */
slcd_disable();

**slcd_init**

The description of `slcd_init` is shown as below:

**Table 3-625. Function `slcd_init`**

<table>
<thead>
<tr>
<th>Function name</th>
<th>slcd_init</th>
</tr>
</thead>
<tbody>
<tr>
<td>Function prototype</td>
<td><code>void slcd_init(uint32_t prescaler, uint32_t divider, uint32_t duty, uint32_t bias);</code></td>
</tr>
<tr>
<td>Function descriptions</td>
<td>initialize SLCD interface</td>
</tr>
<tr>
<td>Precondition</td>
<td>-</td>
</tr>
</tbody>
</table>

<p>| Input parameter(in) | - |
| prescaler | the SLCD prescaler |
| SLCD_PRESCALER_1 | $f_{PSC} = f_{in,ck}$ |
| SLCD_PRESCALER_2 | $f_{PSC} = f_{in,ck}/2$ |</p>
<table>
<thead>
<tr>
<th>SLCD_PRESCALER_4</th>
<th>( f_{\text{PSC}} = \frac{f_{\text{in,clk}}}{4} )</th>
</tr>
</thead>
<tbody>
<tr>
<td>SLCD_PRESCALER_8</td>
<td>( f_{\text{PSC}} = \frac{f_{\text{in,clk}}}{8} )</td>
</tr>
<tr>
<td>SLCD_PRESCALER_1 6</td>
<td>( f_{\text{PSC}} = \frac{f_{\text{in,clk}}}{4} )</td>
</tr>
<tr>
<td>SLCD_PRESCALER_3 2</td>
<td>( f_{\text{PSC}} = \frac{f_{\text{in,clk}}}{32} )</td>
</tr>
<tr>
<td>SLCD_PRESCALER_6 4</td>
<td>( f_{\text{PSC}} = \frac{f_{\text{in,clk}}}{64} )</td>
</tr>
<tr>
<td>SLCD_PRESCALER_1 28</td>
<td>( f_{\text{PSC}} = \frac{f_{\text{in,clk}}}{128} )</td>
</tr>
<tr>
<td>SLCD_PRESCALER_2 56</td>
<td>( f_{\text{PSC}} = \frac{f_{\text{in,clk}}}{256} )</td>
</tr>
<tr>
<td>SLCD_PRESCALER_5 12</td>
<td>( f_{\text{PSC}} = \frac{f_{\text{in,clk}}}{512} )</td>
</tr>
<tr>
<td>SLCD_PRESCALER_1 024</td>
<td>( f_{\text{PSC}} = \frac{f_{\text{in,clk}}}{1024} )</td>
</tr>
<tr>
<td>SLCD_PRESCALER_2 048</td>
<td>( f_{\text{PSC}} = \frac{f_{\text{in,clk}}}{2048} )</td>
</tr>
<tr>
<td>SLCD_PRESCALER_4 096</td>
<td>( f_{\text{PSC}} = \frac{f_{\text{in,clk}}}{4096} )</td>
</tr>
<tr>
<td>SLCD_PRESCALER_8 192</td>
<td>( f_{\text{PSC}} = \frac{f_{\text{in,clk}}}{8192} )</td>
</tr>
<tr>
<td>SLCD_PRESCALER_1 6384</td>
<td>( f_{\text{PSC}} = \frac{f_{\text{in,clk}}}{16384} )</td>
</tr>
<tr>
<td>SLCD_PRESCALER_3 2768</td>
<td>( f_{\text{PSC}} = \frac{f_{\text{in,clk}}}{32768} )</td>
</tr>
</tbody>
</table>

**Input parameter (in)**

<table>
<thead>
<tr>
<th>divider</th>
<th>the SLCD divider</th>
</tr>
</thead>
<tbody>
<tr>
<td>SLCD_DIVIDER_16</td>
<td>( f_{\text{SLCD}} = \frac{f_{\text{PSC}}}{16} )</td>
</tr>
<tr>
<td>SLCD_DIVIDER_17</td>
<td>( f_{\text{SLCD}} = \frac{f_{\text{PSC}}}{17} )</td>
</tr>
<tr>
<td>SLCD_DIVIDER_18</td>
<td>( f_{\text{SLCD}} = \frac{f_{\text{PSC}}}{18} )</td>
</tr>
<tr>
<td>SLCD_DIVIDER_19</td>
<td>( f_{\text{SLCD}} = \frac{f_{\text{PSC}}}{19} )</td>
</tr>
<tr>
<td>SLCD_DIVIDER_20</td>
<td>( f_{\text{SLCD}} = \frac{f_{\text{PSC}}}{20} )</td>
</tr>
<tr>
<td>SLCD_DIVIDER_21</td>
<td>( f_{\text{SLCD}} = \frac{f_{\text{PSC}}}{21} )</td>
</tr>
<tr>
<td>SLCD_DIVIDER_22</td>
<td>( f_{\text{SLCD}} = \frac{f_{\text{PSC}}}{22} )</td>
</tr>
<tr>
<td>SLCD_DIVIDER_23</td>
<td>( f_{\text{SLCD}} = \frac{f_{\text{PSC}}}{23} )</td>
</tr>
<tr>
<td>SLCD_DIVIDER_24</td>
<td>( f_{\text{SLCD}} = \frac{f_{\text{PSC}}}{24} )</td>
</tr>
<tr>
<td>SLCD_DIVIDER_25</td>
<td>( f_{\text{SLCD}} = \frac{f_{\text{PSC}}}{25} )</td>
</tr>
<tr>
<td>SLCD_DIVIDER_26</td>
<td>( f_{\text{SLCD}} = \frac{f_{\text{PSC}}}{26} )</td>
</tr>
<tr>
<td>SLCD_DIVIDER_27</td>
<td>( f_{\text{SLCD}} = \frac{f_{\text{PSC}}}{27} )</td>
</tr>
<tr>
<td>SLCD_DIVIDER_28</td>
<td>( f_{\text{SLCD}} = \frac{f_{\text{PSC}}}{28} )</td>
</tr>
<tr>
<td>SLCD_DIVIDER_29</td>
<td>( f_{\text{SLCD}} = \frac{f_{\text{PSC}}}{29} )</td>
</tr>
</tbody>
</table>
### Input parameter (in)

<table>
<thead>
<tr>
<th>duty</th>
<th>the SLCD duty</th>
</tr>
</thead>
<tbody>
<tr>
<td>SLCD_DUTY_STATIC</td>
<td>static duty</td>
</tr>
<tr>
<td>SLCD_DUTY_1_2</td>
<td>1/2 duty</td>
</tr>
<tr>
<td>SLCD_DUTY_1_3</td>
<td>1/3 duty</td>
</tr>
<tr>
<td>SLCD_DUTY_1_4</td>
<td>1/4 duty</td>
</tr>
<tr>
<td>SLCD_DUTY_1_6</td>
<td>1/6 duty</td>
</tr>
<tr>
<td>SLCD_DUTY_1_8</td>
<td>1/8 duty</td>
</tr>
</tbody>
</table>

### Input parameter (in)

<table>
<thead>
<tr>
<th>bias</th>
<th>the SLCD voltage bias</th>
</tr>
</thead>
<tbody>
<tr>
<td>SLCD_BIAS_1_4</td>
<td>1/4 voltage bias</td>
</tr>
<tr>
<td>SLCD_BIAS_1_2</td>
<td>1/2 voltage bias</td>
</tr>
<tr>
<td>SLCD_BIAS_1_3</td>
<td>1/3 voltage bias</td>
</tr>
</tbody>
</table>

### Output parameter (out)

- 

### Return value

- 

Example:

```c
/* initialize SLCD interface */

slcd_init(SLCD_PRESCALER_16, SLCD_DIVIDER_20, SLCD_DUTY_1_4, SLCD_BIAS_1_4);
```

**slcd_enhance_mode_enable**

The description of slcd_enhance_mode_enable is shown as below:

### Table 3-626. Function slcd_enhance_mode_enable

<table>
<thead>
<tr>
<th>Function name</th>
<th>slcd_enhance_mode_enable</th>
</tr>
</thead>
<tbody>
<tr>
<td>Function prototype</td>
<td>void slcd_enhance_mode_enable(void);</td>
</tr>
<tr>
<td>Function descriptions</td>
<td>enable SLCD enhance mode</td>
</tr>
<tr>
<td>Precondition</td>
<td>-</td>
</tr>
</tbody>
</table>

### Input parameter (in)

- 

### Output parameter (out)

- 

### Return value

- 

Example:

```c
/* enable SLCD enhance mode */
```
slcd_enhance_mode_enable();

slcd_enhance_mode_disable

The description of slcd_enhance_mode_disable is shown as below:

Table 3-627. Function slcd_enhance_mode_disable

<table>
<thead>
<tr>
<th>Function name</th>
<th>slcd_enhance_mode_disable</th>
</tr>
</thead>
<tbody>
<tr>
<td>Function prototype</td>
<td>void slcd_enhance_mode_disable(void);</td>
</tr>
<tr>
<td>Function descriptions</td>
<td>disable SLCD enhance mode</td>
</tr>
<tr>
<td>Precondition</td>
<td>-</td>
</tr>
<tr>
<td>Input parameter(in)</td>
<td>-</td>
</tr>
<tr>
<td>Output parameter(out)</td>
<td>-</td>
</tr>
<tr>
<td>Return value</td>
<td>-</td>
</tr>
</tbody>
</table>

Example:

/* disable SLCD enhance mode */
slcd_enhance_mode_disable();

slcd_bias_voltage_select

The description of slcd_bias_voltage_select is shown as below:

Table 3-628. Function slcd_bias_voltage_select

<table>
<thead>
<tr>
<th>Function name</th>
<th>slcd_bias_voltage_select</th>
</tr>
</thead>
<tbody>
<tr>
<td>Function prototype</td>
<td>void slcd_bias_voltage_select(uint32_t bias_voltage);</td>
</tr>
<tr>
<td>Function descriptions</td>
<td>select SLCD bias voltage</td>
</tr>
<tr>
<td>Precondition</td>
<td>-</td>
</tr>
<tr>
<td>Input parameter(in)</td>
<td>bias_voltage the SLCD voltage bias</td>
</tr>
<tr>
<td></td>
<td>SLCD_BIAS_1_4 1/4 voltage bias</td>
</tr>
<tr>
<td></td>
<td>SLCD_BIAS_1_2 1/2 voltage bias</td>
</tr>
<tr>
<td></td>
<td>SLCD_BIAS_1_3 1/3 voltage bias</td>
</tr>
<tr>
<td>Output parameter(out)</td>
<td>-</td>
</tr>
<tr>
<td>Return value</td>
<td>-</td>
</tr>
</tbody>
</table>

Example:

/* set the SLCD 1/4 bias voltage */
slcd_bias_voltage_select(SLCD_BIAS_1_4);
slcd_duty_select

The description of slcd_duty_select is shown as below:

Table 3-629. Function slcd_duty_select

<table>
<thead>
<tr>
<th>Function name</th>
<th>slcd_duty_select</th>
</tr>
</thead>
<tbody>
<tr>
<td>Function prototype</td>
<td>void slcd_duty_select(uint32_t duty);</td>
</tr>
<tr>
<td>Function descriptions</td>
<td>select SLCD duty</td>
</tr>
<tr>
<td>Precondition</td>
<td>-</td>
</tr>
</tbody>
</table>

Input parameter(in)

<table>
<thead>
<tr>
<th>duty</th>
<th>duty select</th>
</tr>
</thead>
<tbody>
<tr>
<td>SLCD_DUTY_STATIC</td>
<td>static duty</td>
</tr>
<tr>
<td>SLCD_DUTY_1_2</td>
<td>1/2 duty</td>
</tr>
<tr>
<td>SLCD_DUTY_1_3</td>
<td>1/3 duty</td>
</tr>
<tr>
<td>SLCD_DUTY_1_4</td>
<td>1/4 duty</td>
</tr>
<tr>
<td>SLCD_DUTY_1_6</td>
<td>1/6 duty</td>
</tr>
<tr>
<td>SLCD_DUTY_1_8</td>
<td>1/8 duty</td>
</tr>
</tbody>
</table>

Output parameter(out)

- -

Return value

- -

Example:

/* select SLCD duty */

slcd_duty_select(SLCD_DUTY_1_2);

slcd_clock_config

The description of slcd_clock_config is shown as below:

Table 3-630. Function slcd_clock_config

<table>
<thead>
<tr>
<th>Function name</th>
<th>slcd_clock_config</th>
</tr>
</thead>
<tbody>
<tr>
<td>Function prototype</td>
<td>void slcd_clock_config(uint32_t prescaler,uint32_t divider);</td>
</tr>
<tr>
<td>Function descriptions</td>
<td>configure SLCD input clock</td>
</tr>
<tr>
<td>Precondition</td>
<td>-</td>
</tr>
</tbody>
</table>

Input parameter(in)

<table>
<thead>
<tr>
<th>prescaler</th>
<th>the prescaler factor</th>
</tr>
</thead>
<tbody>
<tr>
<td>SLCD_PRESCALER_1</td>
<td>fPSC = fIN_clk</td>
</tr>
<tr>
<td>SLCD_PRESCALER_2</td>
<td>fPSC = fIN_div/2</td>
</tr>
<tr>
<td>SLCD_PRESCALER_4</td>
<td>fPSC = fIN_div/4</td>
</tr>
<tr>
<td>SLCD_PRESCALER_8</td>
<td>fPSC = fIN_div/8</td>
</tr>
<tr>
<td>SLCD_PRESCALER_1 6</td>
<td>fPSC = fIN_div/4</td>
</tr>
<tr>
<td>SLCD_PRESCALER_3</td>
<td>fPSC = fIN_div/32</td>
</tr>
<tr>
<td>SLCD_PRESCALER_6</td>
<td>( f_{PSC} = f_{in.clk}/64 )</td>
</tr>
<tr>
<td>SLCD_PRESCALER_1</td>
<td>( f_{PSC} = f_{in.clk}/128 )</td>
</tr>
<tr>
<td>SLCD_PRESCALER_2</td>
<td>( f_{PSC} = f_{in.clk}/256 )</td>
</tr>
<tr>
<td>SLCD_PRESCALER_5</td>
<td>( f_{PSC} = f_{in.clk}/512 )</td>
</tr>
<tr>
<td>SLCD_PRESCALER_1</td>
<td>( f_{PSC} = f_{in.clk}/1024 )</td>
</tr>
<tr>
<td>SLCD_PRESCALER_2</td>
<td>( f_{PSC} = f_{in.clk}/2048 )</td>
</tr>
<tr>
<td>SLCD_PRESCALER_4</td>
<td>( f_{PSC} = f_{in.clk}/4096 )</td>
</tr>
<tr>
<td>SLCD_PRESCALER_8</td>
<td>( f_{PSC} = f_{in.clk}/8192 )</td>
</tr>
<tr>
<td>SLCD_PRESCALER_1</td>
<td>( f_{PSC} = f_{in.clk}/16384 )</td>
</tr>
<tr>
<td>SLCD_PRESCALER_3</td>
<td>( f_{PSC} = f_{in.clk}/32768 )</td>
</tr>
</tbody>
</table>

**Input parameter(in)**

<table>
<thead>
<tr>
<th>divider</th>
<th>the divider factor</th>
</tr>
</thead>
<tbody>
<tr>
<td>SLCD_DIVIDER_16</td>
<td>( f_{SLCD} = f_{PSC}/16 )</td>
</tr>
<tr>
<td>SLCD_DIVIDER_17</td>
<td>( f_{SLCD} = f_{PSC}/17 )</td>
</tr>
<tr>
<td>SLCD_DIVIDER_18</td>
<td>( f_{SLCD} = f_{PSC}/18 )</td>
</tr>
<tr>
<td>SLCD_DIVIDER_19</td>
<td>( f_{SLCD} = f_{PSC}/19 )</td>
</tr>
<tr>
<td>SLCD_DIVIDER_20</td>
<td>( f_{SLCD} = f_{PSC}/20 )</td>
</tr>
<tr>
<td>SLCD_DIVIDER_21</td>
<td>( f_{SLCD} = f_{PSC}/21 )</td>
</tr>
<tr>
<td>SLCD_DIVIDER_22</td>
<td>( f_{SLCD} = f_{PSC}/22 )</td>
</tr>
<tr>
<td>SLCD_DIVIDER_23</td>
<td>( f_{SLCD} = f_{PSC}/23 )</td>
</tr>
<tr>
<td>SLCD_DIVIDER_24</td>
<td>( f_{SLCD} = f_{PSC}/24 )</td>
</tr>
<tr>
<td>SLCD_DIVIDER_25</td>
<td>( f_{SLCD} = f_{PSC}/25 )</td>
</tr>
<tr>
<td>SLCD_DIVIDER_26</td>
<td>( f_{SLCD} = f_{PSC}/26 )</td>
</tr>
<tr>
<td>SLCD_DIVIDER_27</td>
<td>( f_{SLCD} = f_{PSC}/27 )</td>
</tr>
<tr>
<td>SLCD_DIVIDER_28</td>
<td>( f_{SLCD} = f_{PSC}/28 )</td>
</tr>
<tr>
<td>SLCD_DIVIDER_29</td>
<td>( f_{SLCD} = f_{PSC}/29 )</td>
</tr>
<tr>
<td>SLCD_DIVIDER_30</td>
<td>( f_{SLCD} = f_{PSC}/30 )</td>
</tr>
<tr>
<td>SLCD_DIVIDER_31</td>
<td>( f_{SLCD} = f_{PSC}/31 )</td>
</tr>
</tbody>
</table>

**Output parameter(out)**

- -

**Return value**
Example:

/* configure the prescaler and the divider of SLCD clock */

slcd_clock_config(SLCD_PRESCALER_4, SLCD_DIVIDER_19);

**slcd_blink_mode_config**

The description of `slcd_blink_mode_config` is shown as below:

**Table 3-631. Function slcd_blink_mode_config**

<table>
<thead>
<tr>
<th>Function name</th>
<th>slcd_blink_mode_config</th>
</tr>
</thead>
<tbody>
<tr>
<td>Function prototype</td>
<td>void slcd_blink_mode_config(uint32_t mode, uint32_t blink_divider);</td>
</tr>
<tr>
<td>Function descriptions</td>
<td>configure SLCD blink mode</td>
</tr>
<tr>
<td>Precondition</td>
<td>-</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Input parameter(in)</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>mode</strong></td>
</tr>
<tr>
<td>SLCD_BLINKMODE_OFF</td>
</tr>
<tr>
<td>SLCD_BLINKMODE_SEG0_COM0</td>
</tr>
<tr>
<td>SLCD_BLINKMODE_SEG0_ALLCOM</td>
</tr>
<tr>
<td>SLCD_BLINKMODE_ALLSEG_ALLCOM</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Input parameter(in)</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>divider</strong></td>
</tr>
<tr>
<td>SLCD_BLINK_FREQUENCY_DIV8</td>
</tr>
<tr>
<td>SLCD_BLINK_FREQUENCY_DIV16</td>
</tr>
<tr>
<td>SLCD_BLINK_FREQUENCY_DIV32</td>
</tr>
<tr>
<td>SLCD_BLINK_FREQUENCY_DIV64</td>
</tr>
<tr>
<td>SLCD_BLINK_FREQUENCY_DIV128</td>
</tr>
<tr>
<td>SLCD_BLINK_FREQUENCY_DIV256</td>
</tr>
<tr>
<td>SLCD_BLINK_FREQUENCY_DIV512</td>
</tr>
<tr>
<td>SLCD_BLINK_FREQUENCY_DIV1024</td>
</tr>
</tbody>
</table>
Example:

/* configure SLCD blink mode */
slcd_blink_mode_config(SLCD_BLINKMODE_SEG0_COM0,
SLCD_BLINK_FREQUENCY_DIV8);

**slcd_contrast_ratio_config**

The description of `slcd_contrast_ratio_config` is shown as below:

**Table 3-632. Function slcd_contrast_ratio_config**

<table>
<thead>
<tr>
<th>Function name</th>
<th>slcd_contrast_ratio_config</th>
</tr>
</thead>
<tbody>
<tr>
<td>Function prototype</td>
<td>void slcd_contrast_ratio_config(uint32_t contrast_ratio);</td>
</tr>
<tr>
<td>Function descriptions</td>
<td>configure SLCD contrast ratio</td>
</tr>
<tr>
<td>Precondition</td>
<td>-</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Input parameter(in)</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>contrast_ratio</td>
<td>specify the VSLCD voltage</td>
</tr>
<tr>
<td>SLCD_CONTRAST_LEVEL_0</td>
<td>maximum SLCD Voltage = 2.65V</td>
</tr>
<tr>
<td>SLCD_CONTRAST_LEVEL_1</td>
<td>maximum SLCD Voltage = 2.80V</td>
</tr>
<tr>
<td>SLCD_CONTRAST_LEVEL_2</td>
<td>maximum SLCD Voltage = 2.92V</td>
</tr>
<tr>
<td>SLCD_CONTRAST_LEVEL_3</td>
<td>maximum SLCD Voltage = 3.08V</td>
</tr>
<tr>
<td>SLCD_CONTRAST_LEVEL_4</td>
<td>maximum SLCD Voltage = 3.23V</td>
</tr>
<tr>
<td>SLCD_CONTRAST_LEVEL_5</td>
<td>maximum SLCD Voltage = 3.37V</td>
</tr>
<tr>
<td>SLCD_CONTRAST_LEVEL_6</td>
<td>maximum SLCD Voltage = 3.52V</td>
</tr>
<tr>
<td>SLCD_CONTRAST_LEVEL_7</td>
<td>maximum SLCD Voltage = 3.67V</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Output parameter(out)</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>-</td>
<td>-</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Return value</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>-</td>
<td>-</td>
</tr>
</tbody>
</table>

Example:
/* configure SLCD contrast ratio */

slcd_contrast_ratio_config(SLCD_CONTRAST_LEVEL_0);

**slcd_dead_time_config**

The description of `slcd_dead_time_config` is shown as below:

<table>
<thead>
<tr>
<th>Table 3-633. Function slcd_dead_time_config</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Function name</strong></td>
</tr>
<tr>
<td><strong>Function prototype</strong></td>
</tr>
<tr>
<td><strong>Function descriptions</strong></td>
</tr>
<tr>
<td><strong>Precondition</strong></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th><strong>Input parameter</strong>&lt;br&gt;<strong>dead_time</strong></th>
<th>the length of the dead time between frames</th>
</tr>
</thead>
<tbody>
<tr>
<td>SLCD_DEADTIME_PERIOD_0</td>
<td>no dead time</td>
</tr>
<tr>
<td>SLCD_DEADTIME_PERIOD_1</td>
<td>1 phase inserted between couple of frame</td>
</tr>
<tr>
<td>SLCD_DEADTIME_PERIOD_2</td>
<td>2 phase inserted between couple of frame</td>
</tr>
<tr>
<td>SLCD_DEADTIME_PERIOD_3</td>
<td>3 phase inserted between couple of frame</td>
</tr>
<tr>
<td>SLCD_DEADTIME_PERIOD_4</td>
<td>4 phase inserted between couple of frame</td>
</tr>
<tr>
<td>SLCD_DEADTIME_PERIOD_5</td>
<td>5 phase inserted between couple of frame</td>
</tr>
<tr>
<td>SLCD_DEADTIME_PERIOD_6</td>
<td>6 phase inserted between couple of frame</td>
</tr>
<tr>
<td>SLCD_DEADTIME_PERIOD_7</td>
<td>7 phase inserted between couple of frame</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th><strong>Output parameter</strong>&lt;br&gt;<strong>out</strong></th>
<th>-</th>
</tr>
</thead>
</table>

**Return value**

Example:

/* configure SLCD dead time duration */

slcd_dead_time_config(SLCD_DEADTIME_PERIOD_1);

**slcd_pulse_on_duration_config**

The description of `slcd_pulse_on_duration_config` is shown as below:
### Table 3-634. Function `slcd_pulse_on_duration_config`

<table>
<thead>
<tr>
<th>Function name</th>
<th><code>slcd_pulse_on_duration_config</code></th>
</tr>
</thead>
<tbody>
<tr>
<td>Function prototype</td>
<td><code>void slcd_pulse_on_duration_config(uint32_t duration);</code></td>
</tr>
<tr>
<td>Function descriptions</td>
<td>configure SLCD pulse on duration</td>
</tr>
<tr>
<td>Precondition</td>
<td>-</td>
</tr>
</tbody>
</table>

#### Input parameter (in)

- **duration**
  - the pulse duration in terms of PSC pulses
- **SLCD_PULSEON_DURATION_0**
  - pulse on duration = 0
- **SLCD_PULSEON_DURATION_1**
  - pulse on duration = 1/f_{psc}
- **SLCD_PULSEON_DURATION_2**
  - pulse on duration = 2/f_{psc}
- **SLCD_PULSEON_DURATION_3**
  - pulse on duration = 3/f_{psc}
- **SLCD_PULSEON_DURATION_4**
  - pulse on duration = 4/f_{psc}
- **SLCD_PULSEON_DURATION_5**
  - pulse on duration = 5/f_{psc}
- **SLCD_PULSEON_DURATION_6**
  - pulse on duration = 6/f_{psc}
- **SLCD_PULSEON_DURATION_7**
  - pulse on duration = 7/f_{psc}

#### Output parameter (out)

- -

#### Return value

- -

Example:

```c
/* configure SLCD pulse on duration */
slcd_pulse_on_duration_config(SLCD_PULSEON_DURATION_7);
```

### `slcd_com_seg_remap`

The description of `slcd_com_seg_remap` is shown as below:

### Table 3-635. Function `slcd_com_seg_remap`

<table>
<thead>
<tr>
<th>Function name</th>
<th><code>slcd_com_seg_remap</code></th>
</tr>
</thead>
<tbody>
<tr>
<td>Function prototype</td>
<td><code>void slcd_com_seg_remap(ControlStatus newvalue);</code></td>
</tr>
<tr>
<td>Function descriptions</td>
<td>select SLCD common/segment pad</td>
</tr>
<tr>
<td>Precondition</td>
<td>-</td>
</tr>
</tbody>
</table>

#### Input parameter (in)

<table>
<thead>
<tr>
<th>NewValue</th>
<th>ENABLE or DISABLE</th>
</tr>
</thead>
</table>


**slcd_com_seg_remap**

The description of `slcd_com_seg_remap` is shown as below:

|--------|----------------------------------------|

Output parameter(out)

- -

Return value

- -

**Example:**

```c
/* select SLCD common/segment pad */

slcd_com_seg_remap(ENABLE);
```

**slcd_voltage_source_select**

The description of `slcd_voltage_source_select` is shown as below:

<table>
<thead>
<tr>
<th>Function name</th>
<th>slcd_voltage_source_select</th>
</tr>
</thead>
<tbody>
<tr>
<td>Function prototype</td>
<td>void slcd_voltage_source_select(uint8_t voltage_source);</td>
</tr>
<tr>
<td>Function descriptions</td>
<td>select SLCD voltage source</td>
</tr>
<tr>
<td>Precondition</td>
<td>-</td>
</tr>
</tbody>
</table>

Input parameter(in)

<table>
<thead>
<tr>
<th>voltage_source</th>
<th>the SLCD voltage source</th>
</tr>
</thead>
<tbody>
<tr>
<td>SLCD_VOLTAGE_INTERNAL</td>
<td>internal source</td>
</tr>
<tr>
<td>SLCD_VOLTAGE_EXTERNAL</td>
<td>external source (VSLCD pin)</td>
</tr>
</tbody>
</table>

Output parameter(out)

- -

Return value

- -

**Example:**

```c
/* select SLCD voltage source */

slcd_voltage_source_select(SLCD_VOLTAGE_EXTERNAL);
```

**slcd_high_drive_config**

The description of `slcd_high_drive_config` is shown as below:

<table>
<thead>
<tr>
<th>Function name</th>
<th>slcd_high_drive_config</th>
</tr>
</thead>
<tbody>
<tr>
<td>Function prototype</td>
<td>void slcd_high_drive_config(ControlStatus newvalue);</td>
</tr>
<tr>
<td>Function descriptions</td>
<td>enable or disable permanent high drive</td>
</tr>
<tr>
<td>Precondition</td>
<td>-</td>
</tr>
</tbody>
</table>
### slcd_high_drive_config

<table>
<thead>
<tr>
<th>Input parameter (in)</th>
<th>Output parameter (out)</th>
<th>Return value</th>
</tr>
</thead>
<tbody>
<tr>
<td>NewValue ENABLE or DISABLE</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>ENABLE Permanent high drive enabled</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>DISABLE Permanent high drive disabled</td>
<td>-</td>
<td>-</td>
</tr>
</tbody>
</table>

Example:

```c
/* enable permanent high drive */
slcd_high_drive_config(ENABLE);
```

### slcd_data_register_write

The description of slcd_data_register_write is shown as below:

**Function name**: slcd_data_register_write

**Function prototype**:

```c
void slcd_data_register_write(slcd_data_register_enum register_number,
                              uint32_t data);
```

**Function description**: write SLCD data register

**Precondition**: -

<table>
<thead>
<tr>
<th>Input parameter (in)</th>
<th>Output parameter (out)</th>
<th>Return value</th>
</tr>
</thead>
<tbody>
<tr>
<td>register_number refer to Table 3-621, Enum slcd_data_register_enum</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>SLCD_DATA_REGx(x=0, 1, ..., 7) SLCD_DATAx register</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>data the data write to the register</td>
<td>-</td>
<td>-</td>
</tr>
</tbody>
</table>

Example:

```c
/* write SLCD data register */
slcd_data_register_write(SLCD_DATA_REG0, 0x0000FFFF);
```

### slcd_data_update_request

The description of slcd_data_update_request is shown as below:

**Function name**: slcd_data_update_request

**Table 3-639. Function slcd_data_update_request**

<table>
<thead>
<tr>
<th>Function name</th>
<th>slcd_data_update_request</th>
</tr>
</thead>
</table>

Example:

```c
/* write SLCD data register */
slcd_data_update_request(SLCD_DATA_REG0, 0x0000FFFF);
```
Function prototype

```c
void slcd_data_update_request(void);
```

Function descriptions

update SLCD data request

Precondition

Input parameter(in)

Output parameter(out)

Return value

---

**Example:**

```c
/* update SLCD data request */

slcd_data_update_request();
```

### slcd_flag_get

The description of slcd_flag_get is shown as below:

<table>
<thead>
<tr>
<th>Function name</th>
<th>slcd_flag_get</th>
</tr>
</thead>
<tbody>
<tr>
<td>Function prototype</td>
<td>FlagStatus slcd_flag_get(uint8_t flag);</td>
</tr>
<tr>
<td>Function descriptions</td>
<td>get the SLCD flag</td>
</tr>
<tr>
<td>Precondition</td>
<td>-</td>
</tr>
</tbody>
</table>

**Input parameter(in)**

<table>
<thead>
<tr>
<th>flag</th>
<th>status flag</th>
</tr>
</thead>
<tbody>
<tr>
<td>SLCD_FLAG_ON</td>
<td>SLCD controller on flag</td>
</tr>
<tr>
<td>SLCD_FLAG_SOF</td>
<td>start of frame flag</td>
</tr>
<tr>
<td>SLCD_FLAG_UPR</td>
<td>SLCD data update request flag</td>
</tr>
<tr>
<td>SLCD_FLAG_UPD</td>
<td>update SLCD data done flag</td>
</tr>
<tr>
<td>SLCD_FLAG_VRDY</td>
<td>SLCD voltage ready flag</td>
</tr>
<tr>
<td>SLCD_FLAG_SYN</td>
<td>SLCD CFG register synchronization flag</td>
</tr>
</tbody>
</table>

**Output parameter(out)**

| - | - |

**Return value**

FlagStatus

**Example:**

```c
/* get the SLCD status flag */

slcd_flag_get(SLCD_FLAG_ON);
```

### slcd_flag_clear

The description of slcd_flag_clear is shown as below:
Table 3-641. Function slcd_flag_clear

<table>
<thead>
<tr>
<th>Function name</th>
<th>slcd_flag_clear</th>
</tr>
</thead>
<tbody>
<tr>
<td>Function prototype</td>
<td>void slcd_flag_clear(uint8_t flag);</td>
</tr>
<tr>
<td>Function descriptions</td>
<td>Clear the SLCD status flag</td>
</tr>
<tr>
<td>Precondition</td>
<td>-</td>
</tr>
<tr>
<td>Input parameter(in)</td>
<td></td>
</tr>
<tr>
<td>flag</td>
<td>status flag</td>
</tr>
<tr>
<td>SLCD_FLAG_SOF</td>
<td>start of frame flag</td>
</tr>
<tr>
<td>SLCD_FLAG_UPD</td>
<td>update SLCD data done flag</td>
</tr>
<tr>
<td>Output parameter(out)</td>
<td></td>
</tr>
<tr>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>Return value</td>
<td>-</td>
</tr>
</tbody>
</table>

Example:

/* clear the SLCD status flag */
slcd_flag_clear(SLCD_FLAG_SOF);

slcd_interrupt_enable

The description of slcd_interrupt_enable is shown as below:

Table 3-642. Function slcd_interrupt_enable

<table>
<thead>
<tr>
<th>Function name</th>
<th>slcd_interrupt_enable</th>
</tr>
</thead>
<tbody>
<tr>
<td>Function prototype</td>
<td>void slcd_interrupt_enable(uint32_t interrupt);</td>
</tr>
<tr>
<td>Function descriptions</td>
<td>enable the SLCD interrupt</td>
</tr>
<tr>
<td>Precondition</td>
<td>-</td>
</tr>
<tr>
<td>Input parameter(in)</td>
<td></td>
</tr>
<tr>
<td>interrupt</td>
<td>interrupt source</td>
</tr>
<tr>
<td>SLCD_INT_SOF</td>
<td>start of frame interrupt</td>
</tr>
<tr>
<td>SLCD_INT_UPD</td>
<td>SLCD update done interrupt</td>
</tr>
<tr>
<td>Output parameter(out)</td>
<td></td>
</tr>
<tr>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>Return value</td>
<td>-</td>
</tr>
</tbody>
</table>

Example:

/* enable the start of frame interrupt */
slcd_interrupt_enable(SLCD_INT_SOF);

slcd_interrupt_disable

The description of slcd_interrupt_disable is shown as below:
Table 3-643. Function slcd_interrupt_disable

<table>
<thead>
<tr>
<th>Function name</th>
<th>slcd_interrupt_disable</th>
</tr>
</thead>
<tbody>
<tr>
<td>Function prototype</td>
<td>void slcd_interrupt_disable(uint32_t interrupt);</td>
</tr>
<tr>
<td>Function descriptions</td>
<td>disable the SLCD interrupt</td>
</tr>
<tr>
<td>Precondition</td>
<td>-</td>
</tr>
</tbody>
</table>

**Input parameter (in)**

<table>
<thead>
<tr>
<th>interrupt</th>
<th>interrupt source</th>
</tr>
</thead>
<tbody>
<tr>
<td>SLCD_INT_SOF</td>
<td>start of frame interrupt</td>
</tr>
<tr>
<td>SLCD_INT_UPD</td>
<td>SLCD update done interrupt</td>
</tr>
</tbody>
</table>

**Output parameter (out)**

- -

**Return value**

- -

Example:

/* disable the start of frame interrupt */

slcd_interrupt_disable(SLCD_INT_SOF);

**slcd_interrupt_flag_get**

The description of slcd_interrupt_flag_get is shown as below:

Table 3-644. Function slcd_interrupt_flag_get

<table>
<thead>
<tr>
<th>Function name</th>
<th>slcd_interrupt_flag_get</th>
</tr>
</thead>
<tbody>
<tr>
<td>Function prototype</td>
<td>FlagStatus slcd_interrupt_flag_get(uint8_t interrupt);</td>
</tr>
<tr>
<td>Function descriptions</td>
<td>get the SLCD interrupt flag</td>
</tr>
<tr>
<td>Precondition</td>
<td>-</td>
</tr>
</tbody>
</table>

**Input parameter (in)**

<table>
<thead>
<tr>
<th>interrupt</th>
<th>interrupt source</th>
</tr>
</thead>
<tbody>
<tr>
<td>SLCD_INT_FLAG_SOF</td>
<td>start of frame interrupt</td>
</tr>
<tr>
<td>SLCD_INT_FLAG_UPD</td>
<td>SLCD update done interrupt</td>
</tr>
</tbody>
</table>

**Output parameter (out)**

- -

**Return value**

FlagStatus: SET or RESET

Example:

/* get the SLCD interrupt flag */

slcd_interrupt_flag_get(SLCD_INT_FLAG_SOF);

**slcd_interrupt_flag_clear**

The description of slcd_interrupt_flag_clear is shown as below:
### Table 3-645. Function `slcd_interrupt_flag_clear`

<table>
<thead>
<tr>
<th>Function name</th>
<th>slcd_interrupt_flag_clear</th>
</tr>
</thead>
<tbody>
<tr>
<td>Function prototype</td>
<td>FlagStatus slcd_interrupt_flag_clear(uint8_t interrupt);</td>
</tr>
<tr>
<td>Function descriptions</td>
<td>clear the SLCD interrupt flag</td>
</tr>
<tr>
<td>Precondition</td>
<td>-</td>
</tr>
</tbody>
</table>

#### Input parameter (in)

<table>
<thead>
<tr>
<th>interrupt</th>
<th>interrupt source</th>
</tr>
</thead>
<tbody>
<tr>
<td>SLCD_INT_FLAG_SOF</td>
<td>start of frame interrupt</td>
</tr>
<tr>
<td>SLCD_INT_FLAG_UPD</td>
<td>SLCD update done interrupt</td>
</tr>
</tbody>
</table>

#### Output parameter (out)

- -

**Return value**

- -

Example:

```c
/* clear the SLCD interrupt flag */
slcd_interrupt_flag_clear(SLCD_INT_FLAG_SOF);
```

### 3.22. SPI

The SPI/I2S module can communicate with external devices using the SPI protocol or the I2S audio protocol. The SPI/I2S registers are listed in chapter 3.22.1, the SPI/I2S firmware functions are introduced in chapter 3.22.2.

#### 3.22.1. Descriptions of Peripheral registers

SPI/I2S registers are listed in the table shown as below:

**Table 3-646. SPI/I2S Registers**

<table>
<thead>
<tr>
<th>Registers</th>
<th>Descriptions</th>
</tr>
</thead>
<tbody>
<tr>
<td>SPI_CTL0</td>
<td>SPI control register 0</td>
</tr>
<tr>
<td>SPI_CTL1</td>
<td>SPI control register 1</td>
</tr>
<tr>
<td>SPI_STAT</td>
<td>SPI status register</td>
</tr>
<tr>
<td>SPI_DATA</td>
<td>SPI data register</td>
</tr>
<tr>
<td>SPI_CRCPOLY</td>
<td>SPI CRC polynomial register</td>
</tr>
<tr>
<td>SPI_RCRC</td>
<td>SPI receive CRC register</td>
</tr>
<tr>
<td>SPI_TCRC</td>
<td>SPI transmit CRC register</td>
</tr>
<tr>
<td>SPI_I2SCTL</td>
<td>SPI/I2S control register</td>
</tr>
<tr>
<td>SPI_I2SPSC</td>
<td>SPI/I2S clock prescaler register</td>
</tr>
<tr>
<td>SPI_QCTL</td>
<td>Quad-SPI mode control register</td>
</tr>
</tbody>
</table>
3.22.2. Descriptions of Peripheral functions

SPI/I2S firmware functions are listed in the table shown as below:

Table 3-647. SPI/I2S firmware function

<table>
<thead>
<tr>
<th>Function name</th>
<th>Function description</th>
</tr>
</thead>
<tbody>
<tr>
<td>spi_i2s_deinit</td>
<td>reset SPI and I2S peripheral</td>
</tr>
<tr>
<td>spi_struct_para_init</td>
<td>initialize the parameters of SPI struct</td>
</tr>
<tr>
<td>spi_init</td>
<td>initialize SPI parameter</td>
</tr>
<tr>
<td>spi_enable</td>
<td>enable SPI</td>
</tr>
<tr>
<td>spi_disable</td>
<td>disable SPI</td>
</tr>
<tr>
<td>i2s_init</td>
<td>initialize I2S parameter</td>
</tr>
<tr>
<td>i2s_psc_config</td>
<td>configure I2S prescaler</td>
</tr>
<tr>
<td>i2s_enable</td>
<td>enable I2S</td>
</tr>
<tr>
<td>i2s_disable</td>
<td>disable I2S</td>
</tr>
<tr>
<td>spi_nss_output_enable</td>
<td>enable SPI NSS output function</td>
</tr>
<tr>
<td>spi_nss_output_disable</td>
<td>disable SPI NSS output function</td>
</tr>
<tr>
<td>spi_nss_internal_high</td>
<td>SPI NSS pin high level in software mode</td>
</tr>
<tr>
<td>spi_nss_internal_low</td>
<td>SPI NSS pin low level in software mode</td>
</tr>
<tr>
<td>spi_dma_enable</td>
<td>enable SPI DMA function</td>
</tr>
<tr>
<td>spi_dma_disable</td>
<td>disable SPI DMA function</td>
</tr>
<tr>
<td>spi_i2s_data_frame_format_config</td>
<td>configure SPI/I2S data frame format</td>
</tr>
<tr>
<td>spi_i2s_data_transmit</td>
<td>SPI transmit data</td>
</tr>
<tr>
<td>spi_i2s_data_receive</td>
<td>SPI receive data</td>
</tr>
<tr>
<td>spi_bidirectional_transfer_config</td>
<td>configure SPI bidirectional transfer direction</td>
</tr>
<tr>
<td>spi_crc_polynomial_set</td>
<td>set SPI CRC polynomial</td>
</tr>
<tr>
<td>spi_crc_polynomial_get</td>
<td>get SPI CRC polynomial</td>
</tr>
<tr>
<td>spi_crc_on</td>
<td>turn on SPI CRC function</td>
</tr>
<tr>
<td>spi_crc_off</td>
<td>turn off SPI CRC function</td>
</tr>
<tr>
<td>spi_crc_next</td>
<td>SPI next data is CRC value</td>
</tr>
<tr>
<td>spi_crc_get</td>
<td>get SPI CRC send value or receive value</td>
</tr>
<tr>
<td>spi_i2s_interrupt_enable</td>
<td>enable SPI and I2S interrupt</td>
</tr>
<tr>
<td>spi_i2s_interrupt_disable</td>
<td>disable SPI and I2S interrupt</td>
</tr>
<tr>
<td>spi_i2s_interrupt_flag_get</td>
<td>get SPI and I2S interrupt status</td>
</tr>
<tr>
<td>spi_i2s_flag_get</td>
<td>get SPI and I2S flag status</td>
</tr>
<tr>
<td>spi_crc_error_clear</td>
<td>clear SPI CRC error flag status</td>
</tr>
<tr>
<td>qspi_enable</td>
<td>enable quad wire SPI</td>
</tr>
<tr>
<td>qspi_disable</td>
<td>disable quad wire SPI</td>
</tr>
<tr>
<td>qspi_write_enable</td>
<td>enable quad wire SPI write</td>
</tr>
<tr>
<td>qspi_read_enable</td>
<td>enable quad wire SPI read</td>
</tr>
<tr>
<td>qspi_io23_output_enable</td>
<td>enable quad wire SPI IO2 and SPI IO3 pin output</td>
</tr>
<tr>
<td>qspi_io23_output_disable</td>
<td>disable quad wire SPI IO2 and SPI IO3 pin output</td>
</tr>
</tbody>
</table>
Structure spi_parameter_struct

Table 3-648. spi_parameter_struct

<table>
<thead>
<tr>
<th>Member name</th>
<th>Function description</th>
</tr>
</thead>
<tbody>
<tr>
<td>device_mode</td>
<td>SPI master or slave (SPI_MASTER, SPI_SLAVE)</td>
</tr>
<tr>
<td>trans_mode</td>
<td>SPI transtype (SPI_TRANSMODE_FULLDUPLEX, SPI_TRANSMODE_RECEIVEONLY, SPI_TRANSMODE_BDRECEIVE, SPI_TRANSMODE_BDTRANSMIT)</td>
</tr>
<tr>
<td>frame_size</td>
<td>SPI frame size (SPI_FRAMESIZE_16BIT, SPI_FRAMESIZE_8BIT)</td>
</tr>
<tr>
<td>nss</td>
<td>SPI NSS control by hardware or software (SPI_NSS_SOFT, SPI_NSS_HARD)</td>
</tr>
<tr>
<td>endian</td>
<td>SPI big endian or little endian (SPI_ENDIAN_MSB, SPI_ENDIAN_LSB)</td>
</tr>
<tr>
<td>clock_polarity_phase</td>
<td>SPI clock phase and polarity (SPI_CK_PL_LOW_PH_1EDGE, SPI_CK_PL_HIGH_PH_1EDGE, SPI_CK_PL_LOW_PH_2EDGE, SPI_CK_PL_HIGH_PH_2EDGE)</td>
</tr>
<tr>
<td>prescale</td>
<td>SPI prescale factor (SPI_PSC_n (n=2,4,8,16,32,64,128,256))</td>
</tr>
</tbody>
</table>

spi_i2s_deinit

The description of spi_i2s_deinit is shown as below:

Table 3-649. Function spi_i2s_deinit

<table>
<thead>
<tr>
<th>Function name</th>
<th>spi_i2s_deinit</th>
</tr>
</thead>
<tbody>
<tr>
<td>Function prototype</td>
<td>void spi_i2s_deinit(uint32_t spi_periph);</td>
</tr>
<tr>
<td>Function descriptions</td>
<td>reset SPI and I2S peripheral</td>
</tr>
<tr>
<td>Precondition</td>
<td>-</td>
</tr>
<tr>
<td>The called functions</td>
<td>rcu_periph_reset_enable / rcu_periph_reset_disable</td>
</tr>
</tbody>
</table>

Input parameter(in)

<table>
<thead>
<tr>
<th>spi_periph</th>
<th>SPI/I2S peripheral</th>
</tr>
</thead>
<tbody>
<tr>
<td>SPIx(x=0,1,2)</td>
<td>SPI/I2S peripheral selection</td>
</tr>
</tbody>
</table>

Output parameter(out)

| -                      | - |

Return value

Example:

/* reset SPI0 */

spi_i2s_deinit(SPI0);
spi_struct_para_init

The description of spi_struct_para_init is shown as below:

Table 3-650. Function spi_struct_para_init

<table>
<thead>
<tr>
<th>Function name</th>
<th>spi_struct_para_init</th>
</tr>
</thead>
<tbody>
<tr>
<td>Function prototype</td>
<td>void spi_struct_para_init(spi_parameter_struct* spi_struct);</td>
</tr>
<tr>
<td>Function descriptions</td>
<td>initialize the parameters of SPI struct with the default values</td>
</tr>
<tr>
<td>Precondition</td>
<td>-</td>
</tr>
<tr>
<td>The called functions</td>
<td>-</td>
</tr>
</tbody>
</table>

Input parameter(in)

<table>
<thead>
<tr>
<th>spi_struct</th>
<th>SPI parameter struct, the structure members can refer to members of the structure Table 3-648. spi_parameter_struct</th>
</tr>
</thead>
</table>

Output parameter(out)

- -

Return value

- -

Example:

/* initialize the parameters of SPI struct */

spi_parameter_struct spi_struct;

spi_struct->device_mode = SPI_SLAVE;
spi_struct->trans_mode = SPI_TRANSMODE_FULLDUPLEX;
spi_struct->frame_size = SPI_FRAMESIZE_8BIT;
spi_struct->nss = SPI_NSS_HARD;
spi_struct->clock_polarity_phase = SPI_CK_PL_LOW_PH_1EDGE;
spi_struct->prescale = SPI_PSC_2;
spi_struct_para_init(&spi_struct);

spi_init

The description of spi_init is shown as below:

Table 3-651. Function spi_init

<table>
<thead>
<tr>
<th>Function name</th>
<th>spi_init</th>
</tr>
</thead>
<tbody>
<tr>
<td>Function prototype</td>
<td>void spi_init(uint32_t spi_periph, spi_parameter_struct* spi_struct);</td>
</tr>
<tr>
<td>Function descriptions</td>
<td>initialize SPI peripheral parameter</td>
</tr>
<tr>
<td>Precondition</td>
<td>-</td>
</tr>
<tr>
<td>The called functions</td>
<td>-</td>
</tr>
</tbody>
</table>

Input parameter(in)
**spi_periph**  
SPI peripheral

**SPIx(x=0,1,2)**  
SPI peripheral selection

---

**spi_struct**  
SPI parameter initialization struct, the structure members can refer to members of the structure [Table 3-648. spi_parameter_struct](#).

