

GigaDevice Semiconductor Inc.

GD32F527

Arm[®] Cortex[®]-M33 32-bit MCU

**Firmware Library
User Guide**

Revision 1.1

(Aug. 2025)

Table of Contents

Table of Contents	1
List of Figures	5
List of Tables	6
1. Introduction	35
1.1. Rules of User Manual and Firmware Library	35
1.1.1. Peripherals.....	35
1.1.2. Naming rules.....	36
2. Firmware Library Overview	38
2.1. File Structure of Firmware Library	38
2.1.1. Examples Folder	39
2.1.2. Firmware Folder	39
2.1.3. Template Folder	40
2.1.4. Utilities Folder	42
2.2. File descriptions of Firmware Library	43
3. Firmware Library of Standard Peripherals	44
3.1. Overview of Firmware Library of Standard Peripherals.....	44
3.2. ADC	44
3.2.1. Descriptions of Peripheral registers.....	44
3.2.2. Descriptions of Peripheral functions	45
3.3. CAN	78
3.3.1. Descriptions of Peripheral registers.....	78
3.3.2. Descriptions of Peripheral functions	79
3.4. CAU	102
3.4.1. Descriptions of Peripheral registers.....	102
3.4.2. Descriptions of Peripheral functions	102
3.5. CRC	129
3.5.1. Descriptions of Peripheral registers.....	129
3.5.2. Descriptions of Peripheral functions	130
3.6. CTC.....	134
3.6.1. Descriptions of Peripheral registers.....	134
3.6.2. Descriptions of Peripheral functions	134
3.7. DAC	146
3.7.1. Peripheral register description	147
3.7.2. Descriptions of Peripheral functions	147
3.8. DBG	165

3.8.1.	Descriptions of Peripheral registers	165
3.8.2.	Descriptions of Peripheral functions	165
3.9.	DCI	174
3.9.1.	Descriptions of Peripheral registers	174
3.9.2.	Descriptions of Peripheral functions	174
3.10.	DMA	187
3.10.1.	Descriptions of Peripheral registers	187
3.10.2.	Descriptions of Peripheral functions	187
3.11.	ENET	212
3.11.1.	Descriptions of Peripheral registers	213
3.11.2.	Descriptions of Peripheral functions	215
3.12.	EXMC	302
3.12.1.	Descriptions of Peripheral registers	302
3.12.2.	Descriptions of Peripheral functions	302
3.13.	EXTI	337
3.13.1.	Descriptions of Peripheral registers	338
3.13.2.	Descriptions of Peripheral functions	338
3.14.	FMC	345
3.14.1.	Descriptions of Peripheral registers	345
3.14.2.	Descriptions of Peripheral functions	346
3.15.	FWDGT	383
3.15.1.	Descriptions of Peripheral registers	383
3.15.2.	Descriptions of Peripheral functions	384
3.16.	GPIO	389
3.16.1.	Descriptions of Peripheral registers	389
3.16.2.	Descriptions of Peripheral functions	390
3.17.	HAU	400
3.17.1.	Descriptions of Peripheral registers	400
3.17.2.	Descriptions of Peripheral functions	401
3.18.	I2C	421
3.18.1.	Descriptions of Peripheral registers	421
3.18.2.	Descriptions of Peripheral functions	421
3.19.	IPA	486
3.19.1.	Descriptions of Peripheral registers	486
3.19.2.	Descriptions of Peripheral functions	486
3.20.	IREF	504
3.20.1.	Descriptions of Peripheral registers	504
3.20.2.	Descriptions of Peripheral functions	504
3.21.	MISC	508

3.21.1.	Descriptions of Peripheral registers	508
3.21.2.	Descriptions of Peripheral functions	510
3.22.	PKCAU	517
3.22.1.	Descriptions of Peripheral registers	517
3.22.2.	Descriptions of Peripheral functions	517
3.23.	PMU.....	546
3.23.1.	Descriptions of Peripheral registers.....	546
3.23.2.	Descriptions of Peripheral functions	547
3.24.	RCU	558
3.24.1.	Descriptions of Peripheral registers	559
3.24.2.	Descriptions of Peripheral functions	560
3.25.	RTC	595
3.25.1.	Descriptions of Peripheral registers.....	595
3.25.2.	Descriptions of Peripheral functions	596
3.26.	SDIO	623
3.26.1.	Descriptions of Peripheral registers.....	623
3.26.2.	Descriptions of Peripheral functions	623
3.27.	SPI.....	653
3.27.1.	Descriptions of Peripheral registers.....	653
3.27.2.	Descriptions of Peripheral functions	654
3.28.	SYSCFG	680
3.28.1.	Descriptions of Peripheral registers.....	680
3.28.2.	Descriptions of Peripheral functions	680
3.29.	TIMER.....	686
3.29.1.	Descriptions of Peripheral registers.....	687
3.29.2.	Descriptions of Peripheral functions	688
3.30.	TLI	765
3.30.1.	Descriptions of Peripheral registers.....	765
3.30.2.	Descriptions of Peripheral functions	765
3.31.	TRNG.....	783
3.31.1.	Descriptions of Peripheral registers.....	783
3.31.2.	Descriptions of Peripheral functions	784
3.32.	USART.....	789
3.32.1.	Descriptions of Peripheral registers.....	789
3.32.2.	Descriptions of Peripheral functions	789
3.33.	WWDGT.....	825
3.33.1.	Descriptions of Peripheral registers.....	825
3.33.2.	Descriptions of Peripheral functions	825
3.34.	SAI.....	830

3.34.1.	Descriptions of Peripheral registers.....	830
3.34.2.	Descriptions of Peripheral functions.....	831
4.	Revision history.....	850

List of Figures

Figure 2-1. File structure of firmware library of GD32F527	38
Figure 2-2. Select peripheral example files	40
Figure 2-3. Copy the peripheral example files	41
Figure 2-4. Open the project file	41
Figure 2-5. Configure project files	42
Figure 2-6. Compile-debug-download	42

List of Tables

Table 1-1. Peripherals	35
Table 2-1. Function descriptions of Firmware Library	43
Table 3-1. Peripheral function format of Firmware Library	44
Table 3-2. ADC Registers	44
Table 3-3. ADC firmware function.....	45
Table 3-4. Function adc_deinit.....	46
Table 3-5. Function adc_clock_config.....	47
Table 3-6. Function adc_special_function_config.....	48
Table 3-7. Function adc_data_alignment_config.....	48
Table 3-8. Function adc_enable	49
Table 3-9. Function adc_disable	50
Table 3-10. Function adc_calibration_enable.....	50
Table 3-11. Function adc_channel_16_to_18	51
Table 3-12. Function adc_resolution_config	51
Table 3-13. Function adc_oversample_mode_config	52
Table 3-14. Function adc_oversample_mode_enable	54
Table 3-15. Function adc_oversample_mode_disable	54
Table 3-16. Function adc_dma_mode_enable.....	55
Table 3-17. Function adc_dma_mode_disable.....	55
Table 3-18. Function adc_dma_request_after_last_enable	56
Table 3-19. Function adc_dma_mode_disable.....	56
Table 3-20. Function adc_discontinuous_mode_config	57
Table 3-21. Function adc_channel_length_config.....	58
Table 3-22. Function adc_routine_channel_config	58
Table 3-23. Function adc_inserted_channel_config	59
Table 3-24. Function adc_inserted_channel_offset_config.....	60
Table 3-25. Function adc_external_trigger_source_config	61
Table 3-26. Function adc_external_trigger_config.....	63
Table 3-27. Function adc_software_trigger_enable	64
Table 3-28. Function adc_end_of_conversion_config.....	65
Table 3-29. Function adc_routine_data_read	65
Table 3-30. Function adc_inserted_data_read	66
Table 3-31. Function adc_watchdog_single_channel_disable.....	67
Table 3-32. Function adc_watchdog_single_channel_enable.....	67
Table 3-33. Function adc_watchdog_group_channel_enable.....	68
Table 3-34. Function adc_watchdog_disable	68
Table 3-35. Function adc_watchdog_threshold_config	69
Table 3-36. Function adc_sync_mode_config.....	70
Table 3-37. Function adc_interrupt_disable	71
Table 3-38. Function adc_sync_dma_config	71
Table 3-39. Function adc_sync_dma_request_after_last_enable	72

Table 3-40. Function <code>adc_sync_dma_request_after_last_disable</code>	72
Table 3-41. Function <code>adc_interrupt_disable</code>	73
Table 3-42. Function <code>adc_routine_software_startconv_flag_get</code>	73
Table 3-43. Function <code>adc_inserted_software_startconv_flag_get</code>	74
Table 3-44. Function <code>adc_flag_get</code>	74
Table 3-45. Function <code>adc_flag_clear</code>	75
Table 3-46. Function <code>adc_interrupt_flag_get</code>	76
Table 3-47. Function <code>adc_interrupt_flag_clear</code>	76
Table 3-48. Function <code>adc_interrupt_enable</code>	77
Table 3-49. Function <code>adc_interrupt_disable</code>	78
Table 3-50. CAN Registers	79
Table 3-51. CAN firmware function	79
Table 3-52. Structure <code>can_parameter_struct</code>	80
Table 3-53. Structure <code>can_transmit_message_struct</code>	81
Table 3-54. Structure <code>can_receive_message_struct</code>	81
Table 3-55. Structure <code>can_filter_parameter_struct</code>	81
Table 3-56. Structure <code>can_fd_tdc_struct</code>	82
Table 3-57. Structure <code>can_fdframe_struct</code>	82
Table 3-58. Function <code>can_deinit</code>	82
Table 3-59. Function <code>can_struct_para_init</code>	83
Table 3-60. Function <code>can_init</code>	83
Table 3-61. Function <code>can_fd_init</code>	84
Table 3-62. Function <code>can_filter_init</code>	85
Table 3-63. Function <code>can_filter_mask_mode_init</code>	85
Table 3-64. Function <code>can_monitor_mode_set</code>	86
Table 3-65. Function <code>can_fd_function_enable</code>	87
Table 3-66. Function <code>can_fd_function_disable</code>	87
Table 3-67. Function <code>can1_filter_start_bank</code>	88
Table 3-68. Function <code>can_debug_freeze_enable</code>	88
Table 3-69. Function <code>can_debug_freeze_disable</code>	89
Table 3-70. Function <code>can_time_trigger_mode_enable</code>	89
Table 3-71. Function <code>can_time_trigger_mode_disable</code>	90
Table 3-72. Function <code>can_message_transmit</code>	90
Table 3-73. Function <code>can_transmit_states</code>	91
Table 3-74. Function <code>can_transmission_stop</code>	92
Table 3-75. Function <code>can_message_receive</code>	92
Table 3-76. Function <code>can_fifo_release</code>	93
Table 3-77. Function <code>can_receive_message_length_get</code>	93
Table 3-78. Function <code>can_working_mode_set</code>	94
Table 3-79. Function <code>can_wakeup</code>	95
Table 3-80. Function <code>can_error_get</code>	95
Table 3-81. Function <code>can_receive_error_number_get</code>	96
Table 3-82. Function <code>can_transmit_error_number_get</code>	96
Table 3-83. Function <code>can_flag_get</code>	97

Table 3-84. Function <code>can_flag_clear</code>	97
Table 3-85. Function <code>can_interrupt_enable</code>	98
Table 3-86. Function <code>can_interrupt_disable</code>	99
Table 3-87. Function <code>can_interrupt_flag_get</code>	100
Table 3-88. Function <code>can_interrupt_flag_clear</code>	101
Table 3-89. CAU Registers	102
Table 3-90. CAU firmware function	103
Table 3-91. Structure <code>cau_key_parameter_struct</code>	103
Table 3-92. Structure <code>cau_iv_parameter_struct</code>	104
Table 3-93. Structure <code>cau_context_parameter_struct</code>	104
Table 3-94. Structure <code>cau_parameter_struct</code>	104
Table 3-95. Function <code>cau_deinit</code>	105
Table 3-96. Function <code>cau_struct_para_init</code>	105
Table 3-97. Function <code>cau_key_struct_para_init</code>	106
Table 3-98. Function <code>cau_iv_struct_para_init</code>	106
Table 3-99. Function <code>cau_context_struct_para_init</code>	107
Table 3-100. Function <code>cau_enable</code>	107
Table 3-101. Function <code>cau_disable</code>	108
Table 3-102. Function <code>cau_dma_enable</code>	108
Table 3-103. Function <code>cau_dma_disable</code>	109
Table 3-104. Function <code>cau_init</code>	109
Table 3-105. Function <code>cau_aes_keysize_config</code>	111
Table 3-106. Function <code>cau_key_init</code>	111
Table 3-107. Function <code>cau_iv_init</code>	112
Table 3-108. Function <code>cau_phase_config</code>	113
Table 3-109. Function <code>cau_fifo_flush</code>	113
Table 3-110. Function <code>cau_enable_state_get</code>	114
Table 3-111. Function <code>cau_data_write</code>	114
Table 3-112. Function <code>cau_data_read</code>	115
Table 3-113. Function <code>cau_context_save</code>	115
Table 3-114. Function <code>cau_context_restore</code>	116
Table 3-115. Function <code>cau_aes_ecb</code>	117
Table 3-116. Function <code>cau_aes_cbc</code>	118
Table 3-117. Function <code>cau_aes_ctr</code>	119
Table 3-118. Function <code>cau_aes_cfb</code>	120
Table 3-119. Function <code>cau_aes_ofb</code>	121
Table 3-120. Function <code>cau_aes_gcm</code>	121
Table 3-121. Function <code>cau_aes_ccm</code>	122
Table 3-122. Function <code>cau_tdes_ecb</code>	124
Table 3-123. Function <code>cau_tdes_cbc</code>	124
Table 3-124. Function <code>cau_des_ecb</code>	125
Table 3-125. Function <code>cau_des_cbc</code>	126
Table 3-126. Function <code>cau_flag_get</code>	127
Table 3-127. Function <code>cau_interrupt_enable</code>	128

Table 3-128. Function <code>cau_interrupt_disable</code>	128
Table 3-129. Function <code>cau_interrupt_flag_get</code>	129
Table 3-130. CRC Registers	129
Table 3-131. CRC firmware function	130
Table 3-132. Function <code>crc_deinit</code>	130
Table 3-133. Function <code>crc_data_register_reset</code>	130
Table 3-134. Function <code>crc_data_register_read</code>	131
Table 3-135. Function <code>crc_free_data_register_read</code>	131
Table 3-136. Function <code>crc_free_data_register_write</code>	132
Table 3-137. Function <code>crc_single_data_calculate</code>	132
Table 3-138. Function <code>crc_block_data_calculate</code>	133
Table 3-139. CTC Registers	134
Table 3-140. CTC firmware function.....	134
Table 3-141. Function <code>ctc_deinit</code>	135
Table 3-142. Function <code>ctc_counter_enable</code>	135
Table 3-143. Function <code>ctc_counter_disable</code>	136
Table 3-144. Function <code>ctc_irc48m_trim_value_config</code>	136
Table 3-145. Function <code>ctc_software_refsource_pulse_generate</code>	137
Table 3-146. Function <code>ctc_hardware_trim_mode_config</code>	137
Table 3-147. Function <code>ctc_refsource_polarity_config</code>	138
Table 3-148. Function <code>ctc_refsource_signal_select</code>	138
Table 3-149. Function <code>ctc_refsource_prescaler_config</code>	139
Table 3-150. Function <code>ctc_clock_limit_value_config</code>	140
Table 3-151. Function <code>ctc_counter_reload_value_config</code>	140
Table 3-152. Function <code>ctc_counter_capture_value_read</code>	141
Table 3-153. Function <code>ctc_counter_direction_read</code>	141
Table 3-154. Function <code>ctc_counter_reload_value_read</code>	142
Table 3-155. Function <code>ctc_irc48m_trim_value_read</code>	142
Table 3-156. Function <code>ctc_interrupt_enable</code>	143
Table 3-157. Function <code>ctc_interrupt_disable</code>	143
Table 3-158. Function <code>ctc_interrupt_flag_get</code>	144
Table 3-159. Function <code>ctc_interrupt_flag_clear</code>	144
Table 3-160. Function <code>ctc_flag_get</code>	145
Table 3-161. Function <code>ctc_flag_clear</code>	146
Table 3-162. DAC Registers	147
Table 3-163. DAC firmware functions	147
Table 3-164. Function <code>dac_deinit</code>	148
Table 3-165. Function <code>dac_enable</code>	148
Table 3-166. Function <code>dac_disable</code>	149
Table 3-167. Function <code>dac_dma_enable</code>	150
Table 3-168. Function <code>dac_dma_disable</code>	150
Table 3-169. Function <code>dac_output_buffer_enable</code>	151
Table 3-170. Function <code>dac_output_buffer_disable</code>	151
Table 3-171. Function <code>dac_output_value_get</code>	152

Table 3-172. Function <code>dac_data_set</code>	152
Table 3-173. Function <code>dac_trigger_enable</code>	153
Table 3-174. Function <code>dac_trigger_disable</code>	154
Table 3-175. Function <code>dac_trigger_source_config</code>	154
Table 3-176. Function <code>dac_software_trigger_enable</code>	155
Table 3-177. Function <code>dac_wave_mode_config</code>	156
Table 3-178. Function <code>dac_lfsr_noise_config</code>	157
Table 3-179. Function <code>dac_triangle_noise_config</code>	157
Table 3-180. Function <code>dac_concurrent_enable</code>	158
Table 3-181. Function <code>dac_concurrent_disable</code>	158
Table 3-182. Function <code>dac_concurrent_software_trigger_enable</code>	159
Table 3-183. Function <code>dac_concurrent_output_buffer_enable</code>	159
Table 3-184. Function <code>dac_concurrent_output_buffer_disable</code>	160
Table 3-185. Function <code>dac_concurrent_data_set</code>	160
Table 3-186. Function <code>dac_flag_get</code>	161
Table 3-187. Function <code>dac_flag_clear</code>	162
Table 3-188. Function <code>dac_interrupt_enable</code>	162
Table 3-189. Function <code>dac_interrupt_disable</code>	163
Table 3-190. Function <code>dac_interrupt_flag_get</code>	164
Table 3-191. Function <code>dac_interrupt_flag_clear</code>	164
Table 3-192. DBG Registers.....	165
Table 3-193. DBG firmware function	165
Table 3-194. Enum <code>dbg_periph_enum</code>	166
Table 3-195. Function <code>dbg_deinit</code>	167
Table 3-196. Function <code>dbg_id_get</code>	167
Table 3-197. Function <code>dbg_low_power_enable</code>	168
Table 3-198. Function <code>dbg_low_power_disable</code>	169
Table 3-199. Function <code>dbg_periph_enable</code>	170
Table 3-200. Function <code>dbg_periph_disable</code>	170
Table 3-201. Function <code>dbg_trace_pin_enable</code>	171
Table 3-202. Function <code>dbg_trace_pin_disable</code>	172
Table 3-203. Function <code>dbg_id_get</code>	172
Table 3-204. Function <code>dbg_id_set</code>	173
Table 3-205. DCI Registers.....	174
Table 3-206. DCI firmware function	174
Table 3-207. Structure <code>dc_i_parameter_struct</code>	175
Table 3-208. Function <code>dc_i_deinit</code>	175
Table 3-209. Function <code>dc_i_init</code>	176
Table 3-210. Function <code>dc_i_enable</code>	177
Table 3-211. Function <code>dc_i_disable</code>	177
Table 3-212. Function <code>dc_i_capture_enable</code>	177
Table 3-213. Function <code>dc_i_capture_disable</code>	178
Table 3-214. Function <code>dc_i_jpeg_enable</code>	178
Table 3-215. Function <code>dc_i_jpeg_disable</code>	179

Table 3-216. Function dci_crop_window_enable.....	179
Table 3-217. Function dci_crop_window_disable	180
Table 3-218. Function dci_crop_window_config	180
Table 3-219. Function dci_embedded_sync_enable	181
Table 3-220. Function dci_embedded_sync_disable	181
Table 3-221. Function dci_sync_codes_config.....	182
Table 3-222. Function dci_sync_codes_unmask_config	183
Table 3-223. Function dci_data_read.....	183
Table 3-224. Function dci_flag_get	184
Table 3-225. Function dci_interrupt_enable	184
Table 3-226. Function dci_interrupt_disable	185
Table 3-227. Function dci_interrupt_flag_get.....	185
Table 3-228. Function dci_interrupt_flag_clear	186
Table 3-229. DMA Registers.....	187
Table 3-230. DMA firmware function	187
Table 3-231. Structure dma_multi_data_parameter_struct.....	189
Table 3-232. Structure dma_single_data_parameter_struct.....	189
Table 3-233. Function dma_deinit	189
Table 3-234. Function dma_deinit	190
Table 3-235. Function dma_deinit	191
Table 3-236. Function dma_single_data_mode_init.....	191
Table 3-237. Function dma_multi_data_mode_init.....	192
Table 3-238. Function dma_periph_address_config	192
Table 3-239. Function dma_memory_address_config.....	193
Table 3-240. Function dma_transfer_number_config	194
Table 3-241. Function dma_transfer_number_get.....	195
Table 3-242. Function dma_priority_config	195
Table 3-243. Function dma_memory_burst_beats_config.....	196
Table 3-244. Function dma_periph_burst_beats_config.....	197
Table 3-245. Function dma_memory_width_config	198
Table 3-246. Function dma_periph_width_config	198
Table 3-247. Function dma_memory_address_generation_config	199
Table 3-248. Function dma_peripheral_address_generation_config.....	200
Table 3-249. Function dma_circulation_enable	201
Table 3-250. Function dma_circulation_disable	201
Table 3-251. Function dma_channel_enable	202
Table 3-252. Function dma_channel_disable	203
Table 3-253. Function dma_transfer_direction_config.....	203
Table 3-254. Function dma_switch_buffer_mode_config	204
Table 3-255. Function dma_using_memory_get.....	205
Table 3-256. Function dma_channel_subperipheral_select	205
Table 3-257. Function dma_flow_controller_config.....	206
Table 3-258. Function dma_switch_buffer_mode_enable.....	207
Table 3-259. Function dma_fifo_status_get.....	207

Table 3-260. Function dma_flag_get	208
Table 3-261. Function dma_flag_clear	209
Table 3-262. Function dma_interrupt_enable	209
Table 3-263. Function dma_interrupt_disable	210
Table 3-264. Function dma_interrupt_flag_get	211
Table 3-265. Function dma_interrupt_flag_clear	212
Table 3-266. ENET Registers	213
Table 3-267. ENET firmware function	215
Table 3-268. Structure enet_initpara_struct	218
Table 3-269. Structure enet_descriptors_struct	219
Table 3-270. Structure enet_ptp_systime_struct	219
Table 3-271. Enum enet_flag_enum	219
Table 3-272. Enum enet_flag_clear_enum	221
Table 3-273. Enum enet_int_enum	222
Table 3-274. Enum enet_int_flag_enum	223
Table 3-275. Enum enet_int_flag_clear_enum	224
Table 3-276. Enum enet_desc_reg_enum	225
Table 3-277. Enum enet_msc_counter_enum	226
Table 3-278. Enum enet_option_enum	226
Table 3-279. Enum enet_mediamode_enum	227
Table 3-280. Enum enet_chksumconf_enum	227
Table 3-281. Enum enet_frmrecept_enum	227
Table 3-282. Enum enet_registers_type_enum	228
Table 3-283. Enum enet_dmadirection_enum	228
Table 3-284. Enum enet_phydirection_enum	228
Table 3-285. Enum enet_regdirection_enum	228
Table 3-286. Enum enet_macaddress_enum	228
Table 3-287. Enum enet_descstate_enum	229
Table 3-288. Enum enet_msc_preset_enum	229
Table 3-289. Function enet_deinit	229
Table 3-290. Function enet_initpara_config	230
Table 3-291. Function enet_init	233
Table 3-292. Function enet_software_reset	234
Table 3-293. Function enet_rxframe_size_get	235
Table 3-294. Function enet_descriptors_chain_init	235
Table 3-295. Function enet_descriptors_ring_init	236
Table 3-296. Function enet_frame_receive	237
Table 3-297. Function enet_frame_transmit	237
Table 3-298. Function enet_transmit_checksum_config	238
Table 3-299. Function enet_enable	239
Table 3-300. Function enet_disable	239
Table 3-301. Function enet_mac_address_set	239
Table 3-302. Function enet_mac_address_get	240
Table 3-303. Function enet_flag_get	241

Table 3-304. Function enet_flag_clear	243
Table 3-305. Function enet_interrupt_enable	244
Table 3-306. Function enet_interrupt_disable	246
Table 3-307. Function enet_interrupt_flag_get	247
Table 3-308. Function enet_interrupt_flag_clear	249
Table 3-309. Function enet_tx_enable	251
Table 3-310. Function enet_tx_disable	251
Table 3-311. Function enet_rx_enable	251
Table 3-312. Function enet_rx_disable	252
Table 3-313. Function enet_registers_get	252
Table 3-314. Function enet_debug_status_get	253
Table 3-315. Function enet_address_filter_enable	254
Table 3-316. Function enet_address_filter_disable	255
Table 3-317. Function enet_address_filter_config	256
Table 3-318. Function enet_phy_config	257
Table 3-319. Function enet_phy_write_read	257
Table 3-320. Function enet_phyloopback_enable	258
Table 3-321. Function enet_phyloopback_disable	258
Table 3-322. Function enet_forward_feature_enable	259
Table 3-323. Function enet_forward_feature_disable	260
Table 3-324. Function enet_fliter_feature_enable	260
Table 3-325. Function enet_fliter_feature_disable	261
Table 3-326. Function enet_pauseframe_generate	262
Table 3-327. Function enet_pauseframe_detect_config	263
Table 3-328. Function enet_pauseframe_config	263
Table 3-329. Function enet_flowcontrol_threshold_config	264
Table 3-330. Function enet_flowcontrol_feature_enable	265
Table 3-331. Function enet_flowcontrol_feature_disable	266
Table 3-332. Function enet_dmaprocess_state_get	267
Table 3-333. Function enet_dmaprocess_resume	267
Table 3-334. Function enet_rxprocess_check_recovery	268
Table 3-335. Function enet_txfifo_flush	268
Table 3-336. Function enet_current_desc_address_get	269
Table 3-337. Function enet_desc_information_get	270
Table 3-338. Function enet_missed_frame_counter_get	271
Table 3-339. Function enet_desc_flag_get	271
Table 3-340. Function enet_desc_flag_set	273
Table 3-341. Function enet_desc_flag_clear	274
Table 3-342. Function enet_rx_desc_immediate_receive_complete_interrupt	275
Table 3-343. Function enet_rx_desc_delay_receive_complete_interrupt	275
Table 3-344. Function enet_rxframe_drop	276
Table 3-345. Function enet_dma_feature_enable	277
Table 3-346. Function enet_dma_feature_disable	277
Table 3-347. Function enet_rx_desc_enhanced_status_get	278

Table 3-348. Function enet_desc_select_enhanced_mode	279
Table 3-349. Function enet_ptp_enhanced_descriptors_chain_init	279
Table 3-350. Function enet_ptp_enhanced_descriptors_ring_init	280
Table 3-351. Function enet_ptpframe_receive_enhanced_mode	280
Table 3-352. Function enet_ptpframe_transmit_enhanced_mode	281
Table 3-353. Function enet_desc_select_normal_mode	282
Table 3-354. Function enet_ptp_normal_descriptors_chain_init	282
Table 3-355. Function enet_ptp_normal_descriptors_ring_init	283
Table 3-356. Function enet_ptpframe_receive_normal_mode	284
Table 3-357. Function enet_ptpframe_transmit_normal_mode	284
Table 3-358. Function enet_wum_filter_register_pointer_reset	285
Table 3-359. Function enet_wum_filter_config	285
Table 3-360. Function enet_wum_feature_enable	286
Table 3-361. Function enet_wum_feature_disable	287
Table 3-362. Function enet_msc_counters_reset	287
Table 3-363. Function enet_msc_feature_enable	288
Table 3-364. Function enet_msc_feature_disable	288
Table 3-365. Function enet_msc_counters_preset_config	289
Table 3-366. Function enet_msc_counters_get	290
Table 3-367. Function enet_ptp_subsecond_2_nanosecond	290
Table 3-368. Function enet_ptp_nanosecond_2_subsecond	291
Table 3-369. Function enet_ptp_feature_enable	291
Table 3-370. Function enet_ptp_feature_disable	292
Table 3-371. Function enet_ptp_timestamp_function_config	293
Table 3-372. Function enet_ptp_subsecond_increment_config	294
Table 3-373. Function enet_ptp_timestamp_addend_config	295
Table 3-374. Function enet_ptp_timestamp_update_config	295
Table 3-375. Function enet_ptp_expected_time_config	296
Table 3-376. Function enet_ptp_system_time_get	297
Table 3-377. Function enet_ptp_pps_output_frequency_config	297
Table 3-378. enet_ptp_start	298
Table 3-379. enet_ptp_finecorrection_adjfreq	299
Table 3-380. enet_ptp_coarsecorrection_systime_update	300
Table 3-381. enet_ptp_finecorrection_settime	300
Table 3-382. enet_ptp_flag_get	301
Table 3-383. Function enet_initpara_reset	301
Table 3-384. EXMC Registers	302
Table 3-385. EXMC firmware function	303
Table 3-386. Structure exmc_norsram_timing_parameter_struct	304
Table 3-387. Structure exmc_norsram_parameter_struct	304
Table 3-388. Structure exmc_nand_pccard_timing_parameter_struct	305
Table 3-389. Structure exmc_nand_parameter_struct	305
Table 3-390. Structure exmc_pccard_parameter_struct	305
Table 3-391. Structure exmc_sdram_timing_parameter_struct	306

Table 3-392. Structure exmc_sdram_parameter_struct	306
Table 3-393. Structure exmc_sdram_command_parameter_struct	307
Table 3-394. Structure exmc_sqpsram_parameter_struct.....	307
Table 3-395. Function exmc_norsram_deinit	307
Table 3-396. Function exmc_norsram_struct_para_init.....	308
Table 3-397. Function exmc_norsram_init.....	308
Table 3-398. Function exmc_norsram_enable	310
Table 3-399. Function exmc_norsram_disable	310
Table 3-400. Function exmc_nand_deinit	311
Table 3-401. Function exmc_nand_struct_para_init	311
Table 3-402. Function exmc_nand_init.....	312
Table 3-403. Function exmc_nand_enable	313
Table 3-404. Function exmc_nand_disable	313
Table 3-405. Function exmc_pccard_deinit	314
Table 3-406. Function exmc_pccard_struct_para_init.....	314
Table 3-407. Function exmc_pccard_init	315
Table 3-408. Function exmc_pccard_enable	316
Table 3-409. Function exmc_pccard_disable	316
Table 3-410. Function exmc_sdram_deinit	317
Table 3-411. Function exmc_sdram_struct_para_init.....	317
Table 3-412. Function exmc_sdram_init	318
Table 3-413. Function exmc_sdram_struct_command_para_init.....	319
Table 3-414. Function exmc_sqpsram_deinit	320
Table 3-415. Function exmc_sqpsram_struct_para_init	320
Table 3-416. Function exmc_sqpsram_init.....	321
Table 3-417. Function exmc_norsram_consecutive_clock_config	321
Table 3-418. Function exmc_norsram_page_size_config.....	322
Table 3-419. Function exmc_nand_ecc_config.....	323
Table 3-420. Function exmc_ecc_get	323
Table 3-421. Function exmc_sdram_readsample_enable	324
Table 3-422. Function exmc_sdram_readsample_config.....	324
Table 3-423. Function exmc_sdram_command_config.....	325
Table 3-424. Function exmc_sdram_refresh_count_set	326
Table 3-425. Function exmc_sdram_autorefresh_number_set.....	326
Table 3-426. Function exmc_sdram_write_protection_config	327
Table 3-427. Function exmc_sdram_bankstatus_get	327
Table 3-428. Function exmc_sqpsram_read_command_set.....	328
Table 3-429. Function exmc_sqpsram_write_command_set.....	329
Table 3-430. Function exmc_sqpsram_read_id_command_send	329
Table 3-431. Function exmc_sqpsram_write_cmd_send.....	330
Table 3-432. Function exmc_sqpsram_low_id_get.....	330
Table 3-433. Function exmc_sqpsram_low_id_get.....	331
Table 3-434. Function exmc_sqpsram_send_command_state_get.....	331
Table 3-435. Function exmc_interrupt_enable	332

Table 3-436. Function <code>exmc_interrupt_disable</code>	333
Table 3-437. Function <code>exmc_flag_get</code>	334
Table 3-438. Function <code>exmc_flag_clear</code>	335
Table 3-439. Function <code>exmc_interrupt_flag_get</code>	336
Table 3-440. Function <code>exmc_interrupt_flag_clear</code>	337
Table 3-441. EXTI Registers	338
Table 3-442. EXTI firmware function	338
Table 3-443. Enum <code>exti_line_enum</code>	338
Table 3-444. Enum <code>exti_mode_enum</code>	339
Table 3-445. Enum <code>exti_trig_type_enum</code>	339
Table 3-446. Function <code>exti_deinit</code>	339
Table 3-447. Function <code>exti_init</code>	340
Table 3-448. Function <code>exti_interrupt_enable</code>	341
Table 3-449. Function <code>exti_interrupt_disable</code>	341
Table 3-450. Function <code>exti_event_enable</code>	341
Table 3-451. Function <code>exti_event_disable</code>	342
Table 3-452. Function <code>exti_software_interrupt_enable</code>	342
Table 3-453. Function <code>exti_software_interrupt_disable</code>	343
Table 3-454. Function <code>exti_flag_get</code>	343
Table 3-455. Function <code>exti_flag_clear</code>	344
Table 3-456. Function <code>exti_interrupt_flag_get</code>	344
Table 3-457. Function <code>exti_interrupt_flag_clear</code>	345
Table 3-458. FMC Registers	345
Table 3-459. FMC firmware function	346
Table 3-460. <code>fmc_state_enum</code>	348
Table 3-461. <code>efuse_state_enum</code>	348
Table 3-462. Function <code>fmc_unlock</code>	348
Table 3-463. Function <code>fmc_lock</code>	349
Table 3-464. Function <code>fmc_page_erase</code>	349
Table 3-465. Function <code>fmc_sector_erase</code>	350
Table 3-466. Function <code>fmc_mass_erase</code>	350
Table 3-467. Function <code>fmc_bank0_erase</code>	351
Table 3-468. Function <code>fmc_bank1_erase</code>	351
Table 3-469. Function <code>fmc_doubleword_program</code>	352
Table 3-470. Function <code>fmc_word_program</code>	352
Table 3-471. Function <code>fmc_halfword_program</code>	353
Table 3-472. Function <code>fmc_halfword_program</code>	354
Table 3-473. Function <code>fmc_nwa_enable</code>	354
Table 3-474. Function <code>fmc_nwa_disable</code>	355
Table 3-475. Function <code>otp1_read_disable</code>	355
Table 3-476. Function <code>otp2_rlock_enable</code>	356
Table 3-477. Function <code>ob_unlock</code>	356
Table 3-478. Function <code>ob_lock</code>	357
Table 3-479. Function <code>ob_start</code>	357

Table 3-480. Function ob_erase	358
Table 3-481. Function ob_write_protection_enable	358
Table 3-482. Function ob_write_protection_disable	359
Table 3-483. Function ob_drp_enable.....	360
Table 3-484. Function ob_drp_disable.....	360
Table 3-485. Function ob_security_protection_config	361
Table 3-486. Function ob_user_write	362
Table 3-487. Function ob_user_bor_threshold	362
Table 3-488. Function ob_boot_mode_config	363
Table 3-489. Function ob_ecc_config	364
Table 3-490. Function ob_nwa_select.....	364
Table 3-491. Function ob_user_get	365
Table 3-492. Function ob_write_protection0_get	365
Table 3-493. Function ob_write_protection1_get	366
Table 3-494. Function ob_drp0_get	366
Table 3-495. Function ob_drp0_get	367
Table 3-496. Function ob_spc_get.....	368
Table 3-497. Function ob_user_bor_threshold_get	368
Table 3-498. Function ob_boot_mode_get.....	369
Table 3-499. Function ob_ecc_get.....	369
Table 3-500. Function ob_nwa_get.....	370
Table 3-501. Function fmc_flag_get	370
Table 3-502. Function fmc_flag_clear	371
Table 3-503. Function fmc_interrupt_enable	371
Table 3-504. Function fmc_interrupt_disable	372
Table 3-505. Function fmc_interrupt_flag_get.....	373
Table 3-506. Function fmc_interrupt_flag_clear	373
Table 3-507. Function fmc_state_get	374
Table 3-508. Function fmc_ready_wait	375
Table 3-509. Function ob_unlock.....	375
Table 3-510. Function ob_lock	376
Table 3-511. Function efuse_user_data_unlock.....	376
Table 3-512. Function efuse_user_data_lock.....	377
Table 3-513. Function efuse_read	377
Table 3-514. Function efuse_write.....	378
Table 3-515. Function efuse_control_write.....	378
Table 3-516. Function efuse_user_data_write.....	379
Table 3-517. Function efuse_flag_get.....	379
Table 3-518. Function efuse_flag_clear	380
Table 3-519. Function efuse_interrupt_enable.....	380
Table 3-520. Function efuse_interrupt_disable.....	381
Table 3-521. Function efuse_interrupt_flag_get	381
Table 3-522. Function efuse_interrupt_flag_clear	382
Table 3-523. Function efuse_ready_wait	383

Table 3-524. FWDGT Registers	383
Table 3-525. FWDGT firmware function	384
Table 3-526. Function fwdgt_write_enable	384
Table 3-527. Function fwdgt_write_disable	384
Table 3-528. Function fwdgt_enable	385
Table 3-529. Function fwdgt_prescaler_value_config	385
Table 3-530. Function fwdgt_reload_value_config	386
Table 3-531. Function fwdgt_counter_reload	387
Table 3-532. Function fwdgt_config	387
Table 3-533. Function fwdgt_flag_get fwdgt_write_disable	388
Table 3-534. GPIO Registers	389
Table 3-535. GPIO firmware function	390
Table 3-536. Function gpio_deinit	390
Table 3-537. Function gpio_mode_set	391
Table 3-538. Function gpio_output_options_set	392
Table 3-539. Function gpio_bit_set	393
Table 3-540. Function gpio_bit_reset	393
Table 3-541. Function gpio_bit_write	394
Table 3-542. Function gpio_port_write	394
Table 3-543. Function gpio_input_bit_get	395
Table 3-544. Function gpio_input_port_get	396
Table 3-545. Function gpio_output_bit_get	396
Table 3-546. Function gpio_output_port_get	397
Table 3-547. Function gpio_pin_remap_config	397
Table 3-548. Function gpio_pin_lock	398
Table 3-549. Function gpio_bit_toggle	399
Table 3-550. Function gpio_port_toggle	399
Table 3-551. HAU Registers	400
Table 3-552. HAU firmware function	401
Table 3-553. Structure hau_init_parameter_struct	402
Table 3-554. Structure hau_digest_parameter_struct	402
Table 3-555. Structure hau_context_parameter_struct	402
Table 3-556. Function hau_deinit	402
Table 3-557. Function hau_init	403
Table 3-558. Function hau_init_struct_para_init	403
Table 3-559. Function hau_reset	404
Table 3-560. Function hau_last_word_validbits_num_config	404
Table 3-561. Function hau_data_write	405
Table 3-562. Function hau_infifo_words_num_get	405
Table 3-563. Function hau_digest_read	406
Table 3-564. Function hau_digest_calculation_enable	406
Table 3-565. Function hau_multiple_single_dma_config	407
Table 3-566. Function hau_dma_enable	408
Table 3-567. Function hau_dma_disable	408

Table 3-568. Function hau_context_struct_para_init.....	408
Table 3-569. Function hau_context_save.....	409
Table 3-570. Function hau_context_restore	410
Table 3-571. Function hau_hash_sha_1.....	410
Table 3-572. Function hau_hmac_sha_1.....	411
Table 3-573. Function hau_hash_sha_224.....	412
Table 3-574. Function hau_hmac_sha_224.....	412
Table 3-575. Function hau_hash_sha_256.....	413
Table 3-576. Function hau_hmac_sha_256.....	414
Table 3-577. Function hau_hash_md5.....	415
Table 3-578. Function hau_hmac_md5.....	415
Table 3-579. Function hau_flag_get.....	416
Table 3-580. Function hau_flag_clear.....	417
Table 3-581. Function hau_interrupt_enable	417
Table 3-582. Function hau_interrupt_disable.....	418
Table 3-583. Function hau_interrupt_flag_get	419
Table 3-584. Function hau_interrupt_flag_clear	419
Table 3-585. I2C0-2 Registers.....	421
Table 3-586. I2C3-5 Registers.....	421
Table 3-587. I2C0-2 firmware function	422
Table 3-588. I2C3-5 firmware function	423
Table 3-589. Function i2c_deinit.....	424
Table 3-590. Function i2c_clock_config.....	425
Table 3-591. Function i2c_mode_addr_config	426
Table 3-592. Function i2c_smbus_type_config	426
Table 3-593. Function i2c_ack_config	427
Table 3-594. Function i2c_ackpos_config	428
Table 3-595. Function i2c_master_addressing	428
Table 3-596. Function i2c_dualaddr_enable	429
Table 3-597. Function i2c_dualaddr_disable.....	429
Table 3-598. Function i2c_enable	430
Table 3-599. Function i2c_disable	430
Table 3-600. Function i2c_start_on_bus	431
Table 3-601. Function i2c_stop_on_bus	431
Table 3-602. Function i2c_data_transmit	432
Table 3-603. Function i2c_data_receive	433
Table 3-604. Function i2c_dma_config.....	433
Table 3-605. Function i2c_dma_last_transfer_enable	434
Table 3-606. Function i2c_rbne_clear_config	434
Table 3-607. Function i2c_stretch_scl_low_config	435
Table 3-608. Function i2c_slave_response_to_gcall_config.....	435
Table 3-609. Function i2c_software_reset_config	436
Table 3-610. Function i2c_pec_config	437
Table 3-611. Function i2c_pec_transfer_config	437

Table 3-612. Function i2c_pec_value_get	438
Table 3-613. Function i2c_smbus_alert_config	438
Table 3-614. Function i2c_smbus_arp_config	439
Table 3-615. Function i2c_analog_noise_filter_disable	440
Table 3-616. Function i2c_analog_noise_filter_enable	440
Table 3-617. Function i2c_digital_noise_filter_config	441
Table 3-618. Function i2c_sam_enable	441
Table 3-619. Function i2c_sam_disable	442
Table 3-620. Function i2c_sam_timeout_enable	442
Table 3-621. Function i2c_sam_timeout_disable	443
Table 3-622. Function i2c_flag_get	443
Table 3-623. Function i2c_flag_clear	444
Table 3-624. Function i2c_interrupt_enable	445
Table 3-625. Function i2c_interrupt_disable	446
Table 3-626. Function i2c_interrupt_flag_get	447
Table 3-627. Function i2c_interrupt_flag_clear	448
Table 3-628. i2c_add_deinit	449
Table3-629. i2c_add_timing_config	449
Table3-630. i2c_add_digital_noise_filter_config	450
Table3-631. i2c_add_analog_noise_filter_enable	451
Table3-632. i2c_add_analog_noise_filter_disable	451
Table3-633. i2c_add_master_clock_config	452
Table3-634. i2c_add_master_addressing	453
Table3-635. i2c_add_address10_header_enable	453
Table3-636. i2c_add_address10_header_disable	454
Table3-637. i2c_add_address10_enable	454
Table3-638. i2c_address10_disable	455
Table3-639. i2c_add_automatic_end_enable	455
Table3-640. i2c_add_automatic_end_disable	456
Table3-641. i2c_slave_response_to_gcall_enable	456
Table3-642. i2c_add_slave_response_to_gcall_disable	457
Table3-643. i2c_add_stretch_scl_low_enable	457
Table3-644. i2c_add_stretch_scl_low_disable	458
Table3-645. i2c_add_address_config	458
Table3-646. i2c_add_address_bit_compare_config	459
Table3-647. i2c_add_address_disable	460
Table3-648. i2c_add_second_address_config	460
Table3-649. i2c_add_second_address_disable	461
Table3-650. i2c_add_receved_address_get	462
Table3-651. i2c_add_slave_byte_control_enable	462
Table3-652. i2c_add_slave_byte_control_disable	463
Table3-653. i2c_add_nack_enable	463
Table3-654. i2c_add_wakeup_from_deepsleep_enable	464
Table3-655. i2c_add_wakeup_from_deepsleep_disable	464

Table3-656. i2c_add_enable	465
Table3-657. i2c_add_disable	465
Table3-658. i2c_add_start_on_bus	466
Table3-659. i2c_add_stop_on_bus.....	466
Table3-660. i2c_add_data_transmit	467
Table3-661. i2c_add_data_receive	467
Table3-662. i2c_add_reload_enable.....	468
Table3-663. i2c_add_reload_disable.....	469
Table3-664. i2c_add_transfer_byte_number_config.....	469
Table3-665. i2c_add_dma_enable	470
Table3-666. i2c_add_dma_disable	470
Table3-667. i2c_add_pec_transfer.....	471
Table3-668. i2c_add_pec_enable.....	471
Table3-669. i2c_add_pec_disable.....	472
Table3-670. i2c_add_pec_value_get	472
Table3-671. i2c_add_smbus_alert_enable	473
Table3-672. i2c_add_smbus_alert_disable	473
Table3-673. i2c_add_smbus_default_addr_enable	474
Table3-674. i2c_add_smbus_default_addr_disable	474
Table3-675. i2c_add_smbus_host_addr_enable.....	475
Table3-676. i2c_add_smbus_host_addr_disable	475
Table3-677. i2c_add_extented_clock_timeout_enable	476
Table3-678. i2c_add_extented_clock_timeout_disable	476
Table3-679. i2c_add_clock_timeout_enable.....	477
Table3-680. i2c_add_clock_timeout_disable.....	477
Table3-681. i2c_add_bus_timeout_b_config	478
Table3-682. i2c_add_bus_timeout_a_config	478
Table3-683. i2c_add_idle_clock_timeout_config.....	479
Table3-684. i2c_add_flag_get.....	480
Table3-685. i2c_add_flag_clear.....	481
Table3-686. i2c_add_interrupt_enable.....	482
Table3-687. i2c_add_interrupt_disable.....	483
Table3-688. i2c_add_interrupt_flag_get.....	483
Table3-689. i2c_add_interrupt_flag_clear	484
Table 3-690. IPA Registers	486
Table 3-691. IPA firmware function.....	486
Table 3-692. Structure ipa_foreground_parameter_struct.....	487
Table 3-693. Structure ipa_background_parameter_struct	488
Table 3-694. Structure ipa_destination_parameter_struct.....	488
Table 3-695. Function ipa_deinit.....	488
Table 3-696. Function ipa_transfer_enable.....	489
Table 3-697. Function ipa_transfer_hangup_enable	489
Table 3-698. Function ipa_transfer_hangup_disable	490
Table 3-699. Function ipa_transfer_stop_enable.....	490

Table 3-700. Function ipa_transfer_stop_disable	491
Table 3-701. Function ipa_foreground_lut_loading_enable	491
Table 3-702. Function ipa_background_lut_loading_enable.....	492
Table 3-703. Function ipa_pixel_format_convert_mode_set.....	492
Table 3-704. Function ipa_foreground_struct_para_init	493
Table 3-705. Function ipa_foreground_init.....	493
Table 3-706. Function ipa_background_struct_para_init.....	494
Table 3-707. Function ipa_background_init	495
Table 3-708. Function ipa_destination_struct_para_init	495
Table 3-709. Function ipa_destination_init.....	496
Table 3-710. Function ipa_foreground_lut_init	497
Table 3-711. Function ipa_background_lut_init.....	498
Table 3-712. Function ipa_line_mark_config.....	498
Table 3-713. Function ipa_inter_timer_config.....	499
Table 3-714. Function ipa_interval_clock_num_config	500
Table 3-715. Function ipa_flag_get	500
Table 3-716. Function ipa_flag_clear	501
Table 3-717. Function ipa_interrupt_enable	501
Table 3-718. Function ipa_interrupt_disable	502
Table 3-719. Function ipa_interrupt_flag_get.....	503
Table 3-720. Function ipa_interrupt_flag_clear	503
Table 3-721. IREF Registers	504
Table 3-722. IREF firmware function	504
Table 3-723. Function iref_deinit	504
Table 3-724. Function iref_enable	505
Table 3-725. Function iref_disable	505
Table 3-726. Function iref_mode_set.....	506
Table 3-727. Function iref_sink_set	507
Table 3-728. Function iref_precision_trim_value_set.....	507
Table 3-729. Function iref_step_data_config	508
Table 3-730. NVIC Registers	508
Table 3-731. SysTick Registers	509
Table 3-732. Enum IRQn_Type	510
Table 3-733. MISC firmware function	512
Table 3-734. Function nvic_priority_group_set	513
Table 3-735. Function nvic_irq_enable.....	513
Table 3-736. Function nvic_irq_disable.....	514
Table 3-737. Function nvic_vector_table_set.....	514
Table 3-738. Function system_lowpower_set	515
Table 3-739. Function system_lowpower_reset.....	515
Table 3-740. Function systick_clksource_set	516
Table 3-741. PKCAU Registers.....	517
Table 3-742. PKCAU firmware function	517
Table 3-743. Structure pkcau_mont_parameter_struct	518

Table 3-744. Structure pkcau_mod_parameter_struct	518
Table 3-745. Structure pkcau_mod_exp_parameter_struct	518
Table 3-746. Structure pkcau_arithmetic_parameter_struct	519
Table 3-747. Structure pkcau_crt_parameter_struct	519
Table 3-748. Structure pkcau_ec_group_parameter_struct	519
Table 3-749. Structure pkcau_point_parameter_struct	520
Table 3-750. Structure pkcau_signature_parameter_struct	520
Table 3-751. Structure pkcau_hash_parameter_struct	520
Table 3-752. Structure pkcau_ecc_out_struct	520
Table 3-753. Function pkcau_deinit	521
Table 3-754. Function pkcau_mont_struct_para_init	521
Table 3-755. Function pkcau_mod_struct_para_init	522
Table 3-756. Function pkcau_mod_exp_struct_para_init	522
Table 3-757. Function pkcau_arithmetic_struct_para_init	523
Table 3-758. Function pkcau_crt_struct_para_init	523
Table 3-759. Function pkcau_ec_group_struct_para_init	524
Table 3-760. Function pkcau_point_struct_para_init	524
Table 3-761. Function pkcau_signature_struct_para_init	525
Table 3-762. Function pkcau_hash_struct_para_init	526
Table 3-763. Function pkcau_ecc_out_struct_para_init	526
Table 3-764. Function pkcau_enable	527
Table 3-765. Function pkcau_disable	527
Table 3-766. Function pkcau_start	528
Table 3-767. Function pkcau_mode_set	528
Table 3-768. Function pkcau_mont_param_operation	529
Table 3-769. Function pkcau_mod_operation	530
Table 3-770. Function pkcau_mod_exp_operation	531
Table 3-771. Function pkcau_mod_inver_operation	532
Table 3-772. Function pkcau_mod_reduc_operation	533
Table 3-773. Function pkcau_arithmetic_operation	534
Table 3-774. Function pkcau_crt_exp_operation	535
Table 3-775. Function pkcau_point_check_operation	536
Table 3-776. Function pkcau_point_mul_operation	537
Table 3-777. Function pkcau_ecdsa_sign_operation	539
Table 3-778. Function pkcau_ecdsa_verification_operation	540
Table 3-779. Function pkcau_flag_get	543
Table 3-780. Function pkcau_flag_clear	543
Table 3-781. Function pkcau_interrupt_enable	544
Table 3-782. Function pkcau_interrupt_disable	544
Table 3-783. Function pkcau_interrupt_flag_get	545
Table 3-784. Function pkcau_interrupt_flag_clear	546
Table 3-785. PMU Registers	546
Table 3-786. PMU firmware function	547
Table 3-787. Function pmu_deinit	547

Table 3-788. Function pmu_lvd_select	548
Table 3-789. Function pmu_lvd_disable.....	548
Table 3-790. Function pmu_ldo_output_select.....	549
Table 3-791. Function pmu_highdriver_mode_enable	549
Table 3-792. Function pmu_highdriver_mode_disable	550
Table 3-793. Function pmu_highdriver_switch_select.....	550
Table 3-794. Function pmu_lowdriver_mode_enable	551
Table 3-795. Function pmu_lowdriver_mode_disable	551
Table 3-796. Function pmu_lowpower_driver_config.....	552
Table 3-797. Function pmu_normalpower_driver_config	552
Table 3-798. Function pmu_to_sleepmode.....	553
Table 3-799. Function pmu_to_deepsleepmode	554
Table 3-800. Function pmu_to_standbymode.....	554
Table 3-801. Function pmu_wakeup_pin_enable	555
Table 3-802. Function pmu_wakeup_pin_disable	555
Table 3-803. Function pmu_backup_ldo_config.....	556
Table 3-804. Function pmu_backup_write_enable.....	556
Table 3-805. Function pmu_backup_write_disable.....	557
Table 3-806. Function pmu_flag_get.....	557
Table 3-807. Function pmu_flag_clear.....	558
Table 3-808. RCU Registers	559
Table 3-809. RCU firmware function	560
Table 3-810. Enum rcu_periph_enum	561
Table 3-811. Enum rcu_periph_sleep_enum.....	562
Table 3-812. Enum rcu_periph_reset_enum.....	563
Table 3-813. Enum rcu_flag_enum.....	564
Table 3-814. Enum rcu_int_flag_enum	565
Table 3-815. Enum rcu_int_flag_clear_enum	565
Table 3-816. Enum rcu_int_enum	566
Table 3-817. Enum rcu_osci_type_enum	566
Table 3-818. Enum rcu_clock_freq_enum.....	566
Table 3-819. Enum i2c_idx_enum.....	566
Table 3-820. Function rcu_deinit	567
Table 3-821. Function rcu_periph_clock_enable	567
Table 3-822. Function rcu_periph_clock_disable.....	568
Table 3-823. Function rcu_periph_clock_sleep_enable	568
Table 3-824. Function rcu_periph_clock_sleep_disable	568
Table 3-825. Function rcu_periph_reset_enable.....	569
Table 3-826. Function rcu_periph_reset_disable	569
Table 3-827. Function rcu_bkp_reset_enable	570
Table 3-828. Function rcu_bkp_reset_disable	570
Table 3-829. Function rcu_system_clock_source_config.....	571
Table 3-830. Function rcu_system_clock_source_get	572
Table 3-831. Function rcu_ahb_clock_config	572

Table 3-832. Function rcu_apb1_clock_config	573
Table 3-833. Function rcu_apb2_clock_config	573
Table 3-834. Function rcu_ckout0_config	574
Table 3-835. Function rcu_ckout1_config	575
Table 3-836. Function rcu_pll_config	576
Table 3-837. Function rcu_plli2s_config	576
Table 3-838. Function rcu_plli2s_r_config	577
Table 3-839. Function rcu_pllsai_p_config	578
Table 3-840. Function rcu_pllsai_q_config	578
Table 3-841. Function rcu_pllsai_r_config	579
Table 3-842. Function rcu_rtc_clock_config	579
Table 3-843. Function rcu_rtc_div_config	580
Table 3-844. Function rcu_i2s_clock_config	580
Table 3-845. Function rcu_i2c_clock_config	581
Table 3-846. Function rcu_sai_clock_config	582
Table 3-847. Function rcu_ck48m_clock_config	582
Table 3-848. Function rcu_pll48m_clock_config	583
Table 3-849. Function rcu_timer_clock_prescaler_config	583
Table 3-850. Function rcu_tli_clock_div_config	584
Table 3-851. Function rcu_lxtal_drive_capability_config	585
Table 3-852. Function rcu_osci_stab_wait	585
Table 3-853. Function rcu_osci_on	586
Table 3-854. Function rcu_osci_off	586
Table 3-855. Function rcu_osci_bypass_mode_enable	587
Table 3-856. Function rcu_osci_bypass_mode_disable	587
Table 3-857. Function rcu_hxtal_clock_monitor_enable	588
Table 3-858. Function rcu_hxtal_clock_monitor_disable	588
Table 3-859. Function rcu_irc16m_adjust_value_set	589
Table 3-860. Function rcu_spread_spectrum_config	589
Table 3-861. Function rcu_spread_spectrum_enable	590
Table 3-862. Function rcu_spread_spectrum_disable	590
Table 3-863. Function rcu_voltage_key_unlock	591
Table 3-864. Function rcu_deepsleep_voltage_set	591
Table 3-865. Function rcu_clock_freq_get	592
Table 3-866. Function rcu_flag_get	592
Table 3-867. Function rcu_all_reset_flag_clear	593
Table 3-868. Function rcu_interrupt_flag_get	593
Table 3-869. Function rcu_interrupt_flag_clear	594
Table 3-870. Function rcu_interrupt_enable	594
Table 3-871. Function rcu_interrupt_disable	595
Table 3-872. RTC Registers	596
Table 3-873. RTC firmware function	596
Table 3-874. Struct rtc_parameter_struct	597
Table 3-875. Struct rtc_alarm_struct	598

Table 3-876. Struct rtc_timestamp_struct	598
Table 3-877. Struct rtc_tamper_struct	598
Table 3-878. Function rtc_deinit	599
Table 3-879. Function rtc_init.....	599
Table 3-880. Function rtc_init_mode_enter	600
Table 3-881. Function rtc_init_mode_exit.....	600
Table 3-882. Function rtc_register_sync_wait	601
Table 3-883. Function rtc_current_time_get.....	601
Table 3-884. Function rtc_subsecond_get.....	602
Table 3-885. Function rtc_alarm_config.....	602
Table 3-886. Function rtc_alarm_subsecond_config.....	603
Table 3-887. Function rtc_alarm_get.....	604
Table 3-888. Function rtc_alarm_subsecond_get	605
Table 3-889. Function rtc_alarm_enable	605
Table 3-890. Function rtc_alarm_disable	606
Table 3-891. Function rtc_timestamp_enable	606
Table 3-892. Function rtc_timestamp_disable	607
Table 3-893. Function rtc_timestamp_get.....	607
Table 3-894. Function rtc_timestamp_subsecond_get.....	608
Table 3-895. Function rtc_timestamp_pin_map.....	608
Table 3-896. Function rtc_timestamp_enable	609
Table 3-897. Function rtc_tamper_disable.....	609
Table 3-898. Function rtc_tamper0_pin_map	610
Table 3-899. Function rtc_interrupt_enable	610
Table 3-900. Function rtc_interrupt_disable.....	611
Table 3-901. Function rtc_flag_get.....	612
Table 3-902. Function rtc_flag_clear.....	612
Table 3-903. Function rtc_alter_output_config	613
Table 3-904. Function rtc_calibration_output_config	614
Table 3-905. Function rtc_hour_adjust.....	614
Table 3-906. Function rtc_second_adjust.....	615
Table 3-907. Function rtc_bypass_shadow_enable	616
Table 3-908. Function rtc_bypass_shadow_disable	616
Table 3-909. Function rtc_refclock_detection_enable	617
Table 3-910. Function rtc_refclock_detection_disable.....	617
Table 3-911. Function rtc_wakeup_enable.....	617
Table 3-912. Function rtc_wakeup_disable	618
Table 3-913. Function rtc_wakeup_clock_set	618
Table 3-914. Function rtc_wakeup_timer_set.....	619
Table 3-915. Function rtc_wakeup_timer_get	620
Table 3-916. Function rtc_smooth_calibration_config	620
Table 3-917. Function rtc_coarse_calibration_enable	621
Table 3-918. Function rtc_coarse_calibration_disable.....	621
Table 3-919. Function rtc_coarse_calibration_config	622

Table 3-920. SDIO Registers	623
Table 3-921. SDIO firmware function	623
Table 3-922. Function sdio_deinit	625
Table 3-923. Function sdio_clock_config	625
Table 3-924. Function sdio_hardware_clock_enable	626
Table 3-925. Function sdio_hardware_clock_disable	627
Table 3-926. Function sdio_bus_mode_set	627
Table 3-927. Function sdio_power_state_set	628
Table 3-928. Function sdio_power_state_get	628
Table 3-929. Function sdio_clock_enable	629
Table 3-930. Function sdio_clock_disable	629
Table 3-931. Function sdio_command_response_config	630
Table 3-932. Function sdio_wait_type_set	630
Table 3-933. Function sdio_csm_enable	631
Table 3-934. Function sdio_csm_disable	631
Table 3-935. Function sdio_command_index_get	632
Table 3-936. Function sdio_response_get	632
Table 3-937. Function sdio_data_config	633
Table 3-938. Function sdio_data_transfer_config	634
Table 3-939. Function sdio_dsm_enable	635
Table 3-940. Function sdio_dsm_disable	636
Table 3-941. Function sdio_data_write	636
Table 3-942. Function sdio_data_read	636
Table 3-943. Function sdio_data_counter_get	637
Table 3-944. Function sdio_data_counter_get	637
Table 3-945. Function sdio_dma_enable	638
Table 3-946. Function sdio_dma_disable	638
Table 3-947. Function sdio_flag_get	639
Table 3-948. Function sdio_flag_clear	640
Table 3-949. Function sdio_interrupt_enable	641
Table 3-950. Function sdio_interrupt_disable	642
Table 3-951. Function sdio_interrupt_flag_get	643
Table 3-952. Function sdio_interrupt_flag_clear	645
Table 3-953. Function sdio_readwait_enable	646
Table 3-954. Function sdio_readwait_disable	646
Table 3-955. Function sdio_stop_readwait_enable	647
Table 3-956. Function sdio_stop_readwait_disable	647
Table 3-957. Function sdio_readwait_type_set	648
Table 3-958. Function sdio_operation_enable	648
Table 3-959. Function sdio_operation_disable	649
Table 3-960. Function sdio_suspend_enable	649
Table 3-961. Function sdio_suspend_disable	650
Table 3-962. Function sdio_ceata_command_enable	650
Table 3-963. Function sdio_ceata_command_disable	651

Table 3-964. Function sdio_ceata_interrupt_enable	651
Table 3-965. Function sdio_ceata_interrupt_disable	652
Table 3-966. Function sdio_ceata_command_completion_enable	652
Table 3-967. Function sdio_ceata_command_completion_disable	652
Table 3-968. SPI/I2S Registers	653
Table 3-969. SPI/I2S firmware function.....	654
Table 3-970. spi_parameter_struct.....	655
Table 3-971. Function spi_i2s_deinit	655
Table 3-972. Function spi_struct_para_init	656
Table 3-973. Function spi_init	656
Table 3-974. Function spi_enable	657
Table 3-975. Function spi_disable	658
Table 3-976. Function i2s_init	658
Table 3-977. Function i2s_psc_config	659
Table 3-978. Function i2s_enable	660
Table 3-979. Function i2s_disable	661
Table 3-980. Function spi_nss_output_enable	661
Table 3-981. Function spi_nss_output_disable	662
Table 3-982. Function spi_nss_internal_high	662
Table 3-983. Function spi_nss_internal_low	663
Table 3-984. Function spi_dma_enable	663
Table 3-985. Function spi_dma_disable.....	664
Table 3-986. Function spi_i2s_data_frame_format_config	665
Table 3-987. Function spi_i2s_data_transmit.....	665
Table 3-988. Function spi_i2s_data_receive	666
Table 3-989. Function spi_bidirectional_transfer_config.....	666
Table 3-990. Function i2s_full_duplex_mode_config	667
Table 3-991. Function spi_i2s_format_error_clear.....	668
Table 3-992. Function spi_crc_polynomial_set.....	669
Table 3-993. Function spi_crc_polynomial_get	669
Table 3-994. Function spi_crc_on.....	670
Table 3-995. Function spi_crc_off	670
Table 3-996. Function spi_crc_next	671
Table 3-997. Function spi_crc_get.....	671
Table 3-998. Function spi_crc_error_clear	672
Table 3-999. Function spi_ti_mode_enable	673
Table 3-1000. Function spi_ti_mode_disable	673
Table 3-1001. Function spi_quad_enable.....	674
Table 3-1002. Function spi_quad_disable	674
Table 3-1003. Function spi_quad_write_enable.....	675
Table 3-1004. Function spi_quad_read_enable	675
Table 3-1005. Function spi_quad_io23_output_enable	676
Table 3-1006. Function spi_quad_io23_output_disable.....	676
Table 3-1007. Function spi_i2s_flag_get	677

Table 3-1008. Function spi_i2s_interrupt_enable.....	677
Table 3-1009. Function spi_i2s_interrupt_disable.....	678
Table 3-1010. Function spi_i2s_interrupt_flag_get	679
Table 3-1011. SYSCFG registers	680
Table 3-1012.SYSCFG firmware function	680
Table 3-1013. Function syscfg_deinit	681
Table 3-1014. Function syscfg_bootmode_config	681
Table 3-1015. Function syscfg_fmc_swap_config	682
Table 3-1016. Function syscfg_exmc_swap_config	683
Table 3-1017. Function syscfg_exti_line_config.....	684
Table 3-1018. Function syscfg_enet_phy_interface_config	684
Table 3-1019. Function syscfg_compensation_config	685
Table 3-1020. Function syscfg_flag_get.....	686
Table 3-1021. TIMERx Registers	687
Table 3-1022.TIMERx firmware function.....	688
Table 3-1023. Structure timer_parameter_struct	691
Table 3-1024. Structure timer_break_parameter_struct.....	692
Table 3-1025. Structure timer_oc_parameter_struct.....	692
Table 3-1026. Structure timer_ic_parameter_struct.....	693
Table 3-1027. Function timer_deinit.....	693
Table 3-1028. Function timer_struct_para_init.....	694
Table 3-1029. Function timer_init	694
Table 3-1030. Function timer_enable	695
Table 3-1031. Function timer_disable	696
Table 3-1032. Function timer_auto_reload_shadow_enable	697
Table 3-1033. Function timer_auto_reload_shadow_disable	697
Table 3-1034. Function timer_update_event_enable	698
Table 3-1035. Function timer_update_event_disable	699
Table 3-1036. Function timer_counter_alignment	699
Table 3-1037. Function timer_counter_up_direction	700
Table 3-1038. Function timer_counter_down_direction	701
Table 3-1039. Function timer_prescaler_config.....	702
Table 3-1040. Function timer_repetition_value_config	702
Table 3-1041. Function timer_autoreload_value_config	703
Table 3-1042. Function timer_counter_value_config.....	704
Table 3-1043. Function timer_counter_read	705
Table 3-1044. Function timer_prescaler_read	705
Table 3-1045. Function timer_single_pulse_mode_config	706
Table 3-1046. Function timer_update_source_config.....	707
Table 3-1047. Function timer_dma_enable	708
Table 3-1048. Function timer_dma_disable	709
Table 3-1049. Function timer_channel_dma_request_source_select.....	710
Table 3-1050. Function timer_dma_transfer_config	710
Table 3-1051. Function timer_event_software_generate	712

Table 3-1052. Function timer_break_struct_para_init	714
Table 3-1053. Function timer_break_config	714
Table 3-1054. Function timer_break_enable	715
Table 3-1055. Function timer_break_disable	716
Table 3-1056. Function timer_automatic_output_enable	717
Table 3-1057. Function timer_automatic_output_disable	717
Table 3-1058. Function timer_primary_output_config	718
Table 3-1059. Function timer_channel_control_shadow_config	719
Table 3-1060. Function timer_channel_control_shadow_update_config	720
Table 3-1061. Function timer_channel_output_struct_para_init	720
Table 3-1062. Function timer_channel_output_config	721
Table 3-1063. Function timer_channel_output_mode_config	722
Table 3-1064. Function timer_channel_output_pulse_value_config	724
Table 3-1065. Function timer_channel_output_shadow_config	725
Table 3-1066. Function timer_channel_output_fast_config	726
Table 3-1067. Function timer_channel_output_clear_config	727
Table 3-1068. Function timer_channel_output_polarity_config	728
Table 3-1069. Function timer_channel_complementary_output_polarity_config	729
Table 3-1070. Function timer_channel_output_state_config	730
Table 3-1071. Function timer_channel_complementary_output_state_config	731
Table 3-1072. Function timer_channel_input_struct_para_init	732
Table 3-1073. Function timer_input_capture_config	733
Table 3-1074. Function timer_channel_input_capture_prescaler_config	734
Table 3-1075. Function timer_channel_capture_value_register_read	735
Table 3-1076. Function timer_input_pwm_capture_config	736
Table 3-1077. Function timer_hall_mode_config	737
Table 3-1078. Function timer_input_trigger_source_select	738
Table 3-1079. Function timer_master_output_trigger_source_select	739
Table 3-1080. Function timer_slave_mode_select	740
Table 3-1081. Function timer_master_slave_mode_config	741
Table 3-1082. Function timer_external_trigger_config	742
Table 3-1083. Function timer_quadrature_decoder_mode_config	743
Table 3-1084. Function timer_internal_clock_config	745
Table 3-1085. Function timer_internal_trigger_as_external_clock_config	745
Table 3-1086. Function timer_external_trigger_as_external_clock_config	746
Table 3-1087. Function timer_external_clock_mode0_config	747
Table 3-1088. Function timer_external_clock_mode1_config	749
Table 3-1089. Function timer_external_clock_mode1_disable	750
Table 3-1090. Function timer_channel_remap_config	751
Table 3-1091. Function timer_write_chxval_register_config	752
Table 3-1092. Function timer_output_value_selection_config	753
Table 3-1093. Function timer_channel_composite_pwm_enable	754
Table 3-1094. Function timer_channel_composite_pwm_disable	754
Table 3-1095. Function timer_channel_additional_compare_value_config	755

Table 3-1096. Function timer_channel_additional_output_shadow_config	756
Table 3-1097. Function timer_channel_additional_compare_value_read	757
Table 3-1098. Function timer_flag_get	758
Table 3-1099. Function timer_flag_clear	759
Table 3-1100. Function timer_interrupt_enable	761
Table 3-1101. Function timer_interrupt_disable	762
Table 3-1102. Function timer_interrupt_flag_get	763
Table 3-1103. Function timer_interrupt_flag_clear	764
Table 3-1104. TLI Registers	765
Table 3-1105. TLI firmware function	766
Table 3-1106. Structure tli_parameter_struct	766
Table 3-1107. Structure tli_layer_parameter_struct	767
Table 3-1108. Structure tli_layer_lut_parameter_struct	767
Table 3-1109. Function tli_deinit	768
Table 3-1110. Function tli_struct_para_init	768
Table 3-1111. Function tli_init	769
Table 3-1112. Function tli_dither_config	770
Table 3-1113. Function tli_enable	770
Table 3-1114. Function tli_disable	771
Table 3-1115. Function tli_reload_config	771
Table 3-1116. Function tli_layer_struct_para_init	772
Table 3-1117. Function tli_layer_init	772
Table 3-1118. Function tli_layer_window_offset_modify	774
Table 3-1119. Function tli_lut_struct_para_init	774
Table 3-1120. Function tli_lut_init	775
Table 3-1121. Function tli_color_key_init	776
Table 3-1122. Function tli_layer_enable	776
Table 3-1123. Function tli_layer_disable	777
Table 3-1124. Function tli_color_key_enable	777
Table 3-1125. Function tli_color_key_disable	778
Table 3-1126. Function tli_lut_enable	778
Table 3-1127. Function tli_lut_disable	779
Table 3-1128. Function tli_line_mark_set	779
Table 3-1129. Function tli_current_pos_get	780
Table 3-1130. Function tli_interrupt_enable	780
Table 3-1131. Function tli_interrupt_disable	781
Table 3-1132. Function tli_interrupt_flag_get	781
Table 3-1133. Function tli_interrupt_flag_clear	782
Table 3-1134. Function tli_flag_get	783
Table 3-1135. TRNG Registers	783
Table 3-1136. TRNG firmware function	784
Table 3-1137. Function trng_deinit	784
Table 3-1138. Function trng_enable	784
Table 3-1139. Function trng_disable	785

Table 3-1140 Function trng_get_true_random_data	785
Table 3-1141 Function trng_flag_get.....	786
Table 3-1142 Function trng_interrupt_enable	786
Table 3-1143 Function trng_interrupt_disable	787
Table 3-1144 Function trng_interrupt_flag_get	787
Table 3-1145 Function trng_interrupt_flag_clear	788
Table 3-1146. USART Registers	789
Table 3-1147. USART firmware function.....	789
Table 3-1148. Enum usart_flag_enum.....	791
Table 3-1149. Enum usart_interrupt_flag_enum	791
Table 3-1150. Enum usart_interrupt_enum	792
Table 3-1151. Enum usart_invert_enum	792
Table 3-1152. Function usart_deinit.....	792
Table 3-1153. Function usart_baudrate_set.....	793
Table 3-1154. Function usart_parity_config	793
Table 3-1155. Function usart_word_length_set	794
Table 3-1156. Function usart_stop_bit_set.....	795
Table 3-1157. Function usart_enable	795
Table 3-1158. Function usart_disable	796
Table 3-1159. Function usart_transmit_config.....	796
Table 3-1160. Function usart_receive_config.....	797
Table 3-1161. Function usart_data_first_config.....	798
Table 3-1162. Function usart_invert_config	798
Table 3-1163. Function usart_oversample_config.....	799
Table 3-1164. Function usart_sample_bit_config	800
Table 3-1165. Function usart_receiver_timeout_enable.....	800
Table 3-1166. Function usart_receiver_timeout_disable.....	801
Table 3-1167. Function usart_receiver_timeout_threshold_config	801
Table 3-1168. Function usart_data_transmit	802
Table 3-1169. Function usart_data_receive	802
Table 3-1170. Function usart_address_config	803
Table 3-1171. Function usart_mute_mode_enable.....	804
Table 3-1172. Function usart_mute_mode_disable.....	804
Table 3-1173. Function usart_mute_mode_wakeup_config	805
Table 3-1174. Function usart_lin_mode_enable	805
Table 3-1175. Function usart_lin_mode_disable	806
Table 3-1176. Function usart_lin_break_dection_length_config	806
Table 3-1177. Function usart_send_break	807
Table 3-1178. Function usart_halfduplex_enable	807
Table 3-1179. Function usart_halfduplex_disable	808
Table 3-1180. Function usart_synchronous_clock_enable	809
Table 3-1181. Function usart_synchronous_clock_disable	809
Table 3-1182. Function usart_synchronous_clock_config	810
Table 3-1183. Function usart_guard_time_config	810

Table 3-1184. Function <code>usart_smartcard_mode_enable</code>	811
Table 3-1185. Function <code>usart_smartcard_mode_disable</code>	811
Table 3-1186. Function <code>usart_smartcard_mode_nack_enable</code>	812
Table 3-1187. Function <code>usart_smartcard_mode_nack_disable</code>	812
Table 3-1188. Function <code>usart_smartcard_autoretry_config</code>	813
Table 3-1189. Function <code>usart_block_length_config</code>	814
Table 3-1190. Function <code>usart_irda_mode_enable</code>	814
Table 3-1191. Function <code>usart_irda_mode_disable</code>	815
Table 3-1192. Function <code>usart_prescaler_config</code>	815
Table 3-1193. Function <code>usart_irda_lowpower_config</code>	816
Table 3-1194. Function <code>usart_hardware_flow_rts_config</code>	816
Table 3-1195. Function <code>usart_hardware_flow_cts_config</code>	817
Table 3-1196. Function <code>usart_break_frame_coherence_config</code>	818
Table 3-1197. Function <code>usart_parity_check_coherence_config</code>	818
Table 3-1198. Function <code>usart_hardware_flow_coherence_config</code>	819
Table 3-1199. Function <code>usart_dma_receive_config</code>	819
Table 3-1200. Function <code>usart_dma_transmit_config</code>	820
Table 3-1201. Function <code>usart_flag_get</code>	821
Table 3-1202. Function <code>usart_flag_clear</code>	821
Table 3-1203. Function <code>usart_interrupt_enable</code>	822
Table 3-1204. Function <code>usart_interrupt_disable</code>	823
Table 3-1205. Function <code>usart_interrupt_flag_get</code>	823
Table 3-1206. Function <code>usart_interrupt_flag_clear</code>	824
Table 3-1207. WWDGT Registers	825
Table 3-1208. WWDGT firmware function	825
Table 3-1209. Function <code>wwdgt_deinit</code>	825
Table 3-1210. Function <code>wwdgt_enable</code>	826
Table 3-1211. Function <code>wwdgt_counter_update</code>	826
Table 3-1212. Function <code>wwdgt_config</code>	827
Table 3-1213. Function <code>wwdgt_flag_get</code>	828
Table 3-1214. Function <code>wwdgt_flag_clear</code>	829
Table 3-1215. Function <code>wwdgt_interrupt_enable</code>	830
Table 3-1216. SAI Registers.....	830
Table 3-1217. SAI firmware function	831
Table 3-1218. Structure <code>sai_parameter_struct</code>	832
Table 3-1219. Structure <code>sai_frame_parameter_struct</code>	832
Table 3-1220. Structure <code>sai_slot_parameter_struct</code>	832
Table 3-1221. Enum <code>sai_fifo_state_enum</code>	832
Table 3-1222. Function <code>sai_deinit</code>	833
Table 3-1223. Function <code>sai_struct_para_init</code>	833
Table 3-1224. Function <code>sai_frame_struct_para_init</code>	834
Table 3-1225. Function <code>sai_slot_struct_para_init</code>	834
Table 3-1226. Function <code>sai_init</code>	835
Table 3-1227. Function <code>sai_frame_init</code>	836

Table 3-1228. Function sai_slot_init	836
Table 3-1229. Function sai_enable	837
Table 3-1230. Function sai_disable	837
Table 3-1231. Function sai_sdoutput_config	838
Table 3-1232. Function sai_monomode_config	839
Table 3-1233. Function sai_companing_config	839
Table 3-1234. Function sai_mute_enable	840
Table 3-1235. Function sai_mute_disable	841
Table 3-1236. Function sai_mute_value_config	841
Table 3-1237. Function sai_mute_count_config	842
Table 3-1238. Function sai_data_transmit	842
Table 3-1239. Function sai_data_receive	843
Table 3-1240. Function sai_fifo_status_get	843
Table 3-1241. Function sai_fifo_flush	844
Table 3-1242. Function sai_dma_enable	844
Table 3-1243. Function sai_dma_disable	845
Table 3-1244. Function sai_interrupt_enable	845
Table 3-1245. Function sai_interrupt_disable	846
Table 3-1246. Function sai_interrupt_flag_get	847
Table 3-1247. Function sai_interrupt_flag_clear	847
Table 3-1248. Function sai_flag_get	848
Table 3-1249. Function sai_flag_clear	849
Table 4-1. Revision history	850

1. Introduction

This manual introduces firmware library of GD32F527 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 GD32F527 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 1-1. Peripherals

Peripherals	Descriptions
ADC	Analog-to-digital converter
CAN	Controller area network
CAU	Cryptographic acceleration unit
CRC	CRC calculation unit
CTC	Clock trim controller
DAC	Digital-to-analog converter

Peripherals	Descriptions
DBG	Debug
DCI	Digital camera interface
DMA	Direct memory access controller
ENET	Ethernet
EXMC	External memory controller
EXTI	Interrupt/event controller
FMC	Flash memory controller
FWDGT	Free watchdog timer
GPIO/AFIO	General-purpose and alternate-function I/Os
HAU	Hash acceleration unit
I2C	Inter-integrated circuit interface
IPA	Image processing accelerator
IREF	Programmable current reference
MISC	Nested Vectored Interrupt Controller
PKCAU	Public key cryptographic acceleration unit
PMU	Power management unit
RCU	Reset and clock unit
RTC	Real-time Clock
SAI	Serial audio interface
SDIO	Secure digital input/output interface
SPI/I2S	Serial peripheral interface/Inter-IC sound
SYSCFG	System configuration
TIMER	TIMER
TLI	TFT-LCD interface
TRNG	True random number generator
USART	Universal synchronous/asynchronous receiver /transmitter
WWDGT	Window watchdog timer
USBFS	Universal serial bus full-speed interface
USBHS	Universal serial bus high-speed interface

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 sourcefile and header file are started with “gd32f527_”, such as: gd32f527_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 uppcase of English letters;
- Registers are handled as constants. The naming of them are written in uppcase of English letters. In most cases, register names are shortened accord with the user manual;

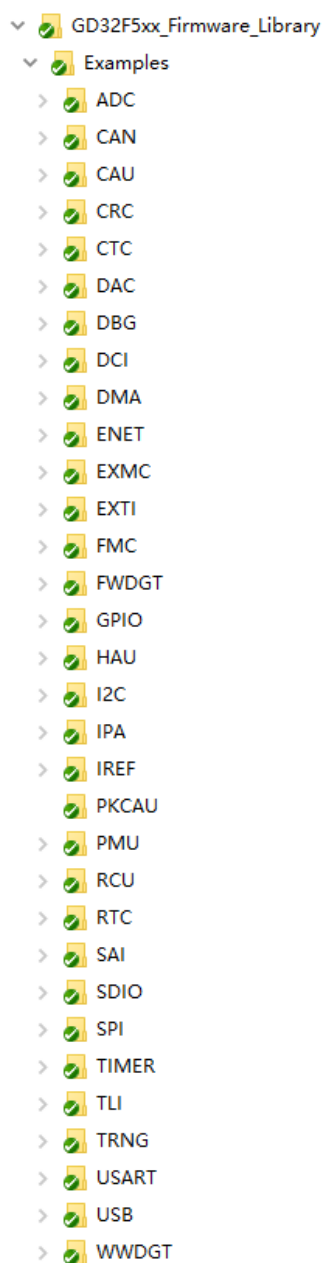
- Variables are written in lowercase, when concluded by several words, underlines should be adapted among words;
- The naming of peripheral functions are started with the peripheral abbreviation added with an underline, when the function name is concluded by several words, underlines should be adapted among words, and all the peripheral functions are written in lowercase.












2. Firmware Library Overview

2.1. File Structure of Firmware Library

GD32F527_Firmware_Library, the file structure is shown as below:

Figure 2-1. File structure of firmware library of GD32F527



- ▼  Firmware
 - >  CMSIS
 - >  GD32F5xx_standard_peripheral
 - >  GD32F5xx_usb_library
- ▼  Template
 - >  IAR_project
 -  Keil_project
- ▼  Utilities
 -  Binary
 -  LCD_common
 - >  Third_Party

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;
- gd32f527_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);
- gd32f527_it.c: the source file include all the interrupt service routines (if no interrupt is used, then all the function bodies are empty);
- gd32f527_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 M33 kernel support files, the startup file based on the Cortex M33 kernel processor, the global header file of GD32F527 and system configuration file;
- GD32F527_standard_peripheral subfolder:
 - Include subfolder includes all the header files of firmware library, users need not modify this folder;
 - Source subfolder includes all the source files of firmware library, users need not modify this folder;
- GD32F527_usb_driver subfolder includes all the related files about USB peripheral:

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

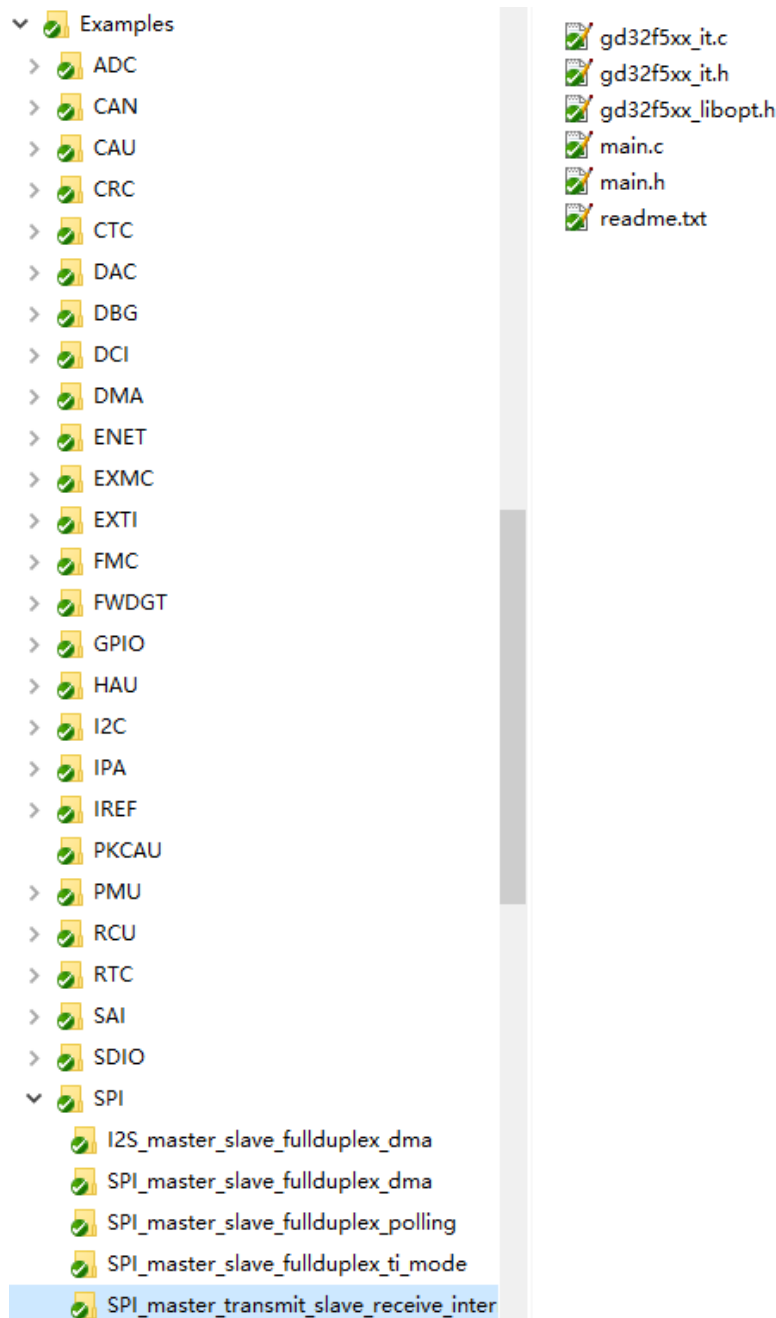
2.1.3. Template Folder

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 formware examples, the steps are shown as below:

Select files

Open “Examples” folder, select the module to be tested, such as SPI, open “SPI” folder, select an example of SPI, such as “SPI_master_transmit_slave_receive_interrupt”, shown as below:

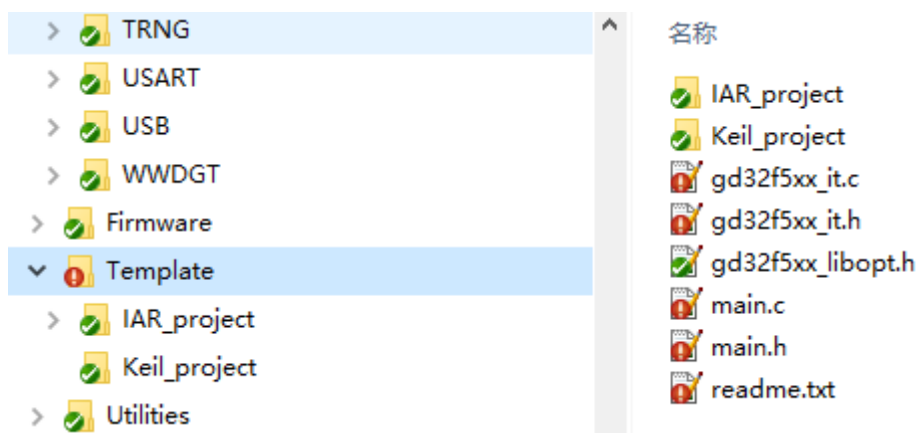
Figure 2-2. Select peripheral example files



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:

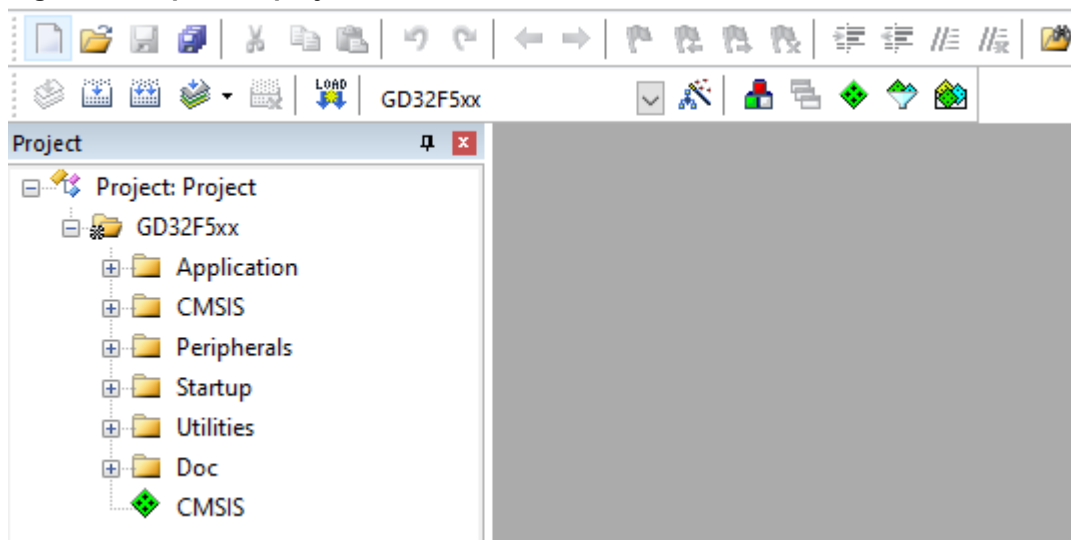
Figure 2-3. Copy the peripheral example files



Open a project

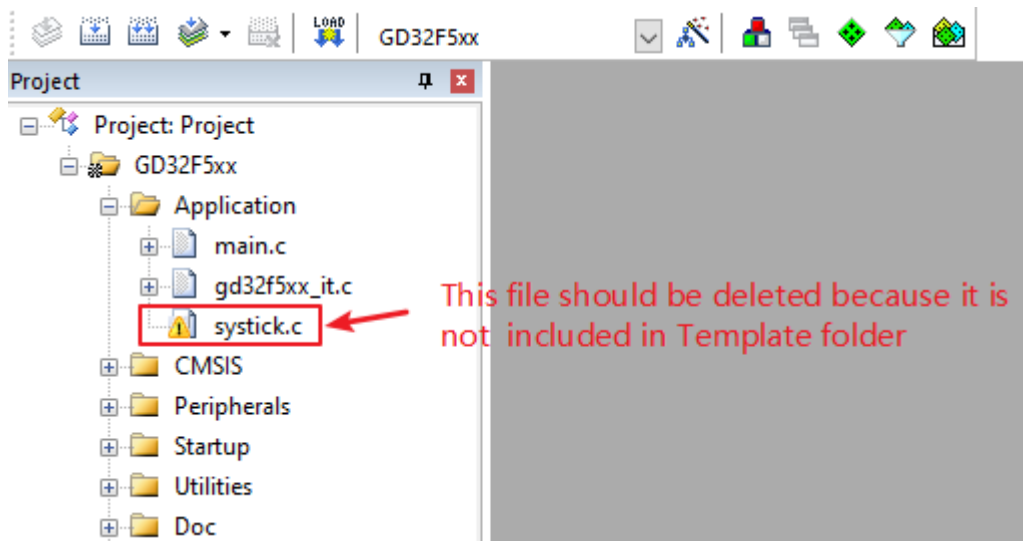
GD provides project in Keil and IAR, users can open project in different IDEs according to their need, such as "Keil_project", open \Template\Keil_project\Project.uvproj, shown as below:

Figure 2-4. Open the project file



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:

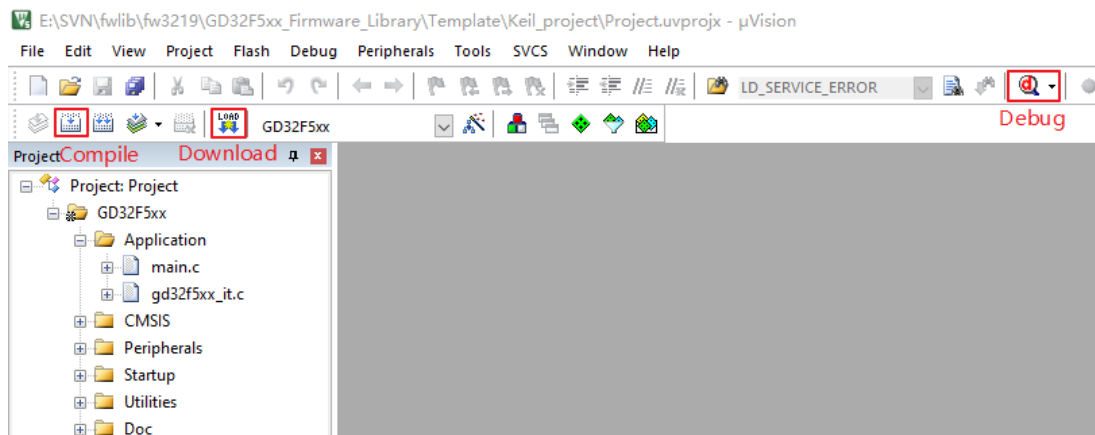
Figure 2-5. Configure project files



Compile-Debug-Download

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:

Figure 2-6. Compile-debug-download



2.1.4. Utilities Folder

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

- Binary, Third_Party subfolders include files for USB tests;
- gd32f527_eval.h is related header files of the evaluation board about running the firmware examples;
- gd32f527_eval.c is related source files 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 2-1. Function descriptions of Firmware Library

Files	Descriptions
gd32f527_libopt.h	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.
main.c	Example of main function.
gd32f527_it.h	Header file, including all the prototypes of interrupt service routines.
gd32f527_it.c	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.
gd32f527_xxx.h	The header file of peripheral PPP, including functions about peripheral PPP, and the variables used for functions.
gd32f527_xxx.c	The C source file for driving peripheral PPP.
systick.h	The header file of systick.c, including prototypes of systick configuration function and delay function.
systick.c	The source file about systick configuration function and delay function.
readme.txt	Description document about how to configure and how to use the firmware example.

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 3-1. Peripheral function format of Firmware Library

Function name	Name of peripheral function
Function prototype	Declaration prototype
Function descriptions	Explain the function how to work
Precondition	Requirements should meet before calling this function
The called functions	Other firmware functions called in this function
Input parameter{in}	
Input parameter name	Description
xxxx	Description of input parameters
Output parameter{out}	
Output parameter name	Description
xxxx	Description of output parameters
Return value	
Return value type	The range of return value

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 3-2. ADC Registers

Registers	Descriptions
ADC_STAT	Status register
ADC_CTL0	Control register 0
ADC_CTL1	Control register 1
ADC_SAMPT0	Sample time register 0
ADC_SAMPT1	Sample time register 1
ADC_IOFFx (x=0..3)	Inserted channel data offset register x(x=0..3)
ADC_WDHT	Watchdog high threshold register
ADC_WDLT	Watchdog low threshold register

Registers	Descriptions
ADC_RSQ0	Routine sequence register 0
ADC_RSQ1	Routine sequence register 1
ADC_RSQ2	Routine sequence register 2
ADC_ISQ	Inserted sequence register
ADC_IDATAx	Inserted data register x(x=0..3)
ADC_RDATA	Routine data register
ADC_OVSAMPCTL	Oversample control register
ADC_SSTAT	Summary status register
ADC_SYNCCTL	Sync control register
ADC_SYNCDATA	Sync routine data register

3.2.2. Descriptions of Peripheral functions

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

Table 3-3. ADC firmware function

Function name	Function description
adc_deinit	reset ADCx peripheral
adc_clock_config	configure the ADC clock for all the ADCs
adc_special_function_config	enable or disable ADC special function
adc_data_alignment_config	configure ADC data alignment
adc_enable	enable ADC interface
adc_disable	disable ADC interface
adc_calibration_enable	ADC calibration and reset calibration
adc_channel_16_to_18	configure temperature sensor and internal reference voltage channel or VBAT channel function
adc_resolution_config	configure ADC resolution
adc_oversample_mode_config	configure ADC oversample mode
adc_oversample_mode_enable	enable ADC oversample mode
adc_oversample_mode_disable	disable ADC oversample mode
adc_dma_mode_enable	enable DMA request
adc_dma_mode_disable	disable DMA request
adc_dma_request_after_last_enable	when DMA=1, the DMA engine issues a request at end of each routine conversion
adc_dma_request_after_last_disable	the DMA engine is disabled after the end of transfer signal from DMA controller is detected
adc_discontinuous_mode_config	configure ADC discontinuous mode
adc_channel_length_config	configure the length of routine channel group or inserted channel group
adc_routine_channel_config	configure ADC routine channel
adc_inserted_channel_config	configure ADC inserted channel
adc_inserted_channel_offset_config	configure ADC inserted channel offset
adc_external_trigger_source_config	configure ADC external trigger source

Function name	Function description
adc_external_trigger_config	enable ADC external trigger
adc_software_trigger_enable	enable ADC software trigger
adc_end_of_conversion_config	configure end of conversion mode
adc_routine_data_read	read ADC routine group data register
adc_inserted_data_read	read ADC inserted group data register
adc_watchdog_single_channel_disable	disable ADC analog watchdog single channel
adc_watchdog_single_channel_enable	configure ADC analog watchdog single channel
adc_watchdog_group_channel_enable	configure ADC analog watchdog group channel
adc_watchdog_disable	disable ADC analog watchdog
adc_watchdog_threshold_config	configure ADC analog watchdog threshold
adc_sync_mode_config	configure the ADC sync mode
adc_sync_delay_config	configure the delay between 2 sampling phases in ADC sync modes
adc_sync_dma_config	configure ADC sync DMA mode selection
adc_sync_dma_request_after_last_enable	when SYNC DMA is not equal to 2'b00, the DMA engine issues requests according to the SYNC DMA bits
adc_sync_dma_request_after_last_disable	the DMA engine is disabled after the end of transfer signal from DMA controller is detected
adc_sync_routine_data_read	read ADC sync routine data register
adc_routine_software_startconv_flag_get	get the bit state of ADCx software start conversion
adc_inserted_software_startconv_flag_get	get the bit state of ADCx software inserted channel start conversion
adc_flag_get	get the ADC flag bits
adc_flag_clear	clear the ADC flag bits
adc_interrupt_enable	enable ADC interrupt
adc_interrupt_disable	disable ADC interrupt
adc_interrupt_flag_get	get the ADC interrupt bits
adc_interrupt_flag_clear	clear the ADC flag

adc_deinit

The description of adc_deinit is shown as below:

Table 3-4. Function adc_deinit

Function name	adc_deinit
Function prototype	void adc_deinit(void);
Function descriptions	reset ADC peripheral
Precondition	-
The called functions	rcu_periph_reset_enable / rcu_periph_reset_disable

Input parameter{in}	
-	-
Output parameter{out}	
-	-
Return value	
-	-

Example:

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

```
adc_deinit();
```

adc_clock_config

The description of adc_clock_config is shown as below:

Table 3-5. Function adc_clock_config

Function name	adc_clock_config
Function prototype	void adc_clock_config(uint32_t prescaler);
Function descriptions	configure the ADC clock for all the ADCs
Precondition	-
The called functions	-
Input parameter{in}	
prescaler	configure ADCs prescaler ratio
ADC_ADCCCK_PCLK2_DIV2	PCLK2 div2
ADC_ADCCCK_PCLK2_DIV4	PCLK2 div4
ADC_ADCCCK_PCLK2_DIV6	PCLK2 div6
ADC_ADCCCK_PCLK2_DIV8	PCLK2 div8
ADC_ADCCCK_HCLK_DIV5	HCLK div5
ADC_ADCCCK_HCLK_DIV6	HCLK div6
ADC_ADCCCK_HCLK_DIV10	HCLK div10
ADC_ADCCCK_HCLK_DIV20	HCLK div20
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* configure the ADC clock: PCLK2 div2 */
adc_clock_config (ADC_ADCCCK_PCLK2_DIV2);
```

adc_special_function_config

The description of adc_special_function_config is shown as below:

Table 3-6. Function adc_special_function_config

Function name	adc_special_function_config
Function prototype	void adc_special_function_config(uint32_t adc_periph, uint32_t function, ControlStatus newvalue);
Function descriptions	enable or disable ADC special function
Precondition	-
The called functions	-
Input parameter{in}	
adc_periph	ADC peripheral
ADCx(x=0..2)	ADC peripheral selection
Input parameter{in}	
function	the function to config
ADC_SCAN_MODE	scan mode select
ADC_INSERTED_CHANNEL_AUTO	inserted channel group convert automatically
ADC_CONTINUOUS_MODE	continuous mode select
Input parameter{in}	
newvalue	control value
ENABLE	enable function
DISABLE	disable function
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* enable ADC0 scan mode */
adc_special_function_config(ADC0, ADC_SCAN_MODE, ENABLE);
```

adc_data_alignment_config

The description of adc_data_alignment_config is shown as below:

Table 3-7. Function adc_data_alignment_config

Function name	adc_data_alignment_config
----------------------	---------------------------

Function prototype	void adc_data_alignment_config(uint32_t adc_periph, uint32_t data_alignment);
Function descriptions	configure ADCx data alignment
Precondition	-
The called functions	-
Input parameter{in}	
adc_periph	ADC peripheral
<i>ADCx(x=0..2)</i>	ADC peripheral selection
Input parameter{in}	
data_alignment	data alignment select
<i>ADC_DATAALIGN_RIGHT</i>	LSB alignment
<i>ADC_DATAALIGN_LEFT</i>	MSB alignment
Output parameter{out}	
-	-
Return value	
-	-

Example:

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

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

adc_enable

The description of adc_enable is shown as below:

Table 3-8. Function adc_enable

Function name	adc_enable
Function prototype	void adc_enable(uint32_t adc_periph);
Function descriptions	enable ADCx interface
Precondition	-
The called functions	-
Input parameter{in}	
adc_periph	ADC peripheral
<i>ADCx(x=0..2)</i>	ADC peripheral selection
Output parameter{out}	
-	-
Return value	
-	-

Example:

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

```
adc_enable(ADC0);
```

adc_disable

The description of adc_disable is shown as below:

Table 3-9. Function adc_disable

Function name	adc_disable
Function prototype	void adc_disable(uint32_t adc_periph);
Function descriptions	disable ADCx interface
Precondition	-
The called functions	-
Input parameter{in}	
adc_periph	ADC peripheral
<i>ADCx(x=0..2)</i>	ADC peripheral selection
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* disable ADC0 */
```

```
adc_disable(ADC0);
```

adc_calibration_enable

The description of adc_calibration_enable is shown as below:

Table 3-10. Function adc_calibration_enable

Function name	adc_calibration_enable
Function prototype	void adc_calibration_enable(uint32_t adc_periph);
Function descriptions	ADCx calibration and reset calibration
Precondition	-
The called functions	-
Input parameter{in}	
adc_periph	ADC peripheral
<i>ADCx(x=0..2)</i>	ADC peripheral selection
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* ADC0 calibration and reset calibration */
```

```
adc_calibration_enable(ADC0);
```

adc_channel_16_to_18

The description of adc_channel_16_to_18 is shown as below:

Table 3-11. Function adc_channel_16_to_18

Function name	adc_channel_16_to_18
Function prototype	void adc_channel_16_to_18(uint32_t function, ControlStatus newvalue);
Function descriptions	configure temperature sensor and internal reference voltage channel or VBAT channel function
Precondition	-
The called functions	-
Input parameter{in}	
function	temperature sensor and internal reference voltage channel or VBAT channel
ADC_VBAT_CHANNEL_SWITCH	channel 18 (1/4 voltate of external battery) switch of ADC0
ADC_TEMP_VREF_CHANNEL_SWITCH	channel 16 (temperature sensor) and 17 (internal reference voltage) switch of ADC0
Input parameter{in}	
newvalue	control value
ENABLE	enable function
DISABLE	disable function
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* enable the temperature sensor and Vrefint channel */
```

```
adc_channel_16_to_18 (ADC_TEMP_VREF_CHANNEL_SWITCH, ENABLE);
```

adc_resolution_config

The description of adc_resolution_config is shown as below:

Table 3-12. Function adc_resolution_config

Function name	adc_resolution_config
Function prototype	void adc_resolution_config(uint32_t adc_periph , uint32_t resolution);
Function descriptions	configure ADC resolution
Precondition	-
The called functions	-
Input parameter{in}	

adc_periph	ADC peripheral
<i>ADCx(x=0..2)</i>	ADC peripheral selection
Input parameter{in}	
resolution	ADC resolution
<i>ADC_RESOLUTION_12B</i>	12-bit ADC resolution
<i>ADC_RESOLUTION_10B</i>	10-bit ADC resolution
<i>ADC_RESOLUTION_8B</i>	8-bit ADC resolution
<i>ADC_RESOLUTION_6B</i>	6-bit ADC resolution
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* configure ADC0 resolution: 10 bits */
```

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

adc_oversample_mode_config

The description of `adc_oversample_mode_config` is shown as below:

Table 3-13. Function `adc_oversample_mode_config`

Function name	<code>adc_oversample_mode_config</code>
Function prototype	<code>void adc_oversample_mode_config(uint32_t adc_periph, uint32_t mode, uint16_t shift, uint8_t ratio);</code>
Function descriptions	configure ADC oversample mode
Precondition	-
The called functions	-
Input parameter{in}	
adc_periph	ADC peripheral
<i>ADCx(x=0..2)</i>	ADC peripheral selection
Input parameter{in}	
mode	ADC oversampling mode
<i>ADC_OVERSAMPLING_ALL_CONVERT</i>	all oversampled conversions for a channel are done consecutively after a trigger
<i>ADC_OVERSAMPLING_ONE_CONVERT</i>	each oversampled conversion for a channel needs a trigger
Input parameter{in}	
shift	ADC oversampling shift

ADC_OVERSAMPLING _SHIFT_NONE	no oversampling shift
ADC_OVERSAMPLING _SHIFT_1B	1-bit oversampling shift
ADC_OVERSAMPLING _SHIFT_2B	2-bit oversampling shift
ADC_OVERSAMPLING _SHIFT_3B	3-bit oversampling shift
ADC_OVERSAMPLING _SHIFT_4B	4-bit oversampling shift
ADC_OVERSAMPLING _SHIFT_5B	5-bit oversampling shift
ADC_OVERSAMPLING _SHIFT_6B	6-bit oversampling shift
ADC_OVERSAMPLING _SHIFT_7B	7-bit oversampling shift
ADC_OVERSAMPLING _SHIFT_8B	8-bit oversampling shift
Input parameter{in}	
ratio	ADC oversampling ratio
ADC_OVERSAMPLING _RATIO_MUL2	oversampling ratio multiple 2
ADC_OVERSAMPLING _RATIO_MUL4	oversampling ratio multiple 4
ADC_OVERSAMPLING _RATIO_MUL8	oversampling ratio multiple 8
ADC_OVERSAMPLING _RATIO_MUL16	oversampling ratio multiple 16
ADC_OVERSAMPLING _RATIO_MUL32	oversampling ratio multiple 32
ADC_OVERSAMPLING _RATIO_MUL64	oversampling ratio multiple 64
ADC_OVERSAMPLING _RATIO_MUL128	oversampling ratio multiple 128
ADC_OVERSAMPLING _RATIO_MUL256	oversampling ratio multiple 256
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* configure ADC1 oversample mode: 16 times sample, 4 bits shift */
```

```
adc_oversample_mode_config(ADC1, ADC_OVERSAMPLING_ALL_CONVERT,
ADC_OVERSAMPLING_SHIFT_4B, ADC_OVERSAMPLING_RATIO_MUL16);
```

adc_oversample_mode_enable

The description of adc_oversample_mode_enable is shown as below:

Table 3-14. Function adc_oversample_mode_enable

Function name	adc_oversample_mode_enable
Function prototype	void adc_oversample_mode_enable(uint32_t adc_periph);
Function descriptions	enable ADC oversample mode
Precondition	-
The called functions	-
Input parameter{in}	
adc_periph	ADC peripheral
<i>ADCx(x=0..2)</i>	ADC peripheral selection
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* enable ADC0 oversample mode */
adc_oversample_mode_enable (ADC0);
```

adc_oversample_mode_disable

The description of adc_oversample_mode_disable is shown as below:

Table 3-15. Function adc_oversample_mode_disable

Function name	adc_oversample_mode_disable
Function prototype	void adc_oversample_mode_disable(uint32_t adc_periph);
Function descriptions	disable ADC oversample mode
Precondition	-
The called functions	-
Input parameter{in}	
adc_periph	ADC peripheral
<i>ADCx(x=0..2)</i>	ADC peripheral selection
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* disable ADC0 oversample mode */
```

adc_oversample_mode_disable (ADC0);

adc_dma_mode_enable

The description of adc_dma_mode_enable is shown as below:

Table 3-16. Function adc_dma_mode_enable

Function name	adc_dma_mode_enable
Function prototype	void adc_dma_mode_enable(uint32_t adc_periph);
Function descriptions	enable ADCx DMA request
Precondition	-
The called functions	-
Input parameter{in}	
adc_periph	ADC peripheral
<i>ADCx(x=0..2)</i>	ADC peripheral selection
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* enable ADC0 DMA request */
adc_dma_mode_enable(ADC0);
```

adc_dma_mode_disable

The description of adc_dma_mode_disable is shown as below:

Table 3-17. Function adc_dma_mode_disable

Function name	adc_dma_mode_disable
Function prototype	void adc_dma_mode_disable(uint32_t adc_periph);
Function descriptions	disable ADCx DMA request
Precondition	-
The called functions	-
Input parameter{in}	
adc_periph	ADC peripheral
<i>ADCx(x=0..2)</i>	ADC peripheral selection
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* disable ADC0 DMA request */
```



```
adc_dma_mode_disable(ADC0);
```

adc_dma_request_after_last_enable

The description of `adc_dma_request_after_last_enable` is shown as below:

Table 3-18. Function `adc_dma_request_after_last_enable`

Function name	<code>adc_dma_request_after_last_enable</code>
Function prototype	<code>void adc_dma_request_after_last_enable(uint32_t adc_periph);</code>
Function descriptions	when DMA=1, the DMA engine issues a request at end of each routine conversion
Precondition	-
The called functions	-
Input parameter{in}	
adc_periph	ADC peripheral
<i>ADCx(x=0..2)</i>	ADC peripheral selection
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* when DMA=1, the DMA engine issues a request at end of each routine conversion for ADC0
*/
```

```
adc_dma_request_after_last_enable (ADC0);
```

adc_dma_request_after_last_disable

The description of `adc_dma_request_after_last_disable` is shown as below:

Table 3-19. Function `adc_dma_mode_disable`

Function name	<code>adc_dma_request_after_last_disable</code>
Function prototype	<code>void adc_dma_request_after_last_disable(uint32_t adc_periph);</code>
Function descriptions	the DMA engine is disabled after the end of transfer signal from DMA controller is detected
Precondition	-
The called functions	-
Input parameter{in}	
adc_periph	ADC peripheral
<i>ADCx(x=0..2)</i>	ADC peripheral selection
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* the DMA engine is disabled after the end of transfer signal from DMA controller is detected
for ADC0 */
```

```
adc_dma_request_after_last_disable (ADC0);
```

adc_discontinuous_mode_config

The description of adc_discontinuous_mode_config is shown as below:

Table 3-20. Function adc_discontinuous_mode_config

Function name	adc_discontinuous_mode_config
Function prototype	void adc_discontinuous_mode_config(uint32_t adc_periph, uint8_t adc_channel_group, uint8_t length);
Function descriptions	configure ADC discontinuous mode
Precondition	-
The called functions	-
Input parameter{in}	
adc_periph	ADC peripheral
ADCx(x=0..2)	ADC peripheral selection
Input parameter{in}	
adc_channel_group	select the channel group
ADC_ROUTINE_CHANNEL	routine channel group
ADC_INSERTED_CHANNEL	inserted channel group
ADC_CHANNEL_DISCON_DISABLE	disable discontinuous mode of routine and inserted channel
Input parameter{in}	
length	number of conversions in discontinuous mode, the number can be 1..8 for routine channel, the number has no effect for inserted channel
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* configure ADC0 routine channel group discontinuous mode */
```

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

adc_channel_length_config

The description of adc_channel_length_config is shown as below:

Table 3-21. Function `adc_channel_length_config`

Function name	<code>adc_channel_length_config</code>
Function prototype	<code>void adc_channel_length_config(uint32_t adc_periph, uint8_t adc_channel_group, uint32_t length);</code>
Function descriptions	configure the length of routine channel group or inserted channel group
Precondition	-
The called functions	-
Input parameter{in}	
<code>adc_periph</code>	ADC peripheral
<code>ADCx(x=0..2)</code>	ADC peripheral selection
Input parameter{in}	
<code>adc_channel_group</code>	select the channel group
<code>ADC_ROUTINE_CHANNEL</code>	routine channel group
<code>ADC_INSERTED_CHANNEL</code>	inserted channel group
Input parameter{in}	
<code>length</code>	the length of the channel, routine channel 1-16, inserted channel 1-4
Output parameter{out}	
-	-
Return value	
-	-

Example:

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

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

`adc_routine_channel_config`

The description of `adc_routine_channel_config` is shown as below:

Table 3-22. Function `adc_routine_channel_config`

Function name	<code>adc_routine_channel_config</code>
Function prototype	<code>void adc_routine_channel_config(uint32_t adc_periph, uint8_t rank, uint8_t adc_channel, uint32_t sample_time);</code>
Function descriptions	configure ADC routine channel
Precondition	-
The called functions	-
Input parameter{in}	
<code>adc_periph</code>	ADC peripheral
<code>ADCx(x=0..2)</code>	ADC peripheral selection
Input parameter{in}	
<code>rank</code>	the routine group sequence rank, this parameter must be between 0 to 15
Input parameter{in}	

adc_channel	the selected ADC channel
<i>ADC_CHANNEL_x(x=0 ..18)</i>	ADC Channelx (x=0..17)(x=16..18 are only for ADC0)
Input parameter{in}	
sample_time	the sample time value
<i>ADC_SAMPLETIME_3</i>	3cycles
<i>ADC_SAMPLETIME_1</i> 5	15cycles
<i>ADC_SAMPLETIME_2</i> 8	28 cycles
<i>ADC_SAMPLETIME_5</i> 6	56 cycles
<i>ADC_SAMPLETIME_8</i> 4	84cycles
<i>ADC_SAMPLETIME_1</i> 12	112 cycles
<i>ADC_SAMPLETIME_1</i> 44	144cycles
<i>ADC_SAMPLETIME_4</i> 80	480 cycles
Output parameter{out}	
-	-
Return value	
-	-

Example:

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

```
adc_routine_channel_config(ADC0, 1, ADC_CHANNEL_0, ADC_SAMPLETIME_56);
```

adc_inserted_channel_config

The description of adc_inserted_channel_config is shown as below:

Table 3-23. Function adc_inserted_channel_config

Function name	adc_inserted_channel_config
Function prototype	void adc_inserted_channel_config(uint32_t adc_periph, uint8_t rank, uint8_t adc_channel, uint32_t sample_time);
Function descriptions	configure ADC inserted channel
Precondition	-
The called functions	-
Input parameter{in}	
adc_periph	ADC peripheral
<i>ADCx(x=0..2)</i>	ADC peripheral selection
Input parameter{in}	

rank	the inserted group sequencer rank, this parameter must be between 0 to 3
Input parameter{in}	
adc_channel	the selected ADC channel
ADC_CHANNEL_x(x=0 ..18)	ADC Channelx (x=0..17)(x=16..18 are only for ADC0)
Input parameter{in}	
sample_time	the sample time value
ADC_SAMPLETIME_3	3 cycles
ADC_SAMPLETIME_1 5	15 cycles
ADC_SAMPLETIME_2 8	28 cycles
ADC_SAMPLETIME_5 6	56 cycles
ADC_SAMPLETIME_8 4	48 cycles
ADC_SAMPLETIME_1 12	112 cycles
ADC_SAMPLETIME_1 44	144 cycles
ADC_SAMPLETIME_4 80	480 cycles
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* configure ADC0 inserted channel */
```

```
adc_inserted_channel_config(ADC0, 1, ADC_CHANNEL_0, ADC_SAMPLETIME_56);
```

adc_inserted_channel_offset_config

The description of adc_inserted_channel_offset_config is shown as below:

Table 3-24. Function adc_inserted_channel_offset_config

Function name	adc_inserted_channel_offset_config
Function prototype	void adc_inserted_channel_offset_config(uint32_t adc_periph, uint8_t inserted_channel, uint16_t offset);
Function descriptions	configure ADC inserted channel offset
Precondition	-
The called functions	-
Input parameter{in}	
adc_periph	ADC peripheral

<i>ADCx(x=0..2)</i>	ADC peripheral selection
Input parameter{in}	
inserted_channel	insert channel select
<i>ADC_INSERTED_CHANNEL_x(x=0..3)</i>	inserted channel, x=0,1,2,3
Input parameter{in}	
offset	the offset data, this parameter must be between 0 to 4095
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* configure ADC0 inserted channel offset */
```

```
adc_inserted_channel_offset_config(ADC0, ADC_INSERTED_CHANNEL_0, 100);
```

adc_external_trigger_source_config

The description of `adc_external_trigger_source_config` is shown as below:

Table 3-25. Function `adc_external_trigger_source_config`

Function name	<code>adc_external_trigger_source_config</code>
Function prototype	<code>void adc_external_trigger_source_config(uint32_t adc_periph, uint8_t adc_channel_group, uint32_t external_trigger_source);</code>
Function descriptions	configure ADC external trigger source
Precondition	-
The called functions	-
Input parameter{in}	
adc_periph	ADC peripheral
<i>ADCx(x=0..2)</i>	ADC peripheral selection
Input parameter{in}	
adc_channel_group	select the channel group
<i>ADC_ROUTINE_CHANNEL</i>	routine channel group
<i>ADC_INSERTED_CHANNEL</i>	inserted channel group
Input parameter{in}	
external_trigger_source	routine or inserted group trigger source
<i>ADC_EXTTRIG_ROUTINE_T0_CH0</i>	external trigger timer 0 CC0 event select for routine channel
<i>ADC_EXTTRIG_ROUTINE_T0_CH1</i>	external trigger timer 0 CC1 event select for routine channel
<i>ADC_EXTTRIG_ROUTINE_T7_CH2</i>	external trigger timer 7 CC2 event select for routine channel

NE_T0_CH2	
ADC_EXTTRIG_ROUTING_NE_T1_CH1	external trigger timer 1 CC1 event select for routine channel
ADC_EXTTRIG_ROUTING_NE_T1_CH2	external trigger timer 1 CC2 event select for routine channel
ADC_EXTTRIG_ROUTING_NE_T1_CH3	external trigger timer 1 CC3 event select for routine channel
ADC_EXTTRIG_ROUTING_NE_T1_TRGO	external trigger timer 1 TRGO event select for routine channel
ADC_EXTTRIG_ROUTING_NE_T2_CH0	external trigger timer 2 CC0 event select for routine channel
ADC_EXTTRIG_ROUTING_NE_T2_TRGO	external trigger timer 2 TRGO event select for routine channel
ADC_EXTTRIG_ROUTING_NE_T3_CH3	external trigger timer 3 CC3 event select for routine channel
ADC_EXTTRIG_ROUTING_NE_T4_CH0	external trigger timer 4 CC0 event select for routine channel
ADC_EXTTRIG_ROUTING_NE_T4_CH1	external trigger timer 4 CC1 event select for routine channel
ADC_EXTTRIG_ROUTING_NE_T4_CH2	external trigger timer 4 CC2 event select for routine channel
ADC_EXTTRIG_ROUTING_NE_T7_CH0	external trigger timer 7 CC0 event select for routine channel
ADC_EXTTRIG_ROUTING_NE_T7_TRGO	external trigger timer 7 TRGO event select for routine channel
ADC_EXTTRIG_ROUTING_NE_EXTI_11	external trigger extline 11 select for routine channel
ADC_EXTTRIG_INSERTED_TED_T0_CH3	external trigger timer 0 CH3 event select for inserted channel
ADC_EXTTRIG_INSERTED_TED_T0_TRGO	external trigger timer 0 TRGO event select for inserted channel
ADC_EXTTRIG_INSERTED_TED_T1_CH0	external trigger timer 1 CH0 event select for inserted channel
ADC_EXTTRIG_INSERTED_TED_T1_TRGO	external trigger timer 1 TRGO event select for inserted channel
ADC_EXTTRIG_INSERTED_TED_T2_CH1	external trigger timer 2 CH1 event select for inserted channel
ADC_EXTTRIG_INSERTED_TED_T2_CH3	external trigger timer 2 CH3 event select for inserted channel
ADC_EXTTRIG_INSERTED_TED_T3_CH0	external trigger timer 3 CH0 event select for inserted channel
ADC_EXTTRIG_INSERTED_TED_T3_CH1	external trigger timer 3 CH1 event select for inserted channel

ADC_EXTTRIG_INSERTED_T3_CH2	external trigger timer 3 CH2 event select for inserted channel
ADC_EXTTRIG_INSERTED_T3_TRGO	external trigger timer 3 TRGO event select for inserted channel
ADC_EXTTRIG_INSERTED_T4_CH3	external trigger timer 4 CH3 event select for inserted channel
ADC_EXTTRIG_INSERTED_T4_TRGO	external trigger timer 4 TRGO event select for inserted channel
ADC_EXTTRIG_INSERTED_T7_CH1	external trigger timer 7 CH1 event select for inserted channel
ADC_EXTTRIG_INSERTED_T7_CH2	external trigger timer 7 CH2 event select for inserted channel
ADC_EXTTRIG_INSERTED_T7_CH3	external trigger timer 7 CH3 event select for inserted channel
ADC_EXTTRIG_INSERTED_EXTI_15	external trigger extiline 15 select for inserted channel
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* configure ADC0 routine channel external trigger source */
```

```
adc_external_trigger_source_config(ADC0,ADC_ROUTINE_CHANNEL,
ADC_EXTTRIG_ROUTINE_T0_CH0);
```

adc_external_trigger_config

The description of adc_external_trigger_config is shown as below:

Table 3-26. Function adc_external_trigger_config

Function name	adc_external_trigger_config
Function prototype	void adc_external_trigger_config(uint32_t adc_periph , uint8_t adc_channel_group , uint32_t trigger_mode);
Function descriptions	configure ADC external trigger
Precondition	-
The called functions	-
Input parameter{in}	
adc_periph	ADC peripheral
ADCx(x=0..2)	ADC peripheral selection
Input parameter{in}	
adc_channel_group	select the channel group
ADC_ROUTINE_CHANNEL	routine channel group

<i>ADC_INSERTED_CHANNEL</i>	inserted channel group
Input parameter{in}	
trigger_mode	external trigger mode
<i>EXTERNAL_TRIGGER_DISABLE</i>	external trigger disable
<i>EXTERNAL_TRIGGER_RISING</i>	rising edge of external trigger
<i>EXTERNAL_TRIGGER_FALLING</i>	falling edge of external trigger
<i>EXTERNAL_TRIGGER_RISING_FALLING</i>	rising and falling edge of external trigger
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* enable ADC0 inserted channel group external trigger */
adc_external_trigger_config(ADC0, ADC_INSERTED_CHANNEL_0,
EXTERNAL_TRIGGER_RISING);
```

adc_software_trigger_enable

The description of `adc_software_trigger_enable` is shown as below:

Table 3-27. Function `adc_software_trigger_enable`

Function name	<code>adc_software_trigger_enable</code>
Function prototype	<code>void adc_software_trigger_enable(uint32_t adc_periph, uint8_t adc_channel_group);</code>
Function descriptions	enable ADC software trigger
Precondition	-
The called functions	-
Input parameter{in}	
adc_periph	ADC peripheral
<i>ADCx(x=0..2)</i>	ADC peripheral selection
Input parameter{in}	
adc_channel_group	select the channel group
<i>ADC_ROUTINE_CHANNEL</i>	routine channel group
<i>ADC_INSERTED_CHANNEL</i>	inserted channel group
Output parameter{out}	
-	-

Return value	
-	-

Example:

```
/* enable ADC0 routine channel group software trigger */
```

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

adc_end_of_conversion_config

The description of adc_end_of_conversion_config is shown as below:

Table 3-28. Function adc_end_of_conversion_config

Function name	adc_end_of_conversion_config
Function prototype	void adc_end_of_conversion_config(uint32_t adc_periph , uint8_t end_selection);
Function descriptions	configure end of conversion mode
Precondition	-
The called functions	-
Input parameter{in}	
adc_periph	ADC peripheral
ADCx(x=0..2)	ADC peripheral selection
Input parameter{in}	
end_selection	end of conversion mode
ADC_EOC_SET_SEQUENCE	only at the end of a sequence of routine conversions, the EOC bit is set.Overflow detection is disabled unless DMA=1
ADC_EOC_SET_CONVERSION	at the end of each routine conversion, the EOC bit is set.Overflow is detected automatically
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* at the end of each routine conversion, the EOC bit is set. */
```

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

adc_routine_data_read

The description of adc_routine_data_read is shown as below:

Table 3-29. Function adc_routine_data_read

Function name	adc_routine_data_read
Function prototype	uint16_t adc_routine_data_read(uint32_t adc_periph);
Function descriptions	read ADC routine group data register

Precondition	-
The called functions	-
Input parameter{in}	
adc_periph	ADC peripheral
<i>ADCx(x=0..2)</i>	ADC peripheral selection
Output parameter{out}	
-	-
Return value	
uint16_t	ADC conversion value (0-0xFFFF)

Example:

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

```
uint16_t adc_value = 0;
```

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

adc_inserted_data_read

The description of adc_inserted_data_read is shown as below:

Table 3-30. Function adc_inserted_data_read

Function name	adc_inserted_data_read
Function prototype	uint16_t adc_inserted_data_read(uint32_t adc_periph, uint8_t inserted_channel);
Function descriptions	read ADC inserted group data register
Precondition	-
The called functions	-
Input parameter{in}	
adc_periph	ADC peripheral
<i>ADCx(x=0..2)</i>	ADC peripheral selection
Input parameter{in}	
inserted_channel	insert channel select
<i>ADC_INSERTED_CHANNEL_x(x=0..3)</i>	inserted Channelx, x=0,1,2,3
Output parameter{out}	
-	-
Return value	
uint16_t	ADC conversion value (0-0xFFFF)

Example:

```
/* read ADC0 inserted group data register */
```

```
uint16_t adc_value = 0;
```

```
adc_value = adc_inserted_data_read (ADC0, ADC_INSERTED_CHANNEL_0);
```

adc_watchdog_single_channel_disable

The description of adc_watchdog_single_channel_disable is shown as below:

Table 3-31. Function adc_watchdog_single_channel_disable

Function name	adc_watchdog_single_channel_disable
Function prototype	void adc_watchdog_single_channel_disable(uint32_t adc_periph);
Function descriptions	disable ADC analog watchdog single channel
Precondition	-
The called functions	-
Input parameter{in}	
adc_periph	ADC peripheral
<i>ADCx(x=0..2)</i>	ADC peripheral selection
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* disable ADC0 analog watchdog single channel */
```

```
adc_watchdog_single_channel_disable(ADC0);
```

adc_watchdog_single_channel_enable

The description of adc_watchdog_single_channel_enable is shown as below:

Table 3-32. Function adc_watchdog_single_channel_enable

Function name	adc_watchdog_single_channel_enable
Function prototype	void adc_watchdog_single_channel_enable(uint32_t adc_periph, uint8_t adc_channel);
Function descriptions	configure ADC analog watchdog single channel
Precondition	-
The called functions	-
Input parameter{in}	
adc_periph	ADC peripheral
<i>ADCx(x=0..2)</i>	ADC peripheral selection
Input parameter{in}	
adc_channel	the selected ADC channel
<i>ADC_CHANNEL_x(x=0..17)</i>	ADC Channelx(x=0..17) (x=16 and x=17 are only for ADC0)
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* configure ADC0 analog watchdog single channel */
adc_watchdog_single_channel_enable(ADC0, ADC_CHANNEL_1);
```

adc_watchdog_group_channel_enable

The description of adc_watchdog_group_channel_enable is shown as below:

Table 3-33. Function adc_watchdog_group_channel_enable

Function name	adc_watchdog_group_channel_enable
Function prototype	void adc_watchdog_group_channel_enable(uint32_t adc_periph, uint8_t adc_channel_group);
Function descriptions	configure ADC analog watchdog group channel
Precondition	-
The called functions	-
Input parameter{in}	
adc_periph	ADC peripheral
ADCx(x=0..2)	ADC peripheral selection
Input parameter{in}	
adc_channel_group	the channel group use analog watchdog
ADC_ROUTINE_CHANNEL	routine channel group
ADC_INSERTED_CHANNEL	inserted channel group
ADC_ROUTINE_INSERTED_CHANNEL	both routine and inserted group
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* configure ADC0 analog watchdog group channel */
adc_watchdog_group_channel_enable(ADC0, ADC_ROUTINE_CHANNEL);
```

adc_watchdog_disable

The description of adc_watchdog_disable is shown as below:

Table 3-34. Function adc_watchdog_disable

Function name	adc_watchdog_disable
Function prototype	void adc_watchdog_disable(uint32_t adc_periph);
Function descriptions	disable ADC analog watchdog
Precondition	-

The called functions	-
Input parameter{in}	
adc_periph	ADC peripheral
ADCx(0,1)	ADC peripheral selection
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* disable ADC0 analog watchdog */
```

```
adc_watchdog_disable(ADC0);
```

adc_watchdog_threshold_config

The description of adc_watchdog_threshold_config is shown as below:

Table 3-35. Function adc_watchdog_threshold_config

Function name	adc_watchdog_threshold_config
Function prototype	void adc_watchdog_threshold_config(uint32_t adc_periph, uint16_t low_threshold, uint16_t high_threshold);
Function descriptions	configure ADC analog watchdog threshold
Precondition	-
The called functions	-
Input parameter{in}	
adc_periph	ADC peripheral
ADCx(x=0..2)	ADC peripheral selection
Input parameter{in}	
low_threshold	analog watchdog low threshold, 0..4095
Input parameter{in}	
high_threshold	analog watchdog high threshold, 0..4095
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* configure ADC0 analog watchdog threshold */
```

```
adc_watchdog_threshold_config(ADC0, 0x0400, 0x0A00);
```

adc_sync_mode_config

The description of adc_sync_mode_config is shown as below:

Table 3-36. Function `adc_sync_mode_config`

Function name	<code>adc_sync_mode_config</code>
Function prototype	<code>void adc_sync_mode_config(uint32_t sync_mode);</code>
Function descriptions	configure the ADC sync mode
Precondition	-
The called functions	-
Input parameter{in}	
sync_mode	ADC sync mode
<code>ADC_SYNC_MODE_INDEPENDENT</code>	all the ADCs work independently
<code>ADC_DAUL_REGULAR_PARALLEL_INSERTED_PARALLEL</code>	ADC0 and ADC1 work in combined routine parallel & inserted parallel mode
<code>ADC_DAUL_REGULAR_PARALLEL_INSERTED_ROTATION</code>	ADC0 and ADC1 work in combined routine parallel & trigger rotation mode
<code>ADC_DAUL_INSERTED_PARALLEL</code>	ADC0 and ADC1 work in inserted parallel mode
<code>ADC_DAUL_REGULAR_PARALLEL</code>	ADC0 and ADC1 work in routine parallel mode
<code>ADC_DAUL_REGULAR_FOLLOW_UP</code>	ADC0 and ADC1 work in follow-up mode
<code>ADC_DAUL_INSERTED_TRIGGER_ROTATION</code>	ADC0 and ADC1 work in trigger rotation mode
<code>ADC_ALL_REGULAR_PARALLEL_INSERTED_PARALLEL</code>	all ADCs work in combined routine parallel & inserted parallel mode
<code>ADC_ALL_REGULAR_PARALLEL_INSERTED_ROTATION</code>	all ADCs work in combined routine parallel & trigger rotation mode
<code>ADC_ALL_INSERTED_PARALLEL</code>	all ADCs work in inserted parallel mode
<code>ADC_ALL_REGULAR_PARALLEL</code>	all ADCs work in routine parallel mode
<code>ADC_ALL_REGULAR_FOLLOW_UP</code>	all ADCs work in follow-up mode
<code>ADC_ALL_INSERTED_TRIGGER_ROTATION</code>	all ADCs work in trigger rotation mode
Output parameter{out}	
-	-
Return value	

-	-
---	---

Example:

```
/* ADC0 and ADC1 work in combined routine parallel & inserted parallel mode */
```

```
adc_sync_mode_config (ADC_DUAL_REGULAR_PARALLEL_INSERTED_PARALLEL);
```

adc_sync_delay_config

The description of adc_sync_delay_config is shown as below:

Table 3-37. Function adc_interrupt_disable

Function name	adc_sync_delay_config
Function prototype	void adc_sync_delay_config(uint32_t sample_delay);
Function descriptions	configure the delay between 2 sampling phases in ADC sync modes
Precondition	-
The called functions	-
Input parameter{in}	
sample_delay	the delay between 2 sampling phases in ADC sync modes
ADC_SYNC_DELAY_x CYCLE(x=5..20)	the delay between 2 sampling phases in ADC sync modes is x ADC clock cycles
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* configure the delay between 2 sampling phases in ADC sync modes */
```

```
adc_sync_delay_config (ADC_SYNC_DELAY_5CYCLE);
```

adc_sync_dma_config

The description of adc_sync_dma_config is shown as below:

Table 3-38. Function adc_sync_dma_config

Function name	adc_sync_dma_config
Function prototype	void adc_sync_dma_config(uint32_t dma_mode);
Function descriptions	configure ADC sync DMA mode selection
Precondition	-
The called functions	-
Input parameter{in}	
dma_mode	the adc interrupt
ADC_SYNC_DMA_DISABLE	ADC sync DMA disabled
ADC_SYNC_DMA_MODE	ADC sync DMA mode 0

DE0	
ADC_SYNC_DMA_MODE1	ADC sync DMA mode 1
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* configure ADC sync DMA mode selection */
```

```
adc_sync_dma_config (ADC_SYNC_DMA_MODE0);
```

adc_sync_dma_request_after_last_enable

The description of adc_sync_dma_request_after_last_enable is shown as below:

Table 3-39. Function adc_sync_dma_request_after_last_enable

Function name	adc_sync_dma_request_after_last_enable
Function prototype	void adc_sync_dma_request_after_last_enable(void);
Function descriptions	when SYNC DMA is not equal to 2'b00, the DMA engine issues requests according to the SYNC DMA bits
Precondition	-
The called functions	-
Input parameter{in}	
-	-
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* when SYNC DMA is not equal to 2'b00, the DMA engine issues requests according to the SYNC DMA bits */
```

```
adc_sync_dma_request_after_last_enable();
```

adc_sync_dma_request_after_last_disable

The description of adc_sync_dma_request_after_last_disable is shown as below:

Table 3-40. Function adc_sync_dma_request_after_last_disable

Function name	adc_sync_dma_request_after_last_disable
Function prototype	void adc_sync_dma_request_after_last_disable (void);
Function descriptions	the DMA engine is disabled after the end of transfer signal from DMA controller is detected

Precondition	-
The called functions	-
Input parameter{in}	
-	-
Output parameter{out}	
-	-
Return value	
-	-

Example:

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

```
adc_sync_dma_request_after_last_disable();
```

adc_sync_routine_data_read

The description of adc_sync_routine_data_read is shown as below:

Table 3-41. Function adc_interrupt_disable

Function name	adc_sync_routine_data_read
Function prototype	uint32_t adc_sync_routine_data_read(void);
Function descriptions	read ADC sync routine data register
Precondition	-
The called functions	-
Input parameter{in}	
-	-
Output parameter{out}	
-	-
Return value	
uint32_t	ADC sync routine data register

Example:

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

adc_routine_software_startconv_flag_get

The description of adc_routine_software_startconv_flag_get is shown as below:

Table 3-42. Function adc_routine_software_startconv_flag_get

Function name	adc_routine_software_startconv_flag_get
Function prototype	FlagStatus adc_routine_software_startconv_flag_get(uint32_t adc_periph);
Function descriptions	get the bit state of ADCx software routine channel start conversion
Precondition	-
The called functions	-
Input parameter{in}	

adc_periph	ADC peripheral
<i>ADCx(x=0..2)</i>	ADC peripheral selection
Output parameter{out}	
-	-
Return value	
FlagStatus	SET or RESET

Example:

```
/* get the bit state of ADC0 software routine channel start conversion */
```

```
FlagStatus flag_value;
```

```
flag_value = adc_routine_software_startconv_flag_get(ADC0);
```

adc_inserted_software_startconv_flag_get

The description of adc_inserted_software_startconv_flag_get is shown as below:

Table 3-43. Function adc_inserted_software_startconv_flag_get

Function name	adc_inserted_software_startconv_flag_get
Function prototype	FlagStatus adc_inserted_software_startconv_flag_get(uint32_t adc_periph);
Function descriptions	get the bit state of ADCx software inserted channel start conversion
Precondition	-
The called functions	-
Input parameter{in}	
adc_periph	ADC peripheral
<i>ADCx(x=0..2)</i>	ADC peripheral selection
Output parameter{out}	
-	-
Return value	
FlagStatus	SET or RESET

Example:

```
/* get the bit state of ADC0 software inserted channel start conversion */
```

```
FlagStatus flag_value;
```

```
flag_value= adc_inserted_software_startconv_flag_get(ADC0);adc_sync_routine_data_read  
();
```

adc_flag_get

The description of adc_flag_get is shown as below:

Table 3-44. Function adc_flag_get

Function name	adc_flag_get
Function prototype	FlagStatus adc_flag_get(uint32_t adc_periph , uint32_t adc_flag);

Function descriptions	get the ADC flag bits
Precondition	-
The called functions	-
Input parameter{in}	
adc_periph	ADC peripheral
<i>ADCx(x=0..2)</i>	ADC peripheral selection
Input parameter{in}	
adc_flag	the adc flag bits
<i>ADC_FLAG_WDE</i>	analog watchdog event flag
<i>ADC_FLAG_EOC</i>	end of group conversion flag
<i>ADC_FLAG_EOIC</i>	end of inserted group conversion flag
<i>ADC_FLAG_STIC</i>	start flag of inserted channel group
<i>ADC_FLAG_STRC</i>	start flag of routine channel group
Output parameter{out}	
-	-
Return value	
FlagStatus	SET or RESET

Example:

```
/* get the ADC0 analog watchdog flag bits*/
```

```
FlagStatus flag_value;
```

```
flag_value = adc_flag_get(ADC0, ADC_FLAG_WDE);
```

adc_flag_clear

The description of adc_flag_clear is shown as below:

Table 3-45. Function adc_flag_clear

Function name	adc_flag_clear
Function prototype	void adc_flag_clear(uint32_t adc_periph, uint32_t adc_flag);
Function descriptions	clear the ADC flag bits
Precondition	-
The called functions	-
Input parameter{in}	
adc_periph	ADC peripheral
<i>ADCx(x=0..2)</i>	ADC peripheral selection
Input parameter{in}	
adc_flag	the adc flag bits
<i>ADC_FLAG_WDE</i>	analog watchdog event flag
<i>ADC_FLAG_EOC</i>	end of group conversion flag
<i>ADC_FLAG_EOIC</i>	end of inserted group conversion flag
<i>ADC_FLAG_STIC</i>	start flag of inserted channel group
<i>ADC_FLAG_STRC</i>	start flag of routine channel group

Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* clear the ADC0 analog watchdog flag bits*/
```

```
adc_flag_clear(ADC0, ADC_FLAG_WDE);
```

adc_interrupt_flag_get

The description of adc_interrupt_flag_get is shown as below:

Table 3-46. Function adc_interrupt_flag_get

Function name	adc_interrupt_flag_get
Function prototype	FlagStatus adc_interrupt_flag_get(uint32_t adc_periph, uint32_t adc_interrupt);
Function descriptions	get the ADC interrupt bits
Precondition	-
The called functions	-
Input parameter{in}	
adc_periph	ADC peripheral
ADCx(x=0..2)	ADC peripheral selection
Input parameter{in}	
adc_interrupt	the adc interrupt bits
ADC_INT_WDE	analog watchdog interrupt
ADC_INT_EOC	end of group conversion interrupt
ADC_INT_EOIC	end of inserted group conversion interrupt
Output parameter{out}	
-	-
Return value	
FlagStatus	SET or RESET

Example:

```
/* get the ADC0 analog watchdog interrupt bits*/
```

```
FlagStatus flag_value;
```

```
flag_value = adc_interrupt_flag_get(ADC0, ADC_INT_WDE);
```

adc_interrupt_flag_clear

The description of adc_interrupt_flag_clear is shown as below:

Table 3-47. Function adc_interrupt_flag_clear

Function name	adc_interrupt_flag_clear
---------------	--------------------------

Function prototype	void adc_interrupt_flag_clear(uint32_t adc_periph, uint32_t adc_interrupt);
Function descriptions	clear the ADC interrupt bits
Precondition	-
The called functions	-
Input parameter{in}	
adc_periph	ADC peripheral
<i>ADCx(x=0..2)</i>	ADC peripheral selection
Input parameter{in}	
adc_interrupt	the adc interrupt bits
<i>ADC_INT_WDE</i>	analog watchdog interrupt
<i>ADC_INT_EOC</i>	end of group conversion interrupt
<i>ADC_INT_EOIC</i>	end of inserted group conversion interrupt
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* clear the ADC0 analog watchdog interrupt bits*/
```

```
adc_interrupt_flag_clear(ADC0, ADC_INT_WDE);
```

adc_interrupt_enable

The description of adc_interrupt_enable is shown as below:

Table 3-48. Function adc_interrupt_enable

Function name	adc_interrupt_enable
Function prototype	void adc_interrupt_enable(uint32_t adc_periph, uint32_t adc_interrupt);
Function descriptions	enable ADC interrupt
Precondition	-
The called functions	-
Input parameter{in}	
adc_periph	ADC peripheral
<i>ADCx(x=0..2)</i>	ADC peripheral selection
Input parameter{in}	
adc_interrupt	the adc interrupt
<i>ADC_INT_WDE</i>	analog watchdog interrupt
<i>ADC_INT_EOC</i>	end of group conversion interrupt
<i>ADC_INT_EOIC</i>	end of inserted group conversion interrupt
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* enable ADC0 analog watchdog interrupt */
adc_interrupt_enable(ADC0, ADC_INT_WDE);
```

adc_interrupt_disable

The description of adc_interrupt_disable is shown as below:

Table 3-49. Function adc_interrupt_disable

Function name	adc_interrupt_disable
Function prototype	void adc_interrupt_disable(uint32_t adc_periph , uint32_t adc_interrupt);
Function descriptions	Disable ADC interrupt
Precondition	-
The called functions	-
Input parameter{in}	
adc_periph	ADC peripheral
<i>ADCx(x=0..2)</i>	ADC peripheral selection
Input parameter{in}	
adc_interrupt	the adc interrupt
<i>ADC_INT_WDE</i>	analog watchdog interrupt
<i>ADC_INT_EOC</i>	end of group conversion interrupt
<i>ADC_INT_EOIC</i>	end of inserted group conversion interrupt
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* disable ADC0 interrupt */
adc_interrupt_disable(ADC0, ADC_INT_WDE);
```

3.3. CAN

CAN bus (for Controller Area Network) is a bus standard designed to allow microcontrollers and devices to communicate with each other without a host computer. The CAN registers are listed in chapter [3.3.1](#), the CAN firmware functions are introduced in chapter [3.3.2](#).

3.3.1. Descriptions of Peripheral registers

CAN registers are listed in the table shown as below:

Table 3-50. CAN Registers

Registers	Descriptions
CAN_CTL	Control register
CAN_STAT	Status register
CAN_TSTAT	Transmit status register
CAN_RFIFO0	Receive message FIFO0 register
CAN_RFIFO1	Receive message FIFO1 register
CAN_INTEN	Interrupt enable register
CAN_ERR	Error register
CAN_BT	Bit timing register
CAN_FDCTL	FD control register
CAN_FDSTAT	FD status register
CAN_FDTDC	FD transmitter delay compensation register
CAN_DBT	Date Bit timing register
CAN_TMIx	Transmit mailbox identifier register
CAN_TMPx	Transmit mailbox property register
CAN_TMDATA0x	Transmit mailbox data0 register
CAN_TMDATA1x	Transmit mailbox data1 register
CAN_RFIFOMIx	Receive FIFO mailbox identifier register
CAN_RFIFOMPx	Receive FIFO mailbox property register
CAN_RFIFOMDAT A0x	Receive FIFO mailbox data0 register
CAN_RFIFOMDAT A1x	Receive FIFO mailbox data1 register
CAN_FCTL	Filter control register
CAN_FMCFG	Filter mode configuration register
CAN_FSCFG	Filter scale configuration register
CAN_FAFIFO	Filter associated FIFO registe
CAN_FW	Filter working register
CAN_FxDATAy	Filter x data y register

3.3.2. Descriptions of Peripheral functions

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

Table 3-51. CAN firmware function

Function name	Function description
can_deinit	deinitialize CAN
can_struct_para_init	initialize CAN parameter struct with a default value
can_init	initialize CAN
can_fd_init	initialize CAN FD function
can_filter_init	initialize CAN filter
can_filter_mask_mode_init	CAN filter mask mode initialization
can_monitor_mode_set	CAN communication mode configure

Function name	Function description
can_fd_function_enable	CAN FD frame function enable
can_fd_function_disable	CAN FD frame function disable
can1_filter_start_bank	set can1 filter start bank number
can_debug_freeze_enable	CAN debug freeze enable
can_debug_freeze_disable	CAN debug freeze disable
can_time_trigger_mode_enable	CAN time trigger mode enable
can_time_trigger_mode_disable	CAN time trigger mode disable
can_message_transmit	transmit CAN message
can_transmit_states	get CAN transmit state
can_transmission_stop	stop CAN transmission
can_message_receive	CAN receive message
can_fifo_release	CAN release fifo
can_receive_message_length_get	CAN receive message length
can_working_mode_set	CAN working mode
can_wakeup	CAN wakeup from sleep mode
can_error_get	CAN get error
can_receive_error_number_get	get CAN receive error number
can_transmit_error_number_get	get CAN transmit error number
can_interrupt_enable	CAN interrupt enable
can_interrupt_disable	CAN interrupt disable
can_flag_get	CAN get flag state
can_flag_clear	CAN clear flag state
can_interrupt_flag_get	CAN get interrupt flag state
can_interrupt_flag_clear	CAN clear interrupt flag state

Structure can_parameter_struct

Table 3-52. Structure can_parameter_struct

Member name	Function description
working_mode	CAN working mode
resync_jump_width	CAN resynchronization jump width
time_segment_1	time segment 1
time_segment_2	time segment 2
time_triggered	time triggered communication mode
auto_bus_off_recovery	automatic bus-off recovery
auto_wake_up	automatic wake-up mode
auto_retrans	automatic retransmission mode
rec_fifo_overwrite	receive FIFO overwrite mode
trans_fifo_order	transmit FIFO order
prescaler	baudrate prescaler

Structure can_trasmit_message_struct

Table 3-53. Structure can_trasmit_message_struct

Member name	Function description
tx_sfids	standard format frame identifier
tx_efids	extended format frame identifier
tx_ff	format of frame, standard or extended format
tx_ft	type of frame, data or remote
tx_dlen	data length
tx_data[64]	transmit data
fd_flag	CAN FD frame flag
fd_brs	bit rate of data switch
fd_esi	error status indicator

Structure can_receive_message_struct

Table 3-54. Structure can_receive_message_struct

Member name	Function description
rx_sfids	standard format frame identifier
rx_efids	extended format frame identifier
rx_ff	format of frame, standard or extended format
rx_ft	type of frame, data or remote
rx_dlen	data length
rx_data[64]	receive data
rx_fi	filtering index
fd_flag	CAN FD frame flag
fd_brs	bit rate of data switch
fd_esi	error status indicator

Structure can_filter_parameter_struct

Table 3-55. Structure can_filter_parameter_struct

Member name	Function description
filter_list_high	filter list number high bits
filter_list_low	filter list number low bits
filter_mask_high	filter mask number high bits
filter_mask_low	filter mask number low bits
filter_fifo_number	receive FIFO associated with the filter
filter_number	filter number
filter_mode	filter mode, list or mask
filter_bits	filter scale
filter_enable	filter work or not

Structure can_fd_tdc_struct

Table 3-56. Structure can_fd_tdc_struct

Member name	Function description
tdc_mode	transmitter delay compensation mode
tdc_filter	transmitter delay compensation filter
tdc_offset	transmitter delay compensation offset

Structure can_fdframe_struct

Table 3-57. Structure can_fdframe_struct

Member name	Function description
fd_frame	FD operation function
excp_event_detect	protocol exception event detection function
delay_compensation	transmitter delay compensation mode
p_delay_compensation	pointer to the struct of the transmitter delay compensation, refer to Table 3-56. Structure can_fd_tdc_struct
iso_bosch	ISO/Bosch mode choice
esi_mode	error state indicator mode
data_resync_jump_width	CAN resynchronization jump width
data_time_segment_1	time segment 1
data_time_segment_2	time segment 2
data_prescaler	baudrate prescaler

can_deinit

The description of can_deinit is shown as below:

Table 3-58. Function can_deinit

Function name	can_deinit
Function prototype	void can_deinit(uint32_t can_periph);
Function descriptions	deinitialize CAN
Precondition	-
The called functions	rcu_periph_reset_enable/ rcu_periph_reset_disable
Input parameter{in}	
can_periph	CAN peripheral
CANx(x=0,1)	CAN peripheral selection
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* CAN0 deinitialize */
can_deinit (CAN0);
```

can_struct_para_init

The description of can_struct_para_init is shown as below:

