

**GigaDevice Semiconductor Inc.**

**GD32E23x**  
**Arm® Cortex®-M23 32-bit MCU**

**Firmware Library**  
**User Guide**

Revison 1.0

(Dec. 2020)

# Table of Contents

<b>Table of Contents .....</b>	<b>1</b>
<b>List of Figures .....</b>	<b>4</b>
<b>List of Tables .....</b>	<b>5</b>
<b>1. Introduction .....</b>	<b>18</b>
<b>1.1. Rules of User Manual and Firmware Library .....</b>	<b>18</b>
1.1.1. Peripherals.....	18
1.1.2. Naming rules.....	19
<b>2. Firmware Library Overview .....</b>	<b>20</b>
<b>2.1. File Structure of Firmware Library .....</b>	<b>20</b>
2.1.1. Examples Folder .....	21
2.1.2. Firmware Folder.....	21
2.1.3. Template Folder .....	21
2.1.4. Utilities Folder .....	24
<b>2.2. File descriptions of Firmware Library .....</b>	<b>24</b>
<b>3. Firmware Library of Standard Peripherals .....</b>	<b>26</b>
<b>3.1. Overview of Firmware Library of Standard Peripherals.....</b>	<b>26</b>
<b>3.2. ADC .....</b>	<b>26</b>
3.2.1. Descriptions of Peripheral registers.....	26
3.2.2. Descriptions of Peripheral functions .....	27
<b>3.3. CMP .....</b>	<b>49</b>
3.3.1. Descriptions of Peripheral registers.....	49
3.3.2. Descriptions of Peripheral functions .....	49
<b>3.4. CRC .....</b>	<b>55</b>
3.4.1. Descriptions of Peripheral registers.....	55
3.4.2. Descriptions of Peripheral functions .....	56
<b>3.5. DBG .....</b>	<b>63</b>
3.5.1. Descriptions of Peripheral registers.....	63
3.5.2. Descriptions of Peripheral functions .....	63
<b>3.6. DMA .....</b>	<b>68</b>
3.6.1. Descriptions of Peripheral registers.....	68
3.6.2. Descriptions of Peripheral functions .....	68
<b>3.7. EXTI .....</b>	<b>85</b>
3.7.1. Descriptions of Peripheral registers.....	85

3.7.2.	Descriptions of Peripheral functions .....	86
<b>3.8. FMC .....</b>		<b>93</b>
3.8.1.	Descriptions of Peripheral registers.....	94
3.8.2.	Descriptions of Peripheral functions .....	94
<b>3.9. FWDGT .....</b>		<b>111</b>
3.9.1.	Descriptions of Peripheral registers.....	111
3.9.2.	Descriptions of Peripheral functions .....	111
<b>3.10. GPIO .....</b>		<b>117</b>
3.10.1.	Descriptions of Peripheral registers.....	117
3.10.2.	Descriptions of Peripheral functions .....	117
<b>3.11. I2C .....</b>		<b>128</b>
3.11.1.	Descriptions of Peripheral registers.....	128
3.11.2.	Descriptions of Peripheral functions .....	128
<b>3.12. MISC .....</b>		<b>152</b>
3.12.1.	Descriptions of Peripheral registers.....	152
3.12.2.	Descriptions of Peripheral functions .....	153
<b>3.13. PMU .....</b>		<b>158</b>
3.13.1.	Descriptions of Peripheral registers.....	158
3.13.2.	Descriptions of Peripheral functions .....	158
<b>3.14. RCU .....</b>		<b>166</b>
3.14.1.	Descriptions of Peripheral registers.....	166
3.14.2.	Descriptions of Peripheral functions .....	166
<b>3.15. RTC .....</b>		<b>191</b>
3.15.1.	Descriptions of Peripheral registers.....	192
3.15.2.	Descriptions of Peripheral functions .....	192
<b>3.16. SPI .....</b>		<b>213</b>
3.16.1.	Descriptions of Peripheral registers.....	213
3.16.2.	Descriptions of Peripheral functions .....	213
<b>3.17. SYSCFG .....</b>		<b>242</b>
3.17.1.	Descriptions of Peripheral registers.....	242
3.17.2.	Descriptions of Peripheral functions .....	242
<b>3.18. TIMER .....</b>		<b>248</b>
3.18.1.	Descriptions of Peripheral registers.....	248
3.18.2.	Descriptions of Peripheral functions .....	249
<b>3.19. USART .....</b>		<b>307</b>
3.19.1.	Descriptions of Peripheral registers.....	307
3.19.2.	Descriptions of Peripheral functions .....	307
<b>3.20. WWDGT .....</b>		<b>357</b>
3.20.1.	Descriptions of Peripheral registers.....	357

---

3.20.2. Descriptions of Peripheral functions .....	357
<b>4. Revision history .....</b>	<b>362</b>

## List of Figures

<b>Figure 2-1. File structure of firmware library of GD32E23x.....</b>	<b>20</b>
<b>Figure 2-2. Select peripheral example files.....</b>	<b>22</b>
<b>Figure 2-3. Copy the peripheral example files .....</b>	<b>22</b>
<b>Figure 2-4. Open the project file.....</b>	<b>23</b>
<b>Figure 2-5. Configure project files .....</b>	<b>23</b>
<b>Figure 2-6. Compile-debug-download .....</b>	<b>24</b>

## List of Tables

<b>Table 1-1. Peripherals.....</b>	<b>18</b>
<b>Table 2-1. Function descriptions of Firmware Library.....</b>	<b>24</b>
<b>Table 3-1. Peripheral function format of Firmware Library .....</b>	<b>26</b>
<b>Table 3-2. ADC Registers .....</b>	<b>26</b>
<b>Table 3-3. ADC firmware function .....</b>	<b>27</b>
<b>Table 3-4. Function adc_deinit .....</b>	<b>28</b>
<b>Table 3-5. Function adc_enable .....</b>	<b>28</b>
<b>Table 3-6. Function adc_disable .....</b>	<b>29</b>
<b>Table 3-7. Function adc_calibration_enable .....</b>	<b>29</b>
<b>Table 3-8. Function adc_dma_mode_enable .....</b>	<b>30</b>
<b>Table 3-9. Function adc_dma_mode_disable .....</b>	<b>30</b>
<b>Table 3-10. Function adc_tempsensor_vrefint_enable.....</b>	<b>30</b>
<b>Table 3-11. Function adc_tempsensor_vrefint_disable .....</b>	<b>31</b>
<b>Table 3-12. Function adc_discontinuous_mode_config.....</b>	<b>31</b>
<b>Table 3-13. Function adc_special_function_config .....</b>	<b>32</b>
<b>Table 3-14. Function adc_data_alignment_config .....</b>	<b>33</b>
<b>Table 3-15. Function adc_channel_length_config .....</b>	<b>33</b>
<b>Table 3-16. Function adc_regular_channel_config .....</b>	<b>34</b>
<b>Table 3-17. Function adc_inserted_channel_config .....</b>	<b>35</b>
<b>Table 3-18. Function adc_inserted_channel_offset_config .....</b>	<b>36</b>
<b>Table 3-19. Function adc_external_trigger_config.....</b>	<b>37</b>
<b>Table 3-20. Function adc_external_trigger_source_config.....</b>	<b>37</b>
<b>Table 3-21. Function adc_software_trigger_enable .....</b>	<b>39</b>
<b>Table 3-22. Function adc_regular_data_read .....</b>	<b>39</b>
<b>Table 3-23. Function adc_inserted_data_read.....</b>	<b>40</b>
<b>Table 3-24. Function adc_flag_get .....</b>	<b>40</b>
<b>Table 3-25. Function adc_flag_clear .....</b>	<b>41</b>
<b>Table 3-26. Function adc_interrupt_flag_get .....</b>	<b>42</b>
<b>Table 3-27. Function adc_interrupt_flag_clear .....</b>	<b>42</b>
<b>Table 3-28. Function adc_interrupt_enable .....</b>	<b>43</b>
<b>Table 3-29. Function adc_interrupt_disable .....</b>	<b>43</b>
<b>Table 3-30. Function adc_watchdog_single_channel_enable .....</b>	<b>44</b>
<b>Table 3-31. Function adc_watchdog_group_channel_enable .....</b>	<b>44</b>
<b>Table 3-32. Function adc_watchdog_disable .....</b>	<b>45</b>
<b>Table 3-33. Function adc_watchdog_threshold_config.....</b>	<b>45</b>
<b>Table 3-34. Function adc_resolution_config .....</b>	<b>46</b>
<b>Table 3-35. Function adc_oversample_mode_config .....</b>	<b>47</b>
<b>Table 3-36. Function adc_oversample_mode_enable.....</b>	<b>48</b>
<b>Table 3-37. Function adc_oversample_mode_disable.....</b>	<b>49</b>
<b>Table 3-38. CMP Registers .....</b>	<b>49</b>

<b>Table 3-39. CMP firmware function .....</b>	<b>50</b>
<b>Table 3-40. Function cmp_deinit .....</b>	<b>50</b>
<b>Table 3-41. Function cmp_mode_init.....</b>	<b>50</b>
<b>Table 3-42. Function cmp_output_init .....</b>	<b>51</b>
<b>Table 3-43. Function can_fd_init .....</b>	<b>52</b>
<b>Table 3-44. Function cmp_disable .....</b>	<b>53</b>
<b>Table 3-45. Function cmp_switch_enable .....</b>	<b>53</b>
<b>Table 3-46. Function cmp_switch_disable .....</b>	<b>54</b>
<b>Table 3-47. Function cmp_output_level_get .....</b>	<b>54</b>
<b>Table 3-48. Function cmp_lock_enable .....</b>	<b>55</b>
<b>Table 3-49. CRC Registers .....</b>	<b>56</b>
<b>Table 3-50. CRC firmware function .....</b>	<b>56</b>
<b>Table 3-51. Function crc_deinit .....</b>	<b>56</b>
<b>Table 3-52. Function crc_reverse_output_data_enable.....</b>	<b>57</b>
<b>Table 3-53. Function crc_reverse_output_data_disable.....</b>	<b>57</b>
<b>Table 3-54. Function crc_data_register_reset .....</b>	<b>58</b>
<b>Table 3-55. Function crc_data_register_read .....</b>	<b>58</b>
<b>Table 3-56. Function crc_free_data_register_read.....</b>	<b>59</b>
<b>Table 3-57. Function crc_free_data_register_write.....</b>	<b>59</b>
<b>Table 3-58. Function crc_init_data_register_write .....</b>	<b>60</b>
<b>Table 3-59. Function crc_input_data_reverse_config.....</b>	<b>60</b>
<b>Table 3-60. Function crc_polynomial_size_set .....</b>	<b>61</b>
<b>Table 3-61. Function crc_polynomial_set .....</b>	<b>61</b>
<b>Table 3-62. Function crc_single_data_calculate .....</b>	<b>62</b>
<b>Table 3-63. Function crc_block_data_calculate .....</b>	<b>62</b>
<b>Table 3-64. DBG Registers .....</b>	<b>63</b>
<b>Table 3-65. DBG firmware function .....</b>	<b>64</b>
<b>Table 3-66. Enum dbg_periph_enum .....</b>	<b>64</b>
<b>Table 3-67. Function dbg_deinit .....</b>	<b>64</b>
<b>Table 3-68. Function dbg_id_get.....</b>	<b>65</b>
<b>Table 3-69. Function dbg_low_power_enable .....</b>	<b>65</b>
<b>Table 3-70. Function dbg_low_power_disable .....</b>	<b>66</b>
<b>Table 3-71. Function dbg_periph_enable .....</b>	<b>66</b>
<b>Table 3-72. Function dbg_periph_disable .....</b>	<b>67</b>
<b>Table 3-73. DMA Registers .....</b>	<b>68</b>
<b>Table 3-74. DMA firmware function .....</b>	<b>68</b>
<b>Table 3-75. Structure dma_parameter_struct .....</b>	<b>69</b>
<b>Table 3-76. Function dma_deinit .....</b>	<b>69</b>
<b>Table 3-77. Function dma_para_init.....</b>	<b>70</b>
<b>Table 3-78. Function dma_init .....</b>	<b>70</b>
<b>Table 3-79. Function dma_circulation_enable .....</b>	<b>71</b>
<b>Table 3-80. Function dma_circulation_disable .....</b>	<b>72</b>
<b>Table 3-81. Function dma_memory_to_memory_enable.....</b>	<b>72</b>
<b>Table 3-82. Function dma_memory_to_memory_disable.....</b>	<b>73</b>

<b>Table 3-83. Function dma_channel_enable .....</b>	<b>73</b>
<b>Table 3-84. Function dma_channel_disable .....</b>	<b>74</b>
<b>Table 3-85. Function dma_periph_address_config.....</b>	<b>74</b>
<b>Table 3-86. Function dma_memory_address_config.....</b>	<b>75</b>
<b>Table 3-87. Function dma_transfer_number_config .....</b>	<b>75</b>
<b>Table 3-88. Function dma_transfer_number_get .....</b>	<b>76</b>
<b>Table 3-89. Function dma_priority_config .....</b>	<b>77</b>
<b>Table 3-90. Function dma_memory_width_config .....</b>	<b>77</b>
<b>Table 3-91. Function dma_periph_width_config .....</b>	<b>78</b>
<b>Table 3-92. Function dma_memory_increase_enable .....</b>	<b>79</b>
<b>Table 3-93. Function dma_memory_increase_disable .....</b>	<b>79</b>
<b>Table 3-94. Function dma_periph_increase_enable .....</b>	<b>80</b>
<b>Table 3-95. Function dma_periph_increase_disable .....</b>	<b>80</b>
<b>Table 3-96. Function dma_transfer_direction_config .....</b>	<b>81</b>
<b>Table 3-97. Function dma_flag_get.....</b>	<b>81</b>
<b>Table 3-98. Function dma_flag_clear.....</b>	<b>82</b>
<b>Table 3-99. Function dma_interrupt_flag_get .....</b>	<b>83</b>
<b>Table 3-100. Function dma_interrupt_flag_clear .....</b>	<b>83</b>
<b>Table 3-101. Function dma_interrupt_enable .....</b>	<b>84</b>
<b>Table 3-102. Function dma_interrupt_disable .....</b>	<b>85</b>
<b>Table 3-103. EXTI Registers .....</b>	<b>85</b>
<b>Table 3-104. EXTI firmware function .....</b>	<b>86</b>
<b>Table 3-105. exti_line_enum .....</b>	<b>86</b>
<b>Table 3-106. exti_mode_enum.....</b>	<b>87</b>
<b>Table 3-107. exti_trig_type_enum .....</b>	<b>87</b>
<b>Table 3-108. Function exti_deinit .....</b>	<b>87</b>
<b>Table 3-109. Function exti_init .....</b>	<b>88</b>
<b>Table 3-110. Function exti_interrupt_enable.....</b>	<b>88</b>
<b>Table 3-111. Function exti_event_enable .....</b>	<b>89</b>
<b>Table 3-112. Function exti_interrupt_disable.....</b>	<b>89</b>
<b>Table 3-113. Function exti_event_disable .....</b>	<b>90</b>
<b>Table 3-114. Function exti_flag_get .....</b>	<b>90</b>
<b>Table 3-115. Function exti_flag_clear .....</b>	<b>91</b>
<b>Table 3-116. Function exti_interrupt_flag_get .....</b>	<b>91</b>
<b>Table 3-117. Function exti_interrupt_flag_clear .....</b>	<b>92</b>
<b>Table 3-118. Function exti_software_interrupt_enable .....</b>	<b>92</b>
<b>Table 3-119. Function exti_software_interrupt_disable .....</b>	<b>93</b>
<b>Table 3-120. FMC Registers .....</b>	<b>94</b>
<b>Table 3-121. FMC firmware function .....</b>	<b>94</b>
<b>Table 3-122. fmc_state_enum.....</b>	<b>95</b>
<b>Table 3-123. Function fmc_unlock .....</b>	<b>95</b>
<b>Table 3-124. Function fmc_lock .....</b>	<b>96</b>
<b>Table 3-125. Function fmc_wscnt_set .....</b>	<b>96</b>
<b>Table 3-126. Function fmc_prefetch_enable .....</b>	<b>97</b>

<b>Table 3-127. Function fmc_prefetch_disable .....</b>	<b>97</b>
<b>Table 3-128. Function fmc_page_erase .....</b>	<b>98</b>
<b>Table 3-129. Function fmc_mass_erase .....</b>	<b>98</b>
<b>Table 3-130. Function fmc_doubleword_program .....</b>	<b>99</b>
<b>Table 3-131. Function fmc_word_program .....</b>	<b>99</b>
<b>Table 3-132. Function ob_unlock .....</b>	<b>100</b>
<b>Table 3-133. Function ob_lock .....</b>	<b>100</b>
<b>Table 3-134. Function ob_reset .....</b>	<b>101</b>
<b>Table 3-135. Function option_byte_value_get .....</b>	<b>101</b>
<b>Table 3-136. Function ob_erase .....</b>	<b>102</b>
<b>Table 3-137. Function ob_write_protection_enable .....</b>	<b>102</b>
<b>Table 3-138. Function ob_security_protection_config .....</b>	<b>103</b>
<b>Table 3-139. Function ob_user_write .....</b>	<b>103</b>
<b>Table 3-140. Function ob_data_program .....</b>	<b>104</b>
<b>Table 3-141. Function ob_user_get .....</b>	<b>104</b>
<b>Table 3-142. Function ob_data_get .....</b>	<b>105</b>
<b>Table 3-143. Function ob_write_protection_get .....</b>	<b>105</b>
<b>Table 3-144. Function ob_obstat_plevel_get .....</b>	<b>106</b>
<b>Table 3-145. Function fmc_interrupt_enable .....</b>	<b>106</b>
<b>Table 3-146. Function fmc_interrupt_disable .....</b>	<b>107</b>
<b>Table 3-147. Function fmc_flag_get .....</b>	<b>108</b>
<b>Table 3-148. Function fmc_flag_clear .....</b>	<b>108</b>
<b>Table 3-149. Function fmc_interrupt_flag_get .....</b>	<b>109</b>
<b>Table 3-150. Function fmc_interrupt_flag_clear .....</b>	<b>109</b>
<b>Table 3-151. Function fmc_state_get .....</b>	<b>110</b>
<b>Table 3-152. Function fmc_ready_wait .....</b>	<b>111</b>
<b>Table 3-153. FWDGT Registers .....</b>	<b>111</b>
<b>Table 3-154. FWDGT firmware function .....</b>	<b>112</b>
<b>Table 3-155. Function fwdgt_write_ensable .....</b>	<b>112</b>
<b>Table 3-156. Function fwdgt_write_disable .....</b>	<b>112</b>
<b>Table 3-157. Function fwdgt_enable .....</b>	<b>113</b>
<b>Table 3-158. Function fwdgt_prescaler_value_config .....</b>	<b>113</b>
<b>Table 3-159. Function fwdgt_reload_value_config .....</b>	<b>114</b>
<b>Table 3-160. Function fwdgt_window_value_config .....</b>	<b>114</b>
<b>Table 3-161. Function fwdgt_counter_reload .....</b>	<b>115</b>
<b>Table 3-162. Function fwdgt_config .....</b>	<b>115</b>
<b>Table 3-163. Function fwdgt_flag_get .....</b>	<b>116</b>
<b>Table 3-164. GPIO Registers .....</b>	<b>117</b>
<b>Table 3-165. GPIO firmware function .....</b>	<b>117</b>
<b>Table 3-166. Function gpio_deinit .....</b>	<b>118</b>
<b>Table 3-167. Function gpio_mode_set .....</b>	<b>118</b>
<b>Table 3-168. Function gpio_output_options_set .....</b>	<b>119</b>
<b>Table 3-169. Function gpio_bit_set .....</b>	<b>120</b>
<b>Table 3-170. Function gpio_bit_reset .....</b>	<b>121</b>

Table 3-171. Function gpio_bit_write.....	121
Table 3-172. Function gpio_port_write .....	122
Table 3-173. Function gpio_input_bit_get.....	123
Table 3-174. Function gpio_input_port_get .....	123
Table 3-175. Function gpio_output_bit_get .....	124
Table 3-176. Function gpio_output_port_get.....	124
Table 3-177. Function gpio_af_set .....	125
Table 3-178. Function gpio_pin_lock .....	126
Table 3-179. Function gpio_bit_toggle .....	127
Table 3-180. Function gpio_port_toggle.....	127
Table 3-181. I2C Registers .....	128
Table 3-182. I2C firmware function .....	128
Table 3-183. Function i2c_deinit .....	129
Table 3-184. Function i2c_clock_config .....	130
Table 3-185. Function i2c_mode_addr_config.....	130
Table 3-186. Function i2c_smbus_type_config .....	131
Table 3-187. Function i2c_ack_config .....	132
Table 3-188. Function i2c_ackpos_config.....	132
Table 3-189. Function i2c_master_addressing.....	133
Table 3-190. Function i2c_dualaddr_enable .....	134
Table 3-191. Function i2c_dualaddr_enable .....	134
Table 3-192. Function i2c_enable.....	135
Table 3-193. Function i2c_disable.....	135
Table 3-194. Function i2c_start_on_bus .....	136
Table 3-195. Function i2c_stop_on_bus.....	136
Table 3-196. Function i2c_data_transmit .....	137
Table 3-197. Function i2c_data_receive .....	137
Table 3-198. Function i2c_dma_enable .....	138
Table 3-199. Function i2c_dma_last_transfer_config .....	139
Table 3-200. Function i2c_stretch_scl_low_config .....	139
Table 3-201. Function i2c_slave_response_to_gcall_config .....	140
Table 3-202. Function i2c_software_reset_config.....	140
Table 3-203. Function i2c_pec_enable .....	141
Table 3-204. Function i2c_pec_transfer_enable .....	142
Table 3-205. Function i2c_pec_value_get .....	142
Table 3-206. Function i2c_smbus_issue_alert .....	143
Table 3-207. Function i2c_smbus_arp_enable .....	144
Table 3-208. Function i2c_sam_enable .....	144
Table 3-209. Function i2c_sam_disable .....	145
Table 3-210. Function i2c_sam_timeout_enable .....	145
Table 3-211. Function i2c_sam_timeout_disable .....	146
Table 3-212. Function i2c_flag_get .....	146
Table 3-213. Function i2c_flag_clear .....	147
Table 3-214. Function i2c_interrupt_enable.....	148

<b>Table 3-215. Function i2c_interrupt_disable.....</b>	<b>149</b>
<b>Table 3-216. Function i2c_interrupt_flag_get .....</b>	<b>150</b>
<b>Table 3-217. Function i2c_interrupt_flag_clear .....</b>	<b>151</b>
<b>Table 3-218. NVIC Registers .....</b>	<b>152</b>
<b>Table 3-219. SysTick Registers .....</b>	<b>153</b>
<b>Table 3-220. IRQn_Type.....</b>	<b>153</b>
<b>Table 3-221. MISC firmware function .....</b>	<b>154</b>
<b>Table 3-222. Function nvic_irq_enable.....</b>	<b>154</b>
<b>Table 3-223. Function nvic_irq_disable.....</b>	<b>154</b>
<b>Table 3-224. Function nvic_system_reset.....</b>	<b>155</b>
<b>Table 3-225. Function nvic_vector_table_set .....</b>	<b>155</b>
<b>Table 3-226. Function system_lowpower_set.....</b>	<b>156</b>
<b>Table 3-227. Function system_lowpower_reset .....</b>	<b>157</b>
<b>Table 3-228. Function systick_clksource_set.....</b>	<b>157</b>
<b>Table 3-229. PMU Registers .....</b>	<b>158</b>
<b>Table 3-230. PMU firmware function .....</b>	<b>158</b>
<b>Table 3-231. Function pmu_deinit.....</b>	<b>159</b>
<b>Table 3-232. Function pmu_lvd_select.....</b>	<b>159</b>
<b>Table 3-233. Function pmu_ldo_output_select .....</b>	<b>160</b>
<b>Table 3-234. Function pmu_lvd_disable.....</b>	<b>160</b>
<b>Table 3-235. Function pmu_to_standbymode .....</b>	<b>161</b>
<b>Table 3-236. Function pmu_to_deepsleepmode.....</b>	<b>161</b>
<b>Table 3-237. Function pmu_to_standbymode .....</b>	<b>162</b>
<b>Table 3-238. Function pmu_wakeup_pin_enable .....</b>	<b>162</b>
<b>Table 3-239. Function pmu_wakeup_pin_disable .....</b>	<b>163</b>
<b>Table 3-240. Function pmu_backup_write_enable.....</b>	<b>164</b>
<b>Table 3-241. Function pmu_backup_write_disable .....</b>	<b>164</b>
<b>Table 3-242. Function pmu_flag_clear.....</b>	<b>165</b>
<b>Table 3-243. Function pmu_flag_get.....</b>	<b>165</b>
<b>Table 3-244. RCU Registers .....</b>	<b>166</b>
<b>Table 3-245. RCU firmware function .....</b>	<b>166</b>
<b>Table 3-246. Function rcu_deinit.....</b>	<b>168</b>
<b>Table 3-247. Function rcu_periph_clock_enable.....</b>	<b>168</b>
<b>Table 3-248. Function rcu_periph_clock_disable.....</b>	<b>169</b>
<b>Table 3-249. Function rcu_periph_clock_sleep_enable .....</b>	<b>170</b>
<b>Table 3-250. Function rcu_periph_clock_sleep_disable .....</b>	<b>170</b>
<b>Table 3-251. Function rcu_periph_reset_enable .....</b>	<b>171</b>
<b>Table 3-252. Function rcu_periph_reset_disable .....</b>	<b>171</b>
<b>Table 3-253. Function rcu_bkp_reset_enable .....</b>	<b>172</b>
<b>Table 3-254. Function rcu_bkp_reset_disable .....</b>	<b>173</b>
<b>Table 3-255. Function rcu_system_clock_source_config .....</b>	<b>173</b>
<b>Table 3-256. Function rcu_system_clock_source_get .....</b>	<b>174</b>
<b>Table 3-257. Function rcu_ahb_clock_config .....</b>	<b>174</b>
<b>Table 3-258. Function rcu_apb1_clock_config .....</b>	<b>175</b>

Table 3-259. Function rcu_apb2_clock_config .....	175
Table 3-260. Function rcu_adc_clock_config .....	176
Table 3-261. Function rcu_ckout_config .....	176
Table 3-262. Function rcu_pll_config .....	177
Table 3-263. Function rcu_usart_clock_config .....	178
Table 3-264. Function rcu_rtc_clock_config .....	179
Table 3-265. Function rcu_hxtal_pdiv_config .....	179
Table 3-266. Function rcu_lxtal_drive_capability_config .....	180
Table 3-267. Function rcu_flag_get.....	180
Table 3-268. Function rcu_all_reset_flag_clear .....	181
Table 3-269. Function rcu_interrupt_flag_get.....	182
Table 3-270. Function rcu_interrupt_flag_clear.....	183
Table 3-271. Function rcu_interrupt_enable .....	183
Table 3-272. Function rcu_interrupt_disable .....	184
Table 3-273. Function rcu_osc1_stab_wait.....	185
Table 3-274. Function rcu_osc1_on .....	185
Table 3-275. Function rcu_osc1_off.....	186
Table 3-276. Function rcu_osc1_bypass_mode_enable.....	187
Table 3-277. Function rcu_osc1_bypass_mode_disable.....	187
Table 3-278. Function rcu_hxtal_clock_monitor_enable.....	188
Table 3-279. Function rcu_hxtal_clock_monitor_disable.....	188
Table 3-280. Function rcu_irc8m_adjust_value_set.....	189
Table 3-281. Function rcu_irc28m_adjust_value_set.....	189
Table 3-282. Function rcu_voltage_key_unlock .....	190
Table 3-283. Function rcu_deepsleep_voltage_set.....	190
Table 3-284. Function rcu_clock_freq_get .....	191
Table 3-285. RTC Registers .....	192
Table 3-286. RTC firmware function.....	192
Table 3-287. rtc_parameter_struct .....	193
Table 3-288. rtc_alarm_struct .....	194
Table 3-289. rtc_timestamp_struct.....	194
Table 3-290. rtc_tamper_struct.....	194
Table 3-291. Function rtc_deinit .....	195
Table 3-292. Function rtc_init .....	195
Table 3-293. Function rtc_init_mode_enter .....	196
Table 3-294. Function rtc_init_mode_exit .....	196
Table 3-295. Function rtc_register_sync_wait .....	197
Table 3-296. Function rtc_current_time_get .....	197
Table 3-297. Function rtc_subsecond_get .....	198
Table 3-298. Function rtc_alarm_config .....	198
Table 3-299. Function rtc_alarm_subsecond_config .....	199
Table 3-300. Function rtc_alarm_enable .....	200
Table 3-301. Function rtc_alarm_disable .....	200
Table 3-302. Function rtc_alarm_get .....	201

Table 3-303. Function <code>rtc_alarm_subsecond_get</code> .....	202
Table 3-304. Function <code>rtc_timestamp_enable</code> .....	202
Table 3-305. Function <code>rtc_timestamp_disable</code> .....	203
Table 3-306. Function <code>rtc_timestamp_get</code> .....	203
Table 3-307. Function <code>rtc_timestamp_subsecond_get</code> .....	204
Table 3-308. Function <code>rtc_timestamp_enable</code> .....	204
Table 3-309. Function <code>rtc_tamper_disable</code> .....	205
Table 3-310. Function <code>rtc_interrupt_enable</code> .....	205
Table 3-311. Function <code>rtc_interrupt_disable</code> .....	206
Table 3-312. Function <code>rtc_flag_get</code> .....	206
Table 3-313. Function <code>rtc_flag_clear</code> .....	207
Table 3-314. Function <code>rtc_alter_output_config</code> .....	208
Table 3-315. <code>rtc_calibration_config</code> .....	209
Table 3-316. <code>rtc_hour_adjust</code> .....	209
Table 3-317. <code>rtc_second_adjust</code> .....	210
Table 3-318. <code>rtc_bypass_shadow_enable</code> .....	211
Table 3-319. <code>rtc_bypass_shadow_disable</code> .....	211
Table 3-320. <code>rtc_refclock_detection_enable</code> .....	212
Table 3-321. <code>rtc_refclock_detection_disable</code> .....	212
Table 3-322. SPI/I2S registers .....	213
Table 3-323. SPI/I2S firmware function .....	213
Table 3-324. <code>spi_parameter_struct</code> .....	214
Table 3-325. Function <code>spi_i2s_deinit</code> .....	215
Table 3-326. Function <code>spi_i2s_deinit</code> .....	215
Table 3-327. Function <code>spi_init</code> .....	216
Table 3-328. Function <code>spi_enable</code> .....	217
Table 3-329. Function <code>spi_disable</code> .....	217
Table 3-330. Function <code>i2s_init</code> .....	218
Table 3-331. Function <code>i2s_psc_config</code> .....	219
Table 3-332. Function <code>i2s_enable</code> .....	220
Table 3-333. Function <code>i2s_disable</code> .....	221
Table 3-334. Function <code>spi_nss_output_enable</code> .....	221
Table 3-335. Function <code>spi_nss_output_disable</code> .....	222
Table 3-336. Function <code>spi_nss_internal_high</code> .....	222
Table 3-337. Function <code>spi_nss_internal_low</code> .....	223
Table 3-338. Function <code>spi_dma_enable</code> .....	223
Table 3-339. Function <code>spi_dma_disable</code> .....	224
Table 3-340. Function <code>spi_i2s_data_frame_format_config</code> .....	225
Table 3-341. Function <code>spi_i2s_data_transmit</code> .....	225
Table 3-342. Function <code>spi_i2s_data_receive</code> .....	226
Table 3-343. Function <code>spi_bidirectional_transfer_config</code> .....	226
Table 3-344. Function <code>spi_crc_polynomial_set</code> .....	227
Table 3-345. Function <code>spi_crc_polynomial_get</code> .....	227
Table 3-346. Function <code>spi_crc_on</code> .....	228

<b>Table 3-347. Function spi_crc_off .....</b>	<b>228</b>
<b>Table 3-348. Function spi_crc_next .....</b>	<b>229</b>
<b>Table 3-349. Function spi_crc_get .....</b>	<b>230</b>
<b>Table 3-350. Function spi_ti_mode_enable .....</b>	<b>230</b>
<b>Table 3-351. Function spi_ti_mode_disable .....</b>	<b>231</b>
<b>Table 3-352. Function spi_nssp_mode_enable .....</b>	<b>231</b>
<b>Table 3-353. Function spi_nssp_mode_disable .....</b>	<b>232</b>
<b>Table 3-354. Function qspi_enable .....</b>	<b>232</b>
<b>Table 3-355. Function qspi_disable .....</b>	<b>233</b>
<b>Table 3-356. Function qspi_write_enable .....</b>	<b>233</b>
<b>Table 3-357. Function qspi_read_enable .....</b>	<b>234</b>
<b>Table 3-358. Function qspi_io23_output_enable .....</b>	<b>234</b>
<b>Table 3-359. Function qspi_io23_output_disable .....</b>	<b>235</b>
<b>Table 3-360. Function spi_i2s_interrupt_enable .....</b>	<b>235</b>
<b>Table 3-361. Function spi_i2s_interrupt_disable .....</b>	<b>236</b>
<b>Table 3-362. Function spi_i2s_interrupt_flag_get .....</b>	<b>236</b>
<b>Table 3-363. Function spi_i2s_flag_get .....</b>	<b>237</b>
<b>Table 3-364. Function spi_crc_error_clear .....</b>	<b>239</b>
<b>Table 3-365. Function spi_fifo_access_size_config .....</b>	<b>239</b>
<b>Table 3-366. Function spi_transmit_odd_config .....</b>	<b>240</b>
<b>Table 3-367. Function spi_receive_odd_config .....</b>	<b>240</b>
<b>Table 3-368. Function spi_crc_length_set .....</b>	<b>241</b>
<b>Table 3-369. SYSCFG Registers .....</b>	<b>242</b>
<b>Table 3-370. SYSCFG firmware function .....</b>	<b>242</b>
<b>Table 3-371. Function syscfg_deinit .....</b>	<b>242</b>
<b>Table 3-372. Function syscfg_dma_remap_enable .....</b>	<b>243</b>
<b>Table 3-373. Function syscfg_dma_remap_disable .....</b>	<b>244</b>
<b>Table 3-374. Function syscfg_high_current_enable .....</b>	<b>244</b>
<b>Table 3-375. Function syscfg_high_current_disable .....</b>	<b>245</b>
<b>Table 3-376. Function syscfg_exti_line_config .....</b>	<b>245</b>
<b>Table 3-377. Function syscfg_lock_config .....</b>	<b>246</b>
<b>Table 3-378. Function irq_latency_set .....</b>	<b>247</b>
<b>Table 3-379. Function syscfg_flag_get .....</b>	<b>247</b>
<b>Table 3-380. Function syscfg_flag_clear .....</b>	<b>248</b>
<b>Table 3-381. TIMERx Registers .....</b>	<b>248</b>
<b>Table 3-382. TIMERx firmware function .....</b>	<b>249</b>
<b>Table 3-383. Structure timer_parameter_struct .....</b>	<b>252</b>
<b>Table 3-384. Structure timer_break_parameter_struct .....</b>	<b>252</b>
<b>Table 3-385. Structure timer_oc_parameter_struct .....</b>	<b>252</b>
<b>Table 3-386. Structure timer_ic_parameter_struct .....</b>	<b>253</b>
<b>Table 3-387. Function timer_deinit .....</b>	<b>253</b>
<b>Table 3-388. Function timer_struct_para_init .....</b>	<b>254</b>
<b>Table 3-389. Function timer_init .....</b>	<b>254</b>
<b>Table 3-390. Function timer_enable .....</b>	<b>255</b>

Table 3-391. Function timer_disable .....	255
Table 3-392. Function timer_auto_reload_shadow_enable .....	256
Table 3-393. Function timer_auto_reload_shadow_disable .....	257
Table 3-394. Function timer_update_event_enable .....	257
Table 3-395. Function timer_update_event_disable .....	258
Table 3-396. Function timer_counter_alignment .....	258
Table 3-397. Function timer_counter_up_direction .....	259
Table 3-398. timer_counter_down_direction .....	260
Table 3-399. Function timer_prescaler_config .....	260
Table 3-400. Function timer_repetition_value_config .....	261
Table 3-401. Function timer_autoreload_value_config .....	261
Table 3-402. Function timer_counter_value_config .....	262
Table 3-403. Function timer_counter_read .....	263
Table 3-404. Function timer_prescaler_read .....	263
Table 3-405. Function timer_single_pulse_mode_config .....	264
Table 3-406. Function timer_update_source_config .....	264
Table 3-407. Function timer_ocpre_clear_source_config .....	265
Table 3-408. Function timer_interrupt_enable .....	266
Table 3-409. Function timer_interrupt_disable .....	267
Table 3-410. Function timer_interrupt_flag_get .....	267
Table 3-411. Function timer_interrupt_flag_clear .....	268
Table 3-412. Function timer_flag_get .....	269
Table 3-413. Function timer_flag_clear .....	270
Table 3-414. Function timer_dma_enable .....	271
Table 3-415. Function timer_dma_disable .....	271
Table 3-416. Function timer_channel_dma_request_source_select .....	272
Table 3-417. Function timer_dma_transfer_config .....	273
Table 3-418. Function timer_event_software_generate .....	274
Table 3-419. Function timer_break_struct_para_init .....	275
Table 3-420. Function timer_break_config .....	276
Table 3-421. Function timer_break_enable .....	277
Table 3-422. Function timer_break_disable .....	277
Table 3-423. Function timer_automatic_output_enable .....	278
Table 3-424. Function timer_automatic_output_disable .....	278
Table 3-425. Function timer_primary_output_config .....	279
Table 3-426. Function timer_channel_control_shadow_config .....	280
Table 3-427. Function timer_channel_control_shadow_update_config .....	280
Table 3-428. Function timer_channel_output_struct_para_init .....	281
Table 3-429. Function timer_channel_output_config .....	281
Table 3-430. Function timer_channel_output_mode_config .....	282
Table 3-431. Function timer_channel_output_pulse_value_config .....	283
Table 3-432. Function timer_channel_output_shadow_config .....	284
Table 3-433. Function timer_channel_output_fast_config .....	285
Table 3-434. Function timer_channel_output_clear_config .....	286

Table 3-435. Function timer_channel_output_polarity_config .....	287
Table 3-436. Function timer_channel_complementary_output_polarity_config.....	288
Table 3-437. Function timer_channel_output_state_config.....	288
Table 3-438. Function timer_channel_complementary_output_state_config .....	289
Table 3-439. Function timer_channel_input_struct_para_init.....	290
Table 3-440. Function timer_input_capture_config .....	291
Table 3-441. Function timer_channel_input_capture_prescaler_config.....	291
Table 3-442. Function timer_channel_capture_value_register_read .....	292
Table 3-443. Function timer_input_pwm_capture_config .....	293
Table 3-444. Function timer_hall_mode_config .....	294
Table 3-445. Function timer_input_trigger_source_select.....	294
Table 3-446. Function timer_master_output_trigger_source_select .....	295
Table 3-447. Function timer_slave_mode_select .....	297
Table 3-448. Function timer_master_slave_mode_config.....	297
Table 3-449. Function timer_external_trigger_config .....	298
Table 3-450. Function timer_quadrature_decoder_mode_config.....	299
Table 3-451. Function timer_internal_clock_config .....	300
Table 3-452. Function timer_internal_trigger_as_external_clock_config.....	301
Table 3-453. Function timer_external_trigger_as_external_clock_config.....	301
Table 3-454. Function timer_external_clock_mode0_config.....	302
Table 3-455. Function timer_external_clock_mode1_config.....	303
Table 3-456. Function timer_external_clock_mode1_disable .....	304
Table 3-457. Function timer_channel_remap_config .....	305
Table 3-458. Function timer_write_chxval_register_config .....	305
Table 3-459. Function timer_output_value_selection_config .....	306
Table 3-460. USART Registers .....	307
Table 3-461. USART firmware function.....	307
Table 3-462. Enum usart_flag_enum .....	310
Table 3-463. Enum usart_interrupt_flag_enum .....	310
Table 3-464. Enum usart_interrupt_enum .....	311
Table 3-465. Enum usart_invert_enum .....	311
Table 3-466. Function usart_deinit.....	312
Table 3-467. Function usart_baudrate_set.....	312
Table 3-468. Function usart_parity_config .....	313
Table 3-469. Function usart_word_length_set .....	313
Table 3-470. Function usart_stop_bit_set .....	314
Table 3-471. Function usart_enable.....	315
Table 3-472. Function usart_disable.....	315
Table 3-473. Function usart_transmit_config .....	316
Table 3-474. Function usart_receive_config .....	316
Table 3-475. Function usart_data_first_config .....	317
Table 3-476. Function usart_invert_config .....	317
Table 3-477. Function usart_overrun_enable .....	318
Table 3-478. Function usart_overrun_disable .....	319

Table 3-479. Function usart_oversample_config .....	319
Table 3-480. Function usart_sample_bit_config .....	320
Table 3-481. Function usart_receiver_timeout_enable.....	321
Table 3-482. Function usart_receiver_timeout_disable.....	321
Table 3-483. Function usart_receiver_timeout_threshold_config .....	322
Table 3-484. Function usart_data_transmit .....	322
Table 3-485. Function usart_data_receive .....	323
Table 3-486. Function usart_autobaud_detection_enable .....	323
Table 3-487. Function usart_autobaud_detection_disable .....	324
Table 3-488. Function usart_autobaud_detection_mode_config .....	324
Table 3-489. Function usart_address_config .....	325
Table 3-490. Function usart_address_detection_mode_config.....	325
Table 3-491. Function usart_mute_mode_enable .....	326
Table 3-492. Function usart_mute_mode_disable .....	327
Table 3-493. Function usart_mute_mode_wakeup_config.....	327
Table 3-494. Function usart_lin_mode_enable .....	328
Table 3-495. Function usart_lin_mode_disable .....	328
Table 3-496. Function usart_lin_break_dection_length_config.....	329
Table 3-497. Function usart_halfduplex_enable.....	329
Table 3-498. Function usart_halfduplex_disable.....	330
Table 3-499. Function usart_clock_enable .....	330
Table 3-500. Function usart_clock_disable .....	331
Table 3-501. Function usart_synchronous_clock_config .....	331
Table 3-502. Function usart_guard_time_config .....	332
Table 3-503. Function usart_smartcard_mode_enable.....	333
Table 3-504. Function usart_smartcard_mode_disable.....	333
Table 3-505. Function usart_smartcard_mode_nack_enable .....	334
Table 3-506. Function usart_smartcard_mode_nack_disable .....	334
Table 3-507. Function usart_smartcard_mode_early_nack_enable .....	335
Table 3-508. Function usart_smartcard_mode_early_nack_disable .....	335
Table 3-509. Function usart_smartcard_autoretry_config .....	336
Table 3-510. Function usart_block_length_config .....	336
Table 3-511. Function usart_irda_mode_enable.....	337
Table 3-512. Function usart_irda_mode_disable .....	338
Table 3-513. Function usart_prescaler_config .....	338
Table 3-514. Function usart_irda_lowpower_config .....	339
Table 3-515. Function usart_hardware_flow_rts_config .....	339
Table 3-516. Function usart_hardware_flow_cts_config.....	340
Table 3-517. Function usart_hardware_flow_coherence_config .....	341
Table 3-518. Function usart_rs485_driver_enable .....	341
Table 3-519. Function usart_rs485_driver_disable .....	342
Table 3-520. Function usart_driver_assertime_config .....	342
Table 3-521. Function usart_driver_deassertime_config .....	343
Table 3-522. Function usart_depolarity_config .....	343

---

Table 3-523. Function usart_dma_receive_config .....	344
Table 3-524. Function usart_dma_transmit_config.....	345
Table 3-525. Function usart_reception_error_dma_disable .....	345
Table 3-526. Function usart_reception_error_dma_enable .....	346
Table 3-527. Function usart_wakeup_enable .....	346
Table 3-528. Function usart_wakeup_disable .....	347
Table 3-529. Function usart_wakeup_mode_config .....	347
Table 3-530. Function usart_receive_fifo_enable.....	348
Table 3-531. Function usart_receive_fifo_disable.....	349
Table 3-532. Function usart_receive_fifo_counter_number .....	349
Table 3-533. Function usart_flag_get .....	350
Table 3-534. Function usart_flag_clear .....	351
Table 3-535. Function usart_interrupt_enable .....	352
Table 3-536. Function usart_interrupt_disable .....	353
Table 3-537. Function usart_command_enable.....	354
Table 3-538. Function usart_interrupt_flag_get .....	354
Table 3-539. Function usart_interrupt_flag_clear .....	356
Table 3-540. WWDGT Registers .....	357
Table 3-541. WWDGT firmware function.....	357
Table 3-542. Function wwdgt_deinit .....	358
Table 3-543. Function wwdgt_enable .....	358
Table 3-544. Function wwdgt_counter_update.....	359
Table 3-545. Function wwdgt_config .....	359
Table 3-546. Function wwdgt_interrupt_enable .....	360
Table 3-547. Function wwdgt_flag_get .....	360
Table 3-548. Function wwdgt_flag_clear .....	361
Table 4-1. Revision history .....	362

## 1. Introduction

This manual introduces firmware library of GD32E23x 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 GD32E23x 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
CMP	Comparator
CRC	CRC calculation unit
DBG	Debug

Peripherals	Descriptions
DMA	Direct memory access controller
EXTI	Interrupt/event controller
FMC	Flash memory controller
FWDGT	Free watchdog timer
GPIO/AFIO	General-purpose and alternate-function I/Os
I2C	Inter-integrated circuit interface
MISC	Nested Vectored Interrupt Controller
PMU	Power management unit
RCU	Reset and clock unit
RTC	Real-time Clock
SPI/I2S	Serial peripheral interface/Inter-IC sound
SYSCFG	System configuration
TIMER	TIMER
USART	Universal synchronous/asynchronous receiver /transmitter
WWDGT	Window watchdog timer

## 1.1.2. Naming rules

The firmware library naming rules are shown as below:

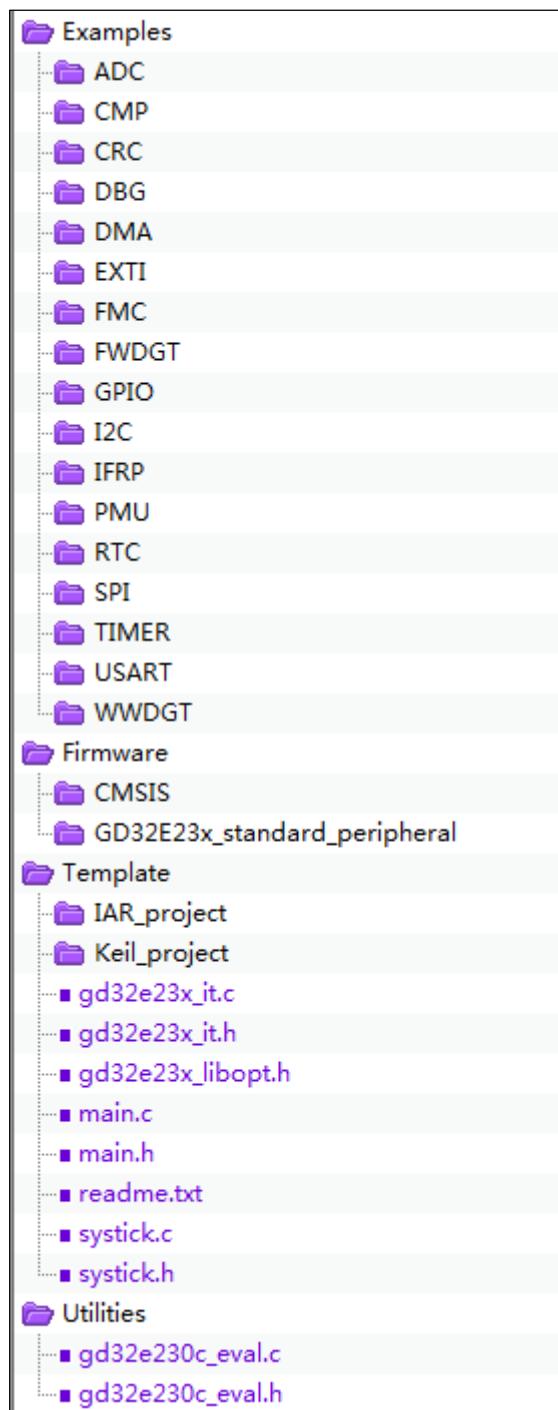
- The peripherals are shortened in XXX format, such as: ADC. More shorten information of peripherals refer to [Peripherals](#);
- The name of sourcefile and header file are started with “gd32e23x\_”, such as: gd32e23x\_adc.h;
- The constants used only in one file should be defined in the used file; the constants used in many files should be defined in corresponding header file. All the constants are written in uppercase of English letters;
- Registers are handled as constants. The naming of them are written in uppercase of English letters. In most cases, register names are shortened accord with the user manual;
- Variables are written in lowercase, when concluded by several words, underlines should be adapted among words;
- The naming of peripheral functions are started with the peripheral abbreviation added with an underline, when the function name is concluded by several words, underlines should be adapted among words, and all the peripheral functions are written in lowercase.

## 2. Firmware Library Overview

### 2.1. File Structure of Firmware Library

GD32E23x\_Firmware\_Library, the file structure is shown as below:

Figure 2-1. File structure of firmware library of GD32E23x



### 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;
- gd32e23x\_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);
- gd32e23x\_it.c: the source file include all the interrupt service routines (if no interrupt is used, then all the function bodies are empty);
- gd32e23x.it.h: the header file include all the prototypes of the interrupt service routines;
- systick.c: the source file include the precise time delay functions by using systick;
- systick.h: the header file include the prototype of the precise time delay functions by using systick;
- main.c: example code. Note: all the examples are not influenced by software IDEs.

### 2.1.2. Firmware Folder

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

- CMSIS subfolder includes the Cortex M23 kernel support files, the startup file based on the Cortex M23 kernel processor, the global header file of GD32E23x and system configuration file;
- GD32E23x\_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;

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

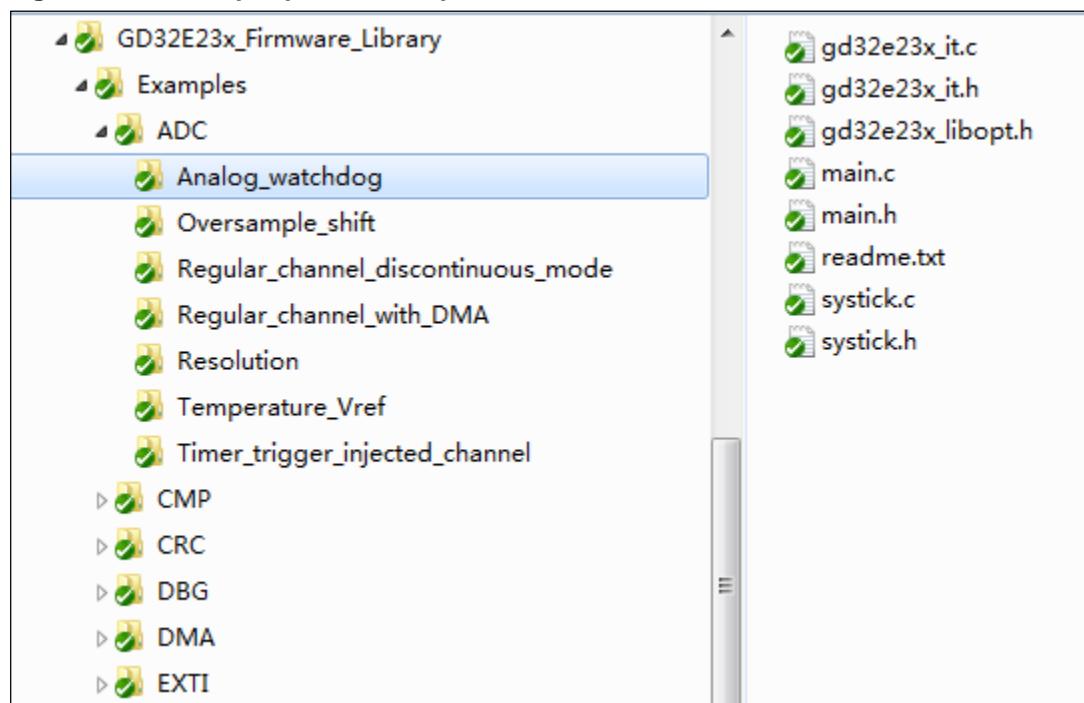
### 2.1.3. Template Folder

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

#### Select files

Open “Examples” folder, select the module to be tested, such as SPI, open “SPI” folder, select an example of SPI, such as “SPI\_master\_transmit\_slave\_receive\_interrupt”, shown as below:

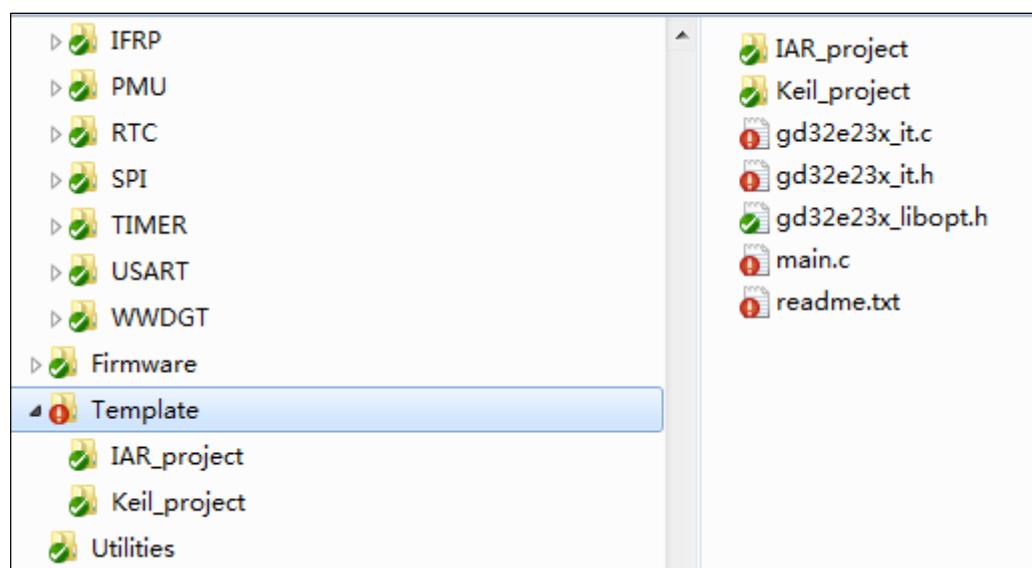
**Figure 2-2. Select peripheral example files**



### 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**

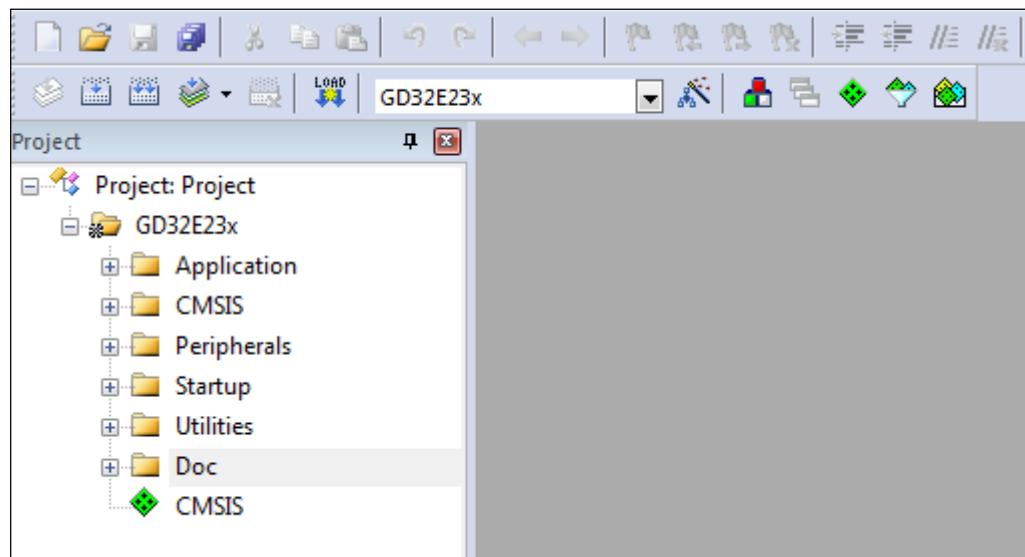


### Open a project

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.uvprojx, shown as

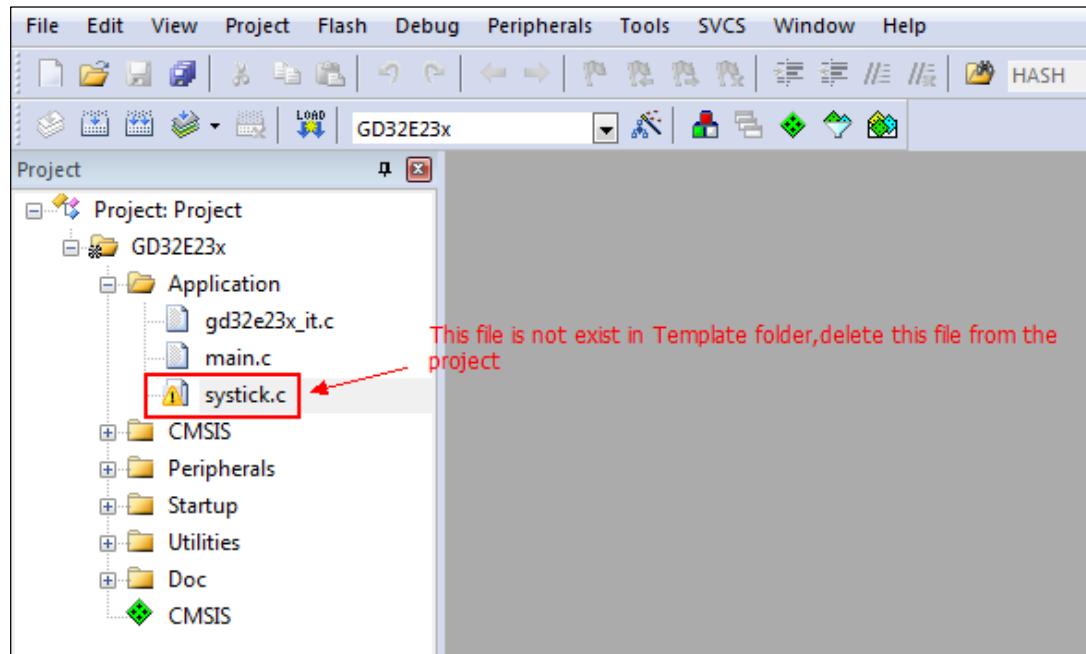
below:

**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:

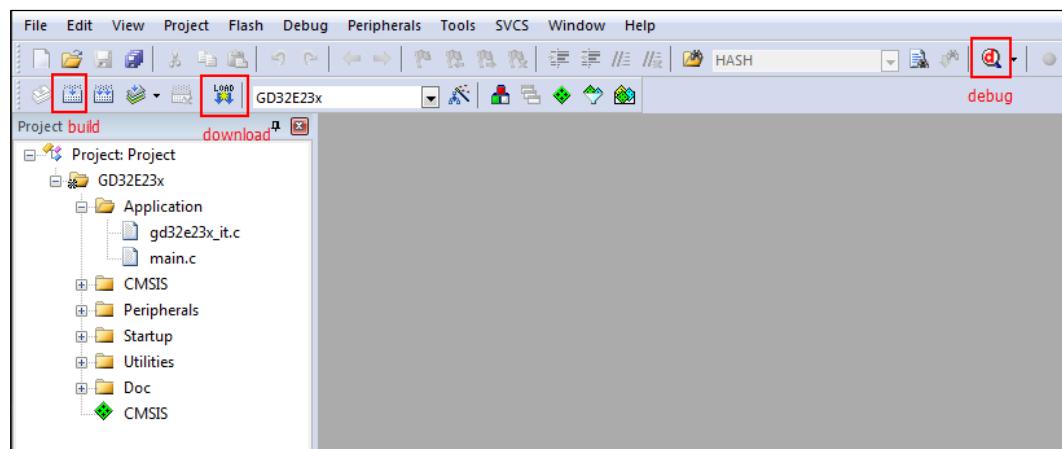
**Figure 2-5. Configure project files**



### 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:

- gd32e230c\_eval.h is related header file of the evaluation board about running the firmware examples;
- gd32e230c\_eval.c is related source file of the evaluation board about running the firmware examples.