---

**spi_parameter_struct**

---

**spi_enable**

The description of spi_enable is shown as below:

### Table 3-652. Function spi_enable

<table>
<thead>
<tr>
<th>Function name</th>
<th>spi_enable</th>
</tr>
</thead>
<tbody>
<tr>
<td>Function prototype</td>
<td>void spi_enable(uint32_t spi_periph);</td>
</tr>
<tr>
<td>Function descriptions</td>
<td>enable SPI</td>
</tr>
<tr>
<td>Precondition</td>
<td>-</td>
</tr>
<tr>
<td>The called functions</td>
<td>-</td>
</tr>
</tbody>
</table>

---

**spi_periph**  
SPI peripheral

**SPIx(x=0,1,2)**  
SPI peripheral selection

---

**spi_enable**  
SPI peripheral

**SPIx(x=0,1,2)**  
SPI peripheral selection

---

**Return value**  
-
Example:

/* enable SPI0 */
spi_enable(SPI0);

**spi_disable**

The description of spi_disable is shown as below:

Table 3-653. Function spi_disable

<table>
<thead>
<tr>
<th>Function name</th>
<th>spi_disable</th>
</tr>
</thead>
<tbody>
<tr>
<td>Function prototype</td>
<td>void spi_disable(uint32_t spi_periph);</td>
</tr>
<tr>
<td>Function descriptions</td>
<td>disable SPIx</td>
</tr>
<tr>
<td>Precondition</td>
<td>-</td>
</tr>
<tr>
<td>The called functions</td>
<td>-</td>
</tr>
</tbody>
</table>

**Input parameter(in)**

- spi_periph: SPI peripheral
  - SPIx (x=0,1,2): SPI peripheral selection

**Output parameter(out)**

- -

**Return value**

- -

Example:

/* disable SPI0 */
spi_disable(SPI0);

**i2s_init**

The description of i2s_init is shown as below:

Table 3-654. Function i2s_init

<table>
<thead>
<tr>
<th>Function name</th>
<th>i2s_init</th>
</tr>
</thead>
<tbody>
<tr>
<td>Function prototype</td>
<td>void i2s_init(uint32_t spi_periph,uint32_t mode, uint32_t standard, uint32_t ckpl);</td>
</tr>
<tr>
<td>Function descriptions</td>
<td>initialize I2S peripheral parameter</td>
</tr>
<tr>
<td>Precondition</td>
<td>-</td>
</tr>
<tr>
<td>The called functions</td>
<td>-</td>
</tr>
</tbody>
</table>

**Input parameter(in)**

- spi_periph: I2S peripheral
  - SPIx (x=1,2): I2S peripheral selection

- mode: I2S operation mode
  - I2S_MODE_SLAVE: I2S slave mode
  - I2S_MODE_SLAVE receive mode

- I2S_MODE_SLAVE: I2S slave transmit mode

- -

427
GD32L23x Firmware Library User Guide

I2S MODE SLAVERX  I2S slave receive mode
I2S_MODE_MASTERTX  I2S master transmit mode
I2S_MODE_MASTERRX  I2S master receive mode

Input parameter(in)

<table>
<thead>
<tr>
<th>standard</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>I2S_STD_PHILLIPS</td>
<td>I2S phillips standard</td>
</tr>
<tr>
<td>I2S_STD_MSB</td>
<td>I2S MSB standard</td>
</tr>
<tr>
<td>I2S_STD_LSB</td>
<td>I2S LSB standard</td>
</tr>
<tr>
<td>I2S_STD_PCMSHORT</td>
<td>I2S PCM short standard</td>
</tr>
<tr>
<td>I2S_STD_PCMLONG</td>
<td>I2S PCM long standard</td>
</tr>
</tbody>
</table>

Input parameter(in)

<table>
<thead>
<tr>
<th>ckpl</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>I2S_CKPL_LOW</td>
<td>I2S clock polarity low level</td>
</tr>
<tr>
<td>I2S_CKPL_HIGH</td>
<td>I2S clock polarity high level</td>
</tr>
</tbody>
</table>

Output parameter(out)

- -

Return value

- -

Example:

/* initialize I2S */
i2s_init(SPI1, I2S_MODE_MASTERTX, I2S_STD_PHILLIPS, I2S_CKPL_LOW);

i2s_psc_config

The description of i2s_psc_config is shown as below:

Table 3-655. Function i2s_psc_config

<table>
<thead>
<tr>
<th>Function name</th>
<th>i2s_psc_config</th>
</tr>
</thead>
<tbody>
<tr>
<td>Function prototype</td>
<td>void i2s_psc_config(uint32_t spi_periph, uint32_t audiosample, uint32_t frameformat, uint32_t mckout);</td>
</tr>
<tr>
<td>Function descriptions</td>
<td>configure I2S prescaler</td>
</tr>
<tr>
<td>Precondition</td>
<td>-</td>
</tr>
<tr>
<td>The called functions</td>
<td>rcu_clock_freq_get</td>
</tr>
<tr>
<td>Input parameter(in)</td>
<td></td>
</tr>
<tr>
<td>spi_periph</td>
<td>I2S peripheral</td>
</tr>
<tr>
<td>SPIx(x=1,2)</td>
<td>I2S peripheral selection</td>
</tr>
<tr>
<td>Input parameter(in)</td>
<td></td>
</tr>
<tr>
<td>audiosample</td>
<td>I2S audio sample rate</td>
</tr>
<tr>
<td>I2S_AUDIOSAMPLE_8K</td>
<td>audio sample rate is 8KHz</td>
</tr>
</tbody>
</table>


**I2S_AUDIOSAMPLE_1**
- **1K**: audio sample rate is 11KHz

**I2S_AUDIOSAMPLE_1**
- **6K**: audio sample rate is 16KHz

**I2S_AUDIOSAMPLE_2**
- **2K**: audio sample rate is 22KHz

**I2S_AUDIOSAMPLE_3**
- **2K**: audio sample rate is 32KHz

**I2S_AUDIOSAMPLE_4**
- **4K**: audio sample rate is 44KHz

**I2S_AUDIOSAMPLE_4**
- **8K**: audio sample rate is 48KHz

**I2S_AUDIOSAMPLE_9**
- **6K**: audio sample rate is 96KHz

**I2S_AUDIOSAMPLE_1**
- **92K**: audio sample rate is 192KHz

---

### Input parameter (in)

<table>
<thead>
<tr>
<th>frameformat</th>
<th>I2S data length and channel length</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>I2S_FRAMEFORMAT_DT16B_CH16B</strong></td>
<td>I2S data length is 16 bit and channel length is 16 bit</td>
</tr>
<tr>
<td><strong>I2S_FRAMEFORMAT_DT16B_CH32B</strong></td>
<td>I2S data length is 16 bit and channel length is 32 bit</td>
</tr>
<tr>
<td><strong>I2S_FRAMEFORMAT_DT24B_CH32B</strong></td>
<td>I2S data length is 24 bit and channel length is 32 bit</td>
</tr>
<tr>
<td><strong>I2S_FRAMEFORMAT_DT32B_CH32B</strong></td>
<td>I2S data length is 32 bit and channel length is 32 bit</td>
</tr>
</tbody>
</table>

---

### Input parameter (in)

<table>
<thead>
<tr>
<th>mckout</th>
<th>I2S master clock output</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>I2S_MCKOUT_ENABLE</strong></td>
<td>I2S master clock output enable</td>
</tr>
<tr>
<td><strong>I2S_MCKOUT_DISABLE</strong></td>
<td>I2S master clock output disable</td>
</tr>
</tbody>
</table>

---

### Output parameter (out)

| - | - |

---

### Return value

| - | - |

---

Example:

```c
/** configure I2S1 prescaler */
i2s_psc_config(SPI1,  I2S_AUDIOSAMPLE_44K,  I2S_FRAMEFORMAT_DT16B_CH16B,  I2S_MCKOUT_DISABLE);
```
**i2s_enable**

The description of `i2s_enable` is shown as below:

**Table 3-656. Function i2s_enable**

<table>
<thead>
<tr>
<th>Function name</th>
<th>i2s_enable</th>
</tr>
</thead>
<tbody>
<tr>
<td>Function prototype</td>
<td>void i2s_enable(uint32_t spi_periph);</td>
</tr>
<tr>
<td>Function descriptions</td>
<td>enable I2S</td>
</tr>
<tr>
<td>Precondition</td>
<td>-</td>
</tr>
<tr>
<td>The called functions</td>
<td>-</td>
</tr>
<tr>
<td>Input parameter (in)</td>
<td>-</td>
</tr>
<tr>
<td>spi_periph</td>
<td>I2S peripheral</td>
</tr>
<tr>
<td>SPIx(x=1,2)</td>
<td>I2S peripheral selection</td>
</tr>
<tr>
<td>Output parameter (out)</td>
<td>-</td>
</tr>
<tr>
<td>Return value</td>
<td>-</td>
</tr>
</tbody>
</table>

Example:

```
/* enable I2S1*/

i2s_enable(SPI1);
```

**i2s_disable**

The description of `i2s_disable` is shown as below:

**Table 3-657. Function i2s_disable**

<table>
<thead>
<tr>
<th>Function name</th>
<th>i2s_disable</th>
</tr>
</thead>
<tbody>
<tr>
<td>Function prototype</td>
<td>void i2s_disable(uint32_t spi_periph);</td>
</tr>
<tr>
<td>Function descriptions</td>
<td>disable I2S</td>
</tr>
<tr>
<td>Precondition</td>
<td>-</td>
</tr>
<tr>
<td>The called functions</td>
<td>-</td>
</tr>
<tr>
<td>Input parameter (in)</td>
<td>-</td>
</tr>
<tr>
<td>spi_periph</td>
<td>I2S peripheral</td>
</tr>
<tr>
<td>SPIx(x=1,2)</td>
<td>I2S peripheral selection</td>
</tr>
<tr>
<td>Output parameter (out)</td>
<td>-</td>
</tr>
<tr>
<td>Return value</td>
<td>-</td>
</tr>
</tbody>
</table>

Example:

```
/* disable I2S1*/

i2s_disable(SPI1);
```
spi_nss_output_enable

The description of spi_nss_output_enable is shown as below:

Table 3-658. Function spi_nss_output_enable

<table>
<thead>
<tr>
<th>Function name</th>
<th>spi_nss_output_enable</th>
</tr>
</thead>
<tbody>
<tr>
<td>Function prototype</td>
<td>void spi_nss_output_enable(uint32_t spi_periph);</td>
</tr>
<tr>
<td>Function descriptions</td>
<td>enable SPI NSS output function</td>
</tr>
<tr>
<td>Precondition</td>
<td>-</td>
</tr>
<tr>
<td>The called functions</td>
<td>-</td>
</tr>
</tbody>
</table>

Input parameter (in)

<table>
<thead>
<tr>
<th>spi_periph</th>
<th>SPI peripheral</th>
</tr>
</thead>
<tbody>
<tr>
<td>SPIx(x=0,1,2)</td>
<td>SPI peripheral selection</td>
</tr>
</tbody>
</table>

Output parameter (out)

- -

Return value

- -

Example:

/* enable SPI0 NSS output */

spi_nss_output_enable(SPI0);

spi_nss_output_disable

The description of spi_nss_output_disable is shown as below:

Table 3-659. Function spi_nss_output_disable

<table>
<thead>
<tr>
<th>Function name</th>
<th>spi_nss_output_disable</th>
</tr>
</thead>
<tbody>
<tr>
<td>Function prototype</td>
<td>void spi_nss_output_disable(uint32_t spi_periph);</td>
</tr>
<tr>
<td>Function descriptions</td>
<td>disable SPI NSS output function</td>
</tr>
<tr>
<td>Precondition</td>
<td>-</td>
</tr>
<tr>
<td>The called functions</td>
<td>-</td>
</tr>
</tbody>
</table>

Input parameter (in)

<table>
<thead>
<tr>
<th>spi_periph</th>
<th>SPI peripheral</th>
</tr>
</thead>
<tbody>
<tr>
<td>SPIx(x=0,1,2)</td>
<td>SPI peripheral selection</td>
</tr>
</tbody>
</table>

Output parameter (out)

- -

Return value

- -

Example:

/* disable SPI0 NSS output */

spi_nss_output_disable(SPI0);
spi_nss_internal_high

The description of spi_nss_internal_high is shown as below:

**Table 3-660. Function spi_nss_internal_high**

<table>
<thead>
<tr>
<th>Function name</th>
<th>spi_nss_internal_high</th>
</tr>
</thead>
<tbody>
<tr>
<td>Function prototype</td>
<td>void spi_nss_internal_high(uint32_t spi_periph);</td>
</tr>
<tr>
<td>Function descriptions</td>
<td>SPI NSS pin high level in software mode</td>
</tr>
<tr>
<td>Precondition</td>
<td>-</td>
</tr>
<tr>
<td>The called functions</td>
<td>-</td>
</tr>
<tr>
<td>Input parameter (in)</td>
<td>Spi_periph</td>
</tr>
<tr>
<td>Spi(x=0,1,2)</td>
<td>SPI peripheral selection</td>
</tr>
<tr>
<td>Output parameter (out)</td>
<td>-</td>
</tr>
<tr>
<td>Return value</td>
<td>-</td>
</tr>
</tbody>
</table>

Example:

/* SPI0 NSS pin is pulled high level in software mode */
spi_nss_internal_high(SPI0);

spi_nss_internal_low

The description of spi_nss_internal_low is shown as below:

**Table 3-661. Function spi_nss_internal_low**

<table>
<thead>
<tr>
<th>Function name</th>
<th>spi_nss_internal_low</th>
</tr>
</thead>
<tbody>
<tr>
<td>Function prototype</td>
<td>void spi_nss_internal_low(uint32_t spi_periph);</td>
</tr>
<tr>
<td>Function descriptions</td>
<td>SPI NSS pin low level in software mode</td>
</tr>
<tr>
<td>Precondition</td>
<td>-</td>
</tr>
<tr>
<td>The called functions</td>
<td>-</td>
</tr>
<tr>
<td>Input parameter (in)</td>
<td>Spi_periph</td>
</tr>
<tr>
<td>Spi(x=0,1,2)</td>
<td>SPI peripheral selection</td>
</tr>
<tr>
<td>Output parameter (out)</td>
<td>-</td>
</tr>
<tr>
<td>Return value</td>
<td>-</td>
</tr>
</tbody>
</table>

Example:

/* SPI0 NSS pin is pulled low level in software mode */
spi_nss_internal_low(SPI0);
spi_dma_enable

The description of spi_dma_enable is shown as below:

Table 3-662. Function spi_dma_enable

<table>
<thead>
<tr>
<th>Function name</th>
<th>spi_dma_enable</th>
</tr>
</thead>
<tbody>
<tr>
<td>Function prototype</td>
<td>void spi_dma_enable(uint32_t spi_periph, uint8_t dma);</td>
</tr>
<tr>
<td>Function descriptions</td>
<td>enable SPI DMA function</td>
</tr>
<tr>
<td>Precondition</td>
<td>-</td>
</tr>
<tr>
<td>The called functions</td>
<td>-</td>
</tr>
</tbody>
</table>

**Input parameter (in)**

- **spi_periph**: SPI peripheral
- **SPIx(x=0,1,2)**: SPI peripheral selection

**Input parameter (in)**

- **dma**: SPI DMA mode
  - **SPI_DMA_TRANSMIT**: SPI transmit data use DMA
  - **SPI_DMA_RECEIVE**: SPI receive data use DMA

**Output parameter (out)**

- -

Return value

Example:

/* enable SPI0 transmit data DMA function */

spi_dma_enable(SPI0, SPI_DMA_TRANSMIT);

spi_dma_disable

The description of spi_dma_disable is shown as below:

Table 3-663. Function spi_dma_disable

<table>
<thead>
<tr>
<th>Function name</th>
<th>spi_dma_disable</th>
</tr>
</thead>
<tbody>
<tr>
<td>Function prototype</td>
<td>void spi_dma_disable(uint32_t spi_periph, uint8_t dma);</td>
</tr>
<tr>
<td>Function descriptions</td>
<td>disable SPI DMA function</td>
</tr>
<tr>
<td>Precondition</td>
<td>-</td>
</tr>
<tr>
<td>The called functions</td>
<td>-</td>
</tr>
</tbody>
</table>

**Input parameter (in)**

- **spi_periph**: SPI peripheral
- **SPIx(x=0,1,2)**: SPI peripheral selection

**Input parameter (in)**

- **dma**: SPI DMA mode
  - **SPI_DMA_TRANSMIT**: SPI transmit data use DMA
  - **SPI_DMA_RECEIVE**: SPI receive data use DMA

Output parameter (out)

- -
Example:

/* disable SPI0 transmit data DMA function */
spi_dma_disable(SPI0, SPI_DMA_TRANSMIT);

**spi_i2s_data_frame_format_config**

The description of `spi_i2s_data_frame_format_config` is shown as below:

<p>| Table 3-664. Function <code>spi_i2s_data_frame_format_config</code> |
|-----------------|-----------------|
| <strong>Function name</strong> | <code>spi_i2s_data_frame_format_config</code> |
| <strong>Function prototype</strong> | <code>void spi_i2s_data_frame_format_config(uint32_t spi_periph, uint16_t frame_format);</code> |
| <strong>Function descriptions</strong> | configure SPI/I2S data frame format |
| <strong>Precondition</strong> | - |
| <strong>The called functions</strong> | - |
| <strong>Input parameter (in)</strong> | |
| <code>spi_periph</code> | SPI peripheral |
| <code>SPIx(x=0,1,2)</code> | SPI peripheral selection |
| <strong>Input parameter (in)</strong> | |
| <code>frame_format</code> | SPI frame size |
| <code>SPI_FRAMESIZE_16BIT</code> | SPI frame size is 16 bits |
| <code>SPI_FRAMESIZE_8BIT</code> | SPI frame size is 8 bits |</p>
<table>
<thead>
<tr>
<th><strong>Output parameter (out)</strong></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Return value</strong></td>
<td>-</td>
</tr>
</tbody>
</table>

Example:

/* configure SPI1/I2S1 data frame format size is 16 bits */
spi_i2s_data_frame_format_config(SPI1, SPI_FRAMESIZE_16BIT);

**spi_i2s_data_transmit**

The description of `spi_i2s_data_transmit` is shown as below:

| Table 3-665. Function `spi_i2s_data_transmit` |
|-----------------|-----------------|
| **Function name** | `spi_i2s_data_transmit` |
| **Function prototype** | `void spi_i2s_data_transmit(uint32_t spi_periph, uint16_t data);` |
| **Function descriptions** | SPI transmit data |
Precondition
The called functions
\[\text{Input parameter}(\text{in})\]
\[
\begin{array}{|c|}
\hline
\text{spi\_periph} & \text{SPI peripheral} \\
\hline
\text{SPIx}(x=0,1,2) & \text{SPI peripheral selection} \\
\hline
\end{array}
\]
\[\text{Input parameter}(\text{in})\]
\[
\begin{array}{|c|}
\hline
\text{data} & \text{16-bit data} \\
\hline
\end{array}
\]
\[\text{Output parameter}(\text{out})\]
\[\text{Return value}\]

Example:

/* SPI0 transmit data */

uint16_t spi0_send_array[10];

uint8_t send_n = 1;

spi_i2s_data_transmit(SPI0, spi0_send_array[send_n]);

\textbf{spi\_i2s\_data\_receive}

The description of \texttt{spi\_i2s\_data\_receive} is shown as below:

\begin{table}[h]
\begin{center}
\begin{tabular}{|c|}
\hline
\textbf{Function name} & \texttt{spi\_i2s\_data\_receive} \\
\hline
\textbf{Function prototype} & \texttt{uint16\_t\ spi\_i2s\_data\_receive(uint32\_t\ spi\_periph);} \\
\hline
\textbf{Function descriptions} & SPI receive data \\
\hline
\textbf{Precondition} & - \\
\hline
\textbf{The called functions} & - \\
\hline
\textbf{Input parameter}(\text{in}) & \\
\hline
\text{spi\_periph} & \text{SPI peripheral} \\
\hline
\text{SPIx}(x=0,1,2) & \text{SPI peripheral selection} \\
\hline
\textbf{Output parameter}(\text{out}) & \\
\hline
- & - \\
\hline
\textbf{Return value} & \\
\hline
\text{uint16\_t} & \text{16-bit data} \\
\hline
\end{tabular}
\end{center}
\end{table}

Example:

/* SPI0 receive data */

uint16_t spi0_receive_array[10];

uint8_t receive_n = 1;

spi0_receive_array[receive_n] = spi_i2s_data_receive(SPI0);
spi_bidirectional_transfer_config

The description of spi_bidirectional_transfer_config is shown as below:

Table 3-667. Function spi_bidirectional_transfer_config

<table>
<thead>
<tr>
<th>Function name</th>
<th>spi_bidirectional_transfer_config</th>
</tr>
</thead>
<tbody>
<tr>
<td>Function prototype</td>
<td>void spi_bidirectional_transfer_config(uint32_t spi_periph, uint32_t transfer_direction);</td>
</tr>
<tr>
<td>Function descriptions</td>
<td>configure SPI bidirectional transfer direction</td>
</tr>
<tr>
<td>Precondition</td>
<td>-</td>
</tr>
<tr>
<td>The called functions</td>
<td>-</td>
</tr>
<tr>
<td>Input parameter(in)</td>
<td></td>
</tr>
<tr>
<td>spi_periph</td>
<td>SPI peripheral</td>
</tr>
<tr>
<td>SPIx(x=0,1,2)</td>
<td>SPI peripheral selection</td>
</tr>
<tr>
<td>transfer_direction</td>
<td>SPI transfer direction</td>
</tr>
<tr>
<td>SPI_BIDIRECTIONAL_TRANSMIT</td>
<td>SPI work in transmit-only mode</td>
</tr>
<tr>
<td>SPI_BIDIRECTIONAL_RECEIVE</td>
<td>SPI work in receive-only mode</td>
</tr>
<tr>
<td>Output parameter(out)</td>
<td>-</td>
</tr>
<tr>
<td>Return value</td>
<td>-</td>
</tr>
</tbody>
</table>

Example:

/* SPI0 works in transmit-only mode */
spi_bidirectional_transfer_config(SPI0, SPI_BIDIRECTIONAL_TRANSMIT);

spi_crc_polynomial_set

The description of spi_crc_polynomial_set is shown as below:

Table 3-668. Function spi_crc_polynomial_set

<table>
<thead>
<tr>
<th>Function name</th>
<th>spi_crc_polynomial_set</th>
</tr>
</thead>
<tbody>
<tr>
<td>Function prototype</td>
<td>void spi_crc_polynomial_set(uint32_t spi_periph, uint16_t crc_poly);</td>
</tr>
<tr>
<td>Function descriptions</td>
<td>set SPI CRC polynomial</td>
</tr>
<tr>
<td>Precondition</td>
<td>-</td>
</tr>
<tr>
<td>The called functions</td>
<td>-</td>
</tr>
<tr>
<td>Input parameter(in)</td>
<td></td>
</tr>
<tr>
<td>spi_periph</td>
<td>SPI peripheral</td>
</tr>
<tr>
<td>SPIx(x=0,1,2)</td>
<td>SPI peripheral selection</td>
</tr>
<tr>
<td>crc_poly</td>
<td>CRC polynomial value</td>
</tr>
</tbody>
</table>
### spi_crc_polynomial_get

The description of `spi_crc_polynomial_get` is shown as below:

**Table 3-669. Function spi_crc_polynomial_get**

<table>
<thead>
<tr>
<th>Function name</th>
<th>spi_crc_polynomial_get</th>
</tr>
</thead>
<tbody>
<tr>
<td>Function prototype</td>
<td>uint16_t spi_crc_polynomial_get(uint32_t spi_periph);</td>
</tr>
<tr>
<td>Function descriptions</td>
<td>get SPI CRC polynomial</td>
</tr>
<tr>
<td>Precondition</td>
<td>-</td>
</tr>
<tr>
<td>The called functions</td>
<td>-</td>
</tr>
<tr>
<td>Input parameter(in)</td>
<td></td>
</tr>
<tr>
<td>spi_periph</td>
<td>SPI peripheral</td>
</tr>
<tr>
<td>SPIx(x=0,1,2)</td>
<td>SPI peripheral selection</td>
</tr>
<tr>
<td>Output parameter(out)</td>
<td>-</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>Return value</td>
<td>-</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>uint16_t</td>
</tr>
<tr>
<td></td>
<td>16-bit CRC polynomial (0-0xFFFF)</td>
</tr>
</tbody>
</table>

Example:

/* get SPI0 CRC polynomial */

```c
uint16_t CRC_VALUE;
CRC_VALUE = spi_crc_polynomial_get(SPI0);
```

### spi_crc_on

The description of `spi_crc_on` is shown as below:

**Table 3-670. Function spi_crc_on**

<table>
<thead>
<tr>
<th>Function name</th>
<th>spi_crc_on</th>
</tr>
</thead>
<tbody>
<tr>
<td>Function prototype</td>
<td>void spi_crc_on(uint32_t spi_periph);</td>
</tr>
<tr>
<td>Function descriptions</td>
<td>turn on SPI CRC function</td>
</tr>
<tr>
<td>Precondition</td>
<td>-</td>
</tr>
<tr>
<td>The called functions</td>
<td>-</td>
</tr>
</tbody>
</table>
GD32L23x Firmware Library User Guide

Input parameter (in)

<table>
<thead>
<tr>
<th>spi_periph</th>
<th>SPI peripheral</th>
</tr>
</thead>
<tbody>
<tr>
<td>SPIx(x=0,1,2)</td>
<td>SPI peripheral selection</td>
</tr>
</tbody>
</table>

Output parameter (out)

| -              | -              |

Return value

| -              | -              |

Example:

/* turn on SPI0 CRC function */

spi_crc_on(SPI0);

**spi_crc_off**

The description of spi_crc_off is shown as below:

Table 3-671. Function spi_crc_off

<table>
<thead>
<tr>
<th>Function name</th>
<th>spi_crc_off</th>
</tr>
</thead>
<tbody>
<tr>
<td>Function prototype</td>
<td>void spi_crc_off(uint32_t spi_periph);</td>
</tr>
<tr>
<td>Function descriptions</td>
<td>turn off SPI CRC function</td>
</tr>
<tr>
<td>Precondition</td>
<td>-</td>
</tr>
<tr>
<td>The called functions</td>
<td>-</td>
</tr>
</tbody>
</table>

Input parameter (in)

<table>
<thead>
<tr>
<th>spi_periph</th>
<th>SPI peripheral</th>
</tr>
</thead>
<tbody>
<tr>
<td>SPIx(x=0,1,2)</td>
<td>SPI peripheral selection</td>
</tr>
</tbody>
</table>

Output parameter (out)

| -              | -              |

Return value

| -              | -              |

Example:

/* turn off SPI0 CRC function */

spi_crc_off(SPI0);

**spi_crc_next**

The description of spi_crc_next is shown as below:

Table 3-672. Function spi_crc_next

<table>
<thead>
<tr>
<th>Function name</th>
<th>spi_crc_next</th>
</tr>
</thead>
<tbody>
<tr>
<td>Function prototype</td>
<td>void spi_crc_next(uint32_t spi_periph);</td>
</tr>
<tr>
<td>Function descriptions</td>
<td>SPI next data is CRC value</td>
</tr>
<tr>
<td>Precondition</td>
<td>-</td>
</tr>
</tbody>
</table>
### The called functions

<table>
<thead>
<tr>
<th>Function name</th>
<th>spi_crc_next</th>
</tr>
</thead>
<tbody>
<tr>
<td>Function prototype</td>
<td>uint16_t spi_crc_next(uint32_t spi_periph, uint8_t crc);</td>
</tr>
<tr>
<td>Function descriptions</td>
<td>get SPI CRC send value or receive value</td>
</tr>
<tr>
<td>Precondition</td>
<td>-</td>
</tr>
<tr>
<td>The called functions</td>
<td>-</td>
</tr>
</tbody>
</table>

### spi_crc_get

The description of `spi_crc_get` is shown as below:

#### Table 3-673. Function `spi_crc_get`

<table>
<thead>
<tr>
<th>Function name</th>
<th>spi_crc_get</th>
</tr>
</thead>
<tbody>
<tr>
<td>Function prototype</td>
<td>uint16_t spi_crc_get(uint32_t spi_periph, uint8_t crc);</td>
</tr>
<tr>
<td>Function descriptions</td>
<td>get SPI CRC send value or receive value</td>
</tr>
<tr>
<td>Precondition</td>
<td>-</td>
</tr>
<tr>
<td>The called functions</td>
<td>-</td>
</tr>
</tbody>
</table>

#### Input parameter (in)

<table>
<thead>
<tr>
<th>spi_periph</th>
<th>SPI peripheral</th>
</tr>
</thead>
<tbody>
<tr>
<td>SPIx(x=0,1,2)</td>
<td>SPI peripheral selection</td>
</tr>
</tbody>
</table>

#### Input parameter (in)

<table>
<thead>
<tr>
<th>crc</th>
<th>SPI crc value</th>
</tr>
</thead>
<tbody>
<tr>
<td>SPI_CRC_TX</td>
<td>get transmit crc value</td>
</tr>
<tr>
<td>SPI_CRC_RX</td>
<td>get receive crc value</td>
</tr>
</tbody>
</table>

#### Output parameter (out)

| - | - |

#### Return value

| uint16_t | 16-bit CRC value (0-0xFFFF) |

Example:

/* SPI0 next data is CRC value */

spi_crc_next(SPI0);

### spi_i2s_interrupt_enable

The description of `spi_i2s_interrupt_enable` is shown as below:

/* get SPI0 CRC send value */

uint16_t value;

value = spi_crc_get(SPI0, SPI_CRC_TX);
## Table 3-674. Function spi_i2s_interrupt_enable

<table>
<thead>
<tr>
<th>Function name</th>
<th>spi_i2s_interrupt_enable</th>
</tr>
</thead>
<tbody>
<tr>
<td>Function prototype</td>
<td>void spi_i2s_interrupt_enable(uint32_t spi_periph, uint8_t interrupt);</td>
</tr>
<tr>
<td>Function descriptions</td>
<td>enable SPI and I2S interrupt</td>
</tr>
<tr>
<td>Precondition</td>
<td>-</td>
</tr>
<tr>
<td>The called functions</td>
<td>-</td>
</tr>
</tbody>
</table>

**Input parameter(in)**

<table>
<thead>
<tr>
<th>spi_periph</th>
<th>SPI peripheral</th>
</tr>
</thead>
<tbody>
<tr>
<td>SPIx(x=0,1,2)</td>
<td>SPI peripheral selection</td>
</tr>
</tbody>
</table>

**Input parameter(in)**

<table>
<thead>
<tr>
<th>interrupt</th>
<th>SPI/I2S interrupt</th>
</tr>
</thead>
<tbody>
<tr>
<td>SPI_I2S_INT_TBE</td>
<td>transmit buffer empty interrupt</td>
</tr>
<tr>
<td>SPI_I2S_INT_RBNE</td>
<td>receive buffer not empty interrupt</td>
</tr>
<tr>
<td>SPI_I2S_INT_ERR</td>
<td>CRC error, configuration error, reception overrun error, transmission underrun error and format error interrupt</td>
</tr>
</tbody>
</table>

**Output parameter(out)**

| - | - |

**Return value**

| - | - |

Example:

```c
/* enable SPI0 transmit buffer empty interrupt */
spi_i2s_interrupt_enable(SPI0, SPI_I2S_INT_TBE);
```

### spi_i2s_interrupt_disable

The description of spi_i2s_interrupt_disable is shown as below:

## Table 3-675. Function spi_i2s_interrupt_disable

<table>
<thead>
<tr>
<th>Function name</th>
<th>spi_i2s_interrupt_disable</th>
</tr>
</thead>
<tbody>
<tr>
<td>Function prototype</td>
<td>void spi_i2s_interrupt_disable(uint32_t spi_periph, uint8_t interrupt);</td>
</tr>
<tr>
<td>Function descriptions</td>
<td>disable SPI and I2S interrupt</td>
</tr>
<tr>
<td>Precondition</td>
<td>-</td>
</tr>
<tr>
<td>The called functions</td>
<td>-</td>
</tr>
</tbody>
</table>

**Input parameter(in)**

<table>
<thead>
<tr>
<th>spi_periph</th>
<th>SPI peripheral</th>
</tr>
</thead>
<tbody>
<tr>
<td>SPIx(x=0,1,2)</td>
<td>SPI peripheral selection</td>
</tr>
</tbody>
</table>

**Input parameter(in)**

<table>
<thead>
<tr>
<th>interrupt</th>
<th>SPI/I2S interrupt</th>
</tr>
</thead>
<tbody>
<tr>
<td>SPI_I2S_INT_TBE</td>
<td>transmit buffer empty interrupt</td>
</tr>
<tr>
<td>SPI_I2S_INT_RBNE</td>
<td>receive buffer not empty interrupt</td>
</tr>
<tr>
<td>SPI_I2S_INT_ERR</td>
<td>CRC error, configuration error, reception overrun error, transmission underrun error and format error interrupt</td>
</tr>
</tbody>
</table>
Example:

```c
/* disable SPI0 transmit buffer empty interrupt */
spi_i2s_interrupt_disable(SPI0, SPI_I2S_INT_TBE);
```

### spi_i2s_interrupt_flag_get

The description of `spi_i2s_interrupt_flag_get` is shown as below:

#### Table 3-676. Function `spi_i2s_interrupt_flag_get`

<table>
<thead>
<tr>
<th>Function name</th>
<th><code>spi_i2s_interrupt_flag_get</code></th>
</tr>
</thead>
<tbody>
<tr>
<td>Function prototype</td>
<td><code>FlagStatus spi_i2s_interrupt_flag_get(uint32_t spi_periph, uint8_t interrupt);</code></td>
</tr>
<tr>
<td>Function descriptions</td>
<td>get SPI and I2S interrupt status</td>
</tr>
<tr>
<td>Precondition</td>
<td>-</td>
</tr>
<tr>
<td>The called functions</td>
<td>-</td>
</tr>
</tbody>
</table>

#### Input parameter (in)

| `spi_periph` | SPI peripheral |
| `SPIx(x=0,1,2)` | SPI peripheral selection |

#### Input parameter (in)

| `interrupt` | SPI/I2S interrupt flag status |
| `SPI_I2S_INT_FLAG_TBE` | transmit buffer empty interrupt |
| `SPI_I2S_INT_FLAG_RBNE` | receive buffer not empty interrupt |
| `SPI_I2S_INT_FLAG_RXORERR` | overrun interrupt |
| `SPI_INT_FLAG_CONFERR` | config error interrupt |
| `SPI_INT_FLAG_CRCERR` | CRC error interrupt |
| `I2S_INT_FLAG_TXURERR` | underrun error interrupt |

#### Output parameter (out)

| - | - |

#### Return value

| FlagStatus | SET or RESET |
FlagStatus Flag_interrupt = RESET;

Flag_interrupt = spi_i2s_interrupt_flag_get(SPI0, SPI_I2S_INT_FLAG_TBE);

**spi_i2s_flag_get**

The description of `spi_i2s_flag_get` is shown as below:

**Table 3-677. Function `spi_i2s_flag_get`**

<table>
<thead>
<tr>
<th>Function name</th>
<th><code>spi_i2s_flag_get</code></th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Function prototype</strong></td>
<td>FlagStatus spi_i2s_flag_get(uint32_t spi_periph, uint32_t flag);</td>
</tr>
<tr>
<td><strong>Function descriptions</strong></td>
<td>get SPI and I2S flag status</td>
</tr>
<tr>
<td><strong>Precondition</strong></td>
<td>-</td>
</tr>
<tr>
<td><strong>The called functions</strong></td>
<td>-</td>
</tr>
</tbody>
</table>

**Input parameter (in)**

| spi_periph | SPI peripheral |
| SPIx (x=0,1,2) | SPI peripheral selection |

**Input parameter (in)**

| flag | SPI/I2S flag status |
| SPI_FLAG_TBE | transmit buffer empty flag |
| SPI_FLAG_RBNE | receive buffer not empty flag |
| SPI_FLAG_TRANS | transmit on-going flag |
| SPI_I2S_INT_FLAG_RXORERR | receive overrun error flag |
| SPI_FLAG_CONFERR | mode config error flag |
| SPI_FLAG_CRCERR | CRC error flag |
| I2S_FLAG_TBE | transmit buffer empty flag |
| I2S_FLAG_RBNE | receive buffer not empty flag |
| I2S_FLAG_TRANS | transmit on-going flag |
| I2S_FLAG_RXORERR | overrun error flag |
| I2S_FLAG_TXURERR | underrun error flag |
| I2S_FLAG_CH | channel side flag |

**Output parameter (out)**

- |

**Return value**

| FlagStatus | SET or RESET |

Example:

```c
/* get SPI0 transmit buffer empty flag status */
FlagStatus Flag = RESET;
Flag = spi_i2s_flag_get(SPI0, SPI_FLAG_TBE);
```
**spi_crc_error_clear**

The description of `spi_crc_error_clear` is shown as below:

<table>
<thead>
<tr>
<th>Function name</th>
<th>spi_crc_error_clear</th>
</tr>
</thead>
<tbody>
<tr>
<td>Function prototype</td>
<td>void spi_crc_error_clear(uint32_t spi_periph);</td>
</tr>
<tr>
<td>Function descriptions</td>
<td>clear SPI CRC error flag status</td>
</tr>
<tr>
<td>Precondition</td>
<td></td>
</tr>
<tr>
<td>The called functions</td>
<td></td>
</tr>
</tbody>
</table>

| Input parameter (in)          | 
|-------------------------------|----------------------------------------------------------|
| spi_periph                    | SPI peripheral                                           |
| SPIx(x=0,1,2)                 | SPI peripheral selection                                 |

| Output parameter (out)        | 
|-------------------------------|----------------------------------------------------------|
|                               | -                                                         |

| Return value                  | -                                                         |

Example:

/* clear SPI0 CRC error flag status */

`spi_crc_error_clear(SPI0);`

**qspi_enable**

The description of `qspi_enable` is shown as below:

<table>
<thead>
<tr>
<th>Function name</th>
<th>qspi_enable</th>
</tr>
</thead>
<tbody>
<tr>
<td>Function prototype</td>
<td>void qspi_enable(uint32_t spi_periph);</td>
</tr>
<tr>
<td>Function descriptions</td>
<td>enable quad wire SPI</td>
</tr>
<tr>
<td>Precondition</td>
<td></td>
</tr>
<tr>
<td>The called functions</td>
<td></td>
</tr>
</tbody>
</table>

| Input parameter (in)          | 
|-------------------------------|----------------------------------------------------------|
| spi_periph                    | SPI peripheral                                           |
| SPIx(x=0)                     | SPI peripheral selection                                 |

| Output parameter (out)        | 
|-------------------------------|----------------------------------------------------------|
|                               | -                                                         |

| Return value                  | -                                                         |

Example:

/* enable quad wire SPI */

`qspi_enable(SPI0);`
**qspi_disable**

The description of `qspi_disable` is shown as below:

**Table 3-680. Function qspi_enable**

<table>
<thead>
<tr>
<th>Function name</th>
<th>qspi_disable</th>
</tr>
</thead>
<tbody>
<tr>
<td>Function prototype</td>
<td>void qspi_disable(uint32_t spi_periph);</td>
</tr>
<tr>
<td>Function descriptions</td>
<td>disable quad wire SPI</td>
</tr>
<tr>
<td>Precondition</td>
<td>-</td>
</tr>
<tr>
<td>The called functions</td>
<td>-</td>
</tr>
</tbody>
</table>

**Input parameter (in)**

<table>
<thead>
<tr>
<th>spi_periph</th>
<th>SPI peripheral</th>
</tr>
</thead>
<tbody>
<tr>
<td>SPIx(x=0)</td>
<td>SPI peripheral selection</td>
</tr>
</tbody>
</table>

**Output parameter (out)**

| - | - |

**Return value**

Example:

```c
/* disable quad wire SPI */
qspi_disable(SPI0);
```

**qspi_write_enable**

The description of `qspi_write_enable` is shown as below:

**Table 3-681. Function qspi_write_enable**

<table>
<thead>
<tr>
<th>Function name</th>
<th>qspi_write_enable</th>
</tr>
</thead>
<tbody>
<tr>
<td>Function prototype</td>
<td>void qspi_write_enable(uint32_t spi_periph);</td>
</tr>
<tr>
<td>Function descriptions</td>
<td>enable quad wire SPI write</td>
</tr>
<tr>
<td>Precondition</td>
<td>-</td>
</tr>
<tr>
<td>The called functions</td>
<td>-</td>
</tr>
</tbody>
</table>

**Input parameter (in)**

<table>
<thead>
<tr>
<th>spi_periph</th>
<th>SPI peripheral</th>
</tr>
</thead>
<tbody>
<tr>
<td>SPIx(x=0)</td>
<td>SPI peripheral selection</td>
</tr>
</tbody>
</table>

**Output parameter (out)**

| - | - |

**Return value**

Example:

```c
/* enable quad wire SPI write */
qspi_write_enable(SPI0);
```
The description of qspi_read_enable is shown as below:

Table 3-682. Function qspi_read_enable

<table>
<thead>
<tr>
<th>Function name</th>
<th>qspi_read_enable</th>
</tr>
</thead>
<tbody>
<tr>
<td>Function prototype</td>
<td>void qspi_read_enable(uint32_t spi_periph);</td>
</tr>
<tr>
<td>Function descriptions</td>
<td>enable quad wire SPI read</td>
</tr>
<tr>
<td>Precondition</td>
<td>-</td>
</tr>
<tr>
<td>The called functions</td>
<td>-</td>
</tr>
</tbody>
</table>

Input parameter (in)

- spi_periph
  - SPI peripheral
  - SPIx(x=0)

Output parameter (out)

- Return value
  - -

Example:

/* enable quad wire SPI read */
qspi_read_enable(SPI0);

The description of qspi_io23_output_enable is shown as below:

Table 3-683. Function qspi_io23_output_enable

<table>
<thead>
<tr>
<th>Function name</th>
<th>qspi_io23_output_enable</th>
</tr>
</thead>
<tbody>
<tr>
<td>Function prototype</td>
<td>void qspi_io23_output_enable(uint32_t spi_periph);</td>
</tr>
<tr>
<td>Function descriptions</td>
<td>enable SPI_IO2 and SPI_IO3 pin output</td>
</tr>
<tr>
<td>Precondition</td>
<td>-</td>
</tr>
<tr>
<td>The called functions</td>
<td>-</td>
</tr>
</tbody>
</table>

Input parameter (in)

- spi_periph
  - SPI peripheral
  - SPIx(x=0)

Output parameter (out)

- Return value
  - -

Example:

/* enable SPI_IO2 and SPI_IO3 pin output */
qspi_io23_output_enable(SPI0);
**qspi_io23_output_disable**

The description of qspi_io23_output_disable is shown as below:

**Table 3-684. Function qspi_io23_output_disable**

<table>
<thead>
<tr>
<th>Function name</th>
<th>qspi_io23_output_disable</th>
</tr>
</thead>
<tbody>
<tr>
<td>Function prototype</td>
<td>void qspi_io23_output_disable(uint32_t spi_periph);</td>
</tr>
<tr>
<td>Function descriptions</td>
<td>disable SPI_IO2 and SPI_IO3 pin output</td>
</tr>
<tr>
<td>Precondition</td>
<td>-</td>
</tr>
<tr>
<td>The called functions</td>
<td>-</td>
</tr>
<tr>
<td>Input parameter (in)</td>
<td></td>
</tr>
<tr>
<td>spi_periph</td>
<td>SPI peripheral</td>
</tr>
<tr>
<td>SPIx(x=0)</td>
<td>SPI peripheral selection</td>
</tr>
<tr>
<td>Output parameter (out)</td>
<td></td>
</tr>
<tr>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>Return value</td>
<td>-</td>
</tr>
</tbody>
</table>

Example:

/* disable SPI_IO2 and SPI_IO3 pin output */

qspi_io23_output_disable(SPI0);

### 3.23. SYSCFG

The SYSCFG registers are listed in chapter 3.23.1, the SYSCFG firmware functions are introduced in chapter 3.23.2.