Table 3-59. Function can_struct_para_init

Function name	can_struct_para_init
Function prototype	void can_struct_para_init(can_struct_type_enum type, void* p_struct)
Function descriptions	initialize CAN parameter struct with a default value
Precondition	-
The called functions	-
Input parameter{in}	
type	CAN peripheral
CAN_INIT_STRUCT	CAN initialize parameters struct
CAN_FILTER_STRUCT	CAN filter parameters struct
CAN_FD_FRAME_STRUCT	CAN initialize FD frame parameters struct
CAN_TX_MESSAGE_STRUCT	CAN transmit message struct
CAN_RX_MESSAGE_STRUCT	CAN receive message struct
Output parameter{out}	
p_struct	the struct pointer that needs initialize
Return value	
-	-

Example:

```
/* Initialize CAN parameter struct */
can_parameter_struct can_init;
can_struct_para_init (CAN_INIT_STRUCT, &can_init);
```

can_init

The description of can_init is shown as below:

Table 3-60. Function can_init

Function name	can_init
Function prototype	ErrStatus can_init(uint32_t can_periph, can_parameter_struct* can_parameter_init);

Function descriptions	initialize CAN
Precondition	can_struct_para_init()
The called functions	-
Input parameter{in}	
can_periph	CAN peripheral
<i>CANx(x=0,1)</i>	CAN peripheral selection
Input parameter{in}	
can_parameter_init	CAN parameter initialization struct, the structure members can refer to members of the structure Table 3-52. Structure can_parameter_struct
Output parameter{out}	
-	-
Return value	
ErrStatus	SUCCESS / ERROR

Example:

```
/* CAN0 initialize */
can_parameter_struct can_parameter_init;
can_init (CAN0, &can_parameter_init);
```

can_fd_init

The description of can_fd_init is shown as below:

Table 3-61. Function can_fd_init

Function name	can_fd_init
Function prototype	ErrStatus can_fd_init(uint32_t can_periph, can_fdframe_struct* can_fdframe_init);
Function descriptions	initialize CAN FD function
Precondition	can_struct_para_init()
The called functions	-
Input parameter{in}	
can_periph	CAN peripheral
<i>CANx(x=0,1)</i>	CAN peripheral selection
Input parameter{in}	
can_fdframe_init	parameters for CAN FD initialization, the structure members can refer to members of the structure Table 3-57. Structure can_fdframe_struct
Output parameter{out}	
-	-
Return value	
ErrStatus	SUCCESS / ERROR

Example:

```

/* CAN0 FD initialize */

can_fdframe_struct fd_init_para;

can_fd_init(CAN0, &fd_init_para);

```

can_filter_init

The description of can_filter_init is shown as below:

Table 3-62. Function can_filter_init

Function name	can_filter_init
Function prototype	void can_filter_init(can_filter_parameter_struct* can_filter_parameter_init);
Function descriptions	initialize CAN filter
Precondition	can_struct_para_init()
The called functions	-
Input parameter{in}	
can_filter_parameter_init	CAN filter initialization struct, the structure members can refer to members of the structure Table 3-55. Structure can_filter_parameter_struct
Output parameter{out}	
-	-
Return value	
-	-

Example:

```

/* initialize CAN filter */

can_filter_init(&can_filter);

```

can_filter_mask_mode_init

The description of can_filter_mask_mode_init is shown as below:

Table 3-63. Function can_filter_mask_mode_init

Function name	can_filter_mask_mode_init
Function prototype	void can_filter_mask_mode_init(uint32_t id, uint32_t mask, can_format_fifo_enum format_fifo, uint16_t filter_number)
Function descriptions	CAN filter mask mode initialization
Precondition	-
The called functions	can_filter_init()
Input parameter{in}	
id	value range (0x00000000 - 0xFFFFFFFF)
Input parameter{in}	
mask	value range (0x00000000 - 0xFFFFFFFF)
Input parameter{in}	
format_fifo	format and fifo states, only one parameter can be selected which is shown as below

<i>CAN_STANDARD_FIF</i> 00	standard format and store to FIFO0
<i>CAN_STANDARD_FIF</i> 01	standard format and store to FIFO1
<i>CAN_EXTENDED_FIF</i> 00	extended format and store to FIFO0
<i>CAN_EXTENDED_FIF</i> 01	extended format and store to FIFO1
Input parameter{in}	
filter_number	filter sequence number, value range(0x00 - 0x1B)
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* CAN filter mask mode initialization */
```

```
can_filter_mask_mode_init(0x11, 0x11, CAN_STANDARD_FIFO0, 0);
```

can_monitor_mode_set

The description of can_monitor_mode_set is shown as below:

Table 3-64. Function can_monitor_mode_set

Function name	can_monitor_mode_set
Function prototype	ErrStatus can_monitor_mode_set(uint32_t can_periph, uint8_t mode)
Function descriptions	CAN communication mode configure
Precondition	-
The called functions	-
Input parameter{in}	
can_periph	CAN peripheral
<i>CANx(x=0,1)</i>	CAN peripheral selection
Input parameter{in}	
mode	communication mode, only one parameter can be selected which is shown as below
<i>CAN_NORMAL_MODE</i>	normal mode
<i>CAN_LOOPBACK_MODE</i>	loopback mode
<i>CAN_SILENT_MODE</i>	silent mode
<i>CAN_SILENT_LOOPBACK_MODE</i>	silent loopback mode
Output parameter{out}	
-	-
Return value	

ErrStatus	SUCCESS / ERROR
------------------	-----------------

Example:

```
/* CAN communication mode configure */
can_monitor_mode_set(CAN0, CAN_NORMAL_MODE);
```

can_fd_function_enable

The description of can_fd_function_enable is shown as below:

Table 3-65. Function can_fd_function_enable

Function name	can_fd_function_enable
Function prototype	void can_fd_function_enable(uint32_t can_periph)
Function descriptions	CAN FD frame function enable
Precondition	-
The called functions	-
Input parameter{in}	
can_periph	CAN peripheral
CANx(x=0,1)	CAN peripheral selection
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* CAN0 FD frame function enable */
can_fd_function_enable(CAN0);
```

can_fd_function_disable

The description of can_fd_function_disable is shown as below:

Table 3-66. Function can_fd_function_disable

Function name	can_fd_function_disable
Function prototype	void can_fd_function_disable(uint32_t can_periph)
Function descriptions	CAN FD frame function disable
Precondition	-
The called functions	-
Input parameter{in}	
can_periph	CAN peripheral
CANx(x=0,1)	CAN peripheral selection
Output parameter{out}	
-	-
Return value	

-	-
---	---

Example:

```
/* CAN0 FD frame function disable */
can_fd_function_disable(CAN0);
```

can1_filter_start_bank

The description of can1_filter_start_bank is shown as below:

Table 3-67. Function can1_filter_start_bank

Function name	can1_filter_start_bank
Function prototype	void can1_filter_start_bank(uint8_t start_bank);
Function descriptions	set CAN1 fliter start bank number
Precondition	-
The called functions	-
Input parameter{in}	
start_bank	CAN1 start bank number
1..27	start number
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* set CAN1 fliter start bank number 15 */
can1_filter_start_bank (15);
```

can_debug_freeze_enable

The description of can_debug_freeze_enable is shown as below:

Table 3-68. Function can_debug_freeze_enable

Function name	can_debug_freeze_enable
Function prototype	void can_debug_freeze_enable(uint32_t can_periph);
Function descriptions	enable CAN debug freeze
Precondition	-
The called functions	dbg_periph_enable
Input parameter{in}	
can_periph	CAN peripheral
CANx(x=0,1)	CAN peripheral selection
Output parameter{out}	
-	-
Return value	

-	-
---	---

Example:

```
/* enable CAN0 debug freeze */
can_debug_freeze_enable (CAN0);
```

can_debug_freeze_disable

The description of can_debug_freeze_disable is shown as below:

Table 3-69. Function can_debug_freeze_disable

Function name	can_debug_freeze_disable
Function prototype	void can_debug_freeze_disable(uint32_t can_periph);
Function descriptions	disable CAN debug freeze
Precondition	-
The called functions	dbg_periph_disable
Input parameter{in}	
can_periph	CAN peripheral
<i>CANx(x=0,1)</i>	CAN peripheral selection
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* disable CAN0 debug freeze */
can_debug_freeze_disable (CAN0);
```

can_time_trigger_mode_enable

The description of can_time_trigger_mode_enable is shown as below:

Table 3-70. Function can_time_trigger_mode_enable

Function name	can_time_trigger_mode_enable
Function prototype	void can_time_trigger_mode_enable(uint32_t can_periph);
Function descriptions	enable CAN time trigger mode
Precondition	-
The called functions	-
Input parameter{in}	
can_periph	CAN peripheral
<i>CANx(x=0,1)</i>	CAN peripheral selection
Output parameter{out}	
-	-
Return value	

-	-
---	---

Example:

```
/* enable CAN0 time trigger mode */
can_time_trigger_mode_enable (CAN0);
```

can_time_trigger_mode_disable

The description of can_time_trigger_mode_disable is shown as below:

Table 3-71. Function can_time_trigger_mode_disable

Function name	can_time_trigger_mode_disable
Function prototype	void can_time_trigger_mode_disable(uint32_t can_periph);
Function descriptions	disable CAN time trigger mode
Precondition	-
The called functions	-
Input parameter{in}	
can_periph	CAN peripheral
<i>CANx(x=0,1)</i>	CAN peripheral selection
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* disable CAN0 time trigger mode */
can_time_trigger_mode_disable (CAN0);
```

can_message_transmit

The description of can_message_transmit is shown as below:

Table 3-72. Function can_message_transmit

Function name	can_message_transmit
Function prototype	uint8_t can_message_transmit(uint32_t can_periph, can_transmit_message_struct* transmit_message);
Function descriptions	transmit CAN message
Precondition	can_struct_para_init()
The called functions	-
Input parameter{in}	
can_periph	CAN peripheral
<i>CANx(x=0,1)</i>	CAN peripheral selection
Input parameter{in}	
transmit_message	CAN transmit message struct, the structure members can refer to members

	of the structure Table 3-53. Structure can_trasnmitt message struct
Output parameter{out}	
-	-
Return value	
uint8_t	0x00-0x03

Example:

```
/* CAN0 transmit message and return the mailbox number */
```

```
uint8_t transmit_mailbox = 0;
```

```
transmit_mailbox = can_message_transmit(CAN0, &transmit_message);
```

can_transmit_states

The description of can_transmit_states is shown as below:

Table 3-73. Function can_transmit_states

Function name	can_transmit_states
Function prototype	can_transmit_state_enum can_transmit_states(uint32_t can_periph, uint8_t mailbox_number);
Function descriptions	get CAN transmit state
Precondition	-
The called functions	-
Input parameter{in}	
can_periph	CAN peripheral
CANx(x=0,1)	CAN peripheral selection
Input parameter{in}	
mailbox_number	Mailbox number
CAN_MAILBOXx	CAN_MAILBOXx(x=0,1,2)
Output parameter{out}	
-	-
Return value	
can_transmit_state_enum	0..4

Example:

```
/* CAN0 mailbox0 transmit state */
```

```
uint8_t transmit_state = 0;
```

```
transmit_state = can_transmit_states (CAN0, CAN_MAILBOX0);
```

can_transmission_stop

The description of can_transmission_stop is shown as below:

Table 3-74. Function can_transmission_stop

Function name	can_transmission_stop
Function prototype	void can_transmission_stop(uint32_t can_periph, uint8_t mailbox_number);
Function descriptions	stop CAN transmission
Precondition	-
The called functions	-
Input parameter{in}	
can_periph	CAN peripheral
CANx(x=0,1)	CAN peripheral selection
Input parameter{in}	
mailbox_number	Mailbox number
CAN_MAILBOXx	CAN_MAILBOXx(x=0,1,2)
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* stop CAN0 mailbox0 transmission */
can_transmission_stop (CAN0, CAN_MAILBOX0);
```

can_message_receive

The description of can_message_receive is shown as below:

Table 3-75. Function can_message_receive

Function name	can_message_receive
Function prototype	void can_message_receive(uint32_t can_periph, uint8_t fifo_number, can_receive_message_struct* receive_message);
Function descriptions	CAN receive message
Precondition	can_struct_para_init()
The called functions	-
Input parameter{in}	
can_periph	CAN peripheral
CANx(x=0,1)	CAN peripheral selection
Input parameter{in}	
fifo_number	Fifo number
CAN_FIFOx	CAN_FIFOx(x=0,1)
Input parameter{in}	
receive_message	CAN message receive struct, the structure members can refer to members of the structure Table 3-54. Structure can_receive_message_struct
Output parameter{out}	
-	-
Return value	

-	-
---	---

Example:

```
/* CAN0 FIFO0 receive message */
```

```
can_message_receive(CAN0, CAN_FIFO0, &receive_message);
```

can_fifo_release

The description of can_fifo_release is shown as below:

Table 3-76. Function can_fifo_release

Function name	can_fifo_release
Function prototype	void can_fifo_release(uint32_t can_periph, uint8_t fifo_number);
Function descriptions	release FIFO0
Precondition	-
The called functions	-
Input parameter{in}	
can_periph	CAN peripheral
<i>CANx(x=0,1)</i>	CAN peripheral selection
Input parameter{in}	
fifo_number	Fifo number
<i>CAN_FIFOx</i>	CAN_FIFOx(x=0,1)
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* CAN0 release FIFO0 */
```

```
can_fifo_release (CAN0, CAN_FIFO0);
```

can_receive_message_length_get

The description of can_receive_message_length_get is shown as below:

Table 3-77. Function can_receive_message_length_get

Function name	can_receive_message_length_get
Function prototype	uint8_t can_receive_message_length_get(uint32_t can_periph, uint8_t fifo_number);
Function descriptions	CAN receive message length
Precondition	-
The called functions	-
Input parameter{in}	
can_periph	CAN peripheral

<i>CANx(x=0,1)</i>	CAN peripheral selection
Input parameter{in}	
fifo_number	Fifo number
<i>CAN_FIFOx</i>	CAN_FIFOx(x=0,1)
Output parameter{out}	
-	-
Return value	
uint8_t	0..3

Example:

```
/* CAN0 FIFO0 receive message length */
```

```
uint8_t frame_number = 0;
```

```
frame_number = can_receive_message_length_get (CAN0, CAN_FIFO0);
```

can_working_mode_set

The description of can_working_mode_set is shown as below:

Table 3-78. Function can_working_mode_set

Function name	can_working_mode_set
Function prototype	ErrStatus can_working_mode_set(uint32_t can_periph, uint8_t working_mode);
Function descriptions	set CAN working mode
Precondition	-
The called functions	-
Input parameter{in}	
can_periph	CAN peripheral
<i>CANx(x=0,1)</i>	CAN peripheral selection
Input parameter{in}	
can_working_mode	Mode select
<i>CAN_MODE_INITIALIZE</i>	Initialize mode
<i>CAN_MODE_NORMAL</i>	Normal mode
<i>CAN_MODE_SLEEP</i>	Sleep mode
Output parameter{out}	
-	-
Return value	
ErrStatus	SUCCESS / ERROR

Example:

```
/* set CAN0 working at initialize mode */
```

```
can_working_mode_set (CAN0, CAN_MODE_INITIALIZE);
```

can_wakeup

The description of can_wakeup is shown as below:

Table 3-79. Function can_wakeup

Function name	can_wakeup
Function prototype	ErrStatus can_wakeup(uint32_t can_periph);
Function descriptions	wake up CAN
Precondition	-
The called functions	-
Input parameter{in}	
can_periph	CAN peripheral
<i>CANx(x=0,1)</i>	CAN peripheral selection
Output parameter{out}	
-	-
Return value	
ErrStatus	SUCCESS / ERROR

Example:

```
/* wake up CAN0 */
can_wakeup (CAN0);
```

can_error_get

The description of can_error_get is shown as below:

Table 3-80. Function can_error_get

Function name	can_error_get
Function prototype	can_error_enum can_error_get(uint32_t can_periph);
Function descriptions	get CAN error type
Precondition	-
The called functions	-
Input parameter{in}	
can_periph	CAN peripheral
<i>CANx(x=0,1)</i>	CAN peripheral selection
Output parameter{out}	
-	-
Return value	
can_error_enum	0..7

Example:

```
/* get CAN0 error type */
can_error_get (CAN0);
```


can_receive_error_number_get

The description of can_receive_error_number_get is shown as below:

Table 3-81. Function can_receive_error_number_get

Function name	can_receive_error_number_get
Function prototype	uint8_t can_receive_error_number_get(uint32_t can_periph);
Function descriptions	get CAN receive error number
Precondition	-
The called functions	-
Input parameter{in}	
can_periph	CAN peripheral
<i>CANx(x=0,1)</i>	CAN peripheral selection
Output parameter{out}	
-	-
Return value	
uint8_t	0..255

Example:

```
/* get CAN0 receive error number */
can_receive_error_number_get (CAN0);
```

can_transmit_error_number_get it

The description of can_transmit_error_number_get is shown as below:

Table 3-82. Function can_transmit_error_number_get

Function name	can_transmit_error_number_get
Function prototype	uint8_t can_transmit_error_number_get(uint32_t can_periph);
Function descriptions	get CAN transmit error number
Precondition	-
The called functions	-
Input parameter{in}	
can_periph	CAN peripheral
<i>CANx(x=0,1)</i>	CAN peripheral selection
Output parameter{out}	
-	-
Return value	
uint8_t	0..255

Example:

```
/* get CAN0 transmit error number */
can_transmit_error_number_get (CAN0);
```

can_flag_get

The description of can_flag_get is shown as below:

Table 3-83. Function can_flag_get

Function name	can_flag_get
Function prototype	FlagStatus can_flag_get(uint32_t can_periph, can_flag_enum flag);
Function descriptions	get CAN flag state
Precondition	-
The called functions	-
Input parameter{in}	
can_periph	CAN peripheral
CANx(x=0,1)	CAN peripheral selection
Input parameter{in}	
flag	CAN flags
CAN_FLAG_MTE2	mailbox 2 transmit error
CAN_FLAG_MTE1	mailbox 1 transmit error
CAN_FLAG_MTE0	mailbox 0 transmit error
CAN_FLAG_MTF2	mailbox 2 transmit finished
CAN_FLAG_MTF1	mailbox 1 transmit finished
CAN_FLAG_MTF0	mailbox 0 transmit finished
CAN_FLAG_RFO0	receive FIFO0 overfull
CAN_FLAG_RFF0	receive FIFO0 full
CAN_FLAG_RFO1	receive FIFO1 overfull
CAN_FLAG_RFF1	receive FIFO1 full
CAN_FLAG_BOERR	bus-off error
CAN_FLAG_PERR	passive error
CAN_FLAG_WERR	warning error
Output parameter{out}	
-	-
Return value	
FlagStatus	SET / RESET

Example:

```
/* get CAN0 mailbox 0 transmit finished flag */
```

```
can_flag_get (CAN0, CAN_FLAG_MTF0);
```

can_flag_clear

The description of can_flag_clear is shown as below:

Table 3-84. Function can_flag_clear

Function name	can_flag_clear
Function prototype	void can_flag_clear(uint32_t can_periph, can_flag_enum flag);

Function descriptions	clear CAN flag state
Precondition	-
The called functions	-
Input parameter{in}	
can_periph	CAN peripheral
<i>CANx(x=0,1)</i>	CAN peripheral selection
Input parameter{in}	
flag	CAN flags
<i>CAN_FLAG_MTE2</i>	mailbox 2 transmit error
<i>CAN_FLAG_MTE1</i>	mailbox 1 transmit error
<i>CAN_FLAG_MTE0</i>	mailbox 0 transmit error
<i>CAN_FLAG_MTF2</i>	mailbox 2 transmit finished
<i>CAN_FLAG_MTF1</i>	mailbox 1 transmit finished
<i>CAN_FLAG_MTF0</i>	mailbox 0 transmit finished
<i>CAN_FLAG_RFO0</i>	receive FIFO0 overfull
<i>CAN_FLAG_RFF0</i>	receive FIFO0 full
<i>CAN_FLAG_RFO1</i>	receive FIFO1 overfull
<i>CAN_FLAG_RFF1</i>	receive FIFO1 full
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* clear CAN0 mailbox 0 transmit error flag */
```

```
can_flag_clear (CAN0, CAN_FLAG_MTE0);
```

can_interrupt_enable

The description of can_interrupt_enable is shown as below:

Table 3-85. Function can_interrupt_enable

Function name	can_interrupt_enable
Function prototype	void can_interrupt_enable(uint32_t can_periph, uint32_t interrupt);
Function descriptions	enable CAN interrupt
Precondition	-
The called functions	-
Input parameter{in}	
can_periph	CAN peripheral
<i>CANx(x=0,1)</i>	CAN peripheral selection
Input parameter{in}	
interrupt	Interrupt type
<i>CAN_INT_TME</i>	transmit mailbox empty interrupt enable

<i>CAN_INT_RFNE0</i>	receive FIFO0 not empty interrupt enable
<i>CAN_INT_RFF0</i>	receive FIFO0 full interrupt enable
<i>CAN_INT_RFO0</i>	receive FIFO0 overfull interrupt enable
<i>CAN_INT_RFNE1</i>	receive FIFO1 not empty interrupt enable
<i>CAN_INT_RFF1</i>	receive FIFO1 full interrupt enable
<i>CAN_INT_RFO1</i>	receive FIFO1 overfull interrupt enable
<i>CAN_INT_WERR</i>	warning error interrupt enable
<i>CAN_INT_PERR</i>	passive error interrupt enable
<i>CAN_INT_BO</i>	bus-off interrupt enable
<i>CAN_INT_ERRN</i>	error number interrupt enable
<i>CAN_INT_ERR</i>	error interrupt enable
<i>CAN_INT_WU</i>	wakeup interrupt enable
<i>CAN_INT_SLPW</i>	sleep working interrupt enable
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* CAN0 transmit mailbox empty interrupt enable */
```

```
can_interrupt_enable (CAN0, CAN_INT_TME);
```

can_interrupt_disable

The description of can_interrupt_disable is shown as below:

Table 3-86. Function can_interrupt_disable

Function name	can_interrupt_disable
Function prototype	void can_interrupt_disable(uint32_t can_periph, uint32_t interrupt);
Function descriptions	disable CAN interrupt
Precondition	-
The called functions	-
Input parameter{in}	
can_periph	CAN peripheral
<i>CANx(x=0,1)</i>	CAN peripheral selection
Input parameter{in}	
interrupt	Interrupt type
<i>CAN_INT_TME</i>	transmit mailbox empty interrupt enable
<i>CAN_INT_RFNE0</i>	receive FIFO0 not empty interrupt enable
<i>CAN_INT_RFF0</i>	receive FIFO0 full interrupt enable
<i>CAN_INT_RFO0</i>	receive FIFO0 overfull interrupt enable
<i>CAN_INT_RFNE1</i>	receive FIFO1 not empty interrupt enable
<i>CAN_INT_RFF1</i>	receive FIFO1 full interrupt enable

<i>CAN_INT_RFO1</i>	receive FIFO1 overfull interrupt enable
<i>CAN_INT_WERR</i>	warning error interrupt enable
<i>CAN_INT_PERR</i>	passive error interrupt enable
<i>CAN_INT_BO</i>	bus-off interrupt enable
<i>CAN_INT_ERRN</i>	error number interrupt enable
<i>CAN_INT_ERR</i>	error interrupt enable
<i>CAN_INT_WU</i>	wakeup interrupt enable
<i>CAN_INT_SLPW</i>	sleep working interrupt enable
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* CAN0 transmit mailbox empty interrupt disable */
```

```
can_interrupt_disable (CAN0, CAN_INT_TME);
```

can_interrupt_flag_get

The description of can_interrupt_flag_get is shown as below:

Table 3-87. Function can_interrupt_flag_get

Function name	can_interrupt_flag_get
Function prototype	FlagStatus can_interrupt_flag_get(uint32_t can_periph, can_interrupt_flag_enum flag);
Function descriptions	get CAN interrupt flag state
Precondition	-
The called functions	-
Input parameter{in}	
can_periph	CAN peripheral
<i>CANx(x=0,1)</i>	CAN peripheral selection
Input parameter{in}	
flag	CAN interrupt flags
<i>CAN_INT_FLAG_SLPIF</i>	status change interrupt flag of sleep working mode entering
<i>CAN_INT_FLAG_WUIF</i>	status change interrupt flag of wakeup from sleep working mode
<i>CAN_INT_FLAG_ERRIF</i>	error interrupt flag
<i>CAN_INT_FLAG_MTF2</i>	mailbox 2 transmit finished interrupt flag
<i>CAN_INT_FLAG_MTF1</i>	mailbox 1 transmit finished interrupt flag
<i>CAN_INT_FLAG_MTF0</i>	mailbox 0 transmit finished interrupt flag
<i>CAN_INT_FLAG_RFO0</i>	receive FIFO0 overfull interrupt flag
<i>CAN_INT_FLAG_RFF0</i>	receive FIFO0 full interrupt flag
<i>CAN_INT_FLAG_RFO1</i>	receive FIFO1 overfull interrupt flag

<i>CAN_INT_FLAG_RFF1</i>	receive FIFO1 full interrupt flag
Output parameter{out}	
-	-
Return value	
FlagStatus	SET / RESET

Example:

```
/* get CAN0 mailbox 0 transmit finished interrupt flag */
can_interrupt_flag_get (CAN0, CAN_INT_FLAG_MTF0);
```

can_interrupt_flag_clear

The description of can_interrupt_flag_clear is shown as below:

Table 3-88. Function can_interrupt_flag_clear

Function name	can_interrupt_flag_clear
Function prototype	void can_interrupt_flag_clear(uint32_t can_periph, can_interrupt_flag_enum flag);
Function descriptions	clear CAN interrupt flag state
Precondition	-
The called functions	-
Input parameter{in}	
can_periph	CAN peripheral
<i>CANx(x=0,1)</i>	CAN peripheral selection
Input parameter{in}	
flag	CAN interrupt flags
<i>CAN_INT_FLAG_SLPIF</i>	status change interrupt flag of sleep working mode entering
<i>CAN_INT_FLAG_WUIF</i>	status change interrupt flag of wakeup from sleep working mode
<i>CAN_INT_FLAG_ERRIF</i>	error interrupt flag
<i>CAN_INT_FLAG_MTF2</i>	mailbox 2 transmit finished interrupt flag
<i>CAN_INT_FLAG_MTF1</i>	mailbox 1 transmit finished interrupt flag
<i>CAN_INT_FLAG_MTF0</i>	mailbox 0 transmit finished interrupt flag
<i>CAN_INT_FLAG_RFO0</i>	receive FIFO0 overfull interrupt flag
<i>CAN_INT_FLAG_RFF0</i>	receive FIFO0 full interrupt flag
<i>CAN_INT_FLAG_RFO1</i>	receive FIFO1 overfull interrupt flag
<i>CAN_INT_FLAG_RFF1</i>	receive FIFO1 full interrupt flag
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* clear CAN0 mailbox 0 transmit finished interrupt flag */
can_interrupt_flag_clear (CAN0, CAN_INT_FLAG_MTF0);
```

3.4. 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 [3.4.1](#) the CAU firmware functions are introduced in chapter [3.4.2](#)

3.4.1. Descriptions of Peripheral registers

CAU registers are listed in the table shown as below:

Table 3-89. CAU Registers

Registers	Descriptions
CAU_CTL	control register
CAU_STAT0	status register 0
CAU_DI	data input register
CAU_DO	data output register
CAU_DMAEN	DMA enable register
CAU_INTEN	interrupt enable register
CAU_STAT1	status register 1
CAU_INTF	interrupt flag register
CAU_KEY0H	key 0 high register
CAU_KEY0L	key 0 low register
CAU_KEY1H	key 1 high register
CAU_KEY1L	key 1 low register
CAU_KEY2H	key 2 high register
CAU_KEY2L	key 2 low register
CAU_KEY3H	key 3 high register
CAU_KEY3L	key 3 low register
CAU_IV0H	initial vector 0 high register
CAU_IV0L	initial vector 0 low register
CAU_IV1H	initial vector 1 high register
CAU_IV1L	initial vector 1 low register
CAU_GCMCCMCT XSx (x = 0..7)	GCM or CCM mode context switch register x
CAU_GCMCTXSx (x = 0..7)	GCM mode context switch register x

3.4.2. Descriptions of Peripheral functions

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

Table 3-90. CAU firmware function

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

Structure cau_key_parameter_struct

Table 3-91. Structure cau_key_parameter_struct

Member name	Function description
-------------	----------------------

key_0_high	key 0 high
key_0_low	key 0 low
key_1_high	key 1 high
key_1_low	key 1 low
key_2_high	key 2 high
key_2_low	key 2 low
key_3_high	key 3 high
key_3_low	key 3 low

Structure cau_iv_parameter_struct

Table 3-92. Structure cau_iv_parameter_struct

Member name	Function description
iv_0_high	init vector 0 high
iv_0_low	init vector 0 low
iv_1_high	init vector 1 high
iv_1_low	init vector 1 low

Structure cau_context_parameter_struct

Table 3-93. Structure cau_context_parameter_struct

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

Structure cau_parameter_struct

Table 3-94. Structure cau_parameter_struct

Member name	Function description
alg_dir	algorithm directory
*key	key

key_size	key size in bytes
*iv	initialization vector
iv_size	iv size in bytes
*input	input data
in_length	input data length in bytes
*aad	additional authentication data
aad_size	aad size

cau_deinit

The description of cau_deinit is shown as below:

Table 3-95. Function cau_deinit

Function name	cau_deinit
Function prototype	void cau_deinit(void);
Function descriptions	reset the CAU peripheral
Precondition	-
The called functions	rcu_periph_reset_enable / rcu_periph_reset_disable
Input parameter{in}	
-	-
Output parameter{out}	
-	-
Return value	
-	-

Example:

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

cau_struct_para_init

The description of cau_struct_para_init is shown as below:

Table 3-96. Function cau_struct_para_init

Function name	cau_struct_para_init
Function prototype	void cau_struct_para_init(cau_parameter_struct *cau_parameter);
Function descriptions	initialize the CAU encrypt and decrypt parameter struct with the default values
Precondition	-
The called functions	-
Input parameter{in}	
-	-
Output parameter{out}	
cau_parameter	structure for encrypt and decrypt parameters, refer to structure Table 3-94. Structure cau_parameter_struct

Return value	
-	-

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 3-97. Function cau_key_struct_para_init

Function name	cau_key_struct_para_init
Function prototype	void cau_key_struct_para_init(cau_key_parameter_struct *key_initpara);
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 3-91. 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 3-98. Function cau_iv_struct_para_init

Function name	cau_iv_struct_para_init
Function prototype	void cau_iv_struct_para_init(cau_iv_parameter_struct *iv_initpara);
Function descriptions	initialize the vectors parameter struct with the default values
Precondition	-
The called functions	-
Input parameter{in}	
-	-

Output parameter{out}	
iv_initpara	structure for vectors initialization of the cau, refer to structure Table 3-92. Structure cau_iv_parameter_struct
Return value	
-	-

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 3-99. Function cau_context_struct_para_init

Function name	cau_context_struct_para_init
Function prototype	void cau_context_struct_para_init(cau_context_parameter_struct *cau_context);
Function descriptions	initialize the context parameter struct with the default values
Precondition	-
The called functions	-
Input parameter{in}	
-	-
Output parameter{out}	
cau_context	structure for cau context swapping, refer to structure Table 3-93. Structure cau_context_parameter_struct
Return value	
-	-

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 3-100. Function cau_enable

Function name	cau_enable
Function prototype	void cau_enable(void);
Function descriptions	enable the CAU peripheral

Precondition	-
The called functions	-
Input parameter{in}	
-	-
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* enable the CAU peripheral */
```

```
cau_enable();
```

cau_disable

The description of cau_disable is shown as below:

Table 3-101. Function cau_disable

Function name	cau_disable
Function prototype	void cau_disable(void);
Function descriptions	disable the CAU peripheral
Precondition	-
The called functions	-
Input parameter{in}	
-	-
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* disable the CAU peripheral */
```

```
cau_disable();
```

cau_dma_enable

The description of cau_dma_enable is shown as below:

Table 3-102. Function cau_dma_enable

Function name	cau_dma_enable
Function prototype	void cau_dma_enable(uint32_t dma_req);
Function descriptions	enable the CAU DMA interface
Precondition	-
The called functions	-

Input parameter{in}	
dma_req	specify the CAU DMA transfer request to be enabled
<i>CAU_DMA_INFIFO</i>	DMA for incoming(Rx) data transfer
<i>CAU_DMA_OUTFIFO</i>	DMA for outgoing(Tx) data transfer
Output parameter{out}	
-	-
Return value	
-	-

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 3-103. Function cau_dma_disable

Function name	cau_dma_disable
Function prototype	void cau_dma_disable(uint32_t dma_req);
Function descriptions	disable the CAU DMA interface
Precondition	-
The called functions	-
Input parameter{in}	
dma_req	specify the CAU DMA transfer request to be disabled
<i>CAU_DMA_INFIFO</i>	DMA for incoming(Rx) data transfer
<i>CAU_DMA_OUTFIFO</i>	DMA for outgoing(Tx) data transfer
Output parameter{out}	
-	-
Return value	
-	-

Example:

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

cau_init

The description of cau_init is shown as below:

Table 3-104. Function cau_init

Function name	cau_init
Function prototype	void cau_init(uint32_t algo_dir, uint32_t algo_mode, uint32_t swapping);
Function descriptions	initialize the CAU peripheral

Precondition	-
The called functions	-
Input parameter{in}	
algo_dir	algorithm direction
<i>CAU_ENCRYPT</i>	encrypt
<i>CAU_DECRYPT</i>	decrypt
Input parameter{in}	
algo_mode	algorithm mode selection
<i>CAU_MODE_TDES_ECB</i>	TDES-ECB (3DES Electronic codebook)
<i>CAU_MODE_TDES_CBC</i>	TDES-CBC (3DES Cipher block chaining)
<i>CAU_MODE_DES_ECB</i>	DES-ECB (simple DES Electronic codebook)
<i>CAU_MODE_DES_CBC</i>	DES-CBC (simple DES Cipher block chaining)
<i>CAU_MODE_AES_ECB</i>	AES-ECB (AES Electronic codebook)
<i>CAU_MODE_AES_CBC</i>	AES-CBC (AES Cipher block chaining)
<i>CAU_MODE_AES_CTR</i>	AES-CTR (AES counter mode)
<i>CAU_MODE_AES_KEY</i>	AES decryption key preparation mode
<i>CAU_MODE_AES_GCM</i>	AES-GCM (AES Galois/counter mode)
<i>CAU_MODE_AES_CCM</i>	AES-CCM (AES combined cipher machine mode)
<i>CAU_MODE_AES_CFB</i>	AES-CFB (cipher feedback mode)
<i>CAU_MODE_AES_OFB</i>	AES-OFB (output feedback mode)
Input parameter{in}	
swapping	data swapping selection
<i>CAU_SWAPPING_32BIT</i>	no swapping
<i>CAU_SWAPPING_16BIT</i>	half-word swapping
<i>CAU_SWAPPING_8BIT</i>	bytes swapping
<i>CAU_SWAPPING_1BIT</i>	bit swapping
Output parameter{out}	
-	-
Return value	

-	-
---	---

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 3-105. Function cau_aes_keysize_config

Function name	cau_aes_keysize_config
Function prototype	void cau_aes_keysize_config(uint32_t key_size);
Function descriptions	configure key size if used AES algorithm
Precondition	-
The called functions	-
Input parameter{in}	
key_size	key length selection when aes mode
CAU_KEYSIZE_128BIT	128 bit key length
CAU_KEYSIZE_192BIT	192 bit key length
CAU_KEYSIZE_256BIT	256 bit key length
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 3-106. Function cau_key_init

Function name	cau_key_init
Function prototype	void cau_key_init(cau_key_parameter_struct* key_initpara);
Function descriptions	initialize the key parameters
Precondition	-
The called functions	-
Input parameter{in}	
key_initpara	structure for keys initialization of the cau, refer to structure Table 3-91. 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 3-107. Function cau_iv_init

Function name	cau_iv_init
Function prototype	void cau_iv_init(cau_iv_parameter_struct* iv_initpara);
Function descriptions	initialize the vectors parameters
Precondition	-
The called functions	-
Input parameter{in}	
iv_initpara	structure for vectors initialization of the cau, refer to structure Table 3-92. Structure cau_iv_parameter_struct
Output parameter{out}	
-	-
Return value	
-	-

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 3-108. Function cau_phase_config

Function name	cau_phase_config
Function prototype	void cau_phase_config(uint32_t phase);
Function descriptions	configure phase
Precondition	-
The called functions	-
Input parameter{in}	
phase	gcm or ccm phase
<i>CAU_PREPARE_PHASE</i>	prepare phase
<i>CAU_AAD_PHASE</i>	AAD phase
<i>CAU_ENCRYPT_DECRYPT_PHASE</i>	encryption/decryption phase
<i>CAU_TAG_PHASE</i>	tag phase
Output parameter{out}	
-	-
Return value	
-	-

Example:

```

/* select prepare phase */

cau_phase_config(CAU_PREPARE_PHASE);

```

cau_fifo_flush

The description of cau_fifo_flush is shown as below:

Table 3-109. Function cau_fifo_flush

Function name	cau_fifo_flush
Function prototype	void cau_fifo_flush(void);
Function descriptions	flush the IN and OUT FIFOs
Precondition	-
The called functions	-
Input parameter{in}	

-	-
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* 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 3-110. Function cau_enable_state_get

Function name	cau_enable_state_get
Function prototype	ControlStatus cau_enable_state_get(void);
Function descriptions	return whether CAU peripheral is enabled or disabled
Precondition	-
The called functions	-
Input parameter{in}	
-	-
Output parameter{out}	
-	-
Return value	
ControlStatus	ENABLE or DISABLE

Example:

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

```
ControlStatus state = DISABLE;
```

```
State = cau_enable_state_get();
```

cau_data_write

The description of cau_data_write is shown as below:

Table 3-111. Function cau_data_write

Function name	cau_data_write
Function prototype	void cau_data_write(uint32_t data);
Function descriptions	write data to the IN FIFO
Precondition	-
The called functions	-
Input parameter{in}	
data	data to write: 0 - 0xFFFFFFFF

Output parameter{out}	
-	-
Return value	
-	-

Example:

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

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

cau_data_read

The description of cau_data_read is shown as below:

Table 3-112. Function cau_data_read

Function name	cau_data_read
Function prototype	uint32_t cau_data_read(void);
Function descriptions	return the last data entered into the output FIFO
Precondition	-
The called functions	-
Input parameter{in}	
-	-
Output parameter{out}	
-	-
Return value	
uint32_t	0x0 – 0xFFFFFFFF

Example:

```
/* 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 3-113. Function cau_context_save

Function name	cau_context_save
Function prototype	void cau_context_save(cau_context_parameter_struct *cau_context, cau_key_parameter_struct* key_initpara);
Function descriptions	save context before context switching
Precondition	-
The called functions	-
Input parameter{in}	
key_initpara	structure for keys initialization of the cau, refer to structure Table 3-91.

	Structure cau_key_parameter_struct
Output parameter{out}	
cau_context	structure for cau context swapping, refer to structure Table 3-93. Structure cau_context_parameter_struct
Return value	
-	-

Example:

```
cau_context_parameter_struct context;

cau_key_parameter_struct key;

cau_parameter_struct cau_parameter;

uint32_t keyaddr;

.....

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

cau_key_struct_para_init(&key);

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

keyaddr += 4U;

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

/* save context before context switching */

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

cau_context_restore

The description of cau_context_restore is shown as below:

Table 3-114. Function cau_context_restore

Function name	cau_context_restore
Function prototype	void cau_context_restore(cau_context_parameter_struct *cau_context);
Function descriptions	restore context after context switching
Precondition	-
The called functions	-
Input parameter{in}	
cau_context	structure for cau context swapping, refer to structure Table 3-93. Structure cau_context_parameter_struct
Output parameter{out}	
-	-
Return value	
-	-

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 3-115. Function cau_aes_ecb

Function name	cau_aes_ecb
Function prototype	ErrStatus cau_aes_ecb(cau_parameter_struct *cau_parameter, uint8_t *output);
Function descriptions	encrypt and decrypt using AES in ECB mode
Precondition	-
The called functions	-
Input parameter{in}	
cau_parameter	structure for encrypt and decrypt parameters, refer to structure Table 3-94. 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.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 3-116. Function cau_aes_cbc

Function name	cau_aes_cbc
Function prototype	ErrStatus cau_aes_cbc(cau_parameter_struct *cau_parameter, uint8_t *output);
Function descriptions	encrypt and decrypt using AES in CBC mode
Precondition	-
The called functions	-
Input parameter{in}	
cau_parameter	structure for encrypt and decrypt parameters, refer to structure Table 3-94. 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 CBC mode */
```

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

cau_aes_ctr

The description of cau_aes_ctr is shown as below:

Table 3-117. Function cau_aes_ctr

Function name	cau_aes_ctr
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 3-94. 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 3-118. Function cau_aes_cfb

Function name	cau_aes_cfb
Function prototype	ErrStatus cau_aes_cfb(cau_parameter_struct *cau_parameter, uint8_t *output);
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 3-94. Structure cau_parameter_struct
Output parameter{out}	
output	pointer to the returned buffer
Return value	
ErrStatus	SUCCESS or ERROR

Example:

```
cau_parameter_struct cau_cfb_parameter;

uint8_t encrypt_result[TEXT_SIZE];

ErrStatus status;

.....

/* encryption in CFB mode */

cau_cfb_parameter.alg_dir = CAU_ENCRYPT;

cau_cfb_parameter.key = (uint8_t *)key_128;

cau_cfb_parameter.key_size = KEY_SIZE;

cau_cfb_parameter.iv = (uint8_t *)vectors;

cau_cfb_parameter.iv_size = IV_SIZE;

cau_cfb_parameter.input = (uint8_t *)plaintext;

cau_cfb_parameter.in_length = PLAINTEXT_SIZE;

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

cau_aes_ofb

The description of cau_aes_ofb is shown as below:

Table 3-119. Function cau_aes_ofb

Function name	cau_aes_ofb
Function prototype	ErrStatus cau_aes_ofb(cau_parameter_struct *cau_parameter, uint8_t *output);
Function descriptions	encrypt and decrypt using AES in OFB mode
Precondition	-
The called functions	-
Input parameter{in}	
cau_parameter	structure for encrypt and decrypt parameters, refer to structure Table 3-94. Structure cau_parameter_struct
Output parameter{out}	
output	pointer to the returned buffer
Return value	
ErrStatus	SUCCESS or ERROR

Example:

```
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 3-120. Function cau_aes_gcm

Function name	cau_aes_gcm
Function prototype	ErrStatus cau_aes_gcm(cau_parameter_struct *cau_parameter, uint8_t *output, uint8_t *tag);
Function descriptions	encrypt and decrypt using AES in GCM mode

Precondition	-
The called functions	-
Input parameter{in}	
cau_parameter	structure for encrypt and decrypt parameters, refer to structure Table 3-94. Structure cau_parameter_struct
Output parameter{out}	
output	pointer to the returned buffer
Output parameter{out}	
tag	pointer to the returned tag buffer
Return value	
ErrStatus	SUCCESS or ERROR

Example:

```
cau_parameter_struct cau_gcm_parameter;

uint8_t encrypt_result[TEXT_SIZE];

uint8_t gcm_tag[GCM_TAG_SIZE];

ErrStatus status;

.....

/* encryption in GCM mode */

cau_gcm_parameter.alg_dir = CAU_ENCRYPT;

cau_gcm_parameter.key = (uint8_t *)key_128;

cau_gcm_parameter.key_size = KEY_SIZE;

cau_gcm_parameter.iv = (uint8_t *)vectors;

cau_gcm_parameter.iv_size = IV_SIZE;

cau_gcm_parameter.input = (uint8_t *)plaintext;

cau_gcm_parameter.in_length = PLAINTEXT_SIZE;

cau_gcm_parameter.aad = (uint8_t *)aadmessage;

cau_gcm_parameter.aad_size = AAD_SIZE;

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

cau_aes_ccm

The description of cau_aes_ccm is shown as below:

Table 3-121. Function cau_aes_ccm

Function name	cau_aes_ccm
Function prototype	ErrStatus cau_aes_ccm(cau_parameter_struct *cau_parameter, uint8_t

	*output, uint8_t tag[], uint32_t tag_size, uint8_t aad_buf[]);
Function descriptions	encrypt and decrypt using AES in CCM mode
Precondition	-
The called functions	-
Input parameter{in}	
cau_parameter	structure for encrypt and decrypt parameters, refer to structure Table 3-94. Structure cau_parameter_struct
Input parameter{in}	
tag_size	tag size (in bytes)
Output parameter{out}	
output	pointer to the returned buffer
Output parameter{out}	
tag	pointer to the returned tag buffer
Output parameter{out}	
aad_buf	pointer to the user buffer used when formatting aad block
Return value	
ErrStatus	SUCCESS or ERROR

Example:

```

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 3-122. Function cau_tdes_ecb

Function name	cau_tdes_ecb
Function prototype	ErrStatus cau_tdes_ecb(cau_parameter_struct *cau_parameter, uint8_t *output);
Function descriptions	encrypt and decrypt using TDES in ECB mode
Precondition	-
The called functions	-
Input parameter{in}	
cau_parameter	structure for encrypt and decrypt parameters, refer to structure Table 3-94. 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[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 3-123. Function cau_tdes_cbc

Function name	cau_tdes_cbc
Function prototype	ErrStatus cau_tdes_cbc(cau_parameter_struct *cau_parameter, uint8_t *output);

Function descriptions	encrypt and decrypt using TDES in CBC mode
Precondition	-
The called functions	-
Input parameter{in}	
cau_parameter	structure for encrypt and decrypt parameters, refer to structure Table 3-94. 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[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 3-124. Function cau_des_ecb

Function name	cau_des_ecb
Function prototype	ErrStatus cau_des_ecb(cau_parameter_struct *cau_parameter, uint8_t *output);
Function descriptions	encrypt and decrypt using DES in ECB mode
Precondition	-
The called functions	-
Input parameter{in}	
cau_parameter	structure for encrypt and decrypt parameters, refer to structure Table 3-94. 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[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 3-125. Function cau_des_cbc

Function name	cau_des_cbc
Function prototype	ErrStatus cau_des_cbc(cau_parameter_struct *cau_parameter, uint8_t *output);
Function descriptions	encrypt and decrypt using DES in CBC mode
Precondition	-
The called functions	-
Input parameter{in}	
cau_parameter	structure for encrypt and decrypt parameters, refer to structure Table 3-94. 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[DATA_SIZE];

ErrStatus status;

.....
```

```

text.alg_dir = CAU_ENCRYPT;

text.key = des_key;

text.iv = vectors;

text.input = plaintext;

text.in_length = DATA_SIZE;

/* encryption in CBC mode */

status = cau_des_cbc(&text, encrypt_result);

```

cau_flag_get

The description of cau_flag_get is shown as below:

Table 3-126. Function cau_flag_get

Function name	cau_flag_get
Function prototype	FlagStatus cau_flag_get(uint32_t flag);
Function descriptions	get the CAU flag status
Precondition	-
The called functions	-
Input parameter{in}	
flag	CAU flag status
CAU_FLAG_INFIFO_EMPTY	input FIFO empty
CAU_FLAG_INFIFO_NOT_FULL	input FIFO is not full
CAU_FLAG_OUTFIFO_NOT_EMPTY	output FIFO not empty
CAU_FLAG_OUTFIFO_FULL	output FIFO is full
CAU_FLAG_BUSY	the CAU core is busy
CAU_FLAG_INFIFO	input FIFO flag status
CAU_FLAG_OUTFIFO	output FIFO flag status
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);

```


cau_interrupt_enable

The description of cau_interrupt_enable is shown as below:

Table 3-127. Function cau_interrupt_enable

Function name	cau_interrupt_enable
Function prototype	void cau_interrupt_enable(uint32_t interrupt);
Function descriptions	enable the CAU interrupts
Precondition	-
The called functions	-
Input parameter{in}	
interrupt	specify the CAU interrupt source to be enabled
CAU_INT_INFIFO	input FIFO interrupt
CAU_INT_OUTFIFO	output FIFO interrupt
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 3-128. Function cau_interrupt_disable

Function name	cau_interrupt_disable
Function prototype	void cau_interrupt_disable(uint32_t interrupt);
Function descriptions	disable the CAU interrupts
Precondition	-
The called functions	-
Input parameter{in}	
interrupt	specify the CAU interrupt source to be disabled
CAU_INT_INFIFO	input FIFO interrupt
CAU_INT_OUTFIFO	output FIFO interrupt
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* 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 3-129. Function cau_interrupt_flag_get

Function name	cau_interrupt_flag_get
Function prototype	FlagStatus cau_interrupt_flag_get(uint32_t int_flag);
Function descriptions	get the interrupt flag
Precondition	-
The called functions	-
Input parameter{in}	
int_flag	CAU interrupt flag
CAU_INT_FLAG_INFIFO	input FIFO interrupt
CAU_INT_FLAG_OUTFIFO	output FIFO interrupt
Output parameter{out}	
-	-
Return value	
FlagStatus	SET or RESET

Example:

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

```
FlagStatus status = RESET;
```

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

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 3-130. CRC Registers

Registers	Descriptions
CRC_DATA	CRC data register
CRC_FDATA	CRC free data register
CRC_CTL	CRC control register

3.5.2. Descriptions of Peripheral functions

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

Table 3-131. CRC firmware function

Function name	Function description
crc_deinit	deinit CRC calculation unit
crc_data_register_reset	reset data register(CRC_DATA) to the value of 0xFFFFFFFF
crc_data_register_read	read the value of the data register
crc_free_data_register_read	read the value of the free data register
crc_free_data_register_write	write data to the free data register
crc_single_data_calculate	calculate the CRC value of a 32-bit data
crc_block_data_calculate	calculate the CRC value of an array of 32-bit values

crc_deinit

The description of crc_deinit is shown as below:

Table 3-132. Function crc_deinit

Function name	crc_deinit
Function prototype	void crc_deinit(void);
Function descriptions	deinit CRC calculation unit
Precondition	-
The called functions	-
Input parameter{in}	
-	-
Output parameter{out}	
-	-
Return value	
-	-

Example:

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

crc_data_register_reset

The description of crc_data_register_reset is shown as below:

Table 3-133. Function crc_data_register_reset

Function name	crc_data_register_reset
Function prototype	void crc_data_register_reset(void);
Function descriptions	reset data register(CRC_DATA) to the value of 0xFFFFFFFF
Precondition	-
The called functions	-

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 3-134. Function crc_data_register_read

Function name	crc_data_register_read
Function prototype	uint32_t crc_data_register_read(void);
Function descriptions	read the value of the data register
Precondition	-
The called functions	-
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 3-135. Function crc_free_data_register_read

Function name	crc_free_data_register_read
Function prototype	uint8_t crc_free_data_register_read(void);
Function descriptions	read the value of the free data register
Precondition	-
The called functions	-
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 3-136. Function crc_free_data_register_write

Function name	crc_free_data_register_write
Function prototype	void crc_free_data_register_write(uint8_t free_data);
Function descriptions	write data to the free data register
Precondition	-
The called functions	-
Input parameter{in}	
free_data	specified 8-bit data
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* write the free data register */

crc_free_data_register_write(0x11);
```

crc_single_data_calculate

The description of crc_single_data_calculate is shown as below:

Table 3-137. Function crc_single_data_calculate

Function name	crc_single_data_calculate
Function prototype	uint32_t crc_single_data_calculate(uint32_t sdata);
Function descriptions	calculate the CRC value of a 32-bit data
Precondition	-
The called functions	-
Input parameter{in}	
sdata	specified 32-bit data

Output parameter{out}	
-	-
Return value	
uint32_t	32-bit value calculated by CRC (0-0xFFFFFFFF)

Example:

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

uint32_t val = 0, valcrc = 0;

val = (uint32_t)0xabcd1234;

valcrc = crc_single_data_calculate(val);
```

crc_block_data_calculate

The description of crc_block_data_calculate is shown as below:

Table 3-138. Function crc_block_data_calculate

Function name	crc_block_data_calculate
Function prototype	uint32_t crc_block_data_calculate(uint32_t array[], uint32_t size);
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 values
Input parameter{in}	
size	size of the array
Output parameter{out}	
-	-
Return value	
uint32_t	32-bit value calculated by CRC (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);
```

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 3-139. CTC Registers

Registers	Descriptions
CTC_CTL0	CTC control register 0
CTC_CTL1	CTC control register 1
CTC_STAT	CTC status register
CTC_INTC	CTC Interrupt clear register

3.6.2. Descriptions of Peripheral functions

CTC registers are listed in the table shown as below:

Table 3-140. CTC firmware function

Function name	Function description
ctc_deinit	reset CTC clock trim controller
ctc_counter_enable	enable CTC trim counter
ctc_counter_disable	disable CTC trim counter
ctc_irc48m_trim_value_config	configure the IRC48M trim value
ctc_software_refsource_pulse_generate	generate software reference source sync pulse
ctc_hardware_trim_mode_config	configure hardware automatically trim mode
ctc_refsource_polarity_config	configure reference signal source polarity
ctc_refsource_signal_select	select reference signal source
ctc_refsource_prescaler_config	configure reference signal source prescaler
ctc_clock_limit_value_config	configure clock trim base limit value
ctc_counter_reload_value_config	configure CTC counter reload value
ctc_counter_capture_value_read	read CTC counter capture value when reference sync pulse occurred
ctc_counter_direction_read	read CTC trim counter direction when reference sync pulse occurred
ctc_counter_reload_value_read	read CTC counter reload value
ctc_irc48m_trim_value_read	read the IRC48M trim value
ctc_interrupt_enable	enable the CTC interrupt
ctc_interrupt_disable	disable the CTC interrupt

Function name	Function description
ctc_interrupt_flag_get	get CTC interrupt flag
ctc_interrupt_flag_clear	clear CTC interrupt flag
ctc_flag_get	get CTC flag
ctc_flag_clear	clear CTC flag

ctc_deinit

The description of ctc_deinit is shown as below:

Table 3-141. Function ctc_deinit

Function name	ctc_deinit
Function prototype	void ctc_deinit (void)
Function descriptions	Reset CTC peripheral
Precondition	-
The called functions	rcu_periph_reset_enable / rcu_periph_reset_disable
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 3-142. Function ctc_counter_enable

Function name	ctc_counter_enable
Function prototype	void ctc_counter_enable (void);
Function descriptions	enable CTC counter
Precondition	-
The called functions	-
Input parameter{in}	
-	-
Output parameter{out}	
-	-
Return value	
-	-

Example:


```
/* enable CTC trim counter*/
```

```
ctc_counter_enable();
```

ctc_counter_disable

The description of ctc_counter_disable is shown as below:

Table 3-143. Function ctc_counter_disable

Function name	ctc_counter_disable
Function prototype	void ctc_counter_disable (void);
Function descriptions	disable CTC counter
Precondition	-
The called functions	-
Input parameter{in}	
-	-
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* disable CTC trim counter */
```

```
ctc_counter_disable();
```

ctc_irc48m_trim_value_config

The description of ctc_irc48m_trim_value_config is shown as below:

Table 3-144. Function ctc_irc48m_trim_value_config

Function name	ctc_irc48m_trim_value_config
Function prototype	void ctc_irc48m_trim_value_config(uint8_t trim_value);
Function descriptions	configure the IRC48M trim value
Precondition	-
The called functions	-
Input parameter{in}	
trim_value	0 ~ 63
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* 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 3-145. Function ctc_software_refsource_pulse_generate

Function name	ctc_software_refsource_pulse_generate
Function prototype	void ctc_software_refsource_pulse_generate (void)
Function descriptions	generate software reference source sync pulse
Precondition	-
The called functions	-
Input parameter{in}	
-	-
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* 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 3-146. Function ctc_hardware_trim_mode_config

Function name	ctc_hardware_trim_mode_config
Function prototype	void ctc_hardware_trim_mode_config(uint32_t hardmode);
Function descriptions	configure hardware automatically trim mode
Precondition	-
The called functions	-
Input parameter{in}	
hardmode	hardware automatically trim mode enable or disable
CTC_HARDWARE_TRIM_MODE_ENABLE	hardware automatically trim mode enable
CTC_HARDWARE_TRIM_MODE_DISABLE	hardware automatically trim mode disable
Output parameter{out}	
-	-
Return value	
-	-

Example:

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

```
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 3-147. Function ctc_refsource_polarity_config

Function name	ctc_refsource_polarity_config
Function prototype	void ctc_refsource_polarity_config(uint32_t polarity);
Function descriptions	configure reference signal source polarity
Precondition	-
The called functions	-
Input parameter{in}	
polarity	reference signal source polarity
<i>CTC_REFSOURCE_POLARITY_FALLING</i>	reference signal source polarity is falling edge
<i>CTC_REFSOURCE_POLARITY_RISING</i>	reference signal source polarity is rising edge
Output parameter{out}	
-	-
Return value	
-	-

Example:

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

```
ctc_refsource_polarity_config(CTC_REFSOURCE_POLARITY_RISING);
```

ctc_refsource_signal_select

The description of ctc_refsource_signal_select is shown as below:

Table 3-148. Function ctc_refsource_signal_select

Function name	ctc_refsource_signal_select
Function prototype	void ctc_refsource_signal_select(uint32_t refs);
Function descriptions	select reference signal source
Precondition	-
The called functions	-
Input parameter{in}	
refs	reference signal source
<i>CTC_REFSOURCE_GPIO</i>	GPIO is selected
<i>CTC_REFSOURCE_LXTAL</i>	LXTAL is selected
Output parameter{out}	
-	-

Return value	
-	-

Example:

```
/* 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 3-149. Function ctc_refsource_prescaler_config

Function name	ctc_refsource_prescaler_config
Function prototype	void ctc_refsource_prescaler_config(uint32_t prescaler);
Function descriptions	configure reference signal source prescaler
Precondition	-
The called functions	-
Input parameter{in}	
prescaler	Prescaler factor
CTC_REFSOURCE_P SC_OFF	reference signal not divided
CTC_REFSOURCE_P SC_DIV2	reference signal divided by 2
CTC_REFSOURCE_P SC_DIV4	reference signal divided by 4
CTC_REFSOURCE_P SC_DIV8	reference signal divided by 8
CTC_REFSOURCE_P SC_DIV16	reference signal divided by 16
CTC_REFSOURCE_P SC_DIV32	reference signal divided by 32
CTC_REFSOURCE_P SC_DIV64	reference signal divided by 64
CTC_REFSOURCE_P SC_DIV128	reference signal divided by 128
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* 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 3-150. Function ctc_clock_limit_value_config

Function name	ctc_clock_limit_value_config
Function prototype	void ctc_clock_limit_value_config(uint8_t limit_value);
Function descriptions	configure clock trim base limit value
Precondition	-
The called functions	-
Input parameter{in}	
limit_value	0x00 - 0xFF
Output parameter{out}	
-	-
Return value	
-	-

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 3-151. Function ctc_counter_reload_value_config

Function name	ctc_counter_reload_value_config
Function prototype	void ctc_counter_reload_value_config(uint16_t reload_value);
Function descriptions	configure CTC counter reload value
Precondition	-
The called functions	-
Input parameter{in}	
reload_value	0x0000 - 0xFFFF
Output parameter{out}	
-	-
Return value	
-	-

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 3-152. Function ctc_counter_capture_value_read

Function name	ctc_counter_capture_value_read
Function prototype	uint16_t ctc_counter_capture_value_read(void);
Function descriptions	read CTC counter capture value
Precondition	-
The called functions	-
Input parameter{in}	
-	-
Output parameter{out}	
-	-
Return value	
uint16_t	the 16-bit CTC counter capture value (0x0000 - 0xFFFF)

Example:

```
/* 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 3-153. Function ctc_counter_direction_read

Function name	ctc_counter_direction_read
Function prototype	FlagStatus ctc_counter_direction_read(void);
Function descriptions	read CTC trim counter direction
Precondition	-
The called functions	-
Input parameter{in}	
-	-
Output parameter{out}	
-	-
Return value	
FlagStatus	SET / RESET

Example:

```
/* 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 3-154. Function ctc_counter_reload_value_read

Function name	ctc_counter_reload_value_read
Function prototype	uint16_t ctc_counter_reload_value_read(void);
Function descriptions	read CTC counter reload value
Precondition	-
The called functions	-
Input parameter{in}	
-	-
Output parameter{out}	
-	-
Return value	
uint16_t	Read 16-bit data of counter reload value (0x0000 - 0xFFFF)

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 3-155. Function ctc_irc48m_trim_value_read

Function name	ctc_irc48m_trim_value_read
Function prototype	uint8_t ctc_irc48m_trim_value_read(void);
Function descriptions	read the IRC48M trim value
Precondition	-
The called functions	-
Input parameter{in}	
-	-
Output parameter{out}	
-	-
Return value	
uint8_t	the 6-bit IRC48M trim value (0-63)

Example:

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

uint8_t ctc_trim_value = 0;

ctc_trim_value = ctc_irc48m_trim_value_read();
```

ctc_interrupt_enable

The description of ctc_interrupt_enable is shown as below:

Table 3-156. Function`ctc_interrupt_enable`

Function name	<code>ctc_interrupt_enable</code>
Function prototype	<code>void ctc_interrupt_enable(uint32_t interrupt);</code>
Function descriptions	enable the CTC interrupt
Precondition	-
The called functions	-
Input parameter{in}	
interrupt	CTC interrupt
<code>CTC_INT_CKOK</code>	clock trim OK interrupt
<code>CTC_INT_CKWARN</code>	clock trim warning interrupt
<code>CTC_INT_ERR</code>	error interrupt
<code>CTC_INT_EREf</code>	expect reference interrupt
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* 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 3-157. Function`ctc_interrupt_disable`

Function name	<code>ctc_interrupt_disable</code>
Function prototype	<code>void ctc_interrupt_disable(uint32_t interrupt);</code>
Function descriptions	disable the CTC interrupt
Precondition	-
The called functions	-
Input parameter{in}	
interrupt	CTC interrupt
<code>CTC_INT_CKOK</code>	clock trim OK interrupt
<code>CTC_INT_CKWARN</code>	clock trim warning interrupt
<code>CTC_INT_ERR</code>	error interrupt
<code>CTC_INT_EREf</code>	expect reference interrupt
Output parameter{out}	
-	-
Return value	
-	-

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 3-158. Function ctc_interrupt_flag_get

Function name	ctc_interrupt_flag_get
Function prototype	FlagStatus ctc_interrupt_flag_get(uint32_t int_flag);
Function descriptions	get CTC interrupt flag
Precondition	-
The called functions	-
Input parameter{in}	
int_flag	CTC interrupt flag
<i>CTC_INT_FLAG_CKOK</i>	clock trim OK interrupt
<i>CTC_INT_FLAG_CKWARN</i>	clock trim warning interrupt
<i>CTC_INT_FLAG_ERR</i>	error interrupt
<i>CTC_INT_FLAG_EREFS</i>	expect reference interrupt
<i>CTC_INT_FLAG_CKEERR</i>	clock trim error bit interrupt
<i>CTC_INT_FLAG_REFMISS</i>	reference sync pulse miss interrupt
<i>CTC_INT_FLAG_TRIMERR</i>	trim value error interrupt
Output parameter{out}	
-	-
Return value	
FlagStatus	SET / RESET

Example:

```
/* 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 3-159. Function ctc_interrupt_flag_clear

Function name	ctc_interrupt_flag_clear
Function prototype	void ctc_interrupt_flag_clear(uint32_t int_flag);
Function descriptions	clear CTC interrupt flag
Precondition	-
The called functions	-

Input parameter{in}	
int_flag	CTC interrupt flag
<i>CTC_INT_FLAG_CKOK</i>	clock trim OK interrupt
<i>CTC_INT_FLAG_CKWARN</i>	clock trim warning interrupt
<i>CTC_INT_FLAG_ERR</i>	error interrupt
<i>CTC_INT_FLAG_EREFP</i>	expect reference interrupt
<i>CTC_INT_FLAG_CKEERR</i>	clock trim error bit interrupt
<i>CTC_INT_FLAG_REFMISS</i>	reference sync pulse miss interrupt
<i>CTC_INT_FLAG_TRIMERR</i>	trim value error interrupt
Output parameter{out}	
-	-
Return value	
-	-

Example:

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

```
ctc_interrupt_flag_clear(CTC_INT_FLAG_CKOK);
```

ctc_flag_get

The description of ctc_flag_get is shown as below:

Table 3-160. Function ctc_flag_get

Function name	ctc_flag_get
Function prototype	FlagStatus ctc_flag_get(uint32_t flag);
Function descriptions	get CTC status flag
Precondition	-
The called functions	-
Input parameter{in}	
flag	CTC status flag
<i>CTC_FLAG_CKOK</i>	clock trim OK interrupt flag
<i>CTC_FLAG_CKWARN</i>	clock trim warning interrupt flag
<i>CTC_FLAG_ERR</i>	error interrupt flag
<i>CTC_FLAG_EREFP</i>	expect reference interrupt flag
<i>CTC_FLAG_CKERR</i>	clock trim error bit
<i>CTC_FLAG_REFMISS</i>	reference sync pulse miss flag
<i>CTC_FLAG_TRIMERR</i>	trim value error flag
Output parameter{out}	
-	-

Return value	
FlagStatus	SET / 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 3-161. Function ctc_flag_clear

Function name	ctc_flag_clear
Function prototype	void ctc_flag_clear (uint32_t flag);
Function descriptions	clear CTC status flag
Precondition	-
The called functions	-
Input parameter{in}	
flag	CTC status flag
CTC_FLAG_CKOK	clock trim OK interrupt flag
CTC_FLAG_CKWARN	clock trim warning interrupt flag
CTC_FLAG_ERR	error interrupt flag
CTC_FLAG_EREFP	expect reference interrupt flag
CTC_FLAG_CKERR	clock trim error bit
CTC_FLAG_REFMIS	reference sync pulse miss flag
CTC_FLAG_TRIMERR	trim value error flag
Output parameter{out}	
-	-
Return value	
-	-

Example:

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

```
ctc_flag_clear(CTC_FLAG_CKOK);
```

3.7. 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 [3.7.1](#) the DAC firmware functions are introduced in chapter [3.7.2](#).

3.7.1. Peripheral register description

DAC registers are listed in the table shown as below:

Table 3-162. DAC Registers

Register	Descriptions
DAC_CTL0	DACx control register 0
DAC_SWT	DACx software trigger register
DAC_OUT0_R12DH	DACx_OUT0 12-bit right-aligned data holding register
DAC_OUT0_L12DH	DACx_OUT0 12-bit left-aligned data holding register
DAC_OUT0_R8DH	DACx_OUT0 8-bit right-aligned data holding register
DAC_OUT1_R12DH	DACx_OUT1 12-bit right-aligned data holding register
DAC_OUT1_L12DH	DACx_OUT1 12-bit left-aligned data holding register
DAC_OUT1_R8DH	DACx_OUT1 8-bit right-aligned data holding register
DACC_R12DH	DACx concurrent mode 12-bit right-aligned data holding register
DACC_L12DH	DACx concurrent mode 12-bit left-aligned data holding register
DACC_R8DH	DACx concurrent mode 8-bit right-aligned data holding register
DAC_OUT0_DO	DACx_OUT0 data output register
DAC_OUT1_DO	DACx_OUT1 data output register
DAC_STAT0	DACx status register 0

3.7.2. Descriptions of Peripheral functions

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

Table 3-163. DAC firmware functions

Function name	Function description
dac_deinit	deinitialize DAC
dac_enable	enable DAC
dac_disable	disable DAC
dac_dma_enable	enable DAC DMA function
dac_dma_disable	disable DAC DMA function
dac_output_buffer_enable	enable DAC output buffer
dac_output_buffer_disable	disable DAC output buffer
dac_output_value_get	get DAC output value
dac_data_set	set DAC data holding register value
dac_trigger_enable	enable DAC trigger
dac_trigger_disable	disable DAC trigger
dac_trigger_source_config	configure DAC trigger source
dac_software_trigger_enable	enable DAC software trigger
dac_wave_mode_config	configure DAC wave mode
dac_lfsr_noise_config	configure DAC LFSR noise mode
dac_triangle_noise_config	configure DAC triangle noise mode
dac_concurrent_enable	enable DAC concurrent mode

Function name	Function description
<code>dac_concurrent_disable</code>	disable DAC concurrent mode
<code>dac_concurrent_software_trigger_enable</code>	enable DAC concurrent software trigger
<code>dac_concurrent_output_buffer_enable</code>	enable DAC concurrent buffer function
<code>dac_concurrent_output_buffer_disable</code>	disable DAC concurrent buffer function
<code>dac_concurrent_data_set</code>	set DAC concurrent mode data holding register value
<code>dac_flag_get</code>	get DAC flag
<code>dac_flag_clear</code>	clear DAC flag
<code>dac_interrupt_enable</code>	enable DAC interrupt
<code>dac_interrupt_disable</code>	disable DAC interrupt
<code>dac_interrupt_flag_get</code>	get DAC interrupt flag
<code>dac_interrupt_flag_clear</code>	clear DAC interrupt flag

dac_deinit

The description of `dac_deinit` is shown as below:

Table 3-164. Function `dac_deinit`

Function name	<code>dac_deinit</code>
Function prototype	<code>void dac_deinit(uint32_t dac_periph);</code>
Function descriptions	deinitialize DAC
Precondition	-
The called functions	<code>rcu_periph_reset_enable</code> / <code>rcu_periph_reset_disable</code>
Input parameter{in}	
<code>dac_periph</code>	DAC peripheral
<code>DACx</code>	DAC peripheral selection(x = 0)
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* deinitialize DAC0 */
dac_deinit(DAC0);
```

dac_enable

The description of `dac_enable` is shown as below:

Table 3-165. Function `dac_enable`

Function name	<code>dac_enable</code>
Function prototype	<code>void dac_enable(uint32_t dac_periph, uint8_t dac_out);</code>
Function descriptions	enable DAC
Precondition	-

The called functions	-
Input parameter{in}	
dac_periph	DAC peripheral
<i>DACx</i>	DAC peripheral selection(x = 0)
Input parameter{in}	
dac_out	DAC output
<i>DAC_OUTx</i>	DAC output channel selection(x = 0,1)
Output parameter{out}	
-	-
Return value	
-	-

Example:

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

dac_disable

The description of dac_disable is shown as below:

Table 3-166. Function dac_disable

Function name	dac_disable
Function prototype	void dac_disable(uint32_t dac_periph, uint8_t dac_out);
Function descriptions	disable DAC
Precondition	-
The called functions	-
Input parameter{in}	
dac_periph	DAC peripheral
<i>DACx</i>	DAC peripheral selection (x = 0)
Input parameter{in}	
dac_out	DAC output
<i>DAC_OUTx</i>	DAC output channel selection (x = 0,1)
Output parameter{out}	
-	-
Return value	
-	-

Example:

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

dac_dma_enable

The description of dac_dma_enable is shown as below:

Table 3-167. Function `dac_dma_enable`

Function name	<code>dac_dma_enable</code>
Function prototype	<code>void dac_dma_enable(uint32_t dac_periph, uint8_t dac_out);</code>
Function descriptions	enable DAC DMA function
Precondition	-
The called functions	-
Input parameter{in}	
<code>dac_periph</code>	DAC peripheral
<code>DACx</code>	DAC peripheral selection (x = 0)
Input parameter{in}	
<code>dac_out</code>	DAC output
<code>DAC_OUTx</code>	DAC output channel selection (x = 0,1)
Output parameter{out}	
-	-
Return value	
-	-

Example:

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

`dac_dma_disable`

The description of `dac_dma_disable` is shown as below:

Table 3-168. Function `dac_dma_disable`

Function name	<code>dac_dma_disable</code>
Function prototype	<code>void dac_dma_disable(uint32_t dac_periph, uint8_t dac_out);</code>
Function descriptions	disable DAC DMA function
Precondition	-
The called functions	-
Input parameter{in}	
<code>dac_periph</code>	DAC peripheral
<code>DACx</code>	DAC peripheral selection(x = 0)
Input parameter{in}	
<code>dac_out</code>	DAC output
<code>DAC_OUTx</code>	DAC output channel selection(x = 0,1)
Output parameter{out}	
-	-
Return value	
-	-

Example:

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

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

dac_output_buffer_enable

The description of `dac_output_buffer_enable` is shown as below:

Table 3-169. Function `dac_output_buffer_enable`

Function name	<code>dac_output_buffer_enable</code>
Function prototype	<code>void dac_output_buffer_enable(uint32_t dac_periph, uint8_t dac_out);</code>
Function descriptions	enable DAC output buffer
Precondition	-
The called functions	-
Input parameter{in}	
dac_periph	DAC peripheral
<i>DACx</i>	DAC peripheral selection(x = 0)
Input parameter{in}	
dac_out	DAC output
<i>DAC_OUTx</i>	DAC output channel selection(x = 0,1)
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* enable DAC0_OUT0 output buffer */
dac_output_buffer_enable(DAC0, DAC_OUT0);
```

dac_output_buffer_disable

The description of `dac_output_buffer_disable` is shown as below:

Table 3-170. Function `dac_output_buffer_disable`

Function name	<code>dac_output_buffer_disable</code>
Function prototype	<code>void dac_output_buffer_disable(uint32_t dac_periph, uint8_t dac_out);</code>
Function descriptions	disable DAC output buffer
Precondition	-
The called functions	-
Input parameter{in}	
dac_periph	DAC peripheral
<i>DACx</i>	DAC peripheral selection(x = 0)
Input parameter{in}	
dac_out	DAC output
<i>DAC_OUTx</i>	DAC output channel selection(x = 0,1)
Output parameter{out}	
-	-

Return value	
-	-

Example:

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

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

dac_output_value_get

The description of `dac_output_value_get` is shown as below:

Table 3-171. Function `dac_output_value_get`

Function name	<code>dac_output_value_get</code>
Function prototype	<code>uint16_t dac_output_value_get(uint32_t dac_periph, uint8_t dac_out);</code>
Function descriptions	get DAC output value
Precondition	-
The called functions	-
Input parameter{in}	
dac_periph	DAC peripheral
<i>DACx</i>	DAC peripheral selection(x = 0)
Input parameter{in}	
dac_out	DAC output
<i>DAC_OUTx</i>	DAC output channel selection(x = 0,1)
Output parameter{out}	
-	-
Return value	
uint16_t	DAC output data (0~4095)

Example:

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

```
uint16_t data=0;
```

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

dac_data_set

The description of `dac_data_set` is shown as below:

Table 3-172. Function `dac_data_set`

Function name	<code>dac_data_set</code>
Function prototype	<code>void dac_data_set(uint32_t dac_periph, uint8_t dac_out, uint32_t dac_align, uint16_t data);</code>
Function descriptions	set DAC data holding register value
Precondition	-
The called functions	-

Input parameter{in}	
dac_periph	DAC peripheral
<i>DACx</i>	DAC peripheral selection(x = 0)
Input parameter{in}	
dac_out	DAC output
<i>DAC_OUTx</i>	DAC output channel selection(x = 0,1)
Input parameter{in}	
dac_align	DAC data alignment mode
<i>DAC_ALIGN_12B_R</i>	12-bit right-aligned data
<i>DAC_ALIGN_12B_L</i>	12-bit left-aligned data
<i>DAC_ALIGN_8B_R</i>	8-bit right-aligned data
Input parameter{in}	
data	data to be loaded (0~4095)
Output parameter{out}	
-	-
Return value	
-	-

Example:

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

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

dac_trigger_enable

The description of `dac_trigger_enable` is shown as below:

Table 3-173. Function `dac_trigger_enable`

Function name	<code>dac_trigger_enable</code>
Function prototype	<code>void dac_trigger_enable(uint32_t dac_periph, uint8_t dac_out);</code>
Function descriptions	enable DAC trigger
Precondition	-
The called functions	-
Input parameter{in}	
dac_periph	DAC peripheral
<i>DACx</i>	DAC peripheral selection (x = 0)
Input parameter{in}	
dac_out	DAC output
<i>DAC_OUTx</i>	DAC output channel selection (x = 0,1)
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* enable DAC0_OUT0 trigger */

dac_trigger_enable(DAC0, DAC_OUT0);
```

dac_trigger_disable

The description of dac_trigger_disable is shown as below:

Table 3-174. Function dac_trigger_disable

Function name	dac_trigger_disable
Function prototype	void dac_trigger_disable(uint32_t dac_periph, uint8_t dac_out);
Function descriptions	disable DAC trigger
Precondition	-
The called functions	-
Input parameter{in}	
dac_periph	DAC peripheral
<i>DACx</i>	DAC peripheral selection (x = 0)
Input parameter{in}	
dac_out	DAC output
<i>DAC_OUTx</i>	DAC output channel selection (x = 0,1)
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* disable DAC0_OUT0 trigger */

dac_trigger_disable(DAC0, DAC_OUT0);
```

dac_trigger_source_config

The description of dac_trigger_source_config is shown as below:

Table 3-175. Function dac_trigger_source_config

Function name	dac_trigger_source_config
Function prototype	void dac_trigger_source_config(uint32_t dac_periph, uint8_t dac_out, uint32_t triggersource);
Function descriptions	configure DAC trigger source
Precondition	-
The called functions	-
Input parameter{in}	
dac_periph	DAC peripheral
<i>DACx</i>	DAC peripheral selection (x = 0)
Input parameter{in}	
dac_out	DAC output

<i>DAC_OUTx</i>	DAC output channel selection (x = 0,1)
Input parameter{in}	
triggersource	external trigger of DAC
<i>DAC_TRIGGER_T5_TRG</i> 0	TIMER5 TRGO
<i>DAC_TRIGGER_T7_TRG</i> 0	TIMER7 TRGO
<i>DAC_TRIGGER_T6_TRG</i> 0	TIMER6 TRGO
<i>DAC_TRIGGER_T4_TRG</i> 0	TIMER4 TRGO
<i>DAC_TRIGGER_T1_TRG</i> 0	TIMER1 TRGO
<i>DAC_TRIGGER_T3_TRG</i> 0	TIMER3 TRGO
<i>DAC_TRIGGER_EXTI_9</i>	EXTI interrupt line9 event
<i>DAC_TRIGGER_SOFTWARE</i>	software trigger
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* configure DAC0_OUT0 trigger source */
```

```
dac_trigger_source_config(DAC0, DAC_OUT0, DAC_TRIGGER_T1_TRGO);
```

dac_software_trigger_enable

The description of `dac_software_trigger_enable` is shown as below:

Table 3-176. Function `dac_software_trigger_enable`

Function name	<code>dac_software_trigger_enable</code>
Function prototype	<code>void dac_software_trigger_enable(uint32_t dac_periph, uint8_t dac_out);</code>
Function descriptions	enable DAC software trigger
Precondition	-
The called functions	-
Input parameter{in}	
dac_periph	DAC peripheral
<i>DACx</i>	DAC peripheral selection(x = 0)
Input parameter{in}	
dac_out	DAC output
<i>DAC_OUTx</i>	DAC output channel selection(x = 0,1)

Output parameter{out}	
-	-
Return value	
-	-

Example:

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

dac_wave_mode_config

The description of dac_wave_mode_config is shown as below:

Table 3-177. Function dac_wave_mode_config

Function name	dac_wave_mode_config
Function prototype	void dac_wave_mode_config(uint32_t dac_periph, uint8_t dac_out, uint32_t wave_mode);
Function descriptions	configure DAC wave mode
Precondition	-
The called functions	-
Input parameter{in}	
dac_periph	DAC peripheral
<i>DACx</i>	DAC peripheral selection (x = 0)
Input parameter{in}	
dac_out	DAC output
<i>DAC_OUTx</i>	DAC output channel selection (x = 0,1)
Input parameter{in}	
wave_mode	DAC wave mode
<i>DAC_WAVE_DISABLE</i>	wave mode disable
<i>DAC_WAVE_MODE_LFSR</i>	LFSR noise mode
<i>DAC_WAVE_MODE_TRIANGLE</i>	triangle noise mode
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* configure DAC0_OUT0 wave mode */
dac_wave_mode_config(DAC0, DAC_OUT0, DAC_WAVE_DISABLE);
```

dac_lfsr_noise_config

The description of dac_lfsr_noise_config is shown as below:

Table 3-178. Function `dac_lfsr_noise_config`

Function name	<code>dac_lfsr_noise_config</code>
Function prototype	<code>void dac_lfsr_noise_config(uint32_t dac_periph, uint8_t dac_out, uint32_t unmask_bits);</code>
Function descriptions	configure DAC LFSR noise mode
Precondition	-
The called functions	-
Input parameter{in}	
<code>dac_periph</code>	DAC peripheral
<code>DACx</code>	DAC peripheral selection (x = 0)
Input parameter{in}	
<code>dac_out</code>	DAC output
<code>DAC_OUTx</code>	DAC output channel selection (x = 0,1)
Input parameter{in}	
<code>unmask_bits</code>	LFSR noise unmask bits
<code>DAC_LFSR_BIT0</code>	unmask the LFSR bit0
<code>DAC_LFSR_BITSx_0</code>	unmask the LFSR bits [x:0] (x = 1,2,3..11)
Output parameter{out}	
-	-
Return value	
-	-

Example:

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

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

`dac_triangle_noise_config`

The description of `dac_triangle_noise_config` is shown as below:

Table 3-179. Function `dac_triangle_noise_config`

Function name	<code>dac_triangle_noise_config</code>
Function prototype	<code>void dac_triangle_noise_config(uint32_t dac_periph, uint8_t dac_out, uint32_t amplitude);</code>
Function descriptions	configure DAC triangle noise mode
Precondition	-
The called functions	-
Input parameter{in}	
<code>dac_periph</code>	DAC peripheral
<code>DACx</code>	DAC peripheral selection (x = 0)
Input parameter{in}	
<code>dac_out</code>	DAC output
<code>DAC_OUTx</code>	DAC output channel selection (x = 0,1)
Input parameter{in}	

amplitude	the amplitude of the triangle
<i>DAC_TRIANGLE_AMPLIT</i> <i>UDE_x</i>	$x = 2^n - 1$ ($n = 1..12$)
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* configure DAC0_OUT0 triangle noise mode */
dac_triangle_noise_config(DAC0, DAC_OUT0, DAC_TRIANGLE_AMPLITUDE_1);
```

dac_concurrent_enable

The description of `dac_concurrent_enable` is shown as below:

Table 3-180. Function `dac_concurrent_enable`

Function name	<code>dac_concurrent_enable</code>
Function prototype	<code>void dac_concurrent_enable(uint32_t dac_periph);</code>
Function descriptions	enable DAC concurrent mode
Precondition	-
The called functions	-
Input parameter{in}	
dac_periph	DAC peripheral
<i>DACx</i>	DAC peripheral selection ($x = 0$)
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* enable DAC0 concurrent mode */
dac_concurrent_enable(DAC0);
```

dac_concurrent_disable

The description of `dac_concurrent_disable` is shown as below:

Table 3-181. Function `dac_concurrent_disable`

Function name	<code>dac_concurrent_disable</code>
Function prototype	<code>void dac_concurrent_disable(uint32_t dac_periph);</code>
Function descriptions	disable DAC concurrent mode
Precondition	-
The called functions	-

Input parameter{in}	
dac_periph	DAC peripheral
<i>DACx</i>	DAC peripheral selection(x = 0)
Output parameter{out}	
-	-
Return value	
-	-

Example:

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

```
dac_concurrent_disable(DAC0);
```

dac_concurrent_software_trigger_enable

The description of `dac_concurrent_software_trigger_enable` is shown as below:

Table 3-182. Function `dac_concurrent_software_trigger_enable`

Function name	<code>dac_concurrent_software_trigger_enable</code>
Function prototype	<code>void dac_concurrent_software_trigger_enable(uint32_t dac_periph);</code>
Function descriptions	enable DAC concurrent software trigger
Precondition	-
The called functions	-
Input parameter{in}	
dac_periph	DAC peripheral
<i>DACx</i>	DAC peripheral selection (x = 0)
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* enable DAC0 concurrent software trigger */
```

```
dac_concurrent_software_trigger_enable(DAC0);
```

dac_concurrent_output_buffer_enable

The description of `dac_concurrent_output_buffer_enable` is shown as below:

Table 3-183. Function `dac_concurrent_output_buffer_enable`

Function name	<code>dac_concurrent_output_buffer_enable</code>
Function prototype	<code>void dac_concurrent_output_buffer_enable(uint32_t dac_periph);</code>
Function descriptions	enable DAC concurrent buffer function
Precondition	-
The called functions	-

Input parameter{in}	
dac_periph	DAC peripheral
<i>DACx</i>	DAC peripheral selection(x = 0)
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* enable DAC0 concurrent buffer function */
dac_concurrent_output_buffer_enable(DAC0);
```

dac_concurrent_output_buffer_disable

The description of `dac_concurrent_output_buffer_disable` is shown as below:

Table 3-184. Function `dac_concurrent_output_buffer_disable`

Function name	<code>dac_concurrent_output_buffer_disable</code>
Function prototype	<code>void dac_concurrent_output_buffer_disable(uint32_t dac_periph);</code>
Function descriptions	disable DAC concurrent buffer function
Precondition	-
The called functions	-
Input parameter{in}	
dac_periph	DAC peripheral
<i>DACx</i>	DAC peripheral selection (x = 0)
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* disable DAC0 concurrent buffer function */
dac_concurrent_output_buffer_disable(DAC0);
```

dac_concurrent_data_set

The description of `dac_concurrent_data_set` is shown as below:

Table 3-185. Function `dac_concurrent_data_set`

Function name	<code>dac_concurrent_data_set</code>
Function prototype	<code>void dac_concurrent_data_set(uint32_t dac_periph, uint32_t dac_align, uint16_t data0, uint16_t data1);</code>
Function descriptions	set DAC concurrent mode data holding register value
Precondition	-

The called functions	-
Input parameter{in}	
dac_periph	DAC peripheral
<i>DACx</i>	DAC peripheral selection(x = 0)
Input parameter{in}	
dac_align	DAC data alignment mode
<i>DAC_ALIGN_12B_R</i>	12-bit right-aligned data
<i>DAC_ALIGN_12B_L</i>	12-bit left-aligned data
<i>DAC_ALIGN_8B_R</i>	8-bit right-aligned data
Input parameter{in}	
data0	DACx_OUT0 data to be loaded (0~4095)
Input parameter{in}	
data1	DACx_OUT1 data to be loaded (0~4095)
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* set DAC0 concurrent mode data holding register value */
```

```
dac_concurrent_data_set(DAC0, DAC_ALIGN_8B_R, 0xFF, 0xFF);
```

dac_flag_get

The description of `dac_flag_get` is shown as below:

Table 3-186. Function `dac_flag_get`

Function name	<code>dac_flag_get</code>
Function prototype	<code>FlagStatus dac_flag_get(uint32_t dac_periph, uint32_t flag);</code>
Function descriptions	get DAC flag
Precondition	-
The called functions	-
Input parameter{in}	
dac_periph	DAC peripheral
<i>DACx</i>	DAC peripheral selection(x = 0)
Input parameter{in}	
flag	the DAC status flags
<i>DAC_FLAG_DDUDR0</i>	DACx_OUT0 DMA underrun flag
<i>DAC_FLAG_DDUDR1</i>	DACx_OUT1 DMA underrun flag
Output parameter{out}	
-	-
Return value	
FlagStatus	the state of DAC bit (SET or RESET)

Example:

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

```
FlagStatus flag;
```

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

dac_flag_clear

The description of `dac_flag_clear` is shown as below:

Table 3-187. Function `dac_flag_clear`

Function name	<code>dac_flag_clear</code>
Function prototype	<code>void dac_flag_clear(uint32_t dac_periph, uint32_t flag);</code>
Function descriptions	clear DAC flag
Precondition	-
The called functions	-
Input parameter{in}	
dac_periph	DAC peripheral
<i>DACx</i>	DAC peripheral selection(x = 0)
Input parameter{in}	
flag	DAC flag
<i>DAC_FLAG_DDUDR0</i>	DACx_OUT0 DMA underrun flag
<i>DAC_FLAG_DDUDR1</i>	DACx_OUT1 DMA underrun flag
Output parameter{out}	
-	-
Return value	
-	-

Example:

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

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

dac_interrupt_enable

The description of `dac_interrupt_enable` is shown as below:

Table 3-188. Function `dac_interrupt_enable`

Function name	<code>dac_interrupt_enable</code>
Function prototype	<code>void dac_interrupt_enable(uint32_t dac_periph, uint32_t interrupt);</code>
Function descriptions	enable DAC interrupt
Precondition	-
The called functions	-
Input parameter{in}	
dac_periph	DAC peripheral

<i>DACx</i>	DAC peripheral selection(x = 0)
Input parameter{in}	
interrupt	the DAC interrupt
<i>DAC_INT_DDUDR0</i>	DACx_OUT0 DMA underrun interrupt
<i>DAC_INT_DDUDR1</i>	DACx_OUT1 DMA underrun interrupt
Output parameter{out}	
-	-
Return value	
-	-

Example:

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

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

dac_interrupt_disable

The description of `dac_interrupt_disable` is shown as below:

Table 3-189. Function `dac_interrupt_disable`

Function name	<code>dac_interrupt_disable</code>
Function prototype	<code>void dac_interrupt_disable(uint32_t dac_periph, uint32_t interrupt);</code>
Function descriptions	disable DAC interrupt
Precondition	-
The called functions	-
Input parameter{in}	
dac_periph	DAC peripheral
<i>DACx</i>	DAC peripheral selection(x = 0)
Input parameter{in}	
interrupt	the DAC interrupt
<i>DAC_INT_DDUDR0</i>	DACx_OUT0 DMA underrun interrupt
<i>DAC_INT_DDUDR1</i>	DACx_OUT1 DMA underrun interrupt
Output parameter{out}	
-	-
Return value	
-	-

Example:

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

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

dac_interrupt_flag_get

The description of `dac_interrupt_flag_get` is shown as below:

Table 3-190. Function `dac_interrupt_flag_get`

Function name	<code>dac_interrupt_flag_get</code>
Function prototype	<code>FlagStatus dac_interrupt_flag_get(uint32_t dac_periph, uint32_t int_flag);</code>
Function descriptions	get DAC interrupt flag
Precondition	-
The called functions	-
Input parameter{in}	
dac_periph	DAC peripheral
<i>DACx</i>	DAC peripheral selection(x = 0)
Input parameter{in}	
int_flag	DAC interrupt flag
<i>DAC_INT_FLAG_DDUDR0</i>	DACx_OUT0 DMA underrun interrupt flag
<i>DAC_INT_FLAG_DDUDR1</i>	DACx_OUT1 DMA underrun interrupt flag
Output parameter{out}	
-	-
Return value	
FlagStatus	the state of DAC interrupt flag(SET or RESET)

Example:

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

```
FlagStatus flag;
```

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

`dac_interrupt_flag_clear`

The description of `dac_interrupt_flag_clear` is shown as below:

Table 3-191. Function `dac_interrupt_flag_clear`

Function name	<code>dac_interrupt_flag_clear</code>
Function prototype	<code>FlagStatus dac_interrupt_flag_clear(uint32_t dac_periph, uint32_t int_flag);</code>
Function descriptions	clear DAC interrupt flag
Precondition	-
The called functions	-
Input parameter{in}	
dac_periph	DAC peripheral
<i>DACx</i>	DAC peripheral selection(x = 0)
Input parameter{in}	
int_flag	DAC interrupt flag
<i>DAC_INT_FLAG_DDUDR0</i>	DACx_OUT0 DMA underrun interrupt flag
<i>DAC_INT_FLAG_DDUDR1</i>	DACx_OUT1 DMA underrun interrupt flag
Output parameter{out}	

-	-
Return value	
-	-

Example:

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

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

3.8. DBG

The DBG hold unit helps debugger to debug power saving mode. The DBG registers are listed in chapter [3.8.1](#). the DBG firmware functions are introduced in chapter [3.8.2](#).

3.8.1. Descriptions of Peripheral registers

DBG registers are listed in the table shown as below:

Table 3-192. DBG Registers

Registers	Descriptions
DBG_ID	DBG ID code register
DBG_CTL0	DBG control register 0
DBG_CTL1	DBG control register 1
DBG_CTL2	DBG control register 2
DBG_CTL3	DBG control register 3

3.8.2. Descriptions of Peripheral functions

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

Table 3-193. DBG firmware function

Function name	Function description
dbg_deinit	deinitialize the DBG
dbg_id_get	read DBG_ID code register
dbg_low_power_enable	enable low power behavior when the MCU is in debug mode
dbg_low_power_disable	disable low power behavior when the MCU is in debug mode
dbg_periph_enable	enable peripheral behavior when the MCU is in debug mode
dbg_periph_disable	disable peripheral behavior when the MCU is in debug mode

Function name	Function description
dbg_deinit	deinitialize the DBG
dbg_trace_pin_enable	enable trace pin assignment
dbg_trace_pin_disable	disable trace pin assignment
dbg_deviceid_get	read DBG DEVICEID
dbg_deviceid_set	write DBG DEVICEID

Enum dbg_periph_enum

Table 3-194. Enum dbg_periph_enum

Member name	Function description
DBG_TIMER1_HOLD	hold TIMER1 counter when core is halted
DBG_TIMER2_HOLD	hold TIMER2 counter when core is halted
DBG_TIMER3_HOLD	hold TIMER3 counter when core is halted
DBG_TIMER4_HOLD	hold TIMER4 counter when core is halted
DBG_TIMER5_HOLD	hold TIMER5 counter when core is halted
DBG_TIMER6_HOLD	hold TIMER6 counter when core is halted
DBG_TIMER11_HOLD	hold TIMER11 counter when core is halted
DBG_TIMER12_HOLD	hold TIMER12 counter when core is halted
DBG_TIMER13_HOLD	hold TIMER13 counter when core is halted
DBG_RTC_HOLD	hold RTC counter when core is halted
DBG_FWDGT_HOLD	debug FWDGT kept when core is halted
DBG_WWDGT_HOLD	debug WWDGT kept when core is halted
DBG_I2C3_HOLD	hold I2C3 smbus when core is halted
DBG_I2C4_HOLD	hold I2C4 smbus when core is halted
DBG_I2C5_HOLD	hold I2C5 smbus when core is halted
DBG_I2C0_HOLD	hold I2C0 smbus when core is halted
DBG_I2C1_HOLD	hold I2C1 smbus when core is halted
DBG_I2C2_HOLD	hold I2C2 smbus when core is halted
DBG_CAN0_HOLD	debug CAN0 kept when core is halted

DBG_CAN1_HOLD	debug CAN1 kept when core is halted
DBG_TIMER0_HOLD	hold TIMER0 counter when core is halted
DBG_TIMER7_HOLD	hold TIMER7 counter when core is halted
DBG_TIMER8_HOLD	hold TIMER8 counter when core is halted
DBG_TIMER9_HOLD	hold TIMER9 counter when core is halted
DBG_TIMER10_HOLD	hold TIMER10 counter when core is halted

dbg_deinit

The description of dbg_deinit is shown as below:

Table 3-195. Function dbg_deinit

Function name	dbg_deinit
Function prototype	void dbg_deinit (void);
Function descriptions	deinitialize the DBG
Precondition	-
The called functions	-
Input parameter{in}	
-	-
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* deinitialize the DBG */
dbg_deinit();
```

dbg_id_get

The description of dbg_id_get is shown as below:

Table 3-196. Function dbg_id_get

Function name	dbg_id_get
Function prototype	uint32_t dbg_id_get(void);

Function descriptions	Read DBG_ID code register
Precondition	-
The called functions	-
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 3-197. Function dbg_low_power_enable

Function name	dbg_low_power_enable
Function prototype	void dbg_low_power_enable(uint32_t dbg_low_power);
Function descriptions	Enable low power behavior when the mcu is in debug mode
Precondition	-
The called functions	-
Input parameter{in}	
dbg_low_power	low power mode
DBG_LOW_POWER_SLEEP	keep debugger connection during sleep mode
DBG_LOW_POWER_DEEPSLEEP	keep debugger connection during deepsleep mode
DBG_LOW_POWER_STANDBY	keep debugger connection during standby mode

TANDBY	
Output parameter{out}	
-	-
Return value	
-	-

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 3-198. Function dbg_low_power_disable

Function name	dbg_low_power_disable
Function prototype	void dbg_low_power_disable(uint32_t dbg_low_power);
Function descriptions	Disable low power behavior when the mcu is in debug mode
Precondition	-
The called functions	-
Input parameter{in}	
dbg_low_power	low power mode
DBG_LOW_POWER_SLEEP	keep debugger connection during sleep mode
DBG_LOW_POWER_DEEPSLEEP	keep debugger connection during deepsleep mode
DBG_LOW_POWER_STANDBY	keep debugger connection during standby mode
Output parameter{out}	
-	-
Return value	
-	-

Example:

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

```
dbg_low_power_disable(DBG_LOW_POWER_SLEEP);
```

dbg_periph_enable

The description of dbg_periph_enable is shown as below:

Table 3-199. Function dbg_periph_enable

Function name	dbg_periph_enable
Function prototype	void dbg_periph_enable(dbg_periph_enum dbg_periph);
Function descriptions	Enable peripheral behavior when the mcu is in debug mode
Precondition	-
The called functions	-
Input parameter{in}	
dbg_periph	Peripheral refer to Table 3-194. Enum dbg_periph_enum
<i>DBG_FWDGT_HOLD</i>	debug FWDGT kept when core is halted
<i>DBG_WWDGT_HOLD</i>	debug WWDGT kept when core is halted
<i>DBG_CANx_HOLD</i>	x=0,1, hold CANx counter when core is halted
<i>DBG_I2Cx_HOLD</i>	x=0,1,2,3,4,5, hold I2Cx smbus when core is halted
<i>DBG_TIMERx_HOLD</i>	x=0,1,2,3,4,5,6,7,8,9,10,11,12,13, hold TIMERx counter when core is halted
Output parameter{out}	
-	-
Return value	
-	-

Example:

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

```
dbg_periph_enable(DBG_TIMER0_HOLD);
```

dbg_periph_disable

The description of dbg_periph_disable is shown as below:

Table 3-200. Function dbg_periph_disable

Function name	dbg_periph_disable
Function prototype	void dbg_periph_disable(dbg_periph_enum dbg_periph);

Function descriptions	Disable peripheral behavior when the mcu is in debug mode
Precondition	-
The called functions	-
Input parameter{in}	
dbg_periph	peripheral refer to Table 3-194. Enum dbg_periph_enum
<i>DBG_FWDGT_HOLD</i>	debug FWDGT kept when core is halted
<i>DBG_WWDGT_HOLD</i>	debug WWDGT kept when core is halted
<i>DBG_CANx_HOLD</i>	x=0,1, hold CANx counter when core is halted
<i>DBG_I2Cx_HOLD</i>	x=0,1,2,3,4,5, hold I2Cx smbus when core is halted
<i>DBG_TIMERx_HOLD</i>	x=0,1,2,3,4,5,6,7,8,9,10,11,12,13, hold TIMERx counter when core is halted
Output parameter{out}	
-	-
Return value	
-	-

Example:

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

```
dbg_periph_disable(DBG_TIMER0_HOLD);
```

dbg_trace_pin_enable

The description of dbg_trace_pin_enable is shown as below:

Table 3-201. Function dbg_trace_pin_enable

Function name	dbg_trace_pin_enable
Function prototype	void dbg_trace_pin_enable(void);
Function descriptions	Enable trace pin assignment
Precondition	-
The called functions	-
Input parameter{in}	
-	-
Output parameter{out}	

-	-
Return value	
-	-

Example:

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

```
dbg_trace_pin_enable();
```

dbg_trace_pin_disable

The description of dbg_trace_pin_disable is shown as below:

Table 3-202. Function dbg_trace_pin_disable

Function name	dbg_trace_pin_disable
Function prototype	void dbg_trace_pin_disable(void);
Function descriptions	Disable trace pin assignment
Precondition	-
The called functions	-
Input parameter{in}	
-	-
Output parameter{out}	
-	-
Return value	
-	-

Example:

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

```
dbg_trace_pin_disable();
```

dbg_deviceid_get

The description of dbg_deviceid_get is shown as below:

Table 3-203. Function dbg_id_get

Function name	dbg_deviceid_get
Function prototype	uint32_t dbg_deviceid_get (void);

Function descriptions	Read DBG DEVICEID
Precondition	-
The called functions	-
Input parameter{in}	
-	-
Output parameter{out}	
-	-
Return value	
uint32_t	DBG DEVICEID (0-0xF)

Example:

```
/* read DBG DEVICEID */
uint32_t id_value = 0;
id_value = dbg_deviceid_get ();
```

dbg_deviceid_set

The description of dbg_deviceid_set is shown as below:

Table 3-204. Function dbg_id_set

Function name	dbg_deviceid_set
Function prototype	void dbg_deviceid_set (uint32_t deviceid);
Function descriptions	write DBG DEVICEID
Precondition	-
The called functions	-
Input parameter{in}	
uint32_t	device id-
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* write DBG DEVICEID */
```

```
uint32_t id_value = 0x03;

dbg_deviceid_set(id_value);
```

3.9. DCI

The DCI is a parallel interface able to capture video or picture from a camera via Digital Camera Interface. The DCI registers are listed in chapter [3.9.1](#). the DCI firmware functions are introduced in chapter [3.9.2](#).

3.9.1. Descriptions of Peripheral registers

DCI registers are listed in the table shown as below:

Table 3-205. DCI Registers

Registers	Descriptions
DCI_CTL	DCI control register
DCI_STAT0	DCI status register 0
DCI_STAT1	DCI status register 1
DCI_INTEN	DCI interrupt enable register
DCI_INTF	DCI interrupt flag register
DCI_INTC	DCI interrupt clear register
DCI_SC	DCI synchronization codes register
DCI_SCUMSK	DCI synchronization codes unmask register
DCI_CWSPOS	DCI cropping window start position register
DCI_CWSZ	DCI cropping window size register
DCI_DATA	DCI data register

3.9.2. Descriptions of Peripheral functions

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

Table 3-206. DCI firmware function

Function name	Function description
dci_deinit	DCI deinit
dci_init	initialize DCI registers
dci_enable	enable DCI function
dci_disable	disable DCI function
dci_capture_enable	enable DCI capture
dci_capture_disable	disable DCI capture
dci_jpeg_enable	enable DCI jpeg mode
dci_jpeg_disable	disable DCI jpeg mode
dci_crop_window_enable	enable cropping window function
dci_crop_window_disable	disable cropping window function

Function name	Function description
dc_i_cro_p_window_config	config DCI cropping window
dc_i_embedded_sync_enable	enable embedded synchronous mode
dc_i_embedded_sync_disable	disable embedded synchronous mode
dc_i_sync_codes_config	config synchronous codes in embedded synchronous mode
dc_i_sync_codes_unmask_config	config synchronous codes unmask in embedded synchronous mode
dc_i_data_read	read DCI data register
dc_i_flag_get	get specified flag
dc_i_interrupt_enable	enable specified DCI interrupt
dc_i_interrupt_disable	disable specified DCI interrupt
dc_i_interrupt_flag_get	get specified interrupt flag
dc_i_interrupt_flag_clear	clear specified interrupt flag

Structure dc_i_parameter_struct

Table 3-207. Structure dc_i_parameter_struct

Member name	Function description
capture_mode	DCI capture mode: DCI_CAPTURE_MODE_CONTINUOUS / DCI_CAPTURE_MODE_SNAPSHOT
clock_polarity	clock polarity selection: DCI_CK_POLARITY_FALLING / DCI_CK_POLARITY_RISING
hsync_polarity	horizontal polarity selection: DCI_HSYNC_POLARITY_LOW / DCI_HSYNC_POLARITY_HIGH
vsync_polarity	vertical polarity selection: DCI_VSYNC_POLARITY_LOW / DCI_VSYNC_POLARITY_HIGH
frame_rate	frame capture rate: DCI_FRAME_RATE_ALL / DCI_FRAME_RATE_1_2 / DCI_FRAME_RATE_1_4
interface_format	digital camera interface format: DCI_INTERFACE_FORMAT_8BITS / DCI_INTERFACE_FORMAT_10BITS / DCI_INTERFACE_FORMAT_12BITS / DCI_INTERFACE_FORMAT_14BITS

dc_i_deinit

The description of dc_i_deinit is shown as below:

Table 3-208. Function dc_i_deinit

Function name	dc_i_deinit
Function prototype	void dc_i_deinit(void);
Function descriptions	DCI deinit
Precondition	-
The called functions	-
Input parameter{in}	

-	-
Output parameter{out}	
-	-
Return value	
-	-

Example:

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

```
dc_i_deinit( );
```

dc_i_init

The description of dc_i_init is shown as below:

Table 3-209. Function dc_i_init

Function name	dc_i_init
Function prototype	void dc_i_init(dc_i_parameter_struct* dc_i_struct);
Function descriptions	initialize DCI registers
Precondition	-
The called functions	-
Input parameter{in}	
dc_i_struct	DCI parameter initialization struct
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* initialize DCI registers */
```

```
dc_i_parameter_struct dc_i_struct;
```

```
dc_i_struct.capture_mode = DCI_CAPTURE_MODE_CONTINUOUS;
```

```
dc_i_struct.clock_polarity = DCI_CK_POLARITY_RISING;
```

```
dc_i_struct.hsync_polarity = DCI_HSYNC_POLARITY_LOW;
```

```
dc_i_struct.vsync_polarity = DCI_VSYNC_POLARITY_LOW;
```

```
dc_i_struct.frame_rate = DCI_FRAME_RATE_ALL;
```

```
dc_i_struct.interface_format = DCI_INTERFACE_FORMAT_8BITS;
```

```
dc_i_init (&dc_i_struct);
```

dc_i_enable

The description of dc_i_enable is shown as below:

Table 3-210. Function dci_enable

Function name	dci_enable
Function prototype	void dci_enable(void);
Function descriptions	enable DCI function
Precondition	-
The called functions	-
Input parameter{in}	
-	-
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* enable DCI function */
```

```
dci_enable( );
```

dci_disable

The description of dci_disable is shown as below:

Table 3-211. Function dci_disable

Function name	dci_disable
Function prototype	void dci_disable(void);
Function descriptions	disable DCI function
Precondition	-
The called functions	-
Input parameter{in}	
-	-
Output parameter{out}	
-	-
Return value	
-	-

Example:

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

```
dci_disable( );
```

dci_capture_enable

The description of dci_capture_enable is shown as below:

Table 3-212. Function dci_capture_enable

Function name	dci_capture_enable
----------------------	--------------------

Function prototype	void dci_capture_enable(void);
Function descriptions	enable DCI capture
Precondition	-
The called functions	-
Input parameter{in}	
-	-
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* enable DCI capture */
dci_capture_enable( );
```

dci_capture_disable

The description of dci_capture_disable is shown as below:

Table 3-213. Function dci_capture_disable

Function name	dci_capture_disable
Function prototype	void dci_capture_disable(void);
Function descriptions	disable DCI capture
Precondition	-
The called functions	-
Input parameter{in}	
-	-
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* disable DCI capture */
dci_capture_disable( );
```

dci_jpeg_enable

The description of dci_jpeg_enable is shown as below:

Table 3-214. Function dci_jpeg_enable

Function name	dci_jpeg_enable
Function prototype	void dci_jpeg_enable(void);
Function descriptions	enable DCI jpeg mode

Precondition	-
The called functions	-
Input parameter{in}	
-	-
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* enable DCI jpeg mode */
```

```
dc_ijpeg_enable( );
```

dc_ijpeg_disable

The description of dc_ijpeg_disable is shown as below:

Table 3-215. Function dc_ijpeg_disable

Function name	dc_ijpeg_disable
Function prototype	void dc_ijpeg_disable(void);
Function descriptions	disable DCI jpeg mode
Precondition	-
The called functions	-
Input parameter{in}	
-	-
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* disable DCI jpeg mode */
```

```
dc_ijpeg_disable( );
```

dc_crop_window_enable

The description of dc_crop_window_enable is shown as below:

Table 3-216. Function dc_crop_window_enable

Function name	dc_crop_window_enable
Function prototype	void dc_crop_window_enable(void);
Function descriptions	enable cropping window function
Precondition	-
The called functions	-

Input parameter{in}	
-	-
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* enable cropping window function */
```

```
dc_i_cro_p_window_enable( );
```

dc_i_cro_p_window_disable

The description of dc_i_cro_p_window_disable is shown as below:

Table 3-217. Function dc_i_cro_p_window_disable

Function name	dc_i_cro_p_window_disable
Function prototype	void dc_i_cro_p_window_disable(void);
Function descriptions	disable cropping window function
Precondition	-
The called functions	-
Input parameter{in}	
-	-
Output parameter{out}	
-	-
Return value	
-	-

Example:

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

```
dc_i_cro_p_window_disable( );
```

dc_i_cro_p_window_config

The description of dc_i_cro_p_window_config is shown as below:

Table 3-218. Function dc_i_cro_p_window_config

Function name	dc_i_cro_p_window_config
Function prototype	void dc_i_cro_p_window_config(uint16_t start_x, uint16_t start_y, uint16_t size_width, uint16_t size_height);
Function descriptions	config DCI cropping window
Precondition	-
The called functions	-
Input parameter{in}	

start_x	window horizontal start position
Input parameter{in}	
start_y	window vertical start position
Input parameter{in}	
size_width	window horizontal size
Input parameter{in}	
size_height	window vertical size
Output parameter{out}	
-	-
Return value	
-	-

Example:

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

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

dc_i_embedded_sync_enable

The description of dc_i_embedded_sync_enable is shown as below:

Table 3-219. Function dc_i_embedded_sync_enable

Function name	dc_i_embedded_sync_enable
Function prototype	void dc_i_embedded_sync_enable(void);
Function descriptions	enable embedded synchronous mode
Precondition	-
The called functions	-
Input parameter{in}	
-	-
Output parameter{out}	
-	-
Return value	
-	-

Example:

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

```
dc_i_embedded_sync_enable( );
```

dc_i_embedded_sync_disable

The description of dc_i_embedded_sync_disable is shown as below:

Table 3-220. Function dc_i_embedded_sync_disable

Function name	dc_i_embedded_sync_disable
Function prototype	void dc_i_embedded_sync_disable(void);

Function descriptions	disable embedded synchronous mode
Precondition	-
The called functions	-
Input parameter{in}	
-	-
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* disable embedded synchronous mode */
dci_embedded_sync_disable( );
```

dci_sync_codes_config

The description of dci_sync_codes_config is shown as below:

Table 3-221. Function dci_sync_codes_config

Function name	dci_sync_codes_config
Function prototype	void dci_sync_codes_config(uint8_t frame_start, uint8_t line_start, uint8_t line_end, uint8_t frame_end);
Function descriptions	config synchronous codes in embedded synchronous mode
Precondition	-
The called functions	-
Input parameter{in}	
frame_start	frame start code in embedded synchronous mode
Input parameter{in}	
line_start	line start code in embedded synchronous mode
Input parameter{in}	
line_end	line end code in embedded synchronous mode
Input parameter{in}	
frame_end	frame end code in embedded synchronous mode
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* config synchronous codes in embedded synchronous mode */
dci_sync_codes_config (0x10, 0x10, 0x20, 0x20);
```

dci_sync_codes_unmask_config

The description of dci_sync_codes_unmask_config is shown as below:

Table 3-222. Function dci_sync_codes_unmask_config

Function name	dci_sync_codes_unmask_config
Function prototype	void dci_sync_codes_unmask_config(uint8_t frame_start, uint8_t line_start, uint8_t line_end, uint8_t frame_end);
Function descriptions	config synchronous codes unmask in embedded synchronous mode
Precondition	-
The called functions	-
Input parameter{in}	
frame_start	frame start code in embedded synchronous mode
Input parameter{in}	
line_start	line start code in embedded synchronous mode
Input parameter{in}	
line_end	line end code in embedded synchronous mode
Input parameter{in}	
frame_end	frame end code in embedded synchronous mode
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* config synchronous codes unmask in embedded synchronous mode */
dci_sync_codes_unmask_config (0x10, 0x10, 0x20, 0x20);
```

dci_data_read

The description of dci_data_read is shown as below:

Table 3-223. Function dci_data_read

Function name	dci_data_read
Function prototype	uint32_t dci_data_read(void);
Function descriptions	read DCI data register
Precondition	-
The called functions	-
Input parameter{in}	
-	-
Output parameter{out}	
-	-
Return value	
uint32_t	0x00 – 0xFFFFFFFF

Example:

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


dc_i_flag_get

The description of dc_i_flag_get is shown as below:

Table 3-224. Function dc_i_flag_get

Function name	dc_i_flag_get
Function prototype	FlagStatus dc_i_flag_get(uint32_t flag);
Function descriptions	get specified flag
Precondition	-
The called functions	-
Input parameter{in}	
flag	DCI flag
<i>DCI_FLAG_HS</i>	HS line status
<i>DCI_FLAG_VS</i>	VS line status
<i>DCI_FLAG_FV</i>	FIFO valid
<i>DCI_FLAG_EF</i>	end of frame flag
<i>DCI_FLAG_OVR</i>	FIFO overrun flag
<i>DCI_FLAG_ESE</i>	embedded synchronous error flag
<i>DCI_FLAG_VSYNC</i>	vsync flag
<i>DCI_FLAG_EL</i>	end of line flag
Output parameter{out}	
-	-
Return value	
FlagStatus	FlagStatus: SET or RESET

Example:

```
/* get specified flag */
FlagStatus status = dc_i_flag_get (DCI_FLAG_HS);
```

dc_i_interrupt_enable

The description of dc_i_interrupt_enable is shown as below:

Table 3-225. Function dc_i_interrupt_enable

Function name	dc_i_interrupt_enable
Function prototype	void dc_i_interrupt_enable(uint32_t interrupt);
Function descriptions	enable specified DCI interrupt
Precondition	-
The called functions	-
Input parameter{in}	
interrupt	DCI interrupt
<i>DCI_INT_EF</i>	end of frame interrupt
<i>DCI_INT_OVR</i>	FIFO overrun interrupt
<i>DCI_INT_ESE</i>	embedded synchronous error interrupt
<i>DCI_INT_VSYNC</i>	vsync interrupt

<i>DCI_INT_EL</i>	end of line interrupt
Output parameter{out}	
-	-
Return value	
-	-

Example:

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

```
dci_interrupt_enable (DCI_INT_EF);
```

dci_interrupt_disable

The description of dci_interrupt_disable is shown as below:

Table 3-226. Function dci_interrupt_disable

Function name	dci_interrupt_disable
Function prototype	void dci_interrupt_disable(uint32_t interrupt);
Function descriptions	disable specified DCI interrupt
Precondition	-
The called functions	-
Input parameter{in}	
interrupt	DCI interrupt
<i>DCI_INT_EF</i>	end of frame interrupt
<i>DCI_INT_OVR</i>	FIFO overrun interrupt
<i>DCI_INT_ESE</i>	embedded synchronous error interrupt
<i>DCI_INT_VSYNC</i>	vsync interrupt
<i>DCI_INT_EL</i>	end of line interrupt
Output parameter{out}	
-	-
Return value	
-	-

Example:

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

```
dci_interrupt_disable (DCI_INT_EF);
```

dci_interrupt_flag_get

The description of dci_interrupt_flag_get is shown as below:

Table 3-227. Function dci_interrupt_flag_get

Function name	dci_interrupt_flag_get
Function prototype	FlagStatus dci_interrupt_flag_get(uint32_t interrupt);
Function descriptions	get specified interrupt flag

Precondition	-
The called functions	-
Input parameter{in}	
interrupt	DCI interrupt
<i>DCI_INT_FLAG_EF</i>	end of frame interrupt flag
<i>DCI_INT_FLAG_OVR</i>	FIFO overrun interrupt flag
<i>DCI_INT_FLAG_ESE</i>	embedded synchronous error interrupt flag
<i>DCI_INT_FLAG_VSYN</i> <i>C</i>	vsync interrupt flag
<i>DCI_INT_FLAG_EL</i>	end of line interrupt flag
Output parameter{out}	
-	-
Return value	
FlagStatus	FlagStatus: SET or RESET

Example:

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

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

dc_i_interrupt_flag_clear

The description of dc_i_interrupt_flag_clear is shown as below:

Table 3-228. Function dc_i_interrupt_flag_clear

Function name	dc_i_interrupt_flag_clear
Function prototype	void dc_i_interrupt_flag_clear(uint32_t interrupt);
Function descriptions	clear specified interrupt flag
Precondition	-
The called functions	-
Input parameter{in}	
interrupt	DCI interrupt
<i>DCI_INT_FLAG_EF</i>	end of frame interrupt flag
<i>DCI_INT_FLAG_OVR</i>	FIFO overrun interrupt flag
<i>DCI_INT_FLAG_ESE</i>	embedded synchronous error interrupt flag
<i>DCI_INT_FLAG_VSYN</i> <i>C</i>	vsync interrupt flag
<i>DCI_INT_FLAG_EL</i>	end of line interrupt flag
Output parameter{out}	
-	-
Return value	
-	-

Example:

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

```
dc_i_interrupt_flag_clear (DCI_INT_FLAG_EF);
```

3.10. DMA

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 [3.10.1](#), the DMA firmware functions are introduced in chapter [3.10.2](#).

3.10.1. Descriptions of Peripheral registers

DMA registers are listed in the table shown as below:

Table 3-229. DMA Registers

Registers	Descriptions
DMA_INTF0	Interrupt flag register0
DMA_INTF1	Interrupt flag register1
DMA_INTC0	Interrupt flag clear register0
DMA_INTC1	Interrupt flag clear register1
DMA_CHxCTL (x=0..7)	Channel x control register
DMA_CHxCNT (x=0..7)	Channel x counter register
DMA_CHxPADDR (x=0..7)	Channel x peripheral base address register
DMA_CHxM0ADDR (x=0..7)	Channel x memory 0 base address register
DMA_CHxM1ADDR (x=0..7)	Channel x memory 1 base address register
DMA_CHxFCTL (x=0..7)	Channel x FIFO control register

3.10.2. Descriptions of Peripheral functions

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

Table 3-230. DMA firmware function

Function name	Function description
dma_deinit	deinitialize DMA a channel registers

Function name	Function description
dma_single_data_para_struct_init	initialize the DMA single data mode parameters struct with the default values
dma_multi_data_para_struct_init	initialize the DMA multi data mode parameters struct with the default values
dma_single_data_mode_init	DMA single data mode initialize
dma_multi_data_mode_init	DMA multi data mode initialize
dma_periph_address_config	set DMA peripheral base address
dma_memory_address_config	set DMA memory base address
dma_transfer_number_config	set the number of remaining data to be transferred by the DMA
dma_transfer_number_get	get the number of remaining data to be transferred by the DMA
dma_priority_config	configure priority level of DMA channel
dma_memory_burst_beats_config	configure transfer burst beats of memory
dma_periph_burst_beats_config	configure transfer burst beats of peripheral
dma_memory_width_config	configure transfer data size of memory
dma_periph_width_config	configure transfer data size of peripheral
dma_memory_address_generation_config	configure next address increasement algorithm of memory
dma_peripheral_address_generation_config	configure next address increasement algorithm of peripheral
dma_circulation_enable	enable DMA circulation mode
dma_circulation_disable	disable DMA circulation mode
dma_channel_enable	enable DMA channel
dma_channel_disable	disable DMA channel
dma_transfer_direction_config	configure the direction of data transfer on the channel
dma_switch_buffer_mode_config	DMA switch buffer mode config
dma_using_memory_get	DMA using memory get
dma_channel_subperipheral_select	DMA channel peripheral select
dma_flow_controller_config	DMA flow controller configure
dma_switch_buffer_mode_enable	DMA flow controller enable
dma_fifo_status_get	DMA FIFO status get
dma_flag_get	check DMA flag is set or not
dma_flag_clear	clear DMA a channel flag
dma_interrupt_enable	enable DMA interrupt
dma_interrupt_disable	disable DMA interrupt
dma_interrupt_flag_get	get DMA interrupt flag is set or not
dma_interrupt_flag_clear	clear DMA a channel interrupt flag

Structure dma_multi_data_parameter_struct

Table 3-231. Structure dma_multi_data_parameter_struct

Member name	Function description
periph_addr	peripheral base address
periph_width	transfer data size of peripheral
periph_inc	peripheral increasing mode
memory0_addr	memory 0 base address
memory_width	transfer data size of memory
memory_inc	memory increasing mode
memory_burst_width	memory burst width
periph_burst_width	periph burst width
critical_value	DMA circular value
circular_mode	DMA circular mode
direction	channel data transfer direction
number	channel transfer number
priority	channel priority level

Structure dma_single_data_parameter_struct

Table 3-232. Structure dma_single_data_parameter_struct

Member name	Function description
periph_addr	peripheral base address
periph_inc	peripheral increasing mode
memory0_addr	memory 0 base address
memory_inc	memory increasing mode
periph_memory_width	transfer data size of peripheral
circular_mode	DMA circular mode
direction	channel data transfer direction
number	channel transfer number
priority	channel priority level

dma_deinit

The description of dma_deinit is shown as below:

Table 3-233. Function dma_deinit

Function name	dma_deinit
Function prototype	void dma_deinit(uint32_t dma_periph, dma_channel_enum channelx);
Function descriptions	deinitialize DMA a channel registers
Precondition	-
The called functions	-
Input parameter{in}	

dma_periph	DMA peripheral
<i>DMAx(x=0, 1)</i>	DMA peripheral selection
Input parameter{in}	
channelx	DMA channel
<i>DMA_CHx(x=0..7)</i>	DMA channel selection
Output parameter{out}	
-	-
Return value	
-	-

Example:

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

dma_single_data_para_struct_init

The description of dma_single_data_para_struct_init is shown as below:

Table 3-234. Function dma_deinit

Function name	dma_single_data_para_struct_init
Function prototype	void dma_single_data_para_struct_init(dma_single_data_parameter_struct* init_struct);
Function descriptions	initialize the DMA single data mode parameters struct with the default values
Precondition	-
The called functions	-
Input parameter{in}	
init_struct	a dma_single_data_parameter_struct address, the structure members can refer to Table 3-232. Structure dma_single_data_parameter_struct
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* initialize the DMA single data mode parameters struct */
dma_single_data_parameter_struct s_dma_init_struct;
dma_single_data_para_struct_init(&s_dma_init_struct);
```

dma_multi_data_para_struct_init

The description of dma_multi_data_para_struct_init is shown as below:

Table 3-235. Function dma_deinit

Function name	dma_multi_data_para_struct_init
Function prototype	void dma_multi_data_para_struct_init (dma_multi_data_parameter_struct* init_struct);
Function descriptions	initialize the DMA multi data mode parameters struct with the default values
Precondition	-
The called functions	-
Input parameter{in}	
init_struct	Structure for initialization, the structure members can refer to Table 3-237. Function dma_multi_data_mode_init
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* initialize the DMA multi data mode parameters struct */
dma_multi_data_parameter_struct m_dma_init_struct;
dma_multi_data_para_struct_init(&m_dma_init_struct);
```

dma_single_data_mode_init

The description of dma_single_data_mode_init is shown as below:

Table 3-236. Function dma_single_data_mode_init

Function name	dma_single_data_mode_init
Function prototype	void dma_single_data_mode_init(uint32_t dma_periph,dma_channel_enum channelx,dma_single_data_parameter_struct* init_struct);
Function descriptions	initialize DMA single data mode
Precondition	-
The called functions	-
Input parameter{in}	
dma_periph	DMA peripheral
DMAx(x=0,1)	DMAx(x=0,1)
Input parameter{in}	
channelx	specify which DMA channel is initialized
DMA_CHx(x=0..7)	DMA_CHx(x=0..7)
Input parameter{ in }	
init_struct	a dma_single_data_parameter_struct address, the structure members can refer to Table 3-232. Structure dma_single_data_parameter_struct
Output parameter{out}	
-	-
Return value	

-	-
---	---

Example:

```
/* initialize DMA single data mode */
```

```
dma_single_data_parameter_struct dma_single_data_struct;
```

```
dma_single_data_mode_init(DMAx0, DMA_CH0, &dma_single_data_struct);
```

dma_multi_data_mode_init

The description of dma_multi_data_mode_init is shown as below:

Table 3-237. Function dma_multi_data_mode_init

Function name	dma_multi_data_mode_init
Function prototype	void dma_multi_data_mode_init(uint32_t dma_periph,dma_channel_enum channelx,dma_multi_data_parameter_struct* init_struct);
Function descriptions	initialize DMA multi data mode
Precondition	-
The called functions	-
Input parameter{in}	
dma_periph	DMA peripheral
<i>DMAx(x=0,1)</i>	DMA peripheral selection
Input parameter{in}	
channelx	DMA channel
<i>DMA_CHx(x=0..7)</i>	DMA channel selection
Input parameter{in}	
init_struct	Structure for initialization, the structure members can refer to Table 3-237. Function dma_multi_data_mode_init
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* initialize DMA multi data mode */
```

```
dma_multi_data_parameter_struct dma_multi_data_struct;
```

```
dma_init(DMA0, DMA_CH0, &dma_multi_data_struct);
```

dma_periph_address_config

The description of dma_periph_address_config is shown as below:

Table 3-238. Function dma_periph_address_config

Function name	dma_periph_address_config
----------------------	---------------------------

Function prototype	void dma_periph_address_config(uint32_t dma_periph, dma_channel_enum channelx, uint32_t address);
Function descriptions	set DMA peripheral base address
Precondition	-
The called functions	-
Input parameter{in}	
dma_periph	DMA peripheral
<i>DMAx(x=0, 1)</i>	DMA peripheral selection
Input parameter{in}	
channelx	DMA channel
<i>DMA_CHx(x=0..7)</i>	DMA channel selection
Input parameter{in}	
address	peripheral base address
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* set DMA peripheral base address */

#define BANK0_WRITE_START_ADDR ((uint32_t)0x08004000)

dma_periph_address_config(DMA0, DMA_CH0, BANK0_WRITE_START_ADDR);
```

dma_memory_address_config

The description of dma_memory_address_config is shown as below:

Table 3-239. Function dma_memory_address_config

Function name	dma_memory_address_config
Function prototype	void dma_memory_address_config(uint32_t dma_periph,dma_channel_enum channelx,uint8_t memory_flag,uint32_t address);
Function descriptions	set DMA memory base address
Precondition	-
The called functions	-
Input parameter{in}	
dma_periph	DMA peripheral
<i>DMAx(x=0, 1)</i>	DMA peripheral selection
Input parameter{in}	
channelx	DMA channel
<i>DMA_CHx(x=0..7)</i>	DMA channel selection
Input parameter{in}	
memory_flag	DMA memory

<i>DMA_MEMORY_x</i> (<i>x</i> =0, 1)	DMA memory selection
Input parameter{in}	
address	memory base address
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* set DMA Memory base address */
```

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

```
dma_memory_address_config(DMA0, DMA_CH0, DMA_MEMORY_0, (uint32_t) g_destbuf);
```

dma_transfer_number_config

The description of `dma_transfer_number_config` is shown as below:

Table 3-240. Function `dma_transfer_number_config`

Function name	<code>dma_transfer_number_config</code>
Function prototype	<code>void dma_transfer_number_config(uint32_t dma_periph, dma_channel_enum channelx, uint32_t number);</code>
Function descriptions	set the number of remaining data to be transferred by the DMA
Precondition	-
The called functions	-
Input parameter{in}	
dma_periph	DMA peripheral
<i>DMAx</i> (<i>x</i> =0, 1)	DMA peripheral selection
Input parameter{in}	
channelx	DMA channel
<i>DMA_CHx</i> (<i>x</i> =0..7)	DMA channel selection
Input parameter{in}	
number	the number of remaining data to be transferred by the DMA
0-0xffff	number
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* set the number of remaining data to be transferred by the DMA */
```

```
#define TRANSFER_NUM 0x400
```

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

dma_transfer_number_get

The description of dma_transfer_number_get is shown as below:

Table 3-241. Function dma_transfer_number_get

Function name	dma_transfer_number_get
Function prototype	uint32_t dma_transfer_number_get(uint32_t dma_periph, dma_channel_enum channelx);
Function descriptions	get the number of remaining data to be transferred by the DMA
Precondition	-
The called functions	-
Input parameter{in}	
dma_periph	DMA peripheral
<i>DMAx(x=0, 1)</i>	DMA peripheral selection
Input parameter{in}	
channelx	DMA channel
<i>DMA_CHx(x=0..7)</i>	DMA channel selection
Output parameter{out}	
-	-
Return value	
uint32_t(0-0xffff)	the number of remaining data to be transferred by the DMA(0-0xffff)

Example:

```
/* get the number of remaining data to be transferred by the DMA */
```

```
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 3-242. Function dma_priority_config

Function name	dma_priority_config
Function prototype	void dma_priority_config(uint32_t dma_periph, dma_channel_enum channelx, uint32_t priority);
Function descriptions	configure priority level of DMA channel
Precondition	-
The called functions	-
Input parameter{in}	
dma_periph	DMA peripheral
<i>DMAx(x=0, 1)</i>	DMA peripheral selection
Input parameter{in}	

channelx	DMA channel
<i>DMA_CHx(x=0..7)</i>	DMA channel selection
Input parameter{in}	
priority	priority Level of this channel
<i>DMA_PRIORITY_LOW</i>	low priority
<i>DMA_PRIORITY_MEDIUM</i>	medium priority
<i>DMA_PRIORITY_HIGH</i>	high priority
<i>DMA_PRIORITY_ULTRA_HIGH</i>	ultra high priority
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* configure priority level of DMA channel */
```

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

dma_memory_burst_beats_config

The description of dma_memory_burst_beats_config is shown as below:

Table 3-243. Function dma_memory_burst_beats_config

Function name	dma_memory_burst_beats_config
Function prototype	void dma_memory_burst_beats_config (uint32_t dma_periph, dma_channel_enum channelx, uint32_t mbeat);
Function descriptions	configure transfer burst beats of memory
Precondition	-
The called functions	-
Input parameter{in}	
dma_periph	DMA peripheral
<i>DMAx(x=0, 1)</i>	DMA peripheral selection
Input parameter{in}	
channelx	DMA channel
<i>DMA_CHx(x=0..7)</i>	DMA channel selection
Input parameter{in}	
mbeat	transfer burst beats
<i>DMA_MEMORY_BURST_SINGLE</i>	memory transfer single burst
<i>DMA_MEMORY_BURST_4_BEAT</i>	memory transfer 4-beat burst
<i>DMA_MEMORY_BURST_8_BEAT</i>	memory transfer 8-beat burst

<i>DMA_MEMORY_BURST_16_BEAT</i>	memory transfer 16-beat burst
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* configure transfer burst beats of memory */
```

```
dma_memory_burst_beats_config (DMA0, DMA_CH0, DMA_MEMORY_BURST_8_BEAT);
```

dma_periph_burst_beats_config

The description of dma_periph_burst_beats_config is shown as below:

Table 3-244. Function dma_periph_burst_beats_config

Function name	dma_periph_burst_beats_config
Function prototype	void dma_periph_burst_beats_config (uint32_t dma_periph, dma_channel_enum channelx, uint32_t pbeat);
Function descriptions	configure transfer burst beats of peripheral
Precondition	-
The called functions	-
Input parameter{in}	
dma_periph	DMA peripheral
<i>DMAx(x=0, 1)</i>	DMA peripheral selection
Input parameter{in}	
channelx	DMA channel
<i>DMA_CHx(x=0..7)</i>	DMA channel selection
Input parameter{in}	
pbeat	transfer burst beats
<i>DMA_PERIPH_BURST_SINGLE</i>	peripheral transfer single burst
<i>DMA_PERIPH_BURST_4_BEAT</i>	peripheral transfer 4-beat burst
<i>DMA_PERIPH_BURST_8_BEAT</i>	peripheral transfer 8-beat burst
<i>DMA_PERIPH_BURST_16_BEAT</i>	peripheral transfer 16-beat burst
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* configure transfer burst beats of peripheral */
```

```
dma_periph_burst_beats_config(DMA0, DMA_CH0, DMA_PERIPH_BURST_8_BEAT);
```

dma_memory_width_config

The description of dma_memory_width_config is shown as below:

Table 3-245. Function dma_memory_width_config

Function name	dma_memory_width_config
Function prototype	void dma_memory_width_config(uint32_t dma_periph,dma_channel_enum channelx,uint32_t msize);
Function descriptions	configure transfer data size of memory
Precondition	-
The called functions	-
Input parameter{in}	
dma_periph	DMA peripheral
<i>DMAx(x=0,1)</i>	DMA peripheral selection
Input parameter{in}	
channelx	DMA channel
<i>DMA_CHx(x=0..7)</i>	DMA channel selection
Input parameter{in}	
msize	transfer data size of memory
<i>DMA_MEMORY_WIDTH_H_8BIT</i>	transfer data size of memory is 8-bit
<i>DMA_MEMORY_WIDTH_H_16BIT</i>	transfer data size of memory is 16-bit
<i>DMA_MEMORY_WIDTH_H_32BIT</i>	transfer data size of memory is 32-bit
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* configure transfer data size of memory */
```

```
dma_memory_width_config(DMA0, DMA_CH0, DMA_MEMORY_WIDTH_8BIT);
```

dma_periph_width_config

The description of dma_periph_width_config is shown as below:

Table 3-246. Function dma_periph_width_config

Function name	dma_periph_width_config
Function prototype	void dma_periph_width_config (uint32_t dma_periph, dma_channel_enum

	channelx, uint32_t pwidth);
Function descriptions	configure transfer data size of peripheral
Precondition	-
The called functions	-
Input parameter{in}	
dma_periph	DMA peripheral
<i>DMAx(x=0,1)</i>	DMA peripheral selection
Input parameter{in}	
channelx	DMA channel
<i>DMA_CHx(x=0..7)</i>	DMA channel selection
Input parameter{in}	
psize	transfer data size of peripheral
<i>DMA_PERIPHERAL_WIDTH_8BIT</i>	transfer data size of peripheral is 8-bit
<i>DMA_PERIPHERAL_WIDTH_16BIT</i>	transfer data size of peripheral is 16-bit
<i>DMA_PERIPHERAL_WIDTH_32BIT</i>	transfer data size of peripheral is 32-bit
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* configure transfer data size of peripheral */
```

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

dma_memory_address_generation_config

The description of dma_memory_address_generation_config is shown as below:

Table 3-247. Function dma_memory_address_generation_config

Function name	dma_memory_address_generation_config
Function prototype	void dma_memory_address_generation_config(uint32_t dma_periph, dma_channel_enum channelx, uint8_t generation_algorithm);
Function descriptions	configure memory address generation generation_algorithm
Precondition	-
The called functions	-
Input parameter{in}	
dma_periph	DMA peripheral
<i>DMAx(x=0,1)</i>	DMA peripheral selection
Input parameter{in}	
channelx	DMA channel
<i>DMA_CHx(x=0..7)</i>	DMA channel selection

Input parameter{in}	
generation_algorithm	the address generation algorithm
<i>DMA_MEMORY_INCR EASE_ENABLE</i>	next address of memory is increasing address mode
<i>DMA_MEMORY_INCR EASE_DISABLE</i>	next address of memory is fixed address mode
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* configure next address increasement algorithm of memory */
```

```
dma_memory_address_generation_config (DMA0, DMA_CH0, DMA_MEMORY_INCREASE  
_ENABLE);
```

dma_peripheral_address_generation_config

The description of dma_peripheral_address_generation_config is shown as below:

Table 3-248. Function dma_peripheral_address_generation_config

Function name	dma_peripheral_address_generation_config
Function prototype	void dma_peripheral_address_generation_config(uint32_t dma_periph,dma_channel_enum channelx,uint8_t generation_algorithm);
Function descriptions	configure peripheral address generation_algorithm
Precondition	-
The called functions	-
Input parameter{in}	
dma_periph	DMA peripheral
<i>DMAx(x=0,1)</i>	DMA peripheral selection
Input parameter{in}	
channelx	DMA channel
<i>DMA_CHx(x=0..7)</i>	DMA channel selection
Input parameter{in}	
generation_algorithm	the address generation algorithm
<i>DMA_PERIPH_INCRE ASE_ENABLE</i>	next address of peripheral is increasing address mode
<i>DMA_PERIPH_INCRE ASE_DISABLE</i>	next address of peripheral is fixed address mode
<i>DMA_PERIPH_INCRE ASE_FIX</i>	increasing steps of peripheral address is fixed
Output parameter{out}	
-	-
Return value	

-	-
---	---

Example:

```
/* configure next address increasement algorithm of peripheral */
```

```
dma_peripheral_address_generation_config (DMA0, DMA_CH0, DMA_PERIPH_INCREMENT_ENABLE);
```

dma_circulation_enable

The description of dma_circulation_enable is shown as below:

Table 3-249. Function dma_circulation_enable

Function name	dma_circulation_enable
Function prototype	void dma_circulation_enable(uint32_t dma_periph,dma_channel_enum channelx);
Function descriptions	enable DMA circulation mode
Precondition	-
The called functions	-
Input parameter{in}	
dma_periph	DMA peripheral
DMAx(x=0, 1)	DMA peripheral selection
Input parameter{in}	
channelx	DMA channel
DMA_CHx(x=0..7)	DMA channel selection
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* enable DMA circulation mode */
```

```
dma_circulation_enable (DMA0, DMA_CH0);
```

dma_circulation_disable

The description of dma_circulation_disable is shown as below:

Table 3-250. Function dma_circulation_disable

Function name	dma_circulation_disable
Function prototype	void dma_circulation_disable(uint32_t dma_periph,dma_channel_enum channelx);
Function descriptions	disable DMA circulation mode
Precondition	-
The called functions	-

Input parameter{in}	
dma_periph	DMA peripheral
<i>DMAx(x=0, 1)</i>	DMA peripheral selection
Input parameter{in}	
channelx	DMA channel
<i>DMA_CHx(x=0..7)</i>	DMA channel selection
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* disable DMA circulation mode */
```

```
dma_circulation_disable (DMA0, DMA_CH0);
```

dma_channel_enable

The description of dma_channel_enable is shown as below:

Table 3-251. Function dma_channel_enable

Function name	dma_channel_enable
Function prototype	void dma_channel_enable(uint32_t dma_periph,dma_channel_enum channelx);
Function descriptions	enable DMA channel
Precondition	-
The called functions	-
Input parameter{in}	
dma_periph	DMA peripheral
<i>DMAx(x=0, 1)</i>	DMA peripheral selection
Input parameter{in}	
channelx	DMA channel
<i>DMA_CHx(x=0..7)</i>	DMA channel selection
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* enable DMA channel */
```

```
dma_channel_enable (DMA0, DMA_CH0);
```

dma_channel_disable

The description of dma_channel_disable is shown as below:

Table 3-252. Function dma_channel_disable

Function name	dma_channel_disable
Function prototype	void dma_channel_disable(uint32_t dma_periph,dma_channel_enum channelx);
Function descriptions	disable DMA channel
Precondition	-
The called functions	-
Input parameter{in}	
dma_periph	DMA peripheral
DMAx(x=0, 1)	DMA peripheral selection
Input parameter{in}	
channelx	DMA channel
DMA_CHx(x=0..7)	DMA channel selection
Output parameter{out}	
-	-
Return value	
FlagStatus	SET or RESET

Example:

```
/* disable DMA channel */
```

```
dma_channel_disable (DMA0, DMA_CH0);
```

dma_transfer_direction_config

The description of dma_transfer_direction_config is shown as below:

Table 3-253. Function dma_transfer_direction_config

Function name	dma_transfer_direction_config
Function prototype	void dma_transfer_direction_config(uint32_t dma_periph,dma_channel_enum channelx,uint8_t direction);
Function descriptions	configure the direction of data transfer on the channel
Precondition	-
The called functions	-
Input parameter{in}	
dma_periph	DMA peripheral
DMAx(x=0, 1)	DMA peripheral selection
Input parameter{in}	
channelx	DMA channel
DMA_CHx(x=0..7)	DMA channel selection
Input parameter{in}	
direction	specify the direction of data transfer
DMA_PERIPH_TO_MEMORY	read from peripheral and write to memory
DMA_MEMORY_TO_PERIPH	read from memory and write to peripheral

<i>ERIPH</i>	
<i>DMA_MEMORY_TO_MEMORY</i>	read from memory and write to memory
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* configure the direction of data transfer on the channel */
```

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

dma_switch_buffer_mode_config

The description of dma_switch_buffer_mode_config is shown as below:

Table 3-254. Function dma_switch_buffer_mode_config

Function name	dma_switch_buffer_mode_config
Function prototype	void dma_switch_buffer_mode_config(uint32_t dma_periph,dma_channel_enum channelx,uint32_t memory1_addr,uint32_t memory_select);
Function descriptions	DMA switch buffer mode config
Precondition	-
The called functions	-
Input parameter{in}	
dma_periph	DMA peripheral
<i>DMAx(x=0,1)</i>	DMA peripheral selection
Input parameter{in}	
channelx	DMA channel
<i>DMA_CHx(x=0..7)</i>	DMA channel selection
Input parameter{in}	
memory1_addr	memory1 base address
Input parameter{in}	
memory_select	DMA_MEMORY_0 or DMA_MEMORY_1
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* DMA switch buffer mode config */
```

```
dma_switch_buffer_mode_config(DMA0, DMA_CH0,0xaabbccdd,DMA_MEMORY_0);
```

dma_using_memory_get

The description of dma_using_memory_get is shown as below:

Table 3-255. Function dma_using_memory_get

Function name	dma_using_memory_get
Function prototype	uint32_t dma_using_memory_get(uint32_t dma_periph,dma_channel_enum channelx);
Function descriptions	DMA using memory get
Precondition	-
The called functions	-
Input parameter{in}	
dma_periph	DMA peripheral
<i>DMAx(x=0,1)</i>	DMA peripheral selection
Input parameter{in}	
channelx	DMA channel
<i>DMA_CHx(x=0..7)</i>	DMA channel selection
Output parameter{out}	
-	-
Return value	
uint32_t	the using memory

Example:

```
/* DMA using memory get */
```

```
uint32_t using_memory = dma_using_memory_get(DMA0, DMA_CH0);
```

dma_channel_subperipheral_select

The description of dma_channel_subperipheral_select is shown as below:

Table 3-256. Function dma_channel_subperipheral_select

Function name	dma_channel_subperipheral_select
Function prototype	void dma_channel_subperipheral_select(uint32_t dma_periph,dma_channel_enum channelx,dma_subperipheral_enum sub_periph);
Function descriptions	DMA channel peripheral select
Precondition	-
The called functions	-
Input parameter{in}	
dma_periph	DMA peripheral
<i>DMAx(x=0,1)</i>	DMA peripheral selection
Input parameter{in}	
channelx	DMA channel
<i>DMA_CHx(x=0..7)</i>	DMA channel selection

Input parameter{in}	
sub_periph	specify DMA channel peripheral
<i>DMA_SUBPERIx(x=0..7)</i>	specify DMA channel peripheral select
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* DMA channel peripheral select */
```

```
dma_channel_subperipheral_select (DMA0, DMA_CH0, DMA_SUBPERI0);
```

dma_flow_controller_config

The description of dma_flow_controller_config is shown as below:

Table 3-257. Function dma_flow_controller_config

Function name	dma_flow_controller_config
Function prototype	void dma_flow_controller_config(uint32_t dma_periph,dma_channel_enum channelx,uint32_t controller);
Function descriptions	DMA flow controller configure
Precondition	-
The called functions	-
Input parameter{in}	
dma_periph	DMA peripheral
<i>DMAx(x=0, 1)</i>	DMA peripheral selection
Input parameter{in}	
channelx	DMA channel
<i>DMA_CHx(x=0..7)</i>	DMA channel selection
Input parameter{in}	
controller	specify DMA flow controler
<i>DMA_FLOW_CONTROLLER_DMA</i>	DMA is the flow controller
<i>DMA_FLOW_CONTROLLER_PERI</i>	peripheral is the flow controller
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* DMA flow controller configure */
```

`dma_flow_controller_config (DMA0, DMA_CH0, DMA_FLOW_CONTROLLER_DMA);`

dma_switch_buffer_mode_enable

The description of `dma_switch_buffer_mode_enable` is shown as below:

Table 3-258. Function `dma_switch_buffer_mode_enable`

Function name	<code>dma_switch_buffer_mode_enable</code>
Function prototype	<code>void dma_switch_buffer_mode_enable(uint32_t dma_periph,dma_channel_enum channelx,ControlStatus newvalue);</code>
Function descriptions	DMA switch buffer mode enable
Precondition	-
The called functions	-
Input parameter{in}	
dma_periph	DMA peripheral
<i>DMAx(x=0,1)</i>	DMA peripheral selection
Input parameter{in}	
channelx	DMA channel
<i>DMA_CHx(x=0..7)</i>	DMA channel selection
Input parameter{in}	
newvalue	ENABLE or DISABLE
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* DMA switch buffer mode enable */
```

```
dma_switch_buffer_mode_enable(DMA0, DMA_CH0, ENABLE);
```

dma_fifo_status_get

The description of `dma_fifo_status_get` is shown as below:

Table 3-259. Function `dma_fifo_status_get`

Function name	<code>dma_fifo_status_get</code>
Function prototype	<code>uint32_t dma_fifo_status_get(uint32_t dma_periph,dma_channel_enum channelx);</code>
Function descriptions	DMA FIFO status get
Precondition	-
The called functions	-
Input parameter{in}	
dma_periph	DMA peripheral
<i>DMAx(x=0,1)</i>	DMA peripheral selection
Input parameter{in}	

channelx	DMA channel
<i>DMA_CHx(x=0..7)</i>	DMA channel selection
Output parameter{out}	
-	-
Return value	
uint32_t	the using memory

Example:

```
/* DMA FIFO status get */
```

```
uint32_t using_memory dma_fifo_status_get (DMA0, DMA_CH0);
```

dma_flag_get

The description of dma_flag_get is shown as below:

Table 3-260. Function dma_flag_get

Function name	dma_flag_get
Function prototype	FlagStatus dma_flag_get(uint32_t dma_periph,dma_channel_enum channelx,uint32_t flag);
Function descriptions	get DMA flag is set or not
Precondition	-
The called functions	-
Input parameter{in}	
dma_periph	DMA peripheral
<i>DMAx(x=0, 1)</i>	DMA peripheral selection
Input parameter{in}	
channelx	DMA channel
<i>DMA_CHx(x=0..7)</i>	DMA channel selection
Input parameter{in}	
flag	specify get which flag
<i>DMA_FLAG_FEE</i>	FIFO error and exception flag
<i>DMA_FLAG_SDE</i>	single data mode exception flag
<i>DMA_FLAG_TAE</i>	transfer access error flag
<i>DMA_FLAG_HTF</i>	half transfer finish flag
<i>DMA_FLAG_FTF</i>	full transger finish flag
Output parameter{out}	
-	-
Return value	
FlagStatus	SET or RESET

Example:

```
/* check DMA0 channel0 flag is set or not */
```

```
FlagStatus flag_status = dma_flag_get (DMA0, DMA_CH0, DMA_FLAG_FEE);
```

dma_flag_clear

The description of dma_flag_clear is shown as below:

Table 3-261. Function dma_flag_clear

Function name	dma_flag_clear
Function prototype	void dma_flag_clear(uint32_t dma_periph, dma_channel_enum channelx, uint32_t flag);
Function descriptions	clear DMA a channel flag
Precondition	-
The called functions	-
Input parameter{in}	
dma_periph	DMA peripheral
<i>DMAx(x=0, 1)</i>	DMA peripheral selection
Input parameter{in}	
channelx	DMA channel
<i>DMA_CHx(x=0..7)</i>	DMA channel selection
Input parameter{in}	
flag	specify get which flag
<i>DMA_FLAG_FEE</i>	FIFO error and exception flag
<i>DMA_FLAG_SDE</i>	single data mode exception flag
<i>DMA_FLAG_TAE</i>	transfer access error flag
<i>DMA_FLAG_HTF</i>	half transfer finish flag
<i>DMA_FLAG_FTF</i>	full transger finish flag
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* clear DMA0 channel0 flag */
dma_flag_clear(DMA0, DMA_CH0, DMA_FLAG_FTF);
```

dma_interrupt_enable

The description of dma_interrupt_enable is shown as below:

Table 3-262. Function dma_interrupt_enable

Function name	dma_interrupt_enable
Function prototype	void dma_interrupt_enable(uint32_t dma_periph, dma_channel_enum channelx, uint32_t interrupt);
Function descriptions	enable DMA interrupt
Precondition	-
The called functions	-

Input parameter{in}	
dma_periph	DMA peripheral
<i>DMAx(x=0, 1)</i>	DMA peripheral selection
Input parameter{in}	
channelx	DMA channel
<i>DMA_CHx(x=0..7)</i>	DMA channel selection
Input parameter{in}	
interrupt	DMA interrupt source
<i>DMA_INT_SDE</i>	single data mode exception interrupt enable
<i>DMA_INT_TAE</i>	transfer access error interrupt enable
<i>DMA_INT_HTF</i>	half transfer finish interrupt enable
<i>DMA_INT_FTF</i>	full transfer finish interrupt enable
<i>DMA_INT_FEE</i>	FIFO exception interrupt enable
Output parameter{out}	
-	-
Return value	
-	-

Example:

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

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

dma_interrupt_disable

The description of dma_interrupt_disable is shown as below:

Table 3-263. Function dma_interrupt_disable

Function name	dma_interrupt_disable
Function prototype	void dma_interrupt_disable(uint32_t dma_periph, dma_channel_enum channelx, uint32_t interrupt);
Function descriptions	disable DMA interrupt
Precondition	-
The called functions	-
Input parameter{in}	
dma_periph	DMA peripheral
<i>DMAx(x=0, 1)</i>	DMA peripheral selection
Input parameter{in}	
channelx	DMA channel
<i>DMA_CHx(x=0..7)</i>	DMA channel selection
Input parameter{in}	
interrupt	DMA interrupt source
<i>DMA_INT_SDE</i>	single data mode exception interrupt enable
<i>DMA_INT_TAE</i>	transfer access error interrupt enable
<i>DMA_INT_HTF</i>	half transfer finish interrupt enable

<i>DMA_INT_FTF</i>	full transfer finish interrupt enable
<i>DMA_INT_FEE</i>	FIFO exception interrupt enable
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* DMA0 channel0 interrupt disable */
```

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

dma_interrupt_flag_get

The description of dma_interrupt_flag_get is shown as below:

Table 3-264. Function dma_interrupt_flag_get

Function name	dma_interrupt_flag_get
Function prototype	FlagStatus dma_interrupt_flag_get(uint32_t dma_periph, dma_channel_enum channelx, uint32_t interrupt);
Function descriptions	get DMA interrupt flag is set or not
Precondition	-
The called functions	-
Input parameter{in}	
dma_periph	DMA peripheral
<i>DMAx(x=0, 1)</i>	DMA peripheral selection
Input parameter{in}	
channelx	DMA channel
<i>DMA_CHx(x=0..7)</i>	DMA channel selection
Input parameter{in}	
interrupt	specify get which flag
<i>DMA_INT_FLAG_FEE</i>	FIFO error and exception interrupt flag
<i>DMA_INT_FLAG_SDE</i>	single data mode exception interrupt flag
<i>DMA_INT_FLAG_TAE</i>	transfer access error interrupt flag
<i>DMA_INT_FLAG_HTF</i>	half transfer finish interrupt flag
<i>DMA_INT_FLAG_FTF</i>	full transger finish interrupt flag
Output parameter{out}	
-	-
Return value	
FlagStatus	SET or RESET

Example:

```
/* get DMA0 channel3 interrupt flag is set or not */
```

```
dma_interrupt_flag_get(DMA0, DMA_CH3, DMA_INT_FLAG_FTF);
```

dma_interrupt_flag_clear

The description of dma_interrupt_flag_clear is shown as below:

Table 3-265. Function dma_interrupt_flag_clear

Function name	dma_interrupt_flag_clear
Function prototype	void dma_interrupt_flag_clear(uint32_t dma_periph, dma_channel_enum channelx, uint32_t interrupt);
Function descriptions	clear DMA a channel interrupt flag
Precondition	-
The called functions	-
Input parameter{in}	
dma_periph	DMA peripheral
<i>DMAx(x=0, 1)</i>	DMA peripheral selection
Input parameter{in}	
channelx	DMA channel
<i>DMA_CHx(x=0..7)</i>	DMA channel selection
Input parameter{in}	
interrupt	specify get which flag
<i>DMA_INT_FLAG_FEE</i>	FIFO error and exception interrupt flag
<i>DMA_INT_FLAG_SDE</i>	single data mode exception interrupt flag
<i>DMA_INT_FLAG_TAE</i>	transfer access error interrupt flag
<i>DMA_INT_FLAG_HTF</i>	half transfer finish interrupt flag
<i>DMA_INT_FLAG_FTF</i>	full transger finish interrupt flag
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* clear DMA0 channel3 interrupt flag */
```

```
dma_interrupt_flag_clear(DMA0, DMA_CH3, DMA_INT_FLAG_FTF);
```

3.11. ENET

There is a media access controller (MAC) designed in Ethernet module to support 10/100Mbps interface speed. For more efficient data transfer between Ethernet and memory, a DMA controller is designed in this module. The support interface protocol for Ethernet is media independent interface (MII) and reduced media independent interface (RMII). The ENET registers are listed in chapter [3.11.1](#), the ENET firmware functions are introduced in chapter [3.11.2](#).