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

## 2.2. File descriptions of Firmware Library

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

**Table 2-1. Function descriptions of Firmware Library**

Files	Descriptions
gd32e23x_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.
gd32e23x_it.h	Header file, including all the prototypes of interrupt service routines.
gd32e23x_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.
gd32e23x_xxx.h	The header file of peripheral PPP, including functions about peripheral PPP, and the variables used for functions.
gd32e23x_xxx.c	The C source file for driving peripheral PPP.

Files	Descriptions
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**

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

<b>Registers</b>	<b>Descriptions</b>
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	Inserted channel data offset register x(x=0..3)
ADC_WDHT	Watchdog high threshold register

<b>Registers</b>	<b>Descriptions</b>
ADC_WDLT	Watchdog low threshold register
ADC_RSQ0	Regular sequence register 0
ADC_RSQ1	Regular sequence register 1
ADC_RSQ2	Regular sequence register 2
ADC_ISQ	Inserted sequence register
ADC_IDATAx	Inserted data register x(x=0..3)
ADC_RDATA	Regular data register
ADC_OVSAMPCTL	Oversample control 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**

<b>Function name</b>	<b>Function description</b>
adc_deinit	reset ADC peripheral
adc_enable	enable ADC interface
adc_disable	disable ADC interface
adc_calibration_enable	ADC calibration and reset calibration
adc_dma_mode_enable	enable DMA request
adc_dma_mode_disable	disable DMA request
adc_tempsensor_vrefint_enable	enable the temperature sensor and Vrefint channel
adc_tempsensor_vrefint_disable	disable the temperature sensor and Vrefint channel
adc_discontinuous_mode_config	configure ADC discontinuous mode
adc_special_function_config	enable or disable ADC special function
adc_data_alignment_config	configure ADC data alignment
adc_channel_length_config	configure the length of regular channel group or inserted channel group
adc_regular_channel_config	configure ADC regular channel
adc_inserted_channel_config	configure ADC inserted channel
adc_inserted_channel_offset_config	configure ADC inserted channel offset
adc_external_trigger_config	enable ADC external trigger
adc_external_trigger_source_config	configure ADC external trigger source
adc_software_trigger_enable	enable ADC software trigger
adc_regular_data_read	read ADC regular group data register
adc_inserted_data_read	read ADC inserted group data register
adc_flag_get	get the ADC flag bits
adc_flag_clear	clear the ADC flag bits
adc_interrupt_flag_get	get the ADC interrupt bits
adc_interrupt_flag_clear	clear the ADC flag
adc_interrupt_enable	enable ADC interrupt
adc_interrupt_disable	disable ADC interrupt

Function name	Function description
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_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\_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
Output parameter{out}	
-	-
Return value	
-	-

Example:

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

## adc\_enable

The description of adc\_enable is shown as below:

**Table 3-5. Function adc\_enable**

Function name	adc_enable
Function prototype	void adc_enable(void);
Function descriptions	enable ADC interface
Precondition	-
The called functions	-
Output parameter{out}	
-	-
Return value	
-	-

-	-
---	---

Example:

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

```
adc_enable();
```

### **adc\_disable**

The description of adc\_disable is shown as below:

**Table 3-6. Function adc\_disable**

<b>Function name</b>	adc_disable
<b>Function prototype</b>	void adc_disable(void);
<b>Function descriptions</b>	disable ADC interface
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

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

```
adc_disable();
```

### **adc\_calibration\_enable**

The description of adc\_calibration\_enable is shown as below:

**Table 3-7. Function adc\_calibration\_enable**

<b>Function name</b>	adc_calibration_enable
<b>Function prototype</b>	void adc_calibration_enable(void);
<b>Function descriptions</b>	ADC calibration and reset calibration
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

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

```
adc_calibration_enable();
```

### **adc\_dma\_mode\_enable**

The description of adc\_dma\_mode\_enable is shown as below:

**Table 3-8. Function adc\_dma\_mode\_enable**

<b>Function name</b>	adc_dma_mode_enable
<b>Function prototype</b>	void adc_dma_mode_enable(void);
<b>Function descriptions</b>	enable ADC DMA request
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

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

### **adc\_dma\_mode\_disable**

The description of adc\_dma\_mode\_disable is shown as below:

**Table 3-9. Function adc\_dma\_mode\_disable**

<b>Function name</b>	adc_dma_mode_disable
<b>Function prototype</b>	void adc_dma_mode_disable(void);
<b>Function descriptions</b>	disable ADC DMA request
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

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

### **adc\_tempsensor\_vrefint\_enable**

The description of adc\_tempsensor\_vrefint\_enable is shown as below:

**Table 3-10. Function adc\_tempsensor\_vrefint\_enable**

<b>Function name</b>	adc_tempsensor_vrefint_enable
----------------------	-------------------------------

<b>Function prototype</b>	void adc_tempsensor_vrefint_enable(void);
<b>Function descriptions</b>	enable the temperature sensor and Vrefint channel
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
-	-
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

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

adc_tempsensor_vrefint_enable();
```

### **adc\_tempsensor\_vrefint\_disable**

The description of `adc_tempsensor_vrefint_disable` is shown as below:

**Table 3-11. Function `adc_tempsensor_vrefint_disable`**

<b>Function name</b>	adc_tempsensor_vrefint_disable
<b>Function prototype</b>	void adc_tempsensor_vrefint_disable(void);
<b>Function descriptions</b>	disable the temperature sensor and Vrefint channel
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
-	-
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

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

adc_tempsensor_vrefint_disable();
```

### **adc\_discontinuous\_mode\_config**

The description of `adc_discontinuous_mode_config` is shown as below:

**Table 3-12. Function `adc_discontinuous_mode_config`**

<b>Function name</b>	adc_discontinuous_mode_config
<b>Function prototype</b>	void adc_discontinuous_mode_config(uint8_t channel_group, uint8_t

	length);
<b>Function descriptions</b>	configure ADC discontinuous mode
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>adc_channel_group</b>	select the channel group
<i>ADC_REGULAR_CHANNEL</i>	regular channel group
<i>ADC_INSERTED_CHANNEL</i>	inserted channel group
<i>ADC_CHANNEL_DISCON_DISABLE</i>	disable discontinuous mode of regular and inserted channel
<b>Input parameter{in}</b>	
<b>length</b>	number of conversions in discontinuous mode, the number can be 1..8 for regular channel, the number has no effect for inserted channel
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

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

### **adc\_special\_function\_config**

The description of `adc_special_function_config` is shown as below:

**Table 3-13. Function `adc_special_function_config`**

<b>Function name</b>	adc_special_function_config
<b>Function prototype</b>	void adc_special_function_config(uint32_t function, ControlStatus newvalue);
<b>Function descriptions</b>	enable or disable ADC special function
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>function</b>	the function to config
<i>ADC_SCAN_MODE</i>	scan mode select
<i>ADC_INSERTED_CHANNEL_AUTO</i>	inserted channel group convert automatically
<i>ADC_CONTINUOUS_MODE</i>	continuous mode select
<b>Input parameter{in}</b>	

<b>newvalue</b>	control value
<i>ENABLE</i>	enable function
<i>DISABLE</i>	disable function
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

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

### **adc\_data\_alignment\_config**

The description of adc\_data\_alignment\_config is shown as below:

**Table 3-14. Function adc\_data\_alignment\_config**

<b>Function name</b>	adc_data_alignment_config
<b>Function prototype</b>	void adc_data_alignment_config(uint32_t data_alignment);
<b>Function descriptions</b>	configure ADC data alignment
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>data_alignment</b>	data alignment select
<i>ADC_DATAALIGN_RIGHT</i>	LSB alignment
<i>ADC_DATAALIGN_LEFT</i>	MSB alignment
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

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

### **adc\_channel\_length\_config**

The description of adc\_channel\_length\_config is shown as below:

**Table 3-15. Function adc\_channel\_length\_config**

<b>Function name</b>	adc_channel_length_config
----------------------	---------------------------

<b>Function prototype</b>	void adc_channel_length_config(uint8_t channel_group, uint32_t length);
<b>Function descriptions</b>	configure the length of regular channel group or inserted channel group
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>channel_group</b>	select the channel group
<i>ADC_REGULAR_CHANNEL</i>	regular channel group
<i>ADC_INSERTED_CHANNEL</i>	inserted channel group
<b>Input parameter{in}</b>	
<b>length</b>	the length of the channel, regular channel 1-16, inserted channel 1-4
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

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

### adc\_regular\_channel\_config

The description of `adc_regular_channel_config` is shown as below:

**Table 3-16. Function `adc_regular_channel_config`**

<b>Function name</b>	adc_regular_channel_config
<b>Function prototype</b>	void adc_regular_channel_config(uint8_t rank, uint8_t channel, uint32_t sample_time);
<b>Function descriptions</b>	configure ADC regular channel
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>rank</b>	the regular group sequence rank, this parameter must be between 0 to 15
<b>Input parameter{in}</b>	
<b>channel</b>	the selected ADC channel
<i>ADC_CHANNEL_x</i>	ADC Channelx (x=0..9,16,17)
<b>Input parameter{in}</b>	
<b>sample_time</b>	the sample time value
<i>ADC_SAMPLETIME_1POINT5</i>	1.5 cycles
<i>ADC_SAMPLETIME_7POINT5</i>	7.5 cycles

<i>ADC_SAMPLETIME_13POINT5</i>	13.5 cycles
<i>ADC_SAMPLETIME_28POINT5</i>	28.5 cycles
<i>ADC_SAMPLETIME_41POINT5</i>	41.5 cycles
<i>ADC_SAMPLETIME_55POINT5</i>	55.5 cycles
<i>ADC_SAMPLETIME_71POINT5</i>	71.5 cycles
<i>ADC_SAMPLETIME_239POINT5</i>	239.5 cycles
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* configure ADC regular channel */
adc_regular_channel_config(1, ADC_CHANNEL_0, ADC_SAMPLETIME_7POINT5);
```

### **adc\_inserted\_channel\_config**

The description of `adc_inserted_channel_config` is shown as below:

**Table 3-17. Function `adc_inserted_channel_config`**

<b>Function name</b>	adc_inserted_channel_config
<b>Function prototype</b>	void adc_inserted_channel_config(uint8_t rank, uint8_t channel, uint32_t sample_time);
<b>Function descriptions</b>	configure ADC inserted channel
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>rank</b>	the inserted group sequencer rank, this parameter must be between 0 to 3
<b>Input parameter{in}</b>	
<b>channel</b>	the selected ADC channel
<b>ADC_CHANNEL_x</b>	ADC Channelx (x=0..9,16,17)
<b>Input parameter{in}</b>	
<b>sample_time</b>	the sample time value
<i>ADC_SAMPLETIME_1POINT5</i>	1.5 cycles
<i>ADC_SAMPLETIME_7POINT5</i>	7.5 cycles

<i>ADC_SAMPLETIME_13POINT5</i>	13.5 cycles
<i>ADC_SAMPLETIME_28POINT5</i>	28.5 cycles
<i>ADC_SAMPLETIME_41POINT5</i>	41.5 cycles
<i>ADC_SAMPLETIME_55POINT5</i>	55.5 cycles
<i>ADC_SAMPLETIME_71POINT5</i>	71.5 cycles
<i>ADC_SAMPLETIME_239POINT5</i>	239.5 cycles
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* configure ADC0 inserted channel */
adc_inserted_channel_config(ADC0, 1, ADC_CHANNEL_0, ADC_SAMPLETIME_7POINT5);
```

### **adc\_inserted\_channel\_offset\_config**

The description of `adc_inserted_channel_offset_config` is shown as below:

**Table 3-18. Function `adc_inserted_channel_offset_config`**

<b>Function name</b>	<code>adc_inserted_channel_offset_config</code>
<b>Function prototype</b>	<code>void adc_inserted_channel_offset_config(uint8_t inserted_channel, uint16_t offset);</code>
<b>Function descriptions</b>	configure ADC inserted channel offset
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>inserted_channel</b>	insert channel select
<i>ADC_INSERTED_CHANNEL_x</i>	inserted channel, x=0,1,2,3
<b>Input parameter{in}</b>	
<b>offset</b>	the offset data, this parameter must be between 0 to 4095
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

---

```
/* configure ADC inserted channel offset */
```

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

### **adc\_external\_trigger\_config**

The description of adc\_external\_trigger\_config is shown as below:

**Table 3-19. Function adc\_external\_trigger\_config**

<b>Function name</b>	adc_external_trigger_config
<b>Function prototype</b>	void adc_external_trigger_config(uint8_t channel_group, ControlStatus newvalue);
<b>Function descriptions</b>	configure ADC external trigger
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>channel_group</b>	select the channel group
<b>ADC_REGULAR_CHANNEL</b>	regular channel group
<b>ADC_INSERTED_CHANNEL</b>	inserted channel group
<b>Input parameter{in}</b>	
<b>newvalue</b>	control value
<b>ENABLE</b>	enable function
<b>DISABLE</b>	disable function
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

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

### **adc\_external\_trigger\_source\_config**

The description of adc\_external\_trigger\_source\_config is shown as below:

**Table 3-20. Function adc\_external\_trigger\_source\_config**

<b>Function name</b>	adc_external_trigger_source_config
<b>Function prototype</b>	void adc_external_trigger_source_config(uint8_t channel_group, uint32_t external_trigger_source);
<b>Function descriptions</b>	configure ADC external trigger source
<b>Precondition</b>	-
<b>The called functions</b>	-

Input parameter{in}	
<b>channel_group</b>	select the channel group
<i>ADC_REGULAR_CHA_NNEL</i>	regular channel group
<i>ADC_INSERTED_CHA_NNEL</i>	inserted channel group
Input parameter{in}	
<b>external_trigger_source</b>	regular or inserted group trigger source
<i>ADC_EXTRIG_REGULAR_T0_CH0</i>	TIMER0 CH0 event select for regular channel
<i>ADC_EXTRIG_REGULAR_T0_CH1</i>	TIMER0 CH1 event select for regular channel
<i>ADC1_EXTRIG_REGULAR_T0_CH2</i>	TIMER0 CH2 event select for regular channel
<i>ADC_EXTRIG_REGULAR_T2_TRGO</i>	TIMER2 TRGO event select for regular channel
<i>ADC_EXTRIG_REGULAR_T14_CH0</i>	TIMER14 CH0 event select for regular channel
<i>ADC_EXTRIG_REGULAR_LAR_EXTI_11</i>	external interrupt line 11 for regular channel
<i>ADC_EXTRIG_REGULAR_LAR_NONE</i>	software trigger for regular channel
<i>ADC_EXTRIG_INSERTED_T0_TRGO</i>	TIMER0 TRGO event select for inserted channel
<i>ADC_EXTRIG_INSERTED_T0_CH3</i>	TIMER0 CH3 event select for inserted channel
<i>ADC_EXTRIG_INSERTED_T2_CH3</i>	TIMER2 CH3 event select for inserted channel
<i>ADC_EXTRIG_INSERTED_T14_TRGO</i>	TIMER14 TRGO event select for inserted channel
<i>ADC_EXTRIG_INSERTED_TED_EXTI_15</i>	external interrupt line 15 for inserted channel
<i>ADC_EXTRIG_INSERTED_TED_NONE</i>	software trigger for inserted channel
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* configure ADC regular channel external trigger source */
```

---

```
adc_external_trigger_source_config(ADC_REGULAR_CHANNEL,
    ADC_EXTTRIG_REGULAR_T0_CH0);
```

### **adc\_software\_trigger\_enable**

The description of adc\_software\_trigger\_enable is shown as below:

**Table 3-21. Function adc\_software\_trigger\_enable**

<b>Function name</b>	adc_software_trigger_enable
<b>Function prototype</b>	void adc_software_trigger_enable(uint8_t channel_group);
<b>Function descriptions</b>	enable ADC software trigger
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>channel_group</b>	select the channel group
<i>ADC_REGULAR_CHANNEL</i>	regular channel group
<i>ADC_INSERTED_CHANNEL</i>	inserted channel group
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

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

### **adc\_regular\_data\_read**

The description of adc\_regular\_data\_read is shown as below:

**Table 3-22. Function adc\_regular\_data\_read**

<b>Function name</b>	adc_regular_data_read
<b>Function prototype</b>	uint16_t adc_regular_data_read(void);
<b>Function descriptions</b>	read ADC regular group data register
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
<b>uint16_t</b>	ADC conversion value (0-0xFFFF)

Example:

---

```
/* read ADC regular group data register */
```

```
uint16_t adc_value = 0;  
  
adc_value = adc_regular_data_read();
```

### **adc\_inserted\_data\_read**

The description of `adc_inserted_data_read` is shown as below:

**Table 3-23. Function `adc_inserted_data_read`**

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

Example:

```
/* read ADC inserted group data register */  
  
uint16_t adc_value = 0;  
  
adc_value = adc_inserted_data_read (ADC_INSERTED_CHANNEL_0);
```

### **adc\_flag\_get**

The description of `adc_flag_get` is shown as below:

**Table 3-24. Function `adc_flag_get`**

<b>Function name</b>	adc_flag_get
<b>Function prototype</b>	FlagStatus adc_flag_get(uint32_t flag);
<b>Function descriptions</b>	get the ADC flag bits
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<code>flag</code>	the adc flag bits
<code>ADC_FLAG_WDE</code>	analog watchdog event flag
<code>ADC_FLAG_EOC</code>	end of group conversion flag
<code>ADC_FLAG_EOIC</code>	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 regular channel group
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
<b>FlagStatus</b>	SET or RESET

Example:

```
/* get the ADC analog watchdog flag bits*/
FlagStatus flag_value;
flag_value = adc_flag_get(ADC_FLAG_WDE);
```

### **adc\_flag\_clear**

The description of **adc\_flag\_clear** is shown as below:

**Table 3-25. Function *adc\_flag\_clear***

<b>Function name</b>	<i>adc_flag_clear</i>
<b>Function prototype</b>	void adc_flag_clear(uint32_t flag);
<b>Function descriptions</b>	clear the ADC flag bits
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>flag</b>	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 regular channel group
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

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

### **adc\_interrupt\_flag\_get**

The description of **adc\_interrupt\_flag\_get** is shown as below:

**Table 3-26. Function adc\_interrupt\_flag\_get**

<b>Function name</b>	adc_interrupt_flag_get
<b>Function prototype</b>	FlagStatus adc_interrupt_flag_get(uint32_t flag);
<b>Function descriptions</b>	get the ADC interrupt bits
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>flag</b>	the adc interrupt bits
<i>ADC_INT_FLAG_WDE</i>	analog watchdog interrupt
<i>ADC_INT_FLAG_EOC</i>	end of group conversion interrupt
<i>ADC_INT_FLAG_EOIC</i>	end of inserted group conversion interrupt
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
<b>FlagStatus</b>	SET or RESET

Example:

```
/* get the ADC analog watchdog interrupt bits*/
FlagStatus flag_value;
flag_value = adc_interrupt_flag_get(ADC_INT_FLAG_WDE);
```

### **adc\_interrupt\_flag\_clear**

The description of adc\_interrupt\_flag\_clear is shown as below:

**Table 3-27. Function adc\_interrupt\_flag\_clear**

<b>Function name</b>	adc_interrupt_flag_clear
<b>Function prototype</b>	void adc_interrupt_flag_clear(uint32_t flag);
<b>Function descriptions</b>	clear the ADC interrupt bits
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>flag</b>	the adc interrupt bits
<i>ADC_INT_FLAG_WDE</i>	analog watchdog interrupt
<i>ADC_INT_FLAG_EOC</i>	end of group conversion interrupt
<i>ADC_INT_FLAG_EOIC</i>	end of inserted group conversion interrupt
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

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

---

```
adc_interrupt_flag_clear(ADC_INT_FLAG_WDE);
```

### **adc\_interrupt\_enable**

The description of adc\_interrupt\_enable is shown as below:

**Table 3-28. Function adc\_interrupt\_enable**

<b>Function name</b>	adc_interrupt_enable
<b>Function prototype</b>	void adc_interrupt_enable(uint32_t interrupt);
<b>Function descriptions</b>	enable ADC interrupt
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>interrupt</b>	the adc interrupt
<b>ADC_INT_WDE</b>	analog watchdog interrupt
<b>ADC_INT_EOC</b>	end of group conversion interrupt
<b>ADC_INT_EOIC</b>	end of inserted group conversion interrupt
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

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

adc_interrupt_enable(ADC_INT_WDE);
```

### **adc\_interrupt\_disable**

The description of adc\_interrupt\_disable is shown as below:

**Table 3-29. Function adc\_interrupt\_disable**

<b>Function name</b>	adc_interrupt_disable
<b>Function prototype</b>	void adc_interrupt_disable(uint32_t interrupt);
<b>Function descriptions</b>	Disable ADC interrupt
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>interrupt</b>	the adc interrupt
<b>ADC_INT_WDE</b>	analog watchdog interrupt
<b>ADC_INT_EOC</b>	end of group conversion interrupt
<b>ADC_INT_EOIC</b>	end of inserted group conversion interrupt
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	

-	-
---	---

Example:

```
/* disable ADC interrupt */

adc_interrupt_disable(ADC_INT_WDE);

adc_watchdog_single_channel_enable
```

The description of **adc\_watchdog\_single\_channel\_enable** is shown as below:

**Table 3-30. Function adc\_watchdog\_single\_channel\_enable**

<b>Function name</b>	adc_watchdog_single_channel_enable
<b>Function prototype</b>	void adc_watchdog_single_channel_enable(uint8_t channel);
<b>Function descriptions</b>	configure ADC analog watchdog single channel
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>channel</b>	the selected ADC channel
<b>ADC_CHANNEL_x</b>	ADC Channelx(x=0..9,16,17)
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

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

adc_watchdog_single_channel_enable(ADC_CHANNEL_1);
```

### **adc\_watchdog\_group\_channel\_enable**

The description of **adc\_watchdog\_group\_channel\_enable** is shown as below:

**Table 3-31. Function adc\_watchdog\_group\_channel\_enable**

<b>Function name</b>	adc_watchdog_group_channel_enable
<b>Function prototype</b>	void adc_watchdog_group_channel_enable(uint8_t channel_group);
<b>Function descriptions</b>	configure ADC analog watchdog group channel
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>channel_group</b>	the channel group use analog watchdog
<b>ADC_REGULAR_CHA_NNEL</b>	regular channel group
<b>ADC_INSERTED_CHA</b>	inserted channel group

<i>NNEL</i>	
<i>ADC_REGULAR_INSE</i>	both regular and inserted group
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* configure ADC analog watchdog group channel */
adc_watchdog_group_channel_enable(ADC_REGULAR_CHANNEL);
```

### **adc\_watchdog\_disable**

The description of `adc_watchdog_disable` is shown as below:

**Table 3-32. Function `adc_watchdog_disable`**

<b>Function name</b>	adc_watchdog_disable
<b>Function prototype</b>	void adc_watchdog_disable(void);
<b>Function descriptions</b>	disable ADC analog watchdog
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
-	-
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

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 3-33. Function `adc_watchdog_threshold_config`**

<b>Function name</b>	adc_watchdog_threshold_config
<b>Function prototype</b>	void adc_watchdog_threshold_config(uint16_t low_threshold, uint16_t high_threshold);
<b>Function descriptions</b>	configure ADC analog watchdog threshold
<b>Precondition</b>	-

<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>low_threshold</b>	analog watchdog low threshold, 0..4095
<b>Input parameter{in}</b>	
<b>high_threshold</b>	analog watchdog high threshold, 0..4095
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* configure ADC analog watchdog threshold */
adc_watchdog_threshold_config(0x0400, 0xA00);
```

### **adc\_resolution\_config**

The description of adc\_resolution\_config is shown as below:

**Table 3-34. Function adc\_resolution\_config**

<b>Function name</b>	adc_resolution_config
<b>Function prototype</b>	void adc_resolution_config(uint32_t resolution);
<b>Function descriptions</b>	configure ADC resolution
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>resolution</b>	ADC resolution
<b>ADC_RESOLUTION_12B</b>	12-bit ADC resolution
<b>ADC_RESOLUTION_10B</b>	10-bit ADC resolution
<b>ADC_RESOLUTION_8B</b>	8-bit ADC resolution
<b>ADC_RESOLUTION_6B</b>	6-bit ADC resolution
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

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

### **adc\_oversample\_mode\_config**

The description of adc\_oversample\_mode\_config is shown as below:

**Table 3-35. Function adc\_oversample\_mode\_config**

<b>Function name</b>	adc_oversample_mode_config
<b>Function prototype</b>	void adc_oversample_mode_config(uint32_t mode, uint16_t shift, uint8_t ratio);
<b>Function descriptions</b>	configure ADC oversample mode
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>mode</b>	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
<b>Input parameter{in}</b>	
<b>shift</b>	ADC oversampling shift
<i>ADC_OVERSAMPLING_SHIFT_NONE</i>	no oversampling shift
<i>ADC_OVERSAMPLING_SHIFT_1B</i>	1-bit oversampling shift
<i>ADC_OVERSAMPLING_SHIFT_2B</i>	2-bit oversampling shift
<i>ADC_OVERSAMPLING_SHIFT_3B</i>	3-bit oversampling shift
<i>ADC_OVERSAMPLING_SHIFT_4B</i>	4-bit oversampling shift
<i>ADC_OVERSAMPLING_SHIFT_5B</i>	5-bit oversampling shift
<i>ADC_OVERSAMPLING_SHIFT_6B</i>	6-bit oversampling shift
<i>ADC_OVERSAMPLING_SHIFT_7B</i>	7-bit oversampling shift
<i>ADC_OVERSAMPLING_SHIFT_8B</i>	8-bit oversampling shift
<b>Input parameter{in}</b>	
<b>ratio</b>	ADC oversampling ratio
<i>ADC_OVERSAMPLING_RATIO_MUL2</i>	oversampling ratio multiple 2
<i>ADC_OVERSAMPLING_RATIO_MUL4</i>	oversampling ratio multiple 4

<i>ADC_OVERSAMPLING_RATIO_MUL8</i>	oversampling ratio multiple 8
<i>ADC_OVERSAMPLING_RATIO_MUL16</i>	oversampling ratio multiple 16
<i>ADC_OVERSAMPLING_RATIO_MUL32</i>	oversampling ratio multiple 32
<i>ADC_OVERSAMPLING_RATIO_MUL64</i>	oversampling ratio multiple 64
<i>ADC_OVERSAMPLING_RATIO_MUL128</i>	oversampling ratio multiple 128
<i>ADC_OVERSAMPLING_RATIO_MUL256</i>	oversampling ratio multiple 256
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

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

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

### **adc\_oversample\_mode\_enable**

The description of `adc_oversample_mode_enable` is shown as below:

**Table 3-36. Function `adc_oversample_mode_enable`**

<b>Function name</b>	adc_oversample_mode_enable
<b>Function prototype</b>	void adc_oversample_mode_enable(void);
<b>Function descriptions</b>	enable ADC oversample mode
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
-	-
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* enable ADC oversample mode */

adc_oversample_mode_enable();
```

### **adc\_oversample\_mode\_disable**

The description of adc\_oversample\_mode\_disable is shown as below:

**Table 3-37. Function adc\_oversample\_mode\_disable**

<b>Function name</b>	adc_oversample_mode_disable
<b>Function prototype</b>	void adc_oversample_mode_disable(void);
<b>Function descriptions</b>	disable ADC oversample mode
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
-	-
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* disable ADC oversample mode */

adc_oversample_mode_disable();
```

## **3.3. CMP**

The general purpose comparator can work either standalone(all terminal are available on I/Os) or together with the timers. It could be used to wake up the MCU from low-power mode by an analog signal, provide a trigger source when an analog signal is in a certain condition, achieves some current control by working together with a PWM output of a timer.The CMP registers are listed in chapter [3.3.1](#), the CMP firmware functions are introduced in chapter [3.3.2](#).

### **3.3.1. Descriptions of Peripheral registers**

CMP registers are listed in the table shown as below:

**Table 3-38. CMP Registers**

Registers	Descriptions
CMP_CS	Control/Status register

### **3.3.2. Descriptions of Peripheral functions**

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

**Table 3-39. CMP firmware function**

<b>Function name</b>	<b>Function description</b>
cmp_deinit	deinitialize comparator
cmp_mode_init	initialize comparator mode
cmp_output_init	initialize comparator output
cmp_enable	enable comparator
cmp_disable	disable comparator
cmp_switch_enable	enable comparator switch
cmp_switch_disable	disable comparator switch
cmp_output_level_get	get output level
cmp_lock_enable	lock the comparator

### **cmp\_deinit**

The description of cmp\_deinit is shown as below:

**Table 3-40. Function cmp\_deinit**

<b>Function name</b>	cmp_deinit
<b>Function prototype</b>	void cmp_deinit(void);
<b>Function descriptions</b>	deinitialize CMP
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
-	-
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* CMP deinitialize*/
cmp_deinit();
```

### **cmp\_mode\_init**

The description of cmp\_mode\_init is shown as below:

**Table 3-41. Function cmp\_mode\_init**

<b>Function name</b>	cmp_mode_init
<b>Function prototype</b>	void cmp_mode_init(operating_mode_enum operating_mode, inverting_input_enum inverting_input, cmp_hysteresis_enum output_hysteresis)
<b>Function descriptions</b>	initialize comparator mode
<b>Precondition</b>	-

The called functions		-
Input parameter{in}		
<b>operating_mode</b>		operating_mode
<i>CMP_HIGHSPEED</i>		high speed mode
<i>CMP_MIDDLE SPEED</i>		medium speed mode
<i>CMP_LOWSPEED</i>		low speed mode
<i>CMP_VERYLOWSPEED</i>		very-low speed mode
<i>D</i>		
Input parameter{in}		
<b>inverting_input</b>		inverting_input
<i>CMP_1_4VREFINT</i>		VREFINT *1/4 input
<i>CMP_1_2VREFINT</i>		VREFINT *1/2 input
<i>CMP_3_4VREFINT</i>		VREFINT *3/4 input
<i>CMP_VREFINT</i>		VREFINT input
<i>CMP_PA4</i>		PA4 input
<i>CMP_PA5</i>		PA5 input
<i>CMP_PA0</i>		PA0 input
<i>CMP_PA2</i>		PA2 input
Input parameter{in}		
<b>output_hysteresis</b>		hysteresis
<i>CMP_HYSTERESIS_NO</i>		output no hysteresis
<i>CMP_HYSTERESIS_LOW</i>		output low hysteresis
<i>CMP_HYSTERESIS_MIDDLE</i>		output middle hysteresis
<i>CMP_HYSTERESIS_HIGH</i>		output high hysteresis
Output parameter{out}		
-		-
Return value		
-		-

Example:

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

### cmp\_output\_init

The description of cmp\_output\_init is shown as below:

**Table 3-42. Function cmp\_output\_init**

Function name	cmp_output_init

<b>Function prototype</b>	void cmp_output_init(cmp_output_enum output_selection, uint32_t output_polarity);
<b>Function descriptions</b>	initialize comparator output
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>output_selection</b>	output_selection
<i>CMP_OUTPUT_NONE</i>	output no selection
<i>CMP_OUTPUT_TIMER_0BKIN</i>	TIMER 0 break input
<i>CMP_OUTPUT_TIMER_0ICO</i>	TIMER 0 channel0 input capture
<i>CMP_OUTPUT_TIMER_0OCPRECLR</i>	TIMER 0 OCPRE_CLR input
<i>CMP_OUTPUT_TIMER_2ICO</i>	TIMER 2 channel0 input capture
<i>CMP_OUTPUT_TIMER_2OCPRECLR</i>	TIMER 2 OCPRE_CLR input
<b>Input parameter{in}</b>	
<b>output_polarity</b>	output_polarity
<i>CMP_OUTPUT_POLARITY_INVERTED</i>	output is inverted
<i>CMP_OUTPUT_POLARITY_NOINVERTED</i>	output is not inverted
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* CMP output initialize*/

cmp_output_init(CMP_OUTPUT_TIMER0BKIN,
CMP_OUTPUT_POLARITY_NOINVERTED);
```

### **cmp\_enable**

The description of cmp\_enable is shown as below:

**Table 3-43. Function can\_fd\_init**

<b>Function name</b>	cmp_enable
<b>Function prototype</b>	void cmp_enable(void);
<b>Function descriptions</b>	enable comparator
<b>Precondition</b>	-

<b>The called functions</b>	-
<b>Input parameter{in}</b>	
-	-
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* enable CMP*/
cmp_enable();
```

### **cmp\_disable**

The description of cmp\_disable is shown as below:

**Table 3-44. Function cmp\_disable**

<b>Function name</b>	cmp_disable
<b>Function prototype</b>	void cmp_disable(void);
<b>Function descriptions</b>	disable comparator
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
-	-
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

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

### **cmp\_switch\_enable**

The description of cmp\_switch\_enable is shown as below:

**Table 3-45. Function cmp\_switch\_enable**

<b>Function name</b>	cmp_switch_enable
<b>Function prototype</b>	void cmp_switch_enable(void);
<b>Function descriptions</b>	enable comparator switch
<b>Precondition</b>	-
<b>The called functions</b>	-

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

Example:

```
/* enable CMP switch */
cmp_switch_enable();
```

### **cmp\_switch\_disable**

The description of cmp\_switch\_disable is shown as below:

**Table 3-46. Function cmp\_switch\_disable**

<b>Function name</b>	cmp_switch_disable
<b>Function prototype</b>	void cmp_switch_disable(void);
<b>Function descriptions</b>	disable comparator switch
<b>Precondition</b>	-
<b>The called functions</b>	-
Input parameter{in}	
-	-
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* disable CMP switch */
cmp_switch_disable();
```

### **cmp\_output\_level\_get**

The description of cmp\_output\_level\_get is shown as below:

**Table 3-47. Function cmp\_output\_level\_get**

<b>Function name</b>	cmp_output_level_get
<b>Function prototype</b>	uint32_t cmp_output_level_get(void);
<b>Function descriptions</b>	get output level
<b>Precondition</b>	-
<b>The called functions</b>	-
Input parameter{in}	

-	-
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
<b>uint32_t</b>	CMP_OUTPUTLEVEL_HIGH / CMP_OUTPUTLEVEL_LOW

Example:

```
/* get CMP output level */
cmp_output_level_get();
```

### **cmp\_lock\_enable**

The description of cmp\_lock\_enable is shown as below:

**Table 3-48. Function cmp\_lock\_enable**

<b>Function name</b>	cmp_lock_enable
<b>Function prototype</b>	void cmp_lock_enable(void);
<b>Function descriptions</b>	lock the comparator
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
-	-
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

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

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

### **3.4.1. Descriptions of Peripheral registers**

CRC registers are listed in the table shown as below:

**Table 3-49. CRC Registers**

<b>Registers</b>	<b>Descriptions</b>
CRC_DATA	CRC data register
CRC_FDATA	CRC free data register
CRC_CTL	CRC control register
CRC_IDATA	CRC initialization data register
CRC_POLY	CRC polynomial register

### 3.4.2. Descriptions of Peripheral functions

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

**Table 3-50. CRC firmware function**

<b>Function name</b>	<b>Function description</b>
crc_deinit	deinit CRC calculation unit
crc_reverse_output_data_enable	enable the reverse operation of output data
crc_reverse_output_data_disable	disable the reverse operation of output data
crc_data_register_reset	reset data register to the value of initializaiton data register
crc_data_register_read	read the data register
crc_free_data_register_read	read the free data register
crc_free_data_register_write	write the free data register
crc_init_data_register_write	write the initial value register
crc_input_data_reverse_config	configure the CRC input data function
crc_polynomial_size_set	configure the CRC size of polynomial function
crc_polynomial_set	configure the CRC polynomial value function
crc_single_data_calculate	CRC calculate a 32-bit data
crc_block_data_calculate	CRC calculate a 32-bit data array

#### **crc\_deinit**

The description of crc\_deinit is shown as below:

**Table 3-51. Function crc\_deinit**

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

Example:

```
/* reset crc */

crc_deinit();
```

### **crc\_reverse\_output\_data\_enable**

The description of `crc_reverse_output_data_enable` is shown as below:

**Table 3-52. Function `crc_reverse_output_data_enable`**

<b>Function name</b>	crc_reverse_output_data_enable
<b>Function prototype</b>	void crc_reverse_output_data_enable (void);
<b>Function descriptions</b>	enable the reverse operation of output data
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
-	-
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* enable CRC reverse operation of output data */

crc_reverse_output_data_enable();
```

### **crc\_reverse\_output\_data\_disable**

The description of `crc_reverse_output_data_disable` is shown as below:

**Table 3-53. Function `crc_reverse_output_data_disable`**

<b>Function name</b>	crc_reverse_output_data_disable
<b>Function prototype</b>	void crc_reverse_output_data_disable (void);
<b>Function descriptions</b>	disable the reverse operation of output data
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
-	-
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

---

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

```
crc_reverse_output_data_disable();
```

### **crc\_data\_register\_reset**

The description of `crc_data_register_reset` is shown as below:

**Table 3-54. Function `crc_data_register_reset`**

<b>Function name</b>	crc_data_register_reset
<b>Function prototype</b>	void crc_data_register_reset(void);
<b>Function descriptions</b>	reset data register to the value of initializaiton data register
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
-	-
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

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 3-55. Function `crc_data_register_read`**

<b>Function name</b>	crc_data_register_read
<b>Function prototype</b>	uint32_t crc_data_register_read(void);
<b>Function descriptions</b>	read the data register
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
-	-
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
<b>uint32_t</b>	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 3-56. Function `crc_free_data_register_read`**

<b>Function name</b>	crc_free_data_register_read	
<b>Function prototype</b>	uint8_t crc_free_data_register_read(void);	
<b>Function descriptions</b>	read the free data register	
<b>Precondition</b>	-	
<b>The called functions</b>	-	
<b>Input parameter{in}</b>		
-	-	
<b>Output parameter{out}</b>		
-	-	
<b>Return value</b>		
<b>uint8_t</b>	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 3-57. Function `crc_free_data_register_write`**

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

Example:

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

---

```
crc_free_data_register_write(0x11);
```

### **crc\_init\_data\_register\_write**

The description of `crc_init_data_register_write` is shown as below:

**Table 3-58. Function `crc_init_data_register_write`**

<b>Function name</b>	crc_init_data_register_write	
<b>Function prototype</b>	void crc_init_data_register_write(uint32_t init_data)	
<b>Function descriptions</b>	write the initialization data register	
<b>Precondition</b>	-	
<b>The called functions</b>	-	
<b>Input parameter{in}</b>		
<b>init_data</b>	specify 32-bit data	
<b>Output parameter{out}</b>		
-	-	
<b>Return value</b>		
-	-	

Example:

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

### **crc\_input\_data\_reverse\_config**

The description of `crc_input_data_reverse_config` is shown as below:

**Table 3-59. Function `crc_input_data_reverse_config`**

<b>Function name</b>	crc_input_data_reverse_config	
<b>Function prototype</b>	void crc_input_data_reverse_config(uint32_t data_reverse)	
<b>Function descriptions</b>	configure the crc input data function	
<b>Precondition</b>	-	
<b>The called functions</b>	-	
<b>Input parameter{in}</b>		
<b>data_reverse</b>	specify input data reverse function	
<b>CRC_INPUT_DATA_N OT</b>	input data is not reversed	
<b>CRC_INPUT_DATA_B YTE</b>	input data is reversed on 8 bits	
<b>CRC_INPUT_DATA_H ALFWORD</b>	input data is reversed on 16 bits	
<b>CRC_INPUT_DATA_W ORD</b>	input data is reversed on 32 bits	
<b>Output parameter{out}</b>		

-	-
<b>Return value</b>	
-	-

Example:

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

### **crc\_polynomial\_size\_set**

The description of **crc\_polynomial\_size\_set** is shown as below:

**Table 3-60. Function **crc\_polynomial\_size\_set****

<b>Function name</b>	crc_polynomial_size_set
<b>Function prototype</b>	void crc_polynomial_size_set(uint32_t poly_size)
<b>Function descriptions</b>	configure the CRC size of polynomial function
<b>Precondition</b>	-
<b>The called functions</b>	-
<i>Input parameter{in}</i>	
<b>poly_size</b>	size of polynomial
CRC_CTL_PS_32	32-bit polynomial for CRC calculation
CRC_CTL_PS_16	16-bit polynomial for CRC calculation
CRC_CTL_PS_8	8-bit polynomial for CRC calculation
CRC_CTL_PS_7	7-bit polynomial for CRC calculation
<i>Output parameter{out}</i>	
-	-
<b>Return value</b>	
-	-

Example:

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

### **crc\_polynomial\_set**

The description of **crc\_polynomial\_set** is shown as below:

**Table 3-61. Function **crc\_polynomial\_set****

<b>Function name</b>	crc_polynomial_set
<b>Function prototype</b>	void crc_polynomial_set(uint32_t poly)
<b>Function descriptions</b>	configure the CRC polynomial value function
<b>Precondition</b>	-
<b>The called functions</b>	-

Input parameter{in}	
<b>poly</b>	configurable polynomial value
Output parameter{out}	
-	-
Return value	
-	-

Example:

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

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

The description of `crc_single_data_calculate` is shown as below:

**Table 3-62. Function `crc_single_data_calculate`**

<b>Function name</b>	crc_single_data_calculate
<b>Function prototype</b>	uint32_t crc_single_data_calculate(uint32_t sdata);
<b>Function descriptions</b>	CRC calculate a 32-bit data
<b>Precondition</b>	-
<b>The called functions</b>	-
Input parameter{in}	
<b>sdata</b>	specify 32-bit data
Output parameter{out}	
-	-
Return value	
<b>uint32_t</b>	32-bit CRC calculate value (0-0xFFFFFFFF)

Example:

```
/* CRC calculate a 32-bit data */
uint32_t val = 0, valcrc = 0;
val = (uint32_t)0xabcd1234;
rcu_periph_clock_enable(RCU_CRC);
valcrc = crc_single_data_calculate(val);
```

### **crc\_block\_data\_calculate**

The description of `crc_block_data_calculate` is shown as below:

**Table 3-63. Function `crc_block_data_calculate`**

<b>Function name</b>	crc_block_data_calculate
<b>Function prototype</b>	uint32_t crc_block_data_calculate(uint32_t array[], uint32_t size);

<b>Function descriptions</b>	calculate the CRC value of an array of 32-bit values
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>array</b>	pointer to an array of 32 bit data words
<b>Input parameter{in}</b>	
<b>size</b>	size of the array
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
<b>uint32_t</b>	32-bit CRC calculate value (0-0xFFFFFFFF)

Example:

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

#define BUFFER_SIZE      6

uint32_t valcrc = 0;

static const uint32_t data_buffer[BUFFER_SIZE] = {

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

rcu_periph_clock_enable(RCU_CRC);

valcrc = crc_block_data_calculate((uint32_t *) data_buffer, BUFFER_SIZE);
```

## 3.5. **DBG**

The DBG hold unit helps debugger to debug power saving mode. The DBG registers are listed in chapter [3.5.1](#), the DBG firmware functions are introduced in chapter [3.5.2](#).

### 3.5.1. **Descriptions of Peripheral registers**

DBG registers are listed in the table shown as below:

**Table 3-64. DBG Registers**

Registers	Descriptions
DBG_ID	DBG ID code register
DBG_CTL0	DBG control register0
DBG_CTL1	DBG control register1

### 3.5.2. **Descriptions of Peripheral functions**

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

**Table 3-65. DBG firmware function**

<b>Function name</b>	<b>Function description</b>
dbg_deinit	reset DBG register
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

**Enum dbg\_periph\_enum**
**Table 3-66. Enum dbg\_periph\_enum**

<b>Member name</b>	<b>Function description</b>
DBG_FWDGT_HOLD	debug FWDGT kept when core is halted
DBG_WWDGT_HOLD	debug WWDGT kept when core is halted
DBG_TIMER0_HOLD	hold TIMER0 counter when core is halted
DBG_TIMER2_HOLD	hold TIMER2 counter when core is halted
DBG_TIMER5_HOLD	hold TIMER5 counter when core is halted
DBG_TIMER13_HOLD	hold TIMER13 counter when core is halted
DBG_TIMER14_HOLD	hold TIMER14 counter when core is halted
DBG_TIMER15_HOLD	hold TIMER15 counter when core is halted
DBG_TIMER16_HOLD	hold TIMER16 counter 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_RTC_HOLD	hold RTC counter when core is halted

**dbg\_deinit**

The description of dbg\_deinit is shown as below:

**Table 3-67. Function dbg\_deinit**

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

Example:

---

```
/* deinitialize the DBG*/
```

```
dbg_deinit();
```

### **dbg\_id\_get**

The description of dbg\_id\_get is shown as below:

**Table 3-68. Function dbg\_id\_get**

<b>Function name</b>	dbg_id_get
<b>Function prototype</b>	uint32_t dbg_id_get(void);
<b>Function descriptions</b>	Read DBG_ID code register
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
-	-
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
<b>uint32_t</b>	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 3-69. Function dbg\_low\_power\_enable**

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

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 3-70. Function dbg\_low\_power\_disable**

<b>Function name</b>	dbg_low_power_disable
<b>Function prototype</b>	void dbg_low_power_disable(uint32_t dbg_low_power);
<b>Function descriptions</b>	Disable low power behavior when the mcu is in debug mode
<b>Precondition</b>	-
<b>The called functions</b>	-
Input parameter{in}	
<b>dbg_low_power</b>	low power mode
<b>DBG_LOW_POWER_SLEEP</b>	keep debugger connection during sleep mode
<b>DBG_LOW_POWER_DEEPSLEEP</b>	keep debugger connection during deepsleep mode
<b>DBG_LOW_POWER_STANDBY</b>	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 3-71. Function dbg\_periph\_enable**

<b>Function name</b>	dbg_periph_enable
<b>Function prototype</b>	void dbg_periph_enable(dbg_periph_enum dbg_periph);
<b>Function descriptions</b>	Enable peripheral behavior when the mcu is in debug mode

<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>dbg_periph</b>	Peripheral refer to <a href="#">Table 3-66. Enum dbg_periph_enum</a>
<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_TIMERx_HOLD</i>	x=0,2,5,13,14,15,16, hold TIMERx counter when core is halted
<i>DBG_I2Cx_HOLD</i>	x=0,1, hold I2Cx smbus when core is halted
<i>DBG_RTC_HOLD</i>	hold RTC counter when core is halted
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

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 3-72. Function `dbg_periph_disable`**

<b>Function name</b>	dbg_periph_disable
<b>Function prototype</b>	void dbg_periph_disable(dbg_periph_enum dbg_periph);
<b>Function descriptions</b>	Disable peripheral behavior when the mcu is in debug mode
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>dbg_periph</b>	peripheral refer to <a href="#">Table 3-66. Enum dbg_periph_enum</a>
<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_TIMERx_HOLD</i>	x=0,2,5,13,14,15,16, hold TIMERx counter when core is halted
<i>DBG_I2Cx_HOLD</i>	x=0,1, hold I2Cx smbus when core is halted
<i>DBG_RTC_HOLD</i>	hold RTC counter when core is halted
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

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

dbg_periph_disable(DBG_TIMER0_HOLD);
```

## 3.6. 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 [3.6.1](#), the DMA firmware functions are introduced in chapter [3.6.2](#).

### 3.6.1. Descriptions of Peripheral registers

DMA registers are listed in the table shown as below:

**Table 3-73. DMA Registers**

Registers	Descriptions
DMA_INTF	Interrupt flag register
DMA_INTC	Interrupt flag clear register
DMA_CHxCTL (x=0..4)	Channel x control register
DMA_CHxCNT (x=0..4)	Channel x counter register
DMA_CHxPADDR (x=0..4)	Channel x peripheral base address register
DMA_CHxMADDR (x=0..4)	Channel x memory base address register

### 3.6.2. Descriptions of Peripheral functions

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

**Table 3-74. DMA firmware function**

Function name	Function description
dma_deinit	deinitialize DMA a channel registers
dma_struct_para_init	initialize the parameters of DMA struct with the default values
dma_init	initialize DMA channel
dma_circulation_enable	enable DMA circulation mode
dma_circulation_disable	disable DMA circulation mode
dma_memory_to_memory_enable	enable memory to memory mode
dma_memory_to_memory_disable	disable memory to memory mode
dma_channel_enable	enable DMA channel
dma_channel_disable	disable DMA channel
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

Function name	Function description
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_width_config	configure transfer data size of memory
dma_periph_width_config	configure transfer data size of peripheral
dma_memory_increase_enable	enable next address increasement algorithm of memory
dma_memory_increase_disable	disable next address increasement algorithm of memory
dma_periph_increase_enable	enable next address increasement algorithm of peripheral
dma_periph_increase_disable	disable next address increasement algorithm of peripheral
dma_transfer_direction_config	configure the direction of data transfer on the channel
dma_flag_get	check DMA flag is set or not
dma_flag_clear	clear DMA a channel flag
dma_interrupt_flag_get	check DMA flag and interrupt enable bit is set or not
dma_interrupt_flag_clear	clear DMA a channel flag
dma_interrupt_enable	enable DMA interrupt
dma_interrupt_disable	disable DMA interrupt

## Structure dma\_parameter\_struct

**Table 3-75. Structure dma\_parameter\_struct**

Member name	Function description
periph_addr	peripheral base address
periph_width	transfer data size of peripheral
memory_addr	memory base address
memory_width	transfer data size of memory
number	channel transfer number
priority	channel priority level
periph_inc	peripheral increasing mode
memory_inc	memory increasing mode
direction	channel data transfer direction

## dma\_deinit

The description of dma\_deinit is shown as below:

**Table 3-76. Function dma\_deinit**

Function name	dma_deinit
Function prototype	void dma_deinit(dma_channel_enum channelx);
Function descriptions	deinitialize DMA a channel registers
Precondition	-
The called functions	-
<b>Input parameter{in}</b>	
channelx	DMA channel

<b>DMA_CHx(x=0..4)</b>	DMA channel selection
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

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

### **dma\_struct\_para\_init**

The description of **dma\_struct\_para\_init** is shown as below:

**Table 3-77. Function dma\_struct\_para\_init**

<b>Function name</b>	dma_struct_para_init
<b>Function prototype</b>	void dma_struct_para_init(dma_parameter_struct* init_struct);
<b>Function descriptions</b>	initialize the parameters of DMA struct with the default values
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>init_struct</b>	the initialization data needed to initialize DMA channel
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

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

### **dma\_init**

The description of **dma\_init** is shown as below:

**Table 3-78. Function dma\_init**

<b>Function name</b>	dma_init
<b>Function prototype</b>	void dma_init(dma_channel_enum channelx, dma_parameter_struct init_struct);
<b>Function descriptions</b>	initialize DMA channel
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	

<b>channelx</b>	DMA channel
<i>DMA_CHx( x=0..4)</i>	DMA channel selection
<b>Input parameter{in}</b>	
<b>init_struct</b>	Structure for initialization, the structure members can refer to <a href="#">Table 3-75.</a> <a href="#"><u>Structure dma_parameter_struct</u></a>
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```

/* DMA channel0 initialize */
dma_parameter_struct dma_init_struct;

dma_struct_para_init(&dma_init_struct);
dma_init_struct.direction = DMA_PERIPHERAL_TO_MEMORY;
dma_init_struct.memory_addr = (uint32_t)g_destbuf;
dma_init_struct.memory_inc = DMA_MEMORY_INCREASE_ENABLE;
dma_init_struct.memory_width = DMA_MEMORY_WIDTH_8BIT;
dma_init_struct.number = TRANSFER_NUM;
dma_init_struct.periph_addr = (uint32_t)BANK0_WRITE_START_ADDR;
dma_init_struct.periph_inc = DMA_PERIPH_INCREASE_ENABLE;
dma_init_struct.periph_width = DMA_PERIPHERAL_WIDTH_8BIT;
dma_init_struct.priority = DMA_PRIORITY_ULTRA_HIGH;
dma_init(DMA_CH0, dma_init_struct);
```

### **dma\_circulation\_enable**

The description of `dma_circulation_enable` is shown as below:

**Table 3-79. Function `dma_circulation_enable`**

<b>Function name</b>	dma_circulation_enable
<b>Function prototype</b>	void dma_circulation_enable(dma_channel_enum channelx);
<b>Function descriptions</b>	enable DMA circulation mode
<b>Precondition</b>	corresponding channel enable bit CHEN should be 0
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>channelx</b>	DMA channel
<i>DMA_CHx( x=0..4)</i>	DMA channel selection
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

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

### **dma\_circulation\_disable**

The description of dma\_circulation\_disable is shown as below:

**Table 3-80. Function dma\_circulation\_disable**

<b>Function name</b>	dma_circulation_disable
<b>Function prototype</b>	void dma_circulation_disable(dma_channel_enum channelx);
<b>Function descriptions</b>	disable DMA circulation mode
<b>Precondition</b>	corresponding channel enable bit CHEN should be 0
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>channelx</b>	DMA channel
<b>DMA_CHx( x=0..4)</b>	DMA channel selection
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

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

### **dma\_memory\_to\_memory\_enable**

The description of dma\_memory\_to\_memory\_enable is shown as below:

**Table 3-81. Function dma\_memory\_to\_memory\_enable**

<b>Function name</b>	dma_memory_to_memory_enable
<b>Function prototype</b>	void dma_memory_to_memory_enable(dma_channel_enum channelx);
<b>Function descriptions</b>	enable memory to memory mode
<b>Precondition</b>	corresponding channel enable bit CHEN should be 0
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>channelx</b>	DMA channel
<b>DMA_CHx( x=0..4)</b>	DMA channel selection
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

---

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

### **dma\_memory\_to\_memory\_disable**

The description of `dma_memory_to_memory_disable` is shown as below:

**Table 3-82. Function `dma_memory_to_memory_disable`**

<b>Function name</b>	dma_memory_to_memory_disable
<b>Function prototype</b>	void dma_memory_to_memory_disable(dma_channel_enum channelx);
<b>Function descriptions</b>	disable memory to memory mode
<b>Precondition</b>	corresponding channel enable bit CHEN should be 0
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>channelx</b>	DMA channel
<i>DMA_CHx( x=0..4)</i>	DMA channel selection
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

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

### **dma\_channel\_enable**

The description of `dma_channel_enable` is shown as below:

**Table 3-83. Function `dma_channel_enable`**

<b>Function name</b>	dma_channel_enable
<b>Function prototype</b>	void dma_channel_enable(dma_channel_enum channelx);
<b>Function descriptions</b>	enable DMA channel
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>channelx</b>	DMA channel
<i>DMA_CHx(x=0..4)</i>	DMA channel selection
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* enable DMA channel0 */
```

---

```
dma_channel_enable(DMA_CH0);
```

### **dma\_channel\_disable**

The description of `dma_channel_disable` is shown as below:

**Table 3-84. Function `dma_channel_disable`**

<b>Function name</b>	dma_channel_disable
<b>Function prototype</b>	void dma_channel_disable(dma_channel_enum channelx);
<b>Function descriptions</b>	disable DMA channel
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>channelx</b>	DMA channel
<b>DMA_CHx( x=0..4)</b>	DMA channel selection
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

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

### **dma\_periph\_address\_config**

The description of `dma_periph_address_config` is shown as below:

**Table 3-85. Function `dma_periph_address_config`**

<b>Function name</b>	dma_periph_address_config
<b>Function prototype</b>	void dma_periph_address_config(dma_channel_enum channelx, uint32_t address);
<b>Function descriptions</b>	set DMA peripheral base address
<b>Precondition</b>	corresponding channel enable bit CHEN should be 0
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>channelx</b>	DMA channel
<b>DMA_CHx(x=0..4)</b>	DMA channel selection
<b>Input parameter{in}</b>	
<b>address</b>	peripheral base address
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* configure DMA channel0 periph address */

#define BANK0_WRITE_START_ADDR ((uint32_t)0x08004000)

dma_periph_address_config(DMA_CH0, BANK0_WRITE_START_ADDR);
```

### **dma\_memory\_address\_config**

The description of `dma_memory_address_config` is shown as below:

**Table 3-86. Function `dma_memory_address_config`**

<b>Function name</b>	dma_memory_address_config
<b>Function prototype</b>	void dma_memory_address_config(dma_channel_enum channelx, uint32_t address);
<b>Function descriptions</b>	set DMA memory base address
<b>Precondition</b>	corresponding channel enable bit CHEN should be 0
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
channelx	DMA channel
DMA_CHx(x=0..4)	DMA channel selection
<b>Input parameter{in}</b>	
address	memory base address
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* configure DMA channel0 memory address */

uint8_t g_destbuf[TRANSFER_NUM];

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

### **dma\_transfer\_number\_config**

The description of `dma_transfer_number_config` is shown as below:

**Table 3-87. Function `dma_transfer_number_config`**

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

Input parameter{in}	
<b>channelx</b>	DMA channel
<b>DMA_CHx( x=0..4)</b>	DMA channel selection
Input parameter{in}	
<b>number</b>	data transfer number(0x0-0xFFFF)
Output parameter{out}	
-	-
Return value	
-	-

Example:

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

#define TRANSFER_NUM          0x400

dma_transfer_number_config(DMA_CH0, TRANSFER_NUM);
```

### **dma\_transfer\_number\_get**

The description of **dma\_transfer\_number\_get** is shown as below:

**Table 3-88. Function dma\_transfer\_number\_get**

<b>Function name</b>	dma_transfer_number_get
<b>Function prototype</b>	uint32_t dma_transfer_number_get(dma_channel_enum channelx);
<b>Function descriptions</b>	get the number of remaining data to be transferred by the DMA
<b>Precondition</b>	-
<b>The called functions</b>	-
Input parameter{in}	
<b>channelx</b>	DMA channel
<b>DMA_CHx( x=0..4)</b>	DMA channel selection
Output parameter{out}	
-	-
Return value	
<b>uint32_t</b>	DMA data transmission remaining quantity (0x0-0xFFFF)

Example:

```
/* get DMA channel0 transfer number */

uint32_t number = 0;

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

### **dma\_priority\_config**

The description of **dma\_priority\_config** is shown as below:

**Table 3-89. Function dma\_priority\_config**

<b>Function name</b>	dma_priority_config
<b>Function prototype</b>	void dma_priority_config(dma_channel_enum channelx, uint32_t priority);
<b>Function descriptions</b>	configure priority level of DMA channel
<b>Precondition</b>	corresponding channel enable bit CHEN should be 0
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
channelx	DMA channel
DMA_CHx(x=0..4)	DMA channel selection
<b>Input parameter{in}</b>	
priority	priority Level of this channel
DMA_PRIORITY_LOW	low priority
DMA_PRIORITY_MEDI UM	medium priority
DMA_PRIORITY_HIGH	high priority
DMA_PRIORITY_ULTR A_HIGH	ultra high priority
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

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

### **dma\_memory\_width\_config**

The description of dma\_memory\_width\_config is shown as below:

**Table 3-90. Function dma\_memory\_width\_config**

<b>Function name</b>	dma_memory_width_config
<b>Function prototype</b>	void dma_memory_width_config( dma_channel_enum channelx, uint32_t mwidth);
<b>Function descriptions</b>	configure transfer data size of memory
<b>Precondition</b>	corresponding channel enable bit CHEN should be 0
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
channelx	DMA channel
DMA_CHx( x=0..4)	DMA channel selection
<b>Input parameter{in}</b>	
mwidth	transfer data width of memory
DMA_MEMORY_WIDT	transfer data width of memory is 8-bit

<i>H_8BIT</i>	
<i>DMA_MEMORY_WIDT</i> <i>H_16BIT</i>	transfer data width of memory is 16-bit
<i>DMA_MEMORY_WIDT</i> <i>H_32BIT</i>	transfer data width of memory is 32-bit
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* configure DMA channel0 memory width */

dma_memory_width_config(DMA_CH0, DMA_MEMORY_WIDTH_8BIT);
```

### dma\_periph\_width\_config

The description of `dma_periph_width_config` is shown as below:

**Table 3-91. Function `dma_periph_width_config`**

<b>Function name</b>	<code>dma_periph_width_config</code>
<b>Function prototype</b>	<code>void dma_periph_width_config(dma_channel_enum channelx, uint32_t pwidth);</code>
<b>Function descriptions</b>	configure transfer data width of peripheral
<b>Precondition</b>	corresponding channel enable bit CHEN should be 0
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>channelx</b>	DMA channel
<b>DMA_CHx( x=0..4)</b>	DMA channel selection
<b>Input parameter{in}</b>	
<b>pwidth</b>	transfer data width of peripheral
<i>DMA_PERIPHERAL_WIDTH_8BIT</i>	transfer data width of peripheral is 8-bit
<i>DMA_PERIPHERAL_WIDTH_16BIT</i>	transfer data width of peripheral is 16-bit
<i>DMA_PERIPHERAL_WIDTH_32BIT</i>	transfer data width of peripheral is 32-bit
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* configure DMA channel0 periph width */
```

---

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

### **dma\_memory\_increase\_enable**

The description of `dma_memory_increase_enable` is shown as below:

**Table 3-92. Function `dma_memory_increase_enable`**

<b>Function name</b>	dma_memory_increase_enable
<b>Function prototype</b>	void dma_memory_increase_enable(dma_channel_enum channelx);
<b>Function descriptions</b>	enable next address increasement algorithm of memory
<b>Precondition</b>	corresponding channel enable bit CHEN should be 0
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>channelx</b>	DMA channel
<b>DMA_CHx(x=0..4)</b>	DMA channel selection
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

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

### **dma\_memory\_increase\_disable**

The description of `dma_memory_increase_disable` is shown as below:

**Table 3-93. Function `dma_memory_increase_disable`**

<b>Function name</b>	dma_memory_increase_disable
<b>Function prototype</b>	void dma_memory_increase_disable(dma_channel_enum channelx);
<b>Function descriptions</b>	disable next address increasement algorithm of memory
<b>Precondition</b>	corresponding channel enable bit CHEN should be 0
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>channelx</b>	DMA channel
<b>DMA_CHx(x=0..4)</b>	DMA channel selection
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* disable DMA channel0 memory increase */
```

---

```
dma_memory_increase_disable(DMA_CH0);
```

### **dma\_periph\_increase\_enable**

The description of `dma_periph_increase_enable` is shown as below:

**Table 3-94. Function `dma_periph_increase_enable`**

<b>Function name</b>	dma_periph_increase_enable
<b>Function prototype</b>	void dma_periph_increase_enable(dma_channel_enum channelx);
<b>Function descriptions</b>	enable next address increasement algorithm of peripheral
<b>Precondition</b>	corresponding channel enable bit CHEN should be 0
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>channelx</b>	DMA channel
<i>DMA_CHx(x=0..4)</i>	DMA channel selection
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

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

### **dma\_periph\_increase\_disable**

The description of `dma_periph_increase_disable` is shown as below:

**Table 3-95. Function `dma_periph_increase_disable`**

<b>Function name</b>	dma_periph_increase_disable
<b>Function prototype</b>	void dma_periph_increase_disable(dma_channel_enum channelx);
<b>Function descriptions</b>	disable next address increasement algorithm of peripheral
<b>Precondition</b>	corresponding channel enable bit CHEN should be 0
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>channelx</b>	DMA channel
<i>DMA_CHx(x=0..4)</i>	DMA channel selection
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* disable next address increasement algorithm of DMA channel0 */
```

---

```
dma_periph_increase_disable(DMA_CH0);
```

### **dma\_transfer\_direction\_config**

The description of `dma_transfer_direction_config` is shown as below:

**Table 3-96. Function `dma_transfer_direction_config`**

<b>Function name</b>	dma_transfer_direction_config
<b>Function prototype</b>	void dma_transfer_direction_config(dma_channel_enum channelx, uint32_t direction);
<b>Function descriptions</b>	configure the direction of data transfer on the channel
<b>Precondition</b>	corresponding channel enable bit CHEN should be 0
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>channelx</b>	DMA channel
<i>DMA_CHx( x=0..4)</i>	DMA channel selection
<b>Input parameter{in}</b>	
<b>direction</b>	specify the direction of data transfer
<i>DMA_PERIPHERAL_TO_MEMORY</i>	read from peripheral and write to memory
<i>DMA_MEMORY_TO_PERIPHERAL</i>	read from memory and write to peripheral
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

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

### **dma\_flag\_get**

The description of `dma_flag_get` is shown as below:

**Table 3-97. Function `dma_flag_get`**

<b>Function name</b>	dma_flag_get
<b>Function prototype</b>	FlagStatus dma_flag_get(dma_channel_enum channelx, uint32_t flag);
<b>Function descriptions</b>	check DMA flag is set or not
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>channelx</b>	DMA channel
<i>DMA_CHx( x=0..4)</i>	DMA channel selection

Input parameter{in}	
<b>flag</b>	specify get which flag
<i>DMA_FLAG_G</i>	global interrupt flag of channel
<i>DMA_FLAG_FTF</i>	full transfer finish flag of channel
<i>DMA_FLAG_HTF</i>	half transfer finish flag of channel
<i>DMA_FLAG_ERR</i>	error flag of channel
Output parameter{out}	
-	-
Return value	
<b>FlagStatus</b>	SET or RESET

Example:

```
/* get DMA channel0 flag */

FlagStatus flag = RESET;

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

### **dma\_flag\_clear**

The description of **dma\_flag\_clear** is shown as below:

**Table 3-98. Function **dma\_flag\_clear****

<b>Function name</b>	dma_flag_clear
<b>Function prototype</b>	void dma_flag_clear(dma_channel_enum channelx, uint32_t flag);
<b>Function descriptions</b>	clear DMA a channel flag
<b>Precondition</b>	-
<b>The called functions</b>	-
Input parameter{in}	
<b>channelx</b>	DMA channel
<i>DMA_CHx( x=0..4)</i>	DMA channel selection
Input parameter{in}	
<b>flag</b>	specify get which flag
<i>DMA_FLAG_G</i>	global interrupt flag of channel
<i>DMA_FLAG_FTF</i>	full transfer finish flag of channel
<i>DMA_FLAG_HTF</i>	half transfer finish flag of channel
<i>DMA_FLAG_ERR</i>	error flag of channel
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* clear DMA channel0 flag */
```

dma\_flag\_clear(DMA\_CH0, DMA\_FLAG\_FTF);

### **dma\_interrupt\_flag\_get**

The description of dma\_interrupt\_flag\_get is shown as below:

**Table 3-99. Function dma\_interrupt\_flag\_get**

<b>Function name</b>	dma_interrupt_flag_get
<b>Function prototype</b>	FlagStatus dma_interrupt_flag_get(dma_channel_enum channelx, uint32_t flag);
<b>Function descriptions</b>	check DMA flag and interrupt enable bit is set or not
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
channelx	DMA channel
DMA_CHx(x=0..4)	DMA channel selection
<b>Input parameter{in}</b>	
flag	specify get which flag
DMA_INT_FLAG_FTF	full transfer finish interrupt flag of channel
DMA_INT_FLAG_HTF	half transfer finish interrupt flag of channel
DMA_INT_FLAG_ERR	error interrupt flag of channel
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
FlagStatus	SET or RESET

Example:

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

### **dma\_interrupt\_flag\_clear**

The description of dma\_interrupt\_flag\_clear is shown as below:

**Table 3-100. Function dma\_interrupt\_flag\_clear**

<b>Function name</b>	dma_interrupt_flag_clear
<b>Function prototype</b>	void dma_interrupt_flag_clear(dma_channel_enum channelx, uint32_t flag);
<b>Function descriptions</b>	clear DMA a channel flag
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
channelx	DMA channel

<i>DMA_CHx( x=0..4)</i>	DMA channel selection
<b>Input parameter{in}</b>	
<i>flag</i>	specify get which flag
<i>DMA_INT_FLAG_G</i>	global interrupt flag of channel
<i>DMA_INT_FLAG_FTF</i>	full transfer finish interrupt flag of channel
<i>DMA_INT_FLAG_HTF</i>	half transfer finish interrupt flag of channel
<i>DMA_INT_FLAG_ERR</i>	error interrupt flag of channel
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* get DMA interrupt_flag */

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

### **dma\_interrupt\_enable**

The description of `dma_interrupt_enable` is shown as below:

**Table 3-101. Function `dma_interrupt_enable`**

<b>Function name</b>	dma_interrupt_enable
<b>Function prototype</b>	void dma_interrupt_enable(dma_channel_enum channelx, uint32_t source);
<b>Function descriptions</b>	enable DMA interrupt
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<i>channelx</i>	DMA channel
<i>DMA_CHx( x=0..4)</i>	DMA channel selection
<b>Input parameter{in}</b>	
<i>source</i>	DMA interrupt source
<i>DMA_INT_FTF</i>	full transfer finish interrupt of channel
<i>DMA_INT_HTF</i>	half transfer finish interrupt of channel
<i>DMA_INT_ERR</i>	error interrupt of channel
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* enable DMA channel0 interrupt */
```

---

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

### **dma\_interrupt\_disable**

The description of `dma_interrupt_disable` is shown as below:

**Table 3-102. Function `dma_interrupt_disable`**

<b>Function name</b>	dma_interrupt_disable
<b>Function prototype</b>	void dma_interrupt_disable(dma_channel_enum channelx, uint32_t source);
<b>Function descriptions</b>	disable DMA interrupt
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>channelx</b>	DMA channel
<b>DMA_CHx( x=0..4)</b>	DMA channel selection
<b>Input parameter{in}</b>	
<b>source</b>	DMA interrupt source
<b>DMA_INT_FTF</b>	full transfer finish interrupt of channel
<b>DMA_INT_HTF</b>	half transfer finish interrupt of channel
<b>DMA_INT_ERR</b>	error interrupt of channel
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* disable DMA channel0 interrupt */

dma_interrupt_disable(DMA_CH0, DMA_INT_FTF);
```

## **3.7. EXTI**

EXTI is the interrupt/event controller in the MCU. It contains up to 21 independent edge detectors and generates interrupt requests or events to the processor. The EXTI registers are listed in chapter [3.7.1](#), the EXTI firmware functions are introduced in chapter [3.7.2](#).

### **3.7.1. Descriptions of Peripheral registers**

EXTI registers are listed in the table shown as below:

**Table 3-103. EXTI Registers**

Registers	Descriptions
EXTI_INTEN	Interrupt enable register
EXTI_EVENT	Event enable register

<b>Registers</b>	<b>Descriptions</b>
EXTI_RTEN	Rising edge trigger enable register
EXTI_FTEN	Falling edge trigger enable register
EXTI_SWIEV	Software interrupt event register
EXTI_PD	Pending register

### 3.7.2. Descriptions of Peripheral functions

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

**Table 3-104. EXTI firmware function**

<b>Function name</b>	<b>Function description</b>
exti_deinit	reset the value of all EXTI registers with initial values
exti_init	initialize EXTI line x
exti_interrupt_enable	enable EXTI line x interrupt
exti_event_enable	enable EXTI line x event
exti_interrupt_disable	disable EXTI line x interrupt
exti_event_disable	disable EXTI line x event
exti_flag_get	get EXTI line x flag
exti_flag_clear	clear EXTI line x flag
exti_interrupt_flag_get	get EXTI line x interrupt flag
exti_interrupt_flag_clear	clear EXTI line x interrupt flag
exti_software_interrupt_enable	enable EXTI line x software interrupt
exti_software_interrupt_disable	disable EXTI line x software interrupt

#### Enum exti\_line\_enum

**Table 3-105. exti\_line\_enum**

<b>enum name</b>	<b>Function description</b>
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
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

enum name	Function description
EXTI_14	EXTI line 14
EXTI_15	EXTI line 15
EXTI_16	EXTI line 16
EXTI_17	EXTI line 17
EXTI_19	EXTI line 19
EXTI_25	EXTI line 25
EXTI_26	EXTI line 26
EXTI_27	EXTI line 27

### Enum exti\_mode\_enum

**Table 3-106. exti\_mode\_enum**

enum name	Function description
EXTI_INTERRUPT	EXTI interrupt mode
EXTI_EVENT	EXTI event mode

### Enum exti\_trig\_type\_enum

**Table 3-107. exti\_trig\_type\_enum**

enum 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\_deinit

The description of exti\_deinit is shown as below:

**Table 3-108. Function exti\_deinit**

<b>Function name</b>	exti_deinit
<b>Function prototype</b>	void exti_deinit(void);
<b>Function descriptions</b>	reset the value of all EXTI registers with initial values
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
-	-
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

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

---

```
exti_deinit();
```

### **exti\_init**

The description of exti\_init is shown as below:

**Table 3-109. Function exti\_init**

<b>Function name</b>	exti_init
<b>Function prototype</b>	void exti_init(exti_line_enum linex, exti_mode_enum mode, exti_trig_type_enum trig_type);
<b>Function descriptions</b>	initialize EXTI line x
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
linex	EXTI line x
EXTI_x	x=0..17,19,21
<b>Input parameter{in}</b>	
mode	EXTI mode
EXTI_INTERRUPT	interrupt mode
EXTI_EVENT	event mode
<b>Input parameter{in}</b>	
trig_type	trigger type
EXTI_TRIG_RISING	rising edge trigger
EXTI_TRIG_FALLING	falling edge trigger
EXTI_TRIG_BOTH	rising edge and falling edge trigger
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

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 3-110. Function exti\_interrupt\_enable**

<b>Function name</b>	exti_interrupt_enable
<b>Function prototype</b>	void exti_interrupt_enable(exti_line_enum linex);
<b>Function descriptions</b>	enable EXTI line x interrupt
<b>Precondition</b>	-
<b>The called functions</b>	-

Input parameter{in}	
<b>linex</b>	EXTI line x
<i>EXTI_x</i>	x=0,1,2..27
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* enable the interrupts from EXTI line 0 */

exti_interrupt_enable(EXTI_0);
```

### **exti\_event\_enable**

The description of exti\_event\_enable is shown as below:

**Table 3-111. Function exti\_event\_enable**

<b>Function name</b>	exti_event_enable
<b>Function prototype</b>	void exti_event_enable(exti_line_enum linex);
<b>Function descriptions</b>	enable EXTI line x event
<b>Precondition</b>	-
<b>The called functions</b>	-
Input parameter{in}	
<b>linex</b>	EXTI line x
<i>EXTI_x</i>	x=0,1,2..27
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* enable the events from EXTI line 0 */

exti_event_enable(EXTI_0);
```

### **exti\_interrupt\_disable**

The description of exti\_interrupt\_disable is shown as below:

**Table 3-112. Function exti\_interrupt\_disable**

<b>Function name</b>	exti_interrupt_disable
<b>Function prototype</b>	void exti_interrupt_disable(exti_line_enum linex);
<b>Function descriptions</b>	disable EXTI line x interrupt
<b>Precondition</b>	-

<b>The called functions</b>		-
<b>Input parameter{in}</b>		
linex		EXTI line x
EXTI_x		x=0,1,2..27
<b>Output parameter{out}</b>		
-		-
<b>Return value</b>		
-		-

Example:

```
/* disable the interrupts from EXTI line 0 */
exti_interrupt_disable(EXTI_0);
```

### **exti\_event\_disable**

The description of exti\_event\_disable is shown as below:

**Table 3-113. Function exti\_event\_disable**

<b>Function name</b>	exti_event_disable	
<b>Function prototype</b>	void exti_event_disable(exti_line_enum linex);	
<b>Function descriptions</b>	disable EXTI line x event	
<b>Precondition</b>	-	
<b>The called functions</b>	-	
<b>Input parameter{in}</b>		
linex		EXTI line x
EXTI_x		x=0,1,2..27
<b>Output parameter{out}</b>		
-		-
<b>Return value</b>		
-		-

Example:

```
/* disable the events from EXTI line 0 */
exti_event_disable(EXTI_0);
```

### **exti\_flag\_get**

The description of exti\_flag\_get is shown as below:

**Table 3-114. Function exti\_flag\_get**

<b>Function name</b>	exti_flag_get	
<b>Function prototype</b>	FlagStatus exti_flag_get(exti_line_enum linex);	
<b>Function descriptions</b>	get EXTI line x flag	

<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>linex</b>	EXTI line x
<i>EXTI_x</i>	x=0,1,2..17, 19, 21
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
<b>FlagStatus</b>	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 3-115. Function exti\_flag\_clear**

<b>Function name</b>	exti_flag_clear
<b>Function prototype</b>	void exti_flag_clear(exti_line_enum linex);
<b>Function descriptions</b>	clear EXTI line x flag
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>linex</b>	EXTI line x
<i>EXTI_x</i>	x=0,1,2..17, 19, 21
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

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 3-116. Function exti\_interrupt\_flag\_get**

<b>Function name</b>	exti_interrupt_flag_get
<b>Function prototype</b>	FlagStatus exti_interrupt_flag_get(exti_line_enum linex);

<b>Function descriptions</b>	get EXTI line x interrupt flag
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>linex</b>	EXTI line x
<b>EXTI_x</b>	x=0,1,2..17, 19, 21
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
<b>FlagStatus</b>	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 3-117. Function exti\_interrupt\_flag\_clear**

<b>Function name</b>	exti_interrupt_flag_clear
<b>Function prototype</b>	void exti_interrupt_flag_clear(exti_line_enum linex);
<b>Function descriptions</b>	clear EXTI line x interrupt flag
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>linex</b>	EXTI line x
<b>EXTI_x</b>	x=0,1,2..17, 19, 21
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

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

### **exti\_software\_interrupt\_enable**

The description of exti\_software\_interrupt\_enable is shown as below:

**Table 3-118. Function exti\_software\_interrupt\_enable**

<b>Function name</b>	exti_software_interrupt_enable
----------------------	--------------------------------

<b>Function prototype</b>	void exti_software_interrupt_enable(exti_line_enum linex);
<b>Function descriptions</b>	enable EXTI line x software interrupt
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>linex</b>	EXTI line x
<i>EXTI_x</i>	x=0,1,2..17, 19, 21
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

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 3-119. Function exti\_software\_interrupt\_disable**

<b>Function name</b>	exti_software_interrupt_disable
<b>Function prototype</b>	void exti_software_interrupt_disable(exti_line_enum linex);
<b>Function descriptions</b>	disable EXTI line x software interrupt
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>linex</b>	EXTI line x
<i>EXTI_x</i>	x=0,1,2..17, 19, 21
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* disable EXTI line 0 software interrupt */
exti_software_interrupt_disable(EXTI_0);
```

## **3.8. FMC**

There is flash controller and option byte for GD32E23x series. The FMC registers are listed in chapter [3.8.1](#) the FMC firmware functions are introduced in chapter [3.8.2](#).

### 3.8.1. Descriptions of Peripheral registers

FMC registers are listed in the table shown as below:

**Table 3-120. FMC Registers**

Registers	Descriptions
FMC_WS	FMC wait state register
FMC_KEY	FMC unlock key register
FMC_OBKEY	FMC option bytes unlock key register
FMC_STAT	FMC status register
FMC_CTL	FMC control register
FMC_ADDR	FMC address register
FMC_OBSTAT	FMC option bytes status register
FMC_WP	FMC write protection register
FMC_PID	FMC product ID register

### 3.8.2. Descriptions of Peripheral functions

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

**Table 3-121. FMC firmware function**

Function name	Function description
fmc_unlock	unlock the main FMC operation
fmc_lock	lock the main FMC operation
fmc_wscnt_set	set the wait state counter value
fmc_prefetch_enable	enable pre-fetch
fmc_prefetch_disable	disable pre-fetch
fmc_page_erase	erase FMC page
fmc_mass_erase	erase FMC whole chip
fmc_doubleword_program	FMC program a double word at the corresponding address
fmc_word_program	FMC program a word at the corresponding address
ob_unlock	unlock the option byte operation
ob_lock	lock the option byte operation
ob_reset	reload the option byte and generate a system reset
option_byte_value_get	get option byte value
ob_erase	erase the option byte
ob_write_protection_enable	enable option byte write protection (OB_WP)
ob_security_protection_config	configure read out protect
ob_user_write	write the FMC option byte user
ob_data_program	write the FMC option byte data
ob_user_get	get the FMC option byte OB_USER
ob_data_get	get the FMC option byte OB_DATA
ob_write_protection_get	get the FMC option byte write protection

Function name	Function description
ob_obstat_plevel_get	get the value of FMC option byte security protection level (PLEVEL) in FMC_OBSTAT register
fmc_interrupt_enable	enable FMC interrupt
fmc_interrupt_disable	disable FMC interrupt
fmc_flag_get	get flag set or reset
fmc_flag_clear	clear the FMC pending flag
fmc_interrupt_flag_get	get interrupt flag set or reset
fmc_interrupt_flag_clear	clear the FMC interrupt pending flag by writing 1
fmc_state_get	return the FMC state
fmc_ready_wait	check FMC ready or not

## **fmc\_state\_enum**

**Table 3-122. fmc\_state\_enum**

enum name	enum description
FMC_READY	the operation has been completed
FMC_BUSY	the operation is in progress
FMC_PGERR	program error
FMC_PGAERR	program alignment error
FMC_WPERR	erase/program protection error
FMC_TOERR	timeout error
FMC_OB_HSPC	option byte security protection code high

## **fmc\_unlock**

The description of fmc\_unlock is shown as below:

**Table 3-123. Function fmc\_unlock**

<b>Function name</b>	fmc_unlock
<b>Function prototype</b>	void fmc_unlock (void);
<b>Function descriptions</b>	unlock the main FMC operation
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
-	-
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* unlock the main FMC operation */
```

---

```
fmc_unlock( );
```

### **fmc\_lock**

The description of fmc\_lock is shown as below:

**Table 3-124. Function fmc\_lock**

<b>Function name</b>	fmc_lock
<b>Function prototype</b>	void fmc_lock(void);
<b>Function descriptions</b>	lock the main FMC operation
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
-	-
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

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

fmc_lock();
```

### **fmc\_wscnt\_set**

The description of fmc\_wscnt\_set is shown as below:

**Table 3-125. Function fmc\_wscnt\_set**

<b>Function name</b>	fmc_wscnt_set
<b>Function prototype</b>	void fmc_wscnt_set(uint32_t wscnt);
<b>Function descriptions</b>	set the wait state counter value
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
wscnt	wait state counter value
WS_WSCNT_0	FMC 0 wait
WS_WSCNT_1	FMC 1 wait
WS_WSCNT_2	FMC 2 wait
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

---

```
/* set the wait state counter value */
```

```
fmc_wscnt_set (WS_WSCNT_1);
```

### **fmc\_prefetch\_enable**

The description of fmc\_prefetch\_enable is shown as below:

**Table 3-126. Function fmc\_prefetch\_enable**

<b>Function name</b>	fmc_prefetch_enable
<b>Function prototype</b>	void fmc_prefetch_enable(void);
<b>Function descriptions</b>	enable pre-fetch
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
-	-
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* enable pre-fetch */
fmc_prefetch_enable( );
```

### **fmc\_prefetch\_disable**

The description of fmc\_prefetch\_disable is shown as below:

**Table 3-127. Function fmc\_prefetch\_disable**

<b>Function name</b>	fmc_prefetch_disable
<b>Function prototype</b>	void fmc_prefetch_disable (void);
<b>Function descriptions</b>	disable pre-fetch
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
-	-
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* disable pre-fetch */
```

---

```
fmc_prefetch_disable( );
```

### **fmc\_page\_erase**

The description of fmc\_page\_erase is shown as below:

**Table 3-128. Function fmc\_page\_erase**

<b>Function name</b>	fmc_page_erase
<b>Function prototype</b>	fmc_state_enum fmc_page_erase(uint32_t page_address);
<b>Function descriptions</b>	erase page
<b>Precondition</b>	fmc_unlock
<b>The called functions</b>	fmc_ready_wait
<b>Input parameter{in}</b>	
page_address	the page address to be erased
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
fmc_state_enum	state of FMC, the enum members can refer to members of the enum <a href="#">Table 3-122. fmc_state_enum</a>

Example:

```
/* erase page */
fmc_state_enum state = fmc_page_erase ( 0x08004000);
```

### **fmc\_mass\_erase**

The description of fmc\_mass\_erase is shown as below:

**Table 3-129. Function fmc\_mass\_erase**

<b>Function name</b>	fmc_mass_erase
<b>Function prototype</b>	fmc_state_enum fmc_mass_erase(void );
<b>Function descriptions</b>	erase whole chip
<b>Precondition</b>	fmc_unlock
<b>The called functions</b>	fmc_ready_wait
<b>Input parameter{in}</b>	
-	-
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
fmc_state_enum	state of FMC, the enum members can refer to members of the enum <a href="#">Table 3-122. fmc_state_enum</a>

Example:

```
/* erase whole chip */
```

---

```
fmc_state_enum state = fmc_mass_erase ( );
```

### **fmc\_doubleword\_program**

The description of fmc\_doubleword\_program is shown as below:

**Table 3-130. Function fmc\_doubleword\_program**

<b>Function name</b>	fmc_doubleword_program
<b>Function prototype</b>	fmc_state_enum fmc_doubleword_program(uint32_t address, uint64_t data);
<b>Function descriptions</b>	program a double word at the corresponding address
<b>Precondition</b>	fmc_unlock
<b>The called functions</b>	fmc_ready_wait
<b>Input parameter{in}</b>	
address	the address to program
<b>Input parameter{in}</b>	
data	the data to program
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
<b>fmc_state_enum</b>	state of FMC, the enum members can refer to members of the enum <a href="#">Table 3-122. fmc_state_enum</a>

Example:

```
/* program a double word at the corresponding address */
fmc_state_enum fmc_state = fmc_doubleword_program( 0x08004000,0xaabbccddeeff0055);
```

### **fmc\_word\_program**

The description of fmc\_word\_program is shown as below:

**Table 3-131. Function fmc\_word\_program**

<b>Function name</b>	fmc_word_program
<b>Function prototype</b>	fmc_state_enum fmc_word_program(uint32_t address, uint32_t data);
<b>Function descriptions</b>	program a word at the corresponding address
<b>Precondition</b>	fmc_unlock
<b>The called functions</b>	fmc_ready_wait
<b>Input parameter{in}</b>	
address	the address to program
data	the data to program
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
<b>fmc_state_enum</b>	state of FMC, the enum members can refer to members of the enum <a href="#">Table 3-122. fmc_state_enum</a>

	<a href="#"><u>3-122. fmc_state_enum</u></a>
--	----------------------------------------------

Example:

```
/* program a word at the corresponding address */
fmc_state_enum fmc_state = fmc_word_program ( 0x08004000,0xaabbccdd);
```

### **ob\_unlock**

The description of ob\_unlock is shown as below:

**Table 3-132. Function ob\_unlock**

<b>Function name</b>	ob_unlock
<b>Function prototype</b>	void ob_unlock(void);
<b>Function descriptions</b>	unlock the option byte operation
<b>Precondition</b>	fmc_unlock
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
-	-
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* unlock the option byte operation */
ob_unlock( );
```

### **ob\_lock**

The description of ob\_lock is shown as below:

**Table 3-133. Function ob\_lock**

<b>Function name</b>	ob_lock
<b>Function prototype</b>	void ob_lock(void);
<b>Function descriptions</b>	lock the option byte operation
<b>Precondition</b>	fmc_lock
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
-	-
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

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

ob_lock( );
```

### **ob\_reset**

The description of ob\_reset is shown as below:

**Table 3-134. Function ob\_reset**

<b>Function name</b>	ob_reset
<b>Function prototype</b>	void ob_reset (void);
<b>Function descriptions</b>	reload the option byte and generate a system reset
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
-	-
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* reload the option byte and generate a system reset */

ob_reset( );
```

### **option\_byte\_value\_get**

The description of option\_byte\_value\_get is shown as below:

**Table 3-135. Function option\_byte\_value\_get**

<b>Function name</b>	option_byte_value_get
<b>Function prototype</b>	uint32_t option_byte_value_get(uint32_t addr);
<b>Function descriptions</b>	get option byte value
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
-	-
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
<b>uint32_t</b>	option byte value

Example:

---

```
/* get option byte value*/

uint32_t temp;

temp = option_byte_value_get(0x1fff f800);
```

### **ob\_erase**

The description of ob\_erase is shown as below:

**Table 3-136. Function ob\_erase**

<b>Function name</b>	ob_erase
<b>Function prototype</b>	void ob_erase(void);
<b>Function descriptions</b>	erase the option byte
<b>Precondition</b>	ob_unlock
<b>The called functions</b>	fmc_ready_wait
<b>Input parameter{in}</b>	
-	-
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
<b>fmc_state_enum</b>	state of FMC, the enum members can refer to members of the enum <a href="#">Table 3-122. fmc_state_enum</a>

Example:

```
/* erase the option byte */

fmc_state_enum fmc_state = ob_erase();
```

### **ob\_write\_protection\_enable**

The description of ob\_write\_protection\_enable is shown as below:

**Table 3-137. Function ob\_write\_protection\_enable**

<b>Function name</b>	ob_write_protection_enable
<b>Function prototype</b>	fmc_state_enum ob_write_protection_enable(uint16_t ob_wp);
<b>Function descriptions</b>	enable option byte write protection (OB_WP)
<b>Precondition</b>	ob_unlock
<b>The called functions</b>	fmc_ready_wait
<b>Input parameter{in}</b>	
<b>ob_wp</b>	write protection configuration data
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
<b>fmc_state_enum</b>	state of FMC, the enum members can refer to members of the enum <a href="#">Table 3-122. fmc_state_enum</a>

Example:

```
/* enable write protection */

fmc_state_enum fmc_state = ob_write_protection_enable (0x01);
```

### **ob\_security\_protection\_config**

The description of ob\_security\_protection\_config is shown as below:

**Table 3-138. Function ob\_security\_protection\_config**

<b>Function name</b>	ob_security_protection_config
<b>Function prototype</b>	fmc_state_enum ob_security_protection_config (uint16_t ob_spc);
<b>Function descriptions</b>	configure security protection
<b>Precondition</b>	ob_unlock
<b>The called functions</b>	fmc_ready_wait
<b>Input parameter{in}</b>	
ob_spc	specify security protection
FMC_NSPC	no security protection
FMC_LSPC	low security protection
FMC_HSPC	high security protection
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
fmc_state_enum	state of FMC, the enum members can refer to members of the enum <a href="#">Table 3-122. fmc_state_enum</a>

Example:

```
/* enable security protection */

fmc_state_enum fmc_state;

fmc_state = ob_security_protection_config (FMC_USPC);
```

### **ob\_user\_write**

The description of ob\_user\_write is shown as below:

**Table 3-139. Function ob\_user\_write**

<b>Function name</b>	ob_user_write
<b>Function prototype</b>	fmc_state_enum ob_user_write(uint8_t ob_user);
<b>Function descriptions</b>	program the FMC user option byte
<b>Precondition</b>	ob_unlock
<b>The called functions</b>	fmc_ready_wait
<b>Input parameter{in}</b>	
ob_user	user option byte
OB_FWDGT_HW	hardware free watchdog timer

<i>OB_DEEPSLEEP_RST</i>	no reset when entering deepsleep mode
<i>OB_STDBY_RST</i>	no reset when entering deepsleep mode
<i>OB_BOOT1_SET_1</i>	BOOT1 bit is 1
<i>OB_VDDA_DISABLE</i>	disable VDDA monitor
<i>OB_SRAM_PARITY_ENABLE</i>	enable SRAM parity check
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
<i>fmc_state_enum</i>	state of FMC, the enum members can refer to members of the enum <a href="#">Table 3-122. fmc_state_enum</a>

Example:

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

fmc_state_enum fmc_state = ob_user_write(OB_FWDGT_HW);
```

### **ob\_data\_program**

The description of **ob\_data\_program** is shown as below:

**Table 3-140. Function ob\_data\_program**

<b>Function name</b>	ob_data_program
<b>Function prototype</b>	<i>fmc_state_enum ob_data_program(uint16_t data);</i>
<b>Function descriptions</b>	program the FMC data option byte
<b>Precondition</b>	<i>ob_unlock</i>
<b>The called functions</b>	<i>fmc_ready_wait</i>
<b>Input parameter{in}</b>	
<b>data</b>	the data to be programmed, OB_DATA[0:15]
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
<i>fmc_state_enum</i>	state of FMC, the enum members can refer to members of the enum <a href="#">Table 3-122. fmc_state_enum</a>

Example:

```
/* program option bytes data */

fmc_state_enum fmc_state = ob_data_program (0x56);
```

### **ob\_user\_get**

The description of **ob\_user\_get** is shown as below:

**Table 3-141. Function ob\_user\_get**

<b>Function name</b>	ob_user_get
----------------------	-------------

<b>Function prototype</b>	uint8_t ob_user_get(void);
<b>Function descriptions</b>	get OB_USER in register FMC_OBSTAT
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
-	-
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
<b>uint8_t</b>	the FMC user option byte values(0x00 – 0xFF)

Example:

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

uint8_t user = ob_user_get();
```

### **ob\_data\_get**

The description of ob\_data\_get is shown as below:

**Table 3-142. Function ob\_data\_get**

<b>Function name</b>	ob_data_get
<b>Function prototype</b>	uint16_t ob_data_get(void);
<b>Function descriptions</b>	get OB_DATA in register FMC_OBSTAT
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
-	-
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
<b>uint16_t</b>	the FMC data option byte values(0x0 – 0xFFFF)

Example:

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

uint16_t data = ob_data_get();
```

### **ob\_write\_protection\_get**

The description of ob\_write\_protection\_get is shown as below:

**Table 3-143. Function ob\_write\_protection\_get**

<b>Function name</b>	ob_write_protection_get
<b>Function prototype</b>	uint16_t ob_write_protection_get(void);

<b>Function descriptions</b>	get the FMC option byte write protection (OB_WP) in register FMC_WP
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
-	-
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
<b>uint16_t</b>	the FMC write protection option byte value(0x0 – 0xFFFF)

Example:

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

uint16_t wp = ob_write_protection_get();
```

### **ob\_obstat\_plevel\_get**

The description of ob\_obstat\_plevel\_get is shown as below:

**Table 3-144. Function ob\_obstat\_plevel\_get**

<b>Function name</b>	ob_obstat_plevel_get
<b>Function prototype</b>	uint32_t ob_obstat_plevel_get(void);
<b>Function descriptions</b>	get the value of FMC option byte security protection level (PLEVEL) in FMC_OBSTAT register
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
-	-
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
<b>uint8_t</b>	the value of PLEVEL(0x0,0x01,0x03)

Example:

```
/* get the FMC option byte security protection level */

uint32_t obstat_plevel = ob_obstat_plevel_get();
```

### **fmc\_interrupt\_enable**

The description of fmc\_interrupt\_enable is shown as below:

**Table 3-145. Function fmc\_interrupt\_enable**

<b>Function name</b>	fmc_interrupt_enable
<b>Function prototype</b>	void fmc_interrupt_enable(uint32_t interrupt);

<b>Function descriptions</b>	enable FMC interrupt
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>interrupt</b>	the FMC interrupt source
<i>FMC_INT_END</i>	FMC end of program interrupt
<i>FMC_INT_ERR</i>	FMC error interrupt
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

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

### **fmc\_interrupt\_disable**

The description of fmc\_interrupt\_disable is shown as below:

**Table 3-146. Function fmc\_interrupt\_disable**

<b>Function name</b>	fmc_interrupt_disable
<b>Function prototype</b>	void fmc_interrupt_disable(uint32_t interrupt);
<b>Function descriptions</b>	disable FMC interrupt
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>interrupt</b>	the FMC interrupt source
<i>FMC_INT_END</i>	FMC end of program interrupt
<i>FMC_INT_ERR</i>	FMC error interrupt
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* disable FMC interrupt */
fmc_interrupt_disable(FMC_INT_END);
```

### **fmc\_flag\_get**

The description of fmc\_flag\_get is shown as below:

**Table 3-147. Function fmc\_flag\_get**

<b>Function name</b>	fmc_flag_get
<b>Function prototype</b>	FlagStatus fmc_flag_get(uint32_t flag);
<b>Function descriptions</b>	check FMC flag
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<i>flag</i>	check FMC flag
<i>FMC_FLAG_BUSY</i>	FMC busy flag bit
<i>FMC_FLAG_PGERR</i>	FMC programming error flag
<i>FMC_FLAG_PGAERR</i>	FMC program alignment error flag bit
<i>FMC_FLAG_WPERR</i>	FMC write protection error flag
<i>FMC_FLAG_END</i>	FMC end of programming flag
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
<b>FlagStatus</b>	SET or RESET

Example:

```
/* get FMC flag */

FlagStatus flag = fmc_flag_get(FMC_FLAG_END);
```

### **fmc\_flag\_clear**

The description of fmc\_flag\_clear is shown as below:

**Table 3-148. Function fmc\_flag\_clear**

<b>Function name</b>	fmc_flag_clear
<b>Function prototype</b>	void fmc_flag_clear(uint32_t flag);
<b>Function descriptions</b>	clear the FMC flag by writing 1
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<i>flag</i>	clear FMC flag
<i>FMC_FLAG_PGERR</i>	FMC operation error flag
<i>FMC_FLAG_PGAERR</i>	FMC program alignment error flag
<i>FMC_FLAG_WPERR</i>	FMC erase/program protection error flag
<i>FMC_FLAG_END</i>	FMC end of operation flag
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* get FMC flag */

fmc_flag_clear(FMC_FLAG_END);
```

### **fmc\_interrupt\_flag\_get**

The description of fmc\_interrupt\_flag\_get is shown as below:

**Table 3-149. Function fmc\_interrupt\_flag\_get**

<b>Function name</b>	fmc_interrupt_flag_get
<b>Function prototype</b>	FlagStatus fmc_interrupt_flag_get(uint32_t int_flag);
<b>Function descriptions</b>	get interrupt flag set or reset
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>flag</b>	FMC flag
<i>FMC_INT_FLAG_PGE RR</i>	FMC operation error flag
<i>FMC_INT_FLAG_PGA ERR</i>	FMC program alignment error flag
<i>FMC_INT_FLAG_WPE RR</i>	FMC erase/program protection error flag
<i>FMC_INT_FLAG_END</i>	FMC end of operation flag
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
<b>FlagStatus</b>	SET or RESET

Example:

```
/* get FMC flag */

FlagStatus flag = fmc_interrupt_flag_get (FMC_INT_FLAG_PGERR);
```

### **fmc\_interrupt\_flag\_clear**

The description of fmc\_interrupt\_flag\_get is shown as below:

**Table 3-150. Function fmc\_interrupt\_flag\_clear**

<b>Function name</b>	fmc_interrupt_flag_clear
<b>Function prototype</b>	void fmc_interrupt_flag_clear(uint32_t int_flag);
<b>Function descriptions</b>	clear the FMC interrupt pending flag by writing 1
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	

<b>flag</b>	clear FMC flag
<i>FMC_INT_FLAG_PGE RR</i>	FMC operation error flag
<i>FMC_INT_FLAG_PGA ERR</i>	FMC program alignment error flag
<i>FMC_INT_FLAG_WPE RR</i>	FMC erase/program protection error flag
<i>FMC_INT_FLAG_END</i>	FMC end of operation flag
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* clear FMC flag */

fmc_interrupt_flag_get (FMC_INT_FLAG_PGERR);
```

### **fmc\_state\_get**

The description of fmc\_state\_get is shown as below:

**Table 3-151. Function fmc\_state\_get**

<b>Function name</b>	fmc_state_get
<b>Function prototype</b>	fmc_state_enum fmc_state_get(void);
<b>Function descriptions</b>	get the FMC state
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
-	-
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
<b>fmc_state_enum</b>	state of FMC, the enum members can refer to members of the enum <a href="#">Table 3-122. fmc_state_enum</a>

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 3-152. Function fmc\_ready\_wait**

<b>Function name</b>	fmc_ready_wait
<b>Function prototype</b>	fmc_state_enum fmc_ready_wait(uint32_t timeout);
<b>Function descriptions</b>	check whether FMC is ready or not
<b>Precondition</b>	-
<b>The called functions</b>	fmc_state_get()
<b>Input parameter{in}</b>	
timeout	timeout count
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
<b>fmc_state_enum</b>	state of FMC, the enum members can refer to members of the enum <a href="#">Table 3-122. fmc_state_enum</a>

Example:

```
/* check whether FMC is ready or not */
fmc_state_enum state = fmc_ready_wait (0x00001000 );
```

## 3.9. 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 [3.9.1](#) the FWDGT firmware functions are introduced in chapter [3.9.2](#).

### 3.9.1. Descriptions of Peripheral registers

FWDGT registers are listed in the table shown as below:

**Table 3-153. FWDGT Registers**

Registers	Descriptions
FWDGT_CTL	Control register
FWDGT_PSC	Prescaler register
FWDGT_RLD	Reload register
FWDGT_STAT	Status register
FWDGT_WND	window register

### 3.9.2. Descriptions of Peripheral functions

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

**Table 3-154. FWDGT firmware function**

<b>Function name</b>	<b>Function description</b>
fwdgt_write_enable	enable write access to FWDGT_PSC and FWDGT_RLD and FWDGT_WND
fwdgt_write_disable	disable write access to FWDGT_PSC and FWDGT_RLD and FWDGT_WND
fwdgt_enable	start the FWDGT counter
fwdgt_prescaler_value_config	configure the FWDGT counter prescaler value
fwdgt_reload_value_config	configure the FWDGT counter reload value
fwdgt_window_value_config	configure the FWDGT counter window 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 3-155. Function fwdgt\_write\_enable**

<b>Function name</b>	fwdgt_write_enable
<b>Function prototype</b>	void fwdgt_write_enable(void);
<b>Function descriptions</b>	enable write access to FWDGT_PSC and FWDGT_RLD and FWDGT_WND
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
-	-
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* enable write access to FWDGT_PSC and FWDGT_RLD and FWDGT_WND */

fwdgt_write_enable();
```

### **fwdgt\_write\_disable**

The description of fwdgt\_write\_disable is shown as below:

**Table 3-156. Function fwdgt\_write\_disable**

<b>Function name</b>	fwdgt_write_disable
<b>Function prototype</b>	void fwdgt_write_disable(void);
<b>Function descriptions</b>	disable write access to FWDGT_PSC,FWDGT_RLD and FWDGT_WND

<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
-	-
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* disable write access to FWDGT_PSC,FWDGT_RLD and FWDGT_WND */
fwdgt_write_disable();
```

### **fwdgt\_enable**

The description of fwdgt\_enable is shown as below:

**Table 3-157. Function fwdgt\_enable**

<b>Function name</b>	fwdgt_enable
<b>Function prototype</b>	void fwdgt_enable(void);
<b>Function descriptions</b>	start the FWDGT counter
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
-	-
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

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 3-158. Function fwdgt\_prescaler\_value\_config**

<b>Function name</b>	fwdgt_prescaler_value_config
<b>Function prototype</b>	ErrStatus fwdgt_prescaler_value_config(uint16_t prescaler_value);
<b>Function descriptions</b>	configure the FWDGT counter clock prescaler value
<b>Precondition</b>	-

The called functions	
Input parameter{in}	
prescaler_value	specify prescaler value
FWDGT_PSC_DIVx	FWDGT prescaler set to x(x=4,8,16,32,64,128,256)
Output parameter{out}	
-	-
Return value	
ErrStatus	ERROR / SUCCESS

Example:

```
/* set FWDGT prescaler to 4 */
ErrStatus flag;
flag = fwdgt_prescaler_value_config (FWDGT_PSC_DIV4);
```

### **fwdgt\_reload\_value\_config**

The description of fwdgt\_reload\_value\_config is shown as below:

**Table 3-159. 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 FWDGT counter reload value
Precondition	-
The called functions	-
Input parameter{in}	
reload_value	reload_value: specify reload value(0x0000 - 0x0FFF)
Output parameter{out}	
-	-
Return value	
ErrStatus	ERROR / SUCCESS

Example:

```
/* set FWDGT reload value to 0xFFFF */
ErrStatus flag;
flag = fwdgt_reload_value_config (0xFFFF);
```

### **fwdgt\_window\_value\_config**

The description of fwdgt\_window\_value\_config is shown as below:

**Table 3-160. Function fwdgt\_window\_value\_config**

Function name	fwdgt_window_value_config
Function prototype	ErrStatus fwdgt_window_value_config(uint16_t window_value);

<b>Function descriptions</b>	configure the FWDGT counter window value
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>window_value</b>	window_value: specify window value(0x0000 - 0xFFFF)
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
<b>ErrStatus</b>	ERROR / SUCCESS

Example:

```
/* set FWDGT window value to 0xFFFF */
ErrStatus flag;
flag = fwdgt_window_value_config (0xFFFF);
```

### **fwdgt\_counter\_reload**

The description of fwdgt\_counter\_reload is shown as below:

**Table 3-161. Function fwdgt\_counter\_reload**

<b>Function name</b>	fwdgt_counter_reload
<b>Function prototype</b>	void fwdgt_counter_reload(void);
<b>Function descriptions</b>	reload the counter of FWDGT
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
-	-
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* reload FWDGT counter */
fwdgt_counter_reload ( );
```

### **fwdgt\_config**

The description of fwdgt\_config is shown as below:

**Table 3-162. Function fwdgt\_config**

<b>Function name</b>	fwdgt_config
<b>Function prototype</b>	ErrStatus fwdgt_config(uint16_t reload_value, uint8_t prescaler_div);

<b>Function descriptions</b>	configure counter reload value, and prescaler divider value
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>reload_value</b>	specify reload value(0x0000 - 0x0FFF)
<b>Input parameter{in}</b>	
<b>prescaler_div</b>	FWDGT 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
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
<b>ErrStatus</b>	ERROR or SUCCESS

Example:

```
/* confiure 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 3-163. Function fwdgt\_flag\_get**

<b>Function name</b>	fwdgt_flag_get
<b>Function prototype</b>	FlagStatus fwdgt_flag_get(uint16_t flag);
<b>Function descriptions</b>	get flag state of FWDGT
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>flag</b>	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
<i>FWDGT_FLAG_WUD</i>	a write operation to FWDGT_WND register is on going
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
<b>FlagStatus</b>	SET or RESET

Example:

```
/* test if a prescaler value update is on going */

FlagStatus status;

status = fwdgt_flag_get (FWDGT_FLAG_PUD);
```

## 3.10. GPIO

GPIO is used to implement logic input/output functions for the devices. The GPIO registers are listed in chapter [3.10.1](#), the GPIO firmware functions are introduced in chapter [3.10.2](#).

### 3.10.1. Descriptions of Peripheral registers

GPIO registers are listed in the table shown as below:

**Table 3-164. GPIO Registers**

Registers	Descriptions
GPIOx_CTL	GPIO port control register
GPIOx_OMODE	GPIO port output mode register
GPIOx_OSPEED0	GPIO port output speed register 0
GPIOx_PUD	GPIO port pull-up/pull-down register
GPIOx_ISTAT	GPIO port input status register
GPIOx_OCTL	GPIO port output control register
GPIOx_BOP	GPIO port bit operation register
GPIOx_LOCK	GPIO port configuration lock register
GPIOx_AFSEL0	GPIO alternate function selected register 0
GPIOx_AFSEL1	GPIO alternate function selected register 1
GPIOx_BC	GPIO bit clear register
GPIOx_TG	GPIO port bit toggle register

### 3.10.2. Descriptions of Peripheral functions

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

**Table 3-165. 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

Function name	Function description
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 3-166. 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
<b>Input parameter{in}</b>	
gpio_periph	GPIO port
GPIOx	GPIOx(x = A,B,C, F)
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* reset GPIOA */
gpio_deinit(GPIOA);
```

## gpio\_mode\_set

The description of gpio\_mode\_set is shown as below:

**Table 3-167. 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
<b>Input parameter{in}</b>	

<b>gpio_periph</b>	GPIO port
<b>GPIOx</b>	GPIOx(x = A,B,C,F)
<b>Input parameter{in}</b>	
<b>mode</b>	gpio pin mode
<b>GPIO_MODE_INPUT</b>	input mode
<b>GPIO_MODE_OUTPUT_T</b>	output mode
<b>GPIO_MODE_AF</b>	alternate function mode
<b>GPIO_MODE_ANALOG</b>	analog mode
<b>Input parameter{in}</b>	
<b>pull_up_down</b>	gpio pin with pull-up or pull-down resistor
<b>GPIO_PUPD_NONE</b>	floating mode, no pull-up and pull-down resistors
<b>GPIO_PUPD_PULLUP</b>	with pull-up resistor
<b>GPIO_PUPD_PULLDOWN</b>	with pull-down resistor
<b>Input parameter{in}</b>	
<b>pin</b>	GPIO pin
<b>GPIO_PIN_x</b>	GPIO_PIN_x(x=0..15) ( PB9/PC13 does not exist on GD32E231)
<b>GPIO_PIN_ALL</b>	All pins ( PB9/PC13 does not exist on GD32E231)
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* config PA0 as input mode with pullup*/
gpio_mode_set(GPIOA, GPIO_MODE_INPUT, GPIO_PUPD_PULLUP, GPIO_PIN_0);
```

### **gpio\_output\_options\_set**

The description of **gpio\_output\_options\_set** is shown as below:

**Table 3-168. Function gpio\_output\_options\_set**

<b>Function name</b>	gpio_output_options_set
<b>Function prototype</b>	void gpio_output_options_set(uint32_t gpio_periph, uint8_t otype, uint32_t speed, uint32_t pin);
<b>Function descriptions</b>	set GPIO output type and speed
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>gpio_periph</b>	GPIO port
<b>GPIOx</b>	GPIOx(x = A,B,C,F)

Input parameter{in}	
<b>otype</b>	gpio pin output mode
<i>GPIO_OTYPE_PP</i>	push pull mode
<i>GPIO_OTYPE_OD</i>	open drain mode
Input parameter{in}	
<b>speed</b>	gpio pin output max speed
<i>GPIO_OSPEED_2MHZ</i>	output max speed 2MHz
<i>GPIO_OSPEED_10MHZ</i>	output max speed 10MHz
<i>GPIO_OSPEED_50MHZ</i>	output max speed 50MHz
Input parameter{in}	
<b>pin</b>	GPIO pin
<i>GPIO_PIN_x</i>	<i>GPIO_PIN_x</i> (x=0..15) ( PB9/PC13 does not exist on GD32E231)
<i>GPIO_PIN_ALL</i>	All pins ( PB9/PC13 does not exist on GD32E231)
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* config PA0 as push pull mode */
gpio_output_options_set(GPIOA, GPIO_OTYPE_PP, GPIO_OSPEED_2MHZ, GPIO_PIN_0);
```

### gpio\_bit\_set

The description of gpio\_bit\_set is shown as below:

**Table 3-169. Function gpio\_bit\_set**

<b>Function name</b>	gpio_bit_set
<b>Function prototype</b>	void gpio_bit_set(uint32_t gpio_periph,uint32_t pin);
<b>Function descriptions</b>	set GPIO pin bit
<b>Precondition</b>	-
<b>The called functions</b>	-
Input parameter{in}	
<b>gpio_periph</b>	GPIO port
<i>GPIOx</i>	<i>GPIOx</i> (x = A,B,C, F)
Input parameter{in}	
<b>pin</b>	GPIO pin
<i>GPIO_PIN_x</i>	<i>GPIO_PIN_x</i> (x=0..15) ( PB9/PC13 does not exist on GD32E231)
<i>GPIO_PIN_ALL</i>	All pins ( PB9/PC13 does not exist on GD32E231)
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 3-170. Function gpio\_bit\_reset**

<b>Function name</b>	gpio_bit_reset
<b>Function prototype</b>	void gpio_bit_reset(uint32_t gpio_periph,uint32_t pin);
<b>Function descriptions</b>	reset GPIO pin
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
gpio_periph	GPIO port
GPIOx	GPIOx(x = A,B,C,F)
<b>Input parameter{in}</b>	
pin	GPIO pin
GPIO_PIN_x	GPIO_PIN_x(x=0..15) ( PB9/PC13 does not exist on GD32E231)
GPIO_PIN_ALL	All pins ( PB9/PC13 does not exist on GD32E231)
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* reset PA0*/
gpio_bit_set(GPIOA, GPIO_PIN_0);
```

### gpio\_bit\_write

The description of gpio\_bit\_write is shown as below:

**Table 3-171. Function gpio\_bit\_write**

<b>Function name</b>	gpio_bit_write
<b>Function prototype</b>	void gpio_bit_write(uint32_t gpio_periph,uint32_t pin,bit_status bit_value);
<b>Function descriptions</b>	write data to the specified GPIO pin
<b>Precondition</b>	-
<b>The called functions</b>	-

Input parameter{in}	
<b>gpio_periph</b>	GPIO port
<b>GPIOx</b>	GPIOx(x = A,B,C,F)
Input parameter{in}	
<b>pin</b>	GPIO pin
<b>GPIO_PIN_x</b>	GPIO_PIN_x(x=0..15) ( PB9/PC13 does not exist on GD32E231)
<b>GPIO_PIN_ALL</b>	All pins( PB9/PC13 does not exist on GD32E231)
Input parameter{in}	
<b>bit_value</b>	SET or RESET
<b>RESET</b>	clear the port pin
<b>SET</b>	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 3-172. Function gpio\_port\_write**

<b>Function name</b>	gpio_port_write
<b>Function prototype</b>	void gpio_port_write(uint32_t gpio_periph,uint16_t data);
<b>Function descriptions</b>	write data to the specified GPIO port
<b>Precondition</b>	-
<b>The called functions</b>	-
Input parameter{in}	
<b>gpio_periph</b>	GPIO port
<b>GPIOx</b>	GPIOx(x = A,B,C,F)
Input parameter{in}	
<b>data</b>	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 3-173. Function gpio\_input\_bit\_get**

<b>Function name</b>	gpio_input_bit_get
<b>Function prototype</b>	FlagStatus gpio_input_bit_get(uint32_t gpio_periph,uint32_t pin);
<b>Function descriptions</b>	get GPIO pin input status
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
gpio_periph	GPIO port
GPIOx	GPIOx(x = A,B,C,F)
<b>Input parameter{in}</b>	
pin	GPIO pin
GPIO_PIN_x	GPIO_PIN_x(x=0..15) ( PB9/PC13 does not exist on GD32E231)
GPIO_PIN_ALL	All pins ( PB9/PC13 does not exist on GD32E231)
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
FlagStatus	SET / RESET

Example:

```
/* get status of PA0 */
FlagStatus 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 3-174. Function gpio\_input\_port\_get**

<b>Function name</b>	gpio_input_port_get
<b>Function prototype</b>	uint16_t gpio_input_port_get(uint32_t gpio_periph);
<b>Function descriptions</b>	get GPIO all pins input status
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
gpio_periph	GPIO port
GPIOx	GPIOx(x = A,B,C,F)
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
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 3-175. Function `gpio_output_bit_get`**

<b>Function name</b>	gpio_output_bit_get
<b>Function prototype</b>	FlagStatus gpio_output_bit_get(uint32_t gpio_periph,uint32_t pin);
<b>Function descriptions</b>	get GPIO pin output status
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>gpio_periph</b>	GPIO port
<b>GPIOx</b>	GPIOx(x = A,B,C,F)
<b>Input parameter{in}</b>	
<b>pin</b>	GPIO pin
<b>GPIO_PIN_x</b>	GPIO_PIN_x(x=0..15) ( PB9/PC13 does not exist on GD32E231)
<b>GPIO_PIN_ALL</b>	All pins ( PB9/PC13 does not exist on GD32E231)
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
<b>FlagStatus</b>	SET / 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 3-176. Function `gpio_output_port_get`**

<b>Function name</b>	gpio_output_port_get
<b>Function prototype</b>	uint16_t gpio_output_port_get(uint32_t gpio_periph);
<b>Function descriptions</b>	get GPIO all pins output status
<b>Precondition</b>	-
<b>The called functions</b>	-

Input parameter{in}	
<b>gpio_periph</b>	GPIO port
<b>GPIOx</b>	GPIOx(x = A,B,C,F)
Output parameter{out}	
-	-
Return value	
<b>Uint16_t</b>	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 3-177. Function gpio\_af\_set**

<b>Function name</b>	gpio_af_set
<b>Function prototype</b>	void gpio_af_set(uint32_t gpio_periph, uint32_t alt_func_num, uint32_t pin);
<b>Function descriptions</b>	set GPIO alternate function
<b>Precondition</b>	-
<b>The called functions</b>	-
Input parameter{in}	
<b>gpio_periph</b>	GPIO port
<b>GPIOx</b>	GPIOx(x = A,B,C)
Input parameter{in}	
<b>alt_func_num</b>	GPIO pin af function, please refer to specific device datasheet
<b>GPIO_AF_0</b>	<i>TIMER13, TIMER14, TIMER16, SPI0, SPI1, I2S0, CK_OUT, USART0, I2C0, I2C1, SWDIO, SWCLK</i>
<b>GPIO_AF_1</b>	<i>USART0, USART1, TIMER2, TIMER14, I2C0, I2C1</i>
<b>GPIO_AF_2</b>	<i>TIMER0, TIMER1, TIMER15, TIMER16, I2S0</i>
<b>GPIO_AF_3</b>	<i>I2C0, TIMER14</i>
<b>GPIO_AF_4 (port A,B only)</b>	<i>USART1, I2C0, I2C1, TIMER13</i>
<b>GPIO_AF_5 (port A,B only)</b>	<i>TIMER15, TIMER16, I2S0</i>
<b>GPIO_AF_6 (port A,B only)</b>	<i>SPI1</i>
<b>GPIO_AF_7 (port A,B only)</b>	<i>CMP</i>
Input parameter{in}	

<b>pin</b>	GPIO pin
<b>GPIO_PIN_x</b>	GPIO_PIN_x(x=0..15) ( PB9/PC13 does not exist on GD32E231)
<b>GPIO_PIN_ALL</b>	All pins ( PB9/PC13 does not exist on GD32E231)
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

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 3-178. Function gpio\_pin\_lock**

<b>Function name</b>	gpio_pin_lock
<b>Function prototype</b>	void gpio_pin_lock(uint32_t gpio_periph, uint32_t pin);
<b>Function descriptions</b>	lock GPIO pin bit
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>gpio_periph</b>	GPIO port
<b>GPIOx</b>	GPIOx(x = A,B)
<b>Input parameter{in}</b>	
<b>pin</b>	GPIO pin
<b>GPIO_PIN_x</b>	GPIO_PIN_x(x=0..15) ( PB9 does not exist on GD32E231)
<b>GPIO_PIN_ALL</b>	All pins ( PB9 does not exist on GD32E231)
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* lock PA0*/
gpio_pin_lock (GPIOA, GPIO_PIN_0);
```