#### 3.23.1. Descriptions of Peripheral registers

SYSCFG registers are listed in the table shown as below:

**Table 3-685. SYSCFG Registers**

<table>
<thead>
<tr>
<th>Registers</th>
<th>Descriptions</th>
</tr>
</thead>
<tbody>
<tr>
<td>SYSCFG_CFG0</td>
<td>system configuration register 0</td>
</tr>
<tr>
<td>SYSCFG_EXTISS0</td>
<td>EXTI sources selection register 0</td>
</tr>
<tr>
<td>SYSCFG_EXTISS1</td>
<td>EXTI sources selection register 1</td>
</tr>
<tr>
<td>SYSCFG_EXTISS2</td>
<td>EXTI sources selection register 2</td>
</tr>
<tr>
<td>SYSCFG_EXTISS3</td>
<td>EXTI sources selection register 3</td>
</tr>
<tr>
<td>SYSCFG_CPU_IRQ_LAT</td>
<td>IRQ Latency register</td>
</tr>
</tbody>
</table>

#### 3.23.2. Descriptions of Peripheral functions

SYSCFG firmware functions are listed in the table shown as below:
Table 3-686. SYSCFG firmware function

<table>
<thead>
<tr>
<th>Function name</th>
<th>Function description</th>
</tr>
</thead>
<tbody>
<tr>
<td>syscfg_deinit</td>
<td>deinit syCfg module</td>
</tr>
<tr>
<td>syscfg_exti_line_config</td>
<td>configure the GPIO pin as EXTI Line</td>
</tr>
<tr>
<td>syscfg_pin_remap_enable</td>
<td>enable remap pin function for small packages</td>
</tr>
<tr>
<td>syscfg_pin_remap_disable</td>
<td>disable remap pin function for small packages</td>
</tr>
<tr>
<td>syscfg_high_current_enable</td>
<td>enable PBx(x=6,7,8,9) high current capability</td>
</tr>
<tr>
<td>syscfg_high_current_disable</td>
<td>disable PBx(x=6,7,8,9) high current capability</td>
</tr>
<tr>
<td>irq_latency_set</td>
<td>set the IRQ_LATENCY value</td>
</tr>
<tr>
<td>syscfg_bootmode_get</td>
<td>get the boot mode</td>
</tr>
</tbody>
</table>

**syscfg_deinit**

The description of `syscfg_deinit` is shown as below:

Table 3-687. Function `syscfg_deinit`

<table>
<thead>
<tr>
<th>Function name</th>
<th>syscfg_deinit</th>
</tr>
</thead>
<tbody>
<tr>
<td>Function prototype</td>
<td>void syscfg_deinit(void);</td>
</tr>
<tr>
<td>Function descriptions</td>
<td>reset the SYSCFG registers</td>
</tr>
<tr>
<td>Precondition</td>
<td>-</td>
</tr>
<tr>
<td>The called functions</td>
<td>-</td>
</tr>
</tbody>
</table>

**Input parameter (in)**

- None

**Output parameter (out)**

- None

**Return value**

- None

Example:

/* reset SYSCFG registers */

syscfg_deinit();

**syscfg_exti_line_config**

The description of `syscfg_exti_line_config` is shown as below:

Table 3-688. Function `syscfg_exti_line_config`

<table>
<thead>
<tr>
<th>Function name</th>
<th>syscfg_exti_line_config</th>
</tr>
</thead>
<tbody>
<tr>
<td>Function prototype</td>
<td>void syscfg_exti_line_config(uint8_t exti_port, uint8_t exti_pin);</td>
</tr>
<tr>
<td>Function descriptions</td>
<td>configure the GPIO pin as EXTI Line</td>
</tr>
<tr>
<td>Precondition</td>
<td>-</td>
</tr>
<tr>
<td>The called functions</td>
<td>-</td>
</tr>
</tbody>
</table>

**Input parameter (in)**

| exti_port                        | specify the GPIO port used in EXTI |

- None

- None

- None
**EXTI_SOURCE_GPIOx**  \( x = A,B,C,D,F \)

**exti_pin**

specify the EXTI line

**EXTI_SOURCE_PINx**

GPIOAx = 0..15, GPIOBx = 0..15, GPIOCx = 0..15, GPIODx = 0..6,8,9, GPIOFx = 0,1

Output parameter (out)

- -

**Return value**

- -

Example:

/* configure the GPIO pin as EXTI Line */

syscfg_exti_line_config(EXTI_SOURCE_GPIOA, EXTI_SOURCE_PIN0);

**syscfg_pin_remap_enable**

The description of `syscfg_pin_remap_enable` is shown as below:

### Table 3-689. Function `syscfg_pin_remap_enable`

<table>
<thead>
<tr>
<th>Function name</th>
<th><code>syscfg_pin_remap_enable</code></th>
</tr>
</thead>
<tbody>
<tr>
<td>Function prototype</td>
<td>void <code>syscfg_pin_remap_enable(uint32_t remap_pin);</code></td>
</tr>
<tr>
<td>Function descriptions</td>
<td>enable remap pin function for small packages</td>
</tr>
<tr>
<td>Precondition</td>
<td>-</td>
</tr>
<tr>
<td>The called functions</td>
<td>-</td>
</tr>
</tbody>
</table>

**Input parameter (in)**

<table>
<thead>
<tr>
<th>remap_pin</th>
<th>remap pin</th>
</tr>
</thead>
<tbody>
<tr>
<td>SYSCFG_PA11_PA12_REMAP</td>
<td>PA11 PA12 remap</td>
</tr>
<tr>
<td>SYSCFG_BOOT0_PD3_REMAP</td>
<td>BOOT0 PD3 remap</td>
</tr>
</tbody>
</table>

**Output parameter (out)**

- -

**Return value**

- -

Example:

/* enable BOOT0 remap to PD3 function */

syscfg_pin_remap_enable(SYSCFG_BOOT0_PD3_REMAP);

**syscfg_pin_remap_disable**

The description of `syscfg_pin_remap_disable` is shown as below:

### Table 3-690. Function `syscfg_pin_remap_disable`

<table>
<thead>
<tr>
<th>Function name</th>
<th><code>syscfg_pin_remap_disable</code></th>
</tr>
</thead>
</table>
Function prototype
void syscfg_pin_remap_disable(void);

Function descriptions
disable remap pin function for small packages

Precondition
-

The called functions
-

Input parameter(in)

remap_pin remap pin
SYSCFG_PA11_PA12_REMAP PA11 PA12 remap
SYSCFG_BOOT0_PD3_REMAP BOOT0 PD3 remap

Output parameter(out)
-

Return value
-

Example:
/* disable BOOT0 remap to PD3 function */
syscfg_pin_remap_disable(SYSCFG_BOOT0_PD3_REMAP);

syscfg_high_current_enable

The description of syscfg_high_current_enable is shown as below:

Table 3-691. Function syscfg_high_current_enable

<table>
<thead>
<tr>
<th>Function name</th>
<th>syscfg_high_current_enable</th>
</tr>
</thead>
<tbody>
<tr>
<td>Function prototype</td>
<td>void syscfg_high_current_enable(uint32_t syscfg_gpio);</td>
</tr>
<tr>
<td>Function descriptions</td>
<td>enable PBx(x=6,7,8,9) high current capability</td>
</tr>
<tr>
<td>Precondition</td>
<td>-</td>
</tr>
<tr>
<td>The called functions</td>
<td>-</td>
</tr>
</tbody>
</table>

Input parameter(in)

syscfg_gpio SYSCFG GPIO
SYSCFG_PB6_HIGH_CURRENT PB6 pin high current capability
SYSCFG_PB7_HIGH_CURRENT PB7 pin high current capability
SYSCFG_PB8_HIGH_CURRENT PB8 pin high current capability
SYSCFG_PB9_HIGH_CURRENT PB9 pin high current capability

Output parameter(out)
-

Return value
-
Example:

/* enable PB9 high current capability */

syscfg_high_current_enable(SYSCFG_PB9_HIGH_CURRENT);

**syscfg_high_current_disable**

The description of `syscfg_high_current_disable` is shown as below:

### Table 3-692. Function `syscfg_high_current_disable`

<table>
<thead>
<tr>
<th>Function name</th>
<th><code>syscfg_high_current_disable</code></th>
</tr>
</thead>
<tbody>
<tr>
<td>Function prototype</td>
<td><code>void syscfg_high_current_enable(uint32_t syscfg_gpio);</code></td>
</tr>
<tr>
<td>Function descriptions</td>
<td>disable PB9 high current capability</td>
</tr>
<tr>
<td>Precondition</td>
<td>-</td>
</tr>
<tr>
<td>The called functions</td>
<td>-</td>
</tr>
</tbody>
</table>

**Input parameter (in)**

<table>
<thead>
<tr>
<th><code>syscfg_gpio</code></th>
<th>SYSCFG GPIO</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>SYSCFG_PB6_HIGH_CURRENT</code></td>
<td>PB6 pin high current capability</td>
</tr>
<tr>
<td><code>SYSCFG_PB7_HIGH_CURRENT</code></td>
<td>PB7 pin high current capability</td>
</tr>
<tr>
<td><code>SYSCFG_PB8_HIGH_CURRENT</code></td>
<td>PB8 pin high current capability</td>
</tr>
<tr>
<td><code>SYSCFG_PB9_HIGH_CURRENT</code></td>
<td>PB9 pin high current capability</td>
</tr>
</tbody>
</table>

**Output parameter (out)**

- |

**Return value**

- |

Example:

/* disable PB9 high current capability */

syscfg_high_current_disable(SYSCFG_PB9_HIGH_CURRENT);

**irq_latency_set**

The description of `irq_latency_set` is shown as below:

### Table 3-693. Function `irq_latency_set`

<table>
<thead>
<tr>
<th>Function name</th>
<th><code>irq_latency_set</code></th>
</tr>
</thead>
<tbody>
<tr>
<td>Function prototype</td>
<td><code>void irq_latency_set(uint8_t irq_latency);</code></td>
</tr>
<tr>
<td>Function descriptions</td>
<td>set the IRQ_LATENCY value</td>
</tr>
<tr>
<td>Precondition</td>
<td>-</td>
</tr>
<tr>
<td>The called functions</td>
<td>-</td>
</tr>
</tbody>
</table>
Example:

/* set the wait state counter value */

irq_latency_set(0xFF);

**syscfg_bootmode_get**

The description of syscfg_bootmode_get is shown as below:

<table>
<thead>
<tr>
<th>Function name</th>
<th>syscfg_bootmode_get</th>
</tr>
</thead>
<tbody>
<tr>
<td>Function prototype</td>
<td>uint8_t syscfg_bootmode_get(void);</td>
</tr>
<tr>
<td>Function descriptions</td>
<td>get the current boot mode</td>
</tr>
<tr>
<td>Precondition</td>
<td>-</td>
</tr>
<tr>
<td>The called functions</td>
<td>-</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Input parameter(in)</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>-</td>
<td>-</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Output parameter(out)</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>-</td>
<td>-</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Return value</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>uint8_t boot_mode</td>
<td></td>
</tr>
</tbody>
</table>

**Example:**

/* get the current boot mode */

uint8_t boot_mode;

boot_mode = syscfg_bootmode_get();
3.24.  TIMER

The timers have a 16-bit counter that can be used as an unsigned counter and supports both input capture and output compare. Timers (TIMERx) are divided into three sorts: general level0 timer (TIMER1, TIMER2), general level1 timer (TIMER8, TIMER11), Basic timer (TIMER5, TIMER6). The specific functions of different types of timer are different. The TIMER registers are listed in chapter 3.24.1, the TIMER firmware functions are introduced in chapter 3.24.2.

3.24.1. Descriptions of Peripheral registers

TIMERx registers are listed in the table shown as below:

<table>
<thead>
<tr>
<th>Table 3-695. TIMERx Registers</th>
</tr>
</thead>
<tbody>
<tr>
<td>Registers</td>
</tr>
<tr>
<td>TIMER_CTL0(timerx, x=1, 2, 5, 6, 8, 11)</td>
</tr>
<tr>
<td>TIMERx_CTL1(timerx, x=1, 2, 5, 6)</td>
</tr>
<tr>
<td>TIMERx_SMCFG(timerx, x=1, 2, 8, 11)</td>
</tr>
<tr>
<td>TIMERx_DMAINTEN(timerx, x=1, 2, 5, 6, 8, 11)</td>
</tr>
<tr>
<td>TIMERx_INTF(timerx, x=1, 2, 5, 6, 8, 11)</td>
</tr>
<tr>
<td>TIMERx_SWEVG(timerx, x=1, 2, 5, 6, 8, 11)</td>
</tr>
<tr>
<td>TIMERx_CHCTRL0(timerx, x=1, 2, 8, 11)</td>
</tr>
<tr>
<td>TIMERx_CHCTRL1(timerx, x=1, 2)</td>
</tr>
<tr>
<td>TIMERx_CHCTRL2(timerx, x=1, 2, 8, 11)</td>
</tr>
<tr>
<td>TIMERx_CNT(timerx, x=1, 2, 5, 6, 8, 11)</td>
</tr>
<tr>
<td>TIMERx_PSC(timerx, x=1, 2, 5, 6, 8, 11)</td>
</tr>
<tr>
<td>TIMERx_CAR(timerx, x=1, 2, 5, 6, 8, 11)</td>
</tr>
<tr>
<td>TIMERx_CH0CV(timerx, x=1, 2, 8, 11)</td>
</tr>
<tr>
<td>TIMERx_CH1CV(timerx, x=1, 2, 8, 11)</td>
</tr>
<tr>
<td>TIMERx_CH2CV(timerx, x=1, 2)</td>
</tr>
<tr>
<td>TIMERx_CH3CV(timerx, x=1, 2)</td>
</tr>
<tr>
<td>TIMERx_IRMP(timerx, x=8, 11)</td>
</tr>
<tr>
<td>TIMERx_DMACFG(timerx, x=1, 2)</td>
</tr>
<tr>
<td>TIMERx_DMATB(timerx, x=1, 2)</td>
</tr>
<tr>
<td>TIMERx_CFG(timerx, x=1, 2, 8, 11)</td>
</tr>
</tbody>
</table>

3.24.2. Descriptions of Peripheral functions

The description format of firmware functions are shown as below:
<table>
<thead>
<tr>
<th>Function name</th>
<th>Function description</th>
</tr>
</thead>
<tbody>
<tr>
<td>timer_deinit</td>
<td>deinit a timer</td>
</tr>
<tr>
<td>timer_init</td>
<td>initialize TIMER counter</td>
</tr>
<tr>
<td>timer_enable</td>
<td>enable a timer</td>
</tr>
<tr>
<td>timer_disable</td>
<td>disable a timer</td>
</tr>
<tr>
<td>timer_auto_reload_shadow_enable</td>
<td>enable the auto reload shadow function</td>
</tr>
<tr>
<td>timer_auto_reload_shadow_disable</td>
<td>disable the auto reload shadow function</td>
</tr>
<tr>
<td>timer_update_event_enable</td>
<td>enable the update event</td>
</tr>
<tr>
<td>timer_update_event_disable</td>
<td>disable the update event</td>
</tr>
<tr>
<td>timer_counter_alignment</td>
<td>set TIMER counter alignment mode</td>
</tr>
<tr>
<td>timer_counter_up_direction</td>
<td>set TIMER counter up direction</td>
</tr>
<tr>
<td>timer_counter_down_direction</td>
<td>set TIMER counter down direction</td>
</tr>
<tr>
<td>timer_prescaler_config</td>
<td>configure TIMER prescaler</td>
</tr>
<tr>
<td>timer_autoreload_value_config</td>
<td>configure TIMER autoreload register value</td>
</tr>
<tr>
<td>timer_counter_value_config</td>
<td>configure TIMER counter register value</td>
</tr>
<tr>
<td>timer_counter_read</td>
<td>read TIMER counter value</td>
</tr>
<tr>
<td>timer_prescaler_read</td>
<td>read TIMER prescaler value</td>
</tr>
<tr>
<td>timer_single_pulse_mode_config</td>
<td>configure TIMER single pulse mode</td>
</tr>
<tr>
<td>timer_update_source_config</td>
<td>configure TIMER update source</td>
</tr>
<tr>
<td>timer_dma_enable</td>
<td>enable the TIMER DMA</td>
</tr>
<tr>
<td>timer_dma_disable</td>
<td>disable the TIMER DMA</td>
</tr>
<tr>
<td>timer_channel_dma_request_source_select</td>
<td>channel DMA request source selection</td>
</tr>
<tr>
<td>timer_dma_transfer_config</td>
<td>configure the TIMER DMA transfer</td>
</tr>
<tr>
<td>timer_event_software_generate</td>
<td>software generate events</td>
</tr>
<tr>
<td>timer_channel_output_struct_para_init</td>
<td>initialize the parameters of TIMER channel output parameter struct with the default values</td>
</tr>
<tr>
<td>timer_channel_output_config</td>
<td>configure TIMER channel output function</td>
</tr>
<tr>
<td>timer_channel_output_mode_config</td>
<td>configure TIMER channel output compare mode</td>
</tr>
<tr>
<td>timer_channel_output_pulse_value_config</td>
<td>configure TIMER channel output pulse value</td>
</tr>
<tr>
<td>timer_channel_output_shadow_config</td>
<td>configure TIMER channel output shadow function</td>
</tr>
<tr>
<td>timer_channel_output_fast_config</td>
<td>configure TIMER channel output fast function</td>
</tr>
<tr>
<td>timer_channel_output_clear_config</td>
<td>configure TIMER channel output clear function</td>
</tr>
<tr>
<td>timer_channel_output_polarity_config</td>
<td>configure TIMER channel output polarity</td>
</tr>
<tr>
<td>timer_channel_output_state_config</td>
<td>configure TIMER channel enable state</td>
</tr>
<tr>
<td>timer_channel_input_struct</td>
<td>initialize the parameters of TIMER channel input parameter</td>
</tr>
</tbody>
</table>
Function name | Function description
---|---
para_init | struct with the default values

timer_channel_capture_config | configure TIMER input capture parameter

timer_channel_input_capture_prescaler_config | configure TIMER channel input capture prescaler value

timer_channel_capture_value_register_read | read TIMER channel capture compare register value

timer_input_pwm_capture_config | configure TIMER input pwm capture function

timer_hall_mode_config | configure TIMER hall sensor mode

timer_input_trigger_source_select | select TIMER input trigger source

timer_master_output_trigger_source_select | select TIMER master mode output trigger source

timer_slave_mode_select | select TIMER slave mode

timer_master_slave_mode_config | configure TIMER master slave mode

timer_external_trigger_config | configure TIMER external trigger input

timer_quadrature_decoder_mode_config | configure TIMER quadrature decoder mode

timer_internal_trigger_as_external_clock_config | configure TIMER the internal trigger as external clock input

timer_external_trigger_as_external_clock_config | configure TIMER the external trigger as external clock input

timer_external_clock_mode0_config | configure TIMER the external clock mode 0

timer_external_clock_mode1_config | configure TIMER the external clock mode 1

timer_external_clock_mode1_disable | disable TIMER the external clock mode 1

timer_write_chxval_register_config | configure TIMER write CHxVAL register selection

timer_flag_get | get TIMER flags

timer_flag_clear | clear TIMER flags

timer_interrupt_enable | enable the TIMER interrupt

timer_interrupt_disable | disable the TIMER interrupt

timer_interrupt_flag_get | get timer interrupt flag

timer_interrupt_flag_clear | clear TIMER interrupt flag

### Structure timer_parameter_struct

<table>
<thead>
<tr>
<th>Member name</th>
<th>Function description</th>
</tr>
</thead>
<tbody>
<tr>
<td>prescaler</td>
<td>prescaler value (0~65535)</td>
</tr>
<tr>
<td>alignedmode</td>
<td>aligned mode (TIMER_COUNTER_EDGE, TIMER_COUNTER_CENTER_DOWN, TIMER_COUNTER_CENTER_UP, TIMER_COUNTER_CENTER_BOTH)</td>
</tr>
<tr>
<td>counterdirection</td>
<td>counter direction (TIMER_COUNTER_UP, TIMER_COUNTER_DOWN)</td>
</tr>
<tr>
<td>period</td>
<td>period value (0~65535)</td>
</tr>
</tbody>
</table>
GD32L23x Firmware Library User Guide

Structure timer_oc_parameter_struct

Table 3-698. Structure timer_oc_parameter_struct

<table>
<thead>
<tr>
<th>Member name</th>
<th>Function description</th>
</tr>
</thead>
<tbody>
<tr>
<td>clockdivision</td>
<td>clock division value (TIMER_CKDIV_DIV1, TIMER_CKDIV_DIV2,</td>
</tr>
<tr>
<td></td>
<td>TIMER_CKDIV_DIV4)</td>
</tr>
</tbody>
</table>

Structure timer_ic_parameter_struct

Table 3-699. Structure timer_ic_parameter_struct

<table>
<thead>
<tr>
<th>Member name</th>
<th>Function description</th>
</tr>
</thead>
<tbody>
<tr>
<td>icpolarity</td>
<td>channel input polarity (TIMER_IC_POLARITY_RISING,</td>
</tr>
<tr>
<td></td>
<td>TIMER_IC_POLARITY_FALLING, TIMER_IC_POLARITY_BOTH_EDGE)</td>
</tr>
<tr>
<td>icselection</td>
<td>channel input mode selection (TIMER_IC_SELECTION_DIRECTTI,</td>
</tr>
<tr>
<td></td>
<td>TIMER_IC_SELECTION_INDIRECTTI, TIMER_IC_SELECTION_ITS)</td>
</tr>
<tr>
<td>icprescaler</td>
<td>channel input capture prescaler (TIMER_IC_PSC_DIV1,</td>
</tr>
<tr>
<td></td>
<td>TIMER_IC_PSC_DIV2, TIMER_IC_PSC_DIV4, TIMER_IC_PSC_DIV8)</td>
</tr>
<tr>
<td>icfilter</td>
<td>channel input capture filter control (0~15)</td>
</tr>
</tbody>
</table>

timer_deinit

The description of timer_deinit is shown as below:

Table 3-700. Function timer_deinit

<table>
<thead>
<tr>
<th>Function name</th>
<th>timer_deinit</th>
</tr>
</thead>
<tbody>
<tr>
<td>Function prototype</td>
<td>void timer_deinit(uint32_t timer_periph);</td>
</tr>
<tr>
<td>Function descriptions</td>
<td>deinit a TIMER</td>
</tr>
<tr>
<td>Precondition</td>
<td>-</td>
</tr>
<tr>
<td>The called functions</td>
<td>rcu_periph_reset_enable / rcu_periph_reset_disable</td>
</tr>
</tbody>
</table>

Input parameter(in)

<table>
<thead>
<tr>
<th>timer_periph</th>
<th>TIMER peripheral</th>
</tr>
</thead>
<tbody>
<tr>
<td>TIMERx(x=1, 2, 5, 6, 8, 11)</td>
<td>TIMER peripheral selection</td>
</tr>
</tbody>
</table>

Output parameter(out)

- -

Return value

- -

Example:

/* reset TIMER1 */
timer_deinit(TIMER1);

timer_struct_para_init

The description of timer_struct_para_init is shown as below:

<table>
<thead>
<tr>
<th>Function name</th>
<th>timer_struct_para_init</th>
</tr>
</thead>
<tbody>
<tr>
<td>Function prototype</td>
<td>void timer_struct_para_init(timer_parameter_struct* initpara);</td>
</tr>
<tr>
<td>Function descriptions</td>
<td>initialize the parameters of TIMER init parameter struct with the default values</td>
</tr>
<tr>
<td>Precondition</td>
<td>-</td>
</tr>
<tr>
<td>The called functions</td>
<td>-</td>
</tr>
</tbody>
</table>

**Example:**

```c
/* initialize TIMER init parameter struct with a default value */
timer_parameter_struct timer_initpara;
timer_struct_para_init(timer_initpara);
```

timer_init

The description of timer_init is shown as below:

<table>
<thead>
<tr>
<th>Function name</th>
<th>timer_init</th>
</tr>
</thead>
<tbody>
<tr>
<td>Function prototype</td>
<td>void timer_init(uint32_t timer_periph, timer_parameter_struct* initpara);</td>
</tr>
<tr>
<td>Function descriptions</td>
<td>initialize TIMER counter</td>
</tr>
<tr>
<td>Precondition</td>
<td>-</td>
</tr>
<tr>
<td>The called functions</td>
<td>-</td>
</tr>
</tbody>
</table>

**Example:**

```c
/* initialize TIMER counter */
timer_init(TIMER1, (timer_parameter_struct*) &timer_initpara);
```
Example:

```c
#include <usart.h>

int main(void)
{
    void *hp = NULL;
    int i;

    /* initialize TIMER1 */
    timer_parameter_struct timer_initpara;
    timer_initpara.prescaler   = 63;
    timer_initpara.alignedmode  = TIMER_COUNTER_EDGE;
    timer_initpara.counterdirection = TIMER_COUNTER_UP;
    timer_initpara.period    = 999;
    timer_initpara.clockdivision     = TIMER_CKDIV_DIV1;
    timer_init(TIMER1,&timer_initpara);

    timer_enable(TIMER1);

    for(i = 0; i < 1000; i++)
    {
        USART_SendData(hp, i);
        delay();
    }

    timer_disable(TIMER1);
}
```

### Function timer_enable

The description of timer_enable is shown as below:

**Table 3-703. Function timer_enable**

<table>
<thead>
<tr>
<th>Function name</th>
<th>timer_enable</th>
</tr>
</thead>
<tbody>
<tr>
<td>Function prototype</td>
<td>void timer_enable(uint32_t timer_periph);</td>
</tr>
<tr>
<td>Function descriptions</td>
<td>enable a timer</td>
</tr>
<tr>
<td>Precondition</td>
<td>-</td>
</tr>
<tr>
<td>The called functions</td>
<td>-</td>
</tr>
<tr>
<td>Input parameter(in)</td>
<td></td>
</tr>
<tr>
<td>timer_periph</td>
<td>TIMER peripheral</td>
</tr>
<tr>
<td>TIMERx(x=1, 2, 5, 6, 8, 11)</td>
<td>TIMER peripheral selection</td>
</tr>
<tr>
<td>Output parameter(out)</td>
<td></td>
</tr>
<tr>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>Return value</td>
<td>-</td>
</tr>
</tbody>
</table>

Example:

```c
/* enable TIMER1 */
timer_enable(TIMER1);
```
Table 3-704. Function timer_disable

<table>
<thead>
<tr>
<th>Function name</th>
<th>timer_disable</th>
</tr>
</thead>
<tbody>
<tr>
<td>Function prototype</td>
<td>void timer_disable(uint32_t timer_periph);</td>
</tr>
<tr>
<td>Function descriptions</td>
<td>disable a timer</td>
</tr>
<tr>
<td>Precondition</td>
<td>-</td>
</tr>
<tr>
<td>The called functions</td>
<td>-</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Input parameter(in)</th>
</tr>
</thead>
<tbody>
<tr>
<td>timer_periph</td>
</tr>
<tr>
<td>TIMERx(x=1, 2, 5, 6, 8, 11)</td>
</tr>
<tr>
<td>TIMER peripheral</td>
</tr>
<tr>
<td>TIMER peripheral selection</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Output parameter(out)</th>
</tr>
</thead>
<tbody>
<tr>
<td>-</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Return value</th>
</tr>
</thead>
<tbody>
<tr>
<td>-</td>
</tr>
</tbody>
</table>

Example:

/* disable TIMER1 */
timer_disable(TIMER1);

timer_auto_reload_shadow_enable

The description of timer_auto_reload_shadow_enable is shown as below:

Table 3-705. Function timer_auto_reload_shadow_enable

<table>
<thead>
<tr>
<th>Function name</th>
<th>timer_auto_reload_shadow_enable</th>
</tr>
</thead>
<tbody>
<tr>
<td>Function prototype</td>
<td>void timer_auto_reload_shadow_enable(uint32_t timer_periph);</td>
</tr>
<tr>
<td>Function descriptions</td>
<td>enable the auto reload shadow function</td>
</tr>
<tr>
<td>Precondition</td>
<td>-</td>
</tr>
<tr>
<td>The called functions</td>
<td>-</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Input parameter(in)</th>
</tr>
</thead>
<tbody>
<tr>
<td>timer_periph</td>
</tr>
<tr>
<td>TIMERx(x=1, 2, 5, 6, 8, 11)</td>
</tr>
<tr>
<td>TIMER peripheral</td>
</tr>
<tr>
<td>TIMER peripheral selection</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Output parameter(out)</th>
</tr>
</thead>
<tbody>
<tr>
<td>-</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Return value</th>
</tr>
</thead>
<tbody>
<tr>
<td>-</td>
</tr>
</tbody>
</table>

Example:

/* enable the TIMER1 auto reload shadow function */
timer_auto_reload_shadow_enable(TIMER1);
**timer_auto_reload_shadow_disable**

The description of `timer_auto_reload_shadow_disable` is shown as below:

<table>
<thead>
<tr>
<th>Function name</th>
<th>timer_auto_reload_shadow_disable</th>
</tr>
</thead>
<tbody>
<tr>
<td>Function prototype</td>
<td>void timer_auto_reload_shadow_disable(uint32_t timer_periph);</td>
</tr>
<tr>
<td>Function descriptions</td>
<td>disable the auto reload shadow function</td>
</tr>
<tr>
<td>Precondition</td>
<td>-</td>
</tr>
<tr>
<td>The called functions</td>
<td>-</td>
</tr>
</tbody>
</table>

### Input parameter (in)

<table>
<thead>
<tr>
<th>timer_periph</th>
</tr>
</thead>
<tbody>
<tr>
<td>TIMERx(x=1, 2, 5, 6, 8, 11)</td>
</tr>
<tr>
<td>TIMER peripheral selection</td>
</tr>
</tbody>
</table>

### Output parameter (out)

| - |

### Return value

| - |

**Example:**

```c
/* disable the TIMER1 auto reload shadow function */
timer_auto_reload_shadow_disable(TIMER1);
```

**timer_update_event_enable**

The description of `timer_update_event_enable` is shown as below:

<table>
<thead>
<tr>
<th>Function name</th>
<th>timer_update_event_enable</th>
</tr>
</thead>
<tbody>
<tr>
<td>Function prototype</td>
<td>void timer_update_event_enable(uint32_t timer_periph);</td>
</tr>
<tr>
<td>Function descriptions</td>
<td>enable the update event</td>
</tr>
<tr>
<td>Precondition</td>
<td>-</td>
</tr>
<tr>
<td>The called functions</td>
<td>-</td>
</tr>
</tbody>
</table>

### Input parameter (in)

<table>
<thead>
<tr>
<th>timer_periph</th>
</tr>
</thead>
<tbody>
<tr>
<td>TIMERx(x=1, 2, 5, 6, 8, 11)</td>
</tr>
<tr>
<td>TIMER peripheral selection</td>
</tr>
</tbody>
</table>

### Output parameter (out)

| - |

### Return value

| - |

**Example:**

```c
/* enable TIMER1 the update event */
timer_update_event_enable(TIMER1);
```
The description of timer_update_event_disable is shown as below:

```c
void timer_update_event_disable(uint32_t timer_periph);
```

**Function Description**

disable the update event

**Precondition**
The called functions

**Input Parameter**

<table>
<thead>
<tr>
<th>timer_periph</th>
<th>TIMER peripheral</th>
</tr>
</thead>
</table>

**Output Parameter**

- 

**Return Value**

- 

Example:

```c
/* disable TIMER1 the update event */
timer_update_event_disable(TIMER1);
```

timer_counter_alignment

The description of timer_counter_alignment is shown as below:

```c
void timer_counter_alignment(uint32_t timer_periph, uint16_t aligned);
```

**Function Description**

set TIMER counter alignment mode

**Precondition**
The called functions

**Input Parameter**

<table>
<thead>
<tr>
<th>timer_periph</th>
<th>TIMER peripheral</th>
</tr>
</thead>
</table>

<table>
<thead>
<tr>
<th>aligned</th>
<th>alignment mode</th>
</tr>
</thead>
<tbody>
<tr>
<td>TIMER_COUNTER_EDGE</td>
<td>No center-aligned mode (edge-aligned mode). The direction of the counter is specified by the DIR bit.</td>
</tr>
<tr>
<td>TIMER_COUNTER_CENTER_DOWN</td>
<td>Center-aligned and counting down assert mode. The counter counts under center aligned and channel is configured in output mode (CHxMS=00 in</td>
</tr>
</tbody>
</table>
TIMER_COUNTER_CENTER_UP

Center-aligned and counting up assert mode. The counter counts under center aligned and channel is configured in output mode (CHxMS=00 in TIMERx_CHCTL0 register). Only when the counter is counting up, compare interrupt flag of channels can be set.

TIMER_COUNTER_CENTER_BOTH

Center-aligned and counting up/down assert mode. The counter counts under center-aligned and channel is configured in output mode (CHxMS=00 in TIMERx_CHCTL0 register). Both when the counter is counting up and counting down, compare interrupt flag of channels can be set.

Table 3-710. Function timer_counter_up_direction

<table>
<thead>
<tr>
<th>Function name</th>
<th>timer_counter_up_direction</th>
</tr>
</thead>
<tbody>
<tr>
<td>Function prototype</td>
<td>void timer_counter_up_direction(uint32_t timer_periph);</td>
</tr>
<tr>
<td>Function descriptions</td>
<td>set TIMER counter up direction</td>
</tr>
<tr>
<td>Precondition</td>
<td>set TIMER counter no center-aligned mode (edge-aligned mode)</td>
</tr>
<tr>
<td>The called functions</td>
<td>-</td>
</tr>
<tr>
<td>Input parameter(in)</td>
<td></td>
</tr>
<tr>
<td>timer_periph</td>
<td>TIMER peripheral</td>
</tr>
<tr>
<td>TIMERx(x=1, 2)</td>
<td>TIMER peripheral selection</td>
</tr>
<tr>
<td>Output parameter(out)</td>
<td></td>
</tr>
<tr>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>Return value</td>
<td>-</td>
</tr>
</tbody>
</table>

Example:

/* set TIMER1 counter center-aligned and counting up assert mode */
timer_counter_alignment(TIMER1, TIMER_COUNTER_CENTER_UP);

timer_counter_up_direction

The description of timer_counter_up_direction is shown as below:

Example:

/* set TIMER1 counter up direction */
timer_counter_up_direction(TIMER1);
**timer_counter_down_direction**

The description of `timer_counter_down_direction` is shown as below:

<table>
<thead>
<tr>
<th>Function name</th>
<th>timer_counter_down_direction</th>
</tr>
</thead>
<tbody>
<tr>
<td>Function prototype</td>
<td>void timer_counter_down_direction(uint32_t timer_periph);</td>
</tr>
<tr>
<td>Function descriptions</td>
<td>set TIMER counter down direction</td>
</tr>
<tr>
<td>Precondition</td>
<td>set TIMER counter no center-aligned mode (edge-aligned mode)</td>
</tr>
<tr>
<td>The called functions</td>
<td>-</td>
</tr>
</tbody>
</table>

### Input parameter (in)
- timer_periph: TIMER peripheral
- TIMERx(x=1, 2): TIMER peripheral selection

### Output parameter (out)
- -

**Return value**
- -

Example:

```c
/* set TIMER1 counter down direction */
timer_counter_down_direction(TIMER1);
```

**timer_prescaler_config**

The description of `timer_prescaler_config` is shown as below:

<table>
<thead>
<tr>
<th>Function name</th>
<th>timer_prescaler_config</th>
</tr>
</thead>
<tbody>
<tr>
<td>Function prototype</td>
<td>void timer_prescaler_config(uint32_t timer_periph, uint16_t prescaler, uint8_t pscreload);</td>
</tr>
<tr>
<td>Function descriptions</td>
<td>configure TIMER prescaler</td>
</tr>
<tr>
<td>Precondition</td>
<td>-</td>
</tr>
<tr>
<td>The called functions</td>
<td>-</td>
</tr>
</tbody>
</table>

### Input parameter (in)
- timer_periph: TIMER peripheral
- TIMERx(x=1, 2, 5, 6, 8, 11): TIMER peripheral selection

### Input parameter (in)
- prescaler: prescaler value (0~65535)

### Input parameter (in)
- pscreload: prescaler reload mode
  - TIMER_PSC_RELOAD_NOW: the prescaler is loaded right now
  - TIMER_PSC_RELOAD: the prescaler is loaded at the next update event
### timer_prescaler_config

The description of `timer_prescaler_config` is shown as below:

**Example:**

```c
/* configure TIMER1 prescaler */
timer_prescaler_config(TIMER1, 3000, TIMER_PSC_RELOAD_NOW);
```

### timer_autoreload_value_config

The description of `timer_autoreload_value_config` is shown as below:

**Example:**

```c
/* configure TIMER1 autoreload register value */
timer_autoreload_value_config(TIMER1, 3000);
```

### timer_counter_value_config

The description of `timer_counter_value_config` is shown as below:

**Example:**

```c
/* configure TIMER1 counter register value */
timer_counter_value_config(TIMER1);
```
Precondition -
The called functions -

**timer_periph**
TIMER peripheral selection

**counter**
the counter value (0-65535)

Example:
```
/* configure TIMER1 counter register value */
timer_counter_value_config(TIMER1, 3000);
```

**timer_counter_read**

The description of `timer_counter_read` is shown as below:

<table>
<thead>
<tr>
<th>Function name</th>
<th>timer_counter_read</th>
</tr>
</thead>
<tbody>
<tr>
<td>Function prototype</td>
<td><code>uint32_t timer_counter_read(uint32_t timer_periph);</code></td>
</tr>
<tr>
<td>Function descriptions</td>
<td>read TIMER counter value</td>
</tr>
<tr>
<td>Precondition</td>
<td>-</td>
</tr>
<tr>
<td>The called functions</td>
<td>-</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>timer_periph</th>
<th>TIMER peripheral selection</th>
</tr>
</thead>
</table>

| Return value                | `counter value (0~65535)`   |

Example:
```
/* read TIMER1 counter value */

uint32_t i = 0;

i = timer_counter_read(TIMER1);
```
timer_prescaler_read

The description of timer_prescaler_read is shown as below:

Table 3-716. Function timer_prescaler_read

<table>
<thead>
<tr>
<th>Function name</th>
<th>timer_prescaler_read</th>
</tr>
</thead>
<tbody>
<tr>
<td>Function prototype</td>
<td>uint16_t timer_prescaler_read(uint32_t timer_periph);</td>
</tr>
<tr>
<td>Function descriptions</td>
<td>read TIMER prescaler value</td>
</tr>
<tr>
<td>Precondition</td>
<td>-</td>
</tr>
<tr>
<td>The called functions</td>
<td>-</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Input parameter(in)</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>timer_periph</td>
<td>TIMER peripheral</td>
</tr>
<tr>
<td>TIMEx(x=1, 2, 5, 6, 8, 11)</td>
<td>TIMER peripheral selection</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Output parameter(out)</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>-</td>
<td>-</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Return value</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>uint16_t</td>
<td>prescaler register value (0~65535)</td>
</tr>
</tbody>
</table>

Example:

```
/* read TIMER1 prescaler value */

uint16_t i = 0;
i = timer_prescaler_read(TIMER1);
```

timer_single_pulse_mode_config

The description of timer_single_pulse_mode_config is shown as below:

Table 3-717. Function timer_single_pulse_mode_config

<table>
<thead>
<tr>
<th>Function name</th>
<th>timer_single_pulse_mode_config</th>
</tr>
</thead>
<tbody>
<tr>
<td>Function prototype</td>
<td>void timer_single_pulse_mode_config(uint32_t timer_periph, uint8_t spmode);</td>
</tr>
<tr>
<td>Function descriptions</td>
<td>configure TIMER single pulse mode</td>
</tr>
<tr>
<td>Precondition</td>
<td>-</td>
</tr>
<tr>
<td>The called functions</td>
<td>-</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Input parameter(in)</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>timer_periph</td>
<td>TIMER peripheral</td>
</tr>
<tr>
<td>TIMEx(x=1, 2, 8, 11)</td>
<td>TIMER peripheral selection</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Input parameter(in)</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>spmode</td>
<td>pulse mode</td>
</tr>
<tr>
<td>TIMER_SP_MODE_SINGLE</td>
<td>single pulse mode</td>
</tr>
<tr>
<td>TIMER_SP_MODE_REPEAT</td>
<td>repetitive pulse mode</td>
</tr>
</tbody>
</table>
PETITIVE

<table>
<thead>
<tr>
<th>Output parameter(out)</th>
</tr>
</thead>
<tbody>
<tr>
<td>-</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Return value</th>
</tr>
</thead>
<tbody>
<tr>
<td>-</td>
</tr>
</tbody>
</table>

Example:

/* configure TIMER1 single pulse mode */
timer_single_pulse_mode_config(TIMER1, TIMER_SP_MODE_SINGLE);

timer_update_source_config

The description of timer_update_source_config is shown as below:

**Table 3-718. Function timer_update_source_config**

<table>
<thead>
<tr>
<th>Function name</th>
<th>timer_update_source_config</th>
</tr>
</thead>
<tbody>
<tr>
<td>Function prototype</td>
<td>void timer_update_source_config(uint32_t timer_periph, uint32_t update);</td>
</tr>
<tr>
<td>Function descriptions</td>
<td>configure TIMER update source</td>
</tr>
<tr>
<td>Precondition</td>
<td>-</td>
</tr>
<tr>
<td>The called functions</td>
<td>-</td>
</tr>
</tbody>
</table>

**Input parameter(in)**

<table>
<thead>
<tr>
<th>timer_periph</th>
<th>TIMER peripheral</th>
</tr>
</thead>
<tbody>
<tr>
<td>TIMERx(x=1, 2, 5, 6, 8, 11)</td>
<td>TIMER peripheral selection</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>update</th>
<th>update source</th>
</tr>
</thead>
<tbody>
<tr>
<td>TIMER.UPDATE_SRC_GLOBAL</td>
<td>Any of the following events generate an update interrupt or DMA request:</td>
</tr>
<tr>
<td></td>
<td>- The UPG bit is set</td>
</tr>
<tr>
<td></td>
<td>- The counter generates an overflow or underflow event</td>
</tr>
<tr>
<td></td>
<td>- The slave mode controller generates an update event</td>
</tr>
<tr>
<td>TIMERN.UPDATE_SRC_REGULAR</td>
<td>Only counter overflow/underflow generates an update interrupt or DMA request.</td>
</tr>
</tbody>
</table>

**Output parameter(out)**

| - | - |

**Return value**

| - | - |

Example:

/* configure TIMER1 update only by counter overflow/underflow */
timer_update_source_config(TIMER1, TIMER_UPDATE_SRC_REGULAR);
timer_dma_enable

The description of timer_dma_enable is shown as below:

Table 3-719. Function timer_dma_enable

<table>
<thead>
<tr>
<th>Function name</th>
<th>timer_dma_enable</th>
</tr>
</thead>
<tbody>
<tr>
<td>Function prototype</td>
<td>void timer_dma_enable(uint32_t timer_periph, uint16_t dma);</td>
</tr>
<tr>
<td>Function descriptions</td>
<td>enable the TIMER DMA</td>
</tr>
<tr>
<td>Precondition</td>
<td>-</td>
</tr>
<tr>
<td>The called functions</td>
<td>-</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Input parameter(in)</th>
</tr>
</thead>
<tbody>
<tr>
<td>timer_periph</td>
</tr>
<tr>
<td>TIMERx</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>dma</th>
<th>timer DMA source enable</th>
</tr>
</thead>
<tbody>
<tr>
<td>TIMER_DMA_UPD</td>
<td>update DMA enable, TIMERx(x=1, 2, 5, 6, 8, 11)</td>
</tr>
<tr>
<td>TIMER_DMA_CH0D</td>
<td>channel 0 DMA enable, TIMERx(x=1, 2, 8, 11)</td>
</tr>
<tr>
<td>TIMER_DMA_CH1D</td>
<td>channel 1 DMA enable, TIMERx(x=1, 2, 8, 11)</td>
</tr>
<tr>
<td>TIMER_DMA_CH2D</td>
<td>channel 2 DMA enable, TIMERx(x=1, 2)</td>
</tr>
<tr>
<td>TIMER_DMA_CH3D</td>
<td>channel 3 DMA enable, TIMERx(x=1, 2)</td>
</tr>
<tr>
<td>TIMER_DMA_TRGD</td>
<td>trigger DMA enable, TIMERx(x=1, 2, 8, 11)</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Output parameter(out)</th>
</tr>
</thead>
<tbody>
<tr>
<td>-</td>
</tr>
</tbody>
</table>

| Return value | - |

Example:

/* enable the TIMER1 update DMA */
timer_dma_enable(TIMER1, TIMER_DMA_UPD);

timer_dma_disable

The description of timer_dma_disable is shown as below:

Table 3-720. Function timer_dma_disable

<table>
<thead>
<tr>
<th>Function name</th>
<th>timer_dma_disable</th>
</tr>
</thead>
<tbody>
<tr>
<td>Function prototype</td>
<td>void timer_dma_disable(uint32_t timer_periph, uint16_t dma);</td>
</tr>
<tr>
<td>Function descriptions</td>
<td>disable the TIMER DMA</td>
</tr>
<tr>
<td>Precondition</td>
<td>-</td>
</tr>
<tr>
<td>The called functions</td>
<td>-</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Input parameter(in)</th>
</tr>
</thead>
<tbody>
<tr>
<td>timer_periph</td>
</tr>
<tr>
<td>TIMERx</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Input parameter(in)</th>
</tr>
</thead>
<tbody>
<tr>
<td>-</td>
</tr>
</tbody>
</table>