3.11.1. Descriptions of Peripheral registers

ENET registers are listed in the table shown as below:

Table 3-266. ENET Registers

Registers	Descriptions
ENET_MAC_CFG	MAC configuration register
ENET_MAC_FRMF	MAC frame filter register
ENET_MAC_HLH	MAC hash list high register
ENET_MAC_HLL	MAC hash list low register
ENET_MAC_PHY_CTL	MAC PHY control register
ENET_MAC_PHY_DATA	MAC PHY data register
ENET_MAC_FCTL	MAC flow control register
ENET_MAC_VLT	MAC VLAN tag register
ENET_MAC_RWFF	MAC remote wakeup frame filter register
ENET_MAC_WUM	MAC wakeup management register
ENET_MAC_DBG	MAC debug register
ENET_MAC_INTF	MAC interrupt flag register
ENET_MAC_INTMSK	MAC interrupt mask register
ENET_MAC_ADDR0H	MAC address 0 high register
ENET_MAC_ADDR0L	MAC address 0 low register
ENET_MAC_ADDR1H	MAC address 1 high register
ENET_MAC_ADDR1L	MAC address 1 low register
ENET_MAC_ADDT2H	MAC address 2 high register
ENET_MAC_ADDR2L	MAC address 2 low register
ENET_MAC_ADDR3H	MAC address 3 high register
ENET_MAC_ADDR3L	MAC address 3 low register
ENET_MAC_FCTH	MAC flow control threshold register
ENET_MSC_CTL	MSC control register
ENET_MSC_RINTF	MSC receive interrupt flag register
ENET_MSC_TINTF	MSC transmit interrupt flag register
ENET_MSC_RINTMSK	MSC receive interrupt mask register

Registers	Descriptions
ENET_MSC_TINTMSK	MSC transmit interrupt mask register
ENET_MSC_SCCNT	MSC transmitted good frames after a single collision counter register
ENET_MSC_MSCCNT	MSC transmitted good frames after more than a single collision counter register
ENET_MSC_TGFCNT	MSC transmitted good frames counter register
ENET_MSC_RFCECNT	MSC received frames with CRC error counter register
ENET_MSC_RFAECNT	MSC received frames with alignment error counter register
ENET_MSC_RGUFcnt	MSC received good unicast frames counter register
ENET_PTP_TSCTL	PTP time stamp control register
ENET_PTP_SSINC	PTP subsecond increment register
ENET_PTP_TSH	PTP time stamp high register
ENET_PTP_TSL	PTP time stamp low register
ENET_PTP_TSUH	PTP time stamp update high register
ENET_PTP_TSUL	PTP time stamp update low register
ENET_PTP_TSADDEND	PTP time stamp addend register
ENET_PTP_ETH	PTP expected time high register
ENET_PTP_ETL	PTP expected time low register
ENET_PTP_TSF	PTP time stamp flag register
ENET_PTP_PPSCTL	PTP PPS control register
ENET_DMA_BCTL	DMA bus control register
ENET_DMA_TPEN	DMA transmit poll enable register
ENET_DMA_RPEN	DMA receive poll enable register
ENET_DMA_RDTAADDR	DMA receive descriptor table address register
ENET_DMA_TDTAADDR	DMA transmit descriptor table address register
ENET_DMA_STAT	DMA status register
ENET_DMA_CTL	DMA control register
ENET_DMA_INTEN	DMA interrupt enable register
ENET_DMA_MFBOCNT	DMA missed frame and buffer overflow counter register
ENET_DMA_RSWDCT	DMA receive state watchdog counter register
ENET_DMA_CTDA	DMA current transmit descriptor address register

Registers	Descriptions
DDR	
ENET_DMA_CRDA DDR	DMA current receive descriptor address register
ENET_DMA_CTBA DDR	DMA current transmit buffer address register
ENET_DMA_CRBA DDR	DMA current receive buffer address register

3.11.2. Descriptions of Peripheral functions

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

Table 3-267. ENET firmware function

Function name	Function description
	main function
enet_deinit	deinitialize the ENET, and reset structure parameters for ENET initialization
enet_initpara_config	configure the parameters which are usually less cared for initialization
enet_init	initialize ENET peripheral with generally concerned parameters and the less cared parameters
enet_software_reset	reset all core internal registers located in CLK_TX and CLK_RX
enet_rxfree_size_get	check receive frame valid and return frame size
enet_descriptors_chain_init	initialize the dma tx/rx descriptors's parameters in chain mode
enet_descriptors_ring_init	initialize the dma tx/rx descriptors's parameters in ring mode
enet_frame_receive	handle current received frame data to application buffer
enet_frame_transmit	handle application buffer data to transmit it
enet_transmit_checksum_config	configure the transmit IP frame checksum offload calculation and insertion
enet_enable	ENET Tx and Rx function enable (include MAC and DMA module)
enet_disable	ENET Tx and Rx function disable (include MAC and DMA module)
enet_mac_address_set	configure MAC address
enet_mac_address_get	get MAC address
enet_flag_get	get the ENET MAC/MSC/PTP/DMA status flag
enet_flag_clear	clear the ENET DMA status flag
enet_interrupt_enable	enable ENET MAC/MSC/DMA interrupt
enet_interrupt_disable	disable ENET MAC/MSC/DMA interrupt
enet_interrupt_flag_get	get ENET MAC/MSC/DMA interrupt flag
enet_interrupt_flag_clear	clear ENET DMA interrupt flag
	MAC function

Function name	Function description
enet_tx_enable	ENET Tx function enable (include MAC and DMA module)
enet_tx_disable	ENET Tx function disable (include MAC and DMA module)
enet_rx_enable	ENET Rx function enable (include MAC and DMA module)
enet_rx_disable	ENET Rx function disable (include MAC and DMA module)
enet_registers_get	put registers value into the application buffer
enet_debug_status_get	get the enet debug status from the debug register
enet_address_filter_enable	enable the MAC address filter
enet_address_filter_disable	disable the MAC address filter
enet_address_filter_config	configure the MAC address filter
enet_phy_config	PHY interface configuration (configure SMI clock and reset PHY chip)
enet_phy_write_read	write to/read from a PHY register
enet_phyloopback_enable	enable the loopback function of phy chip
enet_phyloopback_disable	disable the loopback function of phy chip
enet_forward_feature_enable	enable ENET forward feature
enet_forward_feature_disable	disable ENET forward feature
enet_fliter_feature_enable	enable ENET fliter feature
enet_fliter_feature_disable	disable ENET fliter feature
flow control function	
enet_pauseframe_generate	generate the pause frame, ENET will send pause frame after enable transmit flow control
enet_pauseframe_detect_config	configure the pause frame detect type
enet_pauseframe_config	configure the pause frame parameters
enet_flowcontrol_threshold_config	configure the threshold of the flow control(deactive and active threshold)
enet_flowcontrol_feature_enable	enable ENET flow control feature
enet_flowcontrol_feature_disable	disable ENET flow control feature
DMA function	
enet_dmaprocess_state_get	get the dma transmit/receive process state
enet_dmaprocess_resume	poll the dma transmission/reception enable
enet_rxprocess_check_recovery	check and recover the Rx process
enet_txfifo_flush	flush the ENET transmit fifo, and wait until the flush operation completes
enet_current_desc_address_get	get the transmit/receive address of current descriptor, or current buffer, or descriptor table
enet_desc_information_get	get the Tx or Rx descriptor information
enet_missed_frame_counter_get	get the number of missed frames during receiving
descriptor function	
enet_desc_flag_get	get the bit flag of ENET dma descriptor
enet_desc_flag_set	set the bit flag of ENET dma tx descriptor
enet_desc_flag_clear	clear the bit flag of ENET dma tx descriptor
enet_rx_desc_immediate_receive_co	when receiving the completed, set RS bit in

Function name	Function description
mplete_interrupt	ENET_DMA_STAT register will immediately set
enet_rx_desc_delay_receive_complete_interrupt	when receiving the completed, set RS bit in ENET_DMA_STAT register will is set after a configurable delay time
enet_rxframe_drop	drop current receive frame
enet_dma_feature_enable	enable DMA feature
enet_dma_feature_disable	disable DMA feature
enhanced descriptor	
enet_rx_desc_enhanced_status_get	get the bit of extended status flag in ENET DMA descriptor
enet_desc_select_enhanced_mode	configure descriptor to work in enhanced mode
enet_ptp_enhanced_descriptors_chain_init	initialize the dma Tx/Rx descriptors's parameters in enhanced chain mode with ptp function
enet_ptp_enhanced_descriptors_ring_init	initialize the dma Tx/Rx descriptors's parameters in enhanced ring mode with ptp function
enet_ptpframe_receive_enhanced_mode	receive a packet data with timestamp values to application buffer, when the DMA is in enhanced mode
enet_ptpframe_transmit_enhanced_mode	send data with timestamp values in application buffer as a transmit packet, when the DMA is in enhanced mode
normal descriptor	
enet_desc_select_normal_mode	configure descriptor to work in normal mode
enet_ptp_normal_descriptors_chain_init	initialize the dma Tx/Rx descriptors's parameters in normal chain mode with ptp function
enet_ptp_normal_descriptors_ring_init	initialize the dma Tx/Rx descriptors's parameters in normal ring mode with ptp function
enet_ptpframe_receive_normal_mode	receive a packet data with timestamp values to application buffer, when the DMA is in normal mode
enet_ptpframe_transmit_normal_mode	send data with timestamp values in application buffer as a transmit packet, when the DMA is in normal mode
WUM function	
enet_wum_filter_register_pointer_reset	wakeup frame filter register pointer reset
enet_wum_filter_config	set the remote wakeup frame registers
enet_wum_feature_enable	enable wakeup management features
enet_wum_feature_disable	disable wakeup management features
MSC function	
enet_msc_counters_reset	reset the MAC statistics counters
enet_msc_feature_enable	enable the MAC statistics counter features
enet_msc_feature_disable	disable the MAC statistics counter features
enet_msc_counters_preset_config	configure MAC statistics counters preset mode
enet_msc_counters_get	get MAC statistics counter
PTP function	
enet_ptp_subsecond_2_nanosecond	change subsecond to nanosecond

Function name	Function description
enet_ptp_nanosecond_2_subsecond	change nanosecond to subsecond
enet_ptp_feature_enable	enable the PTP features
enet_ptp_feature_disable	disable the PTP features
enet_ptp_timestamp_function_config	configure the PTP timestamp function
enet_ptp_subsecond_increment_config	configure the PTP system time subsecond increment value
enet_ptp_timestamp_addend_config	adjusting the PTP clock frequency only in fine update mode
enet_ptp_timestamp_update_config	initializing or adding/subtracting to second of the PTP system time
enet_ptp_expected_time_config	configure the PTP expected target time
enet_ptp_system_time_get	get the PTP current system time
enet_ptp_pps_output_frequency_config	configure the PPS output frequency
enet_ptp_start	configure and start PTP timestamp counter
enet_ptp_finecorrection_adjfreq	adjust frequency in fine method by configure addend register
enet_ptp_coarsecorrection_systime_update	update system time in coarse method
enet_ptp_finecorrection_settime	set system time in fine method
enet_ptp_flag_get	get the ptp flag status
Other	
enet_initpara_reset	reset the ENET initpara struct, call it before using enet_initpara_config()

Structure enet_initpara_struct

Table 3-268. Structure enet_initpara_struct

member name	Function description
option_enable	select which function to configure
forward_frame	frame forward related parameters
dmabus_mode	DMA bus mode related parameters
dma_maxburst	DMA max burst related parameters
dma_arbitration	DMA Tx and Rx arbitration related parameters
store_forward_mode	store forward mode related parameters
dma_function	DMA control related parameters
vlan_config	VLAN tag related parameters
flow_control	flow control related parameters
hashtable_high	hash list high 32-bit related parameters
hashtable_low	hash list low 32-bit related parameters
framesfilter_mode	frame filter control related parameters
halfduplex_param	halfduplex related parameters
timer_config	frame timer related parameters
interframegap	inter frame gap related parameters

Structure enet_descriptors_struct

Table 3-269. Structure enet_descriptors_struct

member name	Function description
status	status
control_buffer_size	control and buffer1, buffer2 lengths
buffer1_addr	buffer1 address pointer/timestamp low
buffer2_next_desc_addr	buffer2 or next descriptor address pointer/timestamp high
extended_status	extended status
reserved	reserved
timestamp_low	timestamp low
timestamp_high	timestamp high

Structure enet_ptp_systime_struct

Table 3-270. Structure enet_ptp_systime_struct

member name	Function description
second	second of system time
nanosecond	nanosecond of system time
sign	sign of system time

Enum enet_flag_enum

Table 3-271. Enum enet_flag_enum

member name	Function description
ENET_MAC_FLAG_MPKR	magic packet received flag
ENET_MAC_FLAG_WUFR	wakeup frame received flag
ENET_MAC_FLAG_FLOWCONTROL	flow control status flag
ENET_MAC_FLAG_WUM	WUM status flag
ENET_MAC_FLAG_MSC	MSC status flag
ENET_MAC_FLAG_MSCR	MSC receive status flag
ENET_MAC_FLAG_MSCT	MSC transmit status flag
ENET_MAC_FLAG_TMST	timestamp trigger status flag
ENET_PTP_FLAG_TSSCO	timestamp second counter overflow flag

ENET_PTP_FLAG_ TTM	target time match flag
ENET_MSC_FLAG_ RFCE	received frames CRC error flag
ENET_MSC_FLAG_ RFAE	received frames alignment error flag
ENET_MSC_FLAG_ RGUF	received good unicast frames flag
ENET_MSC_FLAG_ TGFSC	transmitted good frames single collision flag
ENET_MSC_FLAG_ TGFMSC	transmitted good frames more single collision flag
ENET_MSC_FLAG_ TGF	transmitted good frames flag
ENET_DMA_FLAG_ TS	transmit status flag
ENET_DMA_FLAG_ TPS	transmit process stopped status flag
ENET_DMA_FLAG_ TBU	transmit buffer unavailable status flag
ENET_DMA_FLAG_ TJT	transmit jabber timeout status flag
ENET_DMA_FLAG_ RO	receive overflow status flag
ENET_DMA_FLAG_ TU	transmit underflow status flag
ENET_DMA_FLAG_ RS	receive status flag
ENET_DMA_FLAG_ RBU	receive buffer unavailable status flag
ENET_DMA_FLAG_ RPS	receive process stopped status flag
ENET_DMA_FLAG_ RWT	receive watchdog timeout status flag
ENET_DMA_FLAG_ ET	early transmit status flag
ENET_DMA_FLAG_ FBE	fatal bus error status flag
ENET_DMA_FLAG_ ER	early receive status flag
ENET_DMA_FLAG_ AI	abnormal interrupt summary flag
ENET_DMA_FLAG_ 	normal interrupt summary flag

NI	
ENET_DMA_FLAG_EB_DMA_ERROR	error during data transfer by RxDMA/TxDMA flag
ENET_DMA_FLAG_EB_TRANSFER_ERROR	error during write/read transfer flag
ENET_DMA_FLAG_EB_ACCESS_ERROR	error during data buffer/descriptor access flag
ENET_DMA_FLAG_MSC	MSC status flag
ENET_DMA_FLAG_WUM	WUM status flag
ENET_DMA_FLAG_TST	timestamp trigger status flag

Enum enet_flag_clear_enum

Table 3-272. Enum enet_flag_clear_enum

member name	Function description
ENET_DMA_FLAG_TS_CLR	transmit status flag clear
ENET_DMA_FLAG_TPS_CLR	transmit process stopped status flag clear
ENET_DMA_FLAG_TBU_CLR	transmit buffer unavailable status flag clear
ENET_DMA_FLAG_TJT_CLR	transmit jabber timeout status flag clear
ENET_DMA_FLAG_RO_CLR	receive overflow status flag clear
ENET_DMA_FLAG_TU_CLR	transmit underflow status flag clear
ENET_DMA_FLAG_RS_CLR	receive status flag clear
ENET_DMA_FLAG_RBU_CLR	receive buffer unavailable status flag clear
ENET_DMA_FLAG_RPS_CLR	receive process stopped status flag clear
ENET_DMA_FLAG_RWT_CLR	receive watchdog timeout status flag clear
ENET_DMA_FLAG_ET_CLR	early transmit status flag clear
ENET_DMA_FLAG_FBE_CLR	fatal bus error status flag clear

ENET_DMA_FLAG_ER_CLR	early receive status flag clear
ENET_DMA_FLAG_AI_CLR	abnormal interrupt summary flag clear
ENET_DMA_FLAG_NI_CLR	normal interrupt summary flag clear

Enum enet_int_enum

Table 3-273. Enum enet_int_enum

member name	Function description
ENET_MAC_INT_WUMIM	WUM interrupt mask
ENET_MAC_INT_TSTMIM	timestamp trigger interrupt mask
ENET_MSC_INT_RFCEIM	received frame CRC error interrupt mask
ENET_MSC_INT_RFAEIM	received frames alignment error interrupt mask
ENET_MSC_INT_RGUFIM	received good unicast frames interrupt mask
ENET_MSC_INT_TGFSCIM	transmitted good frames single collision interrupt mask
ENET_MSC_INT_TGFMSCIM	transmitted good frames more single collision interrupt mask
ENET_MSC_INT_TGFIM	transmitted good frames interrupt mask
ENET_DMA_INT_TIE	transmit interrupt enable
ENET_DMA_INT_TPSIE	transmit process stopped interrupt enable
ENET_DMA_INT_TBUIE	transmit buffer unavailable interrupt enable
ENET_DMA_INT_TJTIE	transmit jabber timeout interrupt enable
ENET_DMA_INT_ROIE	receive overflow interrupt enable
ENET_DMA_INT_TUIE	transmit underflow interrupt enable
ENET_DMA_INT_RIE	receive interrupt enable
ENET_DMA_INT_RBUIE	receive buffer unavailable interrupt enable
ENET_DMA_INT_RPSIE	receive process stopped interrupt enable

<i>PSIE</i>	
<i>ENET_DMA_INT_RWTIE</i>	receive watchdog timeout interrupt enable
<i>ENET_DMA_INT_ETIE</i>	early transmit interrupt enable
<i>ENET_DMA_INT_FBEIE</i>	fatal bus error interrupt enable
<i>ENET_DMA_INT_ERIE</i>	early receive interrupt enable
<i>ENET_DMA_INT_AIE</i>	abnormal interrupt summary enable
<i>ENET_DMA_INT_NIE</i>	normal interrupt summary enable

Enum enet_int_flag_enum

Table 3-274. Enum enet_int_flag_enum

member name	Function description
<i>ENET_MAC_INT_FLAG_WUM</i>	WUM status flag
<i>ENET_MAC_INT_FLAG_MSC</i>	MSC status flag
<i>ENET_MAC_INT_FLAG_MSCR</i>	MSC receive status flag
<i>ENET_MAC_INT_FLAG_MSCT</i>	MSC transmit status flag
<i>ENET_MAC_INT_FLAG_TMST</i>	time stamp trigger status flag
<i>ENET_MSC_INT_FLAG_RFCE</i>	received frames CRC error flag
<i>ENET_MSC_INT_FLAG_RFAE</i>	received frames alignment error flag
<i>ENET_MSC_INT_FLAG_RGUF</i>	received good unicast frames flag
<i>ENET_MSC_INT_FLAG_TGFSC</i>	transmitted good frames single collision flag
<i>ENET_MSC_INT_FLAG_TGFMSC</i>	transmitted good frames more single collision flag
<i>ENET_MSC_INT_FLAG_TGF</i>	transmitted good frames flag
<i>ENET_DMA_INT_FLAG_TS</i>	transmit status flag
<i>ENET_DMA_INT_FLAG_TPS</i>	transmit process stopped status flag

<i>ENET_DMA_INT_F</i> <i>LAG_TBU</i>	transmit buffer unavailable status flag
<i>ENET_DMA_INT_F</i> <i>LAG_TJT</i>	transmit jabber timeout status flag
<i>ENET_DMA_INT_F</i> <i>LAG_RO</i>	receive overflow status flag
<i>ENET_DMA_INT_F</i> <i>LAG_TU</i>	transmit underflow status flag
<i>ENET_DMA_INT_F</i> <i>LAG_RS</i>	receive status flag
<i>ENET_DMA_INT_F</i> <i>LAG_RBU</i>	receive buffer unavailable status flag
<i>ENET_DMA_INT_F</i> <i>LAG_RPS</i>	receive process stopped status flag
<i>ENET_DMA_INT_F</i> <i>LAG_RWT</i>	receive watchdog timeout status flag
<i>ENET_DMA_INT_F</i> <i>LAG_ET</i>	early transmit status flag
<i>ENET_DMA_INT_F</i> <i>LAG_FBE</i>	fatal bus error status flag
<i>ENET_DMA_INT_F</i> <i>LAG_ER</i>	early receive status flag
<i>ENET_DMA_INT_F</i> <i>LAG_AI</i>	abnormal interrupt summary flag
<i>ENET_DMA_INT_F</i> <i>LAG_NI</i>	normal interrupt summary flag
<i>ENET_DMA_INT_F</i> <i>LAG_MSC</i>	MSC status flag
<i>ENET_DMA_INT_F</i> <i>LAG_WUM</i>	WUM status flag
<i>ENET_DMA_INT_F</i> <i>LAG_TST</i>	timestamp trigger status flag

Enum enet_int_flag_clear_enum

Table 3-275. Enum enet_int_flag_clear_enum

member name	Function description
<i>ENET_DMA_INT_F</i> <i>LAG_TS_CLR</i>	transmit status flag
<i>ENET_DMA_INT_F</i> <i>LAG_TPS_CLR</i>	transmit process stopped status flag
<i>ENET_DMA_INT_F</i> <i>LAG_TBU_CLR</i>	transmit buffer unavailable status flag
<i>ENET_DMA_INT_F</i>	transmit jabber timeout status flag

LAG_TJT_CLR	
ENET_DMA_INT_F LAG_RO_CLR	receive overflow status flag
ENET_DMA_INT_F LAG_TU_CLR	transmit underflow status flag
ENET_DMA_INT_F LAG_RS_CLR	receive status flag
ENET_DMA_INT_F LAG_RBU_CLR	receive buffer unavailable status flag
ENET_DMA_INT_F LAG_RPS_CLR	receive process stopped status flag
ENET_DMA_INT_F LAG_RWT_CLR	receive watchdog timeout status flag
ENET_DMA_INT_F LAG_ET_CLR	early transmit status flag
ENET_DMA_INT_F LAG_FBE_CLR	fatal bus error status flag
ENET_DMA_INT_F LAG_ER_CLR	early receive status flag
ENET_DMA_INT_F LAG_AI_CLR	abnormal interrupt summary flag
ENET_DMA_INT_F LAG_NI_CLR	normal interrupt summary flag

Enum enet_desc_reg_enum

Table 3-276. Enum enet_desc_reg_enum

member name	Function description
ENET_RX_DESC_TABLE	the start address of the receive descriptor table
ENET_RX_CURRENT_DESC	the start descriptor address of the current receive descriptor read by the RxDMA controller
ENET_RX_CURRENT_BUFFER	the current receive buffer address being read by the RxDMA controller
ENET_TX_DESC_TABLE	the start address of the transmit descriptor table
ENET_TX_CURRENT_DESC	the start descriptor address of the current transmit descriptor read by the TxDMA controller
ENET_TX_CURRENT_BUFFER	the current transmit buffer address being read by the TxDMA controller

Enum enet_msc_counter_enum

Table 3-277. Enum enet_msc_counter_enum

member name	Function description
<i>ENET_MSC_TX_SCCNT</i>	MSC transmitted good frames after a single collision counter
<i>ENET_MSC_TX_MSCCNT</i>	MSC transmitted good frames after more than a single collision counter
<i>ENET_MSC_TX_TGFCNT</i>	MSC transmitted good frames counter
<i>ENET_MSC_RX_RFCECNT</i>	MSC received frames with CRC error counter
<i>ENET_MSC_RX_RFAECNT</i>	MSC received frames with alignment error counter
<i>ENET_MSC_RX_RGUFCNT</i>	MSC received good unicast frames counter

Enum enet_option_enum

Table 3-278. Enum enet_option_enum

member name	Function description
<i>FORWARD_OPTION</i>	choose to configure the frame forward related parameters
<i>DMABUS_OPTION</i>	choose to configure the DMA bus mode related parameters
<i>DMA_MAXBURST_OPTION</i>	choose to configure the DMA max burst related parameters
<i>DMA_ARBITRATION_OPTION</i>	choose to configure the DMA arbitration related parameters
<i>STORE_OPTION</i>	choose to configure the store forward mode related parameters
<i>DMA_OPTION</i>	choose to configure the DMA related parameters
<i>VLAN_OPTION</i>	choose to configure vlan related parameters
<i>FLOWCTL_OPTION</i>	choose to configure flow control related parameters
<i>HASHH_OPTION</i>	choose to configure hash high
<i>HASHL_OPTION</i>	choose to configure hash low
<i>FILTER_OPTION</i>	choose to configure frame filter related parameters
<i>HALFDUPLEX_OPTION</i>	choose to configure halfduplex mode related parameters
<i>TIMER_OPTION</i>	choose to configure time counter related parameters
<i>INTERFRAMEGAP_OPTION</i>	choose to configure the inter frame gap related parameters

Enum enet_mediamode_enum

Table 3-279. Enum enet_mediamode_enum

member name	Function description
<i>ENET_AUTO_NEGOTIATION</i>	PHY auto negotiation
<i>ENET_100M_FULLDUPLEX</i>	100Mbit/s, full-duplex
<i>ENET_100M_HALFDUPLEX</i>	100Mbit/s, half-duplex
<i>ENET_10M_FULLDUPLEX</i>	10Mbit/s, full-duplex
<i>ENET_10M_HALFDUPLEX</i>	10Mbit/s, half-duplex
<i>ENET_LOOPBACKMODE</i>	MAC in loopback mode at the MII

Enum enet_chksumconf_enum

Table 3-280. Enum enet_chksumconf_enum

member name	Function description
<i>ENET_NO_AUTOCHECKSUM</i>	disable IP frame checksum function
<i>ENET_AUTOCHECKSUM_DROP_FAILFRAMES</i>	enable IP frame checksum function
<i>ENET_AUTOCHECKSUM_ACCEPT_FAILFRAMES</i>	enable IP frame checksum function, and the received frame with only payload error but no other errors will not be dropped

Enum enet_frmrecept_enum

Table 3-281. Enum enet_frmrecept_enum

member name	Function description
<i>ENET_PROMISCUOUS_MODE</i>	promiscuous mode enabled
<i>ENET_RECEIVEALL</i>	all received frame are forwarded to application
<i>ENET_BROADCAST_FRAMES_PASS</i>	the address filters pass all received broadcast frames
<i>ENET_BROADCAST_FRAMES_DROP</i>	the address filters filter all incoming broadcast frames

Enum enet_registers_type_enum

Table 3-282. Enum enet_registers_type_enum

member name	Function description
<i>ALL_MAC_REG</i>	get the registers within the offset scope range ENET_MAC_CFG to ENET_MAC_FCTH
<i>ALL_MSC_REG</i>	get the registers within the offset scope range ENET_MSC_CTL to ENET_MSC_RGUFCNT
<i>ALL_PTP_REG</i>	get the registers within the offset scope range ENET_PTP_TSCTL to ENET_PTP_PPSCTL
<i>ALL_DMA_REG</i>	get the registers within the offset scope range ENET_DMA_BCTL to ENET_DMA_CRBADDR

Enum enet_dmadirection_enum

Table 3-283. Enum enet_dmadirection_enum

member name	Function description
<i>ENET_DMA_TX</i>	DMA Tx descriptors
<i>ENET_DMA_RX</i>	DMA Rx descriptors

Enum enet_phydirection_enum

Table 3-284. Enum enet_phydirection_enum

member name	Function description
<i>ENET_PHY_WRITE</i>	write data to phy register
<i>ENET_PHY_READ</i>	read data from phy register

Enum enet_regdirection_enum

Table 3-285. Enum enet_regdirection_enum

member name	Function description
<i>ENET_REG_READ</i>	read register
<i>ENET_REG_WRITE</i>	write register

Enum enet_macaddress_enum

Table 3-286. Enum enet_macaddress_enum

member name	Function description
<i>ENET_MAC_ADDR</i> <i>ESS0</i>	set MAC address 0 filter
<i>ENET_MAC_ADDR</i> <i>ESS1</i>	set MAC address 1 filter
<i>ENET_MAC_ADDR</i> <i>ESS2</i>	set MAC address 2 filter
<i>ENET_MAC_ADDR</i>	set MAC address 3 filter

ESS3	
------	--

Enum enet_descstate_enum

Table 3-287. Enum enet_descstate_enum

member name	Function description
<i>TXDESC_COLLISION_COUNT</i>	the number of collisions occurred before the frame was transmitted
<i>TXDESC_BUFFER_1_ADDR</i>	the buffer1 address of the Tx frame
<i>RXDESC_FRAME_LENGTH</i>	the byte length of the received frame that was transferred to the buffer
<i>RXDESC_BUFFER_1_SIZE</i>	receive buffer 1 size
<i>RXDESC_BUFFER_2_SIZE</i>	receive buffer 2 size
<i>RXDESC_BUFFER_1_ADDR</i>	the buffer1 address of the Rx frame

Enum enet_msc_preset_enum

Table 3-288. Enum enet_msc_preset_enum

member name	Function description
<i>ENET_MSC_PRESET_NONE</i>	do not preset MSC counter
<i>ENET_MSC_PRESET_HALF</i>	preset all MSC counters to almost-half(0x7FFF FFF0) value
<i>ENET_MSC_PRESET_FULL</i>	preset all MSC counters to almost-full(0xFFFF FFF0) value

enet_deinit

The description of enet_deinit is shown as below:

Table 3-289. Function enet_deinit

Function name	enet_deinit
Function prototype	void enet_deinit(void);
Function descriptions	deinitialize the ENET, and reset structure parameters for ENET initialization
Precondition	-
The called functions	rcu_periph_reset_enable() /rcu_periph_reset_disable()/enet_initpara_reset()
Input parameter{in}	
-	-
Output parameter{out}	
-	-
Return value	

-	-
---	---

Example:

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

```
enet_deinit( );
```

enet_initpara_config

The description of enet_initpara_config is shown as below:

Table 3-290. Function enet_initpara_config

Function name	enet_initpara_config
Function prototype	void enet_initpara_config(enet_option_enum option, uint32_t para)
Function descriptions	configure the parameters which are usually less cared for initialization, this function must be called before enet_init(), otherwise configuration will be no effect
Precondition	enet_initpara_reset(void)
The called functions	-
Input parameter{in}	
option	different function option, which is related to several parameters only one parameter can be selected which is shown as below
FORWARD_OPTION	choose to configure the frame forward related parameters
DMABUS_OPTION	choose to configure the DMA bus mode related parameters
DMA_MAXBURST_OPTION	choose to configure the DMA max burst related parameters
DMA_ARBITRATION_OPTION	choose to configure the DMA arbitration related parameters
STORE_OPTION	choose to configure the store forward mode related parameters
DMA_OPTION	choose to configure the DMA related parameters
VLAN_OPTION	choose to configure vlan related parameters
FLOWCTL_OPTION	choose to configure flow control related parameters
HASHH_OPTION	choose to configure hash high
HASHL_OPTION	choose to configure hash low
FILTER_OPTION	choose to configure frame filter related parameters
HALFDUPLEX_OPTION	choose to configure halfduplex mode related parameters
TIMER_OPTION	choose to configure time counter related parameters
INTERFRAMEGAP_OPTION	choose to configure the inter frame gap related parameters
Input parameter{in}	
para (the value according to the parameter option)	all the related values should be configured which are shown as below example: para = (value1 value2 value3...)
When value of parameter option is FORWARD_OPTION	

value1	ENET_AUTO_PADCRC_DROP_ENABLE / ENET_AUTO_PADCRC_DROP_DISABLE
value2	ENET_FORWARD_ERRFRAMES_ENABLE / ENET_FORWARD_ERRFRAMES_DISABLE
value3	ENET_FORWARD_UNDEFSZ_GOODFRAMES_ENABLE / ENET_FORWARD_UNDEFSZ_GOODFRAMES_DISABLE
value4	ENET_FORWARD_UNDEFSZ_GOODFRAMES_ENABLE / ENET_FORWARD_UNDEFSZ_GOODFRAMES_DISABLE
When value of parameter option is DMABUS_OPTION	
value1	ENET_ADDRESS_ALIGN_ENABLE / ENET_ADDRESS_ALIGN_DISABLE
value2	ENET_FIXED_BURST_ENABLE / ENET_FIXED_BURST_DISABLE
value3	ENET_MIXED_BURST_ENABLE / ENET_MIXED_BURST_DISABLE
When value of parameter option is DMA_MAXBURST_OPTION	
value1	ENET_RXDP_1BEAT / ENET_RXDP_2BEAT / ENET_RXDP_4BEAT / ENET_RXDP_8BEAT / ENET_RXDP_16BEAT / ENET_RXDP_32BEAT / ENET_RXDP_4xPGBL_4BEAT / ENET_RXDP_4xPGBL_8BEAT / ENET_RXDP_4xPGBL_16BEAT / ENET_RXDP_4xPGBL_32BEAT / ENET_RXDP_4xPGBL_64BEAT / ENET_RXDP_4xPGBL_128BEAT
value2	ENET_PGBL_1BEAT / ENET_PGBL_2BEAT / ENET_PGBL_4BEAT / ENET_PGBL_8BEAT / ENET_PGBL_16BEAT / ENET_PGBL_32BEAT / ENET_PGBL_4xPGBL_4BEAT / ENET_PGBL_4xPGBL_8BEAT / ENET_PGBL_4xPGBL_16BEAT / ENET_PGBL_4xPGBL_32BEAT / ENET_PGBL_4xPGBL_64BEAT / ENET_PGBL_4xPGBL_128BEAT
value3	ENET_RXTX_DIFFERENT_PGBL / ENET_RXTX_SAME_PGBL
When value of parameter option is DMA_ARBITRATION_OPTION	
value1	ENET_ARBITRATION_RXPRIORTX
value2	ENET_ARBITRATION_RXTX_1_1 / ENET_ARBITRATION_RXTX_2_1 / ENET_ARBITRATION_RXTX_3_1 / ENET_ARBITRATION_RXTX_4_1
When value of parameter option is STORE_OPTION	
value1	ENET_RX_MODE_STOREFORWARD / ENET_RX_MODE_CUTTHROUGH
value2	ENET_TX_MODE_STOREFORWARD / ENET_TX_MODE_CUTTHROUGH
value3	ENET_RX_THRESHOLD_64BYTES / ENET_RX_THRESHOLD_32BYTES / ENET_RX_THRESHOLD_96BYTES / ENET_RX_THRESHOLD_128BYTES
value4	ENET_TX_THRESHOLD_64BYTES / ENET_TX_THRESHOLD_128BYTES / ENET_TX_THRESHOLD_192BYTES / ENET_TX_THRESHOLD_256BYTES / ENET_TX_THRESHOLD_40BYTES / ENET_TX_THRESHOLD_32BYTES / ENET_TX_THRESHOLD_24BYTES / ENET_TX_THRESHOLD_16BYTES
When value of parameter option is DMA_OPTION	
value1	ENET_FLUSH_RXFRAME_ENABLE / ENET_FLUSH_RXFRAME_DISABLE
value2	ENET_SECONDFRAME_OPT_ENABLE /

	<i>ENET_SECONDFRAME_OPT_DISABLE</i>
value3	<i>ENET_ENHANCED_DESCRIPTOR/ ENET_NORMAL_DESCRIPTOR</i>
When value of parameter option is <i>VLAN_OPTION</i>	
value1	<i>ENET_VLANTAGCOMPARISON_12BIT/</i> <i>ENET_VLANTAGCOMPARISON_16BIT</i>
value2	<i>MAC_VLT_VLTI(regval)</i>
When value of parameter option is <i>FLOWCTL_OPTION</i>	
value1	<i>MAC_FCTL_PTM(regval)</i>
value2	<i>ENET_ZERO_QUANTA_PAUSE_ENABLE /</i> <i>ENET_ZERO_QUANTA_PAUSE_DISABLE</i>
value3	<i>ENET_PAUSETIME_MINUS4 / ENET_PAUSETIME_MINUS28 /</i> <i>ENET_PAUSETIME_MINUS144/ENET_PAUSETIME_MINUS256</i>
value4	<i>ENET_MAC0_AND_UNIQUE_ADDRESS_PAUSEDTECT /</i> <i>ENET_UNIQUE_PAUSEDTECT</i>
value5	<i>ENET_RX_FLOWCONTROL_ENABLE /</i> <i>ENET_RX_FLOWCONTROL_DISABLE</i>
value6	<i>ENET_TX_FLOWCONTROL_ENABLE /</i> <i>ENET_TX_FLOWCONTROL_DISABLE</i>
When value of parameter option is <i>HASHH_OPTION</i>	
value1	<i>0x0~0xFFFF FFFFU</i>
When value of parameter option is <i>HASHL_OPTION</i>	
value1	<i>0x0~0xFFFF FFFFU</i>
When value of parameter option is <i>FILTER_OPTION</i>	
value1	<i>ENET_SRC_FILTER_NORMAL_ENABLE /</i> <i>ENET_SRC_FILTER_INVERSE_ENABLE / ENET_SRC_FILTER_DISABLE</i>
value2	<i>ENET_DEST_FILTER_INVERSE_ENABLE /</i> <i>ENET_DEST_FILTER_INVERSE_DISABLE</i>
value3	<i>ENET_MULTICAST_FILTER_HASH_OR_PERFECT /</i> <i>ENET_MULTICAST_FILTER_HASH /</i> <i>ENET_MULTICAST_FILTER_PERFECT /</i> <i>ENET_MULTICAST_FILTER_NONE</i>
value4	<i>ENET_UNICAST_FILTER_EITHER / ENET_UNICAST_FILTER_HASH /</i> <i>ENET_UNICAST_FILTER_PERFECT</i>
value5	<i>ENET_PCFRM_PREVENT_ALL /</i> <i>ENET_PCFRM_PREVENT_PAUSEFRAME /</i> <i>ENET_PCFRM_FORWARD_ALL / ENET_PCFRM_FORWARD_FILTERED</i>
When value of parameter option is <i>HALFDUPLEX_OPTION</i>	
value1	<i>ENET_CARRIERSENSE_ENABLE / ENET_CARRIERSENSE_DISABLE</i>
value2	<i>ENET_RECEIVEOWN_ENABLE / ENET_RECEIVEOWN_DISABLE</i>
value3	<i>ENET_RETRYTRANSMISSION_ENABLE /</i> <i>ENET_RETRYTRANSMISSION_DISABLE</i>
value4	<i>ENET_BACKOFFLIMIT_10 / ENET_BACKOFFLIMIT_8 /</i> <i>ENET_BACKOFFLIMIT_4 / ENET_BACKOFFLIMIT_1</i>

value5	<i>ENET_DEFERRALCHECK_ENABLE / ENET_DEFERRALCHECK_DISABLE</i>
When value of parameter option is <i>TIMER_OPTION</i>	
value1	<i>ENET_WATCHDOG_ENABLE / ENET_WATCHDOG_DISABLE</i>
value2	<i>ENET_JABBER_ENABLE / ENET_JABBER_DISABLE</i>
When value of parameter option is <i>INTERFRAMEGAP_OPTION</i>	
value1	<i>ENET_INTERFRAMEGAP_96BIT / ENET_INTERFRAMEGAP_88BIT / ENET_INTERFRAMEGAP_80BIT / ENET_INTERFRAMEGAP_72BIT / ENET_INTERFRAMEGAP_64BIT / ENET_INTERFRAMEGAP_56BIT / ENET_INTERFRAMEGAP_48BIT / ENET_INTERFRAMEGAP_40BIT</i>
Output parameter{out}	
-	-
Return value	
-	-

Example:

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

```
enet_initpara_reset();
```

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

enet_init

The description of enet_init is shown as below:

Table 3-291. Function enet_init

Function name	enet_init
Function prototype	ErrStatus enet_init(enet_mediamode_enum mediamode, enet_chksumconf_enum checksum, enet_frmrecept_enum recept);
Function descriptions	initialize ENET peripheral with generally concerned parameters and the less cared parameters
Precondition	enet_deinit ()
The called functions	enet_phy_config /enet_phy_write_read
Input parameter{in}	
mediamode	PHY mode and mac loopback configurations, only one parameter can be selected
<i>ENET_AUTO_NEGOTIATION</i>	PHY auto negotiation
<i>ENET_100M_FULLDUPLEX</i>	100Mbit/s, full-duplex
<i>ENET_100M_HALFDUPLEX</i>	100Mbit/s, half-duplex
<i>ENET_10M_FULLDUP</i>	10Mbit/s, full-duplex

<i>LEX</i>	
<i>ENET_10M_HALFDUPLEX</i>	10Mbit/s, half-duplex
<i>ENET_LOOPBACKMODE</i>	MAC in loopback mode at the MII
Input parameter{in}	
checksum	IP frame checksum offload function, only one parameter can be selected
<i>ENET_NO_AUTOCHECKSUM</i>	disable IP frame checksum function
<i>ENET_AUTOCHECKSUM_DROP_FAILFRAMES</i>	enable IP frame checksum function
<i>ENET_AUTOCHECKSUM_ACCEPT_FAILFRAMES</i>	enable IP frame checksum function, and the received frame with only payload error but no other errors will not be dropped
Input parameter{in}	
recept	frame filter function, only one parameter can be selected
<i>ENET_PROMISCUOUS_MODE</i>	promiscuous mode enabled
<i>ENET_RECEIVEALL</i>	all received frame are forwarded to application
<i>ENET_BROADCAST_FRAMES_PASS</i>	the address filters pass all received broadcast frames
<i>ENET_BROADCAST_FRAMES_DROP</i>	the address filters filter all incoming broadcast frames
Output parameter{out}	
-	-
Return value	
ErrStatus	ERROR or SUCCESS

Example:

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

```
ErrStatus enet_init_status;
```

```
enet_init_status = enet_init(ENET_AUTO_NEGOTIATION, ENET_AUTOCHECKSUM_DROP_FAILFRAMES, ENET_BROADCAST_FRAMES_PASS);
```

enet_software_reset

The description of enet_software_reset is shown as below:

Table 3-292. Function enet_software_reset

Function name	enet_software_reset
Function prototype	ErrStatus enet_software_reset(void);
Function descriptions	reset all core internal registers located in CLK_TX and CLK_RX

Precondition	-
The called functions	-
Input parameter{in}	
-	-
Output parameter{out}	
-	-
Return value	
ErrStatus	ERROR or SUCCESS

Example:

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

```
ErrStatus reval_state = ERROR;
```

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

enet_rxframe_size_get

The description of enet_rxframe_size_get is shown as below:

Table 3-293. Function enet_rxframe_size_get

Function name	enet_rxframe_size_get
Function prototype	uint32_t enet_rxframe_size_get(void);
Function descriptions	check receive frame valid and return frame size
Precondition	-
The called functions	enet_rxframe_drop()
Input parameter{in}	
-	-
Output parameter{out}	
-	-
Return value	
uint32_t	size of received frame 0x0 - 0x3FFF

Example:

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

```
uint32_t reval;
```

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

enet_descriptors_chain_init

The description of enet_descriptors_chain_init is shown as below:

Table 3-294. Function enet_descriptors_chain_init

Function name	enet_descriptors_chain_init
Function prototype	void enet_descriptors_chain_init(enet_dmadirection_enum direction);

Function descriptions	initialize the DMA Tx/Rx descriptors's parameters in chain mode
Precondition	-
The called functions	-
Input parameter{in}	
direction	the descriptors which users want to init only one parameter can be selected which is shown as below
<i>ENET_DMA_TX</i>	DMA Tx descriptors
<i>ENET_DMA_RX</i>	DMA Rx descriptors
Output parameter{out}	
-	-
Return value	
-	-

Example:

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

```
enet_descriptors_chain_init(ENET_DMA_TX);
```

enet_descriptors_ring_init

The description of enet_descriptors_ring_init is shown as below:

Table 3-295. Function enet_descriptors_ring_init

Function name	enet_descriptors_ring_init
Function prototype	void enet_descriptors_ring_init(enet_dmadirection_enum direction);
Function descriptions	initialize the DMA Tx/Rx descriptors's parameters in ring mode
Precondition	-
The called functions	-
Input parameter{in}	
direction	the descriptors which users want to init only one parameter can be selected which is shown as below
<i>ENET_DMA_TX</i>	DMA Tx descriptors
<i>ENET_DMA_RX</i>	DMA Rx descriptors
Output parameter{out}	
-	-
Return value	
-	-

Example:

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

```
enet_descriptors_ring_init(ENET_DMA_TX);
```

enet_frame_receive

The description of enet_frame_receive is shown as below:

Table 3-296. Function `enet_frame_receive`

Function name	<code>enet_frame_receive</code>
Function prototype	<code>ErrStatus enet_frame_receive(uint8_t *buffer, uint32_t bufsize);</code>
Function descriptions	handle current received frame data to application buffer
Precondition	-
The called functions	-
Input parameter{in}	
bufsize	the size of buffer which is the parameter in function, (0 -- 1524)
Output parameter{out}	
buffer	pointer to the received frame data if the input is NULL, user should copy data in application by himself
Return value	
ErrStatus	ERROR or SUCCESS

Example:

```

/* transfer received frame data to application buffer */

uint8_t data_buffer[1500];

uint32_t data_size;

enet_frame_receive(data_buffer, &data_size);

```

`enet_frame_transmit`

The description of `enet_frame_transmit` is shown as below:

Table 3-297. Function `enet_frame_transmit`

Function name	<code>enet_frame_transmit</code>
Function prototype	<code>ErrStatus enet_frame_transmit(uint8_t *buffer, uint32_t length);</code>
Function descriptions	handle application buffer data to transmit it
Precondition	-
The called functions	-
Input parameter{in}	
buffer	pointer to the frame data to be transmitted if the input is NULL, user should handle the data in application by himself
Input parameter{in}	
length	the length of frame data to be transmitted, (0 -- 1524)
Output parameter{out}	
-	-
Return value	
ErrStatus	ERROR or SUCCESS

Example:

```

/* transfer buffer data of application */

```

```
uint8_t data_buffer[1500];

uint32_t data_size = 800;

enet_frame_transmit (data_buffer, data_size);
```

enet_transmit_checksum_config

The description of enet_transmit_checksum_config is shown as below:

Table 3-298. Function enet_transmit_checksum_config

Function name	enet_transmit_checksum_config
Function prototype	void enet_transmit_checksum_config(enet_descriptors_struct *desc, uint32_t checksum);
Function descriptions	configure the transmit IP frame checksum offload calculation and insertion
Precondition	-
The called functions	-
Input parameter{in}	
desc	the descriptor pointer which users want to configure, the structure members can refer to Structure enet_descriptors_struct
Input parameter{in}	
checksum	IP frame checksum configuration only one parameter can be selected which is shown as below
ENET_CHECKSUM_DISABLE	checksum insertion disabled
ENET_CHECKSUM_IPV4HEADER	only IP header checksum calculation and insertion are enabled
ENET_CHECKSUM_TCPUDPICMP_SEGMENT	TCP/UDP/ICMP checksum insertion calculated but pseudo-header
ENET_CHECKSUM_TCPUDPICMP_FULL	TCP/UDP/ICMP checksum insertion fully calculated
Output parameter{out}	
Return value	

Example:

```
/* configure the transmit IP frame checksum offload calculation and insertion */

enet_descriptors_struct rx_desc;

enet_transmit_checksum_config(rx_desc, ENET_CHECKSUM_TCPUDPICMP_FULL);
```

enet_enable

The description of enet_enable is shown as below:

Table 3-299. Function enet_enable

Function name	enet_enable
Function prototype	void enet_enable(void);
Function descriptions	ENET Tx and Rx function enable (include MAC and DMA module)
Precondition	-
The called functions	enet_tx_enable() /enet_rx_enable()
Input parameter{in}	
-	-
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* enable the ENET */
```

```
enet_enable();
```

enet_disable

The description of enet_disable is shown as below:

Table 3-300. Function enet_disable

Function name	enet_disable
Function prototype	void enet_disable(void);
Function descriptions	ENET Tx and Rx function disable (include MAC and DMA module)
Precondition	-
The called functions	enet_tx_disable() /enet_rx_disable()
Input parameter{in}	
-	-
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* disable the ENET */
```

```
enet_disable();
```

enet_mac_address_set

The description of enet_mac_address_set is shown as below:

Table 3-301. Function enet_mac_address_set

Function name	enet_mac_address_set
----------------------	----------------------

Function prototype	void enet_mac_address_set(enet_macaddress_enum mac_addr, uint8_t paddr[]);
Function descriptions	configure MAC address
Precondition	-
The called functions	-
Input parameter{in}	
mac_addr	select which MAC address will be set only one parameter can be selected which is shown as below
ENET_MAC_ADDRES S0	set MAC address 0 filter
ENET_MAC_ADDRES S1	set MAC address 1 filter
ENET_MAC_ADDRES S2	set MAC address 2 filter
ENET_MAC_ADDRES S3	set MAC address 3 filter
Input parameter{in}	
paddr	the buffer pointer which stores the MAC address little-ending store, such as MAC address is aa:bb:cc:dd:ee:22, the buffer is {22, ee, dd, cc, bb, aa}
Output parameter{out}	
-	-
Return value	
-	-

Example:

```

/* config mac address */
netif->hwaddr[0] = 0x02;
netif->hwaddr[1] = 0xaa;
netif->hwaddr[2] = 0xbb;
netif->hwaddr[3] = 0xcc;
netif->hwaddr[4] = 0xdd;
netif->hwaddr[5] = 0xee;

enet_mac_address_set(ENET_MAC_ADDRESS0, netif->hwaddr);

```

enet_mac_address_get

The description of enet_mac_address_get is shown as below:

Table 3-302. Function enet_mac_address_get

Function name	enet_mac_address_get
----------------------	----------------------

Function prototype	void enet_mac_address_get(enet_macaddress_enum mac_addr, uint8_t paddr[]);
Function descriptions	get MAC address
Precondition	-
The called functions	-
Input parameter{in}	
mac_addr	select which MAC address will be set only one parameter can be selected which is shown as below
ENET_MAC_ADDRES S0	set MAC address 0 filter
ENET_MAC_ADDRES S1	set MAC address 1 filter
ENET_MAC_ADDRES S2	set MAC address 2 filter
ENET_MAC_ADDRES S3	set MAC address 3 filter
Output parameter{out}	
paddr	the buffer pointer which stores the MAC address little-ending store, such as MAC address is aa:bb:cc:dd:ee:22, the buffer is {22, ee, dd, cc, bb, aa}
Return value	
-	-

Example:

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

```
enet_mac_address_get (ENET_MAC_ADDRESS0, netif->hwaddr);
```

enet_flag_get

The description of enet_flag_get is shown as below:

Table 3-303. Function enet_flag_get

Function name	enet_flag_get
Function prototype	FlagStatus enet_flag_get(enet_flag_enum enet_flag);
Function descriptions	get the ENET MAC/MSC/PTP/DMA status flag
Precondition	-
The called functions	-
Input parameter{in}	
enet_flag	ENET status flag, refer to enet_flag_enum only one parameter can be selected which is shown as below
ENET_MAC_FLAG_MP KR	magic packet received flag
ENET_MAC_FLAG_W UFR	wakeup frame received flag

<i>ENET_MAC_FLAG_FLOWCONTROL</i>	flow control status flag
<i>ENET_MAC_FLAG_WUM</i>	WUM status flag
<i>ENET_MAC_FLAG_MSC</i>	MSC status flag
<i>ENET_MAC_FLAG_MSC_CR</i>	MSC receive status flag
<i>ENET_MAC_FLAG_MSC_CT</i>	MSC transmit status flag
<i>ENET_MAC_FLAG_TMST</i>	time stamp trigger status flag
<i>ENET_PTP_FLAG_TSCO</i>	timestamp second counter overflow flag
<i>ENET_PTP_FLAG_TTM</i>	target time match flag
<i>ENET_MSC_FLAG_RFC</i>	received frames CRC error flag
<i>ENET_MSC_FLAG_RFAE</i>	received frames alignment error flag
<i>ENET_MSC_FLAG_RGUF</i>	received good unicast frames flag
<i>ENET_MSC_FLAG_TGFSC</i>	transmitted good frames single collision flag
<i>ENET_MSC_FLAG_TGFMSC</i>	transmitted good frames more single collision flag
<i>ENET_MSC_FLAG_TGF</i>	transmitted good frames flag
<i>ENET_DMA_FLAG_TS</i>	transmit status flag
<i>ENET_DMA_FLAG_TPS</i>	transmit process stopped status flag
<i>ENET_DMA_FLAG_TB</i>	transmit buffer unavailable status flag
<i>ENET_DMA_FLAG_TJT</i>	transmit jabber timeout status flag
<i>ENET_DMA_FLAG_RO</i>	receive overflow status flag
<i>ENET_DMA_FLAG_TU</i>	transmit underflow status flag
<i>ENET_DMA_FLAG_RS</i>	receive status flag
<i>ENET_DMA_FLAG_RBU</i>	receive buffer unavailable status flag
<i>ENET_DMA_FLAG_RPS</i>	receive process stopped status flag
<i>ENET_DMA_FLAG_R</i>	receive watchdog timeout status flag

<i>WT</i>	
<i>ENET_DMA_FLAG_ET</i>	early transmit status flag
<i>ENET_DMA_FLAG_FBE</i>	fatal bus error status flag
<i>ENET_DMA_FLAG_ER</i>	early receive status flag
<i>ENET_DMA_FLAG_AI</i>	abnormal interrupt summary flag
<i>ENET_DMA_FLAG_NI</i>	normal interrupt summary flag
<i>ENET_DMA_FLAG_EB_DMA_ERROR</i>	DMA error flag
<i>ENET_DMA_FLAG_EB_TRANSFER_ERROR</i>	transfer error flag
<i>ENET_DMA_FLAG_EB_ACCESS_ERROR</i>	access error flag
<i>ENET_DMA_FLAG_MSC</i>	MSC status flag
<i>ENET_DMA_FLAG_WUM</i>	WUM status flag
<i>ENET_DMA_FLAG_TST</i>	timestamp trigger status flag
Output parameter{out}	
-	-
Return value	
FlagStatus	SET or RESET

Example:

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

```
enet_flag_get (ENET_DMA_FLAG_RS);
```

enet_flag_clear

The description of enet_flag_clear is shown as below:

Table 3-304. Function enet_flag_clear

Function name	enet_flag_clear
Function prototype	void enet_flag_clear(enet_flag_clear_enum enet_flag);
Function descriptions	clear the ENET DMA status flag
Precondition	-
The called functions	-
Input parameter{in}	
enet_flag	ENET DMA flag clear, refer to enet_flag_clear_enum only one parameter can be selected which is shown as below
<i>ENET_DMA_FLAG_TST_CLR</i>	transmit status flag clear
<i>ENET_DMA_FLAG_TP</i>	transmit process stopped status flag clear

<i>S_CLR</i>	
<i>ENET_DMA_FLAG_TB U_CLR</i>	transmit buffer unavailable status flag clear
<i>ENET_DMA_FLAG_TJ T_CLR</i>	transmit jabber timeout status flag clear
<i>ENET_DMA_FLAG_RO _CLR</i>	receive overflow status flag clear
<i>ENET_DMA_FLAG_TU _CLR</i>	transmit underflow status flag clear
<i>ENET_DMA_FLAG_RS _CLR</i>	receive status flag clear
<i>ENET_DMA_FLAG_RB U_CLR</i>	receive buffer unavailable status flag clear
<i>ENET_DMA_FLAG_RP S_CLR</i>	receive process stopped status flag clear
<i>ENET_DMA_FLAG_R WT_CLR</i>	receive watchdog timeout status flag clear
<i>ENET_DMA_FLAG_ET _CLR</i>	early transmit status flag clear
<i>ENET_DMA_FLAG_FB E_CLR</i>	fatal bus error status flag clear
<i>ENET_DMA_FLAG_ER _CLR</i>	early receive status flag clear
<i>ENET_DMA_FLAG_AI CLR</i>	abnormal interrupt summary flag clear
<i>ENET_DMA_FLAG_NI _CLR</i>	normal interrupt summary flag clear
Output parameter{out}	
-	-
Return value	
-	-

Example:

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

```
enet_flag_clear(ENET_DMA_FLAG_RS_CLR);
```

enet_interrupt_enable

The description of enet_interrupt_enable is shown as below:

Table 3-305. Function enet_interrupt_enable

Function name	enet_interrupt_enable
Function prototype	void enet_interrupt_enable(enet_int_enum enet_int);
Function descriptions	enable ENET MAC/MSD/DMA interrupt

Precondition	-
The called functions	-
Input parameter{in}	
enet_int	ENET interrupt only one parameter can be selected which is shown as below
<i>ENET_MAC_INT_WUMIM</i>	WUM interrupt mask
<i>ENET_MAC_INT_TMS</i> <i>TIM</i>	timestamp trigger interrupt mask
<i>ENET_MSC_INT_RFC</i> <i>EIM</i>	received frame CRC error interrupt mask
<i>ENET_MSC_INT_RFA</i> <i>EIM</i>	received frames alignment error interrupt mask
<i>ENET_MSC_INT_RGU</i> <i>FIM</i>	received good unicast frames interrupt mask
<i>ENET_MSC_INT_TGF</i> <i>SCIM</i>	transmitted good frames single collision interrupt mask
<i>ENET_MSC_INT_TGF</i> <i>MSCIM</i>	transmitted good frames more single collision interrupt mask
<i>ENET_MSC_INT_TGFI</i> <i>M</i>	transmitted good frames interrupt mask
<i>ENET_DMA_INT_TIE</i>	transmit interrupt enable
<i>ENET_DMA_INT_TPSI</i> <i>E</i>	transmit process stopped interrupt enable
<i>ENET_DMA_INT_TBUI</i> <i>E</i>	transmit buffer unavailable interrupt enable
<i>ENET_DMA_INT_TJTI</i> <i>E</i>	transmit jabber timeout interrupt enable
<i>ENET_DMA_INT_ROIE</i>	receive overflow interrupt enable
<i>ENET_DMA_INT_TUIE</i>	transmit underflow interrupt enable
<i>ENET_DMA_INT_RIE</i>	receive interrupt enable
<i>ENET_DMA_INT_RBUI</i> <i>E</i>	receive buffer unavailable interrupt enable
<i>ENET_DMA_INT_RPSI</i> <i>E</i>	receive process stopped interrupt enable
<i>ENET_DMA_INT_RWT</i> <i>IE</i>	receive watchdog timeout interrupt enable
<i>ENET_DMA_INT_ETIE</i>	early transmit interrupt enable
<i>ENET_DMA_INT_FBEI</i> <i>E</i>	fatal bus error interrupt enable
<i>ENET_DMA_INT_ERIE</i>	early receive interrupt enable
<i>ENET_DMA_INT_AIE</i>	abnormal interrupt summary enable
<i>ENET_DMA_INT_NIE</i>	normal interrupt summary enable

Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* enable normal interrupt summary */
enet_interrupt_enable(ENET_DMA_INT_NIE);
```

enet_interrupt_disable

The description of enet_interrupt_disable is shown as below:

Table 3-306. Function enet_interrupt_disable

Function name	enet_interrupt_disable
Function prototype	void enet_interrupt_disable(enet_int_enum enet_int);
Function descriptions	disable ENET MAC/MSD/DMA interrupt
Precondition	-
The called functions	-
Input parameter{in}	
enet_int	ENET interrupt only one parameter can be selected which is shown as below
ENET_MAC_INT_WUM IM	WUM interrupt mask
ENET_MAC_INT_TMS TIM	timestamp trigger interrupt mask
ENET_MSC_INT_RFC EIM	received frame CRC error interrupt mask
ENET_MSC_INT_RFA EIM	received frames alignment error interrupt mask
ENET_MSC_INT_RGU FIM	received good unicast frames interrupt mask
ENET_MSC_INT_TGF SCIM	transmitted good frames single collision interrupt mask
ENET_MSC_INT_TGF MSCIM	transmitted good frames more single collision interrupt mask
ENET_MSC_INT_TGFI M	transmitted good frames interrupt mask
ENET_DMA_INT_TIE	transmit interrupt enable
ENET_DMA_INT_TPSI E	transmit process stopped interrupt enable
ENET_DMA_INT_TBUI E	transmit buffer unavailable interrupt enable
ENET_DMA_INT_TJTI	transmit jabber timeout interrupt enable

<i>E</i>	
<i>ENET_DMA_INT_ROIE</i>	receive overflow interrupt enable
<i>ENET_DMA_INT_TUIE</i>	transmit underflow interrupt enable
<i>ENET_DMA_INT_RIE</i>	receive interrupt enable
<i>ENET_DMA_INT_RBUIE</i>	receive buffer unavailable interrupt enable
<i>E</i>	
<i>ENET_DMA_INT_RPSIE</i>	receive process stopped interrupt enable
<i>E</i>	
<i>ENET_DMA_INT_RWTIE</i>	receive watchdog timeout interrupt enable
<i>IE</i>	
<i>ENET_DMA_INT_ETIE</i>	early transmit interrupt enable
<i>ENET_DMA_INT_FBEIE</i>	fatal bus error interrupt enable
<i>E</i>	
<i>ENET_DMA_INT_ERIE</i>	early receive interrupt enable
<i>ENET_DMA_INT_AIE</i>	abnormal interrupt summary enable
<i>ENET_DMA_INT_NIE</i>	normal interrupt summary enable
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* disable normal interrupt summary */
enet_interrupt_disable(ENET_DMA_INT_NIE);
```

enet_interrupt_flag_get

The description of enet_interrupt_flag_get is shown as below:

Table 3-307. Function enet_interrupt_flag_get

Function name	enet_interrupt_flag_get
Function prototype	FlagStatus enet_interrupt_flag_get(enet_int_flag_enum int_flag);
Function descriptions	get ENET MAC/MSD/DMA interrupt flag
Precondition	-
The called functions	-
Input parameter{in}	
int_flag	ENET interrupt flag only one parameter can be selected which is shown as below
<i>ENET_MAC_INT_FLAG_WUM</i>	WUM status flag
<i>ENET_MAC_INT_FLAG_MSC</i>	MSC status flag
<i>ENET_MAC_INT_FLAG_MSCR</i>	MSC receive status flag

ENET_MAC_INT_FLG G_MSCT	MSC transmit status flag
ENET_MAC_INT_FLG G_TMST	time stamp trigger status flag
ENET_MSC_INT_FLG G_RFCE	received frames CRC error flag
ENET_MSC_INT_FLG G_RFAE	received frames alignment error flag
ENET_MSC_INT_FLG G_RGUF	received good unicast frames flag
ENET_MSC_INT_FLG G_TGFSC	transmitted good frames single collision flag
ENET_MSC_INT_FLG G_TGFMSC	transmitted good frames more single collision flag
ENET_MSC_INT_FLG G_TGF	transmitted good frames flag
ENET_DMA_INT_FLG G_TS	transmit status flag
ENET_DMA_INT_FLG G_TPS	transmit process stopped status flag
ENET_DMA_INT_FLG G_TBU	transmit buffer unavailable status flag
ENET_DMA_INT_FLG G_TJT	transmit jabber timeout status flag
ENET_DMA_INT_FLG G_RO	receive overflow status flag
ENET_DMA_INT_FLG G_TU	transmit underflow status flag
ENET_DMA_INT_FLG G_RS	receive status flag
ENET_DMA_INT_FLG G_RBU	receive buffer unavailable status flag
ENET_DMA_INT_FLG G_RPS	receive process stopped status flag
ENET_DMA_INT_FLG G_RWT	receive watchdog timeout status flag
ENET_DMA_INT_FLG G_ET	early transmit status flag
ENET_DMA_INT_FLG G_FBE	fatal bus error status flag
ENET_DMA_INT_FLG G_ER	early receive status flag
ENET_DMA_INT_FLG	abnormal interrupt summary flag

<i>G_AI</i>	
<i>ENET_DMA_INT_FLG</i> <i>G_NI</i>	normal interrupt summary flag
<i>ENET_DMA_INT_FLG</i> <i>G_MSC</i>	MSC status flag
<i>ENET_DMA_INT_FLG</i> <i>G_WUM</i>	WUM status flag
<i>ENET_DMA_INT_FLG</i> <i>G_TST</i>	timestamp trigger status flag
Output parameter{out}	
-	-
Return value	
FlagStatus	SET or RESET

Example:

```
/* check whether the specified flag bit is set or not */
enet_interrupt_flag_get(ENET_DMA_INT_FLAG_RS);
```

enet_interrupt_flag_clear

The description of enet_interrupt_flag_clear is shown as below:

Table 3-308. Function enet_interrupt_flag_clear

Function name	enet_interrupt_flag_clear
Function prototype	void enet_interrupt_flag_clear(enet_int_flag_clear_enum int_flag_clear);
Function descriptions	clear ENET DMA interrupt flag
Precondition	-
The called functions	-
Input parameter{in}	
int_flag_clear	clear ENET interrupt flag only one parameter can be selected which is shown as below
<i>ENET_DMA_INT_FLG</i> <i>G_TS_CLR</i>	transmit status flag
<i>ENET_DMA_INT_FLG</i> <i>G_TPS_CLR</i>	transmit process stopped status flag
<i>ENET_DMA_INT_FLG</i> <i>G_TBU_CLR</i>	transmit buffer unavailable status flag
<i>ENET_DMA_INT_FLG</i>	transmit jabber timeout status flag

<i>G_TJT_CLR</i>	
<i>ENET_DMA_INT_FLG</i> <i>G_RO_CLR</i>	receive overflow status flag
<i>ENET_DMA_INT_FLG</i> <i>G_TU_CLR</i>	transmit underflow status flag
<i>ENET_DMA_INT_FLG</i> <i>G_RS_CLR</i>	receive status flag
<i>ENET_DMA_INT_FLG</i> <i>G_RBU_CLR</i>	receive buffer unavailable status flag
<i>ENET_DMA_INT_FLG</i> <i>G_RPS_CLR</i>	receive process stopped status flag
<i>ENET_DMA_INT_FLG</i> <i>G_RWT_CLR</i>	receive watchdog timeout status flag
<i>ENET_DMA_INT_FLG</i> <i>G_ET_CLR</i>	early transmit status flag
<i>ENET_DMA_INT_FLG</i> <i>G_FBE_CLR</i>	fatal bus error status flag
<i>ENET_DMA_INT_FLG</i> <i>G_ER_CLR</i>	early receive status flag
<i>ENET_DMA_INT_FLG</i> <i>G_AI_CLR</i>	abnormal interrupt summary flag
<i>ENET_DMA_INT_FLG</i> <i>G_NI_CLR</i>	normal interrupt summary flag
Output parameter{out}	
-	-
Return value	
-	-

Example:

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

```
enet_interrupt_flag_clear(ENET_DMA_INT_FLAG_RS);
```

enet_tx_enable

The description of enet_tx_enable is shown as below:

Table 3-309. Function enet_tx_enable

Function name	enet_tx_enable
Function prototype	void enet_tx_enable(void);
Function descriptions	ENET Tx function enable (include MAC and DMA module)
Precondition	-
The called functions	enet_txfifo_flush()
Input parameter{in}	
-	-
Output parameter{out}	
-	-
Return value	
-	-

Example:

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

```
enet_tx_enable();
```

enet_tx_disable

The description of enet_tx_disable is shown as below:

Table 3-310. Function enet_tx_disable

Function name	enet_tx_disable
Function prototype	void enet_tx_disable(void);
Function descriptions	ENET Tx function disable (include MAC and DMA module)
Precondition	-
The called functions	enet_txfifo_flush()
Input parameter{in}	
-	-
Output parameter{out}	
-	-
Return value	
-	-

Example:

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

```
enet_tx_disable();
```

enet_rx_enable

The description of enet_rx_enable is shown as below:

Table 3-311. Function enet_rx_enable

Function name	enet_rx_enable
----------------------	----------------

Function prototype	void enet_rx_enable(void);
Function descriptions	ENET Rx function enable (include MAC and DMA module)
Precondition	-
The called functions	-
Input parameter{in}	
-	-
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* enable reception function of MAC and DMA */
enet_rx_enable();
```

enet_rx_disable

The description of enet_rx_disable is shown as below:

Table 3-312. Function enet_rx_disable

Function name	enet_rx_disable
Function prototype	void enet_rx_disable(void);
Function descriptions	ENET Rx function disable (include MAC and DMA module)
Precondition	-
The called functions	-
Input parameter{in}	
-	-
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* disable reception function of MAC and DMA */
enet_rx_disable();
```

enet_registers_get

The description of enet_registers_get is shown as below:

Table 3-313. Function enet_registers_get

Function name	enet_registers_get
Function prototype	void enet_registers_get(enet_registers_type_enum type, uint32_t *preg, uint32_t num);

Function descriptions	put registers value into the application buffer
Precondition	-
The called functions	-
Input parameter{in}	
type	register type which will be get only one parameter can be selected which is shown as below
<i>ALL_MAC_REG</i>	get the registers within the offset scope range ENET_MAC_CFG to ENET_MAC_FCTH
<i>ALL_MSC_REG</i>	get the registers within the offset scope range ENET_MSC_CTL to ENET_MSC_RGUF CNT
<i>ALL_PTP_REG</i>	get the registers within the offset scope range ENET_PTP_TSCTL to ENET_PTP_PPSCTL
<i>ALL_DMA_REG</i>	get the registers within the offset scope range ENET_DMA_BCTL to ENET_DMA_CRBADDR
Input parameter{in}	
num	the number of registers that the user want to get, (0 -- 54)
Output parameter{out}	
preg	the application buffer pointer for storing the register value
Return value	
-	-

Example:

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

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

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

enet_debug_status_get

The description of enet_debug_status_get is shown as below:

Table 3-314. Function enet_debug_status_get

Function name	enet_debug_status_get
Function prototype	uint32_t enet_debug_status_get(uint32_t mac_debug);
Function descriptions	get the enet debug status from the debug register
Precondition	-
The called functions	--
Input parameter{in}	
mac_debug	enet debug status only one parameter can be selected which is shown as below
<i>ENET_MAC_RECEIVE R_NOT_IDLE</i>	MAC receiver is not in idle state
<i>ENET_RX_ASYNC HR_FIFO_STATE</i>	Rx asynchronous FIFO status

<i>ENET_RXFIFO_WRITING</i>	RxFIFO is doing write operation
<i>ENET_RXFIFO_READ_STATUS</i>	RxFIFO read operation status
<i>ENET_RXFIFO_STATE</i>	RxFIFO state
<i>ENET_MAC_TRANSMITTER_NOT_IDLE</i>	MAC transmitter is not in idle state
<i>ENET_MAC_TRANSMITTER_STATUS</i>	status of MAC transmitter
<i>ENET_PAUSE_CONDITION_STATUS</i>	pause condition status
<i>ENET_TXFIFO_READ_STATUS</i>	TxFIFO read operation status
<i>ENET_TXFIFO_WRITING</i>	TxFIFO is doing write operation
<i>ENET_TXFIFO_NOT_EMPTY</i>	TxFIFO is not empty
<i>ENET_TXFIFO_FULL</i>	TxFIFO is full
Output parameter{out}	
-	-
Return value	
uint32_t	value of the status users want to get

Example:

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

```
uint32_t debug_value;
```

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

enet_address_filter_enable

The description of enet_address_filter_enable is shown as below:

Table 3-315. Function enet_address_filter_enable

Function name	enet_address_filter_enable
Function prototype	void enet_address_filter_enable(enet_macaddress_enum mac_addr);
Function descriptions	enable the MAC address filter
Precondition	-
The called functions	--
Input parameter{in}	
mac_addr	select which MAC address will be enable only one parameter can be selected which is shown as below
<i>ENET_MAC_ADDRESS1</i>	enable MAC address 1 filter

ENET_MAC_ADDRES S2	enable MAC address 2 filter
ENET_MAC_ADDRES S3	enable MAC address 3 filter
Output parameter{out}	
-	-
Return value	
-	-

Example:

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

```
enet_address_filter_enable(ENET_MAC_ADDRESS1);
```

enet_address_filter_disable

The description of enet_address_filter_disable is shown as below:

Table 3-316. Function enet_address_filter_disable

Function name	enet_address_filter_disable
Function prototype	void enet_address_filter_disable(enet_macaddress_enum mac_addr);
Function descriptions	disable the MAC address filter
Precondition	-
The called functions	-
Input parameter{in}	
mac_addr	select which MAC address will be enable only one parameter can be selected which is shown as below
ENET_MAC_ADDRES S1	enable MAC address 1 filter
ENET_MAC_ADDRES S2	enable MAC address 2 filter
ENET_MAC_ADDRES S3	enable MAC address 3 filter
Output parameter{out}	
-	-
Return value	
-	-

Example:

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

```
enet_address_filter_disable(ENET_MAC_ADDRESS1);
```

enet_address_filter_config

The description of enet_address_filter_config is shown as below:

Table 3-317. Function `enet_address_filter_config`

Function name	<code>enet_address_filter_config</code>
Function prototype	<code>void enet_address_filter_config(enet_macaddress_enum mac_addr, uint32_t addr_mask, uint32_t filter_type);</code>
Function descriptions	configure the MAC address filter
Precondition	-
The called functions	-
Input parameter{in}	
mac_addr	select which MAC address will be enable only one parameter can be selected which is shown as below
<code>ENET_MAC_ADDRES S1</code>	enable MAC address 1 filter
<code>ENET_MAC_ADDRES S2</code>	enable MAC address 2 filter
<code>ENET_MAC_ADDRES S3</code>	enable MAC address 3 filter
Input parameter{in}	
addr_mask	select which MAC address bytes will be mask one or more parameters can be selected which are shown as below
<code>ENET_ADDRESS_MA SK_BYTE0</code>	mask <code>ENET_MAC_ADDR1L[7:0]</code> bits
<code>ENET_ADDRESS_MA SK_BYTE1</code>	mask <code>ENET_MAC_ADDR1L[15:8]</code> bits
<code>ENET_ADDRESS_MA SK_BYTE2</code>	mask <code>ENET_MAC_ADDR1L[23:16]</code> bits
<code>ENET_ADDRESS_MA SK_BYTE3</code>	mask <code>ENET_MAC_ADDR1L [31:24]</code> bits
<code>ENET_ADDRESS_MA SK_BYTE4</code>	mask <code>ENET_MAC_ADDR1H [7:0]</code> bits
<code>ENET_ADDRESS_MA SK_BYTE5</code>	mask <code>ENET_MAC_ADDR1H [15:8]</code> bits
Input parameter{in}	
filter_type	select which MAC address filter type will be selected only one parameter can be selected which is shown as below
<code>ENET_ADDRESS_FILT ER_SA</code>	The MAC address is used to compared with the SA field of the received frame
<code>ENET_ADDRESS_FILT ER_DA</code>	The MAC address is used to compared with the DA field of the received frame
Output parameter{out}	
-	-
Return value	
-	-

Example:

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

```
enet_address_filter_config(ENET_MAC_ADDRESS1, ENET_ADDRESS_MASK_BYTE0 |
ENET_ADDRESS_MASK_BYTE1 | ENET_ADDRESS_MASK_BYTE2, ENET_ADDRESS
_FILTER_DA);
```

enet_phy_config

The description of enet_phy_config is shown as below:

Table 3-318. Function enet_phy_config

Function name	enet_phy_config
Function prototype	ErrStatus enet_phy_config(void);
Function descriptions	PHY interface configuration (configure SMI clock and reset PHY chip)
Precondition	-
The called functions	rcu_clock_freq_get()/enet_phy_write_read()
Input parameter{in}	
-	-
Output parameter{out}	
-	-
Return value	
ErrStatus	ERROR or SUCCESS

Example:

```
/* config PHY interface */

enet_phy_config();
```

enet_phy_write_read

The description of enet_phy_write_read is shown as below:

Table 3-319. Function enet_phy_write_read

Function name	enet_phy_write_read
Function prototype	ErrStatus enet_phy_write_read(enet_phydirection_enum direction, uint16_t phy_address, uint16_t phy_reg, uint16_t *pvalue);
Function descriptions	write to / read from a PHY register
Precondition	-
The called functions	-
Input parameter{in}	
direction	only one parameter can be selected which is shown as below
<i>ENET_PHY_WRITE</i>	write data to phy register
<i>ENET_PHY_READ</i>	read data from phy register
Input parameter{in}	
phy_address	0x0 - 0x1F
Input parameter{in}	

phy_reg	0x0 - 0x1F
Input parameter{in}	
pvalue	the value will be written to the PHY register in ENET_PHY_WRITE direction
Output parameter{out}	
pvalue	the value will be read from the PHY register in ENET_PHY_READ direction
Return value	
ErrStatus	ERROR or SUCCESS

Example:

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

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

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

enet_phyloopback_enable

The description of enet_phyloopback_enable is shown as below:

Table 3-320. Function enet_phyloopback_enable

Function name	enet_phyloopback_enable
Function prototype	ErrStatus enet_phyloopback_enable(void);
Function descriptions	enable the loopback function of PHY chip
Precondition	-
The called functions	enet_phy_write_read()
Input parameter{in}	
-	-
Output parameter{out}	
-	-
Return value	
ErrStatus	ERROR or SUCCESS

Example:

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

```
ErrStatus phy_state = ERROR;
```

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

enet_phyloopback_disable

The description of enet_phyloopback_disable is shown as below:

Table 3-321. Function enet_phyloopback_disable

Function name	enet_phyloopback_disable
Function prototype	ErrStatus enet_phyloopback_disable(void);

Function descriptions	disable the loopback function of PHY chip
Precondition	-
The called functions	enet_phy_write_read
Input parameter{in}	
-	-
Output parameter{out}	
-	-
Return value	
ErrStatus	ERROR or SUCCESS

Example:

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

```
ErrStatus phy_state = ERROR;
```

```
phy_state = enet_phyloopback_disable();
```

enet_forward_feature_enable

The description of enet_forward_feature_enable is shown as below:

Table 3-322. Function enet_forward_feature_enable

Function name	enet_forward_feature_enable
Function prototype	void enet_forward_feature_enable(uint32_t feature);
Function descriptions	enable ENET forward feature
Precondition	-
The called functions	-
Input parameter{in}	
feature	the feature of ENET forward mode one or more parameters can be selected which are shown as below
ENET_AUTO_PADCRC_DROP	the function of the MAC strips the Pad/FCS field on received frames
ENET_TYPEFRAME_CRC_DROP	the function that FCS field(last 4 bytes) of frame will be dropped before forwarding
ENET_FORWARD_ERROR_FRAMES	the function that all frame received with error except runt error are forwarded to memory
ENET_FORWARD_UNDERSIZED_GOODFRAMES	the function that forwarding undersized good frames
Output parameter{out}	
-	-
Return value	
-	-

Example:

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

```
enet_forward_feature_enable(ENET_FORWARD_UNDEERSZ_GOODFRAMES);
```

enet_forward_feature_disable

The description of enet_forward_feature_disable is shown as below:

Table 3-323. Function enet_forward_feature_disable

Function name	enet_forward_feature_disable
Function prototype	void enet_forward_feature_disable(uint32_t feature);
Function descriptions	disable ENET forward feature
Precondition	-
The called functions	-
Input parameter{in}	
feature	the feature of ENET forward mode one or more parameters can be selected which are shown as below
ENET_AUTO_PADCRC_DROP	the function of the MAC strips the Pad/FCS field on received frames
ENET_TYPEFRAME_CRC_DROP	the function that FCS field(last 4 bytes) of frame will be dropped before forwarding
ENET_FORWARD_ERROR_FRAMES	the function that all frame received with error except runt error are forwarded to memory
ENET_FORWARD_UNDEERSZ_GOODFRAMES	the function that forwarding undersized good frames
Output parameter{out}	
-	-
Return value	
-	-

Example:

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

```
enet_forward_feature_enable(ENET_FORWARD_UNDEERSZ_GOODFRAMES);
```

enet_fliter_feature_enable

The description of enet_fliter_feature_enable is shown as below:

Table 3-324. Function enet_fliter_feature_enable

Function name	enet_fliter_feature_enable
Function prototype	void enet_fliter_feature_enable(uint32_t feature);
Function descriptions	enable ENET fliter feature
Precondition	-
The called functions	-

Input parameter{in}	
feature	the feature of ENET fliter mode one or more parameters can be selected which are shown as below
<i>ENET_SRC_FILTER</i>	filter source address function
<i>ENET_SRC_FILTER_INVERSE</i>	inverse source address filtering result function
<i>ENET_DEST_FILTER_INVERSE</i>	inverse DA filtering result function
<i>ENET_MULTICAST_FILTER_PASS</i>	pass all multicast frames function
<i>ENET_MULTICAST_FILTER_HASH_MODE</i>	HASH multicast filter function
<i>ENET_UNICAST_FILTER_HASH_MODE</i>	HASH unicast filter function
<i>ENET_FILTER_MODE_EITHER</i>	HASH or perfect filter function
Output parameter{out}	
-	-
Return value	
-	-

Example:

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

```
enet_fliter_feature_enable(ENET_SRC_FILTER);
```

enet_fliter_feature_disable

The description of enet_fliter_feature_disable is shown as below:

Table 3-325. Function enet_fliter_feature_disable

Function name	enet_fliter_feature_disable
Function prototype	void enet_fliter_feature_disable(uint32_t feature);
Function descriptions	disable ENET fliter feature
Precondition	-
The called functions	-
Input parameter{in}	
feature	the feature of ENET fliter mode one or more parameters can be selected which are shown as below
<i>ENET_SRC_FILTER</i>	filter source address function
<i>ENET_SRC_FILTER_INVERSE</i>	inverse source address filtering result function
<i>ENET_DEST_FILTER_INVERSE</i>	inverse DA filtering result function
<i>ENET_MULTICAST_FILTER_PASS</i>	pass all multicast frames function

<i>LTER_PASS</i>	
<i>ENET_MULTICAST_FILTER_HASH_MODE</i>	HASH multicast filter function
<i>ENET_UNICAST_FILTER_HASH_MODE</i>	HASH unicast filter function
<i>ENET_FILTER_MODE_EITHER</i>	HASH or perfect filter function
Output parameter{out}	
-	-
Return value	
-	-

Example:

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

```
enet_filter_feature_enable(ENET_SRC_FILTER);
```

enet_pauseframe_generate

The description of enet_pauseframe_generate is shown as below:

Table 3-326. Function enet_pauseframe_generate

Function name	enet_pauseframe_generate
Function prototype	ErrStatus enet_pauseframe_generate(void);
Function descriptions	generate the pause frame, ENET will send pause frame after enable transmit flow control this function only use in full-duplex mode
Precondition	-
The called functions	-
Input parameter{in}	
-	-
Output parameter{out}	
-	-
Return value	
ErrStatus	ERROR or SUCCESS

Example:

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

```
ErrStatus reval;
```

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

enet_pauseframe_detect_config

The description of enet_pauseframe_detect_config is shown as below:

Table 3-327. Function `enet_pauseframe_detect_config`

Function name	<code>enet_pauseframe_detect_config</code>
Function prototype	<code>void enet_pauseframe_detect_config(uint32_t detect);</code>
Function descriptions	configure the pause frame detect type
Precondition	-
The called functions	-
Input parameter{in}	
detect	pause frame detect type only one parameter can be selected which is shown as below
<code>ENET_MAC0_AND_UNIQUE_ADDRESS_PAUSEDETECT</code>	besides the unique multicast address, MAC can also use the MAC0 address to detecting pause frame
<code>ENET_UNIQUE_PAUSEDETECT</code>	only the unique multicast address for pause frame which is specified in IEEE802.3 can be detected
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* config the pause frame detect type as ENET_UNIQUE_PAUSEDETECT */
```

```
enet_pauseframe_detect_config(ENET_UNIQUE_PAUSEDETECT);
```

enet_pauseframe_config

The description of `enet_pauseframe_config` is shown as below:

Table 3-328. Function `enet_pauseframe_config`

Function name	<code>enet_pauseframe_config</code>
Function prototype	<code>void enet_pauseframe_config(uint32_t pausetime, uint32_t pause_threshold);</code>
Function descriptions	configure the pause frame parameters
Precondition	-
The called functions	-
Input parameter{in}	
pausetime	pause time in transmit pause control frame, (0 – 0xFFFF)
Input parameter{in}	
pause_threshold	the threshold of the pause timer for retransmitting frames automatically, this value must make sure to be less than configured pause time, only one parameter can be selected which is shown as below
<code>ENET_PAUSETIME_MINUS4</code>	pause time minus 4 slot times
<code>ENET_PAUSETIME_MINUS28</code>	pause time minus 28 slot times

<i>ENET_PAUSETIME_MINUS144</i>	pause time minus 144 slot times
<i>ENET_PAUSETIME_MINUS256</i>	pause time minus 256 slot times
Output parameter{out}	
-	-
Return value	
-	-

Example:

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

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

enet_flowcontrol_threshold_config

The description of enet_flowcontrol_threshold_config is shown as below:

Table 3-329. Function enet_flowcontrol_threshold_config

Function name	enet_flowcontrol_threshold_config
Function prototype	void enet_flowcontrol_threshold_config(uint32_t deactive, uint32_t active);
Function descriptions	configure the threshold of the flow control(deactive and active threshold)
Precondition	-
The called functions	-
Input parameter{in}	
deactive	the threshold of the deactive flow control, this value should always be less than active flow control value, only one parameter can be selected which is shown as below
<i>ENET_DEACTIVE_THRESHOLD_256BYTES</i>	threshold level is 256 bytes
<i>ENET_DEACTIVE_THRESHOLD_512BYTES</i>	threshold level is 512 bytes
<i>ENET_DEACTIVE_THRESHOLD_768BYTES</i>	threshold level is 768 bytes
<i>ENET_DEACTIVE_THRESHOLD_1024BYTES</i>	threshold level is 1024 bytes
<i>ENET_DEACTIVE_THRESHOLD_1280BYTES</i>	threshold level is 1280 bytes
<i>ENET_DEACTIVE_THRESHOLD_1536BYTES</i>	threshold level is 1536 bytes
<i>ENET_DEACTIVE_THRESHOLD_1792BYTES</i>	threshold level is 1792 bytes

S	
Input parameter{in}	
active	the threshold of the active flow control, only one parameter can be selected which is shown as below
<i>ENET_ACTIVE_THRE SHOLD_256BYTES</i>	threshold level is 256 bytes
<i>ENET_ACTIVE_THRE SHOLD_512BYTES</i>	threshold level is 512 bytes
<i>ENET_ACTIVE_THRE SHOLD_768BYTES</i>	threshold level is 768 bytes
<i>ENET_ACTIVE_THRE SHOLD_1024BYTES</i>	threshold level is 1024 bytes
<i>ENET_ACTIVE_THRE SHOLD_1280BYTES</i>	threshold level is 1280 bytes
<i>ENET_ACTIVE_THRE SHOLD_1536BYTES</i>	threshold level is 1536 bytes
<i>ENET_ACTIVE_THRE SHOLD_1792BYTES</i>	threshold level is 1792 bytes
Output parameter{out}	
-	-
Return value	
-	-

Example:

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

```
enet_flowcontrol_threshold_config(ENET_DEACTIVE_THRESHOLD_256BYTES, ENET_A  
CTIVE_THRESHOLD_256BYTES);
```

enet_flowcontrol_feature_enable

The description of enet_flowcontrol_feature_enable is shown as below:

Table 3-330. Function enet_flowcontrol_feature_enable

Function name	enet_flowcontrol_feature_enable
Function prototype	void enet_flowcontrol_feature_enable(uint32_t feature);
Function descriptions	enable ENET flow control feature
Precondition	-
The called functions	-
Input parameter{in}	
feature	the feature of ENET flow control mode one or more parameters can be selected which are shown as below
<i>ENET_ZERO_QUANT A_PAUSE</i>	the automatic zero-quanta generation function
<i>ENET_TX_FLOWCON</i>	the flow control operation in the MAC

<i>TROL</i>	
<i>ENET_RX_FLOWCON TROL</i>	decoding function for the received pause frame and process it
<i>ENET_BACK_PRESSU RE</i>	back pressure operation in the MAC(only use in half-duplex mode)
Output parameter{out}	
-	-
Return value	
-	-

Example:

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

```
enet_flowcontrol_feature_enable(ENET_ZERO_QUANTA_PAUSE);
```

enet_flowcontrol_feature_disable

The description of enet_flowcontrol_feature_disable is shown as below:

Table 3-331. Function enet_flowcontrol_feature_disable

Function name	enet_flowcontrol_feature_disable
Function prototype	void enet_flowcontrol_feature_disable(uint32_t feature);
Function descriptions	disable ENET flow control feature
Precondition	-
The called functions	-
Input parameter{in}	
feature	the feature of ENET flow control mode one or more parameters can be selected which are shown as below
<i>ENET_ZERO_QUANT A_PAUSE</i>	the automatic zero-quanta generation function
<i>ENET_TX_FLOWCON TROL</i>	the flow control operation in the MAC
<i>ENET_RX_FLOWCON TROL</i>	decoding function for the received pause frame and process it
<i>ENET_BACK_PRESSU RE</i>	back pressure operation in the MAC(only use in half-duplex mode)
Output parameter{out}	
-	-
Return value	
-	-

Example:

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

```
enet_flowcontrol_feature_disable(ENET_ZERO_QUANTA_PAUSE);
```

enet_dmaprocess_state_get

The description of enet_dmaprocess_state_get is shown as below:

Table 3-332. Function enet_dmaprocess_state_get

Function name	enet_dmaprocess_state_get
Function prototype	uint32_t enet_dmaprocess_state_get(enet_dmadirection_enum direction);
Function descriptions	get the dma transmit/receive process state
Precondition	-
The called functions	-
Input parameter{in}	
direction	choose the direction of DMA process which users want to resume only one parameter can be selected which is shown as below
ENET_DMA_TX	DMA transmit process
ENET_DMA_RX	DMA receive process
Output parameter{out}	
-	-
Return value	
uint32_t	state of dma process, the value range shows below: ENET_RX_STATE_STOPPED / ENET_RX_STATE_FETCHING / ENET_RX_STATE_WAITING / ENET_RX_STATE_SUSPENDED / ENET_RX_STATE_CLOSING / ENET_RX_STATE_QUEUEING / ENET_TX_STATE_STOPPED / ENET_TX_STATE_FETCHING / ENET_TX_STATE_WAITING / ENET_TX_STATE_READING / ENET_TX_STATE_SUSPENDED / ENET_TX_STATE_CLOSING

Example:

```

/* get the dma receive process state */

uint32_t reval;

reval = enet_dmaprocess_state_get(ENET_DMA_RX);

if(ENET_RX_STATE_SUSPENDED == reval){

    do...

}

```

enet_dmaprocess_resume

The description of enet_dmaprocess_resume is shown as below:

Table 3-333. Function enet_dmaprocess_resume

Function name	enet_dmaprocess_resume
Function prototype	void enet_dmaprocess_resume(enet_dmadirection_enum direction);
Function descriptions	poll the DMA transmission/reception enable by writing any value to the ENET_DMA_TPEN/ENET_DMA_RPEN register, this will make the DMA to

	resume transmission/reception
Precondition	-
The called functions	-
Input parameter{in}	
direction	choose the direction of DMA process which users want to resume only one parameter can be selected which is shown as below
<i>ENET_DMA_TX</i>	DMA transmit process
<i>ENET_DMA_RX</i>	DMA receive process
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* enable DMA receive process */
```

```
enet_dmaprocess_resume(ENET_DMA_RX);
```

enet_rxprocess_check_recovery

The description of enet_rxprocess_check_recovery is shown as below:

Table 3-334. Function enet_rxprocess_check_recovery

Function name	enet_rxprocess_check_recovery
Function prototype	void enet_rxprocess_check_recovery(void);
Function descriptions	check and recover the Rx process
Precondition	-
The called functions	-
Input parameter{in}	
-	-
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* check and recover the Rx process */
```

```
enet_rxprocess_check_recovery();
```

enet_txfifo_flush

The description of enet_txfifo_flush is shown as below:

Table 3-335. Function enet_txfifo_flush

Function name	enet_txfifo_flush
----------------------	-------------------

Function prototype	ErrStatus enet_txfifo_flush(void);
Function descriptions	flush the ENET transmit FIFO, and wait until the flush operation completes
Precondition	-
The called functions	-
Input parameter{in}	
-	-
Output parameter{out}	
-	-
Return value	
ErrStatus	ERROR or SUCCESS

Example:

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

```
ErrStatus reval = ERROR;
```

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

enet_current_desc_address_get

The description of enet_current_desc_address_get is shown as below:

Table 3-336. Function enet_current_desc_address_get

Function name	enet_current_desc_address_get
Function prototype	uint32_t enet_current_desc_address_get(enet_desc_reg_enum addr_get);
Function descriptions	get the transmit/receive address of current descriptor, or current buffer, or descriptor table
Precondition	-
The called functions	-
Input parameter{in}	
addr_get	choose the address which users want to get only one parameter can be selected which is shown as below
<i>ENET_RX_DESC_TABLE</i>	the start address of the receive descriptor table
<i>ENET_RX_CURRENT_DESC</i>	the start descriptor address of the current receive descriptor read by the RxDMA controller
<i>ENET_RX_CURRENT_BUFFER</i>	the current receive buffer address being read by the RxDMA controller
<i>ENET_TX_DESC_TABLE</i>	the start address of the transmit descriptor table
<i>ENET_TX_CURRENT_DESC</i>	the start descriptor address of the current transmit descriptor read by the TxDMA controller
<i>ENET_TX_CURRENT_BUFFER</i>	the current transmit buffer address being read by the TxDMA controller
Output parameter{out}	

-	-
Return value	
uint32_t	0- 0xFFFFFFFF

Example:

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

```
uint32_t reval;
```

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

enet_desc_information_get

The description of enet_desc_information_get is shown as below:

Table 3-337. Function enet_desc_information_get

Function name	enet_desc_information_get
Function prototype	uint32_t enet_desc_information_get(enet_descriptors_struct *desc, enet_descstate_enum info_get);
Function descriptions	get the Tx or Rx descriptor information
Precondition	-
The called functions	-
Input parameter{in}	
desc	the descriptor pointer which users want to get information, the structure members can refer to Structure enet_descriptors_struct
Input parameter{in}	
info_get	the descriptor information type which is selected only one parameter can be selected which is shown as below
RXDESC_BUFFER_1_SIZE	receive buffer 1 size
RXDESC_BUFFER_2_SIZE	receive buffer 2 size
RXDESC_FRAME_LENGTH	the byte length of the received frame that was transferred to the buffer
TXDESC_COLLISION_COUNT	the number of collisions occurred before the frame was transmitted
RXDESC_BUFFER_1_ADDR	the buffer1 address of the Rx frame
TXDESC_BUFFER_1_ADDR	the buffer1 address of the Tx frame
Output parameter{out}	
-	-
Return value	
uint32_t	descriptor information if value is 0xFFFFFFFFU, means the false input parameter

Example:

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

uint32_t reval;

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

enet_missed_frame_counter_get

The description of enet_missed_frame_counter_get is shown as below:

Table 3-338. Function enet_missed_frame_counter_get

Function name	enet_missed_frame_counter_get
Function prototype	void enet_missed_frame_counter_get(uint32_t *rxfifo_drop, uint32_t *rxdma_drop);
Function descriptions	get the number of missed frames during receiving
Precondition	-
The called functions	-
Input parameter{in}	
-	-
Output parameter{out}	
rxfifo_drop	pointer to the number of frames dropped by Rx FIFO
Output parameter{out}	
rxdma_drop	pointer to the number of frames missed by the RxDMA controller
Return value	
-	-

Example:

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

uint32_t rxcnt, txcnt;

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

enet_desc_flag_get

The description of enet_desc_flag_get is shown as below:

Table 3-339. Function enet_desc_flag_get

Function name	enet_desc_flag_get
Function prototype	FlagStatus enet_desc_flag_get(enet_descriptors_struct *desc, uint32_t desc_flag);
Function descriptions	get the bit flag of ENET DMA descriptor
Precondition	-
The called functions	-
Input parameter{in}	
desc	the descriptor pointer which users want to get flag, the structure members

	can refer to Structure enet_descriptors_struct
Input parameter{in}	
desc_flag (the value according to the parameter desc)	the bit flag of ENET DMA descriptor only one parameter can be selected which is shown as below
When type of parameter desc is TX	
<i>ENET_TDES0_DB</i>	deferred
<i>ENET_TDES0_UFE</i>	underflow error
<i>ENET_TDES0_EXD</i>	excessive deferral
<i>ENET_TDES0_VFRM</i>	VLAN frame
<i>ENET_TDES0_ECO</i>	excessive collision
<i>ENET_TDES0_LCO</i>	late collision
<i>ENET_TDES0_NCA</i>	no carrier
<i>ENET_TDES0_LCA</i>	loss of carrier
<i>ENET_TDES0_IPPE</i>	IP payload error
<i>ENET_TDES0_FRMF</i>	frame flushed
<i>ENET_TDES0_JT</i>	jabber timeout
<i>ENET_TDES0_ES</i>	error summary
<i>ENET_TDES0_IPHE</i>	IP header error
<i>ENET_TDES0_TTMSS</i>	transmit timestamp status
<i>ENET_TDES0_TCHM</i>	the second address chained mode
<i>ENET_TDES0_TERM</i>	transmit end of ring mode
<i>ENET_TDES0_TTSSEN</i>	transmit timestamp function enable
<i>ENET_TDES0_DPAD</i>	disable adding pad
<i>ENET_TDES0_DCRC</i>	disable CRC
<i>ENET_TDES0_FSG</i>	first segment
<i>ENET_TDES0_LSG</i>	last segment
<i>ENET_TDES0_INTC</i>	interrupt on completion
<i>ENET_TDES0_DAV</i>	DAV bit
When type of parameter desc is RX	
<i>ENET_RDES0_PCERR</i>	payload checksum error
<i>ENET_RDES0_CERR</i>	CRC error
<i>ENET_RDES0_DBERR</i>	dribble bit error
<i>ENET_RDES0_RERR</i>	receive error
<i>ENET_RDES0_RWDT</i>	receive watchdog timeout
<i>ENET_RDES0_FRMT</i>	frame type
<i>ENET_RDES0_LCO</i>	late collision
<i>ENET_RDES0_IPHER</i> <i>R</i>	IP frame header error
<i>ENET_RDES0_LDES</i>	last descriptor
<i>ENET_RDES0_FDES</i>	first descriptor
<i>ENET_RDES0_VTAG</i>	VLAN tag
<i>ENET_RDES0_OERR</i>	overflow error

<i>ENET_RDES0_LERR</i>	length error
<i>ENET_RDES0_SAFF</i>	SA filter fail
<i>ENET_RDES0_DERR</i>	descriptor error
<i>ENET_RDES0_ERRS</i>	error summary
<i>ENET_RDES0_DAFF</i>	destination address filter fail
<i>ENET_RDES0_DAV</i>	descriptor available
Output parameter{out}	
-	-
Return value	
FlagStatus	SET or RESET

Example:

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

```
FlagStatus reval;
```

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

enet_desc_flag_set

The description of enet_desc_flag_set is shown as below:

Table 3-340. Function enet_desc_flag_set

Function name	enet_desc_flag_set
Function prototype	void enet_desc_flag_set(enet_descriptors_struct *desc, uint32_t desc_flag);
Function descriptions	set the bit flag of ENET DMA descriptor
Precondition	-
The called functions	-
Input parameter{in}	
desc	the descriptor pointer which users want to set flag, the structure members can refer to Structure enet_descriptors_struct
Input parameter{in}	
desc_flag (the value according to the parameter desc)	the bit flag of ENET DMA descriptor only one parameter can be selected which is shown as below
When type of parameter desc is TX	
<i>ENET_TDES0_VFRM</i>	VLAN frame
<i>ENET_TDES0_FRMF</i>	frame flushed
<i>ENET_TDES0_TCHM</i>	the second address chained mode
<i>ENET_TDES0_TERM</i>	transmit end of ring mode
<i>ENET_TDES0_TTSSEN</i>	transmit timestamp function enable
<i>ENET_TDES0_DPAD</i>	disable adding pad
<i>ENET_TDES0_DCRC</i>	disable CRC
<i>ENET_TDES0_FSG</i>	first segment
<i>ENET_TDES0_LSG</i>	last segment

<i>ENET_TDES0_INTC</i>	interrupt on completion
<i>ENET_TDES0_DAV</i>	DAV bit
When type of parameter desc is RX	
<i>ENET_RDES0_DAV</i>	descriptor available
Output parameter{out}	
-	-
Return value	
-	-

Example:

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

enet_desc_flag_clear

The description of enet_desc_flag_clear is shown as below:

Table 3-341. Function enet_desc_flag_clear

Function name	enet_desc_flag_clear
Function prototype	void enet_desc_flag_clear(enet_descriptors_struct *desc, uint32_t desc_flag);
Function descriptions	clear the bit flag of ENET DMA descriptor
Precondition	-
The called functions	-
Input parameter{in}	
desc	the descriptor pointer which users want to set flag, the structure members can refer to Structure enet_descriptors_struct
Input parameter{in}	
desc_flag (the value according to the parameter desc)	the bit flag of ENET DMA descriptor only one parameter can be selected which is shown as below
When type of parameter desc is TX	
<i>ENET_TDES0_VFRM</i>	VLAN frame
<i>ENET_TDES0_FRMF</i>	frame flushed
<i>ENET_TDES0_TCHM</i>	the second address chained mode
<i>ENET_TDES0_TERM</i>	transmit end of ring mode
<i>ENET_TDES0_TTSN</i>	transmit timestamp function enable
<i>ENET_TDES0_DPAD</i>	disable adding pad
<i>ENET_TDES0_DCRC</i>	disable CRC
<i>ENET_TDES0_FSG</i>	first segment
<i>ENET_TDES0_LSG</i>	last segment
<i>ENET_TDES0_INTC</i>	interrupt on completion
<i>ENET_TDES0_DAV</i>	DAV bit
When type of parameter desc is RX	

ENET_RDES0_DAV	descriptor available
Output parameter{out}	
-	-
Return value	
-	-

Example:

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

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

enet_rx_desc_immediate_receive_complete_interrupt

The description of enet_rx_desc_immediate_receive_complete_interrupt is shown as below:

Table 3-342. Function enet_rx_desc_immediate_receive_complete_interrupt

Function name	enet_rx_desc_immediate_receive_complete_interrupt
Function prototype	void enet_rx_desc_immediate_receive_complete_interrupt(enet_descriptors_struct *desc);
Function descriptions	when receiving completed, set RS bit in ENET_DMA_STAT register will immediately set
Precondition	-
The called functions	-
Input parameter{in}	
desc	the descriptor pointer which users want to get flag, the structure members can refer to Structure enet_descriptors_struct
Output parameter{out}	
-	-
Return value	
-	-

Example:

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

```
enet_rx_desc_immediate_receive_complete_interrupt(p_rxdesc);
```

enet_rx_desc_delay_receive_complete_interrupt

The description of enet_rx_desc_delay_receive_complete_interrupt is shown as below:

Table 3-343. Function enet_rx_desc_delay_receive_complete_interrupt

Function name	enet_rx_desc_delay_receive_complete_interrupt
Function prototype	void enet_rx_desc_delay_receive_complete_interrupt(enet_descriptors_struct *desc, uint32_t delay_time);

Function descriptions	when receiving completed, set RS bit in ENET_DMA_STAT register will is set after a configurable delay time
Precondition	-
The called functions	-
Input parameter{in}	
desc	the descriptor pointer which users want to get flag, the structure members can refer to Structure enet_descriptors_struct
Input parameter{in}	
delay_time	delay a time of 256*delay_time HCLK(0x00000000 - 0x000000FF)
Output parameter{out}	
-	-
Return value	
-	-

Example:

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

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

enet_rxframe_drop

The description of enet_rxframe_drop is shown as below:

Table 3-344. Function enet_rxframe_drop

Function name	enet_rxframe_drop
Function prototype	void enet_rxframe_drop(void);
Function descriptions	drop current receive frame
Precondition	-
The called functions	-
Input parameter{in}	
-	-
Output parameter{out}	
-	-
Return value	
-	-

Example:

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

```
enet_rxframe_drop( );
```

enet_dma_feature_enable

The description of enet_dma_feature_enable is shown as below:

Table 3-345. Function `enet_dma_feature_enable`

Function name	<code>enet_dma_feature_enable</code>
Function prototype	<code>void enet_dma_feature_enable(uint32_t feature);</code>
Function descriptions	enable DMA feature
Precondition	-
The called functions	-
Input parameter{in}	
feature	the feature of DMA mode one or more parameters can be selected which are shown as below
<code>ENET_NO_FLUSH_RXFRAME</code>	RxDMA does not flushes frames function
<code>ENET_SECONDFRAME_OPT</code>	TxDMA controller operate on second frame function
Output parameter{out}	
-	-
Return value	
-	-

Example:

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

```
enet_dma_feature_enable(ENET_NO_FLUSH_RXFRAME);
```

`enet_dma_feature_disable`

The description of `enet_dma_feature_disable` is shown as below:

Table 3-346. Function `enet_dma_feature_disable`

Function name	<code>enet_dma_feature_disable</code>
Function prototype	<code>void enet_dma_feature_disable(uint32_t feature);</code>
Function descriptions	disable DMA feature
Precondition	-
The called functions	-
Input parameter{in}	
feature	the feature of DMA mode one or more parameters can be selected which are shown as below
<code>ENET_NO_FLUSH_RXFRAME</code>	RxDMA does not flushes frames function
<code>ENET_SECONDFRAME_OPT</code>	TxDMA controller operate on second frame function
Output parameter{out}	
-	-
Return value	
-	-

Example:

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

```
enet_dma_feature_disable(ENET_NO_FLUSH_RXFRAME);
```

enet_rx_desc_enhanced_status_get

The description of enet_rx_desc_enhanced_status_get is shown as below:

Table 3-347. Function enet_rx_desc_enhanced_status_get

Function name	enet_rx_desc_enhanced_status_get
Function prototype	uint32_t enet_rx_desc_enhanced_status_get(enet_descriptors_struct *desc, uint32_t desc_status);
Function descriptions	get the bit of extended status flag in ENET DMA descriptor
Precondition	-
The called functions	-
Input parameter{in}	
desc	the descriptor pointer which users want to get the extended status flag, the structure members can refer to Structure enet_descriptors_struct
Input parameter{in}	
desc_status	desc_status: the extended status want to get only one parameter can be selected which is shown as below
ENET_RDES4_IPPLDT	IP frame payload type
ENET_RDES4_IPHER R	IP frame header error
ENET_RDES4_IPPLDER RR	IP frame payload error
ENET_RDES4_IPCKSB	IP frame checksum bypassed
ENET_RDES4_IPF4	IP frame in version 4
ENET_RDES4_IPF6	IP frame in version 6
ENET_RDES4_PTPMT	PTP message type
ENET_RDES4_PTPOEF	PTP on ethernet frame
ENET_RDES4_PTPVF	PTP version format
Output parameter{out}	
-	-
Return value	
-	-

Example:

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

```
uint32_t status;
```

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

enet_desc_select_enhanced_mode

The description of enet_desc_select_enhanced_mode is shown as below:

Table 3-348. Function enet_desc_select_enhanced_mode

Function name	enet_desc_select_enhanced_mode
Function prototype	void enet_desc_select_enhanced_mode(void);
Function descriptions	configure descriptor to work in enhanced mode
Precondition	-
The called functions	-
Input parameter{in}	
-	-
Output parameter{out}	
-	-
Return value	
-	-

Example:

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

enet_ptp_enhanced_descriptors_chain_init

The description of enet_ptp_enhanced_descriptors_chain_init is shown as below:

Table 3-349. Function enet_ptp_enhanced_descriptors_chain_init

Function name	enet_ptp_enhanced_descriptors_chain_init
Function prototype	void enet_ptp_enhanced_descriptors_chain_init(enet_dmadirection_enum direction);
Function descriptions	initialize the DMA Tx/Rx descriptors's parameters in enhanced chain mode with ptp function
Precondition	-
The called functions	-
Input parameter{in}	
direction	the descriptors which users want to init, only one parameter can be selected which is shown as below
ENET_DMA_TX	DMA Tx descriptors
ENET_DMA_RX	DMA Rx descriptors
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* initialize the DMA Tx descriptors's parameters in enhanced chain mode with ptp function */
```



```
enet_ptp_enhanced_descriptors_chain_init(ENET_DMA_TX);
```

enet_ptp_enhanced_descriptors_ring_init

The description of enet_ptp_enhanced_descriptors_ring_init is shown as below:

Table 3-350. Function enet_ptp_enhanced_descriptors_ring_init

Function name	enet_ptp_enhanced_descriptors_ring_init
Function prototype	void enet_ptp_enhanced_descriptors_ring_init(enet_dmadirection_enum direction);
Function descriptions	initialize the DMA Tx/Rx descriptors's parameters in enhanced ring mode with ptp function
Precondition	-
The called functions	-
Input parameter{in}	
direction	the descriptors which users want to init, only one parameter can be selected which is shown as below
ENET_DMA_TX	DMA Tx descriptors
ENET_DMA_RX	DMA Rx descriptors
Output parameter{out}	
-	-
Return value	
-	-

Example:

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

enet_ptpframe_receive_enhanced_mode

The description of enet_ptpframe_receive_enhanced_mode is shown as below:

Table 3-351. Function enet_ptpframe_receive_enhanced_mode

Function name	enet_ptpframe_receive_enhanced_mode
Function prototype	ErrStatus enet_ptpframe_receive_enhanced_mode(uint8_t *buffer, uint32_t bufsize, uint32_t timestamp[]);
Function descriptions	receive a packet data with timestamp values to application buffer, when the DMA is in enhanced mode
Precondition	-
The called functions	-
Input parameter{in}	
bufsize	the size of buffer which is the parameter in function
Output parameter{out}	
buffer	pointer to the application buffer
Output parameter{out}	

timestamp	pointer to the table which stores the timestamp high and low
Return value	
ErrStatus	SUCCESS or ERROR

Example:

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

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

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

```
ErrStatus status;
```

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

enet_ptpframe_transmit_enhanced_mode

The description of enet_ptpframe_transmit_enhanced_mode is shown as below:

Table 3-352. Function enet_ptpframe_transmit_enhanced_mode

Function name	enet_ptpframe_transmit_enhanced_mode
Function prototype	ErrStatus enet_ptpframe_transmit_enhanced_mode(uint8_t *buffer, uint32_t length, uint32_t timestamp[]);
Function descriptions	send data with timestamp values in application buffer as a transmit packet, when the DMA is in enhanced mode
Precondition	-
The called functions	-
Input parameter{in}	
buffer	pointer on the application buffer note -- if the input is NULL, user should copy data in application by himself
Input parameter{in}	
length	the length of frame data to be transmitted
Output parameter{out}	
timestamp	pointer to the table which stores the timestamp high and low note -- if the input is NULL, timestamp is ignored
Return value	
ErrStatus	SUCCESS or ERROR

Example:

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

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

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

```
ErrStatus status;
```

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

enet_desc_select_normal_mode

The description of enet_desc_select_normal_mode is shown as below:

Table 3-353. Function enet_desc_select_normal_mode

Function name	enet_desc_select_normal_mode
Function prototype	void enet_desc_select_normal_mode(void);
Function descriptions	configure descriptor to work in normal mode
Precondition	-
The called functions	-
Input parameter{in}	
-	-
Output parameter{out}	
-	-
Return value	
-	-

Example:

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

```
enet_desc_select_normal_mode( );
```

enet_ptp_normal_descriptors_chain_init

The description of enet_ptp_normal_descriptors_chain_init is shown as below:

Table 3-354. Function enet_ptp_normal_descriptors_chain_init

Function name	enet_ptp_normal_descriptors_chain_init
Function prototype	void enet_ptp_normal_descriptors_chain_init(enet_dmadirection_enum direction, enet_descriptors_struct *desc_ptptab);
Function descriptions	initialize the DMA Tx/Rx descriptors's parameters in normal chain mode with PTP function
Precondition	-
The called functions	-
Input parameter{in}	
direction	the descriptors which users want to init only one parameter can be selected which is shown as below
ENET_DMA_TX	DMA Tx descriptors
ENET_DMA_RX	DMA Rx descriptors
Input parameter{in}	
desc_ptptab	pointer to the first descriptor address of PTP Rx descriptor table, the structure members can refer to Structure enet_descriptors_struct
Output parameter{out}	
-	-

Return value	
-	-

Example:

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

enet_ptp_normal_descriptors_chain_init(ENET_DMA_RX, ptp_rxdesc_tab);
```

enet_ptp_normal_descriptors_ring_init

The description of enet_ptp_normal_descriptors_ring_init is shown as below:

Table 3-355. Function enet_ptp_normal_descriptors_ring_init

Function name	enet_ptp_normal_descriptors_ring_init
Function prototype	void enet_ptp_normal_descriptors_ring_init(enet_dmadirection_enum direction, enet_descriptors_struct *desc_ptptab);
Function descriptions	initialize the DMA Tx/Rx descriptors's parameters in normal ring mode with PTP function
Precondition	-
The called functions	-
Input parameter{in}	
direction	the descriptors which users want to init only one parameter can be selected which is shown as below
ENET_DMA_TX	DMA Tx descriptors
ENET_DMA_RX	DMA Rx descriptors
Input parameter{in}	
desc_ptptab	pointer to the first descriptor address of PTP Rx descriptor table, the structure members can refer to Structure enet_descriptors_struct
Output parameter{out}	
-	-
Return value	
-	-

Example:

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

enet_ptp_normal_descriptors_ring_init(ENET_DMA_RX, ptp_rxdesc_tab);
```

enet_ptpframe_receive_normal_mode

The description of enet_ptpframe_receive_normal_mode is shown as below:

Table 3-356. Function enet_ptpframe_receive_normal_mode

Function name	enet_ptpframe_receive_normal_mode
Function prototype	ErrStatus enet_ptpframe_receive_normal_mode(uint8_t *buffer, uint32_t bufsize, uint32_t timestamp[]);
Function descriptions	receive a packet data with timestamp values to application buffer, when the DMA is in normal mode
Precondition	-
The called functions	-
Input parameter{in}	
bufsize	the size of buffer which is the parameter in function
Output parameter{out}	
timestamp	pointer to the table which stores the timestamp high and low
Output parameter{out}	
buffer	pointer to the application buffer if the input is NULL, user should copy data in application by himself
Return value	
ErrStatus	ERROR or SUCCESS

Example:

```

/* receive a packet data with timestamp values to application buffer in DMA normal mode */
uint32_t rx_buffer[500];
uint32_t time_stamp[2];
ErrStatus status;

status = enet_ptpframe_receive_normal_mode(rx_buffer, 500, time_stamp);

```

enet_ptpframe_transmit_normal_mode

The description of enet_ptpframe_transmit_normal_mode is shown as below:

Table 3-357. Function enet_ptpframe_transmit_normal_mode

Function name	enet_ptpframe_transmit_normal_mode
Function prototype	ErrStatus enet_ptpframe_transmit_normal_mode(uint8_t *buffer, uint32_t length, uint32_t timestamp[]);
Function descriptions	send data with timestamp values in application buffer as a transmit packet, when the DMA is in normal mode
Precondition	-
The called functions	--
Input parameter{in}	
buffer	pointer on the application buffer if the input is NULL, user should copy data in application by himself
Input parameter{in}	
length	the length of frame data to be transmitted

Output parameter{out}	
timestamp	pointer to the table which stores the timestamp high and low if the input is NULL, timestamp is ignored
Return value	
ErrStatus	ERROR or SUCCESS

Example:

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

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

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

```
ErrStatus status;
```

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

enet_wum_filter_register_pointer_reset

The description of enet_wum_filter_register_pointer_reset is shown as below:

Table 3-358. Function enet_wum_filter_register_pointer_reset

Function name	enet_wum_filter_register_pointer_reset
Function prototype	void enet_wum_filter_register_pointer_reset(void);
Function descriptions	wakeup frame filter register pointer reset
Precondition	-
The called functions	-
Input parameter{in}	
-	-
Output parameter{out}	
-	-
Return value	
-	-

Example:

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

```
enet_wum_filter_register_pointer_reset ();
```

enet_wum_filter_config

The description of enet_wum_filter_config is shown as below:

Table 3-359. Function enet_wum_filter_config

Function name	enet_wum_filter_config
Function prototype	void enet_wum_filter_config(uint32_t pdata[]);
Function descriptions	set the remote wakeup frame registers

Precondition	-
The called functions	-
Input parameter{in}	
pdata	pointer to buffer data which is written to remote wakeup frame registers (8 words total)
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* set the remote wakeup frame registers */
```

```
uint32_t wum_data[8];
```

```
enet_wum_filter_config (wum_data);
```

enet_wum_feature_enable

The description of enet_wum_feature_enable is shown as below:

Table 3-360. Function enet_wum_feature_enable

Function name	enet_wum_feature_enable
Function prototype	void enet_wum_feature_enable(uint32_t feature);
Function descriptions	enable wakeup management features
Precondition	-
The called functions	-
Input parameter{in}	
feature	one or more parameters can be selected which are shown as below
<i>ENET_WUM_POWER_DOWN</i>	power down mode
<i>ENET_WUM_MAGIC_PACKET_FRAME</i>	enable a wakeup event due to magic packet reception
<i>ENET_WUM_WAKEUP_FRAME</i>	enable a wakeup event due to wakeup frame reception
<i>ENET_WUM_GLOBAL_UNICAST</i>	any received unicast frame passed filter is considered to be a wakeup frame
Output parameter{out}	
-	-
Return value	
-	-

Example:

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

```
enet_wum_feature_enable(ENET_WUM_POWER_DOWN);
```

enet_wum_feature_disable

The description of enet_wum_feature_disable is shown as below:

Table 3-361. Function enet_wum_feature_disable

Function name	enet_wum_feature_disable
Function prototype	void enet_wum_feature_disable(uint32_t feature)
Function descriptions	disable wakeup management features
Precondition	-
The called functions	-
Input parameter{in}	
feature	one or more parameters can be selected which are shown as below
ENET_WUM_MAGIC_PACKET_FRAME	enable a wakeup event due to magic packet reception
ENET_WUM_WAKEUP_FRAME	enable a wakeup event due to wakeup frame reception
ENET_WUM_GLOBAL_UNICAST	any received unicast frame passed filter is considered to be a wakeup frame
Output parameter{out}	
-	-
Return value	
-	-

Example:

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

```
enet_wum_feature_disable(ENET_WUM_POWER_DOWN);
```

enet_msc_counters_reset

The description of enet_msc_counters_reset is shown as below:

Table 3-362. Function enet_msc_counters_reset

Function name	enet_msc_counters_reset
Function prototype	void enet_msc_counters_reset(void);
Function descriptions	reset the MAC statistics counters
Precondition	-
The called functions	-
Input parameter{in}	
-	-
Output parameter{out}	
-	-
Return value	
-	-

Example:


```
/* reset the MAC statistics counters */
```

```
enet_msc_counters_reset();
```

enet_msc_feature_enable

The description of enet_msc_feature_enable is shown as below:

Table 3-363. Function enet_msc_feature_enable

Function name	enet_msc_feature_enable
Function prototype	void enet_msc_feature_enable(uint32_t feature);
Function descriptions	enable the MAC statistics counter features
Precondition	-
The called functions	-
Input parameter{in}	
feature	one or more parameters can be selected which are shown as below
ENET_MSC_COUNTER_STOP_ROLLOVER	counter stop rollover
ENET_MSC_COUNTER_RESET_ON_READ	reset on read
ENET_MSC_COUNTER_FREEZE	MSC counter freeze
Output parameter{out}	
-	-
Return value	
-	-

Example:

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

```
enet_msc_feature_enable(ENET_MSC_COUNTER_STOP_ROLLOVER);
```

enet_msc_feature_disable

The description of enet_msc_feature_disable is shown as below:

Table 3-364. Function enet_msc_feature_disable

Function name	enet_msc_feature_disable
Function prototype	void enet_msc_feature_disable(uint32_t feature);
Function descriptions	disable the MAC statistics counter features
Precondition	-
The called functions	-
Input parameter{in}	
feature	one or more parameters can be selected which are shown as below
ENET_MSC_COUNTER_STOP_ROLLOVER	counter stop rollover

<i>ENET_MSC_RESET_ON_READ</i>	reset on read
<i>ENET_MSC_COUNTER_STOP_ROLLOVER_FREEZE</i>	MSC counter freeze
Output parameter{out}	
-	-
Return value	
-	-

Example:

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

```
enet_msc_feature_disable(ENET_MSC_COUNTER_STOP_ROLLOVER);
```

enet_msc_counters_preset_config

The description of enet_msc_counters_preset_config is shown as below:

Table 3-365. Function enet_msc_counters_preset_config

Function name	enet_msc_counters_preset_config
Function prototype	void enet_msc_counters_preset_config(enet_msc_preset_enum mode);
Function descriptions	configure MAC statistics counters preset mode
Precondition	-
The called functions	-
Input parameter{in}	
mode	MSC counters preset mode, refer to enet_msc_preset_enum only one parameter can be selected which is shown as below
<i>ENET_MSC_PRESET_NONE</i>	do not preset MSC counter
<i>ENET_MSC_PRESET_HALF</i>	preset all MSC counters to almost-half(0x7FFF FFF0) value
<i>ENET_MSC_PRESET_FULL</i>	preset all MSC counters to almost-full(0xFFFF FFF0) value
Output parameter{out}	
-	-
Return value	
-	-

Example:

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

```
enet_msc_counters_preset_config (ENET_MSC_PRESET_HALF);
```

enet_msc_counters_get

The description of enet_msc_counters_get is shown as below:

Table 3-366. Function enet_msc_counters_get

Function name	enet_msc_counters_get
Function prototype	uint32_t enet_msc_counters_get(enet_msc_counter_enum counter);
Function descriptions	get MAC statistics counter
Precondition	-
The called functions	-
Input parameter{in}	
counter	MSC counters which is selected only one parameter can be selected which is shown as below
ENET_MSC_TX_SCCNT	MSC transmitted good frames after a single collision counter
ENET_MSC_TX_MSCCNT	MSC transmitted good frames after more than a single collision counter
ENET_MSC_TX_TGFCNT	MSC transmitted good frames counter
ENET_MSC_RX_RFCECNT	MSC received frames with CRC error counter
ENET_MSC_RX_RFAECNT	MSC received frames with alignment error counter
ENET_MSC_RX_RGUFCNT	MSC received good unicast frames counter
Output parameter{out}	
-	-
Return value	
uint32_t	the MSC counter value

Example:

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

```
uint32_t reval;
```

```
reval = enet_msc_counters_get(ENET_MSC_TX_SCCNT);
```

enet_ptp_subsecond_2_nanosecond

The description of enet_ptp_subsecond_2_nanosecond is shown as below:

Table 3-367. Function enet_ptp_subsecond_2_nanosecond

Function name	enet_ptp_subsecond_2_nanosecond
Function prototype	uint32_t enet_ptp_subsecond_2_nanosecond(uint32_t subsecond);
Function descriptions	change subsecond to nanosecond
Precondition	-
The called functions	-
Input parameter{in}	
subsecond	subsecond value

Output parameter{out}	
-	-
Return value	
uint32_t	the nanosecond value

Example:

```
/* change subsecond to nanosecond */
uint32_t reval;
reval = enet_ptp_subsecond_2_nanosecond (2);
```

enet_ptp_nanosecond_2_subsecond

The description of enet_ptp_nanosecond_2_subsecond is shown as below:

Table 3-368. Function enet_ptp_nanosecond_2_subsecond

Function name	enet_ptp_nanosecond_2_subsecond
Function prototype	uint32_t enet_ptp_nanosecond_2_subsecond(uint32_t nanosecond);
Function descriptions	change nanosecond to subsecond
Precondition	-
The called functions	-
Input parameter{in}	
nanosecond	Nanosecond value
Output parameter{out}	
-	-
Return value	
uint32_t	the subsecond value

Example:

```
/* change nanosecond to subsecond */
uint32_t reval;
reval = enet_ptp_nanosecond_2_subsecond (2);
```

enet_ptp_feature_enable

The description of enet_ptp_feature_enable is shown as below:

Table 3-369. Function enet_ptp_feature_enable

Function name	enet_ptp_feature_enable
Function prototype	void enet_ptp_feature_enable(uint32_t feature);
Function descriptions	enable the PTP features
Precondition	-
The called functions	-
Input parameter{in}	

feature	the feature of ENET PTP mode one or more parameters can be selected which are shown as below
<i>ENET_RXTX_TIMESTAMP</i>	timestamp function for transmit and receive frames
<i>ENET_PTP_TIMESTAMP_INTERRUPT</i>	timestamp interrupt trigger
<i>ENET_ALL_RX_TIMESTAMP</i>	all received frames are taken snapshot
<i>ENET_NONTYPE_FRAME_SNAPSHOT</i>	take snapshot when received non type frame
<i>ENET_IPV6_FRAME_SNAPSHOT</i>	take snapshot for IPv6 frame
<i>ENET_IPV4_FRAME_SNAPSHOT</i>	take snapshot for IPv4 frame
<i>ENET_PTP_FRAME_USE_MACADDRESS_FILTER</i>	use MAC address1-3 to filter the PTP frame
Output parameter{out}	
-	-
Return value	
-	-

Example:

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

```
enet_ptp_feature_enable(ENET_ALL_RX_TIMESTAMP);
```

enet_ptp_feature_disable

The description of enet_ptp_feature_disable is shown as below:

Table 3-370. Function enet_ptp_feature_disable

Function name	enet_ptp_feature_disable
Function prototype	void enet_ptp_feature_disable(uint32_t feature);
Function descriptions	disable the PTP features
Precondition	-
The called functions	-
Input parameter{in}	
feature	the feature of ENET PTP mode one or more parameters can be selected which are shown as below
<i>ENET_RXTX_TIMESTAMP</i>	timestamp function for transmit and receive frames
<i>ENET_PTP_TIMESTAMP_INTERRUPT</i>	timestamp interrupt trigger
<i>ENET_ALL_RX_TIMESTAMP</i>	all received frames are taken snapshot

<i>TAMP</i>	
<i>ENET_NONTYPE_FRAME_SNAPSHOT</i>	take snapshot when received non type frame
<i>ENET_IPV6_FRAME_SNAPSHOT</i>	take snapshot for IPv6 frame
<i>ENET_IPV4_FRAME_SNAPSHOT</i>	take snapshot for IPv4 frame
<i>ENET_PTP_FRAME_USE_MACADDRESS_FILTER</i>	use MAC address1-3 to filter the PTP frame
Output parameter{out}	
-	-
Return value	
-	-

Example:

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

```
enet_ptp_feature_disable(ENET_ALL_RX_TIMESTAMP);
```

enet_ptp_timestamp_function_config

The description of enet_ptp_timestamp_function_config is shown as below:

Table 3-371. Function enet_ptp_timestamp_function_config

Function name	enet_ptp_timestamp_function_config
Function prototype	ErrStatus enet_ptp_timestamp_function_config(enet_ptp_function_enum func);
Function descriptions	configure the PTP timestamp function
Precondition	-
The called functions	-
Input parameter{in}	
func	only one parameter can be selected which is shown as below
<i>ENET_CKNT_ORDINARY</i>	type of ordinary clock node type for timestamp
<i>ENET_CKNT_BOUNDARY</i>	type of boundary clock node type for timestamp
<i>ENET_CKNT_END_TO_END</i>	type of end-to-end transparent clock node type for timestamp
<i>ENET_CKNT_PEER_TO_PEER</i>	type of peer-to-peer transparent clock node type for timestamp
<i>ENET_PTP_ADDEND_UPDATE</i>	addend register update
<i>ENET_PTP_SYSTIME_UPDATE</i>	timestamp update

<i>ENET_PTP_SYSTIME_INIT</i>	timestamp initialize
<i>ENET_PTP_FINEMODE</i>	the system timestamp uses the fine method for updating
<i>ENET_PTP_COARSEMODE</i>	the system timestamp uses the coarse method for updating
<i>ENET_SUBSECOND_DIGITAL_ROLLOVER</i>	digital rollover mode
<i>ENET_SUBSECOND_BINARY_ROLLOVER</i>	digital rollover mode
<i>ENET_SNOOPING_PTP_VERSION_2</i>	version 2
<i>ENET_SNOOPING_PTP_VERSION_1</i>	version 1
<i>ENET_EVENT_TYPE_MESSAGES_SNAPSHOT</i>	only event type messages are taken snapshot
<i>ENET_ALL_TYPE_MESSAGES_SNAPSHOT</i>	all type messages are taken snapshot except announce, management and signaling message
<i>ENET_MASTER_NODE_MESSAGE_SNAPSHOT</i>	snapshot is only take for master node message
<i>ENET_SLAVE_NODE_MESSAGE_SNAPSHOT</i>	snapshot is only taken for slave node message
Output parameter{out}	
-	-
Return value	
ErrStatus	SUCCESS or ERROR

Example:

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

```
ErrStatus reval = ERROR;
```

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

enet_ptp_subsecond_increment_config

The description of enet_ptp_subsecond_increment_config is shown as below:

Table 3-372. Function enet_ptp_subsecond_increment_config

Function name	enet_ptp_subsecond_increment_config
Function prototype	void enet_ptp_subsecond_increment_config(uint32_t subsecond);
Function descriptions	configure system time subsecond increment value

Precondition	-
The called functions	-
Input parameter{in}	
subsecond	the value will be added to the subsecond value of system time, this value must be between 0 and 0xFF
Output parameter{out}	
-	-
Return value	
-	-

Example:

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

```
enet_ptp_subsecond_increment_config(0x1F);
```

enet_ptp_timestamp_addend_config

The description of enet_ptp_timestamp_addend_config is shown as below:

Table 3-373. Function enet_ptp_timestamp_addend_config

Function name	enet_ptp_timestamp_addend_config
Function prototype	void enet_ptp_timestamp_addend_config(uint32_t add);
Function descriptions	adjusting the clock frequency only in fine update mode
Precondition	-
The called functions	-
Input parameter{in}	
add	the value will be added to the accumulator register to achieve time synchronization (0 – 0xFFFF FFFF)
Output parameter{out}	
-	-
Return value	
-	-

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

```
enet_ptp_timestamp_addend_config(0x1FFF);
```

enet_ptp_timestamp_update_config

The description of enet_ptp_timestamp_update_config is shown as below:

Table 3-374. Function enet_ptp_timestamp_update_config

Function name	enet_ptp_timestamp_update_config
Function prototype	void enet_ptp_timestamp_update_config(uint32_t sign, uint32_t second, uint32_t subsecond);
Function descriptions	initialize or add/subtract to second of the system time
Precondition	-

The called functions	-
Input parameter{in}	
sign	timestamp update positive or negative sign, only one parameter can be selected which is shown as below
<i>ENET_PTP_ADD_TO_TIME</i>	update value is added to system time
<i>ENET_PTP_SUBTRACT_FROM_TIME</i>	timestamp update value is subtracted from system time
Input parameter{in}	
second	initializing or adding/subtracting to second of the system time (0 – 0xFFFF FFFF)
Input parameter{in}	
subsecond	the current subsecond of the system time with 0.46 ns accuracy (0 – 0x7FFF FFFF)
Output parameter{out}	
-	-
Return value	
-	-

Example:

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

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

enet_ptp_expected_time_config

The description of enet_ptp_expected_time_config is shown as below:

Table 3-375. Function enet_ptp_expected_time_config

Function name	enet_ptp_expected_time_config
Function prototype	void enet_ptp_expected_time_config(uint32_t second, uint32_t nanosecond);
Function descriptions	configure the expected target time
Precondition	-
The called functions	-
Input parameter{in}	
second	the expected target second time (0 – 0xFFFF FFFF)
Input parameter{in}	
nanosecond	the expected target nanosecond time (signed) (0 – 0xFFFF FFFF)
Output parameter{out}	
-	-
Return value	
-	-

Example:

```

/* configure the expected target time */

enet_ptp_expected_time_config(2000, 0);

```

enet_ptp_system_time_get

The description of enet_ptp_system_time_get is shown as below:

Table 3-376. Function enet_ptp_system_time_get

Function name	enet_ptp_system_time_get
Function prototype	void enet_ptp_system_time_get(enet_ptp_systime_struct *systime_struct);
Function descriptions	get the current system time
Precondition	-
The called functions	-
Input parameter{in}	
-	-
Output parameter{out}	
systime_struct	pointer to a enet_ptp_systime_struct structure which contains parameters of PTP system time, the structure members can refer to Structure enet_ptp_systime_struct
Return value	
-	-

Example:

```

/* get the current system time */

enet_ptp_systime_struct systime;

enet_ptp_system_time_get(&systime);

```

enet_ptp_pps_output_frequency_config

The description of enet_ptp_pps_output_frequency_config is shown as below:

Table 3-377. Function enet_ptp_pps_output_frequency_config

Function name	enet_ptp_pps_output_frequency_config
Function prototype	void enet_ptp_pps_output_frequency_config(uint32_t freq);
Function descriptions	configure the PPS output frequency
Precondition	-
The called functions	-
Input parameter{in}	
freq	
ENET_PPISOFC_1HZ	PPS output 1Hz frequency
ENET_PPISOFC_2HZ	PPS output 2Hz frequency
ENET_PPISOFC_4HZ	PPS output 4Hz frequency
ENET_PPISOFC_8HZ	PPS output 8Hz frequency
ENET_PPISOFC_16HZ	PPS output 16Hz frequency

<i>ENET_PPISOFC_32HZ</i>	PPS output 32Hz frequency
<i>ENET_PPISOFC_64HZ</i>	PPS output 64Hz frequency
<i>ENET_PPISOFC_128HZ</i>	PPS output 128Hz frequency
<i>ENET_PPISOFC_256HZ</i>	PPS output 256Hz frequency
<i>ENET_PPISOFC_512HZ</i>	PPS output 512Hz frequency
<i>ENET_PPISOFC_1024HZ</i>	PPS output 1024Hz frequency
<i>ENET_PPISOFC_2048HZ</i>	PPS output 2048Hz frequency
<i>ENET_PPISOFC_4096HZ</i>	PPS output 4096Hz frequency
<i>ENET_PPISOFC_8192HZ</i>	PPS output 8192Hz frequency
<i>ENET_PPISOFC_16384HZ</i>	PPS output 16384Hz frequency
<i>ENET_PPISOFC_32768HZ</i>	PPS output 32768Hz frequency
Output parameter{out}	
-	-
Return value	
-	-

Example:

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

```
enet_ptp_pps_output_frequency_config(ENET_PPISOFC_1HZ);
```

enet_ptp_start

The enet_ptp_start is shown as below:

Table 3-378. enet_ptp_start

Function name	enet_ptp_start
Function prototype	void enet_ptp_start(int32_t updatemethod, uint32_t init_sec, uint32_t init_subsec, uint32_t carry_cfg, uint32_t accuracy_cfg);
Function descriptions	configure and start PTP timestamp counter
Precondition	-
The called functions	-
Input parameter{in}	
updatemethod	method for updating
<i>ENET_PTP_FINEMODE</i>	fine correction method

<i>ENET_PTP_COARSE MODE</i>	coarse correction method
Input parameter{in}	
init_sec	second value for initializing system time
Input parameter{in}	
init_subsec	subsecond value for initializing system time
Input parameter{in}	
carry_cfg	the value to be added to the accumulator register (in fine method is used)
Input parameter{in}	
accuracy_cfg	the value to be added to the subsecond value of system time
Output parameter{out}	
-	-
Return value	
-	-

Example:

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

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

enet_ptp_finecorrection_adjfreq

The enet_ptp_finecorrection_adjfreq is shown as below:

Table 3-379. enet_ptp_finecorrection_adjfreq

Function name	enet_ptp_finecorrection_adjfreq
Function prototype	void enet_ptp_finecorrection_adjfreq(int32_t carry_cfg);
Function descriptions	adjust frequency in fine method by configure addend register
Precondition	-
The called functions	-
Input parameter{in}	
carry_cfg	the value to be added to the accumulator register
Output parameter{out}	
-	-
Return value	
-	-

Example:

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

```
enet_ptp_finecorrection_adjfreq(10);
```

enet_ptp_coarsecorrection_systime_update

The enet_ptp_coarsecorrection_systime_update is shown as below:

Table 3-380. enet_ptp_coarsecorrection_systime_update

Function name	enet_ptp_coarsecorrection_systime_update
Function prototype	void enet_ptp_coarsecorrection_systime_update(enet_ptp_systime_struct *systime_struct);
Function descriptions	update system time in coarse method
Precondition	-
The called functions	-
Input parameter{in}	
systime_struct	the descriptor pointer which users want to configure, the structure members can refer to Structure enet_ptp_systime_struct
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* update system time in coarse method */
enet_ptp_systime_struct systime_struct;
enet_ptp_coarsecorrection_systime_update (&systime_struct);
```

enet_ptp_finecorrection_settime

The enet_ptp_finecorrection_settime is shown as below:

Table 3-381. enet_ptp_finecorrection_settime

Function name	enet_ptp_finecorrection_settime
Function prototype	void enet_ptp_finecorrection_settime(enet_ptp_systime_struct *systime_struct);
Function descriptions	set system time in fine method
Precondition	-
The called functions	-
Input parameter{in}	
systime_struct	the descriptor pointer which users want to configure, the structure members can refer to Structure enet_ptp_systime_struct
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* set system time in fine method */
enet_ptp_systime_struct systime_struct;
```

```
enet_ptp_finecorrection_settime (&systime_struct);
```

enet_ptp_flag_get

The enet_ptp_flag_get is shown as below:

Table 3-382. enet_ptp_flag_get

Function name	enet_ptp_flag_get
Function prototype	FlagStatus enet_ptp_flag_get(uint32_t flag);
Function descriptions	get the ptp flag status
Precondition	-
The called functions	-
Input parameter{in}	
flag	ptp flag status to be checked
ENET_PTP_ADDEND_UPDATE	addend register update
ENET_PTP_SYSTIME_UPDATE	timestamp update
ENET_PTP_SYSTIME_INIT	timestamp initialize
Output parameter{out}	
-	-
Return value	
FlagStatus	SET or RESET

Example:

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

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

enet_initpara_reset

The description of enet_initpara_reset is shown as below:

Table 3-383. Function enet_initpara_reset

Function name	enet_initpara_reset
Function prototype	void enet_initpara_reset(void);
Function descriptions	reset the ENET initpara struct, call it before using enet_initpara_config()
Precondition	-
The called functions	-
Input parameter{in}	
-	-
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* reset the ENET initpara struct */
enet_initpara_reset();
```

3.12. EXMC

The external memory controller EXMC, is used as a translator for MCU to access a variety of external memory. The EXMC registers are listed in chapter [3.12.1](#), the EXMC firmware functions are introduced in chapter [3.12.2](#).