### **gpio\_bit\_toggle**

The description of gpio\_bit\_toggle is shown as below:

**Table 3-179. Function gpio\_bit\_toggle**

<b>Function name</b>	gpio_bit_toggle
<b>Function prototype</b>	void gpio_bit_toggle(uint32_t gpio_periph, uint32_t pin);
<b>Function descriptions</b>	toggle GPIO pin status
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
gpio_periph	GPIO port
GPIOx	GPIOx(x = A,B,C,F)
<b>Input parameter{in}</b>	
pin	GPIO pin
GPIO_PIN_x	GPIO_PIN_x(x=0..15) ( PB9/PC13 does not exist on GD32E231)
GPIO_PIN_ALL	GPIO_PIN_ALL ( PB9/PC13 does not exist on GD32E231)
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* toggle PA0 */
gpio_bit_toggle (GPIOA, GPIO_PIN_0);
```

### gpio\_port\_toggle

The description of gpio\_port\_toggle is shown as below:

**Table 3-180. Function gpio\_port\_toggle**

<b>Function name</b>	gpio_port_toggle
<b>Function prototype</b>	void gpio_port_toggle(uint32_t gpio_periph);
<b>Function descriptions</b>	toggle GPIO port status
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
gpio_periph	GPIO port
GPIOx	GPIOx(x = A,B,C,F)
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

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

## 3.11. 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 [3.11.1](#), the I2C firmware functions are introduced in chapter [3.11.2](#).

### 3.11.1. Descriptions of Peripheral registers

I2C registers are listed in the table shown as below:

**Table 3-181. I2C 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_SAMCS	SAM control and status register
I2C_FMPCFG	Fast mode plus configure register

### 3.11.2. Descriptions of Peripheral functions

I2C firmware functions are listed in the table shown as below:

**Table 3-182. I2C 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 ACK 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

Function name	Function description
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_enable	I2C DMA mode enable
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	software reset I2C
i2c_pec_enable	whether I2C PEC calculation is enabled
i2c_pec_transfer_enable	I2C whether to transfer PEC value
i2c_pec_value_get	packet error checking value
i2c_smbus_issue_alert	I2C issue alert through SMBA pin
i2c_smbus_arp_enable	whether ARP is enabled under SMBus
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	check I2C flag is set or not
i2c_flag_clear	clear I2C flag
i2c_interrupt_enable	enable I2C interrupt
i2c_interrupt_disable	disable I2C interrupt
i2c_interrupt_flag_get	check I2C interrupt flag
i2c_interrupt_flag_clear	clear I2C interrupt flag

## **i2c\_deinit**

The description of i2c\_deinit is shown as below:

**Table 3-183. 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
<b>Input parameter{in}</b>	
i2c_periph	I2C peripheral
I2Cx	(x=0,1)
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* reset I2C0 */

i2c_deinit(I2C0);
```

### **i2c\_clock\_config**

The description of i2c\_clock\_config is shown as below:

**Table 3-184. Function i2c\_clock\_config**

<b>Function name</b>	i2c_clock_config
<b>Function prototype</b>	void i2c_clock_config(uint32_t i2c_periph, uint32_t clkspeed, uint32_t dutycyc);
<b>Function descriptions</b>	I2C clock configure
<b>Precondition</b>	-
<b>The called functions</b>	rcu_clock_freq_get
<b>Input parameter{in}</b>	
<b>i2c_periph</b>	I2C peripheral
<i>I2Cx</i>	(x=0,1)
<b>Input parameter{in}</b>	
<b>clkspeed</b>	i2c clock speed
<b>Input parameter{in}</b>	
<b>dutycyc</b>	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
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

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 3-185. Function i2c\_mode\_addr\_config**

<b>Function name</b>	i2c_mode_addr_config
<b>Function prototype</b>	void i2c_mode_addr_config(uint32_t i2c_periph, uint32_t mode, uint32_t addformat, uint32_t addr);
<b>Function descriptions</b>	configure I2C address
<b>Precondition</b>	-

<b>The called functions</b>	
<b>Input parameter{in}</b>	
<i>i2c_periph</i>	I2C peripheral
<i>I2Cx</i>	(x=0,1)
<b>Input parameter{in}</b>	
<b>mode</b>	I2C mode select
<i>I2C_I2CMODE_ENAB</i> <i>LE</i>	I2C mode
<i>I2C_SMBUSMODE_E</i> <i>NABLE</i>	SMBus mode
<b>Input parameter{in}</b>	
<b>addformat</b>	7bits or 10bits
<i>I2C_ADDFORMAT_7B</i> <i>ITS</i>	7bits
<i>I2C_ADDFORMAT_10</i> <i>BITS</i>	10bits
<b>Input parameter{in}</b>	
<b>addr</b>	I2C address
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

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 3-186. Function i2c\_smbus\_type\_config**

<b>Function name</b>	i2c_smbus_type_config
<b>Function prototype</b>	void i2c_smbus_type_config(uint32_t i2c_periph, uint32_t type);
<b>Function descriptions</b>	SMBus type selection
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>i2c_periph</b>	I2C peripheral
<i>I2Cx</i>	(x=0,1)
<b>Input parameter{in}</b>	
<b>type</b>	Device or host
<i>I2C_SMBUS_DEVICE</i>	device

<i>I2C_SMBUS_HOST</i>	host
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

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 3-187. Function i2c\_ack\_config**

<b>Function name</b>	i2c_ack_config
<b>Function prototype</b>	void i2c_ack_config(uint32_t i2c_periph, uint32_t ack);
<b>Function descriptions</b>	whether or not to send an ACK
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<i>i2c_periph</i>	I2C peripheral
<i>I2Cx</i>	(x=0,1)
<b>Input parameter{in}</b>	
<i>ack</i>	whether or not to send an ACK
<i>I2C_ACK_ENABLE</i>	ACK will be sent
<i>I2C_ACK_DISABLE</i>	ACK will not be sent
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

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 3-188. Function i2c\_ackpos\_config**

<b>Function name</b>	i2c_ackpos_config
<b>Function prototype</b>	void i2c_ackpos_config(uint32_t i2c_periph, uint32_t pos);

<b>Function descriptions</b>	I2C POAP position configure
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>i2c_periph</b>	I2C peripheral
<i>I2Cx</i>	(x=0,1)
<b>Input parameter{in}</b>	
<b>pos</b>	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
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

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 3-189. Function i2c\_master\_addressing**

<b>Function name</b>	i2c_master_addressing
<b>Function prototype</b>	void i2c_master_addressing(uint32_t i2c_periph, uint32_t addr, uint32_t trandirection);
<b>Function descriptions</b>	master sends slave address
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>i2c_periph</b>	I2C peripheral
<i>I2Cx</i>	(x=0,1)
<b>Input parameter{in}</b>	
<b>addr</b>	slave address
<b>Input parameter{in}</b>	
<b>trandirection</b>	transmitter or receiver
<i>I2C_TRANSMITTER</i>	transmitter
<i>I2C_RECEIVER</i>	receiver
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	

-	-
---	---

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 3-190. Function i2c\_dualaddr\_enable**

<b>Function name</b>	i2c_dualaddr_enable
<b>Function prototype</b>	void i2c_dualaddr_enable(uint32_t i2c_periph, uint32_t addr)
<b>Function descriptions</b>	dual-address mode enable
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>i2c_periph</b>	I2C peripheral
<b>I2Cx</b>	(x=0,1)
<b>Input parameter{in}</b>	
<b>addr</b>	second address in dual-address mode
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* enable I2C0 dual-address*/
```

```
i2c_dualaddr_enable (I2C0, 0x80);
```

### **i2c\_dualaddr\_disable**

The description of i2c\_dualaddr\_disable is shown as below:

**Table 3-191. Function i2c\_dualaddr\_disable**

<b>Function name</b>	i2c_dualaddr_disable
<b>Function prototype</b>	void i2c_dualaddr_disable(uint32_t i2c_periph)
<b>Function descriptions</b>	dual-address mode disable
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>i2c_periph</b>	I2C peripheral
<b>I2Cx</b>	(x=0,1)

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 3-192. 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
I2Cx	(x=0,1)
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* enable I2C0 */
i2c_enable (I2C0);
```

### i2c\_disable

The description of i2c\_disable is shown as below:

**Table 3-193. 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

<i>I2Cx</i>	(x=0,1)
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* disable I2C0 */

i2c_disable (I2C0);
```

### **i2c\_start\_on\_bus**

The description of i2c\_start\_on\_bus is shown as below:

**Table 3-194. Function i2c\_start\_on\_bus**

<b>Function name</b>	i2c_start_on_bus
<b>Function prototype</b>	void i2c_start_on_bus(uint32_t i2c_periph);
<b>Function descriptions</b>	generate a START condition on I2C bus
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<i>i2c_periph</i>	I2C peripheral
<i>I2Cx</i>	(x=0,1)
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

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 3-195. Function i2c\_stop\_on\_bus**

<b>Function name</b>	i2c_stop_on_bus
<b>Function prototype</b>	void i2c_stop_on_bus(uint32_t i2c_periph);
<b>Function descriptions</b>	generate a STOP condition on I2C bus
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	

<b>i2c_periph</b>	I2C peripheral
<i>I2Cx</i>	(x=0,1)
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

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 3-196. Function i2c\_data\_transmit**

<b>Function name</b>	i2c_data_transmit
<b>Function prototype</b>	void i2c_data_transmit(uint32_t i2c_periph, uint8_t data);
<b>Function descriptions</b>	
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>i2c_periph</b>	I2C peripheral
<i>I2Cx</i>	(x=0,1)
<b>Input parameter{in}</b>	
<b>data</b>	transmit data
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* I2C0 transmit data */
```

```
i2c_data_transmit (I2C0, 0x80);
```

### **i2c\_data\_receive**

The description of i2c\_data\_receive is shown as below:

**Table 3-197. Function i2c\_data\_receive**

<b>Function name</b>	i2c_data_receive
<b>Function prototype</b>	uint8_t i2c_data_receive(uint32_t i2c_periph);
<b>Function descriptions</b>	

<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<i>i2c_periph</i>	I2C peripheral
<i>I2Cx</i>	(x=0,1)
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
<b>uint8_t</b>	0x00..0xFF

Example:

```
/* I2C0 receive data */

uint8_t i2c_receiver;

i2c_receiver = i2c_data_receive(I2C0);
```

### **i2c\_dma\_enable**

The description of **i2c\_dma\_enable** is shown as below:

**Table 3-198. Function i2c\_dma\_enable**

<b>Function name</b>	i2c_dma_enable
<b>Function prototype</b>	void i2c_dma_enable(uint32_t i2c_periph, uint32_t dmastate);
<b>Function descriptions</b>	enable I2C DMA mode
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<i>i2c_periph</i>	I2C peripheral
<i>I2Cx</i>	(x=0,1)
<b>Input parameter{in}</b>	
<i>dmastate</i>	on or off
<i>I2C_DMA_ON</i>	DMA mode enable
<i>I2C_DMA_OFF</i>	DMA mode disable
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* I2C0 DMA mode enable */

i2c_dma_enable (I2C0, I2C_DMA_ON);
```

### **i2c\_dma\_last\_transfer\_config**

The description of i2c\_dma\_last\_transfer\_config is shown as below:

**Table 3-199. Function i2c\_dma\_last\_transfer\_config**

<b>Function name</b>	i2c_dma_last_transfer_config
<b>Function prototype</b>	void i2c_dma_last_transfer_config(uint32_t i2c_periph, uint32_t dmalast);
<b>Function descriptions</b>	flag indicating DMA last transfer
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
i2c_periph	I2C peripheral
I2Cx	(x=0,1)
<b>Input parameter{in}</b>	
dmalast	next DMA EOT is the last transfer or not
I2C_DMALST_ON	next DMA EOT is the last transfer
I2C_DMALST_OFF	next DMA EOT is not the last transfer
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* next DMA EOT is the last transfer */
i2c_dma_last_transfer_config (I2C0, I2C_DMALST_ON);
```

### **i2c\_stretch\_scl\_low\_config**

The description of i2c\_stretch\_scl\_low\_config is shown as below:

**Table 3-200. Function i2c\_stretch\_scl\_low\_config**

<b>Function name</b>	i2c_stretch_scl_low_config
<b>Function prototype</b>	void i2c_stretch_scl_low_config(uint32_t i2c_periph, uint32_t stretchpara);
<b>Function descriptions</b>	whether to stretch SCL low when data is not ready in slave mode
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
i2c_periph	I2C peripheral
I2Cx	(x=0,1)
<b>Input parameter{in}</b>	
stretchpara	SCL stretching enable or disable
I2C_SCLSTRETCH_ENABLE	SCL stretching is enabled
I2C_SCLSTRETCH_DISABLE	SCL stretching is disabled

SABLE	
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

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 3-201. Function i2c\_slave\_response\_to\_gcall\_config**

<b>Function name</b>	i2c_slave_response_to_gcall_config
<b>Function prototype</b>	void i2c_slave_response_to_gcall_config(uint32_t i2c_periph, uint32_t gcallpara);
<b>Function descriptions</b>	whether or not to response to a general call
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
i2c_periph	I2C peripheral
I2Cx	(x=0,1)
<b>Input parameter{in}</b>	
gcallpara	response to a general call or not
I2C_GCEN_ENABLE	slave will response to a general call
I2C_GCEN_DISABLE	slave will not response to a general call
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

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 3-202. Function i2c\_software\_reset\_config**

<b>Function name</b>	i2c_software_reset_config
----------------------	---------------------------

<b>Function prototype</b>	void i2c_software_reset_config(uint32_t i2c_periph, uint32_t sreset);
<b>Function descriptions</b>	software reset I2C
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
i2c_periph	I2C peripheral
I2Cx	(x=0,1)
<b>Input parameter{in}</b>	
sreset	reset or not
I2C_SRESET_SET	I2C is under reset
I2C_SRESET_RESET	I2C is not under reset
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```

/* software reset I2C0 */

i2c_software_reset_config (I2C0, I2C_SRESET_SET);

```

### **i2c\_pec\_enable**

The description of i2c\_pec\_enable is shown as below:

**Table 3-203. Function i2c\_pec\_enable**

<b>Function name</b>	i2c_pec_enable
<b>Function prototype</b>	void i2c_pec_enable(uint32_t i2c_periph, uint32_t pecstate);
<b>Function descriptions</b>	I2C PEC calculation on or off
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
i2c_periph	I2C peripheral
I2Cx	(x=0,1)
<b>Input parameter{in}</b>	
pecstate	on or off
I2C_PEC_ENABLE	PEC calculation on
I2C_PEC_DISABLE	PEC calculation off
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

---

```
/* Enable I2C PEC calculation */
```

```
i2c_pec_enable (I2C0, I2C_PEC_ENABLE);
```

### **i2c\_pec\_transfer\_enable**

The description of i2c\_pec\_transfer\_enable is shown as below:

**Table 3-204. Function i2c\_pec\_transfer\_enable**

<b>Function name</b>	i2c_pec_transfer_enable
<b>Function prototype</b>	void i2c_pec_transfer_enable(uint32_t i2c_periph, uint32_t pecpara);
<b>Function descriptions</b>	I2C whether to transfer PEC value
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
i2c_periph	I2C peripheral
I2Cx	(x=0,1)
<b>Input parameter{in}</b>	
pecpara	Transfer PEC or not
I2C_PECTRANS_ENA BLE	transfer PEC
I2C_PECTRANS_DISA BLE	not transfer PEC
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* I2C0 transfer PEC */
```

```
i2c_pec_transfer_enable (I2C0, I2C_PECTRANS_ENABLE);
```

### **i2c\_pec\_value\_get**

The description of i2c\_pec\_value\_get is shown as below:

**Table 3-205. Function i2c\_pec\_value\_get**

<b>Function name</b>	i2c_pec_value_get
<b>Function prototype</b>	uint8_t i2c_pec_value_get(uint32_t i2c_periph);
<b>Function descriptions</b>	get packet error checking value
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
i2c_periph	I2C peripheral

<i>I2Cx</i>	(x=0,1)
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
<i>uint8_t</i>	PEC value

Example:

```
/* I2C0 get packet error checking value */
```

```
uint8_t pec_value;
```

```
pec_value = i2c_pec_value_get (I2C0);
```

### **i2c\_smbus\_issue\_alert**

The description of i2c\_smbus\_issue\_alert is shown as below:

**Table 3-206. Function i2c\_smbus\_issue\_alert**

<b>Function name</b>	i2c_smbus_issue_alert
<b>Function prototype</b>	void i2c_smbus_issue_alert(uint32_t i2c_periph, uint32_t smbuspara);
<b>Function descriptions</b>	I2C issue alert through SMBA pin
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<i>i2c_periph</i>	I2C peripheral
<i>I2Cx</i>	(x=0,1)
<b>Input parameter{in}</b>	
<i>smbuspara</i>	issue alert through SMBA pin or not
<i>I2C_SALTSEND_ENA BLE</i>	issue alert through SMBA pin
<i>I2C_SALTSEND_DISA BLE</i>	not issue alert through SMBA pin
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* I2C0 issue alert through SMBA pin enable*/
```

```
i2c_smbus_issue_alert (I2C0, I2C_SALTSEND_ENABLE);
```

### **i2c\_smbus\_arp\_enable**

The description of i2c\_smbus\_arp\_enable is shown as below:

**Table 3-207. Function i2c\_smbus\_arp\_enable**

<b>Function name</b>	i2c_smbus_arp_enable
<b>Function prototype</b>	void i2c_smbus_arp_enable(uint32_t i2c_periph, uint32_t arpstate);
<b>Function descriptions</b>	enable or disable I2C ARP protocol in SMBus switch
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
i2c_periph	I2C peripheral
I2Cx	(x=0,1)
<b>Input parameter{in}</b>	
arpstate	ARP protocol in SMBus switch
I2C_ARP_ENABLE	enable ARP
I2C_ARP_DISABLE	disable ARP
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* enable I2C0 ARP protocol in SMBus switch */

i2c_smbus_arp_enable (I2C0, I2C_ARP_ENABLE);
```

### **i2c\_sam\_enable**

The description of i2c\_sam\_enable is shown as below:

**Table 3-208. Function i2c\_sam\_enable**

<b>Function name</b>	i2c_sam_enable
<b>Function prototype</b>	void i2c_sam_enable (uint32_t i2c_periph);
<b>Function descriptions</b>	enable SAM_V interface
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
i2c_periph	I2C peripheral
I2Cx	(x=0,1)
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* enable I2C0 SAM_V interface */

i2c_sam_enable (I2C0);
```

### **i2c\_sam\_disable**

The description of i2c\_sam\_disable is shown as below:

**Table 3-209. Function i2c\_sam\_disable**

<b>Function name</b>	i2c_sam_disable	
<b>Function prototype</b>	void i2c_sam_disable (uint32_t i2c_periph);	
<b>Function descriptions</b>	disable SAM_V interface	
<b>Precondition</b>	-	
<b>The called functions</b>	-	
<b>Input parameter{in}</b>		
<b>i2c_periph</b>	I2C peripheral	
<b>I2Cx</b>	(x=0,1)	
<b>Output parameter{out}</b>		
-	-	
<b>Return value</b>		
-	-	

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 3-210. Function i2c\_sam\_timeout\_enable**

<b>Function name</b>	i2c_sam_timeout_enable	
<b>Function prototype</b>	void i2c_sam_timeout_enable (uint32_t i2c_periph);	
<b>Function descriptions</b>	enable SAM_V interface timeout detect	
<b>Precondition</b>	-	
<b>The called functions</b>	-	
<b>Input parameter{in}</b>		
<b>i2c_periph</b>	I2C peripheral	
<b>I2Cx</b>	(x=0,1)	
<b>Output parameter{out}</b>		
-	-	
<b>Return value</b>		
-	-	

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 3-211. Function i2c\_sam\_timeout\_disable**

<b>Function name</b>	i2c_sam_timeout_disable	
<b>Function prototype</b>	void i2c_sam_timeout_disable (uint32_t i2c_periph);	
<b>Function descriptions</b>	disable SAM_V interface timeout detect	
<b>Precondition</b>	-	
<b>The called functions</b>	-	
<b>Input parameter{in}</b>		
<i>i2c_periph</i>	I2C peripheral	
<i>I2Cx</i>	(x=0,1)	
<b>Output parameter{out}</b>		
-	-	
<b>Return value</b>		
-	-	

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 3-212. Function i2c\_flag\_get**

<b>Function name</b>	i2c_flag_get	
<b>Function prototype</b>	FlagStatus i2c_flag_get(uint32_t i2c_periph, i2c_flag_enum flag)	
<b>Function descriptions</b>	check I2C flag is set or not	
<b>Precondition</b>	-	
<b>The called functions</b>	-	
<b>Input parameter{in}</b>		
<i>i2c_periph</i>	I2C peripheral	
<i>I2Cx</i>	(x=0,1)	
<b>Input parameter{in}</b>		
<i>flag</i>	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>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>	overrun 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_TR</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
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
<b>FlagStatus</b>	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 3-213. Function i2c\_flag\_clear**

<b>Function name</b>	i2c_flag_clear
<b>Function prototype</b>	void i2c_flag_clear(uint32_t i2c_periph, i2c_flag_enum flag)
<b>Function descriptions</b>	clear I2C flag
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	

<b>i2c_periph</b>	I2C peripheral
<i>I2Cx</i>	(x=0,1)
<b>Input parameter{in}</b>	
<b>flag</b>	flag type
<i>I2C_FLAG_SMBALT</i>	SMBus Alert status
<i>I2C_FLAG_SMBTO</i>	timeout signal in SMBus mode
<i>I2C_FLAG_PECERR</i>	PEC error when receiving data
<i>I2C_FLAG_OUERR</i>	over-run or under-run situation occurs in slave mode
<i>I2C_FLAG_AERR</i>	acknowledge error
<i>I2C_FLAG_LOSTARB</i>	arbitration lost in master mode
<i>I2C_FLAG_BERR</i>	a bus error
<i>I2C_FLAG_ADDSEND</i>	cleared by reading I2C_STAT0 and reading I2C_STAT1
<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
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

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 3-214. Function i2c\_interrupt\_enable**

<b>Function name</b>	i2c_interrupt_enable
<b>Function prototype</b>	void i2c_interrupt_enable(uint32_t i2c_periph, i2c_interrupt_enum interrupt);
<b>Function descriptions</b>	enable I2C interrupt
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>i2c_periph</b>	I2C peripheral
<i>I2Cx</i>	(x=0,1)
<b>Input parameter{in}</b>	
<b>interrupt</b>	interrupt type
<i>I2C_INT_ERR</i>	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>
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* enable I2C0 event interrupt */

i2c_interrupt_enable (I2C0, I2C_INT_EV);
```

### **i2c\_interrupt\_disable**

The description of i2c\_interrupt\_disable is shown as below:

**Table 3-215. Function i2c\_interrupt\_disable**

<b>Function name</b>	i2c_interrupt_disable
<b>Function prototype</b>	void i2c_interrupt_disable(uint32_t i2c_periph, i2c_interrupt_enum interrupt);
<b>Function descriptions</b>	disable I2C interrupt
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<i>i2c_periph</i>	I2C peripheral
<i>I2Cx</i>	(x=0,1)
<b>Input parameter{in}</b>	
<b>interrupt</b>	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>
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

---

```
/* disable I2C0 event interrupt */

i2c_interrupt_disable (I2C0, I2C_INT_EV);
```

### **i2c\_interrupt\_flag\_get**

The description of i2c\_interrupt\_flag\_get is shown as below:

**Table 3-216. Function i2c\_interrupt\_flag\_get**

<b>Function name</b>	i2c_interrupt_flag_get
<b>Function prototype</b>	FlagStatus i2c_interrupt_flag_get(uint32_t i2c_periph, i2c_interrupt_flag_enum int_flag)
<b>Function descriptions</b>	check I2C interrupt flag
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<i>i2c_periph</i>	I2C peripheral
<i>I2Cx</i>	(x=0,1)
<b>Input parameter{in}</b>	
<i>int_flag</i>	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_BTCS</i>	byte transmission finishes
<i>I2C_INT_FLAG_ADD1_OSEND</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_LOST_ARB</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_PECERR_R</i>	PEC error when receiving data interrupt flag
<i>I2C_INT_FLAG_SMBTO</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
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
<b>FlagStatus</b>	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 3-217. Function i2c\_interrupt\_flag\_clear**

<b>Function name</b>	i2c_interrupt_flag_clear
<b>Function prototype</b>	void i2c_interrupt_flag_clear(uint32_t i2c_periph, i2c_interrupt_flag_enum int_flag);
<b>Function descriptions</b>	clear I2C interrupt flag
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<i>i2c_periph</i>	I2C peripheral
<i>I2Cx</i>	(x=0,1)
<b>Input parameter{in}</b>	
<b>int_flag</b>	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_LOST</i> <i>ARB</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_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
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* clear the acknowledge error interrupt flag */

i2c_interrupt_flag_clear (I2C0, I2C_INT_FLAG_AERR);
```

## 3.12. MISC

MISC is a software package that provide the interfaces for NVIC and SysTick. The NVIC and SysTick registers are listed in chapter [3.12.1](#), the MISC firmware functions are introduced in chapter [3.12.2](#).

### 3.12.1. Descriptions of Peripheral registers

**Table 3-218. NVIC Registers**

Registers	Descriptions
ISER <sup>(1)</sup>	Interrupt Set Enable Register
ICER <sup>(1)</sup>	Interrupt Clear Enable Register
ISPR <sup>(1)</sup>	Interrupt Set Pending Register
ICPR <sup>(1)</sup>	Interrupt Clear Pending Register
IABR <sup>(1)</sup>	Interrupt Active bit Register
ITNS <sup>(1)</sup>	Interrupt Non-Secure State Register
IPR <sup>(1)</sup>	Interrupt Priority Register
CPUID <sup>(2)</sup>	CPUID Base Register
ICSR <sup>(2)</sup>	Interrupt Control and State Register
VTOR <sup>(2)</sup>	Vector Table Offset Register
AIRCR <sup>(2)</sup>	Application Interrupt and Reset Control Register
SCR <sup>(2)</sup>	System Control Register
CCR <sup>(2)</sup>	Configuration Control Register
SHPR <sup>(2)</sup>	System Handlers Priority Registers
SHCSR <sup>(2)</sup>	System Handler Control and State Register

1. refer to the structure NVIC\_Type, is defined in the core\_cm23.h file

2. refer to the structure SCB\_Type, is defined in the core\_cm23.h file

**Table 3-219. SysTick Registers**

<b>Registers</b>	<b>Descriptions</b>
CTRL <sup>(1)</sup>	SysTick Control and Status Register
LOAD <sup>(1)</sup>	SysTick Reload Value Register
VAL <sup>(1)</sup>	SysTick Current Value Register
CALIB <sup>(1)</sup>	SysTick Calibration Register

1. refer to the structure SysTick\_Type, is defined in the core\_cm23.h file

### 3.12.2. Descriptions of Peripheral functions

#### Enum IRQn\_Type

**Table 3-220. IRQn\_Type**

<b>Member name</b>	<b>Function description</b>
WWDGT_IRQn	window watchDog timer interrupt
LVD_IRQn	LVD through EXTI line detect interrupt
RTC_IRQn	RTC through EXTI line interrupt
FMC_IRQn	FMC interrupt
RCU_IRQn	RCU interrupt
EXTI0_1_IRQn	EXTI line 0 and 1 interrupts
EXTI2_3_IRQn	EXTI line 2 and 3 interrupts
EXTI4_15_IRQn	EXTI line 4 and 15 interrupts
DMA_Channel0_IRQn	DMA channel0 interrupt
DMA_Channel1_2_IRQn	DMA channel 1 and channel 2 interrupts
DMA_Channel3_4_IRQn	DMA channel 3 and channel 4 interrupts
ADC_CMP_IRQn	ADC, CMP interrupts
TIMER0_BRK_UP_TRG_COM_IRQn	TIMER0 break, update, trigger and commutation interrupts
TIMER0_Channel_IRQn	TIMER0 channel capture compare interrupts
TIMER2_IRQn	TIMER2 interrupt
TIMER5_IRQn	TIMER5 interrupt
TIMER13_IRQn	TIMER13 interrupt
TIMER14_IRQn	TIMER14 interrupt
TIMER15_IRQn	TIMER15 interrupt
TIMER16_IRQn	TIMER16 interrupt
I2C0_EV_IRQn	I2C0 event interrupt
I2C1_EV_IRQn	I2C1 event interrupt
SPI0_IRQn	SPI0 interrupt
SPI1_IRQn	SPI1 interrupt
USART0_IRQn	USART0 interrupt
USART1_IRQn	USART1 interrupt
I2C0_ER_IRQn	I2C0 error interrupt
I2C1_ER_IRQn	I2C1 error interrupt

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

**Table 3-221. MISC firmware function**

Function name	Function description
nvic_irq_enable	enable NVIC interrupt request
nvic_irq_disable	disable NVIC interrupt request
nvic_system_reset	initiates a system reset request to reset the MCU
nvic_vector_table_set	set the NVIC vector table address
system_lowpower_set	set the state of the low power mode
system_lowpower_reset	reset the state of the low power mode
systick_clksource_set	set the systick clock source

### **nvic\_irq\_enable**

The description of nvcic\_irq\_enable is shown as below:

**Table 3-222. Function nvcic\_irq\_enable**

<b>Function name</b>	nvcic_irq_enable
<b>Function prototype</b>	void nvcic_irq_enable(uint8_t nvcic_irq, uint8_t nvcic_irq_pre_priority);
<b>Function descriptions</b>	enable NVIC request, configure the priority of interrupt
<b>Precondition</b>	-
<b>The called functions</b>	NVIC_SetPriority、NVIC_EnableIRQ
<b>Input parameter{in}</b>	
nvcic_irq	NVIC interrupt, refer to enum <a href="#">Table 3-220. IRQn_Type</a>
<b>Input parameter{in}</b>	
nvcic_irq_pre_priority	the pre-emption priority needed to set (0~3)
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* enable window watchDog timer interrupt , priority is 1 */
nvcic_irq_enable(WWDGT_IRQn, 1);
```

### **nvcic\_irq\_disable**

The description of nvcic\_irq\_disable is shown as below:

**Table 3-223. Function nvcic\_irq\_disable**

<b>Function name</b>	nvcic_irq_disable
<b>Function prototype</b>	void nvcic_irq_disable(uint8_t nvcic_irq);
<b>Function descriptions</b>	disable NVIC request
<b>Precondition</b>	-
<b>The called functions</b>	-

Input parameter{in}	
<b>nvic_irq</b>	NVIC interrupt, refer to enum <a href="#">Table 3-220. IRQn_Type</a>
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* disable window watchDog timer interrupt */
nvic_irq_disable(WWDGT_IRQn);
```

### **nvic\_system\_reset**

The description of nvic\_system\_reset is shown as below:

**Table 3-224. Function nvic\_system\_reset**

<b>Function name</b>	nvic_system_reset
<b>Function prototype</b>	void nvic_system_reset (void);
<b>Function descriptions</b>	initiates a system reset request to reset the MCU
<b>Precondition</b>	-
<b>The called functions</b>	NVIC_SystemReset
Input parameter{in}	
-	-
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* reset the MCU*/
nvic_system_reset();
```

### **nvic\_vector\_table\_set**

The description of nvic\_vector\_table\_set is shown as below:

**Table 3-225. Function nvic\_vector\_table\_set**

<b>Function name</b>	nvic_vector_table_set
<b>Function prototype</b>	void nvic_vector_table_set(uint32_t nvic_vict_tab, uint32_t offset);
<b>Function descriptions</b>	set the NVIC vector table address
<b>Precondition</b>	-
<b>The called functions</b>	-
Input parameter{in}	
<b>nvic_vict_tab</b>	the RAM or FLASH base address

<b>NVIC_VECTTAB_RAM</b>	RAM base address
<b>NVIC_VECTTAB_FLASH</b>	Flash base address
<b>Input parameter{in}</b>	
<b>offset</b>	Vector Table offset (vector table start address= base address+offset)
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

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 3-226. Function system\_lowpower\_set**

<b>Function name</b>	system_lowpower_set
<b>Function prototype</b>	void system_lowpower_set(uint8_t lowpower_mode);
<b>Function descriptions</b>	the state of the low power mode management
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>lowpower_mode</b>	the low power mode state
<b>SCB_LPM_SLEEP_EXIT_ISR</b>	if chose this para, the system always enter low power mode by exiting from ISR
<b>SCB_LPM_DEEPSLEEP_P</b>	if chose this para, the system will enter the DEEPSLEEP mode
<b>SCB_LPM_WAKE_BY_ALL_INTERRUPTS</b>	if chose this para, the lowpower mode can be woke up by all the enable and disable interrupts
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

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 3-227. Function system\_lowpower\_reset**

<b>Function name</b>	system_lowpower_reset
<b>Function prototype</b>	void system_lowpower_reset(uint8_t lowpower_mode);
<b>Function descriptions</b>	the state of the low power mode management
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>lowpower_mode</b>	the low power mode state
SCB_LPM_SLEEP_EXIT_ISR	if chose this para, the system will exit low power mode by exiting from ISR
SCB_LPM_DEEPSLEEP_P	if chose this para, the system will enter the SLEEP mode
SCB_LPM_WAKE_BY_ALL_INT	if chose this para, the lowpower mode only can be woke up by the enable interrupts
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

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 3-228. Function systick\_clksource\_set**

<b>Function name</b>	systick_clksource_set
<b>Function prototype</b>	void systick_clksource_set(uint32_t systick_clksource);
<b>Function descriptions</b>	set the systick clock source
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>systick_clksource</b>	the systick clock source needed to choose
SYSTICK_CLKSOURC_E_HCLK	systick clock source is from HCLK
SYSTICK_CLKSOURC_E_HCLK_DIV8	systick clock source is from HCLK/8
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* systick clock source is HCLK/8 */

systick_clksource_set(SYSTICK_CLKSOURCE_HCLK_DIV8);
```

## 3.13. 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 [3.13.1](#), the PMU firmware functions are introduced in chapter [3.13.2](#).

### 3.13.1. Descriptions of Peripheral registers

PMU registers are listed in the table shown as below:

**Table 3-229. PMU Registers**

Registers	Descriptions
PMU_CTL	PMU control register
PMU_CS	PMU control and status register

### 3.13.2. Descriptions of Peripheral functions

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

**Table 3-230. PMU firmware function**

Function name	Function description
pmu_deinit	reset PMU register
pmu_lvd_select	select low voltage detector threshold
pmu_ldo_output_select	select LDO output voltage
pmu_lvd_disable	disable PMU lvd
pmu_to_sleepmode	PMU work in sleep mode
pmu_to_deepsleepmode	PMU work at deepsleep mode
pmu_to_standbymode	pmu work at standby mode
pmu_wakeup_pin_enable	enable PMU wakeup pin
pmu_wakeup_pin_disable	disable PMU wakeup pin
pmu_backup_write_enable	enable backup domain write
pmu_backup_write_disable	disable backup domain write
pmu_flag_clear	clear flag bit
pmu_flag_get	get flag state

#### pmu\_deinit

The description of pmu\_deinit is shown as below:

**Table 3-231. Function pmu\_deinit**

<b>Function name</b>	pmu_deinit
<b>Function prototype</b>	void pmu_deinit(void);
<b>Function descriptions</b>	reset PMU register
<b>Precondition</b>	-
<b>The called functions</b>	rcu_periph_reset_enable / rcu_periph_reset_disable
<b>Input parameter{in}</b>	
-	-
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* reset PMU */
pmu_deinit();
```

### **pmu\_lvd\_select**

The description of pmu\_lvd\_select is shown as below:

**Table 3-232. Function pmu\_lvd\_select**

<b>Function name</b>	pmu_lvd_select
<b>Function prototype</b>	void pmu_lvd_select(uint32_t lvdt_n);
<b>Function descriptions</b>	select low voltage detector threshold
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<i>lvdt_n</i>	voltage threshold value
<i>PMU_LVDT_0</i>	voltage threshold is 2.1V
<i>PMU_LVDT_1</i>	voltage threshold is 2.3V
<i>PMU_LVDT_2</i>	voltage threshold is 2.4V
<i>PMU_LVDT_3</i>	voltage threshold is 2.6V
<i>PMU_LVDT_4</i>	voltage threshold is 2.7V
<i>PMU_LVDT_5</i>	voltage threshold is 2.9V
<i>PMU_LVDT_6</i>	voltage threshold is 3.0V
<i>PMU_LVDT_7</i>	voltage threshold is 3.1V
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

---

```
/* select low voltage detector threshold as 3.1V */
```

```
pmu_lvd_select (PMU_LVDT_7);
```

### **pmu\_ldo\_output\_select**

The description of pmu\_ldo\_output\_select is shown as below:

**Table 3-233. Function pmu\_ldo\_output\_select**

<b>Function name</b>	pmu_ldo_output_select
<b>Function prototype</b>	void pmu_ldo_output_select(uint32_t ldo_output);
<b>Function descriptions</b>	select LDO output voltage
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>ldo_output</b>	output voltage mode
<i>PMU_LDOVS_LOW</i>	LDO output voltage low mode
<i>PMU_LDOVS_HIGH</i>	LDO output voltage high mode
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* select output low voltage mode */
```

```
pmu_ldo_output_select (PMU_LDOVS_LOW);
```

### **pmu\_lvd\_disable**

The description of pmu\_lvd\_disable is shown as below:

**Table 3-234. Function pmu\_lvd\_disable**

<b>Function name</b>	pmu_lvd_disable
<b>Function prototype</b>	void pmu_lvd_disable (void);
<b>Function descriptions</b>	disable PMU lvd
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
-	-
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

---

```
/* disable PMU lvd */
```

```
pmu_lvd_disable();
```

### **pmu\_to\_sleepmode**

The description of pmu\_to\_sleepmode is shown as below:

**Table 3-235. Function pmu\_to\_sleepmode**

<b>Function name</b>	pmu_to_sleepmode
<b>Function prototype</b>	void pmu_to_sleepmode(uint8_t sleepmodecmd);
<b>Function descriptions</b>	PMU work at sleep mode
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>sleepmodecmd</b>	command to enter sleep mode
<i>WFI_CMD</i>	use WFI command
<i>WFE_CMD</i>	use WFE command
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* PMU work at sleep mode */

pmu_to_sleepmode (WFI_CMD);
```

### **pmu\_to\_deepsleepmode**

The description of pmu\_to\_deepsleepmode is shown as below:

**Table 3-236. Function pmu\_to\_deepsleepmode**

<b>Function name</b>	pmu_to_deepsleepmode
<b>Function prototype</b>	void pmu_to_deepsleepmode(uint32_t ldo,uint8_t deepsleepmodecmd);
<b>Function descriptions</b>	PMU work at deepsleep mode
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>ldo</b>	ldo work mode
<i>PMU_LDO_NORMAL</i>	LDO operates normally when pmu enter deepsleep mode
<i>PMU_LDO_LOWPOW_ER</i>	LDO work at low power mode when pmu enter deepsleep mode
<b>Input parameter{in}</b>	
<b>deepsleepmodecmd</b>	command to enter deepsleep mode

<i>WFI_CMD</i>	use WFI command
<i>WFE_CMD</i>	use WFE command
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* PMU work at deepsleep mode */

pmu_to_deepsleepmode (PMU_LDO_NORMAL, WFI_CMD);
```

### **pmu\_to\_standbymode**

The description of pmu\_to\_standbymode is shown as below:

**Table 3-237. Function pmu\_to\_standbymode**

<b>Function name</b>	pmu_to_standbymode
<b>Function prototype</b>	void pmu_to_standbymode(uint8_t standbymodecmd);
<b>Function descriptions</b>	pmu work at standby mode
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>standbymodecmd</b>	command to enter standby mode
<i>WFI_CMD</i>	use WFI command
<i>WFE_CMD</i>	use WFE command
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* PMU work at standby mode */

pmu_to_standby (WFI_CMD);
```

### **pmu\_wakeup\_pin\_enable**

The description of pmu\_wakeup\_pin\_enable is shown as below:

**Table 3-238. Function pmu\_wakeup\_pin\_enable**

<b>Function name</b>	pmu_wakeup_pin_enable
<b>Function prototype</b>	void pmu_wakeup_pin_enable(uint32_t wakeup_pin);
<b>Function descriptions</b>	enable wakeup pin
<b>Precondition</b>	-

The called functions		-
Input parameter{in}		
wakeup_pin		Wakeup pin
PMU_WAKEUP_PIN0		WKUP Pin 0 (PA0)
PMU_WAKEUP_PIN1		WKUP Pin 1 (PC13)
PMU_WAKEUP_PIN5		WKUP Pin 5 (PB5)
PMU_WAKEUP_PIN6		WKUP Pin 6 (PB15)
Output parameter{out}		
-		-
Return value		
-		-

Example:

```
/* enable wakeup pin6 */
pmu_wakeup_pin_enable (PMU_WAKEUP_PIN6);
```

### **pmu\_wakeup\_pin\_disable**

The description of pmu\_wakeup\_pin\_disable is shown as below:

**Table 3-239. Function pmu\_wakeup\_pin\_disable**

Function name		pmu_wakeup_pin_disable
Function prototype		void pmu_wakeup_pin_disable(uint32_t wakeup_pin);
Function descriptions		disable wakeup pin
Precondition		-
The called functions		-
Input parameter{in}		
wakeup_pin		Wakeup pin
PMU_WAKEUP_PIN0		WKUP Pin 0 (PA0)
PMU_WAKEUP_PIN1		WKUP Pin 1 (PC13)
PMU_WAKEUP_PIN5		WKUP Pin 5 (PB5)
PMU_WAKEUP_PIN6		WKUP Pin 6 (PB15)
Output parameter{out}		
-		-
Return value		
-		-

Example:

```
/* disable wakeup pin6 */
pmu_wakeup_pin_disable (PMU_WAKEUP_PIN6);
```

### **pmu\_backup\_write\_enable**

The description of pmu\_backup\_write\_enable is shown as below:

**Table 3-240. Function pmu\_backup\_write\_enable**

<b>Function name</b>	pmu_backup_write_enable
<b>Function prototype</b>	void pmu_backup_write_enable (void);
<b>Function descriptions</b>	enable backup domain write
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
-	-
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

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 3-241. Function pmu\_backup\_write\_disable**

<b>Function name</b>	pmu_backup_write_disable
<b>Function prototype</b>	void pmu_backup_write_disable (void);
<b>Function descriptions</b>	disable backup domain write
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
-	-
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* disable backup domain write */
pmu_backup_write_disable();
```

### **pmu\_flag\_clear**

The description of pmu\_flag\_clear is shown as below:

**Table 3-242. Function pmu\_flag\_clear**

<b>Function name</b>	pmu_flag_clear
<b>Function prototype</b>	void pmu_flag_clear(uint32_t flag_clear);
<b>Function descriptions</b>	clear flag bit
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>flag_clear</b>	flag
<b>PMU_FLAG_RESET_WAKEUP</b>	reset wakeup flag
<b>PMU_FLAG_RESET_STANDBY</b>	reset standby flag
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* clear flag bit */
pmu_flag_clear (PMU_FLAG_RESET_WAKEUP);
```

### **pmu\_flag\_get**

The description of pmu\_flag\_get is shown as below:

**Table 3-243. Function pmu\_flag\_get**

<b>Function name</b>	pmu_flag_get
<b>Function prototype</b>	FlagStatus pmu_flag_get(uint32_t flag);
<b>Function descriptions</b>	get flag state
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>flag</b>	flag
<b>PMU_FLAG_WAKEUP</b>	wakeup flag
<b>PMU_FLAG_STANDBY</b>	standby flag
<b>PMU_FLAG_LVD</b>	lvd flag
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
<b>FlagStatus</b>	SET or RESET

Example:

```
/* get flag state */

FlagStatus status;

status = pmu_flag_get (PMU_FLAG_WAKEUP);
```

## 3.14. 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 [3.14.1](#), the RCU firmware functions are introduced in chapter [3.14.2](#).

### 3.14.1. Descriptions of Peripheral registers

**Table 3-244. RCU Registers**

Registers	Descriptions
RCU_CTL0	Control register 0
RCU_CFG0	Clock configuration register 0
RCU_INT	Clock interrupt register
RCU_APB2RST	APB2 reset register
RCU_APB1RST	APB1 reset register
RCU_AHBEN	AHB enable register
RCU_APB2EN	APB2 enable register
RCU_APB1EN	APB1 enable register
RCU_BDCTL	Backup domain control register
RCU_RSTSCK	Reset source/clock register
RCU_AHBRST	AHB reset register
RCU_CFG1	Clock configuration register 1
RCU_CFG2	Clock configuration register 2
RCU_CTL1	Control register 1
RCU_VKEY	Unlock voltage register
RCU_DSV	Deep-sleep mode voltage register

### 3.14.2. Descriptions of Peripheral functions

**Table 3-245. 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

<b>Function name</b>	<b>Function description</b>
rcu_periph_clock_sleep_disable	disable the peripherals clock when in sleep mode
rcu_periph_reset_enable	enable the peripherals reset
rcu_periph_reset_disable	disable the peripheral reset
rcu_bkp_reset_enable	enable the BKP domain reset
rcu_bkp_reset_disable	disable the BKP domain 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_adc_clock_config	configure the ADC clock source and prescaler selection
rcu_ckout_config	configure the CK_OUT clock source and divider
rcu_pll_config	configure the main PLL clock
rcu_usart_clock_config	configure the usart clock
rcu_rtc_clock_config	configure the RTC clock source selection
rcu_hxtal_prediv_config	configure the HXTAL divider used as input of PLL
rcu_lxtal_drive_capability_config	configure the LXTAL drive capability
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
rcu_osc_stab_wait	wait for oscillator stabilization flags is SET or oscillator startup is timeout
rcu_osc_on	turn on the oscillator
rcu_osc_off	turn off the oscillator
rcu_osc_bypass_mode_enable	enable the oscillator bypass mode
rcu_osc_bypass_mode_disable	disable the oscillator bypass mode
rcu_hxtal_clock_monitor_enable	enable the HXTAL clock monitor
rcu_hxtal_clock_monitor_disable	disable the HXTAL clock monitor
rcu_irc8m_adjust_value_set	set the IRC8M adjust value
rcu_irc28m_adjust_value_set	set the IRC28M adjust value
rcu_voltage_key_unlock	unlock Deep-sleep mode voltage register
rcu_deepsleep_voltage_set	set the deep-sleep mode voltage value
rcu_clock_freq_get	get the system clock, bus clock frequency

### **rcu\_deinit**

The description of rcu\_deinit is shown as below:

**Table 3-246. Function rcu\_deinit**

<b>Function name</b>	rcu_deinit
<b>Function prototype</b>	void rcu_deinit(void);
<b>Function descriptions</b>	deinitialize the RCU, reset the value of all RCU registers into initial values
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
-	-
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* reset RCU */
rcu_deinit();
```

### **rcu\_periph\_clock\_enable**

The description of rcu\_periph\_clock\_enable is shown as below:

**Table 3-247. Function rcu\_periph\_clock\_enable**

<b>Function name</b>	rcu_periph_clock_enable
<b>Function prototype</b>	void rcu_periph_clock_enable(rcu_periph_enum periph);
<b>Function descriptions</b>	enable the peripherals clock
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>periph</b>	RCU peripherals, refer to rcu_periph_enum
<i>RCU_GPIOx</i>	GPIO ports clock (x=A,B,C,F)
<i>RCU_DMA</i>	DMA clock
<i>RCU_CRC</i>	CRC clock
<i>RCU_CFGCMP</i>	CFGCMP clock
<i>RCU_ADC</i>	ADC clock
<i>RCU_TIMERx</i>	TIMERx clock(x=0,2,5,13,14,15,16)
<i>RCU_SPIx</i>	SPIx clock (x=0,1)
<i>RCU_USARTx</i>	USARTx clock (x=0,1)
<i>RCU_WWDGT</i>	WWDGT clock
<i>RCU_I2Cx</i>	I2Cx clock (x=0,1)
<i>RCU_PMU</i>	PMU clock
<i>RCU_RTC</i>	RTC clock
<i>RCU_DBGMCU</i>	DBGMCU clock
<b>Output parameter{out}</b>	

-	-
<b>Return value</b>	
-	-

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 3-248. Function `rcu_periph_clock_disable`**

<b>Function name</b>	rcu_periph_clock_disable
<b>Function prototype</b>	void rcu_periph_clock_disable(rcu_periph_enum periph);
<b>Function descriptions</b>	disable the peripherals clock
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>periph</b>	RCU peripherals, refer to <code>rcu_periph_enum</code>
<i>RCU_GPIOx</i>	GPIO ports clock (x=A,B,C,F)
<i>RCU_DMA</i>	DMA clock
<i>RCU_CRC</i>	CRC clock
<i>RCU_CFGCMP</i>	CFGCMP clock
<i>RCU_ADC</i>	ADC clock
<i>RCU_TIMERx</i>	TIMERx clock(x=0,2,5,13,14,15,16)
<i>RCU_SPIx</i>	SPIx clock (x=0,1)
<i>RCU_USARTx</i>	USARTx clock (x=0,1)
<i>RCU_WWDGT</i>	WWDGT clock
<i>RCU_I2Cx</i>	I2Cx clock (x=0,1)
<i>RCU_PMU</i>	PMU clock
<i>RCU_RTC</i>	RTC clock
<i>RCU_DBGMCU</i>	DBGMCU clock
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

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 3-249. Function `rcu_periph_clock_sleep_enable`**

<b>Function name</b>	rcu_periph_clock_sleep_enable
<b>Function prototype</b>	void <code>rcu_periph_clock_sleep_enable(rcu_periph_sleep_enum periph);</code>
<b>Function descriptions</b>	enable the peripherals clock when in sleep mode
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>periph</b>	RCU peripherals, refer to <code>rcu_periph_sleep_enum</code>
<i>RCU_FMC_SLP</i>	FMC clock
<i>RCU_SRAM_SLP</i>	SRAM clock
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* enable the FMC clock when in sleep mode */
rcu_periph_clock_sleep_enable(RCU_FMC_SLP);
```

### **rcu\_periph\_clock\_sleep\_disable**

The description of `rcu_periph_clock_sleep_disable` is shown as below:

**Table 3-250. Function `rcu_periph_clock_sleep_disable`**

<b>Function name</b>	rcu_periph_clock_sleep_disable
<b>Function prototype</b>	void <code>rcu_periph_clock_sleep_disable(rcu_periph_sleep_enum periph);</code>
<b>Function descriptions</b>	disable the peripherals clock when in sleep mode
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>periph</b>	RCU peripherals, refer to <code>rcu_periph_sleep_enum</code>
<i>RCU_FMC_SLP</i>	FMC clock
<i>RCU_SRAM_SLP</i>	SRAM clock
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* disable the FMC clock when in sleep mode */
```

---

```
rcu_periph_clock_sleep_disable(RCU_FMC_SLP);
```

### **rcu\_periph\_reset\_enable**

The description of `rcu_periph_reset_enable` is shown as below:

**Table 3-251. Function `rcu_periph_reset_enable`**

<b>Function name</b>	rcu_periph_reset_enable
<b>Function prototype</b>	void rcu_periph_reset_enable(rcu_periph_reset_enum periph_reset);
<b>Function descriptions</b>	enable the peripherals reset
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<code>periph_reset</code>	RCU peripherals reset, refer to <code>rcu_periph_reset_enum</code>
<code>RCU_GPIOxRST</code>	reset GPIO ports clock (x=A,B,C,F)
<code>RCU_CFGCMRST</code>	reset CFGCMP clock
<code>RCU_ADCRST</code>	reset ADC clock
<code>RCU_TIMERxRST</code>	reset TIMERx clock (x=0,2,5,13,14,15,16)
<code>RCU_SPIxRST</code>	reset SPIx clock (x=0,1)
<code>RCU_USARTxRST</code>	reset USARTx clock (x=0,1)
<code>RCU_WWDGTRST</code>	reset WWDGT clock
<code>RCU_I2CxRST</code>	reset I2Cx clock (x=0,1)
<code>RCU_PMURST</code>	reset PMU clock
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

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 3-252. Function `rcu_periph_reset_disable`**

<b>Function name</b>	rcu_periph_reset_disable
<b>Function prototype</b>	void rcu_periph_reset_disable(rcu_periph_reset_enum periph_reset);
<b>Function descriptions</b>	disable the peripheral reset
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<code>periph_reset</code>	RCU peripherals reset, refer to <code>rcu_periph_reset_enum</code>

<i>RCU_GPIOxRST</i>	disable reset GPIO ports clock (x=A,B,C,F)
<i>RCU_CFGCMRST</i>	disable reset CFGCMP clock
<i>RCU_ADCRST</i>	disable reset ADC clock
<i>RCU_TIMERxRST</i>	disable reset TIMERx clock (x=0,2,5,13,14,15,16)
<i>RCU_SPIxRST</i>	disable reset SPIx clock (x=0,1)
<i>RCU_USARTxRST</i>	disable reset USARTx clock (x=0,1)
<i>RCU_WWDGTRST</i>	disable reset WWDGT clock
<i>RCU_I2CxRST</i>	disable reset I2Cx clock (x=0,1)
<i>RCU_PMURST</i>	disable reset PMU clock
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

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 3-253. Function `rcu_bkp_reset_enable`**

<b>Function name</b>	<code>rcu_bkp_reset_enable</code>
<b>Function prototype</b>	<code>void rcu_bkp_reset_enable(void);</code>
<b>Function descriptions</b>	enable the BKP domain reset
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
-	-
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

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 3-254. Function rcu\_bkp\_reset\_disable**

<b>Function name</b>	rcu_bkp_reset_disable
<b>Function prototype</b>	void rcu_bkp_reset_disable(void);
<b>Function descriptions</b>	disable the BKP domain reset
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
-	-
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

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 3-255. Function rcu\_system\_clock\_source\_config**

<b>Function name</b>	rcu_system_clock_source_config
<b>Function prototype</b>	void rcu_system_clock_source_config(uint32_t ck_sys);
<b>Function descriptions</b>	configure the system clock source
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>ck_sys</b>	system clock source select
<i>RCU_CKSYSRC_IRC_8M</i>	select CK_IRC8M as the CK_SYS source
<i>RCU_CKSYSRC_HXTAL</i>	select CK_HXTAL as the CK_SYS source
<i>RCU_CKSYSRC_PLL</i>	select CK_PLL as the CK_SYS source
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* configure the CK_HXTAL as the CK_SYS source */
rcu_system_clock_source_config(RCU_CKSYSRC_HXTAL);
```

### **rcu\_system\_clock\_source\_get**

The description of `rcu_system_clock_source_get` is shown as below:

**Table 3-256. Function `rcu_system_clock_source_get`**

<b>Function name</b>	rcu_system_clock_source_get
<b>Function prototype</b>	uint32_t rcu_system_clock_source_get(void);
<b>Function descriptions</b>	get the system clock source
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
-	-
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
<b>uint32_t</b>	RCU_SCSS_IRC8M/RCU_SCSS_HXTAL/RCU_SCSS_PLL

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 3-257. Function `rcu_ahb_clock_config`**

<b>Function name</b>	rcu_ahb_clock_config
<b>Function prototype</b>	void rcu_ahb_clock_config(uint32_t ck_ahb);
<b>Function descriptions</b>	configure the AHB clock prescaler selection
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>ck_ahb</b>	AHB clock prescaler selection
<b>RCU_AHB_CKSYS_DI Vx</b>	select CK_SYS / x, (x=1, 2, 4, 8, 16, 64, 128, 256, 512)
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

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 3-258. Function rcu\_apb1\_clock\_config**

<b>Function name</b>	rcu_apb1_clock_config
<b>Function prototype</b>	void rcu_apb1_clock_config(uint32_t ck_apb1);
<b>Function descriptions</b>	configure the APB1 clock prescaler selection
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>ck_apb1</b>	APB1 clock prescaler selection
<i>RCU_APB1_CKAHB_D /Vx</i>	select (CK_AHB / x) as CK_APB1 (x=1,2,4,8,16)
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

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 3-259. Function rcu\_apb2\_clock\_config**

<b>Function name</b>	rcu_apb2_clock_config
<b>Function prototype</b>	void rcu_apb2_clock_config(uint32_t ck_apb2);
<b>Function descriptions</b>	configure the APB2 clock prescaler selection
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>ck_apb2</b>	APB2 clock prescaler selection
<i>RCU_APB2_CKAHB_D /Vx</i>	select (CK_AHB / x) as CK_APB2 clock (x=1,2,4,8,16)
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

---

```
/* configure CK_AHB/8 as CK_APB2 */

rcu_apb2_clock_config(RCU_APB2_CKAHB_DIV8);
```

### **rcu\_adc\_clock\_config**

The description of `rcu_adc_clock_config` is shown as below:

**Table 3-260. Function `rcu_adc_clock_config`**

<b>Function name</b>	<code>rcu_adc_clock_config</code>
<b>Function prototype</b>	<code>void rcu_adc_clock_config(rcu_adc_clock_enum ck_adc);</code>
<b>Function descriptions</b>	configure the ADC clock prescaler selection
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<code>ck_adc</code>	ADC clock prescaler selection, refer to <code>rcu_adc_clock_enum</code>
<code>RCU_ADCCK_IRC28M_DIV2</code>	select CK_IRC28M/2 as CK_ADC
<code>RCU_ADCCK_IRC28M</code>	select CK_IRC28M as CK_ADC
<code>RCU_ADCCK_AHB_DIVx</code>	select (CK_AHB / x) as CK_ADC(x=3,5,7,9)
<code>RCU_ADCCK_APB2_DIVx</code>	select (CK_APB2 / x) as CK_ADC(x=2,4,6,8)
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* configure the ADC prescaler factor */

rcu_adc_clock_config(RCU_ADCCK_IRC28M);
```

### **rcu\_ckout\_config**

The description of `rcu_ckout_config` is shown as below:

**Table 3-261. Function `rcu_ckout_config`**

<b>Function name</b>	<code>rcu_ckout_config</code>
<b>Function prototype</b>	<code>void rcu_ckout_config(uint32_t ckout_src, uint32_t ckout_div);</code>
<b>Function descriptions</b>	configure the CK_OUT clock source and division factor
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<code>ckout_src</code>	CK_OUT clock source selection

<i>RCU_CKOUTSRC_NO NE</i>	no clock selected
<i>RCU_CKOUTSRC_IRC 28M</i>	select high speed 28M internal oscillator clock
<i>RCU_CKOUTSRC_IRC 40K</i>	select high speed 40K internal oscillator clock
<i>RCU_CKOUTSRC_LX TAL</i>	select LXTAL clock
<i>RCU_CKOUTSRC_CK SYS</i>	select system clock CK_SYS
<i>RCU_CKOUTSRC_IRC 8M</i>	select high speed 8M internal oscillator clock
<i>RCU_CKOUTSRC_HX TAL</i>	select HXTAL clock
<i>RCU_CKOUTSRC_CK PLL_DIV1</i>	select CK_PLL clock
<i>RCU_CKOUTSRC_CK PLL_DIV2</i>	Select (CK_PLL / 2) clock
<b>Input parameter{in}</b>	
<i>ckout_div</i>	CK_OUT divider
<i>RCU_CKOUT_DIVx</i>	CK_OUT is divided by x(x=1,2,4,8,16,32,64,128)
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* configure the HXTAL as CK_OUT clock source */
rcu_ckout_config(RCU_CKOUTSRC_HXTAL, RCU_CKOUT_DIV1);
```

### **rcu\_pll\_config**

The description of rcu\_pll\_config is shown as below:

**Table 3-262. Function rcu\_pll\_config**

<b>Function name</b>	rcu_pll_config
<b>Function prototype</b>	void rcu_pll_config(uint32_t pll_src, uint32_t pll_mul);
<b>Function descriptions</b>	configure the main PLL clock
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<i>pll_src</i>	PLL clock source selection
<i>RCU_PLLSRC_IRC8M</i>	IRC8M/2 clock is selected as source clock of PLL

<i>_DIV2</i>	
<i>RCU_PLLSRC_HXTAL</i>	HXTAL is selected as source clock of PLL
<b>Input parameter{in}</b>	
<i>pll_mul</i>	PLL clock multiplication factor
<i>RCU_PLL_MULx</i>	PLL source clock * x (x = 2..32)
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* configure the PLL */

rcu_pll_config(RCU_PLLSRC_HXTAL, RCU_PLL_MUL10);
```

### **rcu\_usart\_clock\_config**

The description of **rcu\_usart\_clock\_config** is shown as below:

**Table 3-263. Function **rcu\_usart\_clock\_config****

<b>Function name</b>	rcu_usart_clock_config
<b>Function prototype</b>	void rcu_usart_clock_config(uint32_t ck_usart);
<b>Function descriptions</b>	configure the USART clock source selection
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<i>ck_usart</i>	USART clock source selection
<i>RCU_USART0SRC_C_KAPB2</i>	CK_USART0 select CK_APB2
<i>RCU_USART0SRC_C_KSYS</i>	CK_USART0 select CK_SYS
<i>RCU_USART0SRC_LXTAL</i>	CK_USART0 select CK_LXTAL
<i>RCU_USART0SRC_IRC8M</i>	CK_USART0 select CK_IRC8M
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* configure the LXTAL as USART0 clock */

rcu_usart_clock_config(RCU_USART0SRC_LXTAL);
```

### **rcu\_rtc\_clock\_config**

The description of `rcu_rtc_clock_config` is shown as below:

**Table 3-264. Function `rcu_rtc_clock_config`**

<b>Function name</b>	<code>rcu_rtc_clock_config</code>
<b>Function prototype</b>	<code>void rcu_rtc_clock_config(uint32_t rtc_clock_source);</code>
<b>Function descriptions</b>	configure the RTC clock source selection
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<code>rtc_clock_source</code>	RTC clock source selection
<code>RCU_RTCSRC_NONE</code>	no clock selected
<code>RCU_RTCSRC_LXTAL</code>	select CK_LXTAL as RTC source clock
<code>RCU_RTCSRC_IRC40K</code>	select CK_IRC40K as RTC source clock
<code>RCU_RTCSRC_HXTAL_DIV_32</code>	select (CK_HXTAL / 32) as RTC source clock
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* configure the RTC clock source selection */

rcu_rtc_clock_config(RCU_RTCSRC_IRC40K);
```

### **rcu\_hxtal\_prediv\_config**

The description of `rcu_hxtal_prediv_config` is shown as below:

**Table 3-265. Function `rcu_hxtal_prediv_config`**

<b>Function name</b>	<code>rcu_hxtal_prediv_config</code>
<b>Function prototype</b>	<code>void rcu_hxtal_prediv_config(uint32_t hxtal_prediv)</code>
<b>Function descriptions</b>	configure the HXTAL divider used as input of PLL
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<code>hxtal_prediv</code>	HXTAL divider used as input of PLL
<code>RCU_PLL_PREDVx</code>	HXTAL divided x used as input of PLL (x=1..16)
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* configure the PLL clock source selection */

rcu_hxtal_prediv_config(RCU_PLL_PREDV2);
```

### **rcu\_lxtal\_drive\_capability\_config**

The description of `rcu_lxtal_drive_capability_config` is shown as below:

**Table 3-266. Function `rcu_lxtal_drive_capability_config`**

<b>Function name</b>	rcu_lxtal_drive_capability_config
<b>Function prototype</b>	void rcu_lxtal_drive_capability_config(uint32_t lxtal_dricap);
<b>Function descriptions</b>	configure the LXTAL drive capability
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<i>lxtal_dricap</i>	drive capability of LXTAL
<i>RCU_LXTAL_LOWDRI</i>	lower driving capability
<i>RCU_LXTAL_MED_LOWDRI</i>	medium low driving capability
<i>RCU_LXTAL_MED_GHDRI</i>	medium high driving capability
<i>RCU_LXTAL_HIGHDRI</i>	higher driving capability
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* configure the LXTAL drive capability */

rcu_lxtal_drive_capability_config (RCU_LXTAL_LOWDRI);
```

### **rcu\_flag\_get**

The description of `rcu_flag_get` is shown as below:

**Table 3-267. Function `rcu_flag_get`**

<b>Function name</b>	rcu_flag_get
<b>Function prototype</b>	FlagStatus rcu_flag_get(rcu_flag_enum flag);
<b>Function descriptions</b>	get the clock stabilization and peripheral reset flags
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>flag</b>	the clock stabilization and peripheral reset flags, refer to <code>rcu_flag_enum</code>

<i>RCU_FLAG_IRC40KST</i> <i>B</i>	IRC40K stabilization flag
<i>RCU_FLAG_LXTALST</i> <i>B</i>	LXTAL stabilization flag
<i>RCU_FLAG_IRC8MST</i> <i>B</i>	IRC8M stabilization flag
<i>RCU_FLAG_HXTALST</i> <i>B</i>	HXTAL stabilization flag
<i>RCU_FLAG_PLLSTB</i>	PLL stabilization flag
<i>RCU_FLAG_IRC28MS</i> <i>TB</i>	IRC28M stabilization flag
<i>RCU_FLAG_V12RST</i>	V12 domain power reset flag
<i>RCU_FLAG_OBLRST</i>	option byte loader reset flag
<i>RCU_FLAG_EPRST</i>	external PIN reset flag
<i>RCU_FLAG_PORRST</i>	power reset flag
<i>RCU_FLAG_SWRST</i>	software reset flag
<i>RCU_FLAG_FWDGTR</i> <i>ST</i>	free watchdog timer reset flag
<i>RCU_FLAG_WWDGTR</i> <i>ST</i>	window watchdog timer reset flag
<i>RCU_FLAG_LPRST</i>	low-power reset flag
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
<b>FlagStatus</b>	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 3-268. Function `rcu_all_reset_flag_clear`**

<b>Function name</b>	rcu_all_reset_flag_clear
<b>Function prototype</b>	void rcu_all_reset_flag_clear(void);
<b>Function descriptions</b>	clear all the reset flag
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
-	-

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 3-269. 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 <code>rcu_int_flag_enum</code>
<i>RCU_INT_FLAG_IRC4OKSTB</i>	IRC40K stabilization interrupt flag
<i>RCU_INT_FLAG_LXTALSTB</i>	LXTAL stabilization interrupt flag
<i>RCU_INT_FLAG_IRC8MSTB</i>	IRC8M stabilization interrupt flag
<i>RCU_INT_FLAG_HXTALSTB</i>	HXTAL stabilization interrupt flag
<i>RCU_INT_FLAG_PLLSTB</i>	PLL stabilization interrupt flag
<i>RCU_INT_FLAG_IRC28MSTB</i>	IRC28M stabilization interrupt flag
<i>RCU_INT_FLAG_CKM</i>	HXTAL clock stuck interrupt flag
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 3-270. Function rcu\_interrupt\_flag\_clear**

<b>Function name</b>	rcu_interrupt_flag_clear
<b>Function prototype</b>	void rcu_interrupt_flag_clear(rcu_int_flag_clear_enum int_flag_clear)
<b>Function descriptions</b>	clear the interrupt flags
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>int_flag_clear</b>	clock stabilization and stuck interrupt flags clear, refer to rcu_int_flag_clear_enum
<i>RCU_INT_FLAG_IRC4 OKSTB_CLR</i>	IRC40K stabilization interrupt flag clear
<i>RCU_INT_FLAG_LXTA LSTB_CLR</i>	LXTAL stabilization interrupt flag clear
<i>RCU_INT_FLAG_IRC8 MSTB_CLR</i>	IRC8M stabilization interrupt flag clear
<i>RCU_INT_FLAG_HXT ALSTB_CLR</i>	HXTAL stabilization interrupt flag clear
<i>RCU_INT_FLAG_PLLS TB_CLR</i>	PLL stabilization interrupt flag clear
<i>RCU_INT_FLAG_IRC2 8MSTB_CLR</i>	IRC28M stabilization interrupt flag clear
<i>RCU_INT_FLAG_CKM _CLR</i>	clock stuck interrupt flag clear
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

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 3-271. Function rcu\_interrupt\_enable**

<b>Function name</b>	rcu_interrupt_enable
----------------------	----------------------

<b>Function prototype</b>	void rcu_interrupt_enable(rcu_int_enum stab_int);
<b>Function descriptions</b>	enable the stabilization interrupt
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>stab_int</b>	clock stabilization interrupt, refer to rcu_int_enum
<i>RCU_INT_IRC40KSTB</i>	IRC40K stabilization interrupt enable
<i>RCU_INT_LXTALSTB</i>	LXTAL stabilization interrupt enable
<i>RCU_INT_IRC8MSTB</i>	IRC8M stabilization interrupt enable
<i>RCU_INT_HXTALSTB</i>	HXTAL stabilization interrupt enable
<i>RCU_INT_PLLSTB</i>	PLL stabilization interrupt enable
<i>RCU_INT_IRC28MSTB</i>	IRC28M stabilization interrupt enable
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

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 3-272. Function `rcu_interrupt_disable`**

<b>Function name</b>	rcu_interrupt_disable
<b>Function prototype</b>	void rcu_interrupt_disable(rcu_int_enum stab_int);
<b>Function descriptions</b>	disable the stabilization interrupt
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>stab_int</b>	clock stabilization interrupt, refer to rcu_int_enum
<i>RCU_INT_IRC40KSTB</i>	IRC40K stabilization interrupt disable
<i>RCU_INT_LXTALSTB</i>	LXTAL stabilization interrupt disable
<i>RCU_INT_IRC8MSTB</i>	IRC8M stabilization interrupt disable
<i>RCU_INT_HXTALSTB</i>	HXTAL stabilization interrupt disable
<i>RCU_INT_PLLSTB</i>	PLL stabilization interrupt disable
<i>RCU_INT_IRC28MSTB</i>	IRC28M stabilization interrupt disable
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* disable the HXTAL stabilization interrupt */

rcu_interrupt_disable(RCU_INT_HXTALSTB);
```

### **rcu\_oscி\_stab\_wait**

The description of `rcu_oscி_stab_wait` is shown as below:

**Table 3-273. Function `rcu_oscி_stab_wait`**

<b>Function name</b>	rcu_oscி_stab_wait
<b>Function prototype</b>	ErrStatus rcu_oscி_stab_wait(rcu_oscி_type_enum oscி);
<b>Function descriptions</b>	wait for oscillator stabilization flags is SET or oscillator startup is timeout
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>oscி</b>	oscillator types, refer to <code>rcu_oscி_type_enum</code>
<i>RCU_HXTAL</i>	high speed crystal oscillator(HXTAL)
<i>RCU_LXTAL</i>	low speed crystal oscillator(LXTAL)
<i>RCU_IRC8M</i>	internal 8M RC oscillators(IRC8M)
<i>RCU_IRC28M</i>	internal 28M RC oscillators(IRC28M)
<i>RCU_IRC40K</i>	internal 40K RC oscillator(IRC40K)
<i>RCU_PLL_CK</i>	phase locked loop(PLL)
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
<b>ErrStatus</b>	SUCCESS or ERROR

Example:

```
/* wait for oscillator stabilization flag */

if(SUCCESS == rcu_oscி_stab_wait(RCU_HXTAL)){
}
```

### **rcu\_oscி\_on**

The description of `rcu_oscி_on` is shown as below:

**Table 3-274. Function `rcu_oscி_on`**

<b>Function name</b>	rcu_oscி_on
<b>Function prototype</b>	void rcu_oscி_on(rcu_oscி_type_enum oscி);
<b>Function descriptions</b>	turn on the oscillator
<b>Precondition</b>	-
<b>The called functions</b>	-

Input parameter{in}	
<b>osci</b>	oscillator types, refer to rcu_osc_type_enum
<i>RCU_HXTAL</i>	high speed crystal oscillator(HXTAL)
<i>RCU_LXTAL</i>	low speed crystal oscillator(LXTAL)
<i>RCU_IRC8M</i>	internal 8M RC oscillators(IRC8M)
<i>RCU_IRC28M</i>	internal 28M RC oscillators(IRC28M)
<i>RCU_IRC40K</i>	internal 40K RC oscillator(IRC40K)
<i>RCU_PLL_CK</i>	phase locked loop(PLL)
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* turn on the high speed crystal oscillator */
rcu_osc_on(RCU_HXTAL);
```

### **rcu\_osc\_off**

The description of **rcu\_osc\_off** is shown as below:

**Table 3-275. Function **rcu\_osc\_off****

<b>Function name</b>	rcu_osc_off
<b>Function prototype</b>	void rcu_osc_off(rcu_osc_type_enum osci);
<b>Function descriptions</b>	turn off the oscillator
<b>Precondition</b>	-
<b>The called functions</b>	-
Input parameter{in}	
<b>osci</b>	oscillator types, refer to rcu_osc_type_enum
<i>RCU_HXTAL</i>	high speed crystal oscillator(HXTAL)
<i>RCU_LXTAL</i>	low speed crystal oscillator(LXTAL)
<i>RCU_IRC8M</i>	internal 8M RC oscillators(IRC8M)
<i>RCU_IRC28M</i>	internal 28M RC oscillators(IRC28M)
<i>RCU_IRC40K</i>	internal 40K RC oscillator(IRC40K)
<i>RCU_PLL_CK</i>	phase locked loop(PLL)
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* turn off the high speed crystal oscillator */
rcu_osc_off(RCU_HXTAL);
```

### **rcu\_oscibypass\_mode\_enable**

The description of `rcu_oscibypass_mode_enable` is shown as below:

**Table 3-276. Function `rcu_oscibypass_mode_enable`**

<b>Function name</b>	rcu_oscibypass_mode_enable
<b>Function prototype</b>	<code>void rcu_oscibypass_mode_enable(rcu_oscibypass_mode_enum osci);</code>
<b>Function descriptions</b>	enable the oscillator bypass mode
<b>Precondition</b>	HXTALEN or LXTALEN must be reset before it
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>osci</b>	oscillator types, refer to <code>rcu_oscibypass_mode_enum</code>
<i>RCU_HXTAL</i>	high speed crystal oscillator(HXTAL)
<i>RCU_LXTAL</i>	low speed crystal oscillator(LXTAL)
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* enable the high speed crystal oscillator bypass mode */
rcu_oscibypass_mode_enable(RCU_HXTAL);
```

### **rcu\_oscibypass\_mode\_disable**

The description of `rcu_oscibypass_mode_disable` is shown as below:

**Table 3-277. Function `rcu_oscibypass_mode_disable`**

<b>Function name</b>	rcu_oscibypass_mode_disable
<b>Function prototype</b>	<code>void rcu_oscibypass_mode_disable(rcu_oscibypass_mode_enum osci);</code>
<b>Function descriptions</b>	disable the oscillator bypass mode
<b>Precondition</b>	HXTALEN or LXTALEN must be reset before it
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>osci</b>	oscillator types, refer to <code>rcu_oscibypass_mode_enum</code>
<i>RCU_HXTAL</i>	high speed crystal oscillator(HXTAL)
<i>RCU_LXTAL</i>	low speed crystal oscillator(LXTAL)
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* disable the high speed crystal oscillator bypass mode */
```

---

```
rcu_osc_bypass_mode_disable(RCU_HXTAL);
```

### **rcu\_hxtal\_clock\_monitor\_enable**

The description of `rcu_hxtal_clock_monitor_enable` is shown as below:

**Table 3-278. Function `rcu_hxtal_clock_monitor_enable`**

<b>Function name</b>	rcu_hxtal_clock_monitor_enable
<b>Function prototype</b>	void rcu_hxtal_clock_monitor_enable(void);
<b>Function descriptions</b>	enable the HXTAL clock monitor
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
-	-
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

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 3-279. Function `rcu_hxtal_clock_monitor_disable`**

<b>Function name</b>	rcu_hxtal_clock_monitor_disable
<b>Function prototype</b>	void rcu_hxtal_clock_monitor_disable(void);
<b>Function descriptions</b>	disable the HXTAL clock monitor
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
-	-
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* disable the HXTAL clock monitor */

rcu_hxtal_clock_monitor_disable();
```

### **rcu\_irc8m\_adjust\_value\_set**

The description of `rcu_irc8m_adjust_value_set` is shown as below:

**Table 3-280. Function `rcu_irc8m_adjust_value_set`**

<b>Function name</b>	rcu_irc8m_adjust_value_set
<b>Function prototype</b>	void rcu_irc8m_adjust_value_set(uint32_t irc8m_adjval);
<b>Function descriptions</b>	set the IRC8M adjust value
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
irc8m_adjval	IRC8M adjust value, must be between 0 and 0x1F
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* set the IRC8M adjust value */
rcu_irc8m_adjust_value_set(0x10);
```

### **rcu\_irc28m\_adjust\_value\_set**

The description of `rcu_irc28m_adjust_value_set` is shown as below:

**Table 3-281. Function `rcu_irc28m_adjust_value_set`**

<b>Function name</b>	rcu_irc28m_adjust_value_set
<b>Function prototype</b>	void rcu_irc28m_adjust_value_set(uint32_t irc28m_adjval);
<b>Function descriptions</b>	set the IRC28M adjust value
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
irc28m_adjval	IRC28M adjust value, must be between 0 and 0x1F
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* set the IRC28M adjust value */
rcu_irc28m_adjust_value_set(0x10);
```

### **rcu\_voltage\_key\_unlock**

The description of rcu\_voltage\_key\_unlock is shown as below:

**Table 3-282. Function rcu\_voltage\_key\_unlock**

<b>Function name</b>	rcu_voltage_key_unlock
<b>Function prototype</b>	void rcu_voltage_key_unlock (void);
<b>Function descriptions</b>	unlock the voltage key
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
-	-
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* unlock the voltage key */
rcu_voltage_key_unlock();
```

### **rcu\_deepsleep\_voltage\_set**

The description of rcu\_deepsleep\_voltage\_set is shown as below:

**Table 3-283. Function rcu\_deepsleep\_voltage\_set**

<b>Function name</b>	rcu_deepsleep_voltage_set
<b>Function prototype</b>	void rcu_deepsleep_voltage_set(uint32_t dsvol);
<b>Function descriptions</b>	set voltage in deep sleep mode
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>dsvol</b>	deep sleep mode voltage
<i>RCU_DEEPSLEEP_V_1_0</i>	the core voltage is 1.0V in deep-sleep mode
<i>RCU_DEEPSLEEP_V_0_9</i>	the core voltage is 0.9V in deep-sleep mode
<i>RCU_DEEPSLEEP_V_0_8</i>	the core voltage is 0.8V in deep-sleep mode
<i>RCU_DEEPSLEEP_V_1_2</i>	the core voltage is 1.2V in deep-sleep mode
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	

-	-
---	---

Example:

```
/* set the deep-sleep mode voltage */

rcu_deepsleep_voltage_set(RCU_DEEPSLEEP_V_1_0);

rcu_clock_freq_get
```

The description of `rcu_clock_freq_get` is shown as below:

**Table 3-284. Function `rcu_clock_freq_get`**

<b>Function name</b>	rcu_clock_freq_get
<b>Function prototype</b>	uint32_t rcu_clock_freq_get(rcu_clock_freq_enum clock);
<b>Function descriptions</b>	get the system clock, bus clock and peripheral clock frequency
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>clock</b>	the clock frequency which to get
<i>CK_SYS</i>	system clock frequency
<i>CK_AHB</i>	AHB clock frequency
<i>CK_APB1</i>	APB1 clock frequency
<i>CK_APB2</i>	APB2 clock frequency
<i>CK_ADC</i>	ADC clock frequency
<i>CK_USART</i>	USART0 clock frequency
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
<b>uint32_t</b>	clock frequency of system, AHB, APB1, APB2, ADC or USART0

Example:

```
uint32_t temp_freq;

/* get the system clock frequency */

temp_freq = rcu_clock_freq_get(CK_SYS);
```

## 3.15. 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 [3.15.1](#), the FWDGT firmware functions are introduced in chapter [3.15.2](#).

### 3.15.1. Descriptions of Peripheral registers

RTC registers are listed in the table shown as below:

**Table 3-285. 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_ALRM0TD	RTC alarm 0 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_BKP0	RTC backup 0 register
RTC_BKP1	RTC backup 1 register
RTC_BKP2	RTC backup 2 register
RTC_BKP3	RTC backup 3 register
RTC_BKP4	RTC backup 4 register

### 3.15.2. Descriptions of Peripheral functions

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

**Table 3-286. 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

Function name	Function description
rtc_alarm_subsecond_config	configure subsecond of RTC alarm
rtc_alarm_get	get RTC alarm
rtc_alarm_subsecond_get	get RTC alarm subsecond
rtc_alarm_enable	enable RTC alarm
rtc_alarm_disable	disable RTC alarm
rtc_timestamp_enable	enable RTC time-stamp
rtc_timestamp_disable	disable RTC time-stamp
rtc_timestamp_get	get RTC timestamp time and date
rtc_timestamp_subsecond_get	get RTC time-stamp subsecond
rtc_tamper_enable	enable RTC tamper
rtc_tamper_disable	disable RTC tamper
rtc_interrupt_enable	enable specified RTC interrupt
rtc_interrupt_disable	disable specified RTC interrupt
rtc_flag_get	check specified flag
rtc_flag_clear	clear specified flag
rtc_alter_output_config	configure RTC alternate output source
rtc_calibration_config	configure RTC calibration register
rtc_hour_adjust	adjust the daylight saving time by adding or subtracting one hour from the current time
rtc_second_adjust	adjust RTC second or subsecond value of current time
rtc_bypass_shadow_enable	enable RTC bypass shadow registers function
rtc_bypass_shadow_disable	disable RTC bypass shadow registers function
rtc_refclock_detection_enable	enable RTC reference clock detection function
rtc_refclock_detection_disable	disable RTC reference clock detection function

## Structure `rtc_parameter_struct`

**Table 3-287. `rtc_parameter_struct`**

Member name	Function description
rtc_year	RTC year value: 0x0 - 0x99(BCD format)
rtc_month	RTC month value (BCD format)
rtc_date	RTC date value: 0x1 - 0x31(BCD format)
rtc_day_of_week	RTC weekday value(BCD format)
rtc_hour	RTC hour value: 0x1 - 0x12(BCD format) or 0x0 - 0x23(BCD format)
rtc_minute	RTC minute value: 0x0 - 0x59(BCD format)
rtc_second	RTC second value: 0x0 - 0x59(BCD format)
rtc_factor_asyn	RTC asynchronous prescaler value: 0x0 - 0x7F
rtc_factor_syn	RTC synchronous prescaler value: 0x0 - 0x7FFF
rtc_am_pm	RTC AM/PM value
rtc_display_format	RTC time notation

### **Structure rtc\_alarm\_struct**

**Table 3-288. rtc\_alarm\_struct**

<b>Member name</b>	<b>Function description</b>
rtc_alarm_mask	RTC alarm mask
rtc_weekday_or_dat e	specify RTC alarm is on date or weekday
rtc_alarm_day	RTC alarm date or weekday value(BCD format)
rtc_alarm_hour	RTC alarm hour value: 0x1 - 0x12(BCD format) or 0x0 - 0x23(BCD format)
rtc_alarm_minute	RTC alarm minute value: 0x0 - 0x59(BCD format)
rtc_alarm_second	RTC alarm second value: 0x0 - 0x59(BCD format)
rtc_am_pm	RTC alarm AM/PM value

### **Structure rtc\_timestamp\_struct**

**Table 3-289. rtc\_timestamp\_struct**

<b>Member name</b>	<b>Function description</b>
rtc_timestamp_mont h	RTC time-stamp month value(BCD format)
rtc_timestamp_date	RTC time-stamp date value: 0x1 - 0x31(BCD format)
rtc_timestamp_day	RTC time-stamp weekday value(BCD format)
rtc_timestamp_hour	RTC time-stamp hour value(BCD format): 0x1 - 0x12(BCD format) or 0x0 - 0x23(BCD format)
rtc_timestamp_minu te	RTC time-stamp minute value: 0x0 - 0x59(BCD format)
rtc_timestamp_seco nd	RTC time-stamp second value: 0x0 - 0x59(BCD format)
rtc_am_pm	RTC time-stamp AM/PM value

### **Structure rtc\_tamper\_struct**

**Table 3-290. rtc\_tamper\_struct**

<b>Member name</b>	<b>Function description</b>
rtc_tamper_source	RTC tamper source
rtc_tamper_trigger	RTC tamper trigger
rtc_tamper_filter	RTC tamper consecutive samples needed during a voltage level detection
rtc_tamper_sample_frequency	RTC tamper sampling frequency during a voltage level detection
rtc_tamper_precharge_enable	RTC tamper precharge feature during a voltage level detection
rtc_tamper_precharge_time	RTC tamper precharge duration if precharge feature is enabled
rtc_tamper_with_timestamp	RTC tamper time-stamp feature

### **rtc\_deinit**

The description of rtc\_deinit is shown as below:

**Table 3-291. Function rtc\_deinit**

<b>Function name</b>	rtc_deinit
<b>Function prototype</b>	ErrStatus rtc_deinit(void);
<b>Function descriptions</b>	reset most of the RTC registers
<b>Precondition</b>	-
<b>The called functions</b>	rcu_periph_reset_enable/ rcu_periph_reset_disable -
<b>Input parameter{in}</b>	
-	-
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
<b>ErrStatus</b>	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 3-292. Function rtc\_init**

<b>Function name</b>	rtc_init
<b>Function prototype</b>	ErrStatus rtc_init(rtc_parameter_struct* rtc_initpara_struct);
<b>Function descriptions</b>	initialize RTC registers
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
rtc_initpara_struct	pointer to a rtc_parameter_struct structure which contains parameters for initialization of the rtc peripheral, the structure members can refer to members of the structure <a href="#">Table 3-287. rtc_parameter_struct</a>
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
<b>ErrStatus</b>	ERROR or SUCCESS

Example:

```
/* reset most of the RTC registers*/
ErrStatus error_status = rtc_init();
```

### **rtc\_init\_mode\_enter**

The description of `rtc_init_mode_enter` is shown as below:

**Table 3-293. Function `rtc_init_mode_enter`**

<b>Function name</b>	rtc_init_mode_enter
<b>Function prototype</b>	ErrStatus rtc_init_mode_enter(void);
<b>Function descriptions</b>	enter RTC init mode
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
-	-
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
<b>ErrStatus</b>	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 3-294. Function `rtc_init_mode_exit`**

<b>Function name</b>	rtc_init_mode_exit
<b>Function prototype</b>	void rtc_init_mode_exit(void);
<b>Function descriptions</b>	exit RTC init mode
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
-	-
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

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 3-295. Function `rtc_register_sync_wait`**

<b>Function name</b>	rtc_register_sync_wait
<b>Function prototype</b>	ErrStatus rtc_register_sync_wait(void);
<b>Function descriptions</b>	wait until RTC_TIME and RTC_DATE registers are synchronized with APB clock, and the shadow registers are updated
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
-	-
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
<b>ErrStatus</b>	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 3-296. Function `rtc_current_time_get`**

<b>Function name</b>	rtc_current_time_get
<b>Function prototype</b>	void rtc_current_time_get(rtc_parameter_struct* rtc_initpara_struct);
<b>Function descriptions</b>	get current time and date
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
-	-
<b>Output parameter{out}</b>	
<b>rtc_initpara_struct</b>	pointer to a <code>rtc_parameter_struct</code> structure which contains parameters for initialization of the rtc peripheral, the structure members can refer to members of the structure <a href="#">Table 3-287. <code>rtc_parameter_struct</code></a>
<b>Return value</b>	
-	-

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 3-297. Function `rtc_subsecond_get`**

<b>Function name</b>	rtc_subsecond_get
<b>Function prototype</b>	uint32_t rtc_subsecond_get(void);
<b>Function descriptions</b>	get current subsecond value
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
-	-
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
<b>uint32_t</b>	current subsecond value(0x00-0xFFFF)

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 3-298. Function `rtc_alarm_config`**

<b>Function name</b>	rtc_alarm_config
<b>Function prototype</b>	void rtc_alarm_config(rtc_alarm_struct* rtc_alarm_time)
<b>Function descriptions</b>	configure RTC alarm
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>rtc_alarm_time</b>	pointer to a <code>rtc_alarm_struct</code> structure which contains parameters for RTC alarm configuration, the structure members can refer to members of the structure <a href="#">Table 3-288. <code>rtc_alarm_struct</code></a>
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

---

```

/*rtc_alarm_config*/

rtc_alarm_struct rtc_alarm_time;

rtc_alarm_config(&rtc_alarm_time);

```

### **rtc\_alarm\_subsecond\_config**

The description of `rtc_alarm_subsecond_config` is shown as below:

**Table 3-299. Function `rtc_alarm_subsecond_config`**

<b>Function name</b>	rtc_alarm_subsecond_config
<b>Function prototype</b>	void rtc_alarm_subsecond_config(uint32_t mask_subsecond, uint32_t subsecond);
<b>Function descriptions</b>	configure subsecond of RTC alarm
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>mask_subsecond</b>	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_14</i>	mask RTC_ALRM0SS_SSC[14:10], and RTC_ALRM0SS_SSC[9:0] is to be compared
<i>RTC_MASKSSC_11_14</i>	mask RTC_ALRM0SS_SSC[14:11], and RTC_ALRM0SS_SSC[10:0] is to be compared
<i>RTC_MASKSSC_12_14</i>	mask RTC_ALRM0SS_SSC[14:12], and RTC_ALRM0SS_SSC[11:0] is to be compared
<i>RTC_MASKSSC_13_14</i>	mask RTC_ALRM0SS_SSC[14:13], and RTC_ALRM0SS_SSC[12:0] is to be compared

4	be compared
<i>RTC_MASKSSC_14</i>	mask RTC_ALRM0SS_SSC[14], and RTC_ALRM0SS_SSC[13:0] is to be compared
<i>RTC_MASKSSC_NONE</i>	mask none, and RTC_ALRM0SS_SSC[14:0] is to be compared
<b>Input parameter{in}</b>	
<b>subsecond</b>	alarm subsecond value(0x000 - 0x7FFF)
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/*configure subsecond of RTC alarm*/
rtc_subsecond_config (RTC_MASKSSC_9_14, 0x7FFF);
```

### **rtc\_alarm\_enable**

The description of `rtc_alarm_enable` is shown as below:

**Table 3-300. Function `rtc_alarm_enable`**

<b>Function name</b>	rtc_alarm_enable
<b>Function prototype</b>	void rtc_alarm_enable(void);
<b>Function descriptions</b>	enable RTC alarm
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
-	-
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/*enable RTC alarm*/
rtc_alarm_enable();
```

### **rtc\_alarm\_disable**

The description of `rtc_alarm_disable` is shown as below:

**Table 3-301. Function `rtc_alarm_disable`**

<b>Function name</b>	rtc_alarm_disable
----------------------	-------------------

<b>Function prototype</b>	ErrStatus rtc_alarm_disable(void);
<b>Function descriptions</b>	disable RTC alarm
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
-	-
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
<b>ErrStatus</b>	ERROR or SUCCESS

Example:

```
/*disable RTC alarm*/
ErrStatus error_status = rtc_alarm_disable();
```

### **rtc\_alarm\_get**

The description of `rtc_alarm_get` is shown as below:

**Table 3-302. Function `rtc_alarm_get`**

<b>Function name</b>	rtc_alarm_get
<b>Function prototype</b>	void rtc_alarm_get(rtc_alarm_struct* rtc_alarm_time);
<b>Function descriptions</b>	get RTC alarm
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
-	-
<b>Output parameter{out}</b>	
<b>rtc_alarm_time</b>	pointer to a <code>rtc_alarm_struct</code> structure which contains parameters for RTC alarm configuration, the structure members can refer to members of the structure <a href="#">Table 3-288. rtc_alarm_struct</a>
<b>Return value</b>	
-	-

Example:

```
/*disable RTC alarm*/
rtc_alarm_struct rtc_alarm_time;
rtc_alarm_get (&rtc_alarm_time);
```

### **rtc\_alarm\_subsecond\_get**

The description of `rtc_alarm_subsecond_get` is shown as below:

**Table 3-303. Function rtc\_alarm\_subsecond\_get**

<b>Function name</b>	rtc_alarm_subsecond_get
<b>Function prototype</b>	uint32_t rtc_alarm_subsecond_get(void);
<b>Function descriptions</b>	get RTC alarm subsecond
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
-	-
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
<b>uint32_t</b>	RTC alarm subsecond value(0x0-0x3FFF)

Example:

```
/*get RTC alarm subsecond*/
uint32_t subsecond =  rtc_alarm_subsecond_get();
```

### **rtc\_timestamp\_enable**

The description of **rtc\_timestamp\_enable** is shown as below:

**Table 3-304. Function rtc\_timestamp\_enable**

<b>Function name</b>	rtc_timestamp_enable
<b>Function prototype</b>	void rtc_timestamp_enable(uint32_t edge);
<b>Function descriptions</b>	enable RTC time-stamp
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>edge</b>	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
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

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 3-305. Function rtc\_timestamp\_disable**

<b>Function name</b>	rtc_timestamp_disable
<b>Function prototype</b>	void rtc_timestamp_disable(void);
<b>Function descriptions</b>	disable RTC time-stamp
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
-	-
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/*disable RTC time-stamp*/
rtc_timestamp_disable();
```

### **rtc\_timestamp\_get**

The description of rtc\_timestamp\_get is shown as below:

**Table 3-306. Function rtc\_timestamp\_get**

<b>Function name</b>	rtc_timestamp_get
<b>Function prototype</b>	void rtc_timestamp_get(rtc_timestamp_struct* rtc_timestamp);
<b>Function descriptions</b>	get RTC timestamp time and date
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
-	-
<b>Output parameter{out}</b>	
rtc_timestamp	Pointer to a rtc_timestamp_struct structure which contains parameters for RTC time-stamp configuration, the structure members can refer to members of the structure <a href="#">Table 3-290. rtc_tamper_struct</a>
<b>Return value</b>	
-	-

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 3-307. Function `rtc_timestamp_subsecond_get`**

<b>Function name</b>	rtc_timestamp_subsecond_get
<b>Function prototype</b>	uint32_t rtc_timestamp_subsecond_get(void);
<b>Function descriptions</b>	get RTC time-stamp subsecond
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
-	-
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
<b>uint32_t</b>	RTC time-stamp subsecond value

Example:

```
/* get RTC time-stamp subsecond */
uint32_t subsecond = rtc_timestamp_subsecond_get();
```

### **rtc\_tamper\_enable**

The description of `rtc_tamper_enable` is shown as below:

**Table 3-308. Function `rtc_timestamp_enable`**

<b>Function name</b>	rtc_tamper_enable
<b>Function prototype</b>	void rtc_tamper_enable(rtc_tamper_struct* rtc_tamper);
<b>Function descriptions</b>	enable RTC tamper
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>rtc_tamper</b>	pointer to a <code>rtc_tamper_struct</code> structure which contains parameters for RTC tamper configuration, the structure members can refer to members of the structure <a href="#">Table 3-290. <code>rtc_tamper_struct</code></a>
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

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 3-309. Function `rtc_tamper_disable`**

<b>Function name</b>	rtc_tamper_disable
<b>Function prototype</b>	void rtc_tamper_disable(uint32_t source);
<b>Function descriptions</b>	disable RTC tamper
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>source</b>	specify which tamper source to be disabled
<i>RTC_TAMPER0</i>	RTC tamper0
<i>RTC_TAMPER1</i>	RTC tamper1
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* disable RTC tamper */
rtc_tamper_disable(RTC_TAMPER0);
```

### **rtc\_interrupt\_enable**

The description of `rtc_interrupt_enable` is shown as below:

**Table 3-310. Function `rtc_interrupt_enable`**

<b>Function name</b>	rtc_interrupt_enable
<b>Function prototype</b>	void rtc_interrupt_enable(uint32_t interrupt);
<b>Function descriptions</b>	enable specified RTC interrupt
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>interrupt</b>	specify which interrupt source to be enabled
<i>RTC_INT_TIMESTAMP</i>	timestamp interrupt
<i>RTC_INT_ALARM</i>	alarm interrupt
<i>RTC_INT_TAMP</i>	tamp interrupt
<b>Output parameter{out}</b>	
-	-

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 3-311. Function `rtc_interrupt_disable`**

<b>Function name</b>	rtc_interrupt_disable
<b>Function prototype</b>	void rtc_interrupt_disable(uint32_t interrupt);
<b>Function descriptions</b>	disable specified RTC interrupt
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>interrupt</b>	specify which RTC interrupt to disable
<i>RTC_INT_TIMESTAMP</i>	second interrupt
<i>RTC_INT_ALARM</i>	alarm interrupt
<i>RTC_INT_TAMP</i>	tamp interrupt
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

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 3-312. Function `rtc_flag_get`**

<b>Function name</b>	rtc_flag_get
<b>Function prototype</b>	FlagStatus rtc_flag_get(uint32_t flag);
<b>Function descriptions</b>	check specified flag
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>flag</b>	specify which flag to check

<i>RTC_FLAG_RECALIBRATION</i>	recalibration pending flag
<i>RTC_FLAG_TAMP1</i>	tamper 1 event flag
<i>RTC_FLAG_TAMP0</i>	tamper 0 event flag
<i>RTC_FLAG_TIMESTAMP_OVERFLOW</i>	time-stamp overflow event flag
<i>RTC_FLAG_TIMESTAMP</i>	time-stamp event flag
<i>RTC_FLAG_ALARM0</i>	alarm event flag
<i>RTC_FLAG_INIT</i>	init mode event flag
<i>RTC_FLAG_RSYN</i>	time and date registers synchronized event flag
<i>RTC_FLAG_YCM</i>	year parameter configured event flag
<i>RTC_FLAG_SHIFT</i>	shift operation pending flag
<i>RTC_FLAG_ALARM0_WRITTEN</i>	alarm written available flag
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
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 3-313. Function `rtc_flag_clear`**

<b>Function name</b>	<code>rtc_flag_clear</code>
<b>Function prototype</b>	<code>void rtc_flag_clear(uint32_t flag);</code>
<b>Function descriptions</b>	clear specified flag
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>flag</b>	specify which flag to clear
<i>RTC_FLAG_TAMP1</i>	tamper 1 event flag
<i>RTC_FLAG_TAMP0</i>	tamper 0 event flag
<i>RTC_FLAG_TIMESTAMP_OVERFLOW</i>	time-stamp overflow event flag
<i>RTC_FLAG_TIMESTAMP</i>	time-stamp event flag
<i>RTC_FLAG_ALARM0</i>	alarm event flag

<i>RTC_FLAG_RSYN</i>	time and date registers synchronized event flag
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

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 3-314. Function `rtc_alter_output_config`**

<b>Function name</b>	rtc_alter_output_config
<b>Function prototype</b>	void rtc_alter_output_config(uint32_t source, uint32_t mode);
<b>Function descriptions</b>	configure rtc alternate output source
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>source</b>	specify signal to output
<i>RTC_CALIBRATION_5 12HZ</i>	when the LSE frequency is 32768Hz and the RTC_PSC is the default value, output 512Hz signal
<i>RTC_CALIBRATION_1 HZ</i>	when the LSE frequency is 32768Hz and the RTC_PSC is the default value, output 1Hz signal
<i>RTC_ALARM_HIGH</i>	when the alarm flag is set, the output pin is high
<i>RTC_ALARM_LOW</i>	when the Alarm flag is set, the output pin is low
<b>Input parameter{in}</b>	
<b>mode</b>	specify the output pin (PC13) mode when output alarm signal
<i>RTC_ALARM_OUTPU T_OD</i>	open drain mode
<i>RTC_ALARM_OUTPU T_PP</i>	push pull mode
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* configure rtc alternate output source */

rtc_alter_output_config(RTC_ALARM_LOW, RTC_ALARM_OUTPUT_PP);
```

### **rtc\_calibration\_config**

The description of `rtc_calibration_config` is shown as below:

**Table 3-315. `rtc_calibration_config`**

<b>Function name</b>	<code>rtc_calibration_config</code>
<b>Function prototype</b>	<code>ErrStatus rtc_calibration_config(uint32_t window, uint32_t plus, uint32_t minus);</code>
<b>Function descriptions</b>	configure RTC calibration register
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<code>window</code>	select calibration window
<code>RTC_CALIBRATION_WINDOW_32S</code>	2exp20 RTCCLK cycles, 32s if RTCCLK = 32768 Hz
<code>RTC_CALIBRATION_WINDOW_16S</code>	2exp19 RTCCLK cycles, 16s if RTCCLK = 32768 Hz
<code>RTC_CALIBRATION_WINDOW_8S</code>	2exp18 RTCCLK cycles, 8s if RTCCLK = 32768 Hz
<b>Input parameter{in}</b>	
<code>plus</code>	add RTC clock or not
<code>RTC_CALIBRATION_PLUS_SET</code>	add one RTC clock every 2048 rtc clock
<code>RTC_CALIBRATION_PLUS_RESET</code>	no effect
<b>Input parameter{in}</b>	
<code>minus</code>	the RTC clock to minus during the calibration window(0x0 - 0x1FF)
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
<code>ErrStatus</code>	ERROR or SUCCESS

Example:

```
/* configure RTC calibration register*/
ErrStatus error_status = rtc_calibration_config(RTC_CALIBRATION_WINDOW_32S,
RTC_CALIBRATION_PLUS_SET, 0xFF);
```

### **rtc\_hour\_adjust**

The description of `rtc_hour_adjust` is shown as below:

**Table 3-316. `rtc_hour_adjust`**

<b>Function name</b>	<code>rtc_hour_adjust</code>
<b>Function prototype</b>	<code>void rtc_hour_adjust(uint32_t operation);</code>

<b>Function descriptions</b>	adjust the daylight saving time by adding or subtracting one hour from the current time
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>operation</b>	hour adjustment operation
<i>RTC_CTL_A1H</i>	add one hour
<i>RTC_CTL_S1H</i>	subtract one hour
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

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 3-317. `rtc_second_adjust`**

<b>Function name</b>	<code>rtc_second_adjust</code>
<b>Function prototype</b>	<code>ErrStatus rtc_second_adjust(uint32_t add, uint32_t minus);</code>
<b>Function descriptions</b>	adjust RTC second or subsecond value of current time
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>add</b>	add 1s to current time or not
<i>RTC_SHIFT_ADD1S_R</i> <i>ESET</i>	no effect
<i>RTC_SHIFT_ADD1S_S</i> <i>ET</i>	add 1s to current time
<b>Input parameter{in}</b>	
<b>minus</b>	number of subsecond to minus from current time(0x0 - 0x7FFF)
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

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 3-318. rtc\_bypass\_shadow\_enable**

<b>Function name</b>	rtc_bypass_shadow_enable
<b>Function prototype</b>	<code>void rtc_bypass_shadow_enable(void);</code>
<b>Function descriptions</b>	enable RTC bypass shadow registers function
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
-	-
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

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 3-319. rtc\_bypass\_shadow\_disable**

<b>Function name</b>	rtc_bypass_shadow_disable
<b>Function prototype</b>	<code>void rtc_bypass_shadow_disable (void);</code>
<b>Function descriptions</b>	disable RTC bypass shadow registers function
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
-	-
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* disable RTC bypass shadow registers function*/
rtc_bypass_shadow_disable ();
```

### **rtc\_refclock\_detection\_enable**

The description of `rtc_refclock_detection_enable` shown as below:

**Table 3-320. rtc\_refclock\_detection\_enable**

<b>Function name</b>	rtc_refclock_detection_enable
<b>Function prototype</b>	ErrStatus rtc_refclock_detection_enable(void);
<b>Function descriptions</b>	enable RTC reference clock detection function
<b>Precondition</b>	-
<b>The called functions</b>	rtc_init_mode_enter/rtc_init_mode_exit
<b>Input parameter{in}</b>	
-	-
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
<b>ErrStatus</b>	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 3-321. rtc\_refclock\_detection\_disable**

<b>Function name</b>	rtc_refclock_detection_disable
<b>Function prototype</b>	ErrStatus rtc_refclock_detection_disable(void);
<b>Function descriptions</b>	disable RTC reference clock detection function
<b>Precondition</b>	-
<b>The called functions</b>	rtc_init_mode_enter/rtc_init_mode_exit
<b>Input parameter{in}</b>	
-	-
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
<b>ErrStatus</b>	ERROR or SUCCESS

Example:

```
/* disableRTC reference clock detection function*/
ErrStatus error_status = rtc_refclock_detection_disable();
```

## 3.16. 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 [3.16.1](#), the SPI/I2S firmware functions are introduced in chapter [3.16.2](#).

### 3.16.1. Descriptions of Peripheral registers

SPI/I2S registers are listed in the table shown as below:

**Table 3-322. 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	SPI quad mode control register

### 3.16.2. Descriptions of Peripheral functions

SPI/I2S firmware functions are listed in the table shown as below:

**Table 3-323. 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

<b>Function name</b>	<b>Function description</b>
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
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_ti_mode_enable	enable SPI TI mode
spi_ti_mode_disable	disable SPI TI mode
spi_nssp_mode_enable	enable SPI NSS pulse mode
spi_nssp_mode_disable	disable SPI NSS pulse mode
qspi_enable	enable quad wire SPI
qspi_disable	disable quad wire SPI
qspi_write_enable	enable quad wire SPI write
qspi_read_enable	enable quad wire SPI read
qspi_io23_output_enable	enable quad wire SPI_IO2 and SPI_IO3 pin output
qspi_io23_output_disable	disable quad wire SPI_IO2 and SPI_IO3 pin output
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
spi_i2s_flag_get	get SPI and I2S flag status
spi_crc_error_clear	clear SPI CRC error flag status
spi_fifo_access_size_config	configure SPI access size to FIFO(8-bit or 16-bit)
spi_transmit_odd_config	configure SPI total number of data transmitting by DMA is odd or not
spi_receive_odd_config	configure SPI total number of data receiving by DMA is odd or not
spi_crc_length_set	set CRC length

## Structure spi\_parameter\_struct

**Table 3-324. spi\_parameter\_struct**

<b>Member name</b>	<b>Function description</b>
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)

<b>Member name</b>	<b>Function description</b>
frame_size	SPI frame size (SPI_FRAMESIZE_xBIT, x=4,5..16)
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_e	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 3-325. Function spi\_i2s\_deinit**

<b>Function name</b>	spi_i2s_deinit
<b>Function prototype</b>	void spi_i2s_deinit(uint32_t spi_periph);
<b>Function descriptions</b>	Reset SPIx and I2Sx peripheral
<b>Precondition</b>	-
<b>The called functions</b>	rcu_periph_reset_enable / rcu_periph_reset_disable
<b>Input parameter{in}</b>	
spi_periph	SPI/I2S peripheral
SPIx	x=0,1
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* reset SPI0 */
spi_i2s_deinit(SPI0);
```

### **spi\_struct\_para\_init**

The description of spi\_struct\_para\_init is shown as below:

**Table 3-326. Function spi\_i2s\_deinit**

<b>Function name</b>	spi_struct_para_init
<b>Function prototype</b>	void spi_struct_para_init(spi_parameter_struct* spi_struct);
<b>Function descriptions</b>	Initialize the parameters of SPI struct with the default values

<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>spi_struct</b>	SPI init parameter struct, the structure members can refer to <a href="#"><u>Table 3-324. spi_parameter_struct</u></a> .
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

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 3-327. Function spi\_init**

<b>Function name</b>	spi_init
<b>Function prototype</b>	ErrStatus spi_init(uint32_t spi_periph, spi_parameter_struct* spi_struct);
<b>Function descriptions</b>	Initialize SPIx peripheral parameter
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>spi_periph</b>	SPI peripheral
<b>SPIx</b>	x=0,1
<b>Input parameter{in}</b>	
<b>spi_struct</b>	SPI parameter initialization stuct, the structure members can refer to members of the structure <a href="#"><u>Table 3-324. spi_parameter_struct</u></a>
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
<b>ErrStatus</b>	ERROR or SUCCESS

Example:

```

/* initialize SPI0 */

spi_parameter_struct spi_init_struct;

ErrStatus errstatus = ERROR;

spi_init_struct.trans_mode          = SPI_TRANSMODE_BDTRANSMIT;

```

---

```

spi_init_struct.device_mode          = SPI_MASTER;
spi_init_struct.frame_size         = SPI_FRAMESIZE_8BIT;
spi_init_struct.clock_polarity_phase = SPI_CK_PL_HIGH_PH_2EDGE;
spi_init_struct.nss                 = SPI_NSS_SOFT;
spi_init_struct.prescale            = SPI_PSC_8;
spi_init_struct.endian               = SPI_ENDIAN_MSB;
errstatus = spi_init(SPI0, &spi_init_struct);

```

### **spi\_enable**

The description of spi\_enable is shown as below:

**Table 3-328. Function spi\_enable**

<b>Function name</b>	spi_enable
<b>Function prototype</b>	void spi_enable(uint32_t spi_periph);
<b>Function descriptions</b>	Enable SPIx
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
spi_periph	SPI peripheral
SPIx	x=0,1
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```

/* enable SPI0 */

spi_enable(SPI0);

```

### **spi\_disable**

The description of spi\_disable is shown as below:

**Table 3-329. Function spi\_disable**

<b>Function name</b>	spi_disable
<b>Function prototype</b>	void spi_disable(uint32_t spi_periph);
<b>Function descriptions</b>	Disable SPIx
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	

<b>spi_periph</b>	SPI peripheral
<i>SPIx</i>	x=0,1
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* disable SPI0 */

spi_disable(SPI0);
```

### i2s\_init

The description of i2s\_init is shown as below:

**Table 3-330. Function i2s\_init**

<b>Function name</b>	i2s_init
<b>Function prototype</b>	void i2s_init(uint32_t spi_periph, uint32_t mode, uint32_t standard, uint32_t ckpl);
<b>Function descriptions</b>	Initialize I2S0 peripheral parameter
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>spi_periph</b>	I2S0 peripheral
<i>SPIx</i>	x=0
<b>Input parameter{in}</b>	
<b>mode</b>	I2S operation mode
<i>I2S_MODE_SLAVE_TX</i>	I2S slave transmit mode
<i>I2S_MODE_SLAVE_RX</i>	I2S slave receive mode
<i>I2S_MODE_MASTER_T X</i>	I2S master transmit mode
<i>I2S_MODE_MASTER_R X</i>	I2S master receive mode
<b>Input parameter{in}</b>	
<b>standard</b>	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_PCM_SHORT</i>	I2S PCM short standard
<i>I2S_STD_PCM_LONG</i>	I2S PCM long standard
<b>Input parameter{in}</b>	
<b>ckpl</b>	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
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* initialize I2S0 */

i2s_init(SPI0, I2S_MODE_MASTERTX, I2S_STD_PHILLIPS, I2S_CKPL_LOW);
```

### **i2s\_psc\_config**

The description of i2s\_psc\_config is shown as below:

**Table 3-331. Function i2s\_psc\_config**

<b>Function name</b>	i2s_psc_config
<b>Function prototype</b>	void i2s_psc_config(uint32_t spi_periph, uint32_t audiosample, uint32_t frameformat, uint32_t mckout);
<b>Function descriptions</b>	Configure I2S0 prescaler
<b>Precondition</b>	-
<b>The called functions</b>	rcu_clock_freq_get
<b>Input parameter{in}</b>	
<b>spi_periph</b>	I2S0 peripheral
<b>SPIx</b>	x=0
<b>Input parameter{in}</b>	
<b>audiosample</b>	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_32K</i>	audio sample rate is 32KHz
<i>I2S_AUDIOSAMPLE_44K</i>	audio sample rate is 44KHz
<i>I2S_AUDIOSAMPLE_48K</i>	audio sample rate is 48KHz
<i>I2S_AUDIOSAMPLE_96K</i>	audio sample rate is 96KHz
<i>I2S_AUDIOSAMPLE_192K</i>	audio sample rate is 192KHz

<b>92K</b>	
<b>Input parameter{in}</b>	
<b>frameformat</b>	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
<b>Input parameter{in}</b>	
<b>mckout</b>	I2S master clock output
<i>I2S_MCKOUT_ENABLE</i>	I2S master clock output enable
<i>I2S_MCKOUT_DISABLE</i>	I2S master clock output disable
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* configure I2S0 prescaler */

i2s_psc_config(SPI0, I2S_AUDIOSAMPLE_44K, I2S_FRAMEFORMAT_DT16B_CH16B,
I2S_MCKOUT_DISABLE);
```

### i2s\_enable

The description of i2s\_enable is shown as below:

**Table 3-332. Function i2s\_enable**

<b>Function name</b>	i2s_enable
<b>Function prototype</b>	void i2s_enable(uint32_t spi_periph);
<b>Function descriptions</b>	Enable I2S0
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>spi_periph</b>	I2S0 peripheral
<i>SPIx</i>	x=0
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	

-	-
---	---

Example:

```
/* enable I2S0*/
```

```
i2s_enable(SPI0);
```

### **i2s\_disable**

The description of i2s\_disable is shown as below:

**Table 3-333. Function i2s\_disable**

<b>Function name</b>	i2s_disable
<b>Function prototype</b>	void i2s_disable(uint32_t spi_periph);
<b>Function descriptions</b>	Disable I2S0
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>spi_periph</b>	I2S0 peripheral
<b>SPIx</b>	x=0
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* disable I2S0*/
```

```
i2s_disable(SPI0);
```

### **spi\_nss\_output\_enable**

The description of spi\_nss\_output\_enable is shown as below:

**Table 3-334. Function spi\_nss\_output\_enable**

<b>Function name</b>	spi_nss_output_enable
<b>Function prototype</b>	void spi_nss_output_enable(uint32_t spi_periph);
<b>Function descriptions</b>	Enable SPIx NSS output function
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>spi_periph</b>	SPIx peripheral
<b>SPIx</b>	x=0,1
<b>Output parameter{out}</b>	
-	-

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 3-335. Function spi\_nss\_output\_disable**

<b>Function name</b>	spi_nss_output_disable
<b>Function prototype</b>	void spi_nss_output_disable(uint32_t spi_periph);
<b>Function descriptions</b>	Disable SPIx NSS output function
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
spi_periph	SPIx peripheral
SPIx	x=0,1
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

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 3-336. Function spi\_nss\_internal\_high**

<b>Function name</b>	spi_nss_internal_high
<b>Function prototype</b>	void spi_nss_internal_high(uint32_t spi_periph);
<b>Function descriptions</b>	SPI NSS pin high level in software mode
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
spi_periph	SPI peripheral
SPIx	x=0,1
<b>Output parameter{out}</b>	

-	-
<b>Return value</b>	
-	-

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 3-337. Function spi\_nss\_internal\_low**

<b>Function name</b>	spi_nss_internal_low
<b>Function prototype</b>	void spi_nss_internal_low(uint32_t spi_periph);
<b>Function descriptions</b>	SPI NSS pin low level in software mode
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>spi_periph</b>	SPI peripheral
<b>SPIx</b>	x=0,1
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

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 3-338. Function spi\_dma\_enable**

<b>Function name</b>	spi_dma_enable
<b>Function prototype</b>	void spi_dma_enable(uint32_t spi_periph, uint8_t dma);
<b>Function descriptions</b>	Enable SPIx DMA function
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>spi_periph</b>	SPI peripheral
<b>SPIx</b>	x=0,1

Input parameter{in}	
<b>dma</b>	SPI DMA mode
<b>SPI_DMA_TRANSMIT</b>	SPI transmit data use DMA
<b>SPI_DMA_RECEIVE</b>	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 3-339. Function spi\_dma\_disable**

<b>Function name</b>	spi_dma_disable
<b>Function prototype</b>	void spi_dma_disable(uint32_t spi_periph, uint8_t dma);
<b>Function descriptions</b>	Disable SPIx DMA function
<b>Precondition</b>	-
<b>The called functions</b>	-
Input parameter{in}	
<b>spi_periph</b>	SPI peripheral
<b>SPIx</b>	x=0,1
Input parameter{in}	
<b>dma</b>	SPI DMA mode
<b>SPI_DMA_TRANSMIT</b>	SPI transmit data use DMA
<b>SPI_DMA_RECEIVE</b>	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 3-340. Function spi\_i2s\_data\_frame\_format\_config**