<p>| Return value | - |</p>
<table>
<thead>
<tr>
<th>dma</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>TIMER_DMA_UPD</td>
<td>timer DMA source disable</td>
</tr>
<tr>
<td>TIMER_DMA_UPD</td>
<td>update DMA enable, TIMERx(x=1, 2, 5, 6, 8, 11)</td>
</tr>
<tr>
<td>TIMER_DMA_CH0D</td>
<td>channel 0 DMA enable, TIMERx(x=1, 2, 8, 11)</td>
</tr>
<tr>
<td>TIMER_DMA_CH1D</td>
<td>channel 1 DMA enable, TIMERx(x=1, 2, 8, 11)</td>
</tr>
<tr>
<td>TIMER_DMA_CH2D</td>
<td>channel 2 DMA enable, TIMERx(x=1, 2)</td>
</tr>
<tr>
<td>TIMER_DMA_CH3D</td>
<td>channel 3 DMA enable, TIMERx(x=1, 2)</td>
</tr>
<tr>
<td>TIMER_DMA_TRGD</td>
<td>trigger DMA enable, TIMERx(x=1, 2, 8, 11)</td>
</tr>
</tbody>
</table>

**Output parameter(out)**

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>-</td>
<td></td>
</tr>
</tbody>
</table>

**Return value**

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>-</td>
<td></td>
</tr>
</tbody>
</table>

Example:

/* disable the TIMER1 update DMA */
timer_dma_disable(TIMER1, TIMER_DMA_UPD);

**timer_channel_dma_request_source_select**

The description of timer_channel_dma_request_source_select is shown as below:

**Table 3-721. Function timer_channel_dma_request_source_select**

<table>
<thead>
<tr>
<th>Function name</th>
<th>timer_channel_dma_request_source_select</th>
</tr>
</thead>
<tbody>
<tr>
<td>Function prototype</td>
<td>void timer_channel_dma_request_source_select(uint32_t timer_periph, uint32_t dma_request);</td>
</tr>
<tr>
<td>Function descriptions</td>
<td>channel DMA request source selection</td>
</tr>
<tr>
<td>Precondition</td>
<td>-</td>
</tr>
<tr>
<td>The called functions</td>
<td>-</td>
</tr>
</tbody>
</table>

**Input parameter(in)**

<table>
<thead>
<tr>
<th>timer_periph</th>
<th>TIMER peripheral</th>
</tr>
</thead>
<tbody>
<tr>
<td>TIMERx(x=1, 2)</td>
<td>TIMER peripheral selection</td>
</tr>
</tbody>
</table>

**Input parameter(in)**

<table>
<thead>
<tr>
<th>dma_request</th>
<th>channel DMA request source selection</th>
</tr>
</thead>
<tbody>
<tr>
<td>TIMER_DMAREQUEST_CHANNELEVENT</td>
<td>DMA request of channel n is sent when channel y event occurs</td>
</tr>
<tr>
<td>TIMER_DMAREQUEST_UPDATEEVENT</td>
<td>DMA request of channel n is sent when update event occurs</td>
</tr>
</tbody>
</table>

**Output parameter(out)**

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>-</td>
<td></td>
</tr>
</tbody>
</table>

**Return value**

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>-</td>
<td></td>
</tr>
</tbody>
</table>

Example:

/* TIMER1 channel DMA request of channel n is sent when channel y event occurs */
timer_channel_dma_request_source_select(TIMER1, TIMER_DMAREQUEST_CHANNELEVENT);

timer_dma_transfer_config

The description of timer_dma_transfer_config is shown as below:

**Table 3-722. Function timer_dma_transfer_config**

<table>
<thead>
<tr>
<th>Function name</th>
<th>timer_dma_transfer_config</th>
</tr>
</thead>
<tbody>
<tr>
<td>Function prototype</td>
<td>void timer_dma_transfer_config(uint32_t timer_periph, uint32_t dma_baseaddr, uint32_t dma_length);</td>
</tr>
<tr>
<td>Function descriptions</td>
<td>configure the TIMER DMA transfer</td>
</tr>
<tr>
<td>Precondition</td>
<td>-</td>
</tr>
<tr>
<td>The called functions</td>
<td>-</td>
</tr>
<tr>
<td><strong>Input parameter(in)</strong></td>
<td></td>
</tr>
<tr>
<td>timer_periph</td>
<td>TIMER peripheral</td>
</tr>
<tr>
<td>TIMERx(x=1, 2)</td>
<td>TIMER peripheral selection</td>
</tr>
<tr>
<td><strong>Input parameter(in)</strong></td>
<td></td>
</tr>
<tr>
<td>dma_baseaddr</td>
<td>DMA transfer access start address</td>
</tr>
<tr>
<td>TIMER_DMACFG_DMA TA_CTL0</td>
<td>DMA transfer address is TIMER_CTL0, TIMERx(x=1, 2)</td>
</tr>
<tr>
<td>TIMER_DMACFG_DMA TA_CTL1</td>
<td>DMA transfer address is TIMER_CTL1, TIMERx(x=1, 2)</td>
</tr>
<tr>
<td>TIMER_DMACFG_DMA TA_SMCFG</td>
<td>DMA transfer address is TIMER_SMCFG, TIMERx(x=1, 2)</td>
</tr>
<tr>
<td>TIMER_DMACFG_DMA TA_DMAINTEN</td>
<td>DMA transfer address is TIMER_DMAINTEN, TIMERx(x=1, 2)</td>
</tr>
<tr>
<td>TIMER_DMACFG_DMA TA_INTF</td>
<td>DMA transfer address is TIMER_INTF, TIMERx(x=1, 2)</td>
</tr>
<tr>
<td>TIMER_DMACFG_DMA TA_SWEVG</td>
<td>DMA transfer address is TIMER_SWEVG, TIMERx(x=1, 2)</td>
</tr>
<tr>
<td>TIMER_DMACFG_DMA TA_CHCTL0</td>
<td>DMA transfer address is TIMER_CHCTL0, TIMERx(x=1, 2)</td>
</tr>
<tr>
<td>TIMER_DMACFG_DMA TA_CHCTL1</td>
<td>DMA transfer address is TIMER_CHCTL1, TIMERx(x=1, 2)</td>
</tr>
<tr>
<td>TIMER_DMACFG_DMA TA_CHCTL2</td>
<td>DMA transfer address is TIMER_CHCTL2, TIMERx(x=1, 2)</td>
</tr>
<tr>
<td>TIMER_DMACFG_DMA TA_CNT</td>
<td>DMA transfer address is TIMER_CNT, TIMERx(x=1, 2)</td>
</tr>
<tr>
<td>TIMER_DMACFG_DMA TA_PSC</td>
<td>DMA transfer address is TIMER_PSC, TIMERx(x=1, 2)</td>
</tr>
<tr>
<td>TIMER_DMACFG_DMA TA_CAR</td>
<td>DMA transfer address is TIMER_CAR, TIMERx(x=1, 2)</td>
</tr>
<tr>
<td>TIMER_DMACFG_DMA TA_CH0CV</td>
<td>DMA transfer address is TIMER_CH0CV, TIMERx(x=1, 2)</td>
</tr>
</tbody>
</table>
TA_CH0CV

**Input parameter (in)**

- dma_lenth: DMA transfer count

**Output parameter (out)**

- 

**Return value**

- 

Example:

```c
/* configure the TIMER1 DMA transfer */
timer_dma_transfer_config(TIMER1, TIMER_DMACFG_DMATA_CTL0, TIMER_DMACFG_DMATC_5TRANSFER);
```

**timer_event_software_generate**

The description of `timer_event_software_generate` is shown as below:

### Table 3-723. Function `timer_event_software_generate`

<table>
<thead>
<tr>
<th>Function name</th>
<th><code>timer_event_software_generate</code></th>
</tr>
</thead>
<tbody>
<tr>
<td>Function prototype</td>
<td><code>void timer_event_software_generate(uint32_t timer_periph, uint16_t event);</code></td>
</tr>
<tr>
<td>Function descriptions</td>
<td>Software generate events</td>
</tr>
<tr>
<td>Precondition</td>
<td>-</td>
</tr>
<tr>
<td>The called functions</td>
<td>-</td>
</tr>
</tbody>
</table>

**Input parameter (in)**

- `timer_periph`: TIMER peripheral
- `event`: the timer software event generation sources

**Input parameter (in)**

- `TIMER_EVENT_SRC_UPG`: update event, TIMERx(x=1, 2, 5, 6, 8, 11)
- `TIMER_EVENT_SRC_CH0G`: channel 0 capture or compare event generation, TIMERx(x=1, 2, 8, 11)
- `TIMER_EVENT_SRC_CH1G`: channel 1 capture or compare event generation, TIMERx(x=1, 2, 8, 11)
TIMER_EVENT_SRC_C2G
- channel 2 capture or compare event generation, TIMERx(x=1, 2)

TIMER_EVENT_SRC_C3G
- channel 3 capture or compare event generation, TIMERx(x=1, 2)

TIMER_EVENT_SRC_TGG
- trigger event generation, TIMERx(x=1, 2, 8, 11)

Output parameter(out)
- -

Return value - -

Example:

/* software generate update event*/
timer_event_software_generate(TIMERO, TIMER_EVENT_SRC_UPG);

timer_channel_output_struct_para_init

The description of timer_channel_output_struct_para_init is shown as below:

Table 3-724. Function timer_channel_output_struct_para_init

```
<table>
<thead>
<tr>
<th>Function name</th>
<th>timer_channel_output_struct_para_init</th>
</tr>
</thead>
<tbody>
<tr>
<td>Function prototype</td>
<td>void timer_channel_output_struct_para_init(timer_oc_parameter_struct* ocpara);</td>
</tr>
<tr>
<td>Function descriptions</td>
<td>initialize the parameters of TIMER channel output parameter struct with the default values</td>
</tr>
<tr>
<td>Precondition</td>
<td>-</td>
</tr>
<tr>
<td>The called functions</td>
<td>-</td>
</tr>
</tbody>
</table>
```

Input parameter(in)

ocpara TIMERO channel output parameter struct, the structure members can refer to Table 3-698. Structure timer_oc_parameter_struct

Output parameter(out)
- -

Return value - -

Example:

/* initialize TIMERO channel output parameter struct with a default value */
timer_oc_parameter_struct timer_ocinitpara;
timer_channel_output_struct_para_init(timer_ocinitpara);

timer_channel_output_config

The description of timer_channel_output_config is shown as below:
Table 3-725. Function timer_channel_output_config

<table>
<thead>
<tr>
<th>Function name</th>
<th>timer_channel_output_config</th>
</tr>
</thead>
<tbody>
<tr>
<td>Function prototype</td>
<td>void timer_channel_output_config(uint32_t timer_periph, uint16_t channel, timer_oc_parameter_struct* ocpara);</td>
</tr>
<tr>
<td>Function descriptions</td>
<td>configure TIMER channel output function</td>
</tr>
<tr>
<td>Precondition</td>
<td>-</td>
</tr>
<tr>
<td>The called functions</td>
<td>-</td>
</tr>
</tbody>
</table>

**Input parameter (in)**

- **timer_periph**: TIMER peripheral
  - **TIMERx**: please refer to the following parameters

**Input parameter (in)**

- **channel**: channel to be configured
  - **TIMER_CHANNEL_0**: TIMER channel 0 TIMERx(x=1, 2, 8, 11)
  - **TIMER_CHANNEL_1**: TIMER channel 1 TIMERx(x=1, 2, 8, 11)
  - **TIMER_CHANNEL_2**: TIMER channel 2 (TIMERx(x=1, 2))
  - **TIMER_CHANNEL_3**: TIMER channel 3 (TIMERx(x=1, 2))

**Input parameter (in)**

- **ocpara**: TIMER channel output parameter struct, the structure members can refer to Table 3-698. Structure timer_oc_parameter_struct.

**Output parameter (out)**

- -

**Return value**

- -

Example:

```c
/* configure TIMER1 channel 0 output function */
timer_oc_parameter_struct timer_ocinitpara;
timer_ocinitpara.outputstate = TIMER_CCX_ENABLE;
timer_ocinitpara.ocpolarity = TIMER_OC_POLARITY_HIGH;
timer_channel_output_config(TIMER1, TIMER_CHANNEL_0, &timer_ocinitpara);
```

**timer_channel_output_mode_config**

The description of timer_channel_output_mode_config is shown as below:

Table 3-726. Function timer_channel_output_mode_config

<table>
<thead>
<tr>
<th>Function name</th>
<th>timer_channel_output_mode_config</th>
</tr>
</thead>
<tbody>
<tr>
<td>Function prototype</td>
<td>void timer_channel_output_mode_config(uint32_t timer_periph, uint16_t channel, uint16_t ocmode);</td>
</tr>
<tr>
<td>Function descriptions</td>
<td>configure TIMER channel output compare mode</td>
</tr>
<tr>
<td>Precondition</td>
<td>-</td>
</tr>
<tr>
<td>The called functions</td>
<td>-</td>
</tr>
</tbody>
</table>
Input parameter(in)

<table>
<thead>
<tr>
<th>parameter</th>
<th>description</th>
</tr>
</thead>
<tbody>
<tr>
<td>timer_periph</td>
<td>TIMER peripheral</td>
</tr>
<tr>
<td>TIMERx</td>
<td>please refer to the following parameters</td>
</tr>
</tbody>
</table>

Input parameter(in)

<table>
<thead>
<tr>
<th>parameter</th>
<th>description</th>
</tr>
</thead>
<tbody>
<tr>
<td>channel</td>
<td>channel to be configured</td>
</tr>
<tr>
<td>TIMER_CH_0</td>
<td>TIMER channel 0 (TIMERx (x=1, 2, 8, 11))</td>
</tr>
<tr>
<td>TIMER_CH_1</td>
<td>TIMER channel 1 (TIMERx (x=1, 2, 8, 11))</td>
</tr>
<tr>
<td>TIMER_CH_2</td>
<td>TIMER channel 2 (TIMERx (x=1, 2))</td>
</tr>
<tr>
<td>TIMER_CH_3</td>
<td>TIMER channel 3 (TIMERx (x=1, 2))</td>
</tr>
</tbody>
</table>

Input parameter(in)

<table>
<thead>
<tr>
<th>parameter</th>
<th>description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ocmode</td>
<td>channel output compare mode</td>
</tr>
<tr>
<td>TIMER_OC_MODE_TIMING</td>
<td>timing mode</td>
</tr>
<tr>
<td>TIMER_OC_MODE_ACTIVE</td>
<td>set the channel output</td>
</tr>
<tr>
<td>TIMER_OC_MODE_INACTIVE</td>
<td>clear the channel output</td>
</tr>
<tr>
<td>TIMER_OC_MODE_TOGGLE</td>
<td>toggle on match</td>
</tr>
<tr>
<td>TIMER_OC_MODE_LO W</td>
<td>force low mode</td>
</tr>
<tr>
<td>TIMER_OC_MODE_HIG H</td>
<td>force high mode</td>
</tr>
<tr>
<td>TIMER_OC_MODE_PWM M0</td>
<td>PWM mode 0</td>
</tr>
<tr>
<td>TIMER_OC_MODE_PWM M1</td>
<td>PWM mode 1</td>
</tr>
</tbody>
</table>

Output parameter(out)

<table>
<thead>
<tr>
<th>parameter</th>
<th>description</th>
</tr>
</thead>
<tbody>
<tr>
<td>-</td>
<td>-</td>
</tr>
</tbody>
</table>

Return value

<table>
<thead>
<tr>
<th>parameter</th>
<th>description</th>
</tr>
</thead>
<tbody>
<tr>
<td>-</td>
<td>-</td>
</tr>
</tbody>
</table>

Example:

```c
/* configure TIMER1 channel PWM 0 mode */
timer_channel_output_mode_config(TIMER1, TIMER_CH_0, TIMER_OC_MODE_PWM0);
```

timer_channel_output_pulse_value_config

The description of timer_channel_output_pulse_value_config is shown as below:

Table 3-727. Function timer_channel_output_pulse_value_config

<table>
<thead>
<tr>
<th>Function name</th>
<th>timer_channel_output_pulse_value_config</th>
</tr>
</thead>
<tbody>
<tr>
<td>Function prototype</td>
<td>void timer_channel_output_pulse_value_config(uint32_t timer_periph,</td>
</tr>
</tbody>
</table>
Function descriptions
configure TIMER channel output pulse value

Precondition
-

The called functions
-

Input parameter (in)

<table>
<thead>
<tr>
<th>timer_periph</th>
<th>TIMER peripheral</th>
</tr>
</thead>
<tbody>
<tr>
<td>TIMERx</td>
<td>please refer to the following parameters</td>
</tr>
</tbody>
</table>

Input parameter (in)

channel
channel to be configured

<table>
<thead>
<tr>
<th>TIMER_CHANNEL_0</th>
<th>TIMER channel 0 (TIMERx (x=1, 2, 8, 11))</th>
</tr>
</thead>
<tbody>
<tr>
<td>TIMER_CHANNEL_1</td>
<td>TIMER channel 1 (TIMERx (x=1, 2, 8, 11))</td>
</tr>
<tr>
<td>TIMER_CHANNEL_2</td>
<td>TIMER channel 2 (TIMERx (x=1, 2))</td>
</tr>
<tr>
<td>TIMER_CHANNEL_3</td>
<td>TIMER channel 3 (TIMERx (x=1, 2))</td>
</tr>
</tbody>
</table>

Input parameter (in)

pulse
channel output pulse value (0~65535)

Output parameter (out)

Return value
-

Example:

/* configure TIMER1 channel 0 output pulse value */
timer_channel_output_pulse_value_config(TIMER1, TIMER_CHANNEL_0, 399);

timer_channel_output_shadow_config

The description of timer_channel_output_shadow_config is shown as below:

Table 3-728. Function timer_channel_output_shadow_config

<table>
<thead>
<tr>
<th>Function name</th>
<th>timer_channel_output_shadow_config</th>
</tr>
</thead>
<tbody>
<tr>
<td>Function prototype</td>
<td>void timer_channel_output_shadow_config(uint32_t timer_periph, uint16_t channel, uint16_t ocshadow);</td>
</tr>
<tr>
<td>Function descriptions</td>
<td>configure TIMER channel output shadow function</td>
</tr>
<tr>
<td>Precondition</td>
<td>-</td>
</tr>
<tr>
<td>The called functions</td>
<td>-</td>
</tr>
</tbody>
</table>

Input parameter (in)

<table>
<thead>
<tr>
<th>timer_periph</th>
<th>TIMER peripheral</th>
</tr>
</thead>
<tbody>
<tr>
<td>TIMERx</td>
<td>please refer to the following parameters</td>
</tr>
</tbody>
</table>

Input parameter (in)

channel
channel to be configured

<table>
<thead>
<tr>
<th>TIMER_CHANNEL_0</th>
<th>TIMER channel 0 (TIMERx (x=1, 2, 8, 11))</th>
</tr>
</thead>
<tbody>
<tr>
<td>TIMER_CHANNEL_1</td>
<td>TIMER channel 1 (TIMERx (x=1, 2, 8, 11))</td>
</tr>
<tr>
<td>TIMER_CHANNEL_2</td>
<td>TIMER channel 2 (TIMERx (x=1, 2))</td>
</tr>
</tbody>
</table>
GD32L23x Firmware Library User Guide

**TIMER CH 3**
IMER channel 3 (TIMERx (x=1, 2))

<table>
<thead>
<tr>
<th>Input parameter</th>
<th>channel output shadow state</th>
</tr>
</thead>
<tbody>
<tr>
<td>ocshadow</td>
<td>channel output shadow state enable</td>
</tr>
<tr>
<td>myshadow</td>
<td>channel output shadow state disable</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Output parameter</th>
<th>channel output shadow state enable</th>
</tr>
</thead>
<tbody>
<tr>
<td>ocshadow</td>
<td>channel output shadow state disable</td>
</tr>
</tbody>
</table>

**Return value**

**Example:**

```c
/*configure TIMER1 channel 0 output shadow function*/
timer_channel_output_shadow_config(TIMERA1, TIMER_CH_0, TIMER_OC_SHADOW_ENABLE);
```

**timer_channel_output_fast_config**

The description of timer_channel_output_fast_config is shown as below:

<table>
<thead>
<tr>
<th>Function name</th>
<th>timer_channel_output_fast_config</th>
</tr>
</thead>
<tbody>
<tr>
<td>Function prototype</td>
<td>void timer_channel_output_fast_config(uint32_t timer_periph, uint16_t channel, uint16_t ocfast);</td>
</tr>
<tr>
<td>Function descriptions</td>
<td>configure TIMER channel output fast function</td>
</tr>
<tr>
<td>Precondition</td>
<td>-</td>
</tr>
<tr>
<td>The called functions</td>
<td>-</td>
</tr>
</tbody>
</table>

**Input parameter**

<table>
<thead>
<tr>
<th>timer_periph</th>
<th>TIMER peripheral</th>
</tr>
</thead>
<tbody>
<tr>
<td>TIMERx</td>
<td>please refer to the following parameters</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Input parameter</th>
<th>channel to be configured</th>
</tr>
</thead>
<tbody>
<tr>
<td>channel</td>
<td>channel to be configured</td>
</tr>
<tr>
<td>TIMER_CH_0</td>
<td>TIMER channel 0 (TIMERx (x=1, 2, 8, 11))</td>
</tr>
<tr>
<td>TIMER_CH_1</td>
<td>TIMER channel 1 (TIMERx (x=1, 2, 8, 11))</td>
</tr>
<tr>
<td>TIMER_CH_2</td>
<td>TIMER channel 2 (TIMERx (x=1, 2))</td>
</tr>
<tr>
<td>TIMER_CH_3</td>
<td>TIMER channel 3 (TIMERx (x=1, 2))</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Input parameter</th>
<th>channel output fast function</th>
</tr>
</thead>
<tbody>
<tr>
<td>ocfast</td>
<td>channel output fast function</td>
</tr>
<tr>
<td>TIMER_OC_FAST_ENABLE</td>
<td>channel output fast function enable</td>
</tr>
<tr>
<td>TIMER_OC_FAST_DISABLE</td>
<td>channel output fast function disable</td>
</tr>
</tbody>
</table>
Output parameter(out)
- -

Return value
- -

Example:

/* configure TIMER1 channel 0 output fast function */
timer_channel_output_fast_config(TIMER1, TIMER_CH_0, TIMER_OC_FAST_ENABLE);

timer_channel_output_clear_config

The description of timer_channel_output_clear_config is shown as below:

Table 3-730. Function timer_channel_output_clear_config

<table>
<thead>
<tr>
<th>Function name</th>
<th>timer_channel_output_clear_config</th>
</tr>
</thead>
<tbody>
<tr>
<td>Function prototype</td>
<td>void timer_channel_output_clear_config(uint32_t timer_periph, uint16_t channel, uint16_t occlear);</td>
</tr>
<tr>
<td>Function descriptions</td>
<td>configure TIMER channel output clear function</td>
</tr>
<tr>
<td>Precondition</td>
<td>-</td>
</tr>
<tr>
<td>The called functions</td>
<td>-</td>
</tr>
</tbody>
</table>

Input parameter(in)

- timer_periph: TIMER periphery
  - TIMERx: please refer to the following parameters

Input parameter(in)

- channel: channel to be configured
  - TIMER_CH_0: TIMER channel 0 (TIMERx (x=1, 2))
  - TIMER_CH_1: TIMER channel 1 (TIMERx (x=1, 2))
  - TIMER_CH_2: TIMER channel 2 (TIMERx (x=1, 2))
  - TIMER_CH_3: TIMER channel 3 (TIMERx (x=1, 2))

Input parameter(in)

- occlear: channel output clear function
  - TIMER_OC_CLEAR_ENABLE: channel output clear function enable
  - TIMER_OC_CLEAR_DISABLE: channel output clear function disable

Output parameter(out)
- -

Return value
- -

Example:

/* configure TIMER1 channel 0 output clear function */
timer_channel_output_clear_config(TIMER1, TIMER_CH_0,
timer_channel_output_polarity_config

The description of timer_channel_output_polarity_config is shown as below:

Table 3-731. Function timer_channel_output_polarity_config

<table>
<thead>
<tr>
<th>Function name</th>
<th>timer_channel_output_polarity_config</th>
</tr>
</thead>
<tbody>
<tr>
<td>Function prototype</td>
<td>void timer_channel_output_polarity_config(uint32_t timer_periph, uint16_t channel, uint16_t ocpolarity);</td>
</tr>
<tr>
<td>Function descriptions</td>
<td>configure TIMER channel output polarity</td>
</tr>
<tr>
<td>Precondition</td>
<td>-</td>
</tr>
<tr>
<td>The called functions</td>
<td>-</td>
</tr>
<tr>
<td><strong>Input parameter</strong>(in)</td>
<td></td>
</tr>
<tr>
<td>timer_periph</td>
<td>TIMER peripheral</td>
</tr>
<tr>
<td>TIMERx</td>
<td>please refer to the following parameters</td>
</tr>
<tr>
<td>channel</td>
<td>channel to be configured</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>TIMER_CH_0</td>
<td>TIMER channel 0 (TIMERx (x=1, 2, 8, 11))</td>
</tr>
<tr>
<td>TIMER_CH_1</td>
<td>TIMER channel 1 (TIMERx (x=1, 2, 8, 11))</td>
</tr>
<tr>
<td>TIMER_CH_2</td>
<td>TIMER channel 2 (TIMERx(x=1, 2))</td>
</tr>
<tr>
<td>TIMER_CH_3</td>
<td>TIMER channel 3 (TIMERx (x=1, 2))</td>
</tr>
<tr>
<td><strong>Input parameter</strong>(in)</td>
<td></td>
</tr>
<tr>
<td>ocpolarity</td>
<td>channel output polarity</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>TIMER_OC_POLARITY _HIGH</td>
<td>channel output polarity is high</td>
</tr>
<tr>
<td>TIMER_OC_POLARITY _LOW</td>
<td>channel output polarity is low</td>
</tr>
<tr>
<td><strong>Output parameter</strong>(out)</td>
<td></td>
</tr>
<tr>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>Return value</td>
<td>-</td>
</tr>
</tbody>
</table>

Example:

/* configure TIMER1 channel 0 output polarity */

timer_channel_output_polarity_config(TIMER1, TIMER_CH_0, TIMER_OC_POLARITY_HIGH);

timer_channel_output_state_config

The description of timer_channel_output_state_config is shown as below:

Table 3-732. Function timer_channel_output_state_config

| Function name                                      | timer_channel_output_state_config |

477
Function prototype: void timer_channel_output_state_config(uint32_t timer_periph, uint16_t channel, uint32_t state);

Function descriptions: configure TIMER channel enable state

Precondition -
The called functions -

Input parameter (in)

<table>
<thead>
<tr>
<th>timer_periph</th>
<th>TIMER peripheral</th>
</tr>
</thead>
<tbody>
<tr>
<td>channel</td>
<td>TIMER channel to be configured</td>
</tr>
<tr>
<td>TIMER_CH_0</td>
<td>TIMER channel 0 (TIMERx (x=1, 2, 8, 11))</td>
</tr>
<tr>
<td>TIMER_CH_1</td>
<td>TIMER channel 1 (TIMERx (x=1, 2, 8, 11))</td>
</tr>
<tr>
<td>TIMER_CH_2</td>
<td>TIMER channel 2 (TIMERx (x=1, 2))</td>
</tr>
<tr>
<td>TIMER_CH_3</td>
<td>TIMER channel 3 (TIMERx (x=1, 2))</td>
</tr>
</tbody>
</table>

Input parameter (in)

<table>
<thead>
<tr>
<th>state</th>
<th>TIMER channel enable state</th>
</tr>
</thead>
<tbody>
<tr>
<td>TIMER_CCX_ENABLE</td>
<td>channel enable</td>
</tr>
<tr>
<td>TIMER_CCX_DISABLE</td>
<td>channel disable</td>
</tr>
</tbody>
</table>

Output parameter (out)

| -            | -                          |

Return value -

Example:

/* configure TIMER1 channel 0 enable state */
timer_channel_output_state_config(TIMER1, TIMER_CH_0, TIMER_CCX_ENABLE);

**timer_channel_input_struct_para_init**

The description of timer_channel_input_struct_para_init is shown as below:

<table>
<thead>
<tr>
<th>Function name</th>
<th>timer_channel_input_struct_para_init</th>
</tr>
</thead>
<tbody>
<tr>
<td>Function prototype</td>
<td>void timer_channel_input_struct_para_init(timer_ic_parameter_struct* icpara);</td>
</tr>
<tr>
<td>Function descriptions</td>
<td>initialize the parameters of TIMER channel input parameter struct with the default values</td>
</tr>
<tr>
<td>Precondition</td>
<td>-</td>
</tr>
<tr>
<td>The called functions</td>
<td>-</td>
</tr>
</tbody>
</table>

Input parameter (in)

| icpara | TIMER channel input parameter struct, the structure members can refer to Table 3-699. Structure timer_ic_parameter_struct |

Output parameter (out)
Example:

    /* initialize TIMER channel input parameter struct with a default value */
    timer_ic_parameter_struct timer_icinitpara;
    timer_channel_input_struct_para_init(&timer_icinitpara);

**timer_input_capture_config**

The description of `timer_input_capture_config` is shown as below:

<table>
<thead>
<tr>
<th>Function name</th>
<th>timer_input_capture_config</th>
</tr>
</thead>
<tbody>
<tr>
<td>Function prototype</td>
<td>void timer_input_capture_config(uint32_t timer_periph, uint16_t channel, timer_ic_parameter_struct* icpara);</td>
</tr>
<tr>
<td>Function descriptions</td>
<td>configure TIMER input capture parameter</td>
</tr>
<tr>
<td>Precondition</td>
<td>-</td>
</tr>
<tr>
<td>The called functions</td>
<td>timer_channel_input_capture_prescaler_config</td>
</tr>
</tbody>
</table>

**Input parameter(in)**

<table>
<thead>
<tr>
<th>channel</th>
<th>channel to be configured</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>TIMER_CHANNEL_0</code></td>
<td>TIMER channel 0 (TIMERx (x=1, 2, 8, 11))</td>
</tr>
<tr>
<td><code>TIMER_CHANNEL_1</code></td>
<td>TIMER channel 1 (TIMERx (x=1, 2, 8, 11))</td>
</tr>
<tr>
<td><code>TIMER_CHANNEL_2</code></td>
<td>TIMER channel 2 (TIMERx (x=1, 2))</td>
</tr>
<tr>
<td><code>TIMER_CHANNEL_3</code></td>
<td>TIMER channel 3 (TIMERx (x=1, 2))</td>
</tr>
</tbody>
</table>

**Input parameter(in)**

| icpara | TIMER channel input parameter struct, the structure members can refer to *Table 3-699. Structure timer_ic_parameter_struct.* |

**Output parameter(out)**

| - | - |

| Return value | - |
GD32L23x Firmware Library User Guide

timer_icinitpara.icselection = TIMER_IC_SELECTION_DIRECTTI;

timer_icinitpara.icprescaler = TIMER_IC_PSC_DIV1;

timer_icinitpara.icfilter = 0x0;

timer_input_capture_config(TIMER1, TIMER_CH_0, &timer_icinitpara);

timer_channel_input_capture_prescaler_config

The description of timer_channel_input_capture_prescaler_config is shown as below:

Table 3-735. Function timer_channel_input_capture_prescaler_config

<table>
<thead>
<tr>
<th>Function name</th>
<th>timer_channel_input_capture_prescaler_config</th>
</tr>
</thead>
<tbody>
<tr>
<td>Function prototype</td>
<td>void timer_channel_input_capture_prescaler_config(uint32_t timer_periph, uint16_t channel, uint16_t prescaler);</td>
</tr>
<tr>
<td>Function descriptions</td>
<td>configure TIMER channel input capture prescaler value</td>
</tr>
<tr>
<td>Precondition</td>
<td>-</td>
</tr>
<tr>
<td>The called functions</td>
<td>-</td>
</tr>
</tbody>
</table>

**Input parameter(in)**

<table>
<thead>
<tr>
<th>timer_periph</th>
<th>TIMER peripheral</th>
</tr>
</thead>
</table>

**Input parameter(in)**

<table>
<thead>
<tr>
<th>channel</th>
<th>channel to be configured</th>
</tr>
</thead>
<tbody>
<tr>
<td>TIMER_CH_0</td>
<td>TIMER channel 0 (TIMERx (x=1, 2, 8, 11))</td>
</tr>
<tr>
<td>TIMER_CH_1</td>
<td>TIMER channel 1 (TIMERx (x=1, 2, 8, 11))</td>
</tr>
<tr>
<td>TIMER_CH_2</td>
<td>TIMER channel 2 (TIMERx (x=1, 2))</td>
</tr>
<tr>
<td>TIMER_CH_3</td>
<td>TIMER channel 3 (TIMERx (x=1, 2))</td>
</tr>
</tbody>
</table>

**Input parameter(in)**

<table>
<thead>
<tr>
<th>prescaler</th>
<th>channel input capture prescaler value</th>
</tr>
</thead>
<tbody>
<tr>
<td>TIMER_IC_PSC_DIV1</td>
<td>no prescaler</td>
</tr>
<tr>
<td>TIMER_IC_PSC_DIV2</td>
<td>divided by 2</td>
</tr>
<tr>
<td>TIMER_IC_PSC_DIV4</td>
<td>divided by 4</td>
</tr>
<tr>
<td>TIMER_IC_PSC_DIV8</td>
<td>divided by 8</td>
</tr>
</tbody>
</table>

**Output parameter(out)**

| - | - |

| Return value | - |

Example:

/* configure TIMER1 channel 0 input capture prescaler value */

timer_channel_input_capture_prescaler_config(TIMER1, TIMER_CH_0, TIMER_IC_PSC_DIV2);
**timer_channel_capture_value_register_read**

The description of `timer_channel_capture_value_register_read` is shown as below:

<table>
<thead>
<tr>
<th>Function name</th>
<th>timer_channel_capture_value_register_read</th>
</tr>
</thead>
<tbody>
<tr>
<td>Function prototype</td>
<td><code>uint32_t timer_channel_capture_value_register_read(uint32_t timer_periph, uint16_t channel);</code></td>
</tr>
<tr>
<td>Function descriptions</td>
<td>read TIMER channel capture compare register value</td>
</tr>
<tr>
<td>Precondition</td>
<td>-</td>
</tr>
<tr>
<td>The called functions</td>
<td>-</td>
</tr>
</tbody>
</table>

### Input parameter(in)

- **timer_periph**
  - TIMER peripheral
- **channel**
  - channel to be configured
- **TIMER_CH_0**
  - TIMER channel 0 TIMERx (x=1, 2, 8, 11)
- **TIMER_CH_1**
  - TIMER channel 1 TIMERx (x=1, 2, 8, 11)
- **TIMER_CH_2**
  - TIMER channel 2 (TIMERx (x=1, 2))
- **TIMER_CH_3**
  - TIMER channel 3 (TIMERx (x=1, 2))

### Output parameter(out)

- `uint32_t`
  - channel capture compare register value (0~65535)

**Example:**

```c
/* read TIMER1 channel 0 capture compare register value */

uint32_t ch0_value = 0;
ch0_value = timer_channel_capture_value_register_read (TIMER1, TIMER_CH_0);
```

**timer_input_pwm_capture_config**

The description of `timer_input_pwm_capture_config` is shown as below:

<table>
<thead>
<tr>
<th>Function name</th>
<th>timer_input_pwm_capture_config</th>
</tr>
</thead>
<tbody>
<tr>
<td>Function prototype</td>
<td>void timer_input_pwm_capture_config(uint32_t timer_periph, uint16_t channel, timer_ic_parameter_struct* icpwm);</td>
</tr>
<tr>
<td>Function descriptions</td>
<td>configure TIMER input pwm capture function</td>
</tr>
<tr>
<td>Precondition</td>
<td>-</td>
</tr>
<tr>
<td>The called functions</td>
<td>-</td>
</tr>
</tbody>
</table>

### Input parameter(in)

- **timer_periph**
  - TIMER peripheral
**TIMERx(x=1, 2, 8, 11)** (TIMER peripheral selection)

<table>
<thead>
<tr>
<th>Input parameter(in)</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>channel</strong></td>
</tr>
<tr>
<td>channel to be configured</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Input parameter(in)</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>TIMER_CH_0</strong></td>
</tr>
<tr>
<td>TIMER channel 0</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Input parameter(in)</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>TIMER_CH_1</strong></td>
</tr>
<tr>
<td>TIMER channel 1</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Input parameter(in)</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>icpwm</strong></td>
</tr>
<tr>
<td>TIMER channel input pwm parameter struct, the structure members can refer to Table 3-699. Structure timer_ic_parameter_struct</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Output parameter(out)</th>
</tr>
</thead>
<tbody>
<tr>
<td>-</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Return value</th>
</tr>
</thead>
<tbody>
<tr>
<td>-</td>
</tr>
</tbody>
</table>

Example:

```c
/* configure TIMER1 input pwm capture parameter */
timer_ic_parameter_struct timer_icinitpara;
timer_icinitpara.icpolarity = TIMER_IC_POLARITY_RISING;
timer_icinitpara.icselection = TIMER_IC_SELECTION_DIRECTTI;
timer_icinitpara.icprescaler = TIMER_IC_PSC_DIV1;
timer_icinitpara.icfilter = 0x0;
timer_input_pwm_capture_config(TIMER1, TIMER_CH_0, &timer_icinitpara);
```

**timer_hall_mode_config**

The description of timer_hall_mode_config is shown as below:

**Table 3-738. Function timer_hall_mode_config**

<table>
<thead>
<tr>
<th>Function name</th>
<th>timer_hall_mode_config</th>
</tr>
</thead>
<tbody>
<tr>
<td>Function prototype</td>
<td>void timer_hall_mode_config(uint32_t timer_periph, uint8_t hallmode);</td>
</tr>
<tr>
<td>Function descriptions</td>
<td>configure TIMER hall sensor mode</td>
</tr>
<tr>
<td>Precondition</td>
<td>-</td>
</tr>
<tr>
<td>The called functions</td>
<td>-</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Input parameter(in)</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>timer_periph</strong></td>
</tr>
<tr>
<td>TIMER peripheral</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Input parameter(in)</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>TIMERx(x=1, 2)</strong></td>
</tr>
<tr>
<td>TIMER peripheral selection</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Input parameter(in)</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>hallmode</strong></td>
</tr>
<tr>
<td>TIMER hall sensor mode state</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Input parameter(in)</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>TIMER_HALLINTERFA CE_ENABLE</strong></td>
</tr>
<tr>
<td>TIMER hall sensor mode enable</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Input parameter(in)</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>TIMER_HALLINTERFA CE_DISABLE</strong></td>
</tr>
<tr>
<td>TIMER hall sensor mode disable</td>
</tr>
</tbody>
</table>
Output parameter(out)
- -
Return value
- -

Example:

/* configure TIMER1 hall sensor mode */
timer_hall_mode_config(TIMER1, TIMER_HALLINTERFACE_ENABLE);

**timer_input_trigger_source_select**

The description of `timer_input_trigger_source_select` is shown as below:

<table>
<thead>
<tr>
<th>Function name</th>
<th>timer_input_trigger_source_select</th>
</tr>
</thead>
<tbody>
<tr>
<td>Function prototype</td>
<td>void timer_input_trigger_source_select(uint32_t timer_periph, uint32_t intrigger);</td>
</tr>
<tr>
<td>Function descriptions</td>
<td>select TIMER input trigger source</td>
</tr>
<tr>
<td>Precondition</td>
<td>SMC[2:0] = 000</td>
</tr>
<tr>
<td>The called functions</td>
<td>-</td>
</tr>
</tbody>
</table>

### Input parameter(in)

- **timer_periph**: TIMER peripheral
  - `TIMERx` please refer to the following parameters

- **intrigger**: trigger selection

| `TIMER_SMCFG_TRGS_EL_ITI0` | Internal trigger input 0 (ITI0, TIMERx (x=1, 2, 8)) |
| `TIMER_SMCFG_TRGS_EL_ITI1` | Internal trigger input 0 (ITI1, TIMERx (x=8, 11)) |
| `TIMER_SMCFG_TRGS_EL_ITI2` | Internal trigger input 0 (ITI2, TIMERx (x=11)) |
| `TIMER_SMCFG_TRGS_EL_CI0F_ED` | CI0 edge flag (CI0F_ED, TIMERx (x=1, 2, 8, 11)) |
| `TIMER_SMCFG_TRGS_EL_CI0FE0` | channel 0 input Filtered output(CI0FE0, TIMERx (x=1, 2, 8, 11)) |
| `TIMER_SMCFG_TRGS_EL_CI1FE1` | channel 1 input Filtered output(CI1FE1, TIMERx (x=1, 2, 8, 11)) |
| `TIMER_SMCFG_TRGS_EL_ETIFP` | External trigger input filter output(ETIFP, TIMERx (x=1, 2, 8, 11)) |

Output parameter(out)
- -

Return value
- -
Example:

```
/* select TIMER1 input trigger source */
timer_input_trigger_source_select(TIMER1, TIMER_SMCFG_TRGSEL_ITI0);
```

**timer_master_output_trigger_source_select**

The description of `timer_master_output_trigger_source_select` is shown as below:

### Table 3-740. Function `timer_master_output_trigger_source_select`

<table>
<thead>
<tr>
<th>Function name</th>
<th>timer_master_output_trigger_source_select</th>
</tr>
</thead>
<tbody>
<tr>
<td>Function prototype</td>
<td><code>void timer_master_output_trigger_source_select(uint32_t timer_periph, uint32_t outrigger);</code></td>
</tr>
<tr>
<td>Function descriptions</td>
<td>select TIMER master mode output trigger source</td>
</tr>
<tr>
<td>Precondition</td>
<td>-</td>
</tr>
<tr>
<td>The called functions</td>
<td>-</td>
</tr>
</tbody>
</table>

#### Input parameter(in)

<table>
<thead>
<tr>
<th>timer_periph</th>
<th>TIMER peripheral</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>TIMERx</code></td>
<td>TIMER peripheral selection</td>
</tr>
</tbody>
</table>

#### input parameter(out)

<table>
<thead>
<tr>
<th>outrigger</th>
<th>master mode control</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>TIMER_TRI_OUT_SRC_RESET</code></td>
<td>Reset. When the UPG bit in the TIMERx_SWEVG register is set or a reset is generated by the slave mode controller, a TRGO pulse occurs. And in the latter case, the signal on TRGO is delayed compared to the actual reset.</td>
</tr>
<tr>
<td><code>TIMER_TRI_OUT_SRC_ENABLE</code></td>
<td>Enable. This mode is useful to start several timers at the same time or to control a window in which a slave timer is enabled. In this mode the master mode controller selects the counter enable signal as TRGO. The counter enable signal is set when CEN control bit is set or the trigger input in pause mode is high. There is a delay between the trigger input in pause mode and the TRGO output, except if the master-slave mode is selected.</td>
</tr>
<tr>
<td><code>TIMER_TRI_OUT_SRC_UPDATE</code></td>
<td>Update. In this mode the master mode controller selects the update event as TRGO.</td>
</tr>
<tr>
<td><code>TIMER_TRI_OUT_SRC_CHO</code></td>
<td>Capture/compare pulse. In this mode the master mode controller generates a TRGO pulse when a capture or a compare match occurred in channel 0.</td>
</tr>
<tr>
<td><code>TIMER_TRI_OUT_SRC_O0CPRE</code></td>
<td>Compare. In this mode the master mode controller selects the O0CPRE signal is used as TRGO.</td>
</tr>
<tr>
<td><code>TIMER_TRI_OUT_SRC_O1CPRE</code></td>
<td>Compare. In this mode the master mode controller selects the O1CPRE signal is used as TRGO.</td>
</tr>
<tr>
<td><code>TIMER_TRI_OUT_SRC_O2CPRE</code></td>
<td>Compare. In this mode the master mode controller selects the O2CPRE signal is used as TRGO.</td>
</tr>
<tr>
<td><code>TIMER_TRI_OUT_SRC_O3CPRE</code></td>
<td>Compare. In this mode the master mode controller selects the O3CPRE signal is used as TRGO.</td>
</tr>
</tbody>
</table>

Output parameter(out)
Example:

/* select TIMER1 master mode output trigger source */
timer_master_output_trigger_source_select(TIMER1, TIMER_TRI_OUT_SRC_RESET);

### timer_slave_mode_select

The description of timer_slave_mode_select is shown as below:

#### Table 3-741. Function timer_slave_mode_select

<table>
<thead>
<tr>
<th>Function name</th>
<th>timer_slave_mode_select</th>
</tr>
</thead>
<tbody>
<tr>
<td>Function prototype</td>
<td>void timer_slave_mode_select(uint32_t timer_periph, uint32_t slavemode);</td>
</tr>
<tr>
<td>Function descriptions</td>
<td>select TIMER slave mode</td>
</tr>
<tr>
<td>Precondition</td>
<td>-</td>
</tr>
<tr>
<td>The called functions</td>
<td>-</td>
</tr>
</tbody>
</table>

#### Input parameter(in)

- **timer_periph**
  - TIMER peripheral
- **TIMERx**
  - TIMER peripheral selection

#### slavemode

- **TIMER_SLAVE_MODE_DISABLE**
  - slave mode disable, TIMERx(x=1, 2, 8, 11)
- **TIMER_ENCODER_MODE0**
  - encoder mode 0, TIMERx(x=1, 2)
- **TIMER_ENCODER_MODE1**
  - encoder mode 1, TIMERx(x=1, 2)
- **TIMER_ENCODER_MODE2**
  - encoder mode 2, TIMERx(x=1, 2)
- **TIMER_SLAVE_MODE_RESTART**
  - restart mode, TIMERx(x=1, 2, 8, 11)
- **TIMER_SLAVE_MODE_PAUSE**
  - pause mode, TIMERx(x=1, 2, 8, 11)
- **TIMER_SLAVE_MODE_EVENT**
  - event mode, TIMERx(x=1, 2, 8, 11)
- **TIMER_SLAVE_MODE_EXTERNAL0**
  - external clock mode 0, TIMERx(x=1, 2, 8, 11)

#### Output parameter(out)

- -

#### Return value

- -
Example:

/* select TIMER1 slave mode */
timer_slave_mode_select(TIMER1, TIMER_ENCODER_MODE0);

timer_master_slave_mode_config

The description of timer_master_slave_mode_config is shown as below:

Table 3-742. Function timer_master_slave_mode_config

<table>
<thead>
<tr>
<th>Function name</th>
<th>timer_master_slave_mode_config</th>
</tr>
</thead>
<tbody>
<tr>
<td>Function prototype</td>
<td>void timer_master_slave_mode_config(uint32_t timer_periph, uint8_t masterslave);</td>
</tr>
<tr>
<td>Function descriptions</td>
<td>configure TIMER master slave mode</td>
</tr>
<tr>
<td>Precondition</td>
<td>-</td>
</tr>
<tr>
<td>The called functions</td>
<td>-</td>
</tr>
<tr>
<td>Input parameter(in)</td>
<td></td>
</tr>
<tr>
<td>timer_periph</td>
<td>TIMER peripheral</td>
</tr>
<tr>
<td>TIMERx(x=1, 2, 8, 11)</td>
<td>TIMER peripheral selection</td>
</tr>
<tr>
<td>masterslave</td>
<td>master slave mode state</td>
</tr>
<tr>
<td>TIMER_MASTER_SLAVE_MODE_ENABLE</td>
<td>master slave mode enable</td>
</tr>
<tr>
<td>TIMER_MASTER_SLAVE_MODE_DISABLE</td>
<td>master slave mode disable</td>
</tr>
<tr>
<td>Output parameter(out)</td>
<td>-</td>
</tr>
<tr>
<td>Return value</td>
<td>-</td>
</tr>
</tbody>
</table>

Example:

/* configure TIMER1 master slave mode */
timer_master_slave_mode_config(TIMER1, TIMER_MASTER_SLAVE_MODE_ENABLE);

timer_external_trigger_config

The description of timer_external_trigger_config is shown as below:

Table 3-743. Function timer_external_trigger_config

<table>
<thead>
<tr>
<th>Function name</th>
<th>timer_external_trigger_config</th>
</tr>
</thead>
<tbody>
<tr>
<td>Function prototype</td>
<td>void timer_external_trigger_config(uint32_t timer_periph, uint32_t extprescaler, uint32_t expolarity, uint32_t extfilter);</td>
</tr>
<tr>
<td>Function descriptions</td>
<td>configure TIMER external trigger input</td>
</tr>
<tr>
<td>Precondition</td>
<td>-</td>
</tr>
</tbody>
</table>
The called functions

<table>
<thead>
<tr>
<th>Input parameter(in)</th>
</tr>
</thead>
<tbody>
<tr>
<td>timer_periph</td>
</tr>
<tr>
<td>TIMERx(x=1, 2)</td>
</tr>
</tbody>
</table>

Input parameter(in)

<table>
<thead>
<tr>
<th>extprescaler</th>
<th>external trigger prescaler</th>
</tr>
</thead>
<tbody>
<tr>
<td>TIMER_EXT_TRI_PSC_</td>
<td>no divided</td>
</tr>
<tr>
<td>OFF</td>
<td></td>
</tr>
<tr>
<td>DIV2</td>
<td>divided by 2</td>
</tr>
<tr>
<td>DIV4</td>
<td>divided by 4</td>
</tr>
<tr>
<td>DIV8</td>
<td>divided by 8</td>
</tr>
</tbody>
</table>

Input parameter(in)

<table>
<thead>
<tr>
<th>expolarity</th>
<th>external trigger polarity</th>
</tr>
</thead>
<tbody>
<tr>
<td>TIMER_ETP_FALLING</td>
<td>active low or falling edge active</td>
</tr>
<tr>
<td>TIMER_ETP_RISING</td>
<td>active high or rising edge active</td>
</tr>
</tbody>
</table>

Input parameter(in)

| extfilter           | external trigger filter control (0~15) |

Output parameter(out)

| -                   | Return value |

Example:

/* configure TIMER1 external trigger input */
timer_external_trigger_config(TIMER1, TIMER_EXT_TRI_PSC_DIV2, TIMER_ETP_FALLING, 10);

timer_quadrature_decoder_mode_config

The description of timer_quadrature_decoder_mode_config is shown as below:

Table 3-744. Function timer_quadrature_decoder_mode_config

<table>
<thead>
<tr>
<th>Function name</th>
<th>timer_quadrature_decoder_mode_config</th>
</tr>
</thead>
<tbody>
<tr>
<td>Function prototype</td>
<td>void timer_quadrature_decoder_mode_config(uint32_t timer_periph, uint32_t decomode, uint16_t ic0polarity, uint16_t ic1polarity);</td>
</tr>
<tr>
<td>Function descriptions</td>
<td>configure TIMER quadrature decoder mode</td>
</tr>
<tr>
<td>Precondition</td>
<td>-</td>
</tr>
<tr>
<td>The called functions</td>
<td>-</td>
</tr>
</tbody>
</table>

Input parameter(in)

<p>| timer_periph | TIMER peripheral |</p>
<table>
<thead>
<tr>
<th>TIMERx(x=1, 2)</th>
<th>TIMER peripheral selection</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Input parameter(in)</strong></td>
<td></td>
</tr>
<tr>
<td><strong>decomode</strong></td>
<td>quadrature decoder mode</td>
</tr>
<tr>
<td>TIMER_ENCODER_MODE0</td>
<td>counter counts on CI0FE0 edge depending on CI1FE1 level</td>
</tr>
<tr>
<td>TIMER_ENCODER_MODE1</td>
<td>counter counts on CI1FE1 edge depending on CI0FE0 level</td>
</tr>
<tr>
<td>TIMER_ENCODER_MODE2</td>
<td>counter counts on both CI0FE0 and CI1FE1 edges depending on the level of the other input</td>
</tr>
<tr>
<td><strong>Input parameter(in)</strong></td>
<td></td>
</tr>
<tr>
<td><strong>ic0polarity</strong></td>
<td>IC0 polarity</td>
</tr>
<tr>
<td>TIMER_IC_POLARITY_RISING</td>
<td>capture rising edge</td>
</tr>
<tr>
<td>TIMER_IC_POLARITY_FALLING</td>
<td>capture falling edge</td>
</tr>
<tr>
<td>TIMER_IC_POLARITY_BOTH_EDGE</td>
<td>capture both edge</td>
</tr>
<tr>
<td><strong>Input parameter(in)</strong></td>
<td></td>
</tr>
<tr>
<td><strong>ic1polarity</strong></td>
<td>IC1 polarity</td>
</tr>
<tr>
<td>TIMER_IC_POLARITY_RISING</td>
<td>capture rising edge</td>
</tr>
<tr>
<td>TIMER_IC_POLARITY_FALLING</td>
<td>capture falling edge</td>
</tr>
<tr>
<td>TIMER_IC_POLARITY_BOTH_EDGE</td>
<td>capture both edge</td>
</tr>
<tr>
<td><strong>Output parameter(out)</strong></td>
<td></td>
</tr>
<tr>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td><strong>Return value</strong></td>
<td>-</td>
</tr>
</tbody>
</table>

Example:

/* configure TIMER1 quadrature decoder mode */
timer_quadrature_decoder_mode_config(TIMER1, TIMER_ENCODER_MODE0, TIMER_IC_POLARITY_RISING, TIMER_IC_POLARITY_RISING);

timer_internal_trigger_as_external_clock_config

The description of timer_internal_trigger_as_external_clock_config is shown as below:

<table>
<thead>
<tr>
<th>Function name</th>
<th>timer_internal_trigger_as_external_clock_config</th>
</tr>
</thead>
<tbody>
<tr>
<td>Function prototype</td>
<td>void timer_internal_trigger_as_external_clock_config(uint32_t timer_periph, uint32_t intrigger);</td>
</tr>
</tbody>
</table>
Function descriptions
configure TIMER the internal trigger as external clock input

Precondition
-

The called functions
timer_input_trigger_source_select

Input parameter(in)
timer_periph
TIMER peripheral

TIMERx
TIMER peripheral selection

Input parameter(in)
intrigger
trigger selection

TIMER_SMCFG_TRGS_EL_ITI0
Internal trigger input 0 (ITI0), TIMERx(x=1, 2, 8)

TIMER_SMCFG_TRGS_EL_ITI1
Internal trigger input 0 (ITI1), TIMERx(x=8, 11)

TIMER_SMCFG_TRGS_EL_ITI2
Internal trigger input 0 (ITI2), TIMERx(x=11)

Output parameter(out)
-

Return value
-

Example:

/* configure TIMER1 the internal trigger ITI0 as external clock input */
timer_internal_trigger_as_external_clock_config(TIMER1, TIMER_SMCFG_TRGSEL_ITI0);

timer_external_trigger_as_external_clock_config

The description of timer_external_trigger_as_external_clock_config is shown as below:

Table 3-746. Function timer_external_trigger_as_external_clock_config

<table>
<thead>
<tr>
<th>Function name</th>
<th>timer_external_trigger_as_external_clock_config</th>
</tr>
</thead>
<tbody>
<tr>
<td>Function prototype</td>
<td>void timer_external_trigger_as_external_clock_config(uint32_t timer_periph, uint32_t extrigger, uint16_t expolarity, uint32_t extfilter);</td>
</tr>
<tr>
<td>Function descriptions</td>
<td>configure TIMER the external trigger as external clock input</td>
</tr>
<tr>
<td>Precondition</td>
<td>-</td>
</tr>
<tr>
<td>The called functions</td>
<td>timer_input_trigger_source_select</td>
</tr>
<tr>
<td>Input parameter(in)</td>
<td></td>
</tr>
<tr>
<td>timer_periph</td>
<td>TIMER peripheral</td>
</tr>
<tr>
<td>TIMERx(x=1, 2, 8,11)</td>
<td>TIMER peripheral selection</td>
</tr>
<tr>
<td>Input parameter(in)</td>
<td>extrigger</td>
</tr>
<tr>
<td>TIMER_SMCFG_TRGS_EL_CI0F_ED</td>
<td>C10 edge flag (CI0F_ED)</td>
</tr>
<tr>
<td>TIMER_SMCFG_TRGS_EL_CI0FE0</td>
<td>channel 0 input Filtered output (CI0FE0)</td>
</tr>
</tbody>
</table>
**TIMER_SMCFG_TRGS_EL_CI1FE1**

- **channel 1 input Filtered output (CI1FE1)**

**Input parameter (in)**

<table>
<thead>
<tr>
<th>expolarity</th>
<th>external trigger polarity</th>
</tr>
</thead>
<tbody>
<tr>
<td>TIMER_IC_POLARITY_RISING</td>
<td>active high or rising edge active</td>
</tr>
<tr>
<td>TIMER_IC_POLARITY_FALLING</td>
<td>active low or falling edge active</td>
</tr>
<tr>
<td>TIMER_IC_POLARITY_BOTH_EDGE</td>
<td>falling edge or rising edge active</td>
</tr>
</tbody>
</table>

**Input parameter (in)**

| extfilter | external trigger filter control (0~15) |

**Output parameter (out)**

- -

**Return value**

- -

**Example:**

```c
/* configure TIMER1 the external trigger CI0FE0 as external clock input */
timer_external_trigger_as_external_clock_config(TIMER1, TIMER_SMCFG_TRGSEL_CI0FE0, TIMER_IC_POLARITY_RISING, 0);
```

**timer_external_clock_mode0_config**

The description of timer_external_clock_mode0_config is shown as below:

**Table 3-747. Function timer_external_clock_mode0_config**

<table>
<thead>
<tr>
<th>Function name</th>
<th>timer_external_clock_mode0_config</th>
</tr>
</thead>
<tbody>
<tr>
<td>Function prototype</td>
<td>void timer_external_clock_mode0_config(uint32_t timer_periph, uint32_t extprescaler, uint32_t expolarity, uint32_t extfilter);</td>
</tr>
<tr>
<td>Function descriptions</td>
<td>configure TIMER the external clock mode0</td>
</tr>
<tr>
<td>Precondition</td>
<td>-</td>
</tr>
<tr>
<td>The called functions</td>
<td>timer_external_trigger_config</td>
</tr>
</tbody>
</table>

**Input parameter (in)**

<table>
<thead>
<tr>
<th>timer_periph</th>
<th>TIMER peripheral</th>
</tr>
</thead>
<tbody>
<tr>
<td>TIMERx(x=1, 2, 8, 11)</td>
<td>TIMER peripheral selection</td>
</tr>
</tbody>
</table>

**Input parameter (in)**

<table>
<thead>
<tr>
<th>extprescaler</th>
<th>ETI external trigger prescaler</th>
</tr>
</thead>
<tbody>
<tr>
<td>TIMER_EXT_TRI_PSC_OFF</td>
<td>no divided</td>
</tr>
<tr>
<td>TIMER_EXT_TRI_PSC_DIV2</td>
<td>divided by 2</td>
</tr>
<tr>
<td>TIMER_EXT_TRI_PSC</td>
<td>divided by 4</td>
</tr>
</tbody>
</table>
**DIV4**

| TIMER_EXT_TRI_PSC_DIV8 | divided by 8 |

**Input parameter(in)**

<table>
<thead>
<tr>
<th>expolarity</th>
<th>ETI external trigger polarity</th>
</tr>
</thead>
<tbody>
<tr>
<td>TIMER_ETP_FALLING</td>
<td>active low or falling edge active</td>
</tr>
<tr>
<td>TIMER_ETP_RISING</td>
<td>active high or rising edge active</td>
</tr>
</tbody>
</table>

**Input parameter(in)**

| extfilter | ETI external trigger filter control (0~15) |

**Output parameter(out)**

| - | - |

**Return value**

Example:

```c
/* configure TIMER1 the external clock mode0 */
timer_external_clock_mode0_config(TIMER1, TIMER_EXT_TRI_PSC_DIV2, TIMER_ETP_FALLING, 0);
```

**timer_external_clock_mode1_config**

The description of timer_external_clock_mode1_config is shown as below:

<table>
<thead>
<tr>
<th>Function name</th>
<th>timer_external_clock_mode1_config</th>
</tr>
</thead>
<tbody>
<tr>
<td>Function prototype</td>
<td>void timer_external_clock_mode1_config(uint32_t timer_periph, uint32_t extprescaler, uint32_t expolarity, uint32_t extfilter);</td>
</tr>
<tr>
<td>Function descriptions</td>
<td>configure TIMER the external clock mode1</td>
</tr>
<tr>
<td>Precondition</td>
<td>-</td>
</tr>
<tr>
<td>The called functions</td>
<td>timer_external_trigger_config</td>
</tr>
</tbody>
</table>

**Input parameter(in)**

<table>
<thead>
<tr>
<th>timer_periph</th>
<th>TIMER peripheral</th>
</tr>
</thead>
<tbody>
<tr>
<td>TIMERx(x=1, 2)</td>
<td>TIMER peripheral selection</td>
</tr>
</tbody>
</table>

**Input parameter(in)**

<table>
<thead>
<tr>
<th>extprescaler</th>
<th>ETI external trigger prescaler</th>
</tr>
</thead>
<tbody>
<tr>
<td>TIMER_EXT_TRI_PSC_OFF</td>
<td>no divided</td>
</tr>
<tr>
<td>TIMER_EXT_TRI_PSC_DIV2</td>
<td>divided by 2</td>
</tr>
<tr>
<td>TIMER_EXT_TRI_PSC_DIV4</td>
<td>divided by 4</td>
</tr>
<tr>
<td>TIMER_EXT_TRI_PSC_DIV8</td>
<td>divided by 8</td>
</tr>
</tbody>
</table>
Input parameter (in)

<table>
<thead>
<tr>
<th>expolarity</th>
<th>ETI external trigger polarity</th>
</tr>
</thead>
<tbody>
<tr>
<td>TIMER_ETP_FALLING</td>
<td>active low or falling edge active</td>
</tr>
<tr>
<td>TIMER_ETP_RISING</td>
<td>active high or rising edge active</td>
</tr>
</tbody>
</table>

Input parameter (in)

| extfilter | ETI external trigger filter control (0~15) |

Output parameter (out)

| - | - |

Return value

- -

Example:

/* configure TIMER1 the external clock mode1 */

timer_external_clock_mode1_config(TIMER1, TIMER_EXT_TRI_PSC_DIV2, TIMER_ETP_FALLING, 0);

timer_external_clock_mode1_disable

The description of timer_external_clock_mode1_disable is shown as below:

Table 3-749. Function timer_external_clock_mode1_disable

<table>
<thead>
<tr>
<th>Function name</th>
<th>timer_external_clock_mode1_disable</th>
</tr>
</thead>
<tbody>
<tr>
<td>Function prototype</td>
<td>void timer_external_clock_mode1_disable(uint32_t timer_periph);</td>
</tr>
<tr>
<td>Function descriptions</td>
<td>disable TIMER the external clock mode1</td>
</tr>
<tr>
<td>Precondition</td>
<td>-</td>
</tr>
<tr>
<td>The called functions</td>
<td>-</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Input parameter (in)</th>
</tr>
</thead>
<tbody>
<tr>
<td>timer_periph</td>
</tr>
<tr>
<td>TIMERx(x=1, 2)</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Output parameter (out)</th>
</tr>
</thead>
<tbody>
<tr>
<td>-</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Return value</th>
</tr>
</thead>
<tbody>
<tr>
<td>-</td>
</tr>
</tbody>
</table>

Example:

/* disable TIMER1 the external clock mode1 */

timer_external_clock_mode1_disable(TIMER1);

timer_channel_remap_config

The description of timer_channel_remap_config is shown as below:
Table 3-750. Function timer_channel_remap_config

<table>
<thead>
<tr>
<th>Function name</th>
<th>timer_channel_remap_config</th>
</tr>
</thead>
<tbody>
<tr>
<td>Function prototype</td>
<td>void timer_channel_remap_config(uint32_t timer_periph, uint32_t remap);</td>
</tr>
<tr>
<td>Function descriptions</td>
<td>configure TIMER channel remap function</td>
</tr>
<tr>
<td>Precondition</td>
<td>-</td>
</tr>
<tr>
<td>The called functions</td>
<td>timer_external_trigger_config</td>
</tr>
</tbody>
</table>

**Input parameter(in)**

- 

<table>
<thead>
<tr>
<th>remap</th>
<th>timer_periph</th>
<th>remap function selection</th>
</tr>
</thead>
<tbody>
<tr>
<td>TIMER8_CI0_RMP_GPIO</td>
<td>TIMER8</td>
<td>Timer8 channel 0 input is connected to GPIO(TIMER8_CH0) TIMERx(x=8)</td>
</tr>
<tr>
<td>TIMER8_CI0_RMP_LXTAL</td>
<td>TIMER8</td>
<td>Timer8 channel 0 input is connected to the LXTAL TIMERx(x=8)</td>
</tr>
<tr>
<td>TIMER8_CI0_RMP_HXTAL_DIV32</td>
<td>TIMER8</td>
<td>Timer8 channel 0 input is connected to HXTAL_DIV32 clock TIMERx(x=8)</td>
</tr>
<tr>
<td>TIMER8_CI0_RMP_CKOUTSEL</td>
<td>TIMER8</td>
<td>Timer8 channel 0 input is connected to CKOUTSEL TIMERx(x=8)</td>
</tr>
<tr>
<td>TIMER11_CI0_RMP_GPIO</td>
<td>TIMER11</td>
<td>Timer11 channel 0 input is connected to GPIO(TIMER11_CH0) TIMERx(x=11)</td>
</tr>
<tr>
<td>TIMER11_CI0_RMP_IRC32K</td>
<td>TIMER11</td>
<td>Timer11 channel 0 input is connected to the IRC32K TIMERx(x=11)</td>
</tr>
<tr>
<td>TIMER11_CI0_RMP_LXTAL</td>
<td>TIMER11</td>
<td>Timer11 channel 0 input is connected to LXTAL clock TIMERx(x=11)</td>
</tr>
<tr>
<td>TIMER11_CI0_RMP_RTC_OUT</td>
<td>TIMER11</td>
<td>Timer11 channel 0 input is connected to RTC_OUT TIMERx(x=11)</td>
</tr>
</tbody>
</table>

**Output parameter(out)**

- 

**Return value**

- 

Example:

```c
/* configure TIMER8 channel 0 input is connected to GPIO */

timer_channel_remap_config(TIMBER8, TIMER8_CI0_RMP_GPIO);
```

**timer_write_chxval_register_config**

The description of timer_write_chxval_register_config is shown as below:

Table 3-751. Function timer_write_chxval_register_config

<table>
<thead>
<tr>
<th>Function name</th>
<th>timer_write_chxval_register_config</th>
</tr>
</thead>
<tbody>
<tr>
<td>Function prototype</td>
<td>void timer_write_chxval_register_config(uint32_t timer_periph, uint16_t</td>
</tr>
</tbody>
</table>
Function descriptions
configure TIMER write CHxVAL register selection

Precondition
-

The called functions
-

Input parameter(in)

<table>
<thead>
<tr>
<th>Function name</th>
<th>timer_periph</th>
</tr>
</thead>
<tbody>
<tr>
<td>TIMER peripheral</td>
<td>TIMERx(x=1, 2, 8, 11)</td>
</tr>
</tbody>
</table>

Input parameter(in)

<table>
<thead>
<tr>
<th>ccsel</th>
<th>write CHxVAL register selection</th>
</tr>
</thead>
<tbody>
<tr>
<td>TIMER_CHVSEL_DISABLE</td>
<td>no effect</td>
</tr>
<tr>
<td>TIMER_CHVSEL_ENABLE</td>
<td>when write the CHxVAL register, if the write value is same as the CHxVAL value, the write access is ignored</td>
</tr>
</tbody>
</table>

Output parameter(out)

Return value
-

Example:

/* configure TIMER1 write CHxVAL register selection */
timer_write_chxval_register_config(TIMER1, TIMER_CHVSEL_ENABLE);

timer_flag_get

The description of timer_flag_get is shown as below:

Table 3-752. Function timer_flag_get

<table>
<thead>
<tr>
<th>Function name</th>
<th>timer_flag_get</th>
</tr>
</thead>
<tbody>
<tr>
<td>Function prototype</td>
<td>FlagStatus timer_flag_get(uint32_t timer_periph, uint32_t flag);</td>
</tr>
<tr>
<td>Function descriptions</td>
<td>get TIMER flags</td>
</tr>
<tr>
<td>Precondition</td>
<td>-</td>
</tr>
<tr>
<td>The called functions</td>
<td>-</td>
</tr>
</tbody>
</table>

Input parameter(in)

<table>
<thead>
<tr>
<th>Function name</th>
<th>timer_periph</th>
</tr>
</thead>
<tbody>
<tr>
<td>TIMER peripheral</td>
<td>TIMERx(x=1, 2, 8, 11)</td>
</tr>
</tbody>
</table>

Input parameter(in)

<table>
<thead>
<tr>
<th>flag</th>
<th>the timer interrupt flags</th>
</tr>
</thead>
<tbody>
<tr>
<td>TIMER_FLAG_UP</td>
<td>update flag, TIMERx(x=1, 2, 5, 6, 8, 11)</td>
</tr>
<tr>
<td>TIMER_FLAG_CH0</td>
<td>channel 0 flag, TIMERx(x=1, 2, 8, 11)</td>
</tr>
<tr>
<td>TIMER_FLAG_CH1</td>
<td>channel 1 flag, TIMERx(x=1, 2, 8, 11)</td>
</tr>
<tr>
<td>TIMER_FLAG_CH2</td>
<td>channel 2 flag, TIMERx(x=1, 2)</td>
</tr>
<tr>
<td>TIMER_FLAG_CH3</td>
<td>channel 3 flag, TIMERx(x=1, 2)</td>
</tr>
<tr>
<td>TIMER_FLAG_TRG</td>
<td>trigger flag, TIMERx(x=1, 2, 8, 11)</td>
</tr>
</tbody>
</table>
**TIMER_FLAG_CH0**
channel 0 overcapture flag, TIMERx(x=1, 2, 8, 11)

**TIMER_FLAG_CH1O**
channel 1 overcapture flag, TIMERx(x=1, 2, 8, 11)

**TIMER_FLAG_CH2O**
channel 2 overcapture flag, TIMERx(x=1, 2)

**TIMER_FLAG_CH3O**
channel 3 overcapture flag, TIMERx(x=1, 2)

**Output parameter (out)**

- -

**FlagStatus**

**Return value**

FlagStatus

SET or RESET

Example:

/* get TIMER1 update flags */

FlagStatus Flag_status = RESET;

Flag_status = timer_flag_get(TIMER1, TIMER_FLAG_UP);

timer_flag_clear

The description of timer_flag_clear is shown as below:

**Table 3-753. Function timer_flag_clear**

<table>
<thead>
<tr>
<th>Function name</th>
<th>timer_flag_clear</th>
</tr>
</thead>
<tbody>
<tr>
<td>Function prototype</td>
<td>void timer_flag_clear(uint32_t timer_periph, uint32_t flag);</td>
</tr>
<tr>
<td>Function descriptions</td>
<td>clear TIMER flags</td>
</tr>
<tr>
<td>Precondition</td>
<td>-</td>
</tr>
<tr>
<td>The called functions</td>
<td>-</td>
</tr>
</tbody>
</table>

**Input parameter (in)**

<table>
<thead>
<tr>
<th>timer_periph</th>
<th>TIMER peripheral</th>
</tr>
</thead>
<tbody>
<tr>
<td>TIMERx</td>
<td>please refer to the following parameters</td>
</tr>
</tbody>
</table>

**Input parameter (in)**

<table>
<thead>
<tr>
<th>flag</th>
<th>the timer interrupt flags</th>
</tr>
</thead>
<tbody>
<tr>
<td>TIMER_FLAG_UP</td>
<td>update flag, TIMERx(x=1, 2, 5, 6, 8, 11)</td>
</tr>
<tr>
<td>TIMER_FLAG_CH0</td>
<td>channel 0 flag, TIMERx(x=1, 2, 8, 11)</td>
</tr>
<tr>
<td>TIMER_FLAG_CH1</td>
<td>channel 1 flag, TIMERx(x=1, 2, 8, 11)</td>
</tr>
<tr>
<td>TIMER_FLAG_CH2</td>
<td>channel 2 flag, TIMERx(x=1, 2)</td>
</tr>
<tr>
<td>TIMER_FLAG_CH3</td>
<td>channel 3 flag, TIMERx(x=1, 2)</td>
</tr>
<tr>
<td>TIMER_FLAG_TRG</td>
<td>trigger flag, TIMERx(x=1, 2, 8, 11)</td>
</tr>
<tr>
<td>TIMER_FLAG_CH0O</td>
<td>channel 0 overcapture flag, TIMERx(x=1, 2, 8, 11)</td>
</tr>
<tr>
<td>TIMER_FLAG_CH1O</td>
<td>channel 1 overcapture flag, TIMERx(x=1, 2, 8, 11)</td>
</tr>
<tr>
<td>TIMER_FLAG_CH2O</td>
<td>channel 2 overcapture flag, TIMERx(x=1, 2)</td>
</tr>
<tr>
<td>TIMER_FLAG_CH3O</td>
<td>channel 3 overcapture flag, TIMERx(x=1, 2)</td>
</tr>
</tbody>
</table>

**Output parameter (out)**

- -

**Return value**

FlagStatus

SET or RESET
Example:

```
/* clear TIMER1 update flags */
timer_flag_clear(TIMER1, TIMER_FLAG_UP);
```

**timer_interrupt_enable**

The description of `timer_interrupt_enable` is shown as below:

<table>
<thead>
<tr>
<th>Table 3-754. Function timer_interrupt_enable</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Function name</strong></td>
</tr>
<tr>
<td><strong>Function prototype</strong></td>
</tr>
<tr>
<td><strong>Function descriptions</strong></td>
</tr>
<tr>
<td><strong>Precondition</strong></td>
</tr>
<tr>
<td><strong>The called functions</strong></td>
</tr>
</tbody>
</table>

**Input parameter(in)**

- `timer_periph` TIMEx peripheral
- `interrupt` please refer to the following parameters

<table>
<thead>
<tr>
<th>Input parameter(in)</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td><code>TIMER_INT_UP</code></td>
<td>update interrupt enable, TIMEx(x=1, 2, 5, 6, 8, 11)</td>
</tr>
<tr>
<td><code>TIMER_INT_CH0</code></td>
<td>channel 0 interrupt enable, TIMEx(x=1, 2, 8, 11)</td>
</tr>
<tr>
<td><code>TIMER_INT_CH1</code></td>
<td>channel 1 interrupt enable, TIMEx(x=1, 2, 8, 11)</td>
</tr>
<tr>
<td><code>TIMER_INT_CH2</code></td>
<td>channel 2 interrupt enable, TIMEx(x=1, 2)</td>
</tr>
<tr>
<td><code>TIMER_INT_CH3</code></td>
<td>channel 3 interrupt enable, TIMEx(x=1, 2)</td>
</tr>
<tr>
<td><code>TIMER_INT_TRG</code></td>
<td>trigger interrupt enable, TIMEx(x=1, 2, 8, 11)</td>
</tr>
</tbody>
</table>

**Output parameter(out)**

- -

**Return value**

- -

Example:

```
/* enable the TIMER1 update interrupt */
timer_interrupt_enable(TIMER1, TIMER_INT_UP);
```

**timer_interrupt_disable**

The description of `timer_interrupt_disable` is shown as below:

<table>
<thead>
<tr>
<th>Table 3-755. Function timer_interrupt_disable</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Function name</strong></td>
</tr>
<tr>
<td><strong>Function prototype</strong></td>
</tr>
</tbody>
</table>
**Function descriptions**

disable the TIMER interrupt

**Precondition**

- 

**The called functions**

- 

**Input parameter(in)**

<table>
<thead>
<tr>
<th>timer_periph</th>
<th>TIMER peripheral</th>
</tr>
</thead>
<tbody>
<tr>
<td>TIMERx</td>
<td>please refer to the following parameters</td>
</tr>
</tbody>
</table>

**Input parameter(in)**

<table>
<thead>
<tr>
<th>interrupt</th>
<th>timer interrupt disable source</th>
</tr>
</thead>
<tbody>
<tr>
<td>TIMER_INT_UP</td>
<td>update interrupt enable, TIMERx(x=1, 2, 5, 6, 8, 11)</td>
</tr>
<tr>
<td>TIMER_INT_CH0</td>
<td>channel 0 interrupt enable, TIMERx(x=1, 2, 8, 11)</td>
</tr>
<tr>
<td>TIMER_INT_CH1</td>
<td>channel 1 interrupt enable, TIMERx(x=1, 2, 8, 11)</td>
</tr>
<tr>
<td>TIMER_INT_CH2</td>
<td>channel 2 interrupt enable, TIMERx(x=1, 2)</td>
</tr>
<tr>
<td>TIMER_INT_CH3</td>
<td>channel 3 interrupt enable, TIMERx(x=1, 2)</td>
</tr>
<tr>
<td>TIMER_INT_TRG</td>
<td>trigger interrupt enable, TIMERx(x=1, 2, 8, 11)</td>
</tr>
</tbody>
</table>

**Output parameter(out)**

- 

**Return value**

- 

Example:

```c
/* disable the TIMER1 update interrupt */
timer_interrupt_disable(TIMER1, TIMER_INT_UP);
```

**timer_interrupt_flag_get**

The description of timer_interrupt_flag_get is shown as below:

**Table 3-756. Function timer_interrupt_flag_get**

<table>
<thead>
<tr>
<th>Function name</th>
<th>timer_interrupt_flag_get</th>
</tr>
</thead>
<tbody>
<tr>
<td>Function prototype</td>
<td>FlagStatus timer_interrupt_flag_get(uint32_t timer_periph, uint32_t interrupt);</td>
</tr>
<tr>
<td>Function descriptions</td>
<td>get timer interrupt flag</td>
</tr>
<tr>
<td>Precondition</td>
<td>-</td>
</tr>
<tr>
<td>The called functions</td>
<td>-</td>
</tr>
</tbody>
</table>

**Input parameter(in)**

<table>
<thead>
<tr>
<th>timer_periph</th>
<th>TIMER peripheral</th>
</tr>
</thead>
<tbody>
<tr>
<td>TIMERx</td>
<td>please refer to the following parameters</td>
</tr>
</tbody>
</table>

**Input parameter(in)**

<table>
<thead>
<tr>
<th>interrupt</th>
<th>the timer interrupt bits</th>
</tr>
</thead>
<tbody>
<tr>
<td>TIMER_INT_FLAG_UP</td>
<td>update interrupt enable, TIMERx(x=1, 2, 5, 6, 8, 11)</td>
</tr>
<tr>
<td>TIMER_INT_FLAG_CH0</td>
<td>channel 0 interrupt enable, TIMERx(x=1, 2, 8, 11)</td>
</tr>
<tr>
<td>TIMER_INT_FLAG_CH1</td>
<td>channel 1 interrupt enable, TIMERx(x=1, 2, 8, 11)</td>
</tr>
<tr>
<td>TIMER_INT_FLAG_CH2</td>
<td>channel 2 interrupt enable, TIMERx(x=1, 2)</td>
</tr>
</tbody>
</table>
### TIMER_INT_FLAG_CH3
- Channel 3 interrupt enable, TIMERx(x=1, 2)

### TIMER_INT_FLAG_TRG
- Trigger interrupt enable, TIMERx(x=1, 2, 8, 11)

### Output parameter(out)
- 

### Return value
- FlagStatus
- SET or RESET

**Example:**

```c
/* get TIMER1 update interrupt flag */

FlagStatus Flag_interrupt = RESET;
Flag_interrupt = timer_interrupt_flag_get(TIMER1, TIMER_INT_FLAG_UP);
```

**timer_interrupt_flag_clear**

The description of timer_interrupt_flag_clear is shown as below:

**Table 3-757. Function timer_interrupt_flag_clear**

<table>
<thead>
<tr>
<th>Function name</th>
<th>timer_interrupt_flag_clear</th>
</tr>
</thead>
<tbody>
<tr>
<td>Function prototype</td>
<td>void timer_interrupt_flag_clear(uint32_t timer_periph, uint32_t interrupt);</td>
</tr>
<tr>
<td>Function descriptions</td>
<td>clear TIMER interrupt flag</td>
</tr>
<tr>
<td>Precondition</td>
<td>-</td>
</tr>
<tr>
<td>The called functions</td>
<td>-</td>
</tr>
</tbody>
</table>

**Input parameter(in)**

<table>
<thead>
<tr>
<th>timer_periph</th>
<th>TIMER peripheral please refer to the following parameters</th>
</tr>
</thead>
</table>

<table>
<thead>
<tr>
<th>interrupt</th>
<th>the timer interrupt bits</th>
</tr>
</thead>
<tbody>
<tr>
<td>TIMER_INT_FLAG_UP</td>
<td>update interrupt enable, TIMERx(x=1, 2, 5, 6, 8, 11)</td>
</tr>
<tr>
<td>TIMER_INT_FLAG_CH0</td>
<td>channel 0 interrupt enable, TIMERx(x=1, 2, 8, 11)</td>
</tr>
<tr>
<td>TIMER_INT_FLAG_CH1</td>
<td>channel 1 interrupt enable, TIMERx(x=1, 2, 8, 11)</td>
</tr>
<tr>
<td>TIMER_INT_FLAG_CH2</td>
<td>channel 2 interrupt enable, TIMERx(x=1, 2)</td>
</tr>
<tr>
<td>TIMER_INT_FLAG_CH3</td>
<td>channel 3 interrupt enable, TIMERx(x=1, 2)</td>
</tr>
<tr>
<td>TIMER_INT_FLAG_TRG</td>
<td>trigger interrupt enable, TIMERx(x=1, 2, 8, 11)</td>
</tr>
</tbody>
</table>

**Output parameter(out)**
- 

**Return value**
- 

**Example:**

```c
/* clear TIMER1 update interrupt flag */

timer_interrupt_flag_clear(TIMER1, TIMER_INT_FLAG_UP);
```
3.25. TRNG

The true random number generator (RNG) module can generate a 32-bit value using continuous analog noise. The TRNG registers are listed in chapter 3.25.1, the TRNG firmware functions are introduced in chapter 3.25.2.

3.25.1. Descriptions of Peripheral registers

TRNG registers are listed in the table shown as below:

Table 3-758 TRNG Registers

<table>
<thead>
<tr>
<th>Registers</th>
<th>Descriptions</th>
</tr>
</thead>
<tbody>
<tr>
<td>TRNG_CTL</td>
<td>TRNG control register</td>
</tr>
<tr>
<td>TRNG_STAT</td>
<td>TRNG status register</td>
</tr>
<tr>
<td>TRNG_DATA</td>
<td>TRNG data register</td>
</tr>
</tbody>
</table>

3.25.2. Descriptions of Peripheral functions

TRNG firmware functions are listed in the table shown as below:

Table 3-759. TRNG firmware function

<table>
<thead>
<tr>
<th>Function name</th>
<th>Function description</th>
</tr>
</thead>
<tbody>
<tr>
<td>trng_deinit</td>
<td>deinitialize the TRNG</td>
</tr>
<tr>
<td>trng_enable</td>
<td>enable the TRNG</td>
</tr>
<tr>
<td>trng_disable</td>
<td>disable the TRNG</td>
</tr>
<tr>
<td>trng_get_true_random_data</td>
<td>get the true random data</td>
</tr>
<tr>
<td>trng_flag_get</td>
<td>get the TRNG status flags</td>
</tr>
<tr>
<td>trng_interrupt_enable</td>
<td>enable the TRNG interrupt</td>
</tr>
<tr>
<td>trng_interrupt_disable</td>
<td>disable the TRNG interrupt</td>
</tr>
<tr>
<td>trng_interrupt_flag_get</td>
<td>get the TRNG interrupt flags</td>
</tr>
<tr>
<td>trng_interrupt_flag_clear</td>
<td>clear the TRNG interrupt flags</td>
</tr>
</tbody>
</table>

Enum trng_flag_enum

Table 3-760. Enum trng_flag_enum

<table>
<thead>
<tr>
<th>Member name</th>
<th>Function description</th>
</tr>
</thead>
<tbody>
<tr>
<td>TRNG_FLAG_DRDY</td>
<td>random data ready status</td>
</tr>
<tr>
<td>TRNG_FLAG_CEC5</td>
<td>clock error current status</td>
</tr>
<tr>
<td>TRNG_FLAG_SECS</td>
<td>seed error current status</td>
</tr>
</tbody>
</table>
Enum trng_int_flag_enum

Table 3-761. Enum trng_int_flag_enum

<table>
<thead>
<tr>
<th>Member name</th>
<th>Function description</th>
</tr>
</thead>
<tbody>
<tr>
<td>TRNG_INT_FLAG_CEIF</td>
<td>clock error interrupt flag</td>
</tr>
<tr>
<td>TRNG_INT_FLAG_SEIF</td>
<td>seed error interrupt flag</td>
</tr>
</tbody>
</table>

trng_deinit

The description of trng_deinit is shown as below:

Table 3-762. Function trng_deinit

<table>
<thead>
<tr>
<th>Function name</th>
<th>trng_deinit</th>
</tr>
</thead>
<tbody>
<tr>
<td>Function prototype</td>
<td>void trng_deinit(void);</td>
</tr>
<tr>
<td>Function descriptions</td>
<td>reset TRNG</td>
</tr>
<tr>
<td>Precondition</td>
<td>-</td>
</tr>
<tr>
<td>The called functions</td>
<td>rcu_periph_reset_enable / rcu_periph_reset_disable</td>
</tr>
<tr>
<td>Input parameter(in)</td>
<td>-</td>
</tr>
<tr>
<td>Output parameter(out)</td>
<td>-</td>
</tr>
<tr>
<td>Return value</td>
<td>-</td>
</tr>
</tbody>
</table>

Example:

/* reset TRNG */
trng_deinit();

trng_enable

The description of trng_enable is shown as below:

Table 3-763. Function trng_enable

<table>
<thead>
<tr>
<th>Function name</th>
<th>trng_enable</th>
</tr>
</thead>
<tbody>
<tr>
<td>Function prototype</td>
<td>void trng_enable(void);</td>
</tr>
<tr>
<td>Function descriptions</td>
<td>enable the TRNG</td>
</tr>
<tr>
<td>Precondition</td>
<td>-</td>
</tr>
<tr>
<td>The called functions</td>
<td>-</td>
</tr>
<tr>
<td>Input parameter(in)</td>
<td>-</td>
</tr>
<tr>
<td>Output parameter(out)</td>
<td>-</td>
</tr>
<tr>
<td>Return value</td>
<td>-</td>
</tr>
</tbody>
</table>
Example:

/* enable TRNG */

trng_enable();

**trng_disable**

The description of trng_disable is shown as below:

**Table 3-764 Function trng_disable**

<table>
<thead>
<tr>
<th>Function name</th>
<th>trng_disable</th>
</tr>
</thead>
<tbody>
<tr>
<td>Function prototype</td>
<td>void trng_disable(void);</td>
</tr>
<tr>
<td>Function descriptions</td>
<td>disable the TRNG</td>
</tr>
<tr>
<td>Precondition</td>
<td>-</td>
</tr>
<tr>
<td>The called functions</td>
<td>-</td>
</tr>
</tbody>
</table>

**Input parameter(in)**

-  

**Output parameter(out)**

-  

**Return value**

-  

Example:

/* disable TRNG */

trng_disable();

**trng_get_true_random_data**

The description of trng_get_true_random_data is shown as below:

**Table 3-765 Function trng_get_true_random_data**

<table>
<thead>
<tr>
<th>Function name</th>
<th>trng_get_true_random_data</th>
</tr>
</thead>
<tbody>
<tr>
<td>Function prototype</td>
<td>uint32_t trng_get_true_random_data(void);</td>
</tr>
<tr>
<td>Function descriptions</td>
<td>get the true random data</td>
</tr>
<tr>
<td>Precondition</td>
<td>-</td>
</tr>
<tr>
<td>The called functions</td>
<td>-</td>
</tr>
</tbody>
</table>

**Input parameter(in)**

-  

**Output parameter(out)**

-  

**Return value**

uint32_t 0x0 – 0xFFFFFFFF

Example:
/* get true random data */

uint32_t data;

data = trng_get_true_random_data();

**trng_flag_get**

The description of trng_flag_get is shown as below:

<table>
<thead>
<tr>
<th>Table 3-766 trng_flag_get</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Function name</strong></td>
</tr>
<tr>
<td><strong>Function prototype</strong></td>
</tr>
<tr>
<td><strong>Function descriptions</strong></td>
</tr>
<tr>
<td><strong>Precondition</strong></td>
</tr>
<tr>
<td><strong>The called functions</strong></td>
</tr>
<tr>
<td><strong>Input parameter(in)</strong></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td><strong>Output parameter(out)</strong></td>
</tr>
<tr>
<td><strong>Return value</strong></td>
</tr>
</tbody>
</table>

Example:

/* get TRNG clock error current flag status */

FlagStatus flag_status = RESET;

flag_status == trng_flag_get(TRNG_FLAG_CECS);

**trng_interrupt_enable**

The description of trng_interrupt_enable is shown as below:

<table>
<thead>
<tr>
<th>Table 3-767 trng_interrupt_enable</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Function name</strong></td>
</tr>
<tr>
<td><strong>Function prototype</strong></td>
</tr>
<tr>
<td><strong>Function descriptions</strong></td>
</tr>
<tr>
<td><strong>Precondition</strong></td>
</tr>
<tr>
<td><strong>The called functions</strong></td>
</tr>
<tr>
<td><strong>Input parameter(in)</strong></td>
</tr>
<tr>
<td><strong>Output parameter(out)</strong></td>
</tr>
<tr>
<td><strong>Return value</strong></td>
</tr>
</tbody>
</table>
Example:

/* enable TRNG interrupt */

trng_interrupt_enable();

**trng_interrupt_disable**

The description of trng_interrupt_disable is shown as below:

<table>
<thead>
<tr>
<th>Table 3-768 trng_interrupt_disable</th>
</tr>
</thead>
<tbody>
<tr>
<td>Function name</td>
</tr>
<tr>
<td>Function prototype</td>
</tr>
<tr>
<td>Function descriptions</td>
</tr>
<tr>
<td>Precondition</td>
</tr>
<tr>
<td>The called functions</td>
</tr>
<tr>
<td>Input parameter(in)</td>
</tr>
<tr>
<td>Output parameter(out)</td>
</tr>
<tr>
<td>Return value</td>
</tr>
</tbody>
</table>

Example:

/* disable TRNG interrupt */

trng_interrupt_disable();

**trng_interrupt_flag_get**

The description of trng_interrupt_flag_get is shown as below:

<table>
<thead>
<tr>
<th>Table 3-769 trng_interrupt_flag_get</th>
</tr>
</thead>
<tbody>
<tr>
<td>Function name</td>
</tr>
<tr>
<td>Function prototype</td>
</tr>
<tr>
<td>Function descriptions</td>
</tr>
<tr>
<td>Precondition</td>
</tr>
<tr>
<td>The called functions</td>
</tr>
<tr>
<td>Input parameter(in)</td>
</tr>
<tr>
<td>TRNG interrupt flag, refer to <strong>Table 3-761. Enum trng_int_flag_enum</strong></td>
</tr>
<tr>
<td>Output parameter(out)</td>
</tr>
<tr>
<td>Return value</td>
</tr>
<tr>
<td>SET or RESET</td>
</tr>
</tbody>
</table>

Example:
/* get TRNG clock error interrupt flag */

FlagStatus interrupt_flag = RESET;

interrupt_flag = trng_interrupt_flag_get(TRNG_INT_FLAG_CEIF);

trng_interrupt_flag_clear

The description of trng_interrupt_flag_clear is shown as below:

<table>
<thead>
<tr>
<th>Table 3-770 trng_interrupt_flag_clear</th>
</tr>
</thead>
<tbody>
<tr>
<td>Function name</td>
</tr>
<tr>
<td>Function prototype</td>
</tr>
<tr>
<td>Function descriptions</td>
</tr>
<tr>
<td>Precondition</td>
</tr>
<tr>
<td>The called functions</td>
</tr>
<tr>
<td>Input parameter(in)</td>
</tr>
<tr>
<td>int_flag</td>
</tr>
<tr>
<td>Output parameter(out)</td>
</tr>
<tr>
<td>-</td>
</tr>
</tbody>
</table>

Return value |

- |

Example:

/* clear TRNG clock error interrupt flag */

trng_interrupt_flag_clear(TRNG_INT_FLAG_CEIF);

3.26. USART

The Universal Synchronous/Asynchronous Receiver/Transmitter (USART) provides a flexible serial data exchange interface. The USART registers are listed in chapter 3.26.1, the USART firmware functions are introduced in chapter 3.26.2.