3.12.1. Descriptions of Peripheral registers

EXMC registers are listed in the table shown as below:

Table 3-384. EXMC Registers

Registers	Descriptions
EXMC_SNCTL	SRAM/NOR Flash control registers
EXMC_SNTCFG	SRAM/NOR Flash timing configuration registers
EXMC_SNWTCFG	SRAM/NOR Flash write timing configuration registers
EXMC_NPCTL	NAND flash/PC card control registers
EXMC_NPINTEN	NAND flash/PC card interrupt enable registers
EXMC_NPCTCFG	NAND flash/PC card common space timing configuration registers
EXMC_NPATCFG	NAND flash/PC card attribute space timing configuration registers
EXMC_PIOTCFG3	PC card I/O space timing configuration register
EXMC_NECC	NAND flash ECC registers
EXMC_SDCTL	SDRAM control registers
EXMC_SDTCFG	SDRAM timing configuration registers
EXMC_SDCMD	SDRAM command register
EXMC_SDARI	SDRAM auto-refresh interval register
EXMC_SDSTAT	SDRAM status register
EXMC_SDRSCTL	SDRAM read sample control register
EXMC_SINIT	SPI initialization register
EXMC_SRCMD	SPI read command register
EXMC_SWCMD	SPI write command register
EXMC_SIDL	SPI ID low register
EXMC_SIDH	SPI ID high register

3.12.2. Descriptions of Peripheral functions

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

Table 3-385. EXMC firmware function

Function name	Function description
exmc_norsram_deinit	deinitialize EXMC NOR/SRAM regionx
exmc_norsram_struct_para_init	initialize exmc_norsram_parameter_struct with the default values
exmc_norsram_init	initialize EXMC NOR/SRAM regionx
exmc_norsram_enable	enable EXMC NOR/PSRAM regionx
exmc_norsram_disable	disable EXMC NOR/PSRAM regionx
exmc_nand_deinit	deinitialize EXMC NAND bankx
exmc_nand_struct_para_init	initialize exmc_norsram_parameter_struct with the default values
exmc_nand_init	initialize EXMC NAND bankx
exmc_nand_enable	enable EXMC NAND bankx
exmc_nand_disable	disable EXMC NAND bankx
exmc_pccard_deinit	deinitialize EXMC PC card bank
exmc_pccard_struct_para_init	initialize exmc_pccard_parameter_struct with the default values
exmc_pccard_init	initialize EXMC PC card bank
exmc_pccard_enable	enable EXMC PC card bank
exmc_pccard_disable	disable EXMC PC card bank
exmc_sdram_deinit	deinitialize EXMC SDRAM devicex
exmc_sdram_struct_para_init	initialize exmc_sdram_parameter_struct with the default values
exmc_sdram_init	initialize EXMC SDRAM devicex
exmc_sdram_struct_command_parameter_init	initialize exmc_sdram_command_parameter_struct with the default values
exmc_sqpsram_deinit	deinitialize EXMC SQPSRAM
exmc_sqpsram_struct_para_init	initialize exmc_sqpsram_parameter_struct with the default values
exmc_sqpsram_init	initialize EXMC SQPSRAM
exmc_norsram_consecutive_clock_config	configure consecutive clock
exmc_norsram_page_size_config	configure CRAM page size
exmc_nand_ecc_config	enable or disable the EXMC NAND ECC function
exmc_ecc_get	get the EXMC ECC value
exmc_sdram_readsample_enable	enable or disable read sample
exmc_sdram_readsample_config	configure the delayed sample clock of read data
exmc_sdram_command_config	configure the SDRAM memory command
exmc_sdram_refresh_count_set	set auto-refresh interval
exmc_sdram_autorefresh_number_set	set the number of successive auto-refresh command
exmc_sdram_write_protection_config	config the write protection function
exmc_sdram_bankstatus_get	get the status of SDRAM device0 or device1

Function name	Function description
exmc_sqpsram_read_command_set	set the read command
exmc_sqpsram_write_command_set	set the write command
exmc_sqpsram_read_id_command_send	send SPI read ID command
exmc_sqpsram_write_cmd_send	send SPI special command which does not have address and data phase
exmc_sqpsram_low_id_get	get the EXMC SPI ID low data
exmc_sqpsram_high_id_get	get the EXMC SPI ID high data
exmc_sqpsram_send_command_status_get	get the bit value of EXMC send write command bit or read ID command
exmc_interrupt_enable	enable EXMC interrupt
exmc_interrupt_disable	disable EXMC interrupt
exmc_flag_get	get EXMC flag status
exmc_flag_clear	clear EXMC flag status
exmc_interrupt_flag_get	get EXMC interrupt flag
exmc_interrupt_flag_clear	clear EXMC interrupt flag

Structure exmc_norsram_timing_parameter_struct

Table 3-386. Structure exmc_norsram_timing_parameter_struct

Member name	Function description
asyn_access_mode	asynchronous access mode
syn_data_latency	configure the data latency
syn_data_latency_decrease	configure the data latency decreasing value
syn_clk_division	configure the clock divide ratio
bus_latency	configure the bus latency
asyn_data_setup_time	configure the data setup time, asynchronous access mode valid
asyn_address_hold_time	configure the address hold time, asynchronous access mode valid
asyn_address_setup_time	configure the data setup time, asynchronous access mode valid

Structure exmc_norsram_parameter_struct

Table 3-387. Structure exmc_norsram_parameter_struct

Member name	Function description
norsram_region	select the region of EXMC NOR/SRAM
write_mode	the write mode, synchronous mode or asynchronous mode
extended_mode	enable or disable the extended mode
asyn_wait	enable or disable the asynchronous wait function

Member name	Function description
nwait_signal	enable or disable the NWAIT signal while in synchronous bust mode
memory_write	enable or disable the write operation
nwait_config	NWAIT signal configuration
wrap_burst_mode	enable or disable the wrap burst mode
nwait_polarity	specifies the polarity of NWAIT signal from memory
burst_mode	enable or disable the burst mode
databus_width	specifies the databus width of external memory
memory_type	specifies the type of external memory
address_data_mux	specifies whether the data bus and address bus are multiplexed
read_write_timing	timing parameters for read and write if the extended mode is not used or the timing parameters for read if the extended mode is used
write_timing	timing parameters for write when the extended mode is used

Structure exmc_nand_pccard_timing_parameter_struct

Table 3-388. Structure exmc_nand_pccard_timing_parameter_struct

Member name	Function description
databus_hiztime	configure the databus HiZ time for write operation
holdtime	configure the address hold time(or the data hold time for write operation)
waittime	configure the minimum wait time
setuptime	configure the address setup time

Structure exmc_nand_parameter_struct

Table 3-389. Structure exmc_nand_parameter_struct

Member name	Function description
nand_bank	select the bank of NAND
ecc_size	the page size for the ECC calculation
atr_latency	configure the latency of ALE low to RB low
ctr_latency	configure the latency of CLE low to RB low
ecc_logic	enable or disable the ECC calculation logic
databus_width	the NAND flash databus width
wait_feature	enable or disable the wait feature
common_space_timing	the timing parameters for NAND flash common space
attribute_space_timing	the timing parameters for NAND flash attribute space

Structure exmc_pccard_parameter_struct

Table 3-390. Structure exmc_pccard_parameter_struct

Member name	Function description
atr_latency	configure the latency of ALE low to RB low

Member name	Function description
ctr_latency	configure the latency of CLE low to RB low
wait_feature	enable or disable the wait feature
common_space_timing	the timing parameters for PC card common space
attribute_space_timing	the timing parameters for PC card attribute space
io_space_timing	the timing parameters for PC card IO space

Structure exmc_sdram_timing_parameter_struct

Table 3-391. Structure exmc_sdram_timing_parameter_struct

Member name	Function description
row_to_column_delay	configure the row to column delay
row_precharge_delay	configure the row precharge delay
write_recovery_delay	configure the write recovery delay
auto_refresh_delay	configure the auto refresh delay
row_address_select_delay	configure the row address select delay
exit_selfrefresh_delay	configure the exit self-refresh delay
load_mode_register_delay	configure the load mode register delay

Structure exmc_sdram_parameter_struct

Table 3-392. Structure exmc_sdram_parameter_struct

Member name	Function description
sdram_device	select the device of SDRAM
pipeline_read_delay	the delay for reading data after CAS latency in HCLK clock cycles
burst_read_switch	enable or disable the burst read
sdclk_config	the SDCLK memory clock for both SDRAM banks
write_protection	enable or disable SDRAM bank write protection function
cas_latency	configure the SDRAM CAS latency
internal_bank_number	the number of internal bank
data_width	the databus width of SDRAM memory
row_address_width	the bit width of a row address
column_address_width	the bit width of a column address
timing	the timing parameters for write and read SDRAM

Structure `exmc_sdram_command_parameter_struct`

Table 3-393. Structure `exmc_sdram_command_parameter_struct`

Member name	Function description
<code>mode_register_content</code>	the SDRAM mode register content
<code>auto_refresh_number</code>	the number of successive auto-refresh cycles will be send when CMD = 011
<code>bank_select</code>	the bank which command will be sent to
<code>command</code>	the commands that will be sent to SDRAM

Structure `exmc_sqpsram_parameter_struct`

Table 3-394. Structure `exmc_sqpsram_parameter_struct`

Member name	Function description
<code>sample_polarity</code>	read data sample polarity
<code>id_length</code>	SPI PSRAM ID length
<code>address_bits</code>	bit number of SPI PSRAM address phase
<code>command_bits</code>	bit number of SPI PSRAM command phase

`exmc_norsram_deinit`

The description of `exmc_norsram_deinit` is shown as below:

Table 3-395. Function `exmc_norsram_deinit`

Function name	<code>exmc_norsram_deinit</code>
Function prototype	<code>void exmc_norsram_deinit(uint32_t exmc_norsram_region);</code>
Function descriptions	deinitialize EXMC NOR/SRAM region
Precondition	-
The called functions	-
Input parameter{in}	
<code>exmc_norsram_region</code>	EXMC NOR/SRAM region
<code>EXMC_BANK0_NORSRAM_REGIONx</code>	x=0,1,2,3
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* deinitialize the EXMC NOR/SRAM region1 of bank0 */
exmc_norsram_deinit(EXMC_BANK0_NORSRAM_REGION1);
```

exmc_norsram_struct_para_init

The description of exmc_norsram_struct_para_init is shown as below:

Table 3-396. Function exmc_norsram_struct_para_init

Function name	exmc_norsram_struct_para_init
Function prototype	void exmc_norsram_struct_para_init(exmc_norsram_parameter_struct* exmc_norsram_init_struct);
Function descriptions	initialize the struct exmc_norsram_parameter_struct
Precondition	-
The called functions	-
Input parameter{in}	
exmc_norsram_init_struct	Structure for initialization, the structure members can refer to Table 3-387. Structure exmc_norsram_parameter_struct
Output parameter{out}	
-	-
Return value	
-	-

Example:

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

exmc_norsram_init

The description of exmc_norsram_init is shown as below:

Table 3-397. Function exmc_norsram_init

Function name	exmc_norsram_init
Function prototype	void exmc_norsram_init(exmc_norsram_parameter_struct* exmc_norsram_init_struct);
Function descriptions	initialize EXMC NOR/SRAM region
Precondition	-
The called functions	-
Input parameter{in}	
exmc_norsram_init_struct	Structure for initialization, the structure members can refer to Table 3-387. Structure exmc_norsram_parameter_struct
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* initialize EXMC NOR/SRAM bank */

exmc_norsram_parameter_struct lcd_init_struct;

exmc_norsram_timing_parameter_struct lcd_timing_init_struct;

/* configure timing parameter */

lcd_timing_init_struct.asyn_access_mode = EXMC_ACCESS_MODE_A;

lcd_timing_init_struct.syn_data_latency = EXMC_DATA_LAT_2_CLK;

lcd_timing_init_struct.syn_data_latency_dec = 0;

lcd_timing_init_struct.syn_clk_division = EXMC_SYN_CLOCK_RATIO_DISABLE;

lcd_timing_init_struct.bus_latency = 1;

lcd_timing_init_struct.asyn_data_setup_time = 5;

lcd_timing_init_struct.asyn_address_hold_time = 2;

lcd_timing_init_struct.asyn_address_setup_time = 2;

/* configure EXMC bus parameters */

lcd_init_struct.norsram_region = EXMC_BANK0_NORSRAM_REGION1;

lcd_init_struct.write_mode = EXMC_ASYN_WRITE;

lcd_init_struct.extended_mode = DISABLE;

lcd_init_struct.asyn_wait = DISABLE;

lcd_init_struct.nwait_signal = DISABLE;

lcd_init_struct.memory_write = ENABLE;

lcd_init_struct.nwait_config = EXMC_NWAIT_CONFIG_BEFORE;

lcd_init_struct.wrap_burst_mode = DISABLE;

lcd_init_struct.nwait_polarity = EXMC_NWAIT_POLARITY_LOW;

lcd_init_struct.burst_mode = DISABLE;

lcd_init_struct.databus_width = EXMC_NOR_DATABUS_WIDTH_16B;

lcd_init_struct.memory_type = EXMC_MEMORY_TYPE_SRAM;

lcd_init_struct.address_data_mux = DISABLE;

lcd_init_struct.read_write_timing = &lcd_timing_init_struct;

lcd_init_struct.write_timing = &lcd_timing_init_struct;

exmc_norsram_init(&lcd_init_struct);
```

exmc_norsram_enable

The description of exmc_norsram_enable is shown as below:

Table 3-398. Function exmc_norsram_enable

Function name	exmc_norsram_enable
Function prototype	void exmc_norsram_enable(uint32_t exmc_norsram_region);
Function descriptions	enable EXMC NOR/PSRAM region
Precondition	-
The called functions	-
Input parameter{in}	
exmc_norsram_region	EXMC NOR/SRAM region
<i>EXMC_BANK0_NORSRAM_REGIONx</i>	x=0,1,2,3
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* enable the EXMC NOR/SRAM region1 of bank0 */
exmc_norsram_enable(EXMC_BANK0_NORSRAM_REGION1);
```

exmc_norsram_disable

The description of exmc_norsram_disable is shown as below:

Table 3-399. Function exmc_norsram_disable

Function name	exmc_norsram_disable
Function prototype	void exmc_norsram_disable(uint32_t exmc_norsram_region);
Function descriptions	disable EXMC NOR/PSRAM region
Precondition	-
The called functions	-
Input parameter{in}	
exmc_norsram_region	EXMC NOR/SRAM region
<i>EXMC_BANK0_NORSRAM_REGIONx</i>	x=0,1,2,3
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* disable the EXMC NOR/SRAM region1 of bank0 */
```

```
exmc_norsram_disable(EXMC_BANK0_NORSRAM_REGION1);
```

exmc_nand_deinit

The description of exmc_nand_deinit is shown as below:

Table 3-400. Function exmc_nand_deinit

Function name	exmc_nand_deinit
Function prototype	void exmc_nand_deinit(uint32_t exmc_nand_bank);
Function descriptions	deinitialize EXMC NAND bank
Precondition	-
The called functions	-
Input parameter{in}	
exmc_nand_bank	the bank of NAND
EXMC_BANKx_NAND	x=1,2
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* deinitialize EXMC NOR/SRAM bank1 */
```

```
exmc_norsram_deinit(EXMC_BANK1_NAND);
```

exmc_nand_struct_para_init

The description of exmc_nand_struct_para_init is shown as below:

Table 3-401. Function exmc_nand_struct_para_init

Function name	exmc_nand_struct_para_init
Function prototype	void exmc_nand_struct_para_init(exmc_nand_parameter_struct* exmc_nand_init_struct);
Function descriptions	initialize the struct exmc_nand_parameter_struct
Precondition	-
The called functions	-
Input parameter{in}	
exmc_nand_init_struct	Structure for initialization, the structure members can refer to Table 3-389 . Structure exmc_nand_parameter_struct
Output parameter{out}	
-	-
Return value	
-	-

Example:


```

/* initialize the struct nand_init_struct */

exmc_nand_parameter_struct nand_init_struct;

exmc_nand_struct_para_init (&nand_init_struct);

```

exmc_nand_init

The description of exmc_nand_init is shown as below:

Table 3-402. Function exmc_nand_init

Function name	exmc_nand_init
Function prototype	void exmc_nand_init(exmc_nand_parameter_struct* exmc_nand_init_struct);
Function descriptions	initialize EXMC NAND bank
Precondition	-
The called functions	-
Input parameter{in}	
exmc_nand_init_struct	Structure for initialization, the structure members can refer to Table 3-389. Structure exmc_nand_parameter_struct
Output parameter{out}	
-	-
Return value	
-	-

Example:

```

exmc_nand_parameter_struct nand_init_struct;

exmc_nand_pccard_timing_parameter_struct nand_timing_init_struct;

/* EXMC configuration */

nand_timing_init_struct.setuptime = 5;

nand_timing_init_struct.waittime = 4;

nand_timing_init_struct.holdtime = 2;

nand_timing_init_struct.databus_hiztime = 2;

nand_init_struct.nand_bank = EXMC_BANK1_NAND;

nand_init_struct.ecc_size = EXMC_ECC_SIZE_2048BYTES;

nand_init_struct.atr_latency = EXMC_ALE_RE_DELAY_1_HCLK;

nand_init_struct.ctr_latency = EXMC_CLE_RE_DELAY_1_HCLK;

nand_init_struct.ecc_logic = ENABLE;

nand_init_struct.databus_width = EXMC_NAND_DATABUS_WIDTH_8B;

```

```
nand_init_struct.wait_feature = ENABLE;

nand_init_struct.common_space_timing = &nand_timing_init_struct;

nand_init_struct.attribute_space_timing = &nand_timing_init_struct;

exmc_nand_init(&nand_init_struct);
```

exmc_nand_enable

The description of exmc_nand_enable is shown as below:

Table 3-403. Function exmc_nand_enable

Function name	exmc_nand_enable
Function prototype	void exmc_nand_enable(uint32_t exmc_nand_bank);
Function descriptions	enable EXMC NAND bank
Precondition	-
The called functions	-
Input parameter{in}	
exmc_nand_bank	the bank of NAND
<i>EXMC_BANKx_NAND</i>	x=1,2
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* enable EXMC NAND bank1 */

exmc_nand_enable(EXMC_BANK1_NAND);
```

exmc_nand_disable

The description of exmc_nand_disable is shown as below:

Table 3-404. Function exmc_nand_disable

Function name	exmc_nand_disable
Function prototype	exmc_nand_disable(uint32_t exmc_nand_bank);
Function descriptions	disable EXMC NAND bank
Precondition	-
The called functions	-
Input parameter{in}	
exmc_nand_bank	the bank of NAND
<i>EXMC_BANKx_NAND</i>	x=1,2
Output parameter{out}	
-	-
Return value	

-	-
---	---

Example:

```
/* disable EXMC NAND bank1 */
exmc_nand_disable(EXMC_BANK1_NAND);
```

exmc_pccard_deinit

The description of exmc_pccard_deinit is shown as below:

Table 3-405. Function exmc_pccard_deinit

Function name	exmc_pccard_deinit
Function prototype	void exmc_pccard_deinit(void);
Function descriptions	deinitialize EXMC PC card bank
Precondition	-
The called functions	-
Input parameter{in}	
-	-
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* deinitialize EXMC PC card bank */
exmc_pccard_deinit();
```

exmc_pccard_struct_para_init

The description of exmc_pccard_struct_para_init is shown as below:

Table 3-406. Function exmc_pccard_struct_para_init

Function name	exmc_pccard_struct_para_init
Function prototype	void exmc_pccard_struct_para_init(exmc_pccard_parameter_struct* exmc_pccard_init_struct);
Function descriptions	initialize the struct exmc_pccard_parameter_struct
Precondition	-
The called functions	-
Input parameter{in}	
exmc_pccard_init_struct	Structure for initialization, the structure members can refer to Table 3-390. Structure exmc_pccard_parameter_struct
Output parameter{out}	
-	-
Return value	

-	-
---	---

Example:

```
/* initialize the struct pccard_init_struct */
exmc_pccard_parameter_struct pccard_init_struct;
exmc_pccard_struct_para_init (&pccard_init_struct);
```

exmc_pccard_init

The description of exmc_pccard_init is shown as below:

Table 3-407. Function exmc_pccard_init

Function name	exmc_pccard_init
Function prototype	void exmc_pccard_init(exmc_pccard_parameter_struct* exmc_pccard_init_struct);
Function descriptions	initialize EXMC PC card bank
Precondition	-
The called functions	-
Input parameter{in}	
exmc_pccard_init_struct	Structure for initialization, the structure members can refer to Table 3-390. Structure exmc_pccard_parameter_struct
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
exmc_pccard_parameter_struct pccard_init_struct;
exmc_nand_pccard_timing_parameter_struct pccard_timing_init_struct;
/* EXMC configuration */
pccard_timing_init_struct.setuptime = 5;
pccard_timing_init_struct.waittime = 4;
pccard_timing_init_struct.holdtime = 2;
pccard_timing_init_struct.databus_hiztime = 2;
pccard_init_struct.atr_latency = EXMC_ALE_RE_DELAY_1_HCLK;
pccard_init_struct.ctr_latency = EXMC_CLE_RE_DELAY_1_HCLK;
pccard_init_struct.wait_feature = ENABLE;
pccard_init_struct.common_space_timing = &pccard_timing_init_struct;
```

```

pccard_init_struct.attribute_space_timing = & pccard_timing_init_struct;

pccard_init_struct.io_space_timing = & pccard_timing_init_struct;

exmc_pccard_init(&pccard_init_struct);

```

exmc_pccard_enable

The description of exmc_pccard_enable is shown as below:

Table 3-408. Function exmc_pccard_enable

Function name	exmc_pccard_enable
Function prototype	void exmc_pccard_enable(void);
Function descriptions	enable EXMC PC card bank
Precondition	-
The called functions	-
Input parameter{in}	
-	-
Output parameter{out}	
-	-
Return value	
-	-

Example:

```

/* enable EXMC PC card bank */

exmc_pccard_enable();

```

exmc_pccard_disable

The description of exmc_pccard_disable is shown as below:

Table 3-409. Function exmc_pccard_disable

Function name	exmc_pccard_disable
Function prototype	void exmc_pccard_disable(void);
Function descriptions	disable EXMC PC card bank
Precondition	-
The called functions	-
Input parameter{in}	
-	-
Output parameter{out}	
-	-
Return value	
-	-

Example:

```

/* disable EXMC PC card bank */

```

```
exmc_pccard_disable();
```

exmc_sdram_deinit

The description of exmc_sdram_deinit is shown as below:

Table 3-410. Function exmc_sdram_deinit

Function name	exmc_sdram_deinit
Function prototype	void exmc_sdram_deinit(uint32_t exmc_sdram_device);
Function descriptions	deinitialize EXMC SDRAM device
Precondition	-
The called functions	-
Input parameter{in}	
exmc_sdram_device	EXMC SDRAM device
<i>EXMC_SDRAM_DEVICE</i> <i>Ex</i>	x=0,1
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* deinitialize EXMC SDRAM device1 */
exmc_sdram_deinit(EXMC_SDRAM_DEVICE1);
```

exmc_sdram_struct_para_init

The description of exmc_sdram_struct_para_init is shown as below:

Table 3-411. Function exmc_sdram_struct_para_init

Function name	exmc_sdram_struct_para_init
Function prototype	exmc_sdram_struct_para_init(exmc_sdram_parameter_struct* exmc_sdram_init_struct);
Function descriptions	initialize the struct exmc_sdram_parameter_struct
Precondition	-
The called functions	-
Input parameter{in}	
exmc_sdram_init_struct	Structure for initialization, the structure members can refer to Table 3-392. Structure exmc_sdram_parameter_struct
Output parameter{out}	
-	-
Return value	
-	-

Example:

```

/* initialize the struct sdram_init_struct */

exmc_sdram_parameter_struct sdram_init_struct;

exmc_sdram_struct_para_init(&sdram_init_struct);

```

exmc_sdram_init

The description of exmc_sdram_init is shown as below:

Table 3-412. Function exmc_sdram_init

Function name	exmc_sdram_init
Function prototype	void exmc_sdram_init(exmc_sdram_parameter_struct* exmc_sdram_init_struct);
Function descriptions	initialize EXMC SDRAM device
Precondition	-
The called functions	-
Input parameter{in}	
exmc_sdram_init_struct	Structure for initialization, the structure members can refer to Table 3-392. Structure exmc_sdram_parameter_struct
Output parameter{out}	
-	-
Return value	
-	-

Example:

```

exmc_sdram_parameter_struct      sdram_init_struct;

exmc_sdram_timing_parameter_struct sdram_timing_init_struct;

/* EXMC configuration */

sdram_timing_init_struct.load_mode_register_delay = 2;

/* XSRD: min = 67ns */

sdram_timing_init_struct.exit_selfrefresh_delay = 7;

/* RASD: min=42ns , max=120k (ns) */

sdram_timing_init_struct.row_address_select_delay = 5;

/* ARFD: min=60ns */

sdram_timing_init_struct.auto_refresh_delay = 6;

/* WRD: min=1 Clock cycles +6ns */

sdram_timing_init_struct.write_recovery_delay = 2;

/* RPD: min=18ns */

```

```

sdram_timing_init_struct.row_precharge_delay = 2;

/* RCD: min=18ns */

sdram_timing_init_struct.row_to_column_delay = 2;

sdram_init_struct.sdram_device = sdram_device;

sdram_init_struct.column_address_width = EXMC_SDRAM_COW_ADDRESS_9;

sdram_init_struct.row_address_width = EXMC_SDRAM_ROW_ADDRESS_13;

sdram_init_struct.data_width = EXMC_SDRAM_DATABUS_WIDTH_16B;

sdram_init_struct.internal_bank_number = EXMC_SDRAM_4_INTER_BANK;

sdram_init_struct.cas_latency = EXMC_CAS_LATENCY_3_SDCLK;

sdram_init_struct.write_protection = DISABLE;

sdram_init_struct.sdclk_config = EXMC_SDCLK_PERIODS_2_HCLK;

sdram_init_struct.burst_read_switch = ENABLE;

sdram_init_struct.pipeline_read_delay = EXMC_PIPELINE_DELAY_1_HCLK;

sdram_init_struct.timing = &sdram_timing_init_struct;

/* EXMC SDRAM bank initialization */

exmc_sdram_init(&sdram_init_struct);

```

exmc_sdram_struct_command_para_init

The description of exmc_sdram_struct_command_para_init is shown as below:

Table 3-413. Function exmc_sdram_struct_command_para_init

Function name	exmc_sdram_struct_command_para_init
Function prototype	void exmc_sdram_struct_command_para_init(exmc_sdram_command_parameter_struct *exmc_sdram_command_init_struct);
Function descriptions	initialize exmc_sdram_command_parameter_struct with the default values
Precondition	-
The called functions	-
Input parameter{in}	
-	-
Output parameter{out}	
exmc_sdram_command_parameter_struct	structure for initialization, the structure members can refer to Table 3-393. Structure exmc_sdram_command_parameter_struct
Return value	
-	-

Example:


```
/* initialize the structure exmc_sdram_command_init_struct */
```

```
exmc_sdram_command_parameter_struct t_exmc_sdram_command_init_struct;
```

```
exmc_sdram_struct_command_para_init(&exmc_sdram_command_init_struct);
```

exmc_sqpsram_deinit

The description of exmc_sqpsram_deinit is shown as below:

Table 3-414. Function exmc_sqpsram_deinit

Function name	exmc_sqpsram_deinit
Function prototype	void exmc_sqpsram_deinit(void);
Function descriptions	deinitialize EXMC SQPIPSRAM
Precondition	-
The called functions	-
Input parameter{in}	
-	-
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* deinitialize EXMC SQPIPSRAM */
```

```
exmc_sqpsram_deinit(void);
```

exmc_sqpsram_struct_para_init

The description of exmc_sqpsram_struct_para_init is shown as below:

Table 3-415. Function exmc_sqpsram_struct_para_init

Function name	exmc_sqpsram_struct_para_init
Function prototype	void exmc_sqpsram_struct_para_init(exmc_sqpsram_parameter_struct* exmc_sqpsram_init_struct);
Function descriptions	initialize the struct exmc_sqpsram_parameter_struct
Precondition	-
The called functions	-
Input parameter{in}	
exmc_sqpsram_init_struct	Structure for initialization, the structure members can refer to Table 3-394. Structure exmc_sqpsram_parameter_struct
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* initialize the struct sqpipsram_init_struct */

exmc_sqpipsram_parameter_struct sqpipsram_init_struct;

exmc_sqpipsram_struct_para_init(&sqpipsram_init_struct);
```

exmc_sqpipsram_init

The description of exmc_sqpipsram_init is shown as below:

Table 3-416. Function exmc_sqpipsram_init

Function name	exmc_sqpipsram_init
Function prototype	void exmc_sqpipsram_init(exmc_sqpipsram_parameter_struct* exmc_sqpipsram_init_struct);
Function descriptions	initialize EXMC SQPIPSRAM
Precondition	-
The called functions	-
Input parameter{in}	
exmc_sqpipsram_init_struct	Structure for initialization, the structure members can refer to Table 3-394. Structure exmc_sqpipsram_parameter_struct
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
exmc_sqpipsram_parameter_struct      sqpipsram_init_struct;

/* EXMC configuration */

sqpipsram_init_struct.sample_polarity = EXMC_SQPIPSRAM_SAMPLE_RISING_EDGE;
sqpipsram_init_struct.id_length = EXMC_SQPIPSRAM_ID_LENGTH_64B;
sqpipsram_init_struct.address_bits = EXMC_SQPIPSRAM_ADDR_LENGTH_24B;
sqpipsram_init_struct.command_bits = EXMC_SQPIPSRAM_COMMAND_LENGTH_8B;

/* EXMC SDRAM bank initialization */

exmc_sqpipsram_init(&sqpipsram_init_struct);
```

exmc_norsram_consecutive_clock_config

The description of exmc_norsram_consecutive_clock_config is shown as below:

Table 3-417. Function exmc_norsram_consecutive_clock_config

Function name	exmc_norsram_consecutive_clock_config
Function prototype	void exmc_norsram_consecutive_clock_config(uint32_t clock_mode);

Function descriptions	configure consecutive clock
Precondition	-
The called functions	-
Input parameter{in}	
clock_mode	the mode of consecutive clock
<i>EXMC_CLOCK_SYN_MODE</i>	the clock is generated only during synchronous access
<i>EXMC_CLOCK_UNCONDITIONALLY</i>	the clock is generated unconditionally
Output parameter{out}	
-	-
Return value	
-	-

Example:

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

```
exmc_norsram_consecutive_clock_config(EXMC_CLOCK_SYN_MODE);
```

exmc_norsram_page_size_config

The description of exmc_norsram_page_size_config is shown as below:

Table 3-418. Function exmc_norsram_page_size_config

Function name	exmc_norsram_page_size_config
Function prototype	void exmc_norsram_page_size_config(uint32_t page_size);
Function descriptions	configure CRAM page size
Precondition	-
The called functions	-
Input parameter{in}	
page_size	CRAM page size
<i>EXMC_CRAM_AUTO_SPLIT</i>	the clock is generated only during synchronous access
<i>EXMC_CRAM_PAGE_SIZE_128_BYTES</i>	page size is 128 bytes
<i>EXMC_CRAM_PAGE_SIZE_256_BYTES</i>	page size is 256 bytes
<i>EXMC_CRAM_PAGE_SIZE_512_BYTES</i>	page size is 512 bytes
<i>EXMC_CRAM_PAGE_SIZE_1024_BYTES</i>	page size is 1024 bytes
Output parameter{out}	
-	-
Return value	
-	-

Example:

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

```
exmc_norsram_page_size_config (EXMC_CRAM_PAGE_SIZE_128_BYTES);
```

exmc_nand_ecc_config

The description of exmc_nand_ecc_config is shown as below:

Table 3-419. Function exmc_nand_ecc_config

Function name	exmc_nand_ecc_config
Function prototype	void exmc_nand_ecc_config(uint32_t exmc_nand_bank, ControlStatus newvalue);
Function descriptions	enable or disable the EXMC NAND ECC function
Precondition	-
The called functions	-
Input parameter{in}	
exmc_nand_bank	specifie the NAND bank
EXMC_BANKx_NAND	x=1,2
Input parameter{in}	
newvalue	ENABLE or DISABLE
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* enable the EXMC NAND ECC function */
```

```
exmc_nand_ecc_config(EXMC_BANK1_NAND, ENABLE);
```

exmc_ecc_get

The description of exmc_ecc_get is shown as below:

Table 3-420. Function exmc_ecc_get

Function name	exmc_ecc_get
Function prototype	uint32_t exmc_ecc_get(uint32_t exmc_nand_bank);
Function descriptions	get the EXMC ECC value
Precondition	-
The called functions	-
Input parameter{in}	
exmc_nand_bank	specifie the NAND bank
EXMC_BANKx_NAND	x=1,2
Output parameter{out}	
-	-

Return value	
uint32_t	the error correction code(ECC) value

Example:

```
/* get the EXMC ECC value */
```

```
uint32_t ecc_value;
```

```
ecc_value = exmc_ecc_get(EXMC_BANK1_NAND);
```

exmc_sdram_readsample_enable

The description of exmc_sdram_readsample_enable is shown as below:

Table 3-421. Function exmc_sdram_readsample_enable

Function name	exmc_sdram_readsample_enable
Function prototype	void exmc_sdram_readsample_enable(ControlStatus newvalue);
Function descriptions	enable or disable read sample
Precondition	-
The called functions	-
Input parameter{in}	
newvalue	ENABLE or DISABLE
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* enable read sample */
```

```
exmc_sdram_readsample_enable(ENABLE);
```

exmc_sdram_readsample_config

The description of exmc_sdram_readsample_config is shown as below:

Table 3-422. Function exmc_sdram_readsample_config

Function name	exmc_sdram_readsample_config
Function prototype	void exmc_sdram_readsample_config(uint32_t delay_cell, uint32_t extra_hclk);
Function descriptions	configure the delayed sample clock of read data
Precondition	-
The called functions	-
Input parameter{in}	
delay_cell	SDRAM the delayed sample clock of read data
EXMC_SDRAM_x_DEL AY_CELL	x=0...15

Input parameter{in}	
extra_hclk	sample cycle of read data
<i>EXMC_SDRAM_READ SAMPLE_x_EXTRAHCLK</i>	x=0,1
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* configure the delayed sample clock and sample cycle of read data */
exmc_sdram_readsample_config(EXMC_SDRAM_1_DELAY_CELL,
EXMC_SDRAM_READSAMPLE_1_EXTRAHCLK);
```

exmc_sdram_command_config

The description of exmc_sdram_command_config is shown as below:

Table 3-423. Function exmc_sdram_command_config

Function name	exmc_sdram_command_config
Function prototype	void exmc_sdram_command_config(exmc_sdram_command_parameter_struct* exmc_sdram_command_init_struct);
Function descriptions	configure the SDRAM memory command
Precondition	-
The called functions	-
Input parameter{in}	
exmc_sdram_command_init_struct	Structure for initialization, the structure members can refer to Table 3-393. Structure exmc_sdram_command_parameter_struct
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* configure the SDRAM memory command */
exmc_sdram_command_parameter_struct    sdram_command_init_struct;
sdram_command_init_struct.command = EXMC_SDRAM_CLOCK_ENABLE;
sdram_command_init_struct.bank_select = bank_select;
sdram_command_init_struct.auto_refresh_number =
EXMC_SDRAM_AUTO_REFLESH_1_SDCLK;
```

```
sdrmc_command_init_struct.mode_register_content = 0;

exmc_sdrmc_command_config(&sdrmc_command_init_struct);
```

exmc_sdrmc_refresh_count_set

The description of exmc_sdrmc_refresh_count_set is shown as below:

Table 3-424. Function exmc_sdrmc_refresh_count_set

Function name	exmc_sdrmc_refresh_count_set
Function prototype	void exmc_sdrmc_refresh_count_set(uint32_t exmc_count);
Function descriptions	set auto-refresh interval
Precondition	-
The called functions	-
Input parameter{in}	
exmc_count	the number SDRAM clock cycles unit between two successive auto-refresh commands, 0x0000~0x1FFF
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* set the SDRAM auto-refresh rate counter */

exmc_sdrmc_refresh_count_set(761);
```

exmc_sdrmc_autorefresh_number_set

The description of exmc_sdrmc_autorefresh_number_set is shown as below:

Table 3-425. Function exmc_sdrmc_autorefresh_number_set

Function name	exmc_sdrmc_autorefresh_number_set
Function prototype	void exmc_sdrmc_autorefresh_number_set(uint32_t exmc_number);
Function descriptions	set the number of successive auto-refresh command
Precondition	-
The called functions	-
Input parameter{in}	
exmc_number	the number of successive Auto-refresh cycles will be send
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* set the number of successive auto-refresh command */
```

```
exmc_sdram_autorefresh_number_set(10);
```

exmc_sdram_write_protection_config

The description of exmc_sdram_write_protection_config is shown as below:

Table 3-426. Function exmc_sdram_write_protection_config

Function name	exmc_sdram_write_protection_config
Function prototype	void exmc_sdram_write_protection_config(uint32_t exmc_sdram_device, ControlStatus newvalue);
Function descriptions	config the write protection function
Precondition	-
The called functions	-
Input parameter{in}	
exmc_sdram_device	specifie the SDRAM device
<i>EXMC_SDRAM_DEVICE</i> Ex	x=0,1
Input parameter{in}	
newvalue	ENABLE or DISABLE
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* enable the write protection function */
```

```
exmc_sdram_write_protection_config(EXMC_SDRAM_DEVICE1, ENABLE);
```

exmc_sdram_bankstatus_get

The description of exmc_sdram_bankstatus_get is shown as below:

Table 3-427. Function exmc_sdram_bankstatus_get

Function name	exmc_sdram_bankstatus_get
Function prototype	uint32_t exmc_sdram_bankstatus_get(uint32_t exmc_sdram_device);
Function descriptions	get the status of SDRAM device0 or device1
Precondition	-
The called functions	-
Input parameter{in}	
exmc_sdram_device	specifie the SDRAM device
<i>EXMC_SDRAM_DEVICE</i> Ex	x=0,1
Output parameter{out}	
-	-
Return value	

uint32_t	the status of SDRAM device
-----------------	----------------------------

Example:

```
/* get the status of SDRAM device1 */
```

```
uint32_t status;
```

```
status = exmc_sdram_bankstatus_get (EXMC_SDRAM_DEVICE1);
```

exmc_sqpsram_read_command_set

The description of exmc_sqpsram_read_command_set is shown as below:

Table 3-428. Function exmc_sqpsram_read_command_set

Function name	exmc_sqpsram_read_command_set
Function prototype	void exmc_sqpsram_read_command_set(uint32_t read_command_mode,uint32_t read_wait_cycle,uint32_t read_command_code);
Function descriptions	set the read command
Precondition	-
The called functions	-
Input parameter{in}	
read_command_mode	configure SPI PSRAM read command mode
EXMC_SQPIPSRAM_READ_MODE_DISABLE	not SPI mode
EXMC_SQPIPSRAM_READ_MODE_SPI	SPI mode
EXMC_SQPIPSRAM_READ_MODE_SQPI	SQPI mode
EXMC_SQPIPSRAM_READ_MODE_QPI	QPI mode
Input parameter{in}	
read_wait_cycle	wait cycle number after address phase,0..15
Input parameter{in}	
read_command_code	read command for AHB read transfer
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* set the SQPIPSRAM read command */
```

```
exmc_sqpsram_read_command_set(EXMC_SQPIPSRAM_READ_MODE_SQPI,1,1);
```

exmc_sqpsram_write_command_set

The description of exmc_sqpsram_write_command_set is shown as below:

Table 3-429. Function exmc_sqpsram_write_command_set

Function name	exmc_sqpsram_write_command_set
Function prototype	void exmc_sqpsram_write_command_set(uint32_t write_command_mode,uint32_t write_wait_cycle,uint32_t write_command_code);
Function descriptions	set the write command
Precondition	-
The called functions	-
Input parameter{in}	
write_command_mode	configure SPI PSRAM write command mode
EXMC_SQPIPSRAM_WRITE_MODE_DISABLE	not SPI mode
EXMC_SQPIPSRAM_WRITE_MODE_SPI	SPI mode
EXMC_SQPIPSRAM_WRITE_MODE_SQPI	SQPI mode
EXMC_SQPIPSRAM_WRITE_MODE_QPI	QPI mode
Input parameter{in}	
write_wait_cycle	wait cycle number after address phase,0..15
Input parameter{in}	
write_command_code	write command for AHB write transfer
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* set the SQPIPSRAM write command */
```

```
exmc_sqpsram_write_command_set(EXMC_SQPIPSRAM_WRITE_MODE_SQPI,1,1);
```

exmc_sqpsram_read_id_command_send

The description of exmc_sqpsram_read_id_command_send is shown as below:

Table 3-430. Function exmc_sqpsram_read_id_command_send

Function name	exmc_sqpsram_read_id_command_send
Function prototype	void exmc_sqpsram_read_id_command_send(void);
Function descriptions	send SPI read ID command

Precondition	-
The called functions	-
Input parameter{in}	
-	-
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* send SPI read ID command */
exmc_sqpsram_read_id_command_send();
```

exmc_sqpsram_write_cmd_send

The description of exmc_sqpsram_write_cmd_send is shown as below:

Table 3-431. Function exmc_sqpsram_write_cmd_send

Function name	exmc_sqpsram_write_cmd_send
Function prototype	void exmc_sqpsram_write_cmd_send(void);
Function descriptions	send SPI special command which does not have address and data phase
Precondition	-
The called functions	-
Input parameter{in}	
-	-
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* send SPI special command which does not have address and data phase */
exmc_sqpsram_write_cmd_send();
```

exmc_sqpsram_low_id_get

The description of exmc_sqpsram_low_id_get is shown as below:

Table 3-432. Function exmc_sqpsram_low_id_get

Function name	exmc_sqpsram_low_id_get
Function prototype	uint32_t exmc_sqpsram_low_id_get(void);
Function descriptions	get the EXMC SPI ID low data
Precondition	-
The called functions	-

Input parameter{in}	
-	-
Output parameter{out}	
-	-
Return value	
uint32_t	the ID low data

Example:

```
/* the ID low data */
uint32_t id;
id = exmc_sqpsram_low_id_get();
```

exmc_sqpsram_high_id_get

The description of exmc_sqpsram_high_id_get is shown as below:

Table 3-433. Function exmc_sqpsram_low_id_get

Function name	exmc_sqpsram_high_id_get
Function prototype	uint32_t exmc_sqpsram_high_id_get(void);
Function descriptions	get the EXMC SPI ID high data
Precondition	-
The called functions	-
Input parameter{in}	
-	-
Output parameter{out}	
-	-
Return value	
uint32_t	the ID high data

Example:

```
/* the ID high data */
uint32_t id;
id = exmc_sqpsram_high_id_get();
```

exmc_sqpsram_send_command_state_get

The description of exmc_sqpsram_send_command_state_get is shown as below:

Table 3-434. Function exmc_sqpsram_send_command_state_get

Function name	exmc_sqpsram_send_command_state_get
Function prototype	FlagStatus exmc_sqpsram_send_command_state_get(uint32_t send_command_flag);
Function descriptions	get the bit value of EXMC send write command bit or read ID command

Precondition	-
The called functions	-
Input parameter{in}	
send_command_flag	the send command flag
<i>EXMC_SEND_COMMA ND_FLAG_RDID</i>	EXMC_SRCMD_RDID flag bit
<i>EXMC_SEND_COMMA ND_FLAG_SC</i>	EXMC_SWCMD_SC flag bit
Output parameter{out}	
-	-
Return value	
FlagStatus	SET or RESET

Example:

```
/* check EXMC_SEND_COMMAND_FLAG_SC is set or not*/
```

```
if(RESET !=exmc_sqpsram_send_command_state_get(EXMC_SEND_COMMAND_FLAG  
_SC));
```

exmc_interrupt_enable

The description of exmc_interrupt_enable is shown as below:

Table 3-435. Function exmc_interrupt_enable

Function name	exmc_interrupt_enable
Function prototype	void exmc_interrupt_enable(uint32_t exmc_bank,uint32_t interrupt);
Function descriptions	enable EXMC interrupt
Precondition	-
The called functions	-
Input parameter{in}	
exmc_bank	specifies the NAND bank , PC card bank or SDRAM device
<i>EXMC_BANK1_NAND</i>	the NAND bank1
<i>EXMC_BANK2_NAND</i>	the NAND bank2
<i>EXMC_BANK3_PCCA RD</i>	the PC Card bank
<i>EXMC_SDRAM_DEVIC E0</i>	the SDRAM device0
<i>EXMC_SDRAM_DEVIC E1</i>	the SDRAM device1
Input parameter{in}	
interrupt	specify get which interrupt flag
<i>EXMC_NAND_PCCAR D_INT_FLAG_RISE</i>	rising edge interrupt and flag
<i>EXMC_NAND_PCCAR D_INT_FLAG_LEVEL</i>	high-level interrupt and flag

<i>EXMC_NAND_PCCAR</i> <i>D_INT_FLAG_FALL</i>	falling edge interrupt and flag
<i>EXMC_SDRAM_INT_F</i> <i>LAG_REFRESH</i>	refresh error interrupt and flag
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* enable EXMC interrupt*/
```

```
exmc_interrupt_enable(EXMC_BANK1_NAND, EXMC_NAND_PCCARD_INT_FLAG_RISE);
```

exmc_interrupt_disable

The description of exmc_interrupt_disable is shown as below:

Table 3-436. Function exmc_interrupt_disable

Function name	exmc_interrupt_disable
Function prototype	void exmc_interrupt_disable(uint32_t exmc_bank, uint32_t interrupt);
Function descriptions	disable EXMC interrupt
Precondition	-
The called functions	-
Input parameter{in}	
exmc_bank	specifies the NAND bank , PC card bank or SDRAM device
<i>EXMC_BANK1_NAND</i>	the NAND bank1
<i>EXMC_BANK2_NAND</i>	the NAND bank2
<i>EXMC_BANK3_PCCA</i> <i>RD</i>	the PC Card bank
<i>EXMC_SDRAM_DEVIC</i> <i>E0</i>	the SDRAM device0
<i>EXMC_SDRAM_DEVIC</i> <i>E1</i>	the SDRAM device1
Input parameter{in}	
interrupt	specify get which interrupt flag
<i>EXMC_NAND_PCCAR</i> <i>D_INT_FLAG_RISE</i>	rising edge interrupt and flag
<i>EXMC_NAND_PCCAR</i> <i>D_INT_FLAG_LEVEL</i>	high-level interrupt and flag
<i>EXMC_NAND_PCCAR</i> <i>D_INT_FLAG_FALL</i>	falling edge interrupt and flag
<i>EXMC_SDRAM_INT_F</i> <i>LAG_REFRESH</i>	refresh error interrupt and flag
Output parameter{out}	

-	-
Return value	
-	-

Example:

```
/* disable EXMC interrupt*/
```

```
exmc_interrupt_disable(EXMC_BANK1_NAND,
EXMC_NAND_PCCARD_INT_FLAG_RISE);
```

exmc_flag_get

The description of exmc_flag_get is shown as below:

Table 3-437. Function exmc_flag_get

Function name	exmc_flag_get
Function prototype	FlagStatus exmc_flag_get(uint32_t exmc_bank,uint32_t flag);
Function descriptions	get EXMC flag status
Precondition	-
The called functions	-
Input parameter{in}	
exmc_bank	specifies the NAND bank , PC card bank or SDRAM device
EXMC_BANK1_NAND	the NAND bank1
EXMC_BANK2_NAND	the NAND bank2
EXMC_BANK3_PCCARD	the PC Card bank
EXMC_SDRAM_DEVICE0	the SDRAM device0
EXMC_SDRAM_DEVICE1	the SDRAM device1
Input parameter{in}	
flag	specify get which flag
EXMC_NAND_PCCARD_FLAG_RISE	interrupt rising edge status
EXMC_NAND_PCCARD_FLAG_LEVEL	interrupt high-level status
EXMC_NAND_PCCARD_FLAG_FALL	interrupt falling edge status
EXMC_SDRAM_FLAG_REFRESH	refresh error interrupt flag
EXMC_SDRAM_FLAG_NREADY	not ready status
Output parameter{out}	
-	-
Return value	

FlagStatus	SET or RESET
-------------------	--------------

Example:

```
/* check EXMC_NAND_PCCARD_FLAG_RISE is set or not*/
```

```
if(RESET != exmc_flag_get (EXMC_BANK1_NAND,
EXMC_NAND_PCCARD_FLAG_RISE));
```

exmc_flag_clear

The description of exmc_flag_clear is shown as below:

Table 3-438. Function exmc_flag_clear

Function name	exmc_flag_clear
Function prototype	FlagStatus exmc_flag_clear (uint32_t exmc_bank,uint32_t flag);
Function descriptions	clear EXMC flag status
Precondition	-
The called functions	-
Input parameter{in}	
exmc_bank	specifies the NAND bank , PC card bank or SDRAM device
EXMC_BANK1_NAND	the NAND bank1
EXMC_BANK2_NAND	the NAND bank2
EXMC_BANK3_PCCARD	the PC Card bank
EXMC_SDRAM_DEVICE0	the SDRAM device0
EXMC_SDRAM_DEVICE1	the SDRAM device1
Input parameter{in}	
flag	specify get which flag
EXMC_NAND_PCCARD_FLAG_RISE	interrupt rising edge status
EXMC_NAND_PCCARD_FLAG_LEVEL	interrupt high-level status
EXMC_NAND_PCCARD_FLAG_FALL	interrupt falling edge status
EXMC_SDRAM_FLAG_REFRESH	refresh error interrupt flag
EXMC_SDRAM_FLAG_NREADY	not ready status
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* clear EXMC flag status */
```

```
exmc_flag_clear(EXMC_BANK1_NAND, EXMC_NAND_PCCARD_FLAG_RISE);
```

exmc_interrupt_flag_get

The description of exmc_interrupt_flag_get is shown as below:

Table 3-439. Function exmc_interrupt_flag_get

Function name	exmc_interrupt_flag_get
Function prototype	FlagStatus exmc_interrupt_flag_get(uint32_t exmc_bank, uint32_t interrupt);
Function descriptions	get EXMC interrupt flag
Precondition	-
The called functions	-
Input parameter{in}	
exmc_bank	specifies the NAND bank , PC card bank or SDRAM device
EXMC_BANK1_NAND	the NAND bank1
EXMC_BANK2_NAND	the NAND bank2
EXMC_BANK3_PCCARD	the PC Card bank
EXMC_SDRAM_DEVICE0	the SDRAM device0
EXMC_SDRAM_DEVICE1	the SDRAM device1
Input parameter{in}	
interrupt	specify get which interrupt flag
EXMC_NAND_PCCARD_INT_FLAG_RISE	rising edge interrupt and flag
EXMC_NAND_PCCARD_INT_FLAG_LEVEL	high-level interrupt and flag
EXMC_NAND_PCCARD_INT_FLAG_FALL	falling edge interrupt and flag
EXMC_SDRAM_INT_FLAG_REFRESH	refresh error interrupt and flag
Output parameter{out}	
-	-
Return value	
FlagStatus	SET or RESET

Example:

```
/* check EXMC_NAND_PCCARD_INT_FLAG_RISE is set or not */
```

```
if(RESET != exmc_interrupt_flag_get (EXMC_BANK1_NAND, EXMC_NAND_PCCARD_INT_FLAG_RISE));
```

exmc_interrupt_flag_clear

The description of exmc_interrupt_flag_clear is shown as below:

Table 3-440. Function exmc_interrupt_flag_clear

Function name	exmc_interrupt_flag_clear
Function prototype	void exmc_interrupt_flag_clear(uint32_t exmc_bank, uint32_t int_flag);
Function descriptions	clear EXMC interrupt flag
Precondition	-
The called functions	-
Input parameter{in}	
exmc_bank	specifies the NAND bank , PC card bank or SDRAM device
EXMC_BANK1_NAND	the NAND bank1
EXMC_BANK2_NAND	the NAND bank2
EXMC_BANK3_PCCARD	the PC Card bank
EXMC_SDRAM_DEVICE0	the SDRAM device0
EXMC_SDRAM_DEVICE1	the SDRAM device1
Input parameter{in}	
int_flag	specify get which interrupt flag
EXMC_NAND_PCCARD_INT_FLAG_RISE	rising edge interrupt and flag
EXMC_NAND_PCCARD_INT_FLAG_LEVEL	high-level interrupt and flag
EXMC_NAND_PCCARD_INT_FLAG_FALL	falling edge interrupt and flag
EXMC_SDRAM_INT_FLAG_REFRESH	refresh error interrupt and flag
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* clear EXMC interrupt flag */

exmc_interrupt_flag_clear(EXMC_BANK1_NAND,
EXMC_NAND_PCCARD_INT_FLAG_RISE);
```

3.13. EXTI

EXTI is the interrupt / event controller in the MCU. It contains up to 25 independent edge

detectors and generates interrupt requests or events to the processor. The EXTI registers are listed in chapter [3.13.1](#), the EXTI firmware functions are introduced in chapter [3.13.2](#)

3.13.1. Descriptions of Peripheral registers

EXTI registers are listed in the table shown as below:

Table 3-441. EXTI Registers

Registers	Descriptions
EXTI_INTEN	interrupt enable register
EXTI_EVEN	event enable register
EXTI_RTEN	rising edge trigger enable register
EXTI_FTEN	falling edge trigger enable register
EXTI_SWIEV	software interrupt event register
EXTI_PD	pending register

3.13.2. Descriptions of Peripheral functions

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

Table 3-442. EXTI firmware function

Function name	Function description
exti_deinit	deinitialize the EXTI
exti_init	initialize EXTI line x
exti_interrupt_enable	enable the interrupts from EXTI line x
exti_interrupt_disable	disable the interrupts from EXTI line x
exti_event_enable	enable the events from EXTI line x
exti_event_disable	disable the events from EXTI line x
exti_software_interrupt_enable	enable the software interrupt event from EXTI line x
exti_software_interrupt_disable	disable the software interrupt event from EXTI line x
exti_flag_get	get EXTI line x interrupt pending flag
exti_flag_clear	clear EXTI line x interrupt pending flag
exti_interrupt_flag_get	get EXTI line x interrupt pending flag
exti_interrupt_flag_clear	clear EXTI line x interrupt pending flag

Enum exti_line_enum

Table 3-443. Enum exti_line_enum

Member name	Function description
EXTI_0	EXTI line 0
EXTI_1	EXTI line 1
EXTI_2	EXTI line 2
EXTI_3	EXTI line 3
EXTI_4	EXTI line 4
EXTI_5	EXTI line 5

Member name	Function description
EXTI_6	EXTI line 6
EXTI_7	EXTI line 7
EXTI_8	EXTI line 8
EXTI_9	EXTI line 9
EXTI_10	EXTI line 10
EXTI_11	EXTI line 11
EXTI_12	EXTI line 12
EXTI_13	EXTI line 13
EXTI_14	EXTI line 14
EXTI_15	EXTI line 15
EXTI_16	EXTI line 16
EXTI_17	EXTI line 17
EXTI_18	EXTI line 18
EXTI_19	EXTI line 19
EXTI_20	EXTI line 20
EXTI_21	EXTI line 21
EXTI_22	EXTI line 22
EXTI_23	EXTI line 23
EXTI_24	EXTI line 24
EXTI_25	EXTI line 25

Enum exti_mode_enum

Table 3-444. Enum exti_mode_enum

Member name	Function description
EXTI_INTERRUPT	EXTI interrupt mode
EXTI_EVENT	EXTI event mode

Enum exti_trig_type_enum

Table 3-445. Enum exti_trig_type_enum

Member name	Function description
EXTI_TRIG_RISING	EXTI rising edge trigger
EXTI_TRIG_FALLING	EXTI falling edge trigger
EXTI_TRIG_BOTH	EXTI rising and falling edge trigger
EXTI_TRIG_NONE	EXTI without rising or falling edge trigger

exti_deinit

The description of exti_deinit is shown as below:

Table 3-446. Function exti_deinit

Function name	exti_deinit
Function prototype	void exti_deinit(void);

Function descriptions	deinitialize the EXTI
Precondition	-
The called functions	-
Input parameter{in}	
-	-
Output parameter{out}	
-	-
Return value	
-	-

Example:

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

```
exti_deinit();
```

exti_init

The description of exti_init is shown as below:

Table 3-447. Function exti_init

Function name	exti_init
Function prototype	void exti_init(exti_line_enum linex, exti_mode_enum mode, exti_trig_type_enum trig_type);
Function descriptions	initialize the EXTI line x
Precondition	-
The called functions	-
Input parameter{in}	
linex	EXTI line x, refer to Table 3-443. Enum exti_line_enum
Input parameter{in}	
mode	EXTI mode, refer to Table 3-444. Enum exti_mode_enum
Input parameter{in}	
trig_type	trigger type, refer to Table 3-445. Enum exti_trig_type_enum
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* 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 3-448. Function exti_interrupt_enable

Function name	exti_interrupt_enable
Function prototype	void exti_interrupt_enable(exti_line_enum linex);
Function descriptions	enable the interrupts from EXTI line x
Precondition	-
The called functions	-
Input parameter{in}	
linex	EXTI line x, refer to Table 3-443. Enum exti_line_enum
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 3-449. Function exti_interrupt_disable

Function name	exti_interrupt_disable
Function prototype	void exti_interrupt_disable(exti_line_enum linex);
Function descriptions	disable the interrupts from EXTI line x
Precondition	-
The called functions	-
Input parameter{in}	
linex	EXTI line x, refer to Table 3-443. Enum exti_line_enum
Output parameter{out}	
-	-
Return value	
-	-

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 3-450. Function exti_event_enable

Function name	exti_event_enable
---------------	-------------------

Function prototype	void exti_event_enable(exti_line_enum linex);
Function descriptions	enable the events from EXTI line x
Precondition	-
The called functions	-
Input parameter{in}	
linex	EXTI line x, refer to Table 3-443. Enum exti_line_enum
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* 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 3-451. Function exti_event_disable

Function name	exti_event_disable
Function prototype	void exti_event_disable(exti_line_enum linex);
Function descriptions	disable the events from EXTI line x
Precondition	-
The called functions	-
Input parameter{in}	
linex	EXTI line x, refer to Table 3-443. Enum exti_line_enum
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* 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 3-452. Function exti_software_interrupt_enable

Function name	exti_software_interrupt_enable
Function prototype	void exti_software_interrupt_enable(exti_line_enum linex);
Function descriptions	enable the software interrupt event from EXTI line x

Precondition	-
The called functions	-
Input parameter{in}	
linex	EXTI line x, refer to Table 3-443. Enum exti_line_enum
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 3-453. Function exti_software_interrupt_disable

Function name	exti_software_interrupt_disable
Function prototype	void exti_software_interrupt_disable(exti_line_enum linex);
Function descriptions	disable the software interrupt event from EXTI line x
Precondition	-
The called functions	-
Input parameter{in}	
linex	EXTI line x, refer to Table 3-443. Enum exti_line_enum
Output parameter{out}	
-	-
Return value	
-	-

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 3-454. Function exti_flag_get

Function name	exti_flag_get
Function prototype	FlagStatus exti_flag_get(exti_line_enum linex);
Function descriptions	get EXTI line x interrupt pending flag
Precondition	-
The called functions	-

Input parameter{in}	
linex	EXTI line x, refer to Table 3-443. Enum exti_line_enum
Output parameter{out}	
-	-
Return value	
FlagStatus	SET or RESET

Example:

```
/* 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 3-455. Function exti_flag_clear

Function name	exti_flag_clear
Function prototype	void exti_flag_clear(exti_line_enum linex);
Function descriptions	clear EXTI line x interrupt pending flag
Precondition	-
The called functions	-
Input parameter{in}	
linex	EXTI line x, refer to Table 3-443. Enum exti_line_enum
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* 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 3-456. Function exti_interrupt_flag_get

Function name	exti_interrupt_flag_get
Function prototype	FlagStatus exti_interrupt_flag_get(exti_line_enum linex);
Function descriptions	get EXTI line x interrupt pending flag
Precondition	-
The called functions	-
Input parameter{in}	
linex	EXTI line x, refer to Table 3-443. Enum exti_line_enum

Output parameter{out}	
-	-
Return value	
FlagStatus	SET or RESET

Example:

```
/* 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 3-457. Function exti_interrupt_flag_clear

Function name	exti_interrupt_flag_clear
Function prototype	void exti_interrupt_flag_clear(exti_line_enum linex);
Function descriptions	clear EXTI line x interrupt pending flag
Precondition	-
The called functions	-
Input parameter{in}	
linex	EXTI line x, refer to Table 3-443. Enum exti_line_enum
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* clear EXTI line 0 interrupt flag status */
```

```
exti_interrupt_flag_clear(EXTI_0);
```

3.14. FMC

There is flash controller, option byte and EFUSE for GD32F527 series. The FMC registers are listed in chapter [3.14.1](#), the FMC firmware functions are introduced in chapter [3.14.2](#).

3.14.1. Descriptions of Peripheral registers

FMC registers are listed in the table shown as below:

Table 3-458. FMC Registers

Registers	Descriptions
FMC_KEY	Unlock key register
FMC_OBKEY	Option byte unlock key register

Registers	Descriptions
FMC_STAT	Status register
FMC_CTL	Control register
FMC_OBCTL0	Option byte control register 0
FMC_OBCTL1	Option byte control register 1
FMC_PECFG	Page erase configuration register
FMC_PEKEY	Unlock page erase key register
FMC_OTP1CFG	OTP1 configuration register
FMC_LDECCADDR0	Load code ECC error address0
FMC_LDECCADDR1	Load code ECC error address1
FMC_LDECCADDR2	Load code ECC error address2
FMC_OBSTAT	Option bytes status register
FMC_PID	Product ID register
EFUSE_CS	Efuse control and status register
EFUSE_ADDR	Efuse address register
EFUSE_CTL	Efuse control register
EFUSE_USER_DATA	Efuse user data register

3.14.2. Descriptions of Peripheral functions

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

Table 3-459. FMC firmware function

Function name	Function description
fmc_unlock	unlock the main FMC operation
fmc_lock	lock the main FMC operation
fmc_page_erase	FMC erase page
fmc_sector_erase	FMC erase sector
fmc_mass_erase	FMC erase whole chip
fmc_bank0_erase	FMC erase whole bank0
fmc_bank1_erase	FMC erase whole bank1
fmc_doubleword_program	FMC program a double word at the corresponding address
fmc_word_program	FMC program a word at the corresponding address
fmc_halfword_program	FMC program a half word at the corresponding address
fmc_byte_program	FMC program a byte at the corresponding address
fmc_nwa_enable	enable no wait time area load after system reset
fmc_nwa_disable	disable no wait time area load after system reset
otp1_read_disable	set OTP1 data block not be read
otp2_rlock_enable	enable read lock block protection for OTP2
ob_unlock	unlock the option byte operation
ob_lock	lock the option byte operation
ob_start	send option byte change command
ob_erase	erase and reset option byte

Function name	Function description
ob_write_protection_enable	enable write protect
ob_write_protection_disable	disable write protect
ob_drp_enable	enable erase/program protection and CBUS read protection
ob_drp_disable	disable erase/program protection and CBUS read protection
ob_security_protection_config	set the option byte security protection level
ob_user_write	write the FMC option byte user
ob_user_bor_threshold	option byte BOR threshold value
ob_boot_mode_config	configure the boot mode
ob_ecc_config	configure FMC/SRAM ECC checking
ob_nwa_select	select no wait time area
ob_user_get	get the FMC option byte user
ob_write_protection0_get	get the FMC option byte write protection
ob_write_protection1_get	get the FMC option byte write protection
ob_drp0_get	get the FMC erase/program protection and CBUS read protection option bytes value
ob_drp1_get	get the FMC erase/program protection and CBUS read protection option bytes value
ob_spc_get	get option byte security protection code value
ob_user_bor_threshold_get	get the FMC threshold value
ob_boot_mode_get	get the boot mode
ob_ecc_get	get FMC/SRAM ECC checking
ob_nwa_get	get no wait time area
fmc_flag_get	get flag set or reset
fmc_flag_clear	clear the FMC pending flag
fmc_interrupt_enable	enable FMC interrupt
fmc_interrupt_disable	disable FMC interrupt
fmc_interrupt_flag_get	get interrupt flag set or reset
fmc_interrupt_flag_clear	clear the FMC pending interrupt flag
fmc_state_get	return the FMC state
fmc_ready_wait	check FMC ready or not
efuse_ctrl_unlock	unlock the EFUSE_CTL register
efuse_ctrl_lock	lock the EFUSE_CTL register
efuse_user_data_unlock	unlock the EFUSE_USER_DATA register
efuse_user_data_lock	lock the EFUSE_USER_DATA register
efuse_read	read EFUSE value
efuse_write	write EFUSE
efuse_control_write	write efuse control parameter
efuse_user_data_write	write user data parameter
efuse_flag_get	check EFUSE flag is set or not
efuse_flag_clear	clear EFUSE pending flag
efuse_interrupt_enable	enable EFUSE interrupt
efuse_interrupt_disable	disable EFUSE interrupt

Function name	Function description
efuse_interrupt_flag_get	check EFUSE interrupt flag is set or not
efuse_interrupt_flag_clear	clear EFUSE pending interrupt flag
efuse_ready_wait	check EFUSE operation ready or not

fmc_state_enum

Table 3-460. fmc_state_enum

enum name	enum description
FMC_READY	the operation has been completed
FMC_BUSY	the operation is in progress
FMC_RDCERR	read CBUS protection error
FMC_PGSERR	program sequence error
FMC_PGMERR	program size not match error
FMC_PGAERR	program alignment error
FMC_WPERR	erase/program protection error
FMC_OPERR	operation error
FMC_LDECCDET	two bits ECC error when load code from flash/OTP1/bootloader
FMC_TOERR	timeout error

efuse_state_enum

Table 3-461. efuse_state_enum

enum name	enum description
EFUSE_READY	the operation has been completed
EFUSE_BUSY	the operation is in progress
EFUSE_OBER	overstep boundary error
EFUSE_TOERR	timeout error

fmc_unlock

The description of fmc_unlock is shown as below:

Table 3-462. Function fmc_unlock

Function name	fmc_unlock
Function prototype	void fmc_unlock (void);
Function descriptions	unlock the main FMC operation
Precondition	-
The called functions	-
Input parameter{in}	
-	-
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* unlock the main FMC operation */

fmc_unlock( );
```

fmc_lock

The description of fmc_lock is shown as below:

Table 3-463. Function fmc_lock

Function name	fmc_lock
Function prototype	void fmc_lock(void);
Function descriptions	lock the main FMC operation
Precondition	-
The called functions	-
Input parameter{in}	
-	-
Output parameter{out}	
-	-
Return value	
-	-

Example:

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

fmc_lock( );
```

fmc_page_erase

The description of fmc_page_erase is shown as below:

Table 3-464. Function fmc_page_erase

Function name	fmc_page_erase
Function prototype	fmc_state_enum fmc_page_erase(uint32_t page_addr);
Function descriptions	erase page
Precondition	fmc_unlock
The called functions	fmc_ready_wait
Input parameter{in}	
page_addr	page address
Output parameter{out}	
-	-
Return value	
fmc_state_enum	state of FMC, refer to fmc_state_enum

Example:

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

```
fmc_unlock();

fmc_state_enum state = FMC_READY;

state = fmc_page_erase(0x08008000);
```

fmc_sector_erase

The description of fmc_sector_erase is shown as below:

Table 3-465. Function fmc_sector_erase

Function name	fmc_sector_erase
Function prototype	fmc_state_enum fmc_sector_erase(uint32_t fmc_sector);
Function descriptions	erase sector
Precondition	fmc_unlock
The called functions	fmc_ready_wait
Input parameter{in}	
fmc_sector	select the sector to erase
CTL_SECTOR_NUMB ER_x	sector X(X = 0~53)
Output parameter{out}	
-	-
Return value	
fmc_state_enum	state of FMC, refer to fmc_state_enum

Example:

```
/* erase number 15 sector */

fmc_unlock();

fmc_state_enum state = FMC_READY;

state = fmc_sector_erase(CTL_SECTOR_NUMBER_15);
```

fmc_mass_erase

The description of fmc_mass_erase is shown as below:

Table 3-466. Function fmc_mass_erase

Function name	fmc_mass_erase
Function prototype	fmc_state_enum fmc_mass_erase(void);
Function descriptions	erase whole chip
Precondition	fmc_unlock
The called functions	fmc_ready_wait
Input parameter{in}	
-	-
Output parameter{out}	
-	-

Return value	
fmc_state_enum	state of FMC, refer to fmc_state_enum

Example:

```
/* erase the whole chip */

fmc_unlock( );

fmc_state_enum state = FMC_READY;

state = fmc_mass_erase();
```

fmc_bank0_erase

The description of fmc_bank0_erase is shown as below:

Table 3-467. Function fmc_bank0_erase

Function name	fmc_bank0_erase
Function prototype	fmc_state_enum fmc_bank0_erase(void)
Function descriptions	erase all FMC sectors in bank0
Precondition	fmc_unlock
The called functions	fmc_ready_wait
Input parameter{in}	
-	-
Output parameter{out}	
-	-
Return value	
fmc_state_enum	state of FMC, refer to fmc_state_enum

Example:

```
/* erase bank0 */

fmc_unlock( );

fmc_state_enum state = FMC_READY;

state = fmc_bank0_erase( );
```

fmc_bank1_erase

The description of fmc_bank1_erase is shown as below:

Table 3-468. Function fmc_bank1_erase

Function name	fmc_bank1_erase
Function prototype	fmc_state_enum fmc_bank1_erase(void)
Function descriptions	erase all FMC sectors in bank1 (include bank1_ex)
Precondition	fmc_unlock
The called functions	fmc_ready_wait

Input parameter{in}	
-	-
Output parameter{out}	
-	-
Return value	
fmc_state_enum	state of FMC, refer to fmc_state_enum

Example:

```
/* erase bank1 */

fmc_unlock( );

fmc_state_enum state = FMC_READY;

state = fmc_bank1_erase( );
```

fmc_doubleword_program

The description of fmc_doubleword_program is shown as below:

Table 3-469. Function fmc_doubleword_program

Function name	fmc_doubleword_program
Function prototype	fmc_state_enum fmc_doubleword_program(uint32_t address, uint64_t data)
Function descriptions	program a doubleword at the corresponding address
Precondition	fmc_unlock
The called functions	fmc_ready_wait
Input parameter{in}	
address	the address to program
data	doubleword to program(0x0000000000000000 - 0xFFFFFFFFFFFFFFFF)
Output parameter{out}	
-	-
Return value	
fmc_state_enum	state of FMC, refer to fmc_state_enum

Example:

```
fmc_unlock( );

fmc_state_enum state = FMC_READY;

state = fmc_doubleword_program(0x08004000, 0x1234567812345678);
```

fmc_word_program

The description of fmc_word_program is shown as below:

Table 3-470. Function fmc_word_program

Function name	fmc_word_program
Function prototype	fmc_state_enum fmc_word_program(uint32_t address, uint32_t data);

Function descriptions	program a word at the corresponding address
Precondition	fmc_unlock
The called functions	fmc_ready_wait
Input parameter{in}	
address	the address to program
data	word to program(0x00000000 - 0xFFFFFFFF)
Output parameter{out}	
-	-
Return value	
fmc_state_enum	state of FMC, refer to fmc_state_enum

Example:

```
fmc_unlock( );
```

```
fmc_state_enum state = FMC_READY;
```

```
state = fmc_word_program(0x08004000, 0x12345678);
```

fmc_halfword_program

The description of fmc_halfword_program is shown as below:

Table 3-471. Function fmc_halfword_program

Function name	fmc_halfword_program
Function prototype	fmc_state_enum fmc_halfword_program(uint32_t address, uint16_t data);
Function descriptions	program a halfword at the corresponding address
Precondition	fmc_unlock
The called functions	fmc_ready_wait
Input parameter{in}	
address	the address to program
data	halfword to program(0x0000 - 0xFFFF)
Output parameter{out}	
-	-
Return value	
fmc_state_enum	state of FMC, refer to fmc_state_enum

Example:

```
/* program half word at the corresponding address */
```

```
fmc_unlock( );
```

```
fmc_state_enum state = FMC_READY;
```

```
state = fmc_halfword_program (0x08004000, 0x1234);
```

fmc_byte_program

The description of fmc_byte_program is shown as below:

Table 3-472. Function fmc_halfword_program

Function name	fmc_halfword_program
Function prototype	fmc_state_enum fmc_byte_program(uint32_t address, uint8_t data);
Function descriptions	program a byte at the corresponding address
Precondition	fmc_unlock
The called functions	fmc_ready_wait
Input parameter{in}	
address	the address to program
data	byte to program(0x00 - 0xFF)
Output parameter{out}	
-	-
Return value	
fmc_state_enum	state of FMC, refer to fmc_state_enum

Example:

```
/* program a byte at the corresponding address */
```

```
fmc_unlock( );
```

```
fmc_state_enum state = FMC_READY;
```

```
state = fmc_byte_program (0x08004000, 0x12);
```

fmc_nwa_enable

The description of fmc_nwa_enable is shown as below:

Table 3-473. Function fmc_nwa_enable

Function name	fmc_nwa_enable
Function prototype	void fmc_nwa_enable(void);
Function descriptions	enable no wait time area load after system reset
Precondition	fmc_unlock
The called functions	-
Input parameter{in}	
-	-
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* enable no wait time area load after system reset */
```

```
fmc_unlock();
```

```
fmc_nwa_enable( );
```

fmc_nwa_disable

The description of fmc_nwa_disable is shown as below:

Table 3-474. Function fmc_nwa_disable

Function name	fmc_nwa_disable
Function prototype	void fmc_nwa_disable(void);
Function descriptions	disable no wait time area load after system reset
Precondition	fmc_unlock
The called functions	-
Input parameter{in}	
-	-
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* disable no wait time area load after system reset */
```

```
fmc_unlock();
```

```
fmc_nwa_disable( );
```

otp1_read_disable

The description of otp1_read_disable is shown as below:

Table 3-475. Function otp1_read_disable

Function name	otp1_read_disable
Function prototype	void otp1_read_disable(void);
Function descriptions	set OTP1 data block not be read
Precondition	-
The called functions	-
Input parameter{in}	
block	specify OTP1 data block x not be read
OTP1_DATA_BLOCK_ x	data block x(x = 0,1,2...15)
OTP1_DATA_BLOCK_ ALL	all data block
Output parameter{out}	
-	-
Return value	

-	-
---	---

Example:

```
/* set OTP1 data block0 not be read */
otp1_read_disable(OTP1_DATA_BLOCK_0);
```

otp2_rlock_enable

The description of otp2_rlock_enable is shown as below:

Table 3-476. Function otp2_rlock_enable

Function name	otp2_rlock_enable
Function prototype	void otp2_rlock_enable(void);
Function descriptions	enable read lock block protection for OTP2
Precondition	fmc_unlock
The called functions	-
Input parameter{in}	
-	-
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* enable read lock block protection for OTP2 */
fmc_unlock();
otp2_rlock_enable( );
```

ob_unlock

The description of ob_unlock is shown as below:

Table 3-477. Function ob_unlock

Function name	ob_unlock
Function prototype	void ob_unlock(void);
Function descriptions	unlock the option byte operation
Precondition	-
The called functions	-
Input parameter{in}	
-	-
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* unlock the option byte operation */
ob_unlock( );
```

ob_lock

The description of ob_lock is shown as below:

Table 3-478. Function ob_lock

Function name	ob_lock
Function prototype	void ob_lock(void);
Function descriptions	lock the option byte operation
Precondition	-
The called functions	-
Input parameter{in}	
-	-
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* lock the option byte operation */
ob_lock( );
```

ob_start

The description of ob_start is shown as below:

Table 3-479. Function ob_start

Function name	ob_start
Function prototype	void ob_start(void);
Function descriptions	send option byte change command
Precondition	ob_unlock
The called functions	fmc_ready_wait
Input parameter{in}	
-	-
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* enable write protection and then send the start command */
```

```
ob_unlock( );

ob_write_protection_enable (OB_WP7);

ob_start();
```

ob_erase

The description of ob_erase is shown as below:

Table 3-480. Function ob_erase

Function name	ob_erase
Function prototype	void ob_erase(void);
Function descriptions	erase and reset option byte
Precondition	ob_unlock
The called functions	fmc_ready_wait
Input parameter{in}	
-	-
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* erase and reset option byte */

ob_unlock( );

ob_erase( );

ob_start( );

ob_lock( );
```

ob_write_protection_enable

The description of ob_write_protection_enable is shown as below:

Table 3-481. Function ob_write_protection_enable

Function name	ob_write_protection_enable
Function prototype	ErrStatus ob_write_protection_enable(uint32_t ob_wp);
Function descriptions	enable write protection
Precondition	ob_unlock
The called functions	fmc_ready_wait
Input parameter{in}	
ob_wp	specify sector to be write protected
<i>OB_WP_x</i>	sector x(x = 0,1,2...22)
<i>OB_WP_23_27</i>	sector23~27

<i>OB_WP_ALL</i>	all sector
Output parameter{out}	
-	-
Return value	
ErrStatus	SUCCESS or ERROR

Example:

```
/* enable write protection of sector 7 */
```

```
ErrStatus temp = ERROR;
```

```
ob_unlock( );
```

```
temp = ob_write_protection_enable (OB_WP7);
```

```
ob_start();
```

```
ob_lock( );
```

ob_write_protection_disable

The description of ob_write_protection_disable is shown as below:

Table 3-482. Function ob_write_protection_disable

Function name	ob_write_protection_disable
Function prototype	ErrStatus ob_write_protection_disable(uint32_t ob_wp);
Function descriptions	disable write protection
Precondition	ob_unlock
The called functions	fmc_ready_wait
Input parameter{in}	
ob_wp	specify sector to be write protected
<i>OB_WP_x</i>	sector x(x = 0,1,2...22)
<i>OB_WP_23_27</i>	sector23~27
<i>OB_WP_ALL</i>	all sector
Output parameter{out}	
-	-
Return value	
ErrStatus	SUCCESS or ERROR

Example:

```
/* disable write protection of sector 7 */
```

```
ErrStatus temp = ERROR;
```

```
ob_unlock( );
```

```
temp = ob_write_protection_disable (OB_WP7);
```

```
ob_start();
```



```
ob_lock( );
```

ob_drp_enable

The description of ob_drp_enable is shown as below:

Table 3-483. Function ob_drp_enable

Function name	ob_drp_enable
Function prototype	void ob_drp_enable(uint32_t ob_drp);
Function descriptions	enable erase/program protection and CBUS read protection
Precondition	ob_unlock
The called functions	fmc_ready_wait
Input parameter{in}	
ob_drp	enable the WPx bits used as erase/program protection and CBUS read protection of each sector
<i>OB_DRP_x</i>	sector x(x = 0,1,2...38)
<i>OB_DRP_39_53</i>	sector39~53
<i>OB_DRP_ALL</i>	all sector
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* enable write protection and CBUS read protectio of sector 7 */
```

```
ob_unlock( );
```

```
ob_drp_enable (OB_DRP_7);
```

```
ob_start();
```

```
ob_lock( );
```

ob_drp_disable

The description of ob_drp_disable is shown as below:

Table 3-484. Function ob_drp_disable

Function name	ob_drp_disable
Function prototype	void ob_drp_disable(void);
Function descriptions	disable all erase/program protection and CBUS read protection
Precondition	ob_unlock
The called functions	fmc_ready_wait
Input parameter{in}	
-	-
Output parameter{out}	

-	-
Return value	
-	-

Example:

```
/* disable all erase/program protection and CBUS read protection */
```

```
ob_unlock( );
```

```
ob_drp_disable();
```

```
ob_start();
```

```
ob_lock( );
```

ob_security_protection_config

The description of ob_security_protection_config is shown as below:

Table 3-485. Function ob_security_protection_config

Function name	ob_security_protection_config
Function prototype	void ob_security_protection_config(uint8_t ob_spc);
Function descriptions	configure security protection level
Precondition	ob_unlock
The called functions	fmc_ready_wait
Input parameter{in}	
ob_spc	specify security protection
<i>FMC_NSPC</i>	no security protection
<i>FMC_LSPC</i>	low security protection
<i>FMC_HSPC</i>	high security protection
Output parameter{out}	
-	-
Return value	
-	state of FMC, refer to fmc_state_enum

Example:

```
/* config FMC as low security protection */
```

```
ob_unlock( );
```

```
ob_security_protection_config (FMC_LSPC);
```

```
ob_start( );
```

```
ob_lock( );
```

ob_user_write

The description of ob_user_write is shown as below:

Table 3-486. Function ob_user_write

Function name	ob_user_write
Function prototype	void ob_user_write(uint32_t ob_fwdgt, uint32_t ob_deepsleep, uint32_t ob_stdby);
Function descriptions	program the FMC user option byte
Precondition	ob_unlock
The called functions	fmc_ready_wait
Input parameter{in}	
ob_fwdgt	option byte watchdog value
OB_FWDGT_SW	software free watchdog
OB_FWDGT_HW	hardware free watchdog
Input parameter{in}	
ob_deepsleep	option byte deepsleep reset value
OB_DEEPSLEEP_NRS T	no reset when entering deepsleep mode
OB_DEEPSLEEP_RST	generate a reset instead of entering deepsleep mode
Input parameter{in}	
ob_stdby	option byte standby reset value
OB_STDBY_NRS	no reset when entering standby mode
OB_STDBY_RST	generate a reset instead of entering standby mode
Output parameter{out}	
-	-
Return value	
-	state of FMC, refer to fmc_state_enum

Example:

```

/* config the USER byte of option byte */

ob_unlock( );

ob_user_write(OB_FWDGT_SW, OB_DEEPSLEEP_NRS, OB_STDBY_NRS);

ob_start ( );

ob_lock( );

```

ob_user_bor_threshold

The description of ob_user_bor_threshold is shown as below:

Table 3-487. Function ob_user_bor_threshold

Function name	ob_user_bor_threshold
Function prototype	void ob_user_bor_threshold(uint32_t ob_bor_th);
Function descriptions	program the option byte BOR threshold value
Precondition	ob_unlock
The called functions	-

Input parameter{in}	
ob_bor_th	user option byte
<i>OB_BOR_TH_VALUE3</i>	BOR threshold value 3
<i>OB_BOR_TH_VALUE2</i>	BOR threshold value 2
<i>OB_BOR_TH_VALUE1</i>	BOR threshold value 1
<i>OB_BOR_TH_OFF</i>	no BOR function
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* config the BOR threshold value as OB_BOR_TH_OFF */
```

```
ob_unlock( );
```

```
ob_user_bor_threshold(OB_BOR_TH_OFF);
```

```
ob_start( );
```

```
ob_lock( );
```

ob_boot_mode_config

The description of ob_boot_mode_config is shown as below:

Table 3-488. Function ob_boot_mode_config

Function name	ob_boot_mode_config
Function prototype	void ob_boot_mode_config(uint32_t boot_mode);
Function descriptions	configure the option byte boot bank value
Precondition	ob_unlock
The called functions	-
Input parameter{in}	
boot_mode	specifies the option byte boot bank value
<i>OB_BB_DISABLE</i>	boot from bank0
<i>OB_BB_ENABLE</i>	boot from bank1 or bank0 if bank1 is void
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* config boot from bank0 */
```

```
ob_unlock( );
```

```
ob_boot_mode_config(OB_BB_DISABLE);
```

```
ob_start( );
```

```
ob_lock( );
```

ob_ecc_config

The description of ob_ecc_config is shown as below:

Table 3-489. Function ob_ecc_config

Function name	ob_ecc_config
Function prototype	void ob_ecc_config(uint32_t ecc_config)
Function descriptions	configure FMC/SRAM ECC checking, only valid after power reset
Precondition	ob_unlock
The called functions	-
Input parameter{in}	
ecc_config	specifies the option byte FMC/SRAM ECC checking
<i>OB_ECC_DISABLE</i>	disable FMC/SRAM ECC checking
<i>OB_ECC_ENABLE</i>	enable FMC/SRAM ECC checking
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* disable ECCchecking */
ob_unlock( );
ob_ecc_config(OB_ECC_DISABLE);
ob_start( );
ob_lock( );
```

ob_nwa_select

The description of ob_nwa_select is shown as below:

Table 3-490. Function ob_nwa_select

Function name	ob_nwa_select
Function prototype	void ob_nwa_select(uint32_t nwa_select)
Function descriptions	select no wait time area
Precondition	ob_unlock
The called functions	-
Input parameter{in}	
nwa_select	specifies the option byte no wait time area
<i>OB_NWA_BANK1</i>	bank1 is no wait time area
<i>OB_NWA_BANK0</i>	bank0 is no wait time area

Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* select bank1 as no wait time area*/
```

```
ob_unlock( );
```

```
ob_nwa_select(OB_NWA_BANK1);
```

```
ob_start( );
```

```
ob_lock( );
```

ob_user_get

The description of ob_user_get is shown as below:

Table 3-491. Function ob_user_get

Function name	ob_user_get
Function prototype	uint8_t ob_user_get(void);
Function descriptions	get the FMC user option byte
Precondition	-
The called functions	-
Input parameter{in}	
-	-
Output parameter{out}	
-	-
Return value	
uint8_t	the FMC user option byte values: ob_fwdgt(Bit0), ob_deepsleep(Bit1), ob_stdby(Bit2)

Example:

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

```
uint8_t ob_user_values = 0;
```

```
ob_user_values = ob_user_get( );
```

ob_write_protection0_get

The description of ob_write_protection0_get is shown as below:

Table 3-492. Function ob_write_protection0_get

Function name	ob_write_protection0_get
Function prototype	uint16_t ob_write_protection0_get(void);

Function descriptions	get the FMC option byte write protection of bank0
Precondition	-
The called functions	-
Input parameter{in}	
-	-
Output parameter{out}	
-	-
Return value	
uint16_t	the FMC write protection option byte value

Example:

```
/* get the FMC write protection option byte value */
uint16_t bank0_protection_value = 0;
bank0_protection_value = ob_write_protection0_get( );
```

ob_write_protection1_get

The description of ob_write_protection1_get is shown as below:

Table 3-493. Function ob_write_protection1_get

Function name	ob_write_protection1_get
Function prototype	uint16_t ob_write_protection1_get(void);
Function descriptions	get the FMC option byte write protection of bank1
Precondition	-
The called functions	-
Input parameter{in}	
-	-
Output parameter{out}	
-	-
Return value	
uint16_t	the FMC write protection option byte value

Example:

```
/* get the FMC write protection option byte value */
uint16_t bank1_protection_value = 0;
bank1_protection_value = ob_write_protection1_get( );
```

ob_drp0_get

The description of ob_drp0_get is shown as below:

Table 3-494. Function ob_drp0_get

Function name	ob_drp0_get
----------------------	-------------

Function prototype	uint16_t ob_drp0_get(void);
Function descriptions	get the FMC CBUS read protection protection of bank0
Precondition	-
The called functions	-
Input parameter{in}	
-	-
Output parameter{out}	
-	-
Return value	
uint16_t	the FMC erase/program protection and CBUS read protection option bytes value

Example:

```
/* get the FMC erase/program protection and CBUS read protection option bytes value */
uint16_t bank0_drp_value = 0;
bank0_drp_value = ob_drp0_get( );
```

ob_drp1_get

The description of ob_drp1_get is shown as below:

Table 3-495. Function ob_drp0_get

Function name	ob_drp1_get
Function prototype	uint16_t ob_drp1_get(void);
Function descriptions	get the FMC CBUS read protection protection of bank1
Precondition	-
The called functions	-
Input parameter{in}	
-	-
Output parameter{out}	
-	-
Return value	
uint16_t	the FMC erase/program protection and CBUS read protection option bytes value

Example:

```
/* get the FMC erase/program protection and CBUS read protection option bytes value */
uint16_t bank1_drp_value = 0;
bank1_drp_value = ob_drp1_get( );
```

ob_spc_get

The description of ob_spc_get is shown as below:

Table 3-496. Function ob_spc_get

Function name	ob_spc_get
Function prototype	FlagStatus ob_spc_get(void);
Function descriptions	get the FMC option byte security protection
Precondition	-
The called functions	-
Input parameter{in}	
-	-
Output parameter{out}	
-	-
Return value	
FlagStatus	SET or RESET

Example:

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

```
FlagStatus spc_value = RESET;
```

```
spc_value = ob_spc_get();
```

ob_user_bor_threshold_get

The description of ob_user_bor_threshold_get is shown as below:

Table 3-497. Function ob_user_bor_threshold_get

Function name	ob_user_bor_threshold_get
Function prototype	uint8_t ob_user_bor_threshold_get(void);
Function descriptions	get the FMC option byte BOR threshold value
Precondition	-
The called functions	-
Input parameter{in}	
-	-
Output parameter{out}	
-	-
Return value	
uint8_t	the FMC BOR threshold value:OB_BOR_TH_OFF,OB_BOR_TH_VALUE1,OB_BOR_TH_VALUE2, OB_BOR_TH_VALUE3

Example:

```
/* get the FMC option byte BOR threshold value */
```

```
uint8_t bor_value = 0;
```

```
bor_value = ob_user_bor_threshold_get( );
```

ob_boot_mode_get

The description of ob_boot_mode_get is shown as below:

Table 3-498. Function ob_boot_mode_get

Function name	ob_boot_mode_get
Function prototype	uint32_t ob_boot_mode_get(void)
Function descriptions	get the boot mode
Precondition	-
The called functions	-
Input parameter{in}	
-	-
Output parameter{out}	
-	-
Return value	
uint32_t	the FMC boot bank value:OB_BB_DISABLE,OB_BB_ENABLE

Example:

```
/* get the boot mode */
uint32_t bb_value = 0;
bb_value = ob_boot_mode_get( );
```

ob_ecc_get

The description of ob_ecc_get is shown as below:

Table 3-499. Function ob_ecc_get

Function name	ob_ecc_get
Function prototype	uint32_t ob_ecc_get(void)
Function descriptions	get FMC/SRAM ECC checking
Precondition	-
The called functions	-
Input parameter{in}	
-	-
Output parameter{out}	
-	-
Return value	
uint32_t	the FMC ECCEN value:OB_ECC_DISABLE,OB_ECC_ENABLE

Example:

```
/* get FMC/SRAM ECC checking */
uint32_t ecc_value = 0;
ecc_value = ob_ecc_get( );
```

ob_nwa_get

The description of ob_nwa_get is shown as below:

Table 3-500. Function ob_nwa_get

Function name	ob_nwa_get
Function prototype	uint32_t ob_nwa_get(void)
Function descriptions	get no wait time area
Precondition	-
The called functions	-
Input parameter{in}	
-	-
Output parameter{out}	
-	-
Return value	
uint32_t	the no wait time area:OB_NWA_BANK1,OB_NWA_BANK0

Example:

```
/* get no wait time area */
uint32_t nwa_value = 0;
nwa_value = ob_nwa_get( );
```

fmc_flag_get

The description of fmc_flag_get is shown as below:

Table 3-501. Function fmc_flag_get

Function name	fmc_flag_get
Function prototype	FlagStatus fmc_flag_get(uint32_t flag);
Function descriptions	check flag is set or not
Precondition	-
The called functions	-
Input parameter{in}	
flag	check FMC flag
<i>FMC_FLAG_BUSY</i>	FMC busy flag bit
<i>FMC_FLAG_RDCERR</i>	FMC read CBUS protection error flag bit
<i>FMC_FLAG_PGSERR</i>	FMC program sequence error flag bit
<i>FMC_FLAG_PGMERR</i>	FMC program size not match error flag bit
<i>FMC_FLAG_PGAERR</i>	FMC program alignment error flag bit
<i>FMC_FLAG_WPERR</i>	FMC Erase/Program protection error flag bit
<i>FMC_FLAG_OPERR</i>	FMC operation error flag bit
<i>FMC_FLAG_END</i>	FMC end of operation flag bit
Output parameter{out}	
-	-

Return value	
FlagStatus	SET or RESET

Example:

```
/* check the FMC_FLAG_END flag set or not*/
```

```
FlagStatus flag = RESET;
```

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

fmc_flag_clear

The description of fmc_flag_clear is shown as below:

Table 3-502. Function fmc_flag_clear

Function name	fmc_flag_clear
Function prototype	void fmc_flag_clear(uint32_t flag);
Function descriptions	clear the FMC flag
Precondition	-
The called functions	-
Input parameter{in}	
flag	clear FMC flag
FMC_FLAG_BUSY	FMC busy flag bit
FMC_FLAG_RDCERR	FMC read CBUS protection error flag bit
FMC_FLAG_PGSERR	FMC program sequence error flag bit
FMC_FLAG_PGMERR	FMC program size not match error flag bit
FMC_FLAG_PGAERR	FMC program alignment error flag bit
FMC_FLAG_WPERR	FMC Erase/Program protection error flag bit
FMC_FLAG_OPERR	FMC operation error flag bit
FMC_FLAG_END	FMC end of operation flag bit
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* clear the FMC_FLAG_END flag */
```

```
fmc_flag_clear(FMC_FLAG_END);
```

fmc_interrupt_enable

The description of fmc_interrupt_enable is shown as below:

Table 3-503. Function fmc_interrupt_enable

Function name	fmc_interrupt_enable
Function prototype	void fmc_interrupt_enable(uint32_t interrupt);

Function descriptions	enable FMC interrupt
Precondition	-
The called functions	-
Input parameter{in}	
interrupt	the FMC interrupt source
<i>FMC_INT_END</i>	enable FMC end of program interrupt
<i>FMC_INT_ERR</i>	enable FMC error interrupt
Output parameter{out}	
-	-
Return value	
-	-

Example:

```

/* enable FMC end of program interrupt */

fmc_unlock( );

fmc_interrupt_enable(FMC_INT_END);

fmc_lock( );

```

fmc_interrupt_disable

The description of fmc_interrupt_disable is shown as below:

Table 3-504. Function fmc_interrupt_disable

Function name	fmc_interrupt_disable
Function prototype	void fmc_interrupt_disable(uint32_t interrupt);
Function descriptions	disable FMC interrupt
Precondition	-
The called functions	-
Input parameter{in}	
interrupt	the FMC interrupt source
<i>FMC_INT_END</i>	disable FMC end of program interrupt
<i>FMC_INT_ERR</i>	disable FMC error interrupt
Output parameter{out}	
-	-
Return value	
-	-

Example:

```

/* disable FMC interrupt */

fmc_interrupt_disable(FMC_INT_END);

```

fmc_interrupt_flag_get

The description of fmc_interrupt_flag_get is shown as below:

Table 3-505. Function fmc_interrupt_flag_get

Function name	fmc_interrupt_flag_get
Function prototype	FlagStatus fmc_interrupt_flag_get(uint32_t int_flag);
Function descriptions	check interrupt flag is set or not
Precondition	-
The called functions	-
Input parameter{in}	
int_flag	check FMC flag
<i>FMC_INT_FLAG_RDCERR</i>	FMC read CBUS protection error interrupt flag bit
<i>FMC_INT_FLAG_PGSE</i>	FMC program sequence error interrupt flag bit
<i>FMC_INT_FLAG_PGMERR</i>	FMC program size not match error interrupt flag bit
<i>FMC_INT_FLAG_PGAERR</i>	FMC program alignment error interrupt flag bit
<i>FMC_INT_FLAG_WPERR</i>	FMC Erase/Program protection error interrupt flag bit
<i>FMC_INT_FLAG_OPERR</i>	FMC operation error interrupt flag bit
<i>FMC_INT_FLAG_END</i>	FMC end of operation interrupt flag bit
Output parameter{out}	
-	-
Return value	
FlagStatus	SET or RESET

Example:

```
/* check the FMC_INT_FLAG_END flag set or not*/
```

```
FlagStatus int_flag = RESET;
```

```
int_flag = fmc_interrupt_flag_get(FMC_INT_FLAG_END);
```

fmc_interrupt_flag_clear

The description of fmc_interrupt_flag_clear is shown as below:

Table 3-506. Function fmc_interrupt_flag_clear

Function name	fmc_interrupt_flag_clear
Function prototype	void fmc_interrupt_flag_clear(uint32_t int_flag);
Function descriptions	clear the FMC interrupt flag
Precondition	-

The called functions	-
Input parameter{in}	
int_flag	clear FMC interrupt flag
<i>FMC_INT_FLAG_RDCERR</i>	FMC read CBUS protection error interrupt flag bit
<i>FMC_INT_FLAG_PGSEERR</i>	FMC program sequence error interrupt flag bit
<i>FMC_INT_FLAG_PGMERR</i>	FMC program size not match error interrupt flag bit
<i>FMC_INT_FLAG_PGAERR</i>	FMC program alignment error interrupt flag bit
<i>FMC_INT_FLAG_WPEERR</i>	FMC Erase/Program protection error interrupt flag bit
<i>FMC_INT_FLAG_OPEERR</i>	FMC operation error interrupt flag bit
<i>FMC_INT_FLAG_END</i>	FMC end of operation interrupt flag bit
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* clear the FMC_INT_FLAG_END flag */
fmc_interrupt_flag_clear(FMC_INT_FLAG_END);
```

fmc_state_get

The description of fmc_state_get is shown as below:

Table 3-507. Function fmc_state_get

Function name	fmc_state_get
Function prototype	fmc_state_enum fmc_state_get(void);
Function descriptions	get the FMC state
Precondition	-
The called functions	-
Input parameter{in}	
-	-
Output parameter{out}	
-	-
Return value	
fmc_state_enum	state of FMC, refer to fmc_state_enum

Example:

```
/* get the FMC state */
```

```
fmc_state_enum state = fmc_state_get( );
```

fmc_ready_wait

The description of fmc_ready_wait is shown as below:

Table 3-508. Function fmc_ready_wait

Function name	fmc_ready_wait
Function prototype	fmc_state_enum fmc_ready_wait(uint32_t timeout)
Function descriptions	check whether FMC is ready or not
Precondition	-
The called functions	fmc_state_get()
Input parameter{in}	
timeout	timeout value
Output parameter{out}	
-	-
Return value	
fmc_state_enum	state of FMC, refer to fmc_state_enum

Example:

```
/* check whether FMC is ready or not */
```

```
fmc_state_enum state = fmc_ready_wait(FMC_TIMEOUT_COUNT);
```

efuse_ctrl_unlock

The description of efuse_ctrl_unlock is shown as below:

Table 3-509. Function ob_unlock

Function name	efuse_ctrl_unlock
Function prototype	void efuse_ctrl_unlock(void);
Function descriptions	unlock the EFUSE_CTL register
Precondition	-
The called functions	-
Input parameter{in}	
-	-
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* unlock the EFUSE_CTL register */
```

```
efuse_ctrl_unlock( );
```


efuse_ctrl_lock

The description of efuse_ctrl_lock is shown as below:

Table 3-510. Function ob_lock

Function name	efuse_ctrl_lock
Function prototype	void efuse_ctrl_lock(void);
Function descriptions	lock the EFUSE_CTL register
Precondition	-
The called functions	-
Input parameter{in}	
-	-
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* lock the EFUSE_CTL register */
efuse_ctrl_lock( );
```

efuse_user_data_unlock

The description of efuse_user_data_unlock is shown as below:

Table 3-511. Function efuse_user_data_unlock

Function name	efuse_user_data_unlock
Function prototype	void efuse_user_data_unlock(void);
Function descriptions	unlock the EFUSE_USER_DATA register
Precondition	-
The called functions	-
Input parameter{in}	
-	-
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* unlock the EFUSE_USER_DATA register */
efuse_user_data_unlock( );
```

efuse_user_data_lock

The description of efuse_user_data_lock is shown as below:

Table 3-512. Function efuse_user_data_lock

Function name	efuse_user_data_lock
Function prototype	void efuse_user_data_lock(void);
Function descriptions	lock the EFUSE_USER_DATA register
Precondition	-
The called functions	-
Input parameter{in}	
-	-
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* lock the EFUSE_USER_DATA register */
efuse_user_data_lock( );
```

efuse_read

The description of efuse_read is shown as below:

Table 3-513. Function efuse_read

Function name	efuse_read
Function prototype	efuse_state_enum efuse_read(uint32_t ef_addr, uint32_t size, uint32_t buf[]);
Function descriptions	read EFUSE value
Precondition	-
The called functions	-
Input parameter{in}	
ef_addr	EFUSE address: EFUSE_CTL_EFADDR, USER_DATA_EFADDR
Input parameter{in}	
size	size of EFUSE (0x01 bytes)
Output parameter{out}	
buf	the buffer for storing read-out EFUSE register value
Return value	
efuse_state_enum	state of EFUSE, refer to efuse_state_enum

Example:

```
/* read EFUSE USER DATA value */
uint32_t buffer[1] = {0};
efuse_state_enum flag = EFUSE_READY;
flag = efuse_read(USER_DATA_EFADDR, 1, buffer);
```

efuse_write

The description of efuse_write is shown as below:

Table 3-514. Function efuse_write

Function name	efuse_write
Function prototype	efuse_state_enum efuse_write(uint32_t ef_addr, uint32_t size, uint8_t ef_data)
Function descriptions	write EFUSE
Precondition	-
The called functions	-
Input parameter{in}	
ef_addr	EFUSE address: EFUSE_CTL_EFADDR, USER_DATA_EFADDR
Input parameter{in}	
size	size of EFUSE (0x01 bytes)
Output parameter{out}	
buf	the buffer of data written to EFUSE
Return value	
efuse_state_enum	state of EFUSE, refer to efuse_state_enum

Example:

```
/* write EFUSE USER DATA*/

uint8_t buffer = 0x11;

efuse_state_enum flag = EFUSE_READY;

flag = efuse_write(USER_DATA_EFADDR, 1, buffer);
```

efuse_control_write

The description of efuse_control_write is shown as below:

Table 3-515. Function efuse_control_write

Function name	efuse_control_write
Function prototype	efuse_state_enum efuse_control_write(uint8_t ef_data)
Function descriptions	write control data
Precondition	-
The called functions	efuse_write
Input parameter{in}	
ef_data	EFUSE data to write
Output parameter{out}	
-	-
Return value	
efuse_state_enum	state of EFUSE, refer to efuse_state_enum

Example:

```

/* write control data */

uint8_t ctl = 0x01;

efuse_state_enum flag = EFUSE_READY;

flag = efuse_control_write(ctl);

```

efuse_user_data_write

The description of efuse_user_data_write is shown as below:

Table 3-516. Function efuse_user_data_write

Function name	efuse_user_data_write
Function prototype	efuse_state_enum efuse_user_data_write(uint8_t ef_data)
Function descriptions	write user data
Precondition	-
The called functions	efuse_write
Input parameter{in}	
ef_data	EFUSE data to write
Output parameter{out}	
-	-
Return value	
efuse_state_enum	state of EFUSE, refer to efuse_state_enum

Example:

```

/* write User data value */

uint8_t buffer = 0x11;

efuse_state_enum flag = EFUSE_READY;

flag = efuse_user_data_write(buffer);

```

efuse_flag_get

The description of efuse_flag_get is shown as below:

Table 3-517. Function efuse_flag_get

Function name	efuse_flag_get
Function prototype	FlagStatus efuse_flag_get(efuse_flag_enum efuse_flag);
Function descriptions	check EFUSE flag is set or not
Precondition	-
The called functions	-
Input parameter{in}	
efuse_flag	specifies to get a flag
<i>EFUSE_PGIF</i>	programming operation completion flag
<i>EFUSE_RDIF</i>	read operation completion flag

<i>EFUSE_OBERIF</i>	overstep boundary error flag
Output parameter{out}	
-	-
Return value	
FlagStatus	SET or RESET

Example:

```
/* get EFUSE write operation complete flag status */
```

```
FlagStatus state = efuse_flag_get(EFUSE_PGIF);
```

efuse_flag_clear

The description of efuse_flag_clear is shown as below:

Table 3-518. Function efuse_flag_clear

Function name	efuse_flag_clear
Function prototype	void efuse_flag_clear(efuse_clear_flag_enum efuse_cflag);
Function descriptions	clear EFUSE pending flag
Precondition	-
The called functions	-
Input parameter{in}	
efuse_cflag	specifies to clear a flag
<i>EFUSE_PGIC</i>	clear programming operation completion flag
<i>EFUSE_RDIC</i>	clear read operation completion flag
<i>EFUSE_OBERIC</i>	clear overstep boundary error flag
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* clear EFUSE write operation complete flag status */
```

```
efuse_flag_clear(EFUSE_PGIC);
```

efuse_interrupt_enable

The description of efuse_interrupt_enable is shown as below:

Table 3-519. Function efuse_interrupt_enable

Function name	efuse_interrupt_enable
Function prototype	void efuse_interrupt_enable(efuse_int_enum source);
Function descriptions	enable EFUSE interrupt
Precondition	-
The called functions	-

Input parameter{in}	
source	specifies an interrupt to enable
<i>EFUSE_INTEN_PG</i>	programming operation completion interrupt
<i>EFUSE_INTEN_RD</i>	read operation completion interrupt
<i>EFUSE_INTEN_OBER</i>	overstep boundary error interrupt
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* enable EFUSE write operation complete interrupt */
efuse_interrupt_enable(EFUSE_INTEN_PG);
```

efuse_interrupt_disable

The description of efuse_interrupt_disable is shown as below:

Table 3-520. Function efuse_interrupt_disable

Function name	efuse_interrupt_disable
Function prototype	void efuse_interrupt_disable(efuse_int_enum source);
Function descriptions	disable EFUSE interrupt
Precondition	-
The called functions	-
Input parameter{in}	
source	specifies an interrupt to disable
<i>EFUSE_INTEN_PG</i>	programming operation completion interrupt
<i>EFUSE_INTEN_RD</i>	read operation completion interrupt
<i>EFUSE_INTEN_OBER</i>	overstep boundary error interrupt
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* disable EFUSE write operation complete interrupt */
efuse_interrupt_disable(EFUSE_INTEN_PG);
```

efuse_interrupt_flag_get

The description of efuse_interrupt_flag_get is shown as below:

Table 3-521. Function efuse_interrupt_flag_get

Function name	efuse_interrupt_flag_get
----------------------	--------------------------

Function prototype	FlagStatus efuse_interrupt_flag_get(efuse_int_flag_enum int_flag);
Function descriptions	check EFUSE interrupt flag is set or not
Precondition	-
The called functions	-
Input parameter{in}	
int_flag	specifies to get a flag
<i>EFUSE_INT_PGIF</i>	programming operation completion interrupt flag
<i>EFUSE_INT_RDIF</i>	read operation completion interrupt flag
<i>EFUSE_INT_OBERIF</i>	overstep boundary error interrupt flag
Output parameter{out}	
-	-
Return value	
FlagStatus	SET or RESET

Example:

```
/* get EFUSE write operation complete interrupt flag status */
```

```
FlagStatus state = efuse_interrupt_flag_get(EFUSE_INT_PGIF);
```

efuse_interrupt_flag_clear

The description of efuse_interrupt_flag_clear is shown as below:

Table 3-522. Function efuse_interrupt_flag_clear

Function name	efuse_interrupt_flag_clear
Function prototype	void efuse_interrupt_flag_clear(efuse_clear_int_flag_enum int_cflag);
Function descriptions	clear EFUSE pending interrupt flag
Precondition	-
The called functions	-
Input parameter{in}	
int_cflag	specifies to clear a flag
<i>EFUSE_INT_PGIC</i>	clear programming operation completion interrupt flag
<i>EFUSE_INT_RDIC</i>	clear read operation completion interrupt flag
<i>EFUSE_INT_OBERIC</i>	clear overstep boundary error interrupt flag
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* clear EFUSE write operation complete interrupt flag status */
```

```
efuse_interrupt_flag_clear(EFUSE_INT_PGIC);
```

efuse_ready_wait

The description of efuse_ready_wait is shown as below:

Table 3-523. Function efuse_ready_wait

Function name	efuse_ready_wait
Function prototype	efuse_state_enum efuse_ready_wait(uint32_t efuse_flag, uint32_t timeout)
Function descriptions	check whether EFUSE is ready or not
Precondition	-
The called functions	-
Input parameter{in}	
efuse_flag	specifies to get a flag
<i>EFUSE_PGIF</i>	programming operation completion flag
<i>EFUSE_RDIF</i>	read operation completion flag
<i>EFUSE_OBERIF</i>	overstep boundary error flag
Input parameter{in}	
timeout	timeout value
Output parameter{out}	
-	-
Return value	
efuse_state_enum	state of FMC, refer to fmc_state_enum

Example:

```
/* check whether EFUSE is ready or not */
```

```
efuse_state_enum state = efuse_ready_wait(EFUSE_TIMEOUT_COUNT);
```

3.15. 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 [3.15.1](#) the FWDGT firmware functions are introduced in chapter [3.15.2](#).

3.15.1. Descriptions of Peripheral registers

FWDGT registers are listed in the table shown as below:

Table 3-524. FWDGT Registers

Registers	Descriptions
FWDGT_CTL	Control register
FWDGT_PSC	Prescaler register
FWDGT_RLD	Reload register
FWDGT_STAT	Status register

3.15.2. Descriptions of Peripheral functions

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

Table 3-525. FWDGT firmware function

Function name	Function description
fwdgt_write_enable	enable write access to FWDGT_PSC and FWDGT_RLD
fwdgt_write_disable	disable write access to FWDGT_PSC and FWDGT_RLD
fwdgt_enable	start the free watchdog timer counter
fwdgt_prescaler_value_config	configure the free watchdog timer counter prescaler value
fwdgt_reload_value_config	configure the free watchdog timer counter reload value
fwdgt_counter_reload	reload the counter of FWDGT
fwdgt_config	configure counter reload value, and prescaler divider value
fwdgt_flag_get	get flag state of FWDGT

fwdgt_write_enable

The description of fwdgt_write_enable is shown as below:

Table 3-526. Function fwdgt_write_enable

Function name	fwdgt_write_enable
Function prototype	void fwdgt_write_enable(void);
Function descriptions	enable write access to FWDGT_PSC and FWDGT_RLD
Precondition	-
The called functions	-
Input parameter{in}	
-	-
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* enable write access to FWDGT_PSC and FWDGT_RLD */
fwdgt_write_enable( );
```

fwdgt_write_disable

The description of fwdgt_write_disable is shown as below:

Table 3-527. Function fwdgt_write_disable

Function name	fwdgt_write_disable
Function prototype	void fwdgt_write_disable(void);
Function descriptions	disable write access to FWDGT_PSC and FWDGT_RLD
Precondition	-

The called functions	-
Input parameter{in}	
-	-
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* disable write access to FWDGT_PSC and FWDGT_RLD */
```

```
fwdgt_write_disable( );
```

fwdgt_enable

The description of fwdgt_enable is shown as below:

Table 3-528. Function fwdgt_enable

Function name	fwdgt_enable
Function prototype	void fwdgt_enable(void);
Function descriptions	start the free watchdog timer counter
Precondition	-
The called functions	-
Input parameter{in}	
-	-
Output parameter{out}	
-	-
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 3-529. Function fwdgt_prescaler_value_config

Function name	fwdgt_prescaler_value_config
Function prototype	ErrStatus fwdgt_prescaler_value_config(uint16_t prescaler_value);
Function descriptions	configure the free watchdog timer counter prescaler value
Precondition	-
Input parameter{in}	
prescaler_value	specify prescaler value

<i>FWDGT_PSC_DIV4</i>	FWDGT prescaler set to 4
<i>FWDGT_PSC_DIV8</i>	FWDGT prescaler set to 8
<i>FWDGT_PSC_DIV16</i>	FWDGT prescaler set to 16
<i>FWDGT_PSC_DIV32</i>	FWDGT prescaler set to 32
<i>FWDGT_PSC_DIV64</i>	FWDGT prescaler set to 64
<i>FWDGT_PSC_DIV128</i>	FWDGT prescaler set to 128
<i>FWDGT_PSC_DIV256</i>	FWDGT prescaler set to 256
<i>FWDGT_PSC_DIV512</i>	FWDGT prescaler set to 512
<i>FWDGT_PSC_DIV1024</i> 4	FWDGT prescaler set to 1024
<i>FWDGT_PSC_DIV2048</i> 8	FWDGT prescaler set to 2048
<i>FWDGT_PSC_DIV4096</i> 6	FWDGT prescaler set to 4096
<i>FWDGT_PSC_DIV8192</i> 2	FWDGT prescaler set to 8192
<i>FWDGT_PSC_DIV16384</i> 84	FWDGT prescaler set to 16384
<i>FWDGT_PSC_DIV32768</i> 68	FWDGT prescaler set to 32768
Output parameter{out}	
-	-
Return value	
ErrStatus	ERROR / SUCCESS

Example:

```
/* set FWDGT prescaler to 256 */
```

ErrStatus flag;

```
flag = fwdgt_prescaler_value_config(FWDGT_PSC_DIV256);
```

fwdgt_reload_value_config

The description of fwdgt_reload_value_config is shown as below:

Table 3-530. Function fwdgt_reload_value_config

Function name	fwdgt_reload_value_config
Function prototype	ErrStatus fwdgt_reload_value_config(uint16_t reload_value);
Function descriptions	configure the free watchdog timer counter reload value
Precondition	-
Input parameter{in}	
reload_value	reload_value: specify window value(0x0000 - 0x0FFF)
Output parameter{out}	
-	-

Return value	
ErrStatus	ERROR / SUCCESS

Example:

```
/* set FWDGT reload value to 0x0FFF */
```

ErrStatus flag;

```
flag = fwdgt_reload_value_config(0x0FFF);
```

fwdgt_counter_reload

The description of fwdgt_counter_reload is shown as below:

Table 3-531. Function fwdgt_counter_reload

Function name	fwdgt_counter_reload
Function prototype	void fwdgt_counter_reload(void);
Function descriptions	reload the counter of FWDGT
Precondition	-
The called functions	-
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 3-532. Function fwdgt_config

Function name	fwdgt_config
Function prototype	ErrStatus 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

<i>FWDGT_PSC_DIV8</i>	FWDGT prescaler set to 8
<i>FWDGT_PSC_DIV16</i>	FWDGT prescaler set to 16
<i>FWDGT_PSC_DIV32</i>	FWDGT prescaler set to 32
<i>FWDGT_PSC_DIV64</i>	FWDGT prescaler set to 64
<i>FWDGT_PSC_DIV128</i>	FWDGT prescaler set to 128
<i>FWDGT_PSC_DIV256</i>	FWDGT prescaler set to 256
<i>FWDGT_PSC_DIV512</i>	FWDGT prescaler set to 512
<i>FWDGT_PSC_DIV1024</i>	FWDGT prescaler set to 1024
<i>FWDGT_PSC_DIV2048</i>	FWDGT prescaler set to 2048
<i>FWDGT_PSC_DIV4096</i>	FWDGT prescaler set to 4096
<i>FWDGT_PSC_DIV8192</i>	FWDGT prescaler set to 8192
<i>FWDGT_PSC_DIV16384</i>	FWDGT prescaler set to 16384
<i>FWDGT_PSC_DIV32768</i>	FWDGT prescaler set to 32768
<i>4</i>	
<i>8</i>	
<i>6</i>	
<i>2</i>	
<i>84</i>	
<i>68</i>	
Output parameter{out}	
-	-
Return value	
ErrStatus	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 3-533. Function fwdgt_flag_get fwdgt_write_disable

Function name	fwdgt_flag_get
Function prototype	FlagStatus fwdgt_flag_get(uint16_t flag);
Function descriptions	get flag state of FWDGT
Precondition	-
The called functions	-
Input parameter{in}	
flag	flag to get
<i>FWDGT_FLAG_PUD</i>	a write operation to FWDGT_PSC register is on going
<i>FWDGT_FLAG_RUD</i>	a write operation to FWDGT_RLD register is on going
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);
```

```
if(status == RESET)
```

```
{
```

```
...
```

```
}else
```

```
{
```

```
...
```

```
}
```

3.16. GPIO

GPIO is used to implement logic input/output functions for the devices. The GPIO registers are listed in chapter [3.16.1](#), the GPIO firmware functions are introduced in chapter [3.16.2](#).

3.16.1. Descriptions of Peripheral registers

GPIO registers are listed in the table shown as below:

Table 3-534. GPIO Registers

Registers	Descriptions
GPIOx_CTL	GPIO port control register
GPIO_OMODE	GPIO port output mode register
GPIO_OSPD	GPIO port output speed register
GPIO_PUD	GPIO port pull-up/pull-down register
GPIO_ISTAT	GPIO port input status register
GPIO_OCTL	GPIO port output control register
GPIO_BOP	GPIO port bit operate register
GPIO_LOCK	GPIO port configuration lock register
GPIO_AFSEL0	GPIO alternate function selected register 0
GPIO_AFSEL1	GPIO alternate function selected register 1
GPIO_BC	GPIO bit clear register
GPIO_TG	GPIO port bit toggle register

3.16.2. Descriptions of Peripheral functions

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

Table 3-535. GPIO firmware function

Function name	Function description
gpio_deinit	reset GPIO port
gpio_mode_set	set GPIO mode
gpio_output_options_set	set GPIO output type and speed
gpio_bit_set	set GPIO pin bit
gpio_bit_reset	reset GPIO pin bit
gpio_bit_write	write data to the specified GPIO pin
gpio_port_write	write data to the specified GPIO port
gpio_input_bit_get	get GPIO pin input status
gpio_input_port_get	get GPIO port input status
gpio_output_bit_get	get GPIO pin output status
gpio_output_port_get	get GPIO port output status
gpio_af_set	set GPIO alternate function
gpio_pin_lock	lock GPIO pin bit
gpio_bit_toggle	toggle GPIO pin status
gpio_port_toggle	toggle GPIO port status

gpio_deinit

The description of gpio_deinit is shown as below:

Table 3-536. Function gpio_deinit

Function name	gpio_deinit
Function prototype	void gpio_deinit(uint32_t gpio_periph);
Function descriptions	reset GPIO port
Precondition	-
The called functions	rcu_periph_reset_enable / rcu_periph_reset_disable
Input parameter{in}	
gpio_periph	GPIO port
GPIOx	GPIOx (x = A,B,C,D,E,F,G,H,I)
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 3-537. Function gpio_mode_set

Function name	gpio_mode_set
Function prototype	void gpio_mode_set(uint32_t gpio_periph, uint32_t mode, uint32_t pull_up_down, uint32_t pin);
Function descriptions	set GPIO mode
Precondition	-
The called functions	rcu_periph_reset_enable / rcu_periph_reset_disable
Input parameter{in}	
gpio_periph	GPIO port
GPIOx	GPIOx (x = A,B,C,D,E,F,G,H,I)
Input parameter{in}	
mode	GPIO pin mode
GPIO_MODE_INPUT	input mode
GPIO_MODE_OUTPUT	output mode
GPIO_MODE_AF	alternate function mode
GPIO_MODE_ANALOG	analog mode
Input parameter{in}	
pull_up_down	GPIO pin with pull-up or pull-down resistor
GPIO_PUPD_NONE	floating mode, no pull-up and pull-down resistors
GPIO_PUPD_PULLUP	with pull-up resistor
GPIO_PUPD_PULLDOWN	with pull-down resistor
Input parameter{in}	
pin	GPIO pin
GPIO_PIN_x	GPIO_PIN_x (x=0..15)
GPIO_PIN_ALL	All pins
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* set PA0 with pull-up input mode */
```

```
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 3-538. Function gpio_output_options_set

Function name	gpio_output_options_set
Function prototype	void gpio_output_options_set(uint32_t gpio_periph, uint8_t otype, uint32_t speed, uint32_t pin);
Function descriptions	set GPIO output type and speed
Precondition	-
The called functions	-
Input parameter{in}	
gpio_periph	GPIO port
GPIOx	GPIOx (x = A,B,C,D,E,F,G,H,I)
Input parameter{in}	
otype	GPIO pin output mode
GPIO_OTYPE_PP	push pull mode
GPIO_OTYPE_OD	open drain mode
Input parameter{in}	
speed	gpio output max speed value
GPIO_OSPEED_2MHZ	output max speed 2MHz
GPIO_OSPEED_25MHZ	output max speed 10MHz
GPIO_OSPEED_50MHZ	output max speed 50MHz
GPIO_OSPEED_MAX	output max speed more than 50MHz
Input parameter{in}	
pin	GPIO pin
GPIO_PIN_x	GPIO_PIN_x (x=0..15)
GPIO_PIN_ALL	All pins
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* configure 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 3-539. Function gpio_bit_set

Function name	gpio_bit_set
Function prototype	void gpio_bit_set(uint32_t gpio_periph, uint32_t pin);
Function descriptions	set GPIO pin bit
Precondition	-
The called functions	-
Input parameter{in}	
gpio_periph	GPIO port
GPIOx	GPIOx (x = A,B,C,D,E,F,G,H,I)
Input parameter{in}	
pin	GPIO pin
GPIO_PIN_x	GPIO_PIN_x (x=0..15)
GPIO_PIN_ALL	All pins
Output parameter{out}	
-	-
Return value	
-	-

Example:

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

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

gpio_bit_reset

The description of gpio_bit_reset is shown as below:

Table 3-540. Function gpio_bit_reset

Function name	gpio_bit_reset
Function prototype	void gpio_bit_reset(uint32_t gpio_periph, uint32_t pin);
Function descriptions	reset GPIO pin bit
Precondition	-
The called functions	-
Input parameter{in}	
gpio_periph	GPIO port
GPIOx	GPIOx (x = A,B,C,D,E,F,G,H,I)
Input parameter{in}	
pin	GPIO pin
GPIO_PIN_x	GPIO_PIN_x (x=0..15)
GPIO_PIN_ALL	All pins
Output parameter{out}	
-	-
Return value	
-	-

Example:

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

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

gpio_bit_write

The description of gpio_bit_write is shown as below:

Table 3-541. Function gpio_bit_write

Function name	gpio_bit_write
Function prototype	void gpio_bit_write(uint32_t gpio_periph, uint32_t pin, bit_status bit_value);
Function descriptions	write data to the specified GPIO pin
Precondition	-
The called functions	-
Input parameter{in}	
gpio_periph	GPIO port
GPIOx	GPIOx (x = A,B,C,D,E,F,G,H,I)
Input parameter{in}	
pin	GPIO pin
GPIO_PIN_x	GPIO_PIN_x (x=0..15)
GPIO_PIN_ALL	All pins
Input parameter{in}	
bit_value	SET or RESET
RESET	clear the port pin
SET	set the port pin
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 3-542. Function gpio_port_write

Function name	gpio_port_write
Function prototype	void gpio_port_write(uint32_t gpio_periph, uint16_t data);
Function descriptions	write data to the specified GPIO port
Precondition	-
The called functions	-

Input parameter{in}	
gpio_periph	GPIO port
<i>GPIOx</i>	GPIOx (x = A,B,C,D,E,F,G,H,I)
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 */
```

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

gpio_input_bit_get

The description of gpio_input_bit_get is shown as below:

Table 3-543. Function gpio_input_bit_get

Function name	gpio_input_bit_get
Function prototype	FlagStatus gpio_input_bit_get(uint32_t gpio_periph, uint32_t pin);
Function descriptions	get GPIO pin input status
Precondition	-
The called functions	-
Input parameter{in}	
gpio_periph	GPIO port
<i>GPIOx</i>	GPIOx (x = A,B,C,D,E,F,G,H,I)
Input parameter{in}	
pin	GPIO pin
<i>GPIO_PIN_x</i>	GPIO_PIN_x (x=0..15)
<i>GPIO_PIN_ALL</i>	All pins
Output parameter{out}	
-	-
Return value	
FlagStatus	SET or RESET

Example:

```
/* 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 3-544. Function gpio_input_port_get

Function name	gpio_input_port_get
Function prototype	uint16_t gpio_input_port_get(uint32_t gpio_periph);
Function descriptions	get GPIO port input status
Precondition	-
The called functions	-
Input parameter{in}	
gpio_periph	GPIO port
<i>GPIOx</i>	GPIOx (x = A,B,C,D,E,F,G,H,I)
Output parameter{out}	
-	-
Return value	
uint16_t	0x0000-0xFFFF

Example:

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

uint16_t port_state;

port_state = gpio_input_bit_get (GPIOA);
```

gpio_output_bit_get

The description of gpio_output_bit_get is shown as below:

Table 3-545. Function gpio_output_bit_get

Function name	gpio_output_bit_get
Function prototype	FlagStatus gpio_output_bit_get(uint32_t gpio_periph, uint32_t pin);
Function descriptions	get GPIO pin output status
Precondition	-
The called functions	-
Input parameter{in}	
gpio_periph	GPIO port
<i>GPIOx</i>	GPIOx (x = A,B,C,D,E,F,G,H,I)
Input parameter{in}	
pin	GPIO pin
<i>GPIO_PIN_x</i>	GPIO_PIN_x (x=0..15)
<i>GPIO_PIN_ALL</i>	All pins
Output parameter{out}	
-	-
Return value	
FlagStatus	SET or RESET

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 3-546. Function gpio_output_port_get

Function name	gpio_output_port_get
Function prototype	uint16_t gpio_output_port_get(uint32_t gpio_periph);
Function descriptions	get GPIO port output status
Precondition	-
The called functions	-
Input parameter{in}	
gpio_periph	GPIO port
<i>GPIOx</i>	GPIOx (x = A,B,C,D,E,F,G,H,I)
Output parameter{out}	
-	-
Return value	
Uint16_t	0x0000-0xFFFF

Example:

```
/* 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 3-547. Function gpio_pin_remap_config

Function name	gpio_af_set
Function prototype	void gpio_af_set(uint32_t gpio_periph, uint32_t alt_func_num, uint32_t pin);
Function descriptions	set GPIO alternate function
Precondition	-
The called functions	-
Input parameter{in}	
gpio_periph	GPIO port
<i>GPIOx</i>	GPIOx (x = A,B,C,D,E,F,G,H,I)
Input parameter{in}	
alt_func_num	GPIO pin af function

<i>GPIO_AF_0</i>	SYSTEM
<i>GPIO_AF_1</i>	TIMER0, TIMER1, TIMER7
<i>GPIO_AF_2</i>	TIMER0, TIMER2, TIMER3, TIMER4
<i>GPIO_AF_3</i>	TIMER7, TIMER8, TIMER9, TIMER10
<i>GPIO_AF_4</i>	I2C0, I2C1, I2C2, I2C3, SPI4, TIMER0
<i>GPIO_AF_5</i>	SPI0, SPI1, SPI2, SPI3, SPI4, SPI5, I2C3
<i>GPIO_AF_6</i>	SPI1, SPI2, SPI3, SPI4, SAI0, I2C4
<i>GPIO_AF_7</i>	USART0, USART1, USART2, SPI0, SPI1, SPI2
<i>GPIO_AF_8</i>	UART3, UART4, USART5, UART6, UART7
<i>GPIO_AF_9</i>	CAN0, CAN1, TLI, TIMER11, TIMER12, TIMER13, I2C1, I2C2, CTC
<i>GPIO_AF_10</i>	USB_FS, USB_HS
<i>GPIO_AF_11</i>	ENET
<i>GPIO_AF_12</i>	EXMC, SDIO, USB_HS
<i>GPIO_AF_13</i>	DCI
<i>GPIO_AF_14</i>	TLI
<i>GPIO_AF_15</i>	EVENTOUT
Input parameter{in}	
pin	GPIO pin
<i>GPIO_PIN_x</i>	GPIO_PIN_x (x=0..15)
<i>GPIO_PIN_ALL</i>	All pins
Output parameter{out}	
-	-
Return value	
-	-

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 3-548. Function gpio_pin_lock

Function name	gpio_pin_lock
Function prototype	void gpio_pin_lock(uint32_t gpio_periph, uint32_t pin);
Function descriptions	lock GPIO pin bit
Precondition	-
The called functions	-
Input parameter{in}	
gpio_periph	GPIO port
<i>GPIOx</i>	GPIOx (x = A,B,C,D,E,F,G,H,I)
Input parameter{in}	

pin	GPIO pin
<i>GPIO_PIN_x</i>	GPIO_PIN_x (x=0..15)
<i>GPIO_PIN_ALL</i>	All pins
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* lock PA0 */
gpio_pin_lock (GPIOA, GPIO_PIN_0);
```

gpio_bit_toggle

The description of gpio_bit_toggle is shown as below:

Table 3-549. Function gpio_bit_toggle

Function name	gpio_bit_toggle
Function prototype	void gpio_bit_toggle(uint32_t gpio_periph, uint32_t pin);
Function descriptions	toggle GPIO pin status
Precondition	-
The called functions	-
Input parameter{in}	
gpio_periph	GPIO port
<i>GPIOx</i>	GPIOx (x = A,B,C,D,E,F,G,H,I)
Input parameter{in}	
pin	GPIO pin
<i>GPIO_PIN_x</i>	GPIO_PIN_x (x=0..15)
<i>GPIO_PIN_ALL</i>	All pins
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 3-550. Function gpio_port_toggle

Function name	gpio_port_toggle
Function prototype	void gpio_port_toggle(uint32_t gpio_periph);

Function descriptions	toggle GPIO port status
Precondition	-
The called functions	-
Input parameter{in}	
gpio_periph	GPIO port
<i>GPIOx</i>	GPIOx (x = A,B,C,D,E,F,G,H,I)
Input parameter{in}	
pin	GPIO pin
<i>GPIO_PIN_x</i>	GPIO_PIN_x (x=0..15)
<i>GPIO_PIN_ALL</i>	All pins
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* toggle GPIOA*/
```

```
gpio_port_toggle (GPIOA);
```

3.17. HAU

The HASH Acceleration Unit supports acceleration of SHA-1, SHA-224, SHA-256, MD5 algorithm and the HMAC (keyed-hash message authentication code) algorithm. The HAU registers are listed in chapter [3.17.1](#). the HAU firmware functions are introduced in chapter [3.17.2](#).

3.17.1. Descriptions of Peripheral registers

HAU registers are listed in the table shown as below:

Table 3-551. HAU Registers

Registers	Descriptions
HAU_CTL	control register
HAU_DI	data input register
HAU_CFG	configuration register
HAU_DO0	data output register 0
HAU_DO1	data output register 1
HAU_DO2	data output register 2
HAU_DO3	data output register 3
HAU_DO4	data output register 4
HAU_DO5	data output register 5
HAU_DO6	data output register 6
HAU_DO7	data output register 7

Registers	Descriptions
HAU_INTEN	interrupt enable register
HAU_STAT	status and interrupt flag register
HAU_CTXSx (x = 0..53)	context switch register

3.17.2. Descriptions of Peripheral functions

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

Table 3-552. HAU firmware function

Function name	Function description
hau_deinit	reset the HAU peripheral
hau_init	initialize the HAU peripheral parameters
hau_init_struct_para_init	initialize the struct hau_initpara
hau_reset	reset the HAU processor core
hau_last_word_validbits_num_config	configure the number of valid bits in last word of the message
hau_data_write	write data to the IN FIFO
hau_infifo_words_num_get	return the number of words already written into the IN FIFO
hau_digest_read	read the message digest result
hau_digest_calculation_enable	enable digest calculation
hau_multiple_single_dma_config	configure multiple or single DMA is used, and digest calculation at the end of a DMA transfer or not
hau_dma_enable	enable the HAU DMA interface
hau_dma_disable	disable the HAU DMA interface
hau_context_struct_para_init	initialize the struct context
hau_context_save	save the HAU peripheral context
hau_context_restore	restore the HAU peripheral context
hau_hash_sha_1	calculate digest using SHA1 in HASH mode
hau_hmac_sha_1	calculate digest using SHA1 in HMAC mode
hau_hash_sha_224	calculate digest using SHA224 in HASH mode
hau_hmac_sha_224	calculate digest using SHA224 in HMAC mode
hau_hash_sha_256	calculate digest using SHA256 in HASH mode
hau_hmac_sha_256	calculate digest using SHA256 in HMAC mode
hau_hash_md5	calculate digest using MD5 in HASH mode
hau_hmac_md5	calculate digest using MD5 in HMAC mode
hau_flag_get	get the HAU flag status
hau_flag_clear	clear the HAU flag status
hau_interrupt_enable	enable the HAU interrupts
hau_interrupt_disable	disable the HAU interrupts
hau_interrupt_flag_get	get the HAU interrupt flag status
hau_interrupt_flag_clear	clear the HAU interrupt flag status

Structure hau_init_parameter_struct

Table 3-553. Structure hau_init_parameter_struct

Member name	Function description
algo	algorithm selection: HAU_ALGO_SHA1, HAU_ALGO_SHA224, HAU_ALGO_SHA256, HAU_ALGO_MD5
mode	HAU mode selection: HAU_MODE_HASH, HAU_MODE_HMAC
datatype	data type mode: HAU_SWAPPING_32BIT, HAU_SWAPPING_16BIT, HAU_SWAPPING_8BIT, HAU_SWAPPING_1BIT
keytype	key length mode: HAU_KEY_SHORTER_64, HAU_KEY_LONGGER_64

Structure hau_digest_parameter_struct

Table 3-554. Structure hau_digest_parameter_struct

Member name	Function description
out[8]	message digest result 0-7

Structure hau_context_parameter_struct

Table 3-555. Structure hau_context_parameter_struct

Member name	Function description
hau_ctl_bak	backup of HAU_CTL register
hau_cfg_bak	backup of HAU_CFG register
hau_inten_bak	backup of HAU_INTEN register
hau_ctxs_bak[54]	backup of HAU_CTXSx registers

hau_deinit

The description of hau_deinit is shown as below:

Table 3-556. Function hau_deinit

Function name	hau_deinit
Function prototype	void hau_deinit(void);
Function descriptions	reset the HAU peripheral
Precondition	-
The called functions	rcu_periph_reset_enable / rcu_periph_reset_disable
Input parameter{in}	
-	-
Output parameter{out}	
-	-
Return value	
-	-

Example:

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

```
hau_deinit( );
```

hau_init

The description of hau_init is shown as below:

Table 3-557. Function hau_init

Function name	hau_init
Function prototype	void hau_init(hau_init_parameter_struct* initpara);
Function descriptions	initialize the HAU peripheral parameters
Precondition	-
The called functions	-
Input parameter{in}	
initpara	refer to structure Table 3-553. Structure hau_init_parameter_struct
Output parameter{out}	
-	-
Return value	
-	-

Example:

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

```
hau_init_parameter_struct hau_initpara;
```

```
...
```

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

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

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

```
if(key_len > 64U){
```

```
    hau_initpara.keytype = HAU_KEY_LONGGER_64;
```

```
}else{
```

```
    hau_initpara.keytype = HAU_KEY_SHORTER_64;
```

```
}
```

```
hau_init(&hau_initpara);
```

hau_init_struct_para_init

The description of hau_init_struct_para_init is shown as below:

Table 3-558. Function hau_init_struct_para_init

Function name	hau_init_struct_para_init
---------------	---------------------------

Function prototype	void hau_init_struct_para_init(hau_init_parameter_struct* initpara)
Function descriptions	initialize the struct hau_initpara
Precondition	-
The called functions	-
Input parameter{in}	
-	-
Output parameter{out}	
initpara	refer to structure Table 3-553. Structure hau_init_parameter_struct
Return value	
-	-

Example:

```

/* initialize the HAU peripheral parameters */

hau_init_parameter_struct hau_initpara;

hau_init_struct_para_init(&hau_initpara);

```

hau_reset

The description of hau_reset is shown as below:

Table 3-559. Function hau_reset

Function name	hau_reset
Function prototype	void hau_reset(void);
Function descriptions	reset the HAU processor core
Precondition	-
The called functions	-
Input parameter{in}	
-	-
Output parameter{out}	
-	-
Return value	
-	-

Example:

```

/* reset the HAU processor core */

hau_reset( );

```

hau_last_word_validbits_num_config

The description of hau_last_word_validbits_num_config is shown as below:

Table 3-560. Function hau_last_word_validbits_num_config

Function name	hau_last_word_validbits_num_config
Function prototype	void hau_last_word_validbits_num_config(uint32_t valid_num);

Function descriptions	configure the number of valid bits in last word of the message
Precondition	-
The called functions	-
Input parameter{in}	
valid_num	number of valid bits in last word of the message(0x00 – 0x1F)
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* configure the number of valid bits in last word of the message */
```

```
hau_last_word_validbits_num_config(0x00000010);
```

hau_data_write

The description of hau_data_write is shown as below:

Table 3-561. Function hau_data_write

Function name	hau_data_write
Function prototype	void hau_data_write(uint32_t data);
Function descriptions	write data to the IN FIFO
Precondition	-
The called functions	-
Input parameter{in}	
data	data to write (0x0 – 0xFFFFFFFF)
Output parameter{out}	
-	-
Return value	
-	-

Example:

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

```
hau_data_write(0x00000010);
```

hau_infifo_words_num_get

The description of hau_infifo_words_num_get is shown as below:

Table 3-562. Function hau_infifo_words_num_get

Function name	hau_infifo_words_num_get
Function prototype	uint32_t hau_infifo_words_num_get(void);
Function descriptions	return the number of words already written into the IN FIFO
Precondition	-

The called functions	-
Input parameter{in}	
-	-
Output parameter{out}	
-	-
Return value	
uint32_t	0x0 – 0xFFFFFFFF

Example:

```
/* return the number of words already written into the IN FIFO */
```

```
uint32_t num = 0;
```

```
num = hau_infifo_words_num_get();
```

hau_digest_read

The description of hau_digest_read is shown as below:

Table 3-563. Function hau_digest_read

Function name	hau_digest_read
Function prototype	void hau_digest_read(hau_digest_parameter_struct* digestpara);
Function descriptions	read the message digest result
Precondition	-
The called functions	-
Input parameter{in}	
-	-
Output parameter{out}	
digestpara	refer to structure Table 3-554. Structure hau_digest_parameter_struct
Return value	
-	-

Example:

```
/* read the message digest result */
```

```
hau_digest_parameter_struct digestpara;
```

```
hau_digest_read(&digestpara);
```

hau_digest_calculation_enable

The description of hau_digest_calculation_enable is shown as below:

Table 3-564. Function hau_digest_calculation_enable

Function name	hau_digest_calculation_enable
Function prototype	void hau_digest_calculation_enable(void);
Function descriptions	enable digest calculation

Precondition	-
The called functions	-
Input parameter{in}	
-	-
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* enable digest calculation */
```

```
hau_digest_calculation_enable();
```

hau_multiple_single_dma_config

The description of hau_multiple_single_dma_config is shown as below:

Table 3-565. Function hau_multiple_single_dma_config

Function name	hau_multiple_single_dma_config
Function prototype	void hau_multiple_single_dma_config(uint32_t multi_single);
Function descriptions	configure single or multiple DMA is used, and digest calculation at the end of a DMA transfer or not
Precondition	-
The called functions	-
Input parameter{in}	
multi_single	Multiple or single
<i>SINGLE_DMA_AUTO_DIGEST</i>	message padding and message digest calculation at the end of a DMA transfer
<i>MULTIPLE_DMA_NO_DIGEST</i>	multiple DMA transfers needed and CALEN bit is not automatically set at the end of a DMA transfer
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* configure multiple or single DMA is used, and digest calculation at the end of a DMA transfer or not */
```

```
hau_multiple_single_dma_config(SINGLE_DMA_AUTO_DIGEST);
```

hau_dma_enable

The description of hau_dma_enable is shown as below:

Table 3-566. Function hau_dma_enable

Function name	hau_dma_enable
Function prototype	void hau_dma_enable(void);
Function descriptions	enable the HAU DMA interface
Precondition	-
The called functions	-
Input parameter{in}	
-	-
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* enable the HAU DMA interface */
```

```
hau_dma_enable();
```

hau_dma_disable

The description of hau_dma_disable is shown as below:

Table 3-567. Function hau_dma_disable

Function name	hau_dma_disable
Function prototype	void hau_dma_disable(void);
Function descriptions	disable the HAU DMA interface
Precondition	-
The called functions	-
Input parameter{in}	
-	-
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* disable the HAU DMA interface */
```

```
hau_dma_disable();
```

hau_context_struct_para_init

The description of hau_context_struct_para_init is shown as below:

Table 3-568. Function hau_context_struct_para_init

Function name	hau_context_struct_para_init
----------------------	------------------------------

Function prototype	void hau_context_struct_para_init(hau_context_parameter_struct* context)
Function descriptions	initialize the struct context
Precondition	-
The called functions	-
Input parameter{in}	
context	refer to structure Table 3-555. Structure hau_context_parameter_struct
Output parameter{out}	
-	-
Return value	
-	-

Example:

```

/* initialize the struct context */

hau_context_parameter_struct context_para;

hau_context_struct_para_init(&context_para);

```

hau_context_save

The description of hau_context_save is shown as below:

Table 3-569. Function hau_context_save

Function name	hau_context_save
Function prototype	void hau_context_save(hau_context_parameter_struct* context_save)
Function descriptions	save the HAU peripheral context
Precondition	-
The called functions	-
Input parameter{in}	
-	-
Output parameter{out}	
context_save	refer to structure Table 3-555. Structure hau_context_parameter_struct
Return value	
-	-

Example:

```

hau_context_parameter_struct context_para;

hau_context_struct_para_init(&context_para);

/* save HAU context */

hau_context_save(&context_para);

```

hau_context_restore

The description of hau_context_restore is shown as below:

Table 3-570. Function hau_context_restore

Function name	hau_context_restore
Function prototype	void hau_context_restore(hau_context_parameter_struct* context_restore)
Function descriptions	restore the HAU peripheral context
Precondition	-
The called functions	-
Input parameter{in}	
context_restore	refer to structure Table 3-555. Structure hau_context_parameter_struct
Output parameter{out}	
-	-
Return value	
-	-

Example:

```

hau_context_parameter_struct context_para;

hau_context_struct_para_init(&context_para);

hau_context_save(&context_para);

.....