<b>Function name</b>	spi_i2s_data_frame_format_config
<b>Function prototype</b>	ErrStatus spi_i2s_data_frame_format_config(uint32_t spi_periph, uint16_t frame_format);
<b>Function descriptions</b>	Configure SPIx/I2S0 data frame format
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
spi_periph	SPI peripheral
SPIx	x=0,1
<b>Input parameter{in}</b>	
frame_format	SPI frame size
SPI_FRAMESIZE_xBIT	SPI frame size is x bits,x=4,5,6,...,15,16
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
ErrStatus	ERROR or SUCCESS

Example:

```
/* configure SPI0/I2S0 data frame format size is 16 bits */
spi_i2s_data_frame_format_config(SPI0, SPI_FRAMESIZE_16BIT);
```

### **spi\_i2s\_data\_transmit**

The description of spi\_i2s\_data\_transmit is shown as below:

**Table 3-341. Function spi\_i2s\_data\_transmit**

<b>Function name</b>	spi_i2s_data_transmit
<b>Function prototype</b>	void spi_i2s_data_transmit(uint32_t spi_periph, uint16_t data);
<b>Function descriptions</b>	SPI transmit data
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
spi_periph	SPI peripheral
SPIx	x=0,1
<b>Input parameter{in}</b>	
data	16-bit data
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

---

```
/* SPI0 transmit data */

spi_i2s_data_transmit(SPI0, spi0_send_array[send_n]);
```

### **spi\_i2s\_data\_receive**

The description of spi\_i2s\_data\_receive is shown as below:

**Table 3-342. Function spi\_i2s\_data\_receive**

<b>Function name</b>	spi_i2s_data_receive
<b>Function prototype</b>	uint16_t spi_i2s_data_receive(uint32_t spi_periph);
<b>Function descriptions</b>	SPI receive data
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
spi_periph	SPI peripheral
SPIx	x=0,1
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
uint16_t	16-bit data

Example:

```
/* SPI0 receive data */

spi0_receive_array[receive_n] = spi_i2s_data_receive(SPI0);
```

### **spi\_bidirectional\_transfer\_config**

The description of spi\_bidirectional\_transfer\_config is shown as below:

**Table 3-343. Function spi\_bidirectional\_transfer\_config**

<b>Function name</b>	spi_bidirectional_transfer_config
<b>Function prototype</b>	void spi_bidirectional_transfer_config(uint32_t spi_periph, uint32_t transfer_direction);
<b>Function descriptions</b>	Configure SPIx bidirectional transfer direction
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
spi_periph	SPI peripheral
SPIx	x=0,1
<b>Input parameter{in}</b>	
transfer_direction	SPI transfer direction
SPI_BIDIRECTIONAL_TRANSMIT	SPI work in transmit-only mode

<b>SPI_BIDIRECTIONAL_RECEIVE</b>	SPI work in receive-only mode
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* SPI0 works in transmit-only mode */
spi_bidirectional_transfer_config(SPI0, SPI_BIDIRECTIONAL_TRANSMIT);
```

### **spi\_crc\_polynomial\_set**

The description of spi\_crc\_polynomial\_set is shown as below:

**Table 3-344. Function spi\_crc\_polynomial\_set**

<b>Function name</b>	spi_crc_polynomial_set
<b>Function prototype</b>	void spi_crc_polynomial_set(uint32_t spi_periph, uint16_t crc_poly);
<b>Function descriptions</b>	Set SPI CRC polynomial
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>spi_periph</b>	SPI peripheral
SPIx	x=0,1
<b>Input parameter{in}</b>	
<b>crc_poly</b>	CRC polynomial value
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* set SPI0 CRC polynomial */
spi_crc_polynomial_set(SPI0,CRC_VALUE);
```

### **spi\_crc\_polynomial\_get**

The description of spi\_crc\_polynomial\_get is shown as below:

**Table 3-345. Function spi\_crc\_polynomial\_get**

<b>Function name</b>	spi_crc_polynomial_get
<b>Function prototype</b>	uint16_t spi_crc_polynomial_get(uint32_t spi_periph);
<b>Function descriptions</b>	Get SPI CRC polynomial

<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>spi_periph</b>	SPI peripheral
<b>SPIx</b>	x=0,1
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
<b>uint16_t</b>	16 bit CRC polynomial value

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 3-346. Function spi\_crc\_on**

<b>Function name</b>	spi_crc_on
<b>Function prototype</b>	void spi_crc_on(uint32_t spi_periph);
<b>Function descriptions</b>	Turn on CRC function
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>spi_periph</b>	SPI peripheral
<b>SPIx</b>	x=0,1
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* turn on SPI0 CRC function */

spi_crc_on(SPI0);
```

### **spi\_crc\_off**

The description of **spi\_crc\_off** is shown as below:

**Table 3-347. Function spi\_crc\_off**

<b>Function name</b>	spi_crc_off
----------------------	-------------

<b>Function prototype</b>	void spi_crc_off(uint32_t spi_periph);
<b>Function descriptions</b>	Turn off CRC function
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
spi_periph	SPI peripheral
SPIx	x=0,1
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* turn off SPI0 CRC function */

spi_crc_off(SPI0);
```

### **spi\_crc\_next**

The description of spi\_crc\_next is shown as below:

**Table 3-348. Function spi\_crc\_next**

<b>Function name</b>	spi_crc_next
<b>Function prototype</b>	void spi_crc_next(uint32_t spi_periph);
<b>Function descriptions</b>	SPI next data is CRC value
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
spi_periph	SPI peripheral
SPIx	x=0,1
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

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 3-349. Function spi\_crc\_get**

<b>Function name</b>	spi_crc_get
<b>Function prototype</b>	uint16_t spi_crc_get(uint32_t spi_periph,uint8_t crc);
<b>Function descriptions</b>	Get SPI CRC send value or receive value
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
spi_periph	SPI peripheral
SPIx	x=0,1
<b>Input parameter{in}</b>	
crc	SPI crc value
SPI_CRC_TX	get transmit crc value
SPI_CRC_RX	get receive crc value
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
uint16_t	16-bit CRC value

Example:

```
/* get SPI0 CRC send value */

uint16_t crc_val;

crc_val = spi_crc_get(SPI0, SPI_CRC_TX);
```

### **spi\_ti\_mode\_enable**

The description of spi\_ti\_mode\_enable is shown as below:

**Table 3-350. Function spi\_ti\_mode\_enable**

<b>Function name</b>	spi_ti_mode_enable
<b>Function prototype</b>	void spi_ti_mode_enable(uint32_t spi_periph);
<b>Function descriptions</b>	Enable SPI TI mode
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
spi_periph	SPI peripheral
SPIx	x=0,1
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

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 3-351. Function spi\_ti\_mode\_disable**

<b>Function name</b>	spi_ti_mode_disable
<b>Function prototype</b>	void spi_ti_mode_disable(uint32_t spi_periph);
<b>Function descriptions</b>	Disable SPI TI mode
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
spi_periph	SPI peripheral
SPIx	x=0,1
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* disable SPI0 TI mode */
spi_ti_mode_disable(SPI0);
```

### **spi\_nssp\_mode\_enable**

The description of spi\_nssp\_mode\_enable is shown as below:

**Table 3-352. Function spi\_nssp\_mode\_enable**

<b>Function name</b>	spi_ti_mode_enable
<b>Function prototype</b>	void spi_ti_mode_enable(uint32_t spi_periph);
<b>Function descriptions</b>	Enable SPI NSS pulse mode
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
spi_periph	SPI peripheral
SPIx	x=0,1
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* enable SPI0 NSS pulse mode */
```

```
spi_nssp_mode_enable(SPI0);
```

### **spi\_nssp\_mode\_disable**

The description of spi\_nssp\_mode\_disable is shown as below:

**Table 3-353. Function spi\_nssp\_mode\_disable**

<b>Function name</b>	spi_ti_mode_disable
<b>Function prototype</b>	void spi_ti_mode_disable(uint32_t spi_periph);
<b>Function descriptions</b>	Disable SPI NSS pulse mode
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
spi_periph	SPI peripheral
SPIx	x=0,1
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* disable SPI0 NSS pulse mode */
spi_nssp_mode_disable(SPI0);
```

### **qspi\_enable**

The description of qspi\_enable is shown as below:

**Table 3-354. Function qspi\_enable**

<b>Function name</b>	qspi_enable
<b>Function prototype</b>	void qspi_enable(uint32_t spi_periph);
<b>Function descriptions</b>	Enable quad wire SPI
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
spi_periph	SPI peripheral
SPIx	x=1
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* enable SPI1 quad wire mode */
```

```
qspi_enable(SPI1);
```

### **qspi\_disable**

The description of qspi\_disable is shown as below:

**Table 3-355. Function qspi\_disable**

<b>Function name</b>	qspi_disable
<b>Function prototype</b>	void qspi_disable(uint32_t spi_periph);
<b>Function descriptions</b>	Disable quad wire SPI
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
spi_periph	SPI peripheral
SPIx	x=1
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* disable SPI1 quad wire mode */
qspi_disable(SPI1);
```

### **qspi\_write\_enable**

The description of qspi\_write\_enable is shown as below:

**Table 3-356. Function qspi\_write\_enable**

<b>Function name</b>	qspi_write_enable
<b>Function prototype</b>	void qspi_write_enable(uint32_t spi_periph);
<b>Function descriptions</b>	Enable quad wire SPI write
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
spi_periph	SPI peripheral
SPIx	x=1
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* enable SPI1 quad wire write */
```

```
qspi_write_enable(SPI1);
```

### **qspi\_read\_enable**

The description of qspi\_read\_enable is shown as below:

**Table 3-357. Function qspi\_read\_enable**

<b>Function name</b>	qspi_read_enable
<b>Function prototype</b>	void qspi_read_enable(uint32_t spi_periph);
<b>Function descriptions</b>	Enable quad wire SPI read
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
spi_periph	SPI peripheral
SPIx	x=1
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* enable SPI1 quad wire read */
qspi_read_enable(SPI1);
```

### **qspi\_io23\_output\_enable**

The description of qspi\_io23\_output\_enable is shown as below:

**Table 3-358. Function qspi\_io23\_output\_enable**

<b>Function name</b>	qspi_io23_output_enable
<b>Function prototype</b>	void qspi_io23_output_enable(uint32_t spi_periph);
<b>Function descriptions</b>	Enable SPI_IO2 and SPI_IO3 pin output
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
spi_periph	SPI peripheral
SPIx	x=1
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* enable SPI1 SPI_IO2 and SPI_IO3 pin output */
```

```
qspi_io23_output_enable(SPI1);
```

### **qspi\_io23\_output\_disable**

The description of qspi\_io23\_output\_disable is shown as below:

**Table 3-359. Function qspi\_io23\_output\_disable**

<b>Function name</b>	qspi_io23_output_disable
<b>Function prototype</b>	void qspi_io23_output_disable(uint32_t spi_periph);
<b>Function descriptions</b>	Disable SPI_IO2 and SPI_IO3 pin output
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
spi_periph	SPI peripheral
SPIx	x=1
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* disable SPI1 SPI_IO2 and SPI_IO3 pin output */
qspi_io23_output_disable(SPI1);
```

### **spi\_i2s\_interrupt\_enable**

The description of spi\_i2s\_interrupt\_enable is shown as below:

**Table 3-360. Function spi\_i2s\_interrupt\_enable**

<b>Function name</b>	spi_i2s_interrupt_enable
<b>Function prototype</b>	void spi_i2s_interrupt_enable(uint32_t spi_periph, uint8_t interrupt);
<b>Function descriptions</b>	Enable SPIx and I2S0 interrupt
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
spi_periph	SPI peripheral
SPIx	x=0,1
<b>Input parameter{in}</b>	
interrupt	SPI/I2S interrupt
SPI_I2SINT_TBE	transmit buffer empty interrupt
SPI_I2S_INT_RBNE	receive buffer not empty interrupt
SPI_I2S_INT_ERR	CRC error,configuration error,reception overrun error, transmission underrun error and format error interrupt
<b>Output parameter{out}</b>	

-	-
<b>Return value</b>	
-	-

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 3-361. Function spi\_i2s\_interrupt\_disable**

<b>Function name</b>	spi_i2s_interrupt_disable
<b>Function prototype</b>	void spi_i2s_interrupt_disable(uint32_t spi_periph, uint8_t interrupt);
<b>Function descriptions</b>	Disable SPIx and I2S0 interrupt
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>spi_periph</b>	SPI peripheral
<b>SPIx</b>	x=0,1
<b>Input parameter{in}</b>	
<b>interrupt</b>	SPI/I2S interrupt
<b>SPI_I2SINT_TBE</b>	transmit buffer empty interrupt
<b>SPI_I2S_INT_RBNE</b>	receive buffer not empty interrupt
<b>SPI_I2S_INT_ERR</b>	CRC error,configuration error,reception overrun error, transmission underrun error and format error interrupt
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

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 3-362. Function spi\_i2s\_interrupt\_flag\_get**

<b>Function name</b>	spi_i2s_interrupt_flag_get
<b>Function prototype</b>	FlagStatus spi_i2s_interrupt_flag_get(uint32_t spi_periph, uint8_t interrupt);

<b>Function descriptions</b>	Get SPIx and I2S0 interrupt flag status
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>spi_periph</b>	SPI peripheral
<b>SPIx</b>	x=0,1
<b>Input parameter{in}</b>	
<b>interrupt</b>	SPI/I2S interrupt
<b>SPI_I2S_INT_FLAG_T BE</b>	transmit buffer empty interrupt
<b>SPI_I2S_INT_FLAG_R BNE</b>	receive buffer not empty interrupt
<b>SPI_I2S_INT_FLAG_R XORERR</b>	overrun interrupt
<b>SPI_INT_FLAG_CONF ERR</b>	config error interrupt
<b>SPI_INT_FLAG_CRCE RR</b>	CRC error interrupt
<b>I2S_INT_FLAG_TXUR ERR</b>	underrun error interrupt
<b>SPI_I2S_INT_FLAG_F ERR</b>	format error interrupt
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
<b>FlagStatus</b>	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++]);
}

```

### **spi\_i2s\_flag\_get**

The description of spi\_i2s\_flag\_get is shown as below:

**Table 3-363. Function spi\_i2s\_flag\_get**

<b>Function name</b>	spi_i2s_flag_get
<b>Function prototype</b>	FlagStatus spi_i2s_flag_get(uint32_t spi_periph, uint8_t interrupt);
<b>Function descriptions</b>	Get SPIx and I2S0 flag status

<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>spi_periph</b>	SPI peripheral
<b>SPIx</b>	x=0,1
<b>Input parameter{in}</b>	
<b>flag</b>	SPI/I2S flag status
<b>SPI_FLAG_TBE</b>	transmit buffer empty flag
<b>SPI_FLAG_RBNE</b>	receive buffer not empty flag
<b>SPI_FLAG_TRANS</b>	transmit on-going flag
<b>SPI_FLAG_RXOVERR</b>	receive overrun error flag
<b>SPI_FLAG_CONFERR</b>	mode config error flag
<b>SPI_FLAG_CRCERR</b>	CRC error flag
<b>SPI_FLAG_FERR</b>	SPI format error interrupt flag
<b>I2S_FLAG_TBE</b>	transmit buffer empty flag
<b>I2S_FLAG_RBNE</b>	receive buffer not empty flag
<b>I2S_FLAG_TRANS</b>	transmit on-going flag
<b>I2S_FLAG_RXOVERR</b>	overrun error flag
<b>I2S_FLAG_TXURERR</b>	underrun error flag
<b>I2S_FLAG_CH</b>	channel side flag
<b>I2S_FLAG_FERR</b>	I2S format error interrupt flag
Only for SPI1	
<b>SPI_TXLVL_EMPTY</b>	SPI TXFIFO is empty
<b>SPI_TXLVL_QUARTER_FULL</b>	SPI TXFIFO is a quarter of full
<b>SPI_TXLVL_HAIF_FULL</b>	SPI TXFIFO is a half of full
<b>SPI_TXLVL_FULL</b>	SPI TXFIFO is full
<b>SPI_RXLVL_EMPTY</b>	SPI RXFIFO is empty
<b>SPI_RXLVL_QUARTE_R_FULL</b>	SPI RXFIFO is a quarter of full
<b>SPI_RXLVL_HAIF_FULL</b>	SPI RXFIFO is a half of full
<b>SPI_RXLVL_FULL</b>	SPI RXFIFO is full
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
<b>FlagStatus</b>	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\_crc\_error\_clear**

The description of spi\_crc\_error\_clear is shown as below:

**Table 3-364. Function spi\_crc\_error\_clear**

<b>Function name</b>	spi_crc_error_clear
<b>Function prototype</b>	void spi_crc_error_clear(uint32_t spi_periph);
<b>Function descriptions</b>	Clear SPI CRC error flag status
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
spi_periph	SPI peripheral
SPIx	x=0,1
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* clear SPI0CRC error flag status */
spi_crc_error_clear(SPI0);
```

### **spi\_fifo\_access\_size\_config**

The description of spi\_fifo\_access\_size\_config is shown as below:

**Table 3-365. Function spi\_fifo\_access\_size\_config**

<b>Function name</b>	spi_fifo_access_size_config
<b>Function prototype</b>	void spi_fifo_access_size_config(uint32_t spi_periph, uint16_t fifo_access_size);
<b>Function descriptions</b>	Configure SPI1 access size to FIFO(8bit or 16bit)
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
spi_periph	SPI peripheral
SPIx	x=1
<b>Input parameter{in}</b>	
fifo_access_size	byte access enable
SPI_HALFWORD_ACCESS	half-word access to FIFO
SPI_BYTE_ACCESS	byte access to FIFO
<b>Output parameter{out}</b>	

-	-
<b>Return value</b>	
-	-

Example:

```
/* configure SPI1 access size half word */
spi_fifo_access_config(SPI1, SPI_HALFWORD_ACCESS);
```

### **spi\_transmit\_odd\_config**

The description of spi\_transmit\_odd\_config is shown as below:

**Table 3-366. Function spi\_transmit\_odd\_config**

<b>Function name</b>	spi_transmit_odd_config
<b>Function prototype</b>	void spi_transmit_odd_config(uint32_t spi_periph, uint16_t odd);
<b>Function descriptions</b>	configure SPI1 total number of data to transmit by DMA is odd or not
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>spi_periph</b>	SPI peripheral
<b>SPIx</b>	x=1
<b>Input parameter{in}</b>	
<b>odd</b>	odd bytes in TX DMA channel
<b>SPI_TXDMA EVEN</b>	number of byte in TX DMA channel is even
<b>SPI_TXDMA ODD</b>	number of byte in TX DMA channel is odd
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* configure SPI1 total number of data to transmit by DMA is odd */
spi_transmit_odd_config(SPI1, SPI_TXDMA_ODD);
```

### **spi\_receive\_odd\_config**

The description of spi\_receive\_odd\_config is shown as below:

**Table 3-367. Function spi\_receive\_odd\_config**

<b>Function name</b>	spi_receive_odd_config
<b>Function prototype</b>	void spi_receive_odd_config(uint32_t spi_periph, uint16_t odd);
<b>Function descriptions</b>	configure SPI1 total number of data to receive by DMA is odd or not
<b>Precondition</b>	-

<b>The called functions</b>		-
<b>Input parameter{in}</b>		
<b>spi_periph</b>		SPI peripheral
<b>SPIx</b>		x=1
<b>Input parameter{in}</b>		
<b>odd</b>		odd bytes in TX DMA channel
<b>SPI_RXDMA EVEN</b>		number of byte in RX DMA channel is even
<b>SPI_RXDMA ODD</b>		number of byte in RX DMA channel is odd
<b>Output parameter{out}</b>		
-		-
<b>Return value</b>		
-		-

Example:

```
/* configure SPI1 total number of data to receive by DMA is odd */
spi_receive_odd_config(SPI1, SPI_RXDMA_ODD);
```

### **spi\_crc\_length\_set**

The description of spi\_crc\_length\_set is shown as below:

**Table 3-368. Function spi\_crc\_length\_set**

<b>Function name</b>	spi_crc_length_set	
<b>Function prototype</b>	void spi_crc_length_set(uint32_t spi_periph, uint16_t crc_length);	
<b>Function descriptions</b>	set CRC length	
<b>Precondition</b>	-	
<b>The called functions</b>	-	
<b>Input parameter{in}</b>		
<b>spi_periph</b>		SPI peripheral
<b>SPIx</b>		x=1
<b>Input parameter{in}</b>		
<b>crc_length</b>		CRC length
<b>SPI_CRC_8BIT</b>		CRC length is 8 bits
<b>SPI_CRC_16BIT</b>		CRC length is 16 bits
<b>Output parameter{out}</b>		
-		-
<b>Return value</b>		
-		-

Example:

```
/* set SPI1 CRC length 16 bits */
spi_crc_length_set(SPI1, SPI_CRC_16BIT);
```

## 3.17. SYSCFG

The SYSCFG registers are listed in chapter [3.17.1](#), the SYSCFG firmware functions are introduced in chapter [3.17.2](#).

### 3.17.1. Descriptions of Peripheral registers

SYSCFG registers are listed in the table shown as below:

**Table 3-369. SYSCFG Registers**

Registers	Descriptions
SYSCFG_CFG0	system configuration register 0
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_CFG2	system configuration register 2
SYSCFG_CPU_IRQ_LAT	IRQ Latency register

### 3.17.2. Descriptions of Peripheral functions

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

**Table 3-370. SYSCFG firmware function**

Function name	Function description
syscfg_deinit	deinit syscfg module
syscfg_dma_remap_enable	enable the DMA channels remapping
syscfg_dma_remap_disable	disable the DMA channels remapping
syscfg_high_current_enable	enable PB9 high current capability
syscfg_high_current_disable	disable PB9 high current capability
syscfg_exti_line_config	configure the GPIO pin as EXTI Line
syscfg_lock_config	connect TIMER0/14/15/16 break input to the selected parameter
irq_latency_set	set the IRQ_LATENCY value
syscfg_flag_get	check if the specified flag in SYSCFG_CFG2 is set or not
syscfg_flag_clear	clear the flag in SYSCFG_CFG2 by writing 1

#### **syscfg\_deinit**

The description of syscfg\_deinit is shown as below:

**Table 3-371. Function syscfg\_deinit**

Function name	syscfg_deinit
Function prototype	void syscfg_deinit(void);

<b>Function descriptions</b>	reset the SYSCFG registers
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
-	-
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* reset SYSCFG registers */

syscfg_deinit();
```

### **syscfg\_dma\_remap\_enable**

The description of syscfg\_dma\_remap\_enable is shown as below:

**Table 3-372. Function syscfg\_dma\_remap\_enable**

<b>Function name</b>	syscfg_dma_remap_enable
<b>Function prototype</b>	void syscfg_dma_remap_enable (void);
<b>Function descriptions</b>	enable the DMA channels remapping
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>syscfg_dma_remap</b>	specify the DMA channels to remap
<b>SYSCFG_DMA_REMA_P_TIMER16</b>	remap TIMER16 channel0 and UP DMA requests to channel1(default channel0)
<b>SYSCFG_DMA_REMA_P_TIMER15</b>	remap TIMER15 channel2 and UP DMA requests to channel3(default channel2)
<b>SYSCFG_DMA_REMA_P_USART0RX</b>	remap USART0 Rx DMA request to channel4(default channel2)
<b>SYSCFG_DMA_REMA_P_USART0TX</b>	remap USART0 Tx DMA request to channel3(default channel1)
<b>SYSCFG_DMA_REMA_P_ADC</b>	remap ADC DMA requests from channel0 to channel1
<b>SYSCFG_PA11_REMA_P_PA12</b>	remap PA11 PA12
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

---

```
/* enable DMA channel remap*/
syscfg_dma_remap_enable(SYSCFG_DMA_REMAP_TIMER16);
```

### **syscfg\_dma\_remap\_disable**

The description of syscfg\_dma\_remap\_disable is shown as below:

**Table 3-373. Function syscfg\_dma\_remap\_disable**

<b>Function name</b>	syscfg_dma_remap_disable
<b>Function prototype</b>	void syscfg_dma_remap_disable (void);
<b>Function descriptions</b>	disable the DMA channels remapping
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>syscfg_dma_remap</b>	specify the DMA channels to remap
<b>SYSCFG_DMA_REMA P_TIMER16</b>	remap TIMER16 channel0 and UP DMA requests to channel1(default channel0)
<b>SYSCFG_DMA_REMA P_TIMER15</b>	remap TIMER15 channel2 and UP DMA requests to channel3(default channel2)
<b>SYSCFG_DMA_REMA P_USART0RX</b>	remap USART0 Rx DMA request to channel4(default channel2)
<b>SYSCFG_DMA_REMA P_USART0TX</b>	remap USART0 Tx DMA request to channel3(default channel1)
<b>SYSCFG_DMA_REMA P_ADC</b>	remap ADC DMA requests from channel0 to channel1
<b>SYSCFG_PA11_REMA P_PA12</b>	remap PA11 PA12
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* disable DMA channel remap*/
syscfg_dma_remap_disable(SYSCFG_DMA_REMAP_TIMER16);
```

### **syscfg\_high\_current\_enable**

The description of syscfg\_high\_current\_enable is shown as below:

**Table 3-374. Function syscfg\_high\_current\_enable**

<b>Function name</b>	syscfg_high_current_enable
<b>Function prototype</b>	void syscfg_high_current_enable(void);
<b>Function descriptions</b>	enable PB9 high current capability

<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
-	-
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* enable PB9 high current capability */
syscfg_high_current_enable();
```

### **syscfg\_high\_current\_disable**

The description of syscfg\_high\_current\_disable is shown as below:

**Table 3-375. Function syscfg\_high\_current\_disable**

<b>Function name</b>	syscfg_high_current_disable
<b>Function prototype</b>	void syscfg_high_current_disable(void);
<b>Function descriptions</b>	disable PB9 high current capability
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
-	-
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* disable PB9 high current capability */
syscfg_high_current_disable();
```

### **syscfg\_exti\_line\_config**

The description of syscfg\_exti\_line\_config is shown as below:

**Table 3-376. Function syscfg\_exti\_line\_config**

<b>Function name</b>	syscfg_exti_line_config
<b>Function prototype</b>	void syscfg_exti_line_config(uint8_t exti_port, uint8_t exti_pin);
<b>Function descriptions</b>	configure the GPIO pin as EXTI Line
<b>Precondition</b>	-

The called functions	
Input parameter{in}	
<b>exti_port</b>	specify the GPIO port used in EXTI
<i>EXTI_SOURCE_GPIOx</i>	x=A,B,C,F
<b>exti_pin</b>	specify the EXTI line
<i>EXTI_SOURCE_PINx</i>	x=0..15(GPIOA, GPIOB), x=13..15(GPIOC), x = 0..16..7 (GPIOF)
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* configure the GPIO pin as EXTI Line */
syscfg_exti_line_config(EXTI_SOURCE_GPIOA, EXTI_SOURCE_PIN0);
```

### **syscfg\_lock\_config**

The description of syscfg\_lock\_config is shown as below:

**Table 3-377. Function syscfg\_lock\_config**

Function name	
Input parameter{in}	
<b>syscfg_lock</b>	specify the parameter to be connected
<b>SYSCFG_LOCK_LOCK_UP</b>	Cortex-M23 lockup output connected to the break input
<b>SYSCFG_LOCK_SRA_M_PARITY_ERROR</b>	SRAM_PARITY check error connected to the break input
<b>SYSCFG_LOCK_LVD</b>	LVD interrupt connected to the break input
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* configure syscfg lock*/
syscfg_lock_config(SYSCFG_LOCK_LOCKUP);
```

### **irq\_latency\_set**

The description of irq\_latency\_set is shown as below:

**Table 3-378. Function irq\_latency\_set**

<b>Function name</b>	irq_latency_set
<b>Function prototype</b>	void irq_latency_set(uint8_t irq_latency);
<b>Function descriptions</b>	set the wait state counter value
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
irq_latency	IRQ_LATENCY value
0x00 - 0xFF	IRQ_LATENCY value
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* set the wait state counter value */

irq_latency_set(0xFF);
```

### **syscfg\_flag\_get**

The description of syscfg\_flag\_get is shown as below:

**Table 3-379. Function syscfg\_flag\_get**

<b>Function name</b>	syscfg_flag_get
<b>Function prototype</b>	FlagStatus syscfg_flag_get(uint32_t syscfg_flag);
<b>Function descriptions</b>	check if the specified flag in SYSCFG_CFG2 is set or not
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
syscfg_flag	specify the flag in SYSCFG_CFG2 to check
SYSCFG_SRAM_PCE_F	SRAM parity check error flag
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
FlagStatus	SET or RESET

Example:

```
/* get syscfg flag */
```

```

FlagStatus status;

status = syscfg_flag_get(SYSCFG_SRAM_PCEF);

```

### **syscfg\_flag\_clear**

The description of syscfg\_flag\_clear is shown as below:

**Table 3-380. Function syscfg\_flag\_clear**

<b>Function name</b>	syscfg_flag_clear
<b>Function prototype</b>	void syscfg_flag_clear (uint32_t syscfg_flag);
<b>Function descriptions</b>	clear the flag in SYSCFG_CFG2 by writing 1
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>syscfg_flag</b>	specify the flag in SYSCFG_CFG2 to check
<b>SYSCFG_SRAM_PCE</b> <i>F</i>	SRAM parity check error flag
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```

/* clear syscfg flag */

syscfg_flag_clear(SYSCFG_SRAM_PCEF);

```

## **3.18. 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 (TIMER0), general level0 timer (TIMER2), general level2 timer (TIMER13), general level2 timer (TIMERx, x=15, 16), Basic timer (TIMER5). The specific functions of different types of timer are different. The TIMER registers are listed in chapter [3.18.1](#), the TIMER firmware functions are introduced in chapter [3.18.2](#).

### **3.18.1. Descriptions of Peripheral registers**

TIMERx registers are listed in the table shown as below:

**Table 3-381. TIMERx Registers**

<b>Registers</b>	<b>Descriptions</b>
TIMER_CTL0(timerx, x=0, 2, 5, 13, 14, 15, 16)	Control register 0
TIMERx_CTL1(timerx, x=0, 2, 5, 13, 14, 15,	Control register 1

<b>Registers</b>	<b>Descriptions</b>
16)	
TIMERx_SMCFG(timerx, x=0, 2, 14)	Slave mode configuration register
TIMERx_DMAINTEN(timerx, x=0, 2, 5, 13, 14, 15, 16)	DMA and interrupt enable register
TIMERx_INTF(timerx, x=0, 2, 5, 13, 14, 15, 16)	Interrupt flag register
TIMERx_SWEVG(timerx, x=0, 2, 5, 13, 14, 15, 16)	Software event generation register
TIMERx_CHCTL0(timerx, x=0, 2, 13, 14, 15, 16)	Channel control register 0
TIMERx_CHCTL1(timerx, x=0, 2)	Channel control register 1
TIMERx_CHCTL2(timerx, x=0, 2, 13, 14, 15, 16)	Channel control register 2
TIMERx_CNT(timerx, x=0, 2, 5, 13, 14, 15, 16)	Counter register
TIMERx_PSC(timerx, x=0, 2, 5, 13, 14, 15, 16)	Prescaler register
TIMERx_CAR(timerx, x=0, 2, 5, 13, 14, 15, 16)	Counter auto reload register
TIMERx_CREP(timerx, x=0, 5, 14, 15, 16)	Counter repetition register
TIMERx_CH0CV(timerx, x=0, 2, 13, 14, 15, 16)	Channel 0 capture/compare value register
TIMERx_CH1CV(timerx, x=0, 2, 14)	Channel 1 capture/compare value register
TIMERx_CH2CV(timerx, x=0, 2)	Channel 2 capture/compare value register
TIMERx_CH3CV(timerx, x=0, 2)	Channel 3 capture/compare value register
TIMERx_IRMP(timerx, x=13)	Channel complementary protection register
TIMERx_CCHP(timerx, x=0, 2, 14, 15, 16)	TIMER complementary channel protection register
TIMERx_DMACFG(timerx, x=0, 2, 14, 15, 16)	DMA configuration register
TIMERx_DMATB(timerx, x=0, 2, 14, 15, 16)	DMA transfer buffer register
TIMERx_CFG(timerx, x=0, 2, 13, 14, 15, 16)	Configuration register

### 3.18.2. Descriptions of Peripheral functions

The description format of firmware functions are shown as below:

**Table 3-382. TIMERx firmware function**

<b>Function name</b>	<b>Function description</b>
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

<b>Function name</b>	<b>Function description</b>
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
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_ocpre_clear_source_config	configure TIMER OCPRE clear source selection
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
timer_flag_get	get TIMER flags
timer_flag_clear	clear TIMER flags
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_	configure TIMER channel output pulse value

<b>Function name</b>	<b>Function description</b>
value_config	
timer_channel_output_shadow_config	configure TIMER channel output shadow function
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
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_channel_remap_config	configure TIMER channel remap function
timer_write_chxval_register_config	configure TIMER write CHxVAL register selection
timer_output_value_selection_config	configure TIMER output value selection

### **Structure timer\_parameter\_struct**

**Table 3-383. Structure timer\_parameter\_struct**

<b>Member name</b>	<b>Function description</b>
prescaler	prescaler value (0~65535)
alignedmode	aligned mode (TIMER_COUNTER_EDGE, 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 3-384. Structure timer\_break\_parameter\_struct**

<b>Member name</b>	<b>Function description</b>
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 3-385. Structure timer\_oc\_parameter\_struct**

<b>Member name</b>	<b>Function description</b>
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)
ocidlestate	idle state of channel output (TIMER_OC_IDLE_STATE_LOW, TIMER_OC_IDLE_STATE_HIGH)

<b>Member name</b>	<b>Function description</b>
ocnidlestate	idle state of channel complementary output (TIMER_OCN_IDLE_STATE_LOW, TIMER_OCN_IDLE_STATE_HIGH)

### **Structure timer\_ic\_parameter\_struct**

**Table 3-386. Structure timer\_ic\_parameter\_struct**

<b>Member name</b>	<b>Function description</b>
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 3-387. Function timer\_deinit**

<b>Function name</b>	timer_deinit
<b>Function prototype</b>	void timer_deinit(uint32_t timer_periph);
<b>Function descriptions</b>	deinit a TIMER
<b>Precondition</b>	-
<b>The called functions</b>	rcu_periph_reset_enable / rcu_periph_reset_disable
<b>Input parameter{in}</b>	
timer_periph	TIMER peripheral
TIMERx(x=0, 2, 5, 13..16)	TIMER peripheral selection
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* reset TIMER0 */
timer_deinit (TIMER0);
```

### **timer\_struct\_para\_init**

The description of timer\_struct\_para\_init is shown as below:

**Table 3-388. Function timer\_struct\_para\_init**

<b>Function name</b>	timer_struct_para_init
<b>Function prototype</b>	void timer_struct_para_init(timer_parameter_struct* initpara);
<b>Function descriptions</b>	initialize the parameters of TIMER init parameter struct with the default values
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
initpara	TIMER init parameter struct, the structure members can refer to <a href="#">Table 3-383. Structure timer_parameter_struct</a> .
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

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 3-389. Function timer\_init**

<b>Function name</b>	timer_init
<b>Function prototype</b>	void timer_init(uint32_t timer_periph, timer_parameter_struct* initpara);
<b>Function descriptions</b>	initialize TIMER counter
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
timer_periph	TIMER peripheral
TIMERx(x=0, 2, 5, 13..16)	TIMER peripheral selection
<b>Input parameter{in}</b>	
initpara	TIMER init parameter struct, the structure members can refer to <a href="#">Table 3-383. Structure timer_parameter_struct</a> .
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

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 3-390. Function timer\_enable**

<b>Function name</b>	timer_enable
<b>Function prototype</b>	void timer_enable(uint32_t timer_periph);
<b>Function descriptions</b>	enable a timer
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>timer_periph</b>	TIMER peripheral
<i>TIMERx(x=0, 2, 5, 13..16)</i>	TIMER peripheral selection
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```

/* enable TIMER0 */

timer_enable (TIMER0);

```

### **timer\_disable**

The description of timer\_disable is shown as below:

**Table 3-391. Function timer\_disable**

<b>Function name</b>	timer_disable
<b>Function prototype</b>	void timer_disable(uint32_t timer_periph);

<b>Function descriptions</b>	disable a timer
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>timer_periph</b>	TIMER peripheral
<i>TIMERx(x=0, 2, 5, 13..16)</i>	TIMER peripheral selection
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* disable TIMER0 */

timer_disable (TIMER0);
```

#### **timer\_auto\_reload\_shadow\_enable**

The description of **timer\_auto\_reload\_shadow\_enable** is shown as below:

**Table 3-392. Function timer\_auto\_reload\_shadow\_enable**

<b>Function name</b>	timer_auto_reload_shadow_enable
<b>Function prototype</b>	void timer_auto_reload_shadow_enable(uint32_t timer_periph);
<b>Function descriptions</b>	enable the auto reload shadow function
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>timer_periph</b>	TIMER peripheral
<i>TIMERx(x=0, 2, 5, 13..16)</i>	TIMER peripheral selection
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

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 3-393. Function timer\_auto\_reload\_shadow\_disable**

<b>Function name</b>	timer_auto_reload_shadow_disable
<b>Function prototype</b>	void timer_auto_reload_shadow_disable (uint32_t timer_periph);
<b>Function descriptions</b>	disable the auto reload shadow function
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
timer_periph	TIMER peripheral
<i>TIMERx(x=0, 2, 5, 13..16)</i>	TIMER peripheral selection
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

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 3-394. Function timer\_update\_event\_enable**

<b>Function name</b>	timer_update_event_enable
<b>Function prototype</b>	void timer_update_event_enable(uint32_t timer_periph);
<b>Function descriptions</b>	enable the update event
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
timer_periph	TIMER peripheral
<i>TIMERx(x=0, 2, 5, 13..16)</i>	TIMER peripheral selection
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

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 3-395. Function timer\_update\_event\_disable**

<b>Function name</b>	timer_update_event_disable
<b>Function prototype</b>	void timer_update_event_disable (uint32_t timer_periph);
<b>Function descriptions</b>	disable the update event
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>timer_periph</b>	TIMER peripheral
<i>TIMERx(x=0, 2, 5, 13..16)</i>	TIMER peripheral selection
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

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 3-396. Function timer\_counter\_alignment**

<b>Function name</b>	timer_counter_alignment
<b>Function prototype</b>	void timer_counter_alignment(uint32_t timer_periph, uint16_t aligned);
<b>Function descriptions</b>	set TIMER counter alignment mode
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>timer_periph</b>	TIMER peripheral
<i>TIMERx(x=0, 2)</i>	TIMER peripheral selection
<b>Input parameter{in}</b>	
<b>aligned</b>	alignment mode
<i>TIMER_COUNTER_ED</i> <i>GE</i>	No center-aligned mode (edge-aligned mode). The direction of the counter is specified by the DIR bit.
<i>TIMER_COUNTER_CE</i> <i>NTER_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 TIMERx_CHCTL0register). Only when the counter is counting down, compare interrupt flag of channels can be set.

<i>TIMER_COUNTER_CE</i>	Center-aligned and counting up assert mode. The counter counts under center aligned and channel is configured in output mode (CHxMS=00 in TIMERx_CHCTL0register). Only when the counter is counting up, compare interrupt flag of channels can be set.
<i>TIMER_COUNTER_CE</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 TIMERx_CHCTL0 register). Both when the counter is counting up and counting down, compare interrupt flag of channels can be set.
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

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 3-397. Function timer\_counter\_up\_direction**

<b>Function name</b>	timer_counter_up_direction
<b>Function prototype</b>	void timer_counter_up_direction(uint32_t timer_periph);
<b>Function descriptions</b>	set TIMER counter up direction
<b>Precondition</b>	set TIMER counter no center-aligned mode (edge-aligned mode)
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>timer_periph</b>	TIMER peripheral
<i>TIMERx(x=0, 2)</i>	TIMER peripheral selection
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

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 3-398. timer\_counter\_down\_direction**

<b>Function name</b>	timer_counter_down_direction
<b>Function prototype</b>	void timer_counter_down_direction(uint32_t timer_periph);
<b>Function descriptions</b>	set TIMER counter down direction
<b>Precondition</b>	set TIMER counter no center-aligned mode (edge-aligned mode)
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
timer_periph	TIMER peripheral
<i>TIMERx(x=0, 2)</i>	TIMER peripheral selection
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

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 3-399. Function timer\_prescaler\_config**

<b>Function name</b>	timer_prescaler_config
<b>Function prototype</b>	void timer_prescaler_config(uint32_t timer_periph, uint16_t prescaler, uint8_t pscreload);
<b>Function descriptions</b>	configure TIMER prescaler
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
timer_periph	TIMER peripheral
<i>TIMERx(x=0, 2, 5, 13..16)</i>	TIMER peripheral selection
<b>Input parameter{in}</b>	
prescaler	prescaler value (0~65535)
<b>Input parameter{in}</b>	
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
<b>Output parameter{out}</b>	
-	-

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 3-400. Function timer\_repetition\_value\_config**

<b>Function name</b>	timer_repetition_value_config
<b>Function prototype</b>	void timer_repetition_value_config(uint32_t timer_periph, uint16_t repetition);
<b>Function descriptions</b>	configure TIMER repetition register value
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>timer_periph</b>	TIMER peripheral
<b>TIMERx(x=0,15,16)</b>	TIMER peripheral selection
<b>Input parameter{in}</b>	
<b>repetition</b>	the counter repetition value (0~255)
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

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 3-401. Function timer\_autoreload\_value\_config**

<b>Function name</b>	timer_autoreload_value_config
<b>Function prototype</b>	void timer_autoreload_value_config(uint32_t timer_periph, uint16_t autoreload);
<b>Function descriptions</b>	configure TIMER autoreload register value
<b>Precondition</b>	-
<b>The called functions</b>	-

Input parameter{in}	
<b>timer_periph</b>	TIMER peripheral
<i>TIMERx(x=0, 2, 5, 13..16)</i>	TIMER peripheral selection
Input parameter{in}	
<b>autoreload</b>	the counter auto-reload value (0-65535)
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 3-402. Function timer\_counter\_value\_config**

<b>Function name</b>	timer_counter_value_config
<b>Function prototype</b>	void timer_counter_value_config(uint32_t timer_periph, uint16_t counter);
<b>Function descriptions</b>	configure TIMER counter register value
<b>Precondition</b>	-
<b>The called functions</b>	-
Input parameter{in}	
<b>timer_periph</b>	TIMER peripheral
<i>TIMERx(x=0, 2, 5, 13..16)</i>	TIMER peripheral selection
Input parameter{in}	
<b>counter</b>	the counter value (0-65535)
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* configure TIMER0 counter register value */

timer_counter_value_config (TIMER0, 3000);
```

### **timer\_counter\_read**

The description of timer\_counter\_read is shown as below:

**Table 3-403. Function timer\_counter\_read**

<b>Function name</b>	timer_counter_read
<b>Function prototype</b>	uint32_t timer_counter_read(uint32_t timer_periph);
<b>Function descriptions</b>	read TIMER counter value
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
timer_periph	TIMER peripheral
TIMERx(x=0, 2, 5, 13..16)	TIMER peripheral selection
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
uint32_t	counter value (0~65535)

Example:

```
/* read TIMER0 counter value */

uint32_t i = 0;

i = timer_counter_read (TIMER0);
```

### **timer\_prescaler\_read**

The description of timer\_prescaler\_read is shown as below:

**Table 3-404. Function timer\_prescaler\_read**

<b>Function name</b>	timer_prescaler_read
<b>Function prototype</b>	uint16_t timer_prescaler_read(uint32_t timer_periph);
<b>Function descriptions</b>	read TIMER prescaler value
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
timer_periph	TIMER peripheral
TIMERx(x=0, 2, 5, 13..16)	TIMER peripheral selection
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
uint16_t	prescaler register value (0~65535)

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 3-405. Function timer\_single\_pulse\_mode\_config**

<b>Function name</b>	timer_single_pulse_mode_config
<b>Function prototype</b>	void timer_single_pulse_mode_config(uint32_t timer_periph, uint8_t spmode);
<b>Function descriptions</b>	configure TIMER single pulse mode
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>timer_periph</b>	TIMER peripheral
<i>TIMERx(x=0, 2, 5, 14..16)</i>	TIMER peripheral selection
<b>Input parameter{in}</b>	
<b>spmode</b>	pulse mode
<i>TIMER_SP_MODE_SIN_GLE</i>	single pulse mode
<i>TIMER_SP_MODE_REPETITIVE</i>	repetitive pulse mode
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

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 3-406. Function timer\_update\_source\_config**

<b>Function name</b>	timer_update_source_config
<b>Function prototype</b>	void timer_update_source_config(uint32_t timer_periph, uint32_t update);
<b>Function descriptions</b>	configure TIMER update source
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>timer_periph</b>	TIMER peripheral

<i>TIMERx(x=0, 2, 5, 13..16)</i>	TIMER peripheral selection
<b>Input parameter{in}</b>	
<b>update</b>	update source
<i>TIMER_UPDATE_SRC_GLOBAL</i>	Any of the following events generate an update interrupt or DMA request: - 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.
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* configure TIMER update only by counter overflow/underflow */

timer_update_source_config (TIMER0, TIMER_UPDATE_SRC_REGULAR);
```

### **timer\_ocpre\_clear\_source\_config**

The description of timer\_ocpre\_clear\_source\_config is shown as below:

**Table 3-407. Function t timer\_ocpre\_clear\_source\_config**

<b>Function name</b>	timer_ocpre_clear_source_config
<b>Function prototype</b>	void timer_ocpre_clear_source_config (uint32_t timer_periph, uint8_t ocpreclear);
<b>Function descriptions</b>	configure TIMER OCPRE clear source selection
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>timer_periph</b>	TIMER peripheral
<i>TIMERx(x=0, 2)</i>	TIMER peripheral selection
<b>Input parameter{in}</b>	
<b>ocpreclear</b>	clear source
<i>TIMER_OCPRE_CL_EAR_SOURCE_CLR_R</i>	OCPRE_CLR_INT is connected to the OCPRE_CLR input
<i>TIMER_OCPRE_CL_EAR_SOURCE_ETIF_F</i>	OCPRE_CLR_INT is connected to ETIF

Output parameter{out}	
-	-
Return value	
-	-

例如：

```
/* configure TIMER0 OCPRE_CLR_INT is connected to the OCPRE_CLR input */
timer_ocpre_clear_source_config(TIMER0, TIMER_OCPRE_CLEAR_SOURCE_CLR);
```

### **timer\_interrupt\_enable**

The description of timer\_interrupt\_enable is shown as below:

**Table 3-408. Function timer\_interrupt\_enable**

<b>Function name</b>	timer_interrupt_enable
<b>Function prototype</b>	void timer_interrupt_enable(uint32_t timer_periph, uint32_t interrupt);
<b>Function descriptions</b>	enable the TIMER interrupt
<b>Precondition</b>	-
<b>The called functions</b>	-
Input parameter{in}	
<b>timer_periph</b>	TIMER peripheral
<b>TIMERx</b>	please refer to the following parameters
Input parameter{in}	
<b>interrupt</b>	timer interrupt enable source
<b>TIMER_INT_UP</b>	update interrupt enable, TIMERx (x=0, 2, 5, 13..16)
<b>TIMER_INT_CH0</b>	channel 0 interrupt enable, TIMERx(x=0, 2, 13..16)
<b>TIMER_INT_CH1</b>	channel 1 interrupt enable, TIMERx(x=0, 2, 14)
<b>TIMER_INT_CH2</b>	channel 2 interrupt enable, TIMERx(x=0, 2)
<b>TIMER_INT_CH3</b>	channel 3 interrupt enable , TIMERx(x=0, 2)
<b>TIMER_INT_CMT</b>	commutation interrupt enable, TIMERx (x=0, 14..16)
<b>TIMER_INT_TRG</b>	trigger interrupt enable, TIMERx(x=0, 2, 14)
<b>TIMER_INT_BRK</b>	break interrupt enable, TIMERx (x=0, 14..16)
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 3-409. Function timer\_interrupt\_disable**

<b>Function name</b>	timer_interrupt_disable
<b>Function prototype</b>	void timer_interrupt_disable (uint32_t timer_periph, uint32_t interrupt);
<b>Function descriptions</b>	disable the TIMER interrupt
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>timer_periph</b>	TIMER peripheral
<b>TIMERx</b>	please refer to the following parameters
<b>Input parameter{in}</b>	
<b>interrupt</b>	timer interrupt disable source
<b>TIMER_INT_UP</b>	update interrupt disable, TIMERx (x=0, 2, 5, 13..16)
<b>TIMER_INT_CH0</b>	channel 0 interrupt disable, TIMERx(x=0, 2, 13..16)
<b>TIMER_INT_CH1</b>	channel 1 interrupt disable, TIMERx(x=0, 2, 14)
<b>TIMER_INT_CH2</b>	channel 2 interrupt disable, TIMERx(x=0, 2)
<b>TIMER_INT_CH3</b>	channel 3 interrupt disable, TIMERx(x=0, 2)
<b>TIMER_INT_CMT</b>	commutation interrupt disable, TIMERx (x=0, 14..16)
<b>TIMER_INT_TRG</b>	trigger interrupt disable, TIMERx(x=0, 2, 14)
<b>TIMER_INT_BRK</b>	break interrupt disable, TIMERx(x=0, 14..16)
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

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 3-410. Function timer\_interrupt\_flag\_get**

<b>Function name</b>	timer_interrupt_flag_get
<b>Function prototype</b>	FlagStatus timer_interrupt_flag_get(uint32_t timer_periph, uint32_t interrupt);
<b>Function descriptions</b>	get timer interrupt flag
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	

<b>timer_periph</b>	TIMER peripheral
<i>TIMERx</i>	please refer to the following parameters
<b>Input parameter{in}</b>	
<b>interrupt</b>	the timer interrupt bits
<i>TIMER_INT_FLAG_UP</i>	update interrupt flag,TIMERx(x=0, 2, 5, 13..16)
<i>TIMER_INT_FLAG_CH0</i>	channel 0 interrupt flag,TIMERx(x=0, 2, 13..16)
<i>TIMER_INT_FLAG_CH1</i>	channel 1 interrupt flag,TIMERx(x=0, 2, 14)
<i>TIMER_INT_FLAG_CH2</i>	channel 2 interrupt flag,TIMERx TIMERx(x=0, 2)
<i>TIMER_INT_FLAG_CH3</i>	channel 3 interrupt flag,TIMERx TIMERx(x=0, 2)
<i>TIMER_INT_FLAG_CM<sub>T</sub></i>	channel commutation interrupt flag, TIMERx (x=0, 14..16)
<i>TIMER_INT_FLAG_TRG</i>	trigger interrupt flag, TIMERx(x=0, 2, 14)
<i>TIMER_INT_FLAG_BRK</i>	break interrupt flag, TIMERx(x=0, 14..16)
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
<b>FlagStatus</b>	SET or RESET

Example:

```
/* get TIMER0 update interrupt flag */

FlagStatus Flag_interrupt = RESET;

Flag_interrupt = timer_interrupt_flag_get (TIMER0, TIMER_INT_FLAG_UP);
```

### **timer\_interrupt\_flag\_clear**

The description of **timer\_interrupt\_flag\_clear** is shown as below:

**Table 3-411. Function timer\_interrupt\_flag\_clear**

<b>Function name</b>	timer_interrupt_flag_clear
<b>Function prototype</b>	void timer_interrupt_flag_clear(uint32_t timer_periph, uint32_t interrupt);
<b>Function descriptions</b>	clear TIMER interrupt flag
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>timer_periph</b>	TIMER peripheral
<i>TIMERx</i>	please refer to the following parameters
<b>Input parameter{in}</b>	
<b>interrupt</b>	the timer interrupt bits
<i>TIMER_INT_FLAG_UP</i>	update interrupt flag,TIMERx(x=0, 2, 5, 13..16)
<i>TIMER_INT_FLAG_CH0</i>	channel 0 interrupt flag,TIMERx(x=0, 2, 13..16)
<i>TIMER_INT_FLAG_CH1</i>	channel 1 interrupt flag,TIMERx(x=0, 2, 14)
<i>TIMER_INT_FLAG_CH2</i>	channel 2 interrupt flag,TIMERx TIMERx(x=0, 2)

<i>TIMER_INT_FLAG_CH3</i>	channel 3 interrupt flag, TIMERx TIMERx(x=0, 2)
<i>TIMER_INT_FLAG_CM_T</i>	channel commutation interrupt flag, TIMERx (x=0, 14..16)
<i>TIMER_INT_FLAG_TRG</i>	trigger interrupt flag, TIMERx(x=0, 2, 14)
<i>TIMER_INT_FLAG_BRK</i>	break interrupt flag, TIMERx(x=0, 14..16)
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* clear TIMER0 update interrupt flag */
timer_interrupt_flag_clear (TIMER0, TIMER_INT_FLAG_UP);
```

### **timer\_flag\_get**

The description of timer\_flag\_get is shown as below:

**Table 3-412. Function timer\_flag\_get**

<b>Function name</b>	timer_flag_get
<b>Function prototype</b>	FlagStatus timer_flag_get(uint32_t timer_periph, uint32_t flag);
<b>Function descriptions</b>	get TIMER flags
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>timer_periph</b>	TIMER peripheral
<b>TIMERx</b>	please refer to the following parameters
<b>Input parameter{in}</b>	
<b>flag</b>	the timer interrupt flags
<i>TIMER_FLAG_UP</i>	update flag, TIMERx(x=0, 2, 5, 13..16)
<i>TIMER_FLAG_CH0</i>	channel 0 flag, TIMERx(x=0, 2, 13..16)
<i>TIMER_FLAG_CH1</i>	channel 1 flag, TIMERx(x=0, 2, 14)
<i>TIMER_FLAG_CH2</i>	channel 2 flag, TIMERx(x=0, 2)
<i>TIMER_FLAG_CH3</i>	channel 3 flag, TIMERx(x=0, 2)
<i>TIMER_FLAG_CMT</i>	channel commutation flag, TIMERx(x=0, 14..16)
<i>TIMER_FLAG_TRG</i>	trigger flag, TIMERx(x=0, 2, 14)
<i>TIMER_FLAG_BRK</i>	break flag, TIMERx(x=0, 14..16)
<i>TIMER_FLAG_CH0O</i>	channel 0 overcapture flag, TIMERx(x=0, 2, 3..16)
<i>TIMER_FLAG_CH1O</i>	channel 1 overcapture flag, TIMERx(x=0, 2, 14)
<i>TIMER_FLAG_CH2O</i>	channel 2 overcapture flag, TIMERx(x=0, 2)
<i>TIMER_FLAG_CH3O</i>	channel 3 overcapture flag, TIMERx(x=0, 2)
<b>Output parameter{out}</b>	
-	-

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 3-413. Function timer\_flag\_clear**

<b>Function name</b>	timer_flag_clear
<b>Function prototype</b>	void timer_flag_clear(uint32_t timer_periph, uint32_t flag);
<b>Function descriptions</b>	clear TIMER flags
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>timer_periph</b>	TIMER peripheral
<b>TIMERx</b>	please refer to the following parameters
<b>Input parameter{in}</b>	
<b>flag</b>	the timer interrupt flags
<b>TIMER_FLAG_UP</b>	update flag, TIMERx(x=0, 2, 5, 13..16)
<b>TIMER_FLAG_CH0</b>	channel 0 flag, TIMERx(x=0, 2, 13..16)
<b>TIMER_FLAG_CH1</b>	channel 1 flag, TIMERx(x=0, 2, 14)
<b>TIMER_FLAG_CH2</b>	channel 2 flag, TIMERx(x=0, 2)
<b>TIMER_FLAG_CH3</b>	channel 3 flag, TIMERx(x=0, 2)
<b>TIMER_FLAG_CMT</b>	channel commutation flag, TIMERx(x=0, 14..16)
<b>TIMER_FLAG_TRG</b>	trigger flag, TIMERx(x=0, 2, 14)
<b>TIMER_FLAG_BRK</b>	break flag, TIMERx(x=0, 14..16)
<b>TIMER_FLAG_CH0O</b>	channel 0 overcapture flag, TIMERx(x=0, 2, 13..16)
<b>TIMER_FLAG_CH1O</b>	channel 1 overcapture flag, TIMERx(x=0, 2, 14)
<b>TIMER_FLAG_CH2O</b>	channel 2 overcapture flag, TIMERx(x=0, 2)
<b>TIMER_FLAG_CH3O</b>	channel 3 overcapture flag, TIMERx(x=0, 2)
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* clear TIMER0 update flags */
```