3.26.1. Descriptions of Peripheral registers

USART registers are listed in the table shown as below:

<table>
<thead>
<tr>
<th>Table 3-771. USART Registers</th>
</tr>
</thead>
<tbody>
<tr>
<td>Registers</td>
</tr>
<tr>
<td>USART_CTL0</td>
</tr>
<tr>
<td>USART_CTL1</td>
</tr>
<tr>
<td>USART_CTL2</td>
</tr>
<tr>
<td>USART_BAUD</td>
</tr>
<tr>
<td>USART_GP</td>
</tr>
</tbody>
</table>
3.26.2. Descriptions of Peripheral functions

USART firmware functions are listed in the table shown as below:

<table>
<thead>
<tr>
<th>Function name</th>
<th>Function description</th>
</tr>
</thead>
<tbody>
<tr>
<td>usart_deinit</td>
<td>reset USART</td>
</tr>
<tr>
<td>usart_baudrate_set</td>
<td>configure USART baud rate value</td>
</tr>
<tr>
<td>usart_parity_config</td>
<td>configure USART parity function</td>
</tr>
<tr>
<td>usart_word_length_set</td>
<td>configure USART word length</td>
</tr>
<tr>
<td>usart_stop_bit_set</td>
<td>configure USART stop bit length</td>
</tr>
<tr>
<td>usart_enable</td>
<td>enable USART</td>
</tr>
<tr>
<td>usart_disable</td>
<td>disable USART</td>
</tr>
<tr>
<td>usart_transmit_config</td>
<td>configure USART transmitter</td>
</tr>
<tr>
<td>usart_receive_config</td>
<td>configure USART receiver</td>
</tr>
<tr>
<td>usart_data_first_config</td>
<td>data is transmitted/received with the LSB/MSB first</td>
</tr>
<tr>
<td>usart_invert_config</td>
<td>configure USART inverted</td>
</tr>
<tr>
<td>usart_overrun_enable</td>
<td>enable the USART overrun function</td>
</tr>
<tr>
<td>usart_overrun_disable</td>
<td>disable the USART overrun function</td>
</tr>
<tr>
<td>usart_oversample_config</td>
<td>configure the USART oversample mode</td>
</tr>
<tr>
<td>usart_sample_bit_config</td>
<td>configure sample bit method</td>
</tr>
<tr>
<td>usart_receiver_timeout_enable</td>
<td>enable receiver timeout</td>
</tr>
<tr>
<td>usart_receiver_timeout_disable</td>
<td>disable receiver timeout</td>
</tr>
<tr>
<td>usart_receiver_timeout_threshold_config</td>
<td>configure receiver timeout threshold</td>
</tr>
<tr>
<td>usart_data_transmit</td>
<td>USART transmit data function</td>
</tr>
<tr>
<td>usart_data_receive</td>
<td>USART receive data function</td>
</tr>
<tr>
<td>usart_command_enable</td>
<td>enable USART command</td>
</tr>
<tr>
<td>usart_autobaud_detection_enable</td>
<td>enable auto baud rate detection</td>
</tr>
<tr>
<td>usart_autobaud_detection_disable</td>
<td>disable auto baud rate detection</td>
</tr>
<tr>
<td>usart_autobaud_detection_mode_config</td>
<td>configure auto baud rate detection mode</td>
</tr>
<tr>
<td>usart_address_config</td>
<td>configure address of the USART</td>
</tr>
<tr>
<td>Function name</td>
<td>Function description</td>
</tr>
<tr>
<td>--------------------------------------------</td>
<td>-----------------------------------------------------------</td>
</tr>
<tr>
<td>usart_address_detection_mode_config</td>
<td>configure address detection mode</td>
</tr>
<tr>
<td>usart_mute_mode_enable</td>
<td>enable mute mode</td>
</tr>
<tr>
<td>usart_mute_mode_disable</td>
<td>disable mute mode</td>
</tr>
<tr>
<td>usart_mute_mode_wakeup_config</td>
<td>configure wakeup method in mute mode</td>
</tr>
<tr>
<td>usart_lin_mode_enable</td>
<td>enable LIN mode</td>
</tr>
<tr>
<td>usart_lin_mode_disable</td>
<td>disable LIN mode</td>
</tr>
<tr>
<td>usart_lin_break_detection_length_config</td>
<td>LIN break detection length</td>
</tr>
<tr>
<td>usart_halfduplex_enable</td>
<td>enable half-duplex mode</td>
</tr>
<tr>
<td>usart_halfduplex_disable</td>
<td>disable half-duplex mode</td>
</tr>
<tr>
<td>usart_clock_enable</td>
<td>enable clock</td>
</tr>
<tr>
<td>usart_clock_disable</td>
<td>disable clock</td>
</tr>
<tr>
<td>usart_synchronous_clock_config</td>
<td>configure USART synchronous mode parameters</td>
</tr>
<tr>
<td>usart_guard_time_config</td>
<td>configure guard time value in smartcard mode</td>
</tr>
<tr>
<td>usart_smartcard_mode_enable</td>
<td>enable smartcard mode</td>
</tr>
<tr>
<td>usart_smartcard_mode_disable</td>
<td>disable smartcard mode</td>
</tr>
<tr>
<td>usart_smartcard_mode_nack_enable</td>
<td>enable NACK in smartcard mode</td>
</tr>
<tr>
<td>usart_smartcard_mode_nack_disable</td>
<td>disable NACK in smartcard mode</td>
</tr>
<tr>
<td>usart_smartcard_mode_early_nack_enable</td>
<td>enable early NACK in smartcard mode</td>
</tr>
<tr>
<td>usart_smartcard_mode_early_nack_disable</td>
<td>disable early NACK in smartcard mode</td>
</tr>
<tr>
<td>usart_smartcard_autoretry_config</td>
<td>configure smartcard auto-retry number</td>
</tr>
<tr>
<td>usart_block_length_config</td>
<td>configure block length</td>
</tr>
<tr>
<td>usart_irda_mode_enable</td>
<td>enable IrDA mode</td>
</tr>
<tr>
<td>usart_irda_mode_disable</td>
<td>disable IrDA mode</td>
</tr>
<tr>
<td>usart_prescaler_config</td>
<td>configure the peripheral clock prescaler</td>
</tr>
<tr>
<td>usart_irda_lowpower_config</td>
<td>configure IrDA low-power</td>
</tr>
<tr>
<td>usart_hardwire_flow_rts_config</td>
<td>configure hardware flow control RTS</td>
</tr>
<tr>
<td>usart_hardwire_flow_cts_config</td>
<td>configure hardware flow control CTS</td>
</tr>
<tr>
<td>usart_hardwire_flow_coherence_config</td>
<td>configure hardware flow control coherence mode</td>
</tr>
<tr>
<td>usart_rs485_driver_enable</td>
<td>enable RS485 driver</td>
</tr>
<tr>
<td>usart_rs485_driver_disable</td>
<td>disable RS485 driver</td>
</tr>
<tr>
<td>usart_driver_assertime_config</td>
<td>configure driver enable assertion time</td>
</tr>
<tr>
<td>usart_driver_deassertime_config</td>
<td>configure driver enable de-assertion time</td>
</tr>
<tr>
<td>usart_depolarity_config</td>
<td>configure driver enable polarity mode</td>
</tr>
<tr>
<td>usart_dma_receive_config</td>
<td>configure USART DMA for reception</td>
</tr>
<tr>
<td>usart_dma_transmit_config</td>
<td>configure USART DMA for transmission</td>
</tr>
<tr>
<td>usart_reception_error_dma_disable</td>
<td>disable DMA on reception error</td>
</tr>
<tr>
<td>Function name</td>
<td>Function description</td>
</tr>
<tr>
<td>-------------------------------------------</td>
<td>----------------------------------------------------------</td>
</tr>
<tr>
<td>usart_reception_error_dma_enable</td>
<td>enable DMA on reception error</td>
</tr>
<tr>
<td>usart_wakeup_enable</td>
<td>enable USART to wakeup the mcu from deep-sleep mode</td>
</tr>
<tr>
<td>usart_wakeup_disable</td>
<td>disable USART to wakeup the mcu from deep-sleep mode</td>
</tr>
<tr>
<td>usart_wakeup_mode_config</td>
<td>configure the USART wakeup mode from deep-sleep mode</td>
</tr>
<tr>
<td>usart_receive_fifo_enable</td>
<td>enable receive FIFO</td>
</tr>
<tr>
<td>usart_receive_fifo_disable</td>
<td>disable receive FIFO</td>
</tr>
<tr>
<td>usart_receive_fifo_counter_number</td>
<td>read receive FIFO counter number</td>
</tr>
<tr>
<td>usart_flag_get</td>
<td>get flag in STAT/RFCS register</td>
</tr>
<tr>
<td>usart_flag_clear</td>
<td>clear USART status</td>
</tr>
<tr>
<td>usart_interrupt_enable</td>
<td>enable USART interrupt</td>
</tr>
<tr>
<td>usart_interrupt_disable</td>
<td>disable USART interrupt</td>
</tr>
<tr>
<td>usart_interrupt_flag_get</td>
<td>get USART interrupt and flag status</td>
</tr>
<tr>
<td>usart_interrupt_flag_clear</td>
<td>clear USART interrupt flag</td>
</tr>
</tbody>
</table>

**Enum usart_flag_enum**

Table 3-773. Enum usart_flag_enum

<table>
<thead>
<tr>
<th>Member name</th>
<th>Function description</th>
</tr>
</thead>
<tbody>
<tr>
<td>USART_FLAG_REA</td>
<td>receive enable acknowledge flag</td>
</tr>
<tr>
<td>USART_FLAG_TEA</td>
<td>transmit enable acknowledge flag</td>
</tr>
<tr>
<td>USART_FLAG_WU</td>
<td>wakeup from Deep-sleep mode flag</td>
</tr>
<tr>
<td>USART_FLAG_RWU</td>
<td>receiver wakeup from mute mode</td>
</tr>
<tr>
<td>USART_FLAG_SB</td>
<td>send break flag</td>
</tr>
<tr>
<td>USART_FLAG_AM</td>
<td>ADDR match flag</td>
</tr>
<tr>
<td>USART_FLAG_BSY</td>
<td>busy flag</td>
</tr>
<tr>
<td>USART_FLAG_ABD</td>
<td>auto baudrate detection flag</td>
</tr>
<tr>
<td>USART_FLAG_ABDE</td>
<td>auto baudrate detection error</td>
</tr>
<tr>
<td>USART_FLAG_EB</td>
<td>end of block flag</td>
</tr>
<tr>
<td>USART_FLAG_RT</td>
<td>receiver timeout flag</td>
</tr>
<tr>
<td>USART_FLAG_CTS</td>
<td>CTS level</td>
</tr>
<tr>
<td>USART_FLAG_CTSF</td>
<td>CTS change flag</td>
</tr>
<tr>
<td>USART_FLAG_LBD</td>
<td>LIN break detected flag</td>
</tr>
<tr>
<td>USART_FLAG_TBE</td>
<td>transmit data buffer empty</td>
</tr>
<tr>
<td>USART_FLAG_TC</td>
<td>transmission complete</td>
</tr>
<tr>
<td>USART_FLAG_RBNE</td>
<td>read data buffer not empty</td>
</tr>
<tr>
<td>USART_FLAG_IDLE</td>
<td>IDLE line detected flag</td>
</tr>
<tr>
<td>USART_FLAG_OERR</td>
<td>overrun error</td>
</tr>
<tr>
<td>USART_FLAG_NERR</td>
<td>noise error flag</td>
</tr>
<tr>
<td>USART_FLAG_FERR</td>
<td>frame error flag</td>
</tr>
<tr>
<td>USART_FLAG_PERR</td>
<td>parity error flag</td>
</tr>
<tr>
<td>USART_FLAG_EPERR</td>
<td>early parity error flag</td>
</tr>
<tr>
<td>USART_FLAG_RFFINT</td>
<td>receive FIFO full interrupt flag</td>
</tr>
</tbody>
</table>
**Enum usart_interrupt_flag_enum**

Table 3-774. Enum usart_interrupt_flag_enum

<table>
<thead>
<tr>
<th>Member name</th>
<th>Function description</th>
</tr>
</thead>
<tbody>
<tr>
<td>USART_FLAG_RFF</td>
<td>receive FIFO full flag</td>
</tr>
<tr>
<td>USART_FLAG_RFE</td>
<td>receive FIFO empty flag</td>
</tr>
<tr>
<td>USART_FLAG_RFF</td>
<td>receive FIFO full flag</td>
</tr>
<tr>
<td>USART_FLAG_RFE</td>
<td>receive FIFO empty flag</td>
</tr>
<tr>
<td>USART_INT_FLAG_EB</td>
<td>end of block interrupt and flag</td>
</tr>
<tr>
<td>USART_INT_FLAG_RT</td>
<td>receiver timeout interrupt and flag</td>
</tr>
<tr>
<td>USART_INT_FLAG_AM</td>
<td>address match interrupt and flag</td>
</tr>
<tr>
<td>USART_INT_FLAG_PERR</td>
<td>parity error interrupt and flag</td>
</tr>
<tr>
<td>USART_INT_FLAG_TBE</td>
<td>transmitter buffer empty interrupt and flag</td>
</tr>
<tr>
<td>USART_INT_FLAG_TC</td>
<td>transmission complete interrupt and flag</td>
</tr>
<tr>
<td>USART_INT_FLAG_RBNE</td>
<td>read data buffer not empty interrupt and flag</td>
</tr>
<tr>
<td>USART_INT_FLAG_RBNE_ORE</td>
<td>read data buffer not empty interrupt and overrun error flag</td>
</tr>
<tr>
<td>USART_INT_FLAG_IDLE</td>
<td>IDLE line detected interrupt and flag</td>
</tr>
<tr>
<td>USART_INT_FLAG_LBD</td>
<td>LIN break detected interrupt and flag</td>
</tr>
<tr>
<td>USART_INT_FLAG_WU</td>
<td>wakeup from deep-sleep mode interrupt and flag</td>
</tr>
<tr>
<td>USART_INT_FLAG_CTS</td>
<td>CTS interrupt and flag</td>
</tr>
<tr>
<td>USART_INT_FLAG_ERR_NERR</td>
<td>error interrupt and noise error flag</td>
</tr>
<tr>
<td>USART_INT_FLAG_ERR_ORE</td>
<td>error interrupt and overrun error</td>
</tr>
<tr>
<td>USART_INT_FLAG_ERR_FERR</td>
<td>error interrupt and frame error flag</td>
</tr>
<tr>
<td>USART_INT_FLAG_RFF</td>
<td>receive FIFO full interrupt and flag</td>
</tr>
</tbody>
</table>

**Enum usart_interrupt_enum**

Table 3-775. Enum usart_interrupt_enum

<table>
<thead>
<tr>
<th>Member name</th>
<th>Function description</th>
</tr>
</thead>
<tbody>
<tr>
<td>USART_INT_EB</td>
<td>end of block interrupt</td>
</tr>
<tr>
<td>USART_INT_RT</td>
<td>receiver timeout interrupt</td>
</tr>
<tr>
<td>USART_INT_AM</td>
<td>address match interrupt</td>
</tr>
<tr>
<td>USART_INT_PERR</td>
<td>parity error interrupt</td>
</tr>
<tr>
<td>USART_INT_TBE</td>
<td>transmitter buffer empty interrupt</td>
</tr>
<tr>
<td>USART_INT_TC</td>
<td>transmission complete interrupt</td>
</tr>
<tr>
<td>USART_INT_RBNE</td>
<td>read data buffer not empty interrupt and overrun error interrupt</td>
</tr>
<tr>
<td>USART_INT_IDLE</td>
<td>IDLE line detected interrupt</td>
</tr>
<tr>
<td>USART_INT_LBD</td>
<td>LIN break detected interrupt</td>
</tr>
<tr>
<td>USART_INT_WU</td>
<td>wakeup from deep-sleep mode interrupt</td>
</tr>
<tr>
<td>USART_INT_CTS</td>
<td>CTS interrupt</td>
</tr>
<tr>
<td>USART_INT_ERR</td>
<td>error interrupt</td>
</tr>
<tr>
<td>Member name</td>
<td>Function description</td>
</tr>
<tr>
<td>--------------------------</td>
<td>-----------------------------------------</td>
</tr>
<tr>
<td>USART_INT_RFF</td>
<td>receive FIFO full interrupt</td>
</tr>
</tbody>
</table>

### Enum usart_invert_enum

#### Table 3-776. Enum usart_invert_enum

<table>
<thead>
<tr>
<th>Member name</th>
<th>Function description</th>
</tr>
</thead>
<tbody>
<tr>
<td>USART_DINV_ENABLE</td>
<td>data bit level inversion</td>
</tr>
<tr>
<td>USART_DINV_DISABLE</td>
<td>data bit level not inversion</td>
</tr>
<tr>
<td>USART_TXPIN_ENABLE</td>
<td>TX pin level inversion</td>
</tr>
<tr>
<td>USART_TXPIN_DISABLE</td>
<td>TX pin level not inversion</td>
</tr>
<tr>
<td>USART_RXPIN_ENABLE</td>
<td>RX pin level inversion</td>
</tr>
<tr>
<td>USART_RXPIN_DISABLE</td>
<td>RX pin level not inversion</td>
</tr>
<tr>
<td>USART_SWAP_ENABLE</td>
<td>swap TX/RX pins</td>
</tr>
<tr>
<td>USART_SWAP_DISABLE</td>
<td>not swap TX/RX pins</td>
</tr>
</tbody>
</table>

### usart_deinit

The description of usart_deinit is shown as below:

#### Table 3-777. Function usart_deinit

<table>
<thead>
<tr>
<th>Function name</th>
<th>Function prototype</th>
<th>Function descriptions</th>
<th>Precondition</th>
<th>The called functions</th>
<th>Input parameter(in)</th>
<th>Output parameter(out)</th>
<th>Return value</th>
</tr>
</thead>
<tbody>
<tr>
<td>usart_deinit</td>
<td>void usart_deinit(uint32_t usart_periph);</td>
<td>reset USART</td>
<td>-</td>
<td>rcu_periph_reset_enable / rcu_periph_reset_disable</td>
<td>usart_periph外设USARTx/UARTx</td>
<td>-</td>
<td>-</td>
</tr>
</tbody>
</table>

#### Example:

/* reset USART0 */

usart_deinit(USART0);

### usart_baudrate_set

The description of usart_baudrate_set is shown as below:
### Table 3-778. Function `usart_baudrate_set`

<table>
<thead>
<tr>
<th>Function name</th>
<th><code>usart_baudrate_set</code></th>
</tr>
</thead>
<tbody>
<tr>
<td>Function prototype</td>
<td><code>void usart_baudrate_set(uint32_t usart_periph, uint32_t baudval)</code>;</td>
</tr>
<tr>
<td>Function descriptions</td>
<td>configure USART baud rate value</td>
</tr>
<tr>
<td>Precondition</td>
<td>-</td>
</tr>
<tr>
<td>The called functions</td>
<td><code>rcu_clock_freq_get</code></td>
</tr>
</tbody>
</table>

**Input parameter (in)**

<table>
<thead>
<tr>
<th><code>usart_periph</code></th>
<th>外设USARTx/UARTx</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>USARTx</code></td>
<td>x=0,1</td>
</tr>
<tr>
<td><code>UARTx</code></td>
<td>x=3,4</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th><code>baudval</code></th>
<th>baud rate value</th>
</tr>
</thead>
<tbody>
<tr>
<td>-</td>
<td>-</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Output parameter (out)</th>
<th>-</th>
</tr>
</thead>
</table>

| Return value | - |

**Example:**

```c
/* configure USART0 baud rate value */
usart_baudrate_set(USART0, 115200);
```

### usart_parity_config

The description of `usart_parity_config` is shown as below:

### Table 3-779. Function `usart_parity_config`

<table>
<thead>
<tr>
<th>Function name</th>
<th><code>usart_parity_config</code></th>
</tr>
</thead>
<tbody>
<tr>
<td>Function prototype</td>
<td><code>void usart_parity_config(uint32_t usart_periph, uint32_t paritycfg)</code>;</td>
</tr>
<tr>
<td>Function descriptions</td>
<td>configure USART parity function</td>
</tr>
<tr>
<td>Precondition</td>
<td>-</td>
</tr>
<tr>
<td>The called functions</td>
<td>-</td>
</tr>
</tbody>
</table>

**Input parameter (in)**

<table>
<thead>
<tr>
<th><code>usart_periph</code></th>
<th>外设USARTx/UARTx</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>USARTx</code></td>
<td>x=0,1</td>
</tr>
<tr>
<td><code>UARTx</code></td>
<td>x=3,4</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th><code>paritycfg</code></th>
<th>configure USART parity</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>USART_PM_NONE</code></td>
<td>no parity</td>
</tr>
<tr>
<td><code>USART_PM_ODD</code></td>
<td>odd parity</td>
</tr>
<tr>
<td><code>USART_PM_EVEN</code></td>
<td>even parity</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Output parameter (out)</th>
<th>-</th>
</tr>
</thead>
</table>

| Return value | - |
Example:

/* configure USART0 parity */
usart_parity_config(USART0, USART_PM_EVEN);

**usart_word_length_set**

The description of `usart_word_length_set` is shown as below:

<table>
<thead>
<tr>
<th>Table 3-780. Function <code>usart_word_length_set</code></th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Function name</strong></td>
</tr>
<tr>
<td><strong>Function prototype</strong></td>
</tr>
<tr>
<td><strong>Function descriptions</strong></td>
</tr>
<tr>
<td><strong>Precondition</strong></td>
</tr>
<tr>
<td><strong>The called functions</strong></td>
</tr>
<tr>
<td><strong>Input parameter</strong></td>
</tr>
<tr>
<td><code>usart_periph</code></td>
</tr>
<tr>
<td><code>USARTx</code></td>
</tr>
<tr>
<td><code>UARTx</code></td>
</tr>
<tr>
<td><strong>Input parameter</strong></td>
</tr>
<tr>
<td><code>wlen</code></td>
</tr>
<tr>
<td><code>USART_WL_8BIT</code></td>
</tr>
<tr>
<td><code>USART_WL_9BIT</code></td>
</tr>
<tr>
<td><strong>Output parameter</strong></td>
</tr>
<tr>
<td>-</td>
</tr>
<tr>
<td><strong>Return value</strong></td>
</tr>
</tbody>
</table>

Example:

/* configure USART0 word length */
usart_word_length_set(USART0, USART_WL_9BIT);

**usart_stop_bit_set**

The description of `usart_stop_bit_set` is shown as below:

<table>
<thead>
<tr>
<th>Table 3-781. Function <code>usart_stop_bit_set</code></th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Function name</strong></td>
</tr>
<tr>
<td><strong>Function prototype</strong></td>
</tr>
<tr>
<td><strong>Function descriptions</strong></td>
</tr>
<tr>
<td><strong>Precondition</strong></td>
</tr>
<tr>
<td><strong>The called functions</strong></td>
</tr>
</tbody>
</table>
**GD32L23x Firmware Library User Guide**

### Example:

/* configure USART0 stop bit length */

usart_stop_bit_set(USART0, USART_STB_1_5BIT);

### usart_enable

The description of `usart_enable` is shown as below:

**Table 3-782. Function `usart_enable`**

<table>
<thead>
<tr>
<th>Function name</th>
<th><code>usart_enable</code></th>
</tr>
</thead>
<tbody>
<tr>
<td>Function prototype</td>
<td><code>void usart_enable(uint32_t usart_periph);</code></td>
</tr>
<tr>
<td>Function descriptions</td>
<td>enable USART</td>
</tr>
<tr>
<td>Precondition</td>
<td>-</td>
</tr>
<tr>
<td>The called functions</td>
<td>-</td>
</tr>
</tbody>
</table>

**Input parameter(in)**

<table>
<thead>
<tr>
<th>usart_periph</th>
</tr>
</thead>
<tbody>
<tr>
<td>USARTx x=0,1</td>
</tr>
<tr>
<td>UARTx x=3,4</td>
</tr>
</tbody>
</table>

**Output parameter(out)**

| - |

**Return value**

| - |

Example:

/* enable USART0 */

usart_enable(USART0);
**usart_disable**

The description of `usart_disable` is shown as below:

<table>
<thead>
<tr>
<th>Function name</th>
<th>usart_disable</th>
</tr>
</thead>
<tbody>
<tr>
<td>Function prototype</td>
<td>void usart_disable(uint32_t usart_periph);</td>
</tr>
<tr>
<td>Function descriptions</td>
<td>disable USART</td>
</tr>
<tr>
<td>Precondition</td>
<td>-</td>
</tr>
<tr>
<td>The called functions</td>
<td>-</td>
</tr>
</tbody>
</table>

**Input parameter(in)**

<table>
<thead>
<tr>
<th>usart_periph</th>
<th>外设USARTx/UARTx</th>
</tr>
</thead>
<tbody>
<tr>
<td>USARTx</td>
<td>x=0,1</td>
</tr>
<tr>
<td>UARTx</td>
<td>x=3,4</td>
</tr>
</tbody>
</table>

**Output parameter(out)**

- -

**Return value**

- -

Example:

```
/* disable USART0 */
usart_disable(USART0);
```

**usart_transmit_config**

The description of `usart_transmit_config` is shown as below:

<table>
<thead>
<tr>
<th>Function name</th>
<th>usart_transmit_config</th>
</tr>
</thead>
<tbody>
<tr>
<td>Function prototype</td>
<td>void usart_transmit_config(uint32_t usart_periph, uint32_t txconfig);</td>
</tr>
<tr>
<td>Function descriptions</td>
<td>configure USART transmitter</td>
</tr>
<tr>
<td>Precondition</td>
<td>-</td>
</tr>
<tr>
<td>The called functions</td>
<td>-</td>
</tr>
</tbody>
</table>

**Input parameter(in)**

<table>
<thead>
<tr>
<th>usart_periph</th>
<th>外设USARTx/UARTx</th>
</tr>
</thead>
<tbody>
<tr>
<td>USARTx</td>
<td>x=0,1</td>
</tr>
<tr>
<td>UARTx</td>
<td>x=3,4</td>
</tr>
</tbody>
</table>

**Input parameter(in)**

<table>
<thead>
<tr>
<th>txconfig</th>
</tr>
</thead>
<tbody>
<tr>
<td>enable or disable USART transmitter</td>
</tr>
<tr>
<td><strong>USART_TRANSMIT_ENABLE</strong></td>
</tr>
<tr>
<td>enable USART transmission</td>
</tr>
<tr>
<td><strong>USART_TRANSMIT_DISABLE</strong></td>
</tr>
<tr>
<td>disable USART transmission</td>
</tr>
</tbody>
</table>

**Output parameter(out)**

- -
Example:

/* configure USART0 transmitter */
usart_transmit_config(USART0, USART_TRANSMIT_ENABLE);

### usart_receive_config

The description of usart_receive_config is shown as below:

<table>
<thead>
<tr>
<th>Function name</th>
<th>usart_receive_config</th>
</tr>
</thead>
<tbody>
<tr>
<td>Function prototype</td>
<td>void usart_receive_config(uint32_t usart_periph, uint32_t rxconfig);</td>
</tr>
<tr>
<td>Function descriptions</td>
<td>configure USART receiver</td>
</tr>
<tr>
<td>Precondition</td>
<td>-</td>
</tr>
<tr>
<td>The called functions</td>
<td>-</td>
</tr>
</tbody>
</table>

#### Input parameter(in)

<table>
<thead>
<tr>
<th>uart_periph</th>
<th>USARTx/UARTx</th>
</tr>
</thead>
<tbody>
<tr>
<td>USARTx</td>
<td>x=0,1</td>
</tr>
<tr>
<td>UARTx</td>
<td>x=3,4</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>rxconfig</th>
<th>enable or disable USART receiver</th>
</tr>
</thead>
<tbody>
<tr>
<td>USART_RECEIVE_ENABLE</td>
<td>enable USART reception</td>
</tr>
<tr>
<td>USART_RECEIVE_DISABLE</td>
<td>disable USART reception</td>
</tr>
</tbody>
</table>

#### Output parameter(out)

- 

#### Return value

- 

Example:

/* configure USART0 receiver */
usart_receive_config(USART0, USART_RECEIVE_ENABLE);

### usart_data_first_config

The description of usart_data_first_config is shown as below:

<table>
<thead>
<tr>
<th>Function name</th>
<th>usart_data_first_config</th>
</tr>
</thead>
<tbody>
<tr>
<td>Function prototype</td>
<td>void usart_data_first_config(uint32_t usart_periph, uint32_t msbf);</td>
</tr>
</tbody>
</table>
Function descriptions
data is transmitted/received with the LSB/MSB first

| Precondition | - |
| The called functions | - |

### Input parameter(in)

<table>
<thead>
<tr>
<th>usart_periph</th>
<th>外设USARTx/UARTx</th>
</tr>
</thead>
<tbody>
<tr>
<td>USARTx</td>
<td>x=0,1</td>
</tr>
<tr>
<td>UARTx</td>
<td>x=3,4</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>msbf</th>
<th>LSB/MSB</th>
</tr>
</thead>
<tbody>
<tr>
<td>USART_MSBF_LSB</td>
<td>LSB first</td>
</tr>
<tr>
<td>USART_MSBF_MSB</td>
<td>MSB first</td>
</tr>
</tbody>
</table>

### Output parameter(out)

-  -

Return value

-  -

Example:

/* configure LSB of data first */

```c
usrart_data_first_config(USART0, USART_MSBF_LSB);
```

### usart_invert_config

The description of `usart_invert_config` is shown as below:

#### Table 3-787. Function `usart_invert_config`

<table>
<thead>
<tr>
<th>Function name</th>
<th>usart_invert_config</th>
</tr>
</thead>
<tbody>
<tr>
<td>Function prototype</td>
<td>void usart_invert_config(uint32_t usart_periph, usart_invert_enum invertpara);</td>
</tr>
<tr>
<td>Function descriptions</td>
<td>configure USART inverted</td>
</tr>
<tr>
<td>Precondition</td>
<td>-</td>
</tr>
<tr>
<td>The called functions</td>
<td>-</td>
</tr>
</tbody>
</table>

### Input parameter(in)

<table>
<thead>
<tr>
<th>usart_periph</th>
<th>外设USARTx/UARTx</th>
</tr>
</thead>
<tbody>
<tr>
<td>USARTx</td>
<td>x=0,1</td>
</tr>
<tr>
<td>UARTx</td>
<td>x=3,4</td>
</tr>
</tbody>
</table>

### Input parameter(in)

<table>
<thead>
<tr>
<th>invertpara</th>
<th>refer to <code>Table 3-776. Enum usart_invert_enum</code></th>
</tr>
</thead>
<tbody>
<tr>
<td><code>USART_DINV_ENABL</code></td>
<td>data bit level inversion</td>
</tr>
<tr>
<td><code>USART_DINV_DISABL</code></td>
<td>data bit level not inversion</td>
</tr>
<tr>
<td><code>USART_TXPIN_ENABL</code></td>
<td>TX pin level inversion</td>
</tr>
</tbody>
</table>
### USART TX Pin Level Configuration

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>USART_TXPIN_DISABLE</td>
<td>TX pin level not inversion</td>
</tr>
<tr>
<td>USART_TXPIN_ENABLE</td>
<td>TX pin level inversion</td>
</tr>
</tbody>
</table>

### USART RX Pin Level Configuration

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>USART_RXPIN_DISABLE</td>
<td>RX pin level not inversion</td>
</tr>
<tr>
<td>USART_RXPIN_ENABLE</td>
<td>RX pin level inversion</td>
</tr>
</tbody>
</table>

### USART Swap Pin Configuration

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>USART_SWAP_DISABLE</td>
<td>not swap TX/RX pins</td>
</tr>
<tr>
<td>USART_SWAP_ENABLE</td>
<td>swap TX/RX pins</td>
</tr>
</tbody>
</table>

#### Output Parameter (Out)

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
</tr>
</tbody>
</table>

#### Return Value

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Example:**

```c
/* configure USART0 inversion */
usart_invert_config(USART0, USART_DINV_ENABLE);
```

### USART Overrun Enable

The description of `usart_overrun_enable` is shown as below:

<table>
<thead>
<tr>
<th>Function name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>usart_overrun_enable</code></td>
<td>Enable the USART overrun function</td>
</tr>
</tbody>
</table>

#### Function Prototype

```c
void usart_overrun_enable(uint32_t usart_periph);
```

#### Input Parameter (In)

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>usart_periph</code></td>
<td>USARTx/UARTx</td>
</tr>
</tbody>
</table>

- For `USARTx` with `x=0,1`:
  ```c
  USARTx
  x=0,1
  ```

- For `UARTx` with `x=3,4`:
  ```c
  UARTx
  x=3,4
  ```

#### Output Parameter (Out)

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
</tr>
</tbody>
</table>

#### Return Value

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Example:**

```c
/* enable USART0 overrun */
usart_overrun_enable(USART0);
```
**user_overrun_disable**

The description of `user_overrun_disable` is shown as below:

**Table 3-789. Function user_overrun_disable**

<table>
<thead>
<tr>
<th>Function name</th>
<th>user_overrun_disable</th>
</tr>
</thead>
<tbody>
<tr>
<td>Function prototype</td>
<td>void user_overrun_disable(uint32_t usart_periph);</td>
</tr>
<tr>
<td>Function descriptions</td>
<td>disable the USART overrun function</td>
</tr>
<tr>
<td>Precondition</td>
<td>-</td>
</tr>
<tr>
<td>The called functions</td>
<td>-</td>
</tr>
</tbody>
</table>

**Input parameter (in)**

<table>
<thead>
<tr>
<th>usart_periph</th>
<th>USARTx x=0,1</th>
</tr>
</thead>
<tbody>
<tr>
<td>UARTx</td>
<td>x=3,4</td>
</tr>
</tbody>
</table>

**Output parameter (out)**

- -

**Return value**

Example:

```c
/* disable USART0 overrun */
usart_overrun_disable(USART0);
```

**user_oversample_config**

The description of `user_oversample_config` is shown as below:

**Table 3-790. Function user_oversample_config**

<table>
<thead>
<tr>
<th>Function name</th>
<th>user_oversample_config</th>
</tr>
</thead>
<tbody>
<tr>
<td>Function prototype</td>
<td>void user_oversample_config(uint32_t usart_periph, uint32_t oversamp);</td>
</tr>
<tr>
<td>Function descriptions</td>
<td>configure the USART oversample mode</td>
</tr>
<tr>
<td>Precondition</td>
<td>-</td>
</tr>
<tr>
<td>The called functions</td>
<td>-</td>
</tr>
</tbody>
</table>

**Input parameter (in)**

<table>
<thead>
<tr>
<th>usart_periph</th>
<th>外设USARTx/UARTx</th>
</tr>
</thead>
<tbody>
<tr>
<td>USARTx</td>
<td>x=0,1</td>
</tr>
<tr>
<td>UARTx</td>
<td>x=3,4</td>
</tr>
</tbody>
</table>

**Input parameter (in)**

<table>
<thead>
<tr>
<th>oversamp</th>
<th>oversample value</th>
</tr>
</thead>
<tbody>
<tr>
<td>USART_OVSMOD_8</td>
<td>oversampling by 8</td>
</tr>
<tr>
<td>USART_OVSMOD_16</td>
<td>oversampling by 16</td>
</tr>
</tbody>
</table>

**Output parameter (out)**

- -

**Return value**
Example:

/* config USART0 oversampling by 8 */
usart_oversample_config(USART0, USART_OVSMOD_8);

**usart_sample_bit_config**

The description of `usart_sample_bit_config` is shown as below:

<table>
<thead>
<tr>
<th>Function name</th>
<th>usart_sample_bit_config</th>
</tr>
</thead>
<tbody>
<tr>
<td>Function prototype</td>
<td>void usart_sample_bit_config(uint32_t usart_periph, uint32_t osb);</td>
</tr>
<tr>
<td>Function descriptions</td>
<td>configure the sample bit method</td>
</tr>
</tbody>
</table>

**Precondition**

- 

**The called functions**

- 

**Input parameter**(in)

<table>
<thead>
<tr>
<th>usart_periph</th>
<th>外设USARTx/UARTx</th>
</tr>
</thead>
<tbody>
<tr>
<td>USARTx</td>
<td>x=0,1</td>
</tr>
<tr>
<td>UARTx</td>
<td>x=3,4</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>osb</th>
<th>sample bit</th>
</tr>
</thead>
<tbody>
<tr>
<td>USART_OSB_1BIT</td>
<td>1 bit</td>
</tr>
<tr>
<td>USART_OSB_3BIT</td>
<td>3 bits</td>
</tr>
</tbody>
</table>

**Output parameter**(out)

- 

**Return value**

- 

Example:

/* config USART0 1 bit sample mode */
usart_sample_bit_config(USART0, USART_OSB_1BIT);

**usart_receiver_timeout_enable**

The description of `usart_receiver_timeout_enable` is shown as below:

<table>
<thead>
<tr>
<th>Function name</th>
<th>usart_receiver_timeout_enable</th>
</tr>
</thead>
<tbody>
<tr>
<td>Function prototype</td>
<td>void usart_receiver_timeout_enable(uint32_t usart_periph);</td>
</tr>
<tr>
<td>Function descriptions</td>
<td>enable receiver timeout</td>
</tr>
</tbody>
</table>

**Precondition**

- 

**The called functions**

-
GD32L23x Firmware Library User Guide

**usart_periph**

<table>
<thead>
<tr>
<th>Input parameter (in)</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>usart_periph</strong></td>
<td>usart peripheral</td>
</tr>
<tr>
<td><strong>USARTx</strong></td>
<td>x=0,1</td>
</tr>
</tbody>
</table>

**Output parameter (out)**

<table>
<thead>
<tr>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>-</td>
</tr>
</tbody>
</table>

**Return value**

<table>
<thead>
<tr>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>-</td>
</tr>
</tbody>
</table>

Example:

/* enable USART0 receiver timeout */

usart_receiver_timeout_enable(USART0);

**usart_receiver_timeout_disable**

The description of `usart_receiver_timeout_disable` is shown as below:

<table>
<thead>
<tr>
<th>Function name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>usart_receiver_timeout_disable</code></td>
<td>disable receiver timeout</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Function prototype</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>void usart_receiver_timeout_disable(uint32_t usart_periph);</code></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Precondition</th>
</tr>
</thead>
<tbody>
<tr>
<td>-</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>The called functions</th>
</tr>
</thead>
<tbody>
<tr>
<td>-</td>
</tr>
</tbody>
</table>

**Input parameter (in)**

<table>
<thead>
<tr>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>-</td>
</tr>
</tbody>
</table>

**Output parameter (out)**

<table>
<thead>
<tr>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>-</td>
</tr>
</tbody>
</table>

**Return value**

<table>
<thead>
<tr>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>-</td>
</tr>
</tbody>
</table>

Example:

/* disable USART0 receiver timeout */

usart_receiver_timeout_disable(USART0);

**usart_receiver_timeout_threshold_config**

The description of `usart_receiver_timeout_threshold_config` is shown as below:

<table>
<thead>
<tr>
<th>Function name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>usart_receiver_timeout_threshold_config</code></td>
<td>configure receiver timeout threshold</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Function prototype</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>void usart_receiver_timeout_threshold_config(uint32_t usart_periph, uint32_t rtimeout);</code></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Precondition</th>
</tr>
</thead>
<tbody>
<tr>
<td>-</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>The called functions</th>
</tr>
</thead>
<tbody>
<tr>
<td>-</td>
</tr>
</tbody>
</table>

**Input parameter (in)**

<table>
<thead>
<tr>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>-</td>
</tr>
</tbody>
</table>

**Output parameter (out)**

<table>
<thead>
<tr>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>-</td>
</tr>
</tbody>
</table>

**Return value**

<table>
<thead>
<tr>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>-</td>
</tr>
</tbody>
</table>
The called functions

**Input parameter**

- `usart_periph` - `usart_periph` (USARTx/UARTx)  
  - `x=0,1`

**Input parameter**

- `rttimeout` - `receiver timeout (0x00000000-0xFFFFFFFF)`

**Output parameter**

- `-`

**Return value**

- `-`

Example:

/* set the receiver timeout threshold of USART0*/

`usart_receiver_timeout_threshold_config(USART0, 115200*3);`

### `usart_data_transmit`

The description of `usart_data_transmit` is shown as below:

**Table 3-795. Function `usart_data_transmit`**

<table>
<thead>
<tr>
<th>Function name</th>
<th><code>usart_data_transmit</code></th>
</tr>
</thead>
<tbody>
<tr>
<td>Function prototype</td>
<td><code>void usart_data_transmit(uint32_t usart_periph, uint32_t data);</code></td>
</tr>
<tr>
<td>Function descriptions</td>
<td><code>USART transmit data function</code></td>
</tr>
<tr>
<td>Precondition</td>
<td><code>-</code></td>
</tr>
<tr>
<td>The called functions</td>
<td><code>-</code></td>
</tr>
</tbody>
</table>

**Input parameter**

- `usart_periph` - `usart_periph` (USARTx/UARTx)  
  - `x=0,1`

- `UARTx` - `x=3,4`

**Input parameter**

- `data` - `data of transmission (0x00-0x1FF)`

**Output parameter**

- `-`

**Return value**

- `-`

Example:

/* USART0 transmit data */

`usart_data_transmit(USART0, 0xAA);`
The description of `usart_data_receive` is shown as below:

**Table 3-796. Function `usart_data_receive`**

<table>
<thead>
<tr>
<th>Function name</th>
<th>usart_data_receive</th>
</tr>
</thead>
<tbody>
<tr>
<td>Function prototype</td>
<td>uint16_t usart_data_receive(uint32_t usart_periph);</td>
</tr>
<tr>
<td>Function descriptions</td>
<td>USART receive data function</td>
</tr>
<tr>
<td>Precondition</td>
<td>-</td>
</tr>
<tr>
<td>The called functions</td>
<td>-</td>
</tr>
</tbody>
</table>

**Input parameter (in)**

<table>
<thead>
<tr>
<th>usart_periph</th>
<th>外设USARTx/UARTx</th>
</tr>
</thead>
<tbody>
<tr>
<td>USARTx</td>
<td>x=0,1</td>
</tr>
<tr>
<td>UARTx</td>
<td>x=3,4</td>
</tr>
</tbody>
</table>

**Output parameter (out)**

- -

**Return value**

| uint16_t       | data of received（0x00-0x1FF） |

Example:

```c
/* USART0 receive data */
uint16_t temp;
temp = usart_data_receive(USART0);
```

The description of `usart_command_enable` is shown as below:

**Table 3-797. Function `usart_command_enable`**

<table>
<thead>
<tr>
<th>Function name</th>
<th>usart_command_enable</th>
</tr>
</thead>
<tbody>
<tr>
<td>Function prototype</td>
<td>void usart_command_enable(uint32_t usart_periph, uint32_t cmdtype);</td>
</tr>
<tr>
<td>Function descriptions</td>
<td>enable USART command</td>
</tr>
<tr>
<td>Precondition</td>
<td>-</td>
</tr>
<tr>
<td>The called functions</td>
<td>-</td>
</tr>
</tbody>
</table>

**Input parameter (in)**

<table>
<thead>
<tr>
<th>usart_periph</th>
<th>外设USARTx/UARTx</th>
</tr>
</thead>
<tbody>
<tr>
<td>USARTx</td>
<td>x=0,1</td>
</tr>
<tr>
<td>UARTx</td>
<td>x=3,4</td>
</tr>
</tbody>
</table>

**Input parameter (in)**

<table>
<thead>
<tr>
<th>cmdtype</th>
<th>command type</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>USART_CMD_ABDCM</code></td>
<td>auto baudrate detection command</td>
</tr>
<tr>
<td><code>USART_CMD_SBKCM</code></td>
<td>send break command</td>
</tr>
</tbody>
</table>
### USART_CMD_MMCMD
- **Function**: mute mode command
- **Output parameter** (out): 
  - 
- **Return value**: 
  - 

### USART_CMD_RXFCM
- **Function**: receive data flush command
- Output parameter (out): 
  - 
- Return value: 
  - 

### USART_CMD_TXFCM
- **Function**: transmit data flush request
- Output parameter (out): 
  - 
- Return value: 
  - 

---

**Example:**

/* enable USART0 command */

usrart_command_enable(USART0, USART_CMD_ABDCMD);

**usrart_autobaud_detection_enable**

The description of usart_autobaud_detection_enable is shown as below:

**Table 3-798. Function usart_autobaud_detection_enable**

<table>
<thead>
<tr>
<th>Function name</th>
<th>usart_autobaud_detection_enable</th>
</tr>
</thead>
<tbody>
<tr>
<td>Function prototype</td>
<td>void usart_autobaud_detection_enable(uint32_t usart_periph);</td>
</tr>
<tr>
<td>Function descriptions</td>
<td>enable auto baud rate detection</td>
</tr>
<tr>
<td>Precondition</td>
<td>-</td>
</tr>
<tr>
<td>The called functions</td>
<td>-</td>
</tr>
<tr>
<td>Input parameter (in)</td>
<td></td>
</tr>
<tr>
<td>usart_periph</td>
<td>usart peripheral</td>
</tr>
<tr>
<td>USARTx</td>
<td>x=0,1</td>
</tr>
<tr>
<td>Output parameter (out)</td>
<td></td>
</tr>
<tr>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>Return value</td>
<td>-</td>
</tr>
</tbody>
</table>

**Example:**

/* enable USART0 auto baud rate detection */

usrart_autobaud_detection_enable(USART0);

**usrart_autobaud_detection_disable**

The description of usart_autobaud_detection_disable is shown as below:

**Table 3-799. Function usart_autobaud_detection_disable**

<table>
<thead>
<tr>
<th>Function name</th>
<th>usart_autobaud_detection_disable</th>
</tr>
</thead>
<tbody>
<tr>
<td>Function prototype</td>
<td>void usart_autobaud_detection_disable(uint32_t usart_periph);</td>
</tr>
</tbody>
</table>
**Function description**

disable auto baud rate detection

**Precondition**
- 

**The called functions**
- 

<table>
<thead>
<tr>
<th>Input parameter(in)</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>usart_periph</strong></td>
</tr>
<tr>
<td><strong>USARTx</strong></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Output parameter(out)</th>
</tr>
</thead>
<tbody>
<tr>
<td>-</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Return value</th>
</tr>
</thead>
<tbody>
<tr>
<td>-</td>
</tr>
</tbody>
</table>

**Example:**

```
/* disable USART0 auto baud rate detection */
usart_autobaud_detection_disable(USART0);
```

**usart_autobaud_detection_mode_config**

The description of **usart_autobaud_detection_mode_config** is shown as below:

**Table 3-800. Function usart_autobaud_detection_mode_config**

<table>
<thead>
<tr>
<th>Function name</th>
<th>usart_autobaud_detection_mode_config</th>
</tr>
</thead>
<tbody>
<tr>
<td>Function prototype</td>
<td>void usart_autobaud_detection_mode_config(uint32_t usart_periph, uint32_t abdmod);</td>
</tr>
<tr>
<td>Function description</td>
<td>configure auto baud rate detection mode</td>
</tr>
<tr>
<td>Precondition</td>
<td>-</td>
</tr>
<tr>
<td>The called functions</td>
<td>-</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Input parameter(in)</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>usart_periph</strong></td>
</tr>
<tr>
<td><strong>USARTx</strong></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>abdmod</th>
<th>auto baud rate detection mode</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>USART_ABDM_FTOR</strong></td>
<td>falling edge to rising edge measurement</td>
</tr>
<tr>
<td><strong>USART_ABDM_FTOF</strong></td>
<td>falling edge to falling edge measurement</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Output parameter(out)</th>
</tr>
</thead>
<tbody>
<tr>
<td>-</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Return value</th>
</tr>
</thead>
<tbody>
<tr>
<td>-</td>
</tr>
</tbody>
</table>

**Example:**

```
/* configure USART0 auto baud rate detection mode */
usart_autobaud_detection_mode_config(USART0, USART_ABDM_FTOR);
```
**usrart_address_config**

The description of `usrart_address_config` is shown as below:

**Table 3-801. Function `usrart_address_config`**

<table>
<thead>
<tr>
<th>Function name</th>
<th><code>usrart_address_config</code></th>
</tr>
</thead>
<tbody>
<tr>
<td>Function prototype</td>
<td><code>void usrart_address_config(uint32_t usart_periph, uint8_t addr);</code></td>
</tr>
<tr>
<td>Function descriptions</td>
<td>configure the address of the USART</td>
</tr>
<tr>
<td>Precondition</td>
<td>-</td>
</tr>
<tr>
<td>The called functions</td>
<td>-</td>
</tr>
<tr>
<td><strong>Input parameter(in)</strong></td>
<td></td>
</tr>
<tr>
<td><code>usart_periph</code></td>
<td>外设USARTx/UARTx</td>
</tr>
<tr>
<td><code>USARTx</code></td>
<td>x=0,1</td>
</tr>
<tr>
<td><code>UARTx</code></td>
<td>x=3,4</td>
</tr>
<tr>
<td><strong>Input parameter(in)</strong></td>
<td></td>
</tr>
<tr>
<td><code>addr</code></td>
<td>address of USART (0x00-0xFF)</td>
</tr>
<tr>
<td><strong>Output parameter(out)</strong></td>
<td></td>
</tr>
<tr>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td><strong>Return value</strong></td>
<td>-</td>
</tr>
</tbody>
</table>

Example:

```
/* configure address of the USART0 */
usrart_address_config(USART0, 0x00);
```

**usrart_address_detection_mode_config**

The description of `usrart_address_detection_mode_config` is shown as below:

**Table 3-802. Function `usrart_address_detection_mode_config`**

<table>
<thead>
<tr>
<th>Function name</th>
<th><code>usrart_address_detection_mode_config</code></th>
</tr>
</thead>
<tbody>
<tr>
<td>Function prototype</td>
<td><code>void usrart_address_detection_mode_config(uint32_t usart_periph, uint32_t addmod);</code></td>
</tr>
<tr>
<td>Function descriptions</td>
<td>configure address detection mode</td>
</tr>
<tr>
<td>Precondition</td>
<td>-</td>
</tr>
<tr>
<td>The called functions</td>
<td>-</td>
</tr>
<tr>
<td><strong>Input parameter(in)</strong></td>
<td></td>
</tr>
<tr>
<td><code>usart_periph</code></td>
<td>外设USARTx/UARTx</td>
</tr>
<tr>
<td><code>USARTx</code></td>
<td>x=0,1</td>
</tr>
<tr>
<td><code>UARTx</code></td>
<td>x=3,4</td>
</tr>
<tr>
<td><strong>Input parameter(in)</strong></td>
<td></td>
</tr>
<tr>
<td><code>addmod</code></td>
<td>address detection mode</td>
</tr>
<tr>
<td><code>USART_ADDM_4BIT</code></td>
<td>4 bits</td>
</tr>
<tr>
<td><code>USART_ADDM_FULLB</code></td>
<td>full bits</td>
</tr>
</tbody>
</table>
GD32L23x Firmware Library User Guide

Output parameter(out)

<table>
<thead>
<tr>
<th>IT</th>
</tr>
</thead>
<tbody>
<tr>
<td>-</td>
</tr>
</tbody>
</table>

Return value

<table>
<thead>
<tr>
<th>IT</th>
</tr>
</thead>
<tbody>
<tr>
<td>-</td>
</tr>
</tbody>
</table>

Example:

/*configure address detection mode */

usart_address_config(USART0, USART_ADDM_4BIT);

**usart_mute_mode_enable**

The description of **usart_mute_mode_enable** is shown as below:

<table>
<thead>
<tr>
<th>Function name</th>
<th>usart_mute_mode_enable</th>
</tr>
</thead>
<tbody>
<tr>
<td>Function prototype</td>
<td>void usart_mute_mode_enable(uint32_t usart_periph);</td>
</tr>
<tr>
<td>Function descriptions</td>
<td>enable mute mode</td>
</tr>
<tr>
<td>Precondition</td>
<td>-</td>
</tr>
<tr>
<td>The called functions</td>
<td>-</td>
</tr>
</tbody>
</table>

Input parameter(in)

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>usart_periph</td>
<td>外设USARTx/UARTx</td>
</tr>
<tr>
<td>USARTx</td>
<td>x=0,1</td>
</tr>
<tr>
<td>UARTx</td>
<td>x=3,4</td>
</tr>
</tbody>
</table>

Output parameter(out)

<table>
<thead>
<tr>
<th>IT</th>
</tr>
</thead>
<tbody>
<tr>
<td>-</td>
</tr>
</tbody>
</table>

Return value

<table>
<thead>
<tr>
<th>IT</th>
</tr>
</thead>
<tbody>
<tr>
<td>-</td>
</tr>
</tbody>
</table>

Example:

/* enable USART0 receiver in mute mode */

usart_mute_mode_enable(USART0);

**usart_mute_mode_disable**

The description of **usart_mute_mode_disable** is shown as below:

<table>
<thead>
<tr>
<th>Function name</th>
<th>usart_mute_mode_disable</th>
</tr>
</thead>
<tbody>
<tr>
<td>Function prototype</td>
<td>void usart_mute_mode_disable(uint32_t usart_periph);</td>
</tr>
<tr>
<td>Function descriptions</td>
<td>disable mute mode</td>
</tr>
<tr>
<td>Precondition</td>
<td>-</td>
</tr>
<tr>
<td>The called functions</td>
<td>-</td>
</tr>
</tbody>
</table>
Example:

/* disable USART0 receiver in mute mode */
usart_mute_mode_disable(USART0);

`usart_mute_mode_wakeup_config`

The description of `usart_mute_mode_wakeup_config` is shown as below:

**Table 3-805. Function usart_mute_mode_wakeup_config**

<table>
<thead>
<tr>
<th>Function name</th>
<th>usart_mute_mode_wakeup_config</th>
</tr>
</thead>
<tbody>
<tr>
<td>Function prototype</td>
<td>void usart_mute_mode_wakeup_config(uint32_t usart_periph, uint32_t wmethod);</td>
</tr>
<tr>
<td>Function descriptions</td>
<td>configure wakeup method in mute mode</td>
</tr>
<tr>
<td>Precondition</td>
<td>-</td>
</tr>
<tr>
<td>The called functions</td>
<td>-</td>
</tr>
</tbody>
</table>

**Input parameter (in)**

<table>
<thead>
<tr>
<th>usart_periph</th>
<th>外设USARTx/UARTx</th>
</tr>
</thead>
<tbody>
<tr>
<td>USARTx</td>
<td>x=0,1</td>
</tr>
<tr>
<td>UARTx</td>
<td>x=3,4</td>
</tr>
</tbody>
</table>

**Output parameter (out)**

| - | - |

**Return value**

| - | - |

Example:

/* configure USART0 wakeup method in mute mode */
usart_mute_mode_wakeup_config(USART0, USART_WM_IDLE);
usrat_lin_mode_enable

The description of usrat_lin_mode_enable is shown as below:

Table 3-806. Function usrat_lin_mode_enable

<table>
<thead>
<tr>
<th>Function name</th>
<th>usrat_lin_mode_enable</th>
</tr>
</thead>
<tbody>
<tr>
<td>Function prototype</td>
<td>void usrat_lin_mode_enable(uint32_t usart_periph);</td>
</tr>
<tr>
<td>Function descriptions</td>
<td>enable LIN mode</td>
</tr>
<tr>
<td>Precondition</td>
<td>-</td>
</tr>
<tr>
<td>The called functions</td>
<td>-</td>
</tr>
<tr>
<td>Input parameter (in)</td>
<td></td>
</tr>
<tr>
<td>usart_periph</td>
<td>usart peripheral</td>
</tr>
<tr>
<td>USARTx</td>
<td>x=0,1</td>
</tr>
<tr>
<td>Output parameter (out)</td>
<td>-</td>
</tr>
<tr>
<td>Return value</td>
<td>-</td>
</tr>
</tbody>
</table>

Example:

/* USART0 LIN mode enable */
usrat_lin_mode_enable(USART0);

usrat_lin_mode_disable

The description of usrat_lin_mode_disable is shown as below:

Table 3-807. Function usrat_lin_mode_disable

<table>
<thead>
<tr>
<th>Function name</th>
<th>usrat_lin_mode_disable</th>
</tr>
</thead>
<tbody>
<tr>
<td>Function prototype</td>
<td>void usrat_lin_mode_disable(uint32_t usart_periph);</td>
</tr>
<tr>
<td>Function descriptions</td>
<td>disable LIN mode</td>
</tr>
<tr>
<td>Precondition</td>
<td>-</td>
</tr>
<tr>
<td>The called functions</td>
<td>-</td>
</tr>
<tr>
<td>Input parameter (in)</td>
<td></td>
</tr>
<tr>
<td>usart_periph</td>
<td>usart peripheral</td>
</tr>
<tr>
<td>USARTx</td>
<td>x=0,1</td>
</tr>
<tr>
<td>Output parameter (out)</td>
<td>-</td>
</tr>
<tr>
<td>Return value</td>
<td>-</td>
</tr>
</tbody>
</table>

Example:

/* USART0 LIN mode disable */
usrat_lin_mode_disable(USART0);
**usart_lin_break_dection_length_config**

The description of `usart_lin_break_dection_length_config` is shown as below:

<table>
<thead>
<tr>
<th>Function name</th>
<th><code>usart_lin_break_dection_length_config</code></th>
</tr>
</thead>
<tbody>
<tr>
<td>Function prototype</td>
<td><code>void usart_lin_break_dection_length_config(uint32_t usart_periph, uint32_t lblen);</code></td>
</tr>
<tr>
<td>Function descriptions</td>
<td>LIN break detection length</td>
</tr>
<tr>
<td>Precondition</td>
<td>-</td>
</tr>
<tr>
<td>The called functions</td>
<td>-</td>
</tr>
</tbody>
</table>

**Input parameter (in)**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>usart_periph</code></td>
<td><code>usart</code> peripheral</td>
</tr>
<tr>
<td><code>USARTx</code></td>
<td><code>x=0,1</code></td>
</tr>
</tbody>
</table>

**Input parameter (in)**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>lblen</code></td>
<td>two methods be used to enter or exit the mute mode</td>
</tr>
<tr>
<td><code>USART_LBLEN_10B</code></td>
<td>10 bits</td>
</tr>
<tr>
<td><code>USART_LBLEN_11B</code></td>
<td>11 bits</td>
</tr>
</tbody>
</table>

**Output parameter (out)**

- -

**Return value**

Example:

```c
/* configure LIN break frame length */
usart_lin_break_dection_length_config(USART0, USART_LBLEN_10B);
```

**usart_halfduplex_enable**

The description of `usart_halfduplex_enable` is shown as below:

<table>
<thead>
<tr>
<th>Function name</th>
<th><code>usart_halfduplex_enable</code></th>
</tr>
</thead>
<tbody>
<tr>
<td>Function prototype</td>
<td><code>void usart_halfduplex_enable(uint32_t usart_periph);</code></td>
</tr>
<tr>
<td>Function descriptions</td>
<td>enable half-duplex mode</td>
</tr>
<tr>
<td>Precondition</td>
<td>-</td>
</tr>
<tr>
<td>The called functions</td>
<td>-</td>
</tr>
</tbody>
</table>

**Input parameter (in)**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>usart_periph</code></td>
<td><code>外设USARTx/UARTx</code></td>
</tr>
<tr>
<td><code>USARTx</code></td>
<td><code>x=0,1</code></td>
</tr>
<tr>
<td><code>UARTx</code></td>
<td><code>x=3,4</code></td>
</tr>
</tbody>
</table>

**Output parameter (out)**

- -

**Return value**
Example:

/* enable USART0 half duplex mode*/
usart_halfduplex_enable(USART0);

**usart_halfduplex_disable**

The description of usart_halfduplex_disable is shown as below:

<table>
<thead>
<tr>
<th>Function name</th>
<th>usart_halfduplex_disable</th>
</tr>
</thead>
<tbody>
<tr>
<td>Function prototype</td>
<td>void usart_halfduplex_disable(uint32_t usart_periph);</td>
</tr>
<tr>
<td>Function descriptions</td>
<td>disable half-duplex mode</td>
</tr>
<tr>
<td>Precondition</td>
<td>-</td>
</tr>
<tr>
<td>The called functions</td>
<td>-</td>
</tr>
</tbody>
</table>

**Input parameter** (in)

- `usart_periph` 外设USARTx/UARTx
  - `USARTx` x=0,1
  - `UARTx` x=3,4

**Output parameter** (out)

- -

**Return value**

- -

Example:

/* disable USART0 half duplex mode*/
usart_halfduplex_disable(USART0);

**usart_clock_enable**

The description of usart_clock_enable is shown as below:

<table>
<thead>
<tr>
<th>Function name</th>
<th>usart_clock_enable</th>
</tr>
</thead>
<tbody>
<tr>
<td>Function prototype</td>
<td>void usart_clock_enable(uint32_t usart_periph);</td>
</tr>
<tr>
<td>Function descriptions</td>
<td>enable clock</td>
</tr>
<tr>
<td>Precondition</td>
<td>-</td>
</tr>
<tr>
<td>The called functions</td>
<td>-</td>
</tr>
</tbody>
</table>

**Input parameter** (in)

- `usart_periph` usart peripheral
  - `USARTx` x=0,1

**Output parameter** (out)

- -
Example:

/* enable clock */

usart_clock_enable(USART0);

**usart_clock_disable**

The description of usart_clock_disable is shown as below:

<table>
<thead>
<tr>
<th>Function</th>
<th>usart_clock_disable</th>
</tr>
</thead>
<tbody>
<tr>
<td>Function prototype</td>
<td>void usart_clock_disable(uint32_t usart_periph);</td>
</tr>
<tr>
<td>Function description</td>
<td>disable clock</td>
</tr>
<tr>
<td>Precondition</td>
<td>-</td>
</tr>
<tr>
<td>The called functions</td>
<td>-</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Input parameter (in)</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>usart_periph</td>
<td>usart peripheral</td>
</tr>
<tr>
<td>USARTx</td>
<td>x=0, 1</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Output parameter (out)</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>-</td>
<td>-</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Return value</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>-</td>
<td>-</td>
</tr>
</tbody>
</table>

Example:

/* disable clock */

usart_clock_disable(USART0);

**usart_synchronous_clock_config**

The description of usart_synchronous_clock_config is shown as below:

<table>
<thead>
<tr>
<th>Function name</th>
<th>usart_synchronous_clock_config</th>
</tr>
</thead>
<tbody>
<tr>
<td>Function prototype</td>
<td>void usart_synchronous_clock_config(uint32_t usart_periph, uint32_t clen, uint32_t cph, uint32_t cp1);</td>
</tr>
<tr>
<td>Function description</td>
<td>configure USART synchronous mode parameters</td>
</tr>
<tr>
<td>Precondition</td>
<td>-</td>
</tr>
<tr>
<td>The called functions</td>
<td>-</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Input parameter (in)</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>usart_periph</td>
<td>usart peripheral</td>
</tr>
</tbody>
</table>

|  |  |
Example:

/* configure USART0 synchronous mode parameters */

usart_synchronous_clock_config(USART0, USART_CLEN_EN, USART_CPH_2CK, USART_CPL_HIGH);

**usart_guard_time_config**

The description of usart_guard_time_config is shown as below:

**Table 3-814. Function usart_guard_time_config**

<table>
<thead>
<tr>
<th>Function name</th>
<th>Function prototype</th>
<th>Function descriptions</th>
<th>Precondition</th>
<th>The called functions</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>void usart_guard_time_config(uint32_t usart_periph,uint32_t guat);</td>
<td>configure guard time value in smartcard mode</td>
<td>-</td>
<td>-</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Input parameter(in)</th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>usart_periph</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>USARTx</td>
<td>x=0,1</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Input parameter(in)</th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>guat</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Output parameter(out)</th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Return value</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Example:

/* configure USART0 guard time value in smartcard mode */
usart.guard_time_config(USART0, 0x0000 0055);

**usart.smartcard_mode_enable**

The description of `usart_smartcard_mode_enable` is shown as below:

**Table 3-815. Function usart_smartcard_mode_enable**

<table>
<thead>
<tr>
<th>Function name</th>
<th>usart.smartcard_mode_enable</th>
</tr>
</thead>
<tbody>
<tr>
<td>Function prototype</td>
<td>void usart.smartcard_mode_enable(uint32_t usart_periph);</td>
</tr>
<tr>
<td>Function descriptions</td>
<td>enable smartcard mode</td>
</tr>
<tr>
<td>Precondition</td>
<td>-</td>
</tr>
<tr>
<td>The called functions</td>
<td>-</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Input parameter(in)</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>usart_periph</td>
<td>usart peripheral</td>
</tr>
<tr>
<td><code>USARTx</code></td>
<td>x=0,1</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Output parameter(out)</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>-</td>
<td>-</td>
</tr>
</tbody>
</table>

| Return value                 | -                         |

Example:

/* USART0 smartcard mode enable */
usart_smartcard_mode_enable(USART0);

**usart.smartcard_mode_disable**

The description of `usart_smartcard_mode_disable` is shown as below:

**Table 3-816. Function usart_smartcard_mode_disable**

<table>
<thead>
<tr>
<th>Function name</th>
<th>usart.smartcard_mode_disable</th>
</tr>
</thead>
<tbody>
<tr>
<td>Function prototype</td>
<td>void usart.smartcard_mode_disable(uint32_t usart_periph);</td>
</tr>
<tr>
<td>Function descriptions</td>
<td>disable smartcard mode</td>
</tr>
<tr>
<td>Precondition</td>
<td>-</td>
</tr>
<tr>
<td>The called functions</td>
<td>-</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Input parameter(in)</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>usart_periph</td>
<td>usart peripheral</td>
</tr>
<tr>
<td><code>USARTx</code></td>
<td>x=0,1</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Output parameter(out)</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>-</td>
<td>-</td>
</tr>
</tbody>
</table>

| Return value                 | -                         |
Example:

/* USART0 smartcard mode disable */
usart_smartcard_mode_disable(USART0);

**usart_smartcard_mode_nack_enable**

The description of **usart_smartcard_mode_nack_enable** is shown as below:

<table>
<thead>
<tr>
<th>Table 3-817. Function usart_smartcard_mode_nack_enable</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Function name</strong></td>
</tr>
<tr>
<td><strong>Function prototype</strong></td>
</tr>
<tr>
<td><strong>Function descriptions</strong></td>
</tr>
<tr>
<td><strong>Precondition</strong></td>
</tr>
<tr>
<td><strong>The called functions</strong></td>
</tr>
<tr>
<td><strong>Input parameter(in)</strong></td>
</tr>
<tr>
<td>usart_periph</td>
</tr>
<tr>
<td>USARTx</td>
</tr>
<tr>
<td><strong>Output parameter(out)</strong></td>
</tr>
<tr>
<td><strong>Return value</strong></td>
</tr>
</tbody>
</table>

Example:

/* enable USART0 NACK in smartcard mode */
usart_smartcard_mode_nack_enable(USART0);

**usart_smartcard_mode_nack_disable**

The description of **usart_smartcard_mode_nack_disable** is shown as below:

<table>
<thead>
<tr>
<th>Table 3-818. Function usart_smartcard_mode_nack_disable</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Function name</strong></td>
</tr>
<tr>
<td><strong>Function prototype</strong></td>
</tr>
<tr>
<td><strong>Function descriptions</strong></td>
</tr>
<tr>
<td><strong>Precondition</strong></td>
</tr>
<tr>
<td><strong>The called functions</strong></td>
</tr>
<tr>
<td><strong>Input parameter(in)</strong></td>
</tr>
<tr>
<td>usart_periph</td>
</tr>
<tr>
<td>USARTx</td>
</tr>
<tr>
<td><strong>Output parameter(out)</strong></td>
</tr>
<tr>
<td><strong>Return value</strong></td>
</tr>
</tbody>
</table>
Example:

/* disable USART0 NACK in smartcard mode */
usart_smartcard_mode_nack_disable(USART0);

**usart_smartcard_mode_early_nack_enable**

The description of usart_smartcard_mode_early_nack_enable is shown as below:

**Table 3-819. Function usart_smartcard_mode_early_nack_enable**

<table>
<thead>
<tr>
<th>Function name</th>
<th>usart_smartcard_mode_early_nack_enable</th>
</tr>
</thead>
<tbody>
<tr>
<td>Function prototype</td>
<td>void usart_smartcard_mode_early_nack_enable(uint32_t usart_periph);</td>
</tr>
<tr>
<td>Function descriptions</td>
<td>enable early NACK in smartcard mode</td>
</tr>
<tr>
<td>Precondition</td>
<td>-</td>
</tr>
<tr>
<td>The called functions</td>
<td>-</td>
</tr>
<tr>
<td>Input parameter(in)</td>
<td>-</td>
</tr>
<tr>
<td>usart_periph</td>
<td>usart peripheral</td>
</tr>
<tr>
<td>USARTx x=0,1</td>
<td>-</td>
</tr>
<tr>
<td>Output parameter(out)</td>
<td>-</td>
</tr>
<tr>
<td>Return value</td>
<td>-</td>
</tr>
</tbody>
</table>

Example:

/* enable USART0 early NACK in smartcard mode */
usart_smartcard_mode_early_nack_enable(USART0);

**usart_smartcard_mode_early_nack_disable**

The description of usart_smartcard_mode_early_nack_disable is shown as below:

**Table 3-820. Function usart_smartcard_mode_early_nack_disable**

<table>
<thead>
<tr>
<th>Function name</th>
<th>usart_smartcard_mode_early_nack_disable</th>
</tr>
</thead>
<tbody>
<tr>
<td>Function prototype</td>
<td>void usart_smartcard_mode_early_nack_disable(uint32_t usart_periph);</td>
</tr>
<tr>
<td>Function descriptions</td>
<td>disable early NACK in smartcard mode</td>
</tr>
<tr>
<td>Precondition</td>
<td>-</td>
</tr>
<tr>
<td>The called functions</td>
<td>-</td>
</tr>
<tr>
<td>Input parameter(in)</td>
<td>-</td>
</tr>
<tr>
<td>usart_periph</td>
<td>usart peripheral</td>
</tr>
<tr>
<td>USARTx x=0,1</td>
<td>-</td>
</tr>
<tr>
<td>Output parameter(out)</td>
<td>-</td>
</tr>
<tr>
<td>Return value</td>
<td>-</td>
</tr>
</tbody>
</table>
Example:

/* disable USART0 early NACK in smartcard mode */
usart_smartcard_mode_early_nack_disable(USART0);

**usart_smartcard_autoretry_config**

The description of **usart_smartcard_autoretry_config** is shown as below:

<table>
<thead>
<tr>
<th>Table 3-821. Function <strong>usart_smartcard_autoretry_config</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Function name</strong></td>
</tr>
<tr>
<td><strong>Function prototype</strong></td>
</tr>
<tr>
<td><strong>Function descriptions</strong></td>
</tr>
<tr>
<td><strong>Precondition</strong></td>
</tr>
<tr>
<td><strong>The called functions</strong></td>
</tr>
<tr>
<td><strong>Input parameter(in)</strong></td>
</tr>
<tr>
<td>usart_periph</td>
</tr>
<tr>
<td>USARTx</td>
</tr>
<tr>
<td><strong>Input parameter(in)</strong></td>
</tr>
<tr>
<td>scrtnum</td>
</tr>
<tr>
<td><strong>Output parameter(out)</strong></td>
</tr>
<tr>
<td><strong>Return value</strong></td>
</tr>
</tbody>
</table>

Example:

/* configure smartcard auto-retry number */
usart_smartcard_autoretry_config(USART0, 0x00000007);

**usart_block_length_config**

The description of **usart_block_length_config** is shown as below:

<table>
<thead>
<tr>
<th>Table 3-822. Function <strong>usart_block_length_config</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Function name</strong></td>
</tr>
<tr>
<td><strong>Function prototype</strong></td>
</tr>
<tr>
<td><strong>Function descriptions</strong></td>
</tr>
<tr>
<td><strong>Precondition</strong></td>
</tr>
<tr>
<td><strong>The called functions</strong></td>
</tr>
<tr>
<td><strong>Input parameter(in)</strong></td>
</tr>
<tr>
<td>usart_periph</td>
</tr>
<tr>
<td>USARTx</td>
</tr>
<tr>
<td><strong>Input parameter(in)</strong></td>
</tr>
</tbody>
</table>
GD32L23x Firmware Library User Guide

<table>
<thead>
<tr>
<th>bl</th>
<th>block length(0x00-0x000000FF)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Output parameter(out)</td>
<td>-</td>
</tr>
<tr>
<td>Return value</td>
<td>-</td>
</tr>
</tbody>
</table>

Example:

/* configure block length in Smartcard T=1 reception */

usart_block_length_config(USART0, 0x000000FF);

**usart_irda_mode_enable**

The description of usart_irda_mode_enable is shown as below:

**Table 3-823. Function usart_irda_mode_enable**

<table>
<thead>
<tr>
<th>Function name</th>
<th>usart_irda_mode_enable</th>
</tr>
</thead>
<tbody>
<tr>
<td>Function prototype</td>
<td>void usart_irda_mode_enable(uint32_t usart_periph);</td>
</tr>
<tr>
<td>Function descriptions</td>
<td>enable IrDA mode</td>
</tr>
<tr>
<td>Precondition</td>
<td>-</td>
</tr>
<tr>
<td>The called functions</td>
<td>-</td>
</tr>
<tr>
<td>Input parameter(in)</td>
<td></td>
</tr>
<tr>
<td>usart_periph</td>
<td>usart peripheral</td>
</tr>
<tr>
<td>USARTx</td>
<td>x=0,1</td>
</tr>
<tr>
<td>Output parameter(out)</td>
<td>-</td>
</tr>
<tr>
<td>Return value</td>
<td>-</td>
</tr>
</tbody>
</table>

Example:

/* enable USART0 IrDA mode */

usart_irda_mode_enable(USART0);

**usart_irda_mode_disable**

The description of usart_irda_mode_disable is shown as below:

**Table 3-824. Function usart_irda_mode_disable**

<table>
<thead>
<tr>
<th>Function name</th>
<th>usart_irda_mode_disable</th>
</tr>
</thead>
<tbody>
<tr>
<td>Function prototype</td>
<td>void usart_irda_mode_disable(uint32_t usart_periph);</td>
</tr>
<tr>
<td>Function descriptions</td>
<td>disable IrDA mode</td>
</tr>
<tr>
<td>Precondition</td>
<td>-</td>
</tr>
<tr>
<td>The called functions</td>
<td>-</td>
</tr>
<tr>
<td>Input parameter(in)</td>
<td></td>
</tr>
</tbody>
</table>

Example:

/* enable USART0 IrDA mode */

usart_irda_mode_disable(USART0);
GD32L23x Firmware Library User Guide

<table>
<thead>
<tr>
<th>usart_periph</th>
<th>usart peripheral</th>
</tr>
</thead>
<tbody>
<tr>
<td>USARTx</td>
<td>x=0,1</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Output parameter(out)</th>
</tr>
</thead>
<tbody>
<tr>
<td>-</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Return value</th>
</tr>
</thead>
<tbody>
<tr>
<td>-</td>
</tr>
</tbody>
</table>

Example:

```c
#ifdef USART0
void usart_irda_lowpower_config(void) { /* enable USART0 IrDA low-power mode */
  usart_irda_lowpower_config(USART0, 0x00);
}
#endif
```

### USART Prescaler Config

The description of `usart_prescaler_config` is shown as below:

**Table 3-825. Function `usart_prescaler_config`**

<table>
<thead>
<tr>
<th>Function name</th>
<th><code>usart_prescaler_config</code></th>
</tr>
</thead>
<tbody>
<tr>
<td>Function prototype</td>
<td>void usart_prescaler_config(uint32_t usart_periph, uint8_t psc);</td>
</tr>
<tr>
<td>Function descriptions</td>
<td>configure the peripheral clock prescaler</td>
</tr>
<tr>
<td>Precondition</td>
<td>-</td>
</tr>
<tr>
<td>The called functions</td>
<td>-</td>
</tr>
</tbody>
</table>

### USART IrdA Lowpower Config

The description of `usart_irda_lowpower_config` is shown as below:

**Table 3-826. Function `usart_irda_lowpower_config`**

<table>
<thead>
<tr>
<th>Function name</th>
<th><code>usart_irda_lowpower_config</code></th>
</tr>
</thead>
<tbody>
<tr>
<td>Function prototype</td>
<td>void usart_irda_lowpower_config(uint32_t usart_periph, uint32_t irlp);</td>
</tr>
<tr>
<td>Function descriptions</td>
<td>configure IrDA low-power</td>
</tr>
<tr>
<td>Precondition</td>
<td>-</td>
</tr>
<tr>
<td>The called functions</td>
<td>-</td>
</tr>
</tbody>
</table>

Example:

```c
#ifdef USART0
void usart_irda_lowpower_config(void) { /* enable USART0 IrDA low-power mode */
  usart_irda_lowpower_config(USART0, 0x00);
}
#endif
```
### Recondition

The called functions

**Input parameter (in)**

<table>
<thead>
<tr>
<th><code>usart_periph</code></th>
<th><code>usart_peripheral</code></th>
</tr>
</thead>
<tbody>
<tr>
<td><code>USARTx</code></td>
<td><code>x=0,1</code></td>
</tr>
</tbody>
</table>

**Input parameter (in)**

<table>
<thead>
<tr>
<th><code>irlp</code></th>
<th>IrDA low-power or normal</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>USART_IRLP_LOW</code></td>
<td>low-power</td>
</tr>
<tr>
<td><code>USART_IRLP_NORMAL</code></td>
<td>normal</td>
</tr>
</tbody>
</table>

**Output parameter (out)**

-

**Return value**

- Example:

```c
/* configure USART0 IrDA low-power */
usart_irda_lowpower_config(USART0, USART_IRLP_LOW);
```

### `usart_hardware_flow_rts_config`

The description of `usart_hardware_flow_rts_config` is shown as below:

**Table 3-827. Function `usart_hardware_flow_rts_config`**

<table>
<thead>
<tr>
<th>Function name</th>
<th><code>usart_hardware_flow_rts_config</code></th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Function prototype</strong></td>
<td><code>void usart_hardware_flow_rts_config(uint32_t usart_periph, uint32_t rtsconfig);</code></td>
</tr>
<tr>
<td><strong>Function descriptions</strong></td>
<td>configure hardware flow control RTS</td>
</tr>
<tr>
<td><strong>Precondition</strong></td>
<td>-</td>
</tr>
<tr>
<td><strong>The called functions</strong></td>
<td>-</td>
</tr>
</tbody>
</table>

**Input parameter (in)**

<table>
<thead>
<tr>
<th><code>usart_periph</code></th>
<th><code>usart_peripheral</code></th>
</tr>
</thead>
<tbody>
<tr>
<td><code>USARTx</code></td>
<td><code>x=0,1</code></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th><code>rtsconfig</code></th>
<th>enable or disable RTS</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>USART_RTS_ENABLE</code></td>
<td>enable RTS</td>
</tr>
<tr>
<td><code>USART_RTS_DISABLE</code></td>
<td>disable RTS</td>
</tr>
</tbody>
</table>

**Output parameter (out)**

- `-`

**Return value**

- `-`
/* configure USART0 hardware flow control RTS */

usart_hardware_flow_rts_config(USART0, USART_RTS_ENABLE);

**usart_hardware_flow_cts_config**

The description of **usart_hardware_flow_cts_config** is shown as below:

**Table 3-828. Function **usart_hardware_flow_cts_config**

<table>
<thead>
<tr>
<th>Function name</th>
<th>usart_hardware_flow_cts_config</th>
</tr>
</thead>
<tbody>
<tr>
<td>Function prototype</td>
<td>void usart_hardware_flow_cts_config(uint32_t usart_periph, uint32_t ctsconfig);</td>
</tr>
<tr>
<td>Function descriptions</td>
<td>configure hardware flow control CTS</td>
</tr>
<tr>
<td>Precondition</td>
<td>-</td>
</tr>
<tr>
<td>The called functions</td>
<td>-</td>
</tr>
<tr>
<td>Input parameter <strong>(in)</strong></td>
<td></td>
</tr>
<tr>
<td><strong>usart_periph</strong></td>
<td>usart peripheral</td>
</tr>
<tr>
<td><strong>USARTx</strong></td>
<td>x=0,1</td>
</tr>
<tr>
<td>Input parameter <strong>(in)</strong></td>
<td></td>
</tr>
<tr>
<td><strong>ctsconfig</strong></td>
<td>enable or disable CTS</td>
</tr>
<tr>
<td><strong>USART_CTS_ENABLE</strong></td>
<td>enable CTS</td>
</tr>
<tr>
<td><strong>USART_CTS_DISABLE</strong></td>
<td>disable CTS</td>
</tr>
<tr>
<td>Output parameter <strong>(out)</strong></td>
<td></td>
</tr>
<tr>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>Return value</td>
<td>-</td>
</tr>
</tbody>
</table>

Example:

/* configure USART0 hardware flow control CTS */

usart_hardware_flow_cts_config(USART0, USART_CTS_ENABLE);

**usart_hardware_flow_coherence_config**

The description of **usart_hardware_flow_coherence_config** is shown as below:

**Table 3-829. Function **usart_hardware_flow_coherence_config**

<table>
<thead>
<tr>
<th>Function name</th>
<th>usart_hardware_flow_coherence_config</th>
</tr>
</thead>
<tbody>
<tr>
<td>Function prototype</td>
<td>void usart_hardware_flow_coherence_config(uint32_t usart_periph, uint32_t hcm);</td>
</tr>
<tr>
<td>Function descriptions</td>
<td>configure hardware flow control coherence mode</td>
</tr>
<tr>
<td>Precondition</td>
<td>-</td>
</tr>
<tr>
<td>The called functions</td>
<td>-</td>
</tr>
<tr>
<td>Input parameter <strong>(in)</strong></td>
<td></td>
</tr>
<tr>
<td><strong>usart_periph</strong></td>
<td>usart peripheral</td>
</tr>
</tbody>
</table>
**USARTx**  
\( x=0,1 \)

<table>
<thead>
<tr>
<th>Input parameter(in)</th>
<th>Output parameter(out)</th>
<th>Return value</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>hcm</strong></td>
<td><strong>-</strong></td>
<td><strong>-</strong></td>
</tr>
</tbody>
</table>

**hcm**

- Hardware flow control coherence mode
- nRTS signal equals to the rxne status register
- nRTS signal is set when the last data bit has been sampled

**Example:**

```c
/* configure hardware flow control coherence mode */
usart_hardware_flow_coherence_config(USART0, USART_HCM_NONE);
```

### usart_rs485_driver_enable

The description of `usart_rs485_driver_enable` is shown as below:

**Table 3-830. Function `usart_rs485_driver_enable`**

<table>
<thead>
<tr>
<th>Function name</th>
<th>Function prototype</th>
<th>Function descriptions</th>
<th>Precondition</th>
<th>The called functions</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>usart_rs485_driver_enable</code></td>
<td><code>void usart_rs485_driver_enable(uint32_t usart_periph);</code></td>
<td>enable USART RS485 driver</td>
<td><code>-</code></td>
<td><code>-</code></td>
</tr>
</tbody>
</table>

**Input parameter(in)**

- `usart_periph`  
- `usart_periph`: usart peripheral
- `USARTx`: \( x=0,1 \)

**Output parameter(out)**

- `-`

**Return value**

- `-`

**Example:**

```c
/* enable USART0 RS485 driver */
usart_rs485_driver_enable(USART0);
```

### usart_rs485_driver_disable

The description of `usart_rs485_driver_disable` is shown as below:

**Table 3-831. Function `usart_rs485_driver_disable`**

<table>
<thead>
<tr>
<th>Function name</th>
<th>Function prototype</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>usart_rs485_driver_disable</code></td>
<td><code>void usart_rs485_driver_disable(uint32_t usart_periph);</code></td>
</tr>
</tbody>
</table>
GD32L23x Firmware Library User Guide

Disable USART

Precondition:

The called functions:

Input parameter (in):
- `usart_periph` (usart peripheral)
- `USARTx` (x=0,1)

Output parameter (out):
- 

Return value:
- 

Example:

`/* disable USART0 RS485 driver */
usart_rs485_driver_disable(USART0);

**usart_driver_assertime_config**

The description of `usart_driver_assertime_config` is shown as below:

<table>
<thead>
<tr>
<th>Function name</th>
<th>Function prototype</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>usart_driver_assertime_config</code></td>
<td><code>void usart_driver_assertime_config(uint32_t usart_periph, uint32_t deatime);</code></td>
</tr>
</tbody>
</table>

**Function descriptions**

Configure driver enable assertion time

**Precondition**

**The called functions**

**Input parameter (in):**
- `usart_periph` (usart peripheral)
- `USARTx` (x=0,1)

**Output parameter (out):**
- 

**Return value:**
- 

Example:

`/* set USART0 driver assertime */
usart_driver_assertime_config(USART0, 0x0000001F);

**usart_driver_deassertime_config**

The description of `usart_driver_deassertime_config` is shown as below:
Function name: usart_driver_deassertime_config

Function prototype:
```c
void usart_driver_deassertime_config(uint32_t usart_periph, uint32_t dedtime);
```

Function descriptions:
Configure driver enable de-assertion time.

Precondition:
- 

The called functions:
- 

Input parameter (in):
- `usart_periph` - USART peripheral
  - `USARTx` - x=0,1
- `dedtime` - Driver enable de-assertion time (0x00-0x0000001F)

Output parameter (out):
- 

Return value:
- 

Example:
```c
/* set USART0 driver deasserttime */
usart_driver_deasserttime_config(USART0, 0x0000001F);
```

The description of `usart_depolarity_config` is shown as below:

Function name: usart_depolarity_config

Function prototype:
```c
void usart_depolarity_config(uint32_t usart_periph, uint32_t dep);
```

Function descriptions:
Configure driver enable polarity mode.

Precondition:
- 

The called functions:
- 

Input parameter (in):
- `usart_periph` - USART peripheral
  - `USARTx` - x=0,1
- `dep` - DE signal
  - `USART_DEP_HIGH` - DE signal is active high
  - `USART_DEP_LOW` - DE signal is active low

Output parameter (out):
- 

Return value:
- 

Example:
/* configure driver enable polarity mode */

usart_depolarity_config(USART0, USART_DEP_HIGH);

**usart_dma_receive_config**

The description of `usart_dma_receive_config` is shown as below:

**Table 3-835. Function usart_dma_receive_config**

<table>
<thead>
<tr>
<th>Function name</th>
<th>usart_dma_receive_config</th>
</tr>
</thead>
<tbody>
<tr>
<td>Function prototype</td>
<td>void usart_dma_receive_config(uint32_t usart_periph, uint32_t dmacmd);</td>
</tr>
<tr>
<td>Function descriptions</td>
<td>configure USART DMA for reception</td>
</tr>
<tr>
<td>Precondition</td>
<td>-</td>
</tr>
<tr>
<td>The called functions</td>
<td>-</td>
</tr>
</tbody>
</table>

**Input parameter(in)**

<table>
<thead>
<tr>
<th>USART_periph</th>
<th>外设USARTx/UARTx</th>
</tr>
</thead>
<tbody>
<tr>
<td>USARTx</td>
<td>x=0,1</td>
</tr>
<tr>
<td>UARTx</td>
<td>x=3,4</td>
</tr>
</tbody>
</table>

**Input parameter(in)**

<table>
<thead>
<tr>
<th>dmacmd</th>
<th>enable or disable DMA for reception</th>
</tr>
</thead>
<tbody>
<tr>
<td>USART_DENR_ENABLE</td>
<td>DMA enable for reception</td>
</tr>
<tr>
<td>USART_DENR_DISABLE</td>
<td>DMA disable for reception</td>
</tr>
</tbody>
</table>

**Output parameter(out)**

| -             | -                                 |

**Return value**

| -             | -                                 |

Example:

/* USART0 DMA enable for reception */

usart_dma_receive_config(USART0, USART_DENR_ENABLE);

**usart_dma_transmit_config**

The description of `usart_dma_transmit_config` is shown as below:

**Table 3-836. Function usart_dma_transmit_config**

<table>
<thead>
<tr>
<th>Function name</th>
<th>usart_dma_transmit_config</th>
</tr>
</thead>
<tbody>
<tr>
<td>Function prototype</td>
<td>void usart_dma_transmit_config(uint32_t usart_periph, uint32_t dmacmd);</td>
</tr>
<tr>
<td>Function descriptions</td>
<td>configure USART DMA for transmission</td>
</tr>
<tr>
<td>Precondition</td>
<td>-</td>
</tr>
<tr>
<td>The called functions</td>
<td>-</td>
</tr>
</tbody>
</table>

**Input parameter(in)**
**usart_periph** | 外设USARTx/UARTx
---|---
**USARTx** | x=0,1
**UARTx** | x=3,4

**Input parameter**(in)

<table>
<thead>
<tr>
<th>dmacmd</th>
<th>enable or disable DMA for transmission</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>USART_DENT_ENABLE</strong></td>
<td>DMA enable for transmission</td>
</tr>
<tr>
<td><strong>USART_DENT_DISABLE</strong></td>
<td>DMA disable for transmission</td>
</tr>
</tbody>
</table>

**Output parameter**(out)

- ...

**Return value**

- ...

Example:

//* USART0 DMA enable for transmission */

usart_dma_transmit_config(USART0, USART_DENT_ENABLE);

**usart_reception_error_dma_disable**

The description of **usart_reception_error_dma_disable** is shown as below:

**Table 3-837. Function usart_reception_error_dma_disable**

<table>
<thead>
<tr>
<th>Function name</th>
<th>usart_reception_error_dma_disable</th>
</tr>
</thead>
<tbody>
<tr>
<td>Function prototype</td>
<td>void usart_reception_error_dma_disable(uint32_t usart_periph);</td>
</tr>
<tr>
<td>Function descriptions</td>
<td>disable DMA on reception error</td>
</tr>
<tr>
<td>Precondition</td>
<td>-</td>
</tr>
<tr>
<td>The called functions</td>
<td>-</td>
</tr>
</tbody>
</table>

**Input parameter**(in)

<table>
<thead>
<tr>
<th>usart_periph</th>
<th>外设USARTx/UARTx</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>USARTx</strong></td>
<td>x=0,1</td>
</tr>
<tr>
<td><strong>UARTx</strong></td>
<td>x=3,4</td>
</tr>
</tbody>
</table>

**Output parameter**(out)

- ...

**Return value**

- ...