/* restore HAU context */

hau_context_restore(&context_para);

```

hau_hash_sha_1

The description of hau_hash_sha_1 is shown as below:

Table 3-571. Function hau_hash_sha_1

Function name	hau_hash_sha_1
Function prototype	ErrStatus hau_hash_sha_1(uint8_t *input, uint32_t in_length, uint8_t output[]);
Function descriptions	calculate digest using SHA1 in HASH mode
Precondition	-
The called functions	-
Input parameter{in}	
input	pointer to the input buffer
Input parameter{in}	
in_length	length of the input buffer
Output parameter{out}	
output	the result digest
Return value	
ErrStatus	SUCCESS or ERROR

Example:

```

/* calculate digest using SHA1 in HASH mode */

ErrStatus status = ERROR;

uint8_t output[20];

uint8_t input[] = {0x01, 0x02, 0x03, 0x04, 0x05, 0x06, 0x07, 0x08,
                  0x01, 0x02, 0x03, 0x04, 0x05, 0x06, 0x07, 0x08;};

status = hau_hash_sha_1(input, 0x10, output);

```

hau_hmac_sha_1

The description of hau_hmac_sha_1 is shown as below:

Table 3-572. Function hau_hmac_sha_1

Function name	hau_hmac_sha_1
Function prototype	ErrStatus hau_hmac_sha_1(uint8_t *key, uint32_t keysize, uint8_t *input, uint32_t in_length, uint8_t output[]);
Function descriptions	calculate digest using SHA1 in HMAC mode
Precondition	-
The called functions	-
Input parameter{in}	
key	pointer to the key used for HMAC
Input parameter{in}	
keysize	length of the key used for HMAC
Input parameter{in}	
input	pointer to the input buffer
Input parameter{in}	
in_length	length of the input buffer
Output parameter{out}	
output	the result digest
Return value	
ErrStatus	SUCCESS or ERROR

Example:

```

/* calculate digest using SHA1 in HMAC mode */

ErrStatus status = ERROR;

uint8_t output[20];

uint8_t key[] = {0x01, 0x02, 0x03, 0x04, 0x05, 0x06, 0x07, 0x08,
                0x01, 0x02, 0x03, 0x04, 0x05, 0x06, 0x07, 0x08;};

uint8_t input[] = {0x01, 0x02, 0x03, 0x04, 0x05, 0x06, 0x07, 0x08,
                  0x01, 0x02, 0x03, 0x04, 0x05, 0x06, 0x07, 0x08;};

```

```
status = hau_hmac_sha_1(key, 0x10, input, 0x10, output);
```

hau_hash_sha_224

The description of hau_hash_sha_224 is shown as below:

Table 3-573. Function hau_hash_sha_224

Function name	hau_hash_sha_224
Function prototype	ErrStatus hau_hash_sha_224(uint8_t *input, uint32_t in_length, uint8_t output[]);
Function descriptions	calculate digest using SHA224 in HASH mode
Precondition	-
The called functions	-
Input parameter{in}	
input	pointer to the input buffer
Input parameter{in}	
in_length	length of the input buffer
Output parameter{out}	
output	the result digest
Return value	
ErrStatus	SUCCESS or ERROR

Example:

```
/* calculate digest using SHA224 in HASH mode */
```

```
ErrStatus status = ERROR;
```

```
uint8_t output[28];
```

```
uint8_t input[] = {0x01, 0x02, 0x03, 0x04, 0x05, 0x06, 0x07, 0x08,  
                  0x01, 0x02, 0x03, 0x04, 0x05, 0x06, 0x07, 0x08};
```

```
status = hau_hash_sha_224 (input, 0x10, output);
```

hau_hmac_sha_224

The description of hau_hmac_sha_224 is shown as below:

Table 3-574. Function hau_hmac_sha_224

Function name	hau_hmac_sha_224
Function prototype	ErrStatus hau_hmac_sha_224(uint8_t *key, uint32_t keysize, uint8_t *input, uint32_t in_length, uint8_t output[]);
Function descriptions	calculate digest using SHA224 in HMAC mode
Precondition	-
The called functions	-
Input parameter{in}	
key	pointer to the key used for HMAC

Input parameter{in}	
keysize	length of the key used for HMAC
Input parameter{in}	
input	pointer to the input buffer
Input parameter{in}	
in_length	length of the input buffer
Output parameter{out}	
output	the result digest
Return value	
ErrStatus	SUCCESS or ERROR

Example:

```
/* calculate digest using SHA224 in HMAC mode */
```

```
ErrStatus status = ERROR;
```

```
uint8_t output[28];
```

```
uint8_t key[] = {0x01, 0x02, 0x03, 0x04, 0x05, 0x06, 0x07, 0x08,
                 0x01, 0x02, 0x03, 0x04, 0x05, 0x06, 0x07, 0x08};
```

```
uint8_t input[] = {0x01, 0x02, 0x03, 0x04, 0x05, 0x06, 0x07, 0x08,
                  0x01, 0x02, 0x03, 0x04, 0x05, 0x06, 0x07, 0x08};
```

```
status = hau_hmac_sha_224 (key, 0x10, input, 0x10, output);
```

hau_hash_sha_256

The description of hau_hash_sha_256 is shown as below:

Table 3-575. Function hau_hash_sha_256

Function name	hau_hash_sha_256
Function prototype	ErrStatus hau_hash_sha_256(uint8_t *input, uint32_t in_length, uint8_t output[]);
Function descriptions	calculate digest using SHA256 in HASH mode
Precondition	-
The called functions	-
Input parameter{in}	
input	pointer to the input buffer
Input parameter{in}	
in_length	length of the input buffer
Output parameter{out}	
output	the result digest
Return value	
ErrStatus	SUCCESS or ERROR

Example:

```
/* calculate digest using SHA256 in HASH mode */

ErrStatus status = ERROR;

uint8_t output[32];

uint8_t input[] = {0x01, 0x02, 0x03, 0x04, 0x05, 0x06, 0x07, 0x08,
                  0x01, 0x02, 0x03, 0x04, 0x05, 0x06, 0x07, 0x08};

status = hau_hash_sha_256 (input, 0x10, output);
```

hau_hmac_sha_256

The description of hau_hmac_sha_256 is shown as below:

Table 3-576. Function hau_hmac_sha_256

Function name	hau_hmac_sha_256
Function prototype	ErrStatus hau_hmac_sha_256(uint8_t *key, uint32_t keysize, uint8_t *input, uint32_t in_length, uint8_t output[]);
Function descriptions	calculate digest using SHA256 in HMAC mode
Precondition	-
The called functions	-
Input parameter{in}	
key	pointer to the key used for HMAC
Input parameter{in}	
keysize	length of the key used for HMAC
Input parameter{in}	
input	pointer to the input buffer
Input parameter{in}	
in_length	length of the input buffer
Output parameter{out}	
output	the result digest
Return value	
ErrStatus	SUCCESS or ERROR

Example:

```
/* calculate digest using SHA256 in HMAC mode */

ErrStatus status = ERROR;

uint8_t output[32];

uint8_t key[] = {0x01, 0x02, 0x03, 0x04, 0x05, 0x06, 0x07, 0x08,
                 0x01, 0x02, 0x03, 0x04, 0x05, 0x06, 0x07, 0x08};

uint8_t input[] = {0x01, 0x02, 0x03, 0x04, 0x05, 0x06, 0x07, 0x08,
```

```
0x01, 0x02, 0x03, 0x04, 0x05, 0x06, 0x07, 0x08};
```

```
status = hau_hmac_sha_256 (key, 0x10, input, 0x10, output);
```

hau_hash_md5

The description of hau_hash_md5 is shown as below:

Table 3-577. Function hau_hash_md5

Function name	hau_hash_md5
Function prototype	ErrStatus hau_hash_md5(uint8_t *input, uint32_t in_length, uint8_t output[]);
Function descriptions	calculate digest using MD5 in HASH mode
Precondition	-
The called functions	-
Input parameter{in}	
input	pointer to the input buffer
Input parameter{in}	
in_length	length of the input buffer
Output parameter{out}	
output	the result digest
Return value	
ErrStatus	SUCCESS or ERROR

Example:

```
/* calculate digest using MD5 in HASH mode */
```

```
ErrStatus status = ERROR;
```

```
uint8_t output[16];
```

```
uint8_t input[] = {0x01, 0x02, 0x03, 0x04, 0x05, 0x06, 0x07, 0x08,  
0x01, 0x02, 0x03, 0x04, 0x05, 0x06, 0x07, 0x08};
```

```
status = hau_hash_md5 (input, 0x10, output);
```

hau_hmac_md5

The description of hau_hmac_md5 is shown as below:

Table 3-578. Function hau_hmac_md5

Function name	hau_hmac_md5
Function prototype	ErrStatus hau_hmac_md5(uint8_t *key, uint32_t keysize, uint8_t *input, uint32_t in_length, uint8_t output[]);
Function descriptions	calculate digest using MD5 in HMAC mode
Precondition	-
The called functions	-

Input parameter{in}	
key	pointer to the key used for HMAC
Input parameter{in}	
keysize	length of the key used for HMAC
Input parameter{in}	
input	pointer to the input buffer
Input parameter{in}	
in_length	length of the input buffer
Output parameter{out}	
output	the result digest
Return value	
ErrStatus	SUCCESS or ERROR

Example:

```
/* calculate digest using MD5 in HMAC mode */
```

```
ErrStatus status = ERROR;
```

```
uint8_t output[16];
```

```
uint8_t key[] = {0x01, 0x02, 0x03, 0x04, 0x05, 0x06, 0x07, 0x08,
                 0x01, 0x02, 0x03, 0x04, 0x05, 0x06, 0x07, 0x08};
```

```
uint8_t input[] = {0x01, 0x02, 0x03, 0x04, 0x05, 0x06, 0x07, 0x08,
                  0x01, 0x02, 0x03, 0x04, 0x05, 0x06, 0x07, 0x08};
```

```
status = hau_hmac_md5 (key, 0x10, input, 0x10, output);
```

hau_flag_get

The description of hau_flag_get is shown as below:

Table 3-579. Function hau_flag_get

Function name	hau_flag_get
Function prototype	FlagStatus hau_flag_get(uint32_t flag);
Function descriptions	get the HAU flag status
Precondition	-
The called functions	-
Input parameter{in}	
flag	HAU flag status
HAU_FLAG_DATA_INPUT	there is enough space (16 bytes) in the input FIFO
HAU_FLAG_CALCULATION_COMPLETE	digest calculation is completed
HAU_FLAG_DMA	DMA is enabled (DMAE =1) or a transfer is processing
HAU_FLAG_BUSY	data block is in process

<i>HAU_FLAG_INFIFO_NO_EMPTY</i>	the input FIFO is not empty
Output parameter{out}	
-	-
Return value	
FlagStatus	SET or RESET

Example:

```
/* get the HAU flag status */
```

```
FlagStatus status;
```

```
status = hau_flag_get (HAU_FLAG_DMA);
```

hau_flag_clear

The description of hau_flag_clear is shown as below:

Table 3-580. Function hau_flag_clear

Function name	hau_flag_clear
Function prototype	void hau_flag_clear(uint32_t flag);
Function descriptions	clear the HAU flag status
Precondition	-
The called functions	-
Input parameter{in}	
flag	HAU flag status
<i>HAU_FLAG_DATA_INPUT</i>	there is enough space (16 bytes) in the input FIFO
<i>HAU_FLAG_CALCULATION_COMPLETE</i>	digest calculation is completed
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* clear the HAU flag status */
```

```
hau_flag_clear (HAU_FLAG_DATA_INPUT);
```

hau_interrupt_enable

The description of hau_interrupt_enable is shown as below:

Table 3-581. Function hau_interrupt_enable

Function name	hau_interrupt_enable
Function prototype	void hau_interrupt_enable(uint32_t interrupt);

Function descriptions	enable the HAU interrupts
Precondition	-
The called functions	-
Input parameter{in}	
interrupt	specify the HAU interrupt source to be enabled
<i>HAU_INT_DATA_INP UT</i>	a new block can be entered into the IN buffer
<i>HAU_INT_CALCULATI ON_COMPLETE</i>	calculation complete
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* enable hau interrupt */
hau_interrupt_enable (HAU_INT_DATA_INPUT);
```

hau_interrupt_disable

The description of hau_interrupt_disable is shown as below:

Table 3-582. Function hau_interrupt_disable

Function name	hau_interrupt_disable
Function prototype	void hau_interrupt_disable(uint32_t interrupt);
Function descriptions	disable the HAU interrupts
Precondition	-
The called functions	-
Input parameter{in}	
interrupt	specify the HAU interrupt source to be disabled
<i>HAU_INT_DATA_INP UT</i>	a new block can be entered into the IN buffer
<i>HAU_INT_CALCULATI ON_COMPLETE</i>	calculation complete
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* disable hau interrupt */
hau_interrupt_disable(HAU_INT_DATA_INPUT);
```

hau_interrupt_flag_get

The description of hau_interrupt_flag_get is shown as below:

Table 3-583. Function hau_interrupt_flag_get

Function name	hau_interrupt_flag_get
Function prototype	FlagStatus hau_interrupt_flag_get(uint32_t int_flag)
Function descriptions	get the HAU interrupt flag status
Precondition	-
The called functions	-
Input parameter{in}	
int_flag	HAU interrupt flag status
HAU_INT_FLAG_DATA_INPUT	there is enough space (16 bytes) in the input FIFO
HAU_INT_FLAG_CALCULATION_COMPLETE	digest calculation is completed
Output parameter{out}	
-	-
Return value	
FlagStatus	SET or RESET

Example:

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

```
FlagStatus status = hau_interrupt_flag_get(HAU_INT_FLAG_DATA_INPUT);
```

hau_interrupt_flag_clear

The description of hau_interrupt_flag_clear is shown as below:

Table 3-584. Function hau_interrupt_flag_clear

Function name	hau_interrupt_flag_clear
Function prototype	void hau_interrupt_flag_clear(uint32_t int_flag)
Function descriptions	clear the HAU interrupt flag status
Precondition	-
The called functions	-
Input parameter{in}	
int_flag	HAU interrupt flag status
HAU_INT_FLAG_DATA_INPUT	there is enough space (16 bytes) in the input FIFO
HAU_INT_FLAG_CALCULATION_COMPLETE	digest calculation is completed
Output parameter{out}	
-	-
Return value	

-	-
---	---

Example:

```
/* clear the HAU interrupt flag status */
```

```
hau_interrupt_flag_clear(HAU_INT_FLAG_DATA_INPUT);
```

3.18. 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 [3.18.1](#), the I2C firmware functions are introduced in chapter [0](#).

3.18.1. Descriptions of Peripheral registers

I2C registers are listed in the table shown as below:

Table 3-585. I2C0-2 Registers

Registers	Descriptions
I2C_CTL0	Control register 0
I2C_CTL1	Control register 1
I2C_SADDR0	Slave address register 0
I2C_SADDR1	Slave address register 1
I2C_DATA	Transfer buffer register
I2C_STAT0	Transfer status register 0
I2C_STAT1	Transfer status register 1
I2C_CKCFG	Clock configure register
I2C_RT	Rise time register
I2C_FCTL	Filter control register
I2C_SAMCS	SAM control and status register

Table 3-586. I2C3-5 Registers

Registers	Descriptions
I2C_ADD_CTL0	Control register 0
I2C_ADD_CTL1	Control register 1
I2C_ADD_SADDR0	Slave address register 0
I2C_ADD_SADDR1	Slave address register 1
I2C_ADD_TIMING	Timing register
I2C_ADD_TIMEOUT	Timeout register
I2C_ADD_STAT	Status register
I2C_ADD_STATC	Status clear register
I2C_ADD_PEC	PEC register
I2C_ADD_RDATA	Receive data register
I2C_ADD_TDATA	Transmit data register
I2C_ADD_CTL2	Control register 2

3.18.2. Descriptions of Peripheral functions

I2C firmware functions are listed in the table shown as below:

Table 3-587. I2C0-2 firmware function

Function name	Function description
i2c_deinit	reset I2C
i2c_clock_config	configure I2C clock
i2c_mode_addr_config	configure I2C address
i2c_smbus_type_config	SMBus type selection
i2c_ack_config	whether or not to send an ACK
i2c_ackpos_config	configure I2C POAP position
i2c_master_addressing	master send slave address
i2c_dualaddr_enable	enable dual-address mode
i2c_dualaddr_disable	disable dual-address mode
i2c_enable	enable I2C
i2c_disable	disable I2C
i2c_start_on_bus	generate a START condition on I2C bus
i2c_stop_on_bus	generate a STOP condition on I2C bus
i2c_data_transmit	I2C transmit data function
i2c_data_receive	I2C receive data function
i2c_dma_config	configure I2C DMA mode
i2c_dma_last_transfer_config	configure whether next DMA EOT is DMA last transfer or not
i2c_stretch_scl_low_config	whether to stretch SCL low when data is not ready in slave mode
i2c_slave_response_to_gcall_config	whether or not to response to a general call
i2c_software_reset_config	configure software reset of I2C
i2c_pec_config	configure I2C PEC calculation
i2c_pec_transfer_config	configure whether to transfer PEC value
i2c_pec_value_get	packet error checking value
i2c_smbus_alert_config	configure I2C alert through SMBA pin
i2c_smbus_arp_config	configure I2C ARP protocol in SMBus
i2c_analog_noise_filter_disable	I2C analog noise filter disable
i2c_analog_noise_filter_enable	I2C analog noise filter enable
i2c_digital_noise_filter_config	digital noise filter
i2c_sam_enable	enable SAM_V interface
i2c_sam_disable	disable SAM_V interface
i2c_sam_timeout_enable	enable SAM_V interface timeout detect
i2c_sam_timeout_disable	disable SAM_V interface timeout detect
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 3-588. I2C3-5 firmware function

Function name	Function description
i2c_add_deinit	reset I2C
i2c_add_timing_config	configure the timing parameters
i2c_add_digital_noise_filter_config	configure digital noise filter
i2c_add_analog_noise_filter_enable	enable analog noise filter
i2c_add_master_clock_config	configure the SCL high and low period of clock in master mode
i2c_add_master_addressing	configure i2c slave addresss and transfer direction in master mode
i2c_add_address10_header_enable	10-bit address header executes read direction only in master receive mode
i2c_add_address10_header_disable	10-bit address header executes complete sequence in master receive mode
i2c_add_address10_enable	enable 10-bit addressing mode in master mode
i2c_add_address10_disable	disable 10-bit addressing mode in master mode
i2c_add_automatic_end_enable	enable I2C automatic end mode in master mode
i2c_add_automatic_end_disable	disable I2C automatic end mode in master mode
i2c_add_slave_response_to_gcall_enable	enable the response to a general call
i2c_add_slave_response_to_gcall_disable	disable the response to a general call
i2c_add_stretch_scl_low_enable	enable to stretch SCL low when data is not ready in slave mode
i2c_add_stretch_scl_low_disable	disable to stretch SCL low when data is not ready in slave mode
i2c_add_address_config	configure i2c slave address
i2c_add_address_bit_compare_config	define which bits of ADDRESS[7:1] need to compare with the incoming address byte
i2c_add_address_disable	disable i2c address in slave mode
i2c_add_second_address_config	configure i2c second slave address
i2c_add_second_address_disable	disable i2c second address in slave mode
i2c_add_receved_address_get	get received match address in slave mode
i2c_add_slave_byte_control_enable	enable slave byte control
i2c_add_slave_byte_control_disable	disable slave byte control
i2c_add_nack_enable	generate a NACK in slave mode
i2c_add_reload_enable	enable I2C reload mode
i2c_add_reload_disable	disable I2C reload mode
i2c_add_wakeup_from_deepsleep_enable	enable wakeup from deep-sleep mode
i2c_add_wakeup_from_deepsleep_disable	disable wakeup from deep-sleep mode
i2c_add_enable	enable I2C

Function name	Function description
i2c_add_disable	disable I2C
i2c_add_start_on_bus	generate a START condition on I2C bus
i2c_add_stop_on_bus	generate a STOP condition on I2C bus
i2c_add_data_transmit	I2C transmit data
i2c_add_data_receive	I2C receive data
i2c_add_reload_enable	enable I2C reload mode
i2c_add_reload_disable	disable I2C reload mode
i2c_add_transfer_byte_number_config	configure number of bytes to be transferred
i2c_add_dma_enable	enable I2C DMA for transmission or reception
i2c_add_dma_disable	disable I2C DMA for transmission or reception
i2c_add_pec_transfer	I2C transfers PEC value
i2c_add_pec_enable	enable I2C PEC calculation
i2c_add_pec_disable	disable I2C PEC calculation
i2c_add_pec_value_get	get packet error checking value
i2c_add_smbus_alert_enable	enable SMBus Alert
i2c_add_smbus_alert_disable	disable SMBus Alert
i2c_add_smbus_default_addr_enable	enable SMBus device default address
i2c_add_smbus_default_addr_disable	disable SMBus device default address
i2c_add_smbus_host_addr_enable	enable SMBus Host address
i2c_add_smbus_host_addr_disable	disable SMBus Host address
i2c_add_extented_clock_timeout_enable	enable extended clock timeout detection
i2c_add_extented_clock_timeout_disable	disable extended clock timeout detection
i2c_add_clock_timeout_enable	enable clock timeout detection
i2c_add_clock_timeout_disable	disable clock timeout detection
i2c_add_bus_timeout_b_config	configure bus timeout B
i2c_add_bus_timeout_a_config	configure bus timeout A
i2c_add_idle_clock_timeout_config	configure idle clock timeout detection
i2c_add_flag_get	get I2C flag status
i2c_add_flag_clear	clear I2C flag status
i2c_add_interrupt_enable	enable I2C interrupt
i2c_add_interrupt_disable	disable I2C interrupt
i2c_add_interrupt_flag_get	get I2C interrupt flag status
i2c_add_interrupt_flag_clear	clear I2C interrupt flag status

i2c_deinit

The description of i2c_deinit is shown as below:

Table 3-589. 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
Input parameter{in}	
i2c_periph	I2C peripheral
<i>I2Cx</i>	(x=0,1,2)
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* reset I2C0 */
```

```
i2c_deinit (I2C0);
```

i2c_clock_config

The description of i2c_clock_config is shown as below:

Table 3-590. Function i2c_clock_config

Function name	i2c_clock_config
Function prototype	void i2c_clock_config(uint32_t i2c_periph, uint32_t clkspeed, uint32_t dutycyc);
Function descriptions	I2C clock configure
Precondition	-
The called functions	rcu_clock_freq_get
Input parameter{in}	
i2c_periph	I2C peripheral
<i>I2Cx</i>	(x=0,1,2)
Input parameter{in}	
clkspeed	i2c clock speed
Input parameter{in}	
dutycyc	duty cycle in fast mode
<i>I2C_DTCY_2</i>	T_low/T_high=2
<i>I2C_DTCY_16_9</i>	T_low/T_high=16/9
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* configure I2C0 clock speed as 100KHz*/
```

```
i2c_clock_config(I2C0, 100000, I2C_DTCY_2);
```

i2c_mode_addr_config

The description of i2c_mode_addr_config is shown as below:

Table 3-591. Function i2c_mode_addr_config

Function name	i2c_mode_addr_config
Function prototype	void i2c_mode_addr_config(uint32_t i2c_periph, uint32_t mode, uint32_t addformat, uint32_t addr);
Function descriptions	configure I2C address
Precondition	-
The called functions	-
Input parameter{in}	
i2c_periph	I2C peripheral
<i>I2Cx</i>	(x=0,1,2)
Input parameter{in}	
mode	I2C mode select
<i>I2C_I2CMODE_ENABLE</i>	I2C mode
<i>I2C_SMBUSMODE_ENABLE</i>	SMBus mode
Input parameter{in}	
addformat	7bits or 10bits
<i>I2C_ADDFORMAT_7BITS</i>	address format is 7 bits
<i>I2C_ADDFORMAT_10BITS</i>	address format is 10 bits
Input parameter{in}	
addr	I2C address
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* configure I2C0 address as 0x82, using 7 bits */
```

```
i2c_mode_addr_config(I2C0, I2C_I2CMODE_ENABLE, I2C_ADDFORMAT_7BITS, 0x82);
```

i2c_smbus_type_config

The description of i2c_smbus_type_config is shown as below:

Table 3-592. Function i2c_smbus_type_config

Function name	i2c_smbus_type_config
----------------------	-----------------------

Function prototype	void i2c_smbus_type_config(uint32_t i2c_periph, uint32_t type);
Function descriptions	SMBus type selection
Precondition	-
The called functions	-
Input parameter{in}	
i2c_periph	I2C peripheral
<i>I2Cx</i>	(x=0,1,2)
Input parameter{in}	
type	Device or host
<i>I2C_SMBUS_DEVICE</i>	device
<i>I2C_SMBUS_HOST</i>	host
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* config I2C0 as SMBUS host type*/
```

```
i2c_smbus_type_config (I2C0, I2C_SMBUS_HOST);
```

i2c_ack_config

The description of i2c_ack_config is shown as below:

Table 3-593. Function i2c_ack_config

Function name	i2c_ack_config
Function prototype	void i2c_ack_config(uint32_t i2c_periph, uint32_t ack);
Function descriptions	whether or not to send an ACK
Precondition	-
The called functions	-
Input parameter{in}	
i2c_periph	I2C peripheral
<i>I2Cx</i>	(x=0,1,2)
Input parameter{in}	
ack	I2C peripheral
<i>I2C_ACK_ENABLE</i>	ACK will be sent
<i>I2C_ACK_DISABLE</i>	ACK will not be sent
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* I2C0 will send ACK */
```

```
i2c_ack_config (I2C0, I2C_ACK_ENABLE);
```

i2c_ackpos_config

The description of i2c_ackpos_config is shown as below:

Table 3-594. Function i2c_ackpos_config

Function name	i2c_ackpos_config
Function prototype	void i2c_ackpos_config(uint32_t i2c_periph, uint32_t pos);
Function descriptions	I2C POAP position configure
Precondition	-
The called functions	-
Input parameter{in}	
i2c_periph	I2C peripheral
<i>I2Cx</i>	(x=0,1,2)
Input parameter{in}	
pos	ACK position
<i>I2C_ACKPOS_CURRENT</i>	whether to send ACK or not for the current
<i>I2C_ACKPOS_NEXT</i>	whether to send ACK or not for the next byte
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* The ACK of I2C0 is send for the current frame*/
```

```
i2c_ackpos_config (I2C0, I2C_ACKPOS_CURRENT);
```

i2c_master_addressing

The description of i2c_master_addressing is shown as below:

Table 3-595. Function i2c_master_addressing

Function name	i2c_master_addressing
Function prototype	void i2c_master_addressing(uint32_t i2c_periph, uint32_t addr, uint32_t trandirection);
Function descriptions	void i2c_master_addressing(uint32_t i2c_periph, uint32_t addr, uint32_t trandirection);
Precondition	-
The called functions	-
Input parameter{in}	
i2c_periph	I2C peripheral
<i>I2Cx</i>	(x=0,1,2)
Input parameter{in}	

addr	slave address
Input parameter{in}	
trandirection	transmitter or receiver
<i>I2C_TRANSMITTER</i>	transmitter
<i>I2C_RECEIVER</i>	receiver
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* send slave address to I2C bus and I2C0 act as receiver */
```

```
i2c_master_addressing(I2C0, 0x82, I2C_RECEIVER);
```

i2c_dualaddr_enable

The description of i2c_dualaddr_enable is shown as below:

Table 3-596. Function i2c_dualaddr_enable

Function name	i2c_dualaddr_enable
Function prototype	void i2c_dualaddr_enable(uint32_t i2c_periph, uint32_t dualaddr);
Function descriptions	enable dual-address mode
Precondition	-
The called functions	-
Input parameter{in}	
i2c_periph	I2C peripheral
<i>I2Cx</i>	(x=0,1,2)
Input parameter{in}	
dualaddr	I2C address
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* enable I2C0 dual-address*/
```

```
i2c_dualaddr_enable (I2C0, 0x84);
```

i2c_dualaddr_disable

The description of i2c_dualaddr_disable is shown as below:

Table 3-597. Function i2c_dualaddr_disable

Function name	i2c_dualaddr_disable
----------------------	----------------------

Function prototype	void i2c_dualaddr_disable(uint32_t i2c_periph);
Function descriptions	disable dual-address mode
Precondition	-
The called functions	-
Input parameter{in}	
i2c_periph	I2C peripheral
<i>I2Cx</i>	(x=0,1,2)
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* disable I2C0 dual-address*/
i2c_dualaddr_disable (I2C0);
```

i2c_enable

The description of i2c_enable is shown as below:

Table 3-598. Function i2c_enable

Function name	i2c_enable
Function prototype	void i2c_enable(uint32_t i2c_periph);
Function descriptions	enable I2C
Precondition	-
The called functions	-
Input parameter{in}	
i2c_periph	I2C peripheral
<i>I2Cx</i>	(x=0,1,2)
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* enable I2C0 */
i2c_enable (I2C0);
```

i2c_disable

The description of i2c_disable is shown as below:

Table 3-599. Function i2c_disable

Function name	i2c_disable
----------------------	-------------

Function prototype	void i2c_disable(uint32_t i2c_periph);
Function descriptions	disable I2C
Precondition	-
The called functions	-
Input parameter{in}	
i2c_periph	I2C peripheral
I2Cx	(x=0,1,2)
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 3-600. Function i2c_start_on_bus

Function name	i2c_start_on_bus
Function prototype	void i2c_start_on_bus(uint32_t i2c_periph);
Function descriptions	generate a START condition on I2C bus
Precondition	-
The called functions	-
Input parameter{in}	
i2c_periph	I2C peripheral
I2Cx	(x=0,1,2)
Output parameter{out}	
-	-
Return value	
-	-

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 3-601. Function i2c_stop_on_bus

Function name	i2c_stop_on_bus
----------------------	-----------------

Function prototype	void i2c_stop_on_bus(uint32_t i2c_periph);
Function descriptions	generate a STOP condition on I2C bus
Precondition	-
The called functions	-
Input parameter{in}	
i2c_periph	I2C peripheral
<i>I2Cx</i>	(x=0,1,2)
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 3-602. Function i2c_data_transmit

Function name	i2c_data_transmit
Function prototype	void i2c_data_transmit(uint32_t i2c_periph, uint8_t data);
Function descriptions	I2C transmit data function
Precondition	-
The called functions	-
Input parameter{in}	
i2c_periph	I2C peripheral
<i>I2Cx</i>	(x=0,1,2)
Input parameter{in}	
data	transmit data
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* I2C0 transmit data */
```

```
i2c_data_transmit (I2C0);
```

i2c_data_receive

The description of i2c_data_receive is shown as below:

Table 3-603. Function i2c_data_receive

Function name	i2c_data_receive
Function prototype	uint8_t i2c_data_receive(uint32_t i2c_periph);
Function descriptions	I2C receive data function
Precondition	-
The called functions	-
Input parameter{in}	
i2c_periph	I2C peripheral
I2Cx	(x=0,1,2)
Output parameter{out}	
-	-
Return value	
uint8_t	0x00..0xFF

Example:

```
/* I2C0 receive data */
uint8_t i2c_receiver;
i2c_receiver = i2c_data_receive(I2C0);
```

i2c_dma_config

The description of i2c_dma_config is shown as below:

Table 3-604. Function i2c_dma_config

Function name	i2c_dma_config
Function prototype	void i2c_dma_config(uint32_t i2c_periph, uint32_t dmastate);
Function descriptions	configure I2C DMA mode
Precondition	-
The called functions	-
Input parameter{in}	
i2c_periph	I2C peripheral
I2Cx	(x=0,1,2)
Input parameter{in}	
dmastate	On or off
I2C_DMA_ON	DMA mode enable
I2C_DMA_OFF	DMA mode disable
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* I2C0 DMA mode enable */
```

```
i2c_dma_config(I2C0, I2C_DMA_ON);
```

i2c_dma_last_transfer_enable

The description of i2c_dma_last_transfer_enable is shown as below:

Table 3-605. Function i2c_dma_last_transfer_enable

Function name	i2c_dma_last_transfer_enable
Function prototype	void i2c_dma_last_transfer_enable(uint32_t i2c_periph, uint32_t dmalast);
Function descriptions	flag indicating DMA last transfer
Precondition	-
The called functions	-
Input parameter{in}	
i2c_periph	I2C peripheral
<i>I2Cx</i>	(x=0,1,2)
Input parameter{in}	
dmalast	next DMA EOT is the last transfer or not
<i>I2C_DMALST_ON</i>	next DMA EOT is the last transfer
<i>I2C_DMALST_OFF</i>	next DMA EOT is not the last transfer
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* next DMA EOT is the last transfer */
```

```
i2c_dma_last_transfer_enable (I2C0, I2C_DMALST_ON);
```

i2c_rbne_clear_config

Table 3-606. Function i2c_rbne_clear_config

Function name	i2c_rbne_clear_config
Function prototype	void i2c_rbne_clear_config(uint32_t i2c_periph, uint32_t rbnecondition);
Function descriptions	configure RBNE clear condition
Precondition	-
The called functions	-
Input parameter{in}	
i2c_periph	I2C peripheral
<i>I2Cx</i>	(x=0,1,2)
Input parameter{in}	
rbnecondition	RBNE clear condition
<i>I2C_CFGRBNE_ON</i>	RBNE can be cleared when RXDATA is read
<i>I2C_CFGRBNE_OFF</i>	RBNE can be cleared when RXDATA is read and BTC is cleared
Output parameter{out}	

-	-
Return value	
-	-

Example:

```
/* configure RBNE clear condition */
```

```
i2c_rbne_clear_config(I2C0, I2C_CFGRBNE_ON);
```

i2c_stretch_scl_low_config

The description of i2c_stretch_scl_low_config is shown as below:

Table 3-607. Function i2c_stretch_scl_low_config

Function name	i2c_stretch_scl_low_config
Function prototype	void i2c_stretch_scl_low_config(uint32_t i2c_periph, uint32_t stretchpara);
Function descriptions	whether to stretch SCL low when data is not ready in slave mode
Precondition	-
The called functions	-
Input parameter{in}	
i2c_periph	I2C peripheral
<i>I2Cx</i>	(x=0,1,2)
Input parameter{in}	
stretchpara	SCL stretching enable or disable
<i>I2C_SCLSTRETCH_ENABLE</i>	enable SCL stretching
<i>I2C_SCLSTRETCH_DISABLE</i>	enable SCL stretching
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* stretch SCL low when data is not ready in slave mode */
```

```
i2c_stretch_scl_low_config (I2C0, I2C_SCLSTRETCH_ENABLE);
```

i2c_slave_response_to_gcall_config

The description of i2c_slave_response_to_gcall_config is shown as below:

Table 3-608. Function i2c_slave_response_to_gcall_config

Function name	i2c_slave_response_to_gcall_config
Function prototype	void i2c_slave_response_to_gcall_config(uint32_t i2c_periph, uint32_t gcallpara);

Function descriptions	whether or not to response to a general call
Precondition	-
The called functions	-
Input parameter{in}	
i2c_periph	I2C peripheral
<i>I2Cx</i>	(x=0,1,2)
Input parameter{in}	
gcallpara	response to a general call or not
<i>I2C_GCEN_ENABLE</i>	slave will response to a general call
<i>I2C_GCEN_DISABLE</i>	slave will not response to a general call
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* I2C0 will response to a general call */
```

```
i2c_slave_response_to_gcall_config (I2C0, I2C_GCEN_ENABLE);
```

i2c_software_reset_config

The description of i2c_software_reset_config is shown as below:

Table 3-609. Function i2c_software_reset_config

Function name	i2c_software_reset_config
Function prototype	void i2c_software_reset_config(uint32_t i2c_periph, uint32_t sreset);
Function descriptions	configure software reset of I2C
Precondition	-
The called functions	-
Input parameter{in}	
i2c_periph	I2C peripheral
<i>I2Cx</i>	(x=0,1,2)
Input parameter{in}	
sreset	under reset or not
<i>I2C_SRESET_SET</i>	I2C is under reset
<i>I2C_SRESET_RESET</i>	I2C is not under reset
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* software reset I2C0 */
```

```
i2c_software_reset_config (I2C0, I2C_SRESET_SET);
```

i2c_pec_config

The description of i2c_pec_config is shown as below:

Table 3-610. Function i2c_pec_config

Function name	i2c_pec_config
Function prototype	void i2c_pec_config (uint32_t i2c_periph, uint32_t pecstate);
Function descriptions	configure I2C PEC calculation
Precondition	-
The called functions	-
Input parameter{in}	
i2c_periph	I2C peripheral
<i>I2Cx</i>	(x=0,1,2)
Input parameter{in}	
pecstate	On or off
<i>I2C_PEC_ENABLE</i>	PEC calculation on
<i>I2C_PEC_DISABLE</i>	PEC calculation off
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* Enable I2C PEC calculation */
```

```
i2c_pec_config (I2C0, I2C_PEC_ENABLE);
```

i2c_pec_transfer_config

The description of i2c_pec_transfer_config is shown as below:

Table 3-611. Function i2c_pec_transfer_config

Function name	i2c_pec_transfer_config
Function prototype	void i2c_pec_transfer_config(uint32_t i2c_periph, uint32_t pecpara);
Function descriptions	configure whether to transfer PEC value
Precondition	-
The called functions	-
Input parameter{in}	
i2c_periph	I2C peripheral
<i>I2Cx</i>	(x=0,1,2)
Input parameter{in}	
pecpara	Transfer PEC or not
<i>I2C_PECTRANS_ENABLE</i>	transfer PEC

<i>I2C_PECTRANS_DISA</i> <i>BLE</i>	not transfer PEC
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* I2C0 transfer PEC */
```

```
i2c_pec_transfer_config(I2C0, I2C_PECTRANS_ENABLE);
```

i2c_pec_value_get

The description of i2c_pec_value_get is shown as below:

Table 3-612. Function i2c_pec_value_get

Function name	i2c_pec_value_get
Function prototype	uint8_t i2c_pec_value_get(uint32_t i2c_periph);
Function descriptions	get packet error checking value
Precondition	-
The called functions	-
Input parameter{in}	
i2c_periph	I2C peripheral
<i>I2Cx</i>	(x=0,1,2)
Output parameter{out}	
-	-
Return value	
uint8_t	0..255

Example:

```
/* I2C0 get packet error checking value */
```

```
uint8_t pec_value;
```

```
pec_value = i2c_pec_value_get (I2C0);
```

i2c_smbus_alert_config

The description of i2c_smbus_alert_config is shown as below:

Table 3-613. Function i2c_smbus_alert_config

Function name	i2c_smbus_alert_config
Function prototype	void i2c_smbus_alert_config(uint32_t i2c_periph, uint32_t smbuspara);
Function descriptions	configure I2C alert through SMBA pin
Precondition	-
The called functions	-

Input parameter{in}	
i2c_periph	I2C peripheral
<i>I2Cx</i>	(x=0,1,2)
Input parameter{in}	
smbuspara	issue alert through SMBA pin or not
<i>I2C_SALTSEND_ENABLE</i>	issue alert through SMBA pin
<i>I2C_SALTSEND_DISABLE</i>	not issue alert through SMBA pin
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* I2C0 issue alert through SMBA pin enable*/
```

```
i2c_smbus_alert_config(I2C0, I2C_SALTSEND_ENABLE);
```

i2c_smbus_arp_config

The description of i2c_smbus_arp_config is shown as below:

Table 3-614. Function i2c_smbus_arp_config

Function name	i2c_smbus_arp_config
Function prototype	void i2c_smbus_arp_config(uint32_t i2c_periph, uint32_t arpstate);
Function descriptions	configure I2C ARP protocol in SMBus
Precondition	-
The called functions	-
Input parameter{in}	
i2c_periph	I2C peripheral
<i>I2Cx</i>	(x=0,1,2)
Input parameter{in}	
arpstate	ARP protocol in SMBus switch
<i>I2C_ARP_ENABLE</i>	enable ARP
<i>I2C_ARP_DISABLE</i>	disable ARP
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* enable I2C0 ARP protocol in SMBus switch */
```

```
i2c_smbus_arp_config(I2C0, I2C_ARP_ENABLE);
```


i2c_analog_noise_filter_disable

The description of i2c_analog_noise_filter_disable is shown as below:

Table 3-615. Function i2c_analog_noise_filter_disable

Function name	i2c_analog_noise_filter_disable
Function prototype	void i2c_analog_noise_filter_disable(uint32_t i2c_periph);
Function descriptions	I2C analog noise filter disable
Precondition	-
The called functions	-
Input parameter{in}	
i2c_periph	I2C peripheral
<i>I2Cx</i>	(x=0,1,2)
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* disable I2C0 analog noise filter */
i2c_analog_noise_filter_disable (I2C0);
```

i2c_analog_noise_filter_enable

The description of i2c_analog_noise_filter_enable is shown as below:

Table 3-616. Function i2c_analog_noise_filter_enable

Function name	i2c_analog_noise_filter_enable
Function prototype	void i2c_analog_noise_filter_enable(uint32_t i2c_periph);
Function descriptions	I2C analog noise filter enable
Precondition	-
The called functions	-
Input parameter{in}	
i2c_periph	I2C peripheral
<i>I2Cx</i>	(x=0,1,2)
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* enable I2C0 analog noise filter */
i2c_analog_noise_filter_enable (I2C0);
```

i2c_digital_noise_filter_config

The description of i2c_digital_noise_filter_config is shown as below:

Table 3-617. Function i2c_digital_noise_filter_config

Function name	i2c_digital_noise_filter_config
Function prototype	void i2c_digital_noise_filter_config(uint32_t i2c_periph, i2c_digital_filter_enum dfilterpara);
Function descriptions	config digital noise filter
Precondition	-
The called functions	-
Input parameter{in}	
i2c_periph	I2C peripheral
<i>I2Cx</i>	(x=0,1,2)
Input parameter{in}	
dfilterpara	filtered spiker's length
<i>i2c_digital_filter_enum</i>	maximum filter spikes's length
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* config I2C0 digital noise filter as I2C_DF_1PCLK */
i2c_digital_noise_filter_config (I2C0, I2C_DF_1PCLK);
```

i2c_sam_enable

The description of i2c_sam_enable is shown as below:

Table 3-618. Function i2c_sam_enable

Function name	i2c_sam_enable
Function prototype	void i2c_sam_enable (uint32_t i2c_periph);
Function descriptions	enable SAM_V interface
Precondition	-
The called functions	-
Input parameter{in}	
i2c_periph	I2C peripheral
<i>I2Cx</i>	(x=0,1,2)
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* enable I2C0 SAM_V interface */
```

```
i2c_sam_enable (I2C0);
```

i2c_sam_disable

The description of i2c_sam_disable is shown as below:

Table 3-619. Function i2c_sam_disable

Function name	i2c_sam_disable
Function prototype	void i2c_sam_disable (uint32_t i2c_periph);
Function descriptions	disable SAM_V interface
Precondition	-
The called functions	-
Input parameter{in}	
i2c_periph	I2C peripheral
<i>I2Cx</i>	(x=0,1,2)
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* disable I2C0 SAM_V interface*/
```

```
i2c_sam_disable (I2C0);
```

i2c_sam_timeout_enable

The description of i2c_sam_timeout_enable is shown as below:

Table 3-620. Function i2c_sam_timeout_enable

Function name	i2c_sam_timeout_enable
Function prototype	void i2c_sam_timeout_enable (uint32_t i2c_periph);
Function descriptions	enable SAM_V interface timeout detect
Precondition	-
The called functions	-
Input parameter{in}	
i2c_periph	I2C peripheral
<i>I2Cx</i>	(x=0,1,2)
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* enable I2C0 SAM_V interface timeout detect */
```

```
i2c_sam_timeout_enable (I2C0);
```

i2c_sam_timeout_disable

The description of i2c_sam_timeout_disable is shown as below:

Table 3-621. Function i2c_sam_timeout_disable

Function name	i2c_sam_timeout_disable
Function prototype	void i2c_sam_timeout_disable (uint32_t i2c_periph);
Function descriptions	disable SAM_V interface timeout detect
Precondition	-
The called functions	-
Input parameter{in}	
i2c_periph	I2C peripheral
<i>I2Cx</i>	(x=0,1,2)
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* disable I2C0 SAM_V interface timeout detect */
```

```
i2c_sam_timeout_disable (I2C0);
```

i2c_flag_get

The description of i2c_flag_get is shown as below:

Table 3-622. Function i2c_flag_get

Function name	i2c_flag_get
Function prototype	FlagStatus i2c_flag_get(uint32_t i2c_periph,uint32_t flag);
Function descriptions	get I2C flag status
Precondition	-
The called functions	-
Input parameter{in}	
i2c_periph	I2C peripheral
<i>I2Cx</i>	(x=0,1,2)
Input parameter{in}	
flag	specify get which flag
<i>I2C_FLAG_SBSEND</i>	start condition send out
<i>I2C_FLAG_ADDSEND</i>	address is sent in master mode or received and matches in slave mode
<i>I2C_FLAG_BTC</i>	byte transmission finishes
<i>I2C_FLAG_ADD10SEN</i>	header of 10-bit address is sent in master mode

<i>D</i>	
<i>I2C_FLAG_STPDET</i>	stop condition detected in slave mode
<i>I2C_FLAG_RBNE</i>	I2C_DATA is not Empty during receiving
<i>I2C_FLAG_TBE</i>	I2C_DATA is empty during transmitting
<i>I2C_FLAG_BERR</i>	a bus error occurs indication a unexpected start or stop condition on I2C bus
<i>I2C_FLAG_LOSTARB</i>	arbitration lost in master mode
<i>I2C_FLAG_AERR</i>	acknowledge error
<i>I2C_FLAG_OUERR</i>	overflow or underrun situation occurs in slave mode
<i>I2C_FLAG_PECERR</i>	PEC error when receiving data
<i>I2C_FLAG_SMBTO</i>	timeout signal in SMBus mode
<i>I2C_FLAG_SMBALT</i>	SMBus alert status
<i>I2C_FLAG_MASTER</i>	a flag indicating whether I2C block is in master or slave mode
<i>I2C_FLAG_I2CBSY</i>	busy flag
<i>I2C_FLAG_TRS</i>	whether the I2C is a transmitter or a receiver
<i>I2C_FLAG_RXGC</i>	general call address (00h) received
<i>I2C_FLAG_DEFSMB</i>	default address of SMBus device
<i>I2C_FLAG_HSTSMB</i>	SMBus host header detected in slave mode
<i>I2C_FLAG_DUMOD</i>	dual flag in slave mode indicating which address is matched in dual-address mode
<i>I2C_FLAG_TFF</i>	txframe fall flag
<i>I2C_FLAG_TFR</i>	txframe rise flag
<i>I2C_FLAG_RFF</i>	rxframe fall flag
<i>I2C_FLAG_RFR</i>	rxframe rise flag
Output parameter{out}	
-	-
Return value	
FlagStatus	SET / RESET

Example:

```
/* check whether start condition send out */
```

```
FlagStatus flag_state = RESET;
```

```
flag_state = i2c_flag_get (I2C0, I2C_FLAG_SBSEND);
```

i2c_flag_clear

The description of i2c_flag_clear is shown as below:

Table 3-623. Function i2c_flag_clear

Function name	i2c_flag_clear
Function prototype	void i2c_flag_clear(uint32_t i2c_periph, uint32_t flag);
Function descriptions	clear I2C flag status
Precondition	-

The called functions	-
Input parameter{in}	
i2c_periph	I2C peripheral
I2Cx	(x=0,1,2)
Input parameter{in}	
flag	flag type
I2C_FLAG_SMBALT	SMBus Alert status
I2C_FLAG_SMBTO	timeout signal in SMBus mode
I2C_FLAG_PECERR	PEC error when receiving data
I2C_FLAG_OUERR	over-run or under-run situation occurs in slave mode
I2C_FLAG_AERR	acknowledge error
I2C_FLAG_LOSTARB	arbitration lost in master mode
I2C_FLAG_BERR	a bus error
I2C_FLAG_ADDSEND	cleared by reading I2C_STAT0 and reading I2C_STAT1
I2C_FLAG_TFF	txframe fall flag
I2C_FLAG_TFR	txframe rise flag
I2C_FLAG_RFF	rxframe fall flag
I2C_FLAG_RFR	rxframe rise flag
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* 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 3-624. Function i2c_interrupt_enable

Function name	i2c_interrupt_enable
Function prototype	void i2c_interrupt_enable(uint32_t i2c_periph, uint32_t interrupt);
Function descriptions	enable I2C interrupt
Precondition	-
The called functions	-
Input parameter{in}	
i2c_periph	I2C peripheral
I2Cx	(x=0,1,2)
Input parameter{in}	
interrupt	interrupt type
I2C_INT_ERR	error interrupt enable

<i>I2C_INT_EV</i>	event interrupt enable
<i>I2C_INT_BUF</i>	buffer interrupt enable
<i>I2C_INT_TFF</i>	<i>txframe fall interrupt enable</i>
<i>I2C_INT_TFR</i>	<i>txframe rise interrupt enable</i>
<i>I2C_INT_RFF</i>	<i>rxframe fall interrupt enable</i>
<i>I2C_INT_RFR</i>	<i>rxframe rise interrupt enable</i>
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* enable I2C0 error interrupt */
```

```
i2c_interrupt_enable(I2C0, I2C_INT_EV);
```

i2c_interrupt_disable

The description of i2c_interrupt_disable is shown as below:

Table 3-625. Function i2c_interrupt_disable

Function name	i2c_interrupt_disable
Function prototype	void i2c_interrupt_disable(uint32_t i2c_periph, uint32_t interrupt);
Function descriptions	disable I2C interrupt
Precondition	-
The called functions	-
Input parameter{in}	
i2c_periph	I2C peripheral
<i>I2Cx</i>	(x=0,1,2)
Input parameter{in}	
interrupt	interrupt type
<i>I2C_INT_ERR</i>	error interrupt disable
<i>I2C_INT_EV</i>	event interrupt disable
<i>I2C_INT_BUF</i>	buffer interrupt disable
<i>I2C_INT_TFF</i>	<i>txframe fall interrupt enable</i>
<i>I2C_INT_TFR</i>	<i>txframe rise interrupt enable</i>
<i>I2C_INT_RFF</i>	<i>rxframe fall interrupt enable</i>
<i>I2C_INT_RFR</i>	<i>rxframe rise interrupt enable</i>
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* disable I2C0 error interrupt */
```

i2c_interrupt_disable (I2C0, I2C_INT_EV);

i2c_interrupt_flag_get

The description of i2c_interrupt_flag_get is shown as below:

Table 3-626. Function i2c_interrupt_flag_get

Function name	i2c_interrupt_flag_get
Function prototype	FlagStatus i2c_interrupt_flag_get(uint32_t i2c_periph, uint32_t int_flag);
Function descriptions	get I2C interrupt flag status
Precondition	-
The called functions	-
Input parameter{in}	
i2c_periph	I2C peripheral
<i>I2Cx</i>	(x=0,1,2)
Input parameter{in}	
int_flag	interrupt flag
<i>I2C_INT_FLAG_SBSE ND</i>	start condition sent out in master mode interrupt flag
<i>I2C_INT_FLAG_ADDS END</i>	address is sent in master mode or received and matches in slave mode interrupt flag
<i>I2C_INT_FLAG_BTC</i>	byte transmission finishes
<i>I2C_INT_FLAG_ADD10 SEND</i>	header of 10-bit address is sent in master mode interrupt flag
<i>I2C_INT_FLAG_STPD ET</i>	stop condition detected in slave mode interrupt flag
<i>I2C_INT_FLAG_RBNE</i>	I2C_DATA is not Empty during receiving interrupt flag
<i>I2C_INT_FLAG_TBE</i>	I2C_DATA is empty during transmitting interrupt flag
<i>I2C_INT_FLAG_BERR</i>	a bus error occurs indication a unexpected start or stop condition on I2C bus interrupt flag
<i>I2C_INT_FLAG_LOSTA RB</i>	arbitration lost in master mode interrupt flag
<i>I2C_INT_FLAG_AERR</i>	acknowledge error interrupt flag
<i>I2C_INT_FLAG_OUER R</i>	over-run or under-run situation occurs in slave mode interrupt flag
<i>I2C_INT_FLAG_PEC RR</i>	PEC error when receiving data interrupt flag
<i>I2C_INT_FLAG_SMBT O</i>	timeout signal in SMBus mode interrupt flag
<i>I2C_INT_FLAG_SMBA LT</i>	SMBus Alert status interrupt flag
<i>I2C_INT_FLAG_TFF</i>	txframe fall interrupt flag
<i>I2C_INT_FLAG_TFR</i>	txframe rise interrupt flag
<i>I2C_INT_FLAG_RFF</i>	rxframe fall interrupt flag

<i>I2C_INT_FLAG_RFR</i>	rxframe rise interrupt flag
Output parameter{out}	
-	-
Return value	
FlagStatus	SET / RESET

Example:

```
/* check the byte transmission finishes interrupt flag is set or not*/
```

```
FlagStatus flag_state = RESET;
```

```
flag_state = i2c_interrupt_flag_get (I2C0, I2C_INT_FLAG_BTC);
```

i2c_interrupt_flag_clear

The description of i2c_interrupt_flag_clear is shown as below:

Table 3-627. Function i2c_interrupt_flag_clear

Function name	i2c_interrupt_flag_clear
Function prototype	void i2c_interrupt_flag_clear(uint32_t i2c_periph, uint32_t int_flag);
Function descriptions	clear I2C interrupt flag status
Precondition	-
The called functions	-
Input parameter{in}	
i2c_periph	I2C peripheral
<i>I2Cx</i>	(x=0,1,2)
Input parameter{in}	
intflag	interrupt flag
<i>I2C_INT_FLAG_ADDS</i> <i>END</i>	address is sent in master mode or received and matches in slave mode interrupt flag
<i>I2C_INT_FLAG_BERR</i>	a bus error occurs indication a unexpected start or stop condition on I2C bus interrupt flag
<i>I2C_INT_FLAG_LOSTA</i> <i>RB</i>	arbitration lost in master mode interrupt flag
<i>I2C_INT_FLAG_AERR</i>	acknowledge error interrupt flag
<i>I2C_INT_FLAG_OUER</i> <i>R</i>	over-run or under-run situation occurs in slave mode interrupt flag
<i>I2C_INT_FLAG_PEC</i> <i>RR</i>	PEC error when receiving data interrupt flag
<i>I2C_INT_FLAG_SMBT</i> <i>O</i>	timeout signal in SMBus mode interrupt flag
<i>I2C_INT_FLAG_SMBA</i> <i>LT</i>	SMBus Alert status interrupt flag
<i>I2C_INT_FLAG_TFF</i>	txframe fall interrupt flag
<i>I2C_INT_FLAG_TFR</i>	txframe rise interrupt flag

<i>I2C_INT_FLAG_RFF</i>	rxframe fall interrupt flag
<i>I2C_INT_FLAG_RFR</i>	rxframe rise interrupt flag
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* clear the acknowledge error interrupt flag */
```

```
i2c_interrupt_flag_clear (I2C0, I2C_INT_FLAG_AERR);
```

i2c_add_deinit

The description of i2c_add_deinit is shown as below:

Table 3-628. i2c_add_deinit

Function name	i2c_add_deinit
Function prototype	void i2c_add_deinit(uint32_t i2c_add_periph);
Function descriptions	reset peripheral I2C
Precondition	-
The called functions	rcu_periph_reset_enable / rcu_periph_reset_disable
Input parameter{in}	
i2c_add_periph	I2C peripheral
<i>I2Cx</i>	(x=3,4,5)
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* reset I2C3 */
```

```
i2c_add_deinit(I2C3);
```

i2c_add_timing_config

The description of i2c_add_timing_config is shown as below:

Table3-629. i2c_add_timing_config

Function name	i2c_add_timing_config
Function prototype	void i2c_add_timing_config(uint32_t i2c_add_periph, uint32_t psc, uint32_t scl_dely, uint32_t sda_dely);
Function descriptions	configure the timing parameters
Precondition	-
The called functions	-

Input parameter{in}	
i2c_add_periph	I2C peripheral
<i>I2Cx</i>	(x=3,4,5)
Input parameter{in}	
psc	0-0xf, timing prescaler
Input parameter{in}	
scl_dely	0-0xf,data setup time
Input parameter{in}	
sda_dely	0-0xf,data hold time
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* configure the timing parameters */
```

```
i2c_add_timing_config(I2C3, 0x1, 0x2, 0x1);
```

i2c_add_digital_noise_filter_config

The description of i2c_add_digital_noise_filter_config is shown as below:

Table3-630. i2c_add_digital_noise_filter_config

Function name	i2c_add_digital_noise_filter_config
Function prototype	void i2c_add_digital_noise_filter_config(uint32_t i2c_add_periph, uint32_t filter_length);
Function descriptions	configure digital noise filter
Precondition	-
The called functions	-
Input parameter{in}	
i2c_add_periph	I2C peripheral
<i>I2Cx</i>	(x=3,4,5)
Input parameter{in}	
filter_length	filter_length
<i>FILTER_DISABLE</i>	digital filter is disabled
<i>FILTER_LENGTH_1</i>	digital filter is enabled and filter spikes with a length of up to 1 t _{I2CCLK}
<i>FILTER_LENGTH_2</i>	digital filter is enabled and filter spikes with a length of up to 2 t _{I2CCLK}
<i>FILTER_LENGTH_3</i>	digital filter is enabled and filter spikes with a length of up to 3 t _{I2CCLK}
<i>FILTER_LENGTH_4</i>	digital filter is enabled and filter spikes with a length of up to 4 t _{I2CCLK}
<i>FILTER_LENGTH_5</i>	digital filter is enabled and filter spikes with a length of up to 5 t _{I2CCLK}
<i>FILTER_LENGTH_6</i>	digital filter is enabled and filter spikes with a length of up to 6 t _{I2CCLK}
<i>FILTER_LENGTH_7</i>	digital filter is enabled and filter spikes with a length of up to 7 t _{I2CCLK}
<i>FILTER_LENGTH_8</i>	digital filter is enabled and filter spikes with a length of up to 8 t _{I2CCLK}
<i>FILTER_LENGTH_9</i>	digital filter is enabled and filter spikes with a length of up to 9 t _{I2CCLK}

<i>FILTER_LENGTH_10</i>	digital filter is enabled and filter spikes with a length of up to 10 t_{I2CCLK}
<i>FILTER_LENGTH_11</i>	digital filter is enabled and filter spikes with a length of up to 11 t_{I2CCLK}
<i>FILTER_LENGTH_12</i>	digital filter is enabled and filter spikes with a length of up to 12 t_{I2CCLK}
<i>FILTER_LENGTH_13</i>	digital filter is enabled and filter spikes with a length of up to 13 t_{I2CCLK}
<i>FILTER_LENGTH_14</i>	digital filter is enabled and filter spikes with a length of up to 14 t_{I2CCLK}
<i>FILTER_LENGTH_15</i>	digital filter is enabled and filter spikes with a length of up to 15 t_{I2CCLK}
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* I2C3 digital filter filters spikes with a length of up to 1  $t_{I2CCLK}$  */
i2c_add_digital_noise_filter_config(I2C3, FILTER_LENGTH_1);
```

i2c_add_analog_noise_filter_enable

The description of i2c_add_analog_noise_filter_enable is shown as below:

Table3-631. i2c_add_analog_noise_filter_enable

Function name	i2c_add_analog_noise_filter_enable
Function prototype	void i2c_add_analog_noise_filter_enable(uint32_t i2c_add_periph);
Function descriptions	enable analog noise filter
Precondition	-
The called functions	-
Input parameter{in}	
i2c_add_periph	I2C peripheral
<i>I2Cx</i>	(x=3,4,5)
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* enable analog noise filter */
i2c_add_analog_noise_filter_enable(I2C3);
```

i2c_add_analog_noise_filter_disable

The description of i2c_add_analog_noise_filter_disable is shown as below:

Table3-632. i2c_add_analog_noise_filter_disable

Function name	i2c_add_analog_noise_filter_disable
Function prototype	void i2c_add_analog_noise_filter_disable(uint32_t i2c_add_periph);

Function descriptions	disable analog noise filter
Precondition	-
The called functions	-
Input parameter{in}	
i2c_add_periph	I2C peripheral
<i>I2Cx</i>	(x=3,4,5)
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* disable analog noise filter */
```

```
i2c_add_analog_noise_filter_disable(I2C3);
```

i2c_add_master_clock_config

The description of i2c_add_master_clock_config is shown as below:

Table3-633. i2c_add_master_clock_config

Function name	i2c_add_master_clock_config
Function prototype	void i2c_add_master_clock_config(uint32_t i2c_add_periph, uint32_t sclh, uint32_t scll);
Function descriptions	configure the SCL high and low period of clock in master mode
Precondition	-
The called functions	-
Input parameter{in}	
i2c_add_periph	I2C peripheral
<i>I2Cx</i>	(x=3,4,5)
Input parameter{in}	
sclh	0-0xff, SCL high period
Input parameter{in}	
scll	0-0xff, SCL low period
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* configure the SCL and SDA period of clock in master mode */
```

```
i2c_add_master_clock_config(I2C3, 0x0f, 0x0f);
```

i2c_add_master_addressing

The description of i2c_add_master_addressing is shown as below:

Table3-634. i2c_add_master_addressing

Function name	i2c_add_master_addressing
Function prototype	void i2c_add_master_addressing(uint32_t i2c_add_periph, uint32_t address, uint32_t trans_direction);
Function descriptions	configure i2c slave addresss and transfer direction in master mode
Precondition	-
The called functions	-
Input parameter{in}	
i2c_add_periph	I2C peripheral
<i>I2Cx</i>	(x=3,4,5)
Input parameter{in}	
address	0-0x3FF except reserved address, I2C slave address to be sent
Input parameter{in}	
trans_direction	I2C transfer direction in master mode
<i>I2C_ADD_MASTER_TRANSMIT</i>	master transmit
<i>I2C_ADD_MASTER_RECEIVE</i>	master receive
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* send slave address to I2C bus */
```

```
i2c_add_master_addressing(I2C3, 0x82, I2C_ADD_MASTER_TRANSMIT);
```

i2c_add_address10_header_enable

The description of i2c_add_address10_header_enable is shown as below:

Table3-635. i2c_add_address10_header_enable

Function name	i2c_add_address10_header_enable
Function prototype	void i2c_add_address10_header_enable(uint32_t i2c_add_periph);
Function descriptions	10-bit address header executes read direction only in master receive mode
Precondition	-
The called functions	-
Input parameter{in}	
i2c_add_periph	I2C peripheral
<i>I2Cx</i>	(x=3,4,5)

Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* 10-bit address header executes read direction only in master receive mode */
```

```
i2c_add_address10_header_enable(I2C3);
```

i2c_add_address10_header_disable

The description of i2c_add_address10_header_disable is shown as below:

Table3-636. i2c_add_address10_header_disable

Function name	i2c_add_address10_header_disable
Function prototype	void i2c_add_address10_header_disable(uint32_t i2c_add_periph);
Function descriptions	10-bit address header executes complete sequence in master receive mode
Precondition	-
The called functions	-
Input parameter{in}	
i2c_add_periph	I2C peripheral
I2Cx	(x=3,4,5)
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* 10-bit address header executes complete sequence in master receive mode */
```

```
i2c_add_address10_header_disable(I2C3);
```

i2c_add_address10_enable

The description of i2c_add_address10_enable is shown as below:

Table3-637. i2c_add_address10_enable

Function name	i2c_add_address10_enable
Function prototype	void i2c_add_address10_enable(uint32_t i2c_add_periph);
Function descriptions	enable 10-bit addressing mode in master mode
Precondition	-
The called functions	-
Input parameter{in}	
i2c_add_periph	I2C peripheral
I2Cx	(x=3,4,5)

Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* enable 10-bit addressing mode in master mode */
```

```
i2c_add_address10_enable(I2C3);
```

i2c_add_address10_disable

The description of i2c_add_address10_disable is shown as below:

Table3-638. i2c_add_address10_disable

Function name	i2c_add_address10_disable
Function prototype	void i2c_add_address10_disable(uint32_t i2c_add_periph);
Function descriptions	disable 10-bit addressing mode in master mode
Precondition	-
The called functions	-
Input parameter{in}	
i2c_add_periph	I2C peripheral
I2Cx	(x=3,4,5)
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* disable 10-bit addressing mode in master mode */
```

```
i2c_add_address10_disable(I2C3);
```

i2c_add_automatic_end_enable

The description of i2c_add_automatic_end_enable is shown as below:

Table3-639. i2c_add_automatic_end_enable

Function name	i2c_add_automatic_end_enable
Function prototype	void i2c_add_automatic_end_enable(uint32_t i2c_add_periph);
Function descriptions	enable I2C automatic end mode in master mode
Precondition	-
The called functions	-
Input parameter{in}	
i2c_add_periph	I2C peripheral
I2Cx	(x=3,4,5)

Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* enable I2C automatic end mode in master mode */
```

```
i2c_add_automatic_end_enable(I2C3);
```

i2c_add_automatic_end_disable

The description of i2c_add_automatic_end_disable is shown as below:

Table3-640. i2c_add_automatic_end_disable

Function name	i2c_add_automatic_end_disable
Function prototype	void i2c_add_automatic_end_disable(uint32_t i2c_add_periph);
Function descriptions	disable I2C automatic end mode in master mode
Precondition	-
The called functions	-
Input parameter{in}	
i2c_add_periph	I2C peripheral
I2Cx	(x=3,4,5)
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* disable I2C automatic end mode in master mode */
```

```
i2c_add_automatic_end_disable(I2C3);
```

i2c_add_slave_response_to_gcall_enable

The description of i2c_add_slave_response_to_gcall_enable is shown as below:

Table3-641. i2c_slave_response_to_gcall_enable

Function name	i2c_add_slave_response_to_gcall_enable
Function prototype	void i2c_add_slave_response_to_gcall_enable(uint32_t i2c_add_periph);
Function descriptions	enable the response to a general call
Precondition	-
The called functions	-
Input parameter{in}	
i2c_add_periph	I2C peripheral
I2Cx	(x=3,4,5)

Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* enable the response to a general call */
```

```
i2c_add_slave_response_to_gcall_enable(I2C3);
```

i2c_add_slave_response_to_gcall_disable

The description of i2c_add_slave_response_to_gcall_disable is shown as below:

Table3-642. i2c_add_slave_response_to_gcall_disable

Function name	i2c_add_slave_response_to_gcall_disable
Function prototype	void i2c_add_slave_response_to_gcall_disable(uint32_t i2c_add_periph);
Function descriptions	disable the response to a general call
Precondition	-
The called functions	-
Input parameter{in}	
i2c_add_periph	I2C peripheral
I2Cx	(x=3,4,5)
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* disable the response to a general call */
```

```
i2c_add_slave_response_to_gcall_disable(I2C3);
```

i2c_add_stretch_scl_low_enable

The description of i2c_add_stretch_scl_low_enable is shown as below:

Table3-643. i2c_add_stretch_scl_low_enable

Function name	i2c__addstretch_scl_low_enable
Function prototype	void i2c_add_stretch_scl_low_enable(uint32_t i2c_add_periph);
Function descriptions	enable to stretch SCL low when data is not ready in slave mode
Precondition	-
The called functions	-
Input parameter{in}	
i2c_add_periph	I2C peripheral
I2Cx	(x=3,4,5)

Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* enable to stretch SCL low when data is not ready in slave mode */
```

```
i2c_add_stretch_scl_low_enable(I2C3);
```

i2c_add_stretch_scl_low_disable

The description of i2c_add_stretch_scl_low_disable is shown as below:

Table3-644. i2c_add_stretch_scl_low_disable

Function name	i2c_add_stretch_scl_low_disable
Function prototype	void i2c_add_stretch_scl_low_disable(uint32_t i2c_add_periph);
Function descriptions	disable to stretch SCL low when data is not ready in slave mode
Precondition	-
The called functions	-
Input parameter{in}	
i2c_add_periph	I2C peripheral
I2Cx	(x=3,4,5)
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* disable to stretch SCL low when data is not ready in slave mode */
```

```
i2c_add_stretch_scl_low_disable(I2C3);
```

i2c_add_address_config

The description of i2c_add_address_config is shown as below:

Table3-645. i2c_add_address_config

Function name	i2c_add_address_config
Function prototype	void i2c_add_address_config(uint32_t i2c_add_periph, uint32_t address, uint32_t addr_format);
Function descriptions	configure i2c slave address
Precondition	-
The called functions	-
Input parameter{in}	
i2c_add_periph	I2C peripheral

<i>I2Cx</i>	(x=3,4,5)
Input parameter{in}	
address	I2C address
Input parameter{in}	
addr_format	7 bits or 10 bits
<i>I2C_ADD_ADDFORMAT_7BITS</i>	7bits
<i>I2C_ADD_ADDFORMAT_10BITS</i>	10bits
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* configure i2c slave address */
```

```
i2c_add_address_config(I2C3, 0x82, I2C_ADD_ADDFORMAT_7BITS);
```

i2c_add_address_bit_compare_config

The description of i2c_add_address_bit_compare_config is shown as below:

Table3-646. i2c_add_address_bit_compare_config

Function name	i2c_add_address_bit_compare_config
Function prototype	void i2c_add_address_bit_compare_config(uint32_t i2c_add_periph, uint32_t compare_bits);
Function descriptions	define which bits of ADDRESS[7:1] need to compare with the incoming address byte
Precondition	-
The called functions	-
Input parameter{in}	
i2c_add_periph	I2C peripheral
<i>I2Cx</i>	(x=3,4,5)
Input parameter{in}	
compare_bits	the bits need to compare
<i>ADDRESS_BIT1_COMPARE</i>	address bit1 needs compare
<i>ADDRESS_BIT2_COMPARE</i>	address bit2 needs compare
<i>ADDRESS_BIT3_COMPARE</i>	address bit3 needs compare
<i>ADDRESS_BIT4_COMPARE</i>	address bit4 needs compare
<i>ADDRESS_BIT5_COMPARE</i>	address bit5 needs compare

<i>PARE</i>	
<i>ADDRESS_BIT6_COM</i> <i>PARE</i>	address bit6 needs compare
<i>ADDRESS_BIT7_COM</i> <i>PARE</i>	address bit7 needs compare
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* define which bits of ADDRESS[7:1] need to compare with the incoming address byte */
i2c_add_address_bit_compare_config(I2C3, ADDRESS_BIT1_COMPARE);
```

i2c_add_address_disable

The description of i2c_add_address_disable is shown as below:

Table3-647. i2c_add_address_disable

Function name	i2c_add_address_disable
Function prototype	void i2c_add_address_disable(uint32_t i2c_add_periph);
Function descriptions	disable i2c address in slave mode
Precondition	-
The called functions	-
Input parameter{in}	
i2c_add_periph	I2C peripheral
<i>I2Cx</i>	(x=3,4,5)
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* disable i2c address in slave mode */
i2c_add_address_disable(I2C3);
```

i2c_add_second_address_config

The description of i2c_add_second_address_config is shown as below:

Table3-648. i2c_add_second_address_config

Function name	i2c_add_second_address_config
Function prototype	void i2c_add_second_address_config(uint32_t i2c_add_periph, uint32_t address, uint32_t addr_mask);

Function descriptions	configure i2c second slave address
Precondition	-
The called functions	-
Input parameter{in}	
i2c_add_periph	I2C peripheral
<i>I2Cx</i>	(x=3,4,5)
Input parameter{in}	
address	I2C address
Input parameter{in}	
addr_mask	the bits not need to compare
<i>ADDRESS2_NO_MASK</i>	no mask, all the bits must be compared
<i>ADDRESS2_MASK_BIT1</i>	ADDRESS2[1] is masked, only ADDRESS2[7:2] are compared
<i>ADDRESS2_MASK_BIT1_2</i>	ADDRESS2[2:1] is masked, only ADDRESS2[7:3] are compared
<i>ADDRESS2_MASK_BIT1_3</i>	ADDRESS2[3:1] is masked, only ADDRESS2[7:4] are compared
<i>ADDRESS2_MASK_BIT1_4</i>	ADDRESS2[4:1] is masked, only ADDRESS2[7:5] are compared
<i>ADDRESS2_MASK_BIT1_5</i>	ADDRESS2[5:1] is masked, only ADDRESS2[7:6] are compared
<i>ADDRESS2_MASK_BIT1_6</i>	ADDRESS2[6:1] is masked, only ADDRESS2[7] are compared
<i>ADDRESS2_MASK_ALL</i>	all the ADDRESS2[7:1] bits are masked
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* configure i2c second slave address */
```

```
i2c_add_second_address_config(I2C3, 0x82, ADDRESS2_MASK_BIT1_2);
```

i2c_add_second_address_disable

The description of i2c_add_second_address_disable is shown as below:

Table3-649. i2c_add_second_address_disable

Function name	i2c_add_second_address_disable
Function prototype	void i2c_add_second_address_disable(uint32_t i2c_add_periph);
Function descriptions	disable i2c second address in slave mode
Precondition	-

The called functions	-
Input parameter{in}	
i2c_add_periph	I2C peripheral
I2Cx	(x=3,4,5)
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* disable i2c second address in slave mode */
```

```
i2c_add_second_address_disable(I2C3);
```

i2c_add_receved_address_get

The description of i2c_add_receved_address_get is shown as below:

Table3-650. i2c_add_receved_address_get

Function name	i2c_add_receved_address_get
Function prototype	uint32_t i2c_add_receved_address_get(uint32_t i2c_add_periph);
Function descriptions	get received match address in slave mode
Precondition	-
The called functions	-
Input parameter{in}	
i2c_add_periph	I2C peripheral
I2Cx	(x=3,4,5)
Output parameter{out}	
-	-
Return value	
uint32_t	0x00..0x7F

Example:

```
/* get received match address in slave mode */
```

```
uint32_t address;
```

```
address = i2c_add_receved_address_get(I2C3);
```

i2c_add_slave_byte_control_enable

The description of i2c_add_slave_byte_control_enable is shown as below:

Table3-651. i2c_add_slave_byte_control_enable

Function name	i2c_add_slave_byte_control_enable
Function prototype	void i2c_add_slave_byte_control_enable(uint32_t i2c_add_periph);
Function descriptions	enable slave byte control

Precondition	-
The called functions	-
Input parameter{in}	
i2c_add_periph	I2C peripheral
<i>I2Cx</i>	(x=3,4,5)
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* enable slave byte control */
i2c_add_slave_byte_control_enable(I2C3);
```

i2c_add_slave_byte_control_disable

The description of i2c_add_slave_byte_control_disable is shown as below:

Table3-652. i2c_add_slave_byte_control_disable

Function name	i2c_add_slave_byte_control_disable
Function prototype	void i2c_add_slave_byte_control_disable(uint32_t i2c_add_periph);
Function descriptions	disable slave byte control
Precondition	-
The called functions	-
Input parameter{in}	
i2c_add_periph	I2C peripheral
<i>I2Cx</i>	(x=3,4,5)
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* disable slave byte control */
i2c_slave_byte_control_disable(I2C3);
```

i2c_add_nack_enable

The description of i2c_add_nack_enable is shown as below:

Table3-653. i2c_add_nack_enable

Function name	i2c_add_nack_enable
Function prototype	void i2c_add_nack_enable(uint32_t i2c_add_periph);
Function descriptions	generate a NACK in slave mode

Precondition	-
The called functions	-
Input parameter{in}	
i2c_add_periph	I2C peripheral
<i>I2Cx</i>	(x=3,4,5)
Output parameter{out}	
-	-
Return value	
-	-

Example:

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

```
i2c_add_nack_enable(I2C3);
```

i2c_add_wakeup_from_deepsleep_enable

The description of i2c_add_wakeup_from_deepsleep_enable is shown as below:

Table3-654. i2c_add_wakeup_from_deepsleep_enable

Function name	i2c_add_wakeup_from_deepsleep_enable
Function prototype	void i2c_add_wakeup_from_deepsleep_enable(uint32_t i2c_add_periph);
Function descriptions	enable wakeup from Deep-sleep mode
Precondition	-
The called functions	-
Input parameter{in}	
i2c_add_periph	I2C peripheral
<i>I2Cx</i>	(x=3,4,5)
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* enable wakeup from Deep-sleep mode */
```

```
i2c_add_wakeup_from_deepsleep_enable(I2C3);
```

i2c_add_wakeup_from_deepsleep_disable

The description of i2c_add_wakeup_from_deepsleep_disable is shown as below:

Table3-655. i2c_add_wakeup_from_deepsleep_disable

Function name	i2c_add_wakeup_from_deepsleep_disable
Function prototype	void i2c_add_wakeup_from_deepsleep_disable(uint32_t i2c_add_periph);
Function descriptions	disable wakeup from Deep-sleep mode

Precondition	-
The called functions	-
Input parameter{in}	
i2c_add_periph	I2C peripheral
<i>I2Cx</i>	(x=3,4,5)
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* disable wakeup from Deep-sleep mode */
```

```
i2c_add_wakeup_from_deepsleep_disable(I2C3);
```

i2c_add_enable

The description of i2c_add_enable is shown as below:

Table3-656. i2c_add_enable

Function name	i2c_add_enable
Function prototype	void i2c_add_enable(uint32_t i2c_add_periph);
Function descriptions	enable I2C
Precondition	-
The called functions	-
Input parameter{in}	
i2c_add_periph	I2C peripheral
<i>I2Cx</i>	(x=3,4,5)
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* enable I2C3 */
```

```
i2c_add_enable(I2C3);
```

i2c_add_disable

The description of i2c_add_disable is shown as below:

Table3-657. i2c_add_disable

Function name	i2c_add_disable
Function prototype	void i2c_add_disable(uint32_t i2c_add_periph);
Function descriptions	disable I2C

Precondition	-
The called functions	-
Input parameter{in}	
i2c_add_periph	I2C peripheral
<i>I2Cx</i>	(x=3,4,5)
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* disable I2C3 */
```

```
i2c_add_disable(I2C3);
```

i2c_add_start_on_bus

The description of i2c_add_start_on_bus is shown as below:

Table3-658. i2c_add_start_on_bus

Function name	i2c_add_start_on_bus
Function prototype	void i2c_add_start_on_bus(uint32_t i2c_add_periph);
Function descriptions	generate a START condition on I2C bus
Precondition	-
The called functions	-
Input parameter{in}	
i2c_add_periph	I2C peripheral
<i>I2Cx</i>	(x=3,4,5)
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* I2C3 send a start condition to I2C bus */
```

```
i2c_add_start_on_bus(I2C3);
```

i2c_add_stop_on_bus

The description of i2c_add_stop_on_bus is shown as below:

Table3-659. i2c_add_stop_on_bus

Function name	i2c_add_stop_on_bus
Function prototype	void i2c_add_stop_on_bus(uint32_t i2c_add_periph);
Function descriptions	generate a STOP condition on I2C bus

Precondition	-
The called functions	-
Input parameter{in}	
i2c_add_periph	I2C peripheral
<i>I2Cx</i>	(x=3,4,5)
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* I2C3 generate a STOP condition to I2C bus */
```

```
i2c_add_stop_on_bus(I2C3);
```

i2c_add_data_transmit

The description of i2c_add_data_transmit is shown as below:

Table3-660. i2c_add_data_transmit

Function name	i2c_add_data_transmit
Function prototype	void i2c_add_data_transmit(uint32_t i2c_add_periph, uint32_t data);
Function descriptions	I2C transmit data
Precondition	-
The called functions	-
Input parameter{in}	
i2c_add_periph	I2C peripheral
<i>I2Cx</i>	(x=3,4,5)
Input parameter{in}	
data	transmit data
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* I2C3 transmit data */
```

```
i2c_add_data_transmit(I2C3, 0x80);
```

i2c_add_data_receive

The description of i2c_add_data_receive is shown as below:

Table3-661. i2c_add_data_receive

Function name	i2c_add_data_receive
----------------------	----------------------

Function prototype	uint32_t i2c_add_data_receive(uint32_t i2c_add_periph);
Function descriptions	I2C receive data
Precondition	-
The called functions	-
Input parameter{in}	
i2c_add_periph	I2C peripheral
<i>I2Cx</i>	(x=3,4,5)
Output parameter{out}	
-	-
Return value	
uint32_t	0x0000..0x00FF

Example:

```
/* I2C3 receive data */
```

```
uint32_t i2c_receiver;
```

```
i2c_receiver = i2c_add_data_receive(I2C3);
```

i2c_add_reload_enable

The description of i2c_add_reload_enable is shown as below:

Table3-662. i2c_add_reload_enable

Function name	i2c_add_reload_enable
Function prototype	void i2c_add_reload_enable(uint32_t i2c_add_periph);
Function descriptions	enable I2C reload mode
Precondition	-
The called functions	-
Input parameter{in}	
i2c_add_periph	I2C peripheral
<i>I2Cx</i>	(x=3,4,5)
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* enable I2C reload mode */
```

```
i2c_add_reload_enable(I2C3);
```

i2c_add_reload_disable

The description of i2c_add_reload_disable is shown as below:

Table3-663. i2c_add_reload_disable

Function name	i2c_add_reload_disable
Function prototype	void i2c_add_reload_disable(uint32_t i2c_add_periph);
Function descriptions	disable I2C reload mode
Precondition	-
The called functions	-
Input parameter{in}	
i2c_add_periph	I2C peripheral
I2Cx	(x=3,4,5)
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* disable I2C reload mode */
i2c_add_reload_disable(I2C3);
```

i2c_add_transfer_byte_number_config

The description of i2c_add_transfer_byte_number_config is shown as below:

Table3-664. i2c_add_transfer_byte_number_config

Function name	i2c_add_transfer_byte_number_config
Function prototype	void i2c_add_transfer_byte_number_config(uint32_t i2c_add_periph, uint8_t byte_number);
Function descriptions	configure number of bytes to be transferred
Precondition	-
The called functions	-
Input parameter{in}	
i2c_add_periph	I2C peripheral
I2Cx	(x=3,4,5)
Input parameter{in}	
byte_number	0x0-0xFF, number of bytes to be transferred
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* configure number of bytes to be transferred */
i2c_add_transfer_byte_number_config(I2C3, 0xFF);
```

i2c_add_dma_enable

The description of i2c_add_dma_enable is shown as below:

Table3-665. i2c_add_dma_enable

Function name	i2c_add_dma_enable
Function prototype	void i2c_add_dma_enable(uint32_t i2c_add_periph, uint8_t dma);
Function descriptions	enable I2C DMA for transmission or reception
Precondition	-
The called functions	-
Input parameter{in}	
i2c_add_periph	I2C peripheral
<i>I2Cx</i>	(x=3,4,5)
Input parameter{in}	
dma	I2C DMA
<i>I2C_ADD_DMA_TRANSMIT</i>	transmit data using DMA
<i>I2C_ADD_DMA_RECEIVE</i>	receive data using DMA
Output parameter{out}	
-	-
Return value	
-	-

Example:

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

```
i2c_add_dma_enable(I2C3, I2C_ADD_DMA_RECEIVE);
```

i2c_add_dma_disable

The description of i2c_add_dma_disable is shown as below:

Table3-666. i2c_add_dma_disable

Function name	i2c_add_dma_disable
Function prototype	void i2c_add_dma_disable(uint32_t i2c_add_periph, uint8_t dma);
Function descriptions	disable I2C DMA for transmission or reception
Precondition	-
The called functions	-
Input parameter{in}	
i2c_add_periph	I2C peripheral
<i>I2Cx</i>	(x=3,4,5)
Input parameter{in}	
dma	I2C DMA
<i>I2C_ADD_DMA_TRAN</i>	transmit data using DMA

<i>SMIT</i>	
<i>I2C_ADD_DMA_RECEIVE</i>	receive data using DMA
Output parameter{out}	
-	-
Return value	
-	-

Example:

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

```
i2c_add_dma_disable(I2C3, I2C_ADD_DMA_RECEIVE);
```

i2c_add_pec_transfer

The description of i2c_add_pec_transfer is shown as below:

Table3-667. i2c_add_pec_transfer

Function name	i2c_add_pec_transfer
Function prototype	void i2c_add_pec_transfer(uint32_t i2c_add_periph);
Function descriptions	I2C transfers PEC value
Precondition	-
The called functions	-
Input parameter{in}	
i2c_add_periph	I2C peripheral
<i>I2Cx</i>	(x=3,4,5)
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* I2C transfers PEC value */
```

```
i2c_add_pec_transfer(I2C3);
```

i2c_add_pec_enable

The description of i2c_add_pec_enable is shown as below:

Table3-668. i2c_add_pec_enable

Function name	i2c_add_pec_enable
Function prototype	void i2c_add_pec_enable(uint32_t i2c_add_periph);
Function descriptions	enable I2C PEC calculation
Precondition	-
The called functions	-

Input parameter{in}	
i2c_add_periph	I2C peripheral
<i>I2Cx</i>	(x=3,4,5)
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* enable I2C PEC calculation */
```

```
i2c_add_pec_enable(I2C3);
```

i2c_add_pec_disable

The description of i2c_add_pec_disable is shown as below:

Table3-669. i2c_add_pec_disable

Function name	i2c_add_pec_disable
Function prototype	void i2c_add_pec_disable(uint32_t i2c_add_periph);
Function descriptions	disable I2C PEC calculation
Precondition	-
The called functions	-
Input parameter{in}	
i2c_add_periph	I2C peripheral
<i>I2Cx</i>	(x=3,4,5)
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* disable I2C PEC calculation */
```

```
i2c_add_pec_disable(I2C3);
```

i2c_add_pec_value_get

The description of i2c_add_pec_value_get is shown as below:

Table3-670. i2c_add_pec_value_get

Function name	i2c_add_pec_value_get
Function prototype	uint32_t i2c_add_pec_value_get(uint32_t i2c_add_periph);
Function descriptions	get packet error checking value
Precondition	-
The called functions	-

Input parameter{in}	
i2c_add_periph	I2C peripheral
<i>I2Cx</i>	(x=3,4,5)
Output parameter{out}	
-	-
Return value	
uint32_t	PEC value

Example:

```
/* I2C3 get packet error checking value */
uint32_t pec_value;
pec_value = i2c_add_pec_value_get(I2C3);
```

i2c_add_smbus_alert_enable

The description of i2c_add_smbus_alert_enable is shown as below:

Table3-671. i2c_add_smbus_alert_enable

Function name	i2c_add_smbus_alert_enable
Function prototype	void i2c_add_smbus_alert_enable(uint32_t i2c_add_periph);
Function descriptions	enable SMBus Alert
Precondition	-
The called functions	-
Input parameter{in}	
i2c_add_periph	I2C peripheral
<i>I2Cx</i>	(x=3,4,5)
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* enable SMBus Alert */
i2c_add_smbus_alert_enable(I2C3);
```

i2c_add_smbus_alert_disable

The description of i2c_add_smbus_alert_disable is shown as below:

Table3-672. i2c_add_smbus_alert_disable

Function name	i2c_add_smbus_alert_disable
Function prototype	void i2c_add_smbus_alert_disable(uint32_t i2c_add_periph);
Function descriptions	disable SMBus Alert
Precondition	-

The called functions	-
Input parameter{in}	
i2c_add_periph	I2C peripheral
I2Cx	(x=3,4,5)
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* disable SMBus Alert */
```

```
i2c_add_smbus_alert_disable(I2C3);
```

i2c_add_smbus_default_addr_enable

The description of i2c_add_smbus_default_addr_enable is shown as below:

Table3-673. i2c_add_smbus_default_addr_enable

Function name	i2c_add_smbus_default_addr_enable
Function prototype	void i2c_add_smbus_default_addr_enable(uint32_t i2c_add_periph);
Function descriptions	enable SMBus device default address
Precondition	-
The called functions	-
Input parameter{in}	
i2c_add_periph	I2C peripheral
I2Cx	(x=3,4,5)
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* enable SMBus device default address */
```

```
i2c_add_smbus_default_addr_enable(I2C3);
```

i2c_add_smbus_default_addr_disable

The description of i2c_add_smbus_default_addr_disable is shown as below:

Table3-674. i2c_add_smbus_default_addr_disable

Function name	i2c_add_smbus_default_addr_disable
Function prototype	void i2c_add_smbus_default_addr_disable(uint32_t i2c_add_periph);
Function descriptions	disable SMBus device default address
Precondition	-

The called functions	-
Input parameter{in}	
i2c_add_periph	I2C peripheral
I2Cx	(x=3,4,5)
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* disable SMBus device default address */
```

```
i2c_add_smbus_default_addr_disable(I2C3);
```

i2c_add_smbus_host_addr_enable

The description of i2c_add_smbus_host_addr_enable is shown as below:

Table3-675. i2c_add_smbus_host_addr_enable

Function name	i2c_add_smbus_host_addr_enable
Function prototype	void i2c_add_smbus_host_addr_enable(uint32_t i2c_add_periph);
Function descriptions	enable SMBus Host address
Precondition	-
The called functions	-
Input parameter{in}	
i2c_add_periph	I2C peripheral
I2Cx	(x=3,4,5)
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* enable SMBus Host address */
```

```
i2c_add_smbus_host_addr_enable(I2C3);
```

i2c_add_smbus_host_addr_disable

The description of i2c_add_smbus_host_addr_disable is shown as below:

Table3-676. i2c_add_smbus_host_addr_disable

Function name	i2c_add_smbus_host_addr_disable
Function prototype	void i2c_add_smbus_host_addr_disable(uint32_t i2c_add_periph);
Function descriptions	disable SMBus Host address
Precondition	-

The called functions	-
Input parameter{in}	
i2c_add_periph	I2C peripheral
I2Cx	(x=3,4,5)
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* disable SMBus Host address */
```

```
i2c_add_smbus_host_addr_disable(I2C3);
```

i2c_add_extented_clock_timeout_enable

The description of i2c_add_extented_clock_timeout_enable is shown as below:

Table3-677. i2c_add_extented_clock_timeout_enable

Function name	i2c_add_extented_clock_timeout_enable
Function prototype	void i2c_add_extented_clock_timeout_enable(uint32_t i2c_add_periph);
Function descriptions	enable extended clock timeout detection
Precondition	-
The called functions	-
Input parameter{in}	
i2c_add_periph	I2C peripheral
I2Cx	(x=3,4,5)
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* enable extended clock timeout detection */
```

```
i2c_add_extented_clock_timeout_enable(I2C3);
```

i2c_add_extented_clock_timeout_disable

The description of i2c_add_extented_clock_timeout_disable is shown as below:

Table3-678. i2c_add_extented_clock_timeout_disable

Function name	i2c_add_extented_clock_timeout_disable
Function prototype	void i2c_add_extented_clock_timeout_disable(uint32_t i2c_add_periph);
Function descriptions	disable extended clock timeout detection
Precondition	-

The called functions	-
Input parameter{in}	
i2c_add_periph	I2C peripheral
I2Cx	(x=3,4,5)
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* disable extended clock timeout detection */
```

```
i2c_add_extented_clock_timeout_disable(I2C3);
```

i2c_add_clock_timeout_enable

The description of i2c_add_clock_timeout_enable is shown as below:

Table3-679. i2c_add_clock_timeout_enable

Function name	i2c_add_clock_timeout_enable
Function prototype	void i2c_add_clock_timeout_enable(uint32_t i2c_add_periph);
Function descriptions	enable clock timeout detection
Precondition	-
The called functions	-
Input parameter{in}	
i2c_add_periph	I2C peripheral
I2Cx	(x=3,4,5)
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* enable clock timeout detection */
```

```
i2c_add_clock_timeout_enable(I2C3);
```

i2c_add_clock_timeout_disable

The description of i2c_add_clock_timeout_disable is shown as below:

Table3-680. i2c_add_clock_timeout_disable

Function name	i2c_add_clock_timeout_disable
Function prototype	void i2c_add_clock_timeout_disable(uint32_t i2c_add_periph);
Function descriptions	disable clock timeout detection
Precondition	-

The called functions	-
Input parameter{in}	
i2c_add_periph	I2C peripheral
I2Cx	(x=3,4,5)
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* disable clock timeout detection */
```

```
i2c_add_clock_timeout_disable(I2C3);
```

i2c_add_bus_timeout_b_config

The description of i2c_add_bus_timeout_b_config is shown as below:

Table3-681. i2c_add_bus_timeout_b_config

Function name	i2c_add_bus_timeout_b_config
Function prototype	void i2c_add_bus_timeout_b_config(uint32_t i2c_add_periph, uint32_t timeout);
Function descriptions	configure bus timeout B
Precondition	-
The called functions	-
Input parameter{in}	
i2c_add_periph	I2C peripheral
I2Cx	(x=3,4,5)
Input parameter{in}	
timeout	0-0xff, bus timeout B
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* configure bus timeout B */
```

```
i2c_add_bus_timeout_b_config(I2C3, 0xff);
```

i2c_add_bus_timeout_a_config

The description of i2c_add_bus_timeout_a_config is shown as below:

Table3-682. i2c_add_bus_timeout_a_config

Function name	i2c_add_bus_timeout_a_config
---------------	------------------------------

Function prototype	void i2c_add_bus_timeout_a_config(uint32_t i2c_add_periph, uint32_t timeout);
Function descriptions	configure bus timeout A
Precondition	-
The called functions	-
Input parameter{in}	
i2c_add_periph	I2C peripheral
<i>I2Cx</i>	(x=3,4,,5)
Input parameter{in}	
timeout	0-0xffff, bus timeout A
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* configure bus timeout A */
```

```
i2c_add_bus_timeout_a_config(I2C3, 0xff);
```

i2c_add_idle_clock_timeout_config

The description of i2c_add_idle_clock_timeout_config is shown as below:

Table3-683. i2c_add_idle_clock_timeout_config

Function name	i2c_add_idle_clock_timeout_config
Function prototype	void i2c_add_idle_clock_timeout_config(uint32_t i2c_add_periph, uint32_t timeout);
Function descriptions	configure idle clock timeout detection
Precondition	-
The called functions	-
Input parameter{in}	
i2c_add_periph	I2C peripheral
<i>I2Cx</i>	(x=3,4,5)
Input parameter{in}	
timeout	bus timeout A
<i>BUSTOA_DETECT_SCL_LOW</i>	BUSTOA is used to detect SCL low timeout
<i>BUSTOA_DETECT_IDLE</i>	BUSTOA is used to detect both SCL and SDA high timeout when the bus is idle
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* configure idle clock timeout detection */
```

```
i2c_add_idle_clock_timeout_config(I2C3, BUSTOA_DETECT_SCL_LOW);
```

i2c_add_flag_get

The description of i2c_add_flag_get is shown as below:

Table3-684. i2c_add_flag_get

Function name	i2c_add_flag_get
Function prototype	FlagStatus i2c_add_flag_get(uint32_t i2c_add_periph, uint32_t flag);
Function descriptions	get I2C flag status
Precondition	-
The called functions	-
Input parameter{in}	
i2c_add_periph	I2C peripheral
I2Cx	(x=3,4,5)
Input parameter{in}	
flag	I2C flags
I2C_ADD_FLAG_TBE	I2C_TDATA is empty during transmitting
I2C_ADD_FLAG_TI	transmit interrupt
I2C_ADD_FLAG_RBNE	I2C_RDATA is not empty during receiving
I2C_ADD_FLAG_ADDS END	address received matches in slave mode
I2_ADDC_FLAG_NACK	not acknowledge flag
I2C_ADD_FLAG_STPD ET	STOP condition detected in slave mode
I2C_ADD_FLAG_TC	transfer complete in master mode
I2C_ADD_FLAG_TCR	transfer complete reload
I2C_ADD_FLAG_BERR	bus error
I2C_ADD_FLAG_LOST ARB	arbitration Lost
I2C_ADD_FLAG_OUE RR	overflow/underrun error in slave mode
I2C_ADD_FLAG_PEC RR	PEC error
I2C_ADD_FLAG_TIME OUT	timeout flag
I2C_ADD_FLAG_SMB ALT	SMBus Alert
I2_ADDC_FLAG_I2CB SY	busy flag
I2C_ADD_FLAG_TR	whether the I2C is a transmitter or a receiver in slave mode

Output parameter{out}	
-	-
Return value	
FlagStatus	SET / RESET

Example:

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

```
FlagStatus flag_state = RESET;
```

```
flag_state = i2c_add_flag_get(I2C3, I2C_ADD_FLAG_TBE);
```

i2c_add_flag_clear

The description of i2c_add_flag_clear is shown as below:

Table3-685. i2c_add_flag_clear

Function name	i2c_add_flag_clear
Function prototype	void i2c_add_flag_clear(uint32_t i2c_add_periph, uint32_t flag);
Function descriptions	clear I2C flag status
Precondition	-
The called functions	-
Input parameter{in}	
i2c_add_periph	I2C peripheral
I2Cx	(x=3,4,5)
Input parameter{in}	
flag	I2C flags
I2C_ADD_FLAG_ADDS END	address received matches in slave mode
I2C_ADD_FLAG_NACK	not acknowledge flag
I2C_ADD_FLAG_STPD ET	STOP condition detected in slave mode
I2C_ADD_FLAG_BERR	bus error
I2C_ADD_FLAG_LOST ARB	arbitration Lost
I2C_ADD_FLAG_OUE RR	overflow/underrun error in slave mode
I2C_ADD_FLAG_PEC RR	PEC error
I2C_ADD_FLAG_TIME OUT	timeout flag
I2C_ADD_FLAG_SMB ALT	SMBus Alert
Output parameter{out}	
-	-

Return value	
-	-

Example:

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

```
i2c_add_flag_clear(I2C3, I2C_ADD_FLAG_BERR);
```

i2c_add_interrupt_enable

The description of i2c_add_interrupt_enable is shown as below:

Table3-686. i2c_add_interrupt_enable

Function name	i2c_add_interrupt_enable
Function prototype	void i2c_add_interrupt_enable(uint32_t i2c_add_periph, uint32_t interrupt);
Function descriptions	enable I2C interrupt
Precondition	-
The called functions	-
Input parameter{in}	
i2c_add_periph	I2C peripheral
I2Cx	(x=3,4,5)
Input parameter{in}	
interrupt	I2C interrupts
I2C_ADD_INT_ERR	error interrupt
I2C_ADD_INT_TC	transfer complete interrupt
I2C_ADD_INT_STPDET	stop detection interrupt
I2C_ADD_INT_NACK	not acknowledge received interrupt
I2C_ADD_INT_ADDM	address match interrupt
I2C_ADD_INT_RBNE	receive interrupt
I2C_ADD_INT_TI	transmit interrupt
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* enable I2C3 transmit interrupt */
```

```
i2c_add_interrupt_enable(I2C3, I2C_ADD_INT_TI);
```

i2c_add_interrupt_disable

The description of i2c_add_interrupt_disable is shown as below:

Table3-687. i2c_add_interrupt_disable

Function name	i2c_add_interrupt_disable
Function prototype	void i2c_add_interrupt_disable(uint32_t i2c_add_periph, uint32_t interrupt);
Function descriptions	disable I2C interrupt
Precondition	-
The called functions	-
Input parameter{in}	
i2c_add_periph	I2C peripheral
I2Cx	(x=3,4,5)
Input parameter{in}	
interrupt	I2C interrupts
I2C_ADD_INT_ERR	error interrupt
I2C_ADD_INT_TC	transfer complete interrupt
I2C_ADD_INT_STPDET	stop detection interrupt
I2C_ADD_INT_NACK	not acknowledge received interrupt
I2C_ADD_INT_ADDM	address match interrupt
I2C_ADD_INT_RBNE	receive interrupt
I2C_ADD_INT_TI	transmit interrupt
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* disable I2C3 transmit interrupt */
```

```
i2c_add_interrupt_disable(I2C3, I2C_ADD_INT_TI);
```

i2c_add_interrupt_flag_get

The description of i2c_add_interrupt_flag_get is shown as below:

Table3-688. i2c_add_interrupt_flag_get

Function name	i2c_add_interrupt_flag_get
Function prototype	FlagStatus i2c_add_interrupt_flag_get(uint32_t i2c_add_periph, i2c_add_interrupt_flag_enum int_flag);
Function descriptions	get I2C interrupt flag status
Precondition	-
The called functions	-
Input parameter{in}	
i2c_add_periph	I2C peripheral
I2Cx	(x=3,4,5)
Input parameter{in}	
int_flag	I2C interrupt flags

<i>I2C_ADD_INT_FLAG_TI</i>	transmit interrupt flag
<i>I2C_ADD_INT_FLAG_RBNE</i>	I2C_RDATA is not empty during receiving interrupt flag
<i>I2C_ADD_INT_FLAG_ADDSEND</i>	address received matches in slave mode interrupt flag
<i>I2C_ADD_INT_FLAG_NACK</i>	not acknowledge interrupt flag
<i>I2C_ADD_INT_FLAG_STPDET</i>	stop condition detected in slave mode interrupt flag
<i>I2C_ADD_INT_FLAG_TC</i>	transfer complete in master mode interrupt flag
<i>I2C_ADD_INT_FLAG_TCR</i>	transfer complete reload interrupt flag
<i>I2C_ADD_INT_FLAG_BERR</i>	bus error interrupt flag
<i>I2C_ADD_INT_FLAG_LOSTARB</i>	arbitration lost interrupt flag
<i>I2C_ADD_INT_FLAG_OVERR</i>	overflow/underrun error in slave mode interrupt flag
<i>I2C_ADD_INT_FLAG_PECERR</i>	PEC error interrupt flag
<i>I2C_ADD_INT_FLAG_TIMEOUT</i>	timeout interrupt flag
<i>I2C_ADD_INT_FLAG_SMBALT</i>	SMBus Alert interrupt flag
Output parameter{out}	
-	-
Return value	
FlagStatus	SET / RESET

Example:

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

```
FlagStatus flag_state = RESET;
```

```
flag_state = i2c_add_interrupt_flag_get(I2C3, I2C_ADD_INT_FLAG_TI);
```

i2c_add_interrupt_flag_clear

The description of i2c_add_interrupt_flag_clear is shown as below:

Table3-689. i2c_add_interrupt_flag_clear

Function name	i2c_interrupt_flag_clear
Function prototype	void i2c_add_interrupt_flag_clear(uint32_t i2c_add_periph,

	i2c_add_interrupt_flag_enum int_flag);
Function descriptions	clear I2C interrupt flag status
Precondition	-
The called functions	-
Input parameter{in}	
i2c_add_periph	I2C peripheral
<i>I2Cx</i>	(x=3,4,5)
Input parameter{in}	
int_flag	I2C interrupt flags
<i>I2C_ADD_INT_FLAG_ADDSEND</i>	address received matches in slave mode interrupt flag
<i>I2C_ADD_INT_FLAG_NACK</i>	not acknowledge interrupt flag
<i>I2C_ADD_INT_FLAG_STPDET</i>	stop condition detected in slave mode interrupt flag
<i>I2C_ADD_INT_FLAG_BERR</i>	bus error interrupt flag
<i>I2C_ADD_INT_FLAG_LOSTARB</i>	arbitration lost interrupt flag
<i>I2C_ADD_INT_FLAG_OUERR</i>	overflow/underrun error in slave mode interrupt flag
<i>I2C_ADD_INT_FLAG_PECERR</i>	PEC error interrupt flag
<i>I2C_ADD_INT_FLAG_TIMEOUT</i>	timeout interrupt flag
<i>I2C_ADD_INT_FLAG_SMBALT</i>	SMBus Alert interrupt flag
Output parameter{out}	
-	-
Return value	
-	-

Example:

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

```
i2c_add_interrupt_flag_clear(I2C3, I2C_ADD_INT_FLAG_BERR);
```

3.19. IPA

The IPA provides a configurable and flexible image format conversion from one or two source image to the destination image. The IPA registers are listed in chapter [3.19.1](#), the IPA firmware functions are introduced in chapter [3.19.2](#).

3.19.1. Descriptions of Peripheral registers

IPA registers are listed in the table shown as below:

Table 3-690. IPA Registers

Registers	Descriptions
IPA_CTL	IPA control register
IPA_INTF	IPA interrupt flag register
IPA_INTC	IPA interrupt flag clear register
IPA_FMADDR	IPA foreground memory base address register
IPA_FLOFF	IPA foreground line offset register
IPA_BMADDR	IPA background memory base address register
IPA_BLOFF	IPA background line offset register
IPA_FPCTL	IPA foreground pixel control register
IPA_FPV	IPA foreground pixel value register
IPA_BPCTL	IPA background pixel control register
IPA_BPV	IPA background pixel value register
IPA_FLMADDR	IPA foreground LUT memory base address register
IPA_BLMADDR	IPA background LUT memory base address register
IPA_DPCTL	IPA destination pixel control register
IPA_DPV	IPA destination pixel value register
IPA_DMADDR	IPA destination memory base address register
IPA_DLOFF	IPA destination line offset register
IPA_IMS	IPA image size register
IPA_LM	IPA line mark register
IPA_ITCTL	IPA inter-timer control register

3.19.2. Descriptions of Peripheral functions

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

Table 3-691. IPA firmware function

Function name	Function description
ipa_deinit	deinitialize IPA
ipa_transfer_enable	enable IPA transfer
ipa_transfer_hangup_enable	enable IPA transfer hang up

Function name	Function description
ipa_transfer_hangup_disable	disable IPA transfer hang up
ipa_transfer_stop_enable	enable IPA transfer stop
ipa_transfer_stop_disable	disable IPA transfer stop
ipa_foreground_lut_loading_enable	enable IPA foreground LUT loading
ipa_background_lut_loading_enable	enable IPA background LUT loading
ipa_pixel_format_convert_mode_set	set pixel format convert mode, the function is invalid when the IPA transfer is enabled
ipa_foreground_struct_para_init	initialize the structure of IPA foreground parameter struct with the default values, it is suggested that call this function after an ipa_foreground_parameter_struct structure is defined
ipa_foreground_init	initialize foreground parameters
ipa_background_struct_para_init	initialize the structure of IPA background parameter struct with the default values, it is suggested that call this function after an ipa_background_parameter_struct structure is defined
ipa_background_init	initialize background parameters
ipa_destination_struct_para_init	initialize the structure of IPA destination parameter struct with the default values, it is suggested that call this function after an ipa_destination_parameter_struct structure is defined
ipa_destination_init	initialize destination parameters
ipa_foreground_lut_init	initialize IPA foreground LUT parameters
ipa_background_lut_init	initialize IPA background LUT parameters
ipa_line_mark_config	configure IPA line mark
ipa_inter_timer_config	IPA inter-timer enable or disable
ipa_interval_clock_num_config	configure the number of clock cycles interval
ipa_flag_get	get IPA flag status in IPA_INTF register
ipa_flag_clear	clear IPA flag in IPA_INTF register
ipa_interrupt_enable	enable IPA interrupt
ipa_interrupt_disable	disable IPA interrupt
ipa_interrupt_flag_get	get IPA interrupt flag
ipa_interrupt_flag_clear	clear IPA interrupt flag

Structure ipa_foreground_parameter_struct

Table 3-692. Structure ipa_foreground_parameter_struct

Member name	Function description
foreground_memaddr	foreground memory base address
foreground_lineoff	foreground line offset
foreground_prealpha	foreground pre-defined alpha value
foreground_alpha_algorithm	foreground alpha value calculation algorithm
foreground_pf	foreground pixel format
foreground_prered	foreground pre-defined red value
foreground_pregreen	foreground pre-defined green value

Member name	Function description
foreground_preblue	foreground pre-defined blue value

Structure ipa_background_parameter_struct

Table 3-693. Structure ipa_background_parameter_struct

Member name	Function description
background_memaddr	background memory base address
background_lineoff	background line offset
background_prealpha	background pre-defined alpha value
background_alpha_algorithm	background alpha value calculation algorithm
background_pf	background pixel format
background_prered	background pre-defined red value
background_pregreen	background pre-defined green value
background_preblue	background pre-defined blue value

Structure ipa_destination_parameter_struct

Table 3-694. Structure ipa_destination_parameter_struct

Member name	Function description
destination_memaddr	destination memory base address
destination_lineoff	destination line offset
destination_prealpha	destination pre-defined alpha value
destination_pf	destination pixel format
destination_prered	destination pre-defined red value
destination_pregreen	destination pre-defined green value
destination_preblue	destination pre-defined blue value
image_width	width of the image to be processed
image_height	height of the image to be processed

ipa_deinit

The description of ipa_deinit is shown as below:

Table 3-695. Function ipa_deinit

Function name	ipa_deinit
Function prototype	void ipa_deinit(void);
Function descriptions	deinitialize IPA registers
Precondition	-
The called functions	-
Input parameter{in}	
-	-
Output parameter{out}	
-	-
Return value	

-	-
---	---

Example:

```
/* deinitialize IPA */
ipa_deinit();
```

ipa_transfer_enable

The description of ipa_transfer_enable is shown as below:

Table 3-696. Function ipa_transfer_enable

Function name	ipa_transfer_enable
Function prototype	void ipa_transfer_enable(void);
Function descriptions	enable IPA transfer
Precondition	-
The called functions	-
Input parameter{in}	
-	-
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* enable IPA transfer */
ipa_transfer_enable();
```

ipa_transfer_hangup_enable

The description of ipa_transfer_hangup_enable is shown as below:

Table 3-697. Function ipa_transfer_hangup_enable

Function name	ipa_transfer_hangup_enable
Function prototype	void ipa_transfer_hangup_enable(void);
Function descriptions	enable IPA transfer hang up
Precondition	-
The called functions	-
Input parameter{in}	
-	-
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* enable IPA transfer hang up */
```

```
ipa_transfer_hangup_enable();
```

ipa_transfer_hangup_disable

The description of ipa_transfer_hangup_disable is shown as below:

Table 3-698. Function ipa_transfer_hangup_disable

Function name	ipa_transfer_hangup_disable
Function prototype	void ipa_transfer_hangup_disable(void);
Function descriptions	disable IPA transfer hang up
Precondition	-
The called functions	-
Input parameter{in}	
-	-
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* disable IPA transfer hang up */
```

```
ipa_transfer_hangup_disable();
```

ipa_transfer_stop_enable

The description of ipa_transfer_stop_enable is shown as below:

Table 3-699. Function ipa_transfer_stop_enable

Function name	ipa_transfer_stop_enable
Function prototype	void ipa_transfer_stop_enable(void);
Function descriptions	enable IPA transfer stop
Precondition	-
The called functions	-
Input parameter{in}	
-	-
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* enable IPA transfer stop */
```

```
ipa_transfer_stop_enable();
```

ipa_transfer_stop_disable

The description of ipa_transfer_stop_disable is shown as below:

Table 3-700. Function ipa_transfer_stop_disable

Function name	ipa_transfer_stop_disable
Function prototype	void ipa_transfer_stop_disable(void);
Function descriptions	disable IPA transfer stop
Precondition	-
The called functions	-
Input parameter{in}	
-	-
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* disable IPA transfer stop */
ipa_transfer_stop_disable();
```

ipa_foreground_lut_loading_enable

The description of ipa_foreground_lut_loading_enable is shown as below:

Table 3-701. Function ipa_foreground_lut_loading_enable

Function name	ipa_foreground_lut_loading_enable
Function prototype	void ipa_foreground_lut_loading_enable(void);
Function descriptions	enable IPA foreground LUT loading
Precondition	-
The called functions	-
Input parameter{in}	
-	-
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* enable IPA foreground LUT loading */
ipa_foreground_lut_loading_enable();
```

ipa_background_lut_loading_enable

The description of ipa_background_lut_loading_enable is shown as below:

Table 3-702. Function ipa_background_lut_loading_enable

Function name	ipa_background_lut_loading_enable
Function prototype	void ipa_background_lut_loading_enable(void);
Function descriptions	enable IPA background LUT loading
Precondition	-
The called functions	-
Input parameter{in}	
-	-
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* enable IPA background LUT loading */
ipa_background_lut_loading_enable();
```

ipa_pixel_format_convert_mode_set

The description of ipa_pixel_format_convert_mode_set is shown as below:

Table 3-703. Function ipa_pixel_format_convert_mode_set

Function name	ipa_pixel_format_convert_mode_set
Function prototype	void ipa_pixel_format_convert_mode_set(uint32_t pfcmm);
Function descriptions	set pixel format convert mode, the function is invalid when the IPA transfer is enabled
Precondition	-
The called functions	-
Input parameter{in}	
pfcmm	pixel format convert mode
<i>IPA_FGTMODE</i>	foreground memory to destination memory without pixel format convert
<i>IPA_FGTMODE_PF_CONVERT</i>	foreground memory to destination memory with pixel format convert
<i>IPA_FGBGMODE</i>	foreground and background memory to destination memory
<i>IPA_FILL_UP_DE</i>	fill up destination memory with specific color
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* configure foreground memory to destination memory with pixel format convert */
ipa_pixel_format_convert_mode_set(IPA_FGTMODE_PF_CONVERT);
```

ipa_foreground_struct_para_init

The description of ipa_foreground_struct_para_init is shown as below:

Table 3-704. Function ipa_foreground_struct_para_init

Function name	ipa_foreground_struct_para_init
Function prototype	void ipa_foreground_struct_para_init(ipa_foreground_parameter_struct* foreground_struct);
Function descriptions	initialize the structure of IPA foreground parameter struct with the default values, it is suggested that call this function after an ipa_foreground_parameter_struct structure is defined
Precondition	-
The called functions	-
Input parameter{in}	
*foreground_struct	a pointer to ipa_foreground_parameter_struct
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
ipa_foreground_parameter_struct fg_struct;

/* initialize the structure of IPA foreground parameter struct with the default values */
ipa_foreground_struct_para_init(&fg_struct);
```

ipa_foreground_init

The description of ipa_foreground_init is shown as below:

Table 3-705. Function ipa_foreground_init

Function name	ipa_foreground_init
Function prototype	void ipa_foreground_init(ipa_foreground_parameter_struct* foreground_struct);
Function descriptions	initialize foreground parameters
Precondition	-
The called functions	-
Input parameter{in}	
*foreground_struct	a pointer to ipa_foreground_parameter_struct
Output parameter{out}	
-	-
Return value	
-	-

Example:

```

ipa_foreground_parameter_struct ipa_fg_init_struct;

ipa_foreground_struct_para_init(&ipa_fg_init_struct);

/* configure IPA foreground */

ipa_fg_init_struct.foreground_memaddr = (uint32_t)&gBuffer;

ipa_fg_init_struct.foreground_pf = FOREGROUND_PPF_RGB565;

ipa_fg_init_struct.foreground_alpha_algorithm = IPA_FG_ALPHA_MODE_1;

ipa_fg_init_struct.foreground_prealpha = 0x75;

ipa_fg_init_struct.foreground_lineoff = 0x00;

ipa_fg_init_struct.foreground_preblue = 0x00;

ipa_fg_init_struct.foreground_pregreen = 0x00;

ipa_fg_init_struct.foreground_prered = 0x00;

/* foreground initialization */

ipa_foreground_init(&ipa_fg_init_struct);

```

ipa_background_struct_para_init

The description of ipa_background_struct_para_init is shown as below:

Table 3-706. Function ipa_background_struct_para_init

Function name	ipa_background_struct_para_init
Function prototype	void ipa_background_struct_para_init(ipa_background_parameter_struct* background_struct);
Function descriptions	initialize the structure of IPA background parameter struct with the default values , it is suggested that call this function after an ipa_background_parameter_struct structure is defined
Precondition	-
The called functions	-
Input parameter{in}	
*background_struct	a pointer to ipa_background_parameter_struct
Output parameter{out}	
-	-
Return value	
-	-

Example:

```

ipa_background_parameter_struct bg_struct;

/* initialize the structure of IPA background parameter struct with the default values */

ipa_background_struct_para_init(&bg_struct);

```

ipa_background_init

The description of ipa_background_init is shown as below:

Table 3-707. Function ipa_background_init

Function name	ipa_background_init
Function prototype	void ipa_background_init(ipa_background_parameter_struct* background_struct);
Function descriptions	initialize background parameters
Precondition	-
The called functions	-
Input parameter{in}	
*background_struct	a pointer to ipa_background_parameter_struct
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
ipa_background_parameter_struct  ipa_bg_init_struct;

ipa_background_struct_para_init(&ipa_bg_init_struct);

/* configure IPA background */

ipa_bg_init_struct.background_memaddr = (uint32_t)&gBuffer;

ipa_bg_init_struct.background_pf = BACKGROUND_PPF_RGB565;

ipa_bg_init_struct.background_alpha_algorithm = IPA_BG_ALPHA_MODE_0;

ipa_bg_init_struct.background_prealpha = 255;

ipa_bg_init_struct.background_lineoff = 0x00;

ipa_bg_init_struct.background_preblue = 0x00;

ipa_bg_init_struct.background_pregreen = 0x00;

ipa_bg_init_struct.background_prered = 0x00;

/* background initialization */

ipa_background_init(&ipa_bg_init_struct);
```

ipa_destination_struct_para_init

The description of ipa_destination_struct_para_init is shown as below:

Table 3-708. Function ipa_destination_struct_para_init

Function name	ipa_destination_struct_para_init
----------------------	----------------------------------

Function prototype	void ipa_destination_struct_para_init(ipa_destination_parameter_struct* destination_struct);
Function descriptions	initialize the structure of IPA destination parameter struct with the default values, it is suggested that call this function after an ipa_destination_parameter_struct structure is defined
Precondition	-
The called functions	-
Input parameter{in}	
*destination_struct	a pointer to ipa_destination_parameter_struct
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
ipa_destination_parameter_struct destination_struct;

/* initialize the structure of IPA destination parameter struct with the default values */
ipa_destination_struct_para_init(&destination_struct);
```

ipa_destination_init

The description of ipa_destination_init is shown as below:

Table 3-709. Function ipa_destination_init

Function name	ipa_destination_init
Function prototype	void ipa_destination_init(ipa_destination_parameter_struct* destination_struct);
Function descriptions	initialize destination parameters
Precondition	-
The called functions	-
Input parameter{in}	
*destination_struct	a pointer to ipa_destination_parameter_struct
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
ipa_destination_parameter_struct ipa_destination_init_struct;

ipa_destination_struct_para_init(&ipa_destination_init_struct);

/* configure destination pixel format */

ipa_destination_init_struct.destination_pf = IPA_DPF_RGB565;
```

```

/* configure destination memory base address */

ipa_destination_init_struct.destination_memaddr = (uint32_t)&gBuffer;

/* configure destination pre-defined alpha value RGB */

ipa_destination_init_struct.destination_pregreen = 0;

ipa_destination_init_struct.destination_preblue = 0;

ipa_destination_init_struct.destination_prered = 0;

ipa_destination_init_struct.destination_prealpha = 0;

/* configure destination line offset */

ipa_destination_init_struct.destination_lineoff = 0;

/* configure height of the image to be processed */

ipa_destination_init_struct.image_height = 160;

/* configure width of the image to be processed */

ipa_destination_init_struct.image_width = 229;

/* ipa destination initialization */

ipa_destination_init(&ipa_destination_init_struct);

```

ipa_foreground_lut_init

The description of ipa_foreground_lut_init is shown as below:

Table 3-710. Function ipa_foreground_lut_init

Function name	ipa_foreground_lut_init
Function prototype	void ipa_foreground_lut_init(uint8_t fg_lut_num, uint8_t fg_lut_pf, uint32_t fg_lut_addr);
Function descriptions	initialize IPA foreground LUT parameters
Precondition	-
The called functions	-
Input parameter{in}	
fg_lut_num	foreground LUT number of pixel
Input parameter{in}	
fg_lut_pf	foreground LUT pixel format
IPA_LUT_PF_ARGB8888	LUT pixel format is ARGB8888
IPA_LUT_PF_RGB888	LUT pixel format is RGB888
Input parameter{in}	
fg_lut_addr	foreground LUT memory base address. The address must be aligned to 8-bit, 16-bit or 32-bit corresponding with the foreground LUT pixel format.
Output parameter{out}	

-	-
Return value	
-	-

Example:

```
/* initialize the LUT foreground LUT parameters */
```

```
ipa_foreground_lut_init(1, IPA_LUT_PF_ARGB8888, 0x20002000);
```

ipa_background_lut_init

The description of ipa_background_lut_init is shown as below:

Table 3-711. Function ipa_background_lut_init

Function name	ipa_background_lut_init
Function prototype	void ipa_background_lut_init(uint8_t bg_lut_num, uint8_t bg_lut_pf, uint32_t bg_lut_addr);
Function descriptions	initialize IPA background LUT parameters
Precondition	-
The called functions	-
Input parameter{in}	
bg_lut_num	background LUT number of pixel
Input parameter{in}	
bg_lut_pf	background LUT pixel format
IPA_LUT_PF_ARGB8888	LUT pixel format is ARGB8888
IPA_LUT_PF_RGB888	LUT pixel format is RGB888
Input parameter{in}	
bg_lut_addr	background LUT memory base address. The address must be aligned to 8-bit, 16-bit or 32-bit corresponding with the foreground LUT pixel format.
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* initialize the LUT background LUT parameters */
```

```
ipa_background_lut_init(2, IPA_LUT_PF_RGB888, 0x20001000);
```

ipa_line_mark_config

The description of ipa_line_mark_config is shown as below:

Table 3-712. Function ipa_line_mark_config

Function name	ipa_line_mark_config
----------------------	----------------------

Function prototype	void ipa_line_mark_config(uint16_t line_num);
Function descriptions	configure IPA line mark
Precondition	-
The called functions	-
Input parameter{in}	
line_num	line number
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* configure line mark */
ipa_line_mark_config(10);
```

ipa_inter_timer_config

The description of ipa_inter_timer_config is shown as below:

Table 3-713. Function ipa_inter_timer_config

Function name	ipa_inter_timer_config
Function prototype	void ipa_inter_timer_config(uint8_t timer_cfg);
Function descriptions	inter-timer enable or disable
Precondition	-
The called functions	-
Input parameter{in}	
timer_cfg	inter-timer configuration
<i>IPA_INTER_TIMER_ENABLE</i>	enable inter-timer
<i>IPA_INTER_TIMER_DISABLE</i>	disable inter-timer
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* enable inter-timer */
ipa_inter_timer_config(IPA_INTER_TIMER_ENABLE);
```

ipa_interval_clock_num_config

The description of ipa_interval_clock_num_config is shown as below:

Table 3-714. Function ipa_interval_clock_num_config

Function name	ipa_interval_clock_num_config
Function prototype	void ipa_interval_clock_num_config(uint8_t clk_num);
Function descriptions	configure the number of clock cycles interval
Precondition	-
The called functions	-
Input parameter{in}	
clk_num	the number of clock cycles
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* configure the number of clock cycles interval and it has no meaning if ITEN is '0' */
ipa_interval_clock_num_config(1);
```

ipa_flag_get

The description of ipa_flag_get is shown as below:

Table 3-715. Function ipa_flag_get

Function name	ipa_flag_get
Function prototype	FlagStatus ipa_flag_get(uint32_t flag);
Function descriptions	get IPA flag status in IPA_INTF register
Precondition	-
The called functions	-
Input parameter{in}	
flag	IPA flags
<i>IPA_FLAG_TAE</i>	transfer access error interrupt flag
<i>IPA_FLAG_FTF</i>	full transfer finish interrupt flag
<i>IPA_FLAG_TLM</i>	transfer line mark interrupt flag
<i>IPA_FLAG_LAC</i>	LUT access conflict interrupt flag
<i>IPA_FLAG_LLF</i>	LUT loading finish interrupt flag
<i>IPA_FLAG_WCF</i>	wrong configuration interrupt flag
Output parameter{out}	
-	-
Return value	
FlagStatus	SET or RESET

Example:

```
/* wait for full transfer finish flag set */
while(ipa_flag_get(IPA_FLAG_FTF) == RESET);
```

ipa_flag_clear

The description of ipa_flag_clear is shown as below:

Table 3-716. Function ipa_flag_clear

Function name	ipa_flag_clear
Function prototype	void ipa_flag_clear(uint32_t flag);
Function descriptions	clear IPA flag in IPA_INTF register
Precondition	-
The called functions	-
Input parameter{in}	
flag	IPA flags
<i>IPA_FLAG_TAE</i>	transfer access error interrupt flag
<i>IPA_FLAG_FTF</i>	full transfer finish interrupt flag
<i>IPA_FLAG_TLM</i>	transfer line mark interrupt flag
<i>IPA_FLAG_LAC</i>	LUT access conflict interrupt flag
<i>IPA_FLAG_LLF</i>	LUT loading finish interrupt flag
<i>IPA_FLAG_WCF</i>	wrong configuration interrupt flag
Output parameter{out}	
-	-
Return value	
-	-

Example:

```

/* wait for full transfer finish flag set */
while(ipa_flag_get(IPA_FLAG_FTF) == RESET);

/* clear full transfer finish flag */
ipa_flag_clear(IPA_FLAG_FTF);

```

ipa_interrupt_enable

The description of ipa_interrupt_enable is shown as below:

Table 3-717. Function ipa_interrupt_enable

Function name	ipa_interrupt_enable
Function prototype	void ipa_interrupt_enable(uint32_t int_flag);
Function descriptions	enable IPA interrupt
Precondition	-
The called functions	-
Input parameter{in}	
int_flag	IPA interrupt flags
<i>IPA_INT_TAE</i>	transfer access error interrupt
<i>IPA_INT_FTF</i>	full transfer finish interrupt

<i>IPA_INT_TLM</i>	transfer line mark interrupt
<i>IPA_INT_LAC</i>	LUT access conflict interrupt
<i>IPA_INT_LLF</i>	LUT loading finish interrupt
<i>IPA_INT_WCF</i>	wrong configuration interrupt
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* enable full transfer finish interrupt */
```

```
ipa_interrupt_enable(IPA_INT_FTF);
```

ipa_interrupt_disable

The description of ipa_interrupt_disable is shown as below:

Table 3-718. Function ipa_interrupt_disable

Function name	ipa_interrupt_disable
Function prototype	void ipa_interrupt_disable(uint32_t int_flag);
Function descriptions	disable IPA interrupt
Precondition	-
The called functions	-
Input parameter{in}	
int_flag	IPA interrupt flags
<i>IPA_INT_TAE</i>	transfer access error interrupt
<i>IPA_INT_FTF</i>	full transfer finish interrupt
<i>IPA_INT_TLM</i>	transfer line mark interrupt
<i>IPA_INT_LAC</i>	LUT access conflict interrupt
<i>IPA_INT_LLF</i>	LUT loading finish interrupt
<i>IPA_INT_WCF</i>	wrong configuration interrupt
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* disable full transfer finish interrupt */
```

```
ipa_interrupt_disable(IPA_INT_FTF);
```

ipa_interrupt_flag_get

The description of ipa_interrupt_flag_get is shown as below:

Table 3-719. Function ipa_interrupt_flag_get

Function name	ipa_interrupt_flag_get
Function prototype	FlagStatus ipa_interrupt_flag_get(uint32_t int_flag);
Function descriptions	get IPA interrupt flag
Precondition	-
The called functions	-
Input parameter{in}	
int_flag	IPA interrupt flag flags
IPA_INT_FLAG_TAE	transfer access error interrupt flag
IPA_INT_FLAG_FTF	full transfer finish interrupt flag
IPA_INT_FLAG_TLM	transfer line mark interrupt flag
IPA_INT_FLAG_LAC	LUT access conflict interrupt flag
IPA_INT_FLAG_LLF	LUT loading finish interrupt flag
IPA_INT_FLAG_WCF	wrong configuration interrupt flag
Output parameter{out}	
-	-
Return value	
FlagStatus	SET or RESET

Example:

```
/* check whether full transfer finish interrupt flag is SET */
```

```
while(ipa_interrupt_flag_get(IPA_INT_FLAG_FTF) == RESET);
```

ipa_interrupt_flag_clear

The description of ipa_interrupt_flag_clear is shown as below:

Table 3-720. Function ipa_interrupt_flag_clear

Function name	ipa_interrupt_flag_clear
Function prototype	void ipa_interrupt_flag_clear(uint32_t int_flag);
Function descriptions	clear IPA interrupt flag
Precondition	-
The called functions	-
Input parameter{in}	
int_flag	IPA interrupt flag flags
IPA_INT_FLAG_TAE	transfer access error interrupt flag
IPA_INT_FLAG_FTF	full transfer finish interrupt flag
IPA_INT_FLAG_TLM	transfer line mark interrupt flag
IPA_INT_FLAG_LAC	LUT access conflict interrupt flag
IPA_INT_FLAG_LLF	LUT loading finish interrupt flag
IPA_INT_FLAG_WCF	wrong configuration interrupt flag
Output parameter{out}	
-	-
Return value	

-	-
---	---

Example:

```
if(ipa_interrupt_flag_get(IPA_INT_FLAG_FTF) != RESET){
    /* clear full transfer finish interrupt flag */
    ipa_interrupt_flag_clear(IPA_INT_FLAG_FTF);
}
```

3.20. IREF

IREF is a software package that provide the interfaces for IREF. The IREF registers are listed in chapter [3.20.1](#), the IREF firmware functions are introduced in chapter [3.20.2](#).

3.20.1. Descriptions of Peripheral registers

Table 3-721. IREF Registers

Registers	Descriptions
IREF_CTL	Control register

3.20.2. Descriptions of Peripheral functions

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

Table 3-722. IREF firmware function

Function name	Function description
iref_deinit	deinitialize IREF
iref_enable	enable IREF
iref_disable	disable IREF
iref_mode_set	set IREF mode
iref_sink_set	set IREF sink current mode
iref_precision_trim_value_set	set IREF current precision trim value
iref_step_data_config	set IREF step data

iref_deinit

The description of iref_deinit is shown as below:

Table 3-723. Function iref_deinit

Function name	iref_deinit
Function prototype	void iref_deinit (void);
Function descriptions	deinitialize IREF
Precondition	-

The called functions	rcu_periph_reset_enable / rcu_periph_reset_disable
Input parameter{in}	
-	-
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* reset IREF */
```

```
iref_deinit();
```

iref_enable

The description of iref_enable is shown as below:

Table 3-724. Function iref_enable

Function name	iref_enable
Function prototype	void iref_enable (void);
Function descriptions	Enable IREF
Precondition	-
The called functions	-
Input parameter{in}	
-	-
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* enable IREF */
```

```
iref_enable();
```

iref_disable

The description of iref_disable is shown as below:

Table 3-725. Function iref_disable

Function name	iref_disable
Function prototype	void iref_disable(void);
Function	Disable IREF

descriptions	
Precondition	-
The called functions	-
Input parameter{in}	
-	-
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* disable IREF */
iref_disable();
```

iref_mode_set

The description of iref_mode_set is shown as below:

Table 3-726. Function iref_mode_set

Function name	iref_mode_set
Function prototype	void iref_mode_set(uint32_t step);
Function descriptions	set IREF mode
Precondition	-
The called functions	-
Input parameter{in}	
step	IREF mode
<i>IREF_MODE_LOW_POWER</i>	Low power mode,1uA step
<i>IREF_MODE_HIGH_CURRENT</i>	High current mode,8uA step
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* set IREF mode */
iref_mode_set(IREF_MODE_LOW_POWER);
```

iref_sink_set

The description of iref_sink_set is shown as below:

Table 3-727. Function iref_sink_set

Function name	iref_sink_set
Function prototype	void iref_sink_set(uint32_t sinkmode);
Function descriptions	set IREF sink current mode
Precondition	-
The called functions	-
Input parameter{in}	
sinkmode	Sink mode
<i>IREF_SOURCE_CURRENT</i>	source current
<i>IREF_SINK_CURRENT</i>	sink current
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* set IREF source current mode */
iref_sink_set(IREF_SOURCE_CURRENT);
```

iref_precision_trim_value_set

The description of iref_precision_trim_value_set is shown as below:

Table 3-728. Function iref_precision_trim_value_set

Function name	iref_precision_trim_value_set
Function prototype	void iref_precision_trim_value_set (uint32_t precisiontrim);
Function descriptions	set IREF current precision trim value
Precondition	-
The called functions	-
Input parameter{in}	
precisiontrim	IREF precision_trim_value
<i>IREF_CUR_PRECISION_TRIM_X</i>	(X=0..31): (-15.. +16)%
Output parameter{out}	
-	-

Return value	
-	-

Example:

```
/* set IREF precision trim value -15% */
```

```
iref_precision_trim_value_set(IREF_CUR_PRECISION_TRIM_0);
```

iref_step_data_config

The description of iref_step_data_config is shown as below:

Table 3-729. Function iref_step_data_config

Function name	iref_step_data_config
Function prototype	void iref_step_data_config (uint32_t stepdata);
Function descriptions	set IREF step data
Precondition	-
The called functions	-
Input parameter{in}	
stepdata	Step data
IREF_CUR_STEP_DATA_X	(X=0..63): step*x
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* set IREF step data */
```

```
iref_step_data_config(IREF_CUR_STEP_DATA_0);
```

3.21. MISC

MISC is a software package that provide the interfaces for NVIC and SysTick. The NVIC and SysTick registers are listed in chapter [3.21.1](#), the MISC firmware functions are introduced in chapter [3.21.2](#).

3.21.1. Descriptions of Peripheral registers

Table 3-730. NVIC Registers

Registers	Descriptions
ISER ⁽¹⁾	Interrupt Set Enable Register

Registers	Descriptions
ICER ⁽¹⁾	Interrupt Clear Enable Register
ISPR ⁽¹⁾	Interrupt Set Pending Register
ICPR ⁽¹⁾	Interrupt Clear Pending Register
IABR ⁽¹⁾	Interrupt Active bit Register
IP ⁽¹⁾	Interrupt Priority Register
STIR ⁽¹⁾	Software Trigger Interrupt Register
CPUID ⁽²⁾	CPUID Base Register
ICSR ⁽²⁾	Interrupt Control and State Register
VTOR ⁽²⁾	Vector Table Offset Register
AIRCR ⁽²⁾	Application Interrupt and Reset Control Register
SCR ⁽²⁾	System Control Register
CCR ⁽²⁾	Configuration Control Register
SHP ⁽²⁾	System Handlers Priority Registers
SHCSR ⁽²⁾	System Handler Control and State Register
CFSR ⁽²⁾	Configurable Fault Status Register
HFSR ⁽²⁾	HardFault Status Register
DFSR ⁽²⁾	Debug Fault Status Register
MMFAR ⁽²⁾	MemManage Fault Address Register
BFAR ⁽²⁾	BusFault Address Register
AFSR ⁽²⁾	Auxiliary Fault Status Register
PFR ⁽²⁾	Processor Feature Register
DFR ⁽²⁾	Debug Feature Register
ADR ⁽²⁾	Auxiliary Feature Register
MMFR ⁽²⁾	Memory Model Feature Register
ISAR ⁽²⁾	Instruction Set Attributes Register
CPACR ⁽²⁾	Coprocessor Access Control Register

1. refer to the structure NVIC_Type, is defined in the core_cm33.h file

2. refer to the structure SCB_Type, is defined in the core_cm33.h file

Table 3-731. SysTick Registers

Registers	Descriptions
CTRL ⁽¹⁾	SysTick Control and Status Register
LOAD ⁽¹⁾	SysTick Reload Value Register
VAL ⁽¹⁾	SysTick Current Value Register
CALIB ⁽¹⁾	SysTick Calibration Register

1. refer to the structure SysTick_Type, is defined in the core_cm33.h file

3.21.2. Descriptions of Peripheral functions

Enum IRQn_Type

Table 3-732. Enum IRQn_Type

Member name	Function description
WWDGT_IRQn	window watchDog timer interrupt
LVD_IRQn	LVD through EXTI line detect interrupt
TAMPER_STAMP_IRQn	tamper and timestamp through EXTI line detect
RTC_WKUP_IRQn	RTC wakeup through EXTI line interrupt
FMC_IRQn	FMC interrupt
RCU_CTC_IRQn	RCU and CTC interrupt
EXTI0_IRQn	EXTI line 0 interrupts
EXTI1_IRQn	EXTI line 1 interrupts
EXTI2_IRQn	EXTI line 2 interrupts
EXTI3_IRQn	EXTI line 3 interrupts
EXTI4_IRQn	EXTI line 4 interrupts
DMA0_Channel0_IRQn	DMA0 channel0 interrupt
DMA0_Channel1_IRQn	DMA0 channel1 interrupt
DMA0_Channel2_IRQn	DMA0 channel2 interrupt
DMA0_Channel3_IRQn	DMA0 channel3 interrupt
DMA0_Channel4_IRQn	DMA0 channel4 interrupt
DMA0_Channel5_IRQn	DMA0 channel5 interrupt
DMA0_Channel6_IRQn	DMA0 channel6 interrupt
ADC_IRQn	ADC interrupt
CAN0_TX_IRQn	CAN0 transmit interrupts
CAN0_RX0_IRQn	CAN0 receive0 interrupts
CAN0_RX1_IRQn	CAN0 receive1 interrupts
CAN0_EWMC_IRQn	CAN0 EWMC interrupts
EXTI5_9_IRQn	EXTI[9:5] interrupts
TIMER0_BRK_TIMER8_IRQn	TIMER0 break and TIMER8 interrupts
TIMER0_UP_TIMER9_IRQn	TIMER0 update and TIMER9 interrupts
TIMER0_TRG_CMT_TIMER10_IRQn	TIMER0 trigger and commutation and TIMER10 interrupts
TIMER0_Channel_IRQn	TIMER0 channel capture compare interrupt
TIMER1_IRQn	TIMER1 interrupt
TIMER2_IRQn	TIMER2 interrupt
TIMER3_IRQn	TIMER3 interrupt
I2C0_EV_IRQn	I2C0 event interrupt
I2C0_ER_IRQn	I2C0 error interrupt
I2C1_EV_IRQn	I2C1 event interrupt
I2C1_ER_IRQn	I2C1 error interrupt
SPI0_IRQn	SPI0 interrupt

Member name	Function description
SPI1_IRQn	SPI1 interrupt
USART0_IRQn	USART0 interrupt
USART1_IRQn	USART1 interrupt
USART2_IRQn	USART2 interrupt
EXTI10_15_IRQn	EXTI[15:10] interrupts
RTC_Alarm_IRQn	RTC alarm interrupt
USBFS_WKUP_IRQn	USBFS wakeup interrupt
TIMER7_BRK_TIMER11_IRQn	TIMER7 break and TIMER11 interrupts
TIMER7_UP_TIMER12_IRQn	TIMER7 update and TIMER12 interrupts
TIMER7_TRG_CMT_TIMER13_IRQn	TIMER7 trigger and commutation and TIMER13 interrupts
TIMER7_Channel_IRQn	TIMER7 channel capture compare interrupt
DMA0_Channel7_IRQn	DMA0 channel7 interrupt
EXMC_IRQn	EXMC global interrupt
SDIO_IRQn	SDIO global interrupt
TIMER4_IRQn	TIMER4 global interrupt
SPI2_IRQn	SPI2 global interrupt
UART3_IRQn	UART3 global interrupt
UART4_IRQn	UART4 global interrupt
TIMER5_DAC_IRQn	TIMER5 global interrupt and DAC0_OUT0 DAC0_OUT1 underrun error interrupts
TIMER6_IRQn	TIMER6 global interrupt
DMA1_Channel0_IRQn	DMA1 channel0 global interrupt
DMA1_Channel1_IRQn	DMA1 channel1 global interrupt
DMA1_Channel2_IRQn	DMA1 channel2 global interrupt
DMA1_Channel3_IRQn	DMA1 channel3 global interrupt
DMA1_Channel4_IRQn	DMA1 channel4 global interrupt
ENET_IRQn	ENET interrupt
ENET_WKUP_IRQn	ENET wakeup interrupt
CAN1_TX_IRQn	CAN1 transmit interrupt
CAN1_RX0_IRQn	CAN1 receive0 interrupt
CAN1_RX1_IRQn	CAN1 receive1 interrupt
CAN1_EWMC_IRQn	CAN1 EWMC interrupt
USBFS_IRQn	USBFS interrupt
DMA1_Channel5_IRQn	DMA1 Channel5 interrupt
DMA1_Channel6_IRQn	DMA1 Channel6 interrupt
DMA1_Channel7_IRQn	DMA1 Channel7 interrupt
USART5_IRQn	USART5 interrupt
I2C2_EV_IRQn	I2C2 event interrupt
I2C2_ER_IRQn	I2C3 error interrupt
USBHS_EP1_Out_IRQn	USBHS endpoint 1 out interrupt
USBHS_EP1_In_IRQn	USBHS endpoint 1 in interrupt
USBHS_WKUP_IRQn	USBHS Wakeup interrupt

Member name	Function description
USBHS_IRQn	USBHS interrupt
DCI_IRQn	DCI interrupt
TRNG_IRQn	TRNG interrupt
FPU_IRQn	FPU interrupt
UART6_IRQn	UART6 interrupt
UART7_IRQn	UART7 interrupt
SPI3_IRQn	SPI3 interrupt
SPI4_IRQn	SPI4 interrupt
SPI5_IRQn	SPI5 interrupt
SAI_IRQn	SAI interrupt
TLI_IRQn	TLI interrupt
TLI_ER_IRQn	TLI error interrupt
IPA_IRQn	IPA interrupt
PKCAU_IRQn	PKCAU interrupt
I2C3_EV_IRQn	I2C3 Event interrupt
I2C3_ER_IRQn	I2C3 Error interrupt
I2C4_EV_IRQn	I2C4 Event interrupt
I2C4_ER_IRQn	I2C4 Error interrupt
I2C5_EV_IRQn	I2C5 Event interrupt
I2C5_ER_IRQn	I2C5 Error interrupt
I2C3_WKUP_IRQn	I2C3 Wakeup through EXTI Line interrupt
I2C4_WKUP_IRQn	I2C4 Wakeup through EXTI Line interrupt
I2C5_WKUP_IRQn	I2C5 Wakeup through EXTI Line interrupt
SYSCFG_SRAM_ECC_ER_IRQn	SYSCFG SRAM ECC Error interrupt
HAU_IRQn	HAU interrupt
CAU_IRQn	CAU interrupt

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

Table 3-733. MISC firmware function

Function name	Function description
<code>nvic_priority_group_set</code>	set the priority group
<code>nvic_irq_enable</code>	enable NVIC interrupt request
<code>nvic_irq_disable</code>	disable NVIC interrupt request
<code>nvic_vector_table_set</code>	set the NVIC vector table address
<code>system_lowpower_set</code>	set the state of the low power mode
<code>system_lowpower_reset</code>	reset the state of the low power mode
<code>systick_clksource_set</code>	set the systick clock source

`nvic_priority_group_set`

The description of `nvic_priority_group_set` is shown as below:

Table 3-734. Function `nvic_priority_group_set`

Function name	<code>nvic_priority_group_set</code>
Function prototype	<code>void nvic_priority_group_set(uint32_t nvic_prigroup);</code>
Function descriptions	set the priority group
Precondition	-
The called functions	-
Input parameter{in}	
<code>nvic_prigroup</code>	priority group
<code>NVIC_PRIGROUP_PRE0_SUB4</code>	0 bits for pre-emption priority 4 bits for subpriority
<code>NVIC_PRIGROUP_PRE1_SUB3</code>	1 bits for pre-emption priority 3 bits for subpriority
<code>NVIC_PRIGROUP_PRE2_SUB2</code>	2 bits for pre-emption priority 2 bits for subpriority
<code>NVIC_PRIGROUP_PRE3_SUB1</code>	3 bits for pre-emption priority 1 bits for subpriority
<code>NVIC_PRIGROUP_PRE4_SUB0</code>	4 bits for pre-emption priority 0 bits for subpriority
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* priority group configuration , 0 bits for pre-emption priority 4 bits for subpriority */
nvic_priority_group_set(NVIC_PRIGROUP_PRE0_SUB4);
```

`nvic_irq_enable`

The description of `nvic_irq_enable` is shown as below:

Table 3-735. Function `nvic_irq_enable`

Function name	<code>nvic_irq_enable</code>
Function prototype	<code>void nvic_irq_enable(uint8_t nvic_irq, uint8_t nvic_irq_pre_priority, uint8_t nvic_irq_sub_priority);</code>
Function descriptions	enable NVIC interrupt request
Precondition	-
The called functions	<code>nvic_priority_group_set</code>
Input parameter{in}	
<code>nvic_irq</code>	NVIC interrupt, refer to enum Enum IRQn_Type
Input parameter{in}	
<code>nvic_irq_pre_priority</code>	the pre-emption priority needed to set (0~4)
Input parameter{in}	
<code>nvic_irq_sub_priority</code>	the subpriority needed to set (0~4)

Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* enable window watchDog timer interrupt , pre-emption priority is 1, subpriority is 1 */
nvic_irq_enable(WWDGT_IRQn, 1, 1);
```

nvic_irq_disable

The description of nvic_irq_disable is shown as below:

Table 3-736. Function nvic_irq_disable

Function name	nvic_irq_disable
Function prototype	void nvic_irq_disable(uint8_t nvic_irq);
Function descriptions	disable NVIC interrupt request
Precondition	-
The called functions	-
Input parameter{in}	
nvic_irq	NVIC interrupt, refer to enum Enum IRQn_Type
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* disable window watchDog timer interrupt */
nvic_irq_disable(WWDGT_IRQn);
```

nvic_vector_table_set

The description of nvic_vector_table_set is shown as below:

Table 3-737. Function nvic_vector_table_set

Function name	nvic_vector_table_set
Function prototype	void nvic_vector_table_set(uint32_t nvic_vect_tab, uint32_t offset);
Function descriptions	set the NVIC vector table address
Precondition	-
The called functions	-
Input parameter{in}	
nvic_vect_tab	the RAM or FLASH base address
NVIC_VECTTAB_RAM	RAM base address
NVIC_VECTTAB_FLASH	Flash base address

Input parameter{in}	
offset	Vector Table offset (vector table start address= base address+offset)
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* 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 3-738. Function system_lowpower_set

Function name	system_lowpower_set
Function prototype	void system_lowpower_set(uint8_t lowpower_mode);
Function descriptions	set the state of the low power mode
Precondition	-
The called functions	-
Input parameter{in}	
lowpower_mode	the low power mode state
SCB_LPM_SLEEP_EXIT_ISR	if chose this para, the system always enter low power mode by exiting from ISR
SCB_LPM_DEEPSLEEP	if chose this para, the system will enter the DEEPSLEEP mode
SCB_LPM_WAKE_BY_ALL_INT	if chose this para, the lowpower mode can be woke up by all the enable and disable interrupts
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* 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 3-739. Function system_lowpower_reset

Function name	system_lowpower_reset
Function prototype	void system_lowpower_reset(uint8_t lowpower_mode);

Function descriptions	reset the state of the low power mode
Precondition	-
The called functions	-
Input parameter{in}	
lowpower_mode	the low power mode state
<i>SCB_LPM_SLEEP_EXIT_ISR</i>	if chose this para, the system will exit low power mode by exiting from ISR
<i>SCB_LPM_DEEPSLEEP</i>	if chose this para, the system will enter the SLEEP mode
<i>SCB_LPM_WAKE_BY_ALL_INT</i>	if chose this para, the lowpower mode only can be woke up by the enable interrupts
Output parameter{out}	
-	-
Return value	
-	-

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 3-740. Function systick_clksource_set

Function name	systick_clksource_set
Function prototype	void systick_clksource_set(uint32_t systick_clksource);
Function descriptions	set the systick clock source
Precondition	-
The called functions	-
Input parameter{in}	
systick_clksource	the systick clock source needed to choose
<i>SYSTICK_CLKSOURCE_HCLK</i>	systick clock source is from HCLK
<i>SYSTICK_CLKSOURCE_HCLK_DIV8</i>	systick clock source is from HCLK/8
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* systick clock source is HCLK/8 */
systick_clksource_set(SYSTICK_CLKSOURCE_HCLK_DIV8);
```

3.22. PKCAU

The Public Key Cryptographic Acceleration Unit (PKCAU) can accelerate RSA (Rivest, Shamir and Adleman), Diffie-Hellmann (DH key exchange) and ECC (elliptic curve cryptography) in GF(p) (Galois domain). The PKCAU registers are listed in chapter [3.22.1](#), the PKCAU firmware functions are introduced in chapter [3.22.2](#).