```
timer_flag_clear (TIMER0, TIMER_FLAG_UP);
```

### **timer\_dma\_enable**

The description of timer\_dma\_enable is shown as below:

**Table 3-414. Function timer\_dma\_enable**

<b>Function name</b>	timer_dma_enable
<b>Function prototype</b>	void timer_dma_enable(uint32_t timer_periph, uint16_t dma);
<b>Function descriptions</b>	enable the TIMER DMA
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>timer_periph</b>	TIMER peripheral
<b>TIMERx</b>	please refer to the following parameters
<b>Input parameter{in}</b>	
<b>dma</b>	timer DMA source enable
<b>TIMER_DMA_UPD</b>	update DMA enable, TIMERx(x=0, 2, 5, 14..16)
<b>TIMER_DMA_CH0D</b>	channel 0 DMA enable, TIMERx(x=0, 2, 14..16)
<b>TIMER_DMA_CH1D</b>	channel 1 DMA enable, TIMERx(x=0..2, 4)
<b>TIMER_DMA_CH2D</b>	channel 2 DMA enable, TIMERx(x=0, 2)
<b>TIMER_DMA_CH3D</b>	channel 3 DMA enable, TIMERx(x=0, 2)
<b>TIMER_DMA_CMTD</b>	commutation DMA request enable, TIMERx(x=0, 14)
<b>TIMER_DMA_TRGD</b>	trigger DMA enable, TIMERx(x=0..2, 14)
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

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 3-415. Function timer\_dma\_disable**

<b>Function name</b>	timer_dma_disable
<b>Function prototype</b>	void timer_dma_disable (uint32_t timer_periph, uint16_t dma);
<b>Function descriptions</b>	disable the TIMER DMA
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	

<b>timer_periph</b>	TIMER peripheral
<i>TIMERx</i>	please refer to the following parameters
<b>Input parameter{in}</b>	
<b>dma</b>	timer DMA source disable
<i>TIMER_DMA_UPD</i>	update DMA enable, TIMERx(x=0, 2, 5, 14..16)
<i>TIMER_DMA_CH0D</i>	channel 0 DMA enable, TIMERx(x=0, 2, 14..16)
<i>TIMER_DMA_CH1D</i>	channel 1 DMA enable, TIMERx(x=0..2, 14)
<i>TIMER_DMA_CH2D</i>	channel 2 DMA enable, TIMERx(x=0, 2)
<i>TIMER_DMA_CH3D</i>	channel 3 DMA enable, TIMERx(x=0, 2)
<i>TIMER_DMA_CMTD</i>	commutation DMA request enable, TIMERx(x=0, 14)
<i>TIMER_DMA_TRGD</i>	trigger DMA enable, TIMERx(x=0..2, 14)
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

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 3-416. Function timer\_channel\_dma\_request\_source\_select**

<b>Function name</b>	timer_channel_dma_request_source_select
<b>Function prototype</b>	void timer_channel_dma_request_source_select(uint32_t timer_periph, uint32_t dma_request);
<b>Function descriptions</b>	channel DMA request source selection
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>timer_periph</b>	TIMER peripheral
<i>TIMERx(x=0, 2, 14..16)</i>	TIMER peripheral selection
<b>Input parameter{in}</b>	
<b>dma_request</b>	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
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	

-	-
---	---

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 3-417. Function timer\_dma\_transfer\_config**

<b>Function name</b>	timer_dma_transfer_config
<b>Function prototype</b>	void timer_dma_transfer_config(uint32_t timer_periph, uint32_t dma_baseaddr, uint32_t dma_lenth);
<b>Function descriptions</b>	configure the TIMER DMA transfer
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>timer_periph</b>	TIMER peripheral
<i>TIMERx(x=0, 2, 14..16)</i>	TIMER peripheral selection
<b>Input parameter{in}</b>	
<b>dma_baseaddr</b>	DMA transfer access start address
<i>TIMER_DMACFG_DMA_TA_CTL0</i>	DMA transfer address is TIMER_CTL0, TIMERx(x=0, 2, 14..16)
<i>TIMER_DMACFG_DMA_TA_CTL1</i>	DMA transfer address is TIMER_CTL1, TIMERx(x=0, 2, 14..16)
<i>TIMER_DMACFG_DMA_TA_SMCFG</i>	DMA transfer address is TIMER_SMCFG, TIMERx(x=0, 2, 14)
<i>TIMER_DMACFG_DMA_TA_DMAINTEN</i>	DMA transfer address is TIMER_DMAINTEN, TIMERx(x=0, 2, 14..16)
<i>TIMER_DMACFG_DMA_TA_INTF</i>	DMA transfer address is TIMER_INTF, TIMERx(x=0, 2, 14..16)
<i>TIMER_DMACFG_DMA_TA_SWEVG</i>	DMA transfer address is TIMER_SWEVG, TIMERx(x=0, 2, 14..16)
<i>TIMER_DMACFG_DMA_TA_CHCTL0</i>	DMA transfer address is TIMER_CHCTL0, TIMERx(x=0, 2, 14..16)
<i>TIMER_DMACFG_DMA_TA_CHCTL1</i>	DMA transfer address is TIMER_CHCTL1, TIMERx(x=0, 2)
<i>TIMER_DMACFG_DMA_TA_CHCTL2</i>	DMA transfer address is TIMER_CHCTL2, TIMERx (x=0, 2, 14..16)
<i>TIMER_DMACFG_DMA_TA_CNT</i>	DMA transfer address is TIMER_CNT, TIMERx (x=0, 2, 14..16)

<i>TIMER_DMACFG_DMA_TA_PSC</i>	DMA transfer address is TIMER_PSC, TIMERx (x=0, 2, 14..16)
<i>TIMER_DMACFG_DMA_TA_CAR</i>	MA transfer address is TIMER_CAR, TIMERx (x=0, 2, 14..16)
<i>TIMER_DMACFG_DMA_TA_CREP</i>	DMA transfer address is TIMER_CREP, TIMERx (x=0, 14..16)
<i>TIMER_DMACFG_DMA_TA_CH0CV</i>	DMA transfer address is TIMER_CH0CV, TIMERx (x=0, 2, 14..16)
<i>TIMER_DMACFG_DMA_TA_CH1CV</i>	DMA transfer address is TIMER_CH1CV, TIMERx (x=0, 2, 14)
<i>TIMER_DMACFG_DMA_TA_CH2CV</i>	DMA transfer address is TIMER_CH2CV, TIMERx (x=0, 2)
<i>TIMER_DMACFG_DMA_TA_CH3CV</i>	DMA transfer address is TIMER_CH3CV, TIMERx (x=0, 2)
<i>TIMER_DMACFG_DMA_TA_CCHP</i>	DMA transfer address is TIMER_CCHP, TIMERx (x=0, 14..16)
<i>TIMER_DMACFG_DMA_TA_DMACFG</i>	DMA transfer address is TIMER_DMACFG, TIMERx (x=0, 2, 14..16)
<b>Input parameter{in}</b>	
<i>dma_lenth</i>	DMA transfer count
<i>TIMER_DMACFG_DMA_TC_xTRANSFER</i>	x=1..18, DMA transfer x time
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

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 3-418. Function timer\_event\_software\_generate**

<b>Function name</b>	timer_event_software_generate
<b>Function prototype</b>	void timer_event_software_generate(uint32_t timer_periph, uint16_t event);
<b>Function descriptions</b>	software generate events
<b>Precondition</b>	-
<b>The called functions</b>	-

Input parameter{in}	
<b>timer_periph</b>	TIMER peripheral
<i>TIMERx</i>	please refer to the following parameters
Input parameter{in}	
<b>event</b>	the timer software event generation sources
<i>TIMER_EVENT_SRC_U PG</i>	update event, TIMERx(x=0, 2, 5, 13..16)
<i>TIMER_EVENT_SRC_C H0G</i>	channel 0 capture or compare event generation, TIMERx(x=0, 2, 13..16)
<i>TIMER_EVENT_SRC_C H1G</i>	channel 1 capture or compare event generation, TIMERx(x=0, 2, 14)
<i>TIMER_EVENT_SRC_C H2G</i>	channel 2 capture or compare event generation, TIMERx(x=0, 2)
<i>TIMER_EVENT_SRC_C H3G</i>	channel 3 capture or compare event generation, TIMERx(x=0, 2)
<i>TIMER_EVENT_SRC_C MTG</i>	channel commutation event generation, TIMERx(x=0, 14..16)
<i>TIMER_EVENT_SRC_T RGG</i>	trigger event generation, TIMERx(x=0, 2, 14)
<i>TIMER_EVENT_SRC_B RKG</i>	break event generation, TIMERx(x=0, 14..16)
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 3-419. Function timer\_break\_struct\_para\_init**

<b>Function name</b>	timer_break_struct_para_init
<b>Function prototype</b>	void timer_break_struct_para_init(timer_break_parameter_struct* breakpara);
<b>Function descriptions</b>	initialize the parameters of TIMER break parameter struct with the default values
<b>Precondition</b>	-
<b>The called functions</b>	-
Input parameter{in}	

<b>breakpara</b>	TIMER break parameter struct, the structure members can refer to <a href="#">Table 3-384. Structure timer_break_parameter_struct.</a>
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

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 3-420. Function timer\_break\_config**

<b>Function name</b>	timer_break_config
<b>Function prototype</b>	void timer_break_config(uint32_t timer_periph, timer_break_parameter_struct* breakpara);
<b>Function descriptions</b>	configure TIMER break function
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>timer_periph</b>	TIMER peripheral
<i>TIMERx(x=0, 14..16)</i>	TIMER peripheral selection
<b>Input parameter{in}</b>	
<b>breakpara</b>	TIMER break parameter struct, the structure members can refer to <a href="#">Table 3-384. Structure timer_break_parameter_struct.</a>
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* configure TIMER0 break function */

timer_break_parameter_struct timer_breakpara;

timer_breakpara.runoffstate       = TIMER_ROS_STATE_DISABLE;
timer_breakpara.idloffstate      = 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 3-421. Function timer\_break\_enable**

<b>Function name</b>	timer_break_enable
<b>Function prototype</b>	void timer_break_enable(uint32_t timer_periph);
<b>Function descriptions</b>	enable TIMER break function
<b>Precondition</b>	This function can be called only when PROT [1:0] bit-filed in TIMERx_CCHP register is 00.
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>timer_periph</b>	TIMER peripheral
<i>TIMERx(x=0, 14..16)</i>	TIMER peripheral selection
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```

/* enable TIMER0 break function*/
timer_break_enable (TIMER0);

```

### **timer\_break\_disable**

The description of timer\_break\_disable is shown as below:

**Table 3-422. Function timer\_break\_disable**

<b>Function name</b>	timer_break_disable
<b>Function prototype</b>	void timer_break_disable(uint32_t timer_periph);
<b>Function descriptions</b>	disable TIMER break function
<b>Precondition</b>	This function can be called only when PROT [1:0] bit-filed in TIMERx_CCHP register is 00.
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>timer_periph</b>	TIMER peripheral

<i>TIMERx(x=0, 14..16)</i>	TIMER peripheral selection
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

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 3-423. Function timer\_automatic\_output\_enable**

<b>Function name</b>	timer_automatic_output_enable
<b>Function prototype</b>	void timer_automatic_output_enable(uint32_t timer_periph);
<b>Function descriptions</b>	enable TIMER output automatic function
<b>Precondition</b>	This function can be called only when PROT [1:0] bit-filed in TIMERx_CCHP register is 00.
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>timer_periph</b>	TIMER peripheral
<i>TIMERx(x=0, 14..16)</i>	TIMER peripheral selection
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

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 3-424. Function timer\_automatic\_output\_disable**

<b>Function name</b>	timer_automatic_output_disable
<b>Function prototype</b>	void timer_automatic_output_disable (uint32_t timer_periph);
<b>Function descriptions</b>	disable TIMER output automatic function
<b>Precondition</b>	This function can be called only when PROT [1:0] bit-filed in TIMERx_CCHP register is 00.

<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>timer_periph</b>	TIMER peripheral
<b>TIMERx(x=0, 14..16)</b>	TIMER peripheral selection
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

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 3-425. Function timer\_primary\_output\_config**

<b>Function name</b>	timer_primary_output_config
<b>Function prototype</b>	void timer_primary_output_config(uint32_t timer_periph, ControlStatus newvalue);
<b>Function descriptions</b>	configure TIMER primary output function
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>timer_periph</b>	TIMER peripheral
<b>TIMERx(x=0, 14..16)</b>	TIMER peripheral selection
<b>Input parameter{in}</b>	
<b>newvalue</b>	control value
<b>ENABLE</b>	enable function
<b>DISABLE</b>	disable function
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

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 3-426. Function timer\_channel\_control\_shadow\_config**

<b>Function name</b>	timer_channel_control_shadow_config
<b>Function prototype</b>	void timer_channel_control_shadow_config(uint32_t timer_periph, ControlStatus newvalue);
<b>Function descriptions</b>	channel commutation control shadow register enable
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>timer_periph</b>	TIMER peripheral
<i>TIMERx(x=0, 14..16)</i>	TIMER peripheral selection
<b>Input parameter{in}</b>	
<b>newvalue</b>	control value
<i>ENABLE</i>	enable function
<i>DISABLE</i>	disable function
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

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 3-427. Function timer\_channel\_control\_shadow\_update\_config**

<b>Function name</b>	timer_channel_control_shadow_update_config
<b>Function prototype</b>	void timer_channel_control_shadow_update_config(uint32_t timer_periph, uint8_t ccuctl);
<b>Function descriptions</b>	configure commutation control shadow register update control
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>timer_periph</b>	TIMER peripheral
<i>TIMERx(x=0, 14..16)</i>	TIMER peripheral selection
<b>Input parameter{in}</b>	
<b>ccuctl</b>	channel control shadow register update control
<i>TIMER_UPDATECTL_C CU</i>	the shadow registers update by when CMTG bit is set
<i>TIMER_UPDATECTL_C 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 3-428. 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* ocpara);
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 <a href="#">Table 3-385. Structure timer_oc_parameter_struct</a> .
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 3-429. 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

<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>timer_periph</b>	TIMER peripheral
<b>TIMERx</b>	please refer to the following parameters
<b>Input parameter{in}</b>	
<b>channel</b>	channel to be configured
<b>TIMER_CH_0</b>	TIMER channel 0 (TIMERx(x=0, 2, 13..16))
<b>TIMER_CH_1</b>	TIMER channel 1 (TIMERx(x=0, 2, 14))
<b>TIMER_CH_2</b>	TIMER channel 2 (TIMERx(x=0, 2))
<b>TIMER_CH_3</b>	TIMER channel 3 (TIMERx(x=0, 2))
<b>Input parameter{in}</b>	
<b>ocpara</b>	TIMER channel output parameter struct, the structure members can refer to <a href="#">Table 3-385. Structure timer_oc_parameter_struct</a> .
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

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 3-430. Function `timer_channel_output_mode_config`**

<b>Function name</b>	timer_channel_output_mode_config
<b>Function prototype</b>	void timer_channel_output_mode_config(uint32_t timer_periph, uint16_t channel, uint16_t ocmode);
<b>Function descriptions</b>	configure TIMER channel output compare mode
<b>Precondition</b>	-

The called functions		-
Input parameter{in}		
<b>timer_periph</b>		TIMER peripheral
<b>TIMERx</b>		please refer to the following parameters
Input parameter{in}		
<b>channel</b>		channel to be configured
<b>TIMER_CH_0</b>		TIMER channel 0 (TIMERx (x=0, 2, 13..16))
<b>TIMER_CH_1</b>		TIMER channel 1 (TIMERx (x=0, 2, 14))
<b>TIMER_CH_2</b>		TIMER channel 2 (TIMERx (x=0, 2))
<b>TIMER_CH_3</b>		TIMER channel 3 (TIMERx (x=0, 2))
Input parameter{in}		
<b>ocmode</b>		channel output compare mode
<b>TIMER_OC_MODE_TIMING</b>		timing mode
<b>TIMER_OC_MODE_ACITIVE</b>		set the channel output
<b>TIMER_OC_MODE_INACTIVE</b>		clear the channel output
<b>TIMER_OC_MODE_TOGGLEGGLE</b>		toggle on match
<b>TIMER_OC_MODE_FORCELOW</b>		force low mode
<b>TIMER_OC_MODE_FORCEHIGH</b>		force high mode
<b>TIMER_OC_MODE_PWM0</b>		PWM mode 0
<b>TIMER_OC_MODE_PWM1</b>		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 3-431. Function timer\_channel\_output\_pulse\_value\_config**

Function name	timer_channel_output_pulse_value_config

<b>Function prototype</b>	void timer_channel_output_pulse_value_config(uint32_t timer_periph, uint16_t channel, uint32_t pulse);
<b>Function descriptions</b>	configure TIMER channel output pulse value
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>timer_periph</b>	TIMER peripheral
<i>TIMERx</i>	please refer to the following parameters
<b>Input parameter{in}</b>	
<b>channel</b>	channel to be configured
<i>TIMER_CH_0</i>	TIMER channel 0 (TIMERx (x=0, 2, 13..16))
<i>TIMER_CH_1</i>	TIMER channel 1 (TIMERx TIMERx(x=0, 2, 14))
<i>TIMER_CH_2</i>	TIMER channel 2 (TIMERx (x=0, 2))
<i>TIMER_CH_3</i>	TIMER channel 3 (TIMERx (x=0, 2))
<b>Input parameter{in}</b>	
<b>pulse</b>	channel output pulse value (0~65535)
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

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 3-432. Function `timer_channel_output_shadow_config`**

<b>Function name</b>	timer_channel_output_shadow_config
<b>Function prototype</b>	void timer_channel_output_shadow_config(uint32_t timer_periph, uint16_t channel, uint16_t ocshadow);
<b>Function descriptions</b>	configure TIMER channel output shadow function
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>timer_periph</b>	TIMER peripheral
<i>TIMERx</i>	please refer to the following parameters
<b>Input parameter{in}</b>	
<b>channel</b>	channel to be configured
<i>TIMER_CH_0</i>	TIMER channel 0 (TIMERx (x=0, 2, 13..16))
<i>TIMER_CH_1</i>	TIMER channel 1 (TIMERx (x=0, 2, 14))

<i>TIMER_CH_2</i>	TIMER channel 2 (TIMERx (x=0, 2))
<i>TIMER_CH_3</i>	TIMER channel 3 (TIMERx (x=0, 2))
<b>Input parameter{in}</b>	
<i>ocshadow</i>	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
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

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 3-433. Function timer\_channel\_output\_fast\_config**

<b>Function name</b>	timer_channel_output_fast_config
<b>Function prototype</b>	void timer_channel_output_fast_config(uint32_t timer_periph, uint16_t channel, uint16_t ocfast);
<b>Function descriptions</b>	configure TIMER channel output fast function
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<i>timer_periph</i>	TIMER peripheral
<i>TIMERx</i>	please refer to the following parameters
<b>Input parameter{in}</b>	
<i>channel</i>	channel to be configured
<i>TIMER_CH_0</i>	TIMER channel 0 (TIMERx (x=0, 2, 13..16))
<i>TIMER_CH_1</i>	TIMER channel 1 (TIMERx (x=0, 2, 14))
<i>TIMER_CH_2</i>	TIMER channel 2 (TIMERx (x=0, 2))
<i>TIMER_CH_3</i>	TIMER channel 3 (TIMERx (x=0, 2))
<b>Input parameter{in}</b>	
<i>ocfast</i>	channel output fast function
<i>TIMER_OC_FAST_ENA_BLE</i>	channel output fast function enable
<i>TIMER_OC_FAST_DIS</i>	channel output fast function disable

<i>ABLE</i>	
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

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 3-434. Function timer\_channel\_output\_clear\_config**

<b>Function name</b>	timer_channel_output_clear_config
<b>Function prototype</b>	void timer_channel_output_clear_config(uint32_t timer_periph, uint16_t channel, uint16_t occlear);
<b>Function descriptions</b>	configure TIMER channel output clear function
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<i>timer_periph</i>	TIMER periphera
<i>TIMERx</i>	please refer to the following parameters
<b>Input parameter{in}</b>	
<i>channel</i>	channel to be configured
<i>TIMER_CH_0</i>	TIMER channel 0 (TIMERx (x=0, 2))
<i>TIMER_CH_1</i>	TIMER channel 1 (TIMERx (x=0, 2))
<i>TIMER_CH_2</i>	TIMER channel 2 (TIMERx (x=0, 2))
<i>TIMER_CH_3</i>	TIMER channel 3 (TIMERx (x=0, 2))
<b>Input parameter{in}</b>	
<i>occlear</i>	channel output clear function
<i>TIMER_OC_CLEAR_EN_ABLE</i>	channel output clear function enable
<i>TIMER_OC_CLEAR_DISABLE</i>	channel output clear function disable
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

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 3-435. Function timer\_channel\_output\_polarity\_config**

<b>Function name</b>	timer_channel_output_polarity_config
<b>Function prototype</b>	void timer_channel_output_polarity_config(uint32_t timer_periph, uint16_t channel, uint16_t ocpolarity);
<b>Function descriptions</b>	configure TIMER channel output polarity
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>timer_periph</b>	TIMER peripheral
<b>TIMERx</b>	please refer to the following parameters
<b>Input parameter{in}</b>	
<b>channel</b>	channel to be configured
<b>TIMER_CH_0</b>	TIMER channel 0 (TIMERx (x=0, 2, 13..16))
<b>TIMER_CH_1</b>	TIMER channel 1 (TIMERx (x=0, 2, 14))
<b>TIMER_CH_2</b>	TIMER channel 2 (TIMERx(x=0, 2))
<b>TIMER_CH_3</b>	TIMER channel 3 (TIMERx (x=0, 2))
<b>Input parameter{in}</b>	
<b>ocpolarity</b>	channel output polarity
<b>TIMER_OC_POLARITY_HIGH</b>	channel output polarity is high
<b>TIMER_OC_POLARITY_LOW</b>	channel output polarity is low
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

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 3-436. Function timer\_channel\_complementary\_output\_polarity\_config**

<b>Function name</b>	timer_channel_complementary_output_polarity_config
<b>Function prototype</b>	void timer_channel_complementary_output_polarity_config(uint32_t timer_periph, uint16_t channel, uint16_t ocpolarity);
<b>Function descriptions</b>	configure TIMER channel complementary output polarity
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
timer_periph	TIMER peripheral
TIMERx	TIMER peripheral selection
<b>Input parameter{in}</b>	
channel	channel to be configured
TIMER_CH_0	TIMER channel 0(TIMERx (x=0, 2, 13..16))
TIMER_CH_1	TIMER channel 1(TIMERx (x=0, 2, 14))
TIMER_CH_2	TIMER channel 2(TIMERx (x=0, 2))
TIMER_CH_3	TIMER channel 3(TIMERx (x=0, 2))
<b>Input parameter{in}</b>	
ocpolarity	channel complementary output polarity
TIMER_OCN_POLARITY_HIGH	channel complementary output polarity is high
TIMER_OCN_POLARITY_LOW	channel complementary output polarity is low
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* configure TIMER0 channel 0 complementary output polarity */

timer_channel_complementary_output_polarity_config (TIMER0, TIMER_CH_0,
TICKER_OCN_POLARITY_HIGH);
```

### **timer\_channel\_output\_state\_config**

The description of timer\_channel\_output\_state\_config is shown as below:

**Table 3-437. Function timer\_channel\_output\_state\_config**

<b>Function name</b>	timer_channel_output_state_config
<b>Function prototype</b>	void timer_channel_output_state_config(uint32_t timer_periph, uint16_t channel, uint32_t state);
<b>Function descriptions</b>	configure TIMER channel enable state
<b>Precondition</b>	-
<b>The called functions</b>	-

Input parameter{in}	
<b>timer_periph</b>	TIMER peripheral
<i>TIMERx</i>	please refer to the following parameters
Input parameter{in}	
<b>channel</b>	channel to be configured
<i>TIMER_CH_0</i>	TIMER channel 0( <i>TIMERx</i> (x=0, 2, 13..16))
<i>TIMER_CH_1</i>	TIMER channel 1( <i>TIMERx</i> (x=0, 2, 14))
<i>TIMER_CH_2</i>	TIMER channel 2( <i>TIMERx</i> (x=0, 2))
<i>TIMER_CH_3</i>	TIMER channel 3( <i>TIMERx</i> (x=0, 2))
Input parameter{in}	
<b>state</b>	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 3-438. Function timer\_channel\_complementary\_output\_state\_config**

<b>Function name</b>	timer_channel_complementary_output_state_config
<b>Function prototype</b>	void timer_channel_complementary_output_state_config(uint32_t timer_periph, uint16_t channel, uint16_t ocnstate);
<b>Function descriptions</b>	configure TIMER channel complementary output enable state
<b>Precondition</b>	-
<b>The called functions</b>	-
Input parameter{in}	
<b>timer_periph</b>	TIMER peripheral
<i>TIMERx(x=0, 14..16)</i>	TIMER peripheral selection
Input parameter{in}	
<b>channel</b>	channel to be configured
<i>TIMER_CH_0</i>	TIMER channel 0, <i>TIMERx</i> (x=0, 14..16)
<i>TIMER_CH_1</i>	TIMER channel 1, <i>TIMERx</i> (x=0)
<i>TIMER_CH_2</i>	TIMER channel 2, <i>TIMERx</i> (x=0)
Input parameter{in}	
<b>state</b>	TIMER channel complementary output enable state

<b>TIMER_CCXN_ENABLE</b>	channel complementary enable
<b>TIMER_CCXN_DISABLE</b>	channel complementary disable
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

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 3-439. Function timer\_channel\_input\_struct\_para\_init**

<b>Function name</b>	timer_channel_input_struct_para_init
<b>Function prototype</b>	void timer_channel_input_struct_para_init(timer_ic_parameter_struct* icpara);
<b>Function descriptions</b>	initialize the parameters of TIMER channel input parameter struct with the default values
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>icpara</b>	TIMER channel input parameter struct, the structure members can refer to <a href="#">Table 3-386. Structure timer_ic_parameter_struct</a> .
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

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 3-440. Function timer\_input\_capture\_config**

<b>Function name</b>	timer_input_capture_config
<b>Function prototype</b>	void timer_input_capture_config(uint32_t timer_periph, uint16_t channel, timer_ic_parameter_struct* icpara);
<b>Function descriptions</b>	configure TIMER input capture parameter
<b>Precondition</b>	-
<b>The called functions</b>	timer_channel_input_capture_prescaler_config
<b>Input parameter{in}</b>	
timer_periph	TIMER peripheral
TIMERx	please refer to the following parameters
<b>Input parameter{in}</b>	
channel	channel to be configured
TIMER_CH_0	TIMER channel 0(TIMERx (x=0, 2, 13..16))
TIMER_CH_1	TIMER channel 1(TIMERx (x=0, 2, 14))
TIMER_CH_2	TIMER channel 2(TIMERx (x=0, 2))
TIMER_CH_3	TIMER channel 3(TIMERx (x=0, 2))
<b>Input parameter{in}</b>	
icpara	TIMER channel input parameter struct, the structure members can refer to <a href="#">Table 3-386. Structure timer_ic_parameter_struct</a> .
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

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_DIRECTTI;
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 3-441. Function timer\_channel\_input\_capture\_prescaler\_config**

<b>Function name</b>	timer_channel_input_capture_prescaler_config
<b>Function prototype</b>	void timer_channel_input_capture_prescaler_config(uint32_t timer_periph, uint16_t channel, uint16_t prescaler);

<b>Function descriptions</b>	configure TIMER channel input capture prescaler value
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>timer_periph</b>	TIMER peripheral
<i>TIMERx</i>	please refer to the following parameters
<b>Input parameter{in}</b>	
<b>channel</b>	channel to be configured
<i>TIMER_CH_0</i>	TIMER channel 0(TIMERx (x=0, 2, 13..16))
<i>TIMER_CH_1</i>	TIMER channel 1(TIMERx (x=0, 2, 14))
<i>TIMER_CH_2</i>	TIMER channel 2(TIMERx (x=0, 2))
<i>TIMER_CH_3</i>	TIMER channel 3(TIMERx (x=0, 2))
<b>Input parameter{in}</b>	
<b>prescaler</b>	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
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* configure TIMER0 channel 0 input capture prescaler value */

timer_channel_input_capture_prescaler_config (TIMER0, TIMER_CH_0,
TICKER_IC_PSC_DIV2);
```

### **timer\_channel\_capture\_value\_register\_read**

The description of `timer_channel_capture_value_register_read` is shown as below:

**Table 3-442. Function `timer_channel_capture_value_register_read`**

<b>Function name</b>	timer_channel_capture_value_register_read
<b>Function prototype</b>	uint32_t timer_channel_capture_value_register_read(uint32_t timer_periph, uint16_t channel);
<b>Function descriptions</b>	read TIMER channel capture compare register value
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>timer_periph</b>	TIMER peripheral
<i>TIMERx</i>	please refer to the following parameters
<b>Input parameter{in}</b>	

<b>channel</b>	channel to be configured
<i>TIMER_CH_0</i>	TIMER channel 0(TIMERx (x=0, 2, 13..16))
<i>TIMER_CH_1</i>	TIMER channel 1(TIMERx (x=0, 2, 14))
<i>TIMER_CH_2</i>	TIMER channel 2(TIMERx (x=0, 2))
<i>TIMER_CH_3</i>	TIMER channel 3(TIMERx (x=0, 2))
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
<b>uint32_t</b>	channel capture compare register value (0~65535)

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 3-443. Function `timer_input_pwm_capture_config`**

<b>Function name</b>	<code>timer_input_pwm_capture_config</code>
<b>Function prototype</b>	<code>void timer_input_pwm_capture_config(uint32_t timer_periph, uint16_t channel, timer_ic_parameter_struct* icpwm);</code>
<b>Function descriptions</b>	configure TIMER input pwm capture function
<b>Precondition</b>	-
<b>The called functions</b>	<code>timer_channel_input_capture_prescaler_config</code>
<b>Input parameter{in}</b>	
<b>timer_periph</b>	TIMER peripheral
<i>TIMERx(x=0, 2, 14)</i>	TIMER peripheral selection
<b>Input parameter{in}</b>	
<b>channel</b>	channel to be configured
<i>TIMER_CH_0</i>	TIMER channel 0
<i>TIMER_CH_1</i>	TIMER channel 1
<b>Input parameter{in}</b>	
<b>icpwm</b>	TIMER channel input pwm parameter struct, the structure members can refer to <a href="#">Table 3-386. Structure <code>timer_ic_parameter_struct</code>.</a>
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

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_DIRECTTI;
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 3-444. Function `timer_hall_mode_config`**

<b>Function name</b>	timer_hall_mode_config
<b>Function prototype</b>	void timer_hall_mode_config(uint32_t timer_periph, uint8_t hallmode);
<b>Function descriptions</b>	configure TIMER hall sensor mode
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>timer_periph</b>	TIMER peripheral
<i>TIMERx(x=0, 2)</i>	TIMER peripheral selection
<b>Input parameter{in}</b>	
<b>hallmode</b>	TIMER hall sensor mode state
<i>TIMER_HALLINTERFACE_CE_ENABLE</i>	TIMER hall sensor mode enable
<i>TIMER_HALLINTERFACE_CE_DISABLE</i>	TIMER hall sensor mode disable
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

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 3-445. Function `timer_input_trigger_source_select`**

<b>Function name</b>	timer_input_trigger_source_select
<b>Function prototype</b>	void timer_input_trigger_source_select(uint32_t timer_periph, uint32_t

	intrigger);
<b>Function descriptions</b>	select TIMER input trigger source
<b>Precondition</b>	SMC[2:0] = 000
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>timer_periph</b>	TIMER peripheral
<i>TIMERx(x=0, 2, 14)</i>	please refer to the following parameters
<b>Input parameter{in}</b>	
<b>intrigger</b>	trigger selection
<i>TIMER_SMCFG_TRGS_EL_ITI0</i>	Internal trigger input 0(ITI0, TIMERx(x=0, 2, 14))
<i>TIMER_SMCFG_TRGS_EL_ITI1</i>	Internal trigger input 0 (ITI1, TIMERx(x=0, 2, 14))
<i>TIMER_SMCFG_TRGS_EL_ITI2</i>	Internal trigger input 0 (ITI2, TIMERx(ITI2, TIMERx(x=0, 2)))
<i>TIMER_SMCFG_TRGS_EL_ITI3</i>	Internal trigger input 0(ITI3, TIMERx(x=0, 2, 14))
<i>TIMER_SMCFG_TRGS_EL_CI0F_ED</i>	CI0 edge flag (CI0F_ED, TIMERx(x=0, 2, 14))
<i>TIMER_SMCFG_TRGS_EL_CI0FE0</i>	channel 0 input Filtered output(CI0FE0, TIMERx(x=0, 2, 14))
<i>TIMER_SMCFG_TRGS_EL_CI1FE1</i>	channel 1 input Filtered output(CI1FE1, TIMERx(x=0, 2, 14))
<i>TIMER_SMCFG_TRGS_EL_ETIFP</i>	External trigger input filter output(ETIFP, TIMERx(x=0, 2))
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

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 3-446. Function timer\_master\_output\_trigger\_source\_select**

<b>Function name</b>	timer_master_output_trigger_source_select
<b>Function prototype</b>	void timer_master_output_trigger_source_select(uint32_t timer_periph, uint32_t outrigger);
<b>Function descriptions</b>	select TIMER master mode output trigger source

<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>timer_periph</b>	TIMER peripheral
<i>TIMERx(x=0, 2, 5, 14)</i>	TIMER peripheral selection
<b>Input parameter{in}</b>	
<b>outrigger</b>	master mode control
<i>TIMER_TRI_OUT_SRC_RESET</i>	Reset. When the UPG bit in the TIMERx_SWEVG register is set or a reset is generated by the slave mode controller, a TRGO pulse occurs. And in the latter case, the signal on TRGO is delayed compared to the actual reset
<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_CH0</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.
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

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 3-447. Function timer\_slave\_mode\_select**

<b>Function name</b>	timer_slave_mode_select
<b>Function prototype</b>	void timer_slave_mode_select(uint32_t timer_periph, uint32_t slavemode);
<b>Function descriptions</b>	select TIMER slave mode
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>timer_periph</b>	TIMER peripheral
<i>TIMERx(x=0, 2, 14)</i>	TIMER peripheral selection
<b>Input parameter{in}</b>	
<b>slavemode</b>	slave mode
<i>TIMER_SLAVE_MODE_DISABLE</i>	slave mode disable, <i>TIMERx(x=0, 2, 14)</i>
<i>TIMER_ENCODER_MODE0</i>	encoder mode 0, <i>TIMERx(x=0, 2)</i>
<i>TIMER_ENCODER_MODE1</i>	encoder mode 1, <i>TIMERx(x=0, 2)</i>
<i>TIMER_ENCODER_MODE2</i>	encoder mode 2, <i>TIMERx(x=0, 2)</i>
<i>TIMER_SLAVE_MODE_RESTART</i>	restart mode, <i>TIMERx(x=0, 2, 14)</i>
<i>TIMER_SLAVE_MODE_PAUSE</i>	pause mode, <i>TIMERx(x=0, 2, 14)</i>
<i>TIMER_SLAVE_MODE_EVENT</i>	event mode, <i>TIMERx(x=0, 2, 14)</i>
<i>TIMER_SLAVE_MODE_EXTERNAL0</i>	external clock mode 0, <i>TIMERx(x=0, 2, 14)</i>
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* select TIMER0 slave mode */

timer_slave_mode_select (TIMER0, TIMER_ENCODER_MODE0);
```

### **timer\_master\_slave\_mode\_config**

The description of **timer\_master\_slave\_mode\_config** is shown as below:

**Table 3-448. Function timer\_master\_slave\_mode\_config**

<b>Function name</b>	timer_master_slave_mode_config
<b>Function prototype</b>	void timer_master_slave_mode_config(uint32_t timer_periph, uint8_t

	masterslave);
<b>Function descriptions</b>	configure TIMER master slave mode
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>timer_periph</b>	TIMER peripheral
<b>TIMERx(x=0, 2, 14)</b>	TIMER peripheral selection
<b>Input parameter{in}</b>	
<b>masterslave</b>	master slave mode state
<b>TIMER_MASTER_SLAVE_MODE_ENABLE</b>	master slave mode enable
<b>TIMER_MASTER_SLAVE_MODE_DISABLE</b>	master slave mode disable
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

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 3-449. Function timer\_external\_trigger\_config**

<b>Function name</b>	timer_external_trigger_config
<b>Function prototype</b>	void timer_external_trigger_config(uint32_t timer_periph, uint32_t extprescaler, uint32_t expolarity, uint32_t extfilter);
<b>Function descriptions</b>	configure TIMER external trigger input
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>timer_periph</b>	TIMER peripheral
<b>TIMERx(x=0, 2)</b>	TIMER peripheral selection
<b>Input parameter{in}</b>	
<b>extprescaler</b>	external trigger prescaler
<b>TIMER_EXT_TRI_PSC_OFF</b>	no divided
<b>TIMER_EXT_TRI_PSC_DIV2</b>	divided by 2
<b>TIMER_EXT_TRI_PSC_</b>	divided by 4

<i>DIV4</i>	
<i>TIMER_EXT_TRI_PSC_DIV8</i>	divided by 8
<b>Input parameter{in}</b>	
<b>expolarity</b>	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
<b>Input parameter{in}</b>	
<b>extfilter</b>	external trigger filter control (0~15)
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

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 3-450. Function timer\_quadrature\_decoder\_mode\_config**

<b>Function name</b>	timer_quadrature_decoder_mode_config
<b>Function prototype</b>	void timer_quadrature_decoder_mode_config(uint32_t timer_periph, uint32_t decomode, uint16_t ic0polarity, uint16_t ic1polarity);
<b>Function descriptions</b>	configure TIMER quadrature decoder mode
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>timer_periph</b>	TIMER peripheral
<i>TIMERx(x=0, 2)</i>	TIMER peripheral selection
<b>Input parameter{in}</b>	
<b>decomode</b>	quadrature decoder mode
<i>TIMER_ENCODER_MO_DE0</i>	counter counts on CI0FE0 edge depending on CI1FE1 level
<i>TIMER_ENCODER_MO_DE1</i>	counter counts on CI1FE1 edge depending on CI0FE0 level
<i>TIMER_ENCODER_MO_DE2</i>	counter counts on both CI0FE0 and CI1FE1 edges depending on the level of the other input
<b>Input parameter{in}</b>	
<b>ic0polarity</b>	IC0 polarity

<i>TIMER_IC_POLARITY_RISING</i>	capture rising edge
<i>TIMER_IC_POLARITY_FALLING</i>	capture falling edge
<b>Input parameter{in}</b>	
<b>ic1polarity</b>	IC1 polarity
<i>TIMER_IC_POLARITY_RISING</i>	capture rising edge
<i>TIMER_IC_POLARITY_FALLING</i>	capture falling edge
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* configure TIMER0 quadrature decoder mode */
timer_quadrature_decoder_mode_config (TIMER0, TIMER_ENCODER_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 3-451. Function timer\_internal\_clock\_config**

<b>Function name</b>	timer_internal_clock_config
<b>Function prototype</b>	void timer_internal_clock_config(uint32_t timer_periph);
<b>Function descriptions</b>	configure TIMER internal clock mode
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>timer_periph</b>	TIMER peripheral
<i>TIMERx(x=0, 2, 14)</i>	TIMER peripheral selection
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

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 3-452. Function timer\_internal\_trigger\_as\_external\_clock\_config**

<b>Function name</b>	timer_internal_trigger_as_external_clock_config
<b>Function prototype</b>	void timer_internal_trigger_as_external_clock_config(uint32_t timer_periph, uint32_t intrigger);
<b>Function descriptions</b>	configure TIMER the internal trigger as external clock input
<b>Precondition</b>	-
<b>The called functions</b>	timer_input_trigger_source_select
<b>Input parameter{in}</b>	
<b>timer_periph</b>	TIMER peripheral
<i>TIMERx(x=0, 2, 14)</i>	TIMER peripheral selection
<b>Input parameter{in}</b>	
<b>intrigger</b>	trigger selection
<i>TIMER_SMCFG_TRGS EL_ITI0</i>	Internal trigger input 0 (ITI0), <i>TIMERx(x=0, 2, 14)</i>
<i>TIMER_SMCFG_TRGS EL_ITI1</i>	Internal trigger input 0 (ITI1) , <i>TIMERx(x=0, 2, 14)</i>
<i>TIMER_SMCFG_TRGS EL_ITI2</i>	Internal trigger input 0 (ITI2) , <i>TIMERx(x=0, 2)</i>
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

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 3-453. Function timer\_external\_trigger\_as\_external\_clock\_config**

<b>Function name</b>	timer_external_trigger_as_external_clock_config
<b>Function prototype</b>	void timer_external_trigger_as_external_clock_config(uint32_t timer_periph, uint32_t exttrigger, uint16_t expolarity, uint32_t extfilter);
<b>Function descriptions</b>	configure TIMER the external trigger as external clock input
<b>Precondition</b>	-
<b>The called functions</b>	timer_input_trigger_source_select
<b>Input parameter{in}</b>	
<b>timer_periph</b>	TIMER peripheral

<i>TIMERx(x=0, 2, 14)</i>	TIMER peripheral selection
<b>Input parameter{in}</b>	
<b>extrigger</b>	external trigger selection
<i>TIMER_SMCFG_TRGS_EL_CI0F_ED</i>	CI0 edge flag (CI0F_ED)
<i>TIMER_SMCFG_TRGS_EL_CI0FE0</i>	channel 0 input Filtered output (CI0FE0)
<i>TIMER_SMCFG_TRGS_EL_CI1FE1</i>	channel 1 input Filtered output (CI1FE1)
<b>Input parameter{in}</b>	
<b>expolarity</b>	external trigger polarity
<i>TIMER_IC_POLARITY_RISING</i>	active high or rising edge active
<i>TIMER_IC_POLARITY_FALLING</i>	active low or falling edge active
<i>TIMER_IC_POLARITY_BOTH_EDGE</i>	falling edge or rising edge active
<b>Input parameter{in}</b>	
<b>extfilter</b>	external trigger filter control (0~15)
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* configure TIMER0 the external trigger CI0FE0 as external clock input */
timer_external_trigger_as_external_clock_config(TIMER0,
    TIMER_SMCFG_TRGSEL_CI0FE0, TIMER_IC_POLARITY_RISING, 0);
```

### **timer\_external\_clock\_mode0\_config**

The description of timer\_external\_clock\_mode0\_config is shown as below:

**Table 3-454. Function timer\_external\_clock\_mode0\_config**

<b>Function name</b>	timer_external_clock_mode0_config
<b>Function prototype</b>	void timer_external_clock_mode0_config(uint32_t timer_periph, uint32_t extprescaler, uint32_t expolarity, uint32_t extfilter);
<b>Function descriptions</b>	configure TIMER the external clock mode0
<b>Precondition</b>	-
<b>The called functions</b>	timer_external_trigger_config
<b>Input parameter{in}</b>	
<b>timer_periph</b>	TIMER peripheral
<i>TIMERx(x=0, 2)</i>	TIMER peripheral selection

Input parameter{in}	
<b>extprescaler</b>	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}	
<b>expolarity</b>	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}	
<b>extfilter</b>	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 3-455. Function timer\_external\_clock\_mode1\_config**

<b>Function name</b>	timer_external_clock_mode1_config
<b>Function prototype</b>	void timer_external_clock_mode1_config(uint32_t timer_periph, uint32_t extprescaler, uint32_t expolarity, uint32_t extfilter);
<b>Function descriptions</b>	configure TIMER the external clock mode1
<b>Precondition</b>	-
<b>The called functions</b>	timer_external_trigger_config
Input parameter{in}	
<b>timer_periph</b>	TIMER peripheral
<i>TIMERx(x=0, 2)</i>	TIMER peripheral selection
Input parameter{in}	
<b>extprescaler</b>	ETI external trigger prescaler
<i>TIMER_EXT_TRI_PSC_</i>	no divided

<b>OFF</b>	
<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
<b>Input parameter{in}</b>	
<b>expolarity</b>	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
<b>Input parameter{in}</b>	
<b>extfilter</b>	ETI external trigger filter control (0~15)
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

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 3-456. Function timer\_external\_clock\_mode1\_disable**

<b>Function name</b>	timer_external_clock_mode1_disable
<b>Function prototype</b>	void timer_external_clock_mode1_disable(uint32_t timer_periph);
<b>Function descriptions</b>	disable TIMER the external clock mode1
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>timer_periph</b>	TIMER peripheral
<i>TIMERx(x=0, 2)</i>	TIMER peripheral selection
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

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 3-457. Function timer\_channel\_remap\_config**

<b>Function name</b>	timer_channel_remap_config
<b>Function prototype</b>	void timer_channel_remap_config (uint32_t timer_periph, uint32_t remap);
<b>Function descriptions</b>	configure TIMER channel remap function
<b>Precondition</b>	-
<b>The called functions</b>	timer_external_trigger_config
<b>Input parameter{in}</b>	
timer_periph	TIMER peripheral
TIMERx(x=13)	TIMER peripheral selection
<b>Input parameter{in}</b>	
remap	remap function selection
TIMER13_CI0_RMP_GPIO	timer13 channel 0 input is connected to GPIO(TIMER13_CH0)
TIMER13_CI0_RMP_RTCCLK	timer13 channel 0 input is connected to the RTCCLK
TIMER13_CI0_RMP_HXTAL_DIV32	timer13 channel 0 input is connected to HXTAL/32 clock
TIMER13_CI0_RMP_CKOUTSEL	timer13 channel 0 input is connected to CKOUTSEL
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* configure TIMER13 channel 0 input is connected to GPIO */
timer_channel_remap_config (TIMER13, TIMER13_CI0_RMP_GPIO);
```

### **timer\_write\_chxval\_register\_config**

The description of timer\_write\_chxval\_register\_config is shown as below:

**Table 3-458. Function timer\_write\_chxval\_register\_config**

<b>Function name</b>	timer_write_chxval_register_config
<b>Function prototype</b>	void timer_write_chxval_register_config(uint32_t timer_periph, uint16_t ccsel);
<b>Function descriptions</b>	configure TIMER write CHxVAL register selection
<b>Precondition</b>	-

<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>timer_periph</b>	TIMER peripheral
<i>TIMERx(x=0, 2, 13..16)</i>	TIMER peripheral selection
<b>Input parameter{in}</b>	
<b>ccsel</b>	write CHxVAL register selection
<i>TIMER_CHVSEL_DISA</i> <i>BLE</i>	no effect
<i>TIMER_CHVSEL_ENAB</i> <i>LE</i>	when write the CHxVAL register, if the write value is same as the CHxVAL value, the write access is ignored
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

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 3-459. Function timer\_output\_value\_selection\_config**

<b>Function name</b>	timer_output_value_selection_config
<b>Function prototype</b>	void timer_output_value_selection_config(uint32_t timer_periph, uint16_t outsel);
<b>Function descriptions</b>	configure TIMER output value selection
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>timer_periph</b>	TIMER peripheral
<i>TIMERx(x=0, 14..16)</i>	TIMER peripheral selection
<b>Input parameter{in}</b>	
<b>outsel</b>	output value selection
<i>TIMER_OUTSEL_DISA</i> <i>BLE</i>	no effect
<i>TIMER_OUTSEL_ENAB</i> <i>LE</i>	if POEN and IOS is 0, the output disabled
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* configure TIMER output value selection */

timer_output_value_selection_config(TIMER0, TIMER_OUTSEL_ENABLE);
```

## 3.19. USART

The Universal Synchronous/Asynchronous Receiver/Transmitter (USART) provides a flexible serial data exchange interface. The USART registers are listed in chapter [3.19.1](#), the USART firmware functions are introduced in chapter [3.19.2](#).

### 3.19.1. Descriptions of Peripheral registers

USART registers are listed in the table shown as below:

**Table 3-460. USART Registers**

Registers	Descriptions
USART_CTL0	Control register 0
USART_CTL1	Control register 1
USART_CTL2	Control register 2
USART_BAUD	Baud rate register
USART_GP	Guard time and prescaler register
USART_RT	Receiver timeout register
USART_CMD	Command register
USART_STAT	Status register
USART_INTC	Status clear register
USART_RDATA	Receive data register
USART_TDATA	Transmit data register
USART_CHC	Coherence control register
USART_RFCS	Receive FIFO control and status register

### 3.19.2. Descriptions of Peripheral functions

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

**Table 3-461. USART firmware function**

Function name	Function description
uart_deinit	reset USART
uart_baudrate_set	configure USART baud rate value
uart_parity_config	configure USART parity function
uart_word_length_set	configure USART word length
uart_stop_bit_set	configure USART stop bit length
uart_enable	enable USART

<b>Function name</b>	<b>Function description</b>
uart_disable	disable USART
uart_transmit_config	configure USART transmitter
uart_receive_config	configure USART receiver
uart_data_first_config	data is transmitted/received with the LSB/MSB first
uart_invert_config	configure USART inverted
uart_overrun_enable	enable the USART overrun function
uart_overrun_disable	disable the USART overrun function
uart_oversample_config	configure the USART oversample mode
uart_sample_bit_config	configure sample bit method
uart_receiver_timeout_enable	enable receiver timeout
uart_receiver_timeout_disable	disable receiver timeout
uart_receiver_timeout_threshold_config	configure receiver timeout threshold
uart_data_transmit	USART transmit data function
uart_data_receive	USART receive data function
uart_autobaud_detection_enable	enable auto baud rate detection
uart_autobaud_detection_disable	disable auto baud rate detection
uart_autobaud_detection_mode_config	configure auto baud rate detection mode
uart_address_config	configure the address of the USART in wake up by address match mode
uart_address_detection_mode_config	configure address detection mode
uart_mute_mode_enable	enable mute mode
uart_mute_mode_disable	disable mute mode
uart_mute_mode_wakeup_config	configure wakeup method in mute mode
uart_lin_mode_enable	enable LIN mode
uart_lin_mode_disable	disable LIN mode
uart_lin_break_dection_length_config	LIN break detection length
uart_halfduplex_enable	enable half duplex mode
uart_halfduplex_disable	disable half duplex mode
uart_clock_enable	enable clock
uart_clock_disable	disable clock
uart_synchronous_clock_config	configure USART synchronous mode parameters
uart_guard_time_config	configure guard time value in smartcard mode
uart_smartcard_mode_enable	enable smartcard mode
uart_smartcard_mode_disable	disable smartcard mode
uart_smartcard_mode_nack_enable	enable NACK in smartcard mode
uart_smartcard_mode_nack_disable	disable NACK in smartcard mode
uart_smartcard_mode_early_nack_e	enable early NACK in smartcard mode

<b>Function name</b>	<b>Function description</b>
enable	
uart_smartcard_mode_early_nack_disable	disable early NACK in smartcard mode
uart_smartcard_autoretry_config	configure smartcard auto-retry number
uart_block_length_config	configure block length
uart_irda_mode_enable	enable IrDA mode
uart_irda_mode_disable	disable IrDA mode
uart_prescaler_config	configure the peripheral clock prescaler in USART IrDA low-power mode
uart_irda_lowpower_config	configure IrDA low-power
uart_hardware_flow_rts_config	configure hardware flow control RTS
uart_hardware_flow_cts_config	configure hardware flow control CTS
uart_hardware_flow_coherence_config	configure hardware flow control coherence mode
uart_rs485_driver_enable	enable RS485 driver
uart_rs485_driver_disable	disable RS485 driver
uart_driver_assertime_config	configure driver enable assertion time
uart_driver_deassertime_config	configure driver enable de-assertion time
uart_depolarity_config	configure driver enable polarity mode
uart_dma_receive_config	configure USART DMA for reception
uart_dma_transmit_config	configure USART DMA for transmission
uart_reception_error_dma_disable	disable DMA on reception error
uart_reception_error_dma_enable	enable DMA on reception error
uart_wakeup_enable	USART be able to wake up the mcu from deep-sleep mode
uart_wakeup_disable	USART be not able to wake up the mcu from deep-sleep mode
uart_wakeup_mode_config	wakeup mode from deep-sleep mode
uart_receive_fifo_enable	enable receive FIFO
uart_receive_fifo_disable	disable receive FIFO
uart_receive_fifo_counter_number	read receive FIFO counter number
uart_flag_get	get flag in STAT/RFCS register
uart_flag_clear	clear USART status
uart_interrupt_enable	enable USART interrupt
uart_interrupt_disable	disable USART interrupt
uart_command_enable	enable USART command
uart_interrupt_flag_get	get USART interrupt and flag status
uart_interrupt_flag_clear	clear USART interrupt flag

### **Enum usart\_flag\_enum**

**Table 3-462. Enum usart\_flag\_enum**

<b>Member name</b>	<b>Function description</b>
USART_FLAG_REA	receive enable acknowledge flag
USART_FLAG_TEA	transmit enable acknowledge flag
USART_FLAG_WU	wakeup from Deep-sleep mode flag
USART_FLAG_RWU	receiver wakeup from mute mode
USART_FLAG_SB	send break flag
USART_FLAG_AM	ADDR match flag
USART_FLAG_BSY	busy flag
USART_FLAG_ABD	auto baudrate detection flag
USART_FLAG_ABDE	auto baudrate detection error
USART_FLAG_EB	end of block flag
USART_FLAG_RT	receiver timeout flag
USART_FLAG_CTS	CTS level
USART_FLAG_CTSF	CTS change flag
USART_FLAG_LBD	LIN break detected flag
USART_FLAG_TBE	transmit data buffer empty
USART_FLAG_TC	transmission complete
USART_FLAG_RBNE	read data buffer not empty
USART_FLAG_IDLE	IDLE line detected flag
USART_FLAG_ORERR	overrun error
USART_FLAG_NERR	noise error flag
USART_FLAG_FERR	frame error flag
USART_FLAG_PERR	parity error flag
USART_FLAG_EPERR	early parity error flag
USART_FLAG_RFFINT	receive FIFO full interrupt flag
USART_FLAG_RFF	receive FIFO full flag
USART_FLAG_RFE	receive FIFO empty flag

### **Enum usart\_interrupt\_flag\_enum**

**Table 3-463. Enum usart\_interrupt\_flag\_enum**

<b>Member name</b>	<b>Function description</b>
USART_INT_FLAG_EB	end of block interrupt and flag
USART_INT_FLAG_RT	receiver timeout interrupt and flag
USART_INT_FLAG_AM	address match interrupt and flag
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	read data buffer not empty interrupt and overrun error flag

<b>Member name</b>	<b>Function description</b>
RR	
USART_INT_FLAG_IDLE	IDLE line detected interrupt and flag
USART_INT_FLAG_LBD	LIN break detected interrupt and flag
USART_INT_FLAG_WU	wakeup from deep-sleep mode interrupt and flag
USART_INT_FLAG_CTS	CTS interrupt and flag
USART_INT_FLAG_ERR_NERR	error interrupt and noise error flag
USART_INT_FLAG_ERR_ORER R	error interrupt and overrun error
USART_INT_FLAG_ERR_FERR	error interrupt and frame error flag
USART_INT_FLAG_RFF	receive FIFO full interrupt and flag

### Enum `uart_interrupt_enum`

**Table 3-464. Enum `uart_interrupt_enum`**

<b>Member name</b>	<b>Function description</b>
USART_INT_EB	end of block interrupt
USART_INT_RT	receiver timeout interrupt
USART_INT_AM	address match interrupt
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_WU	wakeup from deep-sleep mode interrupt
USART_INT_CTS	CTS interrupt
USART_INT_ERR	error interrupt
USART_INT_RFF	receive FIFO full interrupt

### Enum `uart_invert_enum`

**Table 3-465. Enum `uart_invert_enum`**

<b>Member name</b>	<b>Function description</b>
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_SWAP_ENABLE	swap TX/RX pins
USART_SWAP_DISABLE	not swap TX/RX pins

### **uart\_deinit**

The description of `uart_deinit` is shown as below:

**Table 3-466. Function `uart_deinit`**

<b>Function name</b>	uart_deinit
<b>Function prototype</b>	<code>void uart_deinit(uint32_t usart_periph);</code>
<b>Function descriptions</b>	reset USART
<b>Precondition</b>	-
<b>The called functions</b>	<code>rcu_periph_reset_enable / rcu_periph_reset_disable</code>
<b>Input parameter{in}</b>	
<code>usart_periph</code>	uart peripheral
<code>USARTx</code>	x=0,1
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* reset USART0 */

uart_deinit(USART0);
```

### **uart\_baudrate\_set**

The description of `uart_baudrate_set` is shown as below:

**Table 3-467. Function `uart_baudrate_set`**

<b>Function name</b>	uart_baudrate_set
<b>Function prototype</b>	<code>void uart_baudrate_set(uint32_t usart_periph, uint32_t baudval);</code>
<b>Function descriptions</b>	configure USART baud rate value
<b>Precondition</b>	-
<b>The called functions</b>	<code>rcu_clock_freq_get</code>
<b>Input parameter{in}</b>	
<code>usart_periph</code>	uart peripheral
<code>USARTx</code>	x=0,1
<b>Input parameter{in}</b>	
<code>baudval</code>	baud rate value
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* configure USART0 baud rate value */
```

---

```
uart_baudrate_set(USART0, 115200);
```

### **uart\_parity\_config**

The description of `uart_parity_config` is shown as below:

**Table 3-468. Function `uart_parity_config`**

<b>Function name</b>	uart_parity_config
<b>Function prototype</b>	void usart_parity_config(uint32_t usart_periph, uint32_t paritycfg);
<b>Function descriptions</b>	configure USART parity
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>usart_periph</b>	uart peripheral
<b>USARTx</b>	x=0,1
<b>Input parameter{in}</b>	
<b>paritycfg</b>	configure USART parity
<b>USART_PM_NONE</b>	no parity
<b>USART_PM_ODD</b>	odd parity
<b>USART_PM EVEN</b>	even parity
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* configure USART0 parity */
usart_parity_config(USART0, USART_PM_EVEN);
```

### **uart\_word\_length\_set**

The description of `uart_word_length_set` is shown as below:

**Table 3-469. Function `uart_word_length_set`**

<b>Function name</b>	uart_word_length_set
<b>Function prototype</b>	void usart_word_length_set(uint32_t usart_periph, uint32_t wlen);
<b>Function descriptions</b>	configure USART word length
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>usart_periph</b>	uart peripheral
<b>USARTx</b>	x=0,1
<b>Input parameter{in}</b>	
<b>wlen</b>	USART word length configure

<b>USART_WL_8BIT</b>	8 bits
<b>USART_WL_9BIT</b>	9 bits
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* configure USART0 word length */

uart_word_length_set(USART0, USART_WL_9BIT);
```

### **uart\_stop\_bit\_set**

The description of `uart_stop_bit_set` is shown as below:

**Table 3-470. Function `uart_stop_bit_set`**

<b>Function name</b>	uart_stop_bit_set
<b>Function prototype</b>	void usart_stop_bit_set(uint32_t usart_periph, uint32_t stblen);
<b>Function descriptions</b>	configure USART stop bit length
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>usart_periph</b>	uart peripheral
<b>USARTx</b>	x=0,1
<b>Input parameter{in}</b>	
<b>stblen</b>	USART stop bit configure
<b>USART_STB_1BIT</b>	1 bit
<b>USART_STB_0_5BIT</b>	0.5 bit
<b>USART_STB_2BIT</b>	2 bits
<b>USART_STB_1_5BIT</b>	1.5 bits
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* configure USART0 stop bit length */

uart_stop_bit_set(USART0, USART_STB_1_5BIT);
```

### **uart\_enable**

The description of `uart_enable` is shown as below:

**Table 3-471. Function usart\_enable**

<b>Function name</b>	usart_enable
<b>Function prototype</b>	void usart_enable(uint32_t usart_periph);
<b>Function descriptions</b>	enable USART
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>usart_periph</b>	uart peripheral
<b>USARTx</b>	x=0,1
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* enable USART0 */
usart_enable(USART0);
```

### **usart\_disable**

The description of usart\_disable is shown as below:

**Table 3-472. Function usart\_disable**

<b>Function name</b>	usart_disable
<b>Function prototype</b>	void usart_disable(uint32_t usart_periph);
<b>Function descriptions</b>	disable USART
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>usart_periph</b>	uart peripheral
<b>USARTx</b>	x=0,1
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* disable USART0 */
usart_disable(USART0);
```

### **usart\_transmit\_config**

The description of usart\_transmit\_config is shown as below:

**Table 3-473. Function usart\_transmit\_config**

<b>Function name</b>	usart_transmit_config
<b>Function prototype</b>	void usart_transmit_config(uint32_t usart_periph, uint32_t txconfig);
<b>Function descriptions</b>	configure USART transmitter
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>usart_periph</b>	uart peripheral
<b>USARTx</b>	x=0,1
<b>Input parameter{in}</b>	
<b>txconfig</b>	enable or disable USART transmitter
<b>USART_TRANSMIT_ENABLE</b>	enable USART transmission
<b>USART_TRANSMIT_DISABLE</b>	disable USART transmission
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

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 3-474. Function usart\_receive\_config**

<b>Function name</b>	usart_receive_config
<b>Function prototype</b>	void usart_receive_config(uint32_t usart_periph, uint32_t rxconfig);
<b>Function descriptions</b>	configure USART receiver
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>usart_periph</b>	uart peripheral
<b>USARTx</b>	x=0,1
<b>Input parameter{in}</b>	
<b>rxconfig</b>	enable or disable USART receiver
<b>USART_RECEIVE_ENABLE</b>	enable USART reception
<b>USART_RECEIVE_DISABLE</b>	disable USART reception

Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* Configure USART0 receiver */
uart_receive_config(USART0, USART_RECEIVE_ENABLE);
```

### **uart\_data\_first\_config**

The description of `uart_data_first_config` is shown as below:

**Table 3-475. Function `uart_data_first_config`**

Function name	uart_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}	
<code>usart_periph</code>	uart peripheral
<code>USARTx</code>	x=0,1
Input parameter{in}	
<code>msbf</code>	LSB/MSB
<code>USART_MSBF_LSB</code>	LSB first
<code>USART_MSBF_MSB</code>	MSB first
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* configure LSB of data first */
uart_data_first_config(USART0, USART_MSBF_LSB);
```

### **uart\_invert\_config**

The description of `uart_invert_config` is shown as below:

**Table 3-476. Function `uart_invert_config`**

Function name	uart_invert_config
Function prototype	void usart_invert_config(uint32_t usart_periph, usart_invert_enum invertpara);

<b>Function descriptions</b>	USART inverted configure
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>usart_periph</b>	uart peripheral
<b>USARTx</b>	x=0,1
<b>Input parameter{in}</b>	
<b>invertpara</b>	refer to <a href="#">Table 3-465. Enum usart_invert_enum</a>
<b>USART_DINV_ENABL_E</b>	data bit level inversion
<b>USART_DINV_DISABL_E</b>	data bit level not inversion
<b>USART_TXPIN_ENABL_E</b>	TX pin level inversion
<b>USART_TXPIN_DISABL_E</b>	TX pin level not inversion
<b>USART_RXPIN_ENABL_E</b>	RX pin level inversion
<b>USART_RXPIN_DISABL_E</b>	RX pin level not inversion
<b>USART_SWAP_ENABL_E</b>	swap TX/RX pins
<b>USART_SWAP_DISABL_E</b>	not swap TX/RX pins
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* configure USART0 inversion */
uart_invert_config(USART0, USART_DINV_ENABLE);
```

#### **uart\_overrun\_enable**

The description of **uart\_overrun\_enable** is shown as below:

**Table 3-477. Function `uart_overrun_enable`**

<b>Function name</b>	uart_overrun_enable
<b>Function prototype</b>	void uart_overrun_enable(uint32_t usart_periph);
<b>Function descriptions</b>	enable the USART overrun function
<b>Precondition</b>	-
<b>The called functions</b>	-

Input parameter{in}	
<b>usart_periph</b>	uart peripheral
<b>USARTx</b>	x=0,1
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* enable USART0 overrun */

uart_overrun_enable(USART0);
```

### **uart\_overrun\_disable**

The description of `uart_overrun_disable` is shown as below:

**Table 3-478. Function `uart_overrun_disable`**

<b>Function name</b>	uart_overrun_disable
<b>Function prototype</b>	void uart_overrun_disable(uint32_t usart_periph);
<b>Function descriptions</b>	disable the USART overrun function
<b>Precondition</b>	-
<b>The called functions</b>	-
Input parameter{in}	
<b>usart_periph</b>	uart peripheral
<b>USARTx</b>	x=0,1
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* disable USART0 overrun */

uart_overrun_disable(USART0);
```

### **uart\_oversample\_config**

The description of `uart_oversample_config` is shown as below:

**Table 3-479. Function `uart_oversample_config`**

<b>Function name</b>	uart_oversample_config
<b>Function prototype</b>	void uart_oversample_config(uint32_t usart_periph, uint32_t oversamp);
<b>Function descriptions</b>	configure the USART oversample mode
<b>Precondition</b>	-

The called functions		-
Input parameter{in}		
<b>uart_periph</b>		uart peripheral
<b>USARTx</b>		x=0,1
Input parameter{in}		
<b>oversamp</b>		oversample value
<b>USART_OVSMOD_8</b>		oversampling by 8
<b>USART_OVSMOD_16</b>		oversampling by 16
Output parameter{out}		
-		-
Return value		
-		-

Example:

```
/* config USART0 oversampling by 8 */

uart_oversample_config(USART0,USART_OVSMOD_8);
```

### **uart\_sample\_bit\_config**

The description of `uart_sample_bit_config` is shown as below:

**Table 3-480. Function `uart_sample_bit_config`**

Function name		uart_sample_bit_config
Function prototype		void uart_sample_bit_config(uint32_t usart_periph, uint32_t osb);
Function descriptions		configure the sample bit method
Precondition		-
The called functions		-
Input parameter{in}		
<b>uart_periph</b>		uart peripheral
<b>USARTx</b>		x=0,1
Input parameter{in}		
<b>osb</b>		sample bit
<b>USART_OSB_1BIT</b>		1 bit
<b>USART_OSB_3BIT</b>		3 bits
Output parameter{out}		
-		-
Return value		
-		-

Example:

```
/* config USART0 1 bit sample mode */

uart_sample_bit_config(USART0,USART_OSB_1BIT);
```

### **uart\_receiver\_timeout\_enable**

The description of `uart_receiver_timeout_enable` is shown as below:

**Table 3-481. Function `uart_receiver_timeout_enable`**

<b>Function name</b>	uart_receiver_timeout_enable
<b>Function prototype</b>	void <code>uart_receiver_timeout_enable(uint32_t usart_periph);</code>
<b>Function descriptions</b>	enable receiver timeout
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<code>usart_periph</code>	uart peripheral
<code>USARTx</code>	x=0
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* enable USART0 receiver timeout */
uart_receiver_timeout_enable(USART0);
```

### **uart\_receiver\_timeout\_disable**

The description of `uart_receiver_timeout_disable` is shown as below:

**Table 3-482. Function `uart_receiver_timeout_disable`**

<b>Function name</b>	uart_receiver_timeout_disable
<b>Function prototype</b>	void <code>uart_receiver_timeout_disable(uint32_t usart_periph);</code>
<b>Function descriptions</b>	disable receiver timeout
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<code>usart_periph</code>	uart peripheral
<code>USARTx</code>	x=0
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* disable USART0 receiver timeout */
uart_receiver_timeout_disable(USART0);
```

### **uart\_receiver\_timeout\_threshold\_config**

The description of `uart_receiver_timeout_threshold_config` is shown as below:

**Table 3-483. Function `uart_receiver_timeout_threshold_config`**

<b>Function name</b>	uart_receiver_timeout_threshold_config
<b>Function prototype</b>	void usart_receiver_timeout_threshold_config(uint32_t usart_periph, uint32_t rtimeout);
<b>Function descriptions</b>	configure receiver timeout threshold
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>usart_periph</b>	uart peripheral
<b>USARTx</b>	x=0
<b>Input parameter{in}</b>	
<b>rtimeout</b>	receiver timeout (0x00-0x00FFFFFF)
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* set the receiver timeout threshold of USART0*/
usart_receiver_timeout_threshold_config(USART0,115200*3);
```

### **uart\_data\_transmit**

The description of `uart_data_transmit` is shown as below:

**Table 3-484. Function `uart_data_transmit`**

<b>Function name</b>	uart_data_transmit
<b>Function prototype</b>	void usart_data_transmit(uint32_t usart_periph, uint32_t data);
<b>Function descriptions</b>	USART transmit data function
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>usart_periph</b>	uart peripheral
<b>USARTx</b>	x=0,1
<b>Input parameter{in}</b>	
<b>data</b>	data of transmission (0x00-0x1FF)
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* USART0 transmit data */

uart_data_transmit(USART0, 0xAA);
```

### **uart\_data\_receive**

The description of `uart_data_receive` is shown as below:

**Table 3-485. Function `uart_data_receive`**

<b>Function name</b>	uart_data_receive
<b>Function prototype</b>	void usart_data_receive(uint32_t usart_periph);
<b>Function descriptions</b>	USART receive data function
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>usart_periph</b>	uart peripheral
<b>USARTx</b>	x=0,1
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
<b>uint32_t</b>	data of received (0x00-0x1FF)

Example:

```
/* USART0 receive data */

uint16_t temp;

temp = usart_data_receive(USART0);
```

### **uart\_autobaud\_detection\_enable**

The description of `uart_autobaud_detection_enable` is shown as below:

**Table 3-486. Function `uart_autobaud_detection_enable`**

<b>Function name</b>	uart_autobaud_detection_enable
<b>Function prototype</b>	void usart_autobaud_detection_enable(uint32_t usart_periph);
<b>Function descriptions</b>	enable auto baud rate detection
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>usart_periph</b>	uart peripheral
<b>USARTx</b>	x=0
<b>Output parameter{out}</b>	
-	-

Return value	
-	-

Example:

```
/* enable USART0 auto baud rate detection */
uart_autobaud_detection_enable(USART0);
```

### **uart\_autobaud\_detection\_disable**

The description of `uart_autobaud_detection_disable` is shown as below:

**Table 3-487. Function `uart_autobaud_detection_disable`**

<b>Function name</b>	uart_autobaud_detection_disable
<b>Function prototype</b>	void uart_autobaud_detection_disable(uint32_t usart_periph);
<b>Function descriptions</b>	disable auto baud rate detection
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>usart_periph</b>	uart peripheral
<b>USARTx</b>	x=0
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* disable USART0 auto baud rate detection */
uart_autobaud_detection_disable(USART0);
```

### **uart\_autobaud\_detection\_mode\_config**

The description of `uart_autobaud_detection_mode_config` is shown as below:

**Table 3-488. Function `uart_autobaud_detection_mode_config`**

<b>Function name</b>	uart_autobaud_detection_mode_config
<b>Function prototype</b>	void uart_autobaud_detection_mode_config(uint32_t usart_periph, uint32_t abdmod);
<b>Function descriptions</b>	configure auto baud rate detection mode
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>usart_periph</b>	uart peripheral
<b>USARTx</b>	x=0

Input parameter{in}	
<b>abdmmod</b>	auto baud rate detection mode
<b>USART_ABDM_FTOR</b>	falling edge to rising edge measurement
<b>USART_ABDM_FTOF</b>	falling edge to falling edge measurement
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* configure USART0 auto baud rate detection mode */
uart_autobaud_detection_mode_config(USART0, USART_ABDM_FTOR);
```

### **uart\_address\_config**

The description of `uart_address_config` is shown as below:

**Table 3-489. Function `uart_address_config`**

<b>Function name</b>	uart_address_config
<b>Function prototype</b>	void uart_address_config(uint32_t usart_periph, uint8_t addr);
<b>Function descriptions</b>	configure the address of the USART terminal
<b>Precondition</b>	-
<b>The called functions</b>	-
Input parameter{in}	
<b>usart_periph</b>	uart peripheral
<b>USARTx</b>	x=0,1
Input parameter{in}	
<b>addr</b>	address of USART (0-0xFF)
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* configure address of the USART0 */
uart_address_config(USART0, 0x00);
```

### **uart\_address\_detection\_mode\_config**

The description of `uart_address_detection_mode_config` is shown as below:

**Table 3-490. Function `uart_address_detection_mode_config`**

<b>Function name</b>	uart_address_detection_mode_config
----------------------	------------------------------------

<b>Function prototype</b>	void usart_address_detection_mode_config(uint32_t usart_periph, uint32_t addmod);
<b>Function descriptions</b>	configure address detection mode
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>usart_periph</b>	uart peripheral
<b>USARTx</b>	x=0,1
<b>Input parameter{in}</b>	
<b>addmod</b>	address detection mode
<b>USART_ADDM_4BIT</b>	4 bits
<b>USART_ADDM_FULLB IT</b>	full bits
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/*configure address detection */
usart_address_config(USART0, USART_ADDM_4BIT);
```

### **usart\_mute\_mode\_enable**

The description of usart\_mute\_mode\_enable is shown as below:

**Table 3-491. Function usart\_mute\_mode\_enable**

<b>Function name</b>	usart_mute_mode_enable
<b>Function prototype</b>	void usart_mute_mode_enable(uint32_t usart_periph);
<b>Function descriptions</b>	enable mute mode
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>usart_periph</b>	uart peripheral
<b>USARTx</b>	x=0,1
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* enable USART0 receiver in mute mode */
usart_mute_mode_enable(USART0);
```

### **uart\_mute\_mode\_disable**

The description of `uart_mute_mode_disable` is shown as below:

**Table 3-492. Function `uart_mute_mode_disable`**

<b>Function name</b>	uart_mute_mode_disable
<b>Function prototype</b>	void <code>uart_mute_mode_disable(uint32_t usart_periph);</code>
<b>Function descriptions</b>	disable mute mode
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<code>usart_periph</code>	uart peripheral
<code>USARTx</code>	x=0,1
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* disable USART0 receiver in mute mode */

uart_mute_mode_disable(USART0);
```

### **uart\_mute\_mode\_wakeup\_config**

The description of `uart_mute_mode_wakeup_config` is shown as below:

**Table 3-493. Function `uart_mute_mode_wakeup_config`**

<b>Function name</b>	uart_mute_mode_wakeup_config
<b>Function prototype</b>	void <code>uart_mute_mode_wakeup_config(uint32_t usart_periph, uint32_t wmethod);</code>
<b>Function descriptions</b>	configure wakeup method in mute mode
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<code>usart_periph</code>	uart peripheral
<code>USARTx</code>	x=0,1
<b>Input parameter{in}</b>	
<code>wmethod</code>	two methods be used to enter or exit the mute mode
<code>USART_WM_IDLE</code>	idle line
<code>USART_WM_ADDR</code>	address mask
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* configure USART0 wakeup method in mute mode */

uart_mute_mode_wakeup_config(USART0, USART_WM_IDLE);
```

### **uart\_lin\_mode\_enable**

The description of `uart_lin_mode_enable` is shown as below:

**Table 3-494. Function `uart_lin_mode_enable`**

<b>Function name</b>	uart_lin_mode_enable
<b>Function prototype</b>	void usart_lin_mode_enable(uint32_t usart_periph);
<b>Function descriptions</b>	enable LIN mode
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>usart_periph</b>	uart peripheral
<b>USARTx</b>	x=0
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* USART0 LIN mode enable */

uart_lin_mode_enable(USART0);
```

### **uart\_lin\_mode\_disable**

The description of `uart_lin_mode_disable` is shown as below:

**Table 3-495. Function `uart_lin_mode_disable`**

<b>Function name</b>	uart_lin_mode_disable
<b>Function prototype</b>	void usart_lin_mode_disable(uint32_t usart_periph);
<b>Function descriptions</b>	disable LIN mode
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>usart_periph</b>	uart peripheral
<b>USARTx</b>	x=0
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* USART0 LIN mode disable */

uart_lin_mode_disable(USART0);
```

### **uart\_lin\_break\_dection\_length\_config**

The description of `uart_lin_break_dection_length_config` is shown as below:

**Table 3-496. Function `uart_lin_break_dection_length_config`**

<b>Function name</b>	uart_lin_break_dection_length_config
<b>Function prototype</b>	void uart_lin_break_dection_length_config(uint32_t usart_periph, uint32_t Iblen);
<b>Function descriptions</b>	LIN break detection length
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>usart_periph</b>	uart peripheral
<b>USARTx</b>	x=0
<b>Input parameter{in}</b>	
<b>Iblen</b>	two methods be used to enter or exit the mute mode
<b>USART_LBLEN_10B</b>	10 bits
<b>USART_LBLEN_11B</b>	11 bits
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* configure LIN break frame length */

uart_lin_break_dection_length_config(USART0, USART_LBLEN_10B);
```

### **uart\_halfduplex\_enable**

The description of `uart_halfduplex_enable` is shown as below:

**Table 3-497. Function `uart_halfduplex_enable`**

<b>Function name</b>	uart_halfduplex_enable
<b>Function prototype</b>	void uart_halfduplex_enable(uint32_t usart_periph);
<b>Function descriptions</b>	enable half-duplex mode
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>usart_periph</b>	uart peripheral

<b>USARTx</b>	x=0,1
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* enable USART0 half duplex mode*/
uart_halfduplex_enable(USART0);
```

### **uart\_halfduplex\_disable**

The description of **uart\_halfduplex\_disable** is shown as below:

**Table 3-498. Function `uart_halfduplex_disable`**

<b>Function name</b>	uart_halfduplex_disable
<b>Function prototype</b>	void uart_halfduplex_disable(uint32_t usart_periph);
<b>Function descriptions</b>	disable half-duplex mode
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>usart_periph</b>	uart peripheral
<b>USARTx</b>	x=0,1
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* disable USART0 half duplex mode*/
uart_halfduplex_disable(USART0);
```

### **uart\_clock\_enable**

The description of **uart\_clock\_enable** is shown as below:

**Table 3-499. Function `uart_clock_enable`**

<b>Function name</b>	uart_clock_enable
<b>Function prototype</b>	void uart_clock_enable(uint32_t usart_periph);
<b>Function descriptions</b>	enable clock
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	

<b>usart_periph</b>	uart peripheral
<b>USARTx</b>	x=0, 1
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* enable USART0 CK pin */

uart_synchronous_clock_enable(USART0);
```

### **uart\_clock\_disable**

The description of `uart_clock_disable` is shown as below:

**Table 3-500. Function `uart_clock_disable`**

<b>Function name</b>	uart_clock_disable
<b>Function prototype</b>	void usart_clock_disable(uint32_t usart_periph);
<b>Function descriptions</b>	disable clock
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>usart_periph</b>	uart peripheral
<b>USARTx</b>	x=0, 1
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* disable USART0 CK pin */

uart_synchronous_clock_disable(USART0);
```

### **uart\_synchronous\_clock\_config**

The description of `uart_synchronous_clock_config` is shown as below:

**Table 3-501. Function `uart_synchronous_clock_config`**

<b>Function name</b>	uart_synchronous_clock_config
<b>Function prototype</b>	void usart_synchronous_clock_config(uint32_t usart_periph, uint32_t clen, uint32_t cph, uint32_t cpl);
<b>Function descriptions</b>	configure USART synchronous mode parameters
<b>Precondition</b>	-

The called functions	
<b>Input parameter{in}</b>	
<b>usart_periph</b>	uart peripheral
<b>USARTTx</b>	x=0,1
<b>Input parameter{in}</b>	
<b>clen</b>	last bit clock pulse
<b>USART_CLEN_NONE</b>	clock pulse of the last data bit (MSB) is not output to the CK pin
<b>USART_CLEN_EN</b>	clock pulse of the last data bit (MSB) is output to the CK pin
<b>Input parameter{in}</b>	
<b>cph</b>	clock phase
<b>USART_CPH_1CK</b>	first clock transition is the first data capture edge
<b>USART_CPH_2CK</b>	second clock transition is the first data capture edge
<b>Input parameter{in}</b>	
<b>cpl</b>	clock polarity
<b>USART_CPL_LOW</b>	steady low value on CK pin
<b>USART_CPL_HIGH</b>	steady high value on CK pin
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* configure USART0 synchronous mode parameters */

uart_synchronous_clock_config(USART0,USART_CLEN_EN,USART_CPH_2CK,
USART_CPL_HIGH);
```

### **uart\_guard\_time\_config**

The description of **uart\_guard\_time\_config** is shown as below:

**Table 3-502. Function `uart_guard_time_config`**

Function name	
<b>Function prototype</b>	void <code>uart_guard_time_config(uint32_t usart_periph,uint32_t guat)</code> ;
<b>Function descriptions</b>	configure guard time value in smartcard mode
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>usart_periph</b>	uart peripheral
<b>USARTTx</b>	x=0
<b>Input parameter{in}</b>	
<b>guat</b>	guard time value (0x00-0x000000FF)
<b>Output parameter{out}</b>	
-	-

Return value	
-	-

Example:

```
/* configure USART0 guard time value in smartcard mode */

uart_guard_time_config(USART0, 0x0000 0055);
```

### **uart\_smartcard\_mode\_enable**

The description of `uart_smartcard_mode_enable` is shown as below:

**Table 3-503. Function `uart_smartcard_mode_enable`**

<b>Function name</b>	uart_smartcard_mode_enable
<b>Function prototype</b>	void uart_smartcard_mode_enable(uint32_t usart_periph);
<b>Function descriptions</b>	enable smartcard mode
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>usart_periph</b>	uart peripheral
<b>USARTx</b>	x=0
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* USART0 smartcard mode enable */

uart_smartcard_mode_enable(USART0);
```

### **uart\_smartcard\_mode\_disable**

The description of `uart_smartcard_mode_disable` is shown as below:

**Table 3-504. Function `uart_smartcard_mode_disable`**

<b>Function name</b>	uart_smartcard_mode_disable
<b>Function prototype</b>	void uart_smartcard_mode_disable(uint32_t usart_periph);
<b>Function descriptions</b>	disable smartcard mode
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>usart_periph</b>	uart peripheral
<b>USARTx</b>	x=0
<b>Output parameter{out}</b>	

-	-
<b>Return value</b>	
-	-

Example:

```
/* USART0 smartcard mode disable */

uart_smartcard_mode_disable(USART0);
```

### **uart\_smartcard\_mode\_nack\_enable**

The description of `uart_smartcard_mode_nack_enable` is shown as below:

**Table 3-505. Function `uart_smartcard_mode_nack_enable`**

<b>Function name</b>	uart_smartcard_mode_nack_enable
<b>Function prototype</b>	void usart_smartcard_mode_nack_enable(uint32_t usart_periph);
<b>Function descriptions</b>	enable NACK in smartcard mode
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>usart_periph</b>	uart peripheral
<b>USARTx</b>	x=0
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* enable USART0 NACK in smartcard mode */

uart_smartcard_mode_nack_enable(USART0);
```

### **uart\_smartcard\_mode\_nack\_disable**

The description of `uart_smartcard_mode_nack_disable` is shown as below:

**Table 3-506. Function `uart_smartcard_mode_nack_disable`**

<b>Function name</b>	uart_smartcard_mode_nack_disable
<b>Function prototype</b>	void usart_smartcard_mode_nack_disable(uint32_t usart_periph);
<b>Function descriptions</b>	disable NACK in smartcard mode
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>usart_periph</b>	uart peripheral
<b>USARTx</b>	x=0

Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* disable USART0 NACK in smartcard mode */
uart_smartcard_mode_nack_disable(USART0);
```

### **uart\_smartcard\_mode\_early\_nack\_enable**

The description of `uart_smartcard_mode_early_nack_enable` is shown as below:

**Table 3-507. Function `uart_smartcard_mode_early_nack_enable`**

Function name	uart_smartcard_mode_early_nack_enable
Function prototype	void uart_smartcard_mode_early_nack_enable(uint32_t usart_periph);
Function descriptions	enable early NACK in smartcard mode
Precondition	-
The called functions	-
Input parameter{in}	
usart_periph	uart peripheral
USARTx	x=0
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* enable USART0 early NACK in smartcard mode */
uart_smartcard_mode_early_nack_enable(USART0);
```

### **uart\_smartcard\_mode\_early\_nack\_disable**

The description of `uart_smartcard_mode_early_nack_disable` is shown as below:

**Table 3-508. Function `uart_smartcard_mode_early_nack_disable`**

Function name	uart_smartcard_mode_early_nack_disable
Function prototype	void uart_smartcard_mode_early_nack_disable(uint32_t usart_periph);
Function descriptions	disable early NACK in smartcard mode
Precondition	-
The called functions	-
Input parameter{in}	
usart_periph	uart peripheral

<b>USARTx</b>	x=0
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* disable USART0 early NACK in smartcard mode */

uart_smartcard_mode_early_nack_disable(USART0);
```

### **uart\_smartcard\_autoretry\_config**

The description of `uart_smartcard_autoretry_config` is shown as below:

**Table 3-509. Function `uart_smartcard_autoretry_config`**

<b>Function name</b>	uart_smartcard_autoretry_config
<b>Function prototype</b>	void usart_smartcard_autoretry_config(uint32_t usart_periph, uint32_t scrnum);
<b>Function descriptions</b>	configure smartcard auto-retry number
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>usart_periph</b>	uart peripheral
<b>USARTx</b>	x=0
<b>Input parameter{in}</b>	
<b>scrnum</b>	smartcard auto-retry number (0x00-0x00000007)
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* configure smartcard auto-retry number */

uart_smartcard_autoretry_config(USART0, 0x00000007);
```

### **uart\_block\_length\_config**

The description of `uart_block_length_config` is shown as below:

**Table 3-510. Function `uart_block_length_config`**

<b>Function name</b>	uart_block_length_config
<b>Function prototype</b>	void usart_block_length_config(uint32_t usart_periph, uint32_t bl);
<b>Function descriptions</b>	configure block length

<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>usart_periph</b>	uart peripheral
<b>USARTx</b>	x=0
<b>Input parameter{in}</b>	
<b>bl</b>	block length(0x00-0x000000FF)
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* configure block length in Smartcard T=1 reception */
uart_block_length_config(USART0, 0x000000FF);
```

### **uart\_irda\_mode\_enable**

The description of **uart\_irda\_mode\_enable** is shown as below:

**Table 3-511. Function `uart_irda_mode_enable`**

<b>Function name</b>	uart_irda_mode_enable
<b>Function prototype</b>	void uart_irda_mode_enable(uint32_t usart_periph);
<b>Function descriptions</b>	enable IrDA mode
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>usart_periph</b>	uart peripheral
<b>USARTx</b>	x=0
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* enable USART0 IrDA mode */
uart_irda_mode_enable(USART0);
```

### **uart\_irda\_mode\_disable**

The description of **uart\_irda\_mode\_disable** is shown as below:

**Table 3-512. Function usart\_irda\_mode\_disable**

<b>Function name</b>	usart_irda_mode_disable
<b>Function prototype</b>	void usart_irda_mode_disable(uint32_t usart_periph);
<b>Function descriptions</b>	disable IrDA mode
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
usart_periph	uart peripheral
USARTx	x=0
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* disable USART0 IrDA mode */
usart_irda_mode_disable(USART0);
```

### **usart\_prescaler\_config**

The description of usart\_prescaler\_config is shown as below:

**Table 3-513. Function usart\_prescaler\_config**

<b>Function name</b>	usart_prescaler_config
<b>Function prototype</b>	void usart_prescaler_config(uint32_t usart_periph, uint8_t psc);
<b>Function descriptions</b>	configure the peripheral clock prescaler in USART IrDA low-power mode
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
usart_periph	uart peripheral
USARTx	x=0
<b>Input parameter{in}</b>	
psc	clock prescaler (0x00-0xFF)
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* configure the USART0 peripheral clock prescaler in USART IrDA low-power mode */
usart_prescaler_config(USART0, 0x00);
```

### **uart\_irda\_lowpower\_config**

The description of `uart_irda_lowpower_config` is shown as below:

**Table 3-514. Function `uart_irda_lowpower_config`**

<b>Function name</b>	uart_irda_lowpower_config
<b>Function prototype</b>	<code>void usart_irda_lowpower_config(uint32_t usart_periph, uint32_t irlp);</code>
<b>Function descriptions</b>	configure IrDA low-power
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<code>usart_periph</code>	uart peripheral
<code>USARTx</code>	x=0
<b>Input parameter{in}</b>	
<code>irlp</code>	IrDA low-power or normal
<code>USART_IRLP_LOW</code>	low-power
<code>USART_IRLP_NORMAL</code>	normal
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* configure USART0 IrDA low-power */
uart_irda_lowpower_config(USART0, USART_IRLP_LOW);
```

### **uart\_hardware\_flow\_rts\_config**

The description of `uart_hardware_flow_rts_config` is shown as below:

**Table 3-515. Function `uart_hardware_flow_rts_config`**

<b>Function name</b>	uart_hardware_flow_rts_config
<b>Function prototype</b>	<code>void usart_hardware_flow_rts_config(uint32_t usart_periph, uint32_t rtsconfig);</code>
<b>Function descriptions</b>	configure hardware flow control RTS
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<code>usart_periph</code>	uart peripheral
<code>USARTx</code>	x=0,1
<b>Input parameter{in}</b>	
<code>rtsconfig</code>	enable or disable RTS
<code>USART_RTS_ENABLE</code>	enable RTS

<i>USART_RTS_DISABLE</i>	disable RTS
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* configure USART0 hardware flow control RTS */
uart_hardware_flow_rts_config(USART0, USART_RTS_ENABLE);
```

### **uart\_hardware\_flow\_cts\_config**

The description of `uart_hardware_flow_cts_config` is shown as below:

**Table 3-516. Function `uart_hardware_flow_cts_config`**

<b>Function name</b>	uart_hardware_flow_cts_config
<b>Function prototype</b>	void uart_hardware_flow_cts_config(uint32_t usart_periph, uint32_t ctsconfig);
<b>Function descriptions</b>	configure hardware flow control RTS
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>usart_periph</b>	uart peripheral
<i>USARTx</i>	x=0,1
<b>Input parameter{in}</b>	
<b>ctsconfig</b>	enable or disable CTS
<i>USART_CTS_ENABLE</i>	enable CTS
<i>USART_CTS_DISABLE</i>	disable CTS
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* configure USART0 hardware flow control CTS */
uart_hardware_flow_cts_config(USART0, USART_CTS_ENABLE);
```

### **uart\_hardware\_flow\_coherence\_config**

The description of `uart_hardware_flow_coherence_config` is shown as below:

**Table 3-517. Function usart\_hardware\_flow\_coherence\_config**

<b>Function name</b>	usart_hardware_flow_coherence_config
<b>Function prototype</b>	void usart_hardware_flow_coherence_config(uint32_t usart_periph, uint32_t hcm);
<b>Function descriptions</b>	configure hardware flow control coherence mode
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
usart_periph	uart peripheral
USARTx	x=0,1
<b>Input parameter{in}</b>	
hcm	Hardware flow control coherence mode
USART_HCM_NONE	nRTS signal equals to the rxne status register
USART_HCM_EN	nRTS signal is set when the last data bit has been sampled
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* configure hardware flow control coherence mode */

usart_hardware_flow_coherence_config(USART0, USART_HCM_NONE);
```

### **usart\_rs485\_driver\_enable**

The description of usart\_rs485\_driver\_enable is shown as below:

**Table 3-518. Function usart\_rs485\_driver\_enable**

<b>Function name</b>	usart_rs485_driver_enable
<b>Function prototype</b>	void usart_rs485_driver_enable(uint32_t usart_periph);
<b>Function descriptions</b>	enable USART RS485 driver
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
usart_periph	uart peripheral
USARTx	x=0,1
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* enable USART0 RS485 driver */
```

---

```
uart_rs485_driver_enable(USART0);
```

### **uart\_rs485\_driver\_disable**

The description of `uart_rs485_driver_disable` is shown as below:

**Table 3-519. Function `uart_rs485_driver_disable`**

<b>Function name</b>	uart_rs485_driver_disable
<b>Function prototype</b>	void uart_rs485_driver_disable(uint32_t usart_periph);
<b>Function descriptions</b>	disable USARTRS485 driver
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
usart_periph	uart peripheral
USARTx	x=0,1
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* disable USART0 RS485 driver */
uart_rs485_driver_disable (USART0);
```

### **uart\_driver\_assertime\_config**

The description of `uart_driver_assertime_config` is shown as below:

**Table 3-520. Function `uart_driver_assertime_config`**

<b>Function name</b>	uart_driver_assertime_config
<b>Function prototype</b>	void uart_driver_assertime_config(uint32_t usart_periph, uint32_t deatime);
<b>Function descriptions</b>	configure driver enable assertion time
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
usart_periph	uart peripheral
USARTx	x=0,1
<b>Input parameter{in}</b>	
deatime	driver enable assertion time (0x00-0x0000001F)
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* set USART0 driver assertime */

uart_driver_assertime_config(USART0,0x0000001F);
```

### **uart\_driver\_deassertime\_config**

The description of `uart_driver_deassertime_config` is shown as below:

**Table 3-521. Function `uart_driver_deassertime_config`**

<b>Function name</b>	uart_driver_deassertime_config
<b>Function prototype</b>	void <code>uart_driver_deassertime_config(uint32_t usart_periph, uint32_t dectime);</code>
<b>Function descriptions</b>	configure driver enable de-assertion time
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<code>usart_periph</code>	uart peripheral
<code>USARTx</code>	x=0,1
<b>Input parameter{in}</b>	
<code>dectime</code>	driver enable de-assertion time (0x00-0x0000001F)
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* set USART0 driver deassertime */

uart_driver_deassertime_config(USART0,0x0000001F);
```

### **uart\_depolarity\_config**

The description of `uart_depolarity_config` is shown as below:

**Table 3-522. Function `uart_depolarity_config`**

<b>Function name</b>	uart_depolarity_config
<b>Function prototype</b>	void <code>uart_depolarity_config(uint32_t usart_periph, uint32_t dep);</code>
<b>Function descriptions</b>	configure driver enable polarity mode
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	

<b>usart_periph</b>	uart peripheral
<i>USARTx</i>	x=0,1
<b>Input parameter{in}</b>	
<b>dep</b>	DE signal
<i>USART_DEP_HIGH</i>	DE signal is active high
<i>USART_DEP_LOW</i>	DE signal is active low
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* configure driver enable polarity mode */
uart_driver_depolarity_config(USART0, USART_DEP_HIGH);
```

### **uart\_dma\_receive\_config**

The description of **uart\_dma\_receive\_config** is shown as below:

**Table 3-523. Function `uart_dma_receive_config`**

<b>Function name</b>	uart_dma_receive_config
<b>Function prototype</b>	void <code>uart_dma_receive_config(uint32_t usart_periph, uint32_t dmacmd);</code>
<b>Function descriptions</b>	configure USART DMA reception
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>usart_periph</b>	uart peripheral
<i>USARTx</i>	x=0,1
<b>Input parameter{in}</b>	
<b>dmacmd</b>	enable or disable DMA for reception
<i>USART_DENR_ENABL E</i>	DMA enable for reception
<i>USART_DENR_DISAB LE</i>	DMA disable for reception
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* USART0 DMA enable for reception */

uart_dma_receive_config(USART0, USART_DENR_ENABLE);
```

### **uart\_dma\_transmit\_config**

The description of `uart_dma_transmit_config` is shown as below:

**Table 3-524. Function `uart_dma_transmit_config`**

<b>Function name</b>	uart_dma_transmit_config
<b>Function prototype</b>	void <code>uart_dma_transmit_config(uint32_t usart_periph, uint32_t dmacmd);</code>
<b>Function descriptions</b>	configure USART DMA transmission
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<code>usart_periph</code>	uart peripheral
<code>USARTx</code>	x=0,1
<b>Input parameter{in}</b>	
<code>dmacmd</code>	enable or disable DMA for transmission
<code>USART_DENT_ENABLE</code>	DMA enable for transmission
<code>USART_DENT_DISABLE</code>	DMA disable for transmission
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* USART0 DMA enable for transmission */

uart_dma_transmit_config(USART0, USART_DENT_ENABLE);
```

### **uart\_reception\_error\_dma\_disable**

The description of `uart_reception_error_dma_disable` is shown as below:

**Table 3-525. Function `uart_reception_error_dma_disable`**

<b>Function name</b>	uart_reception_error_dma_disable
<b>Function prototype</b>	void <code>uart_reception_error_dma_disable(uint32_t usart_periph);</code>
<b>Function descriptions</b>	disable DMA on reception error
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	

<b>usart_periph</b>	uart peripheral
<b>USARTx</b>	x=0,1
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* disable DMA on reception error */

uart_reception_error_dma_disable(USART0);
```

### **uart\_reception\_error\_dma\_enable**

The description of **uart\_reception\_error\_dma\_enable** is shown as below:

**Table 3-526. Function `uart_reception_error_dma_enable`**

<b>Function name</b>	uart_reception_error_dma_enable
<b>Function prototype</b>	void uart_reception_error_dma_enable(uint32_t usart_periph);
<b>Function descriptions</b>	enable DMA on reception error
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>usart_periph</b>	uart peripheral
<b>USARTx</b>	x=0,1
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* enable DMA on reception error */

uart_reception_error_dma_enable(USART0);
```

### **uart\_wakeup\_enable**

The description of **uart\_wakeup\_enable** is shown as below:

**Table 3-527. Function `uart_wakeup_enable`**

<b>Function name</b>	uart_wakeup_enable
<b>Function prototype</b>	void uart_wakeup_enable(uint32_t usart_periph);
<b>Function descriptions</b>	USART be able to wake up the MCU from deep-sleep mode
<b>Precondition</b>	-
<b>The called functions</b>	-

Input parameter{in}	
<b>usart_periph</b>	uart peripheral
<b>USARTx</b>	x=0
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* USART0 wake up enable */
uart_wakeup_enable(USART0);
```

### **uart\_wakeup\_disable**

The description of `uart_wakeup_disable` is shown as below:

**Table 3-528. Function `uart_wakeup_disable`**

<b>Function name</b>	uart_wakeup_disable
<b>Function prototype</b>	void uart_wakeup_disable(uint32_t usart_periph);
<b>Function descriptions</b>	USART not be able to wake up the MCU from deep-sleep mode
<b>Precondition</b>	-
<b>The called functions</b>	-
Input parameter{in}	
<b>usart_periph</b>	uart peripheral
<b>USARTx</b>	x=0
Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* USART0 wake up disable */
uart_wakeup_disable(USART0);
```

### **uart\_wakeup\_mode\_config**

The description of `uart_wakeup_mode_config` is shown as below:

**Table 3-529. Function `uart_wakeup_mode_config`**

<b>Function name</b>	uart_wakeup_mode_config
<b>Function prototype</b>	void uart_wakeup_mode_config(uint32_t usart_periph, uint32_t wum);
<b>Function descriptions</b>	wakeup mode from deep-sleep mode
<b>Precondition</b>	-

<b>The called functions</b>		-
<b>Input parameter{in}</b>		
<b>usart_periph</b>		uart peripheral
<b>USARTx</b>		x=0
<b>Input parameter{in}</b>		
<b>wum</b>		wakeup mode
<b>USART_WUM_ADDR</b>		WUF active on address match
<b>USART_WUM_START_B</b>		WUF active on start bit
<b>USART_WUM_RBNE</b>		WUF active on RBNE
<b>Output parameter{out}</b>		
-		-
<b>Return value</b>		
-		-

Example:

```
/* configure USART0 wake up mode */

uart_wakeup_mode_config(USART0, USART_WUM_ADDR);
```

### **uart\_receive\_fifo\_enable**

The description of **uart\_receive\_fifo\_enable** is shown as below:

**Table 3-530. Function `uart_receive_fifo_enable`**

<b>Function name</b>	uart_receive_fifo_enable	
<b>Function prototype</b>	void <code>uart_receive_fifo_enable(uint32_t usart_periph);</code>	
<b>Function descriptions</b>	enable receive FIFO	
<b>Precondition</b>	-	
<b>The called functions</b>	-	
<b>Input parameter{in}</b>		
<b>usart_periph</b>		uart peripheral
<b>USARTx</b>		x=0,1
<b>Output parameter{out}</b>		
-		-
<b>Return value</b>		
-		-

Example:

```
/* enable receive FIFO */

uart_receive_fifo_enable(USART0);
```

### **uart\_receive\_fifo\_disable**

The description of `uart_receive_fifo_disable` is shown as below:

**Table 3-531. Function `uart_receive_fifo_disable`**

<b>Function name</b>	uart_receive_fifo_disable
<b>Function prototype</b>	void <code>uart_receive_fifo_disable(uint32_t usart_periph);</code>
<b>Function descriptions</b>	disable receive FIFO
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<code>usart_periph</code>	uart peripheral
<code>USARTx</code>	x=0,1
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* disable receive FIFO */

uart_receive_fifo_disable(USART0);
```

### **uart\_receive\_fifo\_counter\_number**

The description of `uart_receive_fifo_counter_number` is shown as below:

**Table 3-532. Function `uart_receive_fifo_counter_number`**

<b>Function name</b>	uart_receive_fifo_counter_number
<b>Function prototype</b>	<code>uint8_t <code>uart_receive_fifo_counter_number(uint32_t usart_periph);</code></code>
<b>Function descriptions</b>	read receive FIFO counter number
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<code>usart_periph</code>	uart peripheral
<code>USARTx</code>	x=0,1
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
<code>uint8_t</code>	receive FIFO counter number

Example:

```
/* read receive FIFO counter number */

uint8_t temp;
```

temp = usart\_receive\_fifo\_counter\_number(USART0);

### **usart\_flag\_get**

The description of usart\_flag\_get is shown as below:

**Table 3-533. Function usart\_flag\_get**

<b>Function name</b>	usart_flag_get
<b>Function prototype</b>	FlagStatus usart_flag_get(uint32_t usart_periph, usart_flag_enum flag);
<b>Function descriptions</b>	get flag in STAT/CHC/RFCS register
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>usart_periph</b>	uart peripheral
<b>USARTx</b>	x=0,1
<b>Input parameter{in}</b>	
<b>flag</b>	USART flags, refer to <a href="#">Table 3-462. Enum usart_flag_enum</a> only one among these parameters can be selected
<b>USART_FLAG_PERR</b>	parity error flag
<b>USART_FLAG_FERR</b>	frame error flag
<b>USART_FLAG_NERR</b>	noise error flag
<b>USART_FLAG_ORER</b> <i>R</i>	overrun error
<b>USART_FLAG_IDLE</b>	idle line detected flag
<b>USART_FLAG_RBNE</b>	read data buffer not empty
<b>USART_FLAG_TC</b>	transmission completed
<b>USART_FLAG_TBE</b>	transmit data register empty
<b>USART_FLAG_LBD</b>	LIN break detected flag
<b>USART_FLAG_CTSF</b>	CTS change flag
<b>USART_FLAG_CTS</b>	CTS level
<b>USART_FLAG_RT</b>	receiver timeout flag
<b>USART_FLAG_EB</b>	end of block flag
<b>USART_FLAG_ABDE</b>	auto baudrate detection error
<b>USART_FLAG_ABD</b>	auto baudrate detection flag
<b>USART_FLAG_BSY</b>	busy flag
<b>USART_FLAG_AM</b>	address match flag
<b>USART_FLAG_SB</b>	send break flag
<b>USART_FLAG_RWU</b>	receiver wakeup from mute mode
<b>USART_FLAG_WU</b>	wakeup from deep-sleep mode flag
<b>USART_FLAG_TEA</b>	transmit enable acknowledge flag
<b>USART_FLAG_REA</b>	receive enable acknowledge flag
<b>USART_FLAG_EPERR</b>	early parity error flag
<b>USART_FLAG_RFE</b>	receive FIFO empty flag

<b>USART_FLAG_RFF</b>	receive FIFO full flag
<b>USART_FLAG_RFFINT</b>	receive FIFO full interrupt flag
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
<b>FlagStatus</b>	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 3-534. Function usart\_flag\_clear**

<b>Function name</b>	usart_flag_clear
<b>Function prototype</b>	void usart_flag_clear(uint32_t usart_periph, usart_flag_enum flag);
<b>Function descriptions</b>	clear flag in STAT register
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>usart_periph</b>	uart peripheral
<b>USARTx</b>	x=0,1
<b>Input parameter{in}</b>	
<b>flag</b>	USART flags, refer to <a href="#">Table 3-462. Enum usart_flag_enum</a> only one among these parameters can be selected
<b>USART_FLAG_PERR</b>	parity error flag
<b>USART_FLAG_FERR</b>	frame error flag
<b>USART_FLAG_NERR</b>	noise detected flag
<b>USART_FLAG_ORER</b> <i>R</i>	overrun error flag
<b>USART_FLAG_IDLE</b>	idle line detected flag
<b>USART_FLAG_TC</b>	transmission complete flag
<b>USART_FLAG_LBD</b>	LIN break detected flag
<b>USART_FLAG_CTSF</b>	CTS change flag
<b>USART_FLAG_RT</b>	receiver timeout flag
<b>USART_FLAG_EB</b>	end of block flag
<b>USART_FLAG_AM</b>	address match flag
<b>USART_FLAG_WU</b>	wakeup from deep-sleep mode flag
<b>USART_FLAG_EPERR</b>	early parity error flag

Output parameter{out}	
-	-
Return value	
-	-

Example:

```
/* clear USART0 flag */

uart_flag_clear(USART0,USART_FLAG_TC);
```

### **uart\_interrupt\_enable**

The description of `uart_interrupt_enable` is shown as below:

**Table 3-535. Function `uart_interrupt_enable`**

Function name	uart_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}	
<code>usart_periph</code>	uart peripheral
<code>USARTx</code>	x=0,1
Input parameter{in}	
<code>interrupt</code>	interrupt type, refer to <a href="#">Table 3-464. Enum usart_interrupt_enum</a> only one among these parameters can be selected
<code>USART_INT_IDLE</code>	idle interrupt
<code>USART_INT_RBNE</code>	read data buffer not empty interrupt and overrun error interrupt enable interrupt
<code>USART_INT_TC</code>	transmission complete interrupt
<code>USART_INT_TBE</code>	transmit data register empty interrupt
<code>USART_INT_PERR</code>	parity error interrupt
<code>USART_INT_AM</code>	address match interrupt
<code>USART_INT_RT</code>	receiver timeout interrupt
<code>USART_INT_EB</code>	end of block interrupt
<code>USART_INT_LBD</code>	LIN break detection interrupt
<code>USART_INT_ERR</code>	error interrupt enable in multibuffer communication
<code>USART_INT_CTS</code>	CTS interrupt
<code>USART_INT_WU</code>	wakeup from deep-sleep mode interrupt
<code>USART_INT_RFF</code>	receive FIFO full interrupt enable
Output parameter{out}	
-	-
Return value	

-	-
---	---

Example:

```
/* enable USART0 TBE interrupt */

uart_interrupt_enable(USART0, USART_INT_TBE);
```

### **uart\_interrupt\_disable**

The description of `uart_interrupt_disable` is shown as below:

**Table 3-536. Function `uart_interrupt_disable`**

<b>Function name</b>	uart_interrupt_disable
<b>Function prototype</b>	void usart_interrupt_disable(uint32_t usart_periph, usart_interrupt_enum interrupt);
<b>Function descriptions</b>	disable USART interrupt
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>usart_periph</b>	uart peripheral
<b>USARTx</b>	x=0,1
<b>Input parameter{in}</b>	
<b>interrupt</b>	interrupt type, refer to <a href="#">Table 3-464. Enum usart_interrupt enum</a> only one among these parameters can be selected
<b>USART_INT_IDLE</b>	idle interrupt
<b>USART_INT_RBNE</b>	read data buffer not empty interrupt and overrun error interrupt enable interrupt
<b>USART_INT_TC</b>	transmission complete interrupt
<b>USART_INT_TBE</b>	transmit data register empty interrupt
<b>USART_INT_PERR</b>	parity error interrupt
<b>USART_INT_AM</b>	address match interrupt
<b>USART_INT_RT</b>	receiver timeout interrupt
<b>USART_INT_EB</b>	end of block interrupt
<b>USART_INT_LBD</b>	LIN break detection interrupt
<b>USART_INT_ERR</b>	error interrupt enable in multibuffer communication
<b>USART_INT_CTS</b>	CTS interrupt
<b>USART_INT_WU</b>	wakeup from deep-sleep mode interrupt
<b>USART_INT_RFF</b>	receive FIFO full interrupt enable
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

---

```
/* disable USART0 TBE interrupt */

uart_interrupt_disable(USART0, USART_INT_TBE);
```

### **uart\_command\_enable**

The description of `uart_command_enable` is shown as below:

**Table 3-537. Function `uart_command_enable`**

<b>Function name</b>	uart_command_enable
<b>Function prototype</b>	void usart_command_enable(uint32_t usart_periph, uint32_t cmdtype);
<b>Function descriptions</b>	enable USART command
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>usart_periph</b>	uart peripheral
<b>USARTx</b>	x=0,1
<b>Input parameter{in}</b>	
<b>cmdtype</b>	command type
<b>USART_CMD_ABDCMD</b> <i>D</i>	auto baudrate detection command
<b>USART_CMD_SBKCM</b> <i>D</i>	send break command
<b>USART_CMD_MMCM</b>	mute mode command
<b>USART_CMD_RXFCM</b> <i>D</i>	receive data flush command
<b>USART_CMD_TXFCM</b> <i>D</i>	transmit data flush request
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* enable USART0 command */

uart_command_enable(USART0, USART_CMD_ABDCMD);
```

### **uart\_interrupt\_flag\_get**

The description of `uart_interrupt_flag_get` is shown as below:

**Table 3-538. Function `uart_interrupt_flag_get`**

<b>Function name</b>	uart_interrupt_flag_get
<b>Function prototype</b>	FlagStatus usart_interrupt_flag_get(uint32_t usart_periph, usart_interrupt_flag_enum int_flag);

<b>Function descriptions</b>	get USART interrupt and flag status
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>uart_periph</b>	uart peripheral
<b>USARTx</b>	x=0,1
<b>Input parameter{in}</b>	
<b>int_flag</b>	USART interrupt flag, refer to <a href="#">Table 3-463. Enum usart_interrupt_flag_enum</a> , only one among these parameters can be selected
<b>USART_INT_FLAG_EB</b>	end of block interrupt and flag
<b>USART_INT_FLAG_RT</b>	receiver timeout interrupt and flag
<b>USART_INT_FLAG_AM</b>	address match interrupt and flag
<b>USART_INT_FLAG_PERR</b>	parity error interrupt and flag
<b>USART_INT_FLAG_TB_E</b>	transmitter buffer empty interrupt and flag
<b>USART_INT_FLAG_TC</b>	transmission complete interrupt and flag
<b>USART_INT_FLAG_RB_NE</b>	read data buffer not empty interrupt and flag
<b>USART_INT_FLAG_RB_NE_ORERR</b>	read data buffer not empty interrupt and overrun error flag
<b>USART_INT_FLAG_IDLE</b>	IDLE line detected interrupt and flag
<b>USART_INT_FLAG_LBD</b>	LIN break detected interrupt and flag
<b>USART_INT_FLAG_WU</b>	wakeup from deep-sleep mode interrupt and flag
<b>USART_INT_FLAG_CTS</b>	CTS interrupt and flag
<b>USART_INT_FLAG_ER_NERR</b>	error interrupt and noise error flag
<b>USART_INT_FLAG_ER_R_ORERR</b>	error interrupt and overrun error
<b>USART_INT_FLAG_ER_R_FERR</b>	error interrupt and frame error flag
<b>USART_INT_FLAG_RF</b>	receive FIFO full interrupt and flag
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
<b>FlagStatus</b>	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 3-539. Function usart\_interrupt\_flag\_clear**

<b>Function name</b>	usart_interrupt_flag_clear
<b>Function prototype</b>	void usart_interrupt_flag_clear(uint32_t usart_periph, usart_interrupt_flag_enum flag);
<b>Function descriptions</b>	clear USART interrupt flag in STAT register
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
usart_periph	uart peripheral
USARTx	x=0,1
<b>Input parameter{in}</b>	
flag	USART interrupt flag, refer to <a href="#">Table 3-463. Enum usart_interrupt_flag_enum</a> , only one among these parameters can be selected
USART_INT_FLAG_PE RR	parity error flag
USART_INT_FLAG_ER R_FERR	frame error flag
USART_INT_FLAG_ER R_NERR	noise detected flag
USART_INT_FLAG_RB NE_ORERR	read data buffer not empty interrupt and overrun error flag
USART_INT_FLAG_ER R_ORERR	error interrupt and overrun error
USART_INT_FLAG_ID LE	idle line detected flag
USART_INT_FLAG_TC	transmission complete flag
USART_INT_FLAG_LB D	LIN break detected flag
USART_INT_FLAG_CT S	CTS change flag
USART_INT_FLAG_RT	receiver timeout flag
USART_INT_FLAG_EB	end of block flag

<i>USART_INT_FLAG_A</i>	address match flag
<i>USART_INT_FLAG_W</i>	wakeup from deep-sleep mode flag
<i>USART_INT_FLAG_RF</i>	receive FIFO full interrupt and flag
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* clear the USART0 interrupt flag */

uart_interrupt_flag_clear(USART0, USART_INT_FLAG_TC);
```

## 3.20. WWDGT

The window watchdog timer (WWDGT) is used to detect system failures due to software malfunctions. The WWDGT registers are listed in chapter [3.20.1](#), the FWDGT firmware functions are introduced in chapter [3.20.2](#).

### 3.20.1. Descriptions of Peripheral registers

WWDGT registers are listed in the table shown as below:

**Table 3-540. WWDGT Registers**

Registers	Descriptions
WWDGT_CTL	WWDGT control register
WWDGT_CFG	WWDGT configuration register
WWDGT_STAT	WWDGT status register

### 3.20.2. Descriptions of Peripheral functions

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

**Table 3-541. 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_interrupt_enable	enable early wakeup interrupt of WWDGT

Function name	Function description
wwdgt_flag_get	check early wakeup interrupt state of WWDGT
wwdgt_flag_clear	clear early wakeup interrupt state of WWDGT

## wwdgt\_deinit

The description of wwdgt\_deinit is shown as below:

**Table 3-542. 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	-
<b>Input parameter{in}</b>	
-	-
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* reset the window watchdog timer configuration */
wwdgt_deinit();
```

## wwdgt\_enable

The description of wwdgt\_enable is shown as below:

**Table 3-543. 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	-
<b>Input parameter{in}</b>	
-	-
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* start the WWDGT counter */
```

```
wwdgt_enable( );
```

### **wwdgt\_counter\_update**

The description of wwdgt\_counter\_update is shown as below:

**Table 3-544. Function wwdgt\_counter\_update**

<b>Function name</b>	wwdgt_counter_update
<b>Function prototype</b>	void wwdgt_counter_update(uint16_t counter_value);
<b>Function descriptions</b>	configure the window watchdog timer counter value
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>counter_value</b>	counter_value: 0x00000000 - 0x0000007F
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* update WWDGT counter to 0x7F */

wwdgt_counter_update(127);
```

### **wwdgt\_config**

The description of wwdgt\_config is shown as below:

**Table 3-545. Function wwdgt\_config**

<b>Function name</b>	wwdgt_config
<b>Function prototype</b>	void wwdgt_config(uint16_t counter, uint16_t window, uint32_t prescaler);
<b>Function descriptions</b>	configure counter value, window value, and prescaler divider value
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
<b>counter</b>	counter: 0x00000000 - 0x0000007F
<b>Input parameter{in}</b>	
<b>window</b>	window: 0x00000000 - 0x0000007F
<b>Input parameter{in}</b>	
<b>prescaler</b>	wwdgt prescaler value
<b>WWDGT_CFG_PSC_D IV1</b>	the time base of WWDGT counter = (PCLK1/4096)/1
<b>WWDGT_CFG_PSC_D IV2</b>	the time base of WWDGT counter = (PCLK1/4096)/2
<b>WWDGT_CFG_PSC_D</b>	the time base of WWDGT counter = (PCLK1/4096)/4

<i>IV4</i>	
<i>WWDGT_CFG_PSC_D</i> <i>IV8</i>	the time base of WWDGT counter = (PCLK1/4096)/8
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* config WWDGT counter value to 0x7F, window value to 0x50, prescaler divider value to
8 */
wwdgt_config(127, 80, WWDGT_CFG_PSC_DIV8);
```

### **wwdgt\_interrupt\_enable**

The description of wwdgt\_interrupt\_enable is shown as below:

**Table 3-546. Function wwdgt\_interrupt\_enable**

<b>Function name</b>	wwdgt_interrupt_enable
<b>Function prototype</b>	void wwdgt_interrupt_enable(void);
<b>Function descriptions</b>	enable early wakeup interrupt of WWDGT
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
-	-
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* enable early wakeup interrupt of WWDGT */
wwdgt_interrupt_enable();
```

### **wwdgt\_flag\_get**

The description of wwdgt\_flag\_get is shown as below:

**Table 3-547. Function wwdgt\_flag\_get**

<b>Function name</b>	wwdgt_flag_get
<b>Function prototype</b>	FlagStatus wwdgt_flag_get(void);
<b>Function descriptions</b>	check early wakeup interrupt state of WWDGT
<b>Precondition</b>	-

<b>The called functions</b>	-
<b>Input parameter{in}</b>	
-	-
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
<b>FlagStatus</b>	SET or RESET

Example:

```
/* test if the counter value update has reached the 0x40 */

FlagStatus status;

status = wwdgt_flag_get ( );
```

### **wwdgt\_flag\_clear**

The description of wwdgt\_flag\_clear is shown as below:

**Table 3-548. Function wwdgt\_flag\_clear**

<b>Function name</b>	wwdgt_flag_clear
<b>Function prototype</b>	void wwdgt_flag_clear(void);
<b>Function descriptions</b>	clear early wakeup interrupt state of WWDGT
<b>Precondition</b>	-
<b>The called functions</b>	-
<b>Input parameter{in}</b>	
-	-
<b>Output parameter{out}</b>	
-	-
<b>Return value</b>	
-	-

Example:

```
/* clear early wakeup interrupt state of WWDGT */

wwdgt_flag_clear( );
```

## 4. Revision history

**Table 4-1. Revision history**

Revision No.	Description	Date
1.0	Initial Release	Dec.7, 2020

### Important Notice

This document is the property of GigaDevice Semiconductor Inc. and its subsidiaries (the "Company"). This document, including any product of the Company described in this document (the "Product"), is owned by the Company under the intellectual property laws and treaties of the People's Republic of China and other jurisdictions worldwide. The Company reserves all rights under such laws and treaties and does not grant any license under its patents, copyrights, trademarks, or other intellectual property rights. The names and brands of third party referred thereto (if any) are the property of their respective owner and referred to for identification purposes only.

The Company makes no warranty of any kind, express or implied, with regard to this document or any Product, including, but not limited to, the implied warranties of merchantability and fitness for a particular purpose. The Company does not assume any liability arising out of the application or use of any Product described in this document. Any information provided in this document is provided only for reference purposes. It is the responsibility of the user of this document to properly design, program, and test the functionality and safety of any application made of this information and any resulting product. Except for customized products which has been expressly identified in the applicable agreement, the Products are designed, developed, and/or manufactured for ordinary business, industrial, personal, and/or household applications only. The Products are not designed, intended, or authorized for use as components in systems designed or intended for the operation of weapons, weapons systems, nuclear installations, atomic energy control instruments, combustion control instruments, airplane or spaceship instruments, transportation instruments, traffic signal instruments, life-support devices or systems, other medical devices or systems (including resuscitation equipment and surgical implants), pollution control or hazardous substances management, or other uses where the failure of the device or Product could cause personal injury, death, property or environmental damage ("Unintended Uses"). Customers shall take any and all actions to ensure using and selling the Products in accordance with the applicable laws and regulations. The Company is not liable, in whole or in part, and customers shall and hereby do release the Company as well as its suppliers and/or distributors from any claim, damage, or other liability arising from or related to all Unintended Uses of the Products. Customers shall indemnify and hold the Company as well as its suppliers and/or distributors harmless from and against all claims, costs, damages, and other liabilities, including claims for personal injury or death, arising from or related to any Unintended Uses of the Products.

Information in this document is provided solely in connection with the Products. The Company reserves the right to make changes, corrections, modifications or improvements to this document and Products and services described herein at any time, without notice.