Example:

//* disable DMA on reception error */

usart_reception_error_dma_disable(USART0);
**GigaDevice GD32L23x Firmware Library User Guide**

**usart_reception_error_dma_enable**

The description of `usart_reception_error_dma_enable` is shown as below:

<table>
<thead>
<tr>
<th>Function name</th>
<th>usart_reception_error_dma_enable</th>
</tr>
</thead>
<tbody>
<tr>
<td>Function prototype</td>
<td><code>void usart_reception_error_dma_enable(uint32_t usart_periph);</code></td>
</tr>
<tr>
<td>Function descriptions</td>
<td>enable DMA on reception error</td>
</tr>
<tr>
<td>Precondition</td>
<td>-</td>
</tr>
<tr>
<td>The called functions</td>
<td>-</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Input parameter(in)</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td><code>usart_periph</code></td>
<td>设备USARTx/UARTx</td>
</tr>
<tr>
<td><code>USARTx</code></td>
<td>x=0,1</td>
</tr>
<tr>
<td><code>UARTx</code></td>
<td>x=3,4</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Output parameter(out)</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>-</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Return value</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>-</td>
</tr>
</tbody>
</table>

Example:

```c
/* enable DMA on reception error */
usart_reception_error_dma_enable(USART0);
```

**usart_wakeup_enable**

The description of `usart_wakeup_enable` is shown as below:

<table>
<thead>
<tr>
<th>Function name</th>
<th>usart_wakeup_enable</th>
</tr>
</thead>
<tbody>
<tr>
<td>Function prototype</td>
<td><code>void usart_wakeup_enable(uint32_t usart_periph);</code></td>
</tr>
<tr>
<td>Function descriptions</td>
<td>enable USART to wakeup the mcu from deep-sleep mode</td>
</tr>
<tr>
<td>Precondition</td>
<td>-</td>
</tr>
<tr>
<td>The called functions</td>
<td>-</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Input parameter(in)</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td><code>usart_periph</code></td>
<td><code>usart_peripheral</code></td>
</tr>
<tr>
<td><code>USARTx</code></td>
<td>x=0,1</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Output parameter(out)</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>-</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Return value</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>-</td>
</tr>
</tbody>
</table>

Example:

```c
/* USART0 wake up enable */
```
GD32L23x Firmware Library User Guide

usart_wakeup_enable(USART0);

usart_wakeup_disable

The description of usart_wakeup_disable is shown as below:

<table>
<thead>
<tr>
<th>Function name</th>
<th>usart_wakeup_disable</th>
</tr>
</thead>
<tbody>
<tr>
<td>Function prototype</td>
<td>void usart_wakeup_disable(uint32_t usart_periph);</td>
</tr>
<tr>
<td>Function descriptions</td>
<td>disable USART to wakeup the mcu from deep-sleep mode</td>
</tr>
<tr>
<td>Precondition</td>
<td>-</td>
</tr>
<tr>
<td>The called functions</td>
<td>-</td>
</tr>
</tbody>
</table>

Input parameter(in)

<table>
<thead>
<tr>
<th>usart_periph</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>USARTx</td>
<td>x=0,1</td>
</tr>
</tbody>
</table>

Output parameter(out)

| -          |

Return value

| -          |

Example:

/* USART0 wake up disable */
usart_wakeup_disable(USART0);

usart_wakeup_mode_config

The description of usart_wakeup_mode_config is shown as below:

<table>
<thead>
<tr>
<th>Function name</th>
<th>usart_wakeup_mode_config</th>
</tr>
</thead>
<tbody>
<tr>
<td>Function prototype</td>
<td>void usart_wakeup_mode_config(uint32_t usart_periph, uint32_t wum);</td>
</tr>
<tr>
<td>Function descriptions</td>
<td>configure the USART wakeup mode from deep-sleep mode</td>
</tr>
<tr>
<td>Precondition</td>
<td>-</td>
</tr>
<tr>
<td>The called functions</td>
<td>-</td>
</tr>
</tbody>
</table>

Input parameter(in)

<table>
<thead>
<tr>
<th>usart_periph</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>USARTx</td>
<td>x=0,1</td>
</tr>
</tbody>
</table>

Input parameter(in)

<table>
<thead>
<tr>
<th>wum</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>USART_WUM_ADDR</td>
<td>WUF active on address match</td>
</tr>
<tr>
<td>USART_WUM_START_B</td>
<td>WUF active on start bit</td>
</tr>
<tr>
<td>USART_WUM_RBNE</td>
<td>WUF active on RBNE</td>
</tr>
</tbody>
</table>

Output parameter(out)

| -          |
Example:

```c
/* configure USART0 wake up mode */
usart_wakeup_mode_config(USART0, USART_WUM_ADDR);
```

**usart_receive_fifo_enable**

The description of usart_receive_fifo_enable is shown as below:

<table>
<thead>
<tr>
<th>Function name</th>
<th>usart_receive_fifo_enable</th>
</tr>
</thead>
<tbody>
<tr>
<td>Function prototype</td>
<td>void usart_receive_fifo_enable(uint32_t usart_periph);</td>
</tr>
<tr>
<td>Function descriptions</td>
<td>enable receive FIFO</td>
</tr>
<tr>
<td>Precondition</td>
<td>-</td>
</tr>
<tr>
<td>The called functions</td>
<td>-</td>
</tr>
</tbody>
</table>

**Input parameter[in]**

<table>
<thead>
<tr>
<th>usart_periph</th>
<th>外设USARTx/UARTx</th>
</tr>
</thead>
<tbody>
<tr>
<td>USARTx</td>
<td>x=0,1</td>
</tr>
<tr>
<td>UARTx</td>
<td>x=3,4</td>
</tr>
</tbody>
</table>

**Output parameter[out]**

| - | - |

**Return value**

Example:

```c
/* enable receive FIFO */
usart_receive_fifo_enable(USART0);
```

**usart_receive_fifo_disable**

The description of usart_receive_fifo_disable is shown as below:

<table>
<thead>
<tr>
<th>Function name</th>
<th>usart_receive_fifo_disable</th>
</tr>
</thead>
<tbody>
<tr>
<td>Function prototype</td>
<td>void usart_receive_fifo_disable(uint32_t usart_periph);</td>
</tr>
<tr>
<td>Function descriptions</td>
<td>disable receive FIFO</td>
</tr>
<tr>
<td>Precondition</td>
<td>-</td>
</tr>
<tr>
<td>The called functions</td>
<td>-</td>
</tr>
</tbody>
</table>

**Input parameter[in]**

| usart_periph | 外设USARTx/UARTx |

---

547
Example:

/* disable receive FIFO */

usart_receive_fifo_disable(USART0);

**usart_receive_fifo_counter_number**

The description of **usart_receive_fifo_counter_number** is shown as below:

<table>
<thead>
<tr>
<th>Function name</th>
<th>usart_receive_fifo_counter_number</th>
</tr>
</thead>
<tbody>
<tr>
<td>Function prototype</td>
<td>uint8_t usart_receive_fifo_counter_number(uint32_t usart_periph);</td>
</tr>
<tr>
<td>Function descriptions</td>
<td>read receive FIFO counter number</td>
</tr>
<tr>
<td>Precondition</td>
<td>-</td>
</tr>
<tr>
<td>The called functions</td>
<td>-</td>
</tr>
</tbody>
</table>

**Input parameter (in)**

<table>
<thead>
<tr>
<th>usart_periph</th>
<th>外设USARTx/UARTx</th>
</tr>
</thead>
<tbody>
<tr>
<td>USARTx</td>
<td>x=0,1</td>
</tr>
<tr>
<td>UARTx</td>
<td>x=3,4</td>
</tr>
</tbody>
</table>

**Output parameter (out)**

- -

**Return value**

| uint8_t | receive FIFO counter number |

Example:

/* read receive FIFO counter number */

uint8_t temp;

temp = usart_receive_fifo_counter_number(USART0);

**usart_flag_get**

The description of **usart_flag_get** is shown as below:

<table>
<thead>
<tr>
<th>Function name</th>
<th>usart_flag_get</th>
</tr>
</thead>
<tbody>
<tr>
<td>Function prototype</td>
<td>FlagStatus usart_flag_get(uint32_t usart_periph, usart_flag_enum flag);</td>
</tr>
<tr>
<td>Function descriptions</td>
<td>get flag in STAT/RFCS register</td>
</tr>
</tbody>
</table>
### Precondition
- 

### The called functions
- 

### Input parameter(in)

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>usart_periph</code></td>
<td>外设USARTx/UARTx</td>
</tr>
<tr>
<td><code>USARTx</code> x=0,1</td>
<td></td>
</tr>
<tr>
<td><code>UARTx</code> x=3,4</td>
<td></td>
</tr>
</tbody>
</table>

### Input parameter(in)

<table>
<thead>
<tr>
<th>Flag</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>USART_FLAG_PERR</code></td>
<td>parity error flag</td>
</tr>
<tr>
<td><code>USART_FLAG_FERR</code></td>
<td>frame error flag</td>
</tr>
<tr>
<td><code>USART_FLAG_NERR</code></td>
<td>noise error flag</td>
</tr>
<tr>
<td><code>USART_FLAG_OVERR</code></td>
<td>overrun error</td>
</tr>
<tr>
<td><code>USART_FLAG_IDLE</code></td>
<td>idle line detected flag</td>
</tr>
<tr>
<td><code>USART_FLAG_RBNE</code></td>
<td>read data buffer not empty</td>
</tr>
<tr>
<td><code>USART_FLAG_TC</code></td>
<td>transmission completed</td>
</tr>
<tr>
<td><code>USART_FLAG_TBE</code></td>
<td>transmit data register empty</td>
</tr>
<tr>
<td><code>USART_FLAG_LBD</code></td>
<td>LIN break detected flag</td>
</tr>
<tr>
<td><code>USART_FLAG_CTSF</code></td>
<td>CTS change flag</td>
</tr>
<tr>
<td><code>USART_FLAG_CTS</code></td>
<td>CTS level</td>
</tr>
<tr>
<td><code>USART_FLAG_RT</code></td>
<td>receiver timeout flag</td>
</tr>
<tr>
<td><code>USART_FLAG_EB</code></td>
<td>end of block flag</td>
</tr>
<tr>
<td><code>USART_FLAG_ABDE</code></td>
<td>auto baudrate detection error</td>
</tr>
<tr>
<td><code>USART_FLAG_ABD</code></td>
<td>auto baudrate detection flag</td>
</tr>
<tr>
<td><code>USART_FLAG_BSY</code></td>
<td>busy flag</td>
</tr>
<tr>
<td><code>USART_FLAG_AM</code></td>
<td>address match flag</td>
</tr>
<tr>
<td><code>USART_FLAG_SB</code></td>
<td>send break flag</td>
</tr>
<tr>
<td><code>USART_FLAG_RWU</code></td>
<td>receiver wakeup from mute mode</td>
</tr>
<tr>
<td><code>USART_FLAG_WU</code></td>
<td>wakeup from deep-sleep mode flag</td>
</tr>
<tr>
<td><code>USART_FLAG_TEA</code></td>
<td>transmit enable acknowledge flag</td>
</tr>
<tr>
<td><code>USART_FLAG_REA</code></td>
<td>receive enable acknowledge flag</td>
</tr>
<tr>
<td><code>USART_FLAG_EPERR</code></td>
<td>early parity error flag</td>
</tr>
<tr>
<td><code>USART_FLAG_RFE</code></td>
<td>receive FIFO empty flag</td>
</tr>
<tr>
<td><code>USART_FLAG_RFF</code></td>
<td>receive FIFO full flag</td>
</tr>
<tr>
<td><code>USART_FLAG_RFFINT</code></td>
<td>receive FIFO full interrupt flag</td>
</tr>
</tbody>
</table>

### Output parameter(out)
- 

### Return value

| FlagStatus          | SET or RESET                                    |

Example:
/* get flag USART0 state */

FlagStatus status;

status = usart_flag_get(USART0, USART_FLAG_TBE);

**usart_flag_clear**

The description of `usart_flag_clear` is shown as below:

**Table 3-846. Function usart_flag_clear**

<table>
<thead>
<tr>
<th>Function name</th>
<th>usart_flag_clear</th>
</tr>
</thead>
<tbody>
<tr>
<td>Function prototype</td>
<td>void usart_flag_clear(uint32_t usart_periph, usart_flag_enum flag);</td>
</tr>
<tr>
<td>Function descriptions</td>
<td>clear flag in STAT register</td>
</tr>
<tr>
<td>Precondition</td>
<td>-</td>
</tr>
<tr>
<td>The called functions</td>
<td>-</td>
</tr>
</tbody>
</table>

**Input parameter**

<table>
<thead>
<tr>
<th>usart_periph</th>
<th>外设USARTx/UARTx</th>
</tr>
</thead>
<tbody>
<tr>
<td>USARTx</td>
<td>x=0,1</td>
</tr>
<tr>
<td>UARTx</td>
<td>x=3,4</td>
</tr>
</tbody>
</table>

**Input parameter**

<table>
<thead>
<tr>
<th>flag</th>
<th>USART flags, refer to Table 3-773. Enum usart_flag_enum</th>
</tr>
</thead>
<tbody>
<tr>
<td>USART_FLAG_PERR</td>
<td>parity error flag</td>
</tr>
<tr>
<td>USART_FLAG_FERR</td>
<td>frame error flag</td>
</tr>
<tr>
<td>USART_FLAG_NERR</td>
<td>noise detected flag</td>
</tr>
<tr>
<td>USART_FLAG_OERR</td>
<td>overrun error flag</td>
</tr>
<tr>
<td>USART_FLAG_IDLE</td>
<td>idle line detected flag</td>
</tr>
<tr>
<td>USART_FLAG_TC</td>
<td>transmission complete flag</td>
</tr>
<tr>
<td>USART_FLAG_LBD</td>
<td>LIN break detected flag</td>
</tr>
<tr>
<td>USART_FLAG_CTSF</td>
<td>CTS change flag</td>
</tr>
<tr>
<td>USART_FLAG_RT</td>
<td>receiver timeout flag</td>
</tr>
<tr>
<td>USART_FLAG_EB</td>
<td>end of block flag</td>
</tr>
<tr>
<td>USART_FLAG_AM</td>
<td>address match flag</td>
</tr>
<tr>
<td>USART_FLAG_WU</td>
<td>wakeup from deep-sleep mode flag</td>
</tr>
<tr>
<td>USART_FLAG_EPERR</td>
<td>early parity error flag</td>
</tr>
</tbody>
</table>

**Output parameter**

| - | |

**Return value**

| - | - |

Example:

/* clear USART0 flag */
usrart_flag_clear(USART0, USART_FLAG_TC);

**usrart_interrupt_enable**

The description of `usrart_interrupt_enable` is shown as below:

**Table 3-847. Function `usrart_interrupt_enable`**

<table>
<thead>
<tr>
<th>Function name</th>
<th><code>usrart_interrupt_enable</code></th>
</tr>
</thead>
<tbody>
<tr>
<td>Function prototype</td>
<td><code>void usrart_interrupt_enable(uint32_t usart_periph, usart_interrupt_enum interrupt);</code></td>
</tr>
<tr>
<td>Function descriptions</td>
<td>enable USART interrupt</td>
</tr>
<tr>
<td>Precondition</td>
<td>-</td>
</tr>
<tr>
<td>The called functions</td>
<td>-</td>
</tr>
</tbody>
</table>

**Input parameter(in)**

<table>
<thead>
<tr>
<th>usart_periph</th>
<th>外设USARTx/UARTx</th>
</tr>
</thead>
<tbody>
<tr>
<td>USARTx</td>
<td>x=0, 1</td>
</tr>
<tr>
<td>UARTx</td>
<td>x=3, 4</td>
</tr>
</tbody>
</table>

**Input parameter(in)**

<table>
<thead>
<tr>
<th>interrupt</th>
<th>interrupt type, refer to <a href="#">Table 3-775. Enum usart_interrupt_enum</a> only one among these parameters can be selected</th>
</tr>
</thead>
<tbody>
<tr>
<td>USART_INT_IDLE</td>
<td>idle interrupt</td>
</tr>
<tr>
<td>USART_INT_RBNE</td>
<td>read data buffer not empty interrupt and overrun error interrupt enable interrupt</td>
</tr>
<tr>
<td>USART_INT_TC</td>
<td>transmission complete interrupt</td>
</tr>
<tr>
<td>USART_INT_TBE</td>
<td>transmit data register empty interrupt</td>
</tr>
<tr>
<td>USART_INT_PERR</td>
<td>parity error interrupt</td>
</tr>
<tr>
<td>USART_INT_AM</td>
<td>address match interrupt</td>
</tr>
<tr>
<td>USART_INT_RT</td>
<td>receiver timeout interrupt</td>
</tr>
<tr>
<td>USART_INT_EB</td>
<td>end of block interrupt</td>
</tr>
<tr>
<td>USART_INT_LBD</td>
<td>LIN break detection interrupt</td>
</tr>
<tr>
<td>USART_INT_ERR</td>
<td>error interrupt enable in multibuffer communication</td>
</tr>
<tr>
<td>USART_INT_CTS</td>
<td>CTS interrupt</td>
</tr>
<tr>
<td>USART_INT_WU</td>
<td>wakeup from deep-sleep mode interrupt</td>
</tr>
<tr>
<td>USART_INT_RFF</td>
<td>receive FIFO full interrupt enable</td>
</tr>
</tbody>
</table>

**Output parameter(out)**

- |

**Return value**

- |

Example:

```c
/* enable USART0 TBE interrupt */

usrart_interrupt_enable(USART0, USART_INT_TBE);
```
### Table 3-848. Function `usart_interrupt_disable`

<table>
<thead>
<tr>
<th>Function name</th>
<th><code>usart_interrupt_disable</code></th>
</tr>
</thead>
<tbody>
<tr>
<td>Function prototype</td>
<td><code>void usart_interrupt_disable(uint32_t usart_periph, usart_interrupt_enum interrupt);</code></td>
</tr>
<tr>
<td>Function descriptions</td>
<td>disable USART interrupt</td>
</tr>
<tr>
<td>Precondition</td>
<td>-</td>
</tr>
<tr>
<td>The called functions</td>
<td>-</td>
</tr>
</tbody>
</table>

#### Input parameter(in)

<table>
<thead>
<tr>
<th><code>usart_periph</code></th>
<th>外设USARTx/UARTx</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>USARTx</code></td>
<td>x=0,1</td>
</tr>
<tr>
<td><code>UARTx</code></td>
<td>x=3,4</td>
</tr>
</tbody>
</table>

#### Input parameter(in)

<table>
<thead>
<tr>
<th><code>interrupt</code></th>
<th>interrupt type, refer to Table 3-775. Enum usart_interrupt_enum only one among these parameters can be selected</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>USART_INT_IDLE</code></td>
<td>idle interrupt</td>
</tr>
<tr>
<td><code>USART_INT_RBNE</code></td>
<td>read data buffer not empty interrupt and overrun error interrupt enable interrupt</td>
</tr>
<tr>
<td><code>USART_INT_TC</code></td>
<td>transmission complete interrupt</td>
</tr>
<tr>
<td><code>USART_INT_TBE</code></td>
<td>transmit data register empty interrupt</td>
</tr>
<tr>
<td><code>USART_INT_PERR</code></td>
<td>parity error interrupt</td>
</tr>
<tr>
<td><code>USART_INT_AM</code></td>
<td>address match interrupt</td>
</tr>
<tr>
<td><code>USART_INT_RT</code></td>
<td>receiver timeout interrupt</td>
</tr>
<tr>
<td><code>USART_INT_EB</code></td>
<td>end of block interrupt</td>
</tr>
<tr>
<td><code>USART_INT_LBD</code></td>
<td>LIN break detection interrupt</td>
</tr>
<tr>
<td><code>USART_INT_ERR</code></td>
<td>error interrupt enable in multibuffer communication</td>
</tr>
<tr>
<td><code>USART_INT_CTS</code></td>
<td>CTS interrupt</td>
</tr>
<tr>
<td><code>USART_INT_WU</code></td>
<td>wakeup from deep-sleep mode interrupt</td>
</tr>
<tr>
<td><code>USART_INT_RFF</code></td>
<td>receive FIFO full interrupt enable</td>
</tr>
</tbody>
</table>

#### Output parameter(out)

| - | - |

#### Return value

| - | - |

Example:

```c
/* disable USART0 TBE interrupt */

usart_interrupt_disable(USART0, USART_INT_TBE);
```
**usart_interrupt_flag_get**

The description of `usart_interrupt_flag_get` is shown as below:

**Table 3-849. Function usart_interrupt_flag_get**

<table>
<thead>
<tr>
<th>Function name</th>
<th>usart_interrupt_flag_get</th>
</tr>
</thead>
<tbody>
<tr>
<td>Function prototype</td>
<td>FlagStatus usart_interrupt_flag_get(uint32_t usart_periph, usart_interrupt_flag_enum int_flag);</td>
</tr>
<tr>
<td>Function descriptions</td>
<td>get USART interrupt and flag status</td>
</tr>
<tr>
<td>Precondition</td>
<td>-</td>
</tr>
<tr>
<td>The called functions</td>
<td>-</td>
</tr>
</tbody>
</table>

**Input parameter(in)**

<table>
<thead>
<tr>
<th></th>
<th>USART interrupt flag, refer to Table 3-774, <strong>Enum usart_interrupt_flag_enum</strong>, only one among these parameters can be selected</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>usart_periph</code></td>
<td>外设USARTx/UARTx</td>
</tr>
<tr>
<td><code>USARTx</code></td>
<td>x=0,1</td>
</tr>
<tr>
<td><code>UARTx</code></td>
<td>x=3,4</td>
</tr>
</tbody>
</table>

**Input parameter(in)**

<table>
<thead>
<tr>
<th>int_flag</th>
<th>end of block interrupt and flag</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>USART_INT_FLAG_EB</code></td>
<td>receiver timeout interrupt and flag</td>
</tr>
<tr>
<td><code>USART_INT_FLAG_AM</code></td>
<td>address match interrupt and flag</td>
</tr>
<tr>
<td><code>USART_INT_FLAG_PE</code></td>
<td>parity error interrupt and flag</td>
</tr>
<tr>
<td><code>USART_INT_FLAG_TB</code></td>
<td>transmitter buffer empty interrupt and flag</td>
</tr>
<tr>
<td><code>USART_INT_FLAG_TC</code></td>
<td>transmission complete interrupt and flag</td>
</tr>
<tr>
<td><code>USART_INT_FLAG_RBNE</code></td>
<td>read data buffer not empty interrupt and flag</td>
</tr>
<tr>
<td><code>USART_INT_FLAG_RBNE_ORERR</code></td>
<td>read data buffer not empty interrupt and overrun error flag</td>
</tr>
<tr>
<td><code>USART_INT_FLAG_IDLE</code></td>
<td>IDLE line detected interrupt and flag</td>
</tr>
<tr>
<td><code>USART_INT_FLAG_LB</code></td>
<td>LIN break detected interrupt and flag</td>
</tr>
<tr>
<td><code>USART_INT_FLAG_WU</code></td>
<td>wakeup from deep-sleep mode interrupt and flag</td>
</tr>
<tr>
<td><code>USART_INT_FLAG_CTSS</code></td>
<td>CTS interrupt and flag</td>
</tr>
<tr>
<td><code>USART_INT_FLAG_ERERR</code></td>
<td>error interrupt and noise error flag</td>
</tr>
<tr>
<td><code>USART_INT_FLAG_ER_NERR</code></td>
<td>error interrupt and overrun error</td>
</tr>
</tbody>
</table>

553
<table>
<thead>
<tr>
<th>Function</th>
<th>USART_INT_FLAG_RBNE</th>
</tr>
</thead>
<tbody>
<tr>
<td>.getDescription</td>
<td>error interrupt and frame error flag</td>
</tr>
<tr>
<td>USART_INT_FLAG_RF</td>
<td></td>
</tr>
<tr>
<td>Description</td>
<td>receive FIFO full interrupt and flag</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Output parameter(out)</th>
<th>-</th>
</tr>
</thead>
</table>

<table>
<thead>
<tr>
<th>Return value</th>
<th>-</th>
</tr>
</thead>
<tbody>
<tr>
<td>FlagStatus</td>
<td>SET or RESET</td>
</tr>
</tbody>
</table>

Example:

```c
/* get the USART0 interrupt flag status */
FlagStatus status;
status = usart_interrupt_flag_get(USART0, USART_INT_FLAG_RBNE);
```

**usart_interrupt_flag_clear**

The description of `usart_interrupt_flag_clear` is shown as below:

**Table 3-850. Function usart_interrupt_flag_clear**

<table>
<thead>
<tr>
<th>Function name</th>
<th>usart_interrupt_flag_clear</th>
</tr>
</thead>
<tbody>
<tr>
<td>Function prototype</td>
<td><code>void usart_interrupt_flag_clear(uint32_t usart_periph, usart_interrupt_flag_enum int_flag);</code></td>
</tr>
<tr>
<td>Function descriptions</td>
<td>clear USART interrupt flag</td>
</tr>
<tr>
<td>Precondition</td>
<td>-</td>
</tr>
<tr>
<td>The called functions</td>
<td>-</td>
</tr>
</tbody>
</table>

**Input parameter(in)**

<table>
<thead>
<tr>
<th>usart_periph</th>
<th>外设USARTx/UARTx</th>
</tr>
</thead>
<tbody>
<tr>
<td>USARTx</td>
<td>x=0,1</td>
</tr>
<tr>
<td>UARTx</td>
<td>x=3,4</td>
</tr>
</tbody>
</table>

**Input parameter(in)**

<table>
<thead>
<tr>
<th>int_flag</th>
<th>USART interrupt flag, refer to Table 3-774. Enum usart_interrupt_flag_enum, only one among these parameters can be selected</th>
</tr>
</thead>
<tbody>
<tr>
<td>USART_INT_FLAG_PE</td>
<td>parity error flag</td>
</tr>
<tr>
<td>USART_INT_FLAG_ER</td>
<td>frame error flag</td>
</tr>
<tr>
<td>USART_INT_FLAG_RERR</td>
<td>noise detected flag</td>
</tr>
<tr>
<td>USART_INT_FLAG_RBNE</td>
<td>read data buffer not empty interrupt and overrun error flag</td>
</tr>
<tr>
<td>USART_INT_FLAG_ER</td>
<td>error interrupt and overrun error</td>
</tr>
</tbody>
</table>

554
<table>
<thead>
<tr>
<th>R_ORERR</th>
<th>idle line detected flag</th>
</tr>
</thead>
<tbody>
<tr>
<td>USART_INT_FLAG_IDLE</td>
<td>transmission complete flag</td>
</tr>
<tr>
<td>USART_INT_FLAG_TC</td>
<td>LIN break detected flag</td>
</tr>
<tr>
<td>USART_INT_FLAG_LB_D</td>
<td>CTS change flag</td>
</tr>
<tr>
<td>USART_INT_FLAG_CT_S</td>
<td>receiver timeout flag</td>
</tr>
<tr>
<td>USART_INT_FLAG_RT</td>
<td>end of block flag</td>
</tr>
<tr>
<td>USART_INT_FLAG_AM</td>
<td>address match flag</td>
</tr>
<tr>
<td>USART_INT_FLAG_WU</td>
<td>wakeup from deep-sleep mode flag</td>
</tr>
<tr>
<td>USART_INT_FLAG_RF</td>
<td>receive FIFO full interrupt and flag</td>
</tr>
</tbody>
</table>

Output parameter(out)

```
-                   -
```

Return value

```
-                   -
```

Example:

```c
/* clear the USART0 interrupt flag */
usart_interrupt_flag_clear(USART0, USART_INT_FLAG_TC);
```

### 3.27. VREF

The precision internal reference is used to provide reference voltage for ADC/DAC, or used by off-chip circuit connecting to VREF pin. The VREF registers are listed in chapter 3.27.1, the VREF firmware functions are introduced in chapter 3.27.2.

#### 3.27.1. Descriptions of Peripheral registers

VREF registers are listed in the table shown as below:

<table>
<thead>
<tr>
<th>Registers</th>
<th>Descriptions</th>
</tr>
</thead>
<tbody>
<tr>
<td>VREF_CS</td>
<td>VREF Control and status register</td>
</tr>
<tr>
<td>VREF_CALIB</td>
<td>VREF Calibration register</td>
</tr>
</tbody>
</table>
3.27.2. Descriptions of Peripheral functions

VREF firmware functions are listed in the table shown as below:

<table>
<thead>
<tr>
<th>Function name</th>
<th>Function description</th>
</tr>
</thead>
<tbody>
<tr>
<td>vref_deinit</td>
<td>deinitialize the VREF</td>
</tr>
<tr>
<td>vref_enable</td>
<td>enable VREF</td>
</tr>
<tr>
<td>vref_disable</td>
<td>disable VREF</td>
</tr>
<tr>
<td>vref_high_impedance_mode_enable</td>
<td>enable VREF high impedance mode</td>
</tr>
<tr>
<td>vref_high_impedance_mode_disable</td>
<td>disable VREF high impedance mode</td>
</tr>
<tr>
<td>vref_status_get</td>
<td>get the status of VREF</td>
</tr>
<tr>
<td>vref_calib_value_set</td>
<td>set the calibration value of VREF</td>
</tr>
<tr>
<td>vref_calib_value_get</td>
<td>get the calibration value of VREF</td>
</tr>
</tbody>
</table>

vref_deinit

The description of vref_deinit is shown as below:

<table>
<thead>
<tr>
<th>Function name</th>
<th>vref_deinit</th>
</tr>
</thead>
<tbody>
<tr>
<td>Function prototype</td>
<td>void vref_deinit(void);</td>
</tr>
<tr>
<td>Function descriptions</td>
<td>deinitialize the VREF</td>
</tr>
<tr>
<td>Precondition</td>
<td>-</td>
</tr>
<tr>
<td>The called functions</td>
<td>-</td>
</tr>
<tr>
<td>Input parameter(in)</td>
<td>-</td>
</tr>
<tr>
<td>Output parameter(out)</td>
<td>-</td>
</tr>
<tr>
<td>Return value</td>
<td>-</td>
</tr>
</tbody>
</table>

Example:

 /**< deinitialize the VREF */

vref_deinit();

vref_enable

The description of vref_enable is shown as below:

<table>
<thead>
<tr>
<th>Function name</th>
<th>vref_enable</th>
</tr>
</thead>
<tbody>
<tr>
<td>Function prototype</td>
<td>void vref_enable(void);</td>
</tr>
<tr>
<td>Function descriptions</td>
<td>enable VREF</td>
</tr>
</tbody>
</table>
example:

/* enable VREF */
vref_enablet();

vref_disable

The description of vref_disable is shown as below:

Table 3-855. Function vref_disable

<table>
<thead>
<tr>
<th>Function name</th>
<th>vref_disable</th>
</tr>
</thead>
<tbody>
<tr>
<td>Function prototype</td>
<td>void vref_disable(void);</td>
</tr>
<tr>
<td>Function descriptions</td>
<td>disable VREF</td>
</tr>
<tr>
<td>Precondition</td>
<td>-</td>
</tr>
<tr>
<td>The called functions</td>
<td>-</td>
</tr>
<tr>
<td>Input parameter(in)</td>
<td>-</td>
</tr>
<tr>
<td>Output parameter(out)</td>
<td>-</td>
</tr>
<tr>
<td>Return value</td>
<td>-</td>
</tr>
</tbody>
</table>

example:

/* disable VREF */
vref_disable();

vref_high_impedance_mode_enable

The description of vref_high_impedance_mode_enable is shown as below:

Table 3-856. Function vref_high_impedance_mode_enable

<table>
<thead>
<tr>
<th>Function name</th>
<th>vref_high_impedance_mode_enable</th>
</tr>
</thead>
<tbody>
<tr>
<td>Function prototype</td>
<td>void vref_high_impedance_mode_enable(void);</td>
</tr>
<tr>
<td>Function descriptions</td>
<td>enable VREF high impedance mode</td>
</tr>
<tr>
<td>Precondition</td>
<td>-</td>
</tr>
</tbody>
</table>
The called functions

<table>
<thead>
<tr>
<th>Function name</th>
<th>vref_high_impedance_mode_disable</th>
</tr>
</thead>
<tbody>
<tr>
<td>Function prototype</td>
<td>void vref_high_impedance_mode_disable(void);</td>
</tr>
<tr>
<td>Function descriptions</td>
<td>disable VREF high impedance mode</td>
</tr>
<tr>
<td>Precondition</td>
<td>-</td>
</tr>
<tr>
<td>The called functions</td>
<td>-</td>
</tr>
<tr>
<td>Input parameter(in)</td>
<td>-</td>
</tr>
<tr>
<td>Output parameter(out)</td>
<td>-</td>
</tr>
<tr>
<td>Return value</td>
<td>-</td>
</tr>
</tbody>
</table>

Example:

/* enable VREF high impedance mode */

vref_high_impedance_mode_enable();

vref_high_impedance_mode_disable

The description of vref_high_impedance_mode_disable is shown as below:

Table 3-857. Function vref_high_impedance_mode_disable

<table>
<thead>
<tr>
<th>Function name</th>
<th>vref_status_get</th>
</tr>
</thead>
<tbody>
<tr>
<td>Function prototype</td>
<td>FlagStatus vref_status_get(void);</td>
</tr>
<tr>
<td>Function descriptions</td>
<td>get the status of VREF</td>
</tr>
<tr>
<td>Precondition</td>
<td>-</td>
</tr>
<tr>
<td>The called functions</td>
<td>-</td>
</tr>
</tbody>
</table>

Example:

/* disable VREF high impedance mode */

vref_high_impedance_mode_disable();

vref_status_get

The description of vref_status_get is shown as below:

Table 3-858. Function vref_status_get
<table>
<thead>
<tr>
<th>Input parameter (in)</th>
<th>-</th>
<th>-</th>
</tr>
</thead>
<tbody>
<tr>
<td>Output parameter (out)</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>Return value</td>
<td>FlagStatus</td>
<td>SET or RESET</td>
</tr>
</tbody>
</table>

example:

```c
/* get the status of VREF */
FlagStatus status;
status = vref_status_get();
```

**vref_calib_value_set**

The description of vref_calib_value_set is shown as below:

<table>
<thead>
<tr>
<th>Function name</th>
<th>vref_calib_value_set</th>
</tr>
</thead>
<tbody>
<tr>
<td>Function prototype</td>
<td>void vref_calib_value_set(uint8_t value);</td>
</tr>
<tr>
<td>Function descriptions</td>
<td>set the calibration value of VREF</td>
</tr>
<tr>
<td>Precondition</td>
<td>-</td>
</tr>
<tr>
<td>The called functions</td>
<td>-</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Input parameter (in)</th>
<th>value</th>
<th>calibration value (0x00 - 0x3F)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Output parameter (out)</td>
<td>-</td>
<td>-</td>
</tr>
</tbody>
</table>

| Return value | - | |

example:

```c
/* set the calibration value of VREF */
vref_calib_value_set(0x0A);
```

**vref_calib_value_get**

The description of vref_calib_value_get is shown as below:

<table>
<thead>
<tr>
<th>Function name</th>
<th>vref_calib_value_get</th>
</tr>
</thead>
<tbody>
<tr>
<td>Function prototype</td>
<td>uint8_t vref_calib_value_get(void);</td>
</tr>
<tr>
<td>Function descriptions</td>
<td>get the calibration value of VREF</td>
</tr>
<tr>
<td>Precondition</td>
<td>-</td>
</tr>
<tr>
<td>The called functions</td>
<td>-</td>
</tr>
</tbody>
</table>
example:

/* get the calibration value of VREF */

uint8_t cal_val;

cal_val = vref_calib_value_get();

### 3.28. WWDGT

The window watchdog timer (WWDGT) is used to detect system failures due to software malfunctions. The WWDGT registers are listed in chapter 3.28.1, the WWDGT firmware functions are introduced in chapter 3.28.2.

#### 3.28.1. Descriptions of Peripheral registers

WWDGT registers are listed in the table shown as below:

<table>
<thead>
<tr>
<th>Registers</th>
<th>Descriptions</th>
</tr>
</thead>
<tbody>
<tr>
<td>WWDGT_CTL</td>
<td>WWDGT control register</td>
</tr>
<tr>
<td>WWDGT_CFG</td>
<td>WWDGT configuration register</td>
</tr>
<tr>
<td>WWDGT_STAT</td>
<td>WWDGT status register</td>
</tr>
</tbody>
</table>

#### 3.28.2. Descriptions of Peripheral functions

WWDGT firmware functions are listed in the table shown as below:

<table>
<thead>
<tr>
<th>Function name</th>
<th>Function description</th>
</tr>
</thead>
<tbody>
<tr>
<td>wwdgt_deinit</td>
<td>reset the window watchdog timer configuration</td>
</tr>
<tr>
<td>wwdgt_enable</td>
<td>start the window watchdog timer counter</td>
</tr>
<tr>
<td>wwdgt_counter_update</td>
<td>configure the window watchdog timer counter value</td>
</tr>
<tr>
<td>wwdgt_config</td>
<td>configure counter value, window value, and prescaler divider value</td>
</tr>
<tr>
<td>wwdgt_interrupt_enable</td>
<td>enable early wakeup interrupt of WWDGT</td>
</tr>
<tr>
<td>wwdgt_flag_get</td>
<td>check early wakeup interrupt state of WWDGT</td>
</tr>
<tr>
<td>wwdgt_flag_clear</td>
<td>clear early wakeup interrupt state of WWDGT</td>
</tr>
</tbody>
</table>
**wwdgt_deinit**

The description of `wwdgt_deinit` is shown as below:

<table>
<thead>
<tr>
<th>Function name</th>
<th><code>wwdgt_deinit</code></th>
</tr>
</thead>
<tbody>
<tr>
<td>Function prototype</td>
<td><code>void wwdgt_deinit(void);</code></td>
</tr>
<tr>
<td>Function descriptions</td>
<td>reset the window watchdog timer configuration</td>
</tr>
<tr>
<td>Precondition</td>
<td>-</td>
</tr>
<tr>
<td>The called functions</td>
<td>-</td>
</tr>
<tr>
<td>Input parameter(in)</td>
<td>-</td>
</tr>
<tr>
<td>Output parameter(out)</td>
<td>-</td>
</tr>
<tr>
<td>Return value</td>
<td>-</td>
</tr>
</tbody>
</table>

Example:

/* reset the window watchdog timer configuration */

`wwdgt_deinit();`

**wwdgt_enable**

The description of `wwdgt_enable` is shown as below:

<table>
<thead>
<tr>
<th>Function name</th>
<th><code>wwdgt_enable</code></th>
</tr>
</thead>
<tbody>
<tr>
<td>Function prototype</td>
<td><code>void wwdgt_enable(void);</code></td>
</tr>
<tr>
<td>Function descriptions</td>
<td>start the window watchdog timer counter</td>
</tr>
<tr>
<td>Precondition</td>
<td>-</td>
</tr>
<tr>
<td>The called functions</td>
<td>-</td>
</tr>
<tr>
<td>Input parameter(in)</td>
<td>-</td>
</tr>
<tr>
<td>Output parameter(out)</td>
<td>-</td>
</tr>
<tr>
<td>Return value</td>
<td>-</td>
</tr>
</tbody>
</table>

Example:

/* start the WWDGT counter */

`wwdgt_enable();`
wwdgt_counter_update

The description of wwdgt_counter_update is shown as below:

### Table 3-865. Function wwdgt_counter_update

<table>
<thead>
<tr>
<th>Function name</th>
<th>wwdgt_counter_update</th>
</tr>
</thead>
<tbody>
<tr>
<td>Function prototype</td>
<td>void wwdgt_counter_update(uint16_t counter_value);</td>
</tr>
<tr>
<td>Function descriptions</td>
<td>configure the window watchdog timer counter value</td>
</tr>
<tr>
<td>Precondition</td>
<td>-</td>
</tr>
<tr>
<td>The called functions</td>
<td>-</td>
</tr>
<tr>
<td>Input parameter(in)</td>
<td>counter_value</td>
</tr>
<tr>
<td>counter_value</td>
<td>counter_value: 0x00000000 - 0x0000007F</td>
</tr>
<tr>
<td>Output parameter(out)</td>
<td>-</td>
</tr>
<tr>
<td>Return value</td>
<td>-</td>
</tr>
</tbody>
</table>

Example:

```c
/* update WWDGT counter to 0x7F */
wwdgt_counter_update(127);
```

wwdgt_config

The description of wwdgt_config is shown as below:

### Table 3-866. Function wwdgt_config

<table>
<thead>
<tr>
<th>Function name</th>
<th>wwdgt_config</th>
</tr>
</thead>
<tbody>
<tr>
<td>Function prototype</td>
<td>void wwdgt_config(uint16_t counter, uint16_t window, uint32_t prescaler);</td>
</tr>
<tr>
<td>Function descriptions</td>
<td>configure counter value, window value, and prescaler divider value</td>
</tr>
<tr>
<td>Precondition</td>
<td>-</td>
</tr>
<tr>
<td>The called functions</td>
<td>-</td>
</tr>
<tr>
<td>Input parameter(in)</td>
<td>counter</td>
</tr>
<tr>
<td>counter</td>
<td>counter: 0x00000000 - 0x0000007F</td>
</tr>
<tr>
<td>Input parameter(in)</td>
<td>window</td>
</tr>
<tr>
<td>window</td>
<td>window: 0x00000000 - 0x0000007F</td>
</tr>
<tr>
<td>Input parameter(in)</td>
<td>prescaler</td>
</tr>
<tr>
<td>prescaler</td>
<td>wwdgt prescaler value</td>
</tr>
</tbody>
</table>

- **WWDGT_CFG_PSC_DIV1**
  - the time base of WWDGT counter = (PCLK1/4096)/1
- **WWDGT_CFG_PSC_DIV2**
  - the time base of WWDGT counter = (PCLK1/4096)/2
- **WWDGT_CFG_PSC_DIV4**
  - the time base of WWDGT counter = (PCLK1/4096)/4
- **WWDGT_CFG_PSC_DIV8**
  - the time base of WWDGT counter = (PCLK1/4096)/8
### Output parameter (out)

- -

### Return value

- -

Example:

```c
/* configure WWDGT counter value to 0x7F, window value to 0x50, prescaler divider value to 8 */
wwdgt_config(127, 80, WWDGT_CFG_PSC_DIV8);
```

**wwdgt_interrupt_enable**

The description of `wwdgt_interrupt_enable` is shown as below:

**Table 3-867. Function `wwdgt_interrupt_enable`**

<table>
<thead>
<tr>
<th>Function name</th>
<th>wwdgt_interrupt_enable</th>
</tr>
</thead>
<tbody>
<tr>
<td>Function prototype</td>
<td>void wwdgt_interrupt_enable(void);</td>
</tr>
<tr>
<td>Function descriptions</td>
<td>enable early wakeup interrupt of WWDGT</td>
</tr>
<tr>
<td>Precondition</td>
<td>-</td>
</tr>
<tr>
<td>The called functions</td>
<td>-</td>
</tr>
</tbody>
</table>

| Input parameter (in) | - |

---

GD32L23x Firmware Library User Guide
### Output parameter(out)

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Return value</td>
<td></td>
</tr>
</tbody>
</table>

**Example:**

/* enable early wakeup interrupt of WWDGT */

wwdgt_interrupt_enable();

**wwdgt_flag_get**

The description of wwdgt_flag_get is shown as below:

**Table 3-868. Function wwdgt_flag_get**

<table>
<thead>
<tr>
<th>Function name</th>
<th>wwdgt_flag_get</th>
</tr>
</thead>
<tbody>
<tr>
<td>Function prototype</td>
<td>FlagStatus wwdgt_flag_get(void);</td>
</tr>
<tr>
<td>Function descriptions</td>
<td>check early wakeup interrupt state of WWDGT</td>
</tr>
<tr>
<td>Precondition</td>
<td>-</td>
</tr>
<tr>
<td>The called functions</td>
<td>-</td>
</tr>
<tr>
<td>Input parameter(in)</td>
<td>-</td>
</tr>
<tr>
<td>Output parameter(out)</td>
<td>-</td>
</tr>
<tr>
<td>Return value</td>
<td>SET or RESET</td>
</tr>
</tbody>
</table>

**Example:**

/* test if the counter value update has reached the 0x40 */

FlagStatus status;

status = wwdgt_flag_get();

**wwdgt_flag_clear**

The description of wwdgt_flag_clear is shown as below:

**Table 3-869. Function wwdgt_flag_clear**

<table>
<thead>
<tr>
<th>Function name</th>
<th>wwdgt_flag_clear</th>
</tr>
</thead>
<tbody>
<tr>
<td>Function prototype</td>
<td>void wwdgt_flag_clear(void);</td>
</tr>
<tr>
<td>Function descriptions</td>
<td>clear early wakeup interrupt state of WWDGT</td>
</tr>
<tr>
<td>Precondition</td>
<td>-</td>
</tr>
<tr>
<td>The called functions</td>
<td>-</td>
</tr>
<tr>
<td>Input parameter(in)</td>
<td>-</td>
</tr>
</tbody>
</table>
Example:

/* clear early wakeup interrupt state of WWDGT */

wwdgt_flag_clear();
4. Revision history

Table 4-1. Revision history

<table>
<thead>
<tr>
<th>Revision No.</th>
<th>Description</th>
<th>Date</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.0</td>
<td>Initial Release</td>
<td>Aug.25, 2021</td>
</tr>
</tbody>
</table>
Important Notice

This document is the property of GigaDevice Semiconductor Inc. and its subsidiaries (the "Company"). This document, including any product of the Company described in this document (the "Product"), is owned by the Company under the intellectual property laws and treaties of the People’s Republic of China and other jurisdictions worldwide. The Company reserves all rights under such laws and treaties and does not grant any license under its patents, copyrights, trademarks, or other intellectual property rights. The names and brands of third party referred thereto (if any) are the property of their respective owner and referred to for identification purposes only.

The Company makes no warranty of any kind, express or implied, with regard to this document or any Product, including, but not limited to, the implied warranties of merchantability and fitness for a particular purpose. The Company does not assume any liability arising out of the application or use of any Product described in this document. Any information provided in this document is provided only for reference purposes. It is the responsibility of the user of this document to properly design, program, and test the functionality and safety of any application made of this information and any resulting product. Except for customized products which has been expressly identified in the applicable agreement, the Products are designed, developed, and/or manufactured for ordinary business, industrial, personal, and/or household applications only. The Products are not designed, intended, or authorized for use as components in systems designed or intended for the operation of weapons, weapons systems, nuclear installations, atomic energy control instruments, combustion control instruments, airplane or spaceship instruments, transportation instruments, traffic signal instruments, life-support devices or systems, other medical devices or systems (including resuscitation equipment and surgical implants), pollution control or hazardous substances management, or other uses where the failure of the device or Product could cause personal injury, death, property or environmental damage (“Unintended Uses”). Customers shall take any and all actions to ensure using and selling the Products in accordance with the applicable laws and regulations. The Company is not liable, in whole or in part, and customers shall and hereby do release the Company as well as it’s suppliers and/or distributors from any claim, damage, or other liability arising from or related to all Unintended Uses of the Products. Customers shall indemnify and hold the Company as well as it’s suppliers and/or distributors harmless from and against all claims, costs, damages, and other liabilities, including claims for personal injury or death, arising from or related to any Unintended Uses of the Products.

Information in this document is provided solely in connection with the Products. The Company reserves the right to make changes, corrections, modifications or improvements to this document and Products and services described herein at any time, without notice.