3.22.1. Descriptions of Peripheral registers

PKCAU registers are listed in the table shown as below:

Table 3-741. PKCAU Registers

Registers	Descriptions
PKCAU_CTL	Control register
PKCAU_STAT	Status register
PKCAU_STATC	Status clear register

3.22.2. Descriptions of Peripheral functions

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

Table 3-742. PKCAU firmware function

Function name	Function description
pkcau_deinit	reset PKCAU
pkcau_mont_struct_para_init	initialize montgomery parameter structure with a default value
pkcau_mod_struct_para_init	initialize modular parameter structure with a default value
pkcau_mod_exp_struct_para_init	initialize modular exponentiation parameter structure with a default value
pkcau_arithmetic_struct_para_init	initialize arithmetic parameter structure with a default value
pkcau_crt_struct_para_init	initialize CRT parameter structure with a default value
pkcau_ec_group_struct_para_init	initialize ECC curve parameter structure with a default value
pkcau_point_struct_para_init	initialize point parameter structure with a default value
pkcau_signature_struct_para_init	initialize signature parameter structure with a default value
pkcau_hash_struct_para_init	initialize hash parameter structure with a default value
pkcau_ecc_out_struct_para_init	initialize ecc output parameter structure with a default value
pkcau_enable	enable PKCAU
pkcau_disable	disable PKCAU
pkcau_start	start operation
pkcau_mode_set	configure the PKCAU operation mode
pkcau_mont_param_operation	execute montgomery parameter operation
pkcau_mod_operation	execute modular operation, include modular addition, modular subtraction and montgomery multiplication
pkcau_mod_exp_operation	execute modular exponentiation operation
pkcau_mod_inver_operation	execute modular inversion operation

Function name	Function description
pkcau_mod_reduc_operation	execute modular reduction operation
pkcau_arithmetic_operation	execute arithmetic addition operation
pkcau_crt_exp_operation	execute RSA CRT exponentiation operation
pkcau_point_check_operation	execute point check operation
pkcau_point_mul_operation	execute point multiplication operation
pkcau_ecdsa_sign_operation	execute ECDSA sign operation
pkcau_ecdsa_verification_operation	execute ECDSA verify operation
pkcau_flag_get	get PKCAU flag status
pkcau_flag_clear	clear PKCAU flag status
pkcau_interrupt_enable	enable PKCAU interrupt
pkcau_interrupt_disable	disable PKCAU interrupt
pkcau_interrupt_flag_get	get PKCAU interrupt flag status
pkcau_interrupt_flag_clear	clear PKCAU interrupt flag status

Structure pkcau_mont_parameter_struct

Table 3-743. Structure pkcau_mont_parameter_struct

Member name	Function description
modulus_n	modulus value n
modulus_n_len	modulus length in byte

Structure pkcau_mod_parameter_struct

Table 3-744. Structure pkcau_mod_parameter_struct

Member name	Function description
opr_d_a	operand A
opr_d_a_len	operand A length in byte
opr_d_b	operand B
opr_d_b_len	operand b length in byte
modulus_n	modulus value n
modulus_n_len	modulus length in byte

Structure pkcau_mod_exp_parameter_struct

Table 3-745. Structure pkcau_mod_exp_parameter_struct

Member name	Function description
opr_d_a	operand A
opr_d_a_len	operand A length in byte
exp_e	exponent e
e_len	exponent length in byte
modulus_n	modulus n
modulus_n_len	modulus length in byte
mont_para	montgomery parameter R2 mod n

Member name	Function description
mont_para_len	montgomery parameter length in byte

Structure pkcau_arithmetic_parameter_struct

Table 3-746. Structure pkcau_arithmetic_parameter_struct

Member name	Function description
oprnd_a	operand A
oprnd_a_len	length of operand A in byte
oprnd_b	operand B
oprnd_b_len	length of operand B in byte

Structure pkcau_crt_parameter_struct

Table 3-747. Structure pkcau_crt_parameter_struct

Member name	Function description
oprnd_a	operand A
oprnd_a_len	length of operand A in byte
oprnd_dp	operand dp
oprnd_dp_len	length of operand dp in byte
oprnd_dq	operand dq
oprnd_dq_len	length of operand dq in byte
oprnd_qinv	operand qinv
oprnd_qinv_len	length of operand qinv in byte
oprnd_p	prime operand p
oprnd_p_len	length of operand p in byte
oprnd_q	prime operand q
oprnd_q_len	length of operand q in byte

Structure pkcau_ec_group_parameter_struct

Table 3-748. Structure pkcau_ec_group_parameter_struct

Member name	Function description
modulus_p	curve modulus p
modulus_p_len	curve modulus p length in byte
coff_a	curve coefficient a
coff_a_len	curve coefficient a length in byte
coff_b	curve coefficient b
coff_b_len	curve coefficient b length in byte
base_point_x	curve base point coordinate x
base_point_x_len	curve base point coordinate x length in byte
base_point_y	curve base point coordinate y
base_point_y_len	curve base point coordinate y length in byte
order_n	curve prime order n

Member name	Function description
order_n_len	curve prime order n length in byte
a_sign	curve coefficient a sign
multi_k	scalar multiplier k
multi_k_len	length of scalar multiplier k
integer_k	integer k
integer_k_len	integer k length in byte
private_key_d	private key d
private_key_d_len	private key d length in byte
mont_para	montgomery parameter R2 mod n
mont_para_len	montgomery parameter R2 mod n length in byte

Structure pkcau_point_parameter_struct

Table 3-749. Structure pkcau_point_parameter_struct

Member name	Function description
point_x	point coordinate x
point_x_len	point coordinate x length in byte
point_y	point coordinate y
point_y_len	point coordinate y length in byte

Structure pkcau_signature_parameter_struct

Table 3-750. Structure pkcau_signature_parameter_struct

Member name	Function description
sign_r	signature part r
sign_r_len	signature part r length in byte
sign_s	signature part s
sign_s_len	signature part s length in byte

Structure pkcau_hash_parameter_struct

Table 3-751. Structure pkcau_hash_parameter_struct

Member name	Function description
hash_z	hash value z
hash_z_len	hash value z length in byte

Structure pkcau_ecc_out_struct

Table 3-752. Structure pkcau_ecc_out_struct

Member name	Function description
sign_extra	flag of extended ECDSA sign (extra outputs)
sign_r	signature part r
sign_s	signature part s
point_x	point coordinate x

Member name	Function description
point_y	point coordinate y

pkcau_deinit

The description of pkcau_deinit is shown as below:

Table 3-753. Function pkcau_deinit

Function name	pkcau_deinit
Function prototype	void pkcau_deinit(void);
Function descriptions	reset PKCAU
Precondition	-
The called functions	rcu_periph_reset_enable / rcu_periph_reset_disable
Input parameter{in}	
-	-
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* reset PKCAU */
pkcau_deinit();
```

pkcau_mont_struct_para_init

The description of pkcau_mont_struct_para_init is shown as below:

Table 3-754. Function pkcau_mont_struct_para_init

Function name	pkcau_mont_struct_para_init
Function prototype	void pkcau_mont_struct_para_init(pkcau_mont_parameter_struct* init_para);
Function descriptions	initialize montgomery parameter structure with a default value
Precondition	-
The called functions	-
Input parameter{in}	
-	-
Output parameter{out}	
init_para	montgomery parameter struct, the structure members can refer to Table 3-743. Structure pkcau_mont_parameter_struct.
Return value	
-	-

Example:

```
/* initialize PKCAU montgomery parameter struct with a default value */
```

```
pkcau_mont_parameter_struct pkcau_mont_parameter;

pkcau_mont_struct_para_init(&pkcau_mont_parameter);
```

pkcau_mod_struct_para_init

The description of pkcau_mod_struct_para_init is shown as below:

Table 3-755. Function pkcau_mod_struct_para_init

Function name	pkcau_mod_struct_para_init
Function prototype	void pkcau_mod_struct_para_init(pkcau_mod_parameter_struct* init_para);
Function descriptions	initialize modular parameter structure with a default value
Precondition	-
The called functions	-
Input parameter{in}	
-	-
Output parameter{out}	
init_para	modular parameter struct, the structure members can refer to Table 3-744. Structure pkcau_mod_parameter_struct.
Return value	
-	-

Example:

```
/* initialize PKCAU modular parameter struct with a default value */

pkcau_mod_parameter_struct pkcau_mod_parameter;

pkcau_mod_struct_para_init(&pkcau_mod_parameter);
```

pkcau_mod_exp_struct_para_init

The description of pkcau_mod_exp_struct_para_init is shown as below:

Table 3-756. Function pkcau_mod_exp_struct_para_init

Function name	pkcau_mod_exp_struct_para_init
Function prototype	void pkcau_mod_exp_struct_para_init(pkcau_mod_exp_parameter_struct* init_para);
Function descriptions	initialize modular exponentation parameter structure with a default value
Precondition	-
The called functions	-
Input parameter{in}	
-	-
Output parameter{out}	
init_para	modular exponentation parameter struct, the structure members can refer to Table 3-745. Structure pkcau_mod_exp_parameter_struct.
Return value	
-	-

Example:

```
/* initialize PKCAU modular exponentiation parameter struct with a default value */
pkcau_mod_exp_parameter_struct pkcau_mod_exp_parameter;
pkcau_mod_exp_struct_para_init(&pkcau_mod_exp_parameter);
```

pkcau_arithmetic_struct_para_init

The description of pkcau_arithmetic_struct_para_init is shown as below:

Table 3-757. Function pkcau_arithmetic_struct_para_init

Function name	pkcau_arithmetic_struct_para_init
Function prototype	void pkcau_arithmetic_struct_para_init(pkcau_arithmetic_parameter_struct* init_para);
Function descriptions	initialize arithmetic parameter structure with a default value
Precondition	-
The called functions	-
Input parameter{in}	
-	-
Output parameter{out}	
init_para	arithmetic parameter struct, the structure members can refer to Table 3-746. Structure pkcau_arithmetic_parameter_struct.
Return value	
-	-

Example:

```
/* initialize PKCAU arithmetic parameter struct struct with a default value */
pkcau_arithmetic_parameter_struct pkcau_arithmetic_parameter;
pkcau_arithmetic_struct_para_init(&pkcau_arithmetic_parameter);
```

pkcau_crt_struct_para_init

The description of pkcau_crt_struct_para_init is shown as below:

Table 3-758. Function pkcau_crt_struct_para_init

Function name	pkcau_crt_struct_para_init
Function prototype	void pkcau_crt_struct_para_init(pkcau_crt_parameter_struct* init_para);
Function descriptions	initialize CRT parameter structure with a default value
Precondition	-
The called functions	-
Input parameter{in}	
-	-
Output parameter{out}	
init_para	CRT parameter struct, the structure members can refer to Table 3-747.

	Structure pkcau crt parameter struct.
Return value	
-	-

Example:

```
/* initialize PKCAU CRT parameter struct with a default value */
```

```
pkcau_crt_parameter_struct pkcau_crt_parameter;
```

```
pkcau_crt_struct_para_init(&pkcau_crt_parameter);
```

pkcau_ec_group_struct_para_init

The description of pkcau_ec_group_struct_para_init is shown as below:

Table 3-759. Function pkcau_ec_group_struct_para_init

Function name	pkcau_ec_group_struct_para_init
Function prototype	void pkcau_ec_group_struct_para_init(pkcau_ec_group_parameter_struct* init_para);
Function descriptions	initialize ECC curve parameter structure with a default value
Precondition	-
The called functions	-
Input parameter{in}	
-	-
Output parameter{out}	
init_para	ECC curve parameter struct, the structure members can refer to Table 3-748. Structure pkcau ec group parameter struct.
Return value	
-	-

Example:

```
/* initialize PKCAU ECC curve parameter struct with a default value */
```

```
pkcau_ec_group_parameter_struct pkcau_ec_group_parameter;
```

```
pkcau_ec_group_struct_para_init(&pkcau_ec_group_parameter);
```

pkcau_point_struct_para_init

The description of pkcau_point_struct_para_init is shown as below:

Table 3-760. Function pkcau_point_struct_para_init

Function name	pkcau_point_struct_para_init
Function prototype	void pkcau_point_struct_para_init(pkcau_point_parameter_struct* init_para);
Function descriptions	initialize point parameter structure with a default value
Precondition	-

The called functions	-
Input parameter{in}	
-	-
Output parameter{out}	
init_para	point parameter struct, the structure members can refer to Table 3-749. Structure pkcau_point_parameter_struct.
Return value	
-	-

Example:

```
/* initialize PKCAU point parameter struct with a default value */
```

```
pkcau_point_parameter_struct pkcau_point_parameter;
```

```
pkcau_point_struct_para_init(&pkcau_point_parameter);
```

pkcau_signature_struct_para_init

The description of pkcau_signature_struct_para_init is shown as below:

Table 3-761. Function pkcau_signature_struct_para_init

Function name	pkcau_signature_struct_para_init
Function prototype	void pkcau_signature_struct_para_init(pkcau_signature_parameter_struct* init_para);
Function descriptions	initialize signature parameter structure with a default value
Precondition	-
The called functions	-
Input parameter{in}	
-	-
Output parameter{out}	
init_para	signature parameter struct, the structure members can refer to Table 3-750. Structure pkcau_signature_parameter_struct.
Return value	
-	-

Example:

```
/* initialize PKCAU signature parameter struct with a default value */
```

```
pkcau_signature_parameter_struct pkcau_signature_parameter;
```

```
pkcau_signature_struct_para_init(&pkcau_signature_parameter);
```

pkcau_hash_struct_para_init

The description of pkcau_hash_struct_para_init is shown as below:

Table 3-762. Function pkcau_hash_struct_para_init

Function name	pkcau_hash_struct_para_init
Function prototype	void pkcau_hash_struct_para_init(pkcau_hash_parameter_struct* init_para);
Function descriptions	initialize hash parameter structure with a default value
Precondition	-
The called functions	-
Input parameter{in}	
-	-
Output parameter{out}	
init_para	hash parameter struct, the structure members can refer to Table 3-751. Structure pkcau_hash_parameter_struct .
Return value	
-	-

Example:

```
/* initialize PKCAU hash parameter struct with a default value */
```

```
pkcau_hash_parameter_struct pkcau_hash_parameter;
```

```
pkcau_hash_struct_para_init(&pkcau_hash_parameter);
```

pkcau_ecc_out_struct_para_init

The description of pkcau_ecc_out_struct_para_init is shown as below:

Table 3-763. Function pkcau_ecc_out_struct_para_init

Function name	pkcau_ecc_out_struct_para_init
Function prototype	void pkcau_ecc_out_struct_para_init(pkcau_ecc_out_struct* init_para)
Function descriptions	initialize ECC out parameter structure with a default value
Precondition	-
The called functions	-
Input parameter{in}	
-	-
Output parameter{out}	
init_para	ecdsa signature, ecc scalar multiplication output structure, the structure members can refer to Table 3-752. Structure pkcau_ecc_out_struct .
Return value	
-	-

Example:

```
/* initialize PKCAU modular reduction parameter struct with a default value */
```

```
pkcau_ecc_out_struct pkcau_ecc_out_parameter;
```

```
pkcau_ecc_out_struct_para_init(&pkcau_ecc_out_parameter);
```

pkcau_enable

The description of pkcau_enable is shown as below:

Table 3-764. Function pkcau_enable

Function name	pkcau_enable
Function prototype	void pkcau_enable(void);
Function descriptions	enable PKCAU
Precondition	-
The called functions	-
Input parameter{in}	
-	-
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* enable PKCAU */
pkcau_enable();
```

pkcau_disable

The description of pkcau_disable is shown as below:

Table 3-765. Function pkcau_disable

Function name	pkcau_disable
Function prototype	void pkcau_disable(void);
Function descriptions	disable PKCAU
Precondition	-
The called functions	-
Input parameter{in}	
-	-
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* disable PKCAU */
pkcau_disable();
```

pkcau_start

The description of pkcau_start is shown as below:

Table 3-766. Function pkcau_start

Function name	pkcau_start
Function prototype	void pkcau_start(void);
Function descriptions	start operation
Precondition	-
The called functions	-
Input parameter{in}	
-	-
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* start operation */
pkcau_start();
```

pkcau_mode_set

The description of pkcau_mode_set is shown as below:

Table 3-767. Function pkcau_mode_set

Function name	pkcau_mode_set
Function prototype	void pkcau_mode_set(uint32_t mode);
Function descriptions	configure the PKCAU operation mode
Precondition	-
The called functions	-
Input parameter{in}	
mode	PKCAU operation mode
<i>PKCAU_MODE_MOD_EXP</i>	Montgomery parameter computation then modular exponentiation
<i>PKCAU_MODE_MONT_PARAM</i>	Montgomery parameter computation only
<i>PKCAU_MODE_MOD_EXP_FAST</i>	modular exponentiation only
<i>PKCAU_MODE_CRT_EXP</i>	RSA CRT exponentiation
<i>PKCAU_MODE_MOD_INVERSION</i>	modular inversion
<i>PKCAU_MODE_ARITHMETIC_ADD</i>	arithmetic addition
<i>PKCAU_MODE_ARITHMETIC_SUB</i>	arithmetic subtraction
<i>PKCAU_MODE_ARITHMETIC_MULT</i>	arithmetic multiplication

<i>METIC_MUL</i>	
<i>PKCAU_MODE_ARITHMETIC_COMP</i>	arithmetic comparison
<i>PKCAU_MODE_MODULAR_REDUCTION</i>	modular reduction
<i>PKCAU_MODE_MODULAR_ADD</i>	modular addition
<i>PKCAU_MODE_MODULAR_SUB</i>	modular subtraction
<i>PKCAU_MODE_MONTGOMERY_MUL</i>	Montgomery multiplication
<i>PKCAU_MODE_ECC_PARAM_COMPUTATION</i>	Montgomery parameter computation then ECC scalar multiplication
<i>PKCAU_MODE_ECC_SCALAR_MUL_FAST</i>	ECC scalar multiplication only
<i>PKCAU_MODE_ECDSA_SIGN</i>	ECDSA sign
<i>PKCAU_MODE_ECDSA_VERIFY</i>	ECDSA verification
<i>PKCAU_MODE_POINT_CHECK</i>	point on elliptic curve Fp check
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* configure the PKCAU operation mode as PKCAU_MODE_ARITHMETIC_ADD */
pkcau_mode_set(PKCAU_MODE_ARITHMETIC_ADD);
```

pkcau_mont_param_operation

The description of pkcau_mont_param_operation is shown as below:

Table 3-768. Function pkcau_mont_param_operation

Function name	pkcau_mont_param_operation
Function prototype	void pkcau_mont_param_operation(pkcau_mont_parameter_struct* mont_para, uint8_t* results)
Function descriptions	execute montgomery parameter operation
Precondition	-
The called functions	-
Input parameter{in}	
mont_para	montgomery parameter struct, the structure members can refer to Table 3-743. Structure pkcau_mont_parameter_struct .

Output parameter{out}	
results	output buffer
Return value	
-	-

Example:

```
uint8_t results[ME_MOD_SIZE];

/* initialize the montgomery parameter structure */
pkcau_mod_parameter_struct pkcau_mod_parameter;
pkcau_mod_struct_para_init(&pkcau_mod_parameter);

/* initialize the montgomery parameters */
pkcau_mod_parameter.modulus_n_len = ME_MOD_SIZE;
pkcau_mod_parameter.modulus_n = (uint8_t *)modulus_n;

/* execute montgomery parameter operation */
pkcau_mod_param_operation(&pkcau_mod_parameter, results);
```

pkcau_mod_operation

The description of pkcau_mod_operation is shown as below:

Table 3-769. Function pkcau_mod_operation

Function name	pkcau_mod_operation
Function prototype	pkcau_mod_operation(pkcau_mod_parameter_struct* mod_para, uint32_t mode, uint8_t* results)
Function descriptions	execute modular operation, include modular addition, modular subtraction and montgomery multiplication
Precondition	-
The called functions	-
Input parameter{in}	
mod_para	modular parameter struct, the structure members can refer to Table 3-744. Structure pkcau_mod_parameter_struct .
Input parameter{in}	
mode	modular operation mode
PKCAU_MODE_MOD_ADD	modular addition
PKCAU_MODE_MOD_SUB	modular subtraction
PKCAU_MODE_MONT_MUL	Montgomery multiplication
Output parameter{out}	

results	output buffer
Return value	
-	-

Example:

```
uint8_t results[MA_MOD_SIZE];

/* initialize the modular parameter structure */

pkcau_mod_parameter_struct pkcau_mod_parameter;

pkcau_mod_struct_para_init(&pkcau_mod_parameter);

/* initialize the modular parameters */

pkcau_mod_parameter.oprd_a = (uint8_t *)oprd_a;

pkcau_mod_parameter.oprd_b = (uint8_t *)oprd_b;

pkcau_mod_parameter.modulus_n_len = MA_MOD_SIZE;

pkcau_mod_parameter.modulus_n = (uint8_t *)modulus_n;

/* execute modular addition operation */

pkcau_mod_operation(&pkcau_mod_parameter, PKCAU_MODE_MOD_ADD, results);
```

pkcau_mod_exp_operation

The description of pkcau_mod_exp_operation is shown as below:

Table 3-770. Function pkcau_mod_exp_operation

Function name	pkcau_mod_exp_operation
Function prototype	pkcau_mod_exp_operation(pkcau_mod_exp_parameter_struct* mod_exp_para, uint32_t mode, uint8_t* results)
Function descriptions	execute modular exponentiation operation
Precondition	-
The called functions	-
Input parameter{in}	
mod_exp_para	modular exponentiation parameter struct, the structure members can refer to Table 3-745. Structure pkcau_mod_exp_parameter_struct.
Input parameter{in}	
mode	modular exponentiation operation mode
<i>PKCAU_MODE_MOD_EXP</i>	montgomery parameter computation then modular exponentiation
<i>PKCAU_MODE_MOD_EXP_FAST</i>	modular exponentiation only
Output parameter{out}	
results	output buffer
Return value	

-	-
---	---

Example:

```
uint8_t mod_exp_res[ME_MOD_SIZE];

/* initialize the modular exponentation parameter structure */
pkcau_mod_exp_parameter_struct pkcau_mod_exp_parameter;
pkcau_mod_exp_struct_para_init(&pkcau_mod_exp_parameter);

/* initialize the modular exponentation parameters */

pkcau_mod_exp_parameter.oprd_a = (uint8_t *)oprd_a;
pkcau_mod_exp_parameter.oprd_a_len = sizeof(oprd_a);
pkcau_mod_exp_parameter.exp_e = (uint8_t *)exp_e;
pkcau_mod_exp_parameter.e_len = ME_E_SIZE;
pkcau_mod_exp_parameter.modulus_n_len = ME_MOD_SIZE;
pkcau_mod_exp_parameter.modulus_n = (uint8_t *)modulus_n;
pkcau_mod_exp_parameter.mont_para = (uint8_t *)mont_para;
pkcau_mod_exp_parameter.mont_para_len = sizeof(mont_para);

/* execute modular exponentation fast operation */

pkcau_mod_exp_operation(&pkcau_mod_exp_parameter,
PKCAU_MODE_MOD_EXP_FAST, mod_exp_res);
```

pkcau_mod_inver_operation

The description of pkcau_mod_inver_operation is shown as below:

Table 3-771. Function pkcau_mod_inver_operation

Function name	pkcau_mod_inver_operation
Function prototype	pkcau_mod_inver_operation(pkcau_mod_parameter_struct* mod_inver_para, uint8_t* results)
Function descriptions	execute modular inversion operation
Precondition	-
The called functions	-
Input parameter{in}	
mod_inver_para	modular inversion parameter struct, the structure members can refer to Table 3-744. Structure pkcau_mod_parameter_struct .
Output parameter{out}	
results	output buffer
Return value	
-	-

Example:

```
uint8_t mod_inver_res[ME_MOD_SIZE];

/* initialize the modular inversion parameter structure */

pkcau_mod_parameter_struct pkcau_mod_inver_parameter;

pkcau_mod_inver_struct_para_init(&pkcau_mod_inver_parameter);

/* initialize the modular parameters */

pkcau_mod_inver_parameter.oprd_a = (uint8_t *)oprd_a;

pkcau_mod_inver_parameter.oprd_a = sizeof(oprd_a);

pkcau_mod_inver_parameter.modulus_n_len = ME_MOD_SIZE;

pkcau_mod_inver_parameter.modulus_n = (uint8_t *)modulus_n;

/* execute modular inversion operation */

pkcau_mod_inver_operation(&pkcau_mod_inver_parameter, mod_inver_res);
```

pkcau_mod_reduc_operation

The description of pkcau_mod_reduc_operation is shown as below:

Table 3-772. Function pkcau_mod_reduc_operation

Function name	pkcau_mod_reduc_operation
Function prototype	void pkcau_mod_reduc_operation(pkcau_mod_parameter_struct* mod_reduc_para, uint8_t* results)
Function descriptions	execute modular reduction operation
Precondition	-
The called functions	-
Input parameter{in}	
mod_reduc_para	modular reduction parameter struct, the structure members can refer to Table 3-744. Structure pkcau_mod_parameter_struct .
Output parameter{out}	
results	output buffer
Return value	
-	-

Example:

```
uint8_t mod_reduc_res[MA_MOD_SIZE];

/* initialize the modular inversion parameter structure */

pkcau_mod_parameter_struct pkcau_mod_reduc_parameter;

pkcau_mod_reduc_struct_para_init(&pkcau_mod_reduc_parameter);
```

```

/* initialize the modular parameters */

pkcau_mod_reduc_parameter.oprd_a = (uint8_t *)oprd_a;

pkcau_mod_reduc_parameter.oprd_a_len = MA_A_SIZE;

pkcau_mod_reduc_parameter.modulus_n_len = MA_MOD_SIZE;

pkcau_mod_reduc_parameter.modulus_n = (uint8_t *)modulus_n;

/* execute modular reduction operation */

pkcau_mod_reduc_operation(&pkcau_mod_reduc_parameter, mod_reduc_res);

```

pkcau_arithmetic_operation

The description of pkcau_arithmetic_operation is shown as below:

Table 3-773. Function pkcau_arithmetic_operation

Function name	pkcau_arithmetic_operation
Function prototype	void pkcau_arithmetic_operation(pkcau_arithmetic_parameter_struct* arithmetic_para, uint32_t mode, uint8_t* results)
Function descriptions	execute arithmetic operation
Precondition	-
The called functions	-
Input parameter{in}	
arithmetic_para	arithmetic parameter struct, the structure members can refer to Table 3-746. Structure pkcau_arithmetic_parameter_struct .
Input parameter{in}	
mode	arithmetic operation mode
PKCAU_MODE_ARITHMETIC_ADD	arithmetic addition
PKCAU_MODE_ARITHMETIC_SUB	arithmetic subtraction
PKCAU_MODE_ARITHMETIC_MUL	arithmetic multiplication
PKCAU_MODE_ARITHMETIC_COMP	arithmetic comparison
Output parameter{out}	
results	output buffer
Return value	
-	-

Example:

```

uint8_t ari_multi_res[AM_SIZE];

/* initialize the arithmetic parameter structure */

pkcau_arithmetic_parameter_struct pkcau_arithmetic_parameter;

```

```

pkcau_arithmetic_struct_para_init(&pkcau_arithmetic_parameter);

/* initialize the arithmetic addition parameters */

pkcau_ari_multi_parameter.oprd_a = (uint8_t *)oprd_a;
pkcau_ari_multi_parameter.oprd_a_len = AM_A_SIZE;
pkcau_ari_multi_parameter.oprd_b = (uint8_t *)oprd_b;
pkcau_ari_multi_parameter.oprd_b_len = AM_B_SIZE;

/* execute arithmetic addition operation */

pkcau_arithmetic_operation(&pkcau_arithmetic_parameter,
PKCAU_MODE_ARITHMETIC_ADD, ari_multi_res);

```

pkcau_crt_exp_operation

The description of pkcau_crt_exp_operation is shown as below:

Table 3-774. Function pkcau_crt_exp_operation

Function name	pkcau_crt_exp_operation
Function prototype	void pkcau_crt_exp_operation(pkcau_crt_parameter_struct* crt_para, uint8_t* results);
Function descriptions	execute RSA CRT exponentiation operation
Precondition	-
The called functions	-
Input parameter{in}	
crt_para	CRT parameter struct, the structure members can refer to Table 3-747. Structure pkcau crt parameter struct.
Output parameter{out}	
results	output buffer
Return value	
-	-

Example:

```

uint8_t crt_res[sizeof(rsa_crt_a)];

/* initialize the arithmetic parameter structure */

pkcau_crt_parameter_struct pkcau_crt_parameter;
pkcau_crt_exp_operation(&pkcau_crt_parameter);

/* initialize the input ECC curve parameters */

pkcau_crt_parameter.oprd_a = (uint8_t *)rsa_crt_a;
pkcau_crt_parameter.oprd_a_len = sizeof(rsa_crt_a);
pkcau_crt_parameter.oprd_dp = (uint8_t *)rsa_crt_dp;

```



```

pkcau crt_parameter.oprd_dp_len = sizeof(rsa crt_dp);

pkcau crt_parameter.oprd_dq = (uint8_t *)rsa crt_dq;

pkcau crt_parameter.oprd_dq_len = sizeof(rsa crt_dq);

pkcau crt_parameter.oprd_qinv = (uint8_t *)rsa crt_qinv;

pkcau crt_parameter.oprd_qinv_len = sizeof(rsa crt_qinv);

pkcau crt_parameter.oprd_p = (uint8_t *)rsa crt_p;

pkcau crt_parameter.oprd_p_len = sizeof(rsa crt_p);

pkcau crt_parameter.oprd_q = (uint8_t *)rsa crt_q;

pkcau crt_parameter.oprd_q_len = sizeof(rsa crt_q);

/* execute RSA CRT exponentiation operation */

pkcau crt_exp_operation(&pkcau crt_parameter, crt_res);

```

pkcau_point_check_operation

The description of pkcau_point_check_operation is shown as below:

Table 3-775. Function pkcau_point_check_operation

Function name	pkcau_point_check_operation
Function prototype	uint8_t pkcau_point_check_operation(pkcau_point_parameter_struct* point_para, const pkcau_ec_group_parameter_struct* curve_group_para);
Function descriptions	execute point check operation
Precondition	-
The called functions	-
Input parameter{in}	
point_para	point parameter struct, the structure members can refer to Table 3-749. Structure pkcau_point_parameter_struct .
Input parameter{in}	
curve_group_para	ECC curve parameter struct, the structure members can refer to Table 3-748. Structure pkcau_ec_group_parameter_struct .
Output parameter{out}	
-	-
Return value	
uint8_t	flag indicating whether the point is on an elliptic curve or not

Example:

```

uint8_t res = 1;

/* initialize point parameter and ECC curve parameter structure */

pkcau_point_parameter_struct pkcau_point_parameter;

```

```

pkcau_ec_group_parameter_struct pkcau_curve_group;

pkcau_point_struct_para_init(&pkcau_point_parameter);

pkcau_ec_group_struct_para_init(&pkcau_curve_group);

/* initialize the input point parameter */

pkcau_point_check_parameter.point_x = pcheck_x;

pkcau_point_check_parameter.point_x_len = sizeof(pcheck_x);

pkcau_point_check_parameter.point_y = pcheck_y;

pkcau_point_check_parameter.point_y_len = sizeof(pcheck_y);

/* initialize the input ECC curve parameter */

pkcau_curve_group.modulus_p = (uint8_t *)brainpoolp256r1_p;

pkcau_curve_group.modulus_p_len = sizeof(brainpoolp256r1_p);

pkcau_curve_group.coff_a = (uint8_t *)brainpoolp256r1_a;

pkcau_curve_group.coff_a_len = sizeof(brainpoolp256r1_a);

pkcau_curve_group.coff_b = (uint8_t *)brainpoolp256r1_b;

pkcau_curve_group.coff_b_len = sizeof(brainpoolp256r1_b);

pkcau_curve_group.a_sign = 0;

/* execute point check operation */

res = pkcau_point_check_operation(&pkcau_point_parameter, &pkcau_curve_group);

```

pkcau_point_mul_operation

The description of pkcau_point_mul_operation is shown as below:

Table 3-776. Function pkcau_point_mul_operation

Function name	pkcau_point_mul_operation
Function prototype	void pkcau_point_mul_operation(pkcau_point_parameter_struct* point_para, const pkcau_ec_group_parameter_struct* curve_group_para, \n uint32_t mode, pkcau_ecc_out_struct* result);
Function descriptions	execute point multiplication operation
Precondition	-
The called functions	-
Input parameter{in}	
point_para	point parameter struct, the structure members can refer to Table 3-749. Structure pkcau_point_parameter_struct .
Input parameter{in}	
curve_group_para	ECC curve parameter struct, the structure members can refer to Table

	3-748. Structure pkcau_ec_group_parameter_struct.
Input parameter{in}	
mode	point multiplication operation mode
<i>PKCAU_MODE_ECC_MUL</i>	montgomery parameter computation then ECC scalar multiplication
<i>PKCAU_MODE_ECC_MUL_FAST</i>	ECC scalar multiplication only
Output parameter{out}	
result	ecdsa signature, ecc scalar multiplication output structure, the structure members can refer to Table 3-752. Structure pkcau_ecc_out_struct.
Return value	
-	-

Example:

```
pkcau_ecc_out_struct pkcau_ecc_out_result;

/* initialize the ECC out parameter */

pkcau_ecc_out_struct_para_init(&pkcau_ecc_out_result);

pkcau_ecc_out_result.point_x = res_x;

pkcau_ecc_out_result.point_y = res_y;

/* initialize point parameter and ECC curve parameter structure */

pkcau_point_parameter_struct pkcau_point_parameter;

pkcau_ec_group_parameter_struct pkcau_curve_group;

pkcau_point_struct_para_init(&pkcau_point_parameter);

pkcau_ec_group_struct_para_init(&pkcau_curve_group);

/* initialize the input point parameter */

pkcau_point_parameter.point_x = ec_pmul_x;

pkcau_point_parameter.point_x_len = sizeof(ec_pmul_x);

pkcau_point_parameter.point_y = ec_pmul_y;

pkcau_point_parameter.point_x_len = sizeof(ec_pmul_x);

/* initialize the input ECC curve parameter */

pkcau_curve_group.modulus_p      = (uint8_t *)brainpoolp256r1_p;

pkcau_curve_group.modulus_p_len = sizeof(brainpoolp256r1_p);

pkcau_curve_group.coff_a = (uint8_t *)brainpoolp256r1_a;

pkcau_curve_group.coff_a_len = sizeof(brainpoolp256r1_a);
```

```

pkcau_curve_group.a_sign = 0;

pkcau_curve_group.multi_k = ec_pmul_k;

pkcau_curve_group.multi_k_len = PMUL_K_SIZE;

/* execute scalar multiplication operation */

pkcau_point_mul_operation(&pkcau_point_parameter, &pkcau_curve_group,
PKCAU_MODE_ECC_MUL, pkcau_ecc_out_result);

```

pkcau_ecdsa_sign_operation

The description of pkcau_ecdsa_sign_operation is shown as below:

Table 3-777. Function pkcau_ecdsa_sign_operation

Function name	pkcau_ecdsa_sign_operation
Function prototype	uint8_t pkcau_ecdsa_sign_operation(pkcau_hash_parameter_struct* hash_para, \ const pkcau_ec_group_parameter_struct* curve_group_para, \ pkcau_ecc_out_struct* result);
Function descriptions	execute ECDSA sign operation
Precondition	-
The called functions	-
Input parameter{in}	
hash_para	hash parameter struct, the structure members can refer to Table 3-751. Structure pkcau_hash_parameter_struct .
Input parameter{in}	
curve_group_para	ECC curve parameter struct, the structure members can refer to Table 3-748. Structure pkcau_ec_group_parameter_struct .
Output parameter{out}	
result	ecdsa signature, ecc scalar multiplication output structure, the structure members can refer to Table 3-752. Structure pkcau_ecc_out_struct .
Return value	
uint8_t	flag indicating whether the signature operation was successful

Example:

```

pkcau_ecc_out_struct pkcau_ecc_out_result;

/* initialize the ECC out parameter */

pkcau_ecc_out_struct_para_init(&pkcau_ecc_out_result);

pkcau_ecc_out_result.sign_r = r;

pkcau_ecc_out_result.sign_s = s;

/* initialize hash parameter and ECC curve parameter structure */

pkcau_hash_parameter_struct pkcau_hash_parameter;

```

```

pkcau_ec_group_parameter_struct pkcau_curve_group;

pkcau_hash_struct_para_init(&pkcau_hash_parameter);

pkcau_ec_group_struct_para_init(&pkcau_curve_group);

/* initialize the input hash parameter */

pkcau_hash_parameter.hash_z = hash;

pkcau_hash_parameter.hash_z_len = DATA_SIZE;

/* initialize the input ECC curve parameter */

pkcau_curve_group.modulus_p = secp112r2_p;

pkcau_curve_group.modulus_p_len = sizeof(secp112r2_p);

pkcau_curve_group.coff_a = secp112r2_a;

pkcau_curve_group.coff_a_len = sizeof(secp112r2_a);

pkcau_curve_group.a_sign = 0;

pkcau_curve_group.base_point_x = secp112r2_gx;

pkcau_curve_group.base_point_x_len = sizeof(secp112r2_gx);

pkcau_curve_group.base_point_y = secp112r2_gy;

pkcau_curve_group.base_point_y_len = sizeof(secp112r2_gy);

pkcau_curve_group.order_n = secp112r2_n,

pkcau_curve_group.order_n_len = sizeof(secp112r2_n);

pkcau_curve_group.integer_k = k;

pkcau_curve_group.integer_k_len = DATA_SIZE;

pkcau_curve_group.private_key_d = d;

pkcau_curve_group.private_key_d_len = DATA_SIZE;

/* execute ECDSA sign operation */

pkcau_ecdsa_sign_operation(&pkcau_hash_parameter, &pkcau_curve_group, results);

```

pkcau_ecdsa_verification_operation

The description of pkcau_ecdsa_verification_operation is shown as below:

Table 3-778. Function pkcau_ecdsa_verification_operation

Function name	pkcau_ecdsa_verification_operation
Function prototype	uint8_t pkcau_ecdsa_verification_operation(pkcau_point_parameter_struct* point_para, pkcau_hash_parameter_struct* hash_para, pkcau_signature_parameter_struct* signature_para, const

	pkcau_ec_group_parameter_struct* curve_group_para)
Function descriptions	execute ECDSA verification operation
Precondition	-
The called functions	-
Input parameter{in}	
point_para	point parameter struct, the structure members can refer to Table 3-749. Structure pkcau_point_parameter_struct.
Input parameter{in}	
hash_para	hash parameter struct, the structure members can refer to Table 3-751. Structure pkcau_hash_parameter_struct.
Input parameter{in}	
signature_para	signature parameter struct, the structure members can refer to Table 3-750. Structure pkcau_signature_parameter_struct.
Input parameter{in}	
curve_group_para	ECC curve parameter struct, the structure members can refer to Table 3-748. Structure pkcau_ec_group_parameter_struct.
Output parameter{out}	
-	-
Return value	
uint8_t	flag indicating whether the signature verification operation was successful

Example:

```
uint8_t verify_res = 1;

/* ECC curve parameter structure */
pkcau_ec_group_parameter_struct pkcau_curve_group;

/* hash parameter structure */
pkcau_hash_parameter_struct pkcau_hash_parameter;

/* signature parameter structure */
pkcau_signature_parameter_struct pkcau_signature_parameter;

/* point parameter structure */
pkcau_point_parameter_struct pkcau_point_parameter;

/* initialize the ECC curve parameter, hash parameter, point parameter and signature
parameter structure */
pkcau_ec_group_struct_para_init(&pkcau_curve_group);
pkcau_hash_struct_para_init(&pkcau_hash_parameter);
pkcau_point_struct_para_init(&pkcau_point_parameter);
pkcau_signature_struct_para_init(&pkcau_signature_parameter);
```

```

/* initialize the input ECC signature parameters */

pkcau_signature_parameter.sign_r = (uint8_t *)ecc_verify_r;
pkcau_signature_parameter.sign_r_len = sizeof(ecc_verify_r);
pkcau_signature_parameter.sign_s = (uint8_t *)ecc_verify_s;
pkcau_signature_parameter.sign_s_len = sizeof(ecc_verify_s);

/* initialize the input point parameters */

pkcau_point_parameter.point_x = ecc_verify_x;
pkcau_point_parameter.point_x_len = sizeof(ecc_verify_x);
pkcau_point_parameter.point_y = ecc_verify_y;
pkcau_point_parameter.point_y_len = sizeof(ecc_verify_y);

/* initialize the input ECC curve parameters */

pkcau_curve_group.modulus_p = (uint8_t *)brainpoolp256r1_p;
pkcau_curve_group.modulus_p_len = sizeof(brainpoolp256r1_p);
pkcau_curve_group.coff_a = (uint8_t *)brainpoolp256r1_a;
pkcau_curve_group.coff_a_len = sizeof(brainpoolp256r1_a);
pkcau_curve_group.a_sign = 0;
pkcau_curve_group.base_point_x = (uint8_t *)brainpoolp256r1_gx;
pkcau_curve_group.base_point_x_len = sizeof(brainpoolp256r1_gx);
pkcau_curve_group.base_point_y = (uint8_t *)brainpoolp256r1_gy;
pkcau_curve_group.base_point_y_len = sizeof(brainpoolp256r1_gy);
pkcau_curve_group.order_n = (uint8_t *)brainpoolp256r1_n,
pkcau_curve_group.order_n_len = sizeof(brainpoolp256r1_n);

/* initialize the input hash parameters */

pkcau_hash_parameter.hash_z = (uint8_t *)ecc_verify_hash;
pkcau_hash_parameter.hash_z_len = sizeof(ecc_verify_hash);

/* execute ECDSA verification operation */

verify_res = pkcau_ecdsa_verification_operation(&pkcau_point_parameter,
&pkcau_hash_parameter, &pkcau_signature_parameter, &pkcau_curve_group);

```

pkcau_flag_get

The description of pkcau_flag_get is shown as below:

Table 3-779. Function pkcau_flag_get

Function name	pkcau_flag_get
Function prototype	FlagStatus pkcau_flag_get(uint32_t flag);
Function descriptions	get PKCAU flag status
Precondition	-
The called functions	-
Input parameter{in}	
flag	PKCAU flags
PKCAU_FLAG_ADDRE RR	address error flag
PKCAU_FLAG_RAME RR	PKCAU RAM error flag
PKCAU_FLAG_END	end of PKCAU operation flag
PKCAU_FLAG_BUSY	busy flag
Output parameter{out}	
-	-
Return value	
FlagStatus	SET / RESET

Example:

```
/* get PKCAU flag status */
```

```
FlagStatus flag_state = RESET;
```

```
flag_state = pkcau_flag_get(PKCAU_FLAG_ADDRERR);
```

pkcau_flag_clear

The description of pkcau_flag_clear is shown as below:

Table 3-780. Function pkcau_flag_clear

Function name	pkcau_flag_clear
Function prototype	void pkcau_flag_clear(uint32_t flag);
Function descriptions	clear PKCAU flag status
Precondition	-
The called functions	-
Input parameter{in}	
flag	PKCAU flags
PKCAU_FLAG_ADDRE RR	address error flag
PKCAU_FLAG_RAME RR	PKCAU RAM error flag
PKCAU_FLAG_END	end of PKCAU operation flag
Output parameter{out}	
-	-

Return value	
-	-

Example:

```
/* clear address error flag*/
pkcau_flag_clear(PKCAU_FLAG_ADDRERR);
```

pkcau_interrupt_enable

The description of pkcau_interrupt_enable is shown as below:

Table 3-781. Function pkcau_interrupt_enable

Function name	pkcau_interrupt_enable
Function prototype	void pkcau_interrupt_enable(uint32_t interrupt);
Function descriptions	enable PKCAU interrupt
Precondition	-
The called functions	-
Input parameter{in}	
interrupt	interrupt type
<i>PKCAU_INT_ADDRERR</i>	address error interrupt
<i>PKCAU_INT_RAMERR</i>	PKCAU RAM error interrupt
<i>PKCAU_INT_END</i>	end of PKCAU operation interrupt
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* enable PKCAU address error interrupt */
pkcau_interrupt_enable(PKCAU_INT_ADDRERR);
```

pkcau_interrupt_disable

The description of pkcau_interrupt_disable is shown as below:

Table 3-782. Function pkcau_interrupt_disable

Function name	pkcau_interrupt_disable
Function prototype	void pkcau_interrupt_disable(uint32_t interrupt);
Function descriptions	disable PKCAU interrupt
Precondition	-
The called functions	-
Input parameter{in}	
interrupt	interrupt type

<i>PKCAU_INT_ADDRERR</i> <i>R</i>	address error interrupt
<i>PKCAU_INT_RAMERR</i>	PKCAU RAM error interrupt
<i>PKCAU_INT_END</i>	end of PKCAU operation interrupt
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* disable PKCAU address error interrupt */
pkcau_interrupt_disable(PKCAU_INT_ADDRERR);
```

pkcau_interrupt_flag_get

The description of pkcau_interrupt_flag_get is shown as below:

Table 3-783. Function pkcau_interrupt_flag_get

Function name	pkcau_interrupt_flag_get
Function prototype	FlagStatus pkcau_interrupt_flag_get(uint32_t int_flag);
Function descriptions	get PKCAU interrupt flag status
Precondition	-
The called functions	-
Input parameter{in}	
int_flag	PKCAU interrupt flags
<i>PKCAU_INT_FLAG_ADDRERR</i>	address error interrupt flag
<i>PKCAU_INT_FLAG_RAMERR</i>	PKCAU RAM error interrupt flag
<i>PKCAU_INT_FLAG_END</i>	end of PKCAU operation interrupt flag
Output parameter{out}	
-	-
Return value	
FlagStatus	SET / RESET

Example:

```
/* get PKCAU interrupt flag status */
FlagStatus flag_state = RESET;
flag_state = pkcau_interrupt_flag_get(PKCAU_INT_FLAG_ADDRERR);
```

pkcau_interrupt_flag_clear

The description of pkcau_interrupt_flag_clear is shown as below:

Table 3-784. Function pkcau_interrupt_flag_clear

Function name	pkcau_interrupt_flag_clear
Function prototype	void pkcau_interrupt_flag_clear(uint32_t int_flag);
Function descriptions	clear PKCAU flag interrupt status
Precondition	-
The called functions	-
Input parameter{in}	
int_flag	PKCAU interrupt flags
<i>PKCAU_INT_FLAG_ADDRERR</i>	address error interrupt flag
<i>PKCAU_INT_FLAG_RAMERR</i>	PKCAU RAM error interrupt flag
<i>PKCAU_INT_FLAG_END</i>	end of PKCAU operation interrupt flag
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* clear address error interrupt flag*/
pkcau_interrupt_flag_clear(PKCAU_INT_FLAG_ADDRERR);
```

3.23. PMU

According to the Power management unit (PMU), provides three types of power saving modes, including Sleep, Deep-sleep and Standby mode. The PMU registers are listed in chapter [3.23.1](#), the PMU firmware functions are introduced in chapter [3.23.2](#).

3.23.1. Descriptions of Peripheral registers

PMU registers are listed in the table shown as below:

Table 3-785. PMU Registers

Registers	Descriptions
PMU_CTL	Control register
PMU_CS	Control and status register

3.23.2. Descriptions of Peripheral functions

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

Table 3-786. PMU firmware function

Function name	Function description
pmu_deinit	deinitialize the PMU
pmu_lvd_select	select low voltage detector threshold
pmu_lvd_disable	disable PMU lvd
pmu_ldo_output_select	select LDO output voltage
pmu_highdriver_mode_enable	high-driver mode enable
pmu_highdriver_mode_disable	high-driver mode disable
pmu_highdriver_switch_select	high-driver mode switch
pmu_low_driver_mode_enable	enable low-driver mode in deep-sleep mode
pmu_lowdriver_mode_disable	disable low-driver mode in deep-sleep
pmu_lowpower_driver_config	in deep-sleep mode, driver mode when use low power LDO
pmu_normalpower_driver_config	in deep-sleep mode, driver mode when use normal power LDO
pmu_to_sleepmode	PMU work at sleep mode
pmu_to_deepsleepmode	PMU work at deepsleep mode
pmu_to_standbymode	pmu work at standby mode
pmu_wakeup_pin_enable	enable wakeup pin
pmu_wakeup_pin_disable	disable PMU wakeup pin
pmu_backup_ldo_config	backup SRAM LDO on
pmu_backup_write_enable	backup domain write enable
pmu_backup_write_disable	disable write access to the registers in backup domain
pmu_flag_get	get flag status
pmu_flag_clear	clear flag bit

pmu_deinit

The description of pmu_deinit is shown as below:

Table 3-787. Function pmu_deinit

Function name	pmu_deinit
Function prototype	void pmu_deinit(void);
Function descriptions	deinitialize the PMU
Precondition	-
The called functions	rcu_periph_reset_enable / rcu_periph_reset_disable
Input parameter{in}	
-	-
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* reset PMU */
pmu_deinit();
```

pmu_lvd_select

The description of pmu_lvd_select is shown as below:

Table 3-788. Function pmu_lvd_select

Function name	pmu_lvd_select
Function prototype	void pmu_lvd_select(uint32_t lvdn);
Function descriptions	select low voltage detector threshold
Precondition	-
The called functions	-
Input parameter{in}	
lvdn	voltage threshold value
PMU_LVDT_0	voltage threshold is 2.1V
PMU_LVDT_1	voltage threshold is 2.3V
PMU_LVDT_2	voltage threshold is 2.4V
PMU_LVDT_3	voltage threshold is 2.6V
PMU_LVDT_4	voltage threshold is 2.7V
PMU_LVDT_5	voltage threshold is 2.9V
PMU_LVDT_6	voltage threshold is 3.0V
PMU_LVDT_7	voltage threshold is 3.1V
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* select low voltage detector threshold as 3.1V */
pmu_lvd_select (PMU_LVDT_7);
```

pmu_lvd_disable

The description of pmu_lvd_disable is shown as below:

Table 3-789. Function pmu_lvd_disable

Function name	pmu_lvd_disable
Function prototype	void pmu_lvd_disable (void);
Function descriptions	disable PMU lvd
Precondition	-
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 3-790. Function pmu_ldo_output_select

Function name	pmu_ldo_output_select
Function prototype	void pmu_ldo_output_select(uint32_t ldo_output);
Function descriptions	select LDO output voltage
Precondition	-
The called functions	-
Input parameter{in}	
ldo_output	output voltage mode
<i>PMU_LDOVS_LOW</i>	low-driver mode enable in deep-sleep mode
<i>PMU_LDOVS_MID</i>	mid-driver mode disable in deep-sleep mode
<i>PMU_LDOVS_HIGH</i>	high-driver mode disable in deep-sleep mode
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* select output low-driver mode enable in deep-sleep mode */
```

```
pmu_ldo_output_select (PMU_LDOVS_LOW);
```

pmu_highdriver_mode_enable

The description of pmu_highdriver_mode_enable is shown as below:

Table 3-791. Function pmu_highdriver_mode_enable

Function name	pmu_highdriver_mode_enable
Function prototype	void pmu_highdriver_mode_enable(void);
Function descriptions	enable high-driver mode
Precondition	-
The called functions	-

Input parameter{in}	
-	-
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* enable high-driver mode */
```

```
pmu_highdriver_mode_enable ( );
```

pmu_highdriver_mode_disable

The description of pmu_highdriver_mode_disable is shown as below:

Table 3-792. Function pmu_highdriver_mode_disable

Function name	pmu_highdriver_mode_disable
Function prototype	void pmu_highdriver_mode_disable(void);
Function descriptions	disable high-driver mode
Precondition	-
The called functions	-
Input parameter{in}	
-	-
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* disable high-driver mode */
```

```
pmu_highdriver_mode_disable ( );
```

pmu_highdriver_switch_select

The description of pmu_highdriver_switch_select is shown as below:

Table 3-793. Function pmu_highdriver_switch_select

Function name	pmu_highdriver_switch_select
Function prototype	void pmu_highdriver_switch_select(uint32_t highdr_switch);
Function descriptions	switch high-driver mode
Precondition	-
The called functions	pmu_flag_get()
Input parameter{in}	
highdr_switch	enable or disable high-driver mode switch

<i>PMU_HIGHDR_SWITC</i> <i>H_NONE</i>	disable high-driver mode switch
<i>PMU_HIGHDR_SWITC</i> <i>H_EN</i>	enable high-driver mode switch
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* enable high-driver mode switch */
```

```
pmu_highdriver_switch_select (PMU_HIGHDR_SWITCH_EN);
```

pmu_lowdriver_mode_enable

The description of pmu_lowdriver_mode_enable is shown as below:

Table 3-794. Function pmu_lowdriver_mode_enable

Function name	pmu_lowdriver_mode_enable
Function prototype	void pmu_lowdriver_mode_enable(void);
Function descriptions	enable low-driver mode in deep-sleep mode
Precondition	-
The called functions	-
Input parameter{in}	
-	-
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* enable low-driver mode in deep-sleep mode */
```

```
pmu_lowdriver_mode_enable ();
```

pmu_lowdriver_mode_disable

The description of pmu_lowdriver_mode_disable is shown as below:

Table 3-795. Function pmu_lowdriver_mode_disable

Function name	pmu_lowdriver_mode_disable
Function prototype	void pmu_lowdriver_mode_disable (void);
Function descriptions	e disable low-driver mode in deep-sleep mode
Precondition	-
The called functions	-

Input parameter{in}	
-	-
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* disable low-driver mode in deep-sleep mode */
```

```
pmu_lowdriver_mode_disable ();
```

pmu_lowpower_driver_config

The description of pmu_lowpower_driver_config is shown as below:

Table 3-796. Function pmu_lowpower_driver_config

Function name	pmu_lowpower_driver_config
Function prototype	void pmu_lowpower_driver_config(uint32_t mode);
Function descriptions	driver mode when use low power LDO
Precondition	-
The called functions	-
Input parameter{in}	
mode	driver mode
<i>PMU_NORMALDR_LO WPWR</i>	normal driver when use low power LDO
<i>PMU_LOWDR_LOWP WR</i>	low-driver mode enabled when LDEN is 11 and use low power LDO
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* normal driver when use low power LDO */
```

```
pmu_lowpower_driver_config (PMU_NORMALDR_LOWPWR);
```

pmu_normalpower_driver_config

The description of pmu_normalpower_driver_config is shown as below:

Table 3-797. Function pmu_normalpower_driver_config

Function name	pmu_normalpower_driver_config
Function prototype	void pmu_normalpower_driver_config (uint32_t mode);
Function descriptions	driver mode when use normal power LDO

Precondition	-
The called functions	-
Input parameter{in}	
mode	driver mode
<i>PMU_NORMALDR_LOWPWR</i>	normal driver when use normal power LDO
<i>PMU_LOWDR_LOWPWR</i>	low-driver mode enabled when LDEN is 11 and use normal power LDO
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* normal driver when use normal power LDO */
```

```
pmu_normalpower_driver_config (PMU_NORMALDR_LOWPWR);
```

pmu_to_sleepmode

The description of pmu_to_sleepmode is shown as below:

Table 3-798. Function pmu_to_sleepmode

Function name	pmu_to_sleepmode
Function prototype	void pmu_to_sleepmode(uint8_t sleepmodecmd);
Function descriptions	PMU work at sleep mode
Precondition	-
The called functions	-
Input parameter{in}	
sleepmodecmd	command to enter sleep mode
<i>WFI_CMD</i>	use WFI command
<i>WFE_CMD</i>	use WFE command
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* use WFI command to enter sleep mode */
```

```
pmu_to_sleepmode (WFI_CMD);
```

pmu_to_deepsleepmode

The description of pmu_to_deepsleepmode is shown as below:

Table 3-799. Function pmu_to_deepsleepmode

Function name	pmu_to_deepsleepmode
Function prototype	void pmu_to_deepsleepmode(uint32_t ldo, uint32_t lowdrive, uint8_t deepsleepmodecmd);
Function descriptions	PMU work at deepsleep mode
Precondition	-
The called functions	-
Input parameter{in}	
ldo	ldo work mode
PMU_LDO_NORMAL	LDO normal work when pmu enter deepsleep mode
PMU_LDO_LOWPOWER	LDO work at low power mode when pmu enter deepsleep mode
Input parameter{in}	
lowdrive	Low driver mode
PMU_LOWDRIVER_DISABLE	Low-driver mode disable in deep-sleep mode
PMU_LOWDRIVER_ENABLE	Low-driver mode enable in deep-sleep mode
Input parameter{in}	
deepsleepmodecmd	command to enter deepsleep mode
WFI_CMD	use WFI command
WFE_CMD	use WFE command
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* use WFI command to enter deepsleep mode and LDO work at normal power */
```

```
pmu_to_deepsleepmode (PMU_LDO_NORMAL, PMU_LOWDRIVER_DISABLE, WFI_CMD);
```

pmu_to_standbymode

The description of pmu_to_standbymode is shown as below:

Table 3-800. Function pmu_to_standbymode

Function name	pmu_to_standbymode
Function prototype	void pmu_to_standbymode(void);
Function descriptions	pmu work at standby mode
Precondition	-
The called functions	-
Input parameter{in}	
-	-

Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* use WFI command to enter standby mode */
```

```
pmu_to_standbymode ();
```

pmu_wakeup_pin_enable

The description of pmu_wakeup_pin_enable is shown as below:

Table 3-801. Function pmu_wakeup_pin_enable

Function name	pmu_wakeup_pin_enable
Function prototype	void pmu_wakeup_pin_enable(void);
Function descriptions	enable wakeup pin
Precondition	-
The called functions	-
Input parameter{in}	
-	-
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* enable wakeup pin */
```

```
pmu_wakeup_pin_enable ();
```

pmu_wakeup_pin_disable

The description of pmu_wakeup_pin_disable is shown as below:

Table 3-802. Function pmu_wakeup_pin_disable

Function name	pmu_wakeup_pin_disable
Function prototype	void pmu_wakeup_pin_disable (void);
Function descriptions	disable wakeup pin
Precondition	-
The called functions	-
Input parameter{in}	
-	-
Output parameter{out}	
-	-

Return value	
-	-

Example:

```
/* disable wakeup pin */
```

```
pmu_wakeup_pin_disable ();
```

pmu_backup_ldo_config

The description of pmu_backup_ldo_config is shown as below:

Table 3-803. Function pmu_backup_ldo_config

Function name	pmu_backup_ldo_config
Function prototype	void pmu_backup_ldo_config(uint32_t bkp_ldo);
Function descriptions	backup SRAM LDO on
Precondition	-
The called functions	-
Input parameter{in}	
mode	driver mode
PMU_BLDON_OFF	backup SRAM LDO closed
PMU_BLDON_ON	open the backup SRAM LDO
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* configuration backup SRAM LDO on */
```

```
pmu_backup_ldo_config (PMU_BLDON_ON);
```

pmu_backup_write_enable

The description of pmu_backup_write_enable is shown as below:

Table 3-804. Function pmu_backup_write_enable

Function name	pmu_backup_write_enable
Function prototype	void pmu_backup_write_enable (void);
Function descriptions	enable backup domain write
Precondition	-
The called functions	-
Input parameter{in}	
-	-
Output parameter{out}	
-	-

Return value	
-	-

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 3-805. Function pmu_backup_write_disable

Function name	pmu_backup_write_disable
Function prototype	void pmu_backup_write_disable (void);
Function descriptions	disable backup domain write
Precondition	-
The called functions	-
Input parameter{in}	
-	-
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* disable backup domain write */
```

```
pmu_backup_write_disable ();
```

pmu_flag_get

The description of pmu_flag_get is shown as below:

Table 3-806. Function pmu_flag_get

Function name	pmu_flag_get
Function prototype	FlagStatus pmu_flag_get(uint32_t flag);
Function descriptions	get flag state
Precondition	-
The called functions	-
Input parameter{in}	
flag	flag
PMU_FLAG_WAKEUP	wakeup flag
PMU_FLAG_STANDBY	standby flag
PMU_FLAG_LVD	lvd flag
PMU_FLAG_BLDORF	backup SRAM LDO ready flag

<i>PMU_FLAG_LDOVSRF</i>	LDO voltage select ready flag
<i>PMU_FLAG_HDRF</i>	high-driver ready flag
<i>PMU_FLAG_HDSRF</i>	high-driver switch ready flag
<i>PMU_FLAG_LDRF</i>	low-driver mode ready flag
Output parameter{out}	
-	-
Return value	
FlagStatus	SET or RESET

Example:

```
/* get wakeup flag state*/
pmu_flag_get (PMU_FLAG_WAKEUP);
```

pmu_flag_clear

The description of pmu_flag_clear is shown as below:

Table 3-807. Function pmu_flag_clear

Function name	pmu_flag_clear
Function prototype	void pmu_flag_clear(uint32_t flag);
Function descriptions	clear flag bit
Precondition	-
The called functions	-
Input parameter{in}	
flag	flag
<i>PMU_FLAG_RESET_WAKEUP</i>	reset wakeup flag
<i>PMU_FLAG_RESET_STANDBY</i>	reset standby flag
Output parameter{out}	
-	-
Return value	
FlagStatus	SET or RESET

Example:

```
/* clear flag bit wakeup flag */
pmu_flag_clear (PMU_FLAG_RESET_WAKEUP);
```

3.24. 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 [3.24.1](#), the RCU

firmware functions are introduced in chapter [3.24.2](#).

3.24.1. Descriptions of Peripheral registers

RCU registers are listed in the table shown as below:

Table 3-808. RCU Registers

Registers	Descriptions
RCU_CTL	Control register
RCU_PLL	PLL register
RCU_CFG0	Clock configuration register 0
RCU_INT	Clock interrupt register
RCU_AHB1RST	AHB1 reset register
RCU_AHB2RST	AHB2 reset register
RCU_AHB3RST	AHB3 reset register
RCU_APB1RST	APB1 reset register
RCU_APB2RST	APB2 reset register
RCU_AHB1EN	AHB1 enable register
RCU_AHB2EN	AHB2 enable register
RCU_AHB3EN	AHB3 enable register
RCU_APB1EN	APB1 enable register
RCU_APB2EN	APB2 enable register
RCU_AHB1SPEN	AHB1 sleep mode enable register
RCU_AHB2SPEN	AHB2 sleep mode enable register
RCU_AHB3SPEN	AHB3 sleep mode enable register
RCU_APB1SPEN	APB1 sleep mode enable register
RCU_APB2SPEN	APB2 sleep mode enable register
RCU_BDCTL	Backup domain control register
RCU_RSTSCK	Reset source/clock register
RCU_PLLSSCTL	PLL clock spread spectrum control register
RCU_PLLI2S	PLLI2S register
RCU_PLLSAI	PLLSAI register
RCU_CFG1	Configuration register 1
RCU_CFG2	Configuration register 2
RCU_ADDCTL	Additional clock control register
RCU_ADDINT	Additional clock interrupt register
RCU_ADDAPB1RST	APB1 additional reset register
RCU_ADDAPB1EN	APB1 additional enable register
RCU_ADDAPB1SPEN	APB1 additional sleep mode enable register
RCU_VKEY	voltage key register
RCU_DSV	Deep-sleep mode voltage register

3.24.2. Descriptions of Peripheral functions

Table 3-809. RCU firmware function

Function name	Function description
rcu_deinit	deinitialize the RCU
rcu_periph_clock_enable	enable the peripherals clock
rcu_periph_clock_disable	disable the peripherals clock
rcu_periph_clock_sleep_enable	enable the peripherals clock when in sleep mode
rcu_periph_clock_sleep_disable	disable the peripherals clock when in sleep mode
rcu_periph_reset_enable	reset the peripherals
rcu_periph_reset_disable	disable reset the peripheral
rcu_bkp_reset_enable	reset the BKP
rcu_bkp_reset_disable	disable the BKP reset
rcu_system_clock_source_config	configure the system clock source
rcu_system_clock_source_get	get the system clock source
rcu_ahb_clock_config	configure the AHB clock prescaler selection
rcu_apb1_clock_config	configure the APB1 clock prescaler selection
rcu_apb2_clock_config	configure the APB2 clock prescaler selection
rcu_ckout0_config	configure the CK_OUT0 clock source
rcu_ckout1_config	configure the CK_OUT1 clock source
rcu_pll_config	configure the main PLL clock
rcu_plli2s_q_config	configure the PLLI2S_Q clock
rcu_plli2s_r_config	configure the PLLI2S_R clock
rcu_pllsai_p_config	configure the PLLSAI_P clock
rcu_pllsai_q_config	configure the PLLSAI_Q clock
rcu_pllsai_r_config	configure the PLLSAI_R clock
rcu_rtc_clock_config	configure the RTC clock source selection
rcu_rtc_div_config	configure the frequency division of RTC clock when HXTAL was selected as its clock source
rcu_i2s_clock_config	configure the I2S clock source selection
rcu_i2c_clock_config	configure the I2C clock selection
rcu_sai_clock_config	configure the SAI clock selection
rcu_ck48m_clock_config	configure the CK48M clock selection
rcu_pll48m_clock_config	configure the PLL48M clock selection
rcu_timer_clock_prescaler_config	configure the TIMER clock prescaler selection
rcu_tli_clock_div_config	configure the TLI clock division selection
rcu_lxtal_drive_capability_config	configure the LXTAL drive capability
rcu_osci_stab_wait	wait for oscillator stabilization flags is SET or oscillator startup is timeout
rcu_osci_on	turn on the oscillator
rcu_osci_off	turn off the oscillator
rcu_osci_bypass_mode_enable	enable the oscillator bypass mode
rcu_osci_bypass_mode_disable	disable the oscillator bypass mode

Function name	Function description
rcu_hxtal_clock_monitor_enable	enable the HXTAL clock monitor
rcu_hxtal_clock_monitor_disable	disable the HXTAL clock monitor
rcu_irc16m_adjust_value_set	set the IRC8M adjust value
rcu_spread_spectrum_config	configure the spread spectrum modulation for the main PLL clock
rcu_spread_spectrum_enable	enable the spread spectrum modulation for the main PLL clock
rcu_spread_spectrum_disable	disable the spread spectrum modulation for the main PLL clock
rcu_deepsleep_voltage_set	set the deep-sleep mode voltage value
rcu_voltage_key_unlock	unlock the voltage key
rcu_clock_freq_get	get the system clock, bus clock frequency
rcu_flag_get	get the clock stabilization and peripheral reset flags
rcu_all_reset_flag_clear	clear all the reset flag
rcu_interrupt_flag_get	get the clock stabilization interrupt and ckm flags
rcu_interrupt_flag_clear	clear the interrupt flags
rcu_interrupt_enable	enable the stabilization interrupt
rcu_interrupt_disable	disable the stabilization interrupt

Enum rcu_periph_enum

Table 3-810. Enum rcu_periph_enum

enum name	Function description
RCU_GPIOx	GPIO ports clock (x=A,B,C,D,E,F,G ,H,I)
RCU_CRC	CRC clock
RCU_BKPSRAM	BKPSRAM clock
RCU_TCMRAM	TCMSRAM clock
RCU_DMAx	DMAx clock (x=0,1)
RCU_IPA	IPA clock
RCU_ENET	ENET clock
RCU_ENETTX	ENETTX clock
RCU_ENETRX	ENETRX clock
RCU_ENETPTP	ENETPTP clock
RCU_USBHS	USBHS clock
RCU_USBHSULPI	USBHSULPI clock
RCU_DCI	DCI clock
RCU_PKCAU	PKCAU clock
RCU_CAU	CAU clock
RCU_HAU	HAU clock
RCU_TRNG	TRNG clock
RCU_USBFS	USBFS clock
RCU_EXMC	EXMC clock

enum name	Function description
RCU_TIMERx	TIMERx clock (x=0,1,2,3,4,5,6,7,8,9,10,11,12,13)
RCU_WWDGT	WWDGT clock
RCU_SPIx	SPIx clock (x=0,1,2,3,4,5)
RCU_SAI	SAI clock
RCU_USARTx	USARTx clock (x=0,1,2,5)
RCU_UARTx	UARTx clock (x=3,4,6,7)
RCU_I2Cx	I2Cx clock (x=0,1,2,3,4,5)
RCU_CANx	CANx clock (x=0,1)
RCU_PMU	PMU clock
RCU_DAC	DAC clock
RCU_RTC	RTC clock
RCU_ADCx	ADCx clock (x=0,1,2)
RCU_SDIO	SDIO clock
RCU_SYSCFG	SYSCFG interface clock
RCU_TLI	TLI clock
RCU_CTC	CTC clock
RCU_IREF	IREF clock

Enum rcu_periph_sleep_enum

Table 3-811. Enum rcu_periph_sleep_enum

enum name	Function description
RCU_GPIOx_SLP	GPIO clock (x=A,B,C,D,E,F,G,H,I)
RCU_CRC_SLP	CRC clock
RCU_FMC_SLP	FMC clock
RCU_SRAM1_SLP	SRAM0 clock
RCU_SRAM1_SLP	SRAM1 clock
RCU_BKPSRAM	BKPSRAM clock
RCU_SRAM2_SLP	SRAM2 clock
RCU_DMAx_SLP	DMAx clock (x=0,1)
RCU_IPA_SLP	IPA clock
RCU_ENET_SLP	ENET clock
RCU_ENETTX_SLP	ENETTX clock
RCU_ENETRX_SLP	ENETRX clock
RCU_ENETPTP_SLP	ENETPTP clock
RCU_USBHS_SLP	USBHS clock
RCU_USBHSULPI_SLP	USBHSULPI clock
RCU_DCI_SLP	DCI clock
RCU_PKCAU_SLP	PKCAU clock
RCU_CAU_SLP	CAU clock
RCU_HAU_SLP	HAU clock

enum name	Function description
RCU_TRNG_SLP	TRNG clock
RCU_USBFS_SLP	USBFS clock
RCU_EXMC_SLP	EXMC clock
RCU_TIMERx_SLP	TIMERx clock (x=0,1,2,3,4,5,6,7,8,9,10,11,12,13)
RCU_WWDGT_SLP	WWDGT clock
RCU_SPIx_SLP	SPIx clock (x=0,1,2,3,4,5)
RCU_SAI_SLP	SAI clock
RCU_USARTx_SLP	USARTx clock (x=0,1,2,5)
RCU_UARTx_SLP	UARTx clock (x=3,4,6,7)
RCU_I2Cx_SLP	I2Cx clock (x=0,1,2,3,4,5)
RCU_CANx_SLP	CANx clock (x=0,1)
RCU_PMU_SLP	PMU clock
RCU_DAC_SLP	DAC clock
RCU_RTC_SLP	RTC clock
RCU_ADCx_SLP	ADCx clock (x=0,1,2)
RCU_SDIO_SLP	SDIO clock
RCU_SYSCFG_SLP	SYSCFG interfacelock
RCU_TLI_SLP	TLI clock
RCU_CTC_SLP	CTC clock
RCU_IREF_SLP	IREF clock

Enum rcu_periph_reset_enum

Table 3-812. Enum rcu_periph_reset_enum

enum name	Function description
RCU_GPIOxRST	reset GPIOx clock (x=A,B,C,D,E,F,G,H,I)
RCU_CRCRST	reset CRC clock
RCU_DMAxRST	reset DMAx clock (x=0,1)
RCU_IPARST	reset IPA clock
RCU_ENETRST	reset ENET clock
RCU_USBHSRST	reset USBHSclock
RCU_DCIRST	reset DCI clock
RCU_PKCAURST	reset PKCAU clock
RCU_CAURST	reset CAU clock
RCU_HAURST	reset HAU clock
RCU_TRNGRST	reset TRNG clock
RCU_USBFSRST	reset USBFS clock
RCU_EXMCRST	reset EXMC clock
RCU_TIMERxRST	reset TIMERx clock (x=0,1,2,3,4,5,6,7,8,9,10,11,12,13)
RCU_WWDGTRST	reset WWDGT clock
RCU_SPIxRST	reset SPIx clock (x=0,1,2,3,4,5)
RCU_SAIRST	reset SAI clock

enum name	Function description
RCU_USARTxRST	reset USARTx clock (x=0,1,2,5)
RCU_UARTxRST	reset UARTx clock (x=3,4,6,7)
RCU_I2CxRST	reset I2Cx clock (x=0,1, 2, 3, 4, 5)
RCU_CANxRST	reset CANx clock (x=0,1)
RCU_PMURST	reset PMU clock
RCU_DACRST	reset DAC clock
RCU_ADCxRST	reset ADCx clock (x=0,1,2)
RCU_SDIORST	reset SDIO clock
RCU_SYSCFGRST	reset SYSCFG clock
RCU_TLIRST	reset TLI clock
RCU_CTCRST	reset CAU clock
RCU_IREFRST	reset IREF clock

Enum rcu_flag_enum

Table 3-813. Enum rcu_flag_enum

enum name	Function description
RCU_FLAG_IRC16MS TB	IRC16M stabilization flag
RCU_FLAG_HXTALST B	HXTAL stabilization flag
RCU_FLAG_PLLI2SST B	PLLI2S stabilization flag
RCU_FLAG_PLLSAIST B	PLLSAI stabilization flag
RCU_FLAG_PLLSTB	PLL stabilization flag
RCU_FLAG_LXTALST B	LXTAL stabilization flag
RCU_FLAG_IRC32KST B	IRC32K stabilization flag
RCU_FLAG_IRC48MS TB	IRC48M stabilization flag
RCU_FLAG_EPRST	external PIN reset flag
RCU_FLAG_PORRST	power reset flag
RCU_FLAG BORRST	BOR reset flag
RCU_FLAG_SWRST	software reset flag
RCU_FLAG_FWDGTR ST	free watchdog timer reset flag
RCU_FLAG_WWDGTR ST	window watchdog timer reset flag
RCU_FLAG_LPRST	low-power reset flag

Enum rcu_int_flag_enum

Table 3-814. Enum rcu_int_flag_enum

enum name	Function description
RCU_INT_FLAG_IRC3 2KSTB	IRC32K stabilization interrupt flag
RCU_INT_FLAG_LXTA LSTB	LXTAL stabilization interrupt flag
RCU_INT_FLAG_IRC1 6MSTB	IRC16M stabilization interrupt flag
RCU_INT_FLAG_HXTA LSTB	HXTAL stabilization interrupt flag
RCU_INT_FLAG_PLLS TB	PLL stabilization interrupt flag
RCU_INT_FLAG_PLLI2 SSTB	PLLI2S stabilization interrupt flag
RCU_INT_FLAG_PLLS AISTB	PLLSAI stabilization interrupt flag
RCU_INT_FLAG_CKM	HXTAL clock stuck interrupt flag
RCU_INT_FLAG_IRC4 8MSTB	IRC48M stabilization interrupt flag

Enum rcu_int_flag_clear_enum

Table 3-815. Enum rcu_int_flag_clear_enum

enum name	Function description
RCU_INT_FLAG_IRC3 2KSTB_CLR	IRC32K stabilization interrupt flag clear
RCU_INT_FLAG_LXTA LSTB_CLR	LXTAL stabilization interrupt flag clear
RCU_INT_FLAG_IRC1 6MSTB_CLR	IRC16M stabilization interrupt flag clear
RCU_INT_FLAG_HXTA LSTB_CLR	HXTAL stabilization interrupt flag clear
RCU_INT_FLAG_PLLS TB_CLR	PLL stabilization interrupt flag clear
RCU_INT_FLAG_PLLI2 SSTB_CLR	PLLI2S stabilization interrupt flag clear
RCU_INT_FLAG_PLLS AISTB_CLR	PLLSAI stabilization interrupt flag clear
RCU_INT_FLAG_CKM _CLR	clock stuck interrupt flag clear
RCU_INT_FLAG_IRC4 8MSTB_CLR	IRC48M stabilization interrupt flag clear

Enum rcu_int_enum

Table 3-816. Enum rcu_int_enum

enum name	Function description
RCU_INT_IRC32KSTB	IRC32K stabilization interrupt enable
RCU_INT_LXTALSTB	LXTAL stabilization interrupt enable
RCU_INT_IRC16MSTB	IRC16M stabilization interrupt enable
RCU_INT_IRC48MSTB	IRC48M stabilization interrupt enable
RCU_INT_HXTALSTB	HXTAL stabilization interrupt enable
RCU_INT_PLLSTB	PLL stabilization interrupt enable
RCU_INT_PLLI2SSTB	PLLI2S stabilization interrupt enable
RCU_INT_PLLSAISTB	PLLSAI stabilization interrupt enable

Enum rcu_osci_type_enum

Table 3-817. Enum rcu_osci_type_enum

enum name	Function description
RCU_HXTAL	high speed crystal oscillator(HXTAL)
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)
RCU_PLLI2S_CK	PLLI2S phase locked loop
RCU_PLLSAI_CK	PLLSAI phase locked loop

Enum rcu_clock_freq_enum

Table 3-818. Enum rcu_clock_freq_enum

enum name	Function description
CK_SYS	system clock frequency
CK_AHB	AHB clock frequency
CK_APB1	APB1 clock frequency
CK_APB2	APB2 clock frequency

Enum i2c_idx_enum

Table 3-819. Enum i2c_idx_enum

enum name	Function description
IDX_I2C3	idnex of I2C3
IDX_I2C4	idnex of I2C4
IDX_I2C5	idnex of I2C5

rcu_deinit

The description of rcu_deinit is shown as below:

Table 3-820. Function rcu_deinit

Function name	rcu_deinit
Function prototype	void rcu_deinit(void);
Function descriptions	deinitialize the RCU
Precondition	-
The called functions	rcu_osci_stab_wait
Input parameter{in}	
-	-
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* reset RCU */
rcu_deinit();
```

rcu_periph_clock_enable

The description of rcu_periph_clock_enable is shown as below:

Table 3-821. Function rcu_periph_clock_enable

Function name	rcu_periph_clock_enable
Function prototype	void rcu_periph_clock_enable(rcu_periph_enum periph);
Function descriptions	enable the peripherals clock
Precondition	-
The called functions	-
Input parameter{in}	
periph	RCU peripherals, refer to Table 3-810. Enum rcu_periph_enum
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 3-822. Function rcu_periph_clock_disable

Function name	rcu_periph_clock_disable
Function prototype	void rcu_periph_clock_disable(rcu_periph_enum periph);
Function descriptions	disable the peripherals clock
Precondition	-
The called functions	-
Input parameter{in}	
periph	RCU peripherals, refer to Table 3-810. Enum rcu_periph_enum
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* 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 3-823. Function rcu_periph_clock_sleep_enable

Function name	rcu_periph_clock_sleep_enable
Function prototype	void rcu_periph_clock_sleep_enable(rcu_periph_sleep_enum periph);
Function descriptions	enable the peripherals clock when in sleep mode
Precondition	-
The called functions	-
Input parameter{in}	
periph	RCU peripherals, refer to Table 3-811. Enum rcu_periph_sleep_enum
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* enable the CRC clock when in sleep mode */
rcu_periph_clock_sleep_enable(RCU_CRC_SLP);
```

rcu_periph_clock_sleep_disable

The description of rcu_periph_clock_sleep_disable is shown as below:

Table 3-824. Function rcu_periph_clock_sleep_disable

Function name	rcu_periph_clock_sleep_disable
----------------------	--------------------------------

Function prototype	void rcu_periph_clock_sleep_disable(rcu_periph_sleep_enum periph);
Function descriptions	disable the peripherals clock when in sleep mode
Precondition	-
The called functions	-
Input parameter{in}	
periph	RCU peripherals, refer to Table 3-811. Enum rcu_periph_sleep_enum
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* disable the CRC clock when in sleep mode */
```

```
rcu_periph_clock_sleep_disable(RCU_CRC_SLP);
```

rcu_periph_reset_enable

The description of rcu_periph_reset_enable is shown as below:

Table 3-825. Function rcu_periph_reset_enable

Function name	rcu_periph_reset_enable
Function prototype	void rcu_periph_reset_enable(rcu_periph_reset_enum periph_reset);
Function descriptions	enable the peripherals reset
Precondition	-
The called functions	-
Input parameter{in}	
periph_reset	RCU peripherals reset, refer to Table 3-812. Enum rcu_periph_reset_enum
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* enable SPI0 reset */
```

```
rcu_periph_reset_enable(RCU_SPI0RST);
```

rcu_periph_reset_disable

The description of rcu_periph_reset_disable is shown as below:

Table 3-826. Function rcu_periph_reset_disable

Function name	rcu_periph_reset_disable
Function prototype	void rcu_periph_reset_disable(rcu_periph_reset_enum periph_reset);

Function descriptions	disable the peripheral reset
Precondition	-
The called functions	-
Input parameter{in}	
periph_reset	RCU peripherals reset, refer to Table 3-812. Enum rcu_periph_reset_enum
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* 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 3-827. Function rcu_bkp_reset_enable

Function name	rcu_bkp_reset_enable
Function prototype	void rcu_bkp_reset_enable(void);
Function descriptions	enable the BKP domain reset
Precondition	-
The called functions	-
Input parameter{in}	
-	-
Output parameter{out}	
-	-
Return value	
-	-

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 3-828. Function rcu_bkp_reset_disable

Function name	rcu_bkp_reset_disable
Function prototype	void rcu_bkp_reset_disable(void);
Function descriptions	disable the BKP domain reset

Precondition	-
The called functions	-
Input parameter{in}	
-	-
Output parameter{out}	
-	-
Return value	
-	-

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 3-829. Function rcu_system_clock_source_config

Function name	rcu_system_clock_source_config
Function prototype	void rcu_system_clock_source_config(uint32_t ck_sys);
Function descriptions	configure the system clock source
Precondition	-
The called functions	-
Input parameter{in}	
ck_sys	system clock source select
RCU_CKSYSSRC_IRC 16M	select CK_IRC16M as the CK_SYS source
RCU_CKSYSSRC_HX TAL	select CK_HXTAL as the CK_SYS source
RCU_CKSYSSRC_PLL P	select CK_PLLP as the CK_SYS source
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* 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 3-830. Function rcu_system_clock_source_get

Function name	rcu_system_clock_source_get
Function prototype	uint32_t rcu_system_clock_source_get(void);
Function descriptions	get the system clock source
Precondition	-
The called functions	-
Input parameter{in}	
-	-
Output parameter{out}	
-	-
Return value	
uint32_t	which clock is selected as CK_SYS source
RCU_SCSS_IRC16M	CK_IRC16M is selected as the CK_SYS source
RCU_SCSS_HXTAL	CK_HXTAL is selected as the CK_SYS source
RCU_SCSS_PLLP	CK_PLLP is selected as the CK_SYS source

Example:

```
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 3-831. Function rcu_ahb_clock_config

Function name	rcu_ahb_clock_config
Function prototype	void rcu_ahb_clock_config(uint32_t ck_ahb);
Function descriptions	configure the AHB clock prescaler selection
Precondition	-
The called functions	-
Input parameter{in}	
ck_ahb	AHB clock prescaler selection
RCU_AHB_CKSYS_DIVx	select CK_SYS / x, (x=1, 2, 4, 8, 16, 64, 128, 256, 512)
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* 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 3-832. Function rcu_apb1_clock_config

Function name	rcu_apb1_clock_config
Function prototype	void rcu_apb1_clock_config(uint32_t ck_apb1);
Function descriptions	configure the APB1 clock prescaler selection
Precondition	-
The called functions	-
Input parameter{in}	
ck_apb1	APB1 clock prescaler selection
RCU_APB1_CKAHB_D IV1	select CK_AHB as CK_APB1
RCU_APB1_CKAHB_D IV2	select CK_AHB/2 as CK_APB1
RCU_APB1_CKAHB_D IV4	select CK_AHB/4 as CK_APB1
RCU_APB1_CKAHB_D IV8	select CK_AHB/8 as CK_APB1
RCU_APB1_CKAHB_D IV16	select CK_AHB/16 as CK_APB1
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* 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 3-833. Function rcu_apb2_clock_config

Function name	rcu_apb2_clock_config
Function prototype	void rcu_apb2_clock_config(uint32_t ck_apb2);
Function descriptions	configure the APB2 clock prescaler selection
Precondition	-
The called functions	-
Input parameter{in}	
ck_apb2	APB2 clock prescaler selection
RCU_APB2_CKAHB_D	select CK_AHB as CK_APB2

IV1	
RCU_APB2_CKAHB_D	select CK_AHB/2 as CK_APB2
IV2	
RCU_APB2_CKAHB_D	select CK_AHB/4 as CK_APB2
IV4	
RCU_APB2_CKAHB_D	select CK_AHB/8 as CK_APB2
IV8	
RCU_APB2_CKAHB_D	select CK_AHB/16 as CK_APB2
IV16	
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* configure CK_AHB/8 as CK_APB2 */
```

```
rcu_apb2_clock_config(RCU_APB2_CKAHB_DIV8);
```

rcu_ckout0_config

The description of rcu_ckout0_config is shown as below:

Table 3-834. Function rcu_ckout0_config

Function name	rcu_ckout0_config
Function prototype	void rcu_ckout0_config(uint32_t ckout0_src, uint32_t ckout0_div);
Function descriptions	configure the CK_OUT0 clock source
Precondition	-
The called functions	-
Input parameter{in}	
ckout0_src	CK_OUT0 clock source selection
RCU_CKOUT0SRC_IRC16M	select IRC16M clock
RCU_CKOUT0SRC_LXTAL	select LXTAL clock
RCU_CKOUT0SRC_HXTAL	select HXTAL clock
RCU_CKOUT0SRC_CKPLL_PLLP	select PLLP clock
Input parameter{in}	
ckout0_div	CK_OUT0 divider
RCU_CKOUT0_DIVx(x=1,2,3,4,5)	CK_OUT0 is divided by x
Output parameter{out}	
-	-

Return value	
-	-

Example:

```
/* configure the HXTAL as CK_OUT0 clock source */
```

```
rcu_ckout0_config(RCU_CKOUT0SRC_HXTAL, RCU_CKOUT0_DIV1);
```

rcu_ckout1_config

The description of rcu_ckout1_config is shown as below:

Table 3-835. Function rcu_ckout1_config

Function name	rcu_ckout1_config
Function prototype	void rcu_ckout1_config(uint32_t ckout1_src, uint32_t ckout1_div);
Function descriptions	configure the CK_OUT1 clock source and divider
Precondition	-
The called functions	-
Input parameter{in}	
ckout1_src	CK_OUT1 clock source selection
RCU_CKOUT1SRC_SY STEMCLOCK	select system clock
RCU_CKOUT1SRC_PL LI2SR	select PLLI2SR clock
RCU_CKOUT1SRC_H XTAL	select HXTAL clock
RCU_CKOUT1SRC_PL LP	select PLLP clock
Input parameter{in}	
ckout1_div	CK_OUT1 divider
RCU_CKOUT1_DIVx(x =1,2,3,4,5)	CK_OUT1 is divided by x
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* configure the HXTAL as CK_OUT1 clock source */
```

```
rcu_ckout1_config(RCU_CKOUT1SRC_HXTAL, RCU_CKOUT1_DIV1);
```

rcu_pll_config

The description of rcu_pll_config is shown as below:

Table 3-836. Function rcu_pll_config

Function name	rcu_pll_config
Function prototype	ErrStatus rcu_pll_config(uint32_t pll_src, uint32_t pll_psc, uint32_t pll_n, uint32_t pll_p, uint32_t pll_q);
Function descriptions	configure the main PLL clock
Precondition	-
The called functions	-
Input parameter{in}	
pll_src	PLL clock source selection
RCU_PLLSRC_IRC16M	IRC16M clock is selected as source clock of PLL, PLLSAI, PLLI2S
RCU_PLLSRC_HXTAL	HXTAL is selected as source clock of PLL, PLLSAI, PLLI2S
Input parameter{in}	
pll_psc	the PLL VCO source clock prescaler
uint32_t	2~63
Input parameter{in}	
pll_n	the PLL VCO clock multi factor
uint32_t	64~500
Input parameter{in}	
pll_p	the PLLP output frequency division factor from PLL VCO clock
uint32_t	2,4,6,8
Input parameter{in}	
pll_q	the PLL Q output frequency division factor from PLL VCO clock
uint32_t	2~15
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* Configure the main PLL, PSC = 8, PLL_N = 240, PLL_P = 2, PLL_Q = 5 */
```

```
rcu_pll_config(RCU_PLLSRC_HXTAL,8,240,2,5);
```

rcu_plli2s_q_config

The description of rcu_plli2s_q_config is shown as below:

Table 3-837. Function rcu_plli2s_config

Function name	rcu_plli2s_q_config
Function prototype	ErrStatus rcu_plli2s_q_config(uint32_t plli2s_n, uint32_t plli2s_q);
Function descriptions	configure the PLLI2S_Q clock
Precondition	-
The called functions	-
Input parameter{in}	

plli2s_n	the PLLI2S VCO clock multi factor
uint32_t	50~500
Input parameter{in}	
plli2s_q	the PLLI2S Q output frequency division factor from PLLI2S VCO clock
uint32_t	2~15
Output parameter{out}	
-	-
Return value	
ErrStatus	SUCCESS or ERROR

Example:

```
/* configure the PLLI2S n = 100, PLLI2S q =2*/
```

```
rcu_plli2s_q_config (100, 2);
```

rcu_plli2s_r_config

The description of rcu_plli2s_r_config is shown as below:

Table 3-838. Function rcu_plli2s_r_config

Function name	rcu_plli2s_r_config
Function prototype	ErrStatus rcu_plli2s_r_config(uint32_t plli2s_n, uint32_t plli2s_r);
Function descriptions	configure the PLLI2S_R clock
Precondition	-
The called functions	-
Input parameter{in}	
plli2s_n	the PLLI2S VCO clock multi factor
uint32_t	50~500
Input parameter{in}	
plli2s_r	the PLLI2S R output frequency division factor from PLLI2S VCO clock
uint32_t	2~7
Output parameter{out}	
-	-
Return value	
ErrStatus	SUCCESS or ERROR

Example:

```
/* configure the PLLI2S n = 100, PLLI2S r =2*/
```

```
rcu_plli2s_r_config (100, 2);
```

rcu_pll Sai_p_config

The description of rcu_pll Sai_p_config is shown as below:

Table 3-839. Function rcu_pllsai_p_config

Function name	rcu_pllsai_p_config
Function prototype	ErrStatus rcu_pllsai_p_config(uint32_t pllsai_n, uint32_t pllsai_p);
Function descriptions	configure the PLLSAI_P clock
Precondition	-
The called functions	-
Input parameter{in}	
pllsai_n	the PLLSAI VCO clock multi factor
uint32_t	50~500
Input parameter{in}	
pllsai_p	the PLLSAI P output frequency division factor from PLL VCO clock
uint32_t	2,4,6,8
Output parameter{out}	
-	-
Return value	
ErrStatus	SUCCESS or ERROR

Example:

```
/* configure the PLLSAI n = 100, PLLSAI p = 2 */
```

```
rcu_pllsai_p_config (100, 2);
```

rcu_pllsai_q_config

The description of rcu_pllsai_config is shown as below:

Table 3-840. Function rcu_pllsai_q_config

Function name	rcu_pllsai_q_config
Function prototype	ErrStatus rcu_pllsai_q_config(uint32_t pllsai_n, uint32_t pllsai_q);
Function descriptions	configure the PLLSAI_Q clock
Precondition	-
The called functions	-
Input parameter{in}	
pllsai_n	the PLLSAI VCO clock multi factor
uint32_t	50~500
Input parameter{in}	
pllsai_q	the PLLSAI P output frequency division factor from PLL VCO clock
uint32_t	2~15
Output parameter{out}	
-	-
Return value	
ErrStatus	SUCCESS or ERROR

Example:

```
/* configure the PLLSAI n = 100, PLLSAI q = 2*/
```

```
rcu_pll Sai config (100, 2);
```

rcu_pll Sai r config

The description of rcu_pll Sai r config is shown as below:

Table 3-841. Function rcu_pll Sai r config

Function name	rcu_pll Sai r config
Function prototype	ErrStatus rcu_pll Sai r config(uint32_t pll Sai n, uint32_t pll Sai r);
Function descriptions	configure the PLLSAI_R clock
Precondition	-
The called functions	-
Input parameter{in}	
pll Sai n	the PLLSAI VCO clock multi factor
uint32_t	50~500
Input parameter{in}	
pll Sai r	the PLLSAI R output frequency division factor from PLL VCO clock
uint32_t	2~7
Output parameter{out}	
-	-
Return value	
ErrStatus	SUCCESS or ERROR

Example:

```
/* configure the PLLSAI n = 100, PLLSAI r = 2 */
```

```
rcu_pll Sai config (100, 2);
```

rcu_rtc_clock_config

The description of rcu_rtc_clock_config is shown as below:

Table 3-842. Function rcu_rtc_clock_config

Function name	rcu_rtc_clock_config
Function prototype	void rcu_rtc_clock_config(uint32_t rtc_clock_source);
Function descriptions	configure the RTC clock source selection
Precondition	-
The called functions	-
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_IRC32K	select CK_IRC32K as RTC source clock
RCU_RTCSRC_HXTAL_DIV_RTCDIV	select CK_HXTAL or RTCDIV as RTC source clock

Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* configure the RTC clock source selection */
rcu_rtc_clock_config(RCU_RTCSRC_IRC32K);
```

rcu_rtc_div_config

The description of rcu_rtc_div_config is shown as below:

Table 3-843. Function rcu_rtc_div_config

Function name	rcu_rtc_div_config
Function prototype	void rcu_rtc_div_config(uint32_t rtc_div);
Function descriptions	configure the frequency division of RTC clock when HXTAL was selected as its clock source
Precondition	-
The called functions	-
Input parameter{in}	
rtc_div	RTC clock frequency division
RCU_RTC_HXTAL_NO NE	no clock for RTC
RCU_RTC_HXTAL_DIVx	RTCDIV clock select CK_HXTAL/x, x = 2,3....31
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* configure the frequency division of RTC clock when HXTAL was selected as its clock source */
rcu_rtc_div_config(RCU_RTC_HXTAL_DIV6);
```

rcu_i2s_clock_config

The description of rcu_i2s_clock_config is shown as below:

Table 3-844. Function rcu_i2s_clock_config

Function name	rcu_i2s_clock_config
Function prototype	void rcu_i2s_clock_config(uint32_t i2s_clock_source);
Function descriptions	configure the I2S clock source selection

Precondition	-
The called functions	-
Input parameter{in}	
i2s_clock_source	I2S clock source selection
RCU_I2SSRC_PLLI2S	CK_PLLI2S selected as I2S source clock
RCU_I2SSRC_I2S_CKIN	external i2s_ckin pin selected as I2S source clock
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* configure the I2S clock source selection */
rcu_i2s_clock_config(RCU_I2SSRC_PLLI2S);
```

rcu_i2c_clock_config

The description of rcu_i2c_clock_config is shown as below:

Table 3-845. Function rcu_i2c_clock_config

Function name	rcu_i2c_clock_config
Function prototype	void rcu_i2c_clock_config(i2c_idx_enum i2c_idx, uint32_t ck_i2c);
Function descriptions	configure the I2Cx(x=3,4,5) clock source selection
Precondition	-
The called functions	-
Input parameter{in}	
i2c_idx	IDX_I2Cx(x=3,4,5)
IDX_I2C3	idnex of I2C3
IDX_I2C4	idnex of I2C4
IDX_I2C5	idnex of I2C5
Input parameter{in}	
ck_i2c	PLLSAIR divider used as input of TLI
RCU_I2CSRC_CKAPB1	CK_I2C select CK_APB1
RCU_I2CSRC_PLLSAIR	CK_I2C select CK_PLLSAIR
RCU_I2CSRC_IRC16M	CK_I2C select IRC16M
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* configure the I2C3 clock source selection */
```

```
rcu_i2c_clock_config(IDX_I2C3, RCU_I2CSRC_CKAPB1);
```

rcu_sai_clock_config

The description of rcu_sai_clock_config is shown as below:

Table 3-846. Function rcu_sai_clock_config

Function name	rcu_sai_clock_config
Function prototype	void rcu_sai_clock_config(uint32_t sai_clock_source);
Function descriptions	configure the SAI clock source selection
Precondition	-
The called functions	-
Input parameter{in}	
sai_clock_source	SAI clock source selection
RCU_SAISRC_PLLSAI Q	CK_PLLSAIQ selected as SAI source clock
RCU_SAISRC_PLLI2S Q	CK_PLLI2SQ selected as SAI source clock
RCU_SAISRC_I2S_CK IN	CK_I2S_CKIN selected as SAI source clock
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* configure the SAI clock source selection */
```

```
rcu_sai_clock_config(RCU_SAISRC_PLLSAIQ);
```

rcu_ck48m_clock_config

The description of rcu_ck48m_clock_config is shown as below:

Table 3-847. Function rcu_ck48m_clock_config

Function name	rcu_ck48m_clock_config
Function prototype	void rcu_ck48m_clock_config(uint32_t ck48m_clock_source);
Function descriptions	configure the CK48M clock source selection
Precondition	-
The called functions	-
Input parameter{in}	
ck48m_clock_source	CK48M clock source selection
RCU_CK48MSRC_PLL 48M	CK_PLL48M selected as CK48M source clock

RCU_CK48MSRC_IRC 48M	CK_IRC48M selected as CK48M source clock
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* configure the CK48M clock source selection */
```

```
rcu_ck48m_clock_config (RCU_CK48MSRC_IRC48M);
```

rcu_pll48m_clock_config

The description of rcu_pll48m_clock_config is shown as below:

Table 3-848. Function rcu_pll48m_clock_config

Function name	rcu_pll48m_clock_config
Function prototype	void rcu_pll48m_clock_config(uint32_t pll48m_clock_source);
Function descriptions	configure the PLL48M clock source selection
Precondition	-
The called functions	-
Input parameter{in}	
pll48m_clock_source	PLL48M clock source selection
RCU_PLL48MSRC_PL LQ	CK_PLLQ selected as PLL48M source clock
RCU_PLL48MSRC_PL LSAIP	CK_PLLSAIP selected as PLL48M source clock
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* configure the PLL48M clock selection */
```

```
rcu_pll48m_clock_config(RCU_PLL48MSRC_PLLQ);
```

rcu_timer_clock_prescaler_config

The description of rcu_timer_clock_prescaler_config is shown as below:

Table 3-849. Function rcu_timer_clock_prescaler_config

Function name	rcu_timer_clock_prescaler_config
Function prototype	void rcu_timer_clock_prescaler_config(uint32_t timer_clock_prescaler);
Function descriptions	configure the TIMER clock prescaler selection

Precondition	-
The called functions	-
Input parameter{in}	
timer_clock_prescaler	TIMER source selection
RCU_TIMER_PSC_MU L2	If CK_APBx = CK_AHB or CK_APBx = CK_AHB/2, CK_TIMERx = CK_AHB, else CK_TIMERx = 2 x CK_APBx
RCU_TIMER_PSC_MU L4	If CK_APBx = CK_AHB or CK_APBx = CK_AHB/2 or CK_APBx = CK_AHB/4, CK_TIMERx = CK_AHB, else CK_TIMERx = 4 x CK_APBx
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* configure TIMER clock source */
```

```
rcu_timer_clock_prescaler_config(RCU_TIMER_PSC_MUL4);
```

rcu_tli_clock_div_config

The description of rcu_tli_clock_div_config is shown as below:

Table 3-850. Function rcu_tli_clock_div_config

Function name	rcu_tli_clock_div_config
Function prototype	void rcu_tli_clock_div_config(uint32_t pllsai_r_div);
Function descriptions	configure the PLLSAIR divider used as input of TLI
Precondition	-
The called functions	-
Input parameter{in}	
pllsai_r_div	PLLSAIR divider used as input of TLI
RCU_PLLSAIR_DIV2	PLLSAIR/2
RCU_PLLSAIR_DIV4	PLLSAIR/4
RCU_PLLSAIR_DIV8	PLLSAIR/8
RCU_PLLSAIR_DIV16	PLLSAIR/16
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* configure the TLI prescaler factor from PLLSAIR clock */
```

```
rcu_tli_clock_div_config (RCU_PLLSAIR_DIV4);
```

rcu_lxtal_drive_capability_config

The description of rcu_lxtal_drive_capability_config is shown as below:

Table 3-851. Function rcu_lxtal_drive_capability_config

Function name	rcu_lxtal_drive_capability_config
Function prototype	void rcu_lxtal_drive_capability_config(uint32_t lxtal_dricap);
Function descriptions	configure the LXTAL drive capability
Precondition	-
The called functions	-
Input parameter{in}	
lxtal_dricap	drive capability of LXTAL
RCU_LXTAL_LOWDRI	lower driving capability
RCU_LXTAL_HIGHDRI	higher driving capability
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* configure the LXTAL drive capability */
rcu_lxtal_drive_capability_config(RCU_LXTAL_LOWDRI);
```

rcu_osci_stab_wait

The description of rcu_osci_stab_wait is shown as below:

Table 3-852. Function rcu_osci_stab_wait

Function name	rcu_osci_stab_wait
Function prototype	ErrStatus rcu_osci_stab_wait(rcu_osci_type_enum osci);
Function descriptions	wait for oscillator stabilization flags is SET or oscillator startup is timeout
Precondition	-
The called functions	rcu_flag_get
Input parameter{in}	
osci	oscillator types, refer to Table 3-817. Enum rcu_osci_type_enum
Output parameter{out}	
-	-
Return value	
ErrStatus	SUCCESS or ERROR

Example:

```
/* wait for oscillator stabilization flag */
if(SUCCESS == rcu_osci_stab_wait(RCU_HXTAL)){
}
}
```

rcu_osc_i_on

The description of rcu_osc_i_on is shown as below:

Table 3-853. Function rcu_osc_i_on

Function name	rcu_osc_i_on
Function prototype	void rcu_osc_i_on(rcu_osc_i_type_enum osci);
Function descriptions	turn on the oscillator
Precondition	-
The called functions	-
Input parameter{in}	
osci	oscillator types, refer to Table 3-817. Enum rcu_osc_i_type_enum
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* turn on the high speed crystal oscillator */
rcu_osc_i_on(RCU_HXTAL);
```

rcu_osc_i_off

The description of rcu_osc_i_off is shown as below:

Table 3-854. Function rcu_osc_i_off

Function name	rcu_osc_i_off
Function prototype	void rcu_osc_i_off(rcu_osc_i_type_enum osci);
Function descriptions	turn off the oscillator
Precondition	-
The called functions	-
Input parameter{in}	
osci	oscillator types, refer to Table 3-817. Enum rcu_osc_i_type_enum
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* turn off the high speed crystal oscillator */
rcu_osc_i_off(RCU_HXTAL);
```

rcu_osc_i_bypass_mode_enable

The description of rcu_osc_i_bypass_mode_enable is shown as below:

Table 3-855. Function rcu_osci_bypass_mode_enable

Function name	rcu_osci_bypass_mode_enable
Function prototype	void rcu_osci_bypass_mode_enable(rcu_osci_type_enum osci);
Function descriptions	enable the oscillator bypass mode
Precondition	HXTALEN or LXTALEN must be reset before it
The called functions	-
Input parameter{in}	
osci	oscillator types, refer to Table 3-817. Enum rcu_osci_type_enum
RCU_HXTAL	high speed crystal oscillator(HXTAL)
RCU_LXTAL	low speed crystal oscillator(LXTAL)
Output parameter{out}	
-	-
Return value	
-	-

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 3-856. Function rcu_osci_bypass_mode_disable

Function name	rcu_osci_bypass_mode_disable
Function prototype	void rcu_osci_bypass_mode_disable(rcu_osci_type_enum osci);
Function descriptions	disable the oscillator bypass mode
Precondition	HXTALEN or LXTALEN must be reset before it
The called functions	-
Input parameter{in}	
osci	oscillator types, refer to Table 3-817. Enum rcu_osci_type_enum
RCU_HXTAL	high speed crystal oscillator(HXTAL)
RCU_LXTAL	low speed crystal oscillator(LXTAL)
Output parameter{out}	
-	-
Return value	
-	-

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 3-857. Function rcu_hxtal_clock_monitor_enable

Function name	rcu_hxtal_clock_monitor_enable
Function prototype	void rcu_hxtal_clock_monitor_enable(void);
Function descriptions	enable the HXTAL clock monitor
Precondition	-
The called functions	-
Input parameter{in}	
-	-
Output parameter{out}	
-	-
Return value	
-	-

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 3-858. Function rcu_hxtal_clock_monitor_disable

Function name	rcu_hxtal_clock_monitor_disable
Function prototype	void rcu_hxtal_clock_monitor_disable(void);
Function descriptions	disable the HXTAL clock monitor
Precondition	-
The called functions	-
Input parameter{in}	
-	-
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* disable the HXTAL clock monitor */
rcu_hxtal_clock_monitor_disable();
```

rcu_irc16m_adjust_value_set

The description of rcu_irc16m_adjust_value_set is shown as below:

Table 3-859. Function rcu_irc16m_adjust_value_set

Function name	rcu_irc16m_adjust_value_set
Function prototype	void rcu_irc16m_adjust_value_set(uint8_t irc16m_adjval);
Function descriptions	set the IRC16M adjust value
Precondition	-
The called functions	-
Input parameter{in}	
irc16m_adjval	IRC16M adjust value, must be between 0 and 0x1F
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* set the IRC16M adjust value */
rcu_irc16m_adjust_value_set(0x10);
```

rcu_spread_spectrum_config

The description of rcu_spread_spectrum_config is shown as below:

Table 3-860. Function rcu_spread_spectrum_config

Function name	rcu_spread_spectrum_config
Function prototype	void rcu_spread_spectrum_config(uint32_t spread_spectrum_type, uint32_t modstep, uint32_t modcnt)
Function descriptions	configure the spread spectrum modulation for the main PLL clock
Precondition	-
The called functions	-
Input parameter{in}	
spread_spectrum_type	PLL spread spectrum modulation type select
RCU_SS_TYPE_CENTER	center spread type is selected
RCU_SS_TYPE_DOWN	down spread type is selected
Input parameter{in}	
modstep	configure PLL spread spectrum modulation profile amplitude
uint32_t	0 ~ 0x7FFF, The following criteria must be met: MODSTEP*MODCNT <= 2 ¹⁵ -1
Input parameter{in}	
modcnt	configure PLL spread spectrum modulation profile frequency
uint32_t	0 ~ 0x1FFF, The following criteria must be met: MODSTEP*MODCNT <= 2 ¹⁵ -1
Output parameter{out}	

-	-
Return value	
-	-

Example:

```
/* configure PLL spread_spectrum */
```

```
rcu_spread_spectrum_config (RCU_SS_TYPE_CENTER, 0x0F,0x0F);
```

rcu_spread_spectrum_enable

The description of rcu_spread_spectrum_enable is shown as below:

Table 3-861. Function rcu_spread_spectrum_enable

Function name	rcu_spread_spectrum_enable
Function prototype	void rcu_spread_spectrum_enable(void);
Function descriptions	enable the PLL spread spectrum modulation
Precondition	-
The called functions	-
Input parameter{in}	
-	-
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* enable the PLL spread spectrum modulation */
```

```
rcu_spread_spectrum_enable ();
```

rcu_spread_spectrum_disable

The description of rcu_spread_spectrum_disable is shown as below:

Table 3-862. Function rcu_spread_spectrum_disable

Function name	rcu_spread_spectrum_disable
Function prototype	void rcu_spread_spectrum_disable(void);
Function descriptions	disable the HXTAL clock monitor
Precondition	-
The called functions	-
Input parameter{in}	
-	-
Output parameter{out}	
-	-
Return value	

-	-
---	---

Example:

```
/* disable the PLL spread spectrum modulation */
```

```
rcu_spread_spectrum_disable();
```

rcu_voltage_key_unlock

The description of rcu_voltage_key_unlock is shown as below:

Table 3-863. Function rcu_voltage_key_unlock

Function name	rcu_voltage_key_unlock
Function prototype	void rcu_voltage_key_unlock (void);
Function descriptions	unlock the voltage key
Precondition	-
The called functions	-
Input parameter{in}	
-	-
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* unlock the voltage key register */
```

```
rcu_voltage_key_unlock ();
```

rcu_deepsleep_voltage_set

The description of rcu_deepsleep_voltage_set is shown as below:

Table 3-864. Function rcu_deepsleep_voltage_set

Function name	rcu_deepsleep_voltage_set
Function prototype	void rcu_deepsleep_voltage_set(uint32_t dsvol);
Function descriptions	set the deep-sleep mode voltage value
Precondition	-
The called functions	-
Input parameter{in}	
dsvol	deep sleep mode voltage
RCU_DEEPSLEEP_V_0	the core voltage is default value in deep-sleep mode
RCU_DEEPSLEEP_V_1	the core voltage is (default value-0.1)V in deep-sleep mode(customers are not recommended to use it)
RCU_DEEPSLEEP_V_2	the core voltage is (default value-0.2)V in deep-sleep mode(customers are not recommended to use it)

2	not recommended to use it)
RCU_DEEPSLEEP_V_3	the core voltage is (default value-0.3)V in deep-sleep mode(customers are not recommended to use it)
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* set the deep-sleep mode voltage */
rcu_deepsleep_voltage_set(RCU_DEEPSLEEP_V_1);
```

rcu_clock_freq_get

The description of rcu_clock_freq_get is shown as below:

Table 3-865. Function rcu_clock_freq_get

Function name	rcu_clock_freq_get
Function prototype	uint32_t rcu_clock_freq_get(rcu_clock_freq_enum clock);
Function descriptions	get the system clock, bus clock frequency
Precondition	-
The called functions	-
Input parameter{in}	
clock	the clock frequency which to get, refer to Table 3-818. Enum rcu_clock_freq_enum
Output parameter{out}	
-	-
Return value	
ck_freq	clock frequency of system, AHB, APB1, APB2

Example:

```
uint32_t temp_freq;

/* get the system clock frequency */
temp_freq = rcu_clock_freq_get(CK_SYS);
```

rcu_flag_get

The description of rcu_flag_get is shown as below:

Table 3-866. Function rcu_flag_get

Function name	rcu_flag_get
Function prototype	FlagStatus rcu_flag_get(rcu_flag_enum flag);
Function descriptions	get the clock stabilization and peripheral reset flags
Precondition	-

The called functions	-
Input parameter{in}	
flag	the clock stabilization and peripheral reset flags, refer to Table 3-813. Enum rcu_flag_enum
Output parameter{out}	
-	-
Return value	
FlagStatus	SET or RESET

Example:

```
/* 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 3-867. Function rcu_all_reset_flag_clear

Function name	rcu_all_reset_flag_clear
Function prototype	void rcu_all_reset_flag_clear(void);
Function descriptions	clear all the reset flag
Precondition	-
The called functions	-
Input parameter{in}	
-	-
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* 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 3-868. Function rcu_interrupt_flag_get

Function name	rcu_interrupt_flag_get
Function prototype	FlagStatus rcu_interrupt_flag_get(rcu_int_flag_enum int_flag);
Function descriptions	get the clock stabilization interrupt and ckm flags
Precondition	-

The called functions	-
Input parameter{in}	
int_flag	interrupt and ckm flags, refer to Table 3-814. Enum rcu_int_flag_enum
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 3-869. Function rcu_interrupt_flag_clear

Function name	rcu_interrupt_flag_clear
Function prototype	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}	
int_flag_clear	clock stabilization and stuck interrupt flags clear, refer to Table 3-815. Enum rcu_int_flag_clear_enum
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* 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 3-870. Function rcu_interrupt_enable

Function name	rcu_interrupt_enable
Function prototype	void rcu_interrupt_enable(rcu_int_enum stab_int);
Function descriptions	enable the stabilization interrupt
Precondition	-

The called functions	-
Input parameter{in}	
stab_int	clock stabilization interrupt, refer to Table 3-816. Enum rcu_int_enum
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 3-871. Function rcu_interrupt_disable

Function name	rcu_interrupt_disable
Function prototype	void rcu_interrupt_disable(rcu_int_enum stab_int);
Function descriptions	disable the stabilization interrupt
Precondition	-
The called functions	-
Input parameter{in}	
stab_int	clock stabilization interrupt, refer to Table 3-816. Enum rcu_int_enum
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* disable the HXTAL stabilization interrupt */
```

```
rcu_interrupt_disable(RCU_INT_HXTALSTB);
```

3.25. 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 [3.25.1](#), the RTC firmware functions are introduced in chapter [3.25.2](#).

3.25.1. Descriptions of Peripheral registers

RTC registers are listed in the table shown as below:

Table 3-872. RTC Registers

Registers	Descriptions
RTC_TIME	RTC time of day register
RTC_DATE	RTC date register
RTC_CTL	RTC control register
RTC_STAT	RTC status register
RTC_PSC	RTC time prescaler register
RTC_WUT	RTC wakeup timer register
RTC_COSC	RTC coarse calibration register
RTC_ALRM0TD	RTC alarm 0 time and date register
RTC_ALRM1TD	RTC alarm 1 time and date register
RTC_WPK	RTC write protection key register
RTC_SS	RTC sub second register
RTC_SHIFTCTL	RTC shift function control register
RTC_TTS	RTC time of timestamp register
RTC_DTS	RTC date of timestamp register
RTC_SSTS	RTC sub second of timestamp register
RTC_HRFC	RTC high resolution frequency compensation register
RTC_TAMP	RTC tamper register
RTC_ALRM0SS	RTC alarm 0 sub second register
RTC_ALRM1SS	RTC alarm 1 sub second register
RTC_BKPx(x = 0, 1, 2, ..., 18, 19)	RTC backup register

3.25.2. Descriptions of Peripheral functions

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

Table 3-873. RTC firmware function

Function name	Function description
rtc_deinit	reset most of the RTC registers
rtc_init	initialize RTC registers
rtc_init_mode_enter	enter RTC init mode
rtc_init_mode_exit	exit RTC init mode
rtc_register_sync_wait	wait until RTC_TIME and RTC_DATE registers are synchronized with APB clock, and the shadow registers are updated
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

Function name	Function description
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_subsecond_get	get RTC time-stamp subsecond
rtc_timestamp_pin_map	RTC time-stamp pin map
rtc_tamper_enable	enable RTC tamper
rtc_tamper_disable	disable RTC tamper
rtc_tamper0_pin_map	RTC tamper0 pin map
rtc_interrupt_enable	enable specified RTC interrupt
rtc_interrupt_disable	disble specified RTC interrupt
rtc_flag_get	check specified flag
rtc_flag_clear	clear specified flag
rtc_alter_output_config	configure RTC alternate output source
rtc_calibration_output_config	configure RTC calibration output source
rtc_hour_adjust	ajust the daylight saving time by adding or substracting one hour from the current time
rtc_second_adjust	ajust 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_coarse_calibration_enable	enable RTC coarse calibration
rtc_coarse_calibration_disable	disable RTC coarse calibration
rtc_coarse_calibration_config	configure RTC coarse calibration direction and step

Structure rtc_parameter_struct

Table 3-874. Struct rtc_parameter_struct

Member name	Function description
year	RTC year value: 0x0 - 0x99(BCD format)
month	RTC month value
date	RTC date value: 0x1 - 0x31(BCD format)
day_of_week	RTC weekday value
hour	RTC hour value
minute	RTC minute value: 0x0 - 0x59(BCD format)

second	RTC second value: 0x0 - 0x59(BCD format)
factor_asyn	RTC asynchronous prescaler value: 0x0 - 0x7F
factor_syn	RTC synchronous prescaler value: 0x0 - 0x7FFF
am_pm	RTC AM / PM value
display_format	RTC time notation

Structure rtc_alarm_struct

Table 3-875. Struct rtc_alarm_struct

Member name	Function description
alarm_mask	RTC alarm mask
weekday_or_date	specify RTC alarm is on date or weekday
alarm_day	RTC alarm date or weekday value
alarm_hour	RTC alarm hour value
alarm_minute	RTC alarm minute value: 0x0 - 0x59(BCD format)
alarm_second	RTC alarm second value: 0x0 - 0x59(BCD format)
am_pm	RTC alarm AM/PM value

Structure rtc_timestamp_struct

Table 3-876. Struct rtc_timestamp_struct

Member name	Function description
timestamp_month	RTC time-stamp month value
timestamp_date	RTC time-stamp date value: 0x1 - 0x31(BCD format)
timestamp_day	RTC time-stamp weekday value
timestamp_hour	RTC time-stamp hour value
timestamp_minute	RTC time-stamp minute value: 0x0 - 0x59(BCD format)
timestamp_second	RTC time-stamp second value: 0x0 - 0x59(BCD format)
am_pm	RTC time-stamp AM/PM value

Structure rtc_tamper_struct

Table 3-877. Struct rtc_tamper_struct

Member name	Function description
tamper_source	RTC tamper source
tamper_trigger	RTC tamper trigger
tamper_filter	RTC tamper consecutive samples needed during a voltage level detection
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

amp	
-----	--

rtc_deinit

The description of rtc_deinit is shown as below:

Table 3-878. Function rtc_deinit

Function name	rtc_deinit
Function prototype	ErrStatus rtc_deinit(void);
Function descriptions	reset most of the RTC registers
Precondition	-
The called functions	rcu_periph_reset_enable/ rcu_periph_reset_disable -
Input parameter{in}	
-	-
Output parameter{out}	
-	-
Return value	
ErrStatus	ERROR or SUCCESS

Example:

```
/* reset most of the RTC registers */
```

```
ErrStatus error_status = rtc_deinit();
```

rtc_init

The description of rtc_init is shown as below:

Table 3-879. Function rtc_init

Function name	rtc_init
Function prototype	ErrStatus rtc_init(rtc_parameter_struct* rtc_initpara_struct);
Function descriptions	initialize RTC registers
Precondition	-
The called functions	-
Input parameter{in}	
rtc_initpara_struct	the structure members can refer to members of the structure Table 3-874. Struct rtc_parameter_struct
Output parameter{out}	
-	-
Return value	
ErrStatus	ERROR or SUCCESS

Example:

```
/* reset most of the RTC registers */
```

```
rtc_parameter_struct rtc_initpara_struct;
```



```
ErrStatus error_status = rtc_init(&rtc_initpara_struct);
```

rtc_init_mode_enter

The description of rtc_init_mode_enter is shown as below:

Table 3-880. Function rtc_init_mode_enter

Function name	rtc_init_mode_enter
Function prototype	ErrStatus rtc_init_mode_enter(void);
Function descriptions	enter RTC init mode
Precondition	-
The called functions	-
Input parameter{in}	
-	-
Output parameter{out}	
-	-
Return value	
ErrStatus	ERROR or SUCCESS

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 3-881. Function rtc_init_mode_exit

Function name	rtc_init_mode_exit
Function prototype	void rtc_init_mode_exit(void);
Function descriptions	exit RTC init mode
Precondition	-
The called functions	-
Input parameter{in}	
-	-
Output parameter{out}	
-	-
Return value	
-	-

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 3-882. Function rtc_register_sync_wait

Function name	rtc_register_sync_wait
Function prototype	ErrStatus rtc_register_sync_wait(void);
Function descriptions	wait until RTC_TIME and RTC_DATE registers are synchronized with APB clock, and the shadow registers are updated
Precondition	-
The called functions	-
Input parameter{in}	
-	-
Output parameter{out}	
-	-
Return value	
ErrStatus	ERROR or SUCCESS

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 3-883. Function rtc_current_time_get

Function name	rtc_current_time_get
Function prototype	void rtc_current_time_get(rtc_parameter_struct* rtc_initpara_struct);
Function descriptions	get current time and date
Precondition	-
The called functions	-
Input parameter{in}	
-	-
Output parameter{out}	
rtc_initpara_struct	the structure members can refer to members of the structure Table 3-874. Struct rtc_parameter_struct
Return value	
-	-

Example:

```
/* 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 3-884. Function rtc_subsecond_get

Function name	rtc_subsecond_get
Function prototype	uint32_t rtc_subsecond_get(void);
Function descriptions	get current subsecond value
Precondition	-
The called functions	-
Input parameter{in}	
-	-
Output parameter{out}	
-	-
Return value	
uint32_t	current subsecond value(0x0000-0xFFFF)

Example:

```
/* 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 3-885. Function rtc_alarm_config

Function name	rtc_alarm_config
Function prototype	void rtc_alarm_config(uint8_t rtc_alarm, rtc_alarm_struct* rtc_alarm_time);
Function descriptions	configure RTC alarm
Precondition	-
The called functions	-
Input parameter{in}	
rtc_alarm	select alarm0 or alarm1
RTC_ALARM0	alarm0
RTC_ALARM1	alarm1
Input parameter{in}	
rtc_alarm_time	the structure members can refer to members of the structure Table 3-875. Struct rtc_alarm_struct
Output parameter{out}	
-	-
Return value	

-	-
---	---

Example:

```
/* configure RTC alarm0 */

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 3-886. Function rtc_alarm_subsecond_config

Function name	rtc_alarm_subsecond_config
Function prototype	void rtc_alarm_subsecond_config(uint8_t rtc_alarm, uint32_t mask_subsecond, uint32_t subsecond);
Function descriptions	configure subsecond of RTC alarm
Precondition	-
The called functions	-
Input parameter{in}	
rtc_alarm	select alarm0 or alarm1
<i>RTC_ALARM0</i>	alarm0
<i>RTC_ALARM1</i>	alarm1
Input parameter{in}	
mask_subsecond	alarm subsecond mask
<i>RTC_MASKSSC_0_14</i>	mask alarm subsecond configuration
<i>RTC_MASKSSC_1_14</i>	mask RTC_ALRM0SS_SSC[14:1], and RTC_ALRM0SS_SSC[0] is to be compared
<i>RTC_MASKSSC_2_14</i>	mask RTC_ALRM0SS_SSC[14:2], and RTC_ALRM0SS_SSC[1:0] is to be compared
<i>RTC_MASKSSC_3_14</i>	mask RTC_ALRM0SS_SSC[14:3], and RTC_ALRM0SS_SSC[2:0] is to be compared
<i>RTC_MASKSSC_4_14</i>	mask RTC_ALRM0SS_SSC[14:4], and RTC_ALRM0SS_SSC[3:0] is to be compared
<i>RTC_MASKSSC_5_14</i>	mask RTC_ALRM0SS_SSC[14:5], and RTC_ALRM0SS_SSC[4:0] is to be compared
<i>RTC_MASKSSC_6_14</i>	mask RTC_ALRM0SS_SSC[14:6], and RTC_ALRM0SS_SSC[5:0] is to be compared
<i>RTC_MASKSSC_7_14</i>	mask RTC_ALRM0SS_SSC[14:7], and RTC_ALRM0SS_SSC[6:0] is to be compared
<i>RTC_MASKSSC_8_14</i>	mask RTC_ALRM0SS_SSC[14:8], and RTC_ALRM0SS_SSC[7:0] is to be compared
<i>RTC_MASKSSC_9_14</i>	mask RTC_ALRM0SS_SSC[14:9], and RTC_ALRM0SS_SSC[8:0] is to be compared

<i>RTC_MASKSSC_10_1</i> 4	mask RTC_ALARM0SS_SSC[14:10], and RTC_ALARM0SS_SSC[9:0] is to be compared
<i>RTC_MASKSSC_11_1</i> 4	mask RTC_ALARM0SS_SSC[14:11], and RTC_ALARM0SS_SSC[10:0] is to be compared
<i>RTC_MASKSSC_12_1</i> 4	mask RTC_ALARM0SS_SSC[14:12], and RTC_ALARM0SS_SSC[11:0] is to be compared
<i>RTC_MASKSSC_13_1</i> 4	mask RTC_ALARM0SS_SSC[14:13], and RTC_ALARM0SS_SSC[12:0] is to be compared
<i>RTC_MASKSSC_14</i>	mask RTC_ALARM0SS_SSC[14], and RTC_ALARM0SS_SSC[13:0] is to be compared
<i>RTC_MASKSSC_NONE</i>	mask none, and RTC_ALARM0SS_SSC[14:0] is to be compared
Input parameter{in}	
subsecond	alarm subsecond value(0x0000 - 0x7FFF)
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* configure subsecond of RTC alarm0 */
```

```
rtc_alarm_subsecond_config(RTC_ALARM0, RTC_MASKSSC_9_14, 0x7FFF);
```

rtc_alarm_get

The description of rtc_alarm_get is shown as below:

Table 3-887. Function rtc_alarm_get

Function name	rtc_alarm_get
Function prototype	void rtc_alarm_get(uint8_t rtc_alarm, rtc_alarm_struct* rtc_alarm_time);
Function descriptions	get RTC alarm
Precondition	-
The called functions	-
Input parameter{in}	
rtc_alarm	select alarm0 or alarm1
<i>RTC_ALARM0</i>	alarm0
<i>RTC_ALARM1</i>	alarm1
Output parameter{out}	
rtc_alarm_time	the structure members can refer to members of the structure Table 3-875. Struct rtc_alarm_struct
Return value	
-	-

Example:

```

/* get RTC alarm0 */

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 3-888. Function rtc_alarm_subsecond_get

Function name	rtc_alarm_subsecond_get
Function prototype	uint32_t rtc_alarm_subsecond_get(uint8_t rtc_alarm);
Function descriptions	get RTC alarm subsecond
Precondition	-
The called functions	-
Input parameter{in}	
rtc_alarm	select alarm0 or alarm1
<i>RTC_ALARM0</i>	alarm0
<i>RTC_ALARM1</i>	alarm1
Output parameter{out}	
-	-
Return value	
uint32_t	RTC alarm subsecond value(0x0000-0x3FFF)

Example:

```

/* get RTC alarm0 subsecond */

uint32_t subsecond = rtc_alarm_subsecond_get(RTC_ALARM0);

```

rtc_alarm_enable

The description of rtc_alarm_enable is shown as below:

Table 3-889. Function rtc_alarm_enable

Function name	rtc_alarm_enable
Function prototype	void rtc_alarm_enable(uint8_t rtc_alarm);
Function descriptions	enable RTC alarm
Precondition	-
The called functions	-
Input parameter{in}	
rtc_alarm	select alarm0 or alarm1
<i>RTC_ALARM0</i>	alarm0
<i>RTC_ALARM1</i>	alarm1
Output parameter{out}	
-	-
Return value	

-	-
---	---

Example:

```
/* enable RTC alarm0 */
rtc_alarm_enable(RTC_ALARM0);
```

rtc_alarm_disable

The description of rtc_alarm_disable is shown as below:

Table 3-890. Function rtc_alarm_disable

Function name	rtc_alarm_disable
Function prototype	ErrStatus rtc_alarm_disable(uint8_t rtc_alarm);
Function descriptions	disable RTC alarm
Precondition	-
The called functions	-
Input parameter{in}	
rtc_alarm	select alarm0 or alarm1
<i>RTC_ALARM0</i>	alarm0
<i>RTC_ALARM1</i>	alarm1
Output parameter{out}	
-	-
Return value	
ErrStatus	ERROR or SUCCESS

Example:

```
/* disable RTC alarm0 */
ErrStatus error_status = rtc_alarm_disable(RTC_ALARM0);
```

rtc_timestamp_enable

The description of rtc_timestamp_enable is shown as below:

Table 3-891. Function rtc_timestamp_enable

Function name	rtc_timestamp_enable
Function prototype	void rtc_timestamp_enable(uint32_t edge);
Function descriptions	enable RTC time-stamp
Precondition	-
The called functions	-
Input parameter{in}	
edge	specify which edge to detect of time-stamp
<i>RTC_TIMESTAMP_RISING_EDGE</i>	rising edge is valid event edge for timestamp event
<i>RTC_TIMESTAMP_FALLING_EDGE</i>	falling edge is valid event edge for timestamp event

<i>LLING_EDGE</i>	
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 3-892. Function `rtc_timestamp_disable`

Function name	<code>rtc_timestamp_disable</code>
Function prototype	<code>void rtc_timestamp_disable(void);</code>
Function descriptions	disable RTC time-stamp
Precondition	-
The called functions	-
Input parameter{in}	
-	-
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* disable RTC time-stamp */
```

```
rtc_timestamp_disable();
```

rtc_timestamp_get

The description of `rtc_timestamp_get` is shown as below:

Table 3-893. Function `rtc_timestamp_get`

Function name	<code>rtc_timestamp_get</code>
Function prototype	<code>void rtc_timestamp_get(rtc_timestamp_struct* rtc_timestamp);</code>
Function descriptions	get RTC timestamp time and date
Precondition	-
The called functions	-
Input parameter{in}	
-	-
Output parameter{out}	

rtc_timestamp	the structure members can refer to members of the structure Table 3-877 . Struct 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 3-894. Function rtc_timestamp_subsecond_get

Function name	rtc_timestamp_subsecond_get
Function prototype	uint32_t rtc_timestamp_subsecond_get(void);
Function descriptions	get RTC time-stamp subsecond
Precondition	-
The called functions	-
Input parameter{in}	
-	-
Output parameter{out}	
-	-
Return value	
uint32_t	RTC time-stamp subsecond value

Example:

```
/* get RTC time-stamp subsecond */
uint32_t subsecond = rtc_timestamp_subsecond_get();
```

rtc_timestamp_pin_map

The description of rtc_timestamp_pin_map is shown as below:

Table 3-895. Function rtc_timestamp_pin_map

Function name	rtc_timestamp_pin_map
Function prototype	void rtc_timestamp_pin_map(uint32_t rtc_af);
Function descriptions	RTC time-stamp mapping
Precondition	-
The called functions	-
Input parameter{in}	
rtc_af	Timestamp input mapping selection
RTC_AF0_TIMESTAMP	RTC_AF0 use for timestamp

<i>P</i>	
<i>RTC_AF1_TIMESTAMP</i> <i>P</i>	RTC_AF1 use for timestamp
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* RTC time-stamp mapping */
```

```
rtc_timestamp_pin_map(RTC_AF0_TIMESTAMP);
```

rtc_tamper_enable

The description of `rtc_tamper_enable` is shown as below:

Table 3-896. Function `rtc_timestamp_enable`

Function name	rtc_tamper_enable
Function prototype	void rtc_tamper_enable(rtc_tamper_struct* rtc_tamper);
Function descriptions	enable RTC tamper
Precondition	-
The called functions	-
Input parameter{in}	
rtc_tamper	the structure members can refer to members of the structure Table 3-877. Struct <code>rtc_tamper_struct</code>
Output parameter{out}	
-	-
Return value	
-	-

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 3-897. Function `rtc_tamper_disable`

Function name	rtc_tamper_disable
Function prototype	void rtc_tamper_disable(uint32_t source);
Function descriptions	disable RTC tamper
Precondition	-

The called functions	-
Input parameter{in}	
source	specify which tamper source to be disabled
<i>RTC_TAMPER0</i>	RTC tamper0
<i>RTC_TAMPER1</i>	RTC tamper1
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* disable RTC tamper */
rtc_tamper_disable(RTC_TAMPER0);
```

rtc_tamper0_pin_map

The description of rtc_tamper0_pin_map is shown as below:

Table 3-898. Function rtc_tamper0_pin_map

Function name	rtc_tamper0_pin_map
Function prototype	void rtc_tamper0_pin_map (uint32_t rtc_af);
Function descriptions	RTC tamper0 mapping
Precondition	-
The called functions	-
Input parameter{in}	
rtc_af	Tamper0 fuction input mapping selection
<i>RTC_AF0_TAMPER0</i>	RTC_AF0 use for tamper0
<i>RTC_AF1_TAMPER0</i>	RTC_AF1 use for tamper0
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* RTC tamper0 mapping */
rtc_tamper0_pin_map(RTC_AF0_TAMPER0);
```

rtc_interrupt_enable

The description of rtc_interrupt_enable is shown as below:

Table 3-899. Function rtc_interrupt_enable

Function name	rtc_interrupt_enable
Function prototype	void rtc_interrupt_enable(uint32_t interrupt);

Function descriptions	enable specified RTC interrupt
Precondition	-
The called functions	-
Input parameter{in}	
interrupt	specify which interrupt source to be enabled
<i>RTC_INT_TIMESTAMP</i>	timestamp interrupt
<i>RTC_INT_ALARM0</i>	Alarm0 interrupt
<i>RTC_INT_ALARM1</i>	Alarm1 interrupt
<i>RTC_INT_TAMP</i>	tamp interrupt
<i>RTC_INT_WAKEUP</i>	wakeup timer interrupt
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* enable specified RTC interrupt */
rtc_interrupt_enable(RTC_INT_TAMP);
```

rtc_interrupt_disable

The description of rtc_interrupt_disable is shown as below:

Table 3-900. Function rtc_interrupt_disable

Function name	rtc_interrupt_disable
Function prototype	void rtc_interrupt_disable(uint32_t interrupt);
Function descriptions	disble specified RTC interrupt
Precondition	-
The called functions	-
Input parameter{in}	
interrupt	specify which RTC interrupt to disable
<i>RTC_INT_TIMESTAMP</i>	second interrupt
<i>RTC_INT_ALARM0</i>	alarm0 interrupt
<i>RTC_INT_ALARM1</i>	alarm1 interrupt
<i>RTC_INT_TAMP</i>	tamp interrupt
<i>RTC_INT_WAKEUP</i>	wakeup timer interrupt
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* disable specified RTC interrupt */
```

```
rtc_interrupt_disable(RTC_INT_TAMP);
```

rtc_flag_get

The description of rtc_flag_get is shown as below:

Table 3-901. Function rtc_flag_get

Function name	rtc_flag_get
Function prototype	FlagStatus rtc_flag_get(uint32_t flag);
Function descriptions	check specified flag
Precondition	-
The called functions	-
Input parameter{in}	
flag	specify which flag to check
<i>RTC_STAT_SCP</i>	smooth calibration pending flag
<i>RTC_FLAG_TP1</i>	RTC tamper 1 detected flag
<i>RTC_FLAG_TP0</i>	RTC tamper 0 detected flag
<i>RTC_FLAG_TSOVR</i>	time-stamp overflow flag
<i>RTC_FLAG_TS</i>	time-stamp flag
<i>RTC_FLAG_ALARM0</i>	alarm0 occurs flag
<i>RTC_FLAG_ALARM1</i>	Alarm1 occurs flag
<i>RTC_FLAG_WT</i>	wakeup timer occurs flag
<i>RTC_FLAG_INIT</i>	initialization state flag
<i>RTC_FLAG_RSYN</i>	register synchronization flag
<i>RTC_FLAG_YCM</i>	year configuration mark status flag
<i>RTC_FLAG_SOP</i>	alarm0 configuration can be write flag
<i>RTC_FLAG_ALRM0W</i>	alarm0 writen available flag
<i>RTC_FLAG_ALRM1W</i>	alarm1 writen available flag
<i>RTC_FLAG_WTW</i>	wakeup timer can be write flag
Output parameter{out}	
-	-
Return value	
FlagStatus	SET or RESET

Example:

```
/* check time-stamp event flag */
```

```
FlagStatus = rtc_flag_get(RTC_FLAG_TIMESTAMP);
```

rtc_flag_clear

The description of rtc_flag_clear is shown as below:

Table 3-902. Function rtc_flag_clear

Function name	rtc_flag_clear
Function prototype	void rtc_flag_clear(uint32_t flag);

Function descriptions	clear specified flag
Precondition	-
The called functions	-
Input parameter{in}	
flag	specify which flag to clear
<i>RTC_FLAG_TP1</i>	RTC tamper 1 detected flag
<i>RTC_FLAG_TP0</i>	RTC tamper 0 detected flag
<i>RTC_FLAG_TSOVR</i>	time-stamp overflow flag
<i>RTC_FLAG_TS</i>	time-stamp flag
<i>RTC_FLAG_WT</i>	wakeup timer occurs flag
<i>RTC_FLAG_ALARM0</i>	alarm0 occurs flag
<i>RTC_FLAG_ALARM1</i>	alarm1 occurs flag
<i>RTC_FLAG_RSYN</i>	register synchronization flag
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* cleartime-stamp event flag */
```

```
rtc_flag_clear(RTC_FLAG_TIMESTAMP);
```

rtc_alter_output_config

The description of rtc_alter_output_config is shown as below:

Table 3-903. Function rtc_alter_output_config

Function name	rtc_alter_output_config
Function prototype	void rtc_alter_output_config(uint32_t source, uint32_t mode);
Function descriptions	configure rtc alternate output source
Precondition	-
The called functions	-
Input parameter{in}	
source	specify signal to output
<i>RTC_ALARM0_HIGH</i>	when the alarm0 flag is set, the output pin is high
<i>RTC_ALARM0_LOW</i>	when the alarm0 flag is set, the output pin is low
<i>RTC_ALARM1_HIGH</i>	when the alarm1 flag is set, the output pin is high
<i>RTC_ALARM1_LOW</i>	when the alarm1 flag is set, the output pin is low
<i>RTC_WAKEUP_HIGH</i>	when the wakeup flag is set, the output pin is high
<i>RTC_WAKEUP_LOW</i>	when the wakeup flag is set, the output pin is low
Input parameter{in}	
mode	specify the output pin (PC13) mode when output alarm signal
<i>RTC_ALARM_OUTPUT_OD</i>	open drain mode

RTC_ALARM_OUTPU T_PP	push pull mode
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* configure rtc alternate output source */
```

```
rtc_alter_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 3-904. Function rtc_calibration_output_config

Function name	rtc_calibration_output_config
Function prototype	void rtc_calibration_output_config(uint32_t source)
Function descriptions	configure rtc calibration output source
Precondition	-
The called functions	-
Input parameter{in}	
source	specify signal to output
RTC_CALIBRATION_5 12HZ	when the LSE frequency is 32768Hz and the RTC_PSC is the default value, output 512Hz signal
RTC_CALIBRATION_1 HZ	when the LSE frequency is 32768Hz and the RTC_PSC is the default value, output 1Hz signal
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* configure rtc calibration output source */
```

```
rtc_calibration_output_config(RTC_CALIBRATION_1HZ);
```

rtc_hour_adjust

The description of rtc_hour_adjust is shown as below:

Table 3-905. Function rtc_hour_adjust

Function name	rtc_hour_adjust
Function prototype	void rtc_hour_adjust(uint32_t operation);
Function descriptions	adjust the daylight saving time by adding or subtracting one hour from the

	current time
Precondition	-
The called functions	-
Input parameter{in}	
operation	hour adjustment operation
<i>RTC_CTL_A1H</i>	add one hour
<i>RTC_CTL_S1H</i>	subtract one hour
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* 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 3-906. Function rtc_second_adjust

Function name	rtc_second_adjust
Function prototype	ErrStatus rtc_second_adjust(uint32_t add, uint32_t minus);
Function descriptions	adjust RTC second or subsecond value of current time
Precondition	-
The called functions	-
Input parameter{in}	
add	add 1s to current time or not
<i>RTC_SHIFT_ADD1S_RESET</i>	no effect
<i>RTC_SHIFT_ADD1S_SET</i>	add 1s to current time
Input parameter{in}	
minus	number of subsecond to minus from current time(0x0 - 0x7FFF)
Output parameter{out}	
-	-
Return value	
-	-

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 3-907. Function rtc_bypass_shadow_enable

Function name	rtc_bypass_shadow_enable
Function prototype	void rtc_bypass_shadow_enable(void);
Function descriptions	enable RTC bypass shadow registers function
Precondition	-
The called functions	-
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 is shown as below:

Table 3-908. Function rtc_bypass_shadow_disable

Function name	rtc_bypass_shadow_disable
Function prototype	void rtc_bypass_shadow_disable (void);
Function descriptions	disable RTC bypass shadow registers function
Precondition	-
The called functions	-
Input parameter{in}	
-	-
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* disable RTC bypass shadow registers function */
rtc_bypass_shadow_disable();
```

rtc_refclock_detection_enable

The description of rtc_refclock_detection_enable is shown as below:

Table 3-909. Function rtc_refclock_detection_enable

Function name	rtc_refclock_detection_enable
Function prototype	ErrStatus rtc_refclock_detection_enable(void);
Function descriptions	enable RTC reference clock detection function
Precondition	-
The called functions	rtc_init_mode_enter / rtc_init_mode_exit
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 3-910. Function rtc_refclock_detection_disable

Function name	rtc_refclock_detection_disable
Function prototype	ErrStatus rtc_refclock_detection_disable(void);
Function descriptions	disable RTC reference clock detection function
Precondition	-
The called functions	rtc_init_mode_enter / rtc_init_mode_exit
Input parameter{in}	
-	-
Output parameter{out}	
-	-
Return value	
ErrStatus	ERROR or SUCCESS

Example:

```
/* disable RTC reference clock detection function */
ErrStatus error_status = rtc_refclock_detection_disable();
```

rtc_wakeup_enable

The description of rtc_wakeup_enable shown as below:

Table 3-911. Function rtc_wakeup_enable

Function name	rtc_wakeup_enable
----------------------	-------------------

Function prototype	void rtc_wakeup_enable(void);
Function descriptions	enable RTC auto wakeup function
Precondition	-
The called functions	-
Input parameter{in}	
-	-
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* enable RTC auto wakeup function */
rtc_wakeup_enable();
```

rtc_wakeup_disable

The description of rtc_wakeup_disable shown as below:

Table 3-912. Function rtc_wakeup_disable

Function name	rtc_wakeup_disable
Function prototype	ErrStatus rtc_wakeup_disable(void);
Function descriptions	disable RTC auto wakeup function
Precondition	-
The called functions	-
Input parameter{in}	
-	-
Output parameter{out}	
-	-
Return value	
ErrStatus	ERROR or SUCCESS

Example:

```
/* disable RTC auto wakeup function */
rtc_wakeup_disable();
```

rtc_wakeup_clock_set

The description of rtc_wakeup_clock_set shown as below:

Table 3-913. Function rtc_wakeup_clock_set

Function name	rtc_wakeup_clock_set
Function prototype	ErrStatus rtc_wakeup_clock_set(uint8_t wakeup_clock);
Function descriptions	set RTC auto wakeup timer clock

Precondition	-
The called functions	-
Input parameter{in}	
wakeup_clock	Auto-wakeup timert clock selection
WAKEUP_RTCK_DIV 16	RTC auto wakeup timer clock is RTC clock divided by 16
WAKEUP_RTCK_DIV 8	RTC auto wakeup timer clock is RTC clock divided by 8
WAKEUP_RTCK_DIV 4	RTC auto wakeup timer clock is RTC clock divided by 4
WAKEUP_RTCK_DIV 2	RTC auto wakeup timer clock is RTC clock divided by 2
WAKEUP_CKSPRE	RTC auto wakeup timer clock is ckspre
WAKEUP_CKSPRE_2 EXP16	RTC auto wakeup timer clock is ckspre and wakeup timer add 2exp16
Output parameter{out}	
-	-
Return value	
ErrStatus	ERROR or SUCCESS

Example:

```
/* set RTC auto wakeup timer clock */
```

```
ErrStatus error_status = rtc_wakeup_clock_set(WAKEUP_RTCK_DIV16);
```

rtc_wakeup_timer_set

The description of rtc_wakeup_timer_set shown as below:

Table 3-914. Function rtc_wakeup_timer_set

Function name	rtc_wakeup_timer_set
Function prototype	ErrStatus rtc_wakeup_timer_set(uint16_t wakeup_timer);
Function descriptions	set wakeup timer value
Precondition	-
The called functions	-
Input parameter{in}	
wakeup_timer	Auto_wakeup timer reloads value(0x0000-0xFFFF)
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_get shown as below:

Table 3-915. Function rtc_wakeup_timer_get

Function name	rtc_wakeup_timer_get
Function prototype	uint16_t rtc_wakeup_timer_get(void);
Function descriptions	get wakeup timer value
Precondition	-
The called functions	-
Input parameter{in}	
-	-
Output parameter{out}	
-	-
Return value	
uint16_t	0-0XFFFF

Example:

```
/* get wakeup timer value */
```

```
rtc_wakeup_timer_get();
```

rtc_smooth_calibration_config

The description of rtc_smooth_calibration_config shown as below:

Table 3-916. Function rtc_smooth_calibration_config

Function name	rtc_smooth_calibration_config
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}	
window	select calibration window
RTC_CALIBRATION_WINDOW_32S	2exp20 RTCCLK cycles, 32s if RTCCLK = 32768 Hz
RTC_CALIBRATION_WINDOW_16S	2exp19 RTCCLK cycles, 16s if RTCCLK = 32768 Hz
RTC_CALIBRATION_WINDOW_8S	2exp18 RTCCLK cycles, 8s if RTCCLK = 32768 Hz
Output parameter{out}	
-	-
Return value	

ErrStatus	ERROR or SUCCESS
------------------	------------------

Example:

```
/* configure RTC smooth calibration */
```

```
ErrStatus error_status;
```

```
error_status = rtc_smooth_calibration_config(RTC_CALIBRATION_WINDOW_32S,
RTC_CALIBRATION_PLUS_SET);
```

rtc_coarse_calibration_enable

The description of rtc_coarse_calibration_enable shown as below:

Table 3-917. Function rtc_coarse_calibration_enable

Function name	rtc_coarse_calibration_enable
Function prototype	ErrStatus rtc_coarse_calibration_enable(void);
Function descriptions	enable RTC coarse calibration
Precondition	-
The called functions	rtc_init_mode_enter / rtc_init_mode_exit
Input parameter{in}	
-	-
Output parameter{out}	
-	-
Return value	
ErrStatus	ERROR or SUCCESS

Example:

```
/* enable RTC coarse calibration */
```

```
rtc_coarse_calibration_enable();
```

rtc_coarse_calibration_disable

The description of rtc_coarse_calibration_disable shown as below:

Table 3-918. Function rtc_coarse_calibration_disable

Function name	rtc_coarse_calibration_disable
Function prototype	ErrStatus rtc_coarse_calibration_disable(void);
Function descriptions	disable RTC coarse calibration
Precondition	-
The called functions	rtc_init_mode_enter / rtc_init_mode_exit
Input parameter{in}	
-	-
Output parameter{out}	
-	-
Return value	

ErrStatus	ERROR or SUCCESS
------------------	------------------

Example:

```
/* rtc_coarse_calibration_disable */
```

```
ErrStatus error_status = rtc_coarse_calibration_disable();
```

rtc_coarse_calibration_config

The description of rtc_coarse_calibration_config shown as below:

Table 3-919. Function rtc_coarse_calibration_config

Function name	rtc_coarse_calibration_config
Function prototype	ErrStatus rtc_coarse_calibration_config(uint8_t direction, uint8_t step);
Function descriptions	config coarse calibration direction and step
Precondition	-
The called functions	rtc_init_mode_enter / rtc_init_mode_exit
Input parameter{in}	
direction	coarse calibration direction
<i>CALIB_INCREASE</i>	Increase calendar update frequency
<i>CALIB_DECREASE</i>	Decrease calendar update frequency
Input parameter{in}	
step	Coarse calibration direction
<i>0x00-0x1F</i>	COSD=0:
	0x00: +0PPM
	0x01: +4PPM(approximate value)
	0x02: +8PPM(approximate value)

	0x1F: +126PPM(approximate value)
	COSD=1:
	0x00: -0PPM
	0x01: -2PPM(approximate value)
	0x02: -4PPM(approximate value)

	0x1F: -63PPM(approximate value)
Output parameter{out}	
-	-
Return value	
ErrStatus	ERROR or SUCCESS

Example:

```
/* configure coarse calibration direction and step */
```

```
ErrStatus error_status = rtc_coarse_calibration_config(INCREASE, 0x01);
```

3.26. SDIO

The secure digital input/output interface (SDIO) defines the SD/SD I/O /MMC CE-ATA card host interface, which provides command/data transfer between the APB2 system bus and SD memory cards, SD I/O cards, Multimedia Card (MMC), and CE-ATA devices. The SDIO registers are listed in chapter [3.26.1](#), the SDIO firmware functions are introduced in chapter [3.26.2](#).

3.26.1. Descriptions of Peripheral registers

SDIO registers are listed in the table shown as below:

Table 3-920. SDIO Registers

Registers	Descriptions
SDIO_PWRCTL	Power control register
SDIO_CLKCTL	Clock control register
SDIO_CMDAGMT	Command argument register
SDIO_CMDCTL	Command control register
SDIO_RSPCMDIDX	Command index response register
SDIO_RESPx x=0..3	Response register
SDIO_DATATO	Data timeout register
SDIO_DATALEN	Data length register
SDIO_DATACTL	Data control register
SDIO_DATACNT	Data counter register
SDIO_STAT	Status register
SDIO_INTC	Interrupt clear register
SDIO_INTEN	Interrupt enable register
SDIO_FIFOCNT	FIFO counter register
SDIO_FIFO	FIFO data register

3.26.2. Descriptions of Peripheral functions

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

Table 3-921. SDIO firmware function

Function name	Function description
sdio_deinit	deinitialize the SDIO
sdio_clock_config	configure the SDIO clock
sdio_hardware_clock_enable	enable hardware clock control
sdio_hardware_clock_disable	disable hardware clock control
sdio_bus_mode_set	set different SDIO card bus mode
sdio_power_state_set	set the SDIO power state
sdio_power_state_get	get the SDIO power state

Function name	Function description
sdio_clock_enable	enable SDIO_CLK clock output
sdio_clock_disable	disable SDIO_CLK clock output
sdio_command_response_config	configure the command and response
sdio_wait_type_set	set the command state machine wait type
sdio_csm_enable	enable the CSM(command state machine)
sdio_csm_disable	disable the CSM(command state machine)
sdio_command_index_get	get the last response command index
sdio_response_get	get the response for the last received command
sdio_data_config	configure the data timeout, data length and data block size
sdio_data_transfer_config	configure the data transfer mode and direction
sdio_dsm_enable	enable the DSM(data state machine) for data transfer
sdio_dsm_disable	disable the DSM(data state machine)
sdio_data_write	write data(one word) to the transmit FIFO
sdio_data_read	read data(one word) from the receive FIFO
sdio_data_counter_get	get the number of remaining data bytes to be transferred to card
sdio_fifo_counter_get	get the number of words remaining to be written or read from FIFO
sdio_dma_enable	enable the DMA request for SDIO
sdio_dma_disable	disable the DMA request for SDIO
sdio_flag_get	get the flags state of SDIO
sdio_flag_clear	clear the pending flags of SDIO
sdio_interrupt_enable	enable the SDIO interrupt
sdio_interrupt_disable	disable the SDIO interrupt
sdio_interrupt_flag_get	get the interrupt flags state of SDIO
sdio_interrupt_flag_clear	clear the interrupt pending flags of SDIO
sdio_readwait_enable	enable the read wait mode(SD I/O only)
sdio_readwait_disable	disable the read wait mode(SD I/O only)
sdio_stop_readwait_enable	enable the function that stop the read wait process(SD I/O only)
sdio_stop_readwait_disable	disable the function that stop the read wait process(SD I/O only)
sdio_readwait_type_set	set the read wait type(SD I/O only)
sdio_operation_enable	enable the SD I/O mode specific operation(SD I/O only)
sdio_operation_disable	disable the SD I/O mode specific operation(SD I/O only)
sdio_suspend_enable	enable the SD I/O suspend operation(SD I/O only)
sdio_suspend_disable	disable the SD I/O suspend operation(SD I/O only)
sdio_ceata_command_enable	enable the CE-ATA command(CE-ATA only)
sdio_ceata_command_disable	disable the CE-ATA command(CE-ATA only)
sdio_ceata_interrupt_enable	enable the CE-ATA interrupt(CE-ATA only)
sdio_ceata_interrupt_disable	disable the CE-ATA interrupt(CE-ATA only)
sdio_ceata_command_completion_en	enable the CE-ATA command completion signal(CE-ATA only)

Function name	Function description
able	only)
sdio_ceata_command_completion_disable	disable the CE-ATA command completion signal(CE-ATA only)

sdio_deinit

The description of sdio_deinit is shown as below:

Table 3-922. Function sdio_deinit

Function name	sdio_deinit
Function prototype	void sdio_deinit(void);
Function descriptions	deinitialize the SDIO
Precondition	-
The called functions	-
Input parameter{in}	
-	-
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* deinitialize the SDIO */
sdio_deinit();
```

sdio_clock_config

The description of sdio_clock_config is shown as below:

Table 3-923. Function sdio_clock_config

Function name	sdio_clock_config
Function prototype	void sdio_clock_config(uint32_t clock_edge, uint32_t clock_bypass, uint32_t clock_powersave, uint16_t clock_division);
Function descriptions	configure the SDIO clock
Precondition	-
The called functions	-
Input parameter{in}	
clock_edge	SDIO_CLK clock edge
SDIO_SDIOLCKEDGE_RISING	select the rising edge of the SDIOCLK to generate SDIO_CLK
SDIO_SDIOLCKEDGE_FALLING	select the falling edge of the SDIOCLK to generate SDIO_CLK
Input parameter{in}	
clock_bypass	clock bypass

SDIO_CLOCKBYPASS _ENABLE	clock bypass
SDIO_CLOCKBYPASS _DISABLE	no bypass
Input parameter{in}	
clock_powersave	SDIO_CLK clock dynamic switch on/off for power saving
SDIO_CLOCKPW RSA VE_ENABLE	SDIO_CLK closed when bus is idle
SDIO_CLOCKPW RSA VE_DISABLE	SDIO_CLK clock is always on
Input parameter{in}	
clock_division	clock division, less than 512
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* configure the SDIO clock */
```

```
sdio_clock_config(SDIO_SDI OCLKEDGE_RISING, SDIO_CLOCKBYPASS_DISABLE,  
SDIO_CLOCKPW RSAVE_DISABLE, SD_CLK_DIV_TRANS);
```

sdio_hardware_clock_enable

The description of sdio_hardware_clock_enable is shown as below:

Table 3-924. Function sdio_hardware_clock_enable

Function name	sdio_hardware_clock_enable
Function prototype	void sdio_hardware_clock_enable(void);
Function descriptions	enable hardware clock control
Precondition	-
The called functions	-
Input parameter{in}	
-	-
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* enable hardware clock control */
```

```
sdio_hardware_clock_enable();
```

sdio_hardware_clock_disable

The description of sdio_hardware_clock_disable is shown as below:

Table 3-925. Function sdio_hardware_clock_disable

Function name	sdio_hardware_clock_disable
Function prototype	void sdio_hardware_clock_disable(void);
Function descriptions	disable hardware clock control
Precondition	-
The called functions	-
Input parameter{in}	
-	-
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* disable hardware clock control */
sdio_hardware_clock_disable();
```

sdio_bus_mode_set

The description of sdio_bus_mode_set is shown as below:

Table 3-926. Function sdio_bus_mode_set

Function name	sdio_bus_mode_set
Function prototype	void sdio_bus_mode_set(uint32_t bus_mode);
Function descriptions	set different SDIO card bus mode
Precondition	-
The called functions	-
Input parameter{in}	
bus_mode	SDIO card bus mode
<i>SDIO_BUSMODE_1BIT</i>	1-bit SDIO card bus mode
<i>SDIO_BUSMODE_4BIT</i>	4-bit SDIO card bus mode
<i>SDIO_BUSMODE_8BIT</i>	8-bit SDIO card bus mode
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* set SDIO bus mode */
```

```
sdio_bus_mode_set(SDIO_BUSMODE_1BIT);
```

sdio_power_state_set

The description of sdio_power_state_set is shown as below:

Table 3-927. Function sdio_power_state_set

Function name	sdio_power_state_set
Function prototype	void sdio_power_state_set(uint32_t power_state);
Function descriptions	set the SDIO power state
Precondition	-
The called functions	-
Input parameter{in}	
power_state	SDIO power state
SDIO_POWER_ON	SDIO power on
SDIO_POWER_OFF	SDIO power off
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* set SDIO power state */
```

```
sdio_power_state_set(SDIO_POWER_ON);
```

sdio_power_state_get

The description of sdio_power_state_get is shown as below:

Table 3-928. Function sdio_power_state_get

Function name	sdio_power_state_get
Function prototype	uint32_t sdio_power_state_get(void);
Function descriptions	get the SDIO power state
Precondition	-
The called functions	-
Input parameter{in}	
-	-
Output parameter{out}	
-	-
Return value	
uint32_t	SDIO_POWER_ON / SDIO_POWER_OFF

Example:

```

/* get the SDIO power state */

uint32_t sdio_power_value;

sdio_power_value = sdio_power_state_get();

```

sdio_clock_enable

The description of sdio_clock_enable is shown as below:

Table 3-929. Function sdio_clock_enable

Function name	sdio_clock_enable
Function prototype	void sdio_clock_enable(void);
Function descriptions	enable SDIO_CLK clock output
Precondition	-
The called functions	-
Input parameter{in}	
-	-
Output parameter{out}	
-	-
Return value	
-	-

Example:

```

/* enable SDIO_CLK clock output */

sdio_clock_enable();

```

sdio_clock_disable

The description of sdio_clock_disable is shown as below:

Table 3-930. Function sdio_clock_disable

Function name	sdio_clock_disable
Function prototype	void sdio_clock_disable(void);
Function descriptions	disable SDIO_CLK clock output
Precondition	-
The called functions	-
Input parameter{in}	
-	-
Output parameter{out}	
-	-
Return value	
-	-

Example:

```

/* disable SDIO_CLK clock output */

```

```
sdio_clock_disable();
```

sdio_command_response_config

The description of sdio_command_response_config is shown as below:

Table 3-931. Function sdio_command_response_config

Function name	sdio_command_response_config
Function prototype	void sdio_command_response_config(uint32_t cmd_index, uint32_t cmd_argument, uint32_t response_type);
Function descriptions	configure the command and response
Precondition	-
The called functions	-
Input parameter{in}	
cmd_index	command index, refer to the related specifications
Input parameter{in}	
cmd_argument	command argument, refer to the related specifications
Input parameter{in}	
response_type	response type
SDIO_RESPONSETYPE_NO	no response
SDIO_RESPONSETYPE_SHORT	short response
SDIO_RESPONSETYPE_LONG	long response
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* CMD2(SD_CMD_ALL_SEND_CID) command response config*/
```

```
sdio_command_response_config(SD_CMD_ALL_SEND_CID, (uint32_t)0x0, SDIO_RESPONSETYPE_LONG);
```

sdio_wait_type_set

The description of sdio_wait_type_set is shown as below:

Table 3-932. Function sdio_wait_type_set

Function name	sdio_wait_type_set
Function prototype	void sdio_wait_type_set(uint32_t wait_type);
Function descriptions	set the command state machine wait type
Precondition	-
The called functions	-

Input parameter{in}	
wait_type	wait type
<i>SDIO_WAITTYPE_NO</i>	not wait interrupt
<i>SDIO_WAITTYPE_INT ERRUPT</i>	wait interrupt
<i>SDIO_WAITTYPE_DATA AEND</i>	wait the end of data transfer
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* set the command state machine wait type */
sdio_wait_type_set(SDIO_WAITTYPE_NO);
```

sdio_csm_enable

The description of sdio_csm_enable is shown as below:

Table 3-933. Function sdio_csm_enable

Function name	sdio_csm_enable
Function prototype	void sdio_csm_enable(void);
Function descriptions	enable the CSM(command state machine)
Precondition	-
The called functions	-
Input parameter{in}	
-	-
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* enable the CSM(command state machine) */
sdio_csm_enable();
```

sdio_csm_disable

The description of sdio_csm_disable is shown as below:

Table 3-934. Function sdio_csm_disable

Function name	sdio_csm_disable
Function prototype	void sdio_csm_disable(void);

Function descriptions	disable the CSM(command state machine)
Precondition	-
The called functions	-
Input parameter{in}	
-	-
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* disable the CSM(command state machine) */
```

```
sdio_csm_disable();
```

sdio_command_index_get

The description of sdio_command_index_get is shown as below:

Table 3-935. Function sdio_command_index_get

Function name	sdio_command_index_get
Function prototype	uint8_t sdio_command_index_get(void);
Function descriptions	get the last response command index
Precondition	-
The called functions	-
Input parameter{in}	
-	-
Output parameter{out}	
-	-
Return value	
uint8_t	last response command index

Example:

```
/* get SDIO command index */
```

```
uint8_t sdio_commond_value;
```

```
sdio_commond_value = sdio_command_index_get();
```

sdio_response_get

The description of sdio_response_get is shown as below:

Table 3-936. Function sdio_response_get

Function name	sdio_response_get
Function prototype	uint32_t sdio_response_get(uint32_t responsex);
Function descriptions	get the response for the last received command

Precondition	-
The called functions	-
Input parameter{in}	
responsex	SDIO response
<i>SDIO_RESPONSE0</i>	card response[31:0]/card response[127:96]
<i>SDIO_RESPONSE1</i>	card response[95:64]
<i>SDIO_RESPONSE2</i>	card response[63:32]
<i>SDIO_RESPONSE3</i>	card response[31:1], plus bit 0
Output parameter{out}	
-	-
Return value	
uint32_t	response for the last received command

Example:

```
/* store the CID0 numbers */
```

```
uint32_t sdio_cid[0];
```

```
sdio_cid[0] = sdio_response_get(SDIO_RESPONSE0);
```

sdio_data_config

The description of sdio_data_config is shown as below:

Table 3-937. Function sdio_data_config

Function name	sdio_data_config
Function prototype	void sdio_data_config(uint32_t data_timeout, uint32_t data_length, uint32_t data_blocksize);
Function descriptions	configure the data timeout, data length and data block size
Precondition	-
The called functions	-
Input parameter{in}	
data_timeout	data timeout period in card bus clock periods
Input parameter{in}	
data_length	number of data bytes to be transferred
Input parameter{in}	
data_blocksize	size of data block for block transfer
<i>SDIO_DATABLOCKSIZE_1BYTE</i>	block size = 1 byte
<i>SDIO_DATABLOCKSIZE_2BYTES</i>	block size = 2 bytes
<i>SDIO_DATABLOCKSIZE_4BYTES</i>	block size = 4 bytes
<i>SDIO_DATABLOCKSIZE_8BYTES</i>	block size = 8 bytes

<i>SDIO_DATABLOCKSIZE</i> <i>E_16BYTES</i>	block size = 16 bytes
<i>SDIO_DATABLOCKSIZE</i> <i>E_32BYTES</i>	block size = 32 bytes
<i>SDIO_DATABLOCKSIZE</i> <i>E_64BYTES</i>	block size = 64 bytes
<i>SDIO_DATABLOCKSIZE</i> <i>E_128BYTES</i>	block size = 128 bytes
<i>SDIO_DATABLOCKSIZE</i> <i>E_256BYTES</i>	block size = 256 bytes
<i>SDIO_DATABLOCKSIZE</i> <i>E_512BYTES</i>	block size = 512 bytes
<i>SDIO_DATABLOCKSIZE</i> <i>E_1024BYTES</i>	block size = 1024 bytes
<i>SDIO_DATABLOCKSIZE</i> <i>E_2048BYTES</i>	block size = 2048 bytes
<i>SDIO_DATABLOCKSIZE</i> <i>E_4096BYTES</i>	block size = 4096 bytes
<i>SDIO_DATABLOCKSIZE</i> <i>E_8192BYTES</i>	block size = 8192 bytes
<i>SDIO_DATABLOCKSIZE</i> <i>E_16384BYTES</i>	block size = 16384 bytes
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* configure SDIO data */
```

```
sdio_data_config(0, 0, SDIO_DATABLOCKSIZE_1BYTE);
```

sdio_data_transfer_config

The description of sdio_data_transfer_config is shown as below:

Table 3-938. Function sdio_data_transfer_config

Function name	sdio_data_transfer_config
Function prototype	void sdio_data_transfer_config(uint32_t transfer_mode, uint32_t transfer_direction);
Function descriptions	configure the data transfer mode and direction
Precondition	-
The called functions	-
Input parameter{in}	
transfer_mode	mode of data transfer

<i>SDIO_TRANSMODE_BLOCK</i>	block transfer
<i>SDIO_TRANSMODE_STREAM</i>	stream transfer or SDIO multibyte transfer
Input parameter{in}	
transfer_direction	data transfer direction, read or write
<i>SDIO_TRANSDIRECTI ON_TOCARD</i>	write data to card
<i>SDIO_TRANSDIRECTI ON_TOSDIO</i>	read data from card
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* configure SDIO data transmisson */

sdio_data_transfer_config(SDIO_TRANSDIRECTI  
ON_TOSDIO,  
SDIO_TRANSMODE_BLOCK);
```

sdio_dsm_enable

The description of sdio_dsm_enable is shown as below:

Table 3-939. Function sdio_dsm_enable

Function name	sdio_dsm_enable
Function prototype	void sdio_dsm_enable(void);
Function descriptions	enable the DSM(data state machine) for data transfer
Precondition	-
The called functions	-
Input parameter{in}	
-	-
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* enable the DSM(data state machine) */

sdio_dsm_enable();
```

sdio_dsm_disable

The description of sdio_dsm_disable is shown as below:

Table 3-940. Function sdio_dsm_disable

Function name	sdio_dsm_disable
Function prototype	void sdio_dsm_disable(void);
Function descriptions	disable the DSM(data state machine)
Precondition	-
The called functions	-
Input parameter{in}	
-	-
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* disable the DSM(data state machine) */
sdio_dsm_disable();
```

sdio_data_write

The description of sdio_data_write is shown as below:

Table 3-941. Function sdio_data_write

Function name	sdio_data_write
Function prototype	void sdio_data_write(uint32_t data);
Function descriptions	write data(one word) to the transmit FIFO
Precondition	-
The called functions	-
Input parameter{in}	
data	32-bit data write to card
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* write data(one word) to the transmit FIFO */
sdio_data_write(0x0000 0001);
```

sdio_data_read

The description of sdio_data_read is shown as below:

Table 3-942. Function sdio_data_read

Function name	sdio_data_read
----------------------	----------------

Function prototype	uint32_t sdio_data_read(void);
Function descriptions	read data(one word) from the receive FIFO
Precondition	-
The called functions	-
Input parameter{in}	
-	-
Output parameter{out}	
-	-
Return value	
uint32_t	received data

Example:

```
/* read data(one word) from the receive FIFO */
sdio_data_read();
```

sdio_data_counter_get

The description of sdio_data_counter_get is shown as below:

Table 3-943. Function sdio_data_counter_get

Function name	sdio_data_counter_get
Function prototype	uint32_t sdio_data_counter_get(void);
Function descriptions	get the number of remaining data bytes to be transferred to card
Precondition	-
The called functions	-
Input parameter{in}	
-	-
Output parameter{out}	
-	-
Return value	
uint32_t	number of remaining data bytes to be transferred

Example:

```
/* get the number of remaining data bytes to be transferred to card */
uint32_t sdio_data_value;
sdio_data_value = sdio_data_counter_get();
```

sdio_fifo_counter_get

The description of sdio_fifo_counter_get is shown as below:

Table 3-944. Function sdio_data_counter_get

Function name	sdio_fifo_counter_get
Function prototype	uint32_t sdio_fifo_counter_get(void);

Function descriptions	get the number of words remaining to be written or read from FIFO
Precondition	-
The called functions	-
Input parameter{in}	
-	-
Output parameter{out}	
-	-
Return value	
uint32_t	remaining number of words

Example:

```
/* get the number of words remaining to be written or read from FIFO */
uint32_t sdio_fifo_value;
sdio_fifo_value = sdio_fifo_counter_get();
```

sdio_dma_enable

The description of sdio_dma_enable is shown as below:

Table 3-945. Function sdio_dma_enable

Function name	sdio_dma_enable
Function prototype	void sdio_dma_enable(void);
Function descriptions	enable the DMA request for SDIO
Precondition	-
The called functions	-
Input parameter{in}	
-	-
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* enable the SDIO DMA */
sdio_dma_enable();
```

sdio_dma_disable

The description of sdio_dma_disable is shown as below:

Table 3-946. Function sdio_dma_disable

Function name	sdio_dma_disable
Function prototype	void sdio_dma_disable(void);
Function descriptions	disable the DMA request for SDIO

Precondition	-
The called functions	-
Input parameter{in}	
-	-
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* disable the SDIO DMA */
```

```
sdio_dma_disable();
```

sdio_flag_get

The description of sdio_flag_get is shown as below:

Table 3-947. Function sdio_flag_get

Function name	sdio_flag_get
Function prototype	FlagStatus sdio_flag_get(uint32_t flag);
Function descriptions	get the flags state of SDIO
Precondition	-
The called functions	-
Input parameter{in}	
flag	flags state of SDIO
<i>SDIO_FLAG_CCR CER</i> <i>R</i>	command response received (CRC check failed) flag
<i>SDIO_FLAG_DTCRCE</i> <i>RR</i>	data block sent/received (CRC check failed) flag
<i>SDIO_FLAG_CMDTMO</i> <i>UT</i>	command response timeout flag
<i>SDIO_FLAG_DTTMOU</i> <i>T</i>	data timeout flag
<i>SDIO_FLAG_TXURE</i>	transmit FIFO underrun error occurs flag
<i>SDIO_FLAG_RXORE</i>	received FIFO overrun error occurs flag
<i>SDIO_FLAG_CMDREC</i> <i>V</i>	command response received (CRC check passed) flag
<i>SDIO_FLAG_CMDSEN</i> <i>D</i>	command sent (no response required) flag
<i>SDIO_FLAG_DTEND</i>	data end (data counter, SDIO_DATA_CNT, is zero) flag
<i>SDIO_FLAG_STBITE</i>	start bit error in the bus flag
<i>SDIO_FLAG_DTBLKE</i> <i>ND</i>	data block sent/received (CRC check passed) flag
<i>SDIO_FLAG_CMDRUN</i>	command transmission in progress flag

<i>SDIO_FLAG_TXRUN</i>	data transmission in progress flag
<i>SDIO_FLAG_RXRUN</i>	data reception in progress flag
<i>SDIO_FLAG_TFH</i>	transmit FIFO is half empty flag: at least 8 words can be written into the FIFO
<i>SDIO_FLAG_RFH</i>	receive FIFO is half full flag: at least 8 words can be read in the FIFO
<i>SDIO_FLAG_TFF</i>	transmit FIFO is full flag
<i>SDIO_FLAG_RFF</i>	receive FIFO is full flag
<i>SDIO_FLAG_TFE</i>	transmit FIFO is empty flag
<i>SDIO_FLAG_RFE</i>	receive FIFO is empty flag
<i>SDIO_FLAG_TXDTVAL</i>	data is valid in transmit FIFO flag
<i>SDIO_FLAG_RXDTVAL</i>	data is valid in receive FIFO flag
<i>SDIO_FLAG_SDIOINT</i>	SD I/O interrupt received flag
<i>SDIO_FLAG_ATAEND</i>	CE-ATA command completion signal received (only for CMD61) flag
Output parameter{out}	
-	-
Return value	
FlagStatus	SET or RESET

Example:

```
/* get the flags state of SDIO */
```

```
FlagStatus flag_value;
```

```
flag_value = sdio_flag_get(SDIO_FLAG_RFH);
```

sdio_flag_clear

The description of `sdio_flag_clear` is shown as below:

Table 3-948. Function `sdio_flag_clear`

Function name	<code>sdio_flag_clear</code>
Function prototype	<code>void sdio_flag_clear(uint32_t flag);</code>
Function descriptions	clear the pending flags of SDIO
Precondition	-
The called functions	-
Input parameter{in}	
flag	flags state of SDIO
<i>SDIO_FLAG_CCR CER</i> <i>R</i>	command response received (CRC check failed) flag
<i>SDIO_FLAG_DTCRCE</i> <i>RR</i>	data block sent/received (CRC check failed) flag
<i>SDIO_FLAG_CMDTMO</i> <i>UT</i>	command response timeout flag
<i>SDIO_FLAG_DTTMOU</i>	data timeout flag

<i>T</i>	
<i>SDIO_FLAG_TXURE</i>	transmit FIFO underrun error occurs flag
<i>SDIO_FLAG_RXORE</i>	received FIFO overrun error occurs flag
<i>SDIO_FLAG_CMDRECV</i>	command response received (CRC check passed) flag
<i>SDIO_FLAG_CMDSEND</i>	command sent (no response required) flag
<i>SDIO_FLAG_DTEND</i>	data end (data counter, <i>SDIO_DATACNT</i> , is zero) flag
<i>SDIO_FLAG_STBITE</i>	start bit error in the bus flag
<i>SDIO_FLAG_DTBLKE</i>	data block sent/received (CRC check passed) flag
<i>SDIO_FLAG_SDIOINT</i>	SD I/O interrupt received flag
<i>SDIO_FLAG_ATAEND</i>	CE-ATA command completion signal received (only for CMD61) flag
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* clear the pending flags of SDIO */
```

```
sdio_flag_clear(SDIO_FLAG_DTCRCERR);
```

sdio_interrupt_enable

The description of *sdio_interrupt_enable* is shown as below:

Table 3-949. Function *sdio_interrupt_enable*

Function name	<i>sdio_interrupt_enable</i>
Function prototype	<code>void sdio_interrupt_enable(uint32_t int_flag);</code>
Function descriptions	enable the SDIO interrupt
Precondition	-
The called functions	-
Input parameter{in}	
int_flag	interrupt flags state of SDIO
<i>SDIO_INT_CCRERR</i>	SDIO CCRERR interrupt
<i>SDIO_INT_DTCRCERR</i>	SDIO DTCRCERR interrupt
<i>SDIO_INT_CMDTMOUT</i>	SDIO CMDTMOUT interrupt
<i>SDIO_INT_DTTMOUT</i>	SDIO DTTMOUT interrupt
<i>SDIO_INT_TXURE</i>	SDIO TXURE interrupt
<i>SDIO_INT_RXORE</i>	SDIO_INT_RXORE
<i>SDIO_INT_CMDRECV</i>	SDIO CMDRECV interrupt
<i>SDIO_INT_CMDSEND</i>	SDIO CMDSEND interrupt

<i>SDIO_INT_DTEND</i>	SDIO DTEND interrupt
<i>SDIO_INT_STBITE</i>	SDIO STBITE interrupt
<i>SDIO_INT_DTBLKEND</i>	SDIO DTBLKEND interrupt
<i>SDIO_INT_CMDRUN</i>	SDIO CMDRUN interrupt
<i>SDIO_INT_TXRUN</i>	SDIO TXRUN interrupt
<i>SDIO_INT_RXRUN</i>	SDIO RXRUN interrupt
<i>SDIO_INT_TFH</i>	SDIO TFH interrupt
<i>SDIO_INT_RFH</i>	SDIO RFH interrupt
<i>SDIO_INT_TFF</i>	SDIO TFF interrupt
<i>SDIO_INT_RFF</i>	SDIO RFF interrupt
<i>SDIO_INT_TFE</i>	SDIO TFE interrupt
<i>SDIO_INT_RFE</i>	SDIO RFE interrupt
<i>SDIO_INT_TXDTVAL</i>	SDIO TXDTVAL interrupt
<i>SDIO_INT_RXDTVAL</i>	SDIO RXDTVAL interrupt
<i>SDIO_INT_SDIOINT</i>	SDIO SDIOINT interrupt
<i>SDIO_INT_ATAEND</i>	SDIO ATAEND interrupt
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* enable the SDIO corresponding interrupts */
```

```
sdio_interrupt_enable(SDIO_INT_CCRRCERR | SDIO_INT_DTTMOUT | SDIO_INT_RXORE
| SDIO_INT_DTEND | SDIO_INT_STBITE);
```

sdio_interrupt_disable

The description of sdio_interrupt_disable is shown as below:

Table 3-950. Function sdio_interrupt_disable

Function name	sdio_interrupt_disable
Function prototype	void sdio_interrupt_disable(uint32_t int_flag);
Function descriptions	disable the SDIO interrupt
Precondition	-
The called functions	-
Input parameter{in}	
int_flag	interrupt flags state of SDIO
<i>SDIO_INT_CCRRCERR</i>	SDIO CCRRCERR interrupt
<i>SDIO_INT_DTCRCERR</i>	SDIO DTCRCERR interrupt
<i>SDIO_INT_CMDTMOUT</i>	SDIO CMDTMOUT interrupt
<i>SDIO_INT_DTTMOUT</i>	SDIO DTTMOUT interrupt

<i>SDIO_INT_TXURE</i>	SDIO TXURE interrupt
<i>SDIO_INT_RXORE</i>	SDIO_INT_RXORE
<i>SDIO_INT_CMDRECV</i>	SDIO CMDRECV interrupt
<i>SDIO_INT_CMDSEND</i>	SDIO CMDSEND interrupt
<i>SDIO_INT_DTEND</i>	SDIO DTEND interrupt
<i>SDIO_INT_STBITE</i>	SDIO STBITE interrupt
<i>SDIO_INT_DTBLKEND</i>	SDIO DTBLKEND interrupt
<i>SDIO_INT_CMDRUN</i>	SDIO CMDRUN interrupt
<i>SDIO_INT_TXRUN</i>	SDIO TXRUN interrupt
<i>SDIO_INT_RXRUN</i>	SDIO RXRUN interrupt
<i>SDIO_INT_TFH</i>	SDIO TFH interrupt
<i>SDIO_INT_RFH</i>	SDIO RFH interrupt
<i>SDIO_INT_TFF</i>	SDIO TFF interrupt
<i>SDIO_INT_RFF</i>	SDIO RFF interrupt
<i>SDIO_INT_TFE</i>	SDIO TFE interrupt
<i>SDIO_INT_RFE</i>	SDIO RFE interrupt
<i>SDIO_INT_TXDTVAL</i>	SDIO TXDTVAL interrupt
<i>SDIO_INT_RXDTVAL</i>	SDIO RXDTVAL interrupt
<i>SDIO_INT_SDIOINT</i>	SDIO SDIOINT interrupt
<i>SDIO_INT_ATAEND</i>	SDIO ATAEND interrupt
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* disable the SDIO interrupt */
```

```
sdio_interrupt_disable(SDIO_INT_DTCCRERR);
```

sdio_interrupt_flag_get

The description of sdio_interrupt_flag_get is shown as below:

Table 3-951. Function sdio_interrupt_flag_get

Function name	sdio_interrupt_flag_get
Function prototype	FlagStatus sdio_interrupt_flag_get(uint32_t int_flag);
Function descriptions	get the interrupt flags state of SDIO
Precondition	-
The called functions	-
Input parameter{in}	
int_flag	interrupt flags state of SDIO
<i>SDIO_INT_FLAG_CCRERR</i>	SDIO CCCRERR interrupt flag
<i>SDIO_INT_FLAG_DTC</i>	SDIO DTCRERR interrupt flag

<i>RCERR</i>	
<i>SDIO_INT_FLAG_CMD TMOUT</i>	SDIO CMDTMOUT interrupt flag
<i>SDIO_INT_FLAG_DTT MOUT</i>	SDIO DTTMOUT interrupt flag
<i>SDIO_INT_FLAG_TXU RE</i>	SDIO TXURE interrupt flag
<i>SDIO_INT_FLAG_RXO RE</i>	SDIO_INT_RXORE flag
<i>SDIO_INT_FLAG_CMD RECV</i>	SDIO CMDRECV interrupt flag
<i>SDIO_INT_FLAG_CMD SEND</i>	SDIO CMDSEND interrupt flag
<i>SDIO_INT_FLAG_DTE ND</i>	SDIO DTEND interrupt flag
<i>SDIO_INT_FLAG_STBI TE</i>	SDIO STBITE interrupt flag
<i>SDIO_INT_FLAG_DTB LKEND</i>	SDIO DTBLKEND interrupt flag
<i>SDIO_INT_FLAG_CMD RUN</i>	SDIO CMDRUN interrupt flag
<i>SDIO_INT_FLAG_TXR UN</i>	SDIO TXRUN interrupt flag
<i>SDIO_INT_FLAG_RXR UN</i>	SDIO RXRUN interrupt flag
<i>SDIO_INT_FLAG_TFH</i>	SDIO TFH interrupt flag
<i>SDIO_INT_FLAG_RFH</i>	SDIO RFH interrupt flag
<i>SDIO_INT_FLAG_TFF</i>	SDIO TFF interrupt flag
<i>SDIO_INT_FLAG_RFF</i>	SDIO RFF interrupt flag
<i>SDIO_INT_FLAG_TFE</i>	SDIO TFE interrupt flag
<i>SDIO_INT_FLAG_RFE</i>	SDIO RFE interrupt flag
<i>SDIO_INT_FLAG_TXD TVAL</i>	SDIO TXDTVAL interrupt flag
<i>SDIO_INT_FLAG_RXD TVAL</i>	SDIO RXDTVAL interrupt flag
<i>SDIO_INT_FLAG_SDI OINT</i>	SDIO SDIOINT interrupt flag
<i>SDIO_INT_FLAG_ATA END</i>	SDIO ATAEND interrupt flag
Output parameter{out}	
-	-
Return value	
FlagStatus	SET or RESET

Example:

```
/* get the interrupt flags state of SDIO */
```

```
FlagStatus flag_value;
```

```
flag_value = sdio_interrupt_flag_get(SDIO_INT_FLAG_DTEND);
```

sdio_interrupt_flag_clear

The description of sdio_interrupt_flag_clear is shown as below:

Table 3-952. Function sdio_interrupt_flag_clear

Function name	sdio_interrupt_flag_clear
Function prototype	void sdio_interrupt_flag_clear(uint32_t int_flag);
Function descriptions	clear the interrupt pending flags of SDIO
Precondition	-
The called functions	-
Input parameter{in}	
int_flag	interrupt flags state of SDIO
SDIO_INT_FLAG_CCR CERR	command response received (CRC check failed) flag
SDIO_INT_FLAG_DTC RCERR	data block sent/received (CRC check failed) flag
SDIO_INT_FLAG_CMD TMOUT	command response timeout flag
SDIO_INT_FLAG_DTT MOUT	data timeout flag
SDIO_INT_FLAG_TXU RE	transmit FIFO underrun error occurs flag
SDIO_INT_FLAG_RXO RE	received FIFO overrun error occurs flag
SDIO_INT_FLAG_CMD RECV	command response received (CRC check passed) flag
SDIO_INT_FLAG_CMD SEND	command sent (no response required) flag
SDIO_INT_FLAG_DTE ND	data end (data counter, SDIO_DATACNT, is zero) flag
SDIO_INT_FLAG_STBI TE	start bit error in the bus flag
SDIO_INT_FLAG_DTB LKEND	data block sent/received (CRC check passed) flag
SDIO_INT_FLAG_SDI OINT	SD I/O interrupt received flag
SDIO_INT_FLAG_ATA END	CE-ATA command completion signal received (only for CMD61) flag

Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* clear the interrupt pending flags of SDIO */
```

```
sdio_interrupt_flag_clear(SDIO_INT_FLAG_DTEND);
```

sdio_readwait_enable

The description of sdio_readwait_enable is shown as below:

Table 3-953. Function sdio_readwait_enable

Function name	sdio_readwait_enable
Function prototype	void sdio_readwait_enable(void);
Function descriptions	enable the read wait mode(SD I/O only)
Precondition	-
The called functions	-
Input parameter{in}	
-	-
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* enable the read wait mode(SD I/O only) */
```

```
sdio_readwait_enable();
```

sdio_readwait_disable

The description of sdio_readwait_disable is shown as below:

Table 3-954. Function sdio_readwait_disable

Function name	sdio_readwait_disable
Function prototype	void sdio_readwait_disable(void);
Function descriptions	disable the read wait mode(SD I/O only)
Precondition	-
The called functions	-
Input parameter{in}	
-	-
Output parameter{out}	
-	-

Return value	
-	-

Example:

```
/* disable the read wait mode(SD I/O only) */
```

```
sdio_readwait_disable();
```

sdio_stop_readwait_enable

The description of sdio_stop_readwait_enable is shown as below:

Table 3-955. Function sdio_stop_readwait_enable

Function name	sdio_stop_readwait_enable
Function prototype	void sdio_stop_readwait_enable(void);
Function descriptions	enable the function that stop the read wait process(SD I/O only)
Precondition	-
The called functions	-
Input parameter{in}	
-	-
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* enable the function that stop the read wait process(SD I/O only) */
```

```
sdio_stop_readwait_enable();
```

sdio_stop_readwait_disable

The description of sdio_stop_readwait_disable is shown as below:

Table 3-956. Function sdio_stop_readwait_disable

Function name	sdio_stop_readwait_disable
Function prototype	void sdio_stop_readwait_disable(void);
Function descriptions	disable the function that stop the read wait process(SD I/O only)
Precondition	-
The called functions	-
Input parameter{in}	
-	-
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* disable the function that stop the read wait process(SD I/O only) */
sdio_stop_readwait_disable();
```

sdio_readwait_type_set

The description of sdio_readwait_type_set is shown as below:

Table 3-957. Function sdio_readwait_type_set

Function name	sdio_readwait_type_set
Function prototype	void sdio_readwait_type_set(uint32_t readwait_type);
Function descriptions	set the read wait type(SD I/O only)
Precondition	-
The called functions	-
Input parameter{in}	
readwait_type	SD I/O read wait type
SDIO_READWAITTYPE_CLK	read wait control by stopping SDIO_CLK
SDIO_READWAITTYPE_DAT2	read wait control using SDIO_DAT[2]
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* set the read wait type(SD I/O only) */
sdio_readwait_type_set(uint32_t readwait_type);
```

sdio_operation_enable

The description of sdio_operation_enable is shown as below:

Table 3-958. Function sdio_operation_enable

Function name	sdio_operation_enable
Function prototype	void sdio_operation_enable(void);
Function descriptions	enable the SD I/O mode specific operation(SD I/O only)
Precondition	-
The called functions	-
Input parameter{in}	
-	-
Output parameter{out}	
-	-
Return value	

-	-
---	---

Example:

```
/* enable the SD I/O mode specific operation(SD I/O only) */
sdio_operation_enable();
```

sdio_operation_disable

The description of sdio_operation_disable is shown as below:

Table 3-959. Function sdio_operation_disable

Function name	sdio_operation_disable
Function prototype	void sdio_operation_disable(void);
Function descriptions	disable the SD I/O mode specific operation(SD I/O only)
Precondition	-
The called functions	-
Input parameter{in}	
-	-
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* disable the SD I/O mode specific operation(SD I/O only) */
void sdio_operation_disable();
```

sdio_suspend_enable

The description of sdio_suspend_enable is shown as below:

Table 3-960. Function sdio_suspend_enable

Function name	sdio_suspend_enable
Function prototype	void sdio_suspend_enable(void);
Function descriptions	enable the SD I/O suspend operation(SD I/O only)
Precondition	-
The called functions	-
Input parameter{in}	
-	-
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* enable the SD I/O suspend operation(SD I/O only) */
```

```
sdio_suspend_enable();
```

sdio_suspend_disable

The description of sdio_suspend_disable is shown as below:

Table 3-961. Function sdio_suspend_disable

Function name	sdio_suspend_disable
Function prototype	void sdio_suspend_disable(void);
Function descriptions	disable the SD I/O suspend operation(SD I/O only)
Precondition	-
The called functions	-
Input parameter{in}	
-	-
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* disable the SD I/O suspend operation(SD I/O only) */
```

```
sdio_suspend_disable();
```

sdio_ceata_command_enable

The description of sdio_ceata_command_enable is shown as below:

Table 3-962. Function sdio_ceata_command_enable

Function name	sdio_ceata_command_enable
Function prototype	void sdio_ceata_command_enable(void);
Function descriptions	enable the CE-ATA command(CE-ATA only)
Precondition	-
The called functions	-
Input parameter{in}	
-	-
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* enable the CE-ATA command(CE-ATA only) */
```

```
sdio_ceata_command_enable();
```

sdio_ceata_command_disable

The description of sdio_ceata_command_disable is shown as below:

Table 3-963. Function sdio_ceata_command_disable

Function name	sdio_ceata_command_disable
Function prototype	void sdio_ceata_command_disable(void);
Function descriptions	disable the CE-ATA command(CE-ATA only)
Precondition	-
The called functions	-
Input parameter{in}	
-	-
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* disable the CE-ATA command(CE-ATA only) */
sdio_ceata_command_disable();
```

sdio_ceata_interrupt_enable

The description of sdio_ceata_interrupt_enable is shown as below:

Table 3-964. Function sdio_ceata_interrupt_enable

Function name	sdio_ceata_interrupt_enable
Function prototype	void sdio_ceata_interrupt_enable(void);
Function descriptions	enable the CE-ATA interrupt(CE-ATA only)
Precondition	-
The called functions	-
Input parameter{in}	
-	-
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* enable the CE-ATA interrupt(CE-ATA only) */
sdio_ceata_interrupt_enable();
```

sdio_ceata_interrupt_disable

The description of sdio_ceata_interrupt_disable is shown as below:

Table 3-965. Function sdio_ceata_interrupt_disable

Function name	sdio_ceata_interrupt_disable
Function prototype	void sdio_ceata_interrupt_disable(void);
Function descriptions	disable the CE-ATA interrupt(CE-ATA only)
Precondition	-
The called functions	-
Input parameter{in}	
-	-
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* disable the CE-ATA interrupt(CE-ATA only) */
sdio_ceata_interrupt_disable();
```

sdio_ceata_command_completion_enable

The description of sdio_ceata_command_completion_enable is shown as below:

Table 3-966. Function sdio_ceata_command_completion_enable

Function name	sdio_ceata_command_completion_enable
Function prototype	void sdio_ceata_command_completion_enable(void);
Function descriptions	enable the CE-ATA command completion signal(CE-ATA only)
Precondition	-
The called functions	-
Input parameter{in}	
-	-
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* enable the CE-ATA command completion signal(CE-ATA only) */
sdio_ceata_command_completion_enable();
```

sdio_ceata_command_completion_disable

The description of sdio_ceata_command_completion_disable is shown as below:

Table 3-967. Function sdio_ceata_command_completion_disable

Function name	sdio_ceata_command_completion_disable
----------------------	---------------------------------------

Function prototype	void sdio_ceata_command_completion_disable(void);
Function descriptions	disable the CE-ATA command completion signal(CE-ATA only)
Precondition	-
The called functions	-
Input parameter{in}	
-	-
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* disable the CE-ATA command completion signal(CE-ATA only) */
sdio_ceata_command_completion_disable();
```

3.27. 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 [3.27.1](#), the SPI/I2S firmware functions are introduced in chapter [3.27.2](#).

3.27.1. Descriptions of Peripheral registers

SPI/I2S registers are listed in the table shown as below:

Table 3-968. SPI/I2S Registers

Registers	Descriptions
SPI_CTL0	SPI control register 0
SPI_CTL1	SPI control register 1
SPI_STAT	SPI status register
SPI_DATA	SPI data register
SPI_CRCPOLY	SPI CRC polynomial register
SPI_RCRC	SPI receive CRC register
SPI_TCRC	SPI transmit CRC register
SPI_I2SCTL	SPI/I2S control register
SPI_I2SPSC	SPI/I2S clock prescaler register
SPI_QCTL	Quad-SPI mode control register
I2S_ADD_CTL0	I2S_ADD control register 0
I2S_ADD_CTL1	I2S_ADD control register 1
I2S_ADD_STAT	I2S_ADD status register
I2S_ADD_DATA	I2S_ADD data register
I2S_ADD_CRCPOLY	I2S_ADD CRC polynomial register
I2S_ADD_RCRC	I2S_ADD receive CRC register

Registers	Descriptions
I2S_ADD_TCRC	I2S_ADD transmit CRC register
I2S_ADD_I2SCTL	I2S_ADD I2S control register
I2S_ADD_I2SPSC	I2S_ADD I2S clock prescaler register

3.27.2. Descriptions of Peripheral functions

SPI/I2S firmware functions are listed in the table shown as below:

Table 3-969. SPI/I2S firmware function

Function name	Function description
spi_i2s_deinit	reset SPI and I2S peripheral
spi_struct_para_init	initialize the parameters of SPI struct with the default values
spi_init	initialize SPI peripheral parameter
spi_enable	enable SPI
spi_disable	disable SPI
i2s_init	initialize I2S peripheral parameter
i2s_psc_config	configure I2S peripheral prescaler
i2s_enable	enable I2S
i2s_disable	disable I2S
spi_nss_output_enable	enable SPI NSS output function
spi_nss_output_disable	disable SPI NSS output function
spi_nss_internal_high	SPI NSS pin high level in software mode
spi_nss_internal_low	SPI NSS pin low level in software mode
spi_dma_enable	enable SPI DMA function
spi_dma_disable	disable SPI DMA function
spi_i2s_data_frame_format_config	configure SPI/I2S data frame format
spi_i2s_data_transmit	SPI transmit data
spi_i2s_data_receive	SPI receive data
spi_bidirectional_transfer_config	configure SPI bidirectional transfer direction
i2s_full_duplex_mode_config	configure i2s full duplex mode
spi_i2s_format_error_clear	clear TI mode format error flag status
spi_crc_polynomial_set	set SPI CRC polynomial
spi_crc_polynomial_get	get SPI CRC polynomial
spi_crc_on	turn on SPI CRC function
spi_crc_off	turn off SPI CRC function
spi_crc_next	SPI next data is CRC value
spi_crc_get	get SPI CRC send value or receive value
spi_crc_error_clear	clear SPI CRC error flag status
spi_ti_mode_enable	enable SPI TI mode
spi_ti_mode_disable	disable SPI TI mode
spi_quad_enable	enable quad wire SPI
spi_quad_disable	disable quad wire SPI

Function name	Function description
spi_quad_write_enable	enable quad wire SPI write
spi_quad_read_enable	enable quad wire SPI read
spi_quad_io23_output_enable	enable quad wire SPI_IO2 and SPI_IO3 pin output
spi_quad_io23_output_disable	disable quad wire SPI_IO2 and SPI_IO3 pin output
spi_i2s_flag_get	get SPI and I2S flag status
spi_i2s_interrupt_enable	enable SPI and I2S interrupt
spi_i2s_interrupt_disable	disable SPI and I2S interrupt
spi_i2s_interrupt_flag_get	get SPI and I2S interrupt status

Structure spi_parameter_struct

Table 3-970. spi_parameter_struct

Member name	Function description
device_mode	SPI master or slave (SPI_MASTER, SPI_SLAVE)
trans_mode	SPI transtype (SPI_TRANSMODE_FULLDUPLEX, SPI_TRANSMODE_RECEIVEONLY, SPI_TRANSMODE_BDRECEIVE, SPI_TRANSMODE_BDTRANSMIT)
frame_size	SPI frame size (SPI_FRAME_SIZE_16BIT, SPI_FRAME_SIZE_8BIT)
nss	SPI NSS control by hardware or software (SPI_NSS_SOFT, SPI_NSS_HARD)
endian	SPI big endian or little endian (SPI_ENDIAN_MSB, SPI_ENDIAN_LSB)
clock_polarity_phase	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)
prescale	SPI prescale factor (SPI_PSC_n (n=2,4,8,16,32,64,128,256))

spi_i2s_deinit

The description of spi_i2s_deinit is shown as below:

Table 3-971. Function spi_i2s_deinit

Function name	spi_i2s_deinit
Function prototype	void spi_i2s_deinit(uint32_t spi_periph);
Function descriptions	reset SPI and I2S peripheral
Precondition	-
The called functions	rcu_periph_reset_enable / rcu_periph_reset_disable
Input parameter{in}	
spi_periph	SPI/I2S peripheral
SPIx	x=0,1,2,3,4,5
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 3-972. Function spi_struct_para_init

Function name	spi_struct_para_init
Function prototype	void spi_struct_para_init(spi_parameter_struct* spi_struct);
Function descriptions	initialize the parameters of SPI struct with the default values
Precondition	-
The called functions	-
Input parameter{in}	
*spi_struct	a spi_parameter_struct address
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* initialize the parameters of SPI */
spi_parameter_struct spi_init_struct;
spi_struct_para_init(&spi_init_struct);
```

spi_init

The description of spi_init is shown as below:

Table 3-973. Function spi_init

Function name	spi_init
Function prototype	void spi_init(uint32_t spi_periph, spi_parameter_struct* spi_struct);
Function descriptions	initialize SPI peripheral parameter
Precondition	-
The called functions	-
Input parameter{in}	
spi_periph	SPI peripheral
SPIx	x=0,1,2,3,4,5
Input parameter{in}	

spi_struct	SPI parameter initialization struct, the structure members can refer to members of the structure Table 3-970. spi_parameter_struct
Output parameter{out}	
-	-
Return value	
-	-

Example:

```

/* initialize SPI0 */

spi_parameter_struct spi_init_struct;

spi_init_struct.trans_mode      = SPI_TRANSMODE_BDTRANSMIT;
spi_init_struct.device_mode     = SPI_MASTER;
spi_init_struct.frame_size      = SPI_FRAME_SIZE_8BIT;
spi_init_struct.clock_polarity_phase = SPI_CK_PL_HIGH_PH_2EDGE;
spi_init_struct.nss             = SPI_NSS_SOFT;
spi_init_struct.prescale        = SPI_PSC_8;
spi_init_struct.endian          = SPI_ENDIAN_MSB;

spi_init(SPI0, &spi_init_struct);

```

spi_enable

The description of spi_enable is shown as below:

Table 3-974. Function spi_enable

Function name	spi_enable
Function prototype	void spi_enable(uint32_t spi_periph);
Function descriptions	enable SPI
Precondition	-
The called functions	-
Input parameter{in}	
spi_periph	SPI peripheral
<i>SPIx</i>	x=0,1,2,3,4,5
Output parameter{out}	
-	-
Return value	
-	-

Example:

```

/* enable SPI0 */

spi_enable(SPI0);

```

spi_disable

The description of spi_disable is shown as below:

Table 3-975. Function spi_disable

Function name	spi_disable
Function prototype	void spi_disable(uint32_t spi_periph);
Function descriptions	disable SPI
Precondition	-
The called functions	-
Input parameter{in}	
spi_periph	SPI peripheral
<i>SPIx</i>	x=0,1,2,3,4,5
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* disable SPI0 */
spi_disable(SPI0);
```

i2s_init

The description of i2s_init is shown as below:

Table 3-976. Function i2s_init

Function name	i2s_init
Function prototype	void i2s_init(uint32_t spi_periph, uint32_t i2s_mode, uint32_t i2s_standard, uint32_t i2s_ckpl);
Function descriptions	initialize I2S peripheral parameter
Precondition	-
The called functions	-
Input parameter{in}	
spi_periph	I2S peripheral
<i>SPIx</i>	x=1,2
Input parameter{in}	
i2s_mode	I2S operation mode
<i>I2S_MODE_SLAVETX</i>	I2S slave transmit mode
<i>I2S_MODE_SLAVERX</i>	I2S slave receive mode
<i>I2S_MODE_MASTERTX</i>	I2S master transmit mode
<i>I2S_MODE_MASTERRX</i>	I2S master receive mode

Input parameter{in}	
i2s_standard	I2S standard
<i>I2S_STD_PHILLIPS</i>	I2S phillips standard
<i>I2S_STD_MSB</i>	I2S MSB standard
<i>I2S_STD_LSB</i>	I2S LSB standard
<i>I2S_STD_PCMSHORT</i>	I2S PCM short standard
<i>I2S_STD_PCMLONG</i>	I2S PCM long standard
Input parameter{in}	
i2s_ckpl	I2S idle state clock polarity
<i>I2S_CKPL_LOW</i>	I2S clock polarity low level
<i>I2S_CKPL_HIGH</i>	I2S clock polarity high level
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* initialize I2S1 */
```

```
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 3-977. Function i2s_psc_config

Function name	i2s_psc_config
Function prototype	void i2s_psc_config(uint32_t spi_periph, uint32_t i2s_audiosample, uint32_t i2s_frameformat, uint32_t i2s_mckout);
Function descriptions	configure I2S prescaler
Precondition	-
The called functions	rcu_i2s_clock_config/ rcu_osci_on/ rcu_osci_stab_wait
Input parameter{in}	
spi_periph	I2S peripheral
<i>SPIx</i>	x=1,2
Input parameter{in}	
i2s_audiosample	I2S audio sample rate
<i>I2S_AUDIOSAMPLE_8K</i>	audio sample rate is 8KHz
<i>I2S_AUDIOSAMPLE_11K</i>	audio sample rate is 11KHz
<i>I2S_AUDIOSAMPLE_16K</i>	audio sample rate is 16KHz
<i>I2S_AUDIOSAMPLE_22K</i>	audio sample rate is 22KHz

<i>I2S_AUDIOSAMPLE_3</i> 2K	audio sample rate is 32KHz
<i>I2S_AUDIOSAMPLE_4</i> 4K	audio sample rate is 44KHz
<i>I2S_AUDIOSAMPLE_4</i> 8K	audio sample rate is 48KHz
<i>I2S_AUDIOSAMPLE_9</i> 6K	audio sample rate is 96KHz
<i>I2S_AUDIOSAMPLE_1</i> 92K	audio sample rate is 192KHz
Input parameter{in}	
i2s_frameformat	I2S data length and channel length
<i>I2S_FRAMEFORMAT_</i> <i>DT16B_CH16B</i>	I2S data length is 16 bit and channel length is 16 bit
<i>I2S_FRAMEFORMAT_</i> <i>DT16B_CH32B</i>	I2S data length is 16 bit and channel length is 32 bit
<i>I2S_FRAMEFORMAT_</i> <i>DT24B_CH32B</i>	I2S data length is 24 bit and channel length is 32 bit
<i>I2S_FRAMEFORMAT_</i> <i>DT32B_CH32B</i>	I2S data length is 32 bit and channel length is 32 bit
Input parameter{in}	
i2s_mckout	I2S master clock output
<i>I2S_MCKOUT_ENABL</i> <i>E</i>	I2S master clock output enable
<i>I2S_MCKOUT_DISABL</i> <i>E</i>	I2S master clock output disable
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* 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 3-978. Function i2s_enable

Function name	i2s_enable
Function prototype	void i2s_enable(uint32_t spi_periph);
Function descriptions	enable I2S

Precondition	-
The called functions	-
Input parameter{in}	
spi_periph	I2S peripheral
<i>SPIx</i>	x=1,2
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* enable I2S1*/
i2s_enable(SPI1);
```

i2s_disable

The description of i2s_disable is shown as below:

Table 3-979. Function i2s_disable

Function name	i2s_disable
Function prototype	void i2s_disable(uint32_t spi_periph);
Function descriptions	disable I2S
Precondition	-
The called functions	-
Input parameter{in}	
spi_periph	I2S peripheral
<i>SPIx</i>	x=1,2
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* disable I2S1*/
i2s_disable(SPI1);
```

spi_nss_output_enable

The description of spi_nss_output_enable is shown as below:

Table 3-980. Function spi_nss_output_enable

Function name	spi_nss_output_enable
Function prototype	void spi_nss_output_enable(uint32_t spi_periph);
Function descriptions	enable SPI NSS output function

Precondition	-
The called functions	-
Input parameter{in}	
spi_periph	SPI peripheral
<i>SPIx</i>	x=0,1,2,3,4,5
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 3-981. Function spi_nss_output_disable

Function name	spi_nss_output_disable
Function prototype	void spi_nss_output_disable(uint32_t spi_periph);
Function descriptions	disable SPI NSS output function
Precondition	-
The called functions	-
Input parameter{in}	
spi_periph	SPI peripheral
<i>SPIx</i>	x=0,1,2,3,4,5
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 3-982. Function spi_nss_internal_high

Function name	spi_nss_internal_high
Function prototype	void spi_nss_internal_high(uint32_t spi_periph);
Function descriptions	SPI NSS pin high level in software mode

Precondition	-
The called functions	-
Input parameter{in}	
spi_periph	SPI peripheral
<i>SPIx</i>	x=0,1,2,3,4,5
Output parameter{out}	
-	-
Return value	
-	-

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 3-983. Function spi_nss_internal_low

Function name	spi_nss_internal_low
Function prototype	void spi_nss_internal_low(uint32_t spi_periph);
Function descriptions	SPI NSS pin low level in software mode
Precondition	-
The called functions	-
Input parameter{in}	
spi_periph	SPI peripheral
<i>SPIx</i>	x=0,1,2,3,4,5
Output parameter{out}	
-	-
Return value	
-	-

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 3-984. Function spi_dma_enable

Function name	spi_dma_enable
Function prototype	void spi_dma_enable(uint32_t spi_periph, uint8_t dma);
Function descriptions	enable SPI DMA function

Precondition	-
The called functions	-
Input parameter{in}	
spi_periph	SPI peripheral
<i>SPIx</i>	x=0,1,2,3,4,5
Input parameter{in}	
dma	SPI DMA mode
<i>SPI_DMA_TRANSMIT</i>	SPI transmit data use DMA
<i>SPI_DMA_RECEIVE</i>	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 3-985. Function spi_dma_disable

Function name	spi_dma_disable
Function prototype	void spi_dma_disable(uint32_t spi_periph, uint8_t dma);
Function descriptions	disable SPI DMA function
Precondition	-
The called functions	-
Input parameter{in}	
spi_periph	SPI peripheral
<i>SPIx</i>	x=0,1,2,3,4,5
Input parameter{in}	
dma	SPI DMA mode
<i>SPI_DMA_TRANSMIT</i>	SPI transmit data use DMA
<i>SPI_DMA_RECEIVE</i>	SPI receive data use DMA
Output parameter{out}	
-	-
Return value	
-	-

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:

Table 3-986. Function spi_i2s_data_frame_format_config

Function name	spi_i2s_data_frame_format_config
Function prototype	void spi_i2s_data_frame_format_config(uint32_t spi_periph, uint16_t frame_format);
Function descriptions	configure SPI/I2S data frame format
Precondition	-
The called functions	-
Input parameter{in}	
spi_periph	SPI peripheral
<i>SPIx</i>	x=0,1,2,3,4,5
Input parameter{in}	
frame_format	SPI frame size
<i>SPI_FRAME_SIZE_16BIT</i>	SPI frame size is 16 bits
<i>SPI_FRAME_SIZE_8BIT</i>	SPI frame size is 8 bits
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* configure SPI1/I2S1 data frame format size is 16 bits */
spi_i2s_data_frame_format_config(SPI1, SPI_FRAME_SIZE_16BIT);
```

spi_i2s_data_transmit

The description of spi_i2s_data_transmit is shown as below:

Table 3-987. 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	-
Input parameter{in}	
spi_periph	SPI peripheral
<i>SPIx</i>	x=0,1,2,3,4,5
Input parameter{in}	
data	16-bit data
Output parameter{out}	

-	-
Return value	
-	-

Example:

```
/* SPI0 transmit data */

uint16_t spi_send_array[] = {0x5050,0xA0A0};

spi_i2s_data_transmit(SPI0, spi0_send_array[0]);
```

spi_i2s_data_receive

The description of spi_i2s_data_receive is shown as below:

Table 3-988. Function spi_i2s_data_receive

Function name	spi_i2s_data_receive
Function prototype	uint16_t spi_i2s_data_receive(uint32_t spi_periph);
Function descriptions	SPI receive data
Precondition	-
The called functions	-
Input parameter{in}	
spi_periph	SPI peripheral
<i>SPIx</i>	x=0,1,2,3,4,5
Output parameter{out}	
-	-
Return value	
uint16_t	16-bit data

Example:

```
/* SPI0 receive data */

uint16_t spi0_receive_data;

spi0_receive_data = spi_i2s_data_receive(SPI0);
```

spi_bidirectional_transfer_config

The description of spi_bidirectional_transfer_config is shown as below:

Table 3-989. Function spi_bidirectional_transfer_config

Function name	spi_bidirectional_transfer_config
Function prototype	void spi_bidirectional_transfer_config(uint32_t spi_periph, uint32_t transfer_direction);
Function descriptions	configure SPI bidirectional transfer direction
Precondition	-
The called functions	-

Input parameter{in}	
spi_periph	SPI peripheral
<i>SPIx</i>	x=0,1,2,3,4,5
Input parameter{in}	
transfer_direction	SPI transfer direction
<i>SPI_BIDIRECTIONAL_TRANSMIT</i>	SPI work in transmit-only mode
<i>SPI_BIDIRECTIONAL_RECEIVE</i>	SPI work in receive-only mode
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* SPI0 works in transmit-only mode */
```

```
spi_bidirectional_transfer_config(SPI0, SPI_BIDIRECTIONAL_TRANSMIT);
```

i2s_full_duplex_mode_config

The description of i2s_full_duplex_mode_config is shown as below:

Table 3-990. Function i2s_full_duplex_mode_config

Function name	i2s_full_duplex_mode_config
Function prototype	void i2s_full_duplex_mode_config (uint32_t i2s_add_periph,uint32_t i2s_mode,uint32_t i2s_standard,uint32_t i2s_ckpl,uint32_t frameformat);
Function descriptions	configure i2s full duplex mode
Precondition	-
The called functions	-
Input parameter{in}	
i2s_add_periph	I2Sx_ADDperipheral
<i>I2Sx_ADD</i>	x=1,2
Input parameter{in}	
i2s_mode	i2s mode
<i>I2S_MODE_SLAVETX</i>	I2S slave transmit mode
<i>I2S_MODE_SLAVERX</i>	I2S slave receive mode
<i>I2S_MODE_MASTERTX</i>	I2S master transmit mode
<i>I2S_MODE_MASTERRX</i>	I2S master receive mode
Input parameter{in}	
i2s_standard	i2s standard
<i>I2S_STD_PHILLIPS</i>	I2S phillips standard
<i>I2S_STD_MSB</i>	I2S MSB standard

<i>I2S_STD_LSB</i>	I2S LSB standard
<i>I2S_STD_PCMSHORT</i>	I2S PCM short standard
<i>I2S_STD_PCMLONG</i>	I2S PCM long standard
Input parameter{in}	
<i>i2s_ckpl</i>	i2s idle state clock polarity
<i>I2S_CKPL_LOW</i>	The idle state of I2S_CK is low level
<i>I2S_CKPL_HIGH</i>	The idle state of I2S_CK is high level
Input parameter{in}	
<i>i2s_frameformat</i>	i2s data length and channel length
<i>I2S_FRAMEFORMAT_DT16B_CH16B</i>	I2S data length is 16 bit and channel length is 16 bit
<i>I2S_FRAMEFORMAT_DT16B_CH32B</i>	I2S data length is 16 bit and channel length is 32 bit
<i>I2S_FRAMEFORMAT_DT24B_CH32B</i>	I2S data length is 24 bit and channel length is 32 bit
<i>I2S_FRAMEFORMAT_DT32B_CH32B</i>	I2S data length is 32 bit and channel length is 32 bit
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* configure i2s full duplex mode */
```

```
i2s_full_duplex_mode_config(I2S1_ADD,I2S_MODE_SLAVETX,I2S_STD_LSB,I2S_CKPL_
LOW,I2S_FRAMEFORMAT_DT16B_CH16B);
```

spi_i2s_format_error_clear

The description of spi_i2s_format_error_clear is shown as below:

Table 3-991. Function spi_i2s_format_error_clear

Function name	spi_i2s_format_error_clear
Function prototype	void spi_i2s_format_error_clear(uint32_t spi_periph, uint32_t flag);
Function descriptions	clear TI mode format error flag status
Precondition	-
The called functions	-
Input parameter{in}	
spi_periph	SPI peripheral
<i>SPIx</i>	x=0,1,2,3,4,5
Input parameter{in}	
flag	SPI/I2S frame format error flag
<i>SPI_FLAG_FERR</i>	SPI TI mode frame format error
<i>I2S_FLAG_FERR</i>	I2S frame format error

Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* clear SPI0 TI mode format error flag */
spi_i2s_format_error_clear(SPI0, SPI_FLAG_FERR);
```

spi_crc_polynomial_set

The description of spi_crc_polynomial_set is shown as below:

Table 3-992. Function spi_crc_polynomial_set

Function name	spi_crc_polynomial_set
Function prototype	void spi_crc_polynomial_set(uint32_t spi_periph, uint16_t crc_poly);
Function descriptions	set SPI CRC polynomial
Precondition	-
The called functions	-
Input parameter{in}	
spi_periph	SPI peripheral
SPIx	x=0,1,2,3,4,5
Input parameter{in}	
crc_poly	CRC polynomial value
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* set SPI0 CRC polynomial */
uint16_t CRC_VALUE = 0x5050;
spi_crc_polynomial_set(SPI0,CRC_VALUE);
```

spi_crc_polynomial_get

The description of spi_crc_polynomial_get is shown as below:

Table 3-993. Function spi_crc_polynomial_get

Function name	spi_crc_polynomial_get
Function prototype	uint16_t spi_crc_polynomial_get(uint32_t spi_periph);
Function descriptions	get SPI CRC polynomial
Precondition	-
The called functions	-

Input parameter{in}	
spi_periph	SPI peripheral
<i>SPIx</i>	x=0,1,2,3,4,5
Output parameter{out}	
-	-
Return value	
uint16_t	16-bit CRC polynomial (0-0xFFFF)

Example:

```
/* get SPI0 CRC polynomial */
uint16_t crc_val;

crc_val = spi_crc_polynomial_get(SPI0);
```

spi_crc_on

The description of spi_crc_on is shown as below:

Table 3-994. Function spi_crc_on

Function name	spi_crc_on
Function prototype	void spi_crc_on(uint32_t spi_periph);
Function descriptions	turn on SPI CRC function
Precondition	-
The called functions	-
Input parameter{in}	
spi_periph	SPI peripheral
<i>SPIx</i>	x=0,1,2,3,4,5
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 3-995. Function spi_crc_off

Function name	spi_crc_off
Function prototype	void spi_crc_off(uint32_t spi_periph);
Function descriptions	turn off SPI CRC function
Precondition	-

The called functions	-
Input parameter{in}	
spi_periph	SPI peripheral
SPIx	x=0,1,2,3,4,5
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 3-996. Function spi_crc_next

Function name	spi_crc_next
Function prototype	void spi_crc_next(uint32_t spi_periph);
Function descriptions	SPI next data is CRC value
Precondition	-
The called functions	-
Input parameter{in}	
spi_periph	SPI peripheral
SPIx	x=0,1,2,3,4,5
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* SPI0 next data is CRC value */
```

```
spi_crc_next(SPI0);
```

spi_crc_get

The description of spi_crc_get is shown as below:

Table 3-997. Function spi_crc_get

Function name	spi_crc_get
Function prototype	uint16_t spi_crc_get(uint32_t spi_periph, uint8_t spi_crc);
Function descriptions	get SPI CRC send value or receive value
Precondition	-

The called functions	-
Input parameter{in}	
spi_periph	SPI peripheral
<i>SPIx</i>	x=0,1,2,3,4,5
Input parameter{in}	
spi_crc	SPI crc value
<i>SPI_CRC_TX</i>	get transmit crc value
<i>SPI_CRC_RX</i>	get receive crc value
Output parameter{out}	
-	-
Return value	
uint16_t	16-bit CRC value (0-0xFFFF)

Example:

```
/* get SPI0 CRC send value */

uint16_t crc_val;

crc_val = spi_crc_get(SPI0, SPI_CRC_TX);
```

spi_crc_error_clear

The description of spi_crc_error_clear is shown as below:

Table 3-998. Function spi_crc_error_clear

Function name	spi_crc_error_clear
Function prototype	void spi_crc_error_clear(uint32_t spi_periph);
Function descriptions	clear SPI CRC error flag status
Precondition	-
The called functions	-
Input parameter{in}	
spi_periph	SPI peripheral
<i>SPIx</i>	x=0,1,2,3,4,5
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* clear SPI0 CRC error flag status */

spi_crc_error_clear(SPI0);
```

spi_ti_mode_enable

The description of spi_ti_mode_enable is shown as below:

Table 3-999. Function spi_ti_mode_enable

Function name	spi_ti_mode_enable
Function prototype	void spi_ti_mode_enable(uint32_t spi_periph);
Function descriptions	enable SPI TI mode
Precondition	-
The called functions	-
Input parameter{in}	
spi_periph	SPI peripheral
SPIx	x=0,1,2,3,4,5
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* enable SPI0 TI mode */
spi_ti_mode_enable(SPI0);
```

spi_ti_mode_disable

The description of spi_ti_mode_disable is shown as below:

Table 3-1000. Function spi_ti_mode_disable

Function name	spi_ti_mode_disable
Function prototype	void spi_ti_mode_disable(uint32_t spi_periph);
Function descriptions	disable SPI TI mode
Precondition	-
The called functions	-
Input parameter{in}	
spi_periph	SPI peripheral
SPIx	x=0,1,2,3,4,5
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* disable SPI0 TI mode */
spi_ti_mode_disable(SPI0);
```

spi_quad_enable

The description of spi_quad_enable is shown as below:

Table 3-1001. Function spi_quad_enable

Function name	spi_quad_enable
Function prototype	void spi_quad_enable(uint32_t spi_periph);
Function descriptions	enable quad wire SPI
Precondition	-
The called functions	-
Input parameter{in}	
spi_periph	SPI peripheral
<i>SPIx</i>	x=5
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* enable SPI5 quad wire mode */
spi_quad_enable(SPI5);
```

spi_quad_disable

The description of spi_quad_disable is shown as below:

Table 3-1002. Function spi_quad_disable

Function name	spi_quad_disable
Function prototype	spi_quad_disable(uint32_t spi_periph);
Function descriptions	disable quad wire SPI
Precondition	-
The called functions	-
Input parameter{in}	
spi_periph	SPI peripheral
<i>SPIx</i>	x=5
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* disable SPI5 quad wire mode */
spi_quad_disable(SPI5);
```

spi_quad_write_enable

The description of spi_quad_write_enable is shown as below:

Table 3-1003. Function spi_quad_write_enable

Function name	spi_quad_write_enable
Function prototype	void spi_quad_write_enable(uint32_t spi_periph);
Function descriptions	enable quad wire SPI write
Precondition	-
The called functions	-
Input parameter{in}	
spi_periph	SPI peripheral
SPIx	x=5
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* enable SPI5 quad wire write */
spi_quad_write_enable(SPI5);
```

spi_quad_read_enable

The description of spi_quad_read_enable is shown as below:

Table 3-1004. Function spi_quad_read_enable

Function name	spi_quad_read_enable
Function prototype	void spi_quad_read_enable(uint32_t spi_periph);
Function descriptions	enable quad wire SPI read
Precondition	-
The called functions	-
Input parameter{in}	
spi_periph	SPI peripheral
SPIx	x=5
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* enable SPI5 quad wire read */
spi_quad_read_enable(SPI5);
```

spi_quad_io23_output_enable

The description of spi_quad_io23_output_enable is shown as below:

Table 3-1005. Function spi_quad_io23_output_enable

Function name	spi_quad_io23_output_enable
Function prototype	void spi_quad_io23_output_enable(uint32_t spi_periph);
Function descriptions	enable SPI_IO2 and SPI_IO3 pin output
Precondition	-
The called functions	-
Input parameter{in}	
spi_periph	SPI peripheral
<i>SPIx</i>	x=5
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* enable SPI5 SPI_IO2 and SPI_IO3 pin output */
spi_quad_io23_output_enable(SPI5);
```

spi_quad_io23_output_disable

The description of spi_quad_io23_output_disable is shown as below:

Table 3-1006. Function spi_quad_io23_output_disable

Function name	spi_quad_io23_output_disable
Function prototype	void spi_quad_io23_output_disable(uint32_t spi_periph);
Function descriptions	disable SPI_IO2 and SPI_IO3 pin output
Precondition	-
The called functions	-
Input parameter{in}	
spi_periph	SPI peripheral
<i>SPIx</i>	x=5
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* disable SPI5 SPI_IO2 and SPI_IO3 pin output */
spi_quad_io23_output_disable(SPI5);
```

spi_i2s_flag_get

The description of spi_i2s_flag_get is shown as below:

Table 3-1007. Function spi_i2s_flag_get

Function name	spi_i2s_flag_get
Function prototype	FlagStatus spi_i2s_flag_get(uint32_t spi_periph, uint32_t spi_i2s_flag);
Function descriptions	get SPI and I2S flag status
Precondition	-
The called functions	-
Input parameter{in}	
spi_periph	SPI peripheral
SPIx	x=0,1,2,3,4,5
Input parameter{in}	
spi_i2s_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_FLAG_RXORERR	receive overrun error flag
SPI_FLAG_CONFERR	mode config error flag
SPI_FLAG_CRCERR	CRC error flag
SPI_FLAG_FERR	format 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
I2S_FLAG_FERR	format error flag
Output parameter{out}	
-	-
Return value	
FlagStatus	SET or RESET

Example:

```
/* get SPI0 transmit buffer empty flag status */
while(RESET == spi_i2s_flag_get(SPI0, SPI_FLAG_TBE));
spi_i2s_data_transmit(SPI0, spi0_send_array[send_n++]);
```

spi_i2s_interrupt_enable

The description of spi_i2s_interrupt_enable is shown as below:

Table 3-1008. Function spi_i2s_interrupt_enable

Function name	spi_i2s_interrupt_enable
Function prototype	void spi_i2s_interrupt_enable(uint32_t spi_periph, uint8_t spi_i2s_int);
Function descriptions	enable SPI and I2S interrupt

Precondition	-
The called functions	-
Input parameter{in}	
spi_periph	SPI peripheral
<i>SPIx</i>	x=0,1,2,3,4,5
Input parameter{in}	
spi_i2s_int	SPI/I2S interrupt
<i>SPI_I2S_INT_TBE</i>	transmit buffer empty interrupt
<i>SPI_I2S_INT_RBNE</i>	receive buffer not empty interrupt
<i>SPI_I2S_INT_ERR</i>	CRC error,configuration error,reception overrun error, transmission underrun error and format error interrupt
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* 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 3-1009. Function spi_i2s_interrupt_disable

Function name	spi_i2s_interrupt_disable
Function prototype	void spi_i2s_interrupt_disable(uint32_t spi_periph, uint8_t spi_i2s_int);
Function descriptions	disable SPI and I2S interrupt
Precondition	-
The called functions	-
Input parameter{in}	
spi_periph	SPI peripheral
<i>SPIx</i>	x=0,1,2,3,4,5
Input parameter{in}	
spi_i2s_int	SPI/I2S interrupt
<i>SPI_I2S_INT_TBE</i>	transmit buffer empty interrupt
<i>SPI_I2S_INT_RBNE</i>	receive buffer not empty interrupt
<i>SPI_I2S_INT_ERR</i>	CRC error,configuration error,reception overrun error, transmission underrun error and format error interrupt
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* 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 3-1010. Function spi_i2s_interrupt_flag_get

Function name	spi_i2s_interrupt_flag_get
Function prototype	FlagStatus spi_i2s_interrupt_flag_get(uint32_t spi_periph, uint8_t spi_i2s_int);
Function descriptions	get SPI and I2S interrupt status
Precondition	-
The called functions	-
Input parameter{in}	
spi_periph	SPI peripheral
<i>SPIx</i>	x=0,1,2,3,4,5
Input parameter{in}	
spi_i2s_int	SPI/I2S interrupt flag status
<i>SPI_I2S_INT_FLAG_TBE</i>	transmit buffer empty interrupt
<i>SPI_I2S_INT_FLAG_RBNE</i>	receive buffer not empty interrupt
<i>SPI_I2S_INT_FLAG_RXORERR</i>	overrun interrupt
<i>SPI_INT_FLAG_CONFIGERR</i>	config error interrupt
<i>SPI_INT_FLAG_CRCERR</i>	CRC error interrupt
<i>I2S_INT_FLAG_TXURERR</i>	underrun error interrupt
<i>I2S_INT_FLAG_FERR</i>	format error interrupt
Output parameter{out}	
-	-
Return value	
FlagStatus	SET or RESET

Example:

```
/* get SPI0 transmit buffer empty interrupt status */
if(RESET != spi_i2s_interrupt_flag_get(SPI0, SPI_I2S_INT_FLAG_TBE)){
    while(RESET == spi_i2s_flag_get(SPI0, SPI_FLAG_TBE));
}
```



```

        spi_i2s_data_transmit(SPI0, spi0_send_array[send_n++]);
    }

```

3.28. SYSCFG

The SYSCFG registers are listed in chapter [3.28.1](#), the TIMER firmware functions are introduced in chapter [3.28.2](#).

3.28.1. Descriptions of Peripheral registers

SYSCFG registers are listed in the table shown as below:

Table 3-1011. SYSCFG registers

Registers	Descriptions
SYSCFG_CFG0	Configuration register 0
SYSCFG_CFG1	Configuration register 1
SYSCFG_EXTISS0	EXTI sources selection register 0
SYSCFG_EXTISS1	EXTI sources selection register 1
SYSCFG_EXTISS2	EXTI sources selection register 2
SYSCFG_EXTISS3	EXTI sources selection register 3
SYSCFG_CPSCTL	I/O compensation control register

3.28.2. Descriptions of Peripheral functions

The description format of firmware functions are shown as below:

Table 3-1012.SYSCFG firmware function

Function name	Function description
syscfg_deinit	deinit syscfg module
syscfg_bootmode_config	configure the boot mode
syscfg_fmc_swap_config	configure FMC memory mapping swap
syscfg_exmc_swap_config	configure the EXMC swap
syscfg_exti_line_config	configure the GPIO pin as EXTI Line
syscfg_enet_phy_interface_config	configure the PHY interface for the ethernet MAC
syscfg_compensation_config	configure the I/O compensation cell

Function name	Function description
syscfg_flag_get	check the I/O compensation cell is ready or not

syscfg_deinit

The description of syscfg_deinit is shown as below:

Table 3-1013. Function syscfg_deinit

Function name	syscfg_deinit
Function prototype	void syscfg_deinit(void);
Function descriptions	deinit syscfg module
Precondition	-
The called functions	rcu_periph_reset_enable / rcu_periph_reset_disable
Input parameter{in}	
-	-
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* reset SYSCFG */
syscfg_deinit();
```

syscfg_bootmode_config

The description of syscfg_bootmode_config is shown as below:

Table 3-1014. Function syscfg_bootmode_config

Function name	syscfg_bootmode_config
Function prototype	void syscfg_bootmode_config(uint8_t syscfg_bootmode);
Function descriptions	configure the boot mode
Precondition	-
The called functions	-
Input parameter{in}	

syscfg_bootmode	selects the memory remapping
<i>SYSCFG_BOOTMODE_FLASH</i>	main flash memory (0x08000000~0x083BFFFF) is mapped at address 0x00000000
<i>SYSCFG_BOOTMODE_BOOTLOADER</i>	boot loader (0x1FFF0000 - 0x1FFF77FF) is mapped at address 0x00000000
<i>SYSCFG_BOOTMODE_EXMC_SRAM</i>	SRAM/NOR 0 and 1 of EXMC (0x60000000~0x67FFFFFF) is mapped at address 0x00000000
<i>SYSCFG_BOOTMODE_SRAM</i>	SRAM0 of on-chip SRAM (0x20000000~0x2001BFFF) is mapped at address 0x00000000
<i>SYSCFG_BOOTMODE_EXMC_SDRAM</i>	SDRAM bank0 of EXMC (0xC0000000~0xC7FFFFFF) is mapped at address 0x00000000
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* configure boot from main flash */
```

```
syscfg_bootmode_config(SYSCFG_BOOTMODE_FLASH);
```

syscfg_fmc_swap_config

The description of syscfg_fmc_swap_config is shown as below:

Table 3-1015. Function syscfg_fmc_swap_config

Function name	syscfg_fmc_swap_config
Function prototype	void syscfg_fmc_swap_config(uint32_t syscfg_fmc_swap);
Function descriptions	FMC memory mapping swap
Precondition	-
The called functions	-
Input parameter{in}	
syscfg_fmc_swap	selects the internal flash bank swapping
<i>SYSCFG_FMC_SWP_BANK0</i>	bank 0 is mapped at address 0x08000000 and bank 1 is mapped at address 0x08100000

<code>SYSCFG_FMC_SWP_BANK1</code>	bank 1 is mapped at address 0x08000000 and bank 0 is mapped at address 0x08100000
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* FMC memory bank 0 is mapped at address 0x08000000 and bank 1 is mapped at address 0x08100000 */
```

```
syscfg_fmc_swap_config(SYSCFG_FMC_SWP_BANK0);
```

syscfg_exmc_swap_config

The description of syscfg_exmc_swap_config is shown as below:

Table 3-1016. Function syscfg_exmc_swap_config

Function name	syscfg_exmc_swap_config
Function prototype	void syscfg_exmc_swap_config(uint32_t syscfg_exmc_swap);
Function descriptions	EXMC memory mapping swap
Precondition	-
The called functions	-
Input parameter{in}	
syscfg_exmc_swap	selects the memories in EXMC swapping
<code>SYSCFG_EXMC_SWP_ENABLE</code>	SDRAM bank 0 and bank 1 are swapped with NAND bank 1 and PC card
<code>SYSCFG_EXMC_SWP_DISABLE</code>	no memory mapping swap
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* enable EXMC memory mapping swap */
```

```
syscfg_extmc_swap_config(SYS_CFG_EXMC_SWAP_ENABLE);
```

syscfg_exti_line_config

The description of syscfg_exti_line_config is shown as below:

Table 3-1017. Function syscfg_exti_line_config

Function name	syscfg_exti_line_config
Function prototype	void syscfg_exti_line_config(uint8_t exti_port, uint8_t exti_pin);
Function descriptions	configure the GPIO pin as EXTI Line
Precondition	-
The called functions	-
Input parameter{in}	
exti_port	specify the GPIO port used in EXTI
EXTI_SOURCE_GPIOx	x = A,B,C,D,E,F,G,H,I
Input parameter{in}	
exti_pin	specify the EXTI line
EXTI_SOURCE_PINx	x = 0..15
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* configure the PA0 pin as EXTI Line */
syscfg_exti_line_config(EXTI_SOURCE_GPIOA, EXTI_SOURCE_PIN0);
```

syscfg_enet_phy_interface_config

The description of syscfg_enet_phy_interface_config is shown as below:

Table 3-1018. Function syscfg_enet_phy_interface_config

Function name	syscfg_enet_phy_interface_config
Function prototype	void syscfg_enet_phy_interface_config(uint32_t syscfg_enet_phy_interface);
Function descriptions	configure the PHY interface for the ethernet MAC

Precondition	-
The called functions	-
Input parameter{in}	
syscfg_enet_phy_interface	specifies the media interface mode
<i>SYSCFG_ENET_PHY_MII</i>	MII mode is selected
<i>SYSCFG_ENET_PHY_RMII</i>	RMII mode is selected
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* configure the MII PHY interface for the ethernet MAC */
```

```
syscfg_enet_phy_interface_config(SYSCFG_ENET_PHY_MII);
```

syscfg_compensation_config

The description of syscfg_compensation_config is shown as below:

Table 3-1019. Function syscfg_compensation_config

Function name	syscfg_compensation_config
Function prototype	void syscfg_compensation_config(uint32_t syscfg_compensation);
Function descriptions	configure the I/O compensation cell
Precondition	-
The called functions	-
Input parameter{in}	
syscfg_compensation	specifies the I/O compensation cell mode
<i>SYSCFG_COMPENSATION_ENABLE</i>	I/O compensation cell is enabled
<i>SYSCFG_COMPENSATION_DISABLE</i>	I/O compensation cell is disabled

Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* enable the I/O compensation cell function */
```

```
syscfg_compensation_config(SYS_CFG_COMPENSATION_ENABLE);
```

syscfg_flag_get

The description of syscfg_flag_get is shown as below:

Table 3-1020. Function syscfg_flag_get

Function name	syscfg_flag_get
Function prototype	FlagStatus syscfg_flag_get(void);
Function descriptions	checks whether the I/O compensation cell ready flag is set or not
Precondition	-
The called functions	-
Input parameter{in}	
-	-
Output parameter{out}	
-	-
Return value	
FlagStatus	SET or RESET

Example:

```
/* check whether the I/O compensation cell ready flag is set or not */
```

```
if(RESET != syscfg_flag_get());
```

3.29. 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 five sorts: advanced timer (TIMERx, x=0, 7), general level0 timer (TIMERx, x=1, 2, 3, 4), general level1 timer (TIMERx, x=8, 11), general level2 timer (TIMERx, x=9, 10, 12, 13), Basic timer (TIMERx, x=5,

6). The specific functions of different types of timer are different. The TIMER registers are listed in chapter [3.29.1](#), the TIMER firmware functions are introduced in chapter [3.29.2](#).

3.29.1. Descriptions of Peripheral registers

TIMERx registers are listed in the table shown as below:

Table 3-1021. TIMERx Registers

Registers	Descriptions
TIMER_CTL0	Control register 0
TIMER_CTL1	Control register 1
TIMER_SMCFG	Slave mode configuration register
TIMER_DMAINTEN	DMA and interrupt enable register
TIMER_INTF	Interrupt flag register
TIMER_SWEVG	Software event generation register
TIMER_CHCTL0	Channel control register 0
TIMER_CHCTL1	Channel control register 1
TIMER_CHCTL2	Channel control register 2
TIMER_CNT	Counter register
TIMER_PSC	Prescaler register
TIMER_CAR	Counter auto reload register
TIMER_CREP	Counter repetition register
TIMER_CH0CV	Channel 0 capture/compare value register
TIMER_CH1CV	Channel 1 capture/compare value register
TIMER_CH2CV	Channel 2 capture/compare value register
TIMER_CH3CV	Channel 3 capture/compare value register
TIMER_CCHP	Channel complementary protection register
TIMER_DMACFG	DMA configuration register
TIMER_DMATB	DMA transfer buffer register
TIMER_CH0COMV_ADD	Channel 0 additional compare value register
TIMER_CH1COMV	Channel 1 additional compare value register

Registers	Descriptions
_ADD	
TIMER_CH2COMV_ADD	Channel 2 additional compare value register
TIMER_CH3COMV_ADD	Channel 3 additional compare value register
TIMER_CTL2	Control register 2
TIMER_IRMP	TIMER input remap register
TIMER_CFG	Configuration register

3.29.2. Descriptions of Peripheral functions

The description format of firmware functions are shown as below:

Table 3-1022.TIMERx firmware function

Function name	Function description
timer_deinit	deinit a timer
timer_struct_para_init	initialize the parameters of TIMER init parameter struct with the default values
timer_init	initialize TIMER counter
timer_enable	enable a timer
timer_disable	disable a timer
timer_auto_reload_shadow_enable	enable the auto reload shadow function
timer_auto_reload_shadow_disable	disable the auto reload shadow function
timer_update_event_enable	enable the update event
timer_update_event_disable	disable the update event
timer_counter_alignment	set TIMER counter alignment mode
timer_counter_up_direction	set TIMER counter up direction
timer_counter_down_direction	set TIMER counter down direction
timer_prescaler_config	configure TIMER prescaler
timer_repetition_value_config	configure TIMER repetition register value
timer_autoreload_value_config	configure TIMER autoreload register value

Function name	Function description
timer_counter_value_config	configure TIMER counter register value
timer_counter_read	read TIMER counter value
timer_prescaler_read	read TIMER prescaler value
timer_single_pulse_mode_config	configure TIMER single pulse mode
timer_update_source_config	configure TIMER update source
timer_dma_enable	enable the TIMER DMA
timer_dma_disable	disable the TIMER DMA
timer_channel_dma_request_source_select	channel DMA request source selection
timer_dma_transfer_config	configure the TIMER DMA transfer
timer_event_software_generate	software generate events
timer_break_struct_para_init	initialize the parameters of TIMER break parameter struct with the default values
timer_break_config	configure TIMER break function
timer_break_enable	enable TIMER break function
timer_break_disable	disable TIMER break function
timer_automatic_output_enable	enable TIMER output automatic function
timer_automatic_output_disable	disable TIMER output automatic function
timer_primary_output_config	configure TIMER primary output function
timer_channel_control_shadow_config	channel capture/compare control shadow register enable
timer_channel_control_shadow_update_config	configure TIMER channel control shadow register update control
timer_channel_output_struct_para_init	initialize the parameters of TIMER channel output parameter struct with the default values
timer_channel_output_config	configure TIMER channel output function
timer_channel_output_mode_config	configure TIMER channel output compare mode
timer_channel_output_pulse_value_config	configure TIMER channel output pulse value
timer_channel_output_shadow_	configure TIMER channel output shadow function

Function name	Function description
config	
timer_channel_output_fast_config	configure TIMER channel output fast function
timer_channel_output_clear_config	configure TIMER channel output clear function
timer_channel_output_polarity_config	configure TIMER channel output polarity
timer_channel_complementary_output_polarity_config	configure TIMER channel complementary output polarity
timer_channel_output_state_config	configure TIMER channel enable state
timer_channel_complementary_output_state_config	configure TIMER channel complementary output enable state
timer_channel_input_struct_para_init	initialize the parameters of TIMER channel input parameter struct with the default values
timer_input_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_clock_config	configure TIMER internal clock 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

Function name	Function description
ock_config	
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_output_value_selection_config	configure TIMER output value selection
timer_channel_remap_config	configure TIMER channel remap function
timer_channel_composite_pwm_enable	enable the TIMER composite pwm
timer_channel_composite_pwm_disable	disable the TIMER composite pwm
timer_channel_additional_compare_value_config	configure TIMER channel additional compare value
timer_channel_additional_output_shadow_config	configure TIMER channel additional output shadow function
timer_channel_additional_compare_value_read	get TIMER channel additional compare value
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 3-1023. Structure timer_parameter_struct

Member name	Function description
prescaler	prescaler value (0~65535)
alignedmode	aligned mode (TIMER_COUNTER_EDGE,

Member name	Function description
	TIMER_COUNTER_CENTER_DOWN, TIMER_COUNTER_CENTER_UP, TIMER_COUNTER_CENTER_BOTH)
counterdirection	counter direction (TIMER_COUNTER_UP, TIMER_COUNTER_DOWN)
period	period value (0~65535)
clockdivision	clock division value (TIMER_CKDIV_DIV1, TIMER_CKDIV_DIV2, TIMER_CKDIV_DIV4)
repetitioncounter	the counter repetition value (0~255)

Structure timer_break_parameter_struct

Table 3-1024. Structure timer_break_parameter_struct

Member name	Function description
runoffstate	run mode off-state (TIMER_ROS_STATE_ENABLE, TIMER_ROS_STATE_DISABLE)
ideloffstate	idle mode off-state (TIMER_IOS_STATE_ENABLE, TIMER_IOS_STATE_DISABLE)
deadtime	dead time (0~255)
breakpolarity	break polarity (TIMER_BREAK_POLARITY_LOW, TIMER_BREAK_POLARITY_HIGH)
outputautostate	output automatic enable (TIMER_OUTAUTO_ENABLE, TIMER_OUTAUTO_DISABLE)
protectmode	complementary register protect control (TIMER_CCHP_PROT_OFF, TIMER_CCHP_PROT_0, TIMER_CCHP_PROT_1, TIMER_CCHP_PROT_2)
breakstate	break enable (TIMER_BREAK_ENABLE, TIMER_BREAK_DISABLE)

Structure timer_oc_parameter_struct

Table 3-1025. Structure timer_oc_parameter_struct

Member name	Function description
outputstate	channel output state (TIMER_CCX_ENABLE, TIMER_CCX_DISABLE)
outputnstate	channel complementary output state (TIMER_CCXN_ENABLE, TIMER_CCXN_DISABLE)
ocpolarity	channel output polarity (TIMER_OC_POLARITY_HIGH, TIMER_OC_POLARITY_LOW)
ocnpolarity	channel complementary output polarity (TIMER_OCN_POLARITY_HIGH, TIMER_OCN_POLARITY_LOW)

Member name	Function description
ocidlestate	idle state of channel output (TIMER_OC_IDLE_STATE_LOW, TIMER_OC_IDLE_STATE_HIGH)
ocnidlestate	idle state of channel complementary output (TIMER_OCN_IDLE_STATE_LOW, TIMER_OCN_IDLE_STATE_HIGH)

Structure timer_ic_parameter_struct

Table 3-1026. Structure timer_ic_parameter_struct

Member name	Function description
icpolarity	channel input polarity (TIMER_IC_POLARITY_RISING, TIMER_IC_POLARITY_FALLING, TIMER_IC_POLARITY_BOTH_EDGE)
icselection	channel input mode selection (TIMER_IC_SELECTION_DIRECTTI, TIMER_IC_SELECTION_INDIRECTTI, TIMER_IC_SELECTION_ITS)
icprescaler	channel input capture prescaler (TIMER_IC_PSC_DIV1, TIMER_IC_PSC_DIV2, TIMER_IC_PSC_DIV4, TIMER_IC_PSC_DIV8)
icfilter	channel input capture filter control (0~15)

timer_deinit

The description of timer_deinit is shown as below:

Table 3-1027. Function timer_deinit

Function name	timer_deinit
Function prototype	void timer_deinit(uint32_t timer_periph);
Function descriptions	deinit a TIMER
Precondition	-
The called functions	rcu_periph_reset_enable / rcu_periph_reset_disable
Input parameter{in}	
timer_periph	TIMER peripheral
TIMERx(x=0..13)	TIMER peripheral selection
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* reset TIMER0 */
timer_deinit (TIMER0);
```

timer_struct_para_init

The description of timer_struct_para_init is shown as below:

Table 3-1028. Function timer_struct_para_init

Function name	timer_struct_para_init
Function prototype	void timer_struct_para_init(timer_parameter_struct* initpara);
Function descriptions	initialize the parameters of TIMER init parameter struct with the default values
Precondition	-
The called functions	-
Input parameter{in}	
initpara	TIMER init parameter struct, the structure members can refer to Table 3-1023. Structure timer parameter struct .
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* 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 3-1029. Function timer_init

Function name	timer_init
Function prototype	void timer_init(uint32_t timer_periph, timer_parameter_struct* initpara);
Function descriptions	initialize TIMER counter

Precondition	-
The called functions	-
Input parameter{in}	
timer_periph	TIMER peripheral
<i>TIMERx(x=0..13)</i>	TIMER peripheral selection
Input parameter{in}	
initpara	TIMER init parameter struct, the structure members can refer to Table 3-1023. Structure timer parameter struct.
Output parameter{out}	
-	-
Return value	
-	-

Example:

```

/* initialize TIMER0 */

timer_parameter_struct timer_initpara;

timer_initpara.prescaler      = 107;

timer_initpara.alignedmode    = TIMER_COUNTER_EDGE;

timer_initpara.counterdirection = TIMER_COUNTER_UP;

timer_initpara.period         = 999;

timer_initpara.clockdivision   = TIMER_CKDIV_DIV1;

timer_initpara.repetitioncounter = 1;

timer_init(TIMER0,&timer_initpara);

```

timer_enable

The description of timer_enable is shown as below:

Table 3-1030. Function timer_enable

Function name	timer_enable
Function prototype	void timer_enable(uint32_t timer_periph);
Function descriptions	enable a timer
Precondition	-

The called functions	-
Input parameter{in}	
timer_periph	TIMER peripheral
<i>TIMERx(x=0..13)</i>	TIMER peripheral selection
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* enable TIMER0 */
```

```
timer_enable (TIMER0);
```

timer_disable

The description of timer_disable is shown as below:

Table 3-1031. Function timer_disable

Function name	timer_disable
Function prototype	void timer_disable(uint32_t timer_periph);
Function descriptions	disable a timer
Precondition	-
The called functions	-
Input parameter{in}	
timer_periph	TIMER peripheral
<i>TIMERx(x=0..13)</i>	TIMER peripheral selection
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* disable TIMER0 */
```

```
timer_disable (TIMER0);
```

timer_auto_reload_shadow_enable

The description of timer_auto_reload_shadow_enable is shown as below:

Table 3-1032. Function timer_auto_reload_shadow_enable

Function name	timer_auto_reload_shadow_enable
Function prototype	void timer_auto_reload_shadow_enable(uint32_t timer_periph);
Function descriptions	enable the auto reload shadow function
Precondition	-
The called functions	-
Input parameter{in}	
timer_periph	TIMER peripheral
<i>TIMERx(x=0..13)</i>	TIMER peripheral selection
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* enable the TIMER0 auto reload shadow function */
```

```
timer_auto_reload_shadow_enable (TIMER0);
```

timer_auto_reload_shadow_disable

The description of timer_auto_reload_shadow_disable is shown as below:

Table 3-1033. Function timer_auto_reload_shadow_disable

Function name	timer_auto_reload_shadow_disable
Function prototype	void timer_auto_reload_shadow_disable (uint32_t timer_periph);
Function descriptions	disable the auto reload shadow function
Precondition	-
The called functions	-
Input parameter{in}	
timer_periph	TIMER peripheral
<i>TIMERx(x=0..13)</i>	TIMER peripheral selection

Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* disable the TIMER0 auto reload shadow function */
```

```
timer_auto_reload_shadow_disable (TIMER0);
```

timer_update_event_enable

The description of timer_update_event_enable is shown as below:

Table 3-1034. Function timer_update_event_enable

Function name	timer_update_event_enable
Function prototype	void timer_update_event_enable(uint32_t timer_periph);
Function descriptions	enable the update event
Precondition	-
The called functions	-
Input parameter{in}	
timer_periph	TIMER peripheral
<i>TIMERx(x=0..13)</i>	TIMER peripheral selection
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* enable TIMER0 the update event */
```

```
timer_update_event_enable (TIMER0);
```

timer_update_event_disable

The description of timer_update_event_disable is shown as below:

Table 3-1035. Function timer_update_event_disable

Function name	timer_update_event_disable
Function prototype	void timer_update_event_disable (uint32_t timer_periph);
Function descriptions	disable the update event
Precondition	-
The called functions	-
Input parameter{in}	
timer_periph	TIMER peripheral
TIMER _x (x=0..13)	TIMER peripheral selection
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* disable TIMER0 the update event */
timer_update_event_disable (TIMER0);
```

timer_counter_alignment

The description of timer_counter_alignment is shown as below:

Table 3-1036. Function timer_counter_alignment

Function name	timer_counter_alignment
Function prototype	void timer_counter_alignment(uint32_t timer_periph, uint16_t aligned);
Function descriptions	set TIMER counter alignment mode
Precondition	-
The called functions	-
Input parameter{in}	
timer_periph	TIMER peripheral
TIMER _x (x=0..4,7..13)	TIMER peripheral selection
Input parameter{in}	
aligned	alignment mode

<i>TIMER_COUNTER_EDGE</i>	No center-aligned mode (edge-aligned mode). The direction of the counter is specified by the DIR bit.
<i>TIMER_COUNTER_CENTER_DOWN</i>	Center-aligned and counting down assert mode. The counter counts under center aligned and channel is configured in output mode (CHxMS=00 in TIMEx_CHCTL0 register). Only when the counter is counting down, compare interrupt flag of channels can be set.
<i>TIMER_COUNTER_CENTER_UP</i>	Center-aligned and counting up assert mode. The counter counts under center aligned and channel is configured in output mode (CHxMS=00 in TIMEx_CHCTL0 register). Only when the counter is counting up, compare interrupt flag of channels can be set.
<i>TIMER_COUNTER_CENTER_BOTH</i>	Center-aligned and counting up/down assert mode. The counter counts under center-aligned and channel is configured in output mode (CHxMS=00 in TIMEx_CHCTL0 register). Both when the counter is counting up and counting down, compare interrupt flag of channels can be set.
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* set TIMER0 counter center-aligned and counting up assert mode */
timer_counter_alignment (TIMER0, TIMER_COUNTER_CENTER_UP);
```

timer_counter_up_direction

The description of timer_counter_up_direction is shown as below:

Table 3-1037. Function timer_counter_up_direction

Function name	timer_counter_up_direction
Function prototype	void timer_counter_up_direction(uint32_t timer_periph);
Function descriptions	set TIMER counter up direction
Precondition	set TIMER counter no center-aligned mode (edge-aligned mode)
The called functions	-
Input parameter{in}	
timer_periph	TIMER peripheral
<i>TIMEx(x=0..4,7..13)</i>	TIMER peripheral selection

Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* set TIMER0 counter up direction */
```

```
timer_counter_up_direction (TIMER0);
```

timer_counter_down_direction

The description of timer_counter_down_direction is shown as below:

Table 3-1038. Function timer_counter_down_direction

Function name	timer_counter_down_direction
Function prototype	void timer_counter_down_direction(uint32_t timer_periph);
Function descriptions	set TIMER counter down direction
Precondition	set TIMER counter no center-aligned mode (edge-aligned mode)
The called functions	-
Input parameter{in}	
timer_periph	TIMER peripheral
TIMERx(x=0..4,7..13)	TIMER peripheral selection
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* set TIMER0 counter down direction */
```

```
timer_counter_down_direction (TIMER0);
```

timer_prescaler_config

The description of timer_prescaler_config is shown as below:

Table 3-1039. Function timer_prescaler_config

Function name	timer_prescaler_config
Function prototype	void timer_prescaler_config(uint32_t timer_periph, uint16_t prescaler, uint8_t pscreload);
Function descriptions	configure TIMER prescaler
Precondition	-
The called functions	-
Input parameter{in}	
timer_periph	TIMER peripheral
<i>TIMERx(x=0..13)</i>	TIMER peripheral selection
Input parameter{in}	
prescaler	prescaler value (0~65535)
Input parameter{in}	
pscreload	prescaler reload mode
<i>TIMER_PSC_RELOAD_NOW</i>	the prescaler is loaded right now
<i>TIMER_PSC_RELOAD_UPDATE</i>	the prescaler is loaded at the next update event
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* configure TIMER0 prescaler */
```

```
timer_prescaler_config (TIMER0, 3000, TIMER_PSC_RELOAD_NOW);
```

timer_repetition_value_config

The description of timer_repetition_value_config is shown as below:

Table 3-1040. Function timer_repetition_value_config

Function name	timer_repetition_value_config
Function prototype	void timer_repetition_value_config(uint32_t timer_periph, uint16_t

	repetition);
Function descriptions	configure TIMER repetition register value
Precondition	-
The called functions	-
Input parameter{in}	
timer_periph	TIMER peripheral
<i>TIMERx(x=0,7)</i>	TIMER peripheral selection
Input parameter{in}	
repetition	the counter repetition value (0~255)
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* configure TIMER0 repetition register value */
timer_repetition_value_config (TIMER0, 98);
```

timer_autoreload_value_config

The description of timer_autoreload_value_config is shown as below:

Table 3-1041. Function timer_autoreload_value_config

Function name	timer_autoreload_value_config
Function prototype	void timer_autoreload_value_config(uint32_t timer_periph, uint32_t autoreload);
Function descriptions	configure TIMER autoreload register value
Precondition	-
The called functions	-
Input parameter{in}	
timer_periph	TIMER peripheral
<i>TIMERx(x=0..13)</i>	TIMER peripheral selection
Input parameter{in}	

autoreload	the counter auto-reload value
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* configure TIMER autoreload register value */
timer_autoreload_value_config (TIMER0, 3000);
```

timer_counter_value_config

The description of timer_counter_value_config is shown as below:

Table 3-1042. Function timer_counter_value_config

Function name	timer_counter_value_config
Function prototype	void timer_counter_value_config(uint32_t timer_periph, uint32_t counter);
Function descriptions	configure TIMER counter register value
Precondition	-
The called functions	-
Input parameter{in}	
timer_periph	TIMER peripheral
<i>TIMERx(x=0..13)</i>	TIMER peripheral selection
Input parameter{in}	
counter	the counter value
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* configure TIMER0 counter register value */
timer_counter_value_config (TIMER0);
```

timer_counter_read

The description of timer_counter_read is shown as below:

Table 3-1043. Function timer_counter_read

Function name	timer_counter_read
Function prototype	uint32_t timer_counter_read(uint32_t timer_periph);
Function descriptions	read TIMER counter value
Precondition	-
The called functions	-
Input parameter{in}	
timer_periph	TIMER peripheral
<i>TIMERx(x=0..13)</i>	TIMER peripheral selection
Output parameter{out}	
-	-
Return value	
uint32_t	counter value

Example:

```
/* read TIMERO0 counter value */
uint32_t i = 0;
i = timer_counter_read (TIMERO0);
```

timer_prescaler_read

The description of timer_prescaler_read is shown as below:

Table 3-1044. Function timer_prescaler_read

Function name	timer_prescaler_read
Function prototype	uint16_t timer_prescaler_read(uint32_t timer_periph);
Function descriptions	read TIMER prescaler value
Precondition	-
The called functions	-
Input parameter{in}	
timer_periph	TIMER peripheral

<i>TIMERx</i> (<i>x</i> =0..13)	TIMER peripheral selection
Output parameter{out}	
-	-
Return value	
uint16_t	prescaler register value (0x0000~0xFFFF)

Example:

```
/* read TIMER0 prescaler value */
```

```
uint16_t i = 0;
```

```
i = timer_prescaler_read (TIMER0);
```

timer_single_pulse_mode_config

The description of timer_single_pulse_mode_config is shown as below:

Table 3-1045. Function timer_single_pulse_mode_config

Function name	timer_single_pulse_mode_config
Function prototype	void timer_single_pulse_mode_config(uint32_t timer_periph, uint8_t spmode);
Function descriptions	configure TIMER single pulse mode
Precondition	-
The called functions	-
Input parameter{in}	
timer_periph	TIMER peripheral
<i>TIMERx</i> (<i>x</i> =0..8,11)	TIMER peripheral selection
Input parameter{in}	
spmode	pulse mode
<i>TIMER_SP_MODE_SINGLE</i>	single pulse mode
<i>TIMER_SP_MODE_REPETITIVE</i>	repetitive pulse mode
Output parameter{out}	
-	-

Return value	
-	-

Example:

```
/* configure TIMER0 single pulse mode */
```

```
timer_single_pulse_mode_config (TIMER0, TIMER_SP_MODE_SINGLE);
```

timer_update_source_config

The description of timer_update_source_config is shown as below:

Table 3-1046. Function timer_update_source_config

Function name	timer_update_source_config
Function prototype	void timer_update_source_config(uint32_t timer_periph, uint32_t update);
Function descriptions	configure TIMER update source
Precondition	-
The called functions	-
Input parameter{in}	
timer_periph	TIMER peripheral
<i>TIMERx(x=0..13)</i>	TIMER peripheral selection
Input parameter{in}	
update	update source
<i>TIMER_UPDATE_SRC_GLOBAL</i>	Any of the following events generate an update interrupt or DMA request: <ul style="list-style-type: none"> - The UPG bit is set - The counter generates an overflow or underflow event - The slave mode controller generates an update event
<i>TIMER_UPDATE_SRC_REGULAR</i>	Only counter overflow/underflow generates an update interrupt or DMA request.
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* configure TIMER update only by counter overflow/underflow */
```

```
timer_update_source_config (TIMER0, TIMER_UPDATE_SRC_REGULAR);
```

timer_dma_enable

The description of timer_dma_enable is shown as below:

Table 3-1047. Function timer_dma_enable

Function name	timer_dma_enable
Function prototype	void timer_dma_enable(uint32_t timer_periph, uint16_t dma);
Function descriptions	enable the TIMER DMA
Precondition	-
The called functions	-
Input parameter{in}	
timer_periph	TIMER peripheral
<i>TIMERx</i>	please refer to the following parameters
Input parameter{in}	
dma	timer DMA source enable
<i>TIMER_DMA_UPD</i>	update DMA enable, <i>TIMERx</i> (x=0..7)
<i>TIMER_DMA_CH0D</i>	channel 0 DMA enable, <i>TIMERx</i> (x=0..4,7)
<i>TIMER_DMA_CH1D</i>	channel 1 DMA enable, <i>TIMERx</i> (x=0..4,7)
<i>TIMER_DMA_CH2D</i>	channel 2 DMA enable, <i>TIMERx</i> (x=0..4,7)
<i>TIMER_DMA_CH3D</i>	channel 3 DMA enable, <i>TIMERx</i> (x=0..4,7)
<i>TIMER_DMA_CMTD</i>	commutation DMA request enable, <i>TIMERx</i> (x=0,7)
<i>TIMER_DMA_TRGD</i>	trigger DMA enable, <i>TIMERx</i> (x=0..4,7)
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* enable the TIMER0 update DMA */
```

```
timer_dma_enable (TIMER0, TIMER_DMA_UPD);
```

timer_dma_disable

The description of timer_dma_disable is shown as below:

Table 3-1048. Function timer_dma_disable

Function name	timer_dma_disable
Function prototype	void timer_dma_disable (uint32_t timer_periph, uint16_t dma);
Function descriptions	disable the TIMER DMA
Precondition	-
The called functions	-
Input parameter{in}	
timer_periph	TIMER peripheral
<i>TIMERx</i>	please refer to the following parameters
Input parameter{in}	
dma	timer DMA source disable
<i>TIMER_DMA_UPD</i>	update DMA disable, TIMERx(x=0..7)
<i>TIMER_DMA_CH0D</i>	channel 0 DMA disable, TIMERx(x=0..4,7)
<i>TIMER_DMA_CH1D</i>	channel 1 DMA disable, TIMERx(x=0..4,7)
<i>TIMER_DMA_CH2D</i>	channel 2 DMA disable, TIMERx(x=0..4,7)
<i>TIMER_DMA_CH3D</i>	channel 3 DMA disable, TIMERx(x=0..4,7)
<i>TIMER_DMA_CMTD</i>	commutation DMA request disable, TIMERx(x=0,7)
<i>TIMER_DMA_TRGD</i>	trigger DMA disable, TIMERx(x=0..4,7)
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* disable the TIMER0 update DMA */
timer_dma_disable (TIMER0, TIMER_DMA_UPD);
```

timer_channel_dma_request_source_select

The description of timer_channel_dma_request_source_select is shown as below:

Table 3-1049. Function timer_channel_dma_request_source_select

Function name	timer_channel_dma_request_source_select
Function prototype	void timer_channel_dma_request_source_select(uint32_t timer_periph, uint32_t dma_request);
Function descriptions	channel DMA request source selection
Precondition	-
The called functions	-
Input parameter{in}	
timer_periph	TIMER peripheral
<i>TIMERx</i>	TIMER peripheral selection
Input parameter{in}	
dma_request	channel DMA request source selection
<i>TIMER_DMAREQUEST_CHANNELEVENT</i>	DMA request of channel n is sent when channel y event occurs
<i>TIMER_DMAREQUEST_UPDATEEVENT</i>	DMA request of channel n is sent when update event occurs
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* TIMER0 channel DMA request of channel n is sent when channel y event occurs */
```

```
timer_channel_dma_request_source_select(TIMER0,
TIMER_DMAREQUEST_CHANNELEVENT);
```

timer_dma_transfer_config

The description of timer_dma_transfer_config is shown as below:

Table 3-1050. Function timer_dma_transfer_config

Function name	timer_dma_transfer_config
Function prototype	void timer_dma_transfer_config(uint32_t timer_periph, uint32_t dma_baseaddr, uint32_t dma_lenth);

Function descriptions	configure the TIMER DMA transfer
Precondition	-
The called functions	-
Input parameter{in}	
timer_periph	TIMER peripheral
<i>TIMERx</i>	please refer to the following parameters
Input parameter{in}	
dma_baseaddr	DMA transfer access start address
<i>TIMER_DMACFG_DMA TA_CTL0</i>	DMA transfer address is TIMER_CTL0, TIMERx(x=0..4,7)
<i>TIMER_DMACFG_DMA TA_CTL1</i>	DMA transfer address is TIMER_CTL1, TIMERx(x=0..4,7)
<i>TIMER_DMACFG_DMA TA_SMCFG</i>	DMA transfer address is TIMER_SMCFG, TIMERx(x=0..4,7)
<i>TIMER_DMACFG_DMA TA_DMAINTEN</i>	DMA transfer address is TIMER_DMAINTEN, TIMERx(x=0..4,7)
<i>TIMER_DMACFG_DMA TA_INTF</i>	DMA transfer address is TIMER_INTF, TIMERx(x=0..4,7)
<i>TIMER_DMACFG_DMA TA_SWEVG</i>	DMA transfer address is TIMER_SWEVG, TIMERx(x=0..4,7)
<i>TIMER_DMACFG_DMA TA_CHCTL0</i>	DMA transfer address is TIMER_CHCTL0, TIMERx(x=0..4,7)
<i>TIMER_DMACFG_DMA TA_CHCTL1</i>	DMA transfer address is TIMER_CHCTL1, TIMERx(x=0..4,7)
<i>TIMER_DMACFG_DMA TA_CHCTL2</i>	DMA transfer address is TIMER_CHCTL2, TIMERx(x=0..4,7)
<i>TIMER_DMACFG_DMA TA_CNT</i>	DMA transfer address is TIMER_CNT, TIMERx(x=0..4,7)
<i>TIMER_DMACFG_DMA TA_PSC</i>	DMA transfer address is TIMER_PSC, TIMERx(x=0..4,7)
<i>TIMER_DMACFG_DMA TA_CAR</i>	MA transfer address is TIMER_CAR, TIMERx(x=0..4,7)
<i>TIMER_DMACFG_DMA</i>	DMA transfer address is TIMER_CREP, TIMERx(x=0,7)

<i>TA_CREP</i>	
<i>TIMER_DMACFG_DMA</i> <i>TA_CH0CV</i>	DMA transfer address is <i>TIMER_CH0CV</i> , <i>TIMERx</i> (<i>x</i> =0..4,7)
<i>TIMER_DMACFG_DMA</i> <i>TA_CH1CV</i>	DMA transfer address is <i>TIMER_CH1CV</i> , <i>TIMERx</i> (<i>x</i> =0..4,7)
<i>TIMER_DMACFG_DMA</i> <i>TA_CH2CV</i>	DMA transfer address is <i>TIMER_CH2CV</i> , <i>TIMERx</i> (<i>x</i> =0..4,7)
<i>TIMER_DMACFG_DMA</i> <i>TA_CH3CV</i>	DMA transfer address is <i>TIMER_CH3CV</i> , <i>TIMERx</i> (<i>x</i> =0..4,7)
<i>TIMER_DMACFG_DMA</i> <i>TA_CCHP</i>	DMA transfer address is <i>TIMER_CCHP</i> , <i>TIMERx</i> (<i>x</i> =0,7)
<i>TIMER_DMACFG_DMA</i> <i>TA_DMACFG</i>	DMA transfer address is <i>TIMER_DMACFG</i> , <i>TIMERx</i> (<i>x</i> =0..4,7)
<i>TIMER_DMACFG_DMA</i> <i>TA_DMATB</i>	DMA transfer address is <i>TIMER_DMATB</i> , <i>TIMERx</i> (<i>x</i> =0..4,7)
Input parameter{in}	
dma_lenth	DMA transfer count
<i>TIMER_DMACFG_DMA</i> <i>TC_xTRANSFER</i>	<i>x</i> =1..18, DMA transfer <i>x</i> time
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* configure the TIMER0 DMA transfer */
```

```
timer_dma_transfer_config(TIMER0, TIMER_DMACFG_DMATA_CTL0,  
TIMER_DMACFG_DMATC_5TRANSFER);
```

timer_event_software_generate

The description of `timer_event_software_generate` is shown as below:

Table 3-1051. Function `timer_event_software_generate`

Function name	<code>timer_event_software_generate</code>
Function prototype	<code>void timer_event_software_generate(uint32_t timer_periph, uint16_t</code>

	event);
Function descriptions	software generate events
Precondition	-
The called functions	-
Input parameter{in}	
timer_periph	TIMER peripheral
<i>TIMERx</i>	please refer to the following parameters
Input parameter{in}	
event	the timer software event generation sources
<i>TIMER_EVENT_SRC_UPG</i>	update event,TIMERx(x=0..13)
<i>TIMER_EVENT_SRC_CH0G</i>	channel 0 capture or compare event generation,TIMERx(x=0..4,7..13)
<i>TIMER_EVENT_SRC_CH1G</i>	channel 1 capture or compare event generation,TIMERx(x=0..4,7,8,11)
<i>TIMER_EVENT_SRC_CH2G</i>	channel 2 capture or compare event generation,TIMERx(x=0..4,7)
<i>TIMER_EVENT_SRC_CH3G</i>	channel 3 capture or compare event generation,TIMERx(x=0..4,7)
<i>TIMER_EVENT_SRC_CMTG</i>	channel commutation event generation,TIMERx(x=0,7)
<i>TIMER_EVENT_SRC_TRG</i>	trigger event generation,TIMERx(x=0..4,7,8,11)
<i>TIMER_EVENT_SRC_BRK</i>	break event generation,TIMERx(x=0,7)
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* software generate update event*/
```

```
timer_event_software_generate (TIMER0, TIMER_EVENT_SRC_UPG);
```

timer_break_struct_para_init

The description of timer_break_struct_para_init is shown as below:

Table 3-1052. Function timer_break_struct_para_init

Function name	timer_break_struct_para_init
Function prototype	void timer_break_struct_para_init(timer_break_parameter_struct* breakpara);
Function descriptions	initialize the parameters of TIMER break parameter struct with the default values
Precondition	-
The called functions	-
Input parameter{in}	
breakpara	TIMER break parameter struct, the structure members can refer to Table 3-1024. Structure timer_break_parameter_struct .
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* initialize TIMER break parameter struct with a default value */
timer_break_parameter_struct timer_breakpara;
timer_break_struct_para_init(timer_breakpara);
```

timer_break_config

The description of timer_break_config is shown as below:

Table 3-1053. Function timer_break_config

Function name	timer_break_config
Function prototype	void timer_break_config(uint32_t timer_periph, timer_break_parameter_struct* breakpara);
Function descriptions	configure TIMER break function
Precondition	-
The called functions	-

Input parameter{in}	
timer_periph	TIMER peripheral
<i>TIMERx(x=0,7)</i>	TIMER peripheral selection
Input parameter{in}	
breakpara	TIMER break parameter struct, the structure members can refer to Table 3-1024. Structure timer break parameter struct.
Output parameter{out}	
-	-
Return value	
-	-

Example:

```

/* configure TIMER0 break function */

timer_break_parameter_struct timer_breakpara;

timer_breakpara.runoffstate      = TIMER_ROS_STATE_DISABLE;
timer_breakpara.ideloffstate     = TIMER_IOS_STATE_DISABLE ;
timer_breakpara.deadtime        = 255;
timer_breakpara.breakpolarity   = TIMER_BREAK_POLARITY_LOW;
timer_breakpara.outputautostate = TIMER_OUTAUTO_ENABLE;
timer_breakpara.protectmode     = TIMER_CCHP_PROT_0;
timer_breakpara.breakstate      = TIMER_BREAK_ENABLE;

timer_break_config(TIMER0, &timer_breakpara);

```

timer_break_enable

The description of timer_break_enable is shown as below:

Table 3-1054. Function timer_break_enable

Function name	timer_break_enable
Function prototype	void timer_break_enable(uint32_t timer_periph);
Function descriptions	enable TIMER break function
Precondition	This function can be called only when PROT [1:0] bit-filed in TIMERx_CCHP register is 00.

The called functions	-
Input parameter{in}	
timer_periph	TIMER peripheral
<i>TIMERx(x=0,7)</i>	TIMER peripheral selection
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* enable TIMER0 break function*/
```

```
timer_break_enable (TIMER0);
```

timer_break_disable

The description of timer_break_disable is shown as below:

Table 3-1055. Function timer_break_disable

Function name	timer_break_disable
Function prototype	void timer_break_disable(uint32_t timer_periph);
Function descriptions	disable TIMER break function
Precondition	This function can be called only when PROT [1:0] bit-filed in TIMERx_CCHP register is 00.
The called functions	-
Input parameter{in}	
timer_periph	TIMER peripheral
<i>TIMERx(x=0,7)</i>	TIMER peripheral selection
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* disable TIMER0 break function*/
```

timer_break_disable (TIMER0);

timer_automatic_output_enable

The description of timer_automatic_output_enable is shown as below:

Table 3-1056. Function timer_automatic_output_enable

Function name	timer_automatic_output_enable
Function prototype	void timer_automatic_output_enable(uint32_t timer_periph);
Function descriptions	enable TIMER output automatic function
Precondition	This function can be called only when PROT [1:0] bit-filed in TIMERx_CCHP register is 00.
The called functions	-
Input parameter{in}	
timer_periph	TIMER peripheral
<i>TIMERx(x=0,7)</i>	TIMER peripheral selection
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* enable TIMER0 output automatic function */
```

```
timer_automatic_output_enable (TIMER0);
```

timer_automatic_output_disable

The description of timer_automatic_output_disable is shown as below:

Table 3-1057. Function timer_automatic_output_disable

Function name	timer_automatic_output_disable
Function prototype	void timer_automatic_output_disable (uint32_t timer_periph);
Function descriptions	disable TIMER output automatic function
Precondition	This function can be called only when PROT [1:0] bit-filed in TIMERx_CCHP register is 00.
The called functions	-

Input parameter{in}	
timer_periph	TIMER peripheral
<i>TIMERx(x=0,7)</i>	TIMER peripheral selection
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* disable TIMER0 output automatic function */
```

```
timer_automatic_output_disable (TIMER0);
```

timer_primary_output_config

The description of timer_primary_output_config is shown as below:

Table 3-1058. Function timer_primary_output_config

Function name	timer_primary_output_config
Function prototype	void timer_primary_output_config(uint32_t timer_periph, ControlStatus newvalue);
Function descriptions	configure TIMER primary output function
Precondition	-
The called functions	-
Input parameter{in}	
timer_periph	TIMER peripheral
<i>TIMERx(x=0,7)</i>	TIMER peripheral selection
Input parameter{in}	
newvalue	control value
<i>ENABLE</i>	enable function
<i>DISABLE</i>	disable function
Output parameter{out}	
-	-
Return value	

-	-
---	---

Example:

```
/* enable TIMER0 primary output function */
```

```
timer_primary_output_config (TIMER0, ENABLE);
```

timer_channel_control_shadow_config

The description of timer_channel_control_shadow_config is shown as below:

Table 3-1059. Function timer_channel_control_shadow_config

Function name	timer_channel_control_shadow_config
Function prototype	void timer_channel_control_shadow_config(uint32_t timer_periph, ControlStatus newvalue);
Function descriptions	channel commutation control shadow register enable
Precondition	-
The called functions	-
Input parameter{in}	
timer_periph	TIMER peripheral
<i>TIMERx(x=0,7)</i>	TIMER peripheral selection
Input parameter{in}	
newvalue	control value
<i>ENABLE</i>	enable function
<i>DISABLE</i>	disable function
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* channel capture/compare control shadow register enable */
```

```
timer_channel_control_shadow_config (TIMER0, ENABLE);
```

timer_channel_control_shadow_update_config

The description of timer_channel_control_shadow_update_config is shown as below:

Table 3-1060. Function timer_channel_control_shadow_update_config

Function name	timer_channel_control_shadow_update_config
Function prototype	void timer_channel_control_shadow_update_config(uint32_t timer_periph, uint8_t ccuctl);
Function descriptions	configure commutation control shadow register update control
Precondition	-
The called functions	-
Input parameter{in}	
timer_periph	TIMER peripheral
<i>TIMERx(x=0,7)</i>	TIMER peripheral selection
Input parameter{in}	
ccuctl	channel control shadow register update control
<i>TIMER_UPDATECTL_CCU</i>	the shadow registers update by when CMTG bit is set
<i>TIMER_UPDATECTL_CUTRI</i>	the shadow registers update by when CMTG bit is set or an rising edge of TRGI occurs
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* configure TIMER0 channel control shadow register update when CMTG bit is set */
timer_channel_control_shadow_update_config (TIMER0, TIMER_UPDATECTL_CCU);
```

timer_channel_output_struct_para_init

The description of timer_channel_output_struct_para_init is shown as below:

Table 3-1061. Function timer_channel_output_struct_para_init

Function name	timer_channel_output_struct_para_init
Function prototype	void timer_channel_output_struct_para_init(timer_oc_parameter_struct* ocpa);
Function descriptions	initialize the parameters of TIMER channel output parameter struct with the

	default values
Precondition	-
The called functions	-
Input parameter{in}	
ocpara	TIMER channel output parameter struct, the structure members can refer to Table 3-1025. Structure timer oc parameter struct .
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* initialize TIMER 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 3-1062. Function timer_channel_output_config

Function name	timer_channel_output_config
Function prototype	void timer_channel_output_config(uint32_t timer_periph, uint16_t channel, timer_oc_parameter_struct* ocpara);
Function descriptions	configure TIMER channel output function
Precondition	-
The called functions	-
Input parameter{in}	
timer_periph	TIMER peripheral
<i>TIMERx</i>	please refer to the following parameters
Input parameter{in}	
channel	channel to be configured
<i>TIMER_CH_0</i>	TIMER channel 0 (TIMERx(x=0..4,7..13))

<i>TIMER_CH_1</i>	TIMER channel 1 (TIMERx(x=0..4,7,8,11))
<i>TIMER_CH_2</i>	TIMER channel 2 (TIMERx(x=0..4,7))
<i>TIMER_CH_3</i>	TIMER channel 3 (TIMERx(x=0..4,7))
Input parameter{in}	
ocpara	TIMER channel output parameter struct, the structure members can refer to Table 3-1025. Structure timer_oc_parameter_struct .
Output parameter{out}	
-	-
Return value	
-	-

Example:

```

/* configure TIMER0 channel 0 output function */

timer_oc_parameter_struct timer_ocinitpara;

timer_ocinitpara.outputstate = TIMER_CCX_ENABLE;

timer_ocinitpara.outputnstate = TIMER_CCXN_ENABLE;

timer_ocinitpara.ocpolarity = TIMER_OC_POLARITY_HIGH;

timer_ocinitpara.ocnpolarity = TIMER_OCN_POLARITY_HIGH;

timer_ocinitpara.ocidlestate = TIMER_OC_IDLE_STATE_HIGH;

timer_ocinitpara.ocnidlestate = TIMER_OCN_IDLE_STATE_LOW;

timer_channel_output_config(TIMER0, TIMER_CH_0, &timer_ocinitpara);

```

timer_channel_output_mode_config

The description of timer_channel_output_mode_config is shown as below:

Table 3-1063. Function timer_channel_output_mode_config

Function name	timer_channel_output_mode_config
Function prototype	void timer_channel_output_mode_config(uint32_t timer_periph, uint16_t channel, uint16_t ocmode);
Function descriptions	configure TIMER channel output compare mode
Precondition	-
The called functions	-

Input parameter{in}	
timer_periph	TIMER peripheral
<i>TIMERx</i>	please refer to the following parameters
Input parameter{in}	
channel	channel to be configured
<i>TIMER_CH_0</i>	TIMER channel 0 (TIMERx(x=0..4,7..13))
<i>TIMER_CH_1</i>	TIMER channel 1 (TIMERx(x=0..4,7,8,11))
<i>TIMER_CH_2</i>	TIMER channel 2 (TIMERx(x=0..4,7))
<i>TIMER_CH_3</i>	TIMER channel 3 (TIMERx(x=0..4,7))
Input parameter{in}	
ocmode	channel output compare mode
<i>TIMER_OC_MODE_TIMING</i>	timing mode
<i>TIMER_OC_MODE_ACTIVE</i>	set the channel output
<i>TIMER_OC_MODE_INACTIVE</i>	clear the channel output
<i>TIMER_OC_MODE_TOGGLE</i>	toggle on match
<i>TIMER_OC_MODE_LOW</i>	force low mode
<i>TIMER_OC_MODE_HIGH</i>	force high mode
<i>TIMER_OC_MODE_PWM0</i>	PWM mode 0
<i>TIMER_OC_MODE_PWM1</i>	PWM mode 1
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* configure TIMER0 channel PWM 0 mode */
```

```
timer_channel_output_mode_config(TIMER0, 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 3-1064. Function timer_channel_output_pulse_value_config

Function name	timer_channel_output_pulse_value_config
Function prototype	void timer_channel_output_pulse_value_config(uint32_t timer_periph, uint16_t channel, uint32_t pulse);
Function descriptions	configure TIMER channel output pulse value
Precondition	-
The called functions	-
Input parameter{in}	
timer_periph	TIMER peripheral
<i>TIMERx</i>	please refer to the following parameters
Input parameter{in}	
channel	channel to be configured
<i>TIMER_CH_0</i>	TIMER channel 0 (TIMERx(x=0..4,7..13))
<i>TIMER_CH_1</i>	TIMER channel 1 (TIMERx(x=0..4,7,8,11))
<i>TIMER_CH_2</i>	TIMER channel 2 (TIMERx(x=0..4,7))
<i>TIMER_CH_3</i>	TIMER channel 3 (TIMERx(x=0..4,7))
Input parameter{in}	
pulse	channel output pulse value (0~65535)
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* configure TIMER0 channel 0 output pulse value */
```

```
timer_channel_output_pulse_value_config(TIMER0, TIMER_CH_0, 399);
```

timer_channel_output_shadow_config

The description of timer_channel_output_shadow_config is shown as below:

Table 3-1065. Function timer_channel_output_shadow_config

Function name	timer_channel_output_shadow_config
Function prototype	void timer_channel_output_shadow_config(uint32_t timer_periph, uint16_t channel, uint16_t ocshadow);
Function descriptions	configure TIMER channel output shadow function
Precondition	-
The called functions	-
Input parameter{in}	
timer_periph	TIMER peripheral
<i>TIMERx</i>	please refer to the following parameters
Input parameter{in}	
channel	channel to be configured
<i>TIMER_CH_0</i>	TIMER channel 0 (TIMERx(x=0..4,7..13))
<i>TIMER_CH_1</i>	TIMER channel 1 (TIMERx(x=0..4,7,8,11))
<i>TIMER_CH_2</i>	TIMER channel 2 (TIMERx(x=0..4,7))
<i>TIMER_CH_3</i>	TIMER channel 3 (TIMERx(x=0..4,7))
Input parameter{in}	
ocshadow	channel output shadow state
<i>TIMER_OC_SHADOW_ENABLE</i>	channel output shadow state enable
<i>TIMER_OC_SHADOW_DISABLE</i>	channel output shadow state disable
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/*configure TIMER0 channel 0 output shadow function */
```

```
timer_channel_output_shadow_config (TIMER0, 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 3-1066. Function timer_channel_output_fast_config

Function name	timer_channel_output_fast_config
Function prototype	void timer_channel_output_fast_config(uint32_t timer_periph, uint16_t channel, uint16_t ocfast);
Function descriptions	configure TIMER channel output fast function
Precondition	-
The called functions	-
Input parameter{in}	
timer_periph	TIMER peripheral
TIMERx	please refer to the following parameters
Input parameter{in}	
channel	channel to be configured
TIMER_CH_0	TIMER channel 0 (TIMERx(x=0..4,7..13))
TIMER_CH_1	TIMER channel 1 (TIMERx(x=0..4,7,8,11))
TIMER_CH_2	TIMER channel 2 (TIMERx(x=0..4,7))
TIMER_CH_3	TIMER channel 3 (TIMERx(x=0..4,7))
Input parameter{in}	
ocfast	channel output fast function
TIMER_OC_FAST_ENABLE	channel output fast function enable
TIMER_OC_FAST_DISABLE	channel output fast function disable
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* configure TIMER0 channel 0 output fast function */
```

```
timer_channel_output_fast_config (TIMER0, 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 3-1067. Function timer_channel_output_clear_config

Function name	timer_channel_output_clear_config
Function prototype	void timer_channel_output_clear_config(uint32_t timer_periph, uint16_t channel, uint16_t occlear);
Function descriptions	configure TIMER channel output clear function
Precondition	-
The called functions	-
Input parameter{in}	
timer_periph	TIMER periphera
<i>TIMERx</i>	please refer to the following parameters
Input parameter{in}	
channel	channel to be configured
<i>TIMER_CH_0</i>	TIMER channel 0 (TIMERx(x=0..4,7..13))
<i>TIMER_CH_1</i>	TIMER channel 1 (TIMERx(x=0..4,7,8,11))
<i>TIMER_CH_2</i>	TIMER channel 2 (TIMERx(x=0..4,7))
<i>TIMER_CH_3</i>	TIMER channel 3 (TIMERx(x=0..4,7))
Input parameter{in}	
occlear	channel output clear function
<i>TIMER_OC_CLEAR_ENABLE</i>	channel output clear function enable
<i>TIMER_OC_CLEAR_DISABLE</i>	channel output clear function disable
Output parameter{out}	
-	-
Return value	

-	-
---	---

Example:

```
/* configure TIMER0 channel 0 output clear function */
```

```
timer_channel_output_clear_config (TIMER0, TIMER_CH_0,  
TIMER_OC_CLEAR_ENABLE);
```

timer_channel_output_polarity_config

The description of timer_channel_output_polarity_config is shown as below:

Table 3-1068. Function timer_channel_output_polarity_config

Function name	timer_channel_output_polarity_config
Function prototype	void timer_channel_output_polarity_config(uint32_t timer_periph, uint16_t channel, uint16_t ocpolarity);
Function descriptions	configure TIMER channel output polarity
Precondition	-
The called functions	-
Input parameter{in}	
timer_periph	TIMER peripheral
TIMERx	please refer to the following parameters
Input parameter{in}	
channel	channel to be configured
TIMER_CH_0	TIMER channel 0 (TIMERx(x=0..4,7..13))
TIMER_CH_1	TIMER channel 1 (TIMERx(x=0..4,7,8,11))
TIMER_CH_2	TIMER channel 2 (TIMERx(x=0..4,7))
TIMER_CH_3	TIMER channel 3 (TIMERx(x=0..4,7))
Input parameter{in}	
ocpolarity	channel output polarity
TIMER_OC_POLARITY_HIGH	channel output polarity is high
TIMER_OC_POLARITY_LOW	channel output polarity is low
Output parameter{out}	

-	-
Return value	
-	-

Example:

```
/* configure TIMER0 channel 0 output polarity */
```

```
timer_channel_output_polarity_config (TIMER0, TIMER_CH_0,  
TIMER_OC_POLARITY_HIGH);
```

timer_channel_complementary_output_polarity_config

The description of timer_channel_complementary_output_polarity_config is shown as below:

Table 3-1069. Function timer_channel_complementary_output_polarity_config

Function name	timer_channel_complementary_output_polarity_config
Function prototype	void timer_channel_complementary_output_polarity_config(uint32_t timer_periph, uint16_t channel, uint16_t ocnpolarity);
Function descriptions	configure TIMER channel complementary output polarity
Precondition	-
The called functions	-
Input parameter{in}	
timer_periph	TIMER peripheral
<i>TIMERx(x=0,7)</i>	TIMER peripheral selection
Input parameter{in}	
channel	channel to be configured
<i>TIMER_CH_0</i>	TIMER channel 0
<i>TIMER_CH_1</i>	TIMER channel 1
<i>TIMER_CH_2</i>	TIMER channel 2
<i>TIMER_CH_3</i>	TIMER channel 3
Input parameter{in}	
ocnpolarity	channel complementary output polarity
<i>TIMER_OCN_POLARITY_HIGH</i>	channel complementary output polarity is high

<i>TIMER_OCN_POLARITY_LOW</i>	channel complementary output polarity is low
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* configure TIMER0 channel 0 complementary output polarity */
```

```
timer_channel_complementary_output_polarity_config (TIMER0, TIMER_CH_0,  
TIMER_OCN_POLARITY_HIGH);
```

timer_channel_output_state_config

The description of timer_channel_output_state_config is shown as below:

Table 3-1070. Function timer_channel_output_state_config

Function name	timer_channel_output_state_config
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}	
timer_periph	TIMER peripheral
<i>TIMERx</i>	please refer to the following parameters
Input parameter{in}	
channel	channel to be configured
<i>TIMER_CH_0</i>	TIMER channel 0 (TIMERx(x=0..4,7..13))
<i>TIMER_CH_1</i>	TIMER channel 1 (TIMERx(x=0..4,7,8,11))
<i>TIMER_CH_2</i>	TIMER channel 2 (TIMERx(x=0..4,7))
<i>TIMER_CH_3</i>	TIMER channel 3 (TIMERx(x=0..4,7))
Input parameter{in}	

state	TIMER channel enable state
<i>TIMER_CCX_ENABLE</i>	channel enable
<i>TIMER_CCX_DISABLE</i>	channel disable
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* configure TIMER0 channel 0 enable state */
```

```
timer_channel_output_state_config (TIMER0, TIMER_CH_0, TIMER_CCX_ENABLE);
```

timer_channel_complementary_output_state_config

The description of timer_channel_complementary_output_state_config is shown as below:

Table 3-1071. Function timer_channel_complementary_output_state_config

Function name	timer_channel_complementary_output_state_config
Function prototype	void timer_channel_complementary_output_state_config(uint32_t timer_periph, uint16_t channel, uint16_t ocnstate);
Function descriptions	configure TIMER channel complementary output enable state
Precondition	-
The called functions	-
Input parameter{in}	
timer_periph	TIMER peripheral
<i>TIMERx(x=0,7)</i>	TIMER peripheral selection
Input parameter{in}	
channel	channel to be configured
<i>TIMER_CH_0</i>	TIMER channel 0
<i>TIMER_CH_1</i>	TIMER channel 1
<i>TIMER_CH_2</i>	TIMER channel 2
<i>TIMER_CH_3</i>	TIMER channel 3
Input parameter{in}	

state	TIMER channel complementary output enable state
<i>TIMER_CCXN_ENABLE</i>	channel complementary enable
<i>TIMER_CCXN_DISABLE</i>	channel complementary disable
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* configure TIMER0 channel 0 complementary output enable state */
timer_channel_complementary_output_state_config (TIMER0, TIMER_CH_0,
TIMER_CCXN_ENABLE);
```

timer_channel_input_struct_para_init

The description of timer_channel_input_struct_para_init is shown as below:

Table 3-1072. Function timer_channel_input_struct_para_init

Function name	timer_channel_input_struct_para_init
Function prototype	void timer_channel_input_struct_para_init(timer_ic_parameter_struct* icpara);
Function descriptions	initialize the parameters of TIMER channel input parameter struct with the default values
Precondition	-
The called functions	-
Input parameter{in}	
icpara	TIMER channel input parameter struct, the structure members can refer to Table 3-1026. Structure timer ic parameter struct .
Output parameter{out}	
-	-
Return value	
-	-

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 3-1073. Function timer_input_capture_config

Function name	timer_input_capture_config
Function prototype	void timer_input_capture_config(uint32_t timer_periph, uint16_t channel, timer_ic_parameter_struct* icpara);
Function descriptions	configure TIMER input capture parameter
Precondition	-
The called functions	timer_channel_input_capture_prescaler_config
Input parameter{in}	
timer_periph	TIMER peripheral
<i>TIMERx</i>	please refer to the following parameters
Input parameter{in}	
channel	channel to be configured
<i>TIMER_CH_0</i>	TIMER channel 0 (TIMERx(x=0..4,7..13))
<i>TIMER_CH_1</i>	TIMER channel 1 (TIMERx(x=0..4,7,8,11))
<i>TIMER_CH_2</i>	TIMER channel 2 (TIMERx(x=0..4,7))
<i>TIMER_CH_3</i>	TIMER channel 3 (TIMERx(x=0..4,7))
Input parameter{in}	
icpara	TIMER channel input parameter struct, the structure members can refer to Table 3-1026. Structure timer_ic_parameter_struct .
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* configure TIMER0 input capture parameter */
```

```

timer_ic_parameter_struct timer_icinitpara;

timer_icinitpara.icpolarity = TIMER_IC_POLARITY_RISING;

timer_icinitpara.icselection = TIMER_IC_SELECTION_DIRECTTTI;

timer_icinitpara.icprescaler = TIMER_IC_PSC_DIV1;

timer_icinitpara.icfilter = 0x0;

timer_input_capture_config(TIMER0, TIMER_CH_0, &timer_icinitpara);

```

timer_channel_input_capture_prescaler_config

The description of timer_channel_input_capture_prescaler_config is shown as below:

Table 3-1074. Function timer_channel_input_capture_prescaler_config

Function name	timer_channel_input_capture_prescaler_config
Function prototype	void timer_channel_input_capture_prescaler_config(uint32_t timer_periph, uint16_t channel, uint16_t prescaler);
Function descriptions	configure TIMER channel input capture prescaler value
Precondition	-
The called functions	-
Input parameter{in}	
timer_periph	TIMER peripheral
<i>TIMERx</i>	please refer to the following parameters
Input parameter{in}	
channel	channel to be configured
<i>TIMER_CH_0</i>	TIMER channel 0 (TIMERx(x=0..4,7..13))
<i>TIMER_CH_1</i>	TIMER channel 1 (TIMERx(x=0..4,7,8,11))
<i>TIMER_CH_2</i>	TIMER channel 2 (TIMERx(x=0..4,7))
<i>TIMER_CH_3</i>	TIMER channel 3 (TIMERx(x=0..4,7))
Input parameter{in}	
prescaler	channel input capture prescaler value
<i>TIMER_IC_PSC_DIV1</i>	no prescaler
<i>TIMER_IC_PSC_DIV2</i>	divided by 2
<i>TIMER_IC_PSC_DIV4</i>	divided by 4

<i>TIMER_IC_PSC_DIV8</i>	divided by 8
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* configure TIMER0 channel 0 input capture prescaler value */
```

```
timer_channel_input_capture_prescaler_config (TIMER0, 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 3-1075. Function timer_channel_capture_value_register_read

Function name	timer_channel_capture_value_register_read
Function prototype	uint32_t timer_channel_capture_value_register_read(uint32_t timer_periph, uint16_t channel);
Function descriptions	read TIMER channel capture compare register value
Precondition	-
The called functions	-
Input parameter{in}	
timer_periph	TIMER peripheral
<i>TIMERx</i>	please refer to the following parameters
Input parameter{in}	
channel	channel to be configured
<i>TIMER_CH_0</i>	TIMER channel 0 (TIMERx(x=0..4,7..13))
<i>TIMER_CH_1</i>	TIMER channel 1 (TIMERx(x=0..4,7,8,11))
<i>TIMER_CH_2</i>	TIMER channel 2 (TIMERx(x=0..4,7))
<i>TIMER_CH_3</i>	TIMER channel 3 (TIMERx(x=0..4,7))
Output parameter{out}	
-	-

Return value	
uint32_t	channel capture compare register value (0x0000~0xFFFF)

Example:

```
/* read TIMER0 channel 0 capture compare register value */
```

```
uint32_t ch0_value = 0;
```

```
ch0_value = timer_channel_capture_value_register_read (TIMER0, TIMER_CH_0);
```

timer_input_pwm_capture_config

The description of timer_input_pwm_capture_config is shown as below:

Table 3-1076. Function timer_input_pwm_capture_config

Function name	timer_input_pwm_capture_config
Function prototype	void timer_input_pwm_capture_config(uint32_t timer_periph, uint16_t channel, timer_ic_parameter_struct* icpwm);
Function descriptions	configure TIMER input pwm capture function
Precondition	-
The called functions	timer_channel_input_capture_prescaler_config
Input parameter{in}	
timer_periph	TIMER peripheral
TIMERx(x=0..4,7,8,11)	TIMER peripheral selection
Input parameter{in}	
channel	channel to be configured
TIMER_CH_0	TIMER channel 0
TIMER_CH_1	TIMER channel 1
Input parameter{in}	
icpwm	TIMER channel input pwm parameter struct, the structure members can refer to Table 3-1026. Structure timer_ic_parameter_struct .
Output parameter{out}	
-	-
Return value	
-	-

Example:

```

/* configure TIMER0 input pwm capture parameter */

timer_ic_parameter_struct timer_icinitpara;

timer_icinitpara.icpolarity = TIMER_IC_POLARITY_RISING;

timer_icinitpara.icselection = TIMER_IC_SELECTION_DIRECTTTI;

timer_icinitpara.icprescaler = TIMER_IC_PSC_DIV1;

timer_icinitpara.icfilter = 0x0;

timer_input_pwm_capture_config (TIMER0, TIMER_CH_0, &timer_icinitpara);

```

timer_hall_mode_config

The description of timer_hall_mode_config is shown as below:

Table 3-1077. Function timer_hall_mode_config

Function name	timer_hall_mode_config
Function prototype	void timer_hall_mode_config(uint32_t timer_periph, uint8_t hallmode);
Function descriptions	configure TIMER hall sensor mode
Precondition	-
The called functions	-
Input parameter{in}	
timer_periph	TIMER peripheral
TIMERx(x=0..4,7)	TIMER peripheral selection
Input parameter{in}	
hallmode	TIMER hall sensor mode state
TIMER_HALLINTERFA CE_ENABLE	TIMER hall sensor mode enable
TIMER_HALLINTERFA CE_DISABLE	TIMER hall sensor mode disable
Output parameter{out}	
-	-
Return value	
-	-

Example:

```

/* configure TIMER0 hall sensor mode */

```

timer_hall_mode_config (TIMER0, TIMER_HALLINTERFACE_ENABLE);

timer_input_trigger_source_select

The description of timer_input_trigger_source_select is shown as below:

Table 3-1078. Function timer_input_trigger_source_select

Function name	timer_input_trigger_source_select
Function prototype	void timer_input_trigger_source_select(uint32_t timer_periph, uint32_t intrigger);
Function descriptions	select TIMER input trigger source
Precondition	SMC[2:0] = 000
The called functions	-
Input parameter{in}	
timer_periph	TIMER peripheral
<i>TIMERx</i>	please refer to the following parameters
Input parameter{in}	
intrigger	trigger selection
<i>TIMER_SMCFG_TRGS_EL_ITI0</i>	Internal trigger input 0 (ITI0, TIMERx(x=0..4,7,8,11))
<i>TIMER_SMCFG_TRGS_EL_ITI1</i>	Internal trigger input 0 (ITI1, TIMERx(x=0..4,7,8,11))
<i>TIMER_SMCFG_TRGS_EL_ITI2</i>	Internal trigger input 0 (ITI2, TIMERx(x=0..4,7,8,11))
<i>TIMER_SMCFG_TRGS_EL_ITI3</i>	Internal trigger input 0 (ITI3, TIMERx(x=0..4,7,8,11))
<i>TIMER_SMCFG_TRGS_EL_CIOF_ED</i>	CIO edge flag (CIOF_ED, TIMERx(x=0..4,7,8,11))
<i>TIMER_SMCFG_TRGS_EL_CIOFE0</i>	channel 0 input Filtered output (CIOFE0, TIMERx(x=0..4,7,8,11))
<i>TIMER_SMCFG_TRGS_EL_C1FE1</i>	channel 1 input Filtered output (C1FE1, TIMERx(x=0..4,7,8,11))
<i>TIMER_SMCFG_TRGS_EL_ETIFP</i>	External trigger input filter output(ETIFP, TIMERx(x=0..4,7))
Output parameter{out}	

-	-
Return value	
-	-

Example:

```
/* select TIMER0 input trigger source */
```

```
timer_input_trigger_source_select (TIMER0, TIMER_SMCFG_TRGSEL_ITI0);
```

timer_master_output_trigger_source_select

The description of timer_master_output_trigger_source_select is shown as below:

Table 3-1079. Function timer_master_output_trigger_source_select

Function name	timer_master_output_trigger_source_select
Function prototype	void timer_master_output_trigger_source_select(uint32_t timer_periph, uint32_t outrigger);
Function descriptions	select TIMER master mode output trigger source
Precondition	-
The called functions	-
Input parameter{in}	
timer_periph	TIMER peripheral
<i>TIMERx(x=0,7)</i>	TIMER peripheral selection
Input parameter{in}	
outrigger	master mode control
<i>TIMER_TRI_OUT_SRC_RESET</i>	Reset. When the UPG bit in the <i>TIMERx_SWEVG</i> 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
<i>TIMER_TRI_OUT_SRC_ENABLE</i>	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.
<i>TIMER_TRI_OUT_SRC_UPDATE</i>	Update. In this mode the master mode controller selects the update event as TRGO.

<i>TIMER_TRI_OUT_SRC_CC0</i>	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.
<i>TIMER_TRI_OUT_SRC_O0CPRE</i>	Compare. In this mode the master mode controller selects the O0CPRE signal is used as TRGO.
<i>TIMER_TRI_OUT_SRC_O1CPRE</i>	Compare. In this mode the master mode controller selects the O1CPRE signal is used as TRGO.
<i>TIMER_TRI_OUT_SRC_O2CPRE</i>	Compare. In this mode the master mode controller selects the O2CPRE signal is used as TRGO.
<i>TIMER_TRI_OUT_SRC_O3CPRE</i>	Compare. In this mode the master mode controller selects the O3CPRE signal is used as TRGO.
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* select TIMER0 master mode output trigger source */
```

```
timer_master_output_trigger_source_select (TIMER0, TIMER_TRI_OUT_SRC_RESET);
```

timer_slave_mode_select

The description of timer_slave_mode_select is shown as below:

Table 3-1080. Function timer_slave_mode_select

Function name	timer_slave_mode_select
Function prototype	void timer_slave_mode_select(uint32_t timer_periph, uint32_t slavemode);
Function descriptions	select TIMER slave mode
Precondition	-
The called functions	-
Input parameter{in}	
timer_periph	TIMER peripheral
<i>TIMERx(x=0..4,7,8,11)</i>	TIMER peripheral selection
Input parameter{in}	
slavemode	slave mode

<i>TIMER_SLAVE_MODE_DISABLE</i>	slave mode disable
<i>TIMER_QUAD_DECODER_MODE0</i>	quadrature decoder mode 0
<i>TIMER_QUAD_DECODER_MODE1</i>	quadrature decoder mode 1
<i>TIMER_QUAD_DECODER_MODE2</i>	quadrature decoder mode 2
<i>TIMER_SLAVE_MODE_RESTART</i>	restart mode
<i>TIMER_SLAVE_MODE_PAUSE</i>	pause mode
<i>TIMER_SLAVE_MODE_EVENT</i>	event mode
<i>TIMER_SLAVE_MODE_EXTERNAL0</i>	external clock mode 0
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* select TIMER0 slave mode */
```

```
timer_slave_mode_select (TIMER0, TIMER_QUAD_DECODER_MODE0);
```

timer_master_slave_mode_config

The description of timer_master_slave_mode_config is shown as below:

Table 3-1081. Function timer_master_slave_mode_config

Function name	timer_master_slave_mode_config
Function prototype	void timer_master_slave_mode_config(uint32_t timer_periph, uint8_t masterslave);
Function descriptions	configure TIMER master slave mode
Precondition	-
The called functions	-

Input parameter{in}	
timer_periph	TIMER peripheral
<i>TIMERx(x=0..4,7,8,11)</i>	TIMER peripheral selection
Input parameter{in}	
masterslave	master slave mode state
<i>TIMER_MASTER_SLAVE_MODE_ENABLE</i>	master slave mode enable
<i>TIMER_MASTER_SLAVE_MODE_DISABLE</i>	master slave mode disable
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* configure TIMER0 master slave mode */
```

```
timer_master_slave_mode_config (TIMER0, TIMER_MASTER_SLAVE_MODE_ENABLE);
```

timer_external_trigger_config

The description of timer_external_trigger_config is shown as below:

Table 3-1082. Function timer_external_trigger_config

Function name	timer_external_trigger_config
Function prototype	void timer_external_trigger_config(uint32_t timer_periph, uint32_t extprescaler, uint32_t expolarity, uint32_t extfilter);
Function descriptions	configure TIMER external trigger input
Precondition	-
The called functions	-
Input parameter{in}	
timer_periph	TIMER peripheral
<i>TIMERx(x=0..4,7)</i>	TIMER peripheral selection
Input parameter{in}	

extprescaler	external trigger prescaler
<i>TIMER_EXT_TRI_PSC_OFF</i>	no divided
<i>TIMER_EXT_TRI_PSC_DIV2</i>	divided by 2
<i>TIMER_EXT_TRI_PSC_DIV4</i>	divided by 4
<i>TIMER_EXT_TRI_PSC_DIV8</i>	divided by 8
Input parameter{in}	
expolarity	external trigger polarity
<i>TIMER_ETP_FALLING</i>	active low or falling edge active
<i>TIMER_ETP_RISING</i>	active high or rising edge active
Input parameter{in}	
extfilter	external trigger filter control (0~15)
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* configure TIMER0 external trigger input */
```

```
timer_external_trigger_config (TIMER0, 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 3-1083. Function timer_quadrature_decoder_mode_config

Function name	timer_quadrature_decoder_mode_config
Function prototype	void timer_quadrature_decoder_mode_config(uint32_t timer_periph, uint32_t decomode, uint16_t ic0polarity, uint16_t ic1polarity);
Function descriptions	configure TIMER quadrature decoder mode

Precondition	-
The called functions	-
Input parameter{in}	
timer_periph	TIMER peripheral
<i>TIMERx(x=0..4,7)</i>	TIMER peripheral selection
Input parameter{in}	
decomode	quadrature decoder mode
<i>TIMER_QUAD_DECODER_MODE0</i>	counter counts on CI0FE0 edge depending on CI1FE1 level
<i>TIMER_QUAD_DECODER_MODE1</i>	counter counts on CI1FE1 edge depending on CI0FE0 level
<i>TIMER_QUAD_DECODER_MODE2</i>	counter counts on both CI0FE0 and CI1FE1 edges depending on the level of the other input
Input parameter{in}	
ic0polarity	IC0 polarity
<i>TIMER_IC_POLARITY_RISING</i>	capture rising edge
<i>TIMER_IC_POLARITY_FALLING</i>	capture falling edge
Input parameter{in}	
ic1polarity	IC1 polarity
<i>TIMER_IC_POLARITY_RISING</i>	capture rising edge
<i>TIMER_IC_POLARITY_FALLING</i>	capture falling edge
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* configure TIMER0 quadrature decoder mode */
```

timer_quadrature_decoder_mode_config (TIMER0, TIMER_QUAD_DECODER_MODE0, TIMER_IC_POLARITY_RISING, TIMER_IC_POLARITY_RISING);

timer_internal_clock_config

The description of timer_internal_clock_config is shown as below:

Table 3-1084. Function timer_internal_clock_config

Function name	timer_internal_clock_config
Function prototype	void timer_internal_clock_config(uint32_t timer_periph);
Function descriptions	configure TIMER internal clock mode
Precondition	-
The called functions	-
Input parameter{in}	
timer_periph	TIMER peripheral
<i>TIMEx(x=0..4,7,8,11)</i>	TIMER peripheral selection
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* configure TIMER0 internal clock mode */
```

```
timer_internal_clock_config (TIMER0);
```

timer_internal_trigger_as_external_clock_config

The description of timer_internal_trigger_as_external_clock_config is shown as below:

Table 3-1085. Function timer_internal_trigger_as_external_clock_config

Function name	timer_internal_trigger_as_external_clock_config
Function prototype	void timer_internal_trigger_as_external_clock_config(uint32_t timer_periph, uint32_t intrigger);
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
<i>TIMERx(x=0..4,7,8,11)</i>	TIMER peripheral selection
Input parameter{in}	
intrigger	trigger selection
<i>TIMER_SMCFG_TRGS</i> <i>EL_ITI0</i>	Internal trigger input 0 (ITI0)
<i>TIMER_SMCFG_TRGS</i> <i>EL_ITI1</i>	Internal trigger input 0 (ITI1)
<i>TIMER_SMCFG_TRGS</i> <i>EL_ITI2</i>	Internal trigger input 0 (ITI2)
<i>TIMER_SMCFG_TRGS</i> <i>EL_ITI3</i>	Internal trigger input 0 (ITI3)
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* configure TIMER0 the internal trigger ITI0 as external clock input */
```

```
timer_internal_trigger_as_external_clock_config (TIMER0, 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 3-1086. Function timer_external_trigger_as_external_clock_config

Function name	timer_external_trigger_as_external_clock_config
Function prototype	void timer_external_trigger_as_external_clock_config(uint32_t timer_periph, uint32_t extrigger, uint16_t expolarity, uint32_t extfilter);
Function descriptions	configure TIMER the external trigger as external clock input
Precondition	-
The called functions	timer_input_trigger_source_select
Input parameter{in}	

timer_periph	TIMER peripheral
<i>TIMERx</i> (<i>x</i> =0..4,7,8,11)	TIMER peripheral selection
Input parameter{in}	
extrigger	external trigger selection
<i>TIMER_SMCFG_TRGS</i> <i>EL_CIOF_ED</i>	CI0 edge flag (CIOF_ED)
<i>TIMER_SMCFG_TRGS</i> <i>EL_CIOFE0</i>	channel 0 input Filtered output (CIOFE0)
<i>TIMER_SMCFG_TRGS</i> <i>EL_C1FE1</i>	channel 1 input Filtered output (C11FE1)
Input parameter{in}	
expolarity	external trigger polarity
<i>TIMER_IC_POLARITY_</i> <i>RISING</i>	active high or rising edge active
<i>TIMER_IC_POLARITY_</i> <i>FALLING</i>	active low or falling edge active
Input parameter{in}	
extfilter	external trigger filter control (0~15)
Output parameter{out}	
-	-
Return value	
-	-

Example:

```

/* configure TIMER0 the external trigger CIOFE0 as external clock input */
timer_external_trigger_as_external_clock_config (TIMER0,
TIMER_SMCFG_TRGSEL_CIOFE0, TIMER_IC_POLARITY_RISING, 0);

```

timer_external_clock_mode0_config

The description of timer_external_clock_mode0_config is shown as below:

Table 3-1087. Function timer_external_clock_mode0_config

Function name	timer_external_clock_mode0_config
----------------------	-----------------------------------

Function prototype	void timer_external_clock_mode0_config(uint32_t timer_periph, uint32_t extprescaler, uint32_t expolarity, uint32_t extfilter);
Function descriptions	configure TIMER the external clock mode0
Precondition	-
The called functions	timer_external_trigger_config
Input parameter{in}	
timer_periph	TIMER peripheral
<i>TIMERx(x=0..4,7,8,11)</i>	TIMER peripheral selection
Input parameter{in}	
extprescaler	ETI external trigger prescaler
<i>TIMER_EXT_TRI_PSC_OFF</i>	no divided
<i>TIMER_EXT_TRI_PSC_DIV2</i>	divided by 2
<i>TIMER_EXT_TRI_PSC_DIV4</i>	divided by 4
<i>TIMER_EXT_TRI_PSC_DIV8</i>	divided by 8
Input parameter{in}	
expolarity	ETI external trigger polarity
<i>TIMER_ETP_FALLING</i>	active low or falling edge active
<i>TIMER_ETP_RISING</i>	active high or rising edge active
Input parameter{in}	
extfilter	ETI external trigger filter control (0~15)
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* configure TIMER0 the external clock mode0 */
```

```
timer_external_clock_mode0_config (TIMER0, 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 3-1088. Function timer_external_clock_mode1_config

Function name	timer_external_clock_mode1_config
Function prototype	void timer_external_clock_mode1_config(uint32_t timer_periph, uint32_t extprescaler, uint32_t expolarity, uint32_t extfilter);
Function descriptions	configure TIMER the external clock mode1
Precondition	-
The called functions	timer_external_trigger_config
Input parameter{in}	
timer_periph	TIMER peripheral
TIMERx(x=0..4,7)	TIMER peripheral selection
Input parameter{in}	
extprescaler	ETI external trigger prescaler
TIMER_EXT_TRI_PSC_OFF	no divided
TIMER_EXT_TRI_PSC_DIV2	divided by 2
TIMER_EXT_TRI_PSC_DIV4	divided by 4
TIMER_EXT_TRI_PSC_DIV8	divided by 8
Input parameter{in}	
expolarity	ETI external trigger polarity
TIMER_ETP_FALLING	active low or falling edge active
TIMER_ETP_RISING	active high or rising edge active
Input parameter{in}	
extfilter	ETI external trigger filter control (0~15)
Output parameter{out}	

-	-
Return value	
-	-

Example:

```
/* configure TIMER0 the external clock mode1 */
```

```
timer_external_clock_mode1_config (TIMER0, 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 3-1089. Function timer_external_clock_mode1_disable

Function name	timer_external_clock_mode1_disable
Function prototype	void timer_external_clock_mode1_disable(uint32_t timer_periph);
Function descriptions	disable TIMER the external clock mode1
Precondition	-
The called functions	-
Input parameter{in}	
timer_periph	TIMER peripheral
<i>TIMER_x(x=0..4,7)</i>	TIMER peripheral selection
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* disable TIMER0 the external clock mode1 */
```

```
timer_external_clock_mode1_disable (TIMER0);
```

timer_channel_remap_config

The description of timer_channel_remap_config is shown as below:

Table 3-1090. Function timer_channel_remap_config

Function name	timer_channel_remap_config
Function prototype	void timer_channel_remap_config(uint32_t timer_periph, uint32_t remap);
Function descriptions	configure TIMER channel remap function
Precondition	-
The called functions	-
Input parameter{in}	
timer_periph	TIMER peripheral
<i>TIMERx(x=1,4,10)</i>	TIMER peripheral selection
Input parameter{in}	
remap	TIMER remap selection
<i>TIMER1_ITI1_RMP_TIMER7_TRGO</i>	timer1 internal trigger input1 remap to TIMER7_TRGO
<i>TIMER1_ITI1_RMP_ETH_ETHERNET_PTP</i>	timer1 internal trigger input1 remap to ethernet PTP
<i>TIMER1_ITI1_RMP_USB_FS_SOF</i>	timer1 internal trigger input1 remap to USB FS SOF
<i>TIMER1_ITI1_RMP_USB_HS_SOF</i>	timer1 internal trigger input1 remap to USB HS SOF
<i>TIMER4_CI3_RMP_GPIO</i>	timer4 channel 3 input remap to GPIO pin
<i>TIMER4_CI3_RMP_IRC32K</i>	timer4 channel 3 input remap to IRC32K
<i>TIMER4_CI3_RMP_LXTAL</i>	timer4 channel 3 input remap to LXTAL
<i>TIMER4_CI3_RMP_RTC_WAKEUP_INT</i>	timer4 channel 3 input remap to RTC wakeup interrupt
<i>TIMER10_ITI1_RMP_GPIO</i>	internal trigger input1 remap based on GPIO setting
<i>TIMER10_ITI1_RMP_RTC_HXTAL_DIV</i>	timer10 internal trigger input1 remap HXTAL_DIV(clock used for RTC which is HXTAL clock divided by RTCDIV bits in RCU_CFG0 register)
Output parameter{out}	

-	-
Return value	
-	-

Example:

```
/* configure timer1 internal trigger input1 remap to TIMER7_TRGO */
```

```
timer_channel_remap_config (TIMER1,TIMER1_ITI1_RMP_TIMER7_TRGO);
```

timer_write_chxval_register_config

The description of timer_write_chxval_register_config is shown as below:

Table 3-1091. Function timer_write_chxval_register_config

Function name	timer_write_chxval_register_config
Function prototype	void timer_write_chxval_register_config(uint32_t timer_periph, uint16_t ccsel);
Function descriptions	configure TIMER write CHxVAL register selection
Precondition	-
The called functions	-
Input parameter{in}	
timer_periph	TIMER peripheral
<i>TIMERx(x=0..4, 7..13)</i>	TIMER peripheral selection
Input parameter{in}	
ccsel	write CHxVAL register selection
<i>TIMER_CHVSEL_DISABLE</i>	no effect
<i>TIMER_CHVSEL_ENABLE</i>	when write the CHxVAL register, if the write value is same as the CHxVAL value, the write access is ignored
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* configure TIMER0 write CHxVAL register selection */
```

```
timer_write_chxval_register_config(TIMER0, TIMER_CHVSEL_ENABLE);
```

timer_output_value_selection_config

The description of timer_output_value_selection_config is shown as below:

Table 3-1092. Function timer_output_value_selection_config

Function name	timer_output_value_selection_config
Function prototype	void timer_output_value_selection_config(uint32_t timer_periph, uint16_t outsel);
Function descriptions	configure TIMER output value selection
Precondition	-
The called functions	-
Input parameter{in}	
timer_periph	TIMER peripheral
<i>TIMERx (x=0,7)</i>	TIMER peripheral selection
Input parameter{in}	
outsel	output value selection
<i>TIMER_OUTSEL_DISABLE</i>	no effect
<i>TIMER_OUTSEL_ENABLE</i>	if POEN and IOS is 0, the output disabled
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* configure TIMER output value selection */
```

```
timer_output_value_selection_config(TIMER0, TIMER_OUTSEL_ENABLE);
```

timer_channel_composite_pwm_enable

The description of timer_channel_composite_pwm_enable is shown as below:

Table 3-1093. Function timer_channel_composite_pwm_enable

Function name	timer_channel_composite_pwm_enable
Function prototype	void timer_channel_composite_pwm_enable(uint32_t timer_periph, uint32_t channel);
Function descriptions	enable the TIMER composite pwm
Precondition	-
The called functions	-
Input parameter{in}	
timer_periph	TIMER peripheral
<i>TIMERx (x=0,7)</i>	TIMER peripheral selection
Input parameter{in}	
channel	channel to be configured
<i>TIMER_CH_0</i>	TIMER channel 0 (TIMERx(x=0,7))
<i>TIMER_CH_1</i>	TIMER channel 1 (TIMERx(x=0,7))
<i>TIMER_CH_2</i>	TIMER channel 2 (TIMERx(x=0,7))
<i>TIMER_CH_3</i>	TIMER channel 3 (TIMERx(x=0,7))
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* enable the TIMER composite pwm */
```

```
timer_channel_composite_pwm_enable(TIMER0, TIMER_CH_0);
```

timer_channel_composite_pwm_disable

The description of timer_channel_composite_pwm_disable is shown as below:

Table 3-1094. Function timer_channel_composite_pwm_disable

Function name	timer_channel_composite_pwm_disable
Function prototype	void timer_channel_composite_pwm_disable(uint32_t timer_periph, uint32_t channel);

Function descriptions	disable the TIMER composite pwm
Precondition	-
The called functions	-
Input parameter{in}	
timer_periph	TIMER peripheral
<i>TIMERx (x=0,7)</i>	TIMER peripheral selection
Input parameter{in}	
channel	channel to be configured
<i>TIMER_CH_0</i>	TIMER channel 0 (TIMERx(x=0,7))
<i>TIMER_CH_1</i>	TIMER channel 1 (TIMERx(x=0,7))
<i>TIMER_CH_2</i>	TIMER channel 2 (TIMERx(x=0,7))
<i>TIMER_CH_3</i>	TIMER channel 3 (TIMERx(x=0,7))
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* disable the TIMER composite pwm */
```

```
timer_channel_composite_pwm_disable(TIMER0, TIMER_CH_0);
```

timer_channel_additional_compare_value_config

The description of timer_channel_additional_compare_value_config is shown as below:

Table 3-1095. Function timer_channel_additional_compare_value_config

Function name	timer_channel_additional_compare_value_config
Function prototype	void timer_channel_additional_compare_value_config(uint32_t timer_periph, uint16_t channel, uint32_t value);
Function descriptions	configure TIMER channel additional compare value
Precondition	-
The called functions	-
Input parameter{in}	

timer_periph	TIMER peripheral
<i>TIMERx</i> (<i>x</i> =0,7)	TIMER peripheral selection
Input parameter{in}	
channel	channel to be configured
<i>TIMER_CH_0</i>	TIMER channel 0 (TIMERx(<i>x</i> =0,7))
<i>TIMER_CH_1</i>	TIMER channel 1 (TIMERx(<i>x</i> =0,7))
<i>TIMER_CH_2</i>	TIMER channel 2 (TIMERx(<i>x</i> =0,7))
<i>TIMER_CH_3</i>	TIMER channel 3 (TIMERx(<i>x</i> =0,7))
Input parameter{in}	
value	channel additional compare value (0~65535)
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* configure TIMER channel additional compare value */
```

```
timer_channel_additional_compare_value_config (TIMER0, TIMER_CH_1, 399);
```

timer_channel_additional_output_shadow_config

The description of timer_channel_additional_output_shadow_config is shown as below:

Table 3-1096. Function timer_channel_additional_output_shadow_config

Function name	timer_channel_additional_output_shadow_config
Function prototype	void timer_channel_additional_output_shadow_config(uint32_t timer_periph, uint16_t channel, uint16_t ocshadow);
Function descriptions	configure TIMER channel additional output shadow function
Precondition	-
The called functions	-
Input parameter{in}	
timer_periph	TIMER peripheral
<i>TIMERx</i> (<i>x</i> =0,7)	TIMER peripheral selection

Input parameter{in}	
channel	channel to be configured
<i>TIMER_CH_0</i>	TIMER channel 0 (TIMERx(x=0,7))
<i>TIMER_CH_1</i>	TIMER channel 1 (TIMERx(x=0,7))
<i>TIMER_CH_2</i>	TIMER channel 2 (TIMERx(x=0,7))
<i>TIMER_CH_3</i>	TIMER channel 3 (TIMERx(x=0,7))
Input parameter{in}	
ocshadow	channel additional compare output shadow state
<i>TIMER_ADD_SHADOW_ENABLE</i>	channel additional compare output shadow enable
<i>TIMER_ADD_SHADOW_DISABLE</i>	channel additional compare output shadow disable
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* configure TIMER channel additional output shadow function */
```

```
timer_channel_additional_output_shadow_config (TIMER0, TIMER_CH_0,  
TIMER_ADD_SHADOW_ENABLE);
```

timer_channel_additional_compare_value_read

The description of timer_channel_additional_compare_value_read is shown as below:

Table 3-1097. Function timer_channel_additional_compare_value_read

Function name	timer_channel_additional_compare_value_read
Function prototype	uint32_t timer_channel_additional_compare_value_read(uint32_t timer_periph, uint16_t channel);
Function descriptions	read TIMER channel additional compare value
Precondition	-
The called functions	-
Input parameter{in}	

timer_periph	TIMER peripheral
<i>TIMERx</i> (<i>x</i> =0,7)	TIMER peripheral selection
Input parameter{in}	
channel	channel to be configured
<i>TIMER_CH_0</i>	TIMER channel 0 (TIMERx(<i>x</i> =0,7))
<i>TIMER_CH_1</i>	TIMER channel 1 (TIMERx(<i>x</i> =0,7))
<i>TIMER_CH_2</i>	TIMER channel 2 (TIMERx(<i>x</i> =0,7))
<i>TIMER_CH_3</i>	TIMER channel 3 (TIMERx(<i>x</i> =0,7))
Output parameter{out}	
-	-
Return value	
uint32_t	channel additional compare value

Example:

```
/* read TIMER channel additional compare value */
```

```
uint32_t i = 0;
```

```
i = timer_channel_additional_compare_value_read(TIMER0, TIMER_CH_0);
```

timer_flag_get

The description of timer_flag_get is shown as below:

Table 3-1098. Function timer_flag_get

Function name	timer_flag_get
Function prototype	FlagStatus timer_flag_get(uint32_t timer_periph, uint32_t flag);
Function descriptions	get TIMER flags
Precondition	-
The called functions	-
Input parameter{in}	
timer_periph	TIMER peripheral
<i>TIMERx</i>	please refer to the following parameters
Input parameter{in}	

flag	the timer interrupt flags
<i>TIMER_FLAG_UP</i>	update flag, $TIMERx(x=0..13)$
<i>TIMER_FLAG_CH0</i>	channel 0 flag, $TIMERx(x=0..4,7..13)$
<i>TIMER_FLAG_CH1</i>	channel 1 flag, $TIMERx(x=0..4,7,8,11)$
<i>TIMER_FLAG_CH2</i>	channel 2 flag, $TIMERx(x=0..4,7)$
<i>TIMER_FLAG_CH3</i>	channel 3 flag, $TIMERx(x=0..4,7)$
<i>TIMER_FLAG_CMT</i>	channel commutation flag, $TIMERx(x=0,7)$
<i>TIMER_FLAG_TRG</i>	trigger flag, $TIMERx(x=0,7,8,11)$
<i>TIMER_FLAG_BRK</i>	break flag, $TIMERx(x=0,7)$
<i>TIMER_FLAG_CH0O</i>	channel 0 overcapture flag, $TIMERx(x=0..4,7..11)$
<i>TIMER_FLAG_CH1O</i>	channel 1 overcapture flag, $TIMERx(x=0..4,7,8,11)$
<i>TIMER_FLAG_CH2O</i>	channel 2 overcapture flag, $TIMERx(x=0..4,7)$
<i>TIMER_FLAG_CH3O</i>	channel 3 overcapture flag, $TIMERx(x=0..4,7)$
Output parameter{out}	
-	-
Return value	
FlagStatus	SET or RESET

Example:

```
/* get TIMER0 update flags */
```

```
FlagStatus Flag_status = RESET;
```

```
Flag_status = timer_flag_get (TIMER0, TIMER_FLAG_UP);
```

timer_flag_clear

The description of timer_flag_clear is shown as below:

Table 3-1099. Function timer_flag_clear

Function name	timer_flag_clear
Function prototype	void timer_flag_clear(uint32_t timer_periph, uint32_t flag);
Function descriptions	clear TIMER flags
Precondition	-

The called functions	-
Input parameter{in}	
timer_periph	TIMER peripheral
<i>TIMERx</i>	please refer to the following parameters
Input parameter{in}	
flag	the timer interrupt flags
<i>TIMER_FLAG_UP</i>	update flag, <i>TIMERx</i> (<i>x</i> =0..13)
<i>TIMER_FLAG_CH0</i>	channel 0 flag, <i>TIMERx</i> (<i>x</i> =0..4,7..13)
<i>TIMER_FLAG_CH1</i>	channel 1 flag, <i>TIMERx</i> (<i>x</i> =0..4,7,8,11)
<i>TIMER_FLAG_CH2</i>	channel 2 flag, <i>TIMERx</i> (<i>x</i> =0..4,7)
<i>TIMER_FLAG_CH3</i>	channel 3 flag, <i>TIMERx</i> (<i>x</i> =0..4,7)
<i>TIMER_FLAG_CMT</i>	channel commutation flag, <i>TIMERx</i> (<i>x</i> =0,7)
<i>TIMER_FLAG_TRG</i>	trigger flag, <i>TIMERx</i> (<i>x</i> =0,7,8,11)
<i>TIMER_FLAG_BRK</i>	break flag, <i>TIMERx</i> (<i>x</i> =0,7)
<i>TIMER_FLAG_CH0O</i>	channel 0 overcapture flag, <i>TIMERx</i> (<i>x</i> =0..4,7..11)
<i>TIMER_FLAG_CH1O</i>	channel 1 overcapture flag, <i>TIMERx</i> (<i>x</i> =0..4,7,8,11)
<i>TIMER_FLAG_CH2O</i>	channel 2 overcapture flag, <i>TIMERx</i> (<i>x</i> =0..4,7)
<i>TIMER_FLAG_CH3O</i>	channel 3 overcapture flag, <i>TIMERx</i> (<i>x</i> =0..4,7)
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* clear TIMER0 update flags */
```

```
timer_flag_clear (TIMER0, TIMER_FLAG_UP);
```

timer_interrupt_enable

The description of timer_interrupt_enable is shown as below:

Table 3-1100. Function timer_interrupt_enable

Function name	timer_interrupt_enable
Function prototype	void timer_interrupt_enable(uint32_t timer_periph, uint32_t interrupt);
Function descriptions	enable the TIMER interrupt
Precondition	-
The called functions	-
Input parameter{in}	
timer_periph	TIMER peripheral
TIMERx	please refer to the following parameters
Input parameter{in}	
interrupt	timer interrupt enable source
TIMER_INT_UP	update interrupt enable, TIMERx (x=0..13)
TIMER_INT_CH0	channel 0 interrupt enable, TIMERx (x=0..4,7..13)
TIMER_INT_CH1	channel 1 interrupt enable, TIMERx (x=0..4,7,8,11)
TIMER_INT_CH2	channel 2 interrupt enable, TIMERx (x=0..4,7)
TIMER_INT_CH3	channel 3 interrupt enable , TIMERx (x=0..4,7)
TIMER_INT_CMT	commutation interrupt enable, TIMERx (x=0,7)
TIMER_INT_TRG	trigger interrupt enable, TIMERx (x=0..4,7,8,11)
TIMER_INT_BRK	break interrupt enable, TIMERx (x=0,7)
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* enable the TIMER0 update interrupt */
timer_interrupt_enable (TIMER0, TIMER_INT_UP);
```

timer_interrupt_disable

The description of timer_interrupt_disable is shown as below:

Table 3-1101. Function timer_interrupt_disable

Function name	timer_interrupt_disable
Function prototype	void timer_interrupt_disable (uint32_t timer_periph, uint32_t interrupt);
Function descriptions	disable the TIMER interrupt
Precondition	-
The called functions	-
Input parameter{in}	
timer_periph	TIMER peripheral
<i>TIMERx</i>	please refer to the following parameters
Input parameter{in}	
interrupt	timer interrupt disable source
<i>TIMER_INT_UP</i>	update interrupt disable, TIMERx(x=0..13)
<i>TIMER_INT_CH0</i>	channel 0 interrupt disable, TIMERx(x=0..4,7..13)
<i>TIMER_INT_CH1</i>	channel 1 interrupt disable, TIMERx(x=0..4,7,8,11)
<i>TIMER_INT_CH2</i>	channel 2 interrupt disable, TIMERx(x=0..4,7)
<i>TIMER_INT_CH3</i>	channel 3 interrupt disable, TIMERx(x=0..4,7)
<i>TIMER_INT_CMT</i>	commutation interrupt disable, TIMERx(x=0,7)
<i>TIMER_INT_TRG</i>	trigger interrupt disable, TIMERx(x=0..4,7,8,11)
<i>TIMER_INT_BRK</i>	break interrupt disable, TIMERx(x=0,7)
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* disable the TIMER0 update interrupt */
timer_interrupt_disable (TIMER0, TIMER_INT_UP);
```

timer_interrupt_flag_get

The description of timer_interrupt_flag_get is shown as below:

Table 3-1102. Function timer_interrupt_flag_get

Function name	timer_interrupt_flag_get
Function prototype	FlagStatus timer_interrupt_flag_get(uint32_t timer_periph, uint32_t interrupt);
Function descriptions	get timer interrupt flag
Precondition	-
The called functions	-
Input parameter{in}	
timer_periph	TIMER peripheral
<i>TIMERx</i>	please refer to the following parameters
Input parameter{in}	
interrupt	the timer interrupt bits
<i>TIMER_INT_FLAG_UP</i>	update interrupt flag, <i>TIMERx</i> (<i>x</i> =0..13)
<i>TIMER_INT_FLAG_CH0</i>	channel 0 interrupt flag, <i>TIMERx</i> (<i>x</i> =0..4,7..13)
<i>TIMER_INT_FLAG_CH1</i>	channel 1 interrupt flag, <i>TIMERx</i> (<i>x</i> =0..4,7,8,11)
<i>TIMER_INT_FLAG_CH2</i>	channel 2 interrupt flag, <i>TIMERx</i> (<i>x</i> =0..4,7)
<i>TIMER_INT_FLAG_CH3</i>	channel 3 interrupt flag, <i>TIMERx</i> (<i>x</i> =0..4,7)
<i>TIMER_INT_FLAG_CMT</i>	channel commutation interrupt flag, <i>TIMERx</i> (<i>x</i> =0,7)
<i>TIMER_INT_FLAG_TRG</i>	trigger interrupt flag, <i>TIMERx</i> (<i>x</i> =0,7,8,11)
<i>TIMER_INT_FLAG_BRK</i>	break interrupt flag, <i>TIMERx</i> (<i>x</i> =0,7)
Output parameter{out}	
-	-
Return value	
FlagStatus	SET or RESET

Example:

```
/* get TIMERO0 update interrupt flag */
```

```
FlagStatus Flag_interrupt = RESET;
```

```
Flag_interrupt = timer_interrupt_flag_get (TIMERO0, TIMER_INT_FLAG_UP);
```

timer_interrupt_flag_clear

The description of timer_interrupt_flag_clear is shown as below:

Table 3-1103. Function timer_interrupt_flag_clear

Function name	timer_interrupt_flag_clear
Function prototype	void timer_interrupt_flag_clear(uint32_t timer_periph, uint32_t interrupt);
Function descriptions	clear TIMER interrupt flag
Precondition	-
The called functions	-
Input parameter{in}	
timer_periph	TIMER peripheral
<i>TIMERx</i>	please refer to the following parameters
Input parameter{in}	
interrupt	the timer interrupt bits
<i>TIMER_INT_FLAG_UP</i>	update interrupt flag, <i>TIMERx</i> (<i>x</i> =0..13)
<i>TIMER_INT_FLAG_CH0</i>	channel 0 interrupt flag, <i>TIMERx</i> (<i>x</i> =0..4,7..13)
<i>TIMER_INT_FLAG_CH1</i>	channel 1 interrupt flag, <i>TIMERx</i> (<i>x</i> =0..4,7,8,11)
<i>TIMER_INT_FLAG_CH2</i>	channel 2 interrupt flag, <i>TIMERx</i> (<i>x</i> =0..4,7)
<i>TIMER_INT_FLAG_CH3</i>	channel 3 interrupt flag, <i>TIMERx</i> (<i>x</i> =0..4,7)
<i>TIMER_INT_FLAG_CMT</i>	channel commutation interrupt flag, <i>TIMERx</i> (<i>x</i> =0,7)
<i>TIMER_INT_FLAG_TRG</i>	trigger interrupt flag, <i>TIMERx</i> (<i>x</i> =0,7,8,11)
<i>TIMER_INT_FLAG_BRK</i>	break interrupt flag, <i>TIMERx</i> (<i>x</i> =0,7)
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* clear TIMER0 update interrupt flag */
```

```
timer_interrupt_flag_clear (TIMER0, TIMER_INT_FLAG_UP);
```

3.30. TLI

The TLI (TFT-LCD Interface) module handles the synchronous LCD interface and provides pixel data, clock and timing signals for passive LCD display. The TLI registers are listed in chapter [3.30.1](#), the TLI firmware functions are introduced in chapter [3.30.2](#).

3.30.1. Descriptions of Peripheral registers

TLI registers are listed in the table shown as below:

Table 3-1104. TLI Registers

Registers	Descriptions
TLI_SPSZ	TLI synchronous pulse size register
TLI_BPSZ	TLI back-porch size register
TLI_ASZ	TLI active size register
TLI_TSZ	TLI total size register
TLI_CTL	TLI control register
TLI_RL	TLI reload Layer register
TLI_BGC	TLI background color register
TLI_INTEN	TLI interrupt enable register
TLI_INTF	TLI interrupt flag register
TLI_INTC	TLI interrupt flag clear register
TLI_LM	TLI line mark register
TLI_CPPOS	TLI current pixel position register
TLI_STAT	TLI status register
TLI_LxCTL	TLI layer x control register
TLI_LxHPOS	TLI layer x horizontal position parameters register
TLI_LxVPOS	TLI layer x vertical position parameters register
TLI_LxCKEY	TLI layer x color key register
TLI_LxPPF	TLI layer x packeted pixel format register
TLI_LxSA	TLI layer x specified alpha register
TLI_LxDC	TLI layer x default color register
TLI_LxBLEND	TLI layer x blending register
TLI_LxFBADDR	TLI layer x frame base address register
TLI_LxFLLN	TLI layer x frame line length register
TLI_LxFTLN	TLI layer x frame total line number register
TLI_LxLUT	TLI layer x Look Up Table register

3.30.2. Descriptions of Peripheral functions

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

Table 3-1105. TLI firmware function

Function name	Function description
tli_deinit	deinitialize TLI registers
tli_struct_para_init	initialize the parameters of TLI parameter structure with the default values, it is suggested that call this function after a tli_parameter_struct structure is defined
tli_init	initialize TLI
tli_dither_config	configure TLI dither function
tli_enable	enable TLI
tli_disable	disable TLI
tli_reload_config	configure TLI reload mode
tli_layer_struct_para_init	initialize the parameters of TLI layer structure with the default values, it is suggested that call this function after a tli_layer_parameter_struct structure is defined
tli_layer_init	initialize TLI layer
tli_layer_window_offset_modify	reconfigure window position
tli_lut_struct_para_init	initialize the parameters of TLI layer LUT structure with the default values, it is suggested that call this function after a tli_layer_lut_parameter_struct structure is defined
tli_lut_init	initialize TLI layer LUT
tli_color_key_init	initialize TLI layer color key
tli_layer_enable	enable TLI layer
tli_layer_disable	disable TLI layer
tli_color_key_enable	enable TLI layer color keying
tli_color_key_disable	disable TLI layer color keying
tli_lut_enable	enable TLI layer LUT
tli_lut_disable	disable TLI layer LUT
tli_line_mark_set	set line mark value
tli_current_pos_get	get current displayed position
tli_interrupt_enable	enable TLI interrupt
tli_interrupt_disable	disable TLI interrupt
tli_interrupt_flag_get	get TLI interrupt flag
tli_interrupt_flag_clear	clear TLI interrupt flag
tli_flag_get	get TLI flag or state in TLI_INTF register or TLI_STAT register

Structure tli_parameter_struct

Table 3-1106. Structure tli_parameter_struct

Member name	Function description
synpsz_vpsz	size of the vertical synchronous pulse
synpsz_hpsz	size of the horizontal synchronous pulse
backpsz_vbpsz	size of the vertical back porch plus synchronous pulse
backpsz_hbpsz	size of the horizontal back porch plus synchronous pulse

Member name	Function description
activesz_vasz	size of the vertical active area width plus back porch and synchronous pulse
activesz_hasz	size of the horizontal active area width plus back porch and synchronous pulse
totalsz_vtsz	vertical total size of the display
totalsz_htsz	horizontal total size of the display
backcolor_red	background value red
backcolor_green	background value green
backcolor_blue	background value blue
signalpolarity_hs	horizontal pulse polarity selection
signalpolarity_vs	vertical pulse polarity selection
signalpolarity_de	data enable polarity selection
signalpolarity_pixelck	pixel clock polarity selection

Structure tli_layer_parameter_struct

Table 3-1107. Structure tli_layer_parameter_struct

Member name	Function description
layer_window_rightpos	window right position
layer_window_leftpos	window left position
layer_window_bottompos	window bottom position
layer_window_toppos	window top position
layer_ppf	packeted pixel format
layer_sa	specified alpha
layer_default_alpha	the default color alpha
layer_default_red	the default color red
layer_default_green	the default color green
layer_default_blue	the default color blue
layer_acf1	alpha calculation factor 1 of blending method
layer_acf2	alpha calculation factor 2 of blending method
layer_frame_bufaddr	frame buffer base address
layer_frame_buf_stride_offset	frame buffer stride offset
layer_frame_line_length	frame line length
layer_frame_total_line_number	frame total line number

Structure tli_layer_lut_parameter_struct

Table 3-1108. Structure tli_layer_lut_parameter_struct

Member name	Function description
layer_table_addr	look up table write address
layer_lut_channel_red	red channel of a LUT entry
layer_lut_channel_green	green channel of a LUT entry
layer_lut_channel_blue	blue channel of a LUT entry

tli_deinit

The description of tli_deinit is shown as below:

Table 3-1109. Function tli_deinit

Function name	tli_deinit
Function prototype	void tli_deinit(void);
Function descriptions	deinitialize TLI registers
Precondition	-
The called functions	-
Input parameter{in}	
-	-
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* deinitialize TLI */
tli_deinit();
```

tli_struct_para_init

The description of tli_struct_para_init is shown as below:

Table 3-1110. Function tli_struct_para_init

Function name	tli_struct_para_init
Function prototype	void tli_struct_para_init(tli_parameter_struct *tli_struct);
Function descriptions	initialize the parameters of TLI parameter structure with the default values, it is suggested that call this function after a tli_parameter_struct structure is defined
Precondition	-
The called functions	-
Input parameter{in}	
*tli_struct	a pointer to tli_parameter_struct
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
tli_parameter_struct tli_init_struct;

/* initialize the parameters of TLI parameter structure with the default values */
tli_struct_para_init(&tli_init_struct);
```

tli_init

The description of tli_init is shown as below:

Table 3-1111. Function tli_init

Function name	tli_init
Function prototype	void tli_init(tli_parameter_struct *tli_struct);
Function descriptions	initialize TLI display timing parameters
Precondition	-
The called functions	-
Input parameter{in}	
*tli_struct	a pointer to tli_parameter_struct
Output parameter{out}	
-	-
Return value	
-	-

Example:

```

tli_parameter_struct tli_init_struct;

/* initialize the parameters of TLI parameter structure with the default values */
tli_struct_para_init(&tli_init_struct);

/* configure TLI parameter struct */
tli_init_struct.signalpolarity_hs = TLI_HSYN_ACTLIVE_LOW;
tli_init_struct.signalpolarity_vs = TLI_VSYN_ACTLIVE_LOW;
tli_init_struct.signalpolarity_de = TLI_DE_ACTLIVE_LOW;
tli_init_struct.signalpolarity_pixelck = TLI_PIXEL_CLOCK_TLI;

/* LCD display timing configuration */
tli_init_struct.synpsz_hpsz = 40;
tli_init_struct.synpsz_vpsz = 9;
tli_init_struct.backpsz_hbpsz = 42;
tli_init_struct.backpsz_vbpsz = 11;
tli_init_struct.activesz_hasz = 522;
tli_init_struct.activesz_vasz = 283;
tli_init_struct.totalsz_htsz = 524;
tli_init_struct.totalsz_vtsz = 285;

/* configure LCD background R,G,B values */

```

```

tli_init_struct.backcolor_red = 0xFF;

tli_init_struct.backcolor_green = 0xFF;

tli_init_struct.backcolor_blue = 0xFF;

tli_init(&tli_init_struct);

```

tli_dither_config

The description of tli_dither_config is shown as below:

Table 3-1112. Function tli_dither_config

Function name	tli_dither_config
Function prototype	void tli_dither_config(uint8_t dither_stat);
Function descriptions	configure TLI dither function
Precondition	-
The called functions	-
Input parameter{in}	
dither_stat	dither function
<i>TLI_DITHER_ENABLE</i>	enable TLI dither function
<i>TLI_DITHER_DISABLE</i>	disable TLI dither function
Output parameter{out}	
-	-
Return value	
-	-

Example:

```

/* enable TLI dither function */

tli_dither_config(TLI_DITHER_ENABLE);

```

tli_enable

The description of tli_enable is shown as below:

Table 3-1113. Function tli_enable

Function name	tli_enable
Function prototype	void tli_enable(void);
Function descriptions	enable TLI
Precondition	-
The called functions	-
Input parameter{in}	
-	-
Output parameter{out}	
-	-
Return value	

-	-
---	---

Example:

```
/* enabled TLI */
tli_enable();
```

tli_disable

The description of tli_disable is shown as below:

Table 3-1114. Function tli_disable

Function name	tli_disable
Function prototype	void tli_disable(void);
Function descriptions	disable TLI
Precondition	-
The called functions	-
Input parameter{in}	
-	-
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* disable TLI */
tli_disable();
```

tli_reload_config

The description of tli_reload_config is shown as below:

Table 3-1115. Function tli_reload_config

Function name	tli_reload_config
Function prototype	void tli_reload_config(uint8_t reload_mod);
Function descriptions	configure TLI reload mode
Precondition	-
The called functions	-
Input parameter{in}	
reload_mod	TLI reload mode
<i>TLI_FRAME_BLANK_RELOAD_EN</i>	the layer configuration will be reloaded at frame blank
<i>TLI_REQUEST_RELOAD_EN</i>	the layer configuration will be reloaded after this bit sets
Output parameter{out}	

-	-
Return value	
-	-

Example:

```
/* configure TLI reload mode */
```

```
tli_reload_config(TLI_REQUEST_RELOAD_EN);
```

tli_layer_struct_para_init

The description of tli_layer_struct_para_init is shown as below:

Table 3-1116. Function tli_layer_struct_para_init

Function name	tli_layer_struct_para_init
Function prototype	void tli_layer_struct_para_init(tli_layer_parameter_struct *layer_struct);
Function descriptions	initialize the parameters of TLI layer structure with the default values, it is suggested that call this function after a tli_layer_parameter_struct structure is defined
Precondition	-
The called functions	-
Input parameter{in}	
*layer_struct	a pointer to tli_layer_parameter_struct
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
tli_layer_parameter_struct tli_layer_init_struct;
```

```
/* initialize the parameters of TLI layer structure with the default values */
```

```
tli_layer_struct_para_init(&tli_layer_init_struct);
```

tli_layer_init

The description of tli_layer_init is shown as below:

Table 3-1117. Function tli_layer_init

Function name	tli_layer_init
Function prototype	void tli_layer_init(uint32_t layerx, tli_layer_parameter_struct *layer_struct);
Function descriptions	initialize TLI layer
Precondition	-
The called functions	-
Input parameter{in}	
layerx	layer

<i>LAYER0</i>	layer 0
<i>LAYER1</i>	layer 1
Input parameter{in}	
*layer_struct	a pointer to tli_layer_parameter_struct
Output parameter{out}	
-	-
Return value	
-	-

Example:

```

tli_layer_parameter_struct tli_layer_init_struct;

tli_layer_struct_para_init(&tli_layer_init_struct);

/* TLI layer0 configuration */

tli_layer_init_struct.layer_window_leftpos = 20 + 43;

tli_layer_init_struct.layer_window_rightpos = (20 + 440 + 43 - 1);

tli_layer_init_struct.layer_window_toppos = 40 + 12;

tli_layer_init_struct.layer_window_bottompos = (40 + 182 + 12 - 1);

tli_layer_init_struct.layer_ppf = LAYER_PPF_RGB565;

/* TLI window specified alpha configuration */

tli_layer_init_struct.layer_sa = 255;

/* TLI layer default alpha R,G,B value configuration */

tli_layer_init_struct.layer_default_blue = 0xFF;

tli_layer_init_struct.layer_default_green = 0xFF;

tli_layer_init_struct.layer_default_red = 0xFF;

tli_layer_init_struct.layer_default_alpha = 0xFF;

/* TLI window blend configuration */

tli_layer_init_struct.layer_acf1 = LAYER_ACF1_PASA;

tli_layer_init_struct.layer_acf2 = LAYER_ACF2_PASA;

/* TLI layer frame buffer base address configuration */

tli_layer_init_struct.layer_frame_bufaddr = (uint32_t)&gBackground;

tli_layer_init_struct.layer_frame_line_length = ((440 * 2) + 3);

tli_layer_init_struct.layer_frame_buf_stride_offset = (440 * 2);

tli_layer_init_struct.layer_frame_total_line_number = 182;

```

```
tli_layer_init(LAYER0, &tli_layer_init_struct);
```

tli_layer_window_offset_modify

The description of tli_layer_window_offset_modify is shown as below:

Table 3-1118. Function tli_layer_window_offset_modify

Function name	tli_layer_window_offset_modify
Function prototype	void tli_layer_window_offset_modify(uint32_t layerx,uint16_t offset_x,uint16_t offset_y);
Function descriptions	reconfigure window position
Precondition	-
The called functions	-
Input parameter{in}	
layerx	layer
<i>LAYERx</i>	x=0, 1
Input parameter{in}	
offset_x	new horizontal offset
Input parameter{in}	
offset_y	new vertical offset
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* reconfigure LAYER1 window position */
```

```
tli_layer_window_offset_modify(LAYER1, 20, 20);
```

tli_lut_struct_para_init

The description of tli_lut_struct_para_init is shown as below:

Table 3-1119. Function tli_lut_struct_para_init

Function name	tli_lut_struct_para_init
Function prototype	void tli_lut_struct_para_init(tli_layer_lut_parameter_struct *lut_struct);
Function descriptions	initialize the parameters of TLI layer LUT structure with the default values, it is suggested that call this function after a tli_layer_lut_parameter_struct structure is defined
Precondition	-
The called functions	-
Input parameter{in}	
*lut_struct	a pointer to tli_layer_lut_parameter_struct
Output parameter{out}	
-	-

Return value	
-	-

Example:

```
tli_layer_lut_parameter_struct  tli_lut_struct;
```

```
/* initialize the parameters of TLI layer LUT structure with the default values */
```

```
tli_lut_struct_para_init(&tli_lut_struct);
```

tli_lut_init

The description of tli_lut_init is shown as below:

Table 3-1120. Function tli_lut_init

Function name	tli_lut_init
Function prototype	void tli_lut_init(uint32_t layerx,tli_layer_lut_parameter_struct *lut_struct);
Function descriptions	initialize TLI layer LUT
Precondition	-
The called functions	-
Input parameter{in}	
layerx	layer
LAYERx	x=0, 1
Input parameter{in}	
*lut_struct	a pointer to tli_layer_lut_parameter_struct
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
tli_layer_lut_parameter_struct  tli_lut_struct;
```

```
tli_lut_struct_para_init(&tli_lut_struct);
```

```
/* initialize TLI layer0 LUT */
```

```
tli_lut_struct.layer_table_addr = 0x20003000;
```

```
tli_lut_struct.layer_lut_channel_red = 0x20;
```

```
tli_lut_struct.layer_lut_channel_green = 0x30;
```

```
tli_lut_struct.layer_lut_channel_blue = 0xFF;
```

```
tli_lut_init(LAYER0, &tli_lut_struct);
```

tli_color_key_init

The description of tli_color_key_init is shown as below:

Table 3-1121. Function tli_color_key_init

Function name	tli_color_key_init
Function prototype	void tli_color_key_init(uint32_t layerx,uint32_t redkey,uint32_t greenkey,uint32_t bluekey);
Function descriptions	initialize TLI layer color key
Precondition	-
The called functions	-
Input parameter{in}	
layerx	layer
LAYERx	x=0, 1
Input parameter{in}	
redkey	color key red
Input parameter{in}	
greenkey	color key green
Input parameter{in}	
bluekey	color key blue
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* initialize TLI layer0 color key */
tli_color_key_init(LAYER0, 0xAA, 0xFF, 0x00);
```

tli_layer_enable

The description of tli_layer_enable is shown as below:

Table 3-1122. Function tli_layer_enable

Function name	tli_layer_enable
Function prototype	void tli_layer_enable(uint32_t layerx);
Function descriptions	enable TLI layer
Precondition	-
The called functions	-
Input parameter{in}	
layerx	layer
LAYERx	x=0, 1
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* enable TLI layer0 */
```

```
tli_layer_enable(LAYER0);
```

tli_layer_disable

The description of tli_layer_disable is shown as below:

Table 3-1123. Function tli_layer_disable

Function name	tli_layer_disable
Function prototype	void tli_layer_disable(uint32_t layerx);
Function descriptions	disable TLI layer
Precondition	-
The called functions	-
Input parameter{in}	
layerx	layer
<i>LAYERx</i>	x=0, 1
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* disable TLI layer0 */
```

```
tli_layer_disable(LAYER0);
```

tli_color_key_enable

The description of tli_color_key_enable is shown as below:

Table 3-1124. Function tli_color_key_enable

Function name	tli_color_key_enable
Function prototype	void tli_color_key_enable(uint32_t layerx);
Function descriptions	enable TLI layer color keying
Precondition	-
The called functions	-
Input parameter{in}	
layerx	layer
<i>LAYERx</i>	x=0, 1
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* enable TLI layer0 color keying */
```

```
tli_color_key_enable(LAYER0);
```

tli_color_key_disable

The description of tli_color_key_disable is shown as below:

Table 3-1125. Function tli_color_key_disable

Function name	tli_color_key_disable
Function prototype	void tli_color_key_disable(uint32_t layerx);
Function descriptions	disable TLI layer color keying
Precondition	-
The called functions	-
Input parameter{in}	
layerx	layer
<i>LAYERx</i>	x=0, 1
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* disable TLI layer0 color keying */
```

```
tli_color_key_disable(LAYER0);
```

tli_lut_enable

The description of tli_lut_enable is shown as below:

Table 3-1126. Function tli_lut_enable

Function name	tli_lut_enable
Function prototype	void tli_lut_enable(uint32_t layerx);
Function descriptions	enable TLI layer LUT
Precondition	-
The called functions	-
Input parameter{in}	
layerx	layer
<i>LAYERx</i>	x=0, 1
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* enable TLI layer0 LUT */
```

```
tli_lut_enable(LAYER0);
```

tli_lut_disable

The description of tli_lut_disable is shown as below:

Table 3-1127. Function tli_lut_disable

Function name	tli_lut_disable
Function prototype	void tli_lut_disable(uint32_t layerx);
Function descriptions	disable TLI layer0 LUT
Precondition	-
The called functions	-
Input parameter{in}	
layerx	layer
<i>LAYERx</i>	x=0, 1
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* disable TLI layer0 LUT */
```

```
tli_lut_disable(LAYER0);
```

tli_line_mark_set

The description of tli_line_mark_set is shown as below:

Table 3-1128. Function tli_line_mark_set

Function name	tli_line_mark_set
Function prototype	void tli_line_mark_set(uint32_t line_num);
Function descriptions	set line mark value
Precondition	-
The called functions	-
Input parameter{in}	
line_num	line number
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* set line mark value */
```

```
tli_line_mark_set(0x20);
```

tli_current_pos_get

The description of tli_current_pos_get is shown as below:

Table 3-1129. Function tli_current_pos_get

Function name	tli_current_pos_get
Function prototype	uint32_t tli_current_pos_get(void);
Function descriptions	get current displayed position
Precondition	-
The called functions	-
Input parameter{in}	
-	-
Output parameter{out}	
-	-
Return value	
uint32_t	position of current pixel

Example:

```
uint32_t pos;

/* get current pixel position */

pos = tli_current_pos_get();
```

tli_interrupt_enable

The description of tli_interrupt_enable is shown as below:

Table 3-1130. Function tli_interrupt_enable

Function name	tli_interrupt_enable
Function prototype	void tli_interrupt_enable(uint32_t int_flag);
Function descriptions	enable TLI interrupt
Precondition	-
The called functions	-
Input parameter{in}	
int_flag	TLI interrupt flags
TLI_INT_LM	line mark interrupt
TLI_INT_FE	FIFO error interrupt
TLI_INT_TE	transaction error interrupt
TLI_INT_LCR	layer configuration reloaded interrupt
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* enable TLI line mark interrupt */
tli_interrupt_enable(TLI_INT_LM);
```

tli_interrupt_disable

The description of tli_interrupt_disable is shown as below:

Table 3-1131. Function tli_interrupt_disable

Function name	tli_interrupt_disable
Function prototype	void tli_interrupt_disable(uint32_t int_flag);
Function descriptions	disable TLI interrupt
Precondition	-
The called functions	-
Input parameter{in}	
int_flag	TLI interrupt flags
<i>TLI_INT_LM</i>	line mark interrupt
<i>TLI_INT_FE</i>	FIFO error interrupt
<i>TLI_INT_TE</i>	transaction error interrupt
<i>TLI_INT_LCR</i>	layer configuration reloaded interrupt
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* disable TLI line mark interrupt */
tli_interrupt_disable(TLI_INT_LM);
```

tli_interrupt_flag_get

The description of tli_interrupt_flag_get is shown as below:

Table 3-1132. Function tli_interrupt_flag_get

Function name	tli_interrupt_flag_get
Function prototype	FlagStatus tli_interrupt_flag_get(uint32_t int_flag);
Function descriptions	get TLI interrupt flag
Precondition	-
The called functions	-
Input parameter{in}	
int_flag	TLI interrupt flags
<i>TLI_INT_FLAG_LM</i>	line mark interrupt flag
<i>TLI_INT_FLAG_FE</i>	FIFO error interrupt flag
<i>TLI_INT_FLAG_TE</i>	transaction error interrupt flag

<i>TLI_INT_FLAG_LCR</i>	layer configuration reloaded interrupt flag
Output parameter{out}	
-	-
Return value	
FlagStatus	SET or RESET

Example:

```
/* get TLI interrupt flag */
if(SET == tli_interrupt_flag_get(TLI_INT_FLAG_LM)){
    tli_interrupt_flag_clear(TLI_INT_FLAG_LM);
}
```

tli_interrupt_flag_clear

The description of tli_interrupt_flag_clear is shown as below:

Table 3-1133. Function tli_interrupt_flag_clear

Function name	tli_interrupt_flag_clear
Function prototype	void tli_interrupt_flag_clear(uint32_t int_flag);
Function descriptions	clear TLI interrupt flag
Precondition	-
The called functions	-
Input parameter{in}	
int_flag	TLI interrupt flags
<i>TLI_INT_FLAG_LM</i>	line mark interrupt flag
<i>TLI_INT_FLAG_FE</i>	FIFO error interrupt flag
<i>TLI_INT_FLAG_TE</i>	transaction error interrupt flag
<i>TLI_INT_FLAG_LCR</i>	layer configuration reloaded interrupt flag
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
if(SET == tli_interrupt_flag_get(TLI_INT_FLAG_LM)){
    /* clear TLI interrupt flag */
    tli_interrupt_flag_clear(TLI_INT_FLAG_LM);
}
```

tli_flag_get

The description of tli_flag_get is shown as below:

Table 3-1134. Function tli_flag_get

Function name	tli_flag_get
Function prototype	FlagStatus tli_flag_get(uint32_t flag);
Function descriptions	get TLI flag or state in TLI_INTF register or TLI_STAT register
Precondition	-
The called functions	-
Input parameter{in}	
flag	TLI flags or states
TLI_FLAG_VDE	current VDE state
TLI_FLAG_HDE	current HDE state
TLI_FLAG_VS	current VS status of the TLI
TLI_FLAG_HS	current HS status of the TLI
TLI_FLAG_LM	line mark interrupt flag
TLI_FLAG_FE	FIFO error interrupt flag
TLI_FLAG_TE	transaction error interrupt flag
TLI_FLAG_LCR	layer configuration reloaded interrupt flag
Output parameter{out}	
-	-
Return value	
FlagStatus	SET or RESET

Example:

```
/* wait the TLI_FLAG_LM flag set */
while(RESET == tli_flag_get(TLI_FLAG_LM)){
}
```

3.31. TRNG

The true random number generator (TRNG) can generate a 32-bit random value by using continuous analog noise. The TRNG registers are listed in chapter [3.31.1](#), the TRNG firmware functions are introduced in chapter [3.31.2](#).

3.31.1. Descriptions of Peripheral registers

TRNG registers are listed in the table shown as below:

Table 3-1135 TRNG Registers

Registers	Descriptions
TRNG_CTL	TRNG control register
TRNG_STAT	TRNG status register
TRNG_DATA	TRNG data register

3.31.2. Descriptions of Peripheral functions

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

Table 3-1136. TRNG firmware function

Function name	Function description
trng_deinit	deinitialize the TRNG
trng_enable	enable the TRNG interface
trng_disable	disable the TRNG interface
trng_get_true_random_data	get the true random data
trng_flag_get	get the TRNG status flags
trng_interrupt_enable	enable the TRNG interrupt
trng_interrupt_disable	disable the TRNG interrupt
trng_interrupt_flag_get	get the TRNG interrupt flags
trng_interrupt_flag_clear	clear the TRNG interrupt flags

trng_deinit

The description of trng_deinit is shown as below:

Table 3-1137. Function trng_deinit

Function name	trng_deinit
Function prototype	void trng_deinit(void)
Function descriptions	reset TRNG peripheral
Precondition	-
The called functions	rcu_periph_reset_enable / rcu_periph_reset_disable
Input parameter{in}	
-	-
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* reset TRNG */
trng_deinit();
```

trng_enable

The description of trng_enable is shown as below:

Table 3-1138. Function trng_enable

Function name	trng_enable
Function prototype	void trng_enable(void)
Function descriptions	enable the TRNG interface

Precondition	-
The called functions	-
Input parameter{in}	
-	-
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* enable TRNG */
```

```
trng_enable();
```

trng_disable

The description of trng_disable is shown as below:

Table 3-1139 Function trng_disable

Function name	trng_disable
Function prototype	void trng_disable(void)
Function descriptions	disable the TRNG interface
Precondition	-
The called functions	-
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 3-1140 Function trng_get_true_random_data

Function name	trng_get_true_random_data
Function prototype	uint32_t trng_get_true_random_data(void)
Function descriptions	get the true random data
Precondition	-
The called functions	-

Input parameter{in}	
-	-
Output parameter{out}	
-	-
Return value	
uint32_t	the generated random data

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 3-1141 Function trng_flag_get

Function name	trng_flag_get
Function prototype	FlagStatus trng_flag_get(trng_flag_enum flag)
Function descriptions	get the trng status flags
Precondition	-
The called functions	-
Input parameter{in}	
flag	trng status flag
TRNG_FLAG_DRDY	Random Data ready status
TRNG_FLAG_CECS	Clock error current status
TRNG_FLAG_SECS	Seed error current status
Output parameter{out}	
-	-
Return value	
FlagStatus	SET or RESET

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 3-1142 Function trng_interrupt_enable

Function name	trng_interrupt_enable
---------------	-----------------------

Function prototype	void trng_interrupt_enable(void)
Function descriptions	enable the TRNG interrupt
Precondition	-
The called functions	-
Input parameter{in}	
-	-
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* enable TRNG interrupt */
trng_interrupt_enable();
```

trng_interrupt_disable

The description of trng_interrupt_disable is shown as below:

Table 3-1143 Function trng_interrupt_disable

Function name	trng_interrupt_disable
Function prototype	void trng_interrupt_disable(void)
Function descriptions	disable the TRNG interrupt
Precondition	-
The called functions	-
Input parameter{in}	
-	-
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* disable TRNG interrupt */
trng_interrupt_disable();
```

trng_interrupt_flag_get

The description of trng_interrupt_flag_get is shown as below:

Table 3-1144 Function trng_interrupt_flag_get

Function name	trng_interrupt_flag_get
Function prototype	FlagStatus trng_interrupt_flag_get(trng_int_flag_enum int_flag)
Function descriptions	get the trng interrupt flags

Precondition	-
The called functions	-
Input parameter{in}	
int_flag	trng interrupt flag
<i>TRNG_INT_FLAG_CEIF</i> F	clock error interrupt flag
<i>TRNG_INT_FLAG_SEI</i> F	Seed error interrupt flag
Output parameter{out}	
-	-
Return value	
FlagStatus	SET or RESET

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 3-1145 Function trng_interrupt_flag_clear

Function name	trng_interrupt_flag_clear
Function prototype	void trng_interrupt_flag_get(trng_int_flag_enum int_flag)
Function descriptions	clear the trng interrupt flags
Precondition	-
The called functions	-
Input parameter{in}	
int_flag	trng interrupt flag
<i>TRNG_INT_FLAG_CEIF</i> F	clock error interrupt flag
<i>TRNG_INT_FLAG_SEI</i> F	Seed error interrupt flag
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* clear TRNG clock error interrupt flag*/
```

```
trng_interrupt_flag_clear(TRNG_INT_FLAG_CEIF);
```

3.32. USART

The Universal Synchronous/Asynchronous Receiver/Transmitter (USART) provides a flexible serial data exchange interface. The USART registers are listed in chapter [3.32.1](#), the USART firmware functions are introduced in chapter [3.32.2](#).

3.32.1. Descriptions of Peripheral registers

USART registers are listed in the table shown as below:

Table 3-1146. USART Registers

Registers	Descriptions
USART_STAT0	Status register 0
USART_DATA	Data register
USART_BAUD	Baud rate register
USART_CTL0	Control register 0
USART_CTL1	Control register 1
USART_CTL2	Control register 2
USART_GP	Guard time and prescaler register
USART_CTL3	Control register 3
USART_RT	Receiver timeout register
USART_STAT1	Status register 1
USART_CHC	Coherence control register

3.32.2. Descriptions of Peripheral functions

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

Table 3-1147. USART firmware function

Function name	Function description
usart_deinit	reset USART/UART
usart_baudrate_set	configure USART baud rate value
usart_parity_config	configure USART parity
usart_word_length_set	configure USART word length
usart_stop_bit_set	configure USART stop bit length
usart_enable	enable USART
usart_disable	disable USART
usart_transmit_config	configure USART transmitter
usart_receive_config	configure USART receiver
usart_data_first_config	data is transmitted/received with the LSB/MSB first
usart_invert_config	configure USART inverted
usart_oversample_config	configure the USART oversample mode
usart_sample_bit_config	configure sample bit method

Function name	Function description
usart_receiver_timeout_enable	enable receiver timeout
usart_receiver_timeout_disable	disable receiver timeout
usart_receiver_timeout_threshold_config	configure receiver timeout threshold
usart_data_transmit	USART transmit data function
usart_data_receive	USART receive data function
usart_address_config	configure the address of the USART in wake up by address match mode
usart_mute_mode_enable	enable mute mode
usart_mute_mode_disable	disable mute mode
usart_mute_mode_wakeup_config	configure wakeup method in mute mode
usart_lin_mode_enable	enable LIN mode
usart_lin_mode_disable	disable LIN mode
usart_lin_break_dection_length_config	configure LIN break frame length
usart_send_break	send break frame
usart_halfduplex_enable	enable half duplex mode
usart_halfduplex_disable	disable half duplex mode
usart_synchronous_clock_enable	enable CK pin in synchronous mode
usart_synchronous_clock_disable	disable CK pin in synchronous mode
usart_synchronous_clock_config	configure USART synchronous mode parameters
usart_guard_time_config	configure guard time value in smartcard mode
usart_smartcard_mode_enable	enable smartcard mode
usart_smartcard_mode_disable	disable smartcard mode
usart_smartcard_mode_nack_enable	enable NACK in smartcard mode
usart_smartcard_mode_nack_disable	disable NACK in smartcard mode
usart_smartcard_autoretry_config	configure smartcard auto-retry number
usart_block_length_config	configure block length
usart_irda_mode_enable	enable IrDA mode
usart_irda_mode_disable	disable IrDA mode
usart_prescaler_config	configure the peripheral clock prescaler in USART IrDA low-power mode
usart_irda_lowpower_config	configure IrDA low-power
usart_hardware_flow_rts_config	configure hardware flow control RTS
usart_hardware_flow_cts_config	configure hardware flow control CTS
usart_break_frame_coherence_config	configure break frame coherence mode
usart_parity_check_coherence_config	configure parity check coherence mode
usart_hardware_flow_coherence_config	configure hardware flow control coherence mode
usart_dma_receive_config	configure USART DMA reception
usart_dma_transmit_config	configure USART DMA transmission
usart_flag_get	get flag in STAT0/STAT1 register

Function name	Function description
usart_flag_clear	clear flag in STAT0/STAT1 register
usart_interrupt_enable	enable USART interrupt
usart_interrupt_disable	disable USART interrupt
usart_interrupt_flag_get	get USART interrupt flag status
usart_interrupt_flag_clear	clear USART interrupt flag

Enum usart_flag_enum

Table 3-1148. Enum usart_flag_enum

Member name	Function description
USART_FLAG_CTS	CTS change flag
USART_FLAG_LBD	LIN break detected flag
USART_FLAG_TBE	transmit data buffer empty
USART_FLAG_RBNE	transmission complete
USART_FLAG_TC	read data buffer not empty
USART_FLAG_IDLE	IDLE frame detected flag
USART_FLAG_ORERR	overflow error
USART_FLAG_NERR	noise error flag
USART_FLAG_FERR	frame error flag
USART_FLAG_PERR	parity error flag
USART_FLAG_BSY	busy flag
USART_FLAG_EB	end of block flag
USART_FLAG_RT	receiver timeout flag
USART_FLAG_EPERR	early parity error flag

Enum usart_interrupt_flag_enum

Table 3-1149. Enum usart_interrupt_flag_enum

Member name	Function description
USART_INT_FLAG_PERR	parity error interrupt and flag
USART_INT_FLAG_TBE	transmitter buffer empty interrupt and flag
USART_INT_FLAG_TC	transmission complete interrupt and flag
USART_INT_FLAG_RBNE	read data buffer not empty interrupt and flag
USART_INT_FLAG_RBNE_ORE RR	read data buffer not empty interrupt and overrun error flag
USART_INT_FLAG_IDLE	IDLE line detected interrupt and flag
USART_INT_FLAG_LBD	LIN break detected interrupt and flag
USART_INT_FLAG_CTS	CTS interrupt and flag
USART_INT_FLAG_ERR_ORER R	error interrupt and overrun error
USART_INT_FLAG_ERR_NERR	error interrupt and noise error flag
USART_INT_FLAG_ERR_FERR	error interrupt and frame error flag
USART_INT_FLAG_EB	interrupt enable bit of end of block event and flag

USART_INT_FLAG_RT	interrupt enable bit of receive timeout event and flag
-------------------	--

Enum usart_interrupt_enum

Table 3-1150. Enum usart_interrupt_enum

Member name	Function description
USART_INT_PERR	parity error interrupt
USART_INT_TBE	transmitter buffer empty interrupt
USART_INT_TC	transmission complete interrupt
USART_INT_RBNE	read data buffer not empty interrupt and overrun error interrupt
USART_INT_IDLE	IDLE line detected interrupt
USART_INT_LBD	LIN break detected interrupt
USART_INT_CTS	CTS interrupt
USART_INT_ERR	error interrupt
USART_INT_EB	interrupt enable bit of end of block event
USART_INT_RT	interrupt enable bit of receive timeout event

Enum usart_invert_enum

Table 3-1151. Enum usart_invert_enum

Member name	Function description
USART_DINV_ENABLE	data bit level inversion
USART_DINV_DISABLE	data bit level not inversion
USART_TXPIN_ENABLE	TX pin level inversion
USART_TXPIN_DISABLE	TX pin level not inversion
USART_RXPIN_ENABLE	RX pin level inversion
USART_RXPIN_DISABLE	RX pin level not inversion

usart_deinit

The description of usart_deinit is shown as below:

Table 3-1152. Function usart_deinit

Function name	usart_deinit
Function prototype	void usart_deinit(uint32_t usart_periph);
Function descriptions	reset USART/UART
Precondition	-
The called functions	rcu_periph_reset_enable / rcu_periph_reset_disable
Input parameter{in}	
usart_periph	USARTx/UARTx peripheral
USARTx	x=0,1,2,5
UARTx	x=3,4,6,7
Output parameter{out}	
-	-
Return value	

-	-
---	---

Example:

```
/* reset USART0 */
usart_deinit (USART0);
```

usart_baudrate_set

The description of usart_baudrate_set is shown as below:

Table 3-1153. Function usart_baudrate_set

Function name	usart_baudrate_set
Function prototype	void usart_baudrate_set(uint32_t usart_periph, uint32_t baudval);
Function descriptions	configure USART baud rate value
Precondition	-
The called functions	rcu_clock_freq_get
Input parameter{in}	
usart_periph	USARTx/UARTx peripheral
USARTx	x=0,1,2,5
UARTx	x=3,4,6,7
Input parameter{in}	
baudval	baud rate value
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* configure USART0 baud rate value */
usart_baudrate_set(USART0, 115200);
```

usart_parity_config

The description of usart_parity_config is shown as below:

Table 3-1154. Function usart_parity_config

Function name	usart_parity_config
Function prototype	void usart_parity_config(uint32_t usart_periph, uint32_t paritycfg);
Function descriptions	configure USART parity
Precondition	-
The called functions	-
Input parameter{in}	
usart_periph	USARTx/UARTx peripheral
USARTx	x=0,1,2,5

<i>UARTx</i>	x=3,4,6,7
Input parameter{in}	
paritycfg	configure USART parity
<i>USART_PM_NONE</i>	no parity
<i>USART_PM_ODD</i>	odd parity
<i>USART_PM_EVEN</i>	even parity
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* configure USART parity */
```

```
usart_parity_config(USART0, USART_PM_EVEN);
```

usart_word_length_set

The description of usart_word_length_set is shown as below:

Table 3-1155. Function usart_word_length_set

Function name	usart_word_length_set
Function prototype	void usart_word_length_set(uint32_t usart_periph, uint32_t wlen);
Function descriptions	configure USART word length
Precondition	-
The called functions	-
Input parameter{in}	
usart_periph	USARTx/UARTx peripheral
<i>USARTx</i>	x=0,1,2,5
<i>UARTx</i>	x=3,4,6,7
Input parameter{in}	
wlen	USART word length
<i>USART_WL_8BIT</i>	8 bits
<i>USART_WL_9BIT</i>	9 bits
Output parameter{out}	
-	-
Return value	
-	-

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 3-1156. Function usart_stop_bit_set

Function name	usart_stop_bit_set
Function prototype	void usart_stop_bit_set(uint32_t usart_periph, uint32_t stblen);
Function descriptions	configure USART stop bit length
Precondition	-
The called functions	-
Input parameter{in}	
usart_periph	USARTx/UARTx peripheral
USARTx	x=0,1,2,5
UARTx	x=3,4,6,7
Input parameter{in}	
stblen	USART stop bit
USART_STB_1BIT	1 bit
USART_STB_0_5BIT	0.5 bit, not available for UARTx(x=3,4,6,7)
USART_STB_2BIT	2 bits
USART_STB_1_5BIT	1.5 bits, not available for UARTx(x=3,4,6,7)
Output parameter{out}	
-	-
Return value	
-	-

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 3-1157. Function usart_enable

Function name	usart_enable
Function prototype	void usart_enable(uint32_t usart_periph);
Function descriptions	enable USART
Precondition	-
The called functions	-
Input parameter{in}	
usart_periph	USARTx/UARTx peripheral
USARTx	x=0,1,2,5
UARTx	x=3,4,6,7
Output parameter{out}	

-	-
Return value	
-	-

Example:

```
/* enable USART0 */
```

```
usart_enable(USART0);
```

usart_disable

The description of usart_disable is shown as below:

Table 3-1158. Function usart_disable

Function name	usart_disable
Function prototype	void usart_disable(uint32_t usart_periph);
Function descriptions	disable USART
Precondition	-
The called functions	-
Input parameter{in}	
usart_periph	USARTx/UARTx peripheral
USARTx	x=0,1,2,5
UARTx	x=3,4,6,7
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 3-1159. Function usart_transmit_config

Function name	usart_transmit_config
Function prototype	void usart_transmit_config(uint32_t usart_periph, uint32_t txconfig);
Function descriptions	configure USART transmitter
Precondition	-
The called functions	-
Input parameter{in}	
usart_periph	USARTx/UARTx peripheral
USARTx	x=0,1,2,5

<i>UARTx</i>	<i>x=3,4,6,7</i>
Input parameter{in}	
txconfig	enable or disable USART transmitter
<i>USART_TRANSMIT_ENABLE</i>	enable USART transmission
<i>USART_TRANSMIT_DISABLE</i>	disable USART transmission
Output parameter{out}	
-	-
Return value	
-	-

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 3-1160. Function usart_receive_config

Function name	usart_receive_config
Function prototype	void usart_receive_config(uint32_t usart_periph, uint32_t rxconfig);
Function descriptions	configure USART receiver
Precondition	-
The called functions	-
Input parameter{in}	
usart_periph	USARTx/UARTx peripheral
<i>USARTx</i>	<i>x=0,1,2,5</i>
<i>UARTx</i>	<i>x=3,4,6,7</i>
Input parameter{in}	
rxconfig	enable or disable USART receiver
<i>USART_RECEIVE_ENABLE</i>	enable USART reception
<i>USART_RECEIVE_DISABLE</i>	disable USART reception
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 3-1161. Function usart_data_first_config

Function name	usart_data_first_config
Function prototype	void usart_data_first_config(uint32_t usart_periph, uint32_t msbf);
Function descriptions	data is transmitted/received with the LSB/MSB first
Precondition	-
The called functions	-
Input parameter{in}	
usart_periph	USARTx peripheral
<i>USARTx</i>	x=0,1,2,5
Input parameter{in}	
msbf	LSB first or MSB first
<i>USART_MSBF_LSB</i>	LSB first
<i>USART_MSBF_MSB</i>	MSB first
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* configure LSB of data first */
```

```
usart_data_first_config(USART0, USART_MSBF_LSB);
```

usart_invert_config

The description of usart_invert_config is shown as below:

Table 3-1162. Function usart_invert_config

Function name	usart_invert_config
Function prototype	void usart_invert_config(uint32_t usart_periph, usart_invert_enum invertpara);
Function descriptions	configure USART inversion
Precondition	-
The called functions	-
Input parameter{in}	
usart_periph	USARTx peripheral
<i>USARTx</i>	x=0,1,2,5
Input parameter{in}	
invertpara	refer to enum usart_invert_enum
<i>USART_DINV_ENABL</i>	data bit level inversion

<i>E</i>	
<i>USART_DINV_DISABL</i>	data bit level not inversion
<i>E</i>	
<i>USART_TXPIN_ENAB</i>	TX pin level inversion
<i>LE</i>	
<i>USART_TXPIN_DISAB</i>	TX pin level not inversion
<i>LE</i>	
<i>USART_RXPIN_ENAB</i>	RX pin level inversion
<i>LE</i>	
<i>USART_RXPIN_DISAB</i>	RX pin level not inversion
<i>LE</i>	
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* configure USART inversion */
```

```
usart_invert_config(USART0, USART_DINV_ENABLE);
```

usart_oversample_config

The description of usart_oversample_config is shown as below:

Table 3-1163. Function usart_oversample_config

Function name	usart_oversample_config
Function prototype	void usart_oversample_config(uint32_t usart_periph, uint32_t oversamp);
Function descriptions	configure the USART oversample mode
Precondition	-
The called functions	-
Input parameter{in}	
usart_periph	USARTx/UARTx peripheral
<i>USARTx</i>	x=0,1,2,5
<i>UARTx</i>	x=3,4,6,7
Input parameter{in}	
oversamp	oversample value
<i>USART_OVSMOD_8</i>	8 bits
<i>USART_OVSMOD_16</i>	16 bits
Output parameter{out}	
-	-
Return value	
-	-

Example:


```
/* configure USART0 oversample mode */
```

```
usart_oversample_config(USART0, USART_OVSMOD_8);
```

usart_sample_bit_config

The description of usart_sample_bit_config is shown as below:

Table 3-1164. Function usart_sample_bit_config

Function name	usart_sample_bit_config
Function prototype	void usart_sample_bit_config(uint32_t usart_periph, uint32_t obsm);
Function descriptions	configure sample bit method
Precondition	-
The called functions	-
Input parameter{in}	
usart_periph	USARTx/UARTx peripheral
USARTx	x=0,1,2,5
UARTx	x=3,4,6,7
Input parameter{in}	
obsm	sample bit
USART_OSB_1bit	1 bits
USART_OSB_3bit	3 bits
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* configure USART0 sample bit */
```

```
usart_sample_bit_config(USART0, USART_OSB_1bit);
```

usart_receiver_timeout_enable

The description of usart_receiver_timeout_enable is shown as below:

Table 3-1165. Function usart_receiver_timeout_enable

Function name	usart_receiver_timeout_enable
Function prototype	void usart_receiver_timeout_enable(uint32_t usart_periph);
Function descriptions	enable receiver timeout
Precondition	-
The called functions	-
Input parameter{in}	
usart_periph	USARTx peripheral
USARTx	x=0,1,2,5
Output parameter{out}	

-	-
Return value	
-	-

Example:

```
/* enable receiver timeout of USART */
usart_receiver_timeout_enable(USART0);
```

usart_receiver_timeout_disable

The description of usart_receiver_timeout_disable is shown as below:

Table 3-1166. Function usart_receiver_timeout_disable

Function name	usart_receiver_timeout_disable
Function prototype	void usart_receiver_timeout_disable(uint32_t usart_periph);
Function descriptions	disable receiver timeout
Precondition	-
The called functions	-
Input parameter{in}	
usart_periph	USARTx peripheral
<i>USARTx</i>	x=0,1,2,5
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* disable receiver timeout of USART */
usart_receiver_timeout_disable(USART0);
```

usart_receiver_timeout_threshold_config

The description of usart_receiver_timeout_threshold_config is shown as below:

Table 3-1167. Function usart_receiver_timeout_threshold_config

Function name	usart_receiver_timeout_threshold_config
Function prototype	void usart_receiver_timeout_threshold_config(uint32_t usart_periph, uint32_t rtimeout);
Function descriptions	configure receiver timeout threshold
Precondition	-
The called functions	-
Input parameter{in}	
usart_periph	USARTx peripheral
<i>USARTx</i>	x=0,1,2,5

Input parameter{in}	
rtimeout	timeout value
<i>0-0xFFFFF</i>	timeout value
Output parameter{out}	
-	-
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 3-1168. Function usart_data_transmit

Function name	usart_data_transmit
Function prototype	void usart_data_transmit(uint32_t usart_periph, uint32_t data);
Function descriptions	USART transmit data function
Precondition	-
The called functions	-
Input parameter{in}	
usart_periph	USARTx/UARTx peripheral
<i>USARTx</i>	x=0,1,2,5
<i>UARTx</i>	x=3,4,6,7
Input parameter{in}	
data	data of transmission
<i>0-0xFF</i>	data of transmission
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* USART0 transmit data */
```

```
usart_data_transmit(USART0, 0xAA);
```

usart_data_receive

The description of usart_data_receive is shown as below:

Table 3-1169. Function usart_data_receive

Function name	usart_data_receive
---------------	--------------------

Function prototype	void usart_data_receive(uint32_t usart_periph);
Function descriptions	USART receive data function
Precondition	-
The called functions	-
Input parameter{in}	
usart_periph	USARTx/UARTx peripheral
<i>USARTx</i>	x=0,1,2,5
<i>UARTx</i>	x=3,4,6,7
Output parameter{out}	
-	-
Return value	
uint32_t	data of received(0-0xFF)

Example:

```
/* USART0 receive data */

uint16_t temp;

temp = usart_data_receive(USART0);
```

usart_address_config

The description of usart_address_config is shown as below:

Table 3-1170. Function usart_address_config

Function name	usart_address_config
Function prototype	void usart_address_config(uint32_t usart_periph, uint8_t addr);
Function descriptions	configure the address of the USART in wake up by address match mode
Precondition	-
The called functions	-
Input parameter{in}	
usart_periph	USARTx/UARTx peripheral
<i>USARTx</i>	x=0,1,2,5
<i>UARTx</i>	x=3,4,6,7
Input parameter{in}	
addr	address of USART/UART
<i>0-0xFF</i>	address of USART/UART
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* configure address of the USART0 */

usart_address_config(USART0, 0x00);
```

usart_mute_mode_enable

The description of usart_mute_mode_enable is shown as below:

Table 3-1171. Function usart_mute_mode_enable

Function name	usart_mute_mode_enable
Function prototype	void usart_mute_mode_enable(uint32_t usart_periph);
Function descriptions	enable mute mode
Precondition	-
The called functions	-
Input parameter{in}	
usart_periph	USARTx/UARTx peripheral
<i>USARTx</i>	x=0,1,2,5
<i>UARTx</i>	x=3,4,6,7
Output parameter{out}	
-	-
Return value	
-	-

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 3-1172. Function usart_mute_mode_disable

Function name	usart_mute_mode_disable
Function prototype	void usart_mute_mode_disable(uint32_t usart_periph);
Function descriptions	disable mute mode
Precondition	-
The called functions	-
Input parameter{in}	
usart_periph	USARTx/UARTx peripheral
<i>USARTx</i>	x=0,1,2,5
<i>UARTx</i>	x=3,4,6,7
Output parameter{out}	
-	-
Return value	
-	-

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 3-1173. Function usart_mute_mode_wakeup_config

Function name	usart_mute_mode_wakeup_config
Function prototype	void usart_mute_mode_wakeup_config(uint32_t usart_periph, uint32_t wmethod);
Function descriptions	configure wakeup method in mute mode
Precondition	-
The called functions	-
Input parameter{in}	
usart_periph	USARTx/UARTx peripheral
<i>USARTx</i>	x=0,1,2,5
<i>UARTx</i>	x=3,4,6,7
Input parameter{in}	
wmethod	two methods be used to enter or exit the mute mode
<i>USART_WM_IDLE</i>	idle line
<i>USART_WM_ADDR</i>	address mask
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* configure USART0 wakeup method in mute mode */
```

```
usart_mute_mode_wakeup_config(USART0, USART_WM_IDLE);
```

usart_lin_mode_enable

The description of usart_lin_mode_enable is shown as below:

Table 3-1174. Function usart_lin_mode_enable

Function name	usart_lin_mode_enable
Function prototype	void usart_lin_mode_enable(uint32_t usart_periph);
Function descriptions	enable LIN mode
Precondition	-
The called functions	-
Input parameter{in}	
usart_periph	USARTx/UARTx peripheral
<i>USARTx</i>	x=0,1,2,5
<i>UARTx</i>	x=3,4,6,7
Output parameter{out}	

-	-
Return value	
-	-

Example:

```
/* USART0 LIN mode enable */
```

```
usart_lin_mode_enable(USART0);
```

usart_lin_mode_disable

The description of usart_lin_mode_disable is shown as below:

Table 3-1175. Function usart_lin_mode_disable

Function name	usart_lin_mode_disable
Function prototype	void usart_lin_mode_disable(uint32_t usart_periph);
Function descriptions	disable LIN mode
Precondition	-
The called functions	-
Input parameter{in}	
usart_periph	USARTx/UARTx peripheral
USARTx	x=0,1,2,5
UARTx	x=3,4,6,7
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* USART0 LIN mode disable */
```

```
usart_lin_mode_disable(USART0);
```

usart_lin_break_dection_length_config

The description of usart_lin_break_dection_length_config is shown as below:

Table 3-1176. Function usart_lin_break_dection_length_config

Function name	usart_lin_break_dection_length_config
Function prototype	void usart_lin_break_dection_length_config(uint32_t usart_periph, uint32_t lflen);
Function descriptions	configure LIN break frame length
Precondition	-
The called functions	-
Input parameter{in}	
usart_periph	USARTx/UARTx peripheral

<i>USARTx</i>	x=0,1,2,5
<i>UARTx</i>	x=3,4,6,7
Input parameter{in}	
lblen	two methods be used to enter or exit the mute mode
<i>USART_LBLEN_10B</i>	break frame length is 10 bits
<i>USART_LBLEN_11B</i>	break frame length is 11 bits
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* configure LIN break frame length */
```

```
usart_lin_break_dection_length_config(USART0, USART_LBLEN_10B);
```

usart_send_break

The description of usart_send_break is shown as below:

Table 3-1177. Function usart_send_break

Function name	usart_send_break
Function prototype	void usart_send_break(uint32_t usart_periph);
Function descriptions	send break frame
Precondition	-
The called functions	-
Input parameter{in}	
usart_periph	USARTx/UARTx peripheral
<i>USARTx</i>	x=0,1,2,5
<i>UARTx</i>	x=3,4,6,7
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* USART0 send break frame */
```

```
usart_send_break(USART0);
```

usart_halfduplex_enable

The description of usart_halfduplex_enable is shown as below:

Table 3-1178. Function usart_halfduplex_enable

Function name	usart_halfduplex_enable
----------------------	-------------------------

Function prototype	void usart_halfduplex_enable(uint32_t usart_periph);
Function descriptions	enable half duplex mode
Precondition	-
The called functions	-
Input parameter{in}	
usart_periph	USARTx/UARTx peripheral
<i>USARTx</i>	x=0,1,2,5
<i>UARTx</i>	x=3,4,6,7
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 3-1179. Function usart_halfduplex_disable

Function name	usart_halfduplex_disable
Function prototype	void usart_halfduplex_disable(uint32_t usart_periph);
Function descriptions	disable half duplex mode
Precondition	-
The called functions	-
Input parameter{in}	
usart_periph	USARTx/UARTx peripheral
<i>USARTx</i>	x=0,1,2,5
<i>UARTx</i>	x=3,4,6,7
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* disable USART0 half duplex mode*/
```

```
usart_halfduplex_disable(USART0);
```

usart_synchronous_clock_enable

The description of usart_synchronous_clock_enable is shown as below:

Table 3-1180. Function usart_synchronous_clock_enable

Function name	usart_synchronous_clock_enable
Function prototype	void usart_synchronous_clock_enable(uint32_t usart_periph);
Function descriptions	enable CK pin in synchronous mode
Precondition	-
The called functions	-
Input parameter{in}	
usart_periph	USARTx peripheral
<i>USARTx</i>	x=0,1,2,5
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* enable USART0 CK pin in synchronous mode */
usart_synchronous_clock_enable(USART0);
```

usart_synchronous_clock_disable

The description of usart_synchronous_clock_disable is shown as below:

Table 3-1181. Function usart_synchronous_clock_disable

Function name	usart_synchronous_clock_disable
Function prototype	void usart_synchronous_clock_disable(uint32_t usart_periph);
Function descriptions	disable CK pin in synchronous mode
Precondition	-
The called functions	-
Input parameter{in}	
usart_periph	USARTx peripheral
<i>USARTx</i>	x=0,1,2,5
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* disable USART0 CK pin in synchronous mode */
usart_synchronous_clock_disable(USART0);
```

usart_synchronous_clock_config

The description of usart_synchronous_clock_config is shown as below:

Table 3-1182. Function usart_synchronous_clock_config

Function name	usart_synchronous_clock_config
Function prototype	void usart_synchronous_clock_config(uint32_t usart_periph, uint32_t clen, uint32_t cph, uint32_t cpl);
Function descriptions	configure USART synchronous mode parameters
Precondition	-
The called functions	-
Input parameter{in}	
usart_periph	USARTx peripheral
<i>USARTx</i>	x=0,1,2,5
Input parameter{in}	
clen	CK length
<i>USART_CLEN_NONE</i>	there are 7 CK pulses for an 8 bit frame and 8 CK pulses for a 9 bit frame
<i>USART_CLEN_EN</i>	there are 8 CK pulses for an 8 bit frame and 9 CK pulses for a 9 bit frame
Input parameter{in}	
cph	clock phase
<i>USART_CPH_1CK</i>	first clock transition is the first data capture edge
<i>USART_CPH_2CK</i>	second clock transition is the first data capture edge
Input parameter{in}	
cpl	clock polarity
<i>USART_CPL_LOW</i>	steady low value on CK pin
<i>USART_CPL_HIGH</i>	steady high value on CK pin
Output parameter{out}	
-	-
Return value	
-	-

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 3-1183. Function usart_guard_time_config

Function name	usart_guard_time_config
Function prototype	void usart_guard_time_config(uint32_t usart_periph,uint32_t gaut);
Function descriptions	configure guard time value in smartcard mode
Precondition	-
The called functions	-
Input parameter{in}	
usart_periph	USARTx peripheral

<i>USARTx</i>	x=0,1,2,5
Input parameter{in}	
gaut	guard time value
<i>0-0x000000FF</i>	guard time value
Output parameter{out}	
-	-
Return value	
-	-

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 3-1184. Function usart_smartcard_mode_enable

Function name	usart_smartcard_mode_enable
Function prototype	void usart_smartcard_mode_enable(uint32_t usart_periph);
Function descriptions	enable smartcard mode
Precondition	-
The called functions	-
Input parameter{in}	
usart_periph	USARTx peripheral
<i>USARTx</i>	x=0,1,2,5
Output parameter{out}	
-	-
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 3-1185. Function usart_smartcard_mode_disable

Function name	usart_smartcard_mode_disable
Function prototype	void usart_smartcard_mode_disable(uint32_t usart_periph);
Function descriptions	disable smartcard mode
Precondition	-

The called functions	-
Input parameter{in}	
usart_periph	USARTx peripheral
USARTx	x=0,1,2,5
Output parameter{out}	
-	-
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 3-1186. Function usart_smartcard_mode_nack_enable

Function name	usart_smartcard_mode_nack_enable
Function prototype	void usart_smartcard_mode_nack_enable(uint32_t usart_periph);
Function descriptions	enable NACK in smartcard mode
Precondition	-
The called functions	-
Input parameter{in}	
usart_periph	USARTx peripheral
USARTx	x=0,1,2,5
Output parameter{out}	
-	-
Return value	
-	-

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 3-1187. Function usart_smartcard_mode_nack_disable

Function name	usart_smartcard_mode_nack_disable
Function prototype	void usart_smartcard_mode_nack_disable(uint32_t usart_periph);
Function descriptions	disable NACK in smartcard mode
Precondition	-

The called functions	-
Input parameter{in}	
usart_periph	USARTx peripheral
USARTx	x=0,1,2,5
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* disable USART0 NACK in smartcard mode */
```

```
usart_smartcard_mode_nack_disable(USART0);
```

usart_smartcard_autoretry_config

The description of usart_smartcard_autoretry_config is shown as below:

Table 3-1188. Function usart_smartcard_autoretry_config

Function name	usart_smartcard_autoretry_config
Function prototype	void usart_smartcard_autoretry_config(uint32_t usart_periph, uint32_t scrtnum);
Function descriptions	configure smartcard auto-retry number
Precondition	-
The called functions	-
Input parameter{in}	
usart_periph	USARTx peripheral
USARTx	x=0,1,2,5
Input parameter{in}	
scrtnum	smartcard auto-retry number
0-0xFFFFFFFF	smartcard auto-retry number
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* configure smartcard auto-retry number */
```

```
usart_smartcard_autoretry_config (USART0, 0xFFFFFFFF);
```

usart_block_length_config

The description of usart_block_length_config is shown as below:

Table 3-1189. Function usart_block_length_config

Function name	usart_block_length_config
Function prototype	void usart_block_length_config(uint32_t usart_periph, uint32_t bl);
Function descriptions	configure block length in Smartcard T=1 reception
Precondition	-
The called functions	-
Input parameter{in}	
usart_periph	USARTx peripheral
USARTx	x=0,1,2,5
Input parameter{in}	
bl	block length
0-0xFFFFFFFF	block length
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* configure block length in Smartcard T=1 reception */
usart_block_length_config(USART0, 0xFFFFFFFF);
```

usart_irda_mode_enable

The description of usart_irda_mode_enable is shown as below:

Table 3-1190. Function usart_irda_mode_enable

Function name	usart_irda_mode_enable
Function prototype	void usart_irda_mode_enable(uint32_t usart_periph);
Function descriptions	enable IrDA mode
Precondition	-
The called functions	-
Input parameter{in}	
usart_periph	USARTx/UARTx peripheral
USARTx	x=0,1,2,5
UARTx	x=3,4,6,7
Output parameter{out}	
-	-
Return value	
-	-

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 3-1191. Function usart_irda_mode_disable

Function name	usart_irda_mode_disable
Function prototype	void usart_irda_mode_disable(uint32_t usart_periph);
Function descriptions	disable IrDA mode
Precondition	-
The called functions	-
Input parameter{in}	
usart_periph	USARTx/UARTx peripheral
<i>USARTx</i>	x=0,1,2,5
<i>UARTx</i>	x=3,4,6,7
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* disable USART0 IrDA mode */
usart_irda_mode_disable(USART0);
```

usart_prescaler_config

The description of usart_prescaler_config is shown as below:

Table 3-1192. Function usart_prescaler_config

Function name	usart_prescaler_config
Function prototype	void usart_prescaler_config(uint32_t usart_periph, uint8_t psc);
Function descriptions	configure the peripheral clock prescaler in USART IrDA low-power mode
Precondition	-
The called functions	-
Input parameter{in}	
usart_periph	USARTx/UARTx peripheral
<i>USARTx</i>	x=0,1,2,5
<i>UARTx</i>	x=3,4,6,7
Input parameter{in}	
psc	0x00-0xFF
Output parameter{out}	
-	-
Return value	
-	-

Example:


```
/* configure the USART0 peripheral clock prescaler */
```

```
usart_prescaler_config(USART0, 0x00);
```

usart_irda_lowpower_config

The description of usart_irda_lowpower_config is shown as below:

Table 3-1193. Function usart_irda_lowpower_config

Function name	usart_irda_lowpower_config
Function prototype	void usart_irda_lowpower_config(uint32_t usart_periph, uint32_t irlp);
Function descriptions	configure IrDA low-power
Precondition	-
The called functions	-
Input parameter{in}	
usart_periph	USARTx/UARTx peripheral
USARTx	x=0,1,2,5
UARTx	x=3,4,6,7
Input parameter{in}	
irlp	IrDA low-power or normal
USART_IRLP_LOW	low-power
USART_IRLP_NORMAL	normal
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* 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 3-1194. Function usart_hardware_flow_rts_config

Function name	usart_hardware_flow_rts_config
Function prototype	void usart_hardware_flow_rts_config(uint32_t usart_periph, uint32_t rtsconfig);
Function descriptions	configure hardware flow control RTS
Precondition	-
The called functions	-
Input parameter{in}	
usart_periph	USARTx peripheral

<i>USARTx</i>	x=0,1,2,5
Input parameter{in}	
rtsconfig	enable or disable RTS
<i>USART_RTS_ENABLE</i>	enable RTS
<i>USART_RTS_DISABL</i> <i>E</i>	disable RTS
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* 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 3-1195. Function usart_hardware_flow_cts_config

Function name	usart_hardware_flow_cts_config
Function prototype	void usart_hardware_flow_cts_config(uint32_t usart_periph, uint32_t ctsconfig);
Function descriptions	configure hardware flow control RTS
Precondition	-
The called functions	-
Input parameter{in}	
usart_periph	USARTx peripheral
<i>USARTx</i>	x=0,1,2,5
Input parameter{in}	
ctsconfig	enable or disable CTS
<i>USART_CTS_ENABLE</i>	enable CTS
<i>USART_CTS_DISABL</i> <i>E</i>	disable CTS
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* configure USART0 hardware flow control CTS */
```

```
usart_hardware_flow_cts_config(USART0, USART_CTS_ENABLE);
```

usart_break_frame_coherence_config

The description of usart_break_frame_coherence_config is shown as below:

Table 3-1196. Function usart_break_frame_coherence_config

Function name	usart_break_frame_coherence_config
Function prototype	void usart_break_frame_coherence_config(uint32_t usart_periph, uint32_t bcm);
Function descriptions	configure break frame coherence mode
Precondition	-
The called functions	-
Input parameter{in}	
usart_periph	USARTx peripheral
<i>USARTx</i>	x=0,1,2,5
Input parameter{in}	
bcm	
<i>USART_BCM_NONE</i>	No parity error is detected
<i>USART_BCM_EN</i>	Parity error is detected
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* configure USART0 break frame coherence mode */
```

```
usart_break_frame_coherence_config(USART0, USART_BCM_NONE);
```

usart_parity_check_coherence_config

The description of usart_parity_check_coherence_config is shown as below:

Table 3-1197. Function usart_parity_check_coherence_config

Function name	usart_parity_check_coherence_config
Function prototype	void usart_parity_check_coherence_config(uint32_t usart_periph, uint32_t pcm);
Function descriptions	configure parity check coherence mode
Precondition	-
The called functions	-
Input parameter{in}	
usart_periph	USARTx/UARTx peripheral
<i>USARTx</i>	x=0,1,2,5
<i>UARTx</i>	x=3,4,6,7
Input parameter{in}	
pcm	

<i>USART_PCM_NONE</i>	not check parity
<i>USART_PCM_EN</i>	check the parity
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* configure USART0 parity check coherence mode */
```

```
usart_parity_check_coherence_config(USART0, USART_PCM_NONE);
```

usart_hardware_flow_coherence_config

The description of usart_hardware_flow_coherence_config is shown as below:

Table 3-1198. Function usart_hardware_flow_coherence_config

Function name	usart_hardware_flow_coherence_config
Function prototype	void usart_hardware_flow_coherence_config(uint32_t usart_periph, uint32_t hcm);
Function descriptions	configure hardware flow control coherence mode
Precondition	-
The called functions	-
Input parameter{in}	
usart_periph	USARTx peripheral
<i>USARTx</i>	x=0,1,2,5
Input parameter{in}	
hcm	
<i>USART_HCM_NONE</i>	nRTS signal equals to the rxne status register
<i>USART_HCM_EN</i>	nRTS signal is set when the last data bit has been sampled
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* configure hardware flow control coherence mode */
```

```
usart_hardware_flow_coherence_config(USART0, USART_HCM_NONE);
```

usart_dma_receive_config

The description of usart_dma_receive_config is shown as below:

Table 3-1199. Function usart_dma_receive_config

Function name	usart_dma_receive_config
----------------------	--------------------------

Function prototype	void usart_dma_receive_config(uint32_t usart_periph, uint32_t dmacmd);
Function descriptions	configure USART DMA reception
Precondition	-
The called functions	-
Input parameter{in}	
usart_periph	USARTx/UARTx peripheral
<i>USARTx</i>	x=0,1,2,5
<i>UARTx</i>	x=3,4,6,7
Input parameter{in}	
dmacmd	enable or disable DMA for reception
<i>USART_DENR_ENABLE</i>	DMA enable for reception
<i>USART_DENR_DISABLE</i>	DMA disable for reception
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 3-1200. Function usart_dma_transmit_config

Function name	usart_dma_transmit_config
Function prototype	void usart_dma_transmit_config(uint32_t usart_periph, uint32_t dmacmd);
Function descriptions	configure USART DMA transmission
Precondition	-
The called functions	-
Input parameter{in}	
usart_periph	USARTx/UARTx peripheral
<i>USARTx</i>	x=0,1,2,5
<i>UARTx</i>	x=3,4,6,7
Input parameter{in}	
dmacmd	enable or disable DMA for transmission
<i>USART_DENT_ENABLE</i>	DMA enable for transmission
<i>USART_DENT_DISABLE</i>	DMA disable for transmission
Output parameter{out}	

-	-
Return value	
-	-

Example:

```
/* USART0 DMA enable for transmission */
```

```
usart_dma_transmit_config(USART0, USART_DENT_ENABLE);
```

usart_flag_get

The description of usart_flag_get is shown as below:

Table 3-1201. Function usart_flag_get

Function name	usart_flag_get
Function prototype	FlagStatus usart_flag_get(uint32_t usart_periph, usart_flag_enum flag);
Function descriptions	get flag in STAT0/STAT1/CHC register
Precondition	-
The called functions	-
Input parameter{in}	
usart_periph	USARTx/UARTx peripheral
USARTx	x=0,1,2,5
UARTx	x=3,4,6,7
Input parameter{in}	
flag	USART flags, refer to Enum usart_flag_enum
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 3-1202. Function usart_flag_clear

Function name	usart_flag_clear
Function prototype	void usart_flag_clear(uint32_t usart_periph, usart_flag_enum flag);
Function descriptions	clear flag in STAT0/STAT1 register
Precondition	-
The called functions	-

Input parameter{in}	
usart_periph	USARTx/UARTx peripheral
<i>USARTx</i>	x=0,1,2,5
<i>UARTx</i>	x=3,4,6,7
Input parameter{in}	
flag	USART flags, refer to Enum usart_flag_enum
<i>USART_FLAG_CTS</i>	CTS change flag
<i>USART_FLAG_LBD</i>	LIN break detected flag
<i>USART_FLAG_TC</i>	transmission complete
<i>USART_FLAG_RBNE</i>	read data buffer not empty
<i>USART_FLAG_EB</i>	end of block flag
<i>USART_FLAG_RT</i>	receiver timeout flag
<i>USART_FLAG_EPERR</i>	early parity error flag
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* clear USART0 flag */
usart_flag_clear(USART0,USART_FLAG_TC);
```

usart_interrupt_enable

The description of usart_interrupt_enable is shown as below:

Table 3-1203. Function usart_interrupt_enable

Function name	usart_interrupt_enable
Function prototype	void usart_interrupt_enable(uint32_t usart_periph, usart_interrupt_enum interrupt);
Function descriptions	enable USART interrupt
Precondition	-
The called functions	-
Input parameter{in}	
usart_periph	USARTx/UARTx peripheral
<i>USARTx</i>	x=0,1,2,5
<i>UARTx</i>	x=3,4,6,7
Input parameter{in}	
interrupt	USART interrupt, refer to Enum usart_interrupt_enum
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* enable USART0 TBE interrupt */

usart_interrupt_enable(USART0, USART_INT_TBE);
```

usart_interrupt_disable

The description of usart_interrupt_disable is shown as below:

Table 3-1204. Function usart_interrupt_disable

Function name	usart_interrupt_disable
Function prototype	void usart_interrupt_disable(uint32_t usart_periph, usart_interrupt_enum interrupt);
Function descriptions	disable USART interrupt
Precondition	-
The called functions	-
Input parameter{in}	
usart_periph	USARTx/UARTx peripheral
USARTx	x=0,1,2,5
UARTx	x=3,4,6,7
Input parameter{in}	
interrupt	USART interrupt, refer to Enum usart_interrupt_enum
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* 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 3-1205. Function usart_interrupt_flag_get

Function name	usart_interrupt_flag_get
Function prototype	FlagStatus usart_interrupt_flag_get(uint32_t usart_periph, usart_interrupt_flag_enum int_flag);
Function descriptions	get USART interrupt and flag status
Precondition	-
The called functions	-
Input parameter{in}	
usart_periph	USARTx/UARTx peripheral
USARTx	x=0,1,2,5

<i>UARTx</i>	x=3,4,6,7
Input parameter{in}	
int_flag	USART interrupt flag, refer to Enum usart_interrupt_flag_enum
Output parameter{out}	
-	-
Return value	
FlagStatus	SET or RESET

Example:

```
/* 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 3-1206. Function usart_interrupt_flag_clear

Function name	usart_interrupt_flag_clear
Function prototype	void usart_interrupt_flag_clear(uint32_t usart_periph, usart_interrupt_flag_enum int_flag);
Function descriptions	clear USART interrupt flag in STAT0/STAT1 register
Precondition	-
The called functions	-
Input parameter{in}	
usart_periph	USARTx/UARTx peripheral
<i>USARTx</i>	x=0,1,2,5
<i>UARTx</i>	x=3,4,6,7
Input parameter{in}	
int_flag	USART interrupt flag, refer to Enum usart_interrupt_flag_enum
<i>USART_INT_FLAG_CTS</i>	CTS change flag
<i>USART_INT_FLAG_LBD</i>	LIN break detected flag
<i>USART_INT_FLAG_TC</i>	transmission complete
<i>USART_INT_FLAG_RBNE</i>	read data buffer not empty
<i>USART_INT_FLAG_EB</i>	end of block event interrupt flag
<i>USART_INT_FLAG_RT</i>	receive timeout event interrupt flag
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* clear the USART0 interrupt enable bit status */
usart_interrupt_flag_clear(USART0, USART_INT_FLAG_RBNE);
```

3.33. WWDGT

The window watchdog timer (WWDGT) is used to detect system failures due to software malfunctions. The WWDGT registers are listed in chapter [3.33.1](#), the WWDGT firmware functions are introduced in chapter [3.33.2](#).

3.33.1. Descriptions of Peripheral registers

WWDGT registers are listed in the table shown as below:

Table 3-1207. WWDGT Registers

Registers	Descriptions
WWDGT_CTL	Control register
WWDGT_CFG	Configuration register
WWDGT_STAT	Status register

3.33.2. Descriptions of Peripheral functions

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

Table 3-1208. WWDGT firmware function

Function name	Function description
wwdgt_deinit	reset the window watchdog timer configuration
wwdgt_enable	start the window watchdog timer counter
wwdgt_counter_update	configure the window watchdog timer counter value
wwdgt_config	configure counter value, window value, and prescaler divider value
wwdgt_flag_get	check early wakeup interrupt state of WWDGT
wwdgt_flag_clear	clear early wakeup interrupt state of WWDGT
wwdgt_interrupt_enable	enable early wakeup interrupt of WWDGT

wwdgt_deinit

The description of wwdgt_deinit is shown as below:

Table 3-1209. Function wwdgt_deinit

Function name	wwdgt_deinit
Function prototype	void wwdgt_deinit(void);
Function descriptions	reset the window watchdog timer configuration
Precondition	-

The called functions	-
Input parameter{in}	
-	-
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* reset the window watchdog timer configuration */
```

```
wwdgt_deinit( );
```

wwdgt_enable

The description of wwdgt_enable is shown as below:

Table 3-1210. Function wwdgt_enable

Function name	wwdgt_enable
Function prototype	void wwdgt_enable(void);
Function descriptions	start the window watchdog timer counter
Precondition	-
The called functions	-
Input parameter{in}	
-	-
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* start the window watchdog timer counter */
```

```
wwdgt_enable( );
```

wwdgt_counter_update

The description of wwdgt_counter_update is shown as below:

Table 3-1211. Function wwdgt_counter_update

Function name	wwdgt_counter_update
Function prototype	void wwdgt_counter_update(uint16_t counter_value);
Function descriptions	configure the window watchdog timer counter value
Precondition	-
The called functions	-
Input parameter{in}	

counter_value	0x00 - 0x7F
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* update WWDGT counter to 0x7F */
```

```
wwdgt_counter_update(127);
```

wwdgt_config

The description of wwdgt_config is shown as below:

Table 3-1212. Function wwdgt_config

Function name	wwdgt_config
Function prototype	void wwdgt_config(uint16_t counter, uint16_t window, uint32_t prescaler);
Function descriptions	configure counter value, window value, and prescaler divider value
Precondition	-
The called functions	-
Input parameter{in}	
counter	0x00 - 0x7F
Input parameter{in}	
window	0x00 - 0x7F
Input parameter{in}	
prescaler	wwdgt prescaler value
WWDGT_CFG_PSC_D IV1	the time base of window watchdog counter = (PCLK1 / 4096) / 1
WWDGT_CFG_PSC_D IV2	the time base of window watchdog counter = (PCLK1 / 4096) / 2
WWDGT_CFG_PSC_D IV4	the time base of window watchdog counter = (PCLK1 / 4096) / 4
WWDGT_CFG_PSC_D IV8	the time base of window watchdog counter = (PCLK1 / 4096) / 8
WWDGT_CFG_PSC_D IV16	the time base of window watchdog counter = (PCLK1 / 4096) / 16
WWDGT_CFG_PSC_D IV32	the time base of window watchdog counter = (PCLK1 / 4096) / 32
WWDGT_CFG_PSC_D IV64	the time base of window watchdog counter = (PCLK1 / 4096) / 64
WWDGT_CFG_PSC_D IV128	the time base of window watchdog counter = (PCLK1 / 4096) / 128
WWDGT_CFG_PSC_D IV256	the time base of window watchdog counter = (PCLK1 / 4096) / 256

WWDGT_CFG_PSC_D IV512	the time base of window watchdog counter = (PCLK1 / 4096) / 512
WWDGT_CFG_PSC_D IV1024	the time base of window watchdog counter = (PCLK1 / 4096) / 1024
WWDGT_CFG_PSC_D IV2048	the time base of window watchdog counter = (PCLK1 / 4096) / 2048
WWDGT_CFG_PSC_D IV4096	the time base of window watchdog counter = (PCLK1 / 4096) / 4096
WWDGT_CFG_PSC_D IV8192	the time base of window watchdog counter = (PCLK1 / 4096) / 8192
WWDGT_CFG_PSC_D IV16384	the time base of window watchdog counter = (PCLK1 / 4096) / 16384
WWDGT_CFG_PSC_D IV32768	the time base of window watchdog counter = (PCLK1 / 4096) / 32768
WWDGT_CFG_PSC_D IV65536	the time base of window watchdog counter = (PCLK1 / 4096) / 65536
WWDGT_CFG_PSC_D IV131072	the time base of window watchdog counter = (PCLK1 / 4096) / 131072
WWDGT_CFG_PSC_D IV262144	the time base of window watchdog counter = (PCLK1 / 4096) / 262144
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* configure WWDGT counter value to 0x7F, window value to 0x50, prescaler divider value to 8 */
```

```
wwdgt_config(127, 80, WWDGT_CFG_PSC_DIV8);
```

wwdgt_flag_get

The description of wwdgt_flag_get is shown as below:

Table 3-1213. Function wwdgt_flag_get

Function name	wwdgt_flag_get
Function prototype	FlagStatus wwdgt_flag_get(void);
Function descriptions	check early wakeup interrupt state of WWDGT
Precondition	-
The called functions	-
Input parameter{in}	
-	-
Output parameter{out}	
-	-

Return value	
FlagStatus	SET or RESET

Example:

```
/* test if the counter value update has reached the 0x40 */
```

```
FlagStatus status;
```

```
status = wwdgt_flag_get( );
```

```
if(status == RESET)
```

```
{
```

```
...
```

```
}else
```

```
{
```

```
...
```

```
}
```

wwdgt_flag_clear

The description of wwdgt_flag_clear is shown as below:

Table 3-1214. Function wwdgt_flag_clear

Function name	wwdgt_flag_clear
Function prototype	void wwdgt_flag_clear(void);
Function descriptions	clear early wakeup interrupt state of WWDGT
Precondition	-
The called functions	-
Input parameter{in}	
-	-
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* clear early wakeup interrupt state of WWDGT */
```

```
wwdgt_flag_clear( );
```

wwdgt_interrupt_enable

The description of wwdgt_interrupt_enable is shown as below:

Table 3-1215. Function wwdgt_interrupt_enable

Function name	wwdgt_interrupt_enable
Function prototype	void wwdgt_interrupt_enable(void);
Function descriptions	enable early wakeup interrupt of WWDGT
Precondition	-
The called functions	-
Input parameter{in}	
-	-
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* enable early wakeup interrupt of WWDGT */
```

```
wwdgt_interrupt_enable ( );
```

3.34. SAI

The serial audio interface(SAI) is designed to target a wide range of commonly used audio protocols, both in mono and stereo modes, such as I2S, PCM/DSP, AC'97, LSB or MSB justified and TDM. The SAI registers are listed in chapter [3.34.1](#), the SAI firmware functions are introduced in chapter [3.34.2](#).

3.34.1. Descriptions of Peripheral registers

SAI registers are listed in the table shown as below:

Table 3-1216. SAI Registers

Registers	Descriptions
SAI_SYNCFG	SAI synchronize configuration register
SAI_B0CFG0	SAI block 0 configuration register0
SAI_B0CFG1	SAI block 0 configuration register1
SAI_B0FCFG	SAI block 0 frame configuration register
SAI_B0SCFG	SAI block 0 slot configuration register
SAI_B0INTEN	SAI block 0 interrupt enable register
SAI_B0STAT	SAI block 0 status register
SAI_B0INTC	SAI block 0 interrupt flag clear register
SAI_B0DATA	SAI block 0 data register
SAI_B1CFG0	SAI block 1 configuration register0
SAI_B1CFG1	SAI block 1 configuration register1
SAI_B1FCFG	SAI block 1 frame configuration register
SAI_B1SCFG	SAI block 1 slot configuration register

Registers	Descriptions
SAI_B1INTEN	SAI block 1 interrupt enable register
SAI_B1STAT	SAI block 1 status register
SAI_B1INTC	SAI block 1 interrupt flag clear register
SAI_B1DATA	SAI block 1 data register

3.34.2. Descriptions of Peripheral functions

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

Table 3-1217. SAI firmware function

Function name	Function description
sai_deinit	reset SAI
sai_struct_para_init	initialize SAI parameter struct with the default values
sai_frame_struct_para_init	initialize SAI frame parameter struct with the default values
sai_slot_struct_para_init	initialize SAI slot parameter struct with the default values
sai_init	initialize SAI
sai_frame_init	initialize SAI frame
sai_slot_init	initialize SAI slot
sai_enable	sai enable
sai_disable	sai disable
sai_sdoutput_config	SAI serial data near inactive slot output management
sai_monomode_config	configure SAI mono mode
sai_companing_config	configure SAI companding mode
sai_mute_enable	enable SAI mute detected or mute send
sai_mute_disable	disable SAI mute detected or mute send
sai_mute_value_config	configure SAI mute value
sai_mute_count_config	configure SAI mute frame count
sai_data_transmit	SAI transmit data
sai_data_receive	SAI receive data
sai_fifo_status_get	get SAI fifo status
sai_fifo_flush	SAI fifo flush
sai_dma_enable	enable SAI dma
sai_dma_disable	disable SAI dma
sai_interrupt_enable	enable the SAI interrupt
sai_interrupt_disable	disable the SAI interrupt
sai_interrupt_flag_get	get SAI interrupt flag status
sai_interrupt_flag_clear	clear SAI interrupt flag status
sai_flag_get	get SAI flag status
sai_flag_clear	clear SAI flag status

Structure sai_parameter_struct

Table 3-1218. Structure sai_parameter_struct

Member name	Function description
operating_mode	operating mode
protocol	protocol selection
data_width	data width
shift_dir	shift direction
sample_edge	sampling clock edge
sync_mode	synchronization mode
output_drive	output Drive
clk_div_bypass	clock divider logic bypass
mclk_div	master clock divider ratio
mclk_oversampling	the master clock oversampling rate
mclk_enable	the master clock enable
fifo_threshold	FIFO threshold

Structure sai_frame_parameter_struct

Table 3-1219. Structure sai_frame_parameter_struct

Member name	Function description
frame_width	frame width
frame_sync_width	frame synchronization active width
frame_sync_function	frame synchronization function
frame_sync_polarity	frame synchronization active polarity
frame_sync_offset	frame synchronization offset

Structure sai_slot_parameter_struct

Table 3-1220. Structure sai_slot_parameter_struct

Member name	Function description
slot_number	slot number
slot_width	slot width
data_offset	data offset
slot_active	slot activation vector

Enum sai_fifo_state_enum

Table 3-1221. Enum sai_fifo_state_enum

Member name	Function description
FIFO_EMPTY	empty
FIFO_EMPTY_TO_1_4_FULL	$\text{empty} < \text{fifo_level} \leq 1/4_full$
FIFO_1_4_FULL_TO_1_2_FULL	$1/4_full < \text{fifo_level} \leq 1/2_full$
FIFO_1_2_FULL_TO_3_4_FULL	$1/2_full < \text{fifo_level} \leq 3/4_full$
FIFO_3_4_FULL_TO_FULL	$3/4_full < \text{fifo_level} < \text{full}$

Member name	Function description
FIFO_FULL	full

sai_deinit

The description of sai_deinit is shown as below:

Table 3-1222. Function sai_deinit

Function name	sai_deinit
Function prototype	void sai_deinit();
Function descriptions	reset SAI
Precondition	-
The called functions	rcu_periph_reset_enable / rcu_periph_reset_disable
Input parameter{in}	
-	-
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* deinitialize the SAI */
sai_deinit();
```

sai_struct_para_init

The description of sai_struct_para_init is shown as below:

Table 3-1223. Function sai_struct_para_init

Function name	sai_struct_para_init
Function prototype	void sai_struct_para_init(sai_parameter_struct* initpara);
Function descriptions	initialize SAI parameter struct with the default values
Precondition	-
The called functions	-
Input parameter{in}	
initpara	the initialization data needed to initialize SAI, refer to Table 3-1218. Structure sai_parameter_struct
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* initialize the parameters of SAI structure */
```

```
sai_struct_para_init sai_init_struct;

sai_struct_para_init (&sai_init_struct);
```

sai_frame_struct_para_init

The description of sai_frame_struct_para_init is shown as below:

Table 3-1224. Function sai_frame_struct_para_init

Function name	sai_frame_struct_para_init
Function prototype	void sai_frame_struct_para_init(sai_frame_parameter_struct* initpara);
Function descriptions	initialize the parameter of SAI frame structure with a default value
Precondition	-
The called functions	-
Input parameter{in}	
initpara	the initialization data needed to initialize SAI frame, refer to Table 3-1219. Structure sai_frame_parameter_struct
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* initialize the parameters of SAI frame structure */

sai_frame_struct_para_init sai_frame_init_struct;

sai_frame_struct_para_init (&sai_frame_init_struct);
```

sai_slot_struct_para_init

The description of sai_slot_struct_para_init is shown as below:

Table 3-1225. Function sai_slot_struct_para_init

Function name	sai_slot_struct_para_init
Function prototype	void sai_slot_struct_para_init(sai_slot_parameter_struct* initpara);
Function descriptions	initialize the parameter of SAI slot structure with a default value
Precondition	-
The called functions	-
Input parameter{in}	
initpara	the initialization data needed to initialize SAI slot, refer to Table 3-1220. Structure sai_slot_parameter_struct
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* initialize the parameters of SAI slot structure */

sai_slot_struct_para_init sai_slot_init_struct;

sai_slot_struct_para_init (&sai_slot_init_struct);
```

sai_init

The description of sai_init is shown as below:

Table 3-1226. Function sai_init

Function name	sai_init
Function prototype	void sai_init(uint32_t block, sai_parameter_struct* initpara);
Function descriptions	initialize SAI
Precondition	-
The called functions	-
Input parameter{in}	
block	SAI block
<i>SAI_BLOCKx</i>	(x=0,1)
Input parameter{in}	
initpara	the initialization data needed to initialize SAI, refer to Table 3-1218. Structure sai_parameter_struct
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* initialize SAI frame */
sai_parameter_struct sai_init_structure;
sai_init_structure.operating_mode = SAI_MASTER_TRANSMITTER;
sai_init_structure.protocol = SAI_PROTOCOL_POLYMORPHIC;
sai_init_structure.data_width = SAI_DATAWIDTH_16BIT;
sai_init_structure.shift_dir = SAI_SHIFT_MSB;
sai_init_structure.sample_edge = SAI_SAMPEDGE_FALLING;
sai_init_structure.sync_mode = SAI_SYNCMODE_ASYNC;
sai_init_structure.output_drive = SAI_OUTPUT_WITH_SAIEN;
sai_init_structure.clk_div_bypass = SAI_CLKDIV_BYPASS_OFF;
sai_init_structure.mclk_div = SAI_MCLKDIV_2;
sai_init_structure.mclk_oversampling = SAI_MASTERCLK_OVERSAMP_256;
sai_init_structure.mclk_enable = SAI_MASTERCLK_DISABLE;
sai_init_structure.fifo_threshold = SAI_FIFOTH_QUARTER;
sai_init (SAI_BLOCK0, sai_init_structure);
```

sai_frame_init

The description of sai_frame_init is shown as below:

Table 3-1227. Function sai_frame_init

Function name	sai_frame_init
Function prototype	void sai_frame_init(uint32_t block, sai_frame_parameter_struct* initpara);
Function descriptions	initialize SAI frame
Precondition	-
The called functions	-
Input parameter{in}	
block	SAI block
SAI_BLOCKx	(x=0,1)
Input parameter{in}	
initpara	the initialization data needed to initialize SAI frame, refer to Table 3-1219. Structure sai_frame_parameter_struct
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* initialize SAI frame */
sai_frame_parameter_struct sai_frame_init_structure;
sai_frame_init_structure.frame_width = 128;
sai_frame_init_structure.frame_sync_width = 1;
sai_frame_init_structure.frame_sync_function = SAI_FSFUNC_START;
sai_frame_init_structure.frame_sync_polarity = SAI_FSPOLR_LOW;
sai_frame_init_structure.frame_sync_offset = SAI_FSOST_BEGINNING;
sai_frame_init(SAI_BLOCK0, sai_frame_init_structure);
```

sai_slot_init

The description of sai_slot_init is shown as below:

Table 3-1228. Function sai_slot_init

Function name	sai_slot_init
Function prototype	void sai_slot_init(uint32_t block, sai_slot_parameter_struct* initpara);
Function descriptions	initialize SAI slot
Precondition	-
The called functions	-
Input parameter{in}	
block	SAI block
SAI_BLOCKx	(x=0,1)
Input parameter{in}	

initpara	the initialization data needed to initialize SAI slot, refer to Table 3-1220. Structure sai_slot_parameter_struct
Output parameter{out}	
-	-
Return value	
-	-

Example:

```

/* initialize SAI slot */
sai_slot_parameter_struct sai_slot_init_structure;
sai_slot_init_structure.slot_number = 8;
sai_slot_init_structure.slot_width = SAI_SLOTWIDTH_16BIT;
sai_slot_init_structure.data_offset = 0;
sai_slot_init_structure.slot_active = SAI_SLOT_ACTIVE_ALL;
void sai_slot_init(SAI_BLOCK0, sai_slot_init_structure);

```

sai_enable

The description of sai_enable is shown as below:

Table 3-1229. Function sai_enable

Function name	sai_enable
Function prototype	void sai_enable(uint32_t block);
Function descriptions	SAI enable
Precondition	-
The called functions	-
Input parameter{in}	
block	SAI block
SAI_BLOCKx	(x=0,1)
Output parameter{out}	
-	-
Return value	
-	-

Example:

```

/* enable SAI */
sai_enable(SAI_BLOCK0);

```

sai_disable

The description of sai_disable is shown as below:

Table 3-1230. Function sai_disable

Function name	sai_disable
Function prototype	void sai_disable(uint32_t block);

Function descriptions	SAI disable
Precondition	-
The called functions	-
Input parameter{in}	
block	SAI block
<i>SAI_BLOCKx</i>	(x=0,1)
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* disable SAI */
```

```
sai_disable (SAI_BLOCK0);
```

sai_sdoutput_config

The description of sai_sdoutput_config is shown as below:

Table 3-1231. Function sai_sdoutput_config

Function name	sai_sdoutput_config
Function prototype	void sai_sdoutput_config(uint32_t block, uint32_t sdout);
Function descriptions	SAI serial data near inactive slot output management
Precondition	-
The called functions	-
Input parameter{in}	
block	SAI block
<i>SAI_BLOCKx</i>	(x=0,1)
Input parameter{in}	
sdout	serial data output management mode selection
<i>SAI_SDLINE_DRIVE</i>	SD line output is driven entirely during the audio frame
<i>SAI_SDLINE_RELEASE</i>	SD line output is released near inactive slots
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* configure SAI serial data output management mode selection */
```

```
sai_sdoutput_config (SAI_BLOCK0, SAI_SDLINE_DRIVE);
```

sai_monomode_config

The description of sai_monomode_config is shown as below:

Table 3-1232. Function sai_monomode_config

Function name	sai_monomode_config
Function prototype	void sai_monomode_config(uint32_t block, uint32_t mono);
Function descriptions	configure SAI mono mode
Precondition	-
The called functions	-
Input parameter{in}	
block	SAI block
SAI_BLOCKx	(x=0,1)
Input parameter{in}	
mono	stereo and mono mode selection
SAI_STEREO_MODE	stereo mode
SAI_MONO_MODE	mono mode
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* configure SAI mono mode */
sai_monomode_config(SAI_BLOCK0, SAI_STEREO_MODE);
```

sai_companding_config

The description of sai_companding_config is shown as below:

Table 3-1233. Function sai_companding_config

Function name	sai_companding_config
Function prototype	void sai_companding_config(uint32_t block, uint32_t compander, uint32_t complement);
Function descriptions	configure SAI companding mode
Precondition	-
The called functions	-
Input parameter{in}	
block	SAI block
SAI_BLOCKx	(x=0,1)
Input parameter{in}	
compander	compander mode
SAI_COMPANDER_OF F	no compansion applies

SAI_COMPANDER_UL AW	u-law algorithm
SAI_COMPANDER_AL AW	A-law algorithm
Input parameter{in}	
complement	complement mode.
SAI_COMPLEMENT_1 S	data represented in 1's complement form
AI_COMPLEMENT_2S	data represented in 2's complement form
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* configure SAI companding mode */
```

```
sai_companding_config(SAI_BLOCK0, SAI_MONO_MODE, SAI_COMPLEMENT_1S);
```

sai_mute_enable

The description of sai_mute_enable is shown as below:

Table 3-1234. Function sai_mute_enable

Function name	sai_mute_enable
Function prototype	void sai_mute_enable(uint32_t block);
Function descriptions	SAI mute detected enable or mute send enable
Precondition	-
The called functions	-
Input parameter{in}	
block	SAI block
SAI_BLOCKx	(x=0,1)
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* enable SAI mute */
```

```
sai_mute_enable(SAI_BLOCK0);
```

sai_mute_disable

The description of sai_mute_disable is shown as below:

Table 3-1235. Function sai_mute_disable

Function name	sai_mute_disable
Function prototype	void sai_mute_disable(uint32_t block);
Function descriptions	SAI mute detected disable or mute send disable
Precondition	-
The called functions	-
Input parameter{in}	
block	SAI block
SAI_BLOCKx	(x=0,1)
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* disable SAI mute */
sai_mute_disable(SAI_BLOCK0);
```

sai_mute_value_config

The description of sai_mute_value_config is shown as below:

Table 3-1236. Function sai_mute_value_config

Function name	sai_mute_value_config
Function prototype	void sai_mute_value_config(uint32_t block, uint32_t value);
Function descriptions	configure SAI mute value
Precondition	-
The called functions	-
Input parameter{in}	
block	SAI block
SAI_BLOCKx	(x=0,1)
Input parameter{in}	
value	mute value
SAI_MUTESENT_0	0 is sent via the Serial Data (SD) line when mute is on
SAI_MUTESENT_LASTFRAME	If SLOTNB is less or equals to two, last frame is sent via the Serial Data (SD) line
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* configure SAI mute value is 0*/
```

```
sai_mute_value_config(SAI_BLOCK0, SAI_MUTESENT_0);
```

sai_mute_count_config

The description of sai_mute_count_config is shown as below:

Table 3-1237. Function sai_mute_count_config

Function name	sai_mute_count_config
Function prototype	void sai_mute_count_config(uint32_t block, uint32_t count);
Function descriptions	configure SAI mute frame count
Precondition	-
The called functions	-
Input parameter{in}	
block	SAI block
SAI_BLOCKx	(x=0,1)
Input parameter{in}	
count	0~63, mute frame count
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* configure SAI mute frame count 16 */
```

```
sai_mute_count_config (SAI_BLOCK0, 16);
```

sai_data_transmit

The description of sai_data_transmit is shown as below:

Table 3-1238. Function sai_data_transmit

Function name	sai_data_transmit
Function prototype	void sai_data_transmit(uint32_t block, uint32_t data);
Function descriptions	SAI transmit data
Precondition	-
The called functions	-
Input parameter{in}	
block	SAI block
SAI_BLOCKx	(x=0,1)
Input parameter{in}	
data	32-bit data
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* SAI transmit data */
```

```
sai_data_transmit (SAI_BLOCK0, sai_send_array[send_n]);
```

sai_data_receive

The description of sai_data_receive is shown as below:

Table 3-1239. Function sai_data_receive

Function name	sai_data_receive
Function prototype	uint32_t sai_data_receive(uint32_t block);
Function descriptions	SAI receive data
Precondition	-
The called functions	-
Input parameter{in}	
block	SAI block
SAI_BLOCKx	(x=0,1)
Output parameter{out}	
-	-
Return value	
uint32_t	0x00000000-0xFFFFFFFF

Example:

```
/* SAI receive data */
```

```
uint32_t sai_receiver;
```

```
sai_receiver = sai_data_receive ();
```

sai_fifo_status_get

The description of sai_fifo_status_get is shown as below:

Table 3-1240. Function sai_fifo_status_get

Function name	sai_fifo_status_get
Function prototype	sai_fifo_state_enum sai_fifo_status_get(uint32_t block);
Function descriptions	get SAI fifo status
Precondition	-
The called functions	-
Input parameter{in}	
block	SAI block
SAI_BLOCKx	(x=0,1)
Output parameter{out}	
-	-
Return value	

sai_fifo_state_enum	fifo status, refer to Table 3-1221. Enum sai_fifo_state_enum
----------------------------	--

Example:

```
/* get SAI fifo status */

sai_fifo_state_enum fifo_state;

fifo_state =sai_fifo_status_get(SAI_BLOCK0);
```

sai_fifo_flush

The description of sai_fifo_flush is shown as below:

Table 3-1241. Function sai_fifo_flush

Function name	sai_fifo_flush
Function prototype	void sai_fifo_flush(uint32_t block);
Function descriptions	SAI fifo flush
Precondition	-
The called functions	-
Input parameter{in}	
block	SAI block
<i>SAI_BLOCKx</i>	(x=0,1)
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* SAI fifo flush */

sai_fifo_flush();
```

sai_dma_enable

The description of sai_dma_enable is shown as below:

Table 3-1242. Function sai_dma_enable

Function name	sai_dma_enable
Function prototype	void sai_dma_enable(uint32_t block);
Function descriptions	enable SAI dma
Precondition	-
The called functions	-
Input parameter{in}	
block	SAI block
<i>SAI_BLOCKx</i>	(x=0,1)
Output parameter{out}	
-	-

Return value	
-	-

Example:

```
/* enable SAI dma */
```

```
sai_dma_enable (SAI_BLOCK0);
```

sai_dma_disable

The description of sai_dma_disable is shown as below:

Table 3-1243. Function sai_dma_disable

Function name	sai_dma_disable
Function prototype	void sai_dma_disable(uint32_t block);
Function descriptions	disable SAI dma
Precondition	-
The called functions	-
Input parameter{in}	
block	SAI block
<i>SAI_BLOCKx</i>	(x=0,1)
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* disable SAI dma */
```

```
sai_dma_disable (SAI_BLOCK0);
```

sai_interrupt_enable

The description of sai_interrupt_enable is shown as below:

Table 3-1244. Function sai_interrupt_enable

Function name	sai_interrupt_enable
Function prototype	void sai_interrupt_enable(uint32_t block, uint32_t interrupt);
Function descriptions	enable the SAI interrupt
Precondition	-
The called functions	-
Input parameter{in}	
block	SAI block
<i>SAI_BLOCKx</i>	(x=0,1)
Input parameter{in}	
interrupt	specify which interrupt to enable

<i>SAI_INT_OUERR</i>	FIFO overrun or underrun interrupt enable
<i>SAI_INT_MTDET</i>	mute detection interrupt enable
<i>SAI_INT_ERRCK</i>	error clock interrupt enable
<i>SAI_INT_FFREQ</i>	FIFO request interrupt enable
<i>SAI_INT_ACNRDY</i>	audio codec not ready interrupt enable
<i>SAI_INT_FSADET</i>	frame synchronization advanced detection interrupt enable
<i>SAI_INT_FSPDET</i>	frame synchronization postpone detection interrupt enable
Output parameter{out}	
-	-
Return value	
FlagStatus	SET or RESET

Example:

```
/* enable SAI block0 mute detection interrupt */
```

```
sai_interrupt_enable(SAI_BLOCK0, SAI_INT_MTDET);
```

sai_interrupt_disable

The description of sai_interrupt_disable is shown as below:

Table 3-1245. Function sai_interrupt_disable

Function name	sai_interrupt_disable
Function prototype	void sai_interrupt_disable(uint32_t block, uint32_t interrupt);
Function descriptions	disable the SAI interrupt
Precondition	-
The called functions	-
Input parameter{in}	
block	SAI block
<i>SAI_BLOCKx</i>	(x=0,1)
Input parameter{in}	
interrupt	specify which interrupt to enable
<i>SAI_INT_OUERR</i>	FIFO overrun or underrun interrupt enable
<i>SAI_INT_MTDET</i>	mute detection interrupt enable
<i>SAI_INT_ERRCK</i>	error clock interrupt enable
<i>SAI_INT_FFREQ</i>	FIFO request interrupt enable
<i>SAI_INT_ACNRDY</i>	audio codec not ready interrupt enable
<i>SAI_INT_FSADET</i>	frame synchronization advanced detection interrupt enable
<i>SAI_INT_FSPDET</i>	frame synchronization postpone detection interrupt enable
Output parameter{out}	
-	-
Return value	
FlagStatus	SET or RESET

Example:

```
/* disable SAI block0 mute detection interrupt */
```

```
sai_interrupt_disable(SAI_BLOCK0, SAI_INT_MTDET);
```

sai_interrupt_flag_get

The description of sai_interrupt_flag_get is shown as below:

Table 3-1246. Function sai_interrupt_flag_get

Function name	sai_interrupt_flag_get
Function prototype	FlagStatus sai_interrupt_flag_get(uint32_t block, uint32_t interrupt);
Function descriptions	get the SAI interrupt flag
Precondition	-
The called functions	-
Input parameter{in}	
block	SAI block
SAI_BLOCKx	(x=0,1)
Input parameter{in}	
interrupt	specify which interrupt flag to get
SAI_FLAG_OUERR	FIFO overrun or underrun interrupt flag
SAI_FLAG_MTDET	mute detection interrupt flag
SAI_FLAG_ERRCK	error clock interrupt enable
SAI_FLAG_FFREQ	FIFO request interrupt flag
SAI_FLAG_ACNRDY	audio codec not ready interrupt flag
SAI_FLAG_FSADET	frame synchronization advanced detection interrupt flag
SAI_FLAG_FSPDET	frame synchronization postpone detection interrupt flag
Output parameter{out}	
-	-
Return value	
FlagStatus	SET or RESET

Example:

```
/* get SAI block0 intrerrupt flag status*/
```

```
FlagStatus flag_state = RESET;
```

```
flag_state = sai_interrupt_flag_get(SAI_BLOCK0, SAI_FLAG_MTDET);
```

sai_interrupt_flag_clear

The description of sai_interrupt_flag_clear is shown as below:

Table 3-1247. Function sai_interrupt_flag_clear

Function name	sai_interrupt_flag_clear
Function prototype	void sai_interrupt_flag_clear(uint32_t block, uint32_t interrupt);
Function descriptions	clear the SAI interrupt flag
Precondition	-

The called functions	-
Input parameter{in}	
block	SAI block
<i>SAI_BLOCKx</i>	(x=0,1)
Input parameter{in}	
interrupt	specify which interrupt flag to clear
<i>SAI_FLAG_OUERR</i>	FIFO overrun or underrun interrupt flag
<i>SAI_FLAG_MTDET</i>	mute detection interrupt flag
<i>SAI_FLAG_ERRCK</i>	error clock interrupt enable
<i>SAI_FLAG_FFREQ</i>	FIFO request interrupt flag
<i>SAI_FLAG_ACNRDY</i>	audio codec not ready interrupt flag
<i>SAI_FLAG_FSADET</i>	frame synchronization advanced detection interrupt flag
<i>SAI_FLAG_FSPDET</i>	frame synchronization postpone detection interrupt flag
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* clear SAI block 0 error clock intrerrupt flag */
```

```
sai_flag_clear (SAI_BLOCK0, SAI_FLAG_ERRCK);
```

sai_flag_get

The description of sai_flag_get is shown as below:

Table 3-1248. Function sai_flag_get

Function name	sai_flag_get
Function prototype	FlagStatus sai_flag_get(uint32_t block, uint32_t flag);
Function descriptions	get the SAI flag
Precondition	-
The called functions	-
Input parameter{in}	
block	SAI block
<i>SAI_BLOCKx</i>	(x=0,1)
Input parameter{in}	
interrupt	specify which flag to get
<i>SAI_FLAG_OUERR</i>	FIFO overrun or underrun flag
<i>SAI_FLAG_MTDET</i>	mute detection flag
<i>SAI_FLAG_ERRCK</i>	error clock enable
<i>SAI_FLAG_FFREQ</i>	FIFO request flag
<i>SAI_FLAG_ACNRDY</i>	audio codec not ready flag
<i>SAI_FLAG_FSADET</i>	frame synchronization advanced detection flag

<i>SAI_FLAG_FSPDET</i>	frame synchronization postpone detection flag
Output parameter{out}	
-	-
Return value	
FlagStatus	SET or RESET

Example:

```
/* get SAI block0 flag status*/
```

```
FlagStatus flag_state = RESET;
```

```
flag_state = sai_flag_get(SAI_BLOCK0, SAI_FLAG_MTDET);
```

sai_flag_clear

The description of sai_flag_clear is shown as below:

Table 3-1249. Function sai_flag_clear

Function name	sai_flag_clear
Function prototype	void sai_flag_clear(uint32_t block, uint32_t flag);
Function descriptions	clear the SAI flag
Precondition	-
The called functions	-
Input parameter{in}	
block	SAI block
<i>SAI_BLOCKx</i>	(x=0,1)
Input parameter{in}	
interrupt	specify which flag to clear
<i>SAI_FLAG_OUERR</i>	FIFO overrun or underrun flag
<i>SAI_FLAG_MTDET</i>	mute detection flag
<i>SAI_FLAG_ERRCK</i>	error clock flag
<i>SAI_FLAG_ACNRDY</i>	audio codec not ready flag
<i>SAI_FLAG_FSADET</i>	frame synchronization advanced detection flag
<i>SAI_FLAG_FSPDET</i>	frame synchronization postpone detection flag
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* clear SAI block 0 error clock flag */
```

```
sai_flag_clear (SAI_BLOCK0, SAI_FLAG_ERRCK);
```

4. Revision history

Table 4-1. Revision history

Revision No.	Description	Date
1.0	Rev1.0 formal release	Dec.22, 2023
1.1	Changed the GD32F5xx to GD32F527	Aug.08, 2025

Important Notice

This document is the property of GigaDevice Semiconductor Inc. and its subsidiaries (the "Company"). This document, including any product of the Company described in this document (the "Product"), is owned by the Company according to the laws of the People's Republic of China and other applicable laws. The Company reserves all rights under such laws and no Intellectual Property Rights are transferred (either wholly or partially) or licensed by the Company (either expressly or impliedly) herein. The names and brands of third party referred thereto (if any) are the property of their respective owner and referred to for identification purposes only.

To the maximum extent permitted by applicable law, the Company makes no representations or warranties of any kind, express or implied, with regard to the merchantability and the fitness for a particular purpose of the Product, nor does the Company assume any liability arising out of the application or use of any Product. Any information provided in this document is provided only for reference purposes. It is the sole responsibility of the user of this document to determine whether the Product is suitable and fit for its applications and products planned, and properly design, program, and test the functionality and safety of its applications and products planned using the Product. The Product is designed, developed, and/or manufactured for ordinary business, industrial, personal, and/or household applications only, and the Product is not designed or intended for use in (i) safety critical applications such as weapons systems, nuclear facilities, atomic energy controller, combustion controller, aeronautic or aerospace applications, traffic signal instruments, pollution control or hazardous substance management; (ii) life-support systems, other medical equipment or systems (including life support equipment and surgical implants); (iii) automotive applications or environments, including but not limited to applications for active and passive safety of automobiles (regardless of front market or aftermarket), for example, EPS, braking, ADAS (camera/fusion), EMS, TCU, BMS, BSG, TPMS, Airbag, Suspension, DMS, ICMS, Domain, ESC, DCDC, e-clutch, advanced-lighting, etc.. Automobile herein means a vehicle propelled by a self-contained motor, engine or the like, such as, without limitation, cars, trucks, motorcycles, electric cars, and other transportation devices; and/or (iv) other uses where the failure of the device or the Product can reasonably be expected to result in personal injury, death, or severe property or environmental damage (collectively "Unintended Uses"). Customers shall take any and all actions to ensure the Product meets the applicable laws and regulations. The Company is not liable for, in whole or in part, and customers shall hereby release the Company as well as its suppliers and/or distributors from, any claim, damage, or other liability arising from or related to all Unintended Uses of the Product. Customers shall indemnify and hold the Company, and its officers, employees, subsidiaries, affiliates as well as its suppliers and/or distributors harmless from and against all claims, costs, damages, and other liabilities, including claims for personal injury or death, arising from or related to any Unintended Uses of the Product.

Information in this document is provided solely in connection with the Product. The Company reserves the right to make changes, corrections, modifications or improvements to this document and the Product described herein at any time without notice. The Company shall have no responsibility whatsoever for conflicts or incompatibilities arising from future changes to them. Information in this document supersedes and replaces information previously supplied in any prior versions of this